解 説 組込みシステム開発の現状

3. 組込みシステムのデバッグ手法

Debug Technique of Embedded System by Masahiro SHUKUGUCHI (Mitsubishi Electric Micro-Computer Application Software Co., Ltd.).

宿口 雅弘[1]

1 三菱電機マイコン機器ソフトウエア(株)

1. はじめに

組込みシステムにおけるデバッグとは，システム(ハードウェアおよびソフトウェア)のバグ(論理的な誤り)の原因を発見し修正することである．システムにバグがあることを示すテストとはアプローチが異なる．テスト技法は論理的研究がなされているが，デバッグ技法の論理的研究は初歩段階である．デバッグは問題解決の過程を踏み，パズルを解くような直観・実験・発想の自由が必要である．そのために必要な情報を効率よく収集しなければならない．ここではデバッグ(とくにターゲットシステム上で実際にプログラムを動作させて行う場合)において，現在よく使用されている情報を収集するためのデバッグ手法とデバッグツールを紹介する．

2. デバッグとテスト[1), 2)]

テストは，製品(組込みシステム)の品質を保証するため，設計仕様書に記述されてる内容が製品に盛り込まれていることを確認するシステム開発の一工程である．多くの場合，ここでバグと呼ばれる不具合が発見される．このバグの(1)原因が何であるかを突き止め，(2)どう修正すればよいかを検討し，(3)実施するのがデバッグである(図-1)．

テストとデバッグの違いを表-1に示す．

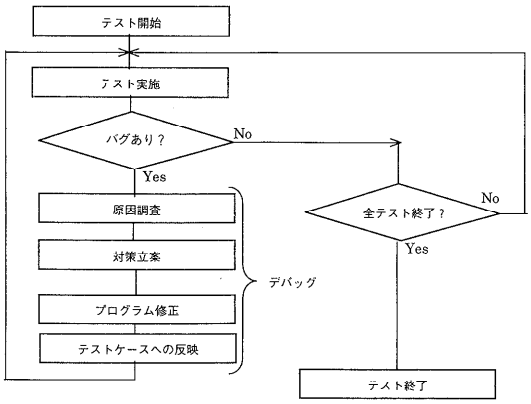


図-1 テスト工程

3. 組込みシステムの特性[4), 5)]

「マイコン搭載」が売り文句にならないほど，最近の電気製品にはマイコンと呼ばれるCPUが搭載されている．その規模は家電製品，情報通信機器，産業・社会システム全般に及んでいる．そして，何らかの不具合が発生した場合の人間社会に及ぼす影響が大きいことは，汎用計算機システムと同一である．以下に組込みシステムの特性を示す．

(1) システム上の特性

• 自立したシステム．

データの入力などに人間が関与するが，それ以外では関与することがない．障害の復旧もシステム自身で行う．

(2) 設計上の特性

• ハードウェア制御がメイン．

プログラムは計算(Computing)よりも機器の制御(Control)を目的とする．このため，ソフト

表-1 テストとデバッグの違い[3)]

	テスト	デバッグ
目 的	バグの存在を示す	バグを修正する
実施条件	既定条件下	未知条件
出 力	既定出力	不定
終了判定	計画	統計的判断
方 法	企画，設計して実行	方法限定不可能
期 間	計画可能	期間限定不可能
実 施	機械的に実行する	直観，実験，発想の自由が必要
設計知識	不要	不可欠
陣 容	設計部隊以外で可能	設計部隊以外では不可能
理論的研究	確立されている	初歩段階
自動化	設計／実行で可能	不可能

