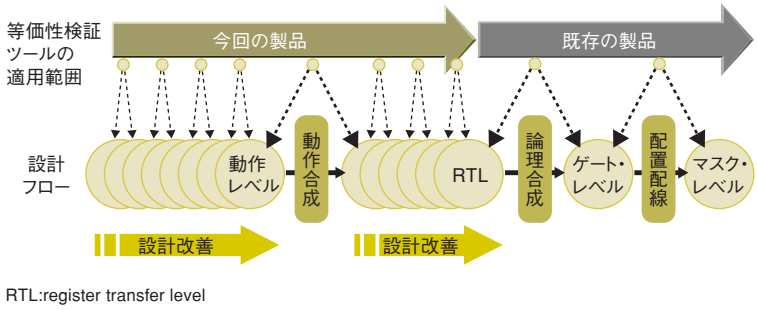
C言語ベースのLSI設計を一気に効率化 論理機能の等価性検証ツールを開発

山本修作
米Calypto Design Systems, Inc.

LSIに搭載できる回路が1000万ゲートを超える時代になり，C言語ベースの大規模論理の設計手法が注目を集めている。この設計手法では動作合成ツールを使うことになるが，これまでは動作合成前後の論理機能の等価性を効率よく検証する手段がなかった。米Calypto Design Systems, Inc.の論理機能等価性検証ツールを用いることで，この検証が簡便に行えるようになる。C言語によるLSI設計が一気に効率化する。（本誌）



これまでRTL（register transfer level）設計以降でしか使えなかった，等価性検証ツールの守備範囲が動作レベル設計への拡大する。米Calypto Design Systems, Inc.のデータ。

われわれは，順序回路の等価性を容易に検証できるEDA（electronic design automation）ツール「SLEC（Sequential Logic Equivalence Checker）」（以下，今回の製品）を開発した。今回の製品は，われわれが開発した最新のフォーマル検証技術（エンジン）[注1)]をベースにしている。

これまでにも，論理機能の等価性を検証するEDAツール[注2)]は市場にあったが，内蔵するフォーマル検証エンジンが組み合わせ回路しか扱えなかった。このため適用範囲は，RTL（register transfer level）設計以降に限られていた（p.000の図参照）。

われわれが開発したフォーマル検証エンジンは順序回路の等価性を検証できるため，適用可能範囲がぐんと広がる（**図1** (a)）。動作合成[注3)]前からRTL設計完了時まで一貫して，論理機能の等価性検証に使える。また，LSIの上流設計でよく使われている言語である「SystemC」や「C言語」，「C++」をサポートしていることも，今回の製品の特徴である。

動作合成や論理合成ベースの設計を効率化

こうした特徴によって，今回の製品を利用することで，動作合成前の動作レベルのSystemCモデルと，合成後のRTL（register transfer level）のVerilog-HDL/VHDLモデルの論理機能等価性を簡単に検証できるようになる。これまで両モデル間の効果的な検証方法がないために，動作合成ツールの導入に踏み切れなかった設計者は少なくなかった。こうした設計者の杞憂を今回の製品は払拭することができる。

今回の製品は，動作合成を利用した設計だけではなく，論理合成を使うこれまでの設計の効率化にも寄与する。例えば，設計現場では，論理合成ツールを適用する前に，パイプラインの変更やリタイミング[注4)]といった最適化を設計者が行っている。最適化によって順序回路が変わることが多いため，最適化前後の論理機能の等価性は論理シミュレータを使って検証する。今回の製品を使えば，こうしたケースでも処理前後の論理機能の等価性を簡単に検証できる。論理シミュレータを使う場合に比べて処理時間を短くできる場合が多い（図1 (b)）。

注目集める動作合成ベースの設計手法

1チップに集積できる回路規模の増大につれて，LSI設計の抽象度は上昇してきた。回路図をベースにしたゲート・レベル設計は，1980年代後半から論理合成の利用を前提にしたRTL設計へと移行していった。最近はその論理合成の限界が明らかになってきており，動作合成を利用する設計手法に注目が集まっている。

この手法では，SystemCやC++，C言語などを使いLSIの動作をモデル化し，それを動作合成ツールに入力する。動作合成ツールは，動作レベル・モデルと制約条件から，最適なマイクロアーキテクチャを選択しRTLモデルを生成する。RTLモデルは論理合成ツールへの入力となる。

