解　説

3. ハードウェア記述言語の比較†

神原弘之††　安浦寛人†††　Pankaj Kukkal††††
Hideaki KOBAYASHI††††　野地保†††††　小栗清†††††††

1. まえがき

ハードウェア記述言語の標準化の活動などをとおして，ハードウェア記述言語を使った設計手法の有用性についてさまざまな議論が行われてきた．しかし，どのハードウェア記述言語がそれぞれの集積回路やシステムの設計手法に適合するのか，各ハードウェア記述言語の特徴の違いがどのように実際の設計に影響するのかといった議論はあまり行われていない．これは，ハードウェア記述言語を比較検討する方法論が確立していないこと，さらには各言語の定義がその成立過程のさまざまな要因により異なっており用語の統一すらなされておらず，比較することが容易でないことによる．ここでは，本特集で取り上げた4つの言語を対象とし，それらの間の本質的な相違が具体的な設計においてどのような差異を生みだすかを明らかにすることを目指す．

本稿ではハードウェア記述言語の本質的な相違をもたらす言語仕様中の要素について議論し，現在標準化や普及活動が進められている4つの言語：UDL/I, VHDL, Verilog HDL, SFL について，対象とする設計レベルと設計手法，意味定義，信号変化のタイミングに関する仮定，データ型，文法などに注目して比較を行う．

† A Comparison of Hardware Description Languages by Hiroyuki KANBARA (Advanced Software Technology & Mechatronics Research Institute of Kyoto), Hiroto YASUURA (Dept. of Information Systems, Kyushu Univ.), Pankaj Kukkal, Hideaki KOBAYASHI (Department of Electrical and Computer Engineering, University of South Carolina), Tamotu NOJI (Mitsubishi Electric Corp.) and Kiyoshi OGURI (NTT).

†† 京都高度技術研究所
††† 九州大学大学院総合理工学研究科
†††† サウスカロライナ大学
††††† 三菱電機
††††††† NTT

比較をより具体的に行うために，簡単なマイクロプロセッサ (KUE-CHIP) をベンチマークとして各ハードウェア記述言語で記述し比較を行った．ベンチマーク回路は，アーキテクチャは簡素であるがディジタル回路の基本的な要素（算術演算回路，レジスタ，カウンタ，メモリなど）を含む実在する8ビットマイクロプロセッサである．各言語による記述のステップ数，記述に要した時間，さらにはシミュレーションや論理合成を行った際の結果などを報告する．また各ハードウェア記述言語の意味や文法の相違が実際の記述上でどのように現れるかを記述の一部を示して解説する．

2. 各ハードウェア記述言語の特徴

2.1 比較項目[1)]

（1） 対象となる設計レベルと設計手法（言語のスコープ）

ディジタルシステムの設計は，階層的に幾つかの設計レベルに分けて行われる．これに対応してハードウェア記述言語もその記述の対象とする設計レベルを想定している．設計レベルを抽象度の高いほうから低いほうへ並べるとアルゴリズム／アーキテクチャレベル，動作レベル，レジスタトランスファレベル，ゲートレベル，スイッチレベル，トランジスタレベル（アナログ信号を取り扱う）などに分けられる．アルゴリズム／アーキテクチャレベルを記述するハードウェア記述言語は，ディジタルシステムの数学的なモデルを基礎とする必要がある．一方，スイッチレベルまたはトランジスタレベルを記述するハードウェア記述言語は，実際のハードウェアで起こる物理現象をモデル化する言語でなければならない．このため，これらのすべての設計レベルを一つのハード

ウェア記述言語でカバーしようとすると種々の困難な問題が起きてくる．

また，大規模なディジタルシステムを正しく設計するために，種々の設計手法が提案され，実際に用いられている．これらの設計手法は，主に設計の自由度を制限し，設計の誤りや物理現象の不確定さによる回路の誤動作をできるだけ起こさないようにして，性能のよい回路を迅速に正しく設計するためのものである．ハードウェア記述言語もある特定の設計手法を仮定すると，その手法で採用されている設計に対する制限を言語自身の前提として受け入れなければならない．一方，広く種々の設計手法に対応しようとすると設計中に記述すべき情報が大きくなる．また，実際に設計している回路だけでなくその回路が利用されたりテストされたりする環境までを含めて記述することまで言語に要求する立場もある．

（2） 意味定義（言語のセマンティクス）

ハードウェア記述言語は，本質的に並列プログラミング言語である．並列プログラミング言語の意味に関する共通の理解を得ることは，人間の直観と比較的親和性のよい逐次型のプログラミング言語の場合と比べ困難がともなう．しかし，標準化を目指すハードウェア記述言語では，その意味が言語仕様の中で明確に定義されていないと，シミュレータや論理合成系などの処理系ごとに，シミュレーションの実行結果が異なったり，論理合成系が生成した回路の動作がユーザの欲するものと異なるといった問題が起きる．ソフトウェアの並列プログラミング言語でも，その意味が明確に言語仕様で定義されているものは多いとはいえず，言語処理系の実現によって説明されるものも多い．しかし，シミュレーション（ソフトウェアによる疑似的な実現）と論理合成（現実のハードウェアによる実現）という異なる2種の実現形態をもつハードウェア記述言語では，言語処理系によりその意味を与える方法は多くの矛盾を生みだす原因となるので，言語の意味定義は重要な問題である．

（3） タイミングに関する仮定

ハードウェア記述言語の意味定義において大きな問題となるのが信号変化のタイミングに関する仮定の問題（以下ではタイミングと呼ぶ）である．

ハードウェアの動作は基本的に回路内の各部ごとに並列に行われる．広く用いられている同期式順序回路設計手法においては，設計者はクロック信号によって並列に動作する独立した各部の動作の同期をとることで，並列処理のタイミングに関する問題を解決する．すなわち設計者は，回路の動作をクロック信号に同期した状態遷移機械の状態の遷移と，同一状態内では並列に実行される動作の組み合わせによって実現する．

設計の初期の段階では，同期式順序回路としての実現の詳細化や非同期回路による実現部分などがまだ明確にされていないことがある．このような段階では，状態遷移機械を用いないで回路の動作を直接記述できることが便利な場合がある．この場合，従来の言語では逐次的な記述により回路動作を記述するようになっていた．多くのシミュレータは逐次処理を行うので，このような記述法はシミュレータの利用のみを前提とした場合には十分であった．

一方，論理合成を前提とした記述では，逐次的な記述を合成の仕様として与えると設計者の意図しない逐次性までも指定することになり，並列処理による性能向上が望めない．また，同期式回路設計手法以外の信頼性の高い論理合成手法が確立していない現在，逐次的な回路動作の記述が合成の入力としてどこまで利用できるかは大きな問題である．

