# 卒　業　論　文

題　　目

## モンテカルロ法を用いた<br>化学反応モデルの FPGA によるシミュレーション

年　　度

平成 17 年度

所　　属

慶應義塾大学

理工学部　情報工学科

指導教員

天野 英晴 教授

氏　　名

60026142

吉見 真聡

# 卒業論文要旨

| 学科 | 情報工学 | 学籍番号 | 60026142 | フリガナ 氏名 | ヨシミ マサト 吉見 真聡 |
|---|---|---|---|---|---|

(論文題名)

## モンテカルロ法を用いた 化学反応モデルの FPGA によるシミュレーション

(内容の要旨)


近年の分子生物学の急速な進展により，種々の生物の遺伝子配列や，細胞内の代謝系などに関する定量的データが蓄積されつつある．これらデータを用いて細胞や個体を計算機上でシミュレートし，実験を行うことなく様々な条件下における生物の分子レベルでの挙動を解明できるようになることが期待されている．

生物学実験は、倫理的な問題が絡むこと，個体の成長に時間を要することなどから，比較的古くから計算機を利用したシミュレーションについて研究が行われてきた．そのため近年では、そのいくつかが実用化され実際に利用されている．

しかし，計算機の処理能力は急速に向上しているにもかかわらず、それらシミュレーション実行に要する時間は，実時間の数倍から数万倍にも及び，計算時間に関わる問題は解決が困難である．細胞シミュレーションという問題は非常に不規則で複雑であるため，SMP やクラスタなどを用いた並列計算処理による高速化は適していない．

その一方で，分子動力学や流体力学といった分野では，専用計算機が性能や処理速度の面で大きな成果を挙げてきており，生物学の分野でも，ハードウェア処理による高速化への期待は大きい．しかし，生物のシミュレーションは様々なアルゴリズムが組み合わせて使用されるため，専用ハードウェアの設計を行うことは難しい．

そこで，近年活発化している FPGA などを用いた可変構造型ハードウェアによる計算システム (リコンフィギャラブルシステム) を応用することで解決を図る．回路を再構成することでアルゴリズムの柔軟性を確保し，問題を専用計算機のように直接ハードウェア処理することで高速化を実現する．

本研究では，生物学計算指向リコンフィギャラブルプラットフォームである ReCSiP をターゲットとして，Gillespie の開発した確率モデルシミュレーションアルゴリズムを実装し，シミュレーションによる評価を行った．その結果，Lotka 反応モデルにおいて AMD 社 AthlonXP2800+を用いたソフトウェアの実行時の 53.48 倍の高速化を実現した．


(内容の要旨は約25行程度で記入のこと)


慶應義塾大学理工学部

# 目 次

# 図目次

# 表 目 次

# 第1章

# 緒論

1980年代後半からの分子生物学の急速な進展により，種々の生物の遺伝子配列や，細胞内の代謝系などに関する定量的データが蓄積されつつある．これらのデータを用いて，細胞や個体を計算機上でシミュレートすることにより，実験を行なうことなしにさまざまな条件下における生物の分子レベルでの挙動を知ることができるようになることが期待されている．

実験と計算機シミュレーションの反復という研究手法の有効性は，既に他の多くの分野において有効性が実証されつつある．生物学の分野では，まだモデル化に必要なデータが決して充分とはいえず，シミュレーション自体はまだ研究途上の課題であるといえるが，生物学実験には常に倫理的な問題が絡むことや，個体の生長に時間を要することなどから，特に高等生物を対象にした実験は困難であり，シミュレーションへの期待はきわめて大きい．

したがって，計算機を利用してシミュレーションを行う，という研究は比較的古くから行われており，1983年に開発されたKinsim[1]に汎用シミュレータのはじまりを見ることができる．1980年代から1990年代にかけてGepasi[2], DBsolve[3]などに代表される，代謝系のモデル化・シミュレーション・パラメータ最適化を行うソフトウェアが開発されるようになり，1990年代後半になると計算機の性能向上や入手可能なデータの増加に伴って，細胞内の代謝反応すべてをシミュレーションするためのE-Cell[4], The Virtual Cell[5]といったシミュレータの開発が行われはじめている．

