特集論文

仮想実行環境の構築容易化技術と応用

奥田勝己* 伊藤真二**
竹山治彦* 町田宏隆***
三輪昌伸**

Automated Generation of Virtual Execution Environments and its Applications

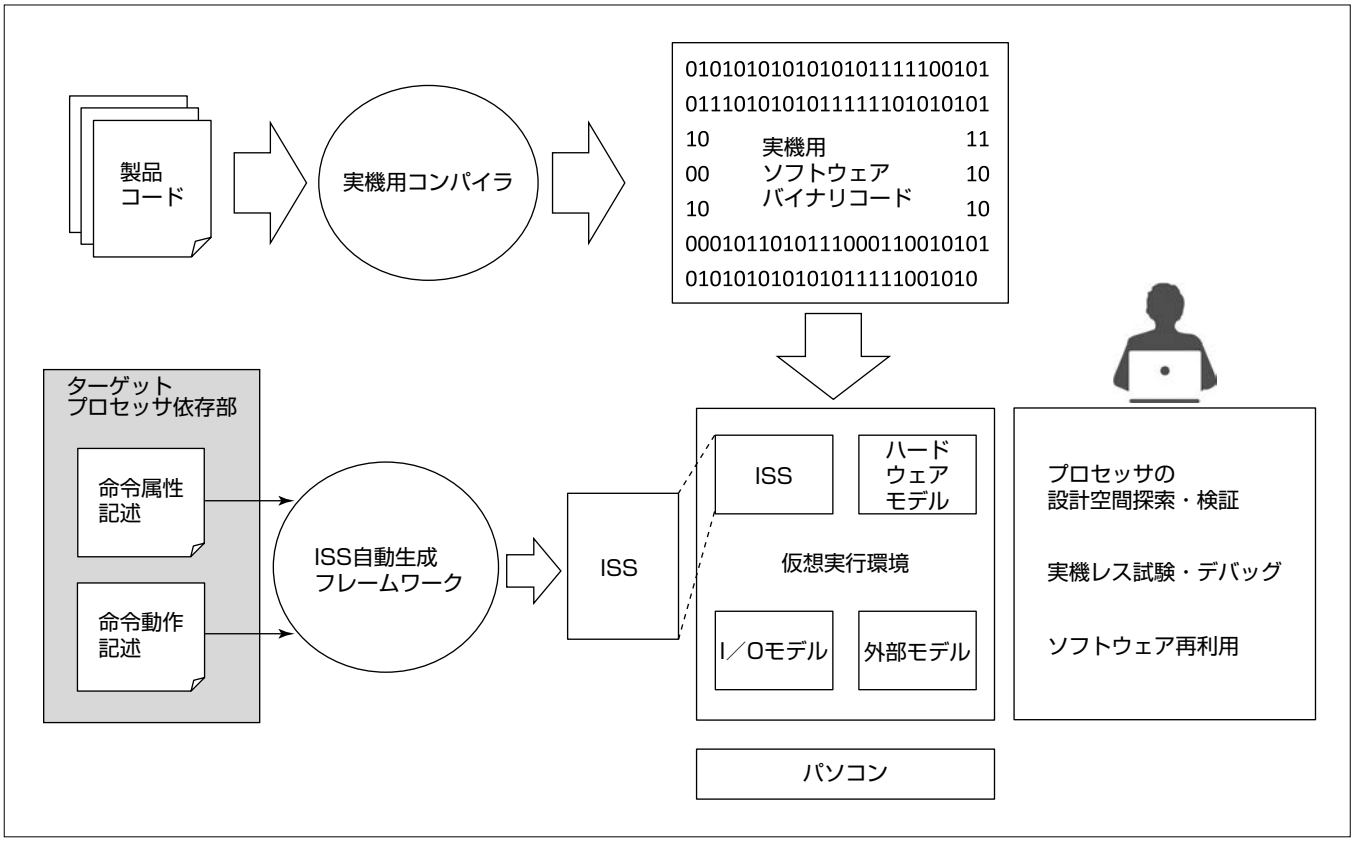
Katsumi Okuda, Haruhiko Takeyama, Masanobu Miwa, Shinji Ito, Hirotaka Machida

要 旨

命令セットシミュレータ(Instruction Set Simulator：ISS)は，ターゲットプロセッサの動作を模擬するソフトウェアである。ISSは組み込みシステムの開発で多目的に利用が可能であり，重要な開発ツールの１つである。例えば，プロセッサ開発者はプロセッサの設計空間探索・検証にISSを利用する。また，ソフトウェア開発者は，ISSを組み込んだ仮想実行環境を用いることで，実機の開発と並行してソフトウェアの試験・デバッグを行う。