また製造プロセスの違い，温度変化またはチップごとの遅延特性のばらつきなどにより，実際のハードウェアの動作を完全に規定することは現在の技術では困難であり，システム内で同一時刻に発生する互いに因果関係のない信号変化の順序を決定することは一般的には不可能である．この立場からハードウェア記述言語の意味論は，「動作の非決定性」の概念を含むべきという主張がなされている．言語の意味を「決定的」なものとするならば，その言語による記述として表される回路の動作はあいまいさを含まないものでなければならない．しかし，このような「決定的な動作」は論理合成系に対する仕様の呈示としては必ずしも適切とは言い難い．一方，「非決定性」を言語仕様に導入することは，高速なシミュレーションの実現を阻害する大きな要因になる．ここにハードウェア記述言語の意味決定のむずかしさの一因がある．

（4） データ型

ハードウェア記述言語の記述力を決定する要素の一つにデータ型の機能があげられる．多くのハードウェア記述言語は，実際のハードウェアに密着したデータ型として，信号線などの記憶を保持しないデータ型と，レジスタなどの記憶を保持するデータ型の二つを備えている．ハードウェアに密着したデータ型を用いることで言語のユーザは，論理合成の結果を予測または制御できる．一方，抽象データ型のようにユーザがデータ型を定義できるような機能を言語が備えていると，より動作の記述の抽象度が高い上位や抽象度が低い下位に言語のスコープを拡張する際などに都合がよい．さらに，抽象度の高いデータ型は，ハードウェアの実現手段を論理合成系にまかせることで，合成の結果に幅広い自由度をもたせることができる．

（5） 文法（言語のシンタックス）

しばしば論議される構文要素の種類や数は，言語の特性を議論するのにそれほど本質的ではない．問題は，ハードウェア記述言語の文法が反映している言語の意味である．すなわち，その言語が記述の対象とするシステムをどのようにモデル化しているかである．たとえば，逐次的な動作と並列動作の2種類のハードウェアの動作を含む言語では，これらを明確に分ける構文要素が必要である．また，クロック信号などのシステム全体の同期信号を仮定したモデル化がなされているならば，状態遷移機械のような離散時間でモデル化した表現をとりいれることは容易であるが，非同期式の順序回路の仕様を表現することはむずかしくなる．以下，スコープ，意味定義，タイミング，データ型，文法に注目して，UDL/I, VHDL, Verilog HDL, SFL の一般的な特徴を比較する．

2.2 UDL/I[2),3)]

（1） スコープ

UDL/I (Unified Design Language for Integrated Circuit) は，論理合成系に与える回路の動作仕様を記述する言語として設計された[2),3)]．単相クロックを用いた同期式回路設計という広く用いられているディジタル回路の設計手法に適したハードウェア記述言語である．しかし，非同期式回路も記述できるように設計されている．言語の設計が処理系とは独立に行われたため，特定の CAD ツールには依存しない言語である．UDL/I がカバーする範囲は，レジスタトランスファレベルからゲートレベルまでである．シミュレーションあるいはテストのための波形記述は，次期言語仕様で加えられる予定である．

（2） 意味定義

明確な意味定義が言語仕様書[2)]の10章に記述されている．意味定義は，UDL/I の言語要素のフルセットから，それと等価なコアサブセットと呼ばれる言語のサブセットへのマッピングにより定義されている[3)~5)]．コアサブセットのフォーマルな意味定義は，NES モデル (Nondeterministic Event Sequence Model)[4)] を用いて説明されているが，NES モデルは現在のところ言語仕様書には含まれていない．言語の意味論は，シミュレーションアルゴリズムと独立なものであることを目指しているが，高速なシミュレーションアルゴリズムと論理合成向けの言語仕様の間の開きが大きいため，その目標は必ずしも達成されているとは言いにくい．

（3） タイミング

タイミングについては，不確定な回路動作を扱うため非決定性の概念が導入されている[3),5)]．高速なシミュレータのインプリメントと非決定性との整合が困難な箇所については，シミュレータの実現の中で処理系によって自由に実装法を定義してよい部分として，言語仕様書に「処理系依存」部分として明記している．タイミングモデルの数学的な定義は NES モデルで与えられている．

（4） データ型

UDL/I のデータ型は固定されており，組合せ回路の出力端子に相当する記憶をもたないデータ型（ターミナル）と記憶をもつデータ型（レジスタ，ラッチ，RAM, ROM）が与えられている．

（5） 文法

UDL/I の基本的な枠組みは一般的な非同期回路も取り扱えるが，広く普及している同期式回路設計が簡単に記述できる文法が提供されている．たとえば，オートマトン記述，タスク転移などの状態遷移機械を表現する要素を備え，同期式回路設計に慣れ親しんだ設計者の記述の労力を削減する工夫が行われている．また，リセット信号など，広く用いられている非同期的な制御信号も簡単に組み合わせて記述できる．

レジスタトランスファレベルの動作を表す記述と，回路の構造つまりゲートレベルの素子の接続関係を表す記述では文法が明確に異なり，動作についての記述であるか構造についての記述であるか陽に区別して記述できる．

2.3 VHDL[6)~9)]

（1） スコープ

VHDL(VHSIC Hardware Description Language)は，並列プログラミング言語 ADA をベースに電子機器のドキュメンテーション用に開発された，特定の CAD および設計手法に依存しないハードウェア記述言語である[6)]．言語設計の初期の段階から，ハードウェアのシミュレーションモデルをプログラミングとして記述すること，またソフトウェアとハードウェアを統合して記述することを目指している．言語が設計された時期の関係で，近年の論理合成技術の進歩は必ずしも十分には反映されていない．VHDL がカバーする範囲は，アーキテクチャ／アルゴリズムレベルからゲート／スイッチレベルまでときわめて広い．そのほかにも設計を検証するためのテストパターンやシステムが動作する環境なども VHDL を用いて記述することができる．

（2） 意味定義

明確な意味定義は，IEEE 標準 1076-1987 には明示されてない[6)]．簡単な意味定義は，言語仕様書に文法の説明とともに示されている．すでに，幾つかの処理系の間で，意味の解釈の違いが問題となっている．

（3） タイミング

タイミングに関する意味定義は，詳細までは明確にされていない．信号変化に関する意味は決定性の考え方に基づいている．これは，この言語がシミュレーション用の言語として開発されたことに起因している．特定の設計手法を仮定しておらず，非同期的な回路動作を基本としている．しかし，抽象データ型を利用したモジュールの抽象化によって抽象度の高いモデルを定義する際に，タイミングについてもある程度の抽象化が図れるようになっている[7)~9)]．

