設計自動化　63－5
（1992. 7.17）

# ハードウェア記述言語の比較

安浦 寛人
九州大学大学院総合理工学研究科
春日市春日公園６－１

神原 弘之
京都高度技術研究所
京都市下京区中堂寺南町１７

あらまし　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。比較をより具体的に行なうために、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なった。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。

# A Comparison of Hardware Description Languages

Hiroto Yasuura　and　Hiroyuki Kanbara
Dept. of Information Systems, Kyushu Univ.　ASTEM RI
Kasuga-Koen, Kasuga 816 Japan　Simogyo-ku, Kyoto 600 Japan

Abstract　We disucuss factors which make differences among Hardware Design Languages (HDL's) and compare four HDL's, UDL/I, VHDL, Verilog HDL and SFL. The comparison is done from the viewpoins of target design levels and methodologies, semantics definitions, assumptions on timing issue, data types and syntax. We use a design of Kue-Chip, an 8-bit microprocessor, as a benchmark of the comparison. We show statistic data for the descriptions of the processor in these languages and discuss the differences of the languages by the descriptions.

## 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通して、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様々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長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かにすることを目指す。

## 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、階層的にいくつかの設計レベルに分けて行なわれる。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方から低い方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、トランジスターレベル（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ー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、実用されている。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伴う。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システムなどの処理系毎に、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ー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、このような手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において特にその明確な意味定義が困難な理由の一つに、信号変化のタイミングに関するの仮定の問題（以下ではタイミングと呼ぶ）が挙げられる。設計プロセスの違い、温度変化またはチップ毎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、このような仮定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に対して大きな阻害要因になる。ここにハードウェア記述言語の意味定義のむづ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としてデータ型の機能が挙げられる。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2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ーは、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ー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持たせることができる。

（5）文法（言語のシンタックス）

しばしば議論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。即ち、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。例えば、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を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難しくなる。

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なわれたため、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][4][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持たないデータ型（ターミナル）と記憶を持つデータ型（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。例えば、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。

## 2．3 VHDL[6]-[9]

（1）スコープ

VHDL(VHSIC Hardware Description Language)は、並行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ム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極めて広い。その他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。既に、いく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、ユーザーによる新しいデータ型を定義することが可能である。

（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。抽象データ型の機能とこの機能を使って、階層的にモジュール化した記述が可能な様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。例えば、言語要素の := は、信号の左辺への代入はシーケンシャル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。

## 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ている。

（4）データ型

データ型は固定であり、記憶を持つデータ型（レジスタ）と記憶を持たないデータ型（ネット）の2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シーケンシャル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。

## 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。

（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。即ち、設計者が規定できない非決定性は、同期信号の周期の中に埋め込んで決定性の世界をユーザーに提供する手法を取っている。

（4）データ型

データ型としては、記憶を持つデータ型（レジスタとメモリ）及び記憶を持たないデータ型（tmp）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。

## 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。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。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。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

UDL/I、VHDL、Verilog HDL、SFL による KUE-CHIP の動作記述の中で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。

## 4．1 記述の 統計的な情報について

UDL/I 記述は、 ASTEM、京都大学、松下電器の共同作業で KUE-CHIP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。948 （wc -l による値））であり、記述作成時間は8 人日であった（記述作業にチップ設計者も加わっている）。シミュレーションは、（社）日本電子工業振興協会の UDL/I 処理系によって行なった。

VHDL記述は、南カロライナ大学で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。2729 （wc -l による値））であり、記述作成時間は、10 人日であった。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、UDL/I 記述を人手で変換して作成した。ステップ数は388 （; ごとに1 ステップと換算。955 （wc -l による値））であり、作成時間は7 人日であった。シミュレーションはCadence 社の Verilog シミュレータで行なった。さらに、Synopsys 社の Design Compilerによって論理合成を行なった（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 （SUN4 10 MIPS で 約 6 時間）で、動作可能最高周波数は 10 MHz （0.8 $\mu$m CMOS ゲートアレイ）と評価された。

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。719 （wc -l による値））であり、作成時間 3 人日であった。シミュレーションは、Parthenonのシミュレータを用い、さらにParthenonによる合成も行なった。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 3 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 242 （; ごとに1 ステップと換算。545 （wc -l による値））ステップが必要であったこと、および算術論理演算の記述に 158 （; ごとに1 ステップと換算。418 （wc -l による値））ステップが必要であったことが大きな要因であった。

## 4．2 意味の相違について

付録に載せた各 HDL の記述の中で、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見てみよう。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 2 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。

一方、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ARG2（mar）に ARG1（pc）の値が代入される

```
57   ARG2 <= ARG1;
```

一方、以下の文は Δ 時刻ごとにシーケンシャルに実行され、変数result に はARG1(pc)の 1 インクリメントした値が代入される。

```
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;
```

そして以下の文で ARG1（pc）の値がインクリメントされる。

```
64   ARG1 <= result;
```

この場合、mar への pc の値の代入と pc の値のインクリメントを行うハードウェアへのインプリメント即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 の値のインクリメントがおこる。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 インクリメントした後の値が 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。