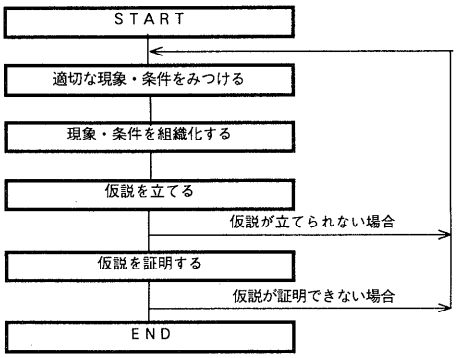


図-2 帰納法によるデバッグ手順

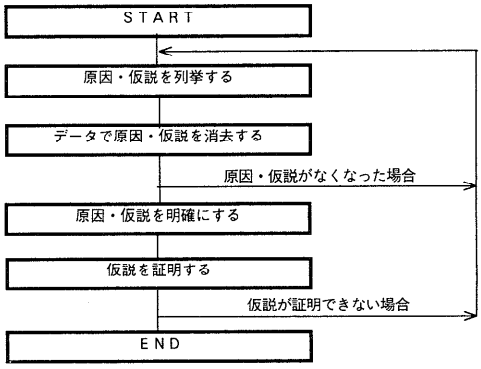


図-3 推論によるデバッグ手順

ウェアの作成，デバッグにハードウェアの知識が不可欠である．

(3) 開発工程上の特性

• ハードウェアとソフトウェアの並行開発．

専用のハードウェアを製作し，開発期間が短いことが多いため，ハードウェアとソフトウェアのデバッグを含め，開発が並行する．

(4) テスト・デバッグ上の特性

• タイミング検証が必要．

論理検証のほかに，ハードウェアの制御が確実にできているかのタイミング検証が必要である．

• 性能検証が必要．

応答速度に厳しい制約があるものが多く，論理検証，タイミング検証のほかに，性能検証が必要である．

• テストの実施が困難．

携帯電話のように小型・軽量のため実装面積が制約され，また，製品価格へも影響するため必要最低限の部品(メモリなど)しか搭載されない．このため，デバッグのための仕掛けを組み込むのが難しい．また，CPU のほかに周辺 I/O やメモリを 1 つのパッケージに収めた，1 チップ CPU が採用されており，デバッグが困難になる要因となっている．また，システム規模によっては，工場で実環境を構築できない場合があり，バグの再現が困難な場合がある．

4. デバッグの手法[6]

デバッグは問題解決の思考過程である．以下にデバッグの思考過程として有効な手法を示す．

(1) 帰納法によるデバッグ

糸口(バグの現象，バグの発生条件から考えられる可能性)から出発し，それらを関係づけることで，バグの原因をみつける(図-2)．

たとえば，ある入力値に対する出力値が異常な場合，いくつかの入力値を変えた場合の結果を調査する．それを関連づけて異常の原因の仮説(演算中の桁あふれなど)を立てる．仮説から導き出せた別の入力値を使用してテストを行い，同じ異常が発生することで仮説を証明する．

(2) 推論によるデバッグ

いくつかの一般的な見解や前提を列挙することから始めて，見解や前提を消去や検討することで，結論(バグの原因)をみつける方法である(図-3)．

たとえば，外部イベントを割込みで検出するシステムで，外部イベントが発生しても割込みが入らない場合，イベント検出部の異常，割込みコントローラの異常，割込みハンドラの異常などの原因を列挙し仮説を立てる．それぞれを調査し正常なもの，異常なものを明確にし仮説を証明する．

5. 組込みシステムのデバッグツール

前述のとおり，デバッグには推論・推理といった高度な人間の知的判断が必要である．推論・推理を行うのに必要な情報をとり出すための手段を提供するのがデバッグツールである．プログラムのデバッグは，プログラムを実行(1 命令ごとか，停止ポイントを設定するか，もしくは，バグが発生するまで)し，プログラムを停止(1 命令ごとか，あらかじめ設定した停止ポイントか，任意の時点か，もしくは，バグが発生するまで)して，レジスタやメモリの内容を確認し，場合により，レジスタやメモリの内容を変更して再度プログラムを実行することを繰り返し，真の原因を突き止

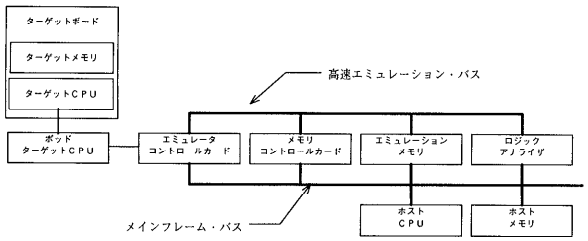


図-4 ICEの概略構成

める．したがって，デバッガに要求される機能は，

(a) プログラムの実行
(b) シングルステップ実行
(c) プログラムの停止
(d) メモリの表示(逆アセンブル)・変更
(e) 現在のレジスタの表示・変更
(f) 命令単位のプログラム実行記録

