アプリケーションノート

TXZ+マイコン機能安全 STL

目次

1. はじめに (Introduction)

1.1. 目的 (Purpose)

本書は TXZ+マイコン機能安全 STL についてソフトウェア開発者向けに技術的な詳細を記述しています。

1.2. 用語、と略語 (Definitions, acronyms, and abbreviations)

用語/略語	定義/詳細な説明
ユーザー	東芝マイコンを使用する顧客
STL	Self-Test Library
STK	スターターキット(評価ボード)
IEC60730	家電製品の安全性を確保するための国際規格
APN	アプリケーションノート
RM	リファレンスマニュアル
TD	データシート
ES	Engineering Sample

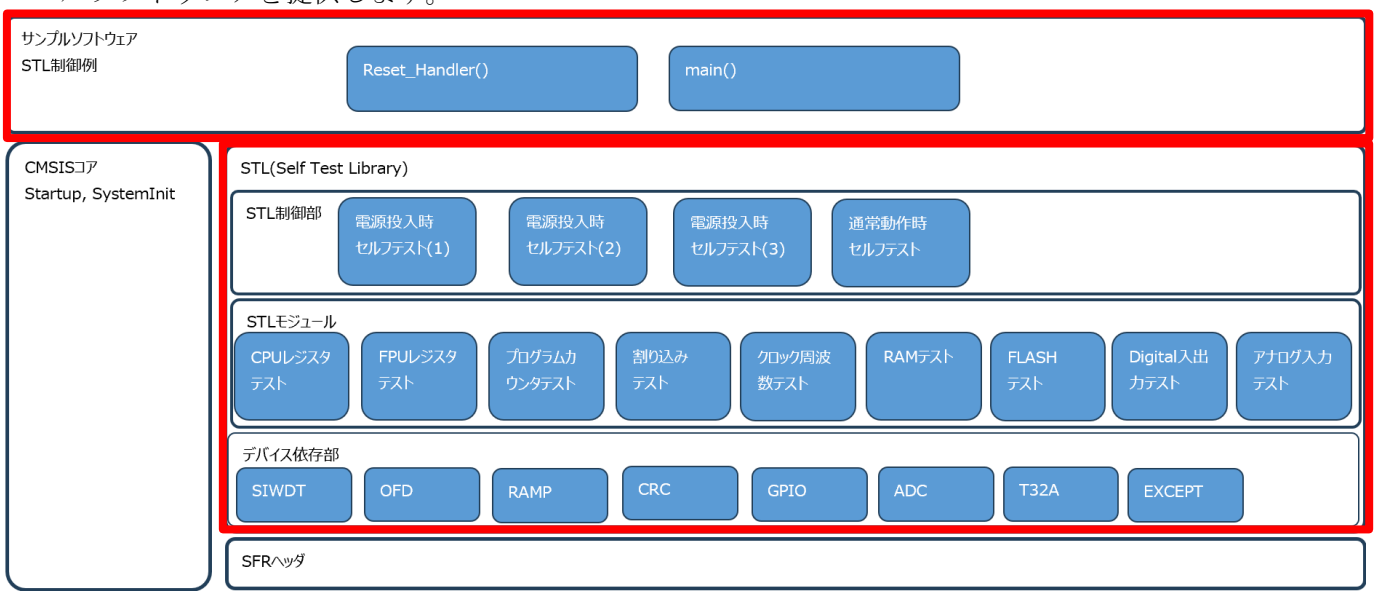
1.3. 参照資料 (References)

文書名	IP 記号
IEC60730 規格書	-
Arm® Cortex® -M4 Processor Technical Reference Manual	-
TXZ+ファミリー TMPM4K グループ(2)データシート	-
CRC 計算回路	CRC-A
周波数検知回路	OFD-A
12bit アナログデジタルコンバーター	ADC-F
クロック制御と動作モード(TMPM4K グループ(2))	CG-M4K(2)-E
例外(TMPM4K グループ(2))	EXCEPT-M4K(2)
フラッシュメモリー	FLASH10MUD32-A
製品個別情報(TMPM4K グループ(2))	PINFO-M4K(2)
入出力ポート(TMPM4K グループ(2))	PORT-M4K(2)
32bit タイマーイベントカウンター	T32A-C
RAM パリティー	RAMP-B
TXZ+ファミリー TMPM4M グループ(1)データシート	-
クロック制御と動作モード(TMPM4M グループ(1))	CG-M4M(1)-E
例外(TMPM4M グループ(1))	EXCEPT-M4M(1)
フラッシュメモリー	FLASH512UD32-B
製品個別情報(TMPM4M グループ(1))	PINFO-M4M(1)
入出力ポート(TMPM4M グループ(1))	PORT-M4M(1)

2. 概要 (Overall description)

2.1. 仕様概要 (Specification overview)

- 本機能安全 STL は、安全規格 IEC/UL60730 安全規格の Annex H クラス B 機器に利用可能なセルフテストライブラリー，およびセルフテストライブラリーを利用するサンプルソフトウェアにより構成されます。
- TXZ+ファミリー M4K(2)グループ，M4M グループを対象とし、評価ボードで動作確認を行えるサンプルソフトウェアを提供します。



注：赤枠が本 機能安全 STL の対象

2.2. STL の機能概要 （STL functions overview）

- 以下の機能安全 STL 機能を提供します。

名称	テスト内容	IEC 60730-1 カテゴリー
CPU レジスターテスト	全ての読み書き可能内部レジスターに対して 0x55555555 および 0xaaaaaaaa のチェッカーパターンを書き込み、それが正常に読み出せることを確認する	1.1 CPU Registers
FPU レジスターテスト	全ての読み書き可能内部レジスターに対して 0x55555555 および 0xaaaaaaaa のチェッカーパターンを書き込み、それが正常に読み出せることを確認する	1.1 CPU Registers
CPU プログラムカウンターテスト	ジャンプ命令実行後に、PC の値がプログラム記述時(ビルド時)に想定された値になっていることを確認する。 ジャンプ先アドレスのパターンを 10 通り用意し、全て正常に実行できたときに呼び出し元に返ってくる。	1.2 CPU Program Counter
割り込みテスト	タイマー割り込みを利用し、一定時間経過内に予想した頻度で割り込み処理が発生し、正常に割り込み処理が終了することを確認する。 割り込みが一度も発生しなかったり、設計よりも非常に多くの回数割り込みが発生したりした場合にはエラーとする。	2. Interrupt handling and execution
クロック周波数テスト	周波数検知回路(OFD)と基準クロック入力(IHOSC2)を用いて、システムクロック(f_c)あるいは外部高速発振器(f_{EHOSC})のクロック周波数異常かどうかを一定時間チェックする。 チェック時間経過後、OFD でクロック異常が発生したかどうかを返す。	3. Clock
RAM テスト	チェッカーパターン 0x55555555 および 0xaaaaaaaa を同一メモリーアドレスへ正常に Write/Read 可能なことと、March C アルゴリズムによる確認をする。これを指定されたメモリー範囲に対して実行する。 デバイスに RAMP 機能がある場合は，RAMP 機能を使用して確認する。	4.2 Variable Memory
FLASH テスト	指定されたアドレス範囲の CRC32 の値を計算し、FLASH 書き込みが異なる値として読み出されていないことを確認する。 プログラムビルド時にあらかじめ計算されていた CRC32 の値と比較する。 デバイスに CRC 機能がない場合は，ソフトウェア CRC で計算する	4.1 Invariable memory
Digital 入出力テスト	I/O ポートの入出力を両方許可状態にして、同一ポートの出力状態を入力することでテストする。 Hi を出力しているときに入力端子が Hi になっているか、また Lo を出力しているときに入力端子が Lo になっているかをチェックする。	7.1 Digital I/O
Analog 入力テスト	入力検証用の 3 種類の入力電圧(VREFL/VREFH/REGOUT1)を発生させ、AD 変換で取得される値をテストする。	7.2 Analog I/O

2.3. 開発環境 (Development environment)

- 開発言語は、C およびアセンブリー言語を使用します。
- ビルド環境として IAR EWARM, ARM KEIL-MDK を使用します。
- 下記の開発環境を使用します。

名称	説明
PC	OS は Windows11 を使用する。
IAR EWARM 9.50.2	IAR 社の IDE(統合開発環境)
Arm® Keil™ MDK 5.40.0	ARM 社の IDE(統合開発環境)

3. 仕様詳細 (Specific requirements)

3.1. 構成詳細

3.1.1. ツリー構造

TXZplusSTL │ ├──Examples │ └──src │ └──STL_app ├──MCU │ ├──CMSIS │ ├──CMSIS_M4K │ │ └──startup │ ├──CMSIS_M4M │ │ └──startup │ └──Driver │ ├──inc │ └──src ├──PJ │ ├──TMPM4K │ │ ├──arm │ │ └──iar │ └──TMPM4M │ ├──arm │ └──iar └──SelfTestLibrary ├──inc └──src	STL サンプルソフトウェア MCU 固有定義 デバイス依存部 Project フォルダー セルフテストライブラリー本体

3.1.2. ファイル一覧

STL サンプルソフトウェア, セルフテストライブラリー, デバイス依存部のファイル一覧

ファイル名	説明
main.c	STL 用サンプルソフトウェア
SelfTest_cpu_register.s	CPU レジスターテスト
SelfTest_fpu_register.s	FPU レジスターテスト
SelfTest_fpu_control.s	FPU コントロールレジスターテスト
SelfTest_cpu_pc.c	プログラムカウンターテスト
SelfTest_interrupt.c	割り込みテスト
SelfTest_clock.c	クロック周波数テスト
SelfTest_ram_cheker.c	RAM テスト チェッカーパターン
SelfTest_ram_march_c.c	RAM テスト MARCH C
SelfTest_ram_ramp.c	RAM テスト RAM パリティー
SelfTest_flash_crc_hw.c	FLASH テスト ハードウェア CRC
SelfTest_flash_crc_sw.c	FLASH テスト ソフトウェア CRC
SelfTest_digital_io.c	デジタル入出力テスト
SelfTest_analog_in.c	アナログ入力テスト
SelfTest_adc.c	デバイス依存部 A/D コンバーター
SelfTest_cg.c	デバイス依存部 クロックジェネレーター
SelfTest_gpio.c	デバイス依存部 ポート
SelfTest_except.c	デバイス依存部 割り込み
SelfTest_crc.c	デバイス依存部 CRC
SelfTest_ofd.c	デバイス依存部 OFD
SelfTest_ramp.c	デバイス依存部 RAM パリティー

3.1.3. API 一覧

セルフテストライブラリーの API 一覧

ファイル名	API 名	機能
SelfTest_cpu_register. s	SelfTest_CPU_Register	CPU レジスターテスト
SelfTest_fpu_register. s	SelfTest_FPU_Register	FPU レジスターテスト
SelfTest_fpu_control. s	SelfTest_FPU_Control	FPU コントロールレジスター テスト
SelfTest_cpu_pc.c	SelfTest_CPU_ProgramCounter	プログラムカウンターテスト
SelfTest_interrupt.c	SelfTest_Interrupt	割り込みテスト
SelfTest_clock.c	SelfTest_Clock	クロック周波数テスト
SelfTest_ram_checker.c	SelfTest_RAM_CheckerPattern	RAM テスト チェッカーパターン
SelfTest_ram_march_c.c	SelfTest_RAM_March_C	RAM テスト March C
SelfTest_ram_ramp.c	SelfTest_RAM_ramp_init	RAM テスト RAM パリティー機能初期化
	SelfTest_RAM_ramp	RAM テスト RAM パリティー
	SelfTest_RAM_ramp_handler	RAM テスト RAM パリティー機能割り込みハンドラー
SelfTest_flash_crc_hw.c	SelfTest_FLASH_CRC_HW	FLASH テスト ハードウェア CRC
SelfTest_flash_crc_sw.c	SelfTest_FLASH_CRC_SW	FLASH テスト ソフトウェア CRC
SelfTest_digital_io.c	SelfTest_DigitalIO	デジタル入出力テスト
SelfTest_analog_in.c	SelfTest_AnalogIn	アナログ入力テスト

3.2. 機能詳細 (Function details)

TXZ+マイコン機能安全 STL は以下の機能を提供します。

3.2.1. CPU レジスターテスト

CPU レジスターの「Stuck at」エラーチェック「注１」を以下の CPU レジスターに対して行います。
0x55555555 および 0xaaaaaaaa のチェッカーパターンを書き込み、それが正常に読み出せることを確認します。

- 汎用レジスター(R0 – R12)
- スタックポインター(R13 (Main SP および Process SP)「注２」)
- リンクレジスター(R14 (LR))
- 制御レジスター(CPSR (上位 5bit のみ))

CPU レジスターテストは順番に行われます。あるレジスターでエラー検出した場合にはその時点でテストを中断しエラーを返します。その際に、エラーの発生したレジスターを表すビットマスク値を同地に返します

- この API は(ユーザー状態ではなく)特権状態で呼び出されることを想定しています。
- この API は割り込み禁止状態で呼び出す必要があります。C 言語で関数呼び出し時に非破壊とされているレジスターは、テスト前後で保存されます。
- この API は割り込みを使用しません。

注意：プログラムを実行すると必ず CPU レジスターを使用するため、自己検証では欠陥を完全に検出できません。

注１：「Stuck at」チェックとは、あるビットが 0 または 1 に固定されている異常を検出するチェックです。

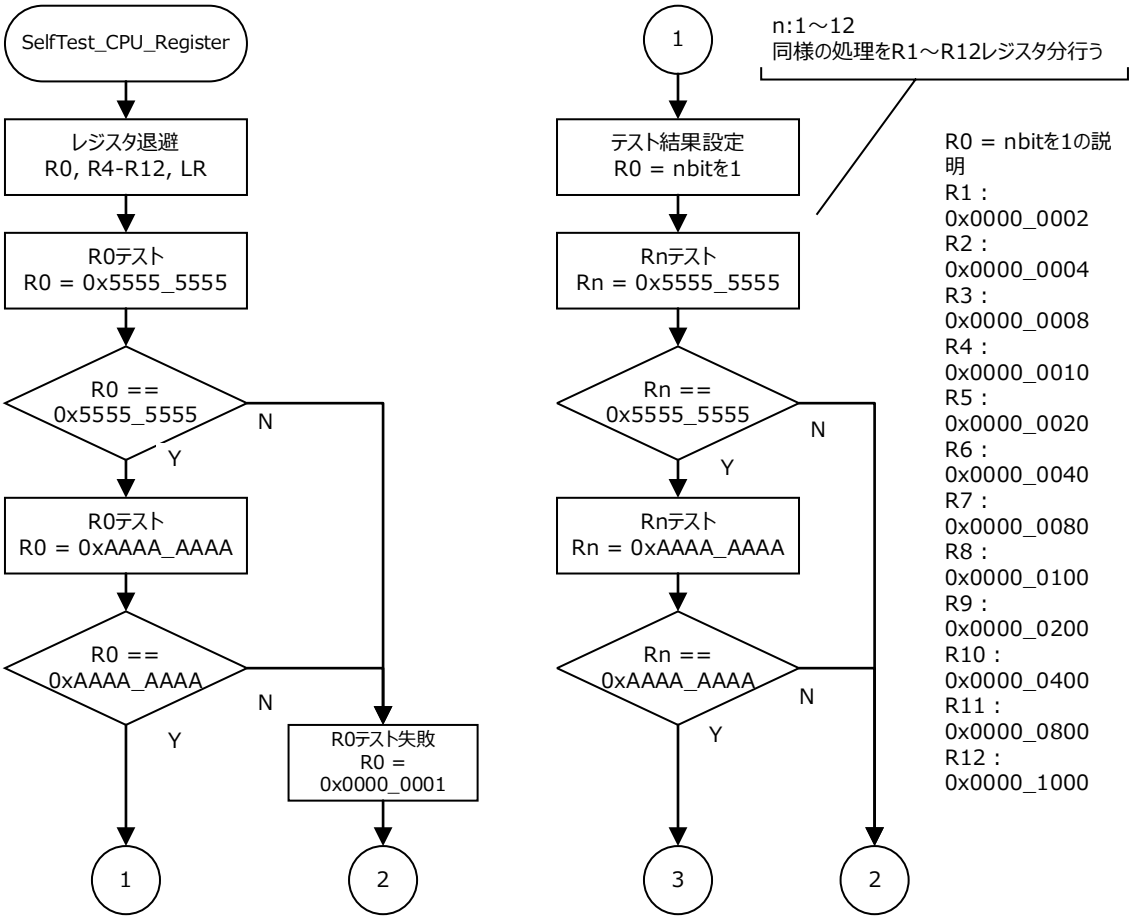
注２：SP レジスターの下位 2bit は 0 固定です。これらの固定ビットはテストされません。