## 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わ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, mem3, mem4
                        : bit_vector(7 downto 0);
          ...
begin
READ_WRITE : process(mar, rw, clk, rw_internal)
begin
    if (rw = '0' or rw_internal = '1') then
        case mar is
            when X"00" =>
                mem0   <= data_in;
                err    <= '0';
            when X"01" =>
                mem1   <= data_in;
                err    <= '0';
        ...
    elsif (rw = '1') then
        case mar is
            when X"00" =>
                data_out <= mem0;
                err    <= '0';
            when X"01" =>
                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 read out = cell[adrs] ;
    instruct write cell[adrs] := in ;
}
```

## 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 を持つシグナル 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 を持つシグナル 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 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 ;
15         state phase0 par {
22                 goto phase1 ;
24             }
```

## 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持ち、CAD ツールのサポートの違いだけでなく、ユーザーの設計手法との親和性などから、ユーザー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、ユーザー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ーメーションモデル[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る予定である。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、高井 裕司氏にご協力いただいた。VHDL による記述は サウスカロライナ大学の Pankaj Kukkal 氏、Verilog HDL による記述については三菱電機の野地 保氏、SFL による記述についてはNTTの小栗 清氏に作成していただいた。ここに感謝の意を表す。


## 参考文献


[1] Hiroto Yasuura : " Comparison of HDL's for VLSI Design", Proc. EDA Standard Forum Japan '92 (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] N. Ishiura, H. Yasuura and S.Yajima : "NES:The Behavioral Model for the Formal Semantics of a Hardware Design language UDL/I", Proc. 27th DAC(1990).

[5] H.Yasuura and N.Ishiura : "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] David R. Coelho : THE VHDL HANDBOOK, Kluwer Academic Publishers(1989).

[10] P. Kukkal, H. Kobayashi and H. Kanbara : VHDL and UDL/I - Feature Description and Analysis, Proc. SASIMI'92(1992).

[11] Open Verilog International: Verilog Hardware Description Language Reference Manual", (1991).

[12] Donald E. Thomas, Philip Moorby : 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、情報処理学会誌、Vol.33, No.2, pp.118-127(1991).

[16] R. Lau and H. Kahn: "Information Modelling of EDIF using EXPRESS", 6th Annual USA EDIF Conf. pp.64-73(1990).


## 付録

1. 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           #alu1       -> p4_alu  ;
23           #alu2, alu3  -> p2_alu  ;
24           #ld1        -> p4_ld   ;
25           #ld2, ld3    -> p2_ld   ;
26           #st         -> p2_st   ;
27           #brn        -> p2_brn  ;
28         END_CASEOF ;
```

2. VHDL による KUE-CHIP の動作記述

（クロックフェーズ p0, p1 ）

```
case CURRENT_STATE_2 is
    when p0 =>
        if (wcond = '1') then
            MOV_INC(pc, mar);
            rw_internal <= '0';
            CURRENT_STATE_2 <= p1;
        elsif (wcond = '0') then
            NULL;
        end if;
    when p1 =>
        if (wcond = '1') then
            if (halt = '1') then
                CURRENT_STATE_2 <= p2_halt;
            elsif (nop = '1') then
                CURRENT_STATE_2 <= p2_nop;
            elsif (in_op = '1') then
                CURRENT_STATE_2 <= p2_in;
            elsif (out_op = '1') then
                CURRENT_STATE_2 <= p2_out;
            elsif (shift = '1') then
                CURRENT_STATE_2 <= p2_shift;
            elsif (alu1 = '1') then
                CURRENT_STATE_2 <= p4_alu;
            elsif ((alu2 = '1') or (alu3 = '1')) then
                CURRENT_STATE_2 <= p2_alu;
            elsif (ld1 = '1') then
                CURRENT_STATE_2 <= p4_ld;
            elsif ((ld2 = '1') or (ld3 = '1')) then
                CURRENT_STATE_2 <= p2_ld;
            elsif (st = '1') then
                CURRENT_STATE_2 <= p2_st;
            elsif (brn = '1') then
                CURRENT_STATE_2 <= p2_brn;
            end if;
        elsif (wcond = '0') then
            NULL;
        end if;
end case;

FETCH : process (clk)
begin
    if(clk'event and clk = '0') then
        if current_state_2 = p1 then
            if (wcond = '1') then
                ir    <= data_out;
            elsif (wcond = '0') then
                NULL;
            end if;
        end if;
    end if;
end process FETCH;

