計算機アーキテクチャ 131−13
（1998. 9. 22）

# プログラマブルデバイスを用いた可変構造シミュレーションシステム

野口 裕 最所圭三 福田 晃
奈良先端科学技術大学院大学 情報科学研究科

## 概要

LSI技術の進歩により、これまでソフトウェアで行なわれていた処理をハードウェアレベルで実現することが可能となってきている。本稿では、その応用の一つとして、シミュレーションシステムのハードウェア化の一手法を提案し、その構成について詳しく述べる。モデルとしては、待ち行列を用いたシミュレーションシステムを扱う。種々のパラメータを少しずつ変え、何度も実行する多大なシミュレーション時間を費すシミュレータのハードウェア化は非常に有益である。クロックドリブン型とイベントドリブン型の2つの方式について設計し、その評価を行なった。その結果、イベント発生間隔が長くなるほどイベントドリブン型の性能が向上した。また、イベントドリブン型に関しては、我々の方式はソフトウェアで行なう方式に比べ100倍以上性能が出た。

# A Reconfigurable Simulation System with Programmable Devices

Yutaka Noguchi, Keizo Saisho, and Akira Fukuda
Graduate School of Information Science, Nara Institute of Science and Technology

## Abstract

With progress in LSI technology, many problems, many processes can be executed on hardware-level instead of software-level. In this paper, we propose a method of a reconfigurable hardware simulation system as one of application and describe the system architecture. We use queueing model as the target simulation model. Since this model takes a huge simulation times, it is suitable for a subject of the proposed system. We design two methods, a clock driven type and an event driven type, and evaluate them. As the result, the event driven type is much superior to the clock driven type especially with large intervals of event. For the event driven type, the performance of our method marks more than one hundred times as much as the performance of the method with software only.

## 1 はじめに

ある処理に特化したハードウェアを組み入れた計算機を用いてその処理を行なった場合、ソフトウェアで同様のことを行なった場合に比べ、はるかに高速に処理できるが、他の処理に用いることができない。つまり、その汎用性という観点においては、ソフトウェアを用いた方法と比べ限界があった。そのため、我々はハードウェア構成を柔軟に変更できるようなシステムの研究を行なっている。

近年、LSI 技術の進歩により、Field Programmable Logic Array(FPGA) や Complex Programmable Logic Device(CPLD) 等のプログラマブルデバイスが開発され、その集積度の向上により、実用的規模の回路を実現できるようになってきた。ソフトウェアレベルでハードウェア機能を記述できるハードウェア記述言語 (HDL) を用いることにより、これまでソフトウェアで行なわれて

いた汎用的な計算処理をハードウェアレベルで実現することが可能となってきた。このため、種々の解法アルゴリズムを問題に応じてハードウェア化し、プログラマブルデバイス上で直接実行する、可変構造マシン (Reconfigurable Machine)、あるいは Custom Computing Machine の研究が盛んになっている [1]。

そこで本稿では、その応用の一つとして、シミュレーションシステムのハードウェア化の一手法を提案し、設計、実装、および、その評価について述べる。シミュレーションモデルとしては、待ち行列を用いたシミュレーションを扱う。待ち行列を用いたシミュレータを用いて解析を行なう場合、種々のパラメータを少しずつ変えて何度も実行するため、多くのシミュレーション時間を要する。このようなシステムをハードウェア化できれば、実行時間を激減させることが期待できる。また、シミュレータ自体は変化させないので、同じハードウェアを用いることができる。

このようなハードウェアシミュレーションシステムに関連する研究は、参考文献 [2] でもなされており、そのシミュレーション速度は、従来のソフトウェアでシミュレーションする場合に比べ、かなり高速であることが報告されている。参考文献 [2] のシステムでは、ユーザは、解析モデルを専用記述言語で記述する。また、共通クロックに同期して動作するクロックドリブン型をハードウェア化している。この専用記述言語は、従来のプログラミング言語で記述するよりは簡易になっているが、やはりその記述にはかなりの知識が必要である。また、クロックドリブン型であるため、呼の発生間隔が長い（呼の発生頻度が希有な）シミュレーションの場合、ほとんど呼が発生しないので、ソフトウェアによるイベントドリブン型より遅くなる可能性がある。そこで、本システムでは、ユーザがより簡易に解析モデルを記述できるようにするため、モデルの記述に GUI ツールを用いることにし、さらに、呼の発生間隔が長いシミュレーションに対応するために、イベントドリブン型も用いることができるようにする。