一般に，マイクロアーキテクチャが異なるとデータ処理の順序（シーケンス）も異なる（**図2**）。たとえば，全演算を一括して処理する動作記述をそのままRTLに展開したモデルと，回路量を削減するために演算器を最大限に共有したRTLのモデルでは，シーケンスがかなり異なっている。

設計が動作レベルへ移行した場合，次のような手順で論理機能の検証を行えれば理想的といえる。まず，与えられたLSIの仕様と動作レベル設計結果の論理機能の等価性を，論理シミュレータや汎用のプロパティ・チェック型フォーマル検証ツール[注5)]を使って検証する。そして動作合成や論理合成を使って設計の抽象度を下げる度に，等価性検証ツールを使って，下げる前後のモデル間の論理機能等価性を自動的に検証する。

テストベンチ書き換えの手間が発生

ところが現実はこうなっていない。動作合成

注1）フォーマル検証とは，数学の証明のように定理や公理，数式の変換などを使って進める検証で，シミュレーションのようなパターンは用いない。市販のフォーマル検証ツールは，このフォーマル検証を実行するアルゴリズム（エンジン）に，いくつかのツールを組み合わせたパッケージになったものが多い。

注2）この記事でいう「等価性検証ツール」は，フォーマル検証技術を使った，論理機能の等価性を検証するためのツールである。

注3）動作合成とは，動作レベルのモデルからRTLモデルを生成する技術。論理合成より上位の合成技術である。制約条件を変えることにより，異なる構造のRTLモデルを生成できる。

注4）広い意味ではタイミングを変更，調整すること。ここでは，クリティカルパスの最適化や負荷バランスの最適化，フリップフロップ・ベース設計からラッチ・ベース設計への変更などを指す。

注5）フォーマル検証エンジンを使った検証ツール。検証したい項目をプロパティとして記述すると，設計がそのプロパティを満たしているかどうかをチェックする。例えば設計仕様を複数のプロパティとして記述すれば，動作レベル設計結果が設計仕様に沿っているかを検証できる。なお，等価性検証ツールは，「二つの論理が等しい」というプロパティだけを処理する，プロパティ・チェック型フォーマル検証ツールといえる。

注6）入力パターン，期待値パターンなどからなる。

前後，すなわち，動作レベルのモデルとRTLモデルの等価性は，等価性検証ツールではなく，論理シミュレーションで検証している。既存の等価性検証ツールが内蔵する，フォーマル検証エンジンが，組み合わせ回路しか扱えなかったためである（**図3**）。このエンジンでは，RTLモデルとゲート・レベル・モデルの論理機能の等価性は検証できても，動作レベルのモデルとRTLモデルの等価性は検証できない。両モデル間では，データ・パスの構造だけではなく，シーケンスが変わるため，それにともなって制御回路が変わってしまうためである。

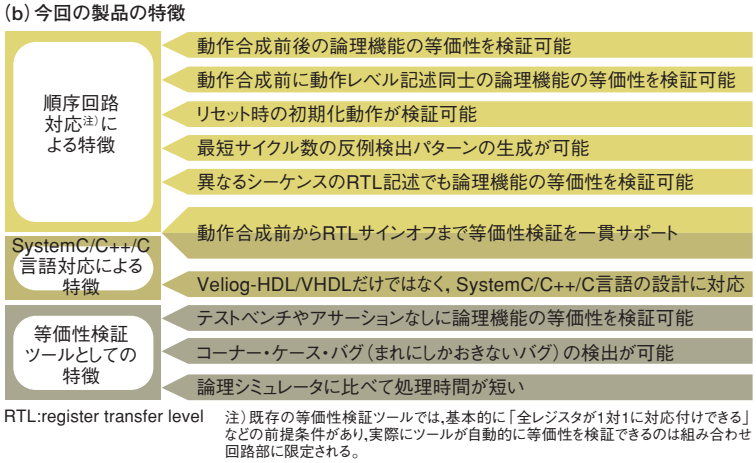
現在，動作レベルのモデルとRTLモデルの等価性は論理シミュレータを使って検証するのが一般的である。どちらのモデルにも同じテストベンチ注6）で論理シミュレーションを実行し，結果が同じだったら等価とみなす。この方法には二つの問題点がある。一つはテストベンチの網羅性を保証する適当な手段がないため，検証漏れの恐れがあること。

