教育用32ビットマイクロプロセッサQP-DLXの設計における設計検証

1C－7

岩井原 瑞穂†, 國貞 勝弘†, 山田 哲也†, 池 兼次郎‡, 斎藤 靖彦†, 安浦 寛人†

† 九州大学 大学院総合理工学研究科 情報システム学専攻
‡ 九州大学 工学部 情報工学科

1 まえがき

ハードウェア記述言語 (Hardware Description Language; HDL) や自動論理合成システムの普及によるシステム開発時間の短縮には目覚しいものがある．設計手法の変遷とともに，設計工程の大きな部分を占める設計検証についても，その方式に変化が見られる．そのひとつは，HDLの機能シミュレータを用いたデバッグであり，これにより検証の工程数の大幅な削減が可能になっている．今後，HDLを設計資産とし，HDLを中核としたシステム設計が主流となる中で，HDLレベルの設計検証も比重を増してゆくであろう．

機能シミュレータを用いた検証の形態として，(1) 与えられた要求仕様および要求性能について，HDLでプロトタイピングを行ない，そのうえで方式設計の正当性の検証，および性能評価を行なう，(2) 自然言語，ソフトウェアシミュレータ，仕様記述言語などで与えられる動作仕様定義をHDL記述が満すかを検証する，(3) ゲートレベルでの遅延を考慮した論理回路シミュレーションの入力に，HDLの機能シミュレータからのシミュレーション出力を加工したものを用いる．などのように機能シミュレータによる検証は多岐にわたるが，いずれにおいても機能シミュレータの実行速度や，シミュレーションのステップ数が問題となることが多い．

本稿では，現在開発中の教育用RISCプロセッサQP-DLXについて，HDLレベルの機能シミュレータを用いた設計検証の事例報告を行ない，検証ステップ数を削減するために用いた方法を紹介する．

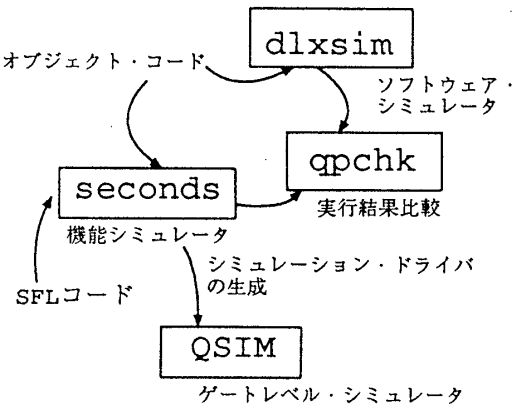

図 1: 設計検証環境

"Design Verification of Education-Purpose 32bit Microprocessor QP-DLX," M. Iwaihara, K. Kunisada, T. Yamada, K. Ike, Y. Saitoh and H. Yasuura, Kyushu University

2 設計検証環境

今回の設計の検証環境を図1に示す．論理合成システムPARTHENONは，ハードウェア記述言語としてSFLを用いており，secondsはその機能シミュレータである．これによりソースコードレベルの検証が可能である．またsecondsの出力フォーマットを指定することにより，レイアウトツールとして用いているCOMPASSの論理シミュレータ用の入力を生成することができる．これにより言語レベルと論理回路レベルのシミュレーション結果の一致比較が可能である．

ソフトウェア・シミュレータについては，カリフォルニア大学で開発された *DLX* のシミュレータ dlxsim を用いている．このソフトウェアは無償で公開されており，ftpにより容易に入手できる．このソフトウェアには，GCCをベースにしたCコンパイラ，アセンブラも含まれている．今回の検証のためには，dlxsimとsecondsのシミュレーション結果を比較するツールqpchkを作成した．またアセンブラについては，QP-DLXのオペコードの割当てのために変更を行なっている．Cコンパイラについては，アーキテクチャの微妙な違い等で修正する必要があった(例えば，分岐命令の方式やロード・インターロック等を行なう／行なわない)．機能レベルシミュレータの実行速度が1秒間で数10ステップ程度であった．そのため，dlxsimとの比較では，この実行速度により検証可能なステップ数が制限される．

3 検証の形態

QP-DLXはDLXアーキテクチャ[1]を基本とし，これに割込みなど[1]で定義されていない機能や，内部動作の観測などの教育のための機能を付加した．ソフトウェア・シミュレータdlxsimは命令セットレベルでの仕様記述と看做すことができる．割込みや観測機能などの独自の部分については実現方式に誤りがないかの検証が必要である．これらのことを考慮して，以下の形態によるQP-DLXの検証を行なった．

- 検証者によるテスト入力系列の作成および確認 (約7000ステップ)
  各命令，内部割込みおよび観測機能について，ソースファイルを目視し，誤りを発生しやすい箇所について，重点的にテストする．

- 命令セット・シミュレータとの実行結果比較
  DLX の命令セットを解釈・実行するソフトウェア・シミュレータにおいて，適当なテスト・プログラムを実行させ，その実行結果をsecondsによるSFLのシミュレーション結果と比較する．テスト・プログラムは1000行程度のCプログラムをコンパイルしたものである．

