東海大学紀要情報通信学部
vol.6,No1,2013,pp.15-22

論文

UMLとSysMLを用いた視覚的なハードウェア設計・検証手法

狩野 大樹[*1]，　清水 尚彦[*2]

The Methodology of Visual Hardware Design and Verification using UML and SysML

by

Daiki KANOU[*1]　and　Naohiko SHIMIZU[*2]

（Received on March 28, 2013 & Accepted on July 18, 2013）

Abstract

We are developing the system-level design methodology with the high abstraction level, in order to raise the HW design productivity. In it, we developed the method for hardware design using UML diagram. By using our methodology, we can design hardware with the system-level. Furthermore, we experimented using our methodology. As a result, it turned out by automating the design verification in the hardware design that HW design productivity is raised more. Then, we express hardware constraints visually using SysML, and propose the methodology of performing design verification automatically.

Keywords: Hardware design, UML, SysML, design verification
キーワード：ハードウェア設計，UML，SysML，設計検証

1. 研究背景

半導体デバイスの微細化・高集積化によりLSIの回路規模は著しく増大している．その結果，要求仕様やハードウェア/ソフトウェアの仕様書の量も膨大なものとなっており，ソフトウェア/ハードウェア両方の設計負担は増加している．ソフトウェア開発では，UML図を用いた抽象度の高いモデル駆動開発手法が提案されつつある．一方ハードウェア開発の現場ではレジスタ転送レベル(RTL)の設計が主流である．抽象度の低いRTL設計では，仕様の間違った解釈やコーディングの不具合が起こりやすくなる．仕様設計におけるバグが実装段階で発覚すると，対策に大きなコストがかかる．仕様設計レベルでのミスを減らし，設計効率を向上させるためには，RTLよりも上位の抽象レベルでの設計を行うことが望ましい．

ITRS 2011によると，現在実現可能な複雑度のLSIを効率的に開発するためには，今日の実に50倍もの設計生産性が必要だという[1]．そのためRTLより高い抽象レベルで設計を行う記述言語が提案されてきた．その中に，オブジェクト指向の分析・設計ができるモデリング言語「UML」[2]がある．UMLを用いてシステム仕様を記述することで，抽象度の高い記述を行うことができる．また，抽象度の高いシステムレベル記述言語で記述したコード生成可能な仕様書を「実行可能仕様」と呼ぶ．実行可能仕様が実現できれば仕様書から曖昧さがなくなり，設計のやり直しが起きにくくなるため，設計生産性を向上させることができる．しかしながら，Fig.1のように実行可能仕様によるモデル駆動設計は未だ研究段階である．

また，検証技術の解決策は未だ見当たらず，増大する半導体設計の複雑度に追従できていない状況である．ITRS2011によると，複雑な設計では開発プロジェクト内の検証エンジニアと設計エンジニアの比率は2:1にも達し，設計作業の50%以上が検証に費やされているという[1](Fig.2)．そのため，高抽象レベルによる新たな設計検証手法を確立することで，検証の信頼性を高めると共に検証工程を削減する必要がある．

1.1 システムレベル設計手法の開発

本論文では，我々が開発している，UMLを用いてハードウェアを上位の抽象レベルで設計する手法[3]について述べる．UML図の各図やその要素と高位レベルハードウェア記述言語NSL[4]の構文を対応するルールを定め，UML図で実行可能仕様を実現する．また，この対応ルールに則ってUML図からNSL記述を生成するコンパイラの開発を行う．ハードウェア記述言語としては，VHDLやVerilogHDLがよく用いられるが，これらの言語はUMLのオブジェクト指向記述との解離が大きく直接生成が困難であること，NSLで記述されたコードはツールを用いて論理合成可能なVHDLやVerilog HDL, System Cへと変換できる[4]ので，FPGA実装やソフトウェア協調シミュレーションの実現が容易であることを勘案し生成言語としてNSLを採用した．

また，我々は大きな工数がかかるハードウェア設計検証を視覚的かつ自動的に行う手法を開発している．本論文では，設計検証に用いるテストベンチプログラムやハードウェア制約をUMLを用いて表す手法[5]ならびに，組み合わせ回路のようなハードウェア制約をSysML[6]のブロック定義図を用いて表す手法について述べる．

[*1] 大学院 工学研究科 情報通信制御システム工学専攻
[*2] 情報通信学部 組込みソフトウェア工学科

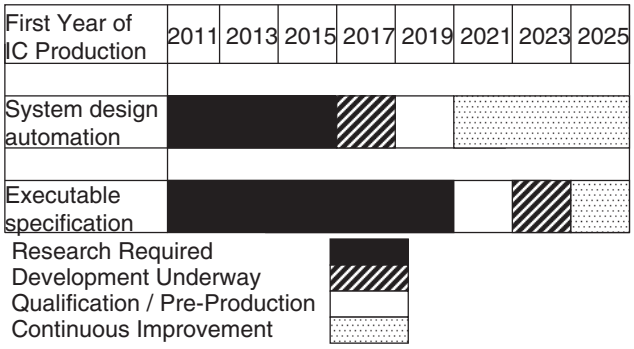


Fig.1　System-Level design potential solutions by ITRS 2011.[1)]

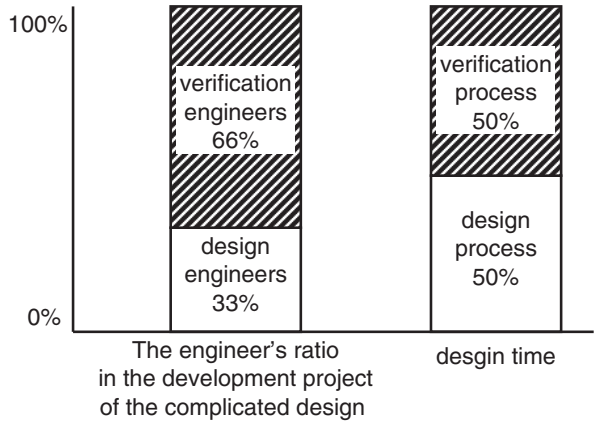


Fig.2　The ratio of the verification cost.[1)]