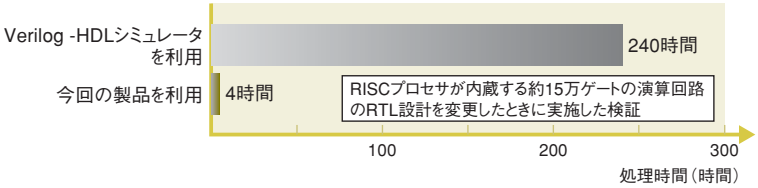
もう一つは，シーケンスが変わるとテストベンチを書き直す必要があることだ。シーケンスが変わると，タイミングが変化したり，新たなコーナー・ケース注7）が発生するためである。動作合成ツールを使ってマイクロアーキテクチャが異なる複数のRTLモデルを短時間で生成できたとしても，その度にテストベンチを書き変える必要が出てくる。

一貫サポートが可能に

こうした問題点を解決できるのが，今回の製品である。冒頭で紹介したように，動作合成前からRTL設計の完了時まで一貫した論理機能の等価性検証が可能になる。テストベンチやアサーション注8）などを用意する必要がないため，設計者は，消費電力，パフォーマンス，チップ面積などの最適化に集中することができ，設計目標への到達がさらに容易となる。

また，動作モデルとRTLモデルの論理機能等価性の検証だけではなく，動作レベル設計やRTL設計の途中で論理機能の等価性を検証する際にも有効なツールとして稼働する。例えばRTL設計では，パイプラインの変更や演算器の





(d) 適用例

設計の内容			「SLEC」の実行結果注1）			
回路規模（ゲート）	機能	レイテンシ/スループット	検証内容	検証結果	メモリー使用量（Mバイト）	処理時間
5万4000	ネットワーク処理回路	2/1	C言語記述とRTLのVerilog-HDLコード（共に手作業で作成）を比較	一致	278	1時間16分26秒
3万8000	FIRフィルタ	15/16	動作合成前のSystemC記述と，合成後のRTLのVerilog-HDLコードを比較	一致	317	7分45秒
22万	IMDCT回路	76/111		一致	620	13分4秒
15万8000	RISCプロセサ	情報なし	人手修正（リタイミング）前後のRTLコード同士を比較	不一致（バグ発見）	196	3分34秒
9万7000				不一致（バグ発見）	1200	24分30秒
6万				一致	967	2時間20分28秒

FIR:finite impulse response
IMDCT:inverse modified discrete cosine transform
RTL:register transfer level

注1）「SLEC1.2」で検証。使ったコンピュータのマイクロプロセサは3GHz動作のPentium4で，主記憶容量は2Gバイト。

図1●LSI設計検証の効率化に寄与
動作合成前後や，順序回路のフローが変わるRTL設計の変更時における，論理機能の等価性検証が簡便かつ高速になった。米Calypto Design Systems, Inc.のデータ。

注7）極めて特殊な状況下でのみ発生する状態。

注8）期待されている動作。プロパティの1種。

共有方法，リタイミング，低消費電力化[注9]などの最適化を実行すると，シーケンスが変わる（**図4**）。最適化前後で論理機能の等価性を検証するのに，今回の製品を使える。

今回の製品を使った検証フロー

次に，今回の製品を使った検証フローを説明する（**図5**）。従来の等価性検証ツールと類似したフローである。

(1) 比較する二つの設計の読み込み[注10]
(2) 検証環境や制約，方法などの設定
(3) 等価性検証の実施 (一致，不一致の出力)
(4) 不一致の場合は，反例パターン[注11]の出力
(5) 反例パターンを使ったデバグ作業

TCLで制御情報を定義

今回の製品は，ユーザーが定義した以下の情報に基づいて検証を実行していく。今回の製品を制御するための情報はTCL（Tool Command Language）[注12]のスクリプトとして記述する。

(1) 二つの設計間で対応する入出力ポートの指定
(2) クロックの設定
(3) リセット・シーケンスもしくは各レジスタの初期値設定
(4) 等価性をチェックするポイントの指定
(5) 入出力タイミングのアラインメント

なお，従来の等価検証ツールとは異なり，今回の製品では，検証対象の設計間で等価な内部レジスタのペアを指定する必要はない。シーケンスが異なる設計同士では，ペアとなるレジスタ自体が存在しない場合が多いためである。特に動作合成前後ではその傾向が強い。ただし，レジスタのペアをユーザーが定義することは可能になっている。これでツールの負荷を軽くすることができるからである。

