「情報処理」第36巻　第7号　別刷　平成7年7月　発行

ハードウェア/ソフトウェア協調設計のコンピュータ支援における問題および方法について

Giovanni De Micheli　　松　永　裕　介　(翻訳)

社団法人　情報処理学会

Vol. 36 No. 7　　情　報　処　理　　July 1995

解　説

ハードウェア/ソフトウェア・コデザイン

1. ハードウェア/ソフトウェア協調設計のコンピュータ支援における問題および方法について†

Giovanni De Micheli ††　　松　永　裕　介（翻訳）†††

1. はじめに

ハードウェアの設計とソフトウェアの設計を別々に行うのではなく，互いに考慮しながらそれらを同時に行う—いわゆるハードウェア/ソフトウェア協調設計（コデザイン）—は目新しいものではなく，従来から熟練したディジタルシステムの設計者の手によって解決されてきた問題であった．しかし，最近になって以下にあげるような要因から注目を集めるようになっている．一点目はエンドユーザの計算機の高性能化である．そのためにオペレーティングシステムと協調したアーキテクチャや，特定のアプリケーションプログラムを効率的に実行するハードウェアサポートなどが必要となってきている．また，製品コストの観点からすべてのハードウェアを作りつけにしてしまうのではなく，プログラム可能な部品を残しておき，改版を容易にするといった方策も用いられるようになってきている．二点目はプログラム可能なハードウェア[18]が特定の計算の高速化や新たなアーキテクチャの評価のために導入されてきたことである．このようなハードウェア上でいろいろと設定を変えながら実際にプログラムを走らせることでハードウェアとソフトウェアのちょうどよい協調関係を探ることができる．三点目は最近の論理合成[5]やシミュレーション技術の進歩が，ハードウェア/ソフトウェア協調設計のための統合的なCAD環境構築を可能にしつつある，ということである．

ハードウェア/ソフトウェア協調設計の技術が適用可能な分野は多岐にわたるので，ここでは非常におおまかな分類にしたがって，その分野における特定の問題や解法について見てゆくことにする．まず，対象となるのはディジタルシステムの設計であるが，これは大きく汎用の計算システムと専用の計算システム（あるいは制御システム）に分けることができる．前者は一般のアプリケーションプログラムを扱うことを前提としており，そのサイズはパームトップ（手の平サイズ）コンピュータからスーパコンピュータまでさまざまである．そのため，多様なオペレーティングシステムやプログラミング言語が用いられることになる．これに対して後者はある特定のアプリケーションに特化された計算のみを行う．このようなシステムのサイズも小さなものから大きなものまでさまざまである．

汎用システムも専用システムも，標準的な命令セットを持つプロセッサや種々の目的のためのコプロセッサをその構成要素として持つことがある．コプロセッサの主なものには浮動小数点計算のための演算コプロセッサがあげられる．その他に特化されたコプロセッサとしてメモリ管理やマルチプロセッサシステムのキャッシュコヒーレンスの制御を行うものがある．さらに極端な例としては，ある特定のアプリケーションに特化した処理や制御のみを行うユニットを持つような専用システムも考えられる．

コプロセッサやアプリケーションに特化したユニットの設計にはいくつかの異なったアプローチが考えられる（表-1）．その一つの端はASIC（Application-Specific Integrated Circuits）を用いたもので，多大の設計期間およびコストとひきかえに，与えられた目的に対して最良のパフォーマンスを実現することができる．ASIP（Application-Specific Instruction Processors）はアプリケーションに合致するように命令セット

† A Survey of Problems and Methods for Computer-Aided Hardware/Software Co-design by Giovanni DE MICHELI (Stanford University).

†† スタンフォード大
††† (株)富士通研究所　CAD研究部

表-1　さまざまな設計手法のトレードオフ

	汎用プロセッサ	コアプロセッサ	ASIP	ASIC
性　能	中	中	高	最高
消費電力	高	中	中／低	最低
柔軟性	中	高	高	低
設計期間	短(ソフトのみ)	中(ハードとソフト)	最長(ハードとソフト)	長(ハード)

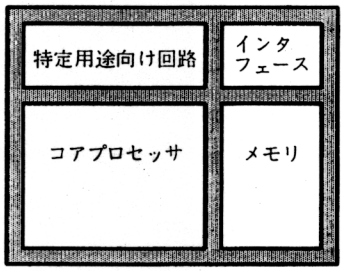

図-1　コアプロセッサを用いた集積回路の例

を選んだ専用のプロセッサで，高いパフォーマンスと共にプログラム可能という特徴を持つ．ASICの対極は，アプリケーション特有の機能をソフトウェアプログラムとして実現する方式である．

