お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。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、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

H8/300L SLP シリーズ

開発ツール/デバッグ技術 (DebugTec)

要旨

このアプリケーションノートでは，ユーザの作り込んだ各種バグを説明します。
次に，バグ検出のためのテストの分類に焦点をあてます。
最後に，効果的なさまざまなエラー防止方法，エラー検出方法，およびデバッグ技術を解説します。

動作確認デバイス

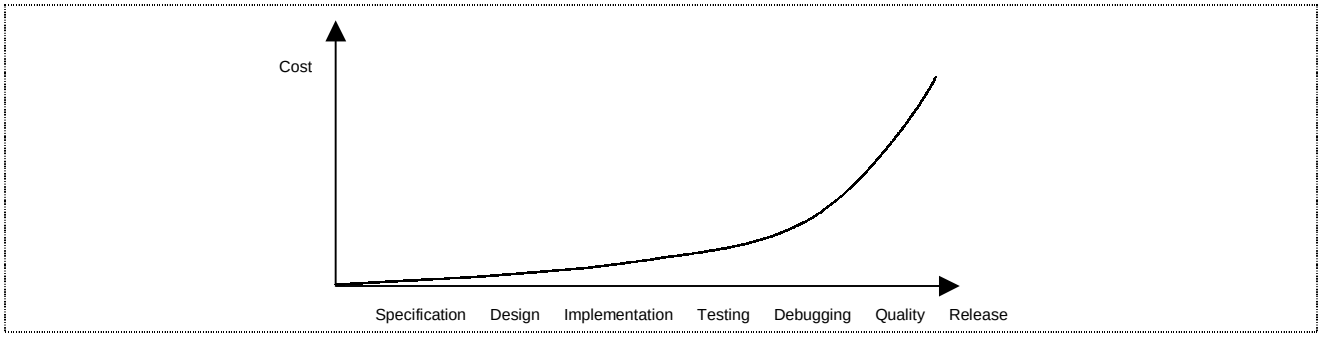
すべての H8/300L SLP MCU

目次

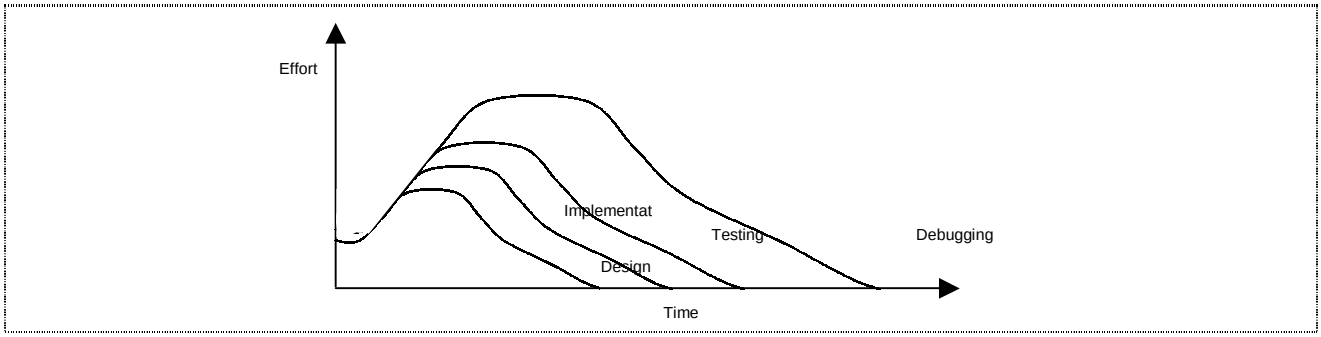
1. はじめに

バグの修正は，開発サイクルでは必須の作業です。これは開発の最後にだけ行う作業ではありません。継続したバグの防止，検出，修正の努力が必要です。
効果的に効率よくバグの修正プロセスをふむために，2 つの明白な統計により開発者がどのような戦略を使用したらよいか、理解に役立ちます。

1. 開発サイクルの終わりのバグの修正は高くつきます。



2. 実行段階よりも統合 (テスト，デバッグ) 段階ではより多くの作業を必要とします。



一般的に，実行段階は，統合段階よりも，完成するまでの期間が短いです。統合段階で時間がかかるのは，この段階でほとんどのバグが見つかるからです。統計によると，クリーンなコンパイルでは，書かれたコード 20 行当たりにバグが 1 つ検出されます。つまり，32k のコードでは，1500 のエラーが検出されるということです。

このアプリケーションノートでは，設計はコーディング段階で起きるエラー防止に関する多くのトピックスを扱います。バグ検出のためのテスト方法にも焦点をあてます。目的は，開発中の作業におけるさまざまな段階を明確に規定することです。これにより，開発者はバグを防止，検出，修正する整然とした計画を作成できます。