さらに，当設計手法に対する実証実験により設計手法を評価した．実験では，同じ仕様のハードウェアをレジスタ転送レベルと当設計手法を用いてそれぞれ設計し，性能や設計時間を比較した．

2 章では，我々が開発している UML を用いたハードウェア設計手法について述べる．3 章では，UML と SysML を用いた２つの設計検証手法について述べる．4 章では，当設計手法に対して行った実証実験について述べる．

1.2　類似の研究

過去に提案された UML 図を用いたモデルベース設計手法をいくつか紹介し，当設計手法との相違点を挙げる．

E.Riccobene は UML 図を用いたシステムレベルのハードウェア設計手法を提案した [7)]．ターゲット言語は System C であり，彼らは System C と UML 図の対応ルールを示した．この手法ではハードウェアの構造をクラス図で表し，振る舞いはステートマシン図を用いて表している．しかしハードウェアの振る舞い記述は，ステートマシンの do ラベルのアクションとして System C のコード記述を直接関連付けている．そのため振る舞いの視覚化がなされていない．また，一般に System C で設計されたコードは論理合成サブセットを満たさないため，シミュレーションを行うに留まる．

Moreira T.G は UML 図から VHDL 記述を生成する手法を提案した [8)]．しかし VHDL は抽象度が低い言語であるため UML 図との親和性が低く，UML 図との対応が上手く取れていない．例えば，この手法では UML クラス図におけるクラスの”操作”欄に VHDL の process 文の記述を行ない，”操作の引数”に process 文のセンシティビティリストの信号を記述している．クラス図の”操作”は，他のオブジェクトから呼び出されることで起動する処理名とその引数を記述する要素であり，彼らの手法は UML の概念と異なっている．

T. Schattkowsky は Handel-C をターゲット言語として，UML 図によるハードウェア設計手法を提案した [9)]．彼らはハードウェアの構造をクラス図，コンポジット構造図で表し，クラスのインスタンス関係をクロックドメイン図で表した．ハードウェアの振る舞いはアクティビティ図を用いている．彼らはアクティビティ図を階層的に用いることで，ハードウェアの振る舞い全体を表現している．そのため，この手法ではモジュール分割が行われないフラットなハードウェアが生成される．モジュール内にある機能毎の検証が出来ないためシミュレーション単位が大きいものとなり，検証工程が膨大になるという欠点を持つ．

2.　システムレベル設計手法の開発

本論文で提案する，UML や SysML を用いシステムレベルでハードウェア設計を行う手法について述べる．本章では，システムレベル設計手法の工程と UML を用いたハードウェア設計手法を示す．

2.1　システムレベル設計の工程

システムレベル設計の工程を Fig.3 に示す．

(1)　システム要求分析・システム仕様定義
SysML を用いて開発対象のシステム要求分析とシステム仕様定義を行う [10)]．

(2)　ハードウェア・ソフトウェア機能分割次に SysML により分析した各機能をハードウェアとソフトウェアどちらで実装するか定め，機能分割を行う．

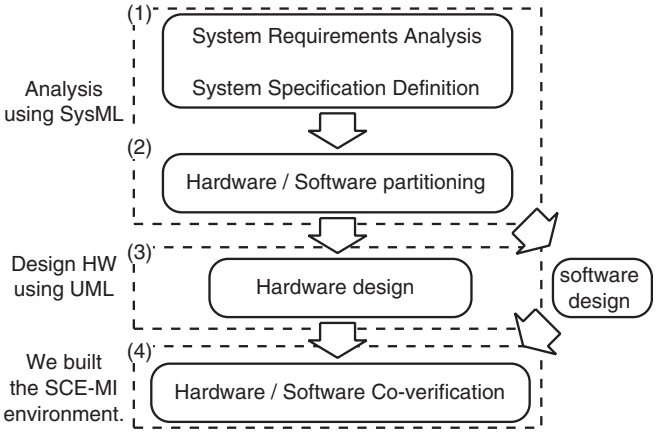


Fig.3　The flow of System Level design.

(3) ハードウェア/ソフトウェア設計
機能分割によって割り振られた機能を UML および SysML を用いてハードウェア/ソフトウェアでそれぞれ設計し，実装および検証を行う [3]．システム分析では SysML を用いて分析するため，ハードウェア設計で用いるハードウェア仕様は SysML 図で表されている．この分析された SysML をさらに詳細化し，ハードウェア設計レベルまで落としこむ．

(4) ハードウェア/ソフトウェア協調検証
ハードウェアとソフトウェアの設計が進んだところで，システム検証のためハードウェアとソフトウェアの協調検証を行う [11]．

以上が我々の開発しているシステムレベル設計手法の工程である．本論文ではハードウェア設計に着目し，ソフトウェア設計については述べない．

2.2 UML を用いたハードウェア設計手法

本項では，本論文が提案する前項の Fig.3(3) の UML を用いたハードウェア設計手法 [3] について述べる．我々は UML の要素とハードウェアの対応ルールを定めることで UML でハードウェアの構造と振る舞いを表現する．この手法を用いたハードウェア設計の工程を Fig.4 に示す．

