EPSON
EXCEED YOUR VISION

IEC60730 規格準拠のための自己診断サンプルプログラムおよび参考情報

（S1C17803 用 / S1C17 Family に適用可能）

ダウンロードした圧縮ファイルに同梱の“ご注意.pdf”に従ってご使用ください。

セイコーエプソン株式会社

Rev.1.0

本資料のご使用につきましては、次の点にご留意願います。

本資料の内容については、予告無く変更することがあります。

1. 本資料の一部、または全部を弊社に無断で転載、または、複製など他の目的に使用することは堅くお断りいたします。

2. 本資料に掲載される応用回路、プログラム、使用方法等はあくまでも参考情報であり、これら起因する第三者の知的財産権およびその他の権利侵害あるいは損害の発生に対し、弊社はいかなる保証を行うものではありません。また、本資料によって第三者または弊社の知的財産権およびその他の権利の実施権の許諾を行うものではありません。

3. 特性値の数値の大小は、数直線上の大小関係で表しています。

4. 本資料に掲載されている製品のうち「外国為替及び外国貿易法」に定める戦略物資に該当するものについては、輸出する場合、同法に基づく輸出許可が必要です。

5. 本資料に掲載されている製品は、生命維持装置その他、きわめて高い信頼性が要求される用途を前提としていません。よって、弊社は本（当該）製品をこれらの用途に用いた場合のいかなる責任についても負いかねます。

6. 本資料に掲載されている会社名、商品名は、各社の商標または登録商標です。

©SEIKO EPSON CORPORATION 2009, All rights reserved.

目　次

1.　概要 1

2.　サンプル 2

2.1　構成 2

2.2　操作手順 2

2.3　サンプルプログラム 3

2.3.1　c17SelfTest.hの定義一覧 3
2.3.2　RAMのR/Wテスト(16 ビットデバイス用) 3
2.3.3　RAMのR/Wテスト(8 ビットデバイス用) 4
2.3.4　レジスタのR/Wテスト 4
2.3.5　ステータスレジスタのR/Wテスト 5
2.3.6　RAMのMarch-Cテスト 7
2.3.7　チェックサム 8
2.3.8　CRC(計算) 8
2.3.9　CRC(テーブル参照) 9
2.3.10　割り込みテスト 9
2.3.11　メインクロック安定性確認テスト 10

3　参考情報 13

3.1　外部インタフェースによる通信異常診断 13

改訂履歴表 14

1. 概要

国際電気標準会議（IEC:International Electrotechnical Commission）は家電機器開発のための IEC60730 指令を発表しています。この安全規格は、欧州で販売・使用される家電機器に対して準拠が義務付けられています。この規格は、最終製品に対して定期的な自己診断を実施することで、故障や不具合を早期に発見し、これによって生じる危険から消費者を守ることを目的としています。

マイクロコントローラ関係のソフトウェア制御は以下の規格によって分類されます。

Class A: 設備の安全性の維持を目的としない制御
（照明器具等の制御）

Class B: 制御する設備の危険な操作を防ぐことを目的とする制御
（洗濯機、冷蔵庫、冷凍庫、食器洗い機等の制御）

Class C: 特別な危険を防ぐことを目的とする制御（燃焼器具等の制御）

これらの Class のうち、家電製品を制御するソフトウェアのほとんどは Class B に分類され、最終製品に対し以下の自己診断テストを実施することが推奨対策とされています。

・マイコンやプログラムカウンタのスタック故障診断

・割り込み周期異常診断

・マイコンの動作クロック周波数の異常診断

・ROM/RAM メモリの異常診断

・外部インタフェースによる通信異常診断

より詳しい情報は、IEC60730 の Annex H を参照してください。

このドキュメントは、セイコーエプソン製 S1C17 Family 対してこれらの自己診断テストを行うサンプルソフトウェアおよび参考情報について記載しています。自己診断テスト用サンプルソフトウェアは、S1C17803 上で動作するよう作成していますが、その他の S1C17 Family の機種においてもこのサンプルプログラムの一部を流用することで、同じテストを実施することが可能です。

自己診断の内容は、

・メモリの故障診断テスト（Read Write テスト、March-C テスト）

・メモリに置かれたデータの確からしさテスト（チェックサム生成、CRC 生成）

・割り込みテスト（割り込み周期、割り込み回数確認）

・メインクロック安定性確認テスト（動作周波数確認）

です。

Read Write テストおよび March-C テストでは、指定範囲のメモリ、レジスタ、スタックポインタ、そしてステータスレジスタについて書き込み、読み出しテストを実施します。

