社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－171 (16)
2007－EMB－ 3 (16)
2007／1／23

組込み計算機システム開発に向けた メモリ破壊系不具合検出機能の開発

近藤拓也† 江島賢司†† 齊藤雅彦†

多様化/複雑化する組込み計算機システム開発に関する課題として，組込みソフトウェアの信頼性向上が挙げられる．組込みソフトウェアの不具合のうち，メモリ破壊系不具合は，不具合結果とその要因（実際に意図しないメモリへアクセスした時点）との相関関係を把握することが困難であり，解析に極めて長い時間を必要とする．特に，近年の組込み向け SoC（System on a Chip）は，各種ハードウェアエンジン，マルチコアなど，並列にメモリアクセスを行う複数のハードウェアモジュールを搭載しており，メモリ破壊系不具合の要因を特定することがますます困難となっている．我々は，このような観点から，組込み SoC 自体のシミュレータを利用して実際の組込みソフトウェアを動作させ，命令実行，メモリアクセスをフックすることで，メモリ破壊系不具合を検出する機能を構築した．本機能を利用することで，各種ハードウェアエンジン，別プロセッサからのメモリ破壊を含む不具合を検出することができ，高信頼な組込みソフトウェアを短期間で開発することができる．

Method to Detect Memory Corruption for Embedded Systems Development

Takuya Kondoh,† Kenji Ejima†† and Masahiko Saito†

Dependability of embedded software has been much more important as the system becomes more diverse and complex. Memory corruption is one of the most difficult problems. The resulting failure is often far away from the error code. It is hard to detect the true cause of the memory corruption. Especially, recent SoC has more than one processor or hardware engine that may access the same memory device. It is much harder to detect memory corruption in such environment. We have developed a method to detect such memory corruption using a SoC simulation tool. It detects memory corruption by hooking instruction execution and memory access. The method makes it fast to develop highly reliable embedded software.

1. はじめに

携帯電話に代表される組込み計算機システムの発展は著しく[1]，インターネットアクセス，Java，デジタルテレビといった高機能なアプリケーションが多数搭載されるようになってきている．ソフトウェア規模が爆発的に増大しているにもかかわらず，市場が要望する多数の新機能に迅速に応えるため，開発期間を短縮させる必要がある．

開発期間の短縮に最も影響を及ぼす組込みソフトウェア開発工程の一つがテスト・デバッグ工程である．ソフトウェア品質を保つためには，一定のテスト・デバッグ工程が欠かせないが，これらを短期間で確実なものとするためには，メモリ破壊系不具合の早期発見が重要である．メモリ破壊系不具合とは，ある時点で意図しないデータの書き込みが行われることにより，そのデータを参照する別の時点で不具合が発生するものである．メモリ破壊系不具合は，不具合結果とその要因との相関関係を把握することが困難であり，解析に極めて長い時間を必要とする．

このようなメモリ破壊系不具合を簡便に検出するための方式として，メモリ確保時にガード領域と呼ばれる緩衝帯を付加する方式[2),3)]や，プログラムのオブジェクトコードを拡張する方式[4)]，独自の保護機能付きコンパイラを用いる方式[5),6)]などが開発されている．これらの方式を用いた場合，実際の組込み計算機システムにおいて利用されるオブジェクトとメモリ配置，アクセスタイミングが異なってくるため，動作タイミングなどによっては不具合を検出できない可能性がある．また，近年の組込み向け SoC は，各種ハードウェアエンジン，マルチコアなど，並列にメモリアクセスを行う複数のモジュールを搭載しているため[7)]，

† 松下電器産業株式会社 (株) プラットフォーム開発センター
Platform Development Center, Matsushita Electric Industrial Co., Ltd
†† 九州大学 情報基盤センター
Computing and Communications Center, Kyushu University

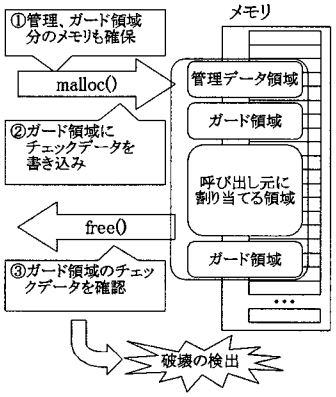

図 1 ガード領域によるメモリ破壊の検出
Fig. 1 Detecting memory corruption by guard region

これらのハードウェアモジュールに対する誤った設定が意図しないメモリ書き込みに繋がることもあり，メモリ破壊系不具合の要因を特定することがますます困難となっている．

我々は，このような観点から，組込みSoC自体のシミュレータを利用して実際の組込みソフトウェアを動作させ，命令実行，メモリアクセスをフックすることで，メモリ破壊系不具合を検出する機能を構築した．本機能を利用することで，各種ハードウェアエンジン，別プロセッサからのメモリ破壊を含む不具合を検出することができ，高信頼な組込みソフトウェアを短期間で開発することができる．

