# デバイス・シミュレーション・エンジンの一構成法

冨田 栄作　山本 利雄　井口 幸洋

明治大学 理工学部 情報科学科
〒 214-8571 川崎市多摩区東三田
E-MAIL: {tomita , y-toshio , iguchi}@cs.meiji.ac.jp

あらまし

高速でかつ安価なデバイス・シミュレーション専用のハードウェアの構成について検討した結果を報告している. 離散化されたデバイス方程式の格子点上の未知変数の計算には FPGA を用い, 未知変数の格納には FPGA 内部の SRAM を用いるという方法で部品数を削減するという現実的な構成方法を提案している. デバイス方程式に先だって, ポアソン方程式の数値解析を取り上げ実装の検討を行なった. またスタティック・タイミング・アナライザとタイミング・シミュレータを用いた予備的な評価も行なっている.

キーワード　デバイス・シミュレーション , シミュレーション・エンジン , FPGA

# A Realization of Device Simulation Engine

Eisaku TOMITA　Toshio YAMAMOTO　Yukihiro IGUCHI

Dept. of Computer Science
Meiji University
Kawasaki 214-8571 , JAPAN
E-MAIL: {tomita , y-toshio, iguchi }@cs.meiji.ac.jp

Abstract

A high-speed and low-cost device simulation engine is proposed together with the investigation for its structure and realization. The proposal is based on the employment of FPGAs which are used for the calculations of the unknown variables of the discretized device equations. The SRAMs in the FPGAs are partly used for the memories, thus resulting in the cut down of the number of components. The performance of simulation engine is estimated by a static timing analyzer and timing simulator.
Device simulation , Simulation Engine , FPGA

key words　Device simulation , Simulation Engine , FPGA

# 1 はじめに

半導体集積回路の集積化技術の発達にともない，集積回路上の半導体デバイスの寸法はディープサブミクロンにまで縮小しようとしている．このような極微細なデバイスの設計にはキャリヤの非定常輸送を考慮したデバイスシミュレータが必要になる．キャリヤの非定常輸送は，キャリヤ密度，運動量密度，そしてエネルギー密度に関するバランス方程式を用いて解析することができ，これらの偏微分方程式を数値的に解く必要がある [1]．しかしこのとき1格子点あたりの未知変数の数は従来のドリフト拡散モデルに対して数倍程度に増大し，従来から用いられている陰解法の適用が困難になる．陽解法を適用するとなると反復回数の増加のためその計算量は膨大となり，ワークステーションを用いてもかなりの計算時間を要することになる．そこで本稿では陽解法によるデバイス・シミュレーションを専用ハードウェアによって行なう方法について検討したので，その結果について報告する．

陽解法のアルゴリズムは基本的に，離散化されたデバイス方程式の格子点上の未知変数を順次反復計算することによって行なわれる．例えば $n \times m$ 個の格子点について離散化されたデバイス方程式を単一のプロセッサで解析するときには，当然のことながら反復計算の1回分で $O(n \times m)$ の計算量となる．文献 [2] では PC の外部に DSP を1個搭載しデバイス・シミュレーションの高速化を図っている．$n \times m$ は $10^4$ 程度になるので，高速化のためすべてを並列に計算するには $10^4$ 個程度のプロセッサを並べればよいが，現実的ではない．$n$ 行 $m$ 列の1列分 (約 100 個程度) のプロセッサを用意する方法を採用することにすれば，計算量は $O(m)$ に低減できる．

計算に用いるプロセッサを FPGA [3] で実現することにする．格子点上の未知変数の値を FPGA 内部の SRAM に格納することにして，外部ピンを節約すれば，部品数を減らすことができる．スタティック・タイミング・アナライザを用いて予備的な性能評価も行なった．

本報告の構成は以下の通りである．2 節でデバイス方程式の解析について簡単に述べ，予備実験のために採用したポアソン方程式の解析との関連について述べる．3 節では今回提案するシミュレーション・エンジンの構成と動作について説明し，簡単な評価を示す．4 節でまとめと今後の課題について言及する．

# 2 デバイス方程式の陽解法とポアソンの方程式の数値解析

1 節で述べたように極微細化な半導体デバイス内のキャリアの非定常輸送は，バランス方程式を用いて解析することができる．バランス方程式はボルツマン方程式のモーメントについての方程式で，未知変量としてキャリア密度，運動量密度，そしてエネルギー密度を持つ．したがって未知変数の数は，従来のドリフト拡散モデルの数倍になり，陰解法のための変数ヤコビアンを組むことが困難になる．そこで陽解法を採用することになるが，このとき計算の反復回数は数千回に達し，ワークステーションにしても膨大な計算量になる [1]．