SysML を用いて要求仕様を分析し，機能分割でハードウェアに割り当てられたものがハードウェア仕様となる．

ハードウェアは対応ルールに則り，UML のクラス図，アクティビティ図，ステートマシン図を用いて表現する．設計者はクラス図を用いてハードウェアの静的構造を表し，アクティビティ図とステートマシン図を用いてハードウェアの振る舞いと状態遷移を表す．ハードウェアを記述した UML は，我々が開発している UML コンパイラによって NSL 記述に変換される．さらに NSLcore[4] を用いて NSL 記述を論理合成可能な Verilog HDL や VHDL に変換し，FPGA に実装する．以上が当設計手法を用いたハードウェア設計の流れである．

Fig.5 に VGA コントローラの機能の一つを表した UML と，この UML を自動変換して得られる回路図を示す．この機能は VGA コントローラの縦側表示区域の制御を行うものである．VGA コントローラの表示制御機能の構造と振る舞いを記述した UML は，我々の手法を用いることで Fig.5 右のようなゲートレベルの回路図に自動変換され，FPGA に実装される．UML と回路図によって抽象レベルが大きく異なることが分かる．

このように，我々の手法を用いることで抽象度の高いハードウェア設計が可能となる．

3. UML と SysML を用いた自動的な設計検証手法の開発

本章では，本論文が提案する UML と SysML を用いた２つの設計検証手法について述べる．２つの設計検証手法はそれぞれ表現できるハードウェア制約に対する得手不得手があるため，設計対象のハードウェアの性質によって手法を使い分ける．

3.1 波形シミュレーションを用いた設計検証手法

波形シミュレーションを用いた設計検証の流れを Fig.6 に示す．UML 設計を行った後，設計者はテストベンチプログラムを NSL で記述する．次に任意のツールを用いてシミュレーションを行い，波形データを生成する．設計者は生成された波形データを元に，自らが理解している仕様通りの動作をしているか確認する．

一般に，設計者が仕様を間違って解釈していた場合，検証の段階でバグを取り除くことが困難となる．また，テストベンチ記述は手書きでコーディングするため，検証工程が複雑になる．これらは設計生産性の観点からも大きな問題である．我々はこれらの問題を解決するため，UML と SysML を用いて設計検証を自動化する．

3.2 UML を用いた設計検証手法

本論文が提案する UML を用いた視覚的な設計検証手法 [5] について述べる．この手法は，シミュレーション時に用いるテストベンチと設計するハードウェアの制約を UML で表し，シミュ

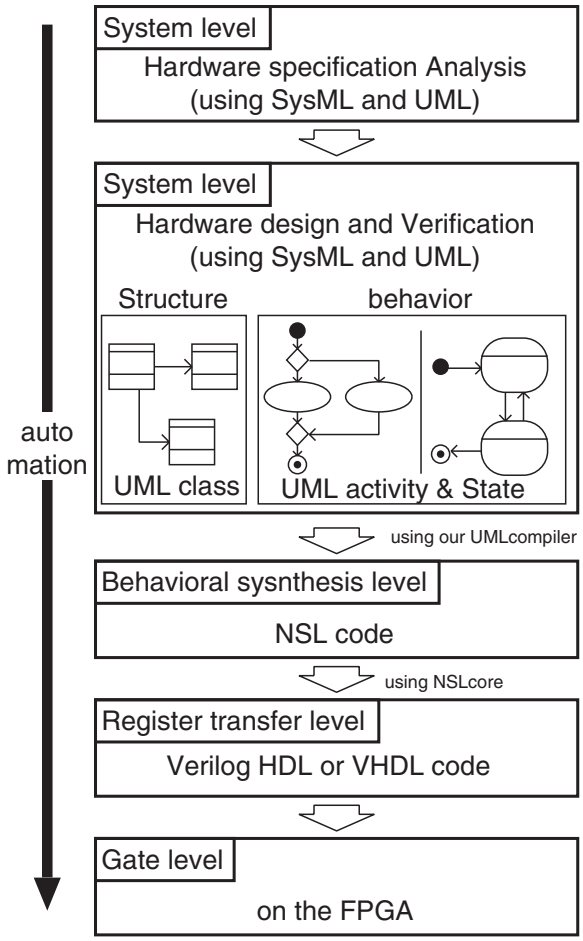
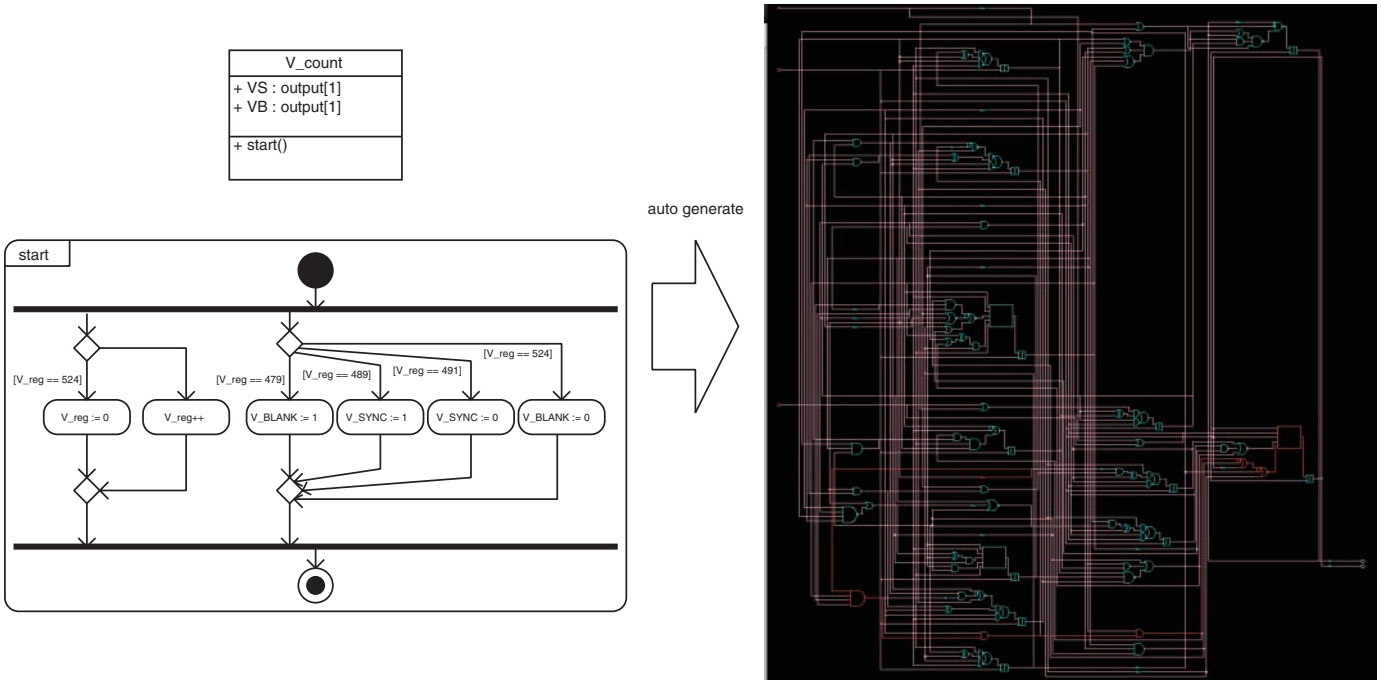