procedure MOV_INC (signal ARG1 : inout bit_vector;
                   signal ARG2 : out bit_vector) is
variable carry  : bit := '0';
variable result : bit_vector(7 downto 0);
begin
ARG2 <= ARG1;
result(0) := ARG1(0) xor '1';
carry    := ARG1(0) and '1';
  for i in 1 to 7 loop
    result(i) := ARG1(i) xor carry;
    carry    := ARG1(i) and carry;
end loop;
ARG1 <= result;
end ;
```

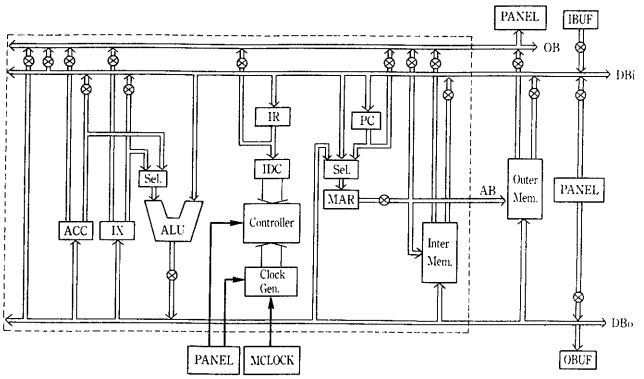
3. Verilog HDL による KUE-CHIPの動作記述

（クロックフェーズ p0, p1 ）

```
initial
 begin
     H;
    forever begin
      P0;
      P1;
      // P2
      begin
       P = 6'b000100;
       casex (IR)
          8'b11111111:  H;    // Halt
          8'b11110000:  NOP;  // No Operation
          8'b11101xxx:  IN;   // Input
          8'b11100xxx:  OUT;  // Output
          8'b10xxx001:  Shift; // Shift
          8'b0001xxxx,        // Inverse Subtract
          8'b0010xxxx,        // Subtract
          8'b0011xxxx,        // Add
          8'b0100xxxx,        // Exclusive Or
          8'b0101xxxx,        // Or
          8'b0110xxxx:  P2_ALU; // And
          8'b0000xxxx:  L;    // Load
          8'b0111xxxx:  ST;   // Store
          8'b11001111,        // Branch Always
          8'b11001110,        // Branch on Not Positive
          8'b11001101,        // Branch on Not Negative
          8'b11001100,        // Branch on Zero
          8'b11001011,        // Branch on Not Zero
          8'b11001010,        // Branch on Negative
          8'b11001001,        // Branch on Positive
          8'b11001000,        // Branch on Overflow
          8'b11000001,        // Branch No Output
          8'b11000000:  P2_B; // Branch No Input
       endcase
      end
      if (halt_request == `yes) H;
    end // forever loop
 end
task  P0;
     begin
         P  = 6'b000001;
        @(posedge CLK) begin
          MAR = PC;
```

```
44                PC  = PC + 1'b1;
45              end
46              @(negedge CLK)
47                if(SP_stop==`yes) H;
48          end
49    endtask
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    endtask
```

参考資料




| | | | |
|---|---|---|---|
| ACC | Accumulator | Clock Gen. | Clock Generator |
| IX | Index Register | Inter Mem. | Internal Memory |
| Sel. | Selector | IBUF | Input Buffer |
| ALU | Arithmetic Logic Unit | OBUF | Output Buffer |
| PC | Program Counter | Outer Mem. | Outer Memory |
| IR | Instruction Register | MCLOCK | Master Clock |
| IDC | Instruction Decoder | | |
| MAR | Memory Address Register | | |
| OB | Observer Bus | DBi | Data Bus for Input |
| AB | Address Bus | DBo | Data Bus for Output |
| ⊗ | 3 State Buffer | | |

KUE-CHIPのブロック図

## 4. 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().mem_r_data ;
29        goto phase2 ;
30      }
31  }
```

| 命令＼フェーズ | | P0 | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|---|---|
| II | | (PC)→MAR<br>PC++ | (Mem)→IR | HALT | | | |
| NOP | | | | NOP | | | |
| IN | | | | (IBUF)→ACC | FLAG CLEAR | | |
| OUT | | | | (ACC)→OBUF | STROBE OUT | | |
| SHIFT | | | | SHIFT | STATUS SET | | |
| IS<br>SA | ACC. IX | | | $(\alpha)\xrightarrow{}(\alpha^*)\xrightarrow{}$ALU→$\alpha$ | STATUS SET | $\alpha$ { ACC, IX | |
| | immidiate | | | (PC)→MAR<br>PC++ | $(\alpha)\xrightarrow{(Mem)}$ALU→$\alpha$ | STATUS SET | |
| EOR<br>OR | direct | | | | (Mem)→MAR | $(\alpha)\xrightarrow{(Mem)}$ALU→$\alpha$ | STATUS SET |
| AND | index | | | | $(IX)\xrightarrow{(Mem)}$ALU→MAR | | |
| L | ACC. IX | | | $(\alpha)\rightarrow(\alpha^*)$ | | | |
| | immidiate | | | (PC)→MAR<br>PC++ | (Mem)→$(\alpha)$ | (Mem)→$\alpha$ | |
| | direct | | | | (Mem)→MAR | | |
| | index | | | | $(IX)\xrightarrow{(Mem)}$ALU→MAR | | |
| ST | direct | | | (PC)→MAR<br>PC++ | (Mem)→MAR | $(\alpha)$→Mem | |
| | index | | | | $(IX)\xrightarrow{(Mem)}$ALU→MAR | | |
| BRANCH | | | | (PC)→MAR<br>PC++ | STATUS CHECK<br>(Mem)→PC | (条件成立) | |

KUE-CHIPの命令実行フェーズ