レジスタの初期値は，今回の製品で必要だが，従来の等価性検証ツールでは必要ない。順序回路の等価性検証では，状態の遷移を含めて検証するため，二つの設計を同じ状態にしてから検証を開始する必要がある。なお初期値を直接設定するよりも，リセット・シーケンスを実行する方が一般的である。今回の製品では，内蔵したサイクル・ベースの論理シミュレータを使って，リセット・シーケンスを実行する。

今回の製品では，ユーザーが指定した出力ポートごとに，等価かどうかをチェックする。これも従来の等価性検証ツールとは異なる。なお上述したように，今回の製品でレジスタのペアを指定した場合には，そのレジスタ・ペアも常に等価性チェックの対象になる。等価性チェックは，次の「タイミング・アラインメント」に沿って行われる。

レイテンシとスループットを指定

異なるシーケンスをもつ設計は，入出力のタイミングが異なる場合が多い。動作合成ツールを用いた場合，レイテンシ，スループットといったタイミング制約を変更するだけで，異なるマイクロアーキテクチャのRTLモデルを生成できるためである。

今回の製品で，こうしたモデル間の等価性を検証する際には，マッピングされた出力ポート，内部レジスタごとに比較すべきタイミング情報が必要となる。これをタイミング・アラインメントと言い，レイテンシやスループット[注13]として指定する（**図6**）。

タイミング・アラインメントは，ユーザー自

注9）不要な部分の動作を停止するなどを行うことで，制御回路の追加が必要になる。

注10）他の検証ツールや合成ツールと同じように，この製品で対応できる記述範囲（サブセット）は明確に決めている。特にSystemC/C++/C言語の場合は，業界標準のサブセットが存在しないので，われわれ独自でサブセットを定義した。

注11）不一致の状態にするための入力パターン。

注12）米University of California, Berkeleyで開発されたスクリプト言語。EDAのツールに広く採用されている。

注13）レイテンシは処理にかかる時間（クロック数）。スループットは回路が入力を受け付けたり出力する頻度で，クロック数で表す。

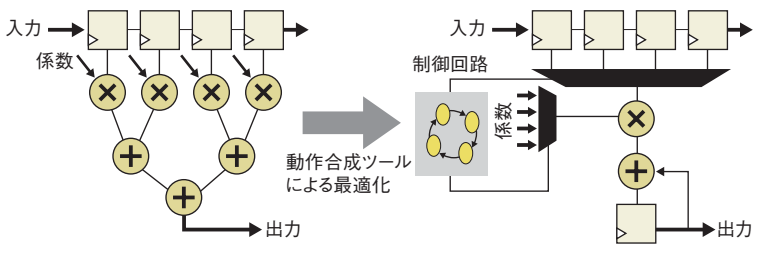


図2●動作合成による最適化の例
FIR（finite impulse response）フィルタに動作合成技術を適用した。左のマイクロアーキテクチャでは乗算器四つと加算器三つを使うが，1クロックで処理が終わる。右の例では，4クロック・サイクル追加されるが，乗算器と加算器は一つずつで済む。演算器を共有するための制御回路は動作合成で自動生成される。米Calypto Design Systems, Inc.のデータ。

身が設定することを推奨している。動作合成ツールとのインタフェースを確立できれば，動作合成ツールから情報を取り込み，自動的に設定することは技術的に難しくはない。しかし非常に便利にはなるが，検証が設計や合成ツールから独立している方がより安全である点に留意する必要がある。

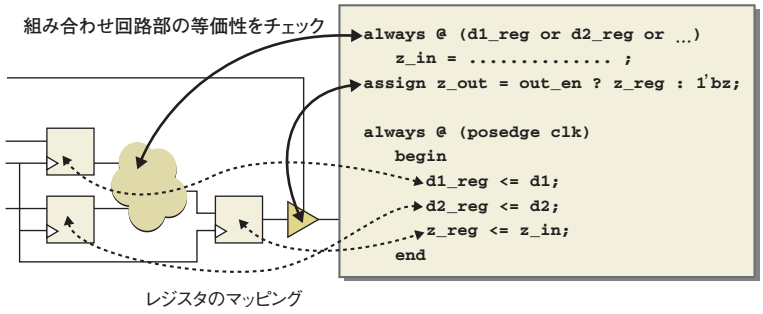


図3●従来の等価性検証ツール
これまでの市販の等価性検証ツールでは，レジスタ間の組み合わせ回路だけを，フォーマル検証エンジンで処理していた。レジスタの対応関係は，レジスタの名前や合成ツールからの情報で別途調べていた。米Calypto Design Systems, Inc.のデータ。