陽解法のアルゴリズムは，ある格子点上の未知変数とそれに隣接するすべての格子点上の未知変数の間の算術計算を全ての格子点について行なう反復計算であるので，ポアソンの方程式の数値解析で用いる点緩和法に似ている．そこでここでは，バランス方程式に先立って，予備実験としてポアソンの方程式を有限差分法で離散化して解く計算を専用エンジンで行なう場合を想定し，このためのハードウェアの構成方法について検討を行なった．とりあえずここでは 2 次元のポアソン方程式に限定して話を進める．

二次元領域でのポアソンの方程式は以下のようになる．

$$\frac{\partial^2 V(x,y)}{\partial x^2} + \frac{\partial^2 V(x,y)}{\partial y^2} = -C(x,y) \qquad (1)$$

ここで $V(x,y)$ は点 $(x,y)$ におけるポテンシャルで，$C(x,y)$ は電荷密度に比例する量である．式 (1) を有限差分で離散化するときのメッシュの様子を，図 2.1に示す．また離散化されたポアソンの方程式は

$$\frac{V_{i-1,j} - 2V_{i,j} + V_{i+1,j}}{\delta x^2} + \frac{V_{i-1} - 2V_{i,j} + V_{i,j+1}}{\delta y^2} = -C_{i,j} \qquad (2)$$

メッシュの格子点 $(i,j)$ 上のポテンシャルと電荷密度は，それぞれ $V_{i,j}$ と $C_{i,j}$ で表されている．また $\delta x$ と $\delta y$ は格子点の間隔である．いま簡単のために $\delta x = \delta y = 1$ とすれば

$$V_{i,j-1} + V_{i-1,j} - 4V_{i,j} + V_{i+1,j} + V_{i,j+1} = -C_{i,j} \quad (3)$$

すべての格子点についてのポテンシャルの値 $V_{i,j}$ を求めるためには，式 (3) がすべての格子点で誤差の範囲内で成り立つまで反復計算せねばならない．陽解法のデバ

イス方程式についての反復計算もこれに似た計算になるが, 一格子点当たりの未知変数が増大するため, 隣接格子点上の未知変数も増大し, 計算そのものはもっと複雑になる.

さて式(3)の場合の格子点数が100×100, つまり10000個であったとすると, 反復計算の1回分で10000回の計算をしなくてはならない. その計算量は膨大であるが, 計算をどの格子点の順に行なっても最終的な解は一致することが一般に知られている. そのため複数の格子点について計算を並列に行なうことで計算の高速化を図ることができる.

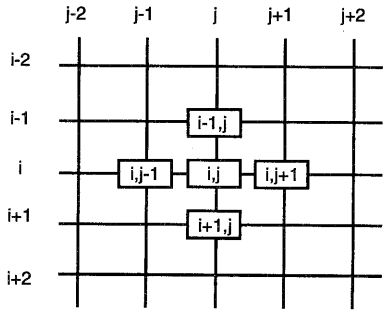


図 2.1: 連続空間のメッシュによる表現

# 3 デバイス・シミュレーション・エンジン

## 3.1 アーキテクチャの概要

図3.1にエンジンの構成を示す. 斜線部分それぞれがメッシュの格子点に相当する. エンジンの四方には境界条件を格納するレジスタを用意する. 境界条件および各メッシュの格子点の初期値を入れ, 値が収束するまでシミュレーションを行なう. 式 (3) の計算を行なうPE(Processor Element) を全ての格子点に配置し, 四方からのデータを入力として反復計算を並列に行なうこともできる. 実際に取り扱うメッシュの交点は $n \times m$ ($n = 20 \sim 100$ 程度, $m = 100$ 程度) であるのですべての格子点にPEを用意すると装置が大規模なものになり現実的ではない. ここでは $1 \sim n$ 個のPEを用意し全体の計算を行なう方法を提案する.