（4） データ型

抽象データ型の機能を備えており，ユーザによる新しいデータ型を定義することが可能である．VHDL では，記憶をもつデータ型と記憶をもたないデータ型を陽に区別することができない．現在の論理合成技術では，記憶をもつデータ型はフリップフロップとして，また記憶をもたないデータ型は信号線として実現する方法が一般的であり，このことが VHDL 記述の論理合成を困難にしている．しかし，C などのプログラミング言語に習熟した利用者の利用を考えると，宣言した変数がレジスタに割り当てられるのかメモリ上に割り当てられるのかをコーディングの際に意識しなくてもよいように，論理合成がすべて解決してくれるのが一つの理想である．今後の論理合成技術の進歩を考慮するならば，記憶をもつデータ型と記憶をもたないデータ型の区別がないことは，ハードウェア記述言語の将来的な一つの望ましい姿であると考えられる．

（5） 文 法

構造化プログラミングの考え方に基づき，ある一つのまとまったハードウェアの動作記述は，そのハードウェアの入出力に関する情報を記述するエンティティと動作に関する情報を記述するアーキテクチャにわけて行う．これにより設計資産の再利用または管理を容易にすることを図っている．この機能と抽象データ型の機能を使って，階層的にモジュール化した記述が可能なように言語設計はなされているが，タイミングに関する抽象化の機能が必ずしも十分とは言えない．

また，並列的な動作と逐次的な動作を区別して記述できる．たとえば，言語要素の := は，信号の左辺への代入は逐次的に行われることを意味し，<= は，信号の左辺への代入が並列に行われることを意味する．

アルゴリズム／アーキテクチャレベルからゲート／スイッチレベルまで文法は同一であり，動作についての記述であるか構造についての記述であるかを区別することはできない．

2.4 Verilog HDL[11),12)]

（1） スコープ

Verilog HDL は Verilog シミュレータのモデリング用言語として設計された特定の CAD ツールに依存したハードウェア記述言語である．実際の設計現場で用いられている実用的な設計手法に対応しており，設計者は，シミュレーションアルゴリズムを考慮してシミュレーションモデルの制御の流れを記述する．Verilog HDL のカバーする範

囲は，アーキテクチャ／アルゴリズムレベルからゲートレベル，スイッチレベルまでである．

（2） 意 味 定 義

言語仕様書中に意味定義はないが，Verilog シミュレータにより，暗黙ではあるが完璧に定義されている．しかし，Verilog シミュレータのアルゴリズムの詳細が完全に公開されているわけではなく，意味定義が公開されているとは言えない．スイッチレベルをカバーするため，信号値には「ストレングス」の概念が導入されている．

（3） タイミング

信号変化のタイミングは Verilog シミュレータの悲観的なシミュレーションアルゴリズムで定義され，ある時刻での信号値が一意に定まらない場合，不定値 X がとられる．

（4） デ ー タ 型

データ型は固定であり，記憶をもつデータ型（レジスタ）と記憶をもたないデータ型（ネット）の二つが言語仕様書で与えられている．

（5） 文　　法

begin と end にはさまれた記述は，ハードウェアの動作が逐次的であることを意味し，fork と join にはさまれた記述はハードウェアの動作が非決定性を含む並列動作であることを意味する．

アルゴリズム／アーキテクチャレベルからレジスタトランスファレベルまでの動作を表す記述と，回路の構造つまりゲート／スイッチレベルの素子の接続関係を表す記述では文法が明確に異なり，動作についての記述であるか構造についての記述であるかを陽に区別して記述できる．

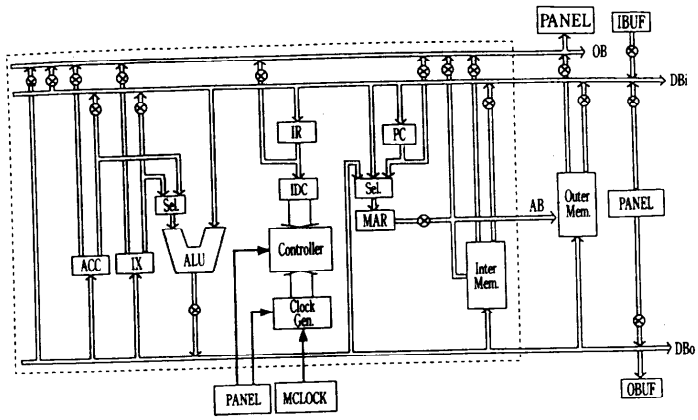
2.5 SFL[13),14)]

（1） ス コ ー プ

SFL (Structured Function Description Language) は，論理合成系 PARTHENON の入力言語として設計された特定の CAD ツールに依存したハードウェア記述言語である．特に，論理合成系を前提に設計された点に大きな特長がある．記述の対象を完全同期回路に限っている．SFL がカバーする範囲は，レジスタトランスファレベルからロジックレベルまでである．

（2） 意 味 定 義

言語仕様書中には明確な意味定義はないが，完全同期回路の設計手法に熟知した設計者にとってなじみやすい単純で明快な意味論が採用されている．



ACC Accumulator
IX Index Register
Sel. Selector
ALU Aithmetic Logic Unit
PC Program Counter
IR Instruction Register
IDC Instruction Decoder
MAR Memory Address Register

Clock Gen. Clock Generator
Inter Mem. Internal Memory
IBUF Input Buffer
OBUF Output Buffer
Outer Mem. Outer Memory
MCLOCK Master Clock

OB Observer Bus
AB Address Bus

DBi Data Bus for Input
DBo Data Bus for Output

⊗ 3 State Buffer

図-1 KUE-CHIP のブロック図

（3） タイミング

信号変化のタイミングは，記述中に暗黙に存在するクロック信号のエッジに限られる．このため，基本的にすべての動作は決定的となる．すなわち，設計者が制御できない非決定な動作はすべて次のクロック信号のエッジまでの一周期の間に終了することをSFLは仮定しており，設計者は決定性の考え方に基づいて設計を行うことができる．