2. 関連研究

メモリ破壊系不具合を検出する簡便な方法として，MemWatch や Debug Malloc Library といった，デバッグ時のみライブラリ関数を置き換える方法がある[2),3)]．これらのツールは，ガード領域と呼ばれる緩衝帯を用いてメモリ破壊系不具合を検出する．メモリ割り当て時に，確保する領域の前後にガード領域を付加し，この領域を特定の値で埋めておく．この領域を解放する際に，ガード領域の値を検査し，内容が書き換わっていた場合に異常があったものとして警告する(図 1)．しかしながら，以下に示すような課題がある．

- ガード領域からの不正な読み出しアクセスが捕捉できない
- ガード領域以外への不正なメモリアクセスは検出できない
- 事後になってから異常があった痕跡は分かるが，不正アクセスの瞬間は捕捉できないため，別途異常の原因を調査する必要がある
- スタック領域やスタティック領域のメモリ破壊に対応していない

これらとは異なるアプローチでメモリ破壊系不具合を検出する代表的なツールとして，Rational Purify がある[4)]．Purify では，プログラムのオブジェクトコードを拡張することでメモリ破壊系不具合を検出することができる．具体的には，プログラムのバイナリを解析し，全てのメモリアクセス命令の前後に検査処理命令を挟み込み，メモリアクセスの正当性を検査する．これにより，スタック領域やスタティック領域にも対応することができ，不正アクセスの瞬間も捕捉できる．しかしながら，以下に示すような課題がある．

- メモリ容量の制限が厳しい組込み計算機システム上において，メモリを大量に使用してしまう．組込み計算機システムは必要最小限のメモリしか搭載しないため，場合によっては，検査対象のプログラムを動作させることができない．
- 検査処理命令の実行に時間を要するため，実動作とのタイミングが一致しない．特に，組込み計算機システムは，一定時間内に処理を完了させなければならない「リアルタイム性保証」を必要とすることが多く，実動作とのタイミングのずれによって，検査自体が無意味なものになる可能性がある．

一方，Saber-C[5)]，Safe-C[6)] に代表されるように，独自の保護機能付きコンパイラを用いる方式なども開発されている．例えば，Safe-C は，コンパイル時に，各データ領域に型式情報を与え，メモリアクセスを行うコードにおいてアクセス可否の検査を行うものである．Purify で検査用に変換されたオブジェクトと異なり，Safe-C でコンパイルされたオブジェクトコードには，必要最小限の検査用コードのみが挿入されているため，そのまま組込み計算機システムのソフトウェアとして活用することも可能である．しかしながら，通常の C コンパイラを利用する場合と異なり，ソースコードを Safe-C 対応に書き換える必要があるほか，オブジェクトコードのみで存在する（ソースコードで提供されない）プログラムの検査を行うことができない．

さらに，前述したように，近年の組込み向け SoC は，各種ハードウェアエンジン，マルチコアなど，並列にメモリアクセスを行う複数のモジュールを搭載しているため，これらのハードウェアモジュールに対する誤った設定が意図しないメモリ書き込みに繋がることがある．例えば，誤った DMA（Direct Memory Access）によってメモリの特定領域が破壊されたり，DSP（Digital Signal Processor）に代表される別プロセッサのプログラムバグによって，メインプロセッサの領域が破壊されたりする可能性がある．これらのメモリ破壊系不具合の要因を特定することは，上記関連技術によっては非常に困難である．