ディジタルシステムの実装方法は使用するテクノロジーに依存する．たとえばプリント板やマルチチップモジュール上に異なる種類の集積回路が実装される場合もあるし，それと等価なシステムが一つのチップとして実現できる場合もある．また，いくつかの汎用プロセッサは「コア」マクロセルとして提供されており，特定用途向けの論理回路と組み合わせて一つのチップに実装可能である．そのような例を図-1に示す．

以下2.では異なるアプリケーション分野—命令セットプロセッサ☆，信号処理用プロセッサ，および組込みシステム—におけるコデザインの問題を明らかにする．続く3.ではプログラム可能なハードウェアに基づいたアーキテクチャとコデザインの問題についての考察を行う．そして4.では，ハードウェアとソフトウェアの混在したシステムのためのシミュレーションや合成（設計）のためのCADツールを紹介し，この分野における研究動向について述べる．

☆ 通常のプロセッサ

2. ハードウェア/ソフトウェア協調設計の問題

いくつもの実用的なCAD（Computer-Aided Design：計算機支援による設計）ツールが開発されているとはいえ，ディジタルシステムのコデザインを行うにはいまだ人手が必要とされているのが現状である．ここでは，いくつかの応用分野におけるコデザインの問題を取りあげる．

2.1 命令セットプロセッサのためのコデザイン

命令セットプロセッサの設計においてハードウェア/ソフトウェア両面にわたった考察が必要となるのは，**命令セットの選択，キャッシュシステムの設計，そしてパイプライン制御**の三つである．

汎用のプロセッサの場合には他の既存のプロセッサとの互換性の要求によって命令セットの選択には大きな制約が課せられている．一方，ASIPのアーキテクチャを決める場合にはかなり自由に命令セットを選べるため，その得失をよく考慮する必要がある．一般にある命令を実装すれば，その命令を用いることによってプログラムの実行速度があがるという利益が生じるが，一方でその命令のために余分なハードウェアを必要とするという損失も生じる[12]．

メモリキャッシュの設計を行う際には，回路の動作速度とキャッシュサイズや（更新や無効化などの）アルゴリズムの性質との間でうまくバランスをとることが要求される．従来から適切なキャッシュサイズを決めるためにはシミュレーションが用いられている．また，マルチプロセッサシステムの場合，キャッシュの一貫性の保持も重要な問題である．ここでは例としてスタンフォード大のマルチプロセッサシステムFLASH[8]の設計を取りあげよう．FLASHのキャッシュ管理方法としてはある無効化のアルゴリズムが選ばれてい

た．FLASHにおける一つの大きな方針はプロトコルをハード化してASICチップの中に押し込めてしまうことを避けることであったので，キャッシュ管理用の汎用プロセッサ，プログラム可能なハードウェア，そしてASIPの三つの方法が検討されることになった．その中で，ASIPがメモリ操作などで汎用のプロセッサに勝り，また高い柔軟性を備えているという理由で最終的に選ばれることになった．

プロセッサのパイプラインの設計では**パイプライン ハザード**を起こさないような制御方法を実現する必要がある．この問題を解決するための手段としてはハードウェア的な方法とソフトウェア的な方法のどちらも適用可能である．ハードウェア的な方法の例としてはパイプラインの中身を排出させるものがある．一方，典型的なソフトウェアの方法としては（コンパイラが）命令を入れ換えたりNOP命令を挿入するものがある．この選択によってプロセッサ全体の性能は大きく影響を受ける．さらに，その性能評価は複雑であり，ハードウェア，ソフトウェア両方に対する適切なモデルが必要となる．このように，最も適切なパイプライン制御方式を見つける問題はハードウェア/ソフトウェア協調設計の一つといえる．PIPERはこの問題を扱うCADツールで，ハードウェアとソフトウェアのトレードオフを調べることによって良い実装方法を提示するものである[13]．また，パイプラインステージへの自動的な分割やパイプラインスケジューリング，さらにはハザード回避のための命令の入換えなどの機能も備わっている．

2.2 信号処理プロセッサのためのコデザイン

信号処理プロセッサ（Digital Signal Processor：DSP）は通信機器やビデオ・オーディオなどの分野で非常に重要な役割を果たしており，高性能かつ廉価なプロセッサの開発が強く求められている．そこで，大量生産によってコストを抑えるためにプログラム可能で汎用なDSPも作られている．また，携帯電話などに用いられるために低消費電力という制約も課せられる場合がある．