チェックサム、CRC 生成では指定範囲のメモリに置かれたデータについて、それぞれ誤り検出符号の値を求め、返します。

割り込みテストは、一定周期間に何回割り込みが発生したかをカウントし、その値を返します。

メインクロック安定性確認テストでは、サブクロック（32KHz）を用いてメインクロックが正常な動作周波数で動作しているかを確認します。

2. サンプル

2.1 構成

selftest フォルダ以下のファイルは下記のとおりです。

表 2.1 サンプルファイル一覧

ファイル名	内容
boot.c	スタートアップモジュール
vector.c	ベクタテーブルの定義
main.c	テスト関数をコール
c17TestRam16.c	RAM の R/W テスト(16 ビットデバイス用)
c17TestRam8.c	RAM の R/W テスト(8 ビットデバイス用)
c17TestRegister.s	汎用レジスタ、スタックポインタの R/W テスト
c17TestPsr.c	ステータスレジスタの R/W テスト
c17RwPsr.c	ステータスレジスタの R/W テストを行うために発生させるソフトウェア例外
c17TestRamMarchc.c	RAM の March-c テスト
c17TestChksum.c	チェックサムの算出
c17TestCrc.c	CRC の算出（計算）
c17TestCrcTbl.c	CRC の算出（テーブルを参照）
c17803TestIntterupt.c	割り込みのテスト
c17803TestClk.c	メインクロック安定性のテスト
vector.h	Vector のヘッダファイル
c17SelfTest.h	自己診断サンプルプログラムで用いられるヘッダファイル
s1c17803.h	標準 I/O 定義ファイル
selftest_gnu17IDE.mak	メイクファイル
selftest_gnu17IDE.lds	リンカスクリプトファイル
selftest_gnu17IDE.par	パラメータ設定ファイル
selftest_gnu17IDE.cmd	GDB コマンドファイル
GDB17 Launch for selftest.launch	GDB 機動ランチャーファイル
その他ファイル	その他プロジェクト関連ファイル

2.2 操作手順

プロジェクトのインポート
(1)GNU17 IDE を起動して、「selftest」プロジェクトをインポートしてください。
※インポートの方法は S5U1C17001C Manual “3.ソフトウェア開発” を参照してください。

ビルド
(1) GNU17 IDE を使用して「selftest」プロジェクトをビルドしてください。
(2)S5U1C17803T (SVT17803 CPU Board)、SVT17 ICD BOARD または S5U1C33001H、USB ケーブル、PC を接続してください。
(3) SVT17 ICD BOARD または S5U1C33001H と SVT33L17 CPU Board をリセットしてください。

実行
(1) GNU17 IDE の［Run］->［External Tools］->［External Tools Configurations］をクリックします。
(2) Program の中から［GDB17 Launch for selftest］を選択し、［Run］を実行します。

2.3 サンプルプログラム

提供するサンプルプログラムについて記します。

2.3.1 c17SelfTest.h の定義一覧

表 2.3 c17SelfTest.h の定義一覧

定義名	値	内容
E_OK	0	テスト結果が良好なことを示す。
E_MEMORY	1	メモリテストで問題があったことを示す。
E_REGISTER	2	レジスタのテストで問題があったことを示す。
E_CLOCK	3	メインクロックのテストで問題があったことを示す。

2.3.2 RAM の R/W テスト(16 ビットデバイス用)

指定された RAM エリアに対して、読み書きのテストを行います。

関数名	c17TestRam16
引数 1	volatile unsigned short *chkAddr
引数 2	unsigned short chkNum
戻り値	short （E_OK または E_MEMORY）
関数概要	chkAddr が指すアドレスから chkNum の数だけ、メモリに 0xaa55（0x55aa）を書き込む。その後、読み込みを行い、書き込んだ値と比較する。差異がなければ E_OK、差異があれば E_MEMORY を返す。
注意事項	chkAddr の最下位ビットは常に 0 として扱われる。 指定されたメモリ領域がスタック領域と重複したとき、動作は保障されない。 c17TestRam16 は chkNum×2 バイト分のメモリチェックを行う。