提案するシステムを用いて、ユーザは、次の手順で解析対象となるシステムの解析を行なうことができる。

i) 解析対象となるシステムをシミュレーションモデル構築 GUI ツールを用いて、待ち行列ネットワークとしてグラフィカルにモデル化する。

ii) 作成したモデルから抽出したパラメータを基に、本システムで提供する GUI-HDL トランスレータを用い、ハードウェア記述言語 (Hardware Description Language:HDL) のプログラムに変換する。この変換されたプログラムに、シミュレーション回路が記述されている。

iii) 得られた HDL プログラムを用いて論理/レイアウト合成し、プログラマブルデバイスを用いた可変構造マシン上にシミュレーション回路を実現する。

iv) 可変構造マシン上でシミュレーションを実行し、その結果を収集する。

本システムの構成は、以下の通りである。

(1) シミュレーションモデル構築 GUI ツール

(2) GUI-HDL トランスレータ

(3) 論理/レイアウト合成ツール

(4) プログラマブルデバイスを用いた可変構造マシン

このうち、(1),(3),(4) は既存のものを使用し、(2) を開発する。

以下、2 章ではシミュレーションモデル構築 GUI ツールについて、3 章では本システムのハードウェアアーキテクチャについて述べ、4 章では、シミュレーションを通して、時刻管理の方法の違いによる実行速度への影響についての考察と実際に基本的な待ち行列モデルを実装した際の実装結果について述べる。

## 2 シミュレーションモデル構築 GUI ツール

シミュレーションモデル構築 GUI ツールとして、一般的な市販汎用 GUI ツールである Visual SLAM を使用することにした。Visual SLAM は、Prisker Corporation 社製の汎用シミュレーションモデル構築ソフトウェアツールである [3] 。

我々の研究室では、マルチプロセッサシステムにおけるスケジューリングアルゴリズムの評価のために Visual SLAM を用いており、これに関するノウハウが蓄積されており、HDL へのトランスレータの作成も、他のシミュレーションツールを用いた場合よりも容易になると考えたからである。

本システムにおいて、ユーザは、この GUI ツールを用い、解析したいシステムの待ち行列モデルをグラフィカルな状態で記述する。図 1は、この GUI ツールを用いて、基本的な A/B/1 モデルの待ち行列モデルを表したものである。

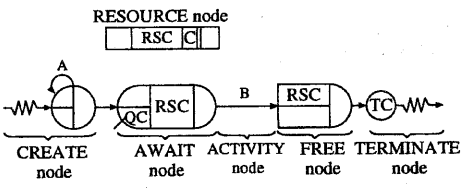


図 1: Visual SLAM で表した A/B/1 待ち行列モデル

図 1は、各イベントの処理の流れを表しており、CREATE ノードにおいて到着間隔分布を指定 (図中の記号 A) して呼の発生を行ない、AWAIT ノードにおいてキュー (Queue) の最大容量と使用する RESOURCE ノードを指定している。また、RESOURCE ノードでは、同時に何個のイベントが滞在できるかを指定している。そして、図 1の ACTIVITY ノードで、サービス時間分布を指定 (図中の記号 B) しており、その後の FREE ノードで RESOURCE ノードの解放の制御を行なっている。また、TERMINATE ノードで、シミュレーション終了の条件として、サービス終了数を指定できる。そして、この図から、Visual SLAM はネットワーク文を生成する。本システムでは、この生成されたネットワーク文を基にして、シミュレーション回路を構成する。例えば、図 1のモデルから生成されたネットワーク文のうち、FREE ノードに関する部分の記述は図 2のようになる。

```
FREE,{{RSC, ... } ... }, ... ;
```

図 2: ネットワーク文の記述 (FREE ノード部)

## 3 可変構造シミュレーションシステム

本章では、本稿における可変構造シミュレーションシステムのターゲットハードウェア、シミュレーション回路アーキテクチャ、および、GUI-HDL トランスレータについて述べる。