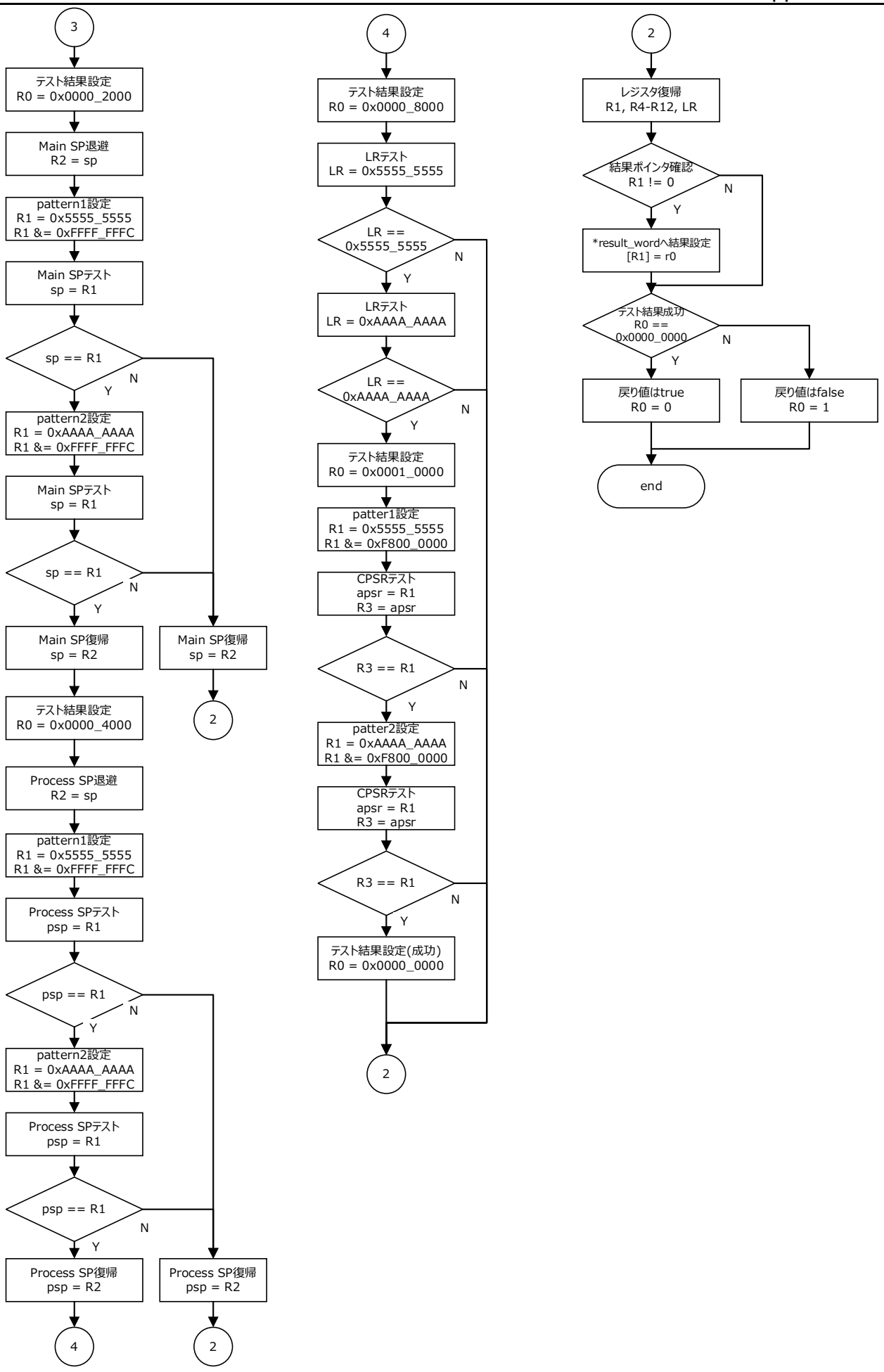
ソースファイル: SelfTest_cpu_register.s
ヘッダーファイル: SelfTest_cpu_register.h

関数名
bool SelfTest_CPU_Register(uint32_t *result_word)
説明
レジスターを退避します 汎用レジスターR0～R12 のテストを実行します R0 に 0x5555_5555 を書き込みます R0 と 0x5555_5555 を比較し不一致の場合, R0 の bit0 に 1 を書き込みエラー処理に Jump します R0 に 0xAAAA_AAAA を書き込みます R0 と 0xAAAA_AAAA を比較し不一致の場合, R0 の bit0 に 1 を書き込みエラー処理に Jump します R1 に 0x5555_5555 を書き込みます R1 と 0x5555_5555 を比較し不一致の場合, R0 の bit1 に 1 を書き込みエラー処理に Jump します R1 に 0xAAAA_AAAA を書き込みます R1 と 0xAAAA_AAAA を比較し不一致の場合, R0 の bit1 に 1 を書き込みエラー処理に Jump します 以降, R2, R3,R4・・・R12 まで同様に実行します。不一致の場合の R0 に書込む値は出力パラメーターの該当 bit に 1 を書き込みエラー処理に Jump します Main SP(MSP)のテストを実行します MSP を退避します MSP に 0x5555_5554 を書き込みます(下位 2bit は 0 にします) MSP と 0x5555_5554 を比較し不一致の場合, R0 の bit13 に 1 を書き込み MSP を復帰しエラー処理に Jump します MSP に 0xAAAA_AAA8 を書き込みます(下位 2bit は 0 にします) MSP と 0xAAAA_AAA8 を比較し不一致の場合, R0 の bit13 に 1 を書き込み MSP を復帰しエラー処理に Jump します テストが成功したとき, MSP を復帰し Process SP のテストを実行します Process SP(PSP)のテストを実行します PSP を退避します PSP に 0x5555_5554 を書き込みます(下位 2bit は 0 にします) PSP と 0x5555_5554 を比較し不一致の場合, R0 の bit14 に 1 を書き込み PSP を復帰しエラー処理に Jump します PSP に 0xAAAA_AAA8 を書き込みます(下位 2bit は 0 にします) PSP と 0xAAAA_AAA8 を比較し不一致の場合, R0 の bit14 に 1 を書き込み PSP を復帰しエラー処理に Jump します テストが成功したとき, PSP を復帰しリンクレジスターのテストを実行します リンクレジスター(LR)のテストを実行します LR に 0x5555_5555 を書き込みます LR と 0x5555_5555 を比較し不一致の場合, R0 の bit15 に 1 を書き込みエラー処理に Jump します LR に 0xAAAA_AAAA を書き込みます LR と 0xAAAA_AAAA を比較し不一致の場合, R0 の bit15 に 1 を書き込みエラー処理に Jump します

CPSR のテストを実行します APSR に 0x5000_0000 を書き込みます(APSR の上位 5bit が CPSR のため他 bit は 0 にします) APSR と 0x5000_0000 を比較し不一致の場合, R0 の bit16 に 1 を書き込みエラー処理に Jump します APSR に 0xA800_0000 を書き込みます(APSR の上位 5bit が CPSR のため他 bit は 0 にします) APSR と 0xA800_0000 を比較し不一致の場合, R0 の bit16 に 1 を書き込みエラー処理に Jump します 退避していたレジスターを復帰します R0 を出力パラメーターにセットします 引数出力パラメーターがセットされていない場合, 戻り値(R0)は 0x01 を返します テストが完了したとき、戻り値(R0)は 0x00 を返します	
入力パラメーター	
なし	
出力パラメーター	
uint32_t *result_word	結果を格納するメモリーバッファー bit0：R0 テスト失敗 - bit12：R12 テスト失敗 bit13：Main SP テスト失敗 bit14：Process SP テスト失敗 bit15：LR テスト失敗 bit16：CPSR テスト失敗
戻り値	
Bool	テスト結果(true: 成功、false: 失敗)

- テストのフローチャート




3
テスト結果設定
R0 = 0x0000_2000
Main SP退避
R2 = sp
pattern1設定
R1 = 0x5555_5555
R1 &= 0xFFFF_FFFC
Main SPテスト
sp = R1
sp == R1
N
Y
pattern2設定
R1 = 0xAAAA_AAAA
R1 &= 0xFFFF_FFFC
Main SPテスト
sp = R1
sp == R1
N
Y
Main SP復帰
sp = R2
Main SP復帰
sp = R2
2
テスト結果設定
R0 = 0x0000_4000
Process SP退避
R2 = sp
pattern1設定
R1 = 0x5555_5555
R1 &= 0xFFFF_FFFC
Process SPテスト
psp = R1
psp == R1
N
Y
pattern2設定
R1 = 0xAAAA_AAAA
R1 &= 0xFFFF_FFFC
Process SPテスト
psp = R1
psp == R1
N
Y
Process SP復帰
psp = R2
Process SP復帰
psp = R2
4
2
4
テスト結果設定
R0 = 0x0000_8000
LRテスト
LR = 0x5555_5555
LR ==
0x5555_5555
N
Y
LRテスト
LR = 0xAAAA_AAAA
LR ==
0xAAAA_AAAA
N
Y
テスト結果設定
R0 = 0x0001_0000
patter1設定
R1 = 0x5555_5555
R1 &= 0xF800_0000
CPSRテスト
apsr = R1
R3 = apsr
R3 == R1
N
Y
patter2設定
R1 = 0xAAAA_AAAA
R1 &= 0xF800_0000
CPSRテスト
apsr = R1
R3 = apsr
R3 == R1
N
Y
テスト結果設定(成功)
R0 = 0x0000_0000
2
2
レジスタ復帰
R1, R4-R12, LR
結果ポインタ確認
R1 != 0
N
Y
*result_wordへ結果設定
[R1] = r0
テスト結果成功
R0 ==
0x0000_0000
N
Y
戻り値はtrue
R0 = 0
戻り値はfalse
R0 = 1
end

3.2.2. FPU レジスターテスト

FPU レジスターの「Stuck at」エラーチェックを以下の FPU レジスターに対して行います。
0x55555555 および 0xaaaaaaaa のチェッカーパターンを書き込み、それが正常に読み出せることを確認します。

- FPU 拡張レジスター(S0 – S31)
- FPU 制御レジスター(CPACR, FPCCR, FPCAR, FPSCR, FPDSCR)

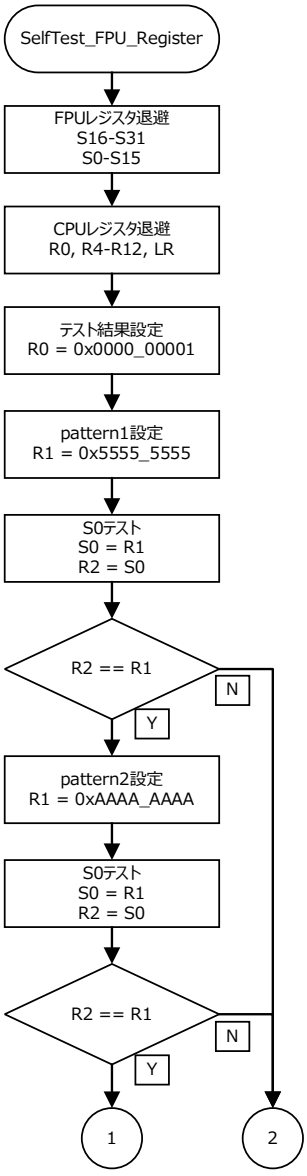
FPU レジスターテストは順番に行われます。あるレジスターでエラー検出した場合にはその時点でテストを中断しエラーを返します。その際に、エラーの発生したレジスターを表すビットマスク値を同時に返します

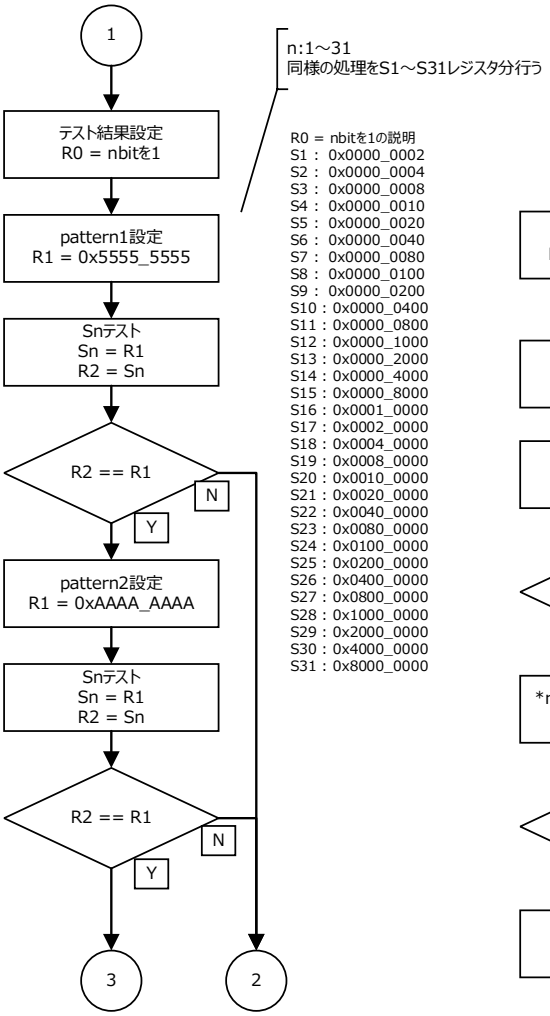
- この API は(ユーザー状態ではなく)特権状態で呼び出されることを想定しています。
- この API は割り込み禁止状態で呼び出す必要があります。C 言語で関数呼び出し時に非破壊とされているレジスターは、テスト前後で保存されます。
- この API は割り込みを使用しません。