また，代謝系を微分方程式にモデル化して解析的にシミュレーションを行う解析モデルに対して，化学反応システムで発生する反応を確率を使って計算する確率モデルが1977年にGillespie[6]によって提案された．1990年代後半より，細胞の生長と分裂の挙動をシミュレーションするSTOCKS[7], 分子の立体構造や共有結合の規則変化をシミュレーションするSTOCHSIM[8]といった，確率モデルで代謝系へアプローチするシミュレータが開発されてきている．

しかし，これらのシミュレータを使用して細胞全体の代謝活動をシミュレーションするために要する時間は実時間の数倍から数万倍にも及んでおり，計算時間の問題は深刻である．その一方で分子動力学や流体力学といった分野では，専用計算機が性能や成果の面で大きな成果を挙げてきており，ハードウェア処理による高速化への期待は大きい．

生物のシミュレーションにおいてはさまざまなアルゴリズムが組み合わせて使用され，新しいアルゴリズムも次々に導入されるなど，専用ハードウェアの設計は難しいと考えられる．しかし，可変構造型ハードウェアによる計算システムは，問題を専用計算機のように直接ハードウェア処理することによって高い性能を発揮することが可能できる上に，回路をソフトウェア的に書き換える柔軟性を兼ね備えており，生物学分野への応用が有望であると考えられる．

本研究では，FPGAを搭載した生物学計算向けハードウェアReCSiP(Reconfigurable Cell Simulation Platform)[9]をターゲットとして，確率モデルによる化学反応モデルのシミュレーションアルゴリズムを実装し，その性能評価を行った．本論文の第2章では，化学反応システムに対する

確率モデルでのアプローチの方法と，近年開発が進められている細胞シミュレータについて述べ，第 3 章では細胞シミュレーションプラットフォーム ReCSiP の構成と機能について述べる．第 4 章では，ReCSiP をターゲットとした化学反応システムの確率モデルシミュレーションの一例として Lotka 反応モデルのシミュレーションを高速化する手法と実装について検討し，第 5 章でその性能評価の結果について述べ，第 6 章で結論と今後の展開について述べる．

# 第2章

# 背景

近年，生物学の分野では遺伝子ネットワーク，代謝経路などを同定するためのデータを高速に得るための高スループットな実験方法の開発が盛んになっており，生物に関する数値的なデータを得ることが容易になりつつある．90 年代に始まった各種ゲノムプロジェクトの成果が出始めており，2002 年にはヒト，イネ等の比較的遺伝資料の大きな生物の遺伝子の高精度解読の完了が宣言された．また，ゲノムと同様，様々な生物の代謝経路も研究されている．

さらに，これらの実験から得られた大量のデータを元に，計算機を用いて網羅的な解析や数理モデリングとシミュレーションを行う動きも本格化している．

この章では，細胞内代謝反応に代表される化学反応システムのシミュレーションの手法について，確率モデルによるアプローチである Gillespie のアルゴリズムについて述べる．次に，解析モデルと確率モデルについて，主なシミュレータの現状について述べる．

## 2.1 Gillespie のアルゴリズム

ある空間上の化学反応システムについて，その挙動を数学的に記述する 2 つの方法がある．

解析モデル (deterministic approach) は，システムの時間の変化を，微分方程式の組で決定されるものと見なす方法である．

また，確率モデル (stochastic approach) は，システムの時間の変化を，単一の微分差分方程式に支配されるランダムウォーク過程の一種と見なす方法である．確率モデルの計算はマスタ方程式と呼ばれる確率・統計的なアプローチから開発された手法であるが，マスタ方程式は数学的に解答することが困難である．しかし，マスタ方程式を直接的に扱うことなく，それと等価な数値計算を行う方法を用いることで，計算機を使った計算を行うことができる．