（4） データ型

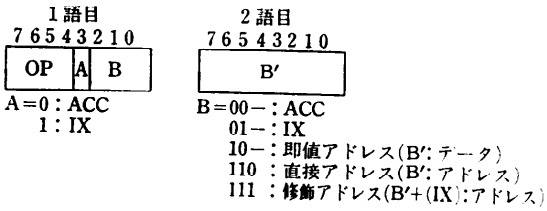
データ型としては，記憶をもつデータ型（レジスタとメモリ）および記憶をもたないデータ型（端子）が言語仕様書で与えられている．SFLではこれらのデータ型に，データ系であるかまたは制御系であるかを示す属性を定義できる．データ系と制御系ではとりうる信号値が異なり，データ系のとりうる信号値は 0，1 または unknown であり，制御系のとりうる信号値は0または1となっている．

（5） 文　法

ステージという状態遷移機械を表現する要素を

表-1 KUE-CHIP 命令形式と命令セット

命令形式



A=0：ACC
　1：IX

B=00－：ACC
　01－：IX
　10－：即値アドレス(B′:データ)
　110　：直接アドレス(B′:アドレス)
　111　：修飾アドレス(B′+(IX):アドレス)

命令セット

略記号	命令コード（1語目）	B′	命令機能の概略	
H	1 1 1 1 1 1 1 1	×	Halt	停止
NOP	1 1 1 1 0 0 0 0	×	No OPeration	
IN	1 1 1 0 1 - - -	×	INput	(IBUF)→ACC，IBUF_FLG_CLR=0
OUT	1 1 1 0 0 - - -	×	OUTput	(ACC)→OBUF，OBUF_WE=0
			Shift	
SRA	1 0 0 0 A 0 0 1	×	Right Arith.	(A)→シフト→A
SRL	1 0 0 1 A 0 0 1	×	Right Logical	(A)→シフト→A
SLL	1 0 1 0 A 0 0 1	×	Left Logical	(A)→シフト→A
RL	1 0 1 1 A 0 0 1	×	Rotate Left	(A)→シフト→A
IS	0 0 0 1 A B	○	Inverse Sub.	(B)－(A)→A
S	0 0 1 0 A B	○	Subtract	(A)－(B)→A
A	0 0 1 1 A B	○	Add	(A)＋(B)→A
EOR	0 1 0 0 A B	○	Exclusive OR	(A)＋(B)→A
OR	0 1 0 1 A B	○	OR	(A)　(B)→A
AND	0 1 1 0 A B	○	AND	(A)　(B)→A
L	0 0 0 0 A B	○	Load	(B)→A
ST	0 1 1 1 A B	◎	STore	(A)→B
	1 1 0 0		Branch	分岐条件　条件が成立すれば
BA	1 1 1 1	◎	always	常に成立　　B′→PC
BNP	1 1 1 0	◎	on not positive	演算結果　≦0
BNN	1 1 0 1	◎	on not negative	≧0
BZ	1 1 0 0	◎	on zero	＝0
BNZ	1 0 1 1	◎	on not zero	≠0
BN	1 0 1 0	◎	on negative	＜0
BP	1 0 0 1	◎	on positive	＞0
BOF	1 0 0 0	◎	on overflow	桁あふれ
BNO	0 0 0 1	◎	no output	IBUF_FLG_IN=1
BNI	0 0 0 0	◎	no input	OBUF_FLG_IN=0

－：don't care，×：不用，○：不用 or 必要，◎：必要

備え，完全同期式回路設計に慣れ親しんだ設計者の記述の労力を削減する工夫が行われている．

SFLはゲート／スイッチレベルをカバーしていないため，回路の構造を記述する文法要素は備えていない．

3. ベンチマーク記述に用いた回路について

ベンチマーク記述の対象に用いたKUE-CHIP (Kyoto University Education CHIP)[15]は，大学などにおける計算機ハードウェア教育の教材として開発されたシングルチップのマイクロコンピュータである．**図-1**のブロック図のうち点線内が集積化されている部分であり，ベンチマーク記述の記述対象である．4本のバス (DBi, DBo, AB, OB) の幅はすべて8ビットである．メモリ空間は256ワードであり，1ワードは8ビットである．KUE-CHIPは256バイトのSRAMを内蔵しており，チップ外部のメモリを使用することも可能である．

KUE-CHIPの命令形式と命令セットを**表-1**に示す．命令は1アドレス形式で，主にレジスタ演算用の1語命令とレジスタとメモリ間の演算用の2語命令がある．2語命令でのアドレッシングモードは3種類あり，即値アドレス，直接アドレス，修飾アドレスが指定できる．

KUE-CHIPの各命令の実行フェーズを**表-2**に示す．P0, P1が命令取り出しフェーズであり，P2, P3, P4, P5が命令の実行フェーズである．1命令の実行に必要なクロックは6フェーズが最長である．

4. ベンチマーク記述の比較

言語の比較のため，UDL/I, VHDL, Verilog HDL, SFLによるKUE-CHIPの動作記述をベンチマークとして作成した．紙面の都合上，各記述の命令取り出しフェーズ：P0, P1に対応した箇所の記述を付録に示す．

表-2 KUE-CHIPの命令実行フェーズ

命令＼フェーズ		P0	P1	P2	P3	P4	P5
H		(PC)→MAR PC++	(Mem)→IR	HALT			
NOP				NOP			
IN				(IBUF)→ACC	FLAG CLEAR		
OUT				(ACC)→OBUF	STROBE OUT		
SHIFT				SHIFT	STATUS SET		
IS	ACC, IX			(α)→ / (α′)→ ALU→α	STATUS SET		
SA	即値			(PC)→MAR PC++	(α) / (Mem)→ ALU→α	STATUS SET	
EOR	直接				(Mem)→MAR	(α)→ / (Mem)→ ALU→α	STATUS SET
OR	修飾				(IX) / (Mem)→ ALU→MAR		
AND							
L	ACC, IX			(α)→(α′)			
	即値			(PC)→MAR PC++	(Mem)→(α)		
	直接				(Mem)→MAR	(Mem)→α	
	修飾				(IX) / (Mem)→ ALU→MAR		
ST	直接			(PC)→MAR PC++	(Mem)→MAR	(α)→Mem	
	修飾				(IX) / (Mem)→ ALU→MAR		
BRANCH				(PC)→MAR PC++	STATUS CHECK (Mem)→PC	(条件成立)	

α { ACC / IX }

4.1 記述の統計的な情報について

UDL/I 記述は，京都高度技術研究所，京都大学，松下電器産業の共同作業で表-1，表-2 の設計仕様に基づき記述した．記述のステップ数は 487（；ごとに 1 ステップと換算），単語数は 2733，文字数は 17878 であった．シミュレーションは，(社) 日本電子工業振興協会の UDL/I 処理系によって行った．