これらの用途ではISSが組み込みシステムの開発早期から利用可能であることが要求されるため，ISS自体を効率的に開発することが課題である。そこで，三菱電機ではISS自動生成フレームワークを開発することでISSの構築を効率化した。ISS自動生成フレームワークは，ターゲットプロセッサ依存の命令属性記述と命令動作記述を主な入力とし，高速なISSを生成することができる。

ISS自動生成フレームワークで構築したISSの代表的な応用事例として，内製プロセッサの検証への適用と人工衛星運用手順検証用衛星シミュレータへの適用の２つが挙げられる。内製プロセッサの検証では，生成されたISSを期待値モデルとして利用している。また，衛星シミュレータへの適用では，生成された高速ISSをシミュレータに搭載することで実衛星のソフトウェアバイナリコードを再利用し，シミュレータ上で直接実行する。このため，衛星の高精度なシミュレーションを行うことができ，運用手順の確実な検証を実現している。



ISSの自動生成と利用

ISSを組み込んだ仮想実行環境は，実機用ソフトウェアバイナリコードを実機レスで実行することができる。組み込みシステム開発では仮想実行環境を用いることで，プロセッサの設計空間探索・検証，ソフトウェアの実機レス試験・デバッグなどを行うことができる。開発したISS自動生成フレームワークでは，ターゲットプロセッサごとの命令属性記述と命令動作記述からISSを自動生成することが可能である。

*先端技術総合研究所 **名古屋製作所 ***鎌倉製作所

1. まえがき

プロセッサの動作を模擬するISSは，組み込みシステムの開発で多用途に適用可能な重要なツールである。例えば，プロセッサ開発者はプロセッサの設計空間探索・検証にISSを利用する。また，組み込みソフトウェアの開発者はISSを組み込んだ仮想実行環境を用いることで，実機レスでソフトウェアの試験・デバッグを行うことができる。特に仮想実行環境を用いた試験では，実機での実現困難な故障注入試験も容易に行うことができる。さらに，ISSを製品に組み込むことで，ソフトウェアの再利用が容易となる。プロセッサAの製品にプロセッサBを模擬するISSを組み込むことで，プロセッサB用のソフトウェアを移植することなく再利用することができる。

組み込みシステム開発へのISS適用における課題は，ISSの生産性と実行速度である。組み込みシステムで使用されるプロセッサの種類は，市販品・内製品を含めて多種多様である。製品や機種ごとにプロセッサが異なる組み込みシステムで広くISSを用いるためには，ISSの効率的な構築手法が必要である。また，ISSはシミュレーション速度が高速であるほど適用範囲が広い。例えば，ISSを用いてソフトウェアを再利用するためには，ISSが十分に高速である必要がある。

ターゲットが市販プロセッサの場合，サードパーティ製のISSやQEMU[(1)]などのOSS(Open Source Software)の利用も考えられる。しかし，それらのISSではプロセッサの種類ごとに調達性が異なることや，性能・品質面でばらつきがあることが問題となる。

そこで当社では，ISS自動生成フレームワーク(以下“ISS生成フレームワーク”という。)を開発し，高性能なISSを効率的に構築できるようにした。ISS生成フレームワークは，プロセッサ記述を入力とし，インタプリタ方式及び動的バイナリ変換方式のISSを構成する主要なコードを自動生成することで，高速・高品質なISSを出力することができる。

本稿では，ISSの実行方式としてインタプリタ方式と動的バイナリ変換方式を述べ，両方式のISS自動生成に対応したISS生成フレームワークについて述べる。また，ISS生成フレームワークの適用事例についても述べる。

2. ISSの実行方式

2.1 インタプリタ方式

古典的なISSの実行方式はインタプリタ方式である。インタプリタ方式のISSがターゲットプログラムを実行する処理フローを**図1**に示す。ISSは，プログラムの終了条件が成立するまで1命令ずつフェッチ，デコード，実行を繰り返すループ処理として構成される。命令フェッチステップではプログラムカウンタが示すアドレスから命令語を読み出す。命令デコードステップでは，フェッチした命令の種類を特定し，命令オペランドを示す命令フィールドを抽出する。命令実行ステップでは，命令デコードステップで特定した命令種類ごとの処理を，抽出した命令フィールドの値をパラメータとして実行する。