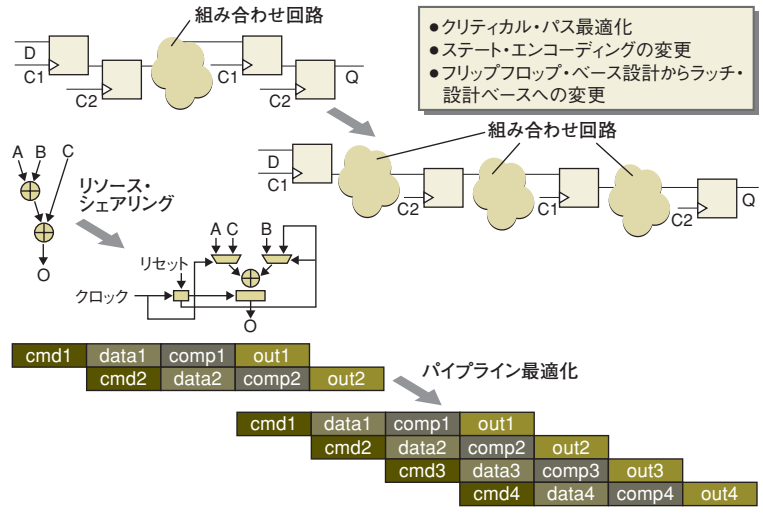


図4●順序回路のシーケンスが変わる
ここで紹介した処理のいくつかは，現在の市販の論理合成ツールでも実行できる。こうした処理前後の論理機能等価性の検証は，順序回路に対応したフォーマル検証エンジンがないと手間がかかる。米Calypto Design Systems, Inc.のデータ。

シミュレーションで設定の誤りなどを除く

TCLスクリプトから検証実行コマンドが発行されると，まず，上述したサイクルベース・シミュレータに，ランダム発生したパターンを入力してシミュレーションを実行する。これによって，検証対象の設計間に明らかな差異はないか，検証環境や制約の設定に間違いがないかを確認する。

シミュレーションでのチェックをパスすると，フォーマル・エンジンを使った等価性の検証に移る。今回の製品のエンジンは，複数の順序回路向け等価検証アルゴリズムを組み合わせ，最適化した独自の「ハイブリッド・エンジン」である。このエンジンの特徴として，ビットごとではなく，ワード単位で処理することを挙げられる。これで処理時間の短縮を図る。

従来の等価性検証ツールは，ビットごとで処理していたため，乗算器など演算量の多い回路には適用が難しかった。しかし動作合成ツールの主な適用先である，ビデオや画像，無線ベース・バンドなどの信号処理系のアプリケーションでは，乗算器はよく使われる回路である。今回の製品は，こうしたアプリケーションにおいても無理なく等価性検証を実行できる。

エンジンには二つのモード

今回の製品のフォーマル検証エンジンは，「バグ発見モード」と「完全検証モード」という独立した2つの実行モードを持っている。

バグ発見モードは，初期状態から，すべての入力条件を考慮しながら1つずつトランザクション[注14)]を進め，最短のサイクルでバグを見つけるモードである。特にマニュアルでシーケンスの変更を行った場合には効果的である。動作合成フローで使用した場合に，合成時のバグだけでなく，元の設計のバグを発見できた例があった。

完全検証モードは，設定された検証環境・制約の下で2つの設計の論理機能が完全に一致していることを保証するモードである。完全検証モードはバグ発見モードに比べ，シーケンスの

注14）今回の製品では，スループットとレイテンシのうち大きい方のクロック・サイクル数を1トランザクションの長さとしている。

より深い状態まで探索しなければならないため，一般的により多くの処理時間を必要とする。

デバグ用に反例のパターンを作成

今回の製品は，指定した出力ポートのペアや内部レジスタのペアごとに一致，不一致を報告する。不一致が見つかれば，波形データとテストベンチの両方で反例を出す。

従来の等価性検証ツールでは，組み合わせ回路部分ごとに独立した検証ができるため，それぞれに一致，不一致を報告していた。一方，今回の製品のような順序回路に向けた等価性検証では，指定したポイントごとに検証し，最初に見つかった不一致で処理を終了させる方法が効果的である。なぜなら，順序回路全体を対象としているため，それ以後の処理を継続しても，同じ原因からの不一致が他の箇所でも見つかることがあるからである。