VHDL 記述は，サウスカロライナ大学で UDL/I 記述を人手で VHDL に変換することで作成した．ステップ数は 1049（；ごとに 1 ステップと換算），単語数は 8676，文字数は 67812 であった．シミュレーションは GenRad 社の HILO シミュレータを用い，テストベクタも VHDL により記述している[10]．

Verilog HDL 記述は，三菱電機において，設計仕様に基づき作成した．ステップ数は 388（；ごとに 1 ステップと換算），単語数は 2253，文字数は 31932 であった．シミュレーションは Cadence 社の Verilog シミュレータで行った．さらに，Synopsys 社の Design Compiler によって論理合成を行った（論理合成用には記述を新しく作り直した）．合成結果は，1520 ゲートで，動作可能最高周波数は 10 MHz (0.8 μm CMOS ゲートアレイ) と評価された．合成時間は SUN 4 10 MIPS で約 6 時間であった．

SFL 記述は，NTT において設計仕様と実際のチップ動作の観測結果に基づき記述した．ステップ数は 396（；ごとに 1 ステップと換算），単語数は 2966，文字数は 25810 であった．シミュレーションは，PARTHENON のシミュレータを用い，さらに PARTHENON による合成も行った．論理合成結果は 1769 ゲート（メモリを除く）で，合成時間は Sparc Station-2 で 866 秒であった．VHDL による記述のステップ数が他の三つの記述に比べて大きい原因は，メモリのモデルの記述にステップ数で 242，単語数で 767，文字数で 11119 必要であったこと，および算術論理演算の記述にステップ数で 158，単語数で 1235，文字数で 10411 が必要であったことが大きな要因であった．

参考のため，以下では各記述の作成に要した工数を紹介する．UDL/I 記述は，UDL/I の言語仕様に詳しくかつ実際にチップの設計を行った設計者が担当した結果，工数は 8 人日を要した．VHDL 記述は，VHDL の言語仕様には詳しいが UDL/I の言語仕様をまったく知らない修士課程の学生が UDL/I 記述を人手で変換した結果，工数は，10 人日を要した．Verilog HDL 記述は，Verilog HDL の言語仕様に詳しい研究者が設計仕様から作成した結果，工数は 7 人日であった．SFL 記述は，SFL の言語仕様の設計者が，設計仕様と実際のチップの動作の観察から作成した結果，工数は 3 人日を要した．

4.2 意味の相違について

付録に載せた各ハードウェア記述言語による記述の中で，クロックフェーズ P0 中での動作の記述を例に取って，各言語の細かな違いをみてみよう．クロックフェーズ P0 は，プログラムカウンタ pc の値をメモリアドレスレジスタ mar に転送し，pc はその値を 1 増やす動作をする．

UDL/I では，クロック信号 clk が定義され，レジスタへの転送が，clk のエッジに同期して同時に行われることが仮定される．よって，

```
4  mar := pc      ;
5  pc  := INC (pc);
```

の二つの記述の順番を入れ換えても記述の意味は同じである．

VHDL では，

```
4  MOV_INC (pc, mar);
```

で procedure MOV_INC が起動される．procedure MOV_INC 内の下記の文は，（シミュレーション時間での）現時刻に ARG 2 (mar) に ARG 1 (pc) の値が代入される．

```
57  ARG 2 <= ARG 1;
```

一方，以下の文は Δ 時刻ごとに逐次的に実行され，変数 result には ARG 1 (pc) の 1 増やした値が代入される．

```
58  result(0) := ARG 1(0) xor '1';
59   carry    := ARG 1(0) and '1';
60   for i in 1 to 7 loop
61    result(i) := ARG 1(i) xor carry;
62    carry     := ARG 1(i) and carry;
63  end loop;
```

そして以下の文で ARG 1 (pc) の値が 1 増やされる．

```
64  ARG 1 <= result;
```

この場合，mar への pc の値の代入と pc の値を 1 増やす動作のハードウェアへのインプリメント

すなわち論理合成の結果は，UDL/I のように必ずしもクロックに同期して同時に実行する必要はなく，非同期回路での実現も考えうる．

Verilog HDL では，

```
42  @ (posedge CLK) begin
43      MAR=PC;
44      PC  =PC+1'b1;
45  end
```

と書かれている．この場合，クロックのエッジに同期して MAR に対する PC の値の代入および PC の値を1増やす動作がおこる．しかし，記述の順番を以下のようにした場合

```
44  PC  =PC+1'b1;
43  MAR=PC;
```

PC の値を1増した後の値が MAR に代入されることを意味し，元の記述の意味と異なってくる．

SFL 記述では，クロック信号のエッジに同期してデータ転送が行われることが暗黙に仮定されている．

```
20  mar  := pc;
21  pc   := inc_pc. up(pc). out;
```

記述の順番を入れ換えても意味は同じである．クロック信号が記述中に明記されずに暗黙に仮定されることが，UDL/I と異なる．

上記の例から，UDL/I と SFL の意味定義は，同期式設計手法を用いているハードウェアの設計者の直観と親和性がよく，VHDL と Verilog は逐次型のプログラミング言語に慣れたソフトウェアの設計者がハードウェアの設計を行う際に親しみやすいと考えられる．

4.3 データ型について

各言語による記述中でデータ型の違いが最も顕著に現れるのはメモリの定義である．UDL/I では，言語仕様で用意されているデータ型 RAM としてメモリを定義することができる．

```
RAM: imem<0: 255, 7: 0>
```

VHDL では，特にメモリ用の構文要素が用意されていないので，下記のようにメモリの動作を表すモデルを記述する必要がある．

```
entity ram is
  port (clk: in bit;
    rw: in bit;
    rw_internal: in bit;
    mar: in bit_vector
  (7 downto 0);
    data_in: in bit_vector
  (7 downto 0);
    data_out: out bit_vector
  (7 downto 0);
    err: out bit;
end ram;
architecture behavioral of ram is
  signal mem0, mem1, mem2:
bit_vector (7 downto 0);
                ...
  begin
  READ_WRITE:
process (mar, rw, clk, rw_internal)
  begin
  if(rw '0' orrw_internal='1') then
    case mar is
      when X "0 0" =>
        mem0  <= data_in;
        err   <= '0';
      when X "0 1" =>
        mem1  <= data_in;
        err   <= '0';
            ...
   elsif (rw='1') then
    case mar is
      when X "0 0" =>
        data_out <= mem0;
        err      <= '0';
      when X "0 1" =>
        data_out <= mem1;
        err      <= '0';
            ...
    end case;
  end if;
  end process READ_WRITE;
end behavioral;
```