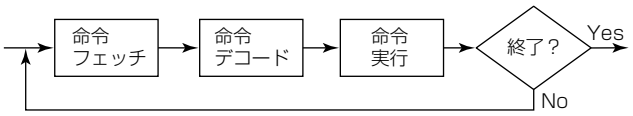


図1．インタプリタ方式の処理フロー

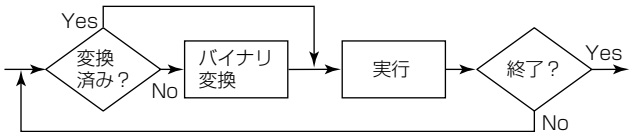


図2．バイナリ変換方式の処理フロー

インタプリタ方式のISSは，プロセッサの機能面での動作を素直に表したモデルであり，記述が比較的容易である。しかし，ターゲットの1命令を数十から数百のホスト命令で模擬実行するため，処理速度は低速である。

2.2 動的バイナリ変換方式

動的バイナリ変換方式(以下“バイナリ変換方式”という。)のISSは，ターゲットプログラムの実行時に命令列をホスト命令列に変換してから実行する。変換で得られたホスト命令列は，後で再利用が可能なように，変換元ターゲット命令のアドレスと対で変換キャッシュと呼ばれるメモリ領域に保存される。バイナリ変換方式のISSは，再度同じターゲット命令のアドレスを実行する場合，変換キャッシュから変換済みのホスト命令列を取得し，変換済み命令を実行する(**図2**)。

バイナリ変換方式では，ターゲット命令の実行ごとに命令フェッチとデコードを行う必要がないため，インタプリタ方式と比較して高速にターゲットプログラムを実行できる。

3. ISS生成フレームワーク

ISS生成フレームワーク(**図3**)は，ターゲットプロセッサ依存部の命令属性記述と命令動作記述を主な入力とし，ISSを出力する。生成対象となるISSは，インタプリタ方式及びバイナリ変換方式のISSである。入力の命令属性記述は命令ごとのフォーマットや分岐命令か否かなどの属性を記述したファイルである。一方，命令動作記述は命令ごとのふるまいを記述したファイルである。

生成されるISSの主な構成要素は命令デコーダ，インタプリタ，動的バイナリトランスレータである。命令デコーダは，インタプリタ方式及びバイナリ変換方式のISSで共通のモジュールである。また，インタプリタとバイナリトランスレータはそれぞれインタプリタ方式ISSとバイナリ

変換方式ISSのモジュールである。

3.1 命令デコーダの自動生成

一般に命令のデコード処理は，デコーディングツリーで表現可能である。デコーディングツリーの各辺はビットパターンをラベルとする。また，リーフは命令の一意な識別子をラベルとする。

命令デコーダは，デコーディングツリーのルートから開始し，リーフに到達するまで子ノードを巡回することで，命令語をデコードする。次の巡回対象ノードは，命令語と一致するビットパターンをラベルとする辺でつながった子ノードである。命令デコーダの性能は使用するデコーディングツリーの深さに依存する。すなわち，ツリーが浅いほど命令デコーダは高速である。

当社では，**図4**の模式図のように命令属性記述に含まれる命令フォーマットを入力とし，デコーディングツリーを生成するアルゴリズムを開発した[2]。このアルゴリズムでは，変則的な命令セットにも対応可能であり，従来アルゴリズム[3]との比較で22%から40%の浅いツリーを生成することが可能である。

3.2 インタプリタの自動生成

ISS生成フレームワークでは，命令動作記述中の処理を補完するコードを生成することで，人手によるISSの記述量を削減する。ISS生成フレームワーク適用前後での記述の比較を**図5**に示す。

ISS生成フレームワークを用いない場合，命令語から命令ワードを抽出する処理①や，プログラムカウンタを更新する処理②の記述が必要になる。一方，ISS生成フレームワークを用いた場合，処理①と処理②のコードは自動生成されるため，記述が不要である。ISS生成フレームワークは，参考文献(4)に記載されている手法でISSを自動生成する。オープンソースのGDB(Gnu DeBugger)との比較では，40%少ない記述量でARM(注1)用のISSを生成できることを確認した。