処理フロー

テストの流れ	1. chkAddr と 0xfffffe との論理積をとる。 2. chkAddr から chkNum 回数分のテストを行うまで(1)から(7)を繰り返す。 (1) chkAddr が指すメモリの値をバックアップする。 (2) chkAddr が指すメモリに 0xaa55 を書き込む。 (3) (2)で書き込んだメモリの値が 0xaa55 となっているかをチェックする。 ◆0xaa55 でなかった場合 （i） chkAddr が指すメモリの値をバックアップした値に戻す。 （ii） E_MEMORY を返す。終了。 (4) chkAddr が指すメモリに 0x55aa を書き込む。 (5) (4)で書き込んだメモリの値が 0x55aa となっているかをチェックする。 ◆0x55aa でなかった場合 （i） chkAddr が指すメモリの値をバックアップした値に戻す。 （ii） E_MEMORY を返す。終了。 (6) chkAddr が指すメモリの値をバックアップした値に戻す。 (7) chkAddr を次のメモリアドレスへインクリメントする。 3. E_OK を返す。終了。

プログラムソース

c17TestRam16.c

2.3.3 RAM の R/W テスト(8 ビットデバイス用)

関数名	c17TestRam8
引数 1	volatile unsigned char *chkAddr
引数 2	unsigned short chkNum
戻り値	short （E_OK または E_MEMORY）
関数概要	chkAddr が指すアドレスから chkNum の数だけ、メモリに 0xa5（0x5a）を書き込む。その後、読み込みを行い、書き込んだ値と比較する。差異がなければ E_OK 、差異があれば E_MEMORY を返す。
注意事項	指定されたメモリ領域がスタック領域と重複したとき、動作は保障されない。 c17TestRam8 は chkNum×1 バイト分のメモリチェックを行う。

処理フロー

テストの流れ	1. chkAddr から chkNum 分のテストを行うまで(1)から(7)を繰り返す。 (1) chkAddr が指すメモリの値をバックアップする。 (2) chkAddr が指すメモリに 0xa5 を書き込む。 (3) (2)で書き込んだメモリの値が 0xa5 となっているかをチェックする。 ◆0xa5 でなかった場合 (i) chkAddr が指すメモリの値をバックアップした値に戻す。 (ii) E_MEMORY を返す。終了。 (4) chkAddr が指すメモリに 0x5a を書き込む。 (5) (4)で書き込んだメモリの値が 0x5a となっているかをチェックする。 ◆0x5a でなかった場合 (i) chkAddr が指すメモリの値をバックアップした値に戻す。 (ii) E_MEMORY を返す。終了。 (6) chkAddr が指すメモリの値をバックアップした値に戻す。 (7) chkAddr を次のメモリアドレスへインクリメントする。 2. E_OK を返す。終了。

プログラムソース

c17TestRam8.c

2.3.4 レジスタの R/W テスト

CPU 内の汎用レジスタ、スタックポインタに対して、読み書きのテストを行います。

関数名	c17TestRegister
引数	void
戻り値	short （E_OK または E_REGISTER）
関数概要	CPU 内の汎用レジスタに 0x555555 と 0xaaaaaa を書き込む。その後、読み込みを行い、書き込んだ値と比較する。同様にスタックポインタに 0x555554 と 0xaaaaa8 を書き込みテストする。差異がなければ E_OK、差異があれば E_REGISTER を返す。

処理フロー

テストの流れ	1. r4 から r7 レジスタの値をバックアップする。 2. r0 から r7 レジスタに 0x555555 を書き込む。 3. それぞれレジスタの値を比較する。 (1) r0 と 0x555555 を比較する。 (2) r0 と r1 を比較する。 (3) r0 と r2 を比較する。 (4) r0 と r3 を比較する。 (5) r0 と r4 を比較する。

	(6) r0 と r5 を比較する。 (7) r0 と r6 を比較する。 (8) r0 と r7 を比較する。 ◆上記(1)から(8)で差異があれば、E_REGISTER を返す。終了。 4. r0 から r7 レジスタに 0xaaaaaa を書き込む。 5. それぞれのレジスタの値を比較する。 (1) r0 と 0xaaaaaa を比較する。 (2) r0 と r1 を比較する。 (3) r0 と r2 を比較する。 (4) r0 と r3 を比較する。 (5) r0 と r4 を比較する。 (6) r0 と r5 を比較する。 (7) r0 と r6 を比較する。 (8) r0 と r7 を比較する。 ◆上記(1)から(8)で差異があれば、E_REGISTER を返す。終了。 6. スタックポインタをバックアップする。 7. スタックポインタに 0x555554 を書き込む。 8. スタックポインタの値と 0x555554 を比較する。 ◆異なる値だった場合 (1) スタックポインタをバックアップした値に戻す。 (2) E_REGISTER を返す。終了。 9. スタックポインタに 0xaaaaa8 を書き込む。 10. スタックポインタの値と 0xaaaaa8 を比較する。 ◆異なる値だった場合 (1) スタックポインタをバックアップした値に戻す。 (2) E_REGISTER を返す。終了。 11. スタックポインタの値をバックアップした値に戻す。 12. r4 から r7 レジスタの値をバックアップした値に戻す。 13. E_OK を返す。終了。