Verilog HDL では，メモリは8ビット幅のレジスタが256並んだレジスタファイルとしてメモリを定義する．

```
reg [7: 0] IM [0: 255];
```

SFL では，メモリを表すデータ型 mem を用いてメモリを定義できるが，メモリは機能回路（合

成の対象としないモジュール）としてのみ定義可能であり，動作記述中で直接メモリを意味するデータ型を定義することはできない．

```
circuit_class memory_256_8 {
  input    adrs<8>;
  input    in<8>;
  output   out<8>;
  mem      cell [256]<8>;
  instrin  read;
  instrin  write;
  instruct_arg read (adrs);
  instruct_arg write (adrs, in);
  instruct readout=cell [adrs];
  instruct write cell [adrs] := in;
}
```

メモリに関して，UDL/I は動作の意味定義を言語仕様の中で明記している．言語仕様の意味定義と動作が異なるメモリを用いた実回路のシミュレーションを目的とした記述や，言語で用意していないような機能をもつ機能メモリなどの記述は困難となる．しかし，メモリの動作が明確に定義されているので，論理合成系が言語の意味定義に適合したメモリを合成することは，比較的容易である．

VHDL は，メモリの動作モデルをユーザが記述することができ，さまざまな仕様のメモリを含んだ回路の記述を行うことができる．その反面 VHDL の論理合成系はユーザが記述したさまざまなメモリの動作モデルを満たすメモリを合成しなければならず，論理合成系の負荷は重たくなる．

Verilog HDL はレジスタファイルとしてメモリを宣言するので，論理合成系がメモリとして実現するかレジスタファイルとして実現するかは言語記述で指定することができない．

SFL は，メモリを合成の直接の対象としておらず，利用するメモリの仕様にあわせて機能回路として記述する．UDL/I と同じような特別な文法を用意しているので，メモリのもちうる機能には制限がでてくる．

4.4 状態遷移機械の記述方法

システムの制御部などの設計に広く用いられる同期式の状態遷移機械の記述方法について比較を行ってみる．ベンチマークのプロセッサでも制御部が状態遷移機械として記述されている．

UDL/I では，状態遷移機械を記述することを目的として用意されているステート文と状態遷移文を用いて下記のようにクロックフェーズ P0 を表すステート p0 からクロックフェーズ P1 を表すステート p1 への遷移を平易に記述できる．この記述では p0 から p1 への遷移はクロック信号のエッジに同期して行われる．

```
3  p0: WAIT (wcond):
6      -> p1;
```

VHDL では，状態遷移機械を記述する要素は文法では用意されていない．この記述中では，状態遷移機械の状態を表すデータ型 state_2 とデータ型 state_2 をもつシグナル current_state_2 を下記のように定義している．

```
type state_2_is (
 p0,
 p1,
 p2_halt, p2_nop, p2_in, p2_out,
 p2_shift, p2_alu, p2_ld,
 p2_st, p2_brn,
 p3_in, p3_out, p3_shift,
 p3_alu, p3_ld, p3_st, p3_brn,
 p4_alu, p4_ld, p4_st,
 p5_alu);
signal: current_state_2:
state_2;
```

付録の VHDL 記述中では，データ型 state_2 をもつシグナル current_state_2 を下記のように参照または信号を代入することで状態遷移を記述している．p0 から p1 への状態の遷移は同期，非同期の両方の場合が考えられる．

```
1  case CURRENT_STATE_2 is
2    when p0 =>
6      CURRENT_STATE_2 <= p1;
```

Verilog HDL では，状態遷移機械を記述する要素は文法では用意されていない．ここではクロックフェーズ P0, P1 に行われるハードウェアの動作を記述するタスク P0, P1 を定義して状態遷移を表現している．この記述はタスク P0 の終了後タスク P1 にハードウェアの動作が移ることを意味しており，P0 から P1 への状態の遷移は同期，非同期の両方の場合が考えられる．

```
4  forever begin
```

```
5    P0;
6    P1;
```

SFLではステージの概念を用いて状態遷移機械を記述している．この場合の状態の遷移は暗黙に存在するクロック信号のエッジに同期して行われることを意味する．

```
1  stage exec {
2      state_name phase 0;
3      state_name phase 1;
4      state_name phase 2;
5      state_name phase 3;
6      state_name phase 4;
7      state_name phase 5;
8      first_state  phase 0;
15     state phase 0 par {
22       goto phase 1;
24   }
```

UDL/I と SFL は，その意味定義と文法により非同期の状態遷移機械を記述することはできないが，同期式の状態遷移機械を少ない記述量で書くことができる．一方，VHDL と Verilog HDL による記述では，回路の実現方法を同期方式とするか非同期方式とするかは指定しない記述が可能で，実現方法を論理合成系に任せることができる．ただし，この場合，動作が決定的であることを仮定しており，このような記述どおりに動作する非同期回路を合成することは必ずしも容易なことではない点には注意する必要がある．

4.5 記述からの論理合成

記述からの論理合成は，合成系の存在する Verilog HDL と SFL について行った (VHDL についても合成実験を行っているが最終的な結果は得られていない)．

SFL の合成では，基本的にすべての記述がそのまま合成されている（メモリだけは機能回路として定義して合成の対象から外した)．SFL およびその処理系 (PARTHENON) の設計思想から，検証のためのシミュレーションと論理合成は完全に意味論を共有しており，問題はなかった．

一方，Verilog HDL の場合には，シミュレーションのみを意識して記述した記述はそのまま実験に用いた論理合成系に受け付けられず，論理合成向きに記述の大幅な書き直しを行う必要があった．また，論理合成された結果の回路が，組み合わせ回路を作るつもりの記述から順序回路が合成されるなど，設計者の意図とは大きく異なる場合もあった．これは，Verilog HDL のシミュレータ向きの意味論を十分に論理合成系が反映していないことに起因していると考えられる．

一般に，論理合成を意識して言語が設計された SFL や UDL/I は，記述から合成される回路への見通しがよい．一方，シミュレーション用の言語として開発された VHDL や Verilog HDL は，合成結果を制御するための情報が書きにくい．将来，論理合成技術が成熟し，実現する回路を設計者がまったく意識しなくてよいようになれば，言語は機能だけを記述する能力さえあればよいが，現状のように論理合成に対する制約により設計者が合成される回路の細部までを意識する必要がある間は，合成向きの記述法や記述への制限が必要である．