である．そのほかに，

(g) プログラムのダウンロード機能
(h) 高位言語デバッグ機能

があるとデバッグの効率がよくなる．さらにリアルタイムOSと連動して，

(i) タスクやそのほかの資源の状態
(j) システムコール(OSのサービス処理)履歴
(k) タスク切替えの履歴

が表示できるとデバッグ効率が向上する．これらのツールは単独で使用されることは少なく，状況に応じて組み合わせて使用するのが望ましい．

5.1 ICE(イン・サーキット・エミュレータ)[7]

最も強力で効果的な組込みシステムのデバッグツールである．CPUの代わりにエミュレータのプローブ挿入し実際の回路上で動作をエミュレートする(図-4)．

デバッガに要求される機能はすべて有しており，後述するほかのデバッガにはない，リアルタイムトレース(命令単位のプログラム実行記録)，ランタイムメモリアクセス(プログラム実行中のメモリ内容の表示・変更)，アクセスブレイク(特定のアドレスにアクセスした場合のプログラム停止)の機能は有用である．CPUの信号線を監視しているので，リアルタイムトレースやアクセスブレイクを確実にするためには，キャッシュを無効にしなければならない．通常，実運用時にはキャッシュを有効にするが，キャッシュ無効にすると実行タイミングが変わり，システムが正しく動作しなくなる，バグが再現しなくなるなどの可能性がありデバッグが困難になる．また，ターゲットとICEの電気的な相性が問題になる場合もある．最近のRISCチップでは高速なエミュレーションメモリの開発が遅れているためICEの開発が困難になってきている．しかし，後述のJTAGのようにデバッグ機能を有するCPUも開発されており，高速なCPUに対応しつつある．

5.2 組込みデバッガ[8),9)]

「ROMモニタ」とも呼ばれるモニタプログラムである．リアルタイムトレース，アクセスブレイクなどの機能がないが，ICEより安価である(図-5)．

ブレイクポイントはソフトウェア割込みを使用して実現するため，デバッグ対象プログラムはRAM上に存在する必要がある(ただし，ロジックアナライザと組み合わせることで，ROMにブレイクポイントを設定することも可能である．)．デバッガ自身もターゲットで動作するプログラムであるため，自身を格納するメモリ領域が必要であり，デバッガの処理(とくにステップ実行時)にCPUサイクルを消費し実行タイミングが変わる．さらに，ホストやターミナルと通信するためのI/Oポートを消費する．ほかに，デバッグ対象プログラムのバグによるメモリ破壊などでモニタ

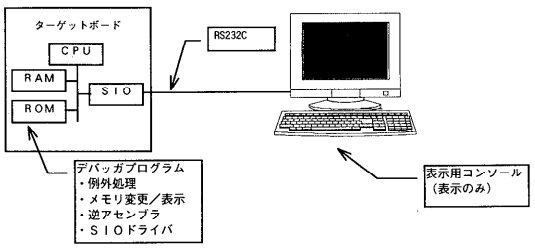


図-5 組込みデバッガ構成図

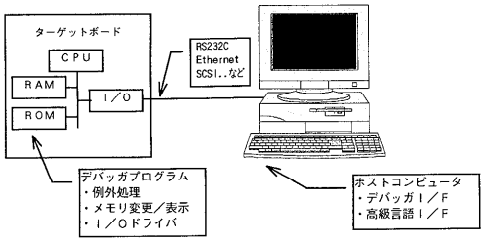


図-6 リモートデバッガ構成図

プログラム自身が暴走する可能性もある．最近では，ターゲットのメモリを少なくするため，ターゲットにはCPUおよびメモリ情報を収集する機能とホストとの通信機能のみをもたせ，デバッガのマンーマシンインタフェースをPCやEWSで実現するリモートデバッガが主流である(図-6)．