3. シミュレータを用いたメモリ破壊系不具合の検出

3.1 概　　要

Windows® CE, Linux®など，組込み計算機システムで利用できるOS（Operating System）においても，ユーザ空間で動作するアプリケーションの不正メモリアクセスを防ぐための機能（仮想記憶管理）をある程度備えている．しかしながら，これらのOSにおいても，通常，デバイスドライバはシステム内の全てのメモリ領域にアクセスすることができる．また，デバイスドライバは各種ハードウェアモジュールに対する設定等を行う特別なソフトウェアモジュールである．このため，デバイスドライバで不具合があったときの影響範囲は広く，システム全体に致命的な影響を及ぼしてしまうおそれがある．このため，我々は，デバイスドライバに代表される，下位層のソフトウェアに関しても，下記の方針でメモリ破壊系不具合を検出できるシステムを構築することを目的とした．

- ヒープ領域，スタック領域，スタティック領域全てに対応する．
- 実機でのメモリ配置やメモリ使用量を大きく変更しない．
- 不正メモリアクセスの瞬間を捕捉する．
- DMAやDSPなどハードウェアモジュール，他コアなどによるメモリ破壊系不具合も検出する．

第2章で説明したように，オブジェクトコードに検査命令を挿入して拡張する方式では実機上のメモリ使用量が大幅に増大し，メモリ配置や実動作のタイミングが変わってしまう．特別なコンパイラを利用する方法では，オブジェクトコードのみが存在するソフトウェアに対応できない．また，DMAやDSPによるメモリ破壊系不具合は，検出すること自体がこれまでの方式では実現できていない．

一方，組込み向けSoC自体が大規模・高性能化しているとはいっても，PC（Personal Computer）・サーバ系プロセッサに比べれば，低速・低機能（および低消費電力）なプロセッサである．この点を利用し，組込み向けSoCをPCでシミュレーションし，ソフトウェア開発に利用する動きがある．代表的なものとして，Texas Instruments社のOMAP ™（Open Multimedia Application Platform）プロセッサをシミュレーションするVirtio[8]，ユーザが定義するSoCのシミュレータを構築できるVirtual Platform[9]などがある．

そこで，我々は，実際の組込み計算機システムと同一のプログラムを動作させるシミュレータに，メモリ破壊系不具合検出機能を搭載するというアプローチを採った．SoCレベルのシミュレータは，プロセッサコアや各種ハードウェアモジュールなどのSoC内ハードウェアモジュールを命令セットレベルで模擬する．SoCレベルシミュレータ自体は通常のPC上で動作するアプリケーションであるため，プロセッサコアが命令を1つ実行するタイミングをフックし，そのときのレジスタやメモリの内容を参照することができる．また，同様に，SDRAM（Synchronous Dynamic Random Access Memory）やFROM（Flash Read-Only Memory）などへのアクセスをフックすることも可能である．我々は，このようなシミュレータ特有の条件を利用し，図2のメモリ破壊系不具合検出機能を構築することとした．

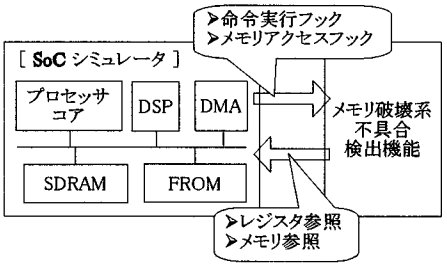

図2　メモリ破壊系不具合検出機能
Fig. 2　Method to detect memory corruption

本機能では，「命令実行フック方式」と呼ぶ機能を搭載し，プロセッサコア（ARM社製プロセッサコア[10]を想定）上で動作するデバイスドライバの動作を追跡し，メモリ破壊系不具合の検出を行う．また，「メモリアクセスフック方式」と呼ぶ機能を搭載することで，DMAやDSPによるメモリアクセスを監視し，不具合の検出を行う．以下，これらの新規開発機能について説明する．

3.2 命令実行フック方式

命令実行フック方式によるメモリ破壊系不具合検出機能では，最初に，シンボル情報とデバッグ情報を静的に解析しておく．その後，実際のコンパイル済みプログラムをシミュレータで実行する．シミュレータは，各命令実行をフックするとともに，静的解析情報を用いることで，利用されているメモリ領域を動的に特定していく．これにより，メモリアクセス命令をフックしたとき，そのメモリアクセスが，正常アクセスか不正アクセスかを判別することができる．以下，これらの処理を詳細に説明する．

3.2.1 シンボル情報とデバッグ情報の静的解析

オブジェクトファイルに含まれるシンボル情報を参照することにより，動的なメモリ破壊系不具合検出に必要な情報を事前に取得しておく．例えば，標準的なデバッグ情報の一つとして，stabsフォーマット[11]などがあり，このデバッグ情報を参照して全てのデータ型や関数の詳細な情報を得ることができる．