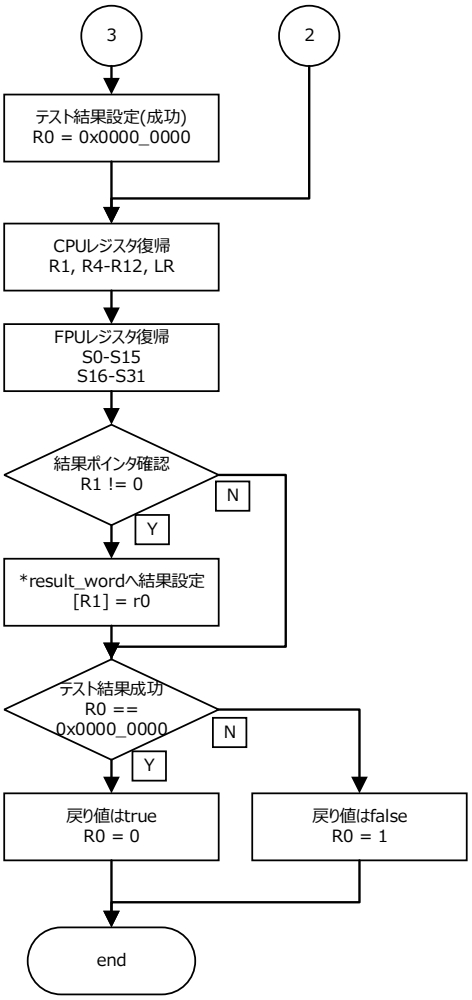
ソースファイル: SelfTest_fpu_register.s
ヘッダーファイル: SelfTest_fpu_register.h

関数名	
bool SelfTest_FPU_Register(uint32_t *result_word)	
説明	
レジスターを退避します FPU 拡張レジスターS0～S31 のテストを実行します S0 に 0x5555_5555 を書き込みます S0 と 0x5555_5555 を比較し不一致の場合, R0 の bit0 に 1 を書き込みエラー処理に Jump します S0 に 0xAAAA_AAAA を書き込みます S0 と 0xAAAA_AAAA を比較し不一致の場合, R0 の bit0 に 1 を書き込みエラー処理に Jump します S1 に 0x5555_5555 を書き込みます S1 と 0x5555_5555 を比較し不一致の場合, R0 の bit1 に 1 を書き込みエラー処理に Jump します S1 に 0xAAAA_AAAA を書き込みます S1 と 0xAAAA_AAAA を比較し不一致の場合, R0 の bit1 に 1 を書き込みエラー処理に Jump します 以降, S2, S3,S4・・・S31 まで同様に実行します。不一致の場合の R0 に書込む値は出力パラメーターの該当 bit に 1 を書き込みエラー処理に Jump します 退避していたレジスターを復帰します R0 を出力パラメーターにセットします 引数出力パラメーターがセットされていない場合, 戻り値(R0)は 0x01 を返します テストが完了したとき、戻り値(R0)は 0x00 を返します	
入力パラメーター	
なし	
出力パラメーター	
uint32_t *result_word	結果を格納するメモリーバッファー bit0：S0 テスト失敗 – bit31：S31 テスト失敗
戻り値	
Bool	テスト結果(true: 成功、false: 失敗)

● テストのフローチャート



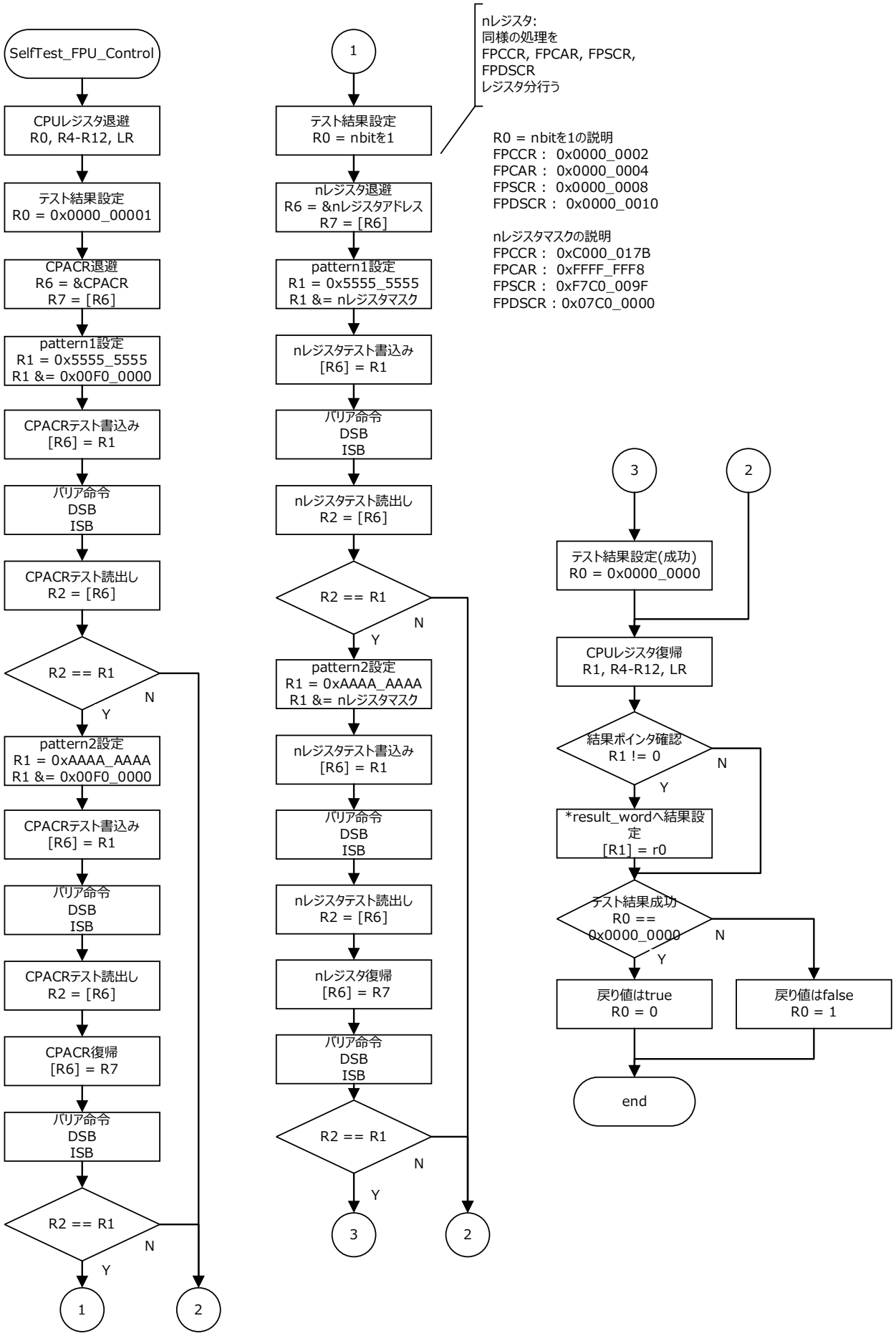




ソースファイル: SelfTest_fpu_control.s
ヘッダーファイル: SelfTest_fpu_control.h

関数名	
bool SelfTest_FPU_Control (uint32_t *result_word)	
説明	
レジスターを退避します FPU 制御レジスター(CPACR, FPCCR, FPCAR, FPSCR, FPDSCR)のテストを実行します CPACR を退避します CPACR に 0x5555_5555 を書き込みます CPACR と 0x5555_5555 を比較し不一致の場合, R0 の bit0 に 1 を書き込みます CPACR を復帰しエラー処理に Jump します CPACR に 0xAAAA_AAAA を書き込みます CPACR と 0xAAAA_AAAA を比較し不一致の場合, R0 の bit0 に 1 を書き込みます CPACR を復帰しエラー処理に Jump します テストが成功したとき, CPACR を復帰し FPCCR テストを実行します 以降, FPCCR, FPCAR, FPSCR, FPDSCR の順にテストを実行します。 不一致の場合, R0 に書込む値は出力パラメーターの該当 bit に 1 を書き込みエラー処理に Jump します 退避していたレジスターを復帰します R0 を出力パラメーターにセットします 引数出力パラメーターがセットされていない場合, 戻り値(R0)は 0x01 を返します テストが完了したとき、戻り値(R0)は 0x00 を返します	
入力パラメーター	
なし	
出力パラメーター	
uint32_t *result_word	結果を格納するメモリーバッファー bit0：CPACR テスト失敗 bit1：FPCCR テスト失敗 bit2：FPCAR テスト失敗 bit3：FPSCR テスト失敗 bit4：FPDSCR テスト失敗
戻り値	
Bool	テスト結果(true: 成功、false: 失敗)

- テストのフローチャート



3.2.3. CPU プログラムカウンターテスト

CPU プログラムカウンターの「Stuck at」エラーチェックを行います。
この関数内ではプログラムカウンターに対して代入を行う代わりに複数回のジャンプ命令を実行します。テストが失敗した場合にはプログラムは復旧しません。逆に実行が終了した場合には必ずテスト成功となります。

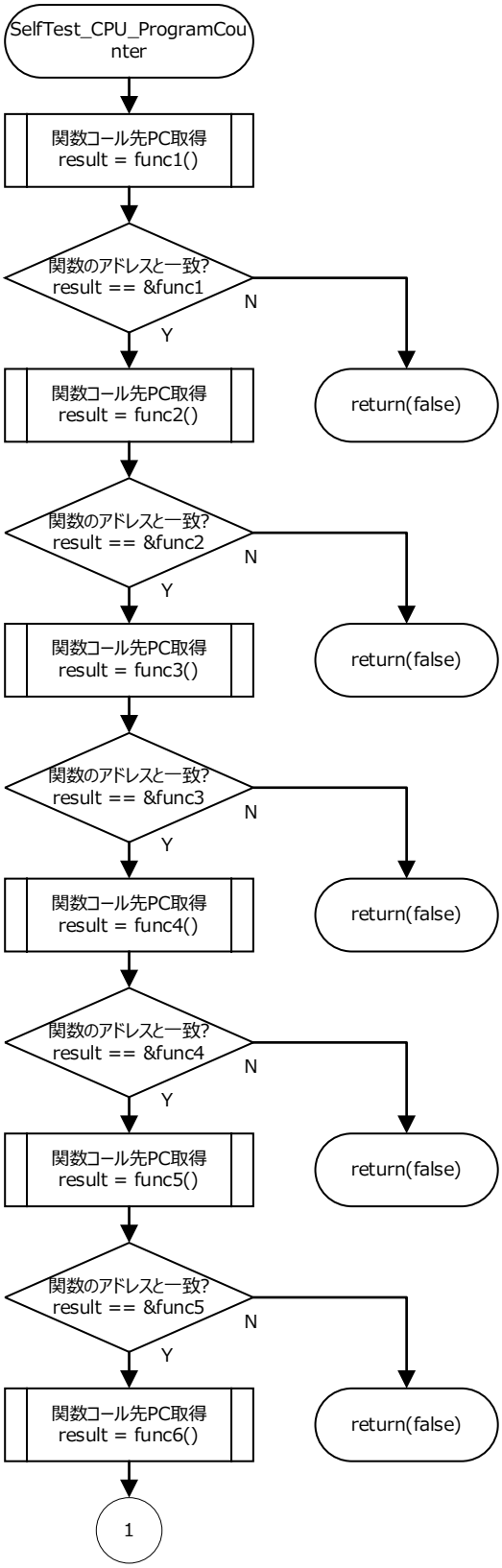
- このライブラリー関数は割り込み禁止状態で呼び出す必要があります。C 言語で関数呼び出し時に非破壊とされているレジスターは、テスト前後で保存されます。
- このライブラリー関数は割り込みを使用しません。

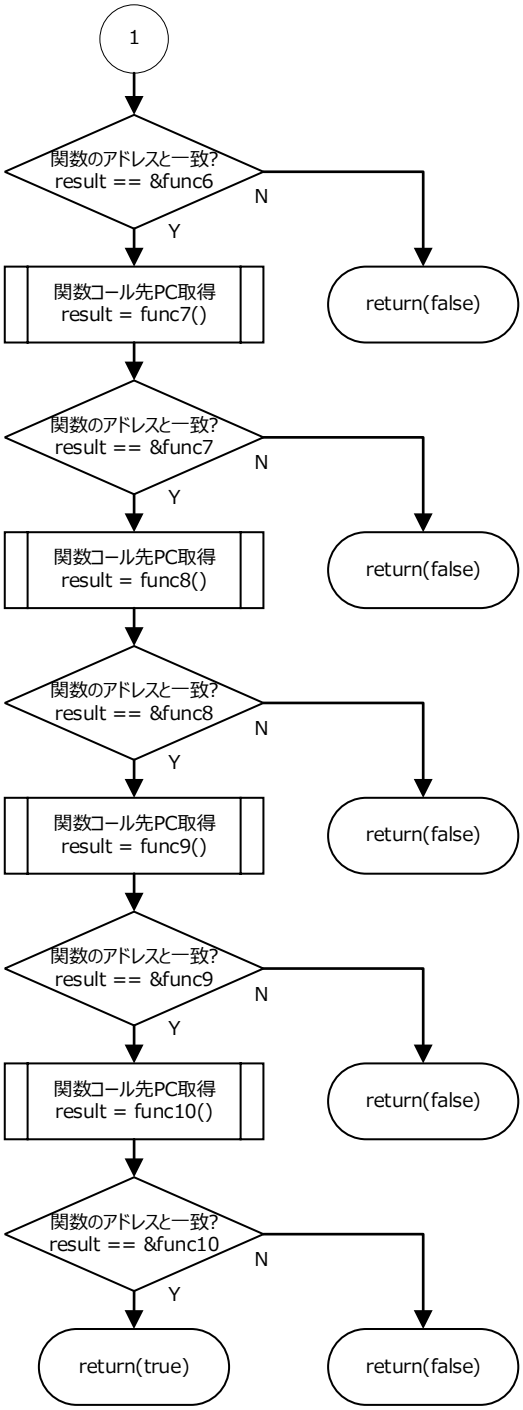
ソースファイル: SelfTest_cpu.pc.c
ヘッダーファイル: SelfTest_cpu_pc.h

関数名	
bool SelfTest_CPU_ProgramCounter(void)	
説明	
関数(func1)をコールします 　関数(func1)は programcounter(PC)を取得し，戻り値として返します 関数(func1)のアドレスと戻り値(PC)を比較し不一致の場合，失敗(false)を返しテストを終了します 成功の場合，関数(func2)をコールします 　関数(func2)は programcounter(PC)を取得し，戻り値として返します 関数(func2)のアドレスと戻り値を比較し不一致の場合，失敗(false)を返しテストを終了します 成功の場合，関数(func3)をコールします 　以降，関数(func3, func4～func10)まで繰り返します 関数(func10)が成功したとき，成功(true)を返しテストを終了します	
入力パラメーター	
なし	
出力パラメーター	
なし	
戻り値	
Bool	テスト結果 正常に実行されたときは true：成功が返る。 失敗時には基本呼び出し側に戻ってこないが、false が返ってくる場合は正常にコンパイルされていないなどの内部エラーが考えられる「注１」。