プログラムソース

c17TestRegister.s

2.3.5 ステータスレジスタの R/W テスト

CPU 内のステータスレジスタに対して、読み書きのテストを行います。

関数名	c17TestPsr
引数	void
戻り値	short （E_OK または E_ REGISTER）
関数概要	CPU 内のステータスレジスタに 0x55 と 0xaa を書き込む。その後、読み込みを行い、書き込んだ値と比較する。差異がなければ E_OK 、差異があれば E_ REGISTER を返す。

関数名	c17WritePsr
引数	void

戻り値	void
関数概要	ステータスレジスタのバックアップと書き込みを行うソフトウェア例外。 スタックに退避されたステータスレジスタの値を backupPsrVal にバックアップする。その後、wPsrVal の値をステータスレジスタに書き込む。

関数名	c17ReadPsr
引数	void
戻り値	void
関数概要	ステータスレジスタの値を比較するソフトウェア例外。 スタックに退避されたステータスレジスタの値と wPsrVal の値を比較する。差異がなければ E_OK、差異があれば E_ REGISTER を result に格納する。 その後、backupPsrVal の値をステータスレジスタに書き込む。

グローバル変数

wPsrVal	ステータスレジスタに書き込む値を格納。
backupPsrVal	ステータスレジスタの値を格納。
result	テストの結果（E_OK または E_REGISTER）。

処理フロー

テストの流れ	1. wPsrVal の値を 0x55 とする。 2. ソフトウェア例外 C17WritePsr を発生させる。 c17WritePsr にてソフトウェア例外直前のステータスレジスタの値が backupPsrVal にバックアップされ、ステータスレジスタに wPsrVal の値（0x55）が書き込まれる。 3. ソフトウェア例外 c17ReadPsr を起こす。 c17ReadPsr にてソフトウェア例外直前のステータスレジスタの値と wPsrVal の値（0x55）が比較され、比較結果が result に格納される。 4. result の値をチェックする。 ◆E_REGISTER だった場合 E_REGISTER を返す。終了。 5. wPsrVal の値を 0xaa とする。 6. ソフトウェア例外 c17WritePsr を発生させる。 c17WritePsr にてソフトウェア例外直前のステータスレジスタの値が backupPsrVal にバックアップされ、ステータスレジスタに wPsrVal の値（0xaa）が書き込まれる。 7. ソフトウェア例外 c17ReadPsr を起こす。 c17ReadPsr にてソフトウェア例外直前のステータスレジスタの値と wPsrVal の値（0xaa）が比較され、比較結果が result に格納される。 8. result の値をチェックする。 ◆E_REGISTER だった場合 E_REGISTER を返す。終了。 9. E_OK を返す。終了。

プログラムソース

c17TestPsr.c
c17RwPsr.s

2.3.6 RAM の March-C テスト

指定された RAM エリアに対して、March-C テストを行います。

関数名	c17TestRamMarchc
引数 1	volatile unsigned char *chkAddr
引数 2	unsigned short chkNum
戻り値	short （E_OK または E_MEMORY）
関数概要	chkAddr が指すアドレスから chkNum 分だけ March-c テストを実施する。問題がなければ E_OK、問題があれば E_MEMORY を返す。
注意事項	指定されたメモリ領域がスタック領域と重複したとき、動作は保障されない。 テスト範囲となったメモリは、0x00 に書き換えられる。

関数名	c17TestBitChkw0
引数	volatile unsigned char *addr
戻り値	short （E_OK または E_MEMORY）
関数概要	指定バイトの下位ビットから 0 を書き込み、その後比較を行う関数。 addr が指すアドレスに下位ビットから順に 0 を書き込んでいく。その後書き込んだ値と読み込んだ値を比較する。差異がなければ E_OK、差異があれば E_MEMORY を返す。