Fig.4 The flow of the hardware design method using the UML diagram which we developed.



(a)
The class and activity diagram expressing the v_count of the VGA controller

(b)
The circuit diagram expressing the v_count of the VGA controller

Fig.5　Automatic conversion on the gate level from the system level.

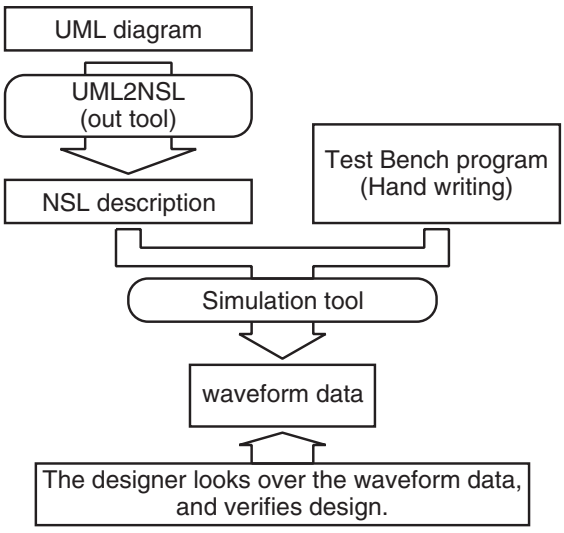


Fig.6　The flow of the design verification methodology by the waveform simulation.[5)]

レーション結果と仕様の比較を自動的に行う．UML を用いた設計検証手法の流れを Fig.7 に示す．

設計対象ハードウェアの要求・システム分析の後，UML を用いてハードウェアの制約とテストベンチ，そしてハードウェアを記述する．我々が開発した UML コンパイラを用いてテストベンチとハードウェアを表した UML から NSL 記述を生成する．同時に UML コンパイラは，ハードウェアの制約を表した UML から制約を抽出した中間ファイルを生成する．テストベンチとハードウェアが記述された NSL 記述を任意のシミュレーションツールによってシミュレートし，シミュレーション結果をテキストファイルで出力する．最後に，出力されたシミュレーション結果とハードウェア制約を示した中間ファイルを比較することで設計検証を行う．

このように UML を用いてハードウェア制約とテストベンチを表し，設計検証の流れを自動化する．

ハードウェア制約，テストベンチ，ハードウェアを記述した UML の例を Fig.8 に示す．ハードウェアの例として 4bit 加算器を用いている．4bit 加算器を表したクラスと，テストベンチや制約を表したクラスは我々が定義したステレオタイプによって区別し，パッケージを分けて表現する．ステレオタイプ constraint を持つ制約クラスの操作には制約条件を表すアクティビティ図やステートマシン図が関連付けられ，これらの UML によってハードウェア制約を表現する．

Fig.9 に 4bit 加算器の制約条件を表すアクティビティ図やステートマシン図の例を示す．Fig.9 のステートマシン図には入力 A と B の値がそれぞれ 2 進数で”1111”と”1100”の場合に遷移する状態 state A がある．この state A には Fig.9 のアクティビティ図の振る舞いが対応しており，このアクティビティ図では出力 S と Co の値を条件とした分岐がある．つまり，4bit 加算器の入力 A が”1111”で B が”1100”のとき，出力 S が”1011”かつ Co が”1”であれば true となり，出力が異なる場合は false となる．

この手法を用いることで設計検証を自動的に行うことができる．ステートマシン図によって制約に条件与えることができるため，この手法は順次回路の制約を表現することができる．しかし，この手法では組み合わせ回路の検証を行う場合，多くの