注１：呼び出すサブルーチンがインライン展開されたときなどにこのようになることがあります。このテストの実装では関数のインライン展開を無効にしています。

- テストのフローチャート





3.2.4. 割り込みテスト

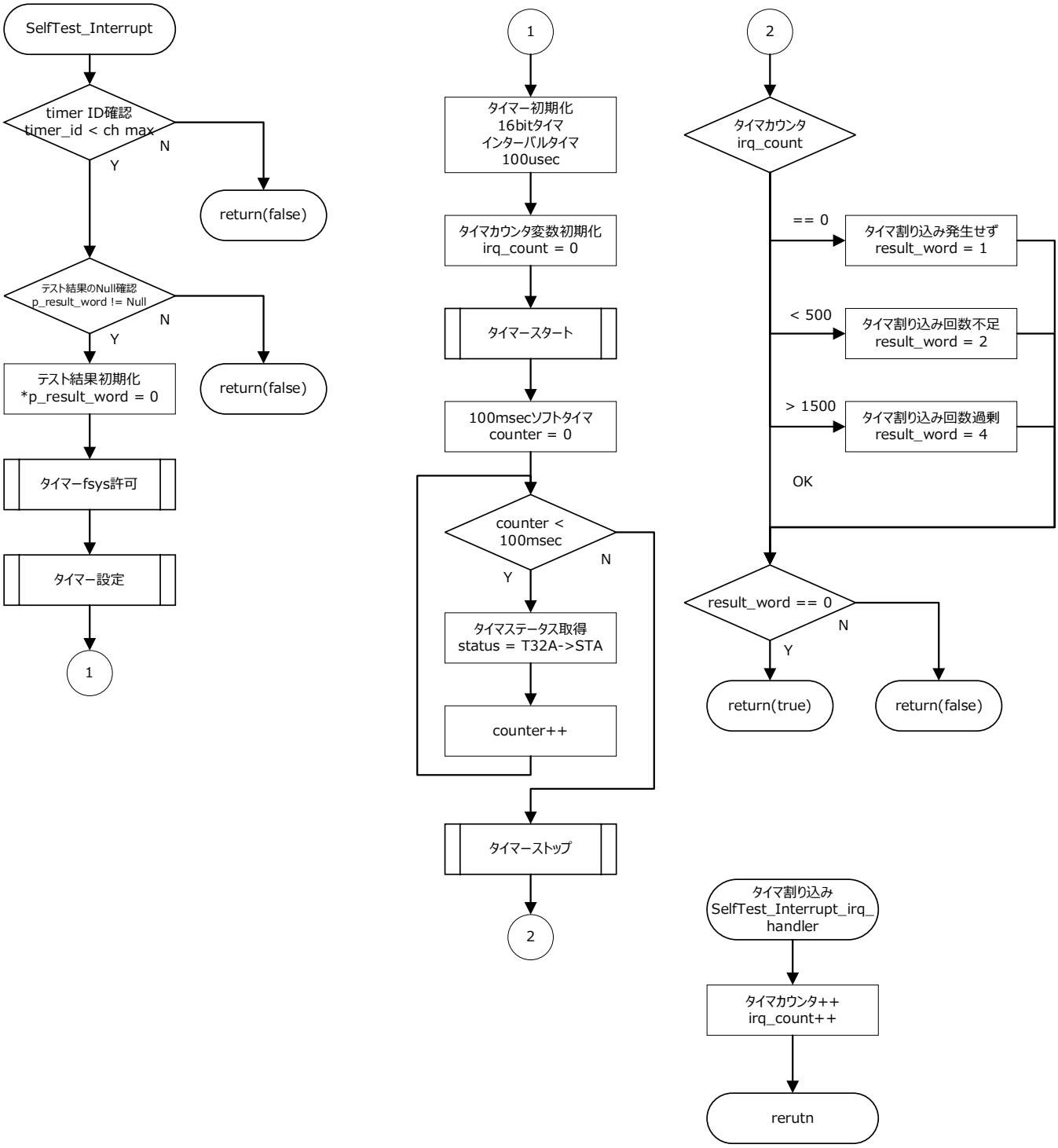
タイマーを割り込み源とした割り込みテストを行います。テスト中に一度も割り込みがかからないか、想定したより割り込み回数が少な過ぎる、または多過ぎるときにエラーとします。

- テスト時間の計測はビジーループで行います。あらかじめ時間計測した結果のループカウンタ値を用いて、約 100msec の間 100 マイクロ秒間隔で発生する割り込み回数をカウントします。標準では約 1000 回の割り込みが発生するはずですが、500 回以上 1500 回以下の割り込み発生回数が検出された場合 OK とし、これを外れるか、または 1 回も発生しなかった場合はエラーとします。
  - 本テストはシステム起動時に最初の割り込み処理のセットアップ前に実行されることを想定します。
  - ビジーループ時間はマイコンの動作周波数に依存するため，ユーザーにより変更できるようにします。
  - 発生回数の OK 検出条件は，ユーザーにより変更できるようにします。

ソースファイル: SelfTest_interrupt.c
ヘッダーファイル:　SelfTest_interrupt.h

関数名	
bool SelfTest_Interrupt(int timer_id, uint32_t *result_word)	
説明	
入力パラメーター(timer_id)で指定された T32 タイマーチャネル番号の初期化を行います 16bit タイマーモード，インターバルタイマー，100usec 割り込み タイマー割り込み内でカウントするカウンター変数を 0 に初期化 T32 タイマーのカウントをスタート ソフトウェアタイマーにより 100msec 間待ちます T32 タイマーを停止 カウンター変数が 0 の時，出力パラメーターの bit0 に 1 をセットし失敗(false)を返してテストを終了します カウンター変数が 500 回未満の時，出力パラメーターの bit1 に 1 をセットし失敗(false)を返してテストを終了します カウンター変数が 1500 回より多い時，出力パラメーターの bit2 に 1 をセットし失敗(false)を返してテストを終了します カウンター変数が 500 回以上 1500 回以下の時，出力パラメーターの bit0,1,2 に 0 をセットし成功(true)を返したテストを終了します	
入力パラメーター	
int timer_id	使用する T32 タイマーチャネル番号（0 – 5）
出力パラメーター	
uint32_t *result_word	結果を格納するメモリーバッファー bit0： タイマー割り込み発生せず bit1： タイマー割り込みの回数が少な過ぎる bit2： タイマー割り込みの回数が多過ぎる
戻り値	
Bool	テスト結果(true: 成功、false: 失敗)

- テストのフローチャート



3.2.5. クロック周波数テスト

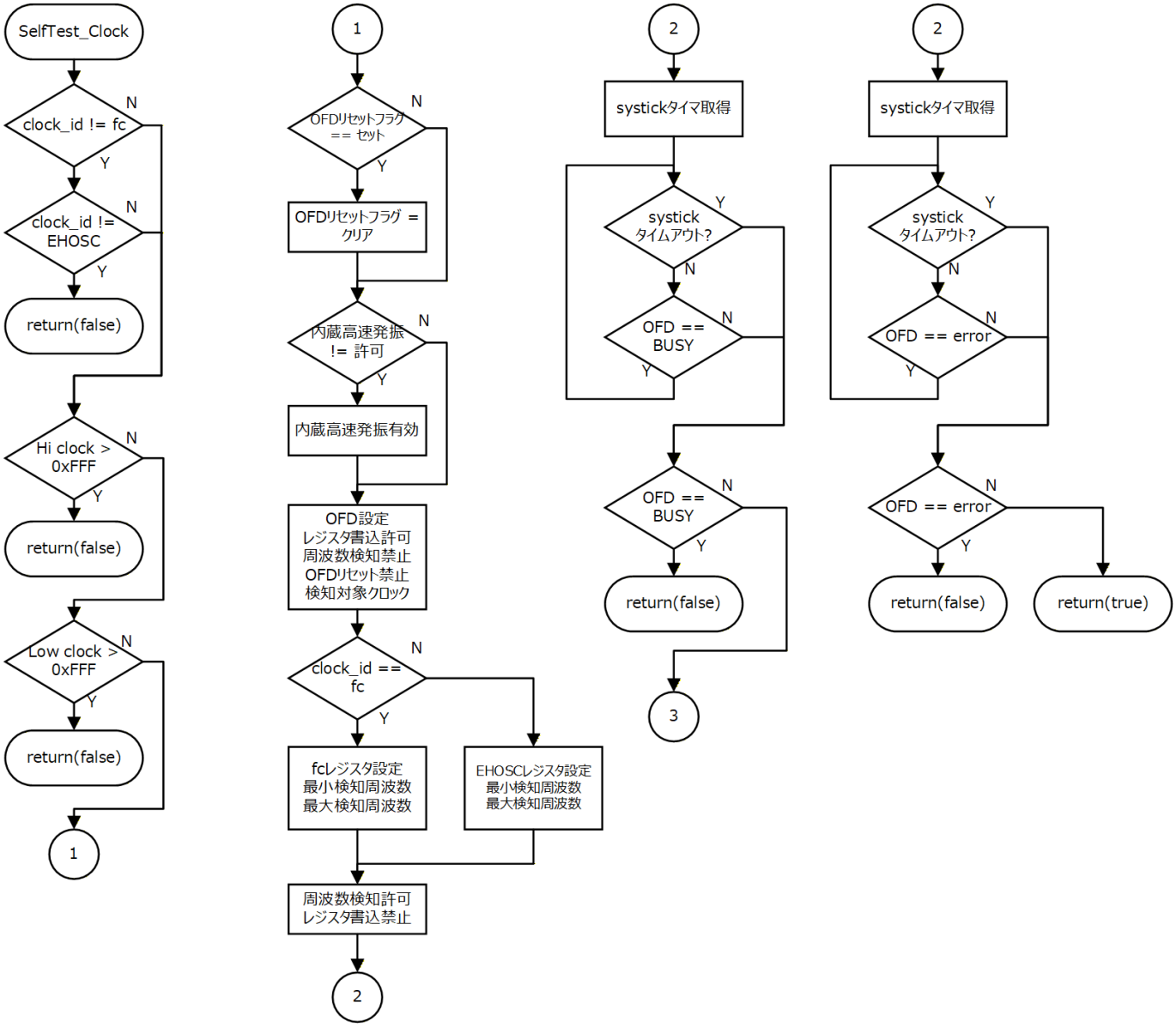
周波数検知回路(OFD)と基準クロック入力(IHOSC2)を用いて、システムクロック(fc)または外部高速発振器(fEHOSC)のクロック周波数異常かどうかを一定時間(100msec)チェックします。

- テスト対象のクロックが fc または fEHOSC のいずれかから選択可能です。
- 基準クロック（IHOSC2）が無効の場合、テスト中に自動的に有効にされ、テスト後はもとの設定(有効または無効)に戻します。
  - 実装例:
    基準クロックの周波数が 10MHz の場合、この値を用いて high_clock および low_clock の値を計算する例が、「OFD リファレンスマニュアル」に記載されています。

ソースファイル: SelfTest_clock.c
ヘッダーファイル: SelfTest_clock.h

関数名	
bool SelfTest_clock(int clock_id, int high_clock, int low_clock)	
説明	
入力パラメーターの設定値を確認します 入力パラメーター(clock_id)が 0(fc), 1(fEHOSC)以外の時，失敗(false)を返しテストを終了します 入力パラメーター(high_clock)が 0x1000 より大きい時，失敗(false)を返しテストを終了します 入力パラメーター(low_clock)が 0x1000 より大きい時，失敗(false)を返しテストを終了します 入力パラメーターが正常な時， テストするクロック入力種別の設定，最大検知周波数の設定，最小検知周波数の設定 周波数検知動作制御を許可 OFD 動作状態が動作中まで待ちます 100msec 間，異常検知フラグを確認します 異常検知フラグが異常の時，失敗(false)を返しテストを終了します 異常検知フラグが異常でない時，成功(true)を返しテストを終了します	
入力パラメーター	
int clock_id	テストするクロック入力の種別 0：fc 1：fEHOSC これ以外の値の場合エラーとします。
int high_clock	高周波側の設定値(12bit) 0x1000 以上の値の場合はエラーとします。
int low_clock	低周波数の設定値(12bit) 0x1000 以上の値の場合はエラーとします。
出力パラメーター	
なし	
戻り値	
bool	テスト結果(true：成功、false：失敗)

● テストのフローチャート



3.2.6. RAM テスト

チェッカーパターン 0x55555555 および 0xaaaaaaaa を同一メモリーアドレスに正常に Write/Read 可能なこと、March C アルゴリズムによる確認します。これを指定されたメモリー範囲に対して実行します。

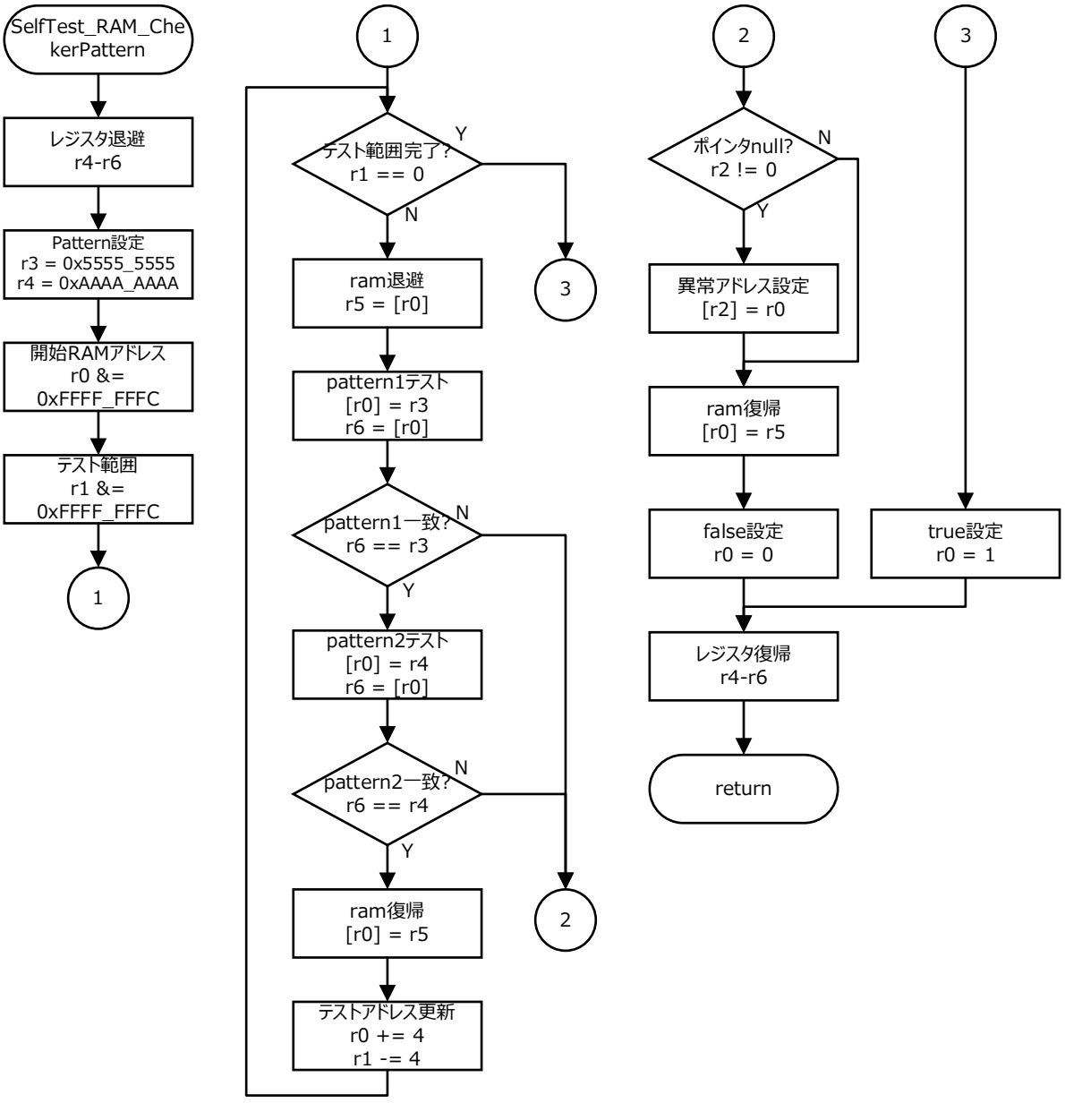
- エラー検出時にはそのアドレスを結果のメモリーバッファーに書き込みます。
  - 各メモリーアドレスの値はテスト後にはテスト前の値を戻しますが、テスト失敗時には元の値に戻っていることを保証できません。
  - テスト範囲に割り込み処理時に変更される RAM 領域(スタック領域や割り込み処理中に書き込む変数領域)が含まれていた場合、予期しない変更が発生する場合があります。このような場合テスト領域を分割し、割り込み禁止で問題の RAM 領域のテストを実行することを推奨します。
  - この API は割り込みを使用しません。
  - デバイスに RAMP 機能がある場合, RAMP 機能を使用して確認します。