テスト：正しいとされるプログラムの組織的な実行。バグ対策の努力は行われない。

デバッグ：プログラムの間違いを実証する段階で，エラーの存在が明らかとなる。

このアプリケーションノートの内容のほとんどはソフトウェア関連です。
しかし，常に，トラブルの原因にハードウェアが疑われる場合があります。後半の章では，ハードウェアのトラブルシューティングについても触れます。

2. バグの種類

バグを見い出して削除するには，バグを理解することが必要です。

以下にプログラマが発見するバグの一般的分類を紹介します。

1. 実行できないエラーソース
   — 不明瞭/不明確な仕様
   — 例外処理の不処理
2. アルゴリズム/論理/処理中のバグ
   — もう 1 つのループを実行する条件ループ
   — AND と OR 条件の論理エラー
   — 複雑なアルゴリズムの不理解
3. データのバグ
   — ポインタのエラー
   — データ範囲のオーバーフロー/アンダーフロー
   — LSB & MSB の定義
   — 意味論
4. リアルタイムバグ
   — 割り込み処理と抑制
   — タスク同期
5. システムバグ
   — スタックオーバーフロー/アンダーフロー
   — 資源共有の問題
6. その他のバグ
   — 構文/タイプミス
   — メモリリーク
   — 周辺の初期化

3. 設計およびコーディング段階 – バグ防止とデバッグ機能

初期段階での正しいコーディングと計画はバグ修正プロセスを削減します。

定まった規定はありませんが，以下にバグ防止策を列挙します。これらの詳細は後半の章で説明します。

1. 正しいコーディング規格を守る
2. 防御的なプログラミングをする
3. エラー処理を計画する
4. デバッグとテストを計画する
5. 変数を正しく定義する (unsigned / signed integer long)
6. 起動時にすべての項目を定義された初期化状態にする
7. ソフトウェアプログラムコードで意味のある正しい備考を書く
8. デバッグツールの利用を計画する
9. モニタやデバッグコードを計画する
10. printf 関数やアサートを利用する
11. クロック，グランド Vcc, IRQ, 外部トリガなどのテスト用端子やパッドを準備する
12. ピン番号，信号名，ジャンパ名をシルクスクリーンで印を付ける

4. テスト段階 – バグの検出

テスト段階には 3 つの異なる分類があります。

1. 第一段階
   — 初めのコーディングプログラムの機能的段階へのテスト
   — たとえばユニットテスト
2. 品質保証段階
   — 完成したシステム上で誤りの無い確実な段階へ到達するためのテスト
   — たとえばカバレッジテスト
3. 保守または退行テスト段階
   — 拡張または修理中にテストされたシステムに新しいバグが入り込まないようにするためのテスト
   — 品質保証段階で行う自己テストまたはカスタムメイドのテスト

第一段階および品質保証段階で行うすべてのテストは適切に格納しドキュメント化する必要があります。なぜならば，これらは保守または退行テスト段階の基本となるからです。

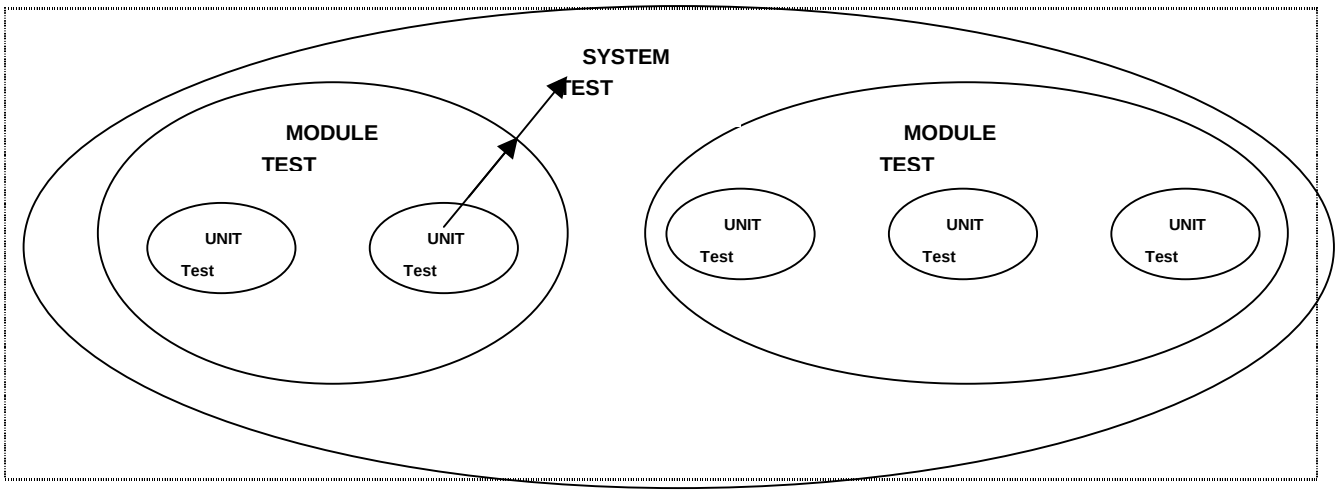
これらのテストは 2 つの方法で実行されます。

1. 静的テスト
2. 動的テスト

テストの内容に立ち入る前に，テストの対象を明確に定義する必要があります。

1. 単体テスト
2. 結合テスト
3. システムテスト

完全なシステムをデバッグするのは，より小さなモジュールをデバッグするより，常に困難です。これは，エラーの組み合わせの可能性があるからです。したがって，小さな単体テストから始め，次により大きなモジュールのテストを行うことが重要です。これにより，最終的な統合システムテストにおいて，短期間で簡単なデバッグ段階に到達します。