5. あとがき

標準を目指す4つのハードウェア記述言語：UDL/I, VHDL, Verilog HDL, SFL について，それぞれを特徴づける特質に焦点をあてた比較および実際にある一つの回路をそれぞれの言語で記述した例を用いての具体的な比較を行った．各言語は個々に異なった特徴をもち，CAD ツールのサポートの違いだけでなく，ユーザの設計手法との親和性などから，ユーザがうける利便もまた異なる．

異なるハードウェア記述言語間で CAD ツールを共有し，ユーザの趣味にあわせた言語を使えるようにするためには各言語間に相互互換性が保証されなければならない．このためには，言語の意味定義に関する共通の土台が必要になる．UDL/I の中で提案された NES モデル[4]や Kahn らの提案するインフォメーションモデル[16]などによる意味の統一的記述法の研究開発が今後重要な課題である．今回作成した UDL/I, VHDL, Verilog HDL, SFL による KUE-CHIP の動作記述については情報処理学会設計自動化研究会 CAD モデル分科会（委員長：小野寺秀俊京都大学工学部助教授）から HDL 比較のベンチマークデータとして一般に公開されている．今後の言語比較，論理合成などの研究資料として活用していただければ幸いである．

謝辞 UDL/I による KUE-CHIP の動作記述の作成については，京都大学工学部電気系教室（当時）の則安学氏および松下電器産業の村岡道明氏ならびに高井裕司氏にご協力いただいた．また VHDL 記述の公開にあたっては，リコーの岡善治氏にご協力いただいた．Verilog HDL による記述の作成については，三菱電機の青木洋氏にご協力いただいた．ここに感謝の意を表す．

参 考 文 献

1) 安浦，神原：ハードウェア記述言語の比較，電子情報通信学会技術報告，VLD 92-46 (1992).
2) UDL/I 標準化委員会：UDL/I 言語仕様第 1.0m 版，日本電子工業振興協会 (1992).
3) 唐津，星野，石浦，安浦：論理合成時代のハードウェア記述言語：UDL/I，電子情報通信学会論文誌，Vol. J74-A, No. 2, pp. 170-178 (1991).
4) Ishiura, N., Yasuura, H. and Yajima, S.: NES: The Behavioral Model for the Formal Semantics of a Hardware Design language UDL/I, Proc. 27th DAC (1990).
5) Yasuura, H. and Ishiura, N.: Formal Semantics of UDL/I and Its Application to CAD/DA Tools, Proc. of ICCD '90 (1990).
6) IEEE: VHDL ハードウェア記述言語，日本規格協会 (1991).
7) ジェームス R アームストロング：VHDL デザインテクニック，電波新聞社 (1990).
8) R リップセット他：VHDL 言語記述によるハードウェア設計へのアプローチ，マグロウヒル出版 (1990).
9) Coelho, D. R.: THE VHDL HANDBOOK, Kluwer Academic Publishers (1989).
10) Kukkal, P., Kobayashi, H., and Kanbara, H.: VHDL and UDL/I—Feature Description and Analysis, Proc. SASIMI '92 (1992).
11) Open Verilog International: Verilog Hardware Description Language Reference Manual (1991).
12) Thomas, D. E. and Moorby, P.: The Verilog Hardware Description Language, Kluwer Academic Publishers (1989).
13) NTT: SFL 言語概説書，NTT (1989).
14) 中村，小栗，野村：RTL 動作記述言語，電子情報通信学会論文誌，Vol. J72-A, No. 10, pp. 1570-1593 (1989).
15) 神原，安浦：計算機教育用コンピュータの開発とその応用，情報処理，Vol. 33, No. 2, pp. 118-127 (1991).
16) Lau, R. and Kahn, H.: Information Modelling of EDIF using EXPRESS, 6th Annual USA EDIF Conf. pp. 64-73 (1990).

(平成4年7月6日受付)

付録 各言語によるベンチマーク記述（抄）

UDL/I による KUE-CHIP の動作記述(クロックフェーズ p0, p1)

```
1  AUTOMATON:rtl:^.rst:^.clk:
                              RISE(.clk);
2
3  p0: WAIT(wcond):
4        mar:=pc;
5        pc  :=INC(pc);
6        ->p1;
7
8  p1: WAIT(wcond):
9        CASEOF
10         #inmem     ir:=imem<.ab>;
11         #exmem     ir:=.dbi;
12       END_CASEOF;
13       CASEOF
14         #exmem.mem_re:=1b0;
15       OFFSTATE 1 END_CASEOF;
16       CASEOF
17         #halt          -> p2_halt;
18         #nop           -> p2_nop;
19         #in            -> p2_in;
20         #out           -> p2_out;
21         #shift         -> p2_shift;
22         #alu 1         -> p4_alu;
23         #alu 2, alu 3  -> p2_alu;
24         #ld 1          -> p4_ld;
25         #ld 2, ld 3    -> p2_ld;
26         #st            -> p2_st;
27         #brn           -> p2_bm;
28       END_CASEOF;
```

VHDL による KUE-CHIP の動作記述 (クロックフェーズ p0, p1)

```
1  case CURRENT_STATE_2 is
2    when p0=>
3      if(wcond='1')then
4        MOV_INC(pc, mar);
5        rw_internal<='0';
6        CURRENT_STATE_2<=p1;
7      elsif(wcond='0')then
8        NULL;
9      end if;
```

```
10    when p1=>
11      if(wcond='1')then
12      if(halt='1')then
13        CURRENT_STATE_2<=p2_halt;
14      elsif(nop='1')then
15        CURRENT_STATE_2<=p2_nop;
16      elsif(in_op='1')then
17        CURRENT_STATE_2<=p2_in;
18      elsif(out_op='1')then
19        CURRENT_STATE_2<=p2_out;
20      elsif(shift='1')then
21        CURRENT_STATE_2<=p2_shift;
22      elsif(alu1='1')then
23        CURRENT_STATE_2<=p4_alu;
24      elsif((alu2='1')or(alu3='1'))then
25        CURRENT_STATE_2<=p2_alu;
26      elsif(ld1='1')then
27        CURRENT_STATE_2<=p4_ld;
28      elsif((ld2='1')or(ld3='1'))then
29        CURRENT_STATE_2<=p2_ld;
30      elsif(st='1')then
31        CURRENT_STATE_2<=p2_st;
32      elsif(brn='1')then
33        CURRENT_STATE_2<=p2_brn;
34      end if;
35    elsif(wcond='0')then
36      NULL;
37      end if;
38  end case;
39
40  FETCH: process(clk)
41  begin
42   if(clk'event and clk='0')then
43      if current_state_2=p1 then
44       if(wcond='1')then
45          ir  <=data_out;
46       elsif(wcond='0')then
47          NULL;
48       end if;
49      end if;
50    end if;
51   end process FETCH;
52
53   procedure MOV_INC(signal ARG1:
                        inout bit_vector;
               signal ARG2: outbit_vector) is
54     variable carry: bit:='0';
55   variable result: bit_vector(7 downto 0);
56   begin
57       ARG2<=ARG1;
58       result(0):=ARG1(0)xor '1';
59       carry  :=ARG1(0) and '1';
60       for i in 1 to 7 loop
61       result(i):=ARG1(i)xor carry;
62       carry:=ARG1(i) and carry;
63       end loop;
64       ARG1<=result;
65   end;
```