(注1) ARMは，ARM Ltd. の登録商標である。

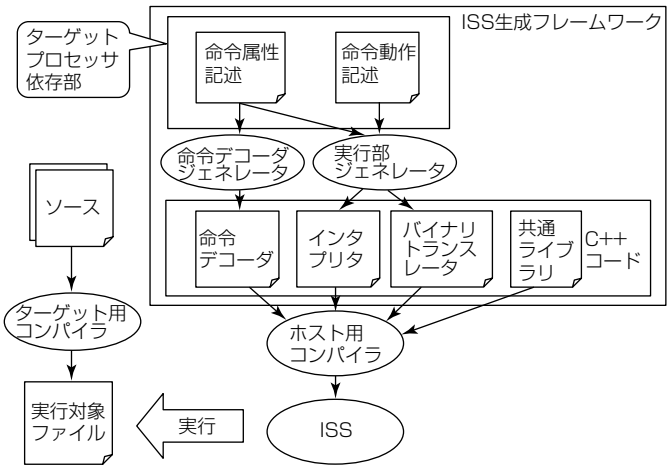


図3．ISS生成フレームワーク

3.3 バイナリトランスレータの自動生成

一般にバイナリ変換方式ISSは複雑で開発に労力を必要とする。バイナリ変換時における1命令変換の様子を**図6**に示す。ISSがターゲット命令daddをホスト命令列に変換する場合，daddに対応するホスト命令列の変換テンプレートに対して，変換時に確定する値をパラメータに代入することで目的のホスト命令列を得る。しかし，ISS開発者が変換テンプレートを作成するためには，ターゲットの命令セットだけでなくホストの命令セットも習得する必要がある。

ISS生成フレームワークでは，ISS開発者がホスト命令を直接操作することなくバイナリトランスレータを記述可能としている[5]。ISS生成フレームワークは，インタプリタ用に記述した命令動作記述をホストネイティブコンパイラでオブジェクトファイルにコンパイルし，当該オブ

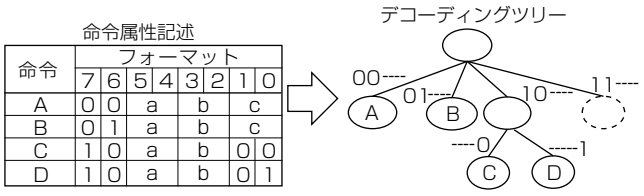


図4．命令デコード処理のデコーディングツリー

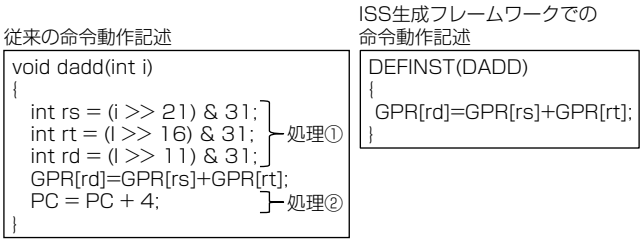


図5．命令動作記述の比較

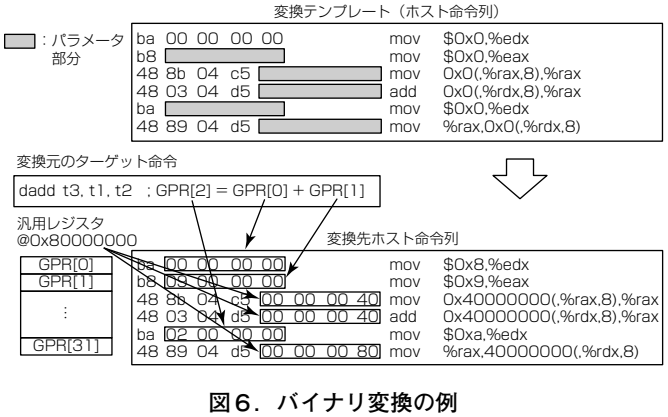


図6．バイナリ変換の例

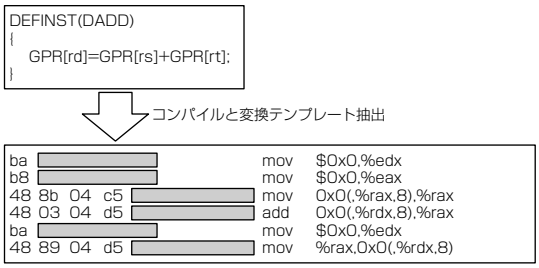


図7．変換テンプレートの抽出

ジェクトファイルから変換テンプレートを抽出することで，バイナリトランスレータを自動生成する（**図7**）。ARM，SH(注2)，MIPS64(注3)をターゲットにベンチマークプログラムDhrystoneとCHStoneを用いた実験評価では，ベースとなるインタプリタに対して1.4倍から13.4倍の範囲で高速化することを確認した。

（注2） SHは，ルネサス エレクトロニクス㈱の登録商標である。
（注3） MIPS64は，Imagination Technologies LLCの登録商標である。