ソースファイル: SelfTest_ram_checker.s
ヘッダーファイル: SelfTest_ram_checker.h

関数名	
bool SelfTest_RAM_Checker(uint32_t start, uint32_t length, uint32_t *result_word)	
説明	
レジスターを退避します 入力パラメーター(start)の下位 2bit を 0 にします(4 の倍数に丸めます) 入力パラメーター(length)の下位 2bit を 0 にします(4 の倍数に丸めます) start で指定されている RAM アドレスへ 0x5555_5555 を書き込みます(*start = 0x55555555) start で指定されている RAM アドレスの値を読み出します 読み出した値と 0x5555_5555 を比較し不一致の場合, 出力パラメーター*result_word に stat で指定された RAM アドレスを書き込み, 失敗(false)を返しテストを終了します 一致した場合, start で指定されている RAM アドレスへ 0xAAAA_AAAA を書き込みます(*start = 0xAAAAAAAA) start で指定されている RAM アドレスの値を読み出します 読み出した値と 0xAAAA_AAAA を比較し不一致の場合, 出力パラメーター*result_word に stat で指定された RAM アドレスを書き込み, 失敗(false)を返しテストを終了します 一致した場合, start で指定された RAM アドレスをインクリメントします(start += 0x0000_00004) 入力パラメーター(length)をデクリメントします(length -= 0x0000_00004) length が 0 になるまで 0x5555_5555 と 0xAAAA_AAAA の比較を繰り返します length が 0 になるまで比較が一致した時, 成功(true)を返しテストを終了します	
入力パラメーター	
uint32_t start	テストを開始する RAM アドレス 4 の倍数の値に丸められます
uint32_t length	テスト範囲(バイト単位) 4 の倍数の値に丸められます
出力パラメーター	
uint32_t *result_word	結果を格納するメモリーバッファー 異常が検出されたアドレスを書き込む（正常の場合は何もしない）

戻り値	
bool	テスト結果(true: 成功、false: 失敗)

- テストのフローチャート



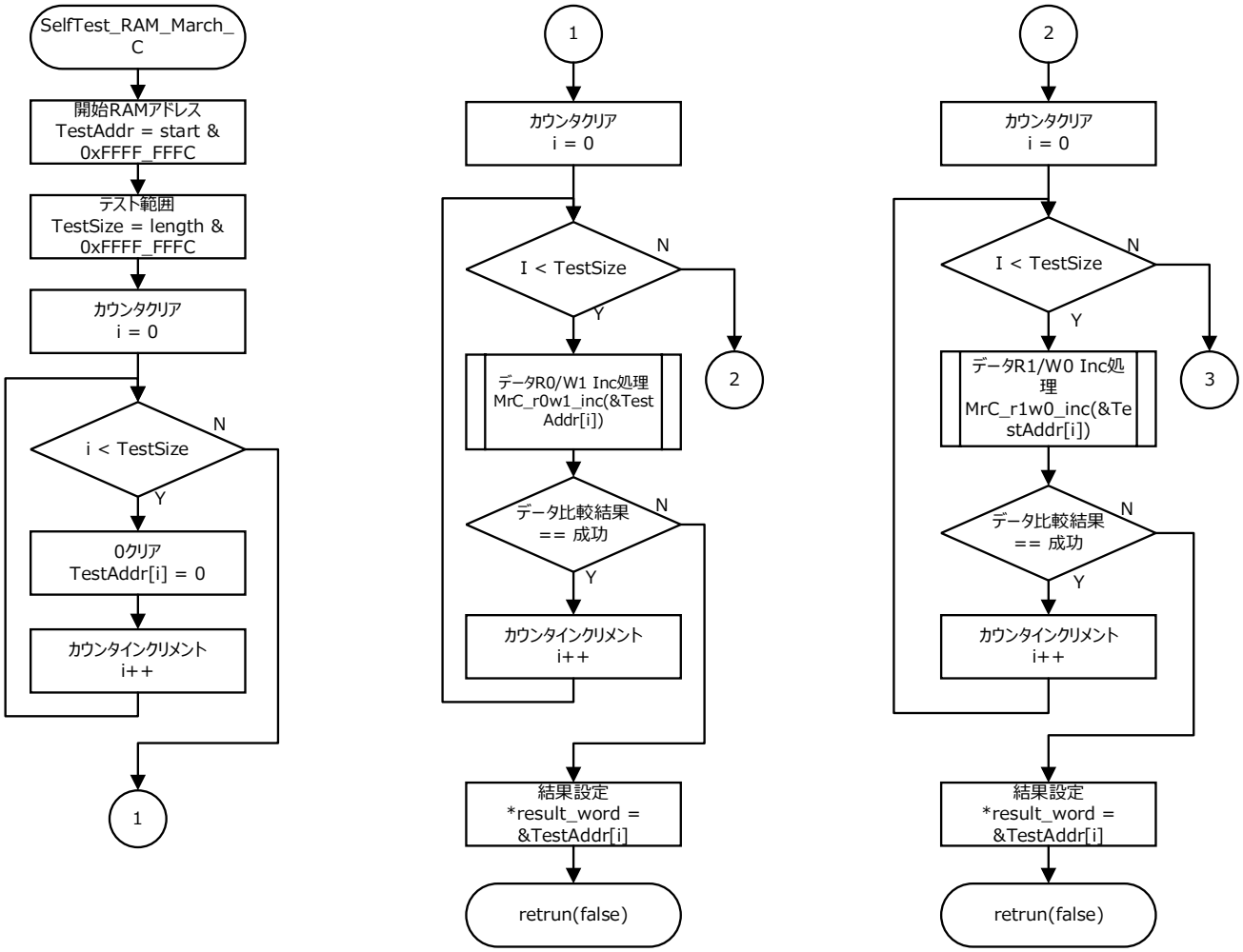
ソースファイル: SelfTest_ram_march_c.s
ヘッダーファイル: SelfTest_ram_march_c.h

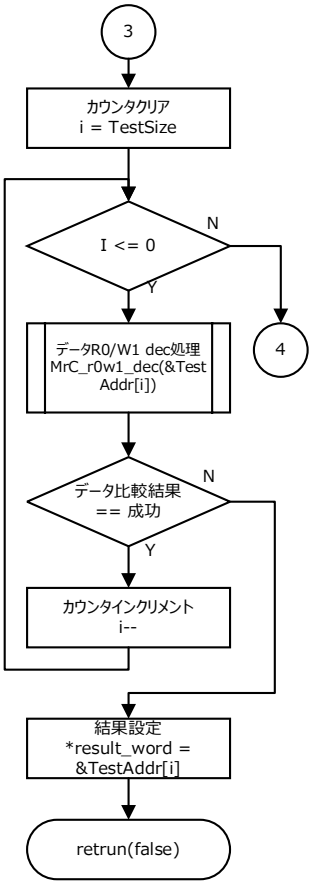
関数名
bool SelfTest_RAM_March_C(uint32_t start, uint32_t length, uint32_t *result_word)
説明
レジスターを退避します 入力パラメーター(start)の下位 2bit を 0 にします(4 の倍数に丸めます) 入力パラメーター(length)の下位 2bit を 0 にします(4 の倍数に丸めます) start で指定された RAM アドレスから length で指定されたサイズ分に 0x0000_0000 を書き込みます start で指定された RAM アドレスを読み出し 0x0000_0000 と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 一致した場合，RAM アドレスに 0x0000_0001 を書き込みます RAM アドレスを読み出し 0x0000_0001 と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 一致した場合，RAM アドレスに 0x0000_0003 を書き込みます RAM アドレスを読み出し 0x0000_0003 と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 一致した場合，RAM アドレスに 0x0000_0007 を書き込みます RAM アドレスを読み出し 0x0000_0007 と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 以降，1bit ずつ 32bit 分繰り返します 32bit 全て一致したとき，RAM アドレスをインクリメントします 以降，サイズ分繰り返します 全て一致したとき，RAM アドレスを読み出し 0x1111_1111 と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 一致した場合，RAM アドレスに 0xFFFF_FFFE を書き込みます RAM アドレスを読み出し 0xFFFF_FFFE と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 一致した場合，RAM アドレスに 0xFFFF_FFFC を書き込みます RAM アドレスを読み出し 0xFFFF_FFFC と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 一致した場合，RAM アドレスに 0xFFFF_FFF8 を書き込みます RAM アドレスを読み出し 0xFFFF_FFF8 と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 以降，1bit ずつ 32bit 分繰り返します 32bit 全て一致したとき，RAM アドレスをインクリメントします 以降，サイズ分繰り返します 全て一致したとき，RAM アドレス(start で指定された RAM アドレス+length で指定されたサイズ)の読み出し 0x0000_0000 と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します 一致した場合，RAM アドレスに 0x0000_0001 を書き込みます RAM アドレスを読み出し 0x0000_0001 と比較し不一致の場合，出力パラメーター*result_word に

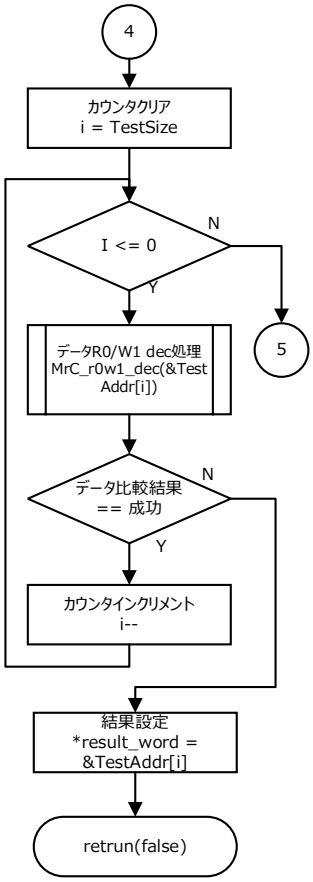
RAM アドレスを書き込み，失敗(false)を返しテストを終了します
一致した場合，RAM アドレスに 0x0000_0003 を書き込みます
RAM アドレスを読み出し 0x0000_0003 と比較し不一致の場合，出力パラメーター*result_word に
RAM アドレスを書き込み，失敗(false)を返しテストを終了します
一致した場合，RAM アドレスに 0x0000_0007 を書き込みます
RAM アドレスを読み出し 0x0000_0007 と比較し不一致の場合，出力パラメーター*result_word に
RAM アドレスを書き込み，失敗(false)を返しテストを終了します
以降，1bit ずつ 32bit 分繰り返します
32bit 全て一致したとき，RAM アドレスをデクリメントします
以降，サイズ分繰り返します
全て一致したとき，RAM アドレスを読み出し 0x1111_1111 と比較し不一致の場合，出力パラメーター*result_word に RAM アドレスを書き込み，失敗(false)を返しテストを終了します
一致した場合，RAM アドレスに 0xFFFF_FFFE を書き込みます
RAM アドレスを読み出し 0xFFFF_FFFE と比較し不一致の場合，出力パラメーター*result_word に
RAM アドレスを書き込み，失敗(false)を返しテストを終了します
一致した場合，RAM アドレスに 0xFFFF_FFFC を書き込みます
RAM アドレスを読み出し 0xFFFF_FFFC と比較し不一致の場合，出力パラメーター*result_word に
RAM アドレスを書き込み，失敗(false)を返しテストを終了します
一致した場合，RAM アドレスに 0xFFFF_FFF8 を書き込みます
RAM アドレスを読み出し 0xFFFF_FFF8 と比較し不一致の場合，出力パラメーター*result_word に
RAM アドレスを書き込み，失敗(false)を返しテストを終了します
以降，1bit ずつ 32bit 分繰り返します
32bit 全て一致したとき，RAM アドレスをデクリメントします
以降，サイズ分繰り返します
全て一致したとき，成功(true)を返しテストを終了します

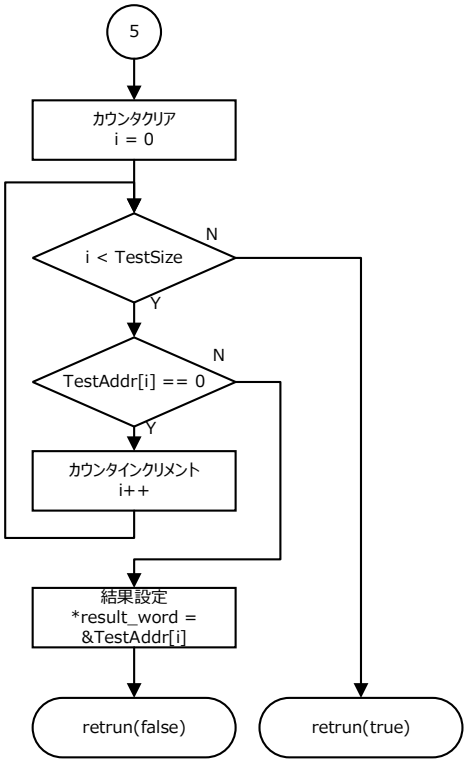
入力パラメーター	
uint32_t start	テストを開始する RAM アドレス 4 の倍数の値に丸められます
uint32_t length	テスト範囲(バイト単位) 4 の倍数の値に丸められます
出力パラメーター	
uint32_t *result_word	結果を格納するメモリーバッファー 異常が検出されたアドレスを書き込む（正常の場合は何もしない）
戻り値	
bool	テスト結果(true：成功、false：失敗)