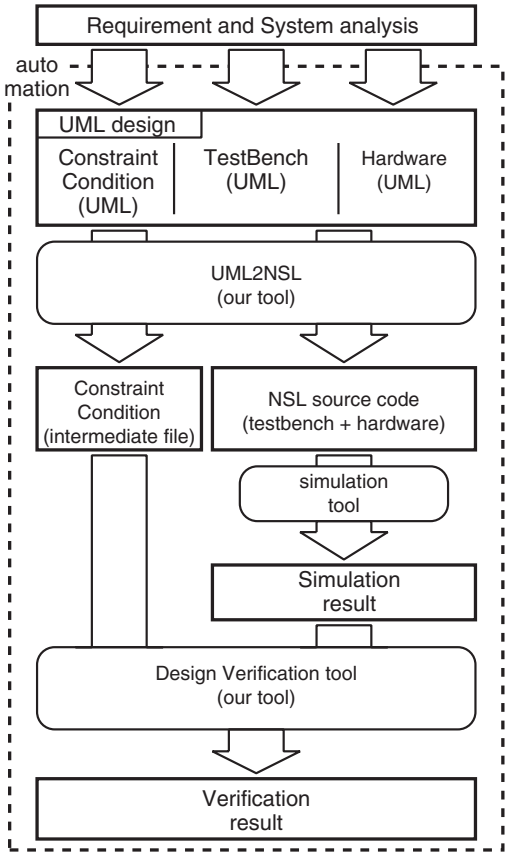


Fig.7 The flow of the automatic design verification methodology using UML.[5)]

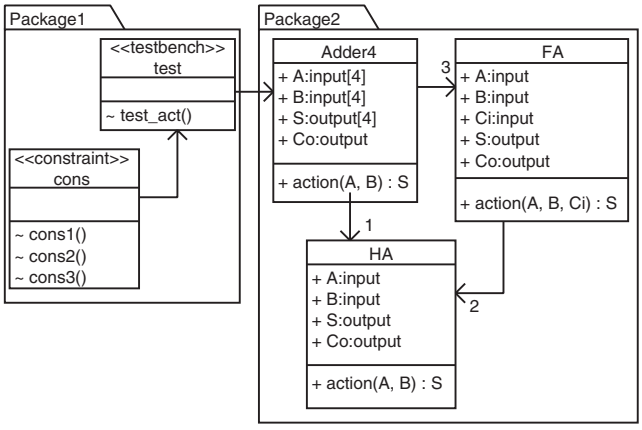


Fig.8 The class diagram showing the test bench and hardware constraints.[5)]

UML 図を記述する必要がある．例えば 4bit 加算器を検証する際，検証する入力パターンの数だけステートマシン図とそれに対応するアクティビティ図を記述しなければならない．

このような組み合わせ回路の検証を行う場合は，次に提案する SysML を用いた設計検証手法を用いる．

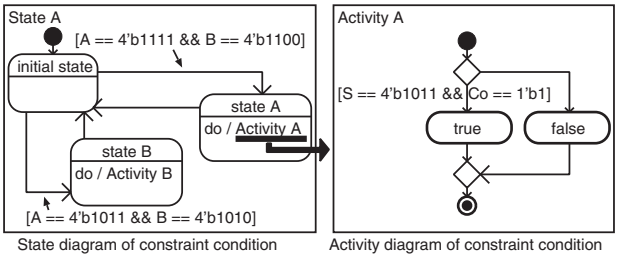


Fig.9 The state machine and activity diagram showing the hardware constraints.[5)]

3.3 SysML を用いた設計検証手法

本論文が提案する，SysML のブロック定義図を用いて加算器のような組み合わせ回路の検証を自動的に行う手法について述べる．SysML を用いた設計検証手法の流れを Fig.10 に示す．

この手法では設計対象ハードウェアの要求・システム分析を行った後，ハードウェアを UML で表し，ハードウェア制約を SysML で表す．ハードウェアを表した UML は，これまでのように我々が開発した UML コンパイラを用いて NSL 記述を生成する．ハードウェア制約を表した SysML はコンパイラによって NSL で記述されたテストベンチプログラムを生成する．最後に，それぞれ生成されたテストベンチとハードウェアの NSL 記述を任意のツールを用いてシミュレートする．生成されるテストベンチプログラムには入力と出力の値を見て制約通りの結果が得られているか判定する記述が含まれる．そのため，設計者はシミュレーション結果に含まれる記述を見ることで制約通りの動作をしているか判断できる．

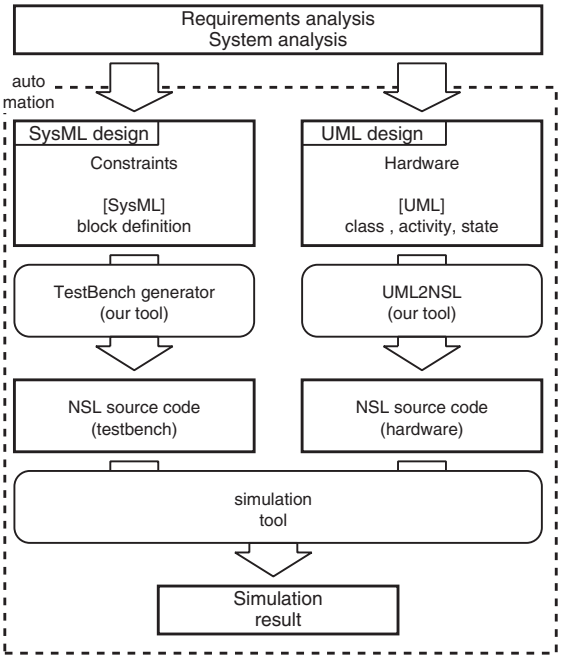


Fig.10 The flow of the automatic design verification methodology using SysML.[5)]

次に，SysML を用いてハードウェア制約を表す手法を解説する．組み合わせ回路は入力のみで出力が決まる回路である．例えば，加算器の制約は２つの入力の値を加算した結果を出力することである．これを NSL の構文に従った式で表すと以下のようになる．

$$\{Co, S\} == A + B \tag{1}$$

{ } で囲われた２つの出力は，NSL における bit 連結を意味する．この式は，入力 A と B の加算値と，出力 S と桁上がり出力 Co を連結した値が等しいことを表している．我々の提案する検証手法ではこのような制約式を SysML で表し，SysML 図から NSL で記述されたテストベンチプログラムを自動生成する．また，ハードウェアを表した UML 図と制約を表した SysML 図の繋がりは SysML の allocation table を用いて表す．