関数名	c17TestBitChkw1
引数	volatile unsigned char *addr
戻り値	short （E_OK または E_MEMORY）
関数概要	指定バイトの下位ビットから 1 を書き込み、その後比較を行う関数。 addr が指すアドレスに下位ビットから順に 1 を書き込んでいく。その後書き込んだ値と読み込んだ値を比較する。差異がなければ E_OK 、差異があれば E_MEMORY を返す。

処理フロー

テストの流れ	1. 開始アドレスとテスト範囲から終了アドレスを求める。 2. テスト範囲の全てのメモリに 0 を書き込む。 3. 開始アドレスからテスト範囲の全てのメモリに対して、c17TestBitChkw1 をコールする。 下位ビットから順に 1 を書き込んでいく。 ◆E_MEMORY が返ってきた場合 E_MEMORY を返す。終了。 4. 開始アドレスからテスト範囲の全てのメモリに対して、c17TestBitChkw0 をコールする。 下位ビットから順に 0 を書き込んでいく。 ◆E_MEMORY が返ってきた場合 E_MEMORY を返す。終了。 5. 終了アドレスからテスト範囲の全てのメモリに対して、c17TestBitChkw1 をコールする。 下位ビットから順に 1 を書き込んでいく。 ◆E_MEMORY が返ってきた場合 E_MEMORY を返す。終了。 6. 終了アドレスからテスト範囲の全てのメモリに対して、c17TestBitChkw0 をコールする。 下位ビットから順に 0 を書き込んでいく。 ◆E_MEMORY が返ってきた場合 E_MEMORY を返す。終了。

	7. テスト範囲の全てのメモリの値が 0x00 となっているかどうかをチェックする。 ◆0x00 でなかった場合 E_MEMORY を返す。終了。 8. E_OK を返す。終了。

プログラムソース

c17TestRamMarchc.c

2.3.7 チェックサム

指定されたメモリエリアのチェックサムの計算を行います。

関数名	c17TestChksum
引数 1	unsigned char *chkAddr
引数 2	unsigned short chkNum
戻り値	unsigned short
関数概要	chkAddr が指すアドレスから chkNum 分だけ、メモリの値を読み、その総和を返す。

処理フロー

テストの流れ	1. テスト範囲のメモリの値を全て加算して、チェックサムの値とする。 2. チェックサムの値を返す。

プログラムソース

c17TestChksum.c

2.3.8 CRC(計算)

指定されたメモリエリアのCRC値を数式計算によって求めます。サンプルプログラムでの生成多項式は$x^{16} + x^{12} + x^5 + x^0$、初期値は 0xffff とします。

関数名	c17TestCrc
引数 1	unsigned char *chkAddr
引数 2	unsigned short chkNum
戻り値	unsigned short
関数概要	chkAddr が指すアドレスから chkNum 分だけメモリの値を読み、CRC の計算を行う。

処理フロー

テストの流れ	1. crc の初期値を 0xffff とする。 以下 2.と 3.の処理をテスト範囲の全てのメモリに対して行う。 2. 指定アドレスを 8 ビット左シフトしたものと crc との排他的論理和をとったものを新たな crc とする。 3. 以下の処理を 8 回繰り返す。 crc の最上位ビットに 1 が立っているかどうかをチェックする。 ◆1 が立っていた場合 crc を 1 ビット左シフトしたものと 0x1021 との排他的論理和をとったものを新たな crc とする。 ◆それ以外だった場合 crc を 1 ビット左シフトしたものを新たな crc とする。

	4. crc の反転したものを、新たな crc とする。 5. crc の値を返す。終了。

プログラムソース

c17TestCrc.c

2.3.9 CRC(テーブル参照)

指定されたメモリエリアの CRC を CRC-CCITT テーブルを用いて求めます。計算で算出するよりも速く求めることができます。

関数名	c17TestCrcTbl
引数 1	unsigned char *chkAddr
引数 2	unsigned short chkNum
戻り値	unsigned short
関数概要	chkAddr が指すアドレスから chkNum 分だけ、メモリの値を読み込む。その後、テーブルを参照して CRC の計算を行う。