4.1 静的テスト

静的テストはもっとも簡単なテストです。コードを実行する必要はありません。しかし，基本的なチェックのプロセスで，多くのバグが発見できることがあります。

1. コードのウォークスルー
   開発者は，書かれたコードをざっと読む間，コードを分析できる。
   — フローのシーケンス
   — パス
   — 渡される引数
   — たとえば，括弧がないなどの論理的エラーを発見するかもしれない。

2. コードのコンパイル
   コードをコンパイルすることにより，構文が正しいかどうかチェックできる。コンパイラによるワーニングを無視してはいけません。
   — たとえば，未使用の定数セクションが変数の宣言を間違いを示唆しているかもしれません。

3. 分析ツール – HEW の CallWalker と MapViewer
   開発者の分析に役に立つツールがあります。ルネサス社製の High-performance Embedded Workshop (HEW) が次のツールを提供します。たとえば, MapViewer, CallWalker などは, コンパイルされたコードをチェックするのに役立ちます。
   — たとえば，MapViewer はコードが間違ったセクションに位置していることを示す場合があります。

4. 設計仕様と評価
   最初の仕様を参照しながらコードをチェックするのは良いことです。これは後半の段階でより良い統合化を確実にします。
   — たとえば，ハードウェアレジスタの定義の誤った思い込みは，思わぬところでシステムがハングアップする原因になることがあります。

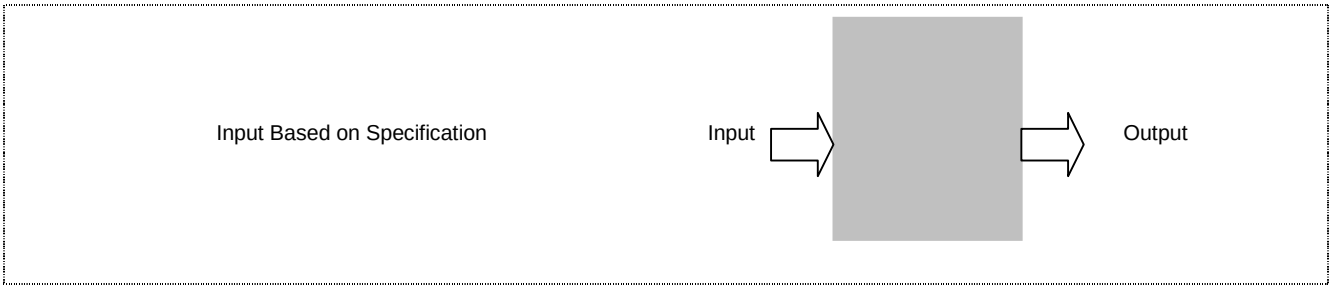
[#1] MapViewer： HEW **内蔵のコンポーネントで，コンパイルされたコードがどこに位置するかグラフィックな視覚を提供します。**

Call Walker： 呼ばれた関数のパスを分析することにより，必要なスタックの深さを示す。

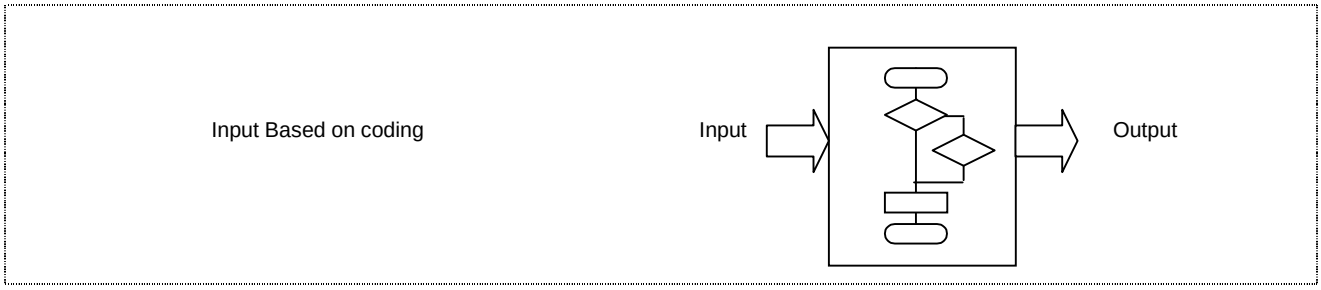
4.2 動的テスト

各ユニット，モジュール，システム用に書かれたコードは，動作条件を評価するために実行されます。それらのテストは以下のように分類できます。

1. ブラックボックステスト (機能的テスト)
   A. テストプログラムの動作と必要な仕様とを比較する
   B. 動作方法に関係なくプログラムの完成度を検討する