この分野においてはASIPが一つの解であろう．ここではASIPを用いた設計に関して，命令セットの選択とコード生成の二点について簡単に述べる．前者は汎用プロセッサの場合と同様の問題であるが，DSPに用いられる場合にはその重要度は比べ物にならないほど高い．それは，DSPに要求されるのがいくつかの機能だけで，特定のプログラムしか実行されないことに起因する．そのプログラムの実行にのみ特化して実行速度を早め，不必要なハードウェアを削るような命令セットの選び方も可能だからである．命令セットによる得失を整数計画問題を用いて定式化することによってこの問題を解く手法がPEASシステム[14]で提案されている☆．

命令セットの選択はそのためのコンパイラにも影響を与えている．一般にコンパイラは，言語依存のフロントエンド，中間コードの最適化，そしてコード生成のバックエンドの三つの段階からなるが，ASIPの設計においては命令セットに依存したバックエンドの部分の開発が重要な問題となる．効率のよいコードを生成するためにはASIPのデータパスに関する正確なモデルを用意することが必要である．最新のコード生成手法[15]では高位レベルのハードウェア記述言語より生成されたデータパス記述が用いられている．このようにハードウェアとソフトウェアは共に高位レベル記述言語によってモデル化される．そして，あたかもハードウェアのマクロライブラリ割当てを行うごとくにコード生成が行われるのである．

2.3 組込みシステムとコントローラ

組込みシステムはある特定のアプリケーションに特化した計算/制御システムであり，そのサイズや対象は多岐にわたる．手近のところでは皿洗い機のコントローラのような固定のプログラムを実行するマイクロプロセッサが組込みシステムであるし，もっと複雑で大規模な例としては航空機のガイドおよび管制システムがあげられよう．一般に，組込みシステムとは専用のハードウェアおよび専用のソフトウェアプログラムを持ち，さらには外の環境とやりとりするためのセンサおよびアクチュエータ☆☆を備えたシステムである，と定義することができる（図-2）．

組込みシステムは多くの場合，反応型システム（reactive system）という範疇に分類することが

☆ 訳者注：PEASシステムはDSP専用のシステムというわけではなく，ASIP用の合成システムである．本特集の別記事参照．

☆☆ 訳者注：与えられた電気信号に従って動くもの．たとえばサーボモータや発音装置や発光装置など．

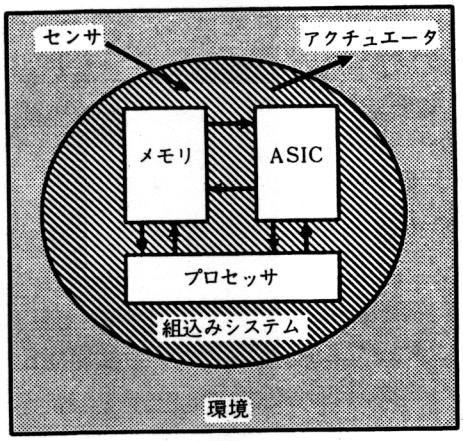

図-2　組込みシステムの模式図

できる．反応型システムとは，特定の入力印加に対して定められた機能を実行することで環境に反応するようなシステムを意味する．そのような機能のうち，あらかじめ定められた時間内に実行されなければならないようなものもある．この場合，そのようなシステムはリアルタイムシステムと呼ばれることもある．反応型でかつリアルタイム性が必要とされるシステムの例としては自動車のエンジンなどの制御や，工業用ロボット，携帯電話などがある．

組込みシステムの設計に固有の問題は，周辺機器（センサやアクチュエータ）とあらかじめ定められたプロトコルに従ってやりとりを行わなければならない，ということである．多くの場合，システム設計者は周辺機器には標準的な部品を用いて，インタフェース部分の標準化をはかっている．デバイスドライバはASICを用いてハードウェア的に実装されることもあるし，マイクロプロセッサ上のソフトウェアとして実装されることもある．そのどちらを選ぶかは利用可能な入出力ポートの数や周辺機器との通信にかけられる時間の余裕に左右される．Chinookシステム[3]はこのような周辺機器を持った組込みシステムのためのCADツールである．