### 3.1 可変構造マシン Paradigm RP2000

可変構造テストベッドとして市販の Paradigm RP2000 を用いる。Paradigm RP2000 は、Zycad 社製の論理エミュレータである。図 3に RP2000 の構成を示す [4] 。

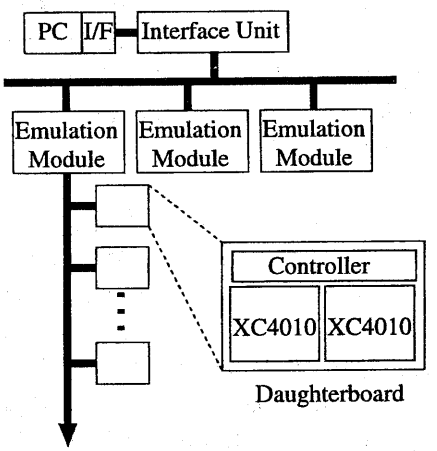


図 3: 可変構造テストベッド RP2000

RP2000 は、Xilinx 社製 FPGA XC4010 を 16 個実装したモジュール (マザーボード) を 3 枚搭載している。モジュール 1 枚あたり 30,000 ゲート相当、全体で 90,000 ゲート相当である。FPGA は Daughterboard と呼ばれるボードに制御用のコントローラとともに 2 個搭載されており、これらをモジュール上でクロスバ結合し、モジュール間は各々の 512 本の I/O コネクタに結合される。各モジュールには 8 個の Daughterboard が搭載されて

いる。この他に、全FPGAにクロック供給などに利用できる配線が装備されている。動作速度は回路によるが、簡単な回路であれば 10MHz 程度で動作する [5]。

### 3.2 シミュレーション回路アーキテクチャ

待ち行列は、客 (customer) または呼 (call) がサーバでサービスを受けるシステムをモデル化したもので、元々は、電話交換の問題のために作られた理論であるが、現在は、計算機の性能評価にも広く用いられている。解析には図 4で示すような待ち行列ネットワークを用いて行なう。本システムにおける待ち行列用シミュレーション回路は、シミュレーションモデル構築 GUI ツールで作成したモデルの各要素を、それぞれシミュレーション回路に置き換えたものになる。乱数発生器が生成する乱数を用いてシミュレーションを行なう。この際、呼の発生間隔が長い場合に、シミュレーション時間をクロックと同期させて単純に実行していると、ほとんど呼が発生しない。一般に、精度を 10 倍にする（有効桁を 1 桁増やす）には 100 倍の計算時間が必要である [6] 。したがって、それに用いる乱数も周期の長いものが必要となるので、呼の発生間隔が長いシミュレーションも多いと考えられる。

そこで、シミュレーション回路に、各要素の動作について単純に共通クロックに同期して動作する回路（以後、クロックドリブン型回路と呼ぶことにする）だけではなく、クロックエッジに合わせて各要素のイベントの発生間隔に用いる乱数値を比較計算して上述の無駄なシミュレーション時間をいっきに進めて実行時間短縮をする回路（以後、イベントドリブン型回路と呼ぶことにする）も生成するようにする。図 5 に本システムにおけるイベントドリブンを用いた処理系の概念を示す。

イベントドリブン型の処理系では、比較回路を含むようになるため、回路構成が複雑になる、動作周波数が小さくなる等の問題を持つ。このため、シミュレーションできる問題の大きさがクロックドリブン型より小さくなる。また、呼の発生間隔が短い場合、クロックドリブン型より遅くなることが考えられる。そのため、GUI-HDL トランスレータでは、どちらの型にも変換できるようにする。