2. ホワイトボックステスト (構造的テスト)
   A. テストプログラムの動作とソースコードの明白な意図とを比較する
   B. 構造上または論理上の欠点を考慮しながらプログラムの動作を検討する



テストはさまざまな手段で実行できます。

1. ソフトウェアの実行
2. テストスクリプトの生成
3. ソフトウェアの性能
4. データの正確さの検証
5. ユーザシステムのエミュレーション

一般的に，エミュレータやシミュレータなどのデバッグ用開発ツールはテストに使用します。コードはプログラムをシングルステップ実行でテストする，またはあるポイントから別のポイントまで実行することでテストします。統合化開発環境 HEW のレジスタ/メモリ/IO/局所変数/ウィンドウなどで入力を簡単に確認できます。出力は，同じウィンドウまたは詳細なカバレッジを表示するトレースウィンドウでモニタリングできます。これは，HEW のテストスクリプト TCL/TK で自動化できます。このテストは再利用してコードの完全性を再評価します。さらに，HEW のカバレッジやパフォーマンス測定機能により，開発者はより効率的なソフトウェア性能の評価を得られます。メモリの比較やファイル検証機能はデータの正確さを検証するのに良い機能です。ユーザシステム動作テストには，エミュレータの使用は不可欠です。

テスト完了後，実際にシステムを稼動する前に，ベータサイトのテスタ (または第三者) がそのシステムを使用することを推奨します。これにより，別の視点からの広範囲のテストを保証できます。

4.3 カバレッジ

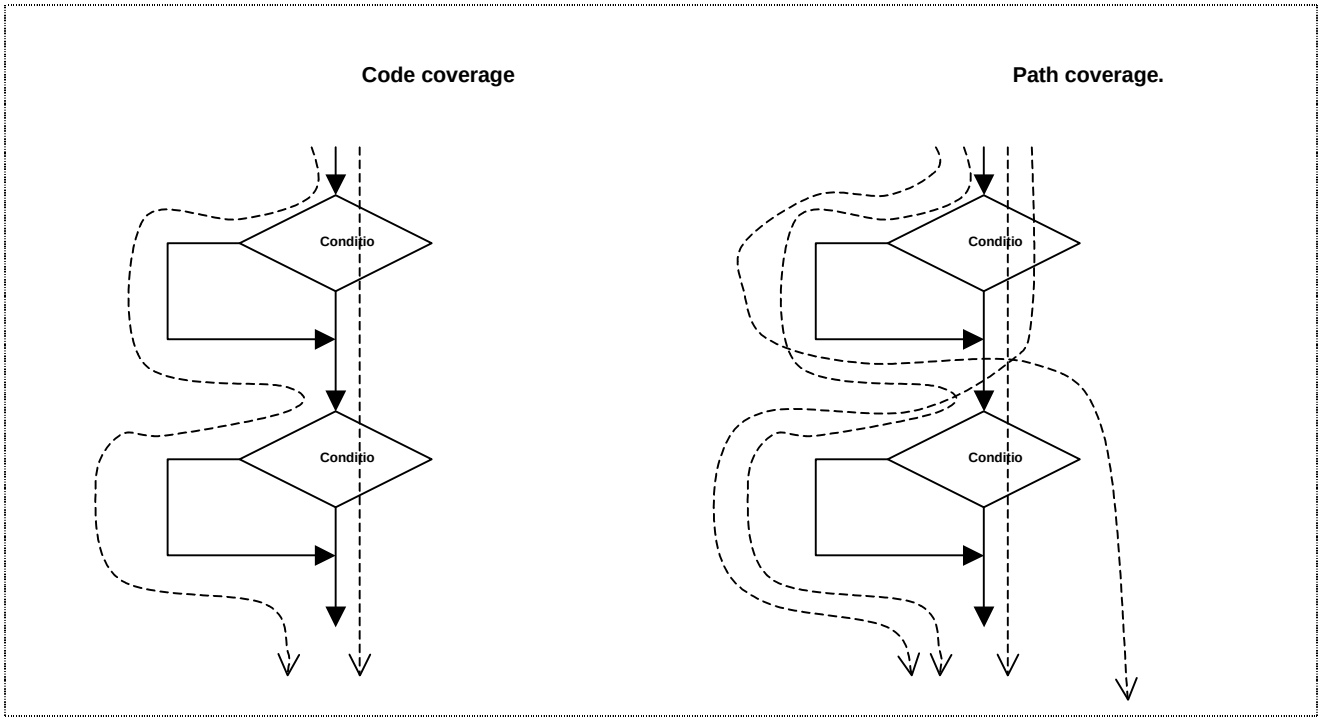
カバレッジは重要なテストの概念です。

以下のような各種用語を使用します。

条件テスト	-	すべての論理条件を実行する
パステスト	-	すべての可能性のあるパスを実行する
判定テスト	-	すべての可能性のある分岐を実行する
命令文テスト	-	プログラムのすべての命令文を実行する