現在，非常に多くの品種の組込みシステムの設計が行われており，その設計の自動化および計算機支援の重要性は非常に高く認識されている．組込みシステムではハードウェアとソフトウェアという異質のものが相互作用を持つのでその設計の自動化は容易ではない[19]が，いくつかの手法が提案されている．それらは4.で簡単に紹介を行う．

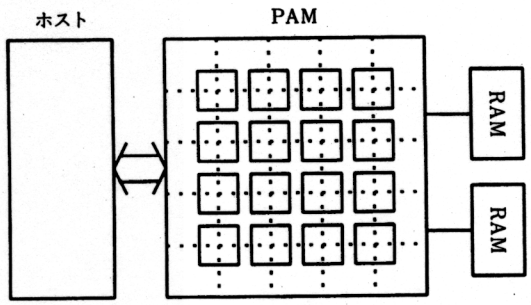

図-3　PAM (Programmable Array Memories)

3. プログラム可能なハードウェアの利用

最近ではFPGA（Field Programmable Gate Array）のようなプログラム可能なハードウェアの登場によってハードウェアとソフトウェアの差異は小さくなっている．ここでは素子間の接続関係や素子の機能を外からロードすることで設定可能なFPGA[18]について考えてみよう．

プログラム可能なハードウェアの応用はいろいろなものが考えられるが，FPGAのチップを搭載したシステムで実際に用いられている例としては，ソフトウェアで書かれていた機能の高速実行や，プロトタイプ評価のためのハードウェアのエミュレーションがあげられる．前者の場合，ソフトウェアプログラムのどの部分をプログラム可能なハードウェアに置き換えるかを決めることがコデザインの問題となる．また，後者の場合，ハードウェアとその上で実行されるソフトウェアからなる複雑なシステムの評価・確認を，FPGA上で模倣したハードウェアに対して実際のプログラムを実行させることで行える．FPGA上での設定は可変なのでハードウェアを固定することなくハードウェア/ソフトウェアのトレードオフを探ることが可能である．

3.1 ソフトウェアの高速実行

一般にソフトウェアプログラムにはその実行性能を左右するボトルネックが存在する．コプロセッサはそのようなボトルネックを高速に実行するために役に立つ場合もある．特に計算の局所的な並列性をうまく活かせるような場合にはコプロセッサによる高速化は有効である．しかし，通常のコプロセッサは特定の目的のために用意されてお

り，どのようなアプリケーションに対しても有効に働くわけではない．

これに対してプログラム可能なハードウェアを用いたコプロセッサの場合には，高速化に関しては通常のコプロセッサと同等の能力を有し，さらに任意のアプリケーションにも適応可能という特徴を持つ．このようなコプロセッサの例としてPAM (Programmable Active Memories) が提案されている[2](図-3)．このシステムはFPGAを搭載した基板と，ホストコンピュータとのインタフェース用のローカルメモリより構成される．ソフトウェアプログラムのうち，ボトルネックとなる部分はFPGAで実行される．このため，FPGAを設定するためのデータがソフトウェアプログラムからコンパイルされる．暗号解読，データ圧縮，文字列のマッチング，物理系システムをモデル化した連立方程式の解法などのさまざまな分野でPAM応用した例が報告されてる．実験結果によるとホストコンピュータによる実行に比べて10倍から100倍の高速化を達成している[2]．

ここでのハードウェア/ソフトウェア協調設計の大きな問題は，ハードウェアで実行させると効果的なプログラムの部分を見つけること，および，それをFPGAで高速に実行させるためのコンパイルを行うことである．前者の問題は自動化されておらず，研究課題として残っている．後者はハードウェアの合成アルゴリズムを用いることができ，ハードウェアの実行速度の最適化技術が有効に利用されている．

3.2 ハードウェアのエミュレーションとプロトタイプ評価

プログラム可能なハードウェアを用いたプロトタイピングは，実際にハードウェアを作る前に確認や評価を行えるので，コストのかかる設計のやり直しを削減する上でとても有用である．プロトタイピングはシステムシミュレーションに比べて，設計の正しさや性能評価に関するより確度の高いデータを提供することができる[16]．

さらに，複数のハードウェアとその上で走るソフトウェアからなるような複雑なディジタルシステムのプロトタイピングも設計者には魅力的であろう．このようなエミュレーションシステムを用いれば，ハードウェア（と同時にソフトウェア）の仕様を変更する可能性を残したままで，実際のソフトウェアをプログラム可能なハードウェア上で実行（エミュレート）することができる．最終的なハードウェアの仕様が固まったら合成システムを用いてシリコンの上に本当のハードウェアとして実装することが可能である[5]．このとき，合成システムの入力となるモデルとしてはVerilog HDL☆のようにエミュレーションのときに用いられるモデルと互換性のあるモデルが用いられる．