今回の製品が出力する反例には，レジスタの初期値もしくはリセット・シーケンス，不一致が起こる場合の入力パターンなどが含まれる。波形データには，実行結果も含まれる。

テストベンチは，読み込んだ設計と同じ言語で出力され，同時に「Makefile」[注15)]も生成される。テストベンチは，設計と共にシミュレータで実行する必要があるが，デバグ用として重要であるだけでなく，報告された不一致を今回の製品から独立してチェックするという意味でも重要である。

バック・トレース機能が有効

今回の製品が出力した反例パターンを用いて不一致の原因を見つけ，それを修正することがデバグ作業となる。波形と記述を見比べただけで見つかるような単純な原因もあるが，短いサイクルの反例であっても，原因を突き止めることが容易でない場合もある。

このようなときに有効なのが，波形表示機能を持つデバグ・ツールである。原因追求の作業は，時間軸と回路の両方を不一致点からさかのぼっていくことが多いため，バック・トレース機能が特に有効である。市販のほとんどのHDLデバガがそのような機能を備えており，既に活用しているユーザーも多い。われわれは米Novas Software, Inc.の製品を薦めている。デバグ作業を通して不一致箇所をつぶすことができれば，再度同じスクリプトを使って等価性検証を行うことになる。M

注15）コンパイルを管理・効率化するための情報。今回の製品では，反例パターンと検証対象モデルのコンパイルで利用できる。

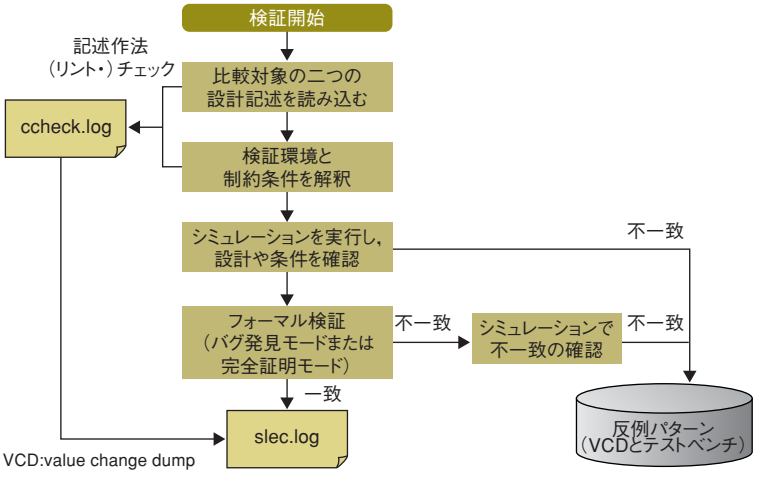


図5●今回の製品を使う検証フロー
フォーマル検証エンジンを適用する前に，サイクル・ベースの論理シミュレータを稼働して，ユーザーが設定した条件などをチェックする。米Calypto Design Systems, Inc.のデータ。

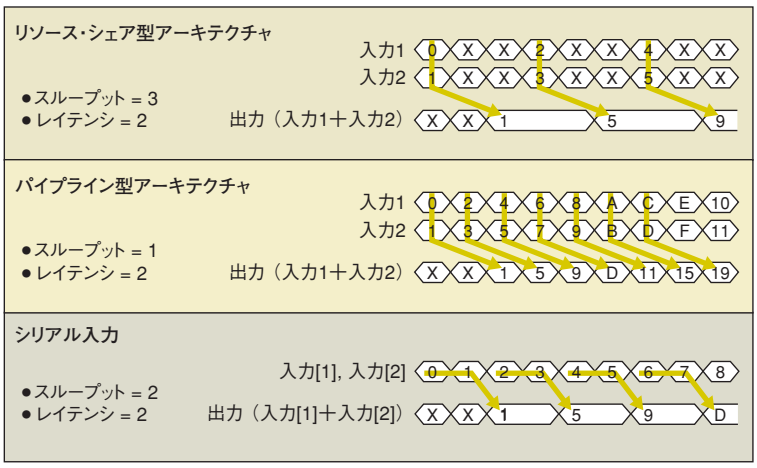


図6●スループットとレイテンシ
入力1と入力2を加算する動作の波形を，マイクロアーキテクチャ別に示した。それぞれでスループット（入力できる頻度）とレイテンシ（処理時間）が異なっている。米Calypto Design Systems, Inc.のデータ。