情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-ARC-184 No.29
2009/8/5

論理設計検証テストプログラム生成ツールの開発と評価方式

元安　優[†1]　　山下　純一[†1,†2]　　北村　俊明[†1]

プロセッサの多機能化や高性能化によって論理設計が複雑になっており，設計者があらゆる実行状況を設定して論理シミュレーションを行い検証することは不可能である．そこで，プロセッサの実装を進める上で必要となる，プロセッサの論理設計を検証するためのテストプログラムの開発を行った．また，テストプログラムによる検証をより効率の良いものにするため，テストプログラムの評価方法について検討を行った．今回開発した 2 種類のランダムテストを用いることで全体の検証率の向上が期待できることが示された．ハードウェア内の制御信号に着目するという評価方式は，テストプログラムを評価する上で良い指標になることがわかった．さらに，設計ソース，制御信号の遷移状態を解析することで状態数の上限値を算出し，テストプログラムが全体の何割程度検証できたかを求めることができるようになることがわかった．

Development and Evaluation of Generator Test Program for Logic Design Verification

Yu Motoyasu,[†1] Junichi Ymashita[†1,†2]
and Toshiaki Kitamura[†1]

Logic design of processor becomes complicated by many functions and the high performance of the processor. So, it is impossible that we test all instructions to set up all execution situations on a logic simulation. In implementing a processor, the test program for verifying the logic design of that is indispensable. In this research, we developed the several types of test programs, to overcome this situation. Moreover, in order to verify processor by a test program efficiently, we examined the evaluation system of a test program. Using this method, we can characterize the two kinds of random tests, that we prepared. As a result, we show the verification rate of the whole processor improves by using both. The evaluation system is a method of observing the control signals within the hardware. The evaluation system is a good barometer of the test program. Moreover, we calculate the numerical upper limit by analyzing a design source. We can compute the rate of verification progress by using this value.

1. はじめに

近年，数多くの電化製品に制御用の組み込みシステムが搭載され，高性能と低消費電力の両立が求められている．この課題に対して，我々は VLIW プロセッサの実行効率のよさに着目し評価[1)] を行ってきた．我々は異種命令セット SMT(Simultaneous Multi-Threading) プロセッサ OROCHI[2)] を提案し設計開発を行ってきた．プロセッサの設計を行った時に，仕様書通りに論理設計できているか検証する必要がある．しかし，プロセッサの多機能化や高性能化によって論理設計が複雑になっており，全ての命令に対してあらゆる実行状況設定して論理シミュレーションを行い，検証することは不可能である．そこで，実行状況の設定を行い，実行結果を比較することで論理設計の検証を行うテストプログラムの開発を行った．各種仕様書に記載されているプロセッサが満たすべき性質を検証するファンクションテストと，その他エンジニアが想定し忘れているテストケースなど洗い出すためのランダムテストを 2 種類作成しプロセッサの論理設計検証に努めた．

従来のランダムテストの運用は，検証終了時期を見極める判断要素がなく終了時期を保証することがきなかった．そこで，その判断要因にすべく，ランダムテストを実施することでどれだけの検証を行えたのかを調査した．具体的にはプロセッサがどのような動きや状態になったかを調べ，その結果をテストプログラムの評価とした．状態を定量化するための指標として，制御信号に着目した．この制御信号を回路の出力を決定づける要因となる信号線と定義する．その信号が回路の状態を形成する要素だと判断し，複数の信号線の論理がその回路の状態だと考える．テストプログラム実行時にどれだけの状態が出現するかを評価尺度とする．提案した評価尺度を基に，ランダムテストの終了時期を検討できる判断材料になり得るか検討を行う．

以降，第 2 章では，今回開発したテストプログラムについて述べる．第 3 章では，テストプログラムを評価するための評価方式について述べる．第 4 章では，測定方法や検証対象のプロセッサについて述べる．第 5 章では，テストプログラムの評価，今回行った評価方式について考察を行う．第 6 章において，まとめとする．

†1 広島市立大学
Hiroshima City University
†2 現在，ローム株式会社
Rohm Corporation

2. テストプログラム

本章ではそのテストプログラムについて説明を行う．テスト対象となった命令セットアーキテクチャは，ARMv4 命令セット[3](以降，ARM 命令) と FR-V FR550 命令のサブセット[4](以降，FRV 命令) を対象とした．