今回の検証では，時間的制約により上記ステップ数程度の検証しか行なっていないが，それでも検証を効果的にするた

めに，次節に述べる方法により効率の良い検証データの生成を行なった．

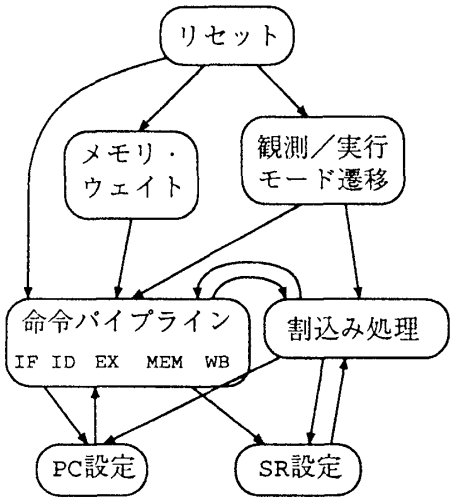

図 2: 機能関連グラフ

4 検証ステップ数の削減

機能シミュレータによる検証では，実際に起り得る全ての場合についてテストすることは不可能であため，効率的な検証データを生成する必要があるが，今回の検証では，以下のように機能単位の入出力に着目した検証ステップの削減を行なった．

以下，本検証手法の立場を述べる．1つのシステムは複数の機能から成り立っており，システム全体の検証とは，個々の機能の検証の集合とする．1つの機能は1つのモジュールまたは複数のモジュールを用いて実現されている．個々の機能にはその動作仕様が定義されている．各機能にはそれぞれ他の機能との間に値の入出力がある．各機能間の値の入出力に着目して，各機能を節点とし，2つの機能間の値の伝達がある場合に有向枝を1つ張ったグラフを機能関連グラフと呼ぶ．図2にQP-DLXにおける機能関連グラフを示す．ただしこの図は説明のためいくらか簡略化している．

各機能の動作仕様は，システムの各値が定まっているある時刻から一定時刻後の各値を定める．検証ステップ数を減らすためには，調べなくてよいシステムの入力値の組合せを見つければよい．今回の検証では以下の4つの方法でステップ数を削減した．

4.1 独立機能

ある機能Fの検証において，機能関連グラフ上でFへの経路を持たない機能Gは，Fの動作に影響を与えない．そのため，Fの検証においては機能Gへは適当な値を1つ設定する．

[例] 図2において，観測／実行モード遷移の入力には，メモリ・ウェイトは影響を与えない．そのため観測／実行モード遷移の検証のためにメモリ・ウェイトの内部状態をいくつか変える必要はない．

4.2 独立入力

ある機能Fにおいて，1つの出力値Oを決定するのにある入力Iの値が影響しないならば，Oの検証においてIの値を変化させる必要はない．

[例] 割込み処理において，実行再開番地の退避は割込み要因の種類に無関係である．このため，実行再開番地と割込み要因の組み合わせについて検証する必要はない．

4.3 集約化

ある機能Fの検証において，別の機能Gの複数の値が集約されてFの1つの値として入力されている場合，そのGの値を変化させる必要はない．

[例] 命令パイプラインでは，分岐を起す命令がジャンプや条件分岐など複数からなるが，機能PC設定への入力ではそれらの値はフラグJMPとして集約されているため，PC設定の検証において分岐を起す命令の種類を変化させる必要はない．

4.4 抽象化 [2]

ある機能Fの検証において，ある入力IがFの内部で検証済みの関数で処理され，その結果により何種類かの異なる動作が定義されているとき，そのそれぞれの動作を生じさせる数だけの値を入力Iに与える．

[例] 命令パイプライン内部において，機能ALUが検証済みのとき，例外処理の検証のためALUがオーバーフローを生じる値と生じない値の2種類のみ調べる．

以上の4つの性質それぞれについて，仕様定義およびHDL記述の両方について成り立つ箇所を検証者が調べ，これに基づいて検証用データを作成した．

[例] 命令パイプラインのEXステージでオーバーフロー割込みが生じたとき，その2クロック後に割込み処理が行なわれ，ステータスレジスタSRに割込みベクトルが設定されるかを検証する．QP-DLXは32ビットのレジスタが40個，4ビットのレジスタが5個，外部入力が115ビットからなる．これらの値の全ての組み合わせで検証するのは不可能である．そこで上記方法を適用すると，例題の検証に関係しているのは，32ビットレジスタが6個，4ビットレジスタが4個，外部入力が7ビットとなる．このうち32ビットおよび4ビットレジスタはすべて1ビットに抽象化できる．さらに独立入力と集約化を考慮すると，全部で128通りの組み合わせまで減らせる．

5 まとめ

以上，QP-DLXの設計における機能シミュレータを用いた設計検証について事例報告を行ない，その中で用いた検証ステップ数を削減するための方法について述べた．また，QP-DLXの設計データは，京都大学および奈良先端技術大学において，形式的検証の素材として用いられている．

参考文献

[1] Hennessy, J. L. and Patterson, D. A., *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.

[2] Clarke, E.M., Grumberg, O. and Long, D. E., "Model Checking and Abstraction," Proc. Principles of Programming Languages, pp. 343–354, 1992.