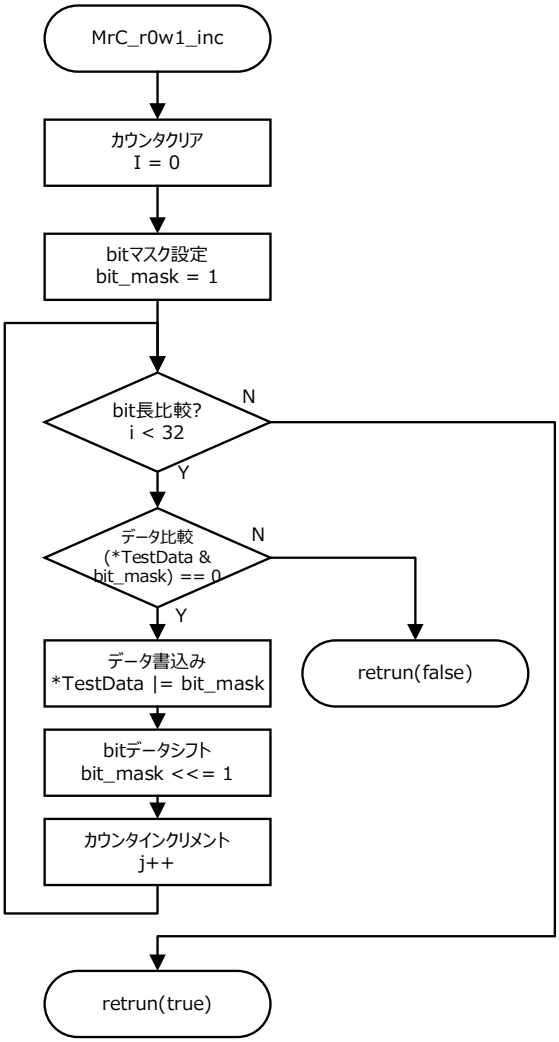
- テストのフローチャート

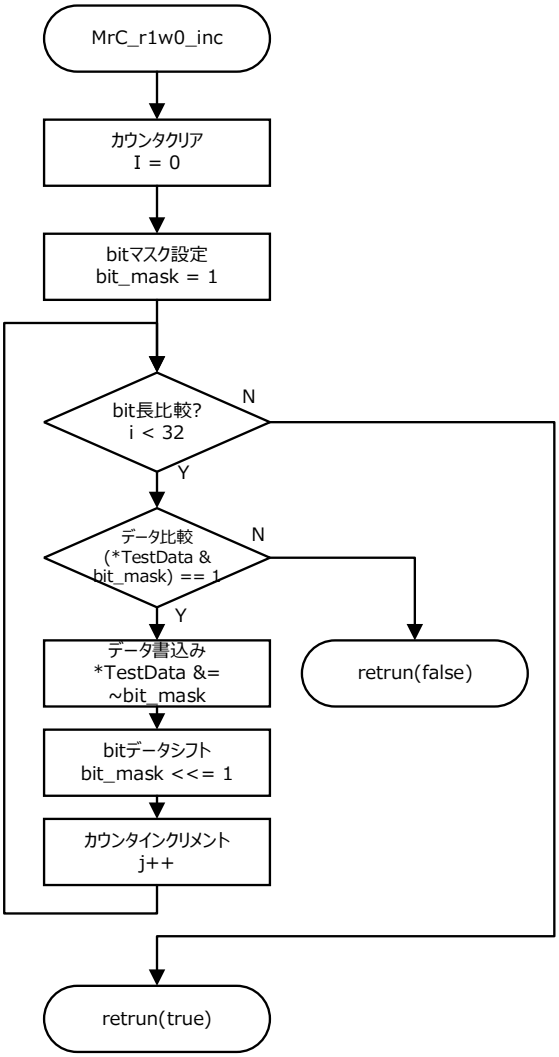


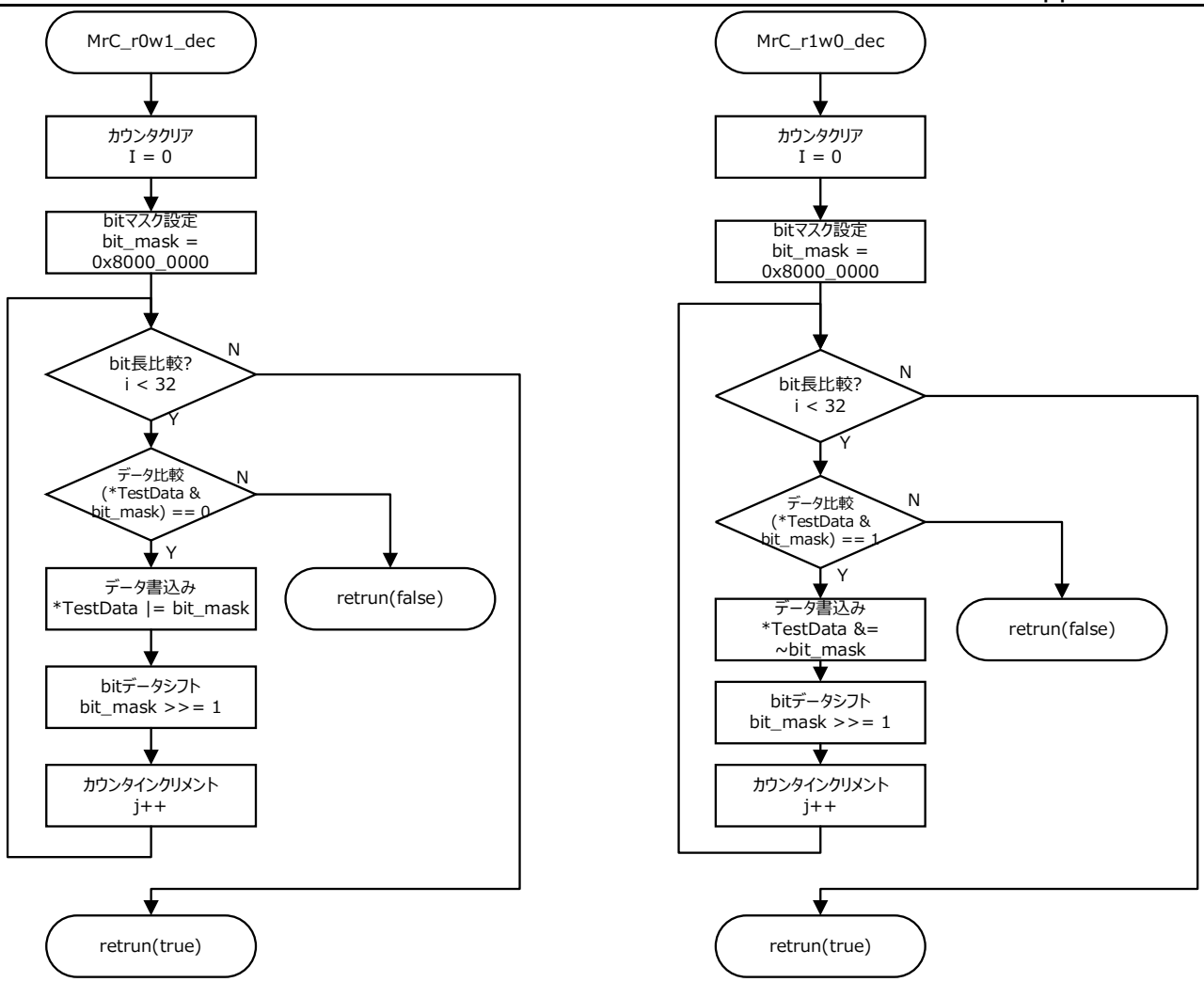

3
カウンタクリア
i = TestSize
I <= 0
N
Y
データR0/W1 dec処理
MrC_r0w1_dec(&Test
Addr[i])
4
データ比較結果
== 成功
N
Y
カウンタインクリメント
i--
結果設定
*result_word =
&TestAddr[i]
retrun(false)


4
カウンタクリア
i = TestSize
I <= 0
N
Y
データR0/W1 dec処理
MrC_r0w1_dec(&Test
Addr[i])
5
データ比較結果
== 成功
N
Y
カウンタインクリメント
i--
結果設定
*result_word =
&TestAddr[i]
retrun(false)


5
カウンタクリア
i = 0
i < TestSize
N
Y
TestAddr[i] == 0
N
Y
カウンタインクリメント
i++
結果設定
*result_word =
&TestAddr[i]
retrun(false)
retrun(true)


MrC_r0w1_inc
カウンタクリア
I = 0
bitマスク設定
bit_mask = 1
bit長比較?
i < 32
N
Y
データ比較
(*TestData &
bit_mask) == 0
N
Y
データ書込み
*TestData |= bit_mask
retrun(false)
bitデータシフト
bit_mask <<= 1
カウンタインクリメント
j++
retrun(true)

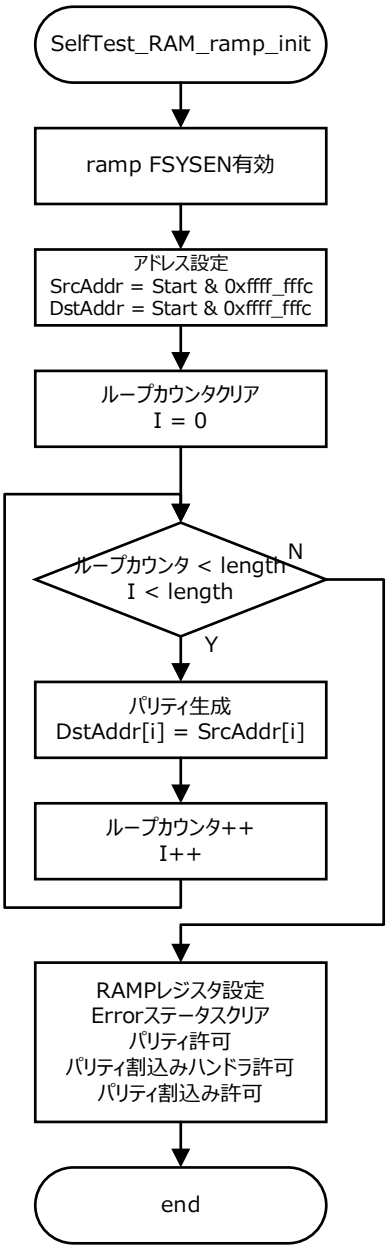

MrC_r1w0_inc
カウンタクリア
I = 0
bitマスク設定
bit_mask = 1
bit長比較?
i < 32
N
Y
データ比較
(*TestData &
bit_mask) == 1
N
Y
データ書込み
*TestData &=
~bit_mask
retrun(false)
bitデータシフト
bit_mask <<= 1
カウンタインクリメント
j++
retrun(true)


MrC_r0w1_dec
カウンタクリア
I = 0
bitマスク設定
bit_mask =
0x8000_0000
bit長比較?
i < 32
N
Y
データ比較
(*TestData &
bit_mask) == 0
N
Y
データ書込み
*TestData |= bit_mask
retrun(false)
bitデータシフト
bit_mask >>= 1
カウンタインクリメント
j++
retrun(true)
MrC_r1w0_dec
カウンタクリア
I = 0
bitマスク設定
bit_mask =
0x8000_0000
bit長比較?
i < 32
N
Y
データ比較
(*TestData &
bit_mask) == 1
N
Y
データ書込み
*TestData &=
~bit_mask
retrun(false)
bitデータシフト
bit_mask >>= 1
カウンタインクリメント
j++
retrun(true)

ソースファイル: SelfTest_ram_ramp.c
ヘッダーファイル: SelfTest_ram_ramp.h

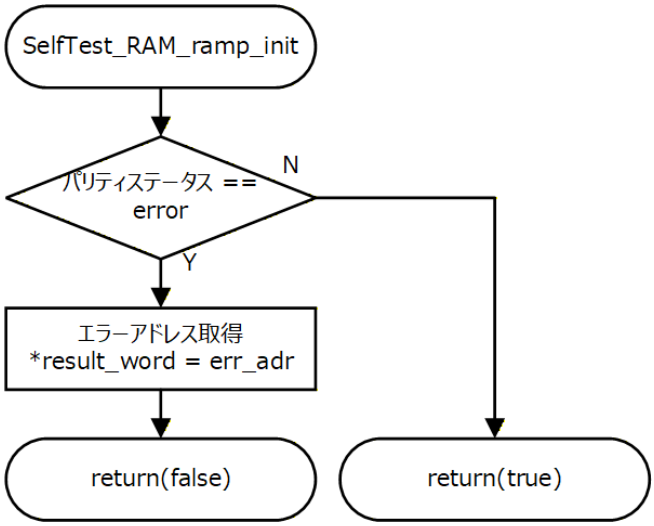
関数名	
void SelfTest_RAM_ramp_init(uint32_t start, uint32_t length, uint32_t)	
説明	
RAMparity 機能のクロック(fsys)を有効にします パリティーを生成します 入力パラメーター(start)で指定された RAM アドレスを読み出し, 同じアドレスに書き込みます RAM アドレスをインクリメントします 入力パラメータ(length)で指定されたサイズ分繰り返します RAMparity 機能ステータスをクリアします RAMparity 機能を有効にします RAMparity 割り込みハンドラーを許可します 異常を検知したとき, RAMparity 割り込みハンドラーへ Jump します	
入力パラメーター	
uint32_t start	テストを開始する RAM アドレス 4 の倍数の値に丸められます
uint32_t length	テスト範囲(バイト単位) 4 の倍数の値に丸められます
出力パラメーター	
---	---
戻り値	
---	---

- テストのフローチャート



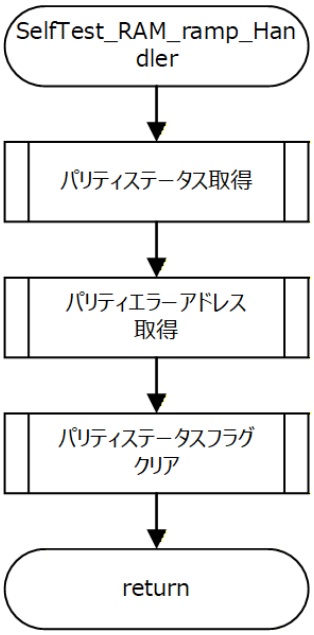
関数名	
bool SelfTest_RAM_ramp(*result_word)	
説明	
RAMparity 割り込みハンドラーで異常を検知したとき，出力パラメーター(*result_word)に異常が検出されたアドレスを書き込み失敗(false)を返しテストを終了します 異常が検知されなかった時，成功(true)を返しテストを終了します	
入力パラメーター	
---	---
---	---
出力パラメーター	
uint32_t *result_word	結果を格納するメモリーバッファー 異常が検出されたアドレスを書き込む（正常の場合は何もしない）
戻り値	
bool	テスト結果(true：成功、false：失敗)

- テストのフローチャート



関数名	
void SelfTest_RAM_ramp_Handler(void)	
説明	
RAMparity 割り込みハンドラーで異常を検知したとき, RAMparity 機能によりコールされます RAMparity 機能ステータスを確認します RAMparity 機能ステータスにエラー発生していた時, RAM パリティーエラーアドレスを SelfTest_RAM_ramp()の出力パラメーター(*result_word)に設定します RAMparity 機能ステータスをクリアし, 割り込みハンドラーを終了します	
入力パラメーター	
---	---
---	---
出力パラメーター	
---	---
戻り値	
---	---