Fig.11 に，UML と SysML で表した 4bit 加算器から生成されるテストベンチプログラムの例を示す．Fig.11(a) は 4bit 加算器の構造を表したクラス図であり，Fig.11(b) は 4bit 加算器の制約を表したブロック定義図である．Fig.11(c) は (a) と (b) の関係を表す allocation table である．(c) により (b) は (a) の制約であることを表している．ここでもし 4bit 加算器内の全加算器を検証する場合は全加算器の制約を表すブロック定義図を記述し，allocation table でクラス FA と関係付ければ良い．Fig.11(d) は (b) から生成される，NSL で記述されたテストベンチプログラムである．

(b) のブロックはステレオタイプ constraint が与えられている．このステレオタイプを持つことで，(b) のブロックは制約ブロックとして扱われる．要素 constraints には数式 (1) のような制約式を記述する．要素 parameters には制約式で用いられるパラメータの定義を行う．(b) の制約ブロックには入力 A，B と出力 S，Co，そして endcnt というパラメータがある．A，B，S，Co は対象ハードウェアである 4bit 加算器が持つ入力出力端子である．endcnt はシミュレーション時間を指定するパラメータであり，我々が定義する型”simutime”を持つ．この型を持つパラメータの初期値によって，乱数生成の試行回数が決定される．

(d) は，(b) から生成される NSL 記述のテストベンチプログラムである．(d) のような制約ブロックから生成されるテストベンチプログラムによる検証では NSL の乱数生成関数を用いる．4bit 加算器の場合 256 種の入力パターンを試す検証が最も信頼度が高いが，入力パターンの多い回路の場合全てのパターンを試すことは現実的ではない．そのためいくつかの乱数を生成して検証を行う．乱数を生成する試行回数は設計者が定める．

(b) の制約ブロック名 test_Adder4 が，(d)3 行目と 6 行目のテストベンチモジュール名に対応している．また，1 行目と 12 行目の記述で設計対象のハードウェアである Adder4 をサブモジュールとして持つことを表している．サブモジュール名などの情報は allocation table で関連付けられているクラスを参照することで得られる．(d) では target というインスタンス名で 4bit 加算器をサブモジュールとして宣言している．4 行目の define ディレクティブ endcnt は，(b) の parameters にある endcnt に対応しており，endcnt は初期値 10 が与えられている．この定数は 24 行目で用いられている．24 行目の記述によってシミュレーションの実行クロック数を制御する．実行クロック数は endcnt に与えられた初期値となる．

7 行目から 10 行目の記述はテストベンチモジュールが持つデータ端子であり，制約ブロックが持つ input と output 型のパラメータに対応している．input 型である A と B には NSL の乱数生成関数によって乱数が与えられる (15, 16 行目)．output 型である S と Co には，サブモジュールとした 4bit 加算器の出力と接続される (18, 19 行目)．18 行目は，サブモジュールの 4bit 加算器が持つ制御入力端子 action を起動することで加算器を動かしている．引数として乱数を生成した端子 A, B を与え，action の戻り値を端子 S に接続している．

21 行目はシミュレーション中のメッセージ出力である．生成した乱数の値と加算器の出力，そしてシミュレーションタイムを表示する．

23 行目は加算器の制約を満たしているか判定する記述である．(b) の制約式に記述された内容を条件とした if 判定を行う．生成した乱数の入力値に対して，得られた出力値が制約を満たしていない場合はエラーメッセージを出力する．

２つの設計手法はそれぞれ組み合わせ回路と順序回路の検証に適している．このように，我々の手法を用いて UML と SysML でハードウェアの構造と振る舞い，そして制約を表すことで，自動的に設計検証を行うことが可能となる．

4. UML を用いたハードウェア設計手法による実証実験

本論文が提案する，UML を用いたハードウェア設計手法によって実際にハードウェアを設計する実証実験を行い，当設計手法を評価した [3)12)13)14)]．3 章では実験方法の概略と実験結果，考察を示す．

4.1 実験方法の概略

実証実験では当設計手法と NSL 記述によるコーディング設計によって同様の仕様をもつハードウェアをそれぞれ設計し，設計時間の比較を行った．各実験を行ったのは当研究室に所属するハードウェア設計初心者の学部生 8 名と院生 3 名である．UML 設計と NSL 記述設計によって同様のハードウェアを設計するが，一人の設計者が同じハードウェアを設計することがないように実験を行った．

実証実験における UML 設計と NSL 設計の設計手順の流れを Fig.12 に示す．設計対象のハードウェア仕様は，UML 設計と NSL 記述設計共に自然言語で記述された仕様書を用いている．各設計者は仕様書を読み仕様を理解した後，UML や NSL 記述によって設計を行う．そのため，これらの実験では仕様分析の比較は行なっていない．また，設計検証も UML 設計と NSL 記述設計共に同様の手法で行った．この実験では，本論文で提案した設計検証手法は用いておらず，それぞれハードウェアを設計した後テストベンチプログラムを NSL で記述して波形シミュレーションを行い，バグがなければ FPGA に実装した．