処理フロー

テストの流れ	1. crc の初期値を 0xffff とする。 以下 2.の処理をテスト範囲の全てのメモリに対して行う。 2. 下記のテーブルと crc を 8 ビット左シフトしたものとの排他的論理和を新たな crc とする。テーブルのインデックスはメモリの値と crc を 8 ビット右シフトしたものとの排他的論理和である。 3. crc の反転したものを、新たな crc とする。 4. crc の値を返す。終了。

プログラムソース

c17TestCrcTbl.c

2.3.10 割り込みテスト

一定周期で発生する割り込みが指定回数発生するまでに、検査する割り込みが何回発生するかをカウントします。

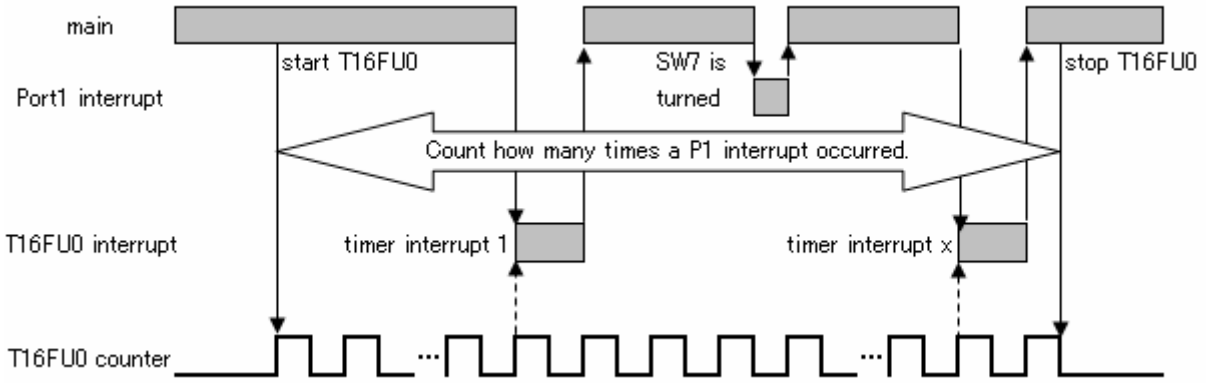

図 2. 3. 10 割り込みテストのイメージ

関数名	c17803TestIntterupt
引数	unsigned short numInt
戻り値	unsigned short
関数概要	numInt の回数分だけ割り込み（CLG の 16 ビットタイマを使用）が発生するまでに、割り込み（ポート 13(SW7)割り込みを使用）が何回発生するかをカウントする。

関数名	c17803T16FInt
引数	void
戻り値	void
関数概要	16 ビットタイマ割り込みハンドラ。

関数名	c17803P1Int
引数	void
戻り値	void
関数概要	P1 ポート割り込みハンドラ。

関数名	c17803InitIntTestInt
引数	void
戻り値	void
関数概要	c17803TestInterrupt で使用する割り込み関数の初期化関数。

処理フロー

テストの流れ	1. カウンタ値、16 ビットタイマ割り込み、ポート 1 割り込みの初期設定を行う。 2. 16 ビットタイマ割り込みが numInt の回数分だけ発生するのを待機する。 ◆この間ポート割り込みが発生した場合 戻り値のカウンタを 1 つ増やす。 3. 16 ビットタイマを停止させ、ポートの割り込みを禁止する。 4. カウンタの値を返す。終了。