- テストのフローチャート



3.2.7. FLASH テスト

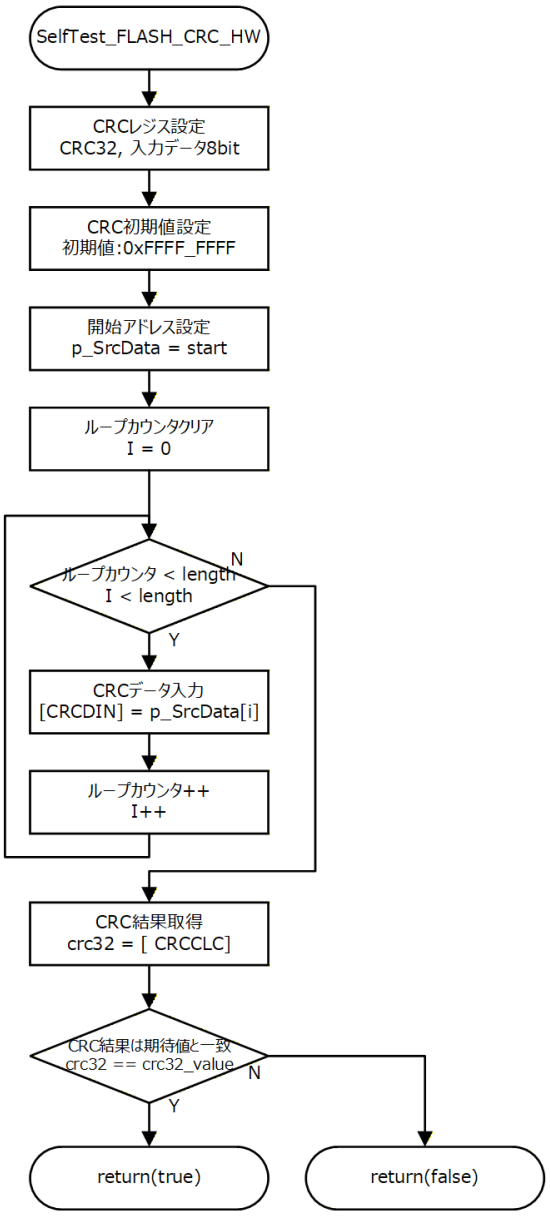
指定されたアドレス範囲の CRC32 の値を計算し、FLASH メモリー領域へ書き込んだ内容とは異なる値が読み出されないことを確認します。

- この API はテストを行う範囲の値チェックは行われず、結果任意の範囲を指定可能です。適切な範囲で呼び出されているかは呼び出し側で正しく指定する必要があります。
  - ➢ この API は割り込みを使用しません。
  - ➢ この API を利用するには、ある FLASH メモリー範囲の CRC32 の値を事前に求めておくことが必要です。
  - ➢ デバイスに CRC 機能がない場合は，ソフトウェア CRC で計算します。

ソースファイル: SelfTest_flash_crc_hw.c
ヘッダーファイル: SelfTest_flash_crc_hw.h

関数名	
bool SelfTest_FLASH_CRC_HW(uint32_t start, uint32_t length, uint32_t crc32_value)	
説明	
CRC 機能を初期化します CRC 形式(CRC32)，入力データ幅(8bit), CRC 初期値結果レジスター(0xFFFF_FFFF) 入力パラメーター(start)で指定された FLASH アドレスを読み出し，CRC 入力データレジスターへ書き込みます FLASH アドレスをインクリメントし，以降入力パラメーター(length)で指定されたサイズ分繰り返します サイズ分繰り返しが完了した時，CRC 初期値結果レジスターを読み出し，入力パラメーター(crc32_value)と比較し不一致の場合，失敗(false)を返しテストを終了します 成功の場合，成功(true)を返しテストを終了します	
入力パラメーター	
uint32_t start	テストを開始する FLASH アドレス 1 バイト単位の値を指定可能
uint32_t length	テスト範囲(バイト) 1 バイト単位の値を指定可能
uint32_t crc32_value	あらかじめ計算しておいたテスト範囲の CRC32 値
出力パラメーター	
なし	
戻り値	
bool	テスト結果(true：成功、false：失敗)

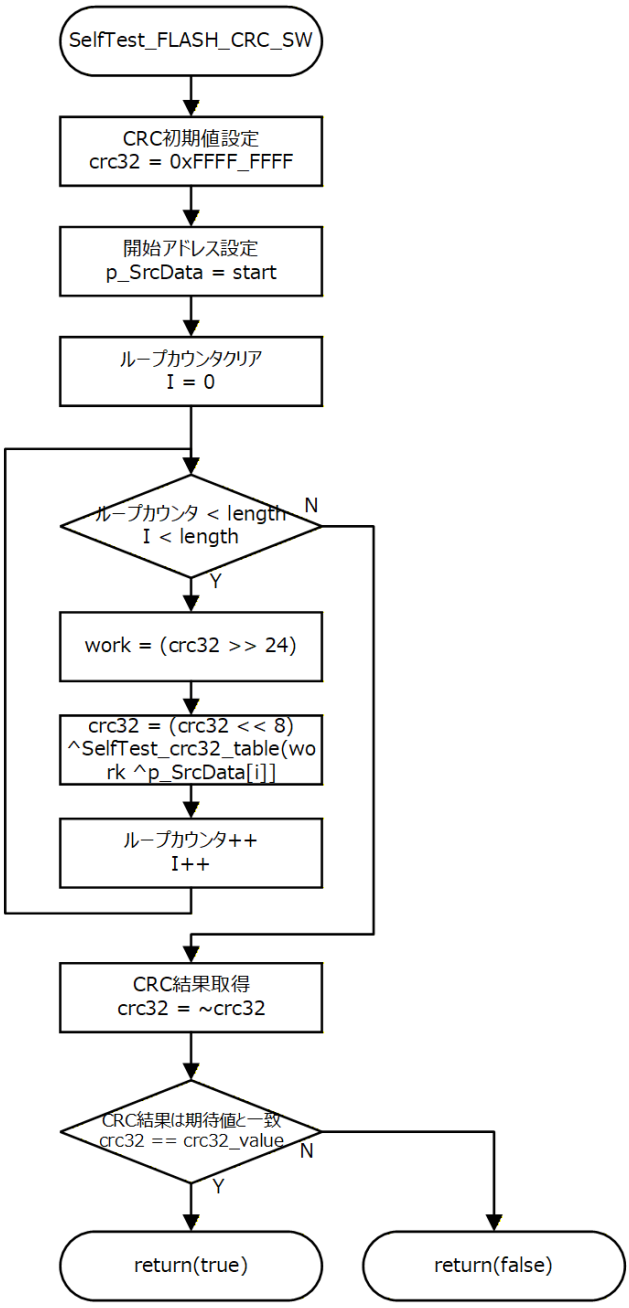
- テストのフローチャート



ソースファイル: SelfTest_flash_crc_sw.c
ヘッダーファイル: SelfTest_flash_crc_sw.h

関数名	
bool SelfTest_FLASH_CRC_SW(uint32_t start, uint32_t length, uint32_t crc32_value)	
説明	
変数 crc に初期値(0xFFFF_FFFF)を設定します 1byte ごとに計算します 　ワークエリア　= crc >> 24 　FLASH データ　=　入力パラメーター(start)で指定された FLASH アドレスを読み出します 　crc = CRC32 用参照テーブル[ワークエリア　^ FLASH データ] ^ (crc << 8) 　FLASH アドレスをインクリメントします 入力パラメーター(length)で指定されたサイズ分繰り返します サイズ分繰り返しが完了した時, CRC 初期値結果レジスターを読み出し, 入力パラメーター(crc32_value)と比較し不一致の場合, 失敗(false)を返しテストを終了します 成功の場合, 成功(true)を返しテストを終了します	
入力パラメーター	
uint32_t start	テストを開始する FLASH アドレス 1 バイト単位の値を指定可能
uint32_t length	テスト範囲(バイト) 1 バイト単位の値を指定可能
uint32_t crc32_value	あらかじめ計算しておいたテスト範囲の CRC32 値
出力パラメーター	
なし	
戻り値	
bool	テスト結果(true: 成功、false: 失敗)

- テストのフローチャート



CRC32 用参照テーブルを使用します

```
uint32_t SelfTest_crc32_table[256] = {
    0x00000000UL, 0x04C11DB7UL, 0x09823B6EUL, 0x0D4326D9UL,
    0x130476DCUL, 0x17C56B6BUL, 0x1A864DB2UL, 0x1E475005UL,
    0x2608EDB8UL, 0x22C9F00FUL, 0x2F8AD6D6UL, 0x2B4BCB61UL,
    0x350C9B64UL, 0x31CD86D3UL, 0x3C8EA00AUL, 0x384FBDBDUL,
    0x4C11DB70UL, 0x48D0C6C7UL, 0x4593E01EUL, 0x4152FDA9UL,
    0x5F15ADACUL, 0x5BD4B01BUL, 0x569796C2UL, 0x52568B75UL,
    0x6A1936C8UL, 0x6ED82B7FUL, 0x639B0DA6UL, 0x675A1011UL,
    0x791D4014UL, 0x7DDC5DA3UL, 0x709F7B7AUL, 0x745E66CDUL,
    0x9823B6E0UL, 0x9CE2AB57UL, 0x91A18D8EUL, 0x95609039UL,
    0x8B27C03CUL, 0x8FE6DD8BUL, 0x82A5FB52UL, 0x8664E6E5UL,
    0xBE2B5B58UL, 0xBAEA46EFUL, 0xB7A96036UL, 0xB3687D81UL,
    0xAD2F2D84UL, 0xA9EE3033UL, 0xA4AD16EAUL, 0xA06C0B5DUL,
    0xD4326D90UL, 0xD0F37027UL, 0xDDB056FEUL, 0xD9714B49UL,
    0xC7361B4CUL, 0xC3F706FBUL, 0xCEB42022UL, 0xCA753D95UL,
    0xF23A8028UL, 0xF6FB9D9FUL, 0xFBB8BB46UL, 0xFF79A6F1UL,
    0xE13EF6F4UL, 0xE5FFEB43UL, 0xE8BCCD9AUL, 0xEC7DD02DUL,
    0x34867077UL, 0x30476DC0UL, 0x3D044B19UL, 0x39C556AEUL,
    0x278206ABUL, 0x23431B1CUL, 0x2E003DC5UL, 0x2AC12072UL,
    0x128E9DCFUL, 0x164F8078UL, 0x1B0CA6A1UL, 0x1FCDBB16UL,
    0x018AEB13UL, 0x054BF6A4UL, 0x0808D07DUL, 0x0CC9CDCAUL,
    0x7897AB07UL, 0x7C56B6B0UL, 0x71159069UL, 0x75D48DDEUL,
    0x6B93DDDBUL, 0x6F52C06CUL, 0x6211E6B5UL, 0x66D0FB02UL,
    0x5E9F46BFUL, 0x5A5E5B08UL, 0x571D7DD1UL, 0x53DC6066UL,
    0x4D9B3063UL, 0x495A2DD4UL, 0x44190B0DUL, 0x40D816BAUL,
    0xACA5C697UL, 0xA864DB20UL, 0xA527FDF9UL, 0xA1E6E04EUL,
    0xBFA1B04BUL, 0xBB60ADFCUL, 0xB6238B25UL, 0xB2E29692UL,
    0x8AAD2B2FUL, 0x8E6C3698UL, 0x832F1041UL, 0x87EE0DF6UL,
    0x99A95DF3UL, 0x9D684044UL, 0x902B669DUL, 0x94EA7B2AUL,
    0xE0B41DE7UL, 0xE4750050UL, 0xE9362689UL, 0xEDF73B3EUL,
    0xF3B06B3BUL, 0xF771768CUL, 0xFA325055UL, 0xFEF34DE2UL,
    0xC6BCF05FUL, 0xC27DEDE8UL, 0xCF3ECB31UL, 0xCBFFD686UL,
    0xD5B88683UL, 0xD1799B34UL, 0xDC3ABDEDUL, 0xD8FBA05AUL,
    0x690CE0EEUL, 0x6DCDFD59UL, 0x608EDB80UL, 0x644FC637UL,
    0x7A089632UL, 0x7EC98B85UL, 0x738AAD5CUL, 0x774BB0EBUL,
    0x4F040D56UL, 0x4BC510E1UL, 0x46863638UL, 0x42472B8FUL,
    0x5C007B8AUL, 0x58C1663DUL, 0x558240E4UL, 0x51435D53UL,
    0x251D3B9EUL, 0x21DC2629UL, 0x2C9F00F0UL, 0x285E1D47UL,
    0x36194D42UL, 0x32D850F5UL, 0x3F9B762CUL, 0x3B5A6B9BUL,
    0x0315D626UL, 0x07D4CB91UL, 0x0A97ED48UL, 0x0E56F0FFUL,
```

```
    0x1011A0FAUL, 0x14D0BD4DUL, 0x19939B94UL, 0x1D528623UL,
    0xF12F560EUL, 0xF5EE4BB9UL, 0xF8AD6D60UL, 0xFC6C70D7UL,
    0xE22B20D2UL, 0xE6EA3D65UL, 0xEBA91BBCUL, 0xEF68060BUL,
    0xD727BBB6UL, 0xD3E6A601UL, 0xDEA580D8UL, 0xDA649D6FUL,
    0xC423CD6AUL, 0xC0E2D0DDUL, 0xCDA1F604UL, 0xC960EBB3UL,
    0xBD3E8D7EUL, 0xB9FF90C9UL, 0xB4BCB610UL, 0xB07DABA7UL,
    0xAE3AFBA2UL, 0xAAFBE615UL, 0xA7B8C0CCUL, 0xA379DD7BUL,
    0x9B3660C6UL, 0x9FF77D71UL, 0x92B45BA8UL, 0x9675461FUL,
    0x8832161AUL, 0x8CF30BADUL, 0x81B02D74UL, 0x857130C3UL,
    0x5D8A9099UL, 0x594B8D2EUL, 0x5408ABF7UL, 0x50C9B640UL,
    0x4E8EE645UL, 0x4A4FFBF2UL, 0x470CDD2BUL, 0x43CDC09CUL,
    0x7B827D21UL, 0x7F436096UL, 0x7200464FUL, 0x76C15BF8UL,
    0x68860BFDUL, 0x6C47164AUL, 0x61043093UL, 0x65C52D24UL,
    0x119B4BE9UL, 0x155A565EUL, 0x18197087UL, 0x1CD86D30UL,
    0x029F3D35UL, 0x065E2082UL, 0x0B1D065BUL, 0x0FDC1BECUL,
    0x3793A651UL, 0x3352BBE6UL, 0x3E119D3FUL, 0x3AD08088UL,
    0x2497D08DUL, 0x2056CD3AUL, 0x2D15EBE3UL, 0x29D4F654UL,
    0xC5A92679UL, 0xC1683BCEUL, 0xCC2B1D17UL, 0xC8EA00A0UL,
    0xD6AD50A5UL, 0xD26C4D12UL, 0xDF2F6BCBUL, 0xDBEE767CUL,
    0xE3A1CBC1UL, 0xE760D676UL, 0xEA23F0AFUL, 0xEEE2ED18UL,
    0xF0A5BD1DUL, 0xF464A0AAUL, 0xF9278673UL, 0xFDE69BC4UL,
    0x89B8FD09UL, 0x8D79E0BEUL, 0x803AC667UL, 0x84FBDBD0UL,
    0x9ABC8BD5UL, 0x9E7D9662UL, 0x933EB0BBUL, 0x97FFAD0CUL,
    0xAFB010B1UL, 0xAB710D06UL, 0xA6322BDFUL, 0xA2F33668UL,
    0xBCB4666DUL, 0xB8757BDAUL, 0xB5365D03UL, 0xB1F740B4UL,
};
```