Verilog HDLによるKUE-CHIPの動作記述(クロックフェーズ p0, p1)

```
 1  initial
 2   begin
 3    H;
 4    foreverbegin
 5     P0;
 6     P1;
 7     //P2
 8     begin
 9      P=6'b000100;
10      casex(IR)
11       8'b11111111: H;     //Halt
12       8'b11110000: NOP;  //No Operation
13       8'b11101xxx: IN;    //Input
14       8'b11100xxx: OUT;  //Output
15       8'b10xxx001: Shift; //Shift
16       8'b0001xxxx,       //Inverse Subtract
17       8'b0010xxxx,       //Subtract
18       8'b0011xxxx,       //Add
19       8'b0100xxxx,       //Exclusive Or
20       8'b0101xxxx,       //Or
21       8'b0110xxxx: P2_ALU;  //And
22       8'b0000xxxx: L;  //Load
23       8'b0111xxxx: ST; //Store
24       8'b11001111,       //Branch Always
25       8'b11001110,
                    //Branch on Not Positive
26       8'b11001101,
                    //Branch on Not Negative
```

```
27        8'b11001100,  //Branch on Zero
28        8'b11001011,  //Branch on Not Zero
29        8'b11001010,  //Branch on Negative
30        8'b11001001,  //Branch on Positive
31        8'b11001000,  //Branch on Overflow
32        8'b11000001,  //Branch No Output
33        8'b11000000: P2_B;
                        //Branch No Input
34      end case
35    end
36    if(halt_request==yes)H;
37      end//forever loop
38    end
39  task P0;
40   begin
41        P=6'b000001;
42        @(posedge CLK)begin
43         MAR=PC;
44         PC=PC+1'b1;
45        end
46        @(negedge CLK)
47         if(SP_stop==yes)H;
48   end
49  end task
50  task P1;
51   begin
52        P=6'b000010;
53        if(MEM_SEL==internal_memory)
54         dbi_sel=sel_IM;
55        else
56         MEM_RE=1'b0;
57        @(posedge CLK)
58         IR=DBi;
59    @(negedge CLK) begin
60        dbi_sel=sel_NONE;
61        MEM_RE=1'b1;
62        if(SP_stop==yes)H;
63       end
64   end
65  end task
```

SFL による KUE-CHIP の動作記述（クロックフェーズ p0, p1）

```
1  stage exec {
2     state_name phase0;
3     state_name phase1;
4     state_name phase2;
5     state_name phase3;
6     state_name phase4;
7     state_name phase5;
8     first_state phase0;
9     par{
10     op();
11     any{
12     stop_immediately: finish;
13    }
14   }
15   state phase0 par{
16    pt0();
17    any{
18     stop_instruction: finish;
19     else: par{
20          mar:=pc;
21          pc:=inc_pc.up(pc).out;
22          goto phase1;
23    }
24   }
25  }
26  state phase1 par{
27   pt1();
28   ir:=memory_read().me m_r_data;
29   goto phase2;
30  }
31 }
```

神原 弘之

1963 年生. 1987 年京都大学工学部電子工学科卒業. 1989 年同大学院修士課程電子工学専攻修了. 同年(財)京都高度技術研究所入所. 論理設計, VLSI 用 CAD の研究に従事. 電子情報通信学会会員.

安浦 寛人 (正会員)

1953 年生. 1976 年京都大学工学部情報工学科卒業. 1978 年同大学院修士課程情報工学専攻修了. 1980 年より京都大学工学部助手. 1986 年より同電子工学科助教授. 1991 年より九州大学大学院総合理工学研究科教授. 並列アルゴリズム, 並列計算機アーキテクチャ, 論理設計, VLSI 用 CAD の研究に従事. 1987 年度電子情報通信学会, 1990 年度情報処理学会論文賞受賞. IEEE, ACM, 電子情報通信学会, ソフトウェア科学会, LA シンポジウム, EATCS 各会員.

Pankaj Kukkal

1968年生. 1989年インド Kurukshetru 大学卒業. 1991年米国サウスカロライナ大学工学部電気コンピュータ工学科修士課程修了. 現在同博士課程在学中. 図形と言語を用いた ASIC の設計データ入力方法の研究に従事. IEEE 会員.

Hideaki Kobayashi

1950 年生. 1973 年早稲田大学理工学部電子通信学科卒業. 1979 年同大学院理工学研究科電気工学専攻, 博士課程修了. 工学博士. 1980 年より米国サウスカロライナ大学工学部電気コンピュータ工学科客員助教授. 1981 年同学科助教授. 1985 年同学科準教授. 1992 年夏より早稲田大学理工学部情報科学研究教育センター訪問学者 (サウスカロライナ大学より, サバティカル休暇). VLSI 設計方法論, 高位 CAD の研究に従事. IEEE, Eta, Kappa, Nu 各会員.

野地 保 (正会員)

1947 年生. 1970 年長崎大学工学部電気工学科卒業. 同年三菱電機(株)計算機製作所入社. 同情報電子研究所を経て, 現在同 CL 研(情)所属. 1992 年より長崎大学大学院後期博士課程在学中. コンピュータ・アーキテクチャ, トップダウン設計手法, 論理合成などの研究に従事. OVI 日本 WG 委員, 電子情報通信学会会員.

小栗 清 (正会員)

昭和 49 年九州大学理学部物理学科卒業. 昭和 51 年同大大学院修士課程修了. 同年日本電信電話公社入社. 同社電気通信研究所において, DIPS アーキテクチャ, 論理装置などの研究, 設計開発を経て, 並列処理アーキテクチャ設計技術の研究に従事. 現在, NTT コミュニケーション科学研究所主幹研究員. 平成 4 年 4 月より電気通信大学大学院情報システム学研究科客員助教授. 1987 年第 2 回元岡賞受賞. 平成元年度情報処理学会論文賞受賞, 平成 3 年度大河内記念技術賞受賞. 電子情報通信学会会員.