一般的なねらいは，コードのすべての行を実行することです。実行されないコードがあると，バグが検出されない可能性が大きくなります。どのくらいのカバレッジテストが必要か，または，十分か，という明確な基準はありません。理想的には，100%のカバレッジが必要です。しかし，これを達成するのは困難です。基本カバレッジを 85%とするガイドを作成して，開発者が効率良くこの値に到達すればその開発者は作業を中止してしまいます。リスクの高いコードを十分にテストすることが一般的なガイドです。(しかし，これは明確な規則ではありません。)

例：



5. デバッグのテクニック

デバッグは回避できないものであり，予測できない時間を費やします。デバッグ時間を減らすためには，開発者による効率の良い計画と実行が必要です。開発者のバグへの対処方法は重要です。明晰で新鮮なアプローチにより，効率良いデバッグ作業が可能です。

以下に論理的なアプローチを示します。

1. 変更するとバグの原因になります– 変更点に注意
2. デバッグを後回しにしないで今やること。後で仕事が遅れる。
3. 同じ失敗を２度繰り返さないようにすること (同じ間違いは再度起こる。よく突き詰めること)。
4. 変更前に読み出し分析すること
5. 第三者にバグを説明すること。そうすれば，理由付けや質問されることにより，新しい洞察力が生まれる
6. 再生できるバグは解決しやすい
7. バグを突き止めるためモジュールを細分化する
8. バグの兆候と出力パターンを研究する
9. 誤った領域をデバッグしていないか確認する
10. その他のシステムバグ–コンパイラとエミュレータ (複数の開発者が同時に使用している)
11. 使用している開発ツールを理解する

バグを防止する，突き止めるためのテクニックを紹介します。

1. 良いコーディング基準に従う
   C ファイルやヘッダファイルの構成，使用する定義など良いコーディングに関する十分な経験をつまなくてはならない
2. 移植性の良い，再利用できる明瞭なプログラムを作成する
   関数には，明白な，入出力パラメータが必要です。これにより，わかりやすく移植性の良いプログラムになります。使用する定数は関数のなかに組み込むのではなく，宣言してヘッダファイルで明確に使用します。関数のなかに組み込むと変更が難しくなります。
3. 防御的プログラムを作成する
   case 命令文で起こりえない例外処理を含む，ヌルポインタをチェックするなど。
4. エラー処理を計画する
   待ち時間が長すぎる場合，タイムアウトエラーを発生させる，または，エラーコードを戻すなど。
5. デバッグとテストを計画する
   デバッグ条件を作成してこれらの条件をプログラムのなかで個別に処理してください。これによって，ハードウエアのバグを分離するなど，テストが簡単になります。以下に自己説明的な条件付き定義を示します。

```
#Define DEBUG                TRUE
#Define HARDWARE_READY       FALSE
#Define HOST_READY           FALSE
#Define SELF_TEST            TRUE
```

6. 変数を正しく定義する (unsigned / signed integer long)
   すべての関数のプロトタイプとパラメータの引渡しで同じ種類の変数を扱っていることを確認してください。
7. 起動時にすべてのコンポーネントを一定の初期状態に設定する
   — すべての変数，ハードウェアレジスタ，データ領域が初期化されているか確認してください。
   — 初期状態が異なると，バグ再生が不可能になることがあります。
8. 意味のなす正しいコメントを入れる
   — 間違ったコメントは読み手に誤解を与えます。
   — プログラマ名と変更日時も重要な情報です。
9. 開発ツールの使用を計画する
   — エミュレータ用のユーザケーブルは余分な空間を必要とします。
   — “JTAG ライク” エミュレータではバグを突き止めるためのトレース情報が不足するかもしれません。したがって，ソフトウェアによるトレース機能の追加が必要かもしれません。
10. モニタ/デバッグコードを計画する
   — 基本デバッグを実行するために，アプリケーションにモニタコードを組み込むことがあります。初期段階で，モニタコードの制限事項を考慮する必要があります。
   — 他のデバッグコードを準備する必要があるかもしれません。たとえば，LED の点滅，シリアルポートまたはメモリ空間の printf 命令文など。
11. テスト用の端子やパッドを用意する- クロック，グランド Vcc, IRQ, 外部トリガなど
   — 適切な動作条件を保障するため，電源端子やクロック端子は必須のテストポイントです。
   — 電流の測定が必要な場合，特定のテストポイントが必要となります。
   — MCU または FPGA の未使用端子のための余分なパッドを作成すると，再配線やプローブに便利です。
12. ジャンパ設計
   — シルクスクリーンのラベルはジャンパやスイッチの設定を理解するのに役立ちます。
   — 次のことを考慮してください。
      - アクセスが容易な位置
      - オン状態は，通常，ショート，High レベル，イネーブルを意味します。
      - 輸送中，ジャンパが落ちるかもしれません。これにより，システムはデフォルト動作状態になります。 (たとえば，ジャンパ挿入をデバッグモードに設定してください。)