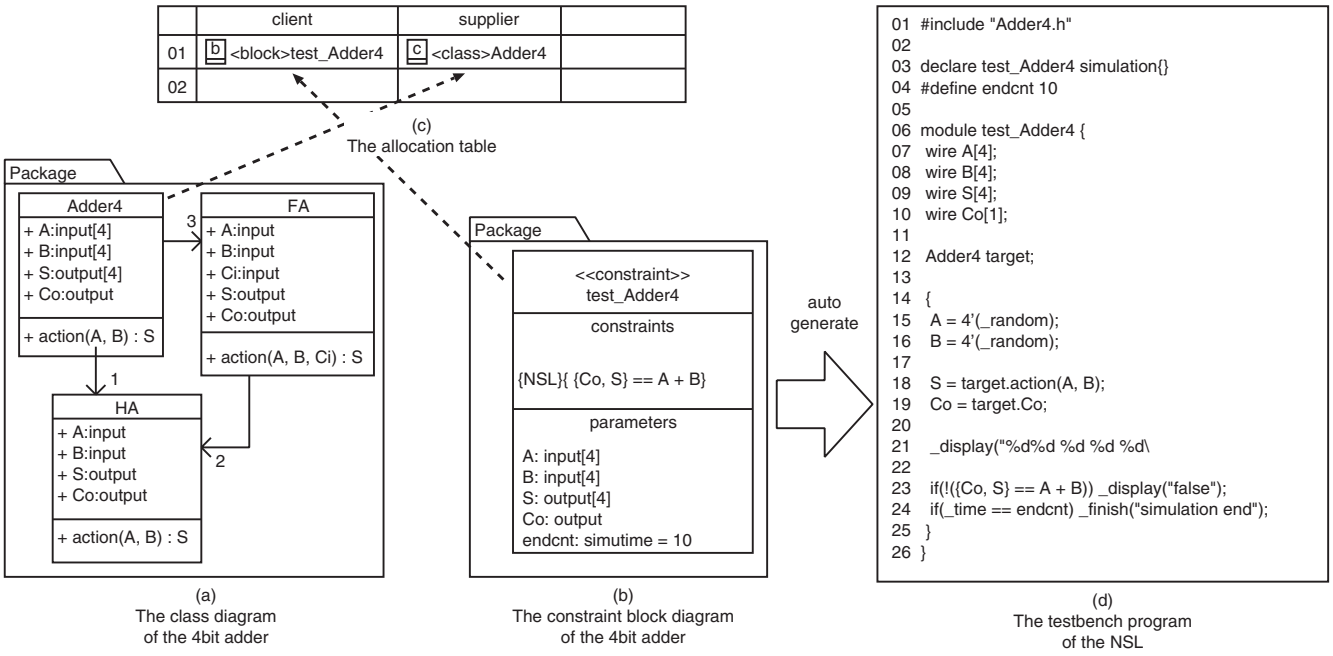


Fig.11 The test bench program of the NSL generated from the constraint block diagram.

4.2 実験結果

各実験の設計時間と検証時間の結果の平均を Fig.13 に示す．設計時間とは設計者が仕様の理解を終えた後，UML 記述や NSL 記述により設計に入った段階から検証に至るまでの時間である．検証時間は検証に入ってからデバッグを行い，波形シミュレーションや FPGA 実装によって正常に動作することを確認するまでの時間である．

実験結果を平均したところ，設計時間では UML 設計よりも NSL 設計の方が短い時間で設計できていることが分かる．一方で，検証時間では UML 設計の方が NSL 設計よりも短い時間で検証し終えていることが分かる．

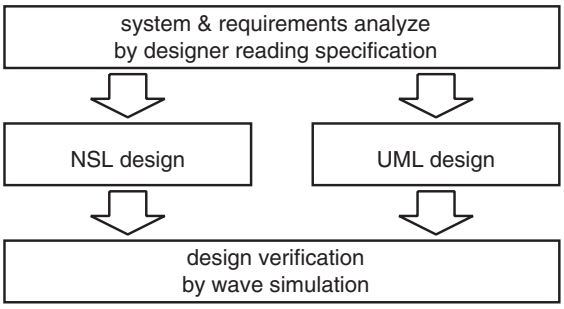


Fig.12 The flow of the procedure of UML design and NSL design in the experiments.

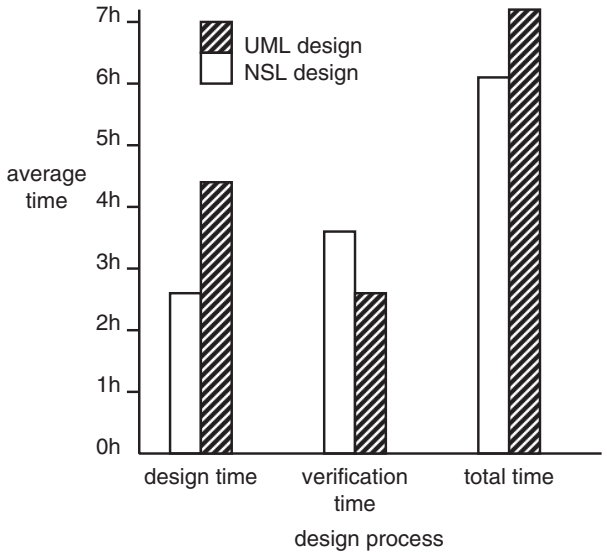


Fig.13 The average of all the experimental results using the our methodology.[5)]

4.3 考察

平均結果を見ると，設計時間は UML 設計よりも NSL 設計の方が短い．これらの実験での仕様分析手法は，UML 設計においても自然言語で記述された抽象度の低い仕様書を設計者が読むことで行なっている．仕様を理解した段階から UML 記述を行なっているため UML と NSL が一対一の関係になっており，設計レベルで記述する UML の量が多いことが設計時間に影響していた．また，仕様分析が手動であるため，設計者が仕様を間違って解釈して設計のやり直しが発生する事例もあった．