2.1 ファンクションテストテストプログラム

アーキテクチャ仕様で明示されている部分の設計を検証するテストプログラムをファンクションテストプログラム[5][6] と呼ぶ．検証対象のプロセッサのマイクロアーキテクチャが仕様通りに動作していることを，このプログラムを用いて検証する．

論理設計検証を目的とするテスト命令列を実行しその正確性を検査するためには，必要な命令が使用範囲内で正常に動作していることを確認している必要がある．比較命令の動作が不確かのままではそれを用いた結果確認処理は信頼できず，分岐命令の動作が不確かのままでは比較命令の後の一致不一致判定は信頼できないことになる．そのために，テスト実施に必要な命令の動作を最低限検証した後，より詳細な検証や他の命令の動作の検証を実施する．

2.2 ランダムテストプログラム

エンジニアが想定していないケースの洗い出しを行う．本研究では 2 種類のランダムテストを作成した．VLIW ランダムテストとナゲットランダムテストを作成し，順に説明を行う．

2.2.1 VLIW ランダムテスト

一般的に用いられやすい生成方法である．1 命令ごとにオペコードおよびオペランドをランダム決定するランダムテストを作成して検証を行う．

VLIW ランダムテストは，1 命令ごとにオペコードおよびオペランドをランダムに構成し，それらの命令を複数個まとめることでランダムな VLIW 命令を生成し実行するテストである．ただし，プロセッサがサポートしない命令は生成しない．また VLIW 命令には発行制限があるため，制限から逸脱しない VLIW 命令を作成する．本来なら制限から逸脱したケースも作成し，割り込みなどの例外処理が発生した場合の検証を行う必要性があるが本研究では行っていない．出現する可能性がある命令は，制御命令を除く全てが対象となる．しかし，ランダムな入力で実行することが好ましくない命令は，その入力を事前に挿入することで不都合を回避する．該当する命令は，メモリアクセス命令と分岐命令である．メモリアクセス命令では，アクセスを許可する領域だけにアクセスするようにアドレス値を制限し，分岐命令ではコード領域以外に分岐したり永久ループが発生したりしないように分岐先を制限する．

本研究で作成した VLIW ランダムテストは，2 種類のタイプに分類できる．決められた命令スロットのみを必ず使用しパッキングする命令数を固定化した固定長 VLIW 命令方式と，仕様スロットを限定せず任意の命令数をパッキングした VLIW 命令でプログラムを構成する可変長 VLIW 命令方式である．パッキングする命令数を固定にした理由は，段階的にランダムテストの難易度を上げていくためにこのような方法を採った．複雑なケースを含んだテストで複数のバグが同時に発生することは，デバッグの容易性を損なうと考えたからである．

固定長 VLIW 方式

あらかじめ決まった数の命令数をパッキングして VLIW 命令にするテストでは，スロットの中から使用するスロットを決定し，そのスロットにだけ命令を発行する VLIW 命令を生成しテストを実施する．

可変長 VLIW 命令方式

任意の命令数をパッキングして VLIW 命令にするテストでは，VLIW 命令ごとに，使用するスロットを変更して VLIW 命令を生成する．1 命令から最大 8 命令をパッキングした VLIW 命令を多数実行するテストである．

命令ブロックを複数個連結することでテストプログラムを作成している．定期的にレジスタをランダムにリセットしつつ，ランダムに生成した命令を連続で実行しその結果を確認する．

2.2.2 ナゲットランダムテスト

前述の VLIW ランダムテストでは，1 命令ごとにオペランドをランダムに決定するため，命令間のデータ依存関係など発生しにくい傾向にある．そこで，データ依存を明示的に発生させたり，割り込みのタイミングをずらすタイプのランダムテストがナゲットランダムテストである．

ナゲットランダムテストは，事前に用意した一連の命令シーケンスを一塊 (ナゲット) として，ナゲット単位でランダムに組み合わせ生成するテストである．既存のルーチンをランダムな順番で実行するテストとも言える．1 ナゲットは複数の VLIW 命令で構成され，データ依存関係の情報を持たせることで，依存を含むプログラムを生成する．