13. 端子番号，信号名，ジャンパ名を明記(シルクスクリーン)する
    — 重要なポイントは，信号名を印字することです。 (例：GND)
    — 容易に外部プローブを接続できるようにするため, 面実装タイプの IC は 5 ピンごとに間隔をあけて印を付けてください。
14. 電源オン用 LED を付ける
    — トラブルシューティングして，結果として，電源が投入されていないことに気づくことがよくあります。
    — LED が点滅して電源ラインが断続していることを示すことがあります。
15. コンパイラのワーニングを抑えない，無視しない
    各ワーニングは不適合を通知します。実行は一時的に「正常」かもしれませんが，1 つの例外が深刻なバグの原因になることがあります。
16. セッションやスクリプトの保存/復帰
    HEW には，セッションやスクリプトの実行を保存/復帰する機能があります。これは擬似的な「初期状態」を保障する良い自動化ツールです。

一連の設定を手動で行うと，手順を飛ばしてシステム動作が異なる場合があります。

17. 仮定ではなく推定する
    転送，CPU 速度などに関する基本的な計算をして，推定することにより，適切なタスクの実行が保障されます。
18. 入力データをエコーバックする
    別の周辺機器またはホストから受信したデータをエコーバックすることは有効です。データの解釈が異なったり間違ったりする場合があるからです。入力を間違えると，当然，出力も正しくありません。
19. DMA または割り込みに気をつける
    割り込みイベントが，厳密なタイミング動作を乱すことがあります。DMA (Direct Memory Access) 機能が I^2C デバイスへのシーケンシャル書き込みを失敗させる可能性があります。開発者は精密なタイミングのイベントに注意が必要です。
20. 未使用領域に特定のコードを書き込む
    — 未使用の割り込みベクタにデバッグルーチンへのポインタを書き込む
    — 未使用領域にデバッグルーチンへの BSR 命令を書き込む
    デバッグルーチンでブレークすると, エラーがただちに検出できます。さらに, スタック領域をチェックすると，最新の実行位置がわかります。(エミュレータのトレースウィンドウはより多くの情報を表示します。)
21. 未使用領域にあらかじめ決められたデータを書き込む
    H’1234，H’5A5A，または H’A0A0 のデータパターンで，データ領域，スタック領域，またはヒープ領域に書き込んでください。そのようなパターンのデータをポインタが取得すると，ポインタの使用方法が明らかに間違っているとわかります。このデータはアドレスポインタとして使用する場合があるので，偶数のデータを選択してください。退行テストを実行すると，開発者はスタック領域やヒープ領域の利用方法を観察できます。
22. アナログ信号の処理
    グランドやデジタル干渉の可能性の点から，ハードウェアは扱いに注意しなければなりません。
    ハードウェア回路の静的テストにより，アナログ信号が正しくキャプチャできることがあります。しかし，動的には，A/D 変換器を介して読み出すと，この信号は正しくないことがあります。したがって，取得データをメモリにそっくりそのままうつす，または一連のシミュレーションデータをシステムに送信することが賢明です。これによって，エラーの原因を隔離することができます。
23. 割り込みを数える
    ハードウェアカウンタにより，割り込み端子をプロービングすることができます。また，MCU へのすべての割り込みは外部カウンタをインクリメントします。これによって，システムが最高速度で実行中に見過ごした割り込みの数を開発者は割り出すことができます。

24. 自己テスト/診断プログラム

テスト用プログラム/ルーチンは以下の 3 つに分類できます。

— 実際のアプリケーションより簡単なプログラム (例：RTOS プロジェクト)で，基本的なプラットフォームの動作のテスト/検証用に書かれたものです。このプログラムの目的は，初期のハードウェア設計とプロトタイピングで起き得る課題を隔離することです。

— 標準のアプリケーションルーチンの動作を起動して検証する自己テストルーチンです。これは，各ルーチンの正しい動作や特性を保障し保持するためのものです。この自己テストルーチンは，外部デバイスとの通信において発生し得る問題を隔離するために，システム内で定義されます。

— システムをテストするために書かれた PC ベースのテストスクリプトまたはプログラム。

25. ウォッチドッグ機能

ウォッチドッグ機能を何らかの問題から回復するためにアプリケーションで使用すると，トラブルシューティング中により多くの問題が発生することがあります。したがって，開発段階において，ウォッチドッグ機能をオフにする方が賢明です。ウォッチドッグ機能を外部チップセットで持たせる場合，(ジャンパにより)オフにする機能を付けるべきです。

しかし，このウォッチドッグ機能を使ってデバッグを容易に進めることもできます。起動したウォッチドッグ機能をリセットから実行させるのではなく，リセットルーチン内でソフトウェアによるチェックを行い，そのルーチンをデバッグルーチンに分岐させます。