4. ハードウェア/ソフトウェア混在のシステムの設計確認と合成

ここではハードウェア/ソフトウェア混在のシステムに対する既存のCADツールのアプローチについて見てゆくことにする．

4.1 混在システムのシミュレーション

回路やシステムの設計確認 (validation) とは，設計に誤りがなく製造段階に移れるかどうかの確認を行う作業である．回路やシステムの規模が増大するにしたがって設計確認の重要度も増しているが同時に複雑さも増している．設計確認は形式的検証やシミュレーション，さらには3.2で述べたようなプロトタイピングの技術を用いて行われる．検証ツールは二つの設計の等価性を調べたり，デッドロックが起きることがない，といった特定の性質を満たしているかを証明するのに用いられる．現在，規則性や抽象化を利用して問題の複雑度が下げられるような問題に対しては形式的検証も実用的に用いられ始めているが，複雑なハードウェア/ソフトウェアシステムを完全に検証するには程遠い．一方，シミュレーションは従来から用いられてきた手法で，あらかじめ用意された入力パタンに対する出力パタンを調べることでシステムの設計が仕様どおりか確かめるものである．ハードウェア/ソフトウェア混在のシステムをサポートするシミュレーションツールは数えるほどしかなく，また，限られた入出力パタンを調べているだけなのでシミュレーションでは設計の正しさを完全には保証することはできない，などの問題点もあるが，ハードウェア/ソフトウェア協調設計にはシミュレーションツールは必要不可欠なものと言えよう．

現在ではVerilog HDLやVHDLといったハ

☆ 訳者注：ハードウェア記述言語の一つ

ードウェア記述言語で記述されたレジスタ転送レベルのハードウェアモデルを高速にシミュレーションする商用ツールが広く用いられている．また，標準的なマイクロプロセッサのレジスタ転送レベルのモデルもそれらの言語で記述されたものが用意されている．したがって，シミュレーションのために，ハードウェア記述言語を用いてハードウェア/ソフトウェア混在システムをモデル化することは原理的には可能である．しかし，実際には，ある程度複雑なソフトウェアプログラムを実行させるにはレジスタ転送レベルのシミュレーションは遅すぎて使い物にならない．

そこで，プロセッサの動作は命令セットレベルで，特定用途向けのハードウェアの動作はレジスタ転送レベルでシミュレーションを行う協調シミュレーション（co-simulation）と呼ばれる手法が提案されている[9]．一つのシミュレーション環境に複数のシミュレータを統合化するにはいくつかの方法が考えられる．特に，シミュレーションしようとしているプロセッサがシミュレーション用のホスト計算機のCPUと同一の場合（たとえばSPARCを含んだシステムのシミュレーションをSPARCを搭載したワークステーションで行う場合），ソフトウェアを実際にホストで実行させる処理とハードウェアシミュレーションを組み合わせることで協調シミュレーションを行うことができる．実際にプロセス間通信機構を用いてこのメカニズムを実現した例が報告されている[1),17)]．

4.2 混在システムの合成

計算システムの計算機支援による合成（ここでは協調合成：　co-synthesisと呼ぶ）は既存のハードウェア合成手法の自然な進化である[5]．このような協調合成を行うためには，システム全体が統一的にモデル化され，（人手か自動で）ハードウェアとソフトウェアに分割される必要がある．ハードウェアの部分は既存のハードウェア合成ツールを用いて専用回路で実現され，ソフトウェアの部分はそのハードウェア上で実行可能なコードとして生成される．協調合成はまた，ハードウェアとソフトウェアで実現される機能の間でのインタフェースと同期をとる手段を提供しなければならない．

4.2.1 システムレベルのモデル化

システムレベルでのモデル化は協調合成を行う上で非常に重要である．提案されている抽象モデルは次のような二つのグループに分けることができる．

- コントロール/データフローグラフ（CDFG）
- 協調する有限状態機械

コントロール/データフローグラフ[5]はディジタルシステムの動作をタスクの集合とデータや制御の依存関係の集合として抽象化するもので，ハードウェア，ソフトウェアのどちらもモデル化することが可能である．実際に，CDFGをハードウェアやソフトウェアの言語モデルから抽出することは容易であり，ハードウェアの高位レベル合成やソフトウェアのコンパイルで有効に用いられている．