データ型に関しては，それぞれの型のビット長や，

構造体のメンバ配置，配列型の要素数などの情報を取得する．関数情報に関しては，関数のローカル変数や引数などがどのようにスタックフレーム内に配置されるのかといった情報を取得する．

これらの組合わせにより，スタティック領域におけるグローバル変数，スタティック変数の配置アドレスやサイズ，型情報と，スタック領域におけるスタックフレームの詳細情報を特定する．

3.2.2 確保されている全メモリ領域の識別と未確保メモリ領域アクセスの検出

シミュレータ上で実際の組込み計算機システムのプログラムを実行させる．シミュレータには，各命令実行毎にフックされ，メモリ破壊系不具合検出機能を呼び出すインタフェースを搭載する．フック時，メモリ破壊系不具合検出機能は，どのアドレスの命令が実行されるのかという情報と，実行される命令コードを受け取る．この命令実行フックにより，どの関数のどの命令が実行されているのかを特定する．関数の入口でスタックフレームを確保する命令があったときに，事前に静的解析してあったスタックフレームの詳細情報を参照することにより，スタック上に配置されるローカル変数，引数の情報を特定する．また，malloc や free といった，ヒープメモリ確保・解放関数等の引数と戻り値を追跡することでヒープ領域に確保されるメモリ領域も特定する．これらから，システム内に存在している全メモリ領域の異常アクセスを監視することができる．例えば，データ書き込み命令をフックしたときに，アクセス先のアドレスにメモリ領域が適切に割り当たっているのかを調べることで，未確保メモリ領域アクセスやスタック境界越えアクセスのような不正アクセスを検出することが可能となる．

3.2.3 メモリ破壊系不具合検出方法

メモリ破壊系不具合の検出について，配列境界越えを中心に説明する．例えば，C 言語で次のような記述を考える．(ここで，array は int 型とする．)

$array[i] = x;$

このとき，この配列 array の先頭アドレスと，実際にアクセスするアドレスが異なるメモリ領域であった場合，配列境界越えアクセスであると判断して警告を行えば良い．しかしながら，ポインタを利用した場合など，配列境界越えアクセスは実際のコードを動作させないと発見できないため，コンパイラで静的に検出することはできない．このため，我々は，シミュレータで実際のプログラムを動作させてメモリ破壊系不具合を検出しようとしている．ところが，コンパイラによってアセンブリに変換された後の書き込み命令コード（図 3）を参照すると，配列の先頭アドレスとオフセットが加算された値がレジスタにまとめられてしまい，書き込み命令を発行するタイミングでは配列の先頭と添字が区別できるようにはなっていない．このままでは，実際にコードを動作させたとき，単に書き込み命令だけを監視しても，それが配列境界越えアクセスなのか，それとも正常なメモリアクセスなのかを判別することができない．

コンパイルされたアセンブリコード

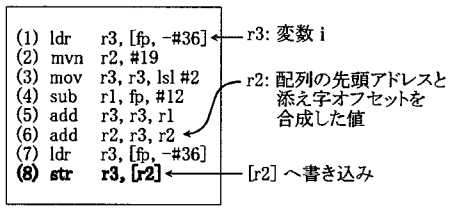

図 3 例：array[i] = x; のコンパイル結果
Fig. 3 Example: compiled code of array[i] = x;

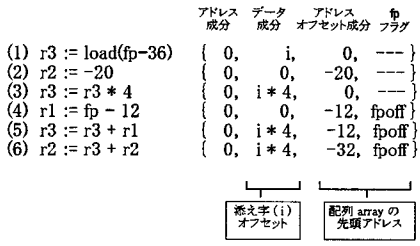

図 4 配列の先頭と添え字を区別
Fig. 4 Distinguishing array head from index

そこで，本機能では命令実行フックを利用して，関数の入り口からどのような処理が行われたのかを追跡し，それぞれのレジスタやメモリ領域に格納されている値の内訳を記録する．具体的には，各レジスタやメモリ領域ごとに，そこに格納するデータのアドレス成分，データ成分，アドレスオフセット成分，fp (frame pointer) オフセットか否かのフラグ，といった詳細情報を設ける．これにより，レジスタに格納されるデータがどういった詳細情報を持ったデータなのかを，実行する命令に応じて区別して記憶する．