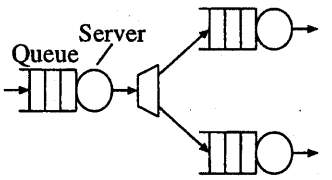


図 4: 待ち行列ネットワークの例

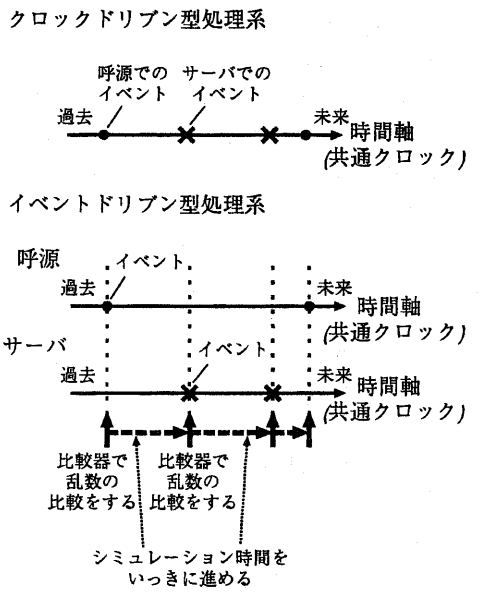


図 5: 本システムにおけるイベントドリブン処理系

### 3.3 GUI-HDL トランスレータ

GUI-HDL トランスレータは、シミュレーションモデル構築 GUI ツールを用いて作成したモデルから作成されたネットワーク文を基にして、前述の可変構造マシン上に実現するシミュレーション回路用の HDL 記述に変換する。本システムにおいては、HDL としては VHDL を採用し、トランスレータ自身はプログラミング言語 Perl で記述されている。

この GUI-HDL トランスレータを用いて変換された HDL プログラムの例として、図 2の FREE ノードに関するネットワーク文を HDL 変換したプログラムの一部を図 6に示す。この部分では、RESOURCE ノードの解放を要求する INPUT 信号が入力すれば RESOURCE を解放させる OUTPT 信号を出力するという動作が記述されている。

```
-- FREE.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity FREE is
   port(OUTPUT: out std_logic;
        INPUT,RESET,CLK: in std_logic);
end FREE;
architecture FREE_body of FREE is
begin
RSCFREE:process(RESET,CLK)
     begin
     if RESET = '0' then
        OUTPUT <= '1';
     elsif CLK'event and CLK = '1' then
        if INPUT = '1' then
           OUTPUT <= '0';
        else
           OUTPUT <= '1';
        end if;
     end if;
     end process;
 .......

end FREE_body;
```

図 6: HDL プログラムへの変換例 (FREE ノード)

# 4 評価

## 4.1 時刻管理に関する考察

表1に乱数の周期を基にしてRTLレベルでクロックドリブン型とイベントドリブン型の回路を比較した結果を示す。対象モデルとしては、GI/G/1待ち行列モデルを用いた。表1は、同数のサービスを終了した時点での両モデルにおける実行クロック数とイベントドリブン型モデルの回路を用いて行なったシミュレーション実行時間を1としたときのクロックドリブン型モデルの回路を用いて行なったシミュレーション実行時間の比率を表している。表1より、乱数の周期が一桁上がるに従い、実行時間が急激に増加しているのが確認できる。

このように、乱数の周期が長くなるに従って、シミュレーションシステムの回路構成はイベントドリブン型処理系を用いたものの方がクロックドリブン型処理系を用いたものよりもシミュレーション実行時間の速度向上率は高くなった。これは、3.2節で述べた理由によるものである。しかしながら、シミュレーション回路全体を一つのまとまりとしてイベントドリブン型回路で構成するとシステム全体が逐次処理になってしまい、処理に並列性が引き出せない。このため、解析モデルを呼の発生間隔などを基に並列処理を行なうことのできる部分に分けて、各部分をイベントドリブン型回路にするなどの工夫が必要と思われる。

表 1: 同数のサービス終了数のときの実行時間比

| 乱数周期 | 実行クロック数 | | 実行時間比 |
|---|---|---|---|
| | イベントドリブン型回路 | クロックドリブン型回路 | |
| 1,000 | 106 | 26,729 | 252 |
| 10,000 | 90 | 226,729 | 2,519 |
| 100,000 | 103 | 1,762,473 | 17,111 |