簡単な例を示す．図 1 に示すようにナゲットが 3 つあるとする．それぞれ 3 項の加減算を行うための処理で，1 命令だけでは実現できず複数の命令で実現している．レジスタ r1

Vol.2009-ARC-184 No.29
2009/8/5

```
NUGGET0:                ;  Sequence:  d=a+b-c
        ADD     r1,r2,r3 ;  t=a+b
        SUB     r4,r1,r5 ;  d=t-c

NUGGET1:                ;  Sequence:  d=a+b+c
        ADD     r1,r2,r3 ;  t=a+b
        ADD     r4,r1,r5 ;  d=t+c

NUGGET2:                ;  Sequence:  d=a-b-c
        SUB     r1,r2,r3 ;  t=a-b
        SUB     r4,r1,r5 ;  d=t-c
```

図 1 ナゲットのサンプル例

のように，先行命令の出力を後続命令が使用するといった依存関係を指定し，データフローが存在するナゲットを定義している．

図 1 に示すナゲットをランダムに選択しつなぎ合わせてテストプログラムを作成する．使用レジスタは使用可能なレジスタがランダムに適宜指定される．ナゲット内でのみ依存関係は指定でき，その制約の効果はナゲット間をまたいで及ぶものではない．ナゲットが変われば，それまでの依存関係はリセットされ，そのナゲットにのみ指定されている関係を保つように命令列を生成する．

ランダムに決定することが可能な要素は，ナゲットを構成する命令のオペコードとオペランド，ナゲットの選出順が挙げられる．選出後のナゲットの並べ方は，連続で並べる単純な方法を採っている．

ナゲットランダムテストでは，ナゲットの種類が少ないと同じ命令列ばかり出現する可能性が高まってしまう．用意するナゲットの種類の増加やそのナゲットの組み合わせ方次第で，テストの多様さが増すと考える．

3. ランダムテストのテストカバレージの評価方式

従来のランダムテストの運用は，検証実施期間とバグ検出数を基に，熟練エンジニアによって検証終了時期を見極めることが一般的である．しかし，その時期を保証する事項は明らかではなかった．そこで，その判断要因にすべく，ランダムテストを実施することでどれだけの検証を行えたのかを調査する．具体的には，プロセッサがどのような動きや状態を示したのかを調べて，その結果をテストプログラムの評価とする．また，今回作成した 2 種類のランダムテストについて，それぞれの特性を比較し評価を行った．

テストを実施することによって，テスト対象をどの程度検証できたのかを示すためには，

- 多彩な入力をテスト対象に与えた

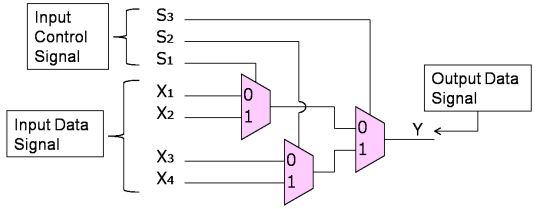

図 2 制御信号の例

- テスト対象が多彩な動作を示したかもしくは多様な状態になった

以上のことが判明すれば可能だと考える．本研究では，テスト対象が多様な状態になったことを調べる方法を検討する．

3.1 評価指標

テスト対象が示した状態を定量化する指標として制御信号に着目した．これはデータ信号はファンクションテストで検証すべき項目と考えられるからである．この制御信号は，回路の出力を決定づける要因となる信号線であると定義する．その信号が回路の状態を形成する要素だと見なし，任意の瞬間における複数の信号線の論理値がその回路の状態だと考える．テスト実行時にどれだけのユニークな状態が出現するかを評価尺度とする．

制御信号と制御信号で表す回路の状態について，それぞれ説明する．

制御信号

本稿において制御信号と定義している信号線について説明する．ハードウェア内に存在する信号線には 2 種類の信号線があると考え，データ信号と制御信号である．制御信号は回路の出力を決定する信号線であると定義し，データ信号は制御信号によって御される信号線であると定義する．同じ信号線でも，着目する箇所によってはデータ信号であったり制御信号であったりもする．それぞれの信号が実際のハードウェアにおいてどの信号にあたるかを，図 2 を例に説明を行う．