例えば，メモリからレジスタへデータを読み込む命令をフックした場合，そのメモリの型情報を調べ，ポインタであれば読み込んだ値をアドレス成分値として記憶し，整数型であれば読み込んだ値をデータ成分値として記憶する．また，レジスタに即値を格納する場合は，即値をアドレスオフセット成分値として記憶する．レジスタ間での演算時においては，詳細情報の各項目ごとに独立してデータを演算し，異なる詳細情報が混ざらないように記憶する．

具体的な例として，図 3 のアセンブリコードのうち (1) ～ (6) の命令を実行するときにおけるレジスタ追跡の様子を図 4 に示す．

(1) このメモリ読み込み命令では，読み込み先メモリの型情報から，そこに格納されたデータがア

ドレスでなく整数であることを判別し，レジスタのデータ成分の項目に読み込んだ値を格納する．

(2) この1の補数の転送命令では，レジスタに即値の1の補数を転送するため，アドレスオフセット成分の項目に値を格納する．

(3) この転送命令では，レジスタの値を4倍して転送するため，データ成分の項目を4倍する．

(4) この減算命令では，レジスタにfpオフセットのアドレスを格納するため，fpオフセットのフラグをセットし，オフセット値をアドレスオフセット成分の項目に格納する．

(5) この加算命令では，レジスタ間で値を加算するため，レジスタの各項目の値を項目ごとに加算する．

(6) この加算命令では，(5) の加算命令同様の処理を行う．

以上のような追跡により，命令（8）でメモリに値を書き込む際，書き込み先アドレスを格納しているレジスタ r2 にどういった詳細情報が格納されているのかを判別できる．この後，読み書き命令をフックしたときに，レジスタが格納している配列の先頭アドレスと添え字を参照し，配列の境界を越えていると判断したとき，不正メモリアクセスとして検出する．

3.3 メモリアクセスフック方式

近年の組込み向け SoC は，高機能化に伴い，各種ハードウェアエンジン，マルチコアなど，並列にメモリアクセスを行う複数のハードウェアモジュールを搭載している．当然，これらのハードウェアモジュールは，それぞれ独立してメモリにアクセスするため，これらのハードウェアモジュールに対する誤った設定が意図しないメモリ書き込みに繋がることがある．これらは2章で示したソフトウェアのみの技術では検出できない不具合である．我々は，これら CPU 以外のバスマスタによるメモリ破壊系不具合を検出するため，同様に，SoC レベルシミュレータを利用する「メモリアクセスフック方式」を開発した．

DMA や DSP など，CPU 以外のバスマスタによるメモリ破壊系不具合は，あらかじめ各ハードウェアモジュールのアクセス可能なメモリ領域を特定し，ハードウェアモジュールのメモリアクセスフック時に，その特定しておいたメモリ領域と比較，検証することで検出する．メモリアクセスフック方式の概要を図5に示す．本方式では，常にアドレスが一定で固定となる静的な領域，アドレスが固定でない動的な領域に分けて，個々のハードウェアモジュールがアクセス可能なメモリ領域を特定する．

前者は，アクセス可能なハードウェアモジュール，開始アドレス，サイズ，といった情報をあらかじめメモリ破壊系不具合検出機能側に与えておくことで特定する．

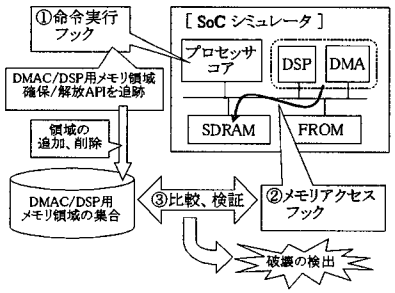

図5 メモリアクセスフックによるメモリ破壊系不具合の検出
Fig. 5 Detecting memory corruption by hooking memory access

後者は，命令実行フック方式を用いて特定する．DMA などのハードウェアモジュールは，メモリ領域を使用する際，メモリ領域を確保する API (Application Programming Interface) を実行する．そこで，この API を命令実行フック方式で追跡し，メモリ領域を特定する．メモリ破壊系不具合検出機能側にはあらかじめ，アクセス可能なハードウェアモジュール，メモリ領域を確保する API，メモリ領域を確保する API の呼び出し元，といった情報を与えておく．メモリ領域を確保する API が呼び出され，呼び出し元がその情報と合致したら，API の引数や戻り値から，確保されたメモリ領域の開始アドレス，サイズを特定する．