26. printf と Assert の利用

Assert() は，条件付きの printf()命令文と似ています。

組み込みシステムは標準の出力コンソールがないので，開発者はこの機能について計画しなければなりません。デバッグ情報は LCD や PC のハイパーターミナルに出力できます。しかし，この方法はかなりの CPU の時間を使います。別の方法として，printf()関数を RAM 領域に書き込むことができます。(下記のソフトウェアトレースを参照してください。)

27. ソフトウェアトレース

printf() 関数は関連するデバッグデータを出力するのに便利です。出力媒体としてシリアルポートを使用すると，時間がかかります。これによって，リアルタイム性の問題が発生することがあります。しかし，メモリアクセスを使用すると，この方法の煩わしさが軽減します。

動作上の注意点：

— ユーザが作成した printf 関数で循環バッファに固定長のデータを書き込むと，バッファポインタがインクリメントされバッファの終わりに達するとバッファの初めにリロードします。

— 開発者はすべての可能性のある評価ポイントで printf 関数を挿入します。

— プログラムを終了するにあたって，開発者は，シリアルポートを介してコマンドによりバッファからトレースデータを回収できます。

— プログラムが壊れたときのために，バッファは保存されなければなりません。

— 起動ルーチンで，バッファやポインタをクリアしないようにします。

— 意図するトレースの深さによって，適切な大きさの RAM を使用します。

28. 開発ツールの効果的利用方法

以下の開発ツールの特長を十分活用してください。

— 評価用に条件を取得するための複雑なブレークシステム

— プログラムの暴走を制限するためのガードアクセス

— プログラムの流れを監視するためのイベントトレース

— CPU の活用を測定するための実行時間測定

— データ変更時に直ちに影響が観測できる実行中のメモリアクセス Parallel-on-the-fly (POTF)

— 手動でコードのカバレッジをモニタする各種ステップ実行 (Step In/Out/Over)

— データの完全性を評価するためのメモリまたはファイルの比較，またはユーザシステムの動作条件

下記アプリケーションノートを参照してください。

” Effective Usage of Hardware Development Tool”

29. HEW 機能の使用

MapViewer	–MCU マップで生成コードの位置を検証する
プロファイリングとパフォーマンス分析	–最適化できるようにプログラムの危険地帯を明らかにする
CallWalker (スタック分析)	–必要なスタックの深さを分析する
カバレッジ	–プログラムのカバレッジを検討する
TCL/TK	–スクリプトを動作の一貫した実行に使用する

下記アプリケーションノートを参照してください。

” Effective HEW Usage of Map Viewer”

“ Stack analysis using Call walker”

“Effective HEW Usage of Profile and Performance”

“Effective Usage of HEW Coverage”

30. バグを数えるバグの管理は，バグを制御するもう 1 つの方法です。
開発者はすべてのバグを突き止めて，数えて，ドキュメント化すべきです。
このプロセスによりバグがより低減するので，より早く出荷できます。

6. ハードウェアのトラブルシューティングのガイド

ハードウェアのトラブルシューティングを行うときに考慮すべきいくつかのキーポイントを以下に説明します。どのような新規プロトタイプでも，ハードウェアの課題は日常的なものです。プロトタイプが品質設計検証の段階を完了しなければ，長時間 (誤った) 使用後，壊れる可能性があります。

1. 検査
   目視検査は，コードのシミュレーションと同じくらい重要です。
   回路のオープン，ショートだけではなく，結線や仕様の再検証に注意を払います。
   拡大鏡または画像システムにより，冷却はんだ，PCB クラック，半田ブリッジ，部品の誤り，向きの誤りなどをより精密に検査できます。
2. 電源チェック
   電源チェックとは，電圧計による VCC 検証ではありません。起動から動作までの機器による監視が必要です。なんらかの低下やリップルがシステム全体への不要なノイズを引き起こす可能性があります。1/10 または 1/100 ファラッドのコンデンサ (蓄積) が必要かもしれません。各 IC の各 Vcc 端子の近くに 0.1uF のデカップリングコンデンサがあることを確認してください。

$$C = \frac{I\ (\text{required current drive}) \times t\ (\text{transition time})}{V\ (\text{tolerable ripple})}$$