3.2.8. Digital 入出力

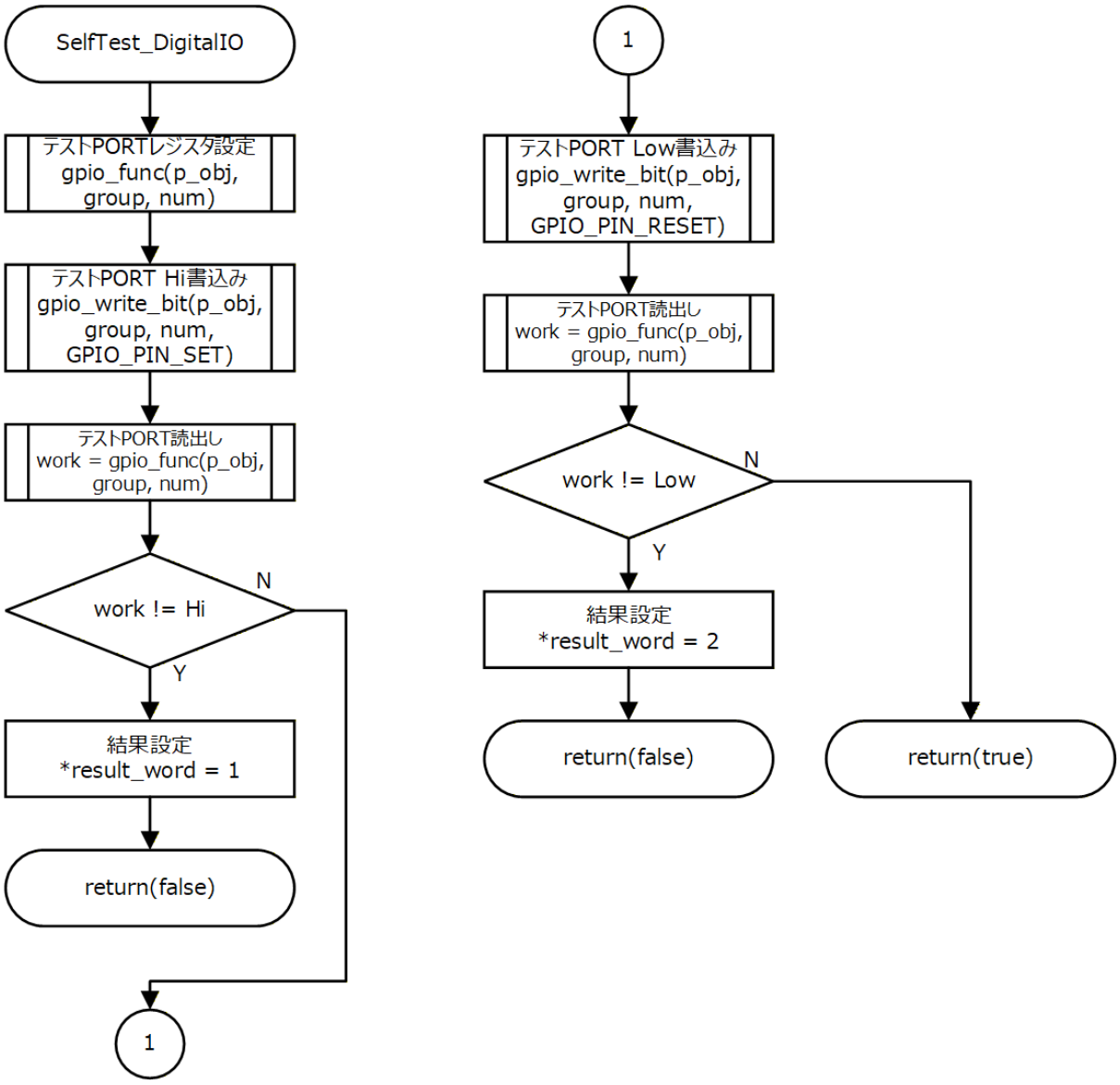
任意ポートを利用して GPIO から Digital データを出力し、その後値を読み込みます。これを 0 出力および 1 出力に対して行います。

- テスト後は任意ポートの GPIO 設定が変更されますので、アプリケーション起動前にこのテストを行うか、テスト後に再度 GPIO 設定を行うようにしてください。
  - この API は割り込みを使用しません。
  - 任意ポートはユーザーで指定できるようにします。

ソースファイル: SelfTest_digital_io.c
ヘッダーファイル: SelfTest_digital_io.h

関数名	
bool SelfTest_DigitalIO(DigitalIO_t port)	
説明	
定数定義によりポートグループ, ポート番号, ポート機能を指定します 指定されたポートの入力, 出力を許可します 指定されたポートへ'1'出力します 指定されたポートの入力レジスターを読み取り'1'と比較し不一致の場合, 出力パラメーター(*result_word)の bit0 に 1 を設定して失敗(false)を返しテストを終了します 指定されたポートへ'0'出力します 指定されたポートの入力レジスターを読み取り'0'と比較し不一致の場合, 出力パラメーター(*result_word)の bit1 に 1 を設定して失敗(false)を返しテストを終了します 成功したとき, 成功(true)を返しテストを終了します 入力パラメーター, 出力パラメーターは引数 DigitalIO_t 構造体のメンバーを示します	
入力パラメーター	
gpio_t *p_obj	テストするポートインスタンス
gpio_gr_t group	テストするポートグループ
gpio_num_t num	テストするポート番号
出力パラメーター	
uint32_t *result_word	エラー発生時には以下のいずれかの bit がセットされる bit0：'1'出力時にエラー bit1：'0'出力時にエラー いずれかのテストでエラーが検出された場合、テストは中断されそれ以降のテストは実施されません。
戻り値	
Bool	結果(true：成功、false：失敗)

- テストのフローチャート



3.2.9. Analog 入力テスト

テスト機能付きの ADC チャネルを利用し、入力検証用の 3 種類の入力電圧(VREFL/VREFH/REGOUT1)を発生させ、AD 変換で取得される値をテストします。

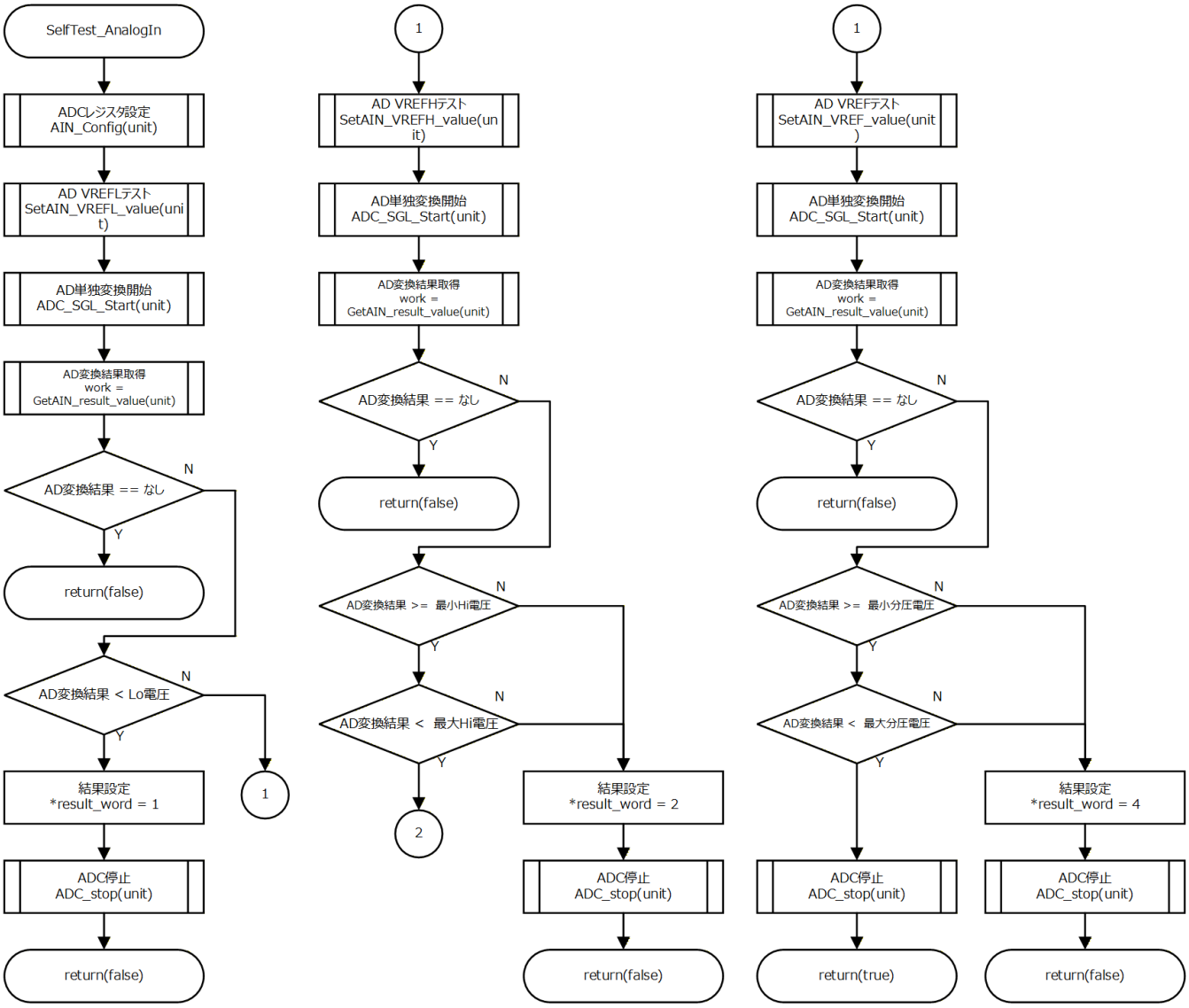
- この API は割り込みを利用しません。
- テスト機能とは自己診断支援機能を有したハードウェアです。

ソースファイル: SelfTest_analog_in.c
ヘッダーファイル: SelfTest_analog_in.h

関数名	
bool SelfTest_AnalogIn(uint32_t unit, uint32_t *result_word)	
説明	
ADC 機能を初期化します Lo 電圧テストを実行します VREFL 接続 AD チャネル初期設定 ADC 単独変換開始 VREFL 接続 AD チャネル変換結果取得 変換結果と Lo 電圧を比較し Lo 電圧以下の時, ADC 変換を停止, 出力パラメーター(*result_word)の bit0 に 1 を設定して失敗(false)を返しテストを終了します Lo 電圧以上の時, Hi 電圧テストを実行します VREFH 接続 AD チャネル初期設定 ADC 単独変換開始 VREFH 接続 AD チャネル変換結果取得 変換結果と Hi 電圧を比較し Hi 電圧以上の時, ADC 変換を停止,出力パラメーター(*result_word)の bit1 に 1 を設定して失敗(false)を返しテストを終了します Hi 電圧以下の時, 分圧電圧テストを実行します リファレンス電源接続 AD チャネル初期設定 ADC 単独変換開始 リファレンス電源接続 AD チャネル変換結果取得 変換結果と分圧電圧を比較し分圧電圧+10%以上または分圧電圧-10%以下の時, ADC 変換を停止, 出力パラメーター(*result_word)の bit2 に 1 を設定して失敗(false)を返しテストを終了します 全て成功した時, ADC 変換を停止, 成功(true)を返しテストを終了します	
入力パラメーター	
uint32_t unit	テストする ADC のユニット名 UNIT_A, UNIT_B, UNIT_C 上記以外の場合は, エラーとします
出力パラメーター	
uint32_t *result_word	エラー発生時には以下のいずれかの bit がセットされる bit0：Lo 電圧テスト時にエラー bit1：Hi 電圧テスト時にエラー bit2： 分圧電圧テスト時にエラー いずれかのテストでエラーが検出された場合､テストは中断されそれ以降のテストは実施されません。
戻り値	

Bool	結果(true：成功、false：失敗)

- テストのフローチャート



4. 改訂履歴

Version	日付	変更内容
1.0	2026/04/01	初版

製品取り扱い上のお願い

株式会社東芝およびその子会社ならびに関係会社を以下「当社」といいます。
本資料に掲載されているハードウエア、ソフトウエアおよびシステムを以下「本製品」といいます。

- 本製品に関する情報等、本資料の掲載内容は、技術の進歩などにより予告なしに変更されることがあります。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。また、文書による当社の事前の承諾を得て本資料を転載複製する場合でも、記載内容に一切変更を加えたり、削除したりしないでください。
- 当社は品質、信頼性の向上に努めていますが、半導体・ストレージ製品は一般に誤作動または故障する場合があります。本製品をご使用頂く場合は、本製品の誤作動や故障により生命・身体・財産が侵害されることのないように、お客様の責任において、お客様のハードウエア・ソフトウエア・システムに必要な安全設計を行うことをお願いします。なお、設計および使用に際しては、本製品に関する最新の情報（本資料、仕様書、データシート、アプリケーションノート、半導体信頼性ハンドブックなど）および本製品が使用される機器の取扱説明書、操作説明書などをご確認の上、これに従ってください。また、上記資料などに記載の製品データ、図、表などに示す技術的な内容、プログラム、アルゴリズムその他応用回路例などの情報を使用する場合は、お客様の製品単独およびシステム全体で十分に評価し、お客様の責任において適用可否を判断してください。
- 本製品は、特別に高い品質・信頼性が要求され、またはその故障や誤作動が生命・身体に危害を及ぼす恐れ、膨大な財産損害を引き起こす恐れ、もしくは社会に深刻な影響を及ぼす恐れのある機器（以下“特定用途”という）に使用されることは意図されていませんし、保証もされていません。特定用途には原子力関連機器、航空・宇宙機器、医療機器（生命直結機器）、車載・輸送機器、防衛関連機器などが含まれますが、本資料に個別に記載する用途は除きます。特定用途に使用された場合には、当社は一切の責任を負いません。なお、詳細は当社営業窓口まで、または当社 Web サイトのお問い合わせフォームからお問い合わせください。
- 本製品を、国内外の法令、規則及び命令により、製造、使用、販売を禁止されている製品に使用することはできません。
- 本資料に掲載してある技術情報は、製品の代表的動作・応用を説明するためのもので、その使用に際して当社及び第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。
- 別途、書面による契約またはお客様と当社が合意した仕様書がない限り、当社は、本製品および技術情報に関して、明示的にも黙示的にも一切の保証（機能動作の保証、商品性の保証、特定目的への合致の保証、情報の正確性の保証、第三者の権利の非侵害保証を含むがこれに限らない。）をしておりません。
- 本製品、または本資料に掲載されている技術情報を、大量破壊兵器の開発等の目的、軍事利用の目的、あるいはその他軍事用途の目的で使用しないでください。また、輸出に際しては、「外国為替及び外国貿易法」、「米国輸出管理規則」等、適用ある輸出関連法令を遵守し、それらの定めるところにより必要な手続を行ってください。
- 本製品の RoHS 適合性など、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い合わせください。本製品のご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用ある環境関連法令を十分調査の上、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は一切の責任を負いかねます。

東芝デバイス&ストレージ株式会社
https://toshiba.semicon-storage.com/jp/