有限状態機械はシステムの状態および遷移を表すもので，ハードウェア，ソフトウェアともに適用可能である．Statechart[7]は並行動作する階層的な有限状態機械をモデル化するスタイルの一つである．CDFGと同様にハードウェア，ソフトウェアの言語モデルから有限状態機械を抽出することもできる．ただし，協調した有限状態機械の場合にはその動作モデルは唯一ではない．通常はすべての有限状態機械が同期して遷移するようなモデルが用いられるが，ソフトウェアはその実行時間が未知であったりデータに依存したりする場合があるのであまり適切ではない．そこで，各有限状態機械が非同期的に遷移するモデルも提案されている．また，ハードウェア部分の実行時間とソフトウェア部分の実行時間の単位が不釣合なので，任意の遅延時間を持つイベントを通して有限状態機械が通信をするようなハードウェア/ソフトウェア混在システムのモデルが提案されている[4]．有限状態機械を用いたモデルでは既存のハードウェア合成/検証の技術を拡張して適用することが可能であるが，最初のシステム仕様によってハードウェアとソフトウェアの分割の仕方が大きく制約を受けてしまうという欠点もある．そこで，ハードウェアとソフトウェアのトレードオフをとるために自動的に分割を行う技術の研究ではCDFGのモデルが用いられている．

4.2.2 システムレベルの分割

ある処理を専用のハードウェアで実現しようとすれば並列性をうまく活かして高速処理が行えるかもしれないし，一方，汎用の高性能プロセッサを用いてソフトウェア的に処理すれば廉価にシステムを構築することができるかもしれない．ただし，汎用プロセッサの場合には基本的に逐次処理であり，特定の処理を高速に行う支援機構を持たないので，性能的には専用ハードウェアにはかなわない．このように，どの部分をハードウェアで作ってどの部分をソフトウェアで実現するかを切り分けることはコストと性能を大きく左右する問題である．

システムレベルでのハードウェアとソフトウェアの分割に関してはいくつかの研究がなされている[10),11),17)]．ここでは，ハードウェアを用いた高速化のための専用コプロセッサの合成[11)]と，性能を左右しない機能のソフトウェア化[10)]という二つのアプローチについて簡単に紹介する．前者は性能を最大限にあげるためのものであり，後者は性能を一定に保ったままコストを最小化しようというものである．

COSYMA[11)]は与えられたシステムの仕様を分割して，自動合成された専用のコプロセッサを用いることで実行速度の向上をはかる．入力となるシステムモデルは，C 言語に性能の制約を記述できるようにした C* という言語を用いたソフトウ