図 2 に記された回路は，出力信号 Y を 3 つのマルチプレクサを用いて 4 つの入力信号から出力信号 Y を選択している回路である．出力信号 Y の値を決定しているのは 3 つのマルチプレクサであり，Y は X_1, X_2, X_3, X_4 の中から選択される．つまり，Y を決定づけているマルチプレクサのセレクト信号, S_1, S_2, S_3 を制御信号だと定義し，X_1, X_2, X_3, X_4 はデータ信号であると定義する．

回路の状態

制御信号が回路の状態を示す要素であると定義し，その内容について述べる．回路内に存在する制御信号を複数集めて多ビットの信号バスを生成する．この多ビットのバスのビットパターンを，その回路の状態であると見なす．図 2 を例に説明を行う．

前述の通り，S_1, S_2, S_3 を制御信号と定義し，回路内には制御信号が 3 ビット存在する．この 3 ビット分の制御信号で示される 3 ビットのパターン $\{S_1, S_2, S_3\}$ が，その回路の状態であると定義する．S_1, S_2, S_3 のそれぞれが独立に真理値を示すならば，2 に示す回路は 2 の 3 乗の 8 状態を取り得る可能性がある回路となる．

この回路のテストを行う際には，入力を与え 8 状態すべてが観測できれば十分な検証を行えたのだと判断することができる．この観測された状態数こそが回路の検証量になるのだと考える．状態数の最大値は着目する制御信号の数に依存し，N ビットの制御信号ならば 2 の n 乗となる．しかし，各制御信号が独立に変化しない場合は 2 の n 乗にはならない．仮に，S_1, S_2 が互いに排反であり $S_1 == (!S_2)$ な関係を満たすならば，図 2 の回路は最大で 4 状態しか示さない．

テストプログラムを実行した時に，テスト開始時から終了まで回路の状態を記録し，テスト時間経過による状態のユニーク数の増加を算出する．その増加傾向を視覚化 (グラフ化) することでテストプログラムの評価とする．

例えば，図 2 の回路にテスト入力を与え，実行サイクルの経過と共に図 3 のように制御信号が遷移したとする．3 ビットのパターンと考え，時間経過による「出現済な状態の数 (最右列) ＝ユニークな数」を算出する．その増加傾向を視覚化すると図 4 になる．この評価方法をプロセッサに適応し評価を行う．しかし，これまでの研究から制御信号を単純に抽出しただけでは，回路の状態数の上限値が不明瞭であることが分かっている．そのため，回路の状態数の増加傾向を正しく視覚化できず，ランダムテストプログラムの評価が正しく行われない．

3.2 状態数の上限

制御信号の抽出を行うと，演算器ユニットだけでも数十ビットの制御信号が存在する．つまり，2 の数十乗個 (約 1000 兆個) 回路の状態が存在することになり，かなり膨大な数となる．しかし，前述した通り，各制御信号が独立に変化しない場合は 2 の制御信号数乗にならない．

そこで設計ソースを解析することで，制御信号の状態数の上限値を求めることにした．状態数の上限値を求めるアルゴリズムとして，次のような場合が考えられる．

Cycle	S_1	S_2	S_3	State num.
0	0	0	0	1
1	0	1	0	2
2	1	0	0	3
3	0	1	0	3
4	1	1	0	4

図 3 制御信号の遷移例

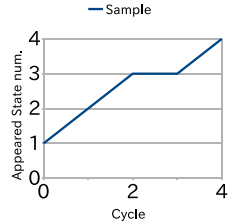

図 4 状態の増加傾向

- 制御信号 S_1, S_2 が互いに排反であり，$S_1 == (!S_2)$ な関係を満たす制御信号
- 制御信号 S_3, S_4 のソースが等しく，$S_3 == S_4$ な関係を満たす制御信号
- 特定の制御信号の集合に着目した時，着目した制御信号群に対して起こりえないパターンが存在する

以上の条件を基に設計ソースの解析し，制御信号の抽出を行い状態数の上限値を算出する．算出した上限値に対して，ランダムテストプログラムがどれだけ上限値に近づいたかを評価する．

3.3 制御信号の遷移状態解析

ランダムなパターンのテストを行っても，必ず全てのパターンが検証できているとは限らない．ランダムテストでは一時的に偏ったパターンの検証しか行っていない場合も考えられる．そこで，制御信号の遷移状態を解析することで回路の状態が偏っていないかを調査する．