プログラムソース

c17803TestInterrupt.c

2.3.11 メインクロック安定性確認テスト

メインクロックが指定誤差範囲内で動作しているかどうかをチェックするテストです。
このテストにはサブクロック（32KHz）で動作する RTC を使用します。

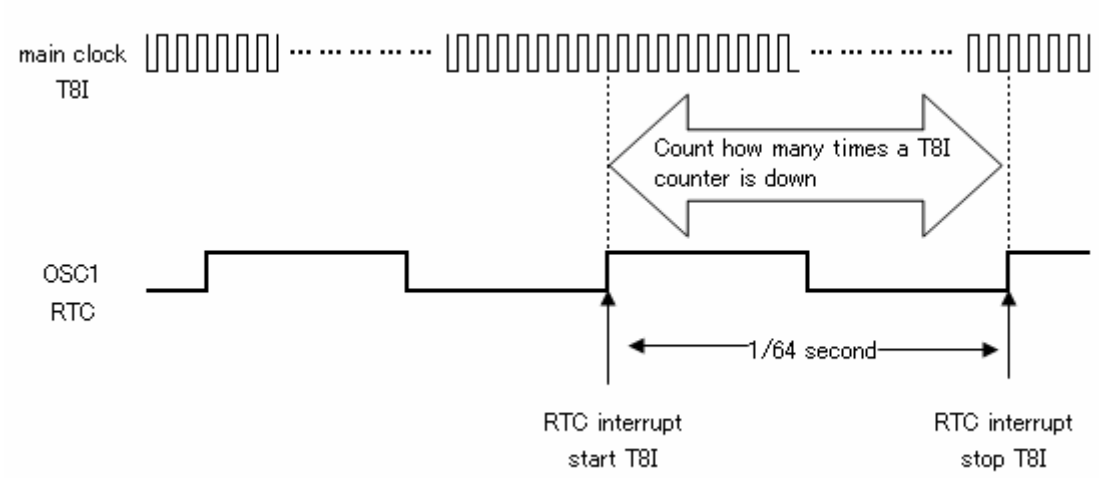

図 2. 3. 11 メインクロック安定性テストのイメージ

関数名	c17803TestClk
引数 1	unsigned long baseFreq
引数 2	unsigned short range
戻り値	int （E_OK または E_CLOCK）
関数概要	使用しているメインクロックが許容誤差範囲内で動作しているかを確認する。baseFreq で理想的な周波数の値を指定し、range で許容誤差（%）を指定する。テスト結果が許容誤差範囲内ならば、E_OK を返し、範囲外ならば E_CLOCK を返す。

関数名	c17803RtcInt
引数	void
戻り値	void
関数概要	リアルタイムクロック割り込みハンドラ

関数名	c17803T8IInt
引数	void
戻り値	void
関数概要	8 ビットタイマ割り込みハンドラ

関数名	c17803InitIntTestClk
引数	void
戻り値	void
関数概要	c17803TestClk で使用する割り込み関数の初期化関数。

処理フロー

テストの流れ	1. リアルタイムクロック、8 ビットタイマの初期設定を行う。 2. リアルタイムクロックで 1/64 秒割り込みを発生させる。 リアルタイムクロック割り込みの安定待ちのため、1 回目の割り込みで 8 ビットタイマをスタートさせ、2 回目の割り込みで計測を行う。 ◆この間、メインクロックで 8 ビットタイマをカウントダウンさせる。 3. 8 ビットタイマの値を読み出す。 4. 3.の値からメインクロックの周波数を求める。 3.の値に 64 と分周比の逆数をかけたものがメインクロックの周波数となる。

	5. 4.の値が許容誤差以内かどうかをチェックする。 ◆許容誤差内の場合 E_OK を返す。終了。 ◆許容誤差外の場合 E_CLOCK を返す。終了。

プログラムソース

c17803TestClk.c

3　参考情報

ここでは、プログラミングおよび、ハードウェアに対する参考情報について記載します。

3.1　外部インタフェースによる通信異常診断

データ転送中にデータ汚染がないかどうかチェックするためには、できるかぎり転送データの冗長性によってすべてのコミュニケーションを保証する必要があります。保証する手段の一つとして、サンプルプログラムの CRC を使用することができます。

また、通信エラーによるエンドレスループを避けるためには、タイムアウト検出を導入することも対策として有効な場合があります。

補足：

通信異常診断とともに、通信異常が発生しにくい構成とすることも大切です。信号線は基板上で他のデジタル線と並走したために、それらの信号の 1→0、0→1 の変化点の影響を受ける場合があります。特に、同時変化する複数の信号線に挟まれると、大きなノイズが信号に乗ります。並走距離を短くする、信号同士を離すといった対策が有効です。

また、相互インダクタンスによって生じる電磁誘導ノイズを防止するために、発振部、アナログ入力部等のノイズの影響を比較的受けやすい回路近くには、大きな電流が流れる信号線を配置しないでください。

改訂履歴表

付－1

コード No.	ページ	改訂内容（旧内容を含む） および改訂理由
411808600	全ページ	新規制定

セイコーエプソン株式会社
半導体事業部　IC 営業部

<IC 国内営業グループ>

東京　〒191-8501　東京都日野市日野 421-8
TEL（042）587-5313（直通）　FAX（042）587-5116

大阪　〒541-0059　大阪市中央区博労町 3-5-1　エプソン大阪ビル 15F
TEL（06）6120-6000（代表）　FAX（06）6120-6100

ドキュメントコード：411808600
2009 年 9 月 作成