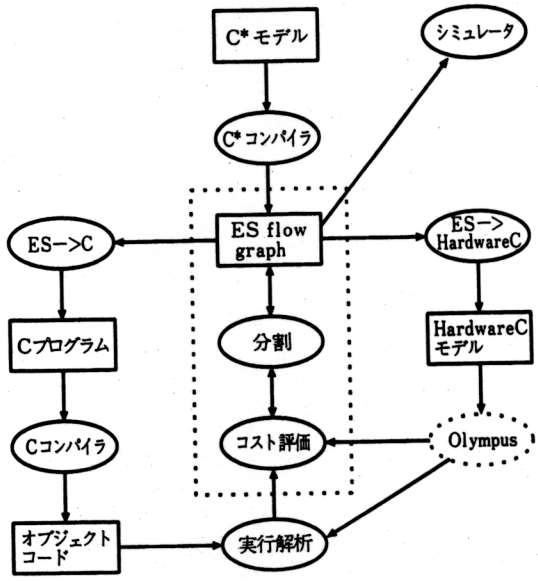

図-4 COSYMA システム

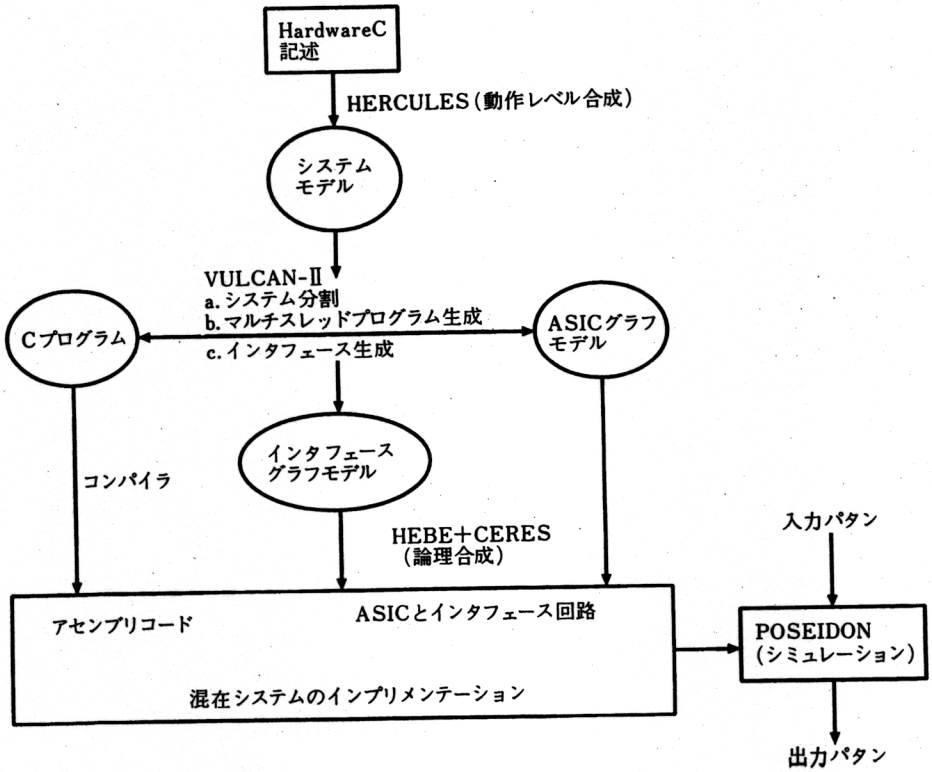

図-5 VULCAN システム

ェアプログラムとして記述される．システムモデルはCDFGに変換され，その中からボトルネックとなっている処理が抽出される．そして，その処理を行う部分がハードウェア側に移されて，実際にそのための専用ハードウェアが合成される．実験として，SPARC上で動作する高品位テレビ用のクロマキーアルゴリズムが用いられた例では，全体の90%の計算時間を占める内部ループを抽出することで，約3倍の高速化が達成された．このときに作られた専用ハードウェアは約17,000ゲートであった．COSYMAシステムの構成図を図-4に示す．

VULCAN[10]は対照的にハードウェアモデルを用いたシステムモデルを出発点として，性能に影響のない部分をi 8086やR 3000のような汎用プロセッサ上のソフトウェアとすることでコストの削減をはかるツールである．設計対象のシステムはHardwareCと呼ばれるC言語に似たシンタックスを持つハードウェア記述言語を用いて記述され，レイテンシーやスループットなどの性能の制約も同時に指定される．システムモデルはCDFGに変換され，汎用プロセッサ上で実行されるサブルーティンと，Olympus合成システム[6]で合成されるハードウェアの仕様に分割される．実験的に，CSMA/CDプロトコルを扱うイーサネット用の制御プロセッサの設計に適用した例では，R 3000をプロセッサとして用いたハードウェア/ソフトウェア混在システムで，専用ハードと同等の性能を持ちながら約20%の回路量を削減できたと報告されている．VULCANシステムの構成図を図-5に示す．

5. おわりに

現時点ではハードウェア/ソフトウェア協調設計に対するCADのサポートは十分とはいえないが，その必要性は高く認識されており，将来の研究開発にかかる期待は大きい．しかし，いくつかの未解決の問題がこの分野の発展を阻害していることも事実である．まっさきに取り組まなければならないのは，ハードウェア/ソフトウェア混在システムのよりよい抽象モデルと，それを矛盾なく記述できる言語の開発である．2番目はコストと性能の評価の問題である．これはハードウェアとソフトウェアの分割，合成を行う上では不可欠なものであるが，システムの抽象モデルが物理的な実装から乖離しているために解決は容易ではない．3番目は協調合成と協調シミュレーション技術である．現在の手法にはまだまだ改良の余地が残っているものと思われる．もう一つ，協調シミュレーションでは完全に正しいという検証は行えないので，ハードウェア/ソフトウェア混在システムに適用可能な形式的検証手法は非常に重要なものとなろう．