5.3 シミュレータ

EWSでターゲットとするCPUの動作環境を構築し，その上でプログラムを実行し論理的なデバッグを行う．シミュレータに疑似言語によるマクロ機能がある場合は，I/Oなどのハードウェアを含めたシミュレートも可能である．CPUのキャッシュヒット，メモリやI/Oアクセス時のウェイトを考慮することも可能であるが，EWSにかかる負荷が高くまた，ターゲットシステムに合わせたシミュレーションシステムの構築に時間がかかり，ハードウェアのタイミングによるバグのデバッグは近い将来的には可能であるが，現状では困難である．しかしハードウェア完成までにソフトウェアの論理検証が可能であるため，テスト工程を短縮できる．シミュレータ自身の負荷が大きいので，快適に処理を実施しようとすると高速なEWSが必要である．OSシミュレータを有するリアルタイムOSもあるが，これはPCやEWSのCPUのコードで動作するものである．

5.4 JTAG/EJTAG[10), 11)]

JTAG(Joint Test Action Group)は，バウンダリ・スキャン・テスト法と呼ばれており，もともとは高密度実装のため困難になったチップのテストのために提案され，1990年にIEEE1149.1として規格化された．チップ内の情報はTAP(Test Access Port)と呼ばれる5本の信号線のシリアル回線を通じてホストと通信する．

EJTAG(EnhancedJTAG)は名前のとおりJTAGを拡張したもので，

(a) JTAGの上位コンパチブル
(b) 実行中の情報の取得(リアルタイムモード)
(c) DMAアクセス対応

を規定している．リアルタイムモード実現のために6本の制御線が追加されている．

これらは，CPUの内部情報やメモリ情報をPCやEWSと通信し，用意したホストプログラムで加工してデバッグ機能を実現する．CPU内部に組み込まれているので，CPUの信号線を外部から監視する必要はない．このため，キャッシュの有効／無効やCPUのクロックに影響されることなく安定して動作する(図-7)．デバッガプログラムがメモリを占有せず，通信ポートも専用であるため，ターゲットシステムの資源を消費することもない．

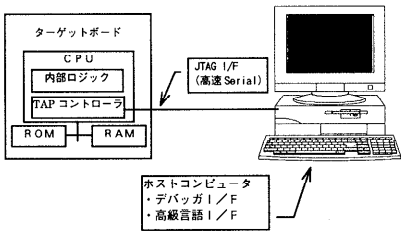


図-7 JTAGデバッガ構成図

5.5 ROMエミュレータ

ROMの形状をしたプローブをターゲットボードに差し込み，ターゲットのデバッグ情報をホストと通信する．機構はICEほど複雑でなく，プロセッサに依存しないので，ROMがエミュレートできるシステムならば容易に対応可能である．またエミュレートメモリがあるので，プログラムのダウンロード，ROMへのブレイクポイント設定が可能である．ROMだけをエミュレートしているので，ICEのようなリアルタイムメモリアクセスなどのリアルタイムデバッグ機能はもたない．また，ROMの周りのハードウェア構成が限定される．

5.6 ロジック・アナライザ

デジタル信号をチェックするための計測器であるが，CPUの信号線をすべて観測することで，高速なCPUに対応するサブセット機能のICE(リアルタイムトレース)として使用できる．最近の高速なRISCチップに対応するシステムも提案されている．ただし，CPUの信号線を監視するので，キャッシュを使用したシステムには適応できない．

5.7 汎用バスアナライザ

汎用バスを用いて複数のボードを接続した場合に，ボード間のデータ通信が正常であるかを確認するために用いる．ロジックアナライザでも代用可能であるが，バスの動作を視覚表示する(ニーモニックに置き換える)のでバスの動作が追跡しやすくなる．汎用バスとして，VMEバス，

CompactPCIバスアナライザが市販されている．

5.8 I/Oバスアナライザ

I/Oバスを用いてI/Oデバイスを接続した場合に，ターゲットボードとI/Oデバイス間のデータ通信が正常であるかを確認するために用いる．ロジックアナライザでも代用可能であるが，汎用バスアナライザと同様に，バスの動作を視覚表示する(ニーモニックに置き換える)のでバスの動作が追跡しやすくなる．I/OバスとしてSCSI，ATAバスアナライザが市販されている．I/Oデバイスドライバ作成時は必須である．