式(3)の計算を行なうためのPEの設計を行なう. 式(3)は式(4)のように変形できる.

$$V_{ij} = \frac{\frac{V_{i+1,j}+V_{i-1,j}}{2} + \frac{V_{i,j+1}+V_{i,j-1}}{2}}{2} \qquad (4)$$

ここでは簡単のために $C_{i,j} = 0$ として考える. この式を実現するための回路を図3.2に示す. AVGは平均をとるユニットである. この演算を3回行ない, 演算結果をレジスタに格納する. AVGでは2で割る作業は1ビット右シフトするだけでよいので加算器のみで実現できる. 3個の演算器を用いてパイプライン処理を行なえば高速化も可能だが, 今回は1個の演算器を順番に3回使って計算する方式を採用した.

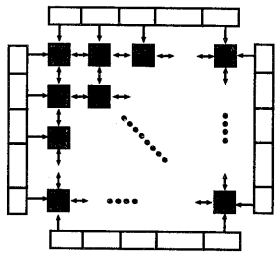

図 3.1: エンジンの構成

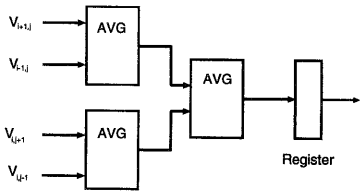


図 3.2: Processor Element の概略図

## 3.2 計算のながれ

$1 \sim n$ 個のPEを用いた計算について説明する. まずPE1個の場合を例に計算の流れを示す.

**例 3.1** 図3.3に示す $4 \times 4$ のメッシュを用いて説明する.

(a) 初期状態として四方のうち上辺と左辺を1.0, 右辺と下辺を0.0, 内部の空間を全て0.0とする.

(b) 左上から順番に四方の値の平均を取り, 結果のその格子点の値とする. $V_{0,0}$ の値は $(1.0+1.0+0.0+0.0)/4 = 0.5$ .

(c) 右に1つ計算する格子点を移動し, 同様の計算を行なう. $V_{0,0}$ が0.5になっているので $(1.0 + 0.5 + 0.0 + 0.0)/4 = 0.375$ となる. $V_{0,3}$ まで計算した後は1段下の

$V_{1,0}$ から $V_{1,3}$ までを同様に計算する. この計算を全ての格子点について収束するまで反復する. 収束結果を (d) に示す.

次に PE を 2 個用意した場合を考える. 各 PE がメッシュのどの部分の計算を担当するかが問題である.

**例 3.2** 図 3.4に図 3.3の初期状態から同時に 2 つの格子点について計算を行なう様子を示す. 計算は繰り返し行なわれるため, 一つ前の結果を記憶しておけば都合がよい.

(a) $V_{0,0}$ と $V_{1,0}$ は, 以下の計算を同時に行う. のではなく, 前回の計算結果, つまりこの場合では初期状態である 0.0 を用いる.

$V_{0,0} = (1.0 + 1.0 + 0.0 + 0.0)/4 = 0.5$

$V_{1,0} = (1.0 + 0.0 + 0.0 + 0.0)/4 = 0.25$

$V_{1,0}$ の計算においても $V_{0,0} = 0.0$ を用いることを注意.

(b)$V_{0,3}$ と $V_{1,3}$ の計算まで終ると下の段に移動し,$V_{2,0}$ と $V_{3,0}$ の計算を行なう.

(c) 以下, 同様に値が収束するまで反復計算を行なう. 収束結果は図 3.3(d) と一致する.

3 個以上の PE を用意した場合も同じように計算できる. このように PE を増やしていき, 縦 1 列 n 個の PE を用意した場合, 左から右まで計算を行なえば全ての格子点について計算ができる. これを値が収束するまで反復計算する.