メモリ破壊系不具合の検出方法は，静的領域でも動的領域でも同一である．この場合，メモリアクセスをフックし，ハードウェアモジュールがどのメモリ領域に対して書き込みを行なったのかを調べる．アクセスされたメモリ領域が，事前に記憶していたハードウェアモジュール用のメモリ領域の範囲を越えていないかを検証することで，ハードウェアモジュールによるメモリ破壊系不具合を検出する．

4. 結　　果

本研究で開発したメモリ破壊系不具合検出機能により，配列境界越えアクセスや未確保メモリ領域へのアクセス，スタック境界越えアクセスなどの不正アクセスを検出することができる．図6に配列境界越えアクセスを引き起こすプログラムの例を示す．このプログラムを実行し，メモリ破壊系不具合検出機能によって配列境界越えアクセスを検出した結果を図7に示す．このように，配列 buf の境界を越えてアクセスしたとき (i = 5 の時点) に，その異常を検出し，エラーメッセージを通知することができる．

5. まとめと今後の展開

我々は，ソフトウェア品質を保つため，メモリ破壊

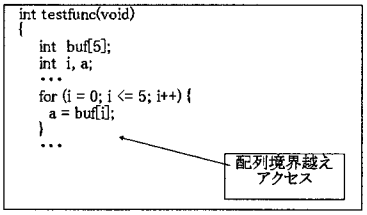

図 6 配列境界越えプログラムの例
Fig. 6 Sample program that involves illegal array access

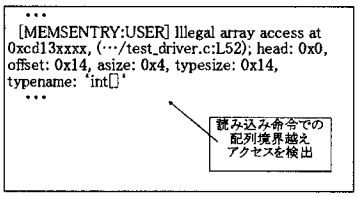

図 7 配列境界越えアクセスの検出結果
Fig. 7 Log message that shows detecting illegal array access

系不具合の早期検出ツールを開発，評価中である．特に，近年の組込み向け SoC は，各種ハードウェアエンジン，マルチコアなど，並列にメモリアクセスを行う複数のモジュールを搭載しているため，これらのハードウェアモジュールに対する誤った設定が意図しないメモリ書き込みに繋がることもあり，メモリ破壊系不具合の要因を特定することがますます困難となっている．

我々は，このような観点から，組込み SoC 自体のシミュレータを利用して実際の組込みソフトウェアを動作させ，命令実行，メモリアクセスをフックすることで，メモリ破壊系不具合を検出する機能を構築した．本機能を利用することで，各種ハードウェアエンジン，別プロセッサからのメモリ破壊系不具合を含む不具合を検出することができ，高信頼な組込みソフトウェアを短期間で開発することができる．

現在，具体的な組込み計算機システムへの適用を試行中であり，今後，これらのツールを核とした自動検証システムなどへの展開を行っていく予定である．

参 考 文 献

1) 高田広章, 組込みシステム開発技術の現状と展望, 情報処理学会論文誌, Vol.42, No.4, pp.930–938, 2001.
2) Lindh, J., MemWatch Version 2.7.1, 2003.
3) Watson, G., Debug Malloc Library Version 5.4.2, 2004.
4) Hastings, R. and Joyce, B., Purify: Fast Detection of Memory Leaks and Access Errors, Proceedings of USENIX Winter '92 Conference, pp.125–136, 1992.
5) Kaufer, S. et al., Saber-C: An Interpreter-Based Programming Environment for the C Language, Proceedings of USENIX Summer '88 Conference, pp.161–171, 1988.
6) Austin, T. et al., Effcient Detection of All Pointer and Array Access Errors, ACM SIGPLAN Notices, Vol.29, No.6, pp.290–301, 1994.
7) 清原督三ほか, ソフトウェア開発効率を重視したデジタル家電向けメディアプロセッサを開発, 日経エレクトロニクス 2004 年 10 月 11 日号, No.884, 2004.
8) Virtio, Corp., System Verification with Virtio Virtual Prototyping Technology, Virtio White Paper, 2004.
9) CoWare, Inc., Virtual Platforms for Software Development – Adapting to the Changing Face of Software Development, CoWare White Paper, 2005.
10) ARM Limited, ARM アーキテクチャリファレンスマニュアル, 2005.
11) Menapace, J. et al., The "stabs" debug format, Cygnus Support, 2005.