5.9 プロトコルアナライザ

通信制御プログラムのデバッグに使用する．通信回線上のビット列の流れを，使用するプロトコルに応じて視覚表示する．プロトコルスタック作成時は必須である．

5.10 ソフト検証ツール[12)]

ターゲットシステム上で実行時に

(a) システムパフォーマンス

(b) カバレッジ☆

を検証する．プログラムの品質向上のためにテスト状況を把握するのが主目的であるが，システムパフォーマンスの向上や，時間のかかる処理の発見に用いることができる．プログラムにタグと呼ぶチェックポイントを組み込み，外づけのハードウェアに状況を通知する．このため，

(1) プログラムのサイズが若干大きくなる，

(2) タグの処理時間が加算される，

(3) 出荷時はタグを外すため，テスト時とプログラムが異なる．

などの問題がある．

6. 組込みシステムのデバッグ手順

デバッグ手法は，システム規模の大小にかかわらずさほど変わりがなくICEなどのデバッグツールを使用する．また，デバッグツールを外した後は，プログラム内で履歴を採る，ステータスランプを点灯するなどして情報を残しそれを後から解析する．大規模なシステムをデバッグする場合は，大規模なシステムはいくつかの機能モジュールの集合であるので，バグの原因となる機能モジュールをトップダウンに洗出し，それぞれの機能モジュールが確実に動作しているか，各機能モジュール間のインタフェースは正しいかなどを確認し，ボトムアップ的にデバッグを行う．

また，実際のターゲットシステムでのデバッグは，システム規模にかかわらず使用するハードウェアの形態により手順が若干異なる．

(1) カスタムボードを使用する場合

組込みシステムの開発では，ハードウェアとソフトウェアの開発が並行することが多い．また，大規模な開発を行う場合は，ハードウェアが1つしかないなどの理由により，ソフトウェアのデバッグ工程が圧迫される．これを解消するために以下の手順が採られる．

(a) シミュレータ→ターゲットシステム

シミュレータ上で論理デバッグを行い，その後ハードウェアの完成後に実機でデバッグを行う．マクロ機能をもつシミュレータでは，I/Oアクセスのシミュレーションも可能であるので，大方のテスト(デバッグ)は完了する．高速なEWSがある場合は作業効率がよい．

(b) 汎用ボード→ターゲットシステム

汎用ボード(VMEボードやCPUメーカが販売している評価ボード)でデバッグを行い，その後ハードウェアの完成後に実機でデバッグを行う．ハードウェア構成が異なる場合はハードウェア依存部分のプログラムを書き直す必要がある．汎用ボードには大抵の場合リアルタイムOS(BSP(Board Support Package)やデバイスドライバつき)が用意されており，統合されたリアルタイムOSのデバッグ環境が容易に使用できる．

(2) システムに汎用ボードを使用する場合

VMEシステムの汎用ボードを使用する場合は，ハードウェアやリアルタイムOSなどの基本ソフトウェアはすでに完成されており，アプリケーションプログラムの開発に注力できる．また，前述のとおり，市販リアルタイムOSの統合環境を使用することができる．VMEボード間の相性(電気的特性の違い)により，VMEボードが正常に動作しないなどの問題が発生する場合があり，この原因解決のためには，VMEバスの規格を含め，ハードウェアの知識が不可欠である．

☆ 全プログラムコードに対する，テストで実施したプログラムコード数または，その割合．

7. 現状での問題点および動向

現状では以下の点が問題となっている．第1に，(a)CPUの高速化に対応できるデバッグツールの開発が遅れており，さらに，(b)周辺I/Oだけでなく DRAM さえも1チップ化の方向にあり，ICEによるデバッグが困難になっている．これらの問題解決のため，JTAG/EJTAG などのデバッグ機能をもったCPUが期待される．