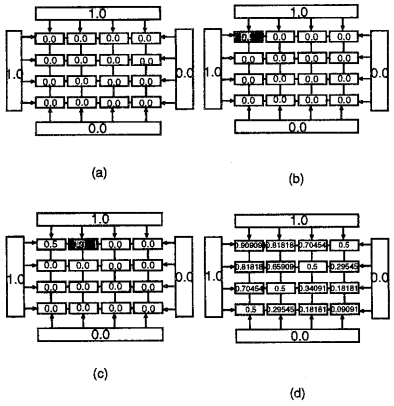


図 3.3: 計算の流れ

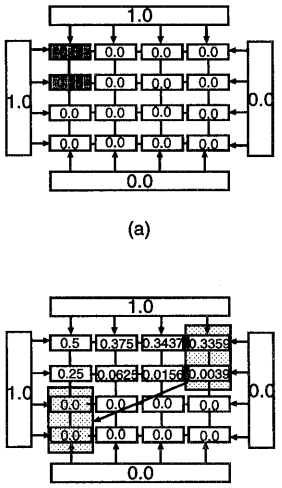


図 3.4: 2 個の PE を用いたエンジン

## 3.3 メモリの構成

メッシュの格子点上の値を保持する記憶ユニットが必要である. 全ての格子点に PE を用意する場合は, 1 つの PE は 1 つの格子点上の値を覚えるだけでよいので, 格子点の値は PE 内のレジスタに保持できる. PE 1 個の場合は, 全ての格子点上の値を格納しなければならない. これを PE 内のレジスタに記憶させるのは現実的でないので内部か外部にメモリを用意する. しかし, PE が複数個ある場合, メモリが 1 ユニットのみであると複数の PE が同時にデータを読み出すことができない. そこで,PE 1 個に対しメモリ 1 個を用意することにした. この方式では, 各メモリにアクセスする PE が 1 つに限定されるので各 PE が同時にメモリアクセスすることが可能になる. 図 3.5は 2 個のメモリを用いた場合の例を示す. 2 個の場合は 1 つのメモリに一行飛ばしにデータを格納すればよい. メモリ 1 個に格納するデータ量はメモリを 1 つ用意するときの半分の容量で済む. これは FPGA 内にメモリを使って実現する場合, メモリ容量が少ないので有効に働く. PE を 4 個, 8 個と増やしていくに伴い, 1 つのメモリの容量は全体の 1/4, 1/8 となっていき, 全体ではメモリの総量は変わらない.

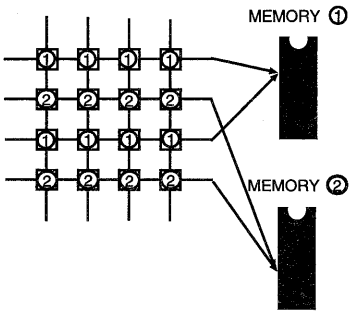


図 3.5: メモリの分割

1つの格子点を計算するために上下左右の4方向の値が必要である. その他に計算結果が収束しているか判定するためにその点の前回の計算結果もメモリから読み出す必要もある. 単純に必要なデータをメモリから読み出すとすると5ステップかかってしまう. 数十万回といった計算回数を行なうエンジンで1回の計算を行なうためのステップ数を削減できるならば, 計算速度を上げることができる.

例としてPEを2個用意した図3.6を用いて説明する. 2個のPEが格子点 $(i,j)$ と $(i+1,j)$ の計算を行なうとする. $V_{i,j}$ を計算するために $V_{i-1,j}, V_{i+1,j}, V_{i,j-1}, V_{i,j+1}$ が必要である. $V_{i,j-1}$ の値は $V_{i,j}$ を計算する1つ前に計算した結果であり, 値がレジスタに残っているため読み出す必要がない. $V_{i+1,j}$ は $(i+1,j)$ のPEで値を保持しているためメモリから読み出す必要はない. また,$V_{i,j}$ の前回計算結果は $V_{i,j-1}$ を計算したときにメモリから読み出してあるためこれも読み出す必要がない. このことからメモリからは $V_{i,j+1}, V_{i-1,j}$ の2つのデータ読み出しで計算ができる.

各PEの実現には, 内部にメモリを実現できるFPGAを採用する. PEの個数を増やすことで1個のメモリの容量を少なくすることができるため1個のPEとそれに伴うメモリとを1つのFPGA内に収めることができる. メモリをFPGA内に収めることで, メモリとの通信に必要であった入出力ピンを外部に出す必要がなくなり, 図3.7に示すように初期値の入力, クロック, 計算結果の入出力以外には外部ピンは不要となる. 外部ピン数を削減することができ, より安価なFPGAを利用することができる. また, 外部メモリとの通信がないため計算速度の点でも有利である.