テストプログラムを実行した時に，テスト開始時から終了までまったく変化しなかった制御信号を記録する．変化しなかった制御信号を確認し，レベル毎に分類する．LEVEL1:変化しなければならない制御信号 (つまりナゲット不足)．LEVEL2:他の制御信号との関係で変化しにく制御信号．LEVEL3:仕様ではほとんど変化しない制御信号．以上の LEVEL に分類を行い，主に LEVEL1 に分類された制御信号に対して着目した．まったく変化していない制御信号に対して，変化するようにナゲットを追加することでランダムテストプログラムの効率向上を図るようにした．以上，状態数の上限値の算出と制御信号の遷移状態を解析することでランダムテストのテストカバレージについて評価する．

4. 測 定 方 法

4.1 制御信号の抽出

回路の状態を観測する評価は，プロセッサの論理シミュレーションを行う時に実施する．観測処理を行うためには回路から制御信号を抽出しなければならない．しかし，ハードウェア記述言語で記述された設計データから仮に実物のハードウェアを制作するとした場合，生成されるマルチプレクサのセレクト信号が設計データのどの信号線に適応するのかを導出することは困難だと考える．そこで，ハードウェア記述言語の文法上において，出力信号を選択している信号線が制御信号であると考える．制御信号とみなすハードウェア記述言語の文法を次に記す．

(1) if-else 文の条件式部分に記述される評価式 (1 ビット分)
- if (!RST) の!RST

(2) case 文の条件式部分の信号バス (多ビット分)
- case (wire_b) の wirec_b

(3) 三項演算子 (?:) の条件式部分に記述される評価式 (1 ビット分)
- assign wire_a = (wire_c==4'b0000) ? 1'b1 : 1'b0; の wire_c==4'b0000

この条件に適応する評価式や信号線を設計ソースから抽出し，前述の評価方法に従い複数本数まとめて観測対象にする．

4.2 テスト対象プロセッサ

本研究で作成したテストプログラムを用いて検証を行ったプロセッサについて説明する．我々の研究室では高性能と低消費電力の両立を実現するプロセッサアーキテクチャの研究を行ってきた．我々は VLIW プロセッサの実行効率の良さに着目して，異種命令セット SMT(Simultaneous Multi-Threading) プロセッサ OROCHI[2] の研究を進めてきた．異種命令セット SMT は VLIW のバックエンド上に命令セットが異なる SMT を構築し，それぞれのスレッドにシステム制御プログラムを実行する汎用プロセッサ，マルチメディア処理を実行するアプリケーションプロセッサの役割を割り付ける．これにより 2 つのプロセッサの機能を 1 プロセッサに集約し，仮想的な「汎用プロセッサ+アプリケーションプロセッサの環境を実現することで，効率の良いシステム構築を目指している.」

異種命令セット SMT プロセッサを構成する 2 つのスレッドを，汎用スレッドおよびアプリケーションスレッドと定義し，前者には ARM 命令，後者には FRV 命令を採用した．OROCHI システムの概要を図 5 に示す．OROCHI プロセッサは，それぞれのスレッド毎にフェッチエンジンを搭載し，命令でコードなどの処理を経た後に命令キューに投入され共通の VLIW バックエンドエンジンで実行する．VLIW バックエンドは VLIW アーキテクチャである FRV 命令を直接サポートし，ARM 命令はでコード時に命令変換を行い実行可能な形式に変換して実行する．

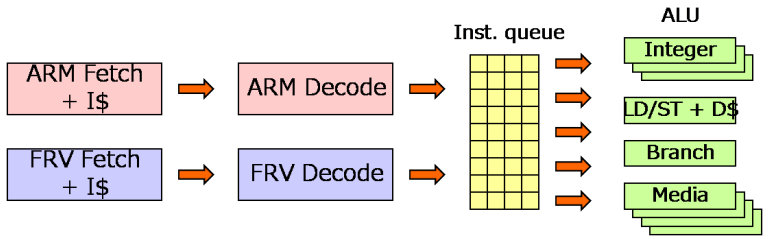

図 5 OROCHI ブロック図

5. 評　　価

5.1 ランダムテストプログラムの評価

プログラムを実行することでプロセッサの論理設計をどの程度検証できたのかを調べるべく，作成した 2 種類の FRV 命令ランダムテストプログラムの評価を行う．

ランダムテストプログラムと一般的なベンチマークプログラムを比較し，テストプログラムの本質である「検証に有用」という性質が満たされているかを確認する．また，2 種類のランダムテストを実施することで全体的な検証率向上が果たせているかを調査する．