4. 適 用 事 例

4.1 内製プロセッサの検証

FA(Factory Automation)用コントローラでは，処理性能が重要なポイントである。そこで，当社では，FA用の演算に特化した専用プロセッサを開発し，製品に搭載している。プロセッサの品質は製品全体の品質に大きな影響を与えるため，プロセッサの検証は特に重要である。このため，プロセッサの検証では大量の検証パターンをプロセッサに入力し，正しい出力が得られるか否かの確認が行われる。検証パターン数は膨大であるため，プロセッサの出力が正か否かを人手で判定することは困難である。そこで，ISSを期待値モデルとして利用することで，効率的に検証を進めることが可能となる。**図8**はISSを期待値モデルとして用いたプロセッサ検証を示している。検証パターンを実プロセッサとISSに入力し，実プロセッサの出力とISSが出力した期待値を一致比較することで，検証ケースの合否を判定する。

プロセッサ検証時にISSを期待値モデルとして利用とするためには，実プロセッサの開発と並行してISSを開発する必要がある。そこで，ISS生成フレームワークを適用することでISSを短期開発し，プロセッサ検証時にISSを利用可能としている。

4.2 人工衛星運用の事前検証

人工衛星は，運用ミス等で一時的に可用時間が失われるだけでも大きな損失となることから，送信するコマンドシーケンスや搭載ソフトウェアバイナリコードのパッチ修正などを，衛星シミュレータで事前に検証することが一般的である。当社でも，人工衛星とともに衛星シミュレータ（**図9**）を提供しているが，衛星シミュレータには衛星搭載ソフトウェアが組み込まれており，ISSは衛星シミュレータを実現するためのコア技術の1つとなっている。

この衛星シミュレータでは，用途が事前検証であることから高速シミュレーションの要求は強い。また，衛星によっては15年以上の長期にわたって運用を行うことから，衛星シミュレータのプラットフォーム(OSやプロセッサ)変更に対応した長期サポート・ポーティングも要求される。一方，衛星搭載プロセッサやそのISSはニッチであることもあり，これらの要求に応えるISSを外部ベンダーに期待することは難しい。

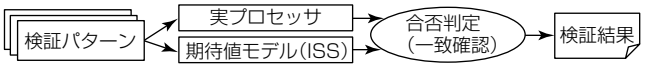


図8．ISSを期待値モデルとしたプロセッサ検証

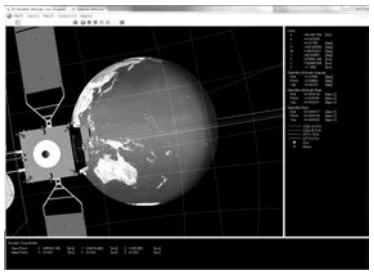


図9．人工衛星シミュレータの3D確認画面

ISS生成フレームワークはこれらの要求に応えるものであり，バイナリ変換方式のISS生成によって衛星シミュレータのシミュレーション速度はリアルタイム比10倍以上を実現，また，自社でISSを生成することによって外部ベンダーに依存しない長期サポート・ポーティングを可能としている。

5. む す び

ISSは，プロセッサの設計・検証，組み込みソフトウェア開発，ソフトウェア再利用などに不可欠なツールである。組み込みシステム用途ではプロセッサの種類が多種多様であるため，ISSの効率的な開発が課題である。

本稿では，ISSを効率的に構築するためのISS生成フレームワークとその適用事例について述べた。ISS生成フレームワークは，ターゲットプロセッサごとに用意した記述からISSの主要構成要素である命令デコーダ，インタプリタ，バイナリトランスレータを自動生成可能である。

参 考 文 献

(1) Bellard，F.：QEMU，A Fast and Portable Dynamic Translator，2005 USENIX Annual Technical Conference，41～46 (2005)
(2) Okuda，K.，et al.：Decision tree generation for decoding irregular instructions，2016 Design，Automation & Test in Europe Conference & Exhibition(DATE)，1592～1597 (2016)
(3) Fournel，N.，et al.：Automated generation of efficient instruction decoders for instruction set simulators，2013 IEEE/ACM International Conference on Computer-Aided Design(ICCAD)，739～746 (2013)
(4) 奥田勝己，ほか：命令セットシミュレータ生成フレームワークの設計と実装，情報処理学会論文誌，**57**，No. 8，1718～1736 (2016)
(5) Okuda，K.，et al.：Automated Generation of Dynamic Binary Translators for Instruction Set Simulation，2017 22st Asia and South Pacific Design Automation Conference(ASP-DAC)，to appear (2017)