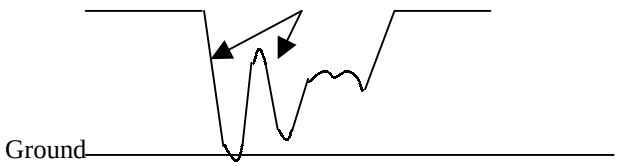
3. クロックチェック
   機器を使用して，クロックの周波数，立ち上がり時間，立ち下がり時間を検証してください。水晶発振子が他の容量性または誘導的部品によって異なる周波数で発振する場合があるので，基本的な特性をチェックしてください。
4. リセットチェック
   リセット条件は，ハードウェアマニュアルの仕様を満たさなければなりません。これはシステムの正しい初期化を保障します。
   システムの容量性の負荷が高いと，電源の立ち上がり時間が遅延し，それによって，不当なリセットが起こる場合があります。
5. タイミング
   外部メモリや周辺機器とインタフェースする場合，ハードウェアタイミングの問題が発生します。
   必要不可欠なチェックポイントは，MCU と外付けデバイスの AC 特性です。
   一般的な注意事項として，読み出し/書き込みサイクルのセットアップ時間とホールド時間をチェックしてください。
6. 電源投入/リセットシーケンス/状態
   電源投入とリセットシーケンスは注意深く分析しなければいけません。以下の質問が根源の問題につながることがあります。
   — パワーオンリセットが直接すべてのシステムにかかっているか。電源投入シーケンスはあるか。
   — メイン MCU が他のサブシステムのリセットを制御しているか。
   — 電源は 2 系統あるか。(起動時競合は？)
   — 起動状態で衝突や競合が生じるのか。
   — 遷移時の電流の量はどのくらいか。
   — リセット状態は他のシステムにリスクを発生させるか。プルアップ/プルダウン抵抗が必要か。
   — リセット状態の MCU や IC 端子の状態はどうか。ハイインピーダンスか。
   — 初期リセット状態でより高い電流が必要か。追加の電流をレギュレータが供給できるか。
   — システムは電源を入れたまま外部周辺機器を取り外しできるように設計されているか。
   — システムに，逆電流や逆電圧，高電流や高電圧に対する制限などの安全策は取り入れているか。

7. IC の駆動とインタフェース
   — すべてのインタフェースが正しいロジックに準拠しているか (DC 特性)。
   — 3V および 5V システム用に設計されているか。
   — CMOS およびバイポーラシステムを設計されているか。
   — 入力は 5V トレラントか。
   — 1 つの出力で多くの入力端子を駆動していないか。 (駆動電流が十分でも，容量性負荷が信号の速度を低下させる場合がある)
   — プルアップ抵抗に正しい値を使用しているか。
8. 耐熱性
   より複雑なソフトウェアデバッグの前に，システムの予備の耐熱テストを行ったか。
   耐熱テストは，24 時間以上放置する，または温室で放置する自己テストです。
9. 結合
   — プリント基板はノイズの干渉に耐えられる設計をしているか。
   — 高周波と低周波は隔離されているか。
   — デジタルとアナログは隔離されているか。
   — 差動信号を特別に処理しているか。
   — グランドパスやリターンパスを考慮しているか。
10. 信号の測定
    測定機器は注意深く選んでください。ロジックアナライザ，オシロスコープ，マルチメータ，およびプローブは，所望の信号特性をキャプチャできることが必要です。

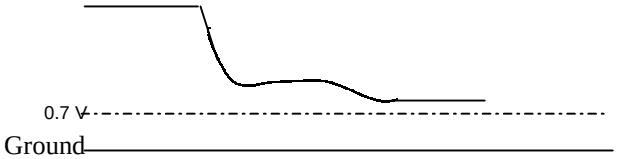
- 優れたロジックアナライザは最大 4ns の分解能まで測定できます。したがって，タイミングがあっていることの検証にはロジックアナライザのエラーの可能性を考慮に入れてください。
- 50MHz の動作を測定するために 100MHz のオシロスコープを使用すると，リアルタイムエラーは正しく信号をとらえられません。500MHz または 1GHz のオシロスコープが必要です。
- 使用するプローブは非常に重要です。端子のノイズを測定するには，短いプローブとグランドのリードを持つ GHz 程度のアクティブプローブを使用してください。

測定されたエラーの例：

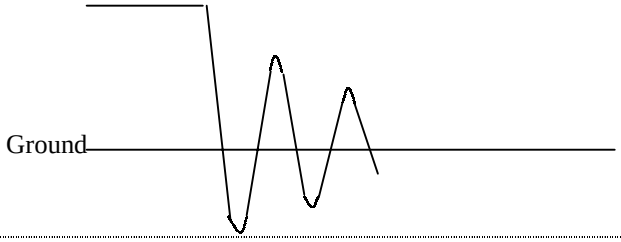
— 2 つのエッジトリガ



— フローティング信号.



— ノイズ信号 (反響)



7. 要約

このアプリケーションノートの記載項目

- バグの種類の概要
- 設計段階におけるポインタ
- さまざまなテスト段階。単体，モジュール，システムテストの静的，動的テスト。
- デバッグテクニックの論理と実践
- ハードウェアのトラブルシューティングガイド

優良な計画性と実践が効果的なバグ対策へのアプローチです。

8. 参考文献

1. www.embedded.com
2. www.ganssle.com
3. www.testing.com
4. www.ednmag.com
5. “The Practice of Programming” by Brian W. Kernighan, Rob Pike, Addison-Wesley
6. “Writing Solid Code” by Steve Maguire, Microsoft

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2004.08.06	—	初版発行

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、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。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。