## 4.2 性能予測

今回、実装が間に合わなかったので、実機での動作評価はできなかった。そこで、RTLレベルでのシミュレーションによる簡単な性能予測を行なった。解析対象モデルとしては、GI/G/1待ち行列モデルを用いた。比較対象として、本システムにおいてシミュレーションモデル構築GUIツールとして用いたVisual SLAM（ソフトウェアツール）で同じのモデルを用いることにした。Visual SLAMはPentium133MHzを載せたパーソナルコンピュータ上で実行している。そして、本システムの動作速度は3.1節の記述を参考にし少し遅めに見積もって2.5MHzに設定した。この動作速度は、本システムのようなシステムの場合、システム全体の動作速度は乱数発生器によって律速される[2]ので、その動作速度を関連文献で達成されているクロック(3.2MHz)を参考にして定める。また、本システムのシミュレーション回路は、3.2節で述べたイベントドリブン型回路で構成した。結果を表

2に示す。表2より、本システムを使用することにより160～230倍の速度向上が見込まれることが分かった。動作クロックが50倍ほど遅いにもかかわらず、100倍以上の性能が出ている。乱数発生器の回路を工夫できて速度向上ができれば、この差はさらに広げることができる。

表2: 動作予測比較

| サービス終了数 | 実行時間 (ms) | |
|---|---|---|
| | ソフトウェア | 本システム |
| 4,000 | 2951 | 13 |
| 40,000 | 20869 | 128 |

### 4.3 乱数発生器に関する考察

シミュレーションにおいて疑似乱数は非常に重要な要素である。現在、本システムでは乱数発生器として合同法を用いている。しかしながら、合同法での疑似乱数の周期は乱数のビット幅で律されてしまい、例えば、乱数のビット幅がaビットのときは得られる乱数の最大周期は$2^a-1$になり、aビットの乗算器と加算器が必要となり回路構造が複雑になってしまう。また、aの値が大きくなるにつれ、動作速度の面で不利になってしまう。疑似乱数発生の他の方法として有名な方法にM系列がある。M系列を用いた方法では、用いる原始多項式の次数bによって周期が決まり$(2^b-1)$、乱数のビット幅とは無関係である。bは乱数発生に必要なシフトレジスタの段数に等しい [7] [8]。排他的論理回路とシフトレジスタによる単純な構造の回路構成で実装できる。また、シフトレジスタの1動作クロックごとに乱数を生成できるため、長い周期の乱数発生に適している。本システムでは、かなり長い周期の乱数を用いることができるようにする予定なので、以上の点を考慮して、今後はM系列を用いた方法で乱数発生器を再構成していくつもりである。

## 5 おわりに

本稿では、プログラマブルデバイスを用いた可変構造マシンの応用の一つとして、シミュレーションシステムのハードウェア化の一手法を提案し、その構成について述べた。また、シミュレーションを通して、時刻管理の方法の違いによる実行速度への影響についての考察と本システムの簡単な評価も行なった。4.1節で考察したように、シミュレーション回路全体をひとまとまりとしてイベントドリブン型回路で構成するとシステム全体が逐次処理になってしまうので、呼の発生間隔などを基に並列処理を行なうことのできる部分に分けて、各部分をイベントドリブン型回路にするなどの工夫が必要と思われる。

今後は、上記のようなことを考慮しつつ、複雑なアルゴリズムを必要とする大規模な待ち行列モデルシミュレーションへの展開に向けて研究していくつもりである。

## 参考文献


[1] 沼 昌宏: "FPGAを利用したアーキテクチャとシステム設計", 情報処理, Vol.35, No.6, pp. 511-518, Jun. 1994.

[2] 山本 欧, 柴田 裕一郎, 天野 英晴: "可変構造を持つ, 並列システム解析用確率モデルシミュレーションシステム", 並列処理シンポジウムJSPP'98 論文集, pp. 295-302, Jun. 1998.

[3] 構造計画研究所: Visual SLAM システム解説書, 1997.

[4] Zycad Corporation: Paradigm RP - Paradigm RP Hardware Reference Manual.

[5] 高木 一義: Paradigm RP2000を用いた論理エミュレーション ハンドブック, 1997.

[6] 広中平祐 他: 現代数理科学辞典, 大阪書籍, pp. 682-686, 1994.

[7] 伏見 正則: 乱数, 東京大学出版会, 1989.

[8] 戸川 隼人 他: インターネット時代の数学, bit別冊, 共立出版, pp. 188-195, 1997.