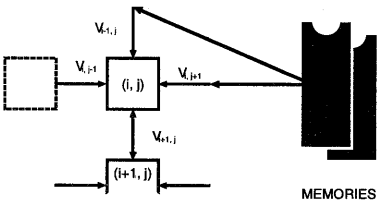


図 3.6: メモリ読みだしの削減

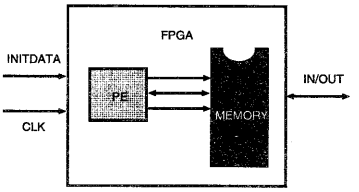


図 3.7: メモリの実現

## 3.4 小規模なエンジンの設計と評価

小規模なエンジンの設計を行ない, PEをどの程度用意すればよいかを検討した. FPGAは内部にメモリを実現できるXILINX社のXC4010 -PC84 -1 を使用した. また, メッシュの数, データ型, PEの数は以下の通りである.

- メッシュの数 : $16 \times 16$
- データ型 : 16ビット 整数型
- PEの数 : 2, 4, 8, 16個を用意

それぞれのエンジンについて設計を行ない動作周波数等の評価を行なった. その結果を表3.1に示す. 例えばPEが1個の場合, FPGA上に実現したプロセッサの動作周波数は30.3[MHz]であり, 計算時間は1.61[ms]であった. 同じの計算を行なうC言語のプログラムを実現し,SS5(170MHz 32MB)で実行した場合の計算時間は5.0[ms]である. エンジンの高速化率を

$$\text{高速化率} = \frac{\text{ソフトウェアでの計算時間}}{\text{エンジンでの計算時間}}$$

すると, 高速化率 $= 5.0/1.61 = 3.1$ 倍であった. すべての場合でエンジンを使用した方が高速であり, その中でもPEを縦1列(16個)用意したものは約28倍速くなった.

実際に取り扱う100×100のシミュレーションでも1列分用意できればかなりの速度が期待できる. また,FPGAを100個実装することは現実からかけはなれたものではないと考える.

表 3.1: 合成結果

| PEの数 | 周波数 (MHz) | 計算時間 (ms) | 高速化率 |
|---|---|---|---|
| 1 | 30.3 | 1.61 | 3.1 |
| 2 | 22.6 | 0.87 | 5.8 |
| 4 | 26.7 | 0.68 | 7.4 |
| 8 | 22.1 | 0.43 | 11.6 |
| 16 | 25.3 | 0.18 | 28.1 |

FPGA : XILINX 社 XC4010E -1 -PC84

小規模なエンジンの設計よりメッシュの格子点の1列分を用意すれば速度向上が期待できるという結論を得た. そこでデバイス・シミュレーションを行なう際に用いるデータである32ビット浮動小数点演算を実装した. メッシュの数は16 × 16, PE16個を用いた. 論理合成の結果は表3.2に示す. 同じ計算を行なうC言語のプログラムはSS5(170MHz, 32MB)上で計算時間は29[ms]であった. エンジンを1列用意することで約2倍の速度の向上がみられた. 実際のシミュレーションは100×100といった大規模なものでありこれに対しPEを1列用意することでプログラムと比較し20倍程度の速度の向上がみられると予想している. また, 開発システムやFPGAの改良は盛んに行なわれているので, 今後, 性能は更に改善されると思われる.

表 3.2: 合成結果

| 周波数 (MHz) | 計算時間 (ms) | 高速化率 |
|---|---|---|
| 28.8 | 14.0 | 2.1 |

FPGA : XILINX 社 XC4025E -2 -CB228

## 4 まとめ

バランス方程式に先だって, 予備実験としてポアソンの方程式を解く専用エンジンの設計を行なった. メッシュの格子点縦1列 $n$ 個にFPGAで実現したPEを配置し, これらを並列に動作させることで高速化を図った. また, FPGA内にメモリを格納することで外部ピンを節約でき, 部品数を減らすことができた.

今後の課題としては, ポアソンの方程式だけでなく, バランス方程式に対応するエンジンを設計し, 性能評価を行なうことである.

## 参考文献


[1] 冨澤 一隆 : "半導体デバイスシミュレーション," コロナ社 (1996).

[2] 倉田衛 : "デバイスシミュレーションボード," コンピュータエンジニアリング社, コンピュータシミュレーション, Vol. 3-3 (1992).

[3] S. D. Brown, R. J. Franeis, J. Rose, and Z. G. Vranesic "FIELD-PROGRAMMABLE GATE ARRAYS," Kluwer Academic Publ. (1992).

[4] 坂東和彦:"PARTHENONによるディバイスシミュレーション専用エンジンの設計と評価," 明治大学理工学研究科修士論文 (1996).

[5] 長谷川裕恭:"VHLDによるハードウェア設計入門," CQ出版社 (1995).