FRV 命令プログラムの単独実行

ランダムテストプログラムと一般的なベンチマークプログラムがどれだけテスト対象を検証できていたのかを評価し，その結果を図式化し比較する．一般的なベンチマークは Standord ベンチマークの 10 種 (Bubble, FFT, Intmm, Mm, Perm, Puzzle, Queens, Quick, Towers, Trees) を対象に選んだ．理由は，プログラムサイズが小さく，評価が取りやすいためである．またテストプログラム以外の検証プログラムとして使用していたため，テストプログラムと比較してどの程度検証効果があったのかを調べることも目的である．

FRV 命令プログラム単独実行の評価結果として着目した制御信号の集合として，整数ユニットの 1 つの中の信号，命令キュー内の信号を選択した．順に，順に，図 6 が整数ユニット 1 つ，図 7 が命令キューの結果である．Stanford ベンチマークを全て掲載すると図の見

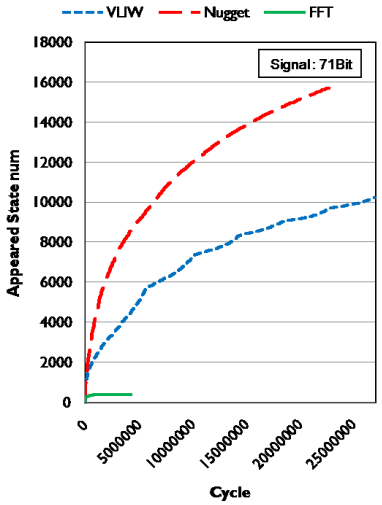

図 6 整数ユニット 1 の傾向 (FRV)

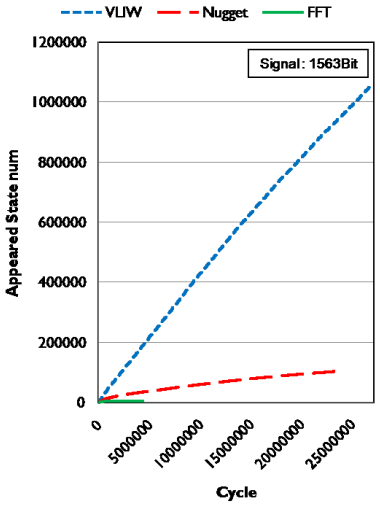

図 7 命令キューの傾向 (FRV)

易さが損なわれてしまうため，10 種のうちプログラム終了時点でもっとも多くの状態数が観測された 1 種のみを示すこととする．図中の線はプログラム実行開始から終了までの軌跡を描いている．図中の凡例のうち，[VLIW] が VLIW ランダムテスト，[Nugget] がナゲットランダムテストを表している．

図 6 は整数ユニット 1 内の制御信号 71 ビット分に着目した時の傾向である．演算ユニットはデータ依存を解決するためのフォワーディング回路を内部に有しており，データ依存の発生の仕方が結果に強く影響すると考えられる回路である．ナゲット方式の方が VLIW 方式よりも出現数が多いことが見て取れる．データ依存関係のケースでは，ナゲット方式の方が効果的であることが示された．また，どちらのランダムテストも汎用ベンチマークに比べて多数の状態になったことがわかる．各ランダムテストはテストを目的としたプログラムとしての性質は有していることが言える．

図 7 は命令キュー内の制御信号 1563 ビット分に着目した時の傾向である．命令キューは命令の多彩さが結果に強く影響すると考えられる回路である．命令キューでは VLIW 方式の方がナゲット方式より出現数が多くなった．これはオペコードとオペランドをランダムに決定しテスト命令列を生成する VLIW 方式の方が様々なパターンの命令列が生成されるためであると考えられる．