参考文献

1) Becher, D., Singh, R. and Tell, S.: An Engineering Environment for Hardware-Software Co-simulation, Proceedings of DAC, pp. 129-134 (1992).
2) Bertin, P., Ronchin, D. and Vuillemin, J.: Introduction to Programmable Active Memories, in J. McCanny, J. McWhirter, E. Schwartzlander (Editors), Systolic Array Processors, Prentice Hall (1989).
3) Chou, P., Walkup, E. and Borriello, G.: Scheduling Strategies in the Co-Synthesis of Reactive Real-Time Systems, IEEE Micro (Aug. 1994).
4) Chiodo, M., Giusto, P., Jurecska, A., Hsieh, H., Lavagno, L. and Sangiovanni, A.: A Formal Methodology for Hardware/Software Co-design of Embedded Systems, IEEE Micro (Aug. 1994).
5) De Micheli, G.: Synthesis and Optimization of Digital Circuits, McGraw-Hill (1994).
6) De Micheli, G., Ku, D., Mailhot, F. and Truong, T.: The Olympus Synthesis System for Digital Design, IEEE Design & Test, pp. 37-53 (Oct. 1990).
7) Harel, D., Pneuli, A., Schmidt, J. and Sherman, R.: On the Formal Semantics of Statecharts, Logic and Computer Science (1986).
8) Horowitz, M. and Keutzer, K.: Hardware-Software Co-Design, Proceedings of SASIMI, Nara, pp. 5-14 (1993).
9) Gupta, R., Coelho, C. and De Micheli, G.: Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Component , Proceedings of DAC, pp. 225-230 (1992).
10) Gupta, R. and De Micheli, G.: System Cosynthesis for Digital Systems, IEEE Design & test, Vol. 10, No. 3, pp. 29-41 (Sep. 1993).
11) Ernst, R., Henkel, J. and Benner, T.: Hardware-Software Co-synthesis for Microcontrollers, IEEE Design & Test, pp. 64-75 (Dec. 1993).
12) Hennessy, J. and Patterson, D.: Computer Architecture : A Quantitative Approach,

Morgan Kaufmann (1990).
13) Huang, I. and Despain, A.: High-level Synthesis of Pipelined Instruction Set Processors and back-end Compilers, Proceedings of DAC, pp. 135-140 (1992).
14) Alomary, A., Nakata, T., Honma, Y., Imai, M. and Hikichi, N.: An ASIP Instruction set Optimization Algorithm with Functional Module Sharing Constraints, Proceedings of ICCAD, pp. 526-532 (1993).
15) Marwedel, P.: Tree-based Mapping of Algorithms to Predefined Structures, Proceedings of ICCAD, pp. 586-593 (1993).
16) Maliniak, L.: Logic Emulator Meets the Demands of CPU Designers, Electronic Design (Apr. 1993).
17) Thomas, D., Adams, J. and Schmitt, H.: A Model and Methodology for Hardware-Software Co-design, IEEE Design & Test, Vol. 10, No. 3, pp. 6-15 (Sep. 1993).
18) Trimberger, S.: A Reprogrammable Gate Array and Applications, IEEE Proceedings, Vol. 81, No. 7, pp. 1030-1041 (July 1993).
19) Wolf, W. and Frey, E.: Tutorial on Embedded System Design, Proceedings of ICCD, pp. 18-21 (1992).

(平成7年3月9日受付)

Giovanni De Micheli

1979年Politecnico di Milano大学原子工学博士．1980年California大学Berkeley校電気工学・計算機科学科修士．1983年同大博士課程修了．1984年～1986年IBM T. J. Watson研究センターで設計自動化のプロジェクトリーダーを勤める．その後，Politecnico di Milano大学電子工学科やHarris Semiconductorに籍を置く．現在，Stanford大電気工学科およびコンピューター科学科準教授．1988年にPresidential Young Investigator賞授賞．1987年IEEE Transactions on CAD/ICASの最優秀論文賞を授賞．1983年と1993年にDesign Automation Conferenceの最優秀論文賞を授賞．1988年，1989年にInternational Conference on Computer Design (ICCD) のtechinal chairmanおよびgeneral chairmanを勤める．集積回路の計算機支援による設計 (CAD)，特にVLIS回路の自動合成，最適化および検証に興味を持つ．著書および共著「Synthesis and Optimization of Digial Circuits」(McGraw-Hill)，「High-level Synthesis of ASICs under Timing and Synchronization Constraints」(Kluwer)，「Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation」(Martinus Nijhoff Publishers) などがある．IEEEフェロー，IEEE Transactions on VLSI SystemsとIntegration: the VLSI Journal誌の編集委員．