そのため，UML や SysML を用いて仕様分析をシステムレベルで行う必要がある．UML を用いて段階的に仕様を設計レベルまで落としこむことで設計レベルでの UML 記述量を減らし，設計時間を削減する．

検証時間は設計時間とは反対に NSL 設計よりも UML 設計の方が短い．ハードウェア設計初心者の学部生が NSL 設計を行った際，細かな構文エラーが多かった．また，そのような構文エラーを取り除く作業に時間を要した．一方で UML 設計を行った場合は構文ミスが少なく，UML を用いて図でハードウェアを表す利点が発揮されたと言える．しかし，実験での設計検証手法は UML 設計と NSL 設計共に同じである．そのため，この検証時間の差は手法による差ではなく，設計時のバグの量の差によって発生したものである．

5. まとめ

我々はハードウェア設計生産性の向上を目的として，抽象度の高いシステムレベル設計を開発している．我々が開発している UML を用いたハードウェア設計手法は，UML で実装可能なハードウェアを表現する．また，これを実現するための UML コンパイラの開発を行なっている．この手法を用いることで，抽象度の高いハードウェア設計を行うことができる．さらに同手法を拡張し，ハードウェア設計検証を視覚的かつ自動的に行う手法の開発を行なっている．

UML を用いた設計検証手法では，UML を用いてハードウェアの構造・振る舞いとハードウェア制約，そしてテストベンチを表す．そしてシミュレーション結果と制約を比較することで自動的に設計検証を行う．この設計検証手法はステートマシン図によって状態毎の制約を記述することができる．そのため，USB パケット通信などの順序回路の検証に適している．

SysML を用いた設計検証手法では，SysML を用いてハードウェアの制約と表し，UML によってハードウェアの構造と振る舞いを表す．そして UML 図と SysML 図から生成した NSL 記述をもとにシミュレーションを行うことで設計検証を行う．この設計検証手法は，乱数を生成して入力に与え，そのタイミングで得られる出力を得て検証を行う．そのため，加算器などの組み合わせ回路の検証に適している．

我々が開発している手法を用いることで, UML および SysML によってハードウェアを視覚的に表し，設計，実装および検証を自動的に行うことができる．

さらに，UML を用いたハードウェア設計手法に対して実証実験を行った．その結果，我々の手法は NSL 設計よりも設計時間は多くかかるが、検証時間は短くなることが分かった．

5.1 今後の課題

今後，我々はハードウェア仕様を表した SysML 図を設計レベルまで段階的に詳細化していく手法の開発を行なっていく．同時に本論文で示した開発・検証手法に用いるコンパイラや検証ツールの開発を進め，完成度を高めていく．また．本論文で提案した設計検証手法に対して実証実験を行い，設計検証手法を評価していく．

参考文献

1) INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS 2011 EDITION , ”2011 Design-1”， http://www.itrs.net/links/2011ITRS/Home2011.htm ，2013/02/08 アクセス
2) Object Management Group (OMG) , Unified Modeling Language(UML) , http://www.uml.org , 2013/02/08 アクセス
3) Daiki KANO, Ryota YAMAZAKI, Naohiko SHIMIZU , ”THE METHOD FOR HARDWARE DESIGN TO GENERATE NSL FROM UML DIAGRAM AND THE EXPERIMENTS WITH FPGA” , ICCEETS2012, pp.594-601, 2012.
4) Overtone Corporation, Next Synthesis Language, http://www.overtone.co.jp, 2013/02/08 アクセス
5) 狩野 大樹, 清水 尚彦, ”UML 図を用いた設計検証の自動化手法の開発” , リコンフィギャラブルシステム研究会, 信学技報, vol.112, no.377, RECONF2012-85, pp.141-146, 2013 年 1 月
6) The SysML Partners , Systems Modeling Language , http://www.sysml.org/ 2013/02/08 アクセス
7) E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio , ”A SoC Design Methodology Involving a UML 2.0 Profile for SystemC”, the conference on Design, Automation and Test in Europe - Volume 2, IEEE Computer Society, pp.704-709, 2005.
8) Moreira T.G. , Wehrmeister M.A. , Pereira C.E. , Petin J.F. , Levrat E. , ”Automatic code generation for embedded systems: From UML specifications to VHDL code”, 2010 8th IEEE International Conference on, Industrial Informatics(INDIN), pp.1085-1090, 2010.
9) T. Schattkowsky , J. H. Hausmann , G. Engels , ”Using UML Activities for System-on-Chip Design and Synthesis” , MoDELS 2006 , LNCS 4199 , pp 737-752, 2006.
10) 山崎 亮太，清水 尚彦，”SysML を用いたモデルベースハードウェア開発の試行”， IEICE Technical Report Vol.112, No.325, pp.63-69, 2012
11) Takahito Nakajima, N.Shimziu , ”Design of Co-Emulation System with SCE-MI API”, Proceedings of the 2009 RISP International Workshop on Nonlinear Circuites and Signal Processing, pp.372-375, 2009.
12) 山崎 亮太, 狩野大樹, 清水尚彦, ”クラス図，アクティビティ図からのハードウェア合成” , 第 37 回パルテノン研究会, Vol.37, pp.1-8, 2011.
13) 狩野 大樹, 山崎 亮太, 清水尚彦, ”ハードウェア設計のためのモデルコンパイラ開発と実証実験 UML ステートマシン図からのハードウェア記述言語の生成” , IEICE Technical Report Vol.111, No.218, pp31-36, 2011.
14) 狩野 大樹, 山崎 亮太, 清水尚彦, ”UML モデル図からハードウェアを設計する手法による実証実験と評価” , IEICE Technical Report Vol.111, No.397, pp.101-106, 2012.