まとめとして，データ依存関係のテストケースはナゲット方式のランダムテストで効率よく検証できることが示された．データ依存関係のテストケースにあった命令ナゲットを用意した結果であり，テストケースに適応するナゲットが用意されていない場合には検証効率の低下が懸念される．ナゲットランダムテストの更なる効率向上には，テストケースを見極めそれに適応するナゲットを作成することが重要と考えられる．

5.2 FRV/ARM 同時実行時の検証量の評価

OROCHI プロセッサで FRV 命令テストプログラムと ARM 命令テストプログラムを同時に実行したときに，それぞれ単独で実行する場合に比べ検証効果が高いのかを調査する．FRV/ARM ともにナゲットランダムテストを使用した．FRV 命令テストプログラム単独実行 (凡例:FRV)，ARM 命令テストプログラム単独実行 (凡例:ARM)，FRV/ARM テストプログラム同時実行 (凡例:SMT) の 3 種を図に記す．通常，プロセッサ内には FRV 専用回路，ARM 専用回路，FRV/ARM 共有回路の 3 種類のハードウェアが存在するはずであり，全てが同時に動作しうる時は同時実行の場合しかない．当然制御信号もそう考えられる．そのため，SMT は各単独実行の結果に比べ状態数は増えると予測する．図 8 に結果を記す．同時実行時の結果が FRV/ARM 単独実行時に比べ大きく上回っていることがわかる．単位時間あたりの検証効果は同時実行時の方が高いことが示された．

5.3 状態数の上限値算出，制御信号の遷移状態解析

以上の結果から，ランダムテストプログラムは汎用ベンチマーク比べハードウェアの検証量が多く，効率的であると評価できた．しかし，制御信号 71 ビット (状態数の上限値 2^{71} 個) の整数ユニット 1 に対して，VLIW，ナゲットランダムテストのどちらもユニーク数 (出現済みな状態数) は上限値に達していないことがわかる．3 章で説明しように，各制御信号が独立に変化しない場合は上限値が 2 の制御信号数乗にならない．3.2 章で挙げた条件を基に，上限値を算出するために整数ユニット 1 の設計ソースに対して解析を行った．さらに図 6，図 8 のテスト開始時から終了までの制御信号の遷移状態を解析し，LEVEL1 に分類された制御信号に着目しナゲットの追加を行った．追加したナゲットの数は 47 個，ナゲットの総数は 147 個である．ナゲットを追加する前と追加した後の実行結果を比較し，特定のケースに対するナゲットを増やすことでどれだけテスト効率が向上するかを調査する．

まず，整数ユニット 1 に対して状態数の上限値の算出を行う．3.2 章で挙げた条件を基に解析を行うと，起こりえないパターンやある制御信号同士が相反する関係なものを除いていく．すると，整数ユニットの状態数の上限値が 28 億個 (約 2^{24} 個) になることがわかった．

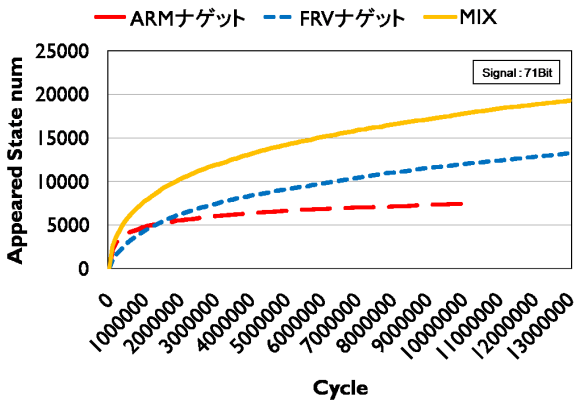

図 8 整数ユニット 1 の傾向 (FRV 単独，ARM 単独，SMT)

制御信号をただ抽出しただけでは，状態数の上限値が 2 の制御信号数乗だったのに対してかなり上限値が低くなったことがわかる．起こりえないパターン等を省くことで回路の状態数の上限値を明確化することができた．テスト開始時から終了時までの制御信号の変化パターンを解析し，変化していない制御信号に着目しナゲットの追加を行った．特定のケースに対してナゲットを追加することで増加傾向が変化するかどうか調査する．図 9 はナゲットを追加する前と追加した後の傾向である．図から読み取ることができるように，ナゲットを追加した後では追加する前に比べて増加傾向が顕著に増加していることがわかる．短いサイクル数で状態数が増加している．また状態数もナゲットの追加によって増加していることがわかる．制御信号の変化パターンを解析することで，変化していない制御信号に着目し，特定ケースのナゲットを増やすことで検証効率が増加することがわかった．しかし，状態数の上限値に比べるとまだまだユニーク数は足りないと考えられる．また，今回状態数の上限値を算出するために解析したのは設計ソースのみでプロセッサの仕様上制御信号の起こりえないパターンはまだ存在すると考えられるため，今回算出した状態数の上限値が厳密な上限値とは限らない．