第2に，組込みシステムにもRISCなどの高速なCPUが採用されつつあるが，これらは(a)多段階パイプライン，(b)スーパスケーラ，(c)これらのためのコンパイラの最適化技術で高性能を実現している．最近では組込みシステムも高級言語で開発し，高級言語レベルでデバッグすることが多くなっているが，CPUの細かな動きをみるためには，アセンブリ言語レベルでデバッグを行う場合がある．この時，コンパイラで最適化されたプログラムはアセンブリ言語とソースプログラムでは構造がかけ離れていることが多いため，アセンブリ言語レベルでのデバッグが困難である．アセンブリ言語と高級言語の対比をとれるなど，アセンブリ言語レベルでのデバッグをサポートするツールの登場が期待される．

第3に，CASEツールのプラットフォーム(PCやEWS)の高性能化により高度なCASEツールが出現している．これらのCASEツールでは，設計レベルでのシミュレーション機能を有している．そのため，仕様書・状態遷移表や疑似言語でのシミュレーションが可能になり，上流設計での机上デバッグが効率よく行える．その結果システムのテスト工程に至るまでに品質を織り込むことができるようになる．しかし，技術者がCPUやハードウェア構成を意識することが少なくなるため，ターゲットシステムで発生した不具合の解析は困難になる可能性がある．

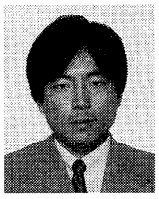
第4に，コデザイン(ハードウェアとソフトウェアの同時設計)もプラットフォームの高性能化，手法の確立，シミュレーションプログラムの高速化などにより，順次導入され始めており，すでにカスタムLSIとそれを制御するプログラムの同時シミュレーションが可能である．ハードウェアとソフトウェアを同時に設計でき，また，ハードウェアの完成前にターゲットシステム相当のデバッグが実施できるため，工程の短縮が可能である．

謝辞 執筆に際し助言をいただいた三菱電機(株)井上章氏，近藤聖久氏，(株)コア北川清康氏，資料を提供いただいた日本アプライドマイクロシステム(株)浅野義雄氏，横河デジタルコンピュータ(株)鴨林英雄氏に深く感謝いたします．

参考文献

1) 三巻達夫，桑原 洋：制御用計算機におけるリアルタイム技術（コンピュータ制御機械システムシリーズ8)，p.280，コロナ社(1986)．
2) 早川正春：ワンチップマイコンの基礎とその応用技術システム設計からデバッグまで，p.217，CQ出版(1984)．
3) Boris, B.: Software Testing Techniques（訳：小野間彰，山浦恒央，ソフトウェアテスト技法)，p.26，第10回ソフトウェア生産性向上・シンポジウム専門セミナ配布資料(1994)．
4) 鈴木裕信，加藤光明：The BUG，p.190，オーム社(1995)．
5) Ivers Peterson: Fatal Defect: Chasing Killer Computer Bugs（殺人バグを追え，訳：伊豆原弓)，p.278，日経BP社(1997)．
6) Myers, G. J.：ソフトウェア・テストの技法（訳：松尾正信)，p.192，近代科学社(1980)．
7) コンピュータ・デザイン編集部編：マイコン開発のすべて，p.243，電波新聞社(1989)．
8) Lethaby, N.：組込みシステムに最適な開発ツールの選択，New Bits，Vol.26，pp.5-10 (Spring 1995)．
9) 中島信行：68K/86系対応リモート・デバッガ，p.182，CQ出版社(1993)．
10) 坂巻佳寿美：バウンダリ・スキャン・テスト手法とその使い方，トランジスタ技術，pp.296-306 (May 1995)．
11) PHILIPS社編：EnhancedJTAG Specification，p.54，PHILIPS社(Apr. 1996)．
12) 日本アプライド・マイクロシステム社編：「RTOS開発環境pSOS/VRTX/VxWorks & Code Testソフト検証ツール」Seminar note，p.32 (Nov. 1995)．

(平成9年7月29日受付)



宿口 雅弘

1965年生．1989年大阪電気通信大学工学部応用電子工学科卒業．同年三菱電機マイコン機器ソフトウエア(株)入社．以来，リアルタイムOS，周辺I/Oドライバの開発・保守およびユーザサポートに従事．

e-mail:ms89019@hon.mms.co.jp
URL http://www.mms.co.jp/