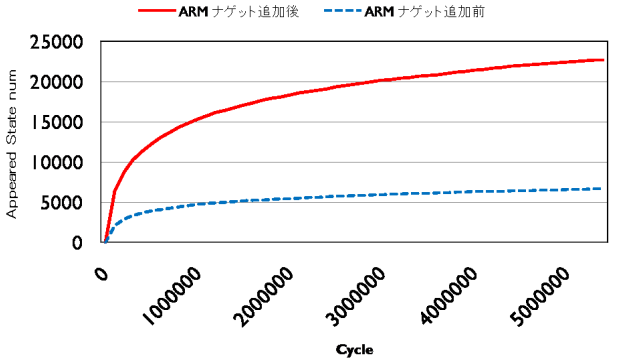

図 9 整数ユニット 1 の傾向 (ナゲット追加前，追加後)

6. ま と め

OROCHI プロセッサを開発する上で必要となる，論理設計を検証するためのテストプログラムの開発を行った．本研究では，各種仕様書から満たすべき性質を検証するテストケースを含むファンクションテストと，その他エンジニアが想定し忘れているケースを洗い出すためのランダムテストを作成しプロセッサの論理検証に努めた．

ランダムテストの運用を効率よく行うために，検証時におけるハードウェアの検証量を調べる尺度を考え，その評価尺度を用いてランダムテストの評価を行った．命令を細かくランダムに生成する VLIW ランダムテストだけではテストし難い部分を，ナゲットランダムテストで補えていることがわかり，双方を用いることで全体の検証率の向上を期待できることが示された．しかし，これまでの研究では制御信号で定量化するハードウェアの状態について，取り得る状態数の上限値が算出できていなかったので，上限値が不明であった．そのため，テストにより出現した状態数が全体の何割程度であるかを求めることができなかった．今回，設計ソースを解析することで制御信号同士の関係を調査し，状態数の上限値を算出することにした．求めた上限値に対して，ランダムテストのユニーク数の増加傾向を評価した．さらにテスト開始時から終了までまったく変化しなかった制御信号に着目し，着目した制御信号に対してナゲットの追加を行うことで検証効率がどのくらい向上するかを評価した．設計ソースや制御信号のパターンを解析し，その結果を基にナゲットの追加を行う等してテスト効率の向上に役立てることができるとわかった．

Vol.2009-ARC-184 No.29
2009/8/5

今後は，設計ソース，制御信号のより詳細な解析を行うことで状態数の上限値を算出し，ランダムテストが全体の何割検証できているのかを評価することが挙げられる．

謝辞 本研究の一部は半導体理工学センターとの共同による．また，本研究は東京大学大規模集積システム設計教育研究センターを通し，日本ケイデンス株式会社およびシノプシス株式会社の協力で行われた．

参 考 文 献

1) 島田貴史，酒井智也，北村俊明: 組み込みプロセッサを用いた動体検出システムの構築と評価．情報処理学会研究会報告 ARC-164,pp.31-36,2005
2) 田端猛一，塩田耕太朗，北村俊明:異種命令セット同時実行プロセッサの実装に向けた評価．情報処理学会研究報告 ARC-167,pp.95-100,Nov.2007.
3) ARM Architecture Reference Manual, 2000, ARM Ltd.
4) FR550 Series Instruction Set Manual, Ver.1.1, 富士通株式会社 (2002)
5) 元安優，北村俊明:ARM プロセッサ向けテストプログラムスイートの開発．広島市立大学　情報科学部情報工学科　卒業論文，2008
6) 山下純一，北村俊明:プロセッサの論理設計検証用テストプログラムの開発と評価方式．広島市立大学大学院　情報科学研究科情報工学専攻　修士論文，2009