Gillespie の開発した “stochastic simulation algorithm” [6] (以下 Gillespie のアルゴリズム) は，この確率モデルを用いて化学反応システムの挙動を計算するアルゴリズムである．

Gillespie のアルゴリズムは次のような問題に解答することを目的として開発された．

> $N$ 種の化学物質が一様に分布する一定の体積 $V$ の中で，相互反応する $M$ 種類の化学反応経路があり，ある初期時間に各化学物質の分子数が与えられた場合，これらの分子数は時間の経過とともにどのように変化するか．

### 2.1.1 解析的モデルでの解法

この問題を解くために，従来，問題を常微分方程式で表現して解析する解析モデルでのアプローチが使用されてきた．

ある時刻 $t$ における $V$ 内の $i$ 番目の化学物質の数 (分子集合レベル．一般的には分子数など) が，時刻 $t$ を変数とする通常非線形の関数 $X_i(t)$ $(i = 1, \ldots, N)$ の式で記述されている化学反応システムを仮定する．このシステムで $M$ 種類の各化学反応が連続的に速度変化すると見なせるならば，以下のような一次の常微分方程式を構成することができる．

$$\frac{dX_i}{dt} = f_i(X_1, \cdots, X_N) \qquad (i = 1, \cdots, N) \tag{2.1}$$

右辺の関数 $f_i$ は，反応経路 $M$ の構造と反応速度定数によって決定される．これらの方程式は，反応速度方程式 (reaction-rate equation) と呼ばれる．$X_1(t), \ldots, X_N(t)$ の解を導くことが，各化学物質の時間変化になる．

このような反応速度方程式の解析的な解法は，通常単純なシステムで利用されることが多いため，計算機でこれらの方程式を数値的に計算することが必要になる．

この方法は，微分方程式で近似していることから，化学反応システムの時間変化が連続的かつ決定的であることを暗黙に仮定している．しかし，分子数のような分子集合レベルは離散的な整数値であるから，化学反応システムは時間的に非連続なプロセスである．さらに，発生する反応は決定的なものではない．もし，分子の挙動がなんらかの方程式に支配されるものと見なしたとしても，システム内のすべての分子の位置と速度を計算しない限りは，システムの挙動を計算することは不可能である．

式 2.1 のような方程式を解く方法は，いくつかの化学反応システムの解析には非常に良い精度を示している．しかし，解析モデルによるシミュレーションは，システムの時間挙動を微分方程式の組で記述するため，初期値の小さな違いが，シミュレーションの進行とともに指数関数的に拡大してしまう．また，遺伝子発現に代表されるような，ごく短い時間で小数の分子が急激に反応する場合には，反応発生にランダムな要素が含まれ，適切なシミュレーションができない．

生態系や微小な生化学システム，非線形なシステムの挙動に注目が集まっている現在においては，分子集合レベルを十分高精度に計算できる方法が必要になる．

Gillespie のアルゴリズムは，化学反応システムが離散的かつ確率論的な挙動を示す，という前提から，化学反応システムの挙動を計算する計算機解法である．

### 2.1.2 確率的モデルによる化学運動学の定式化のための物理的基礎

化学反応は，適当な 2 種類以上の分子が衝突したときに発生する．確率的方法は，熱平衡状態の分子反応システムで起こる分子の衝突を計算するものである．

#### 分子の衝突

ある体積 $V$ 内で，平衡状態にある 2 種類の気体分子 $S_1$ と $S_2$ の混合物からなるシステムがあるとする．単純化のため，$S_1$ 分子と $S_2$ 分子はそれぞれ半径 $r_1$，$r_2$ の球体であると考える．$S_1$ 分子と $S_2$ 分子の中心が重なったときに必ず反応が発生し，半径 $r_{12} = r_1 + r_2$ の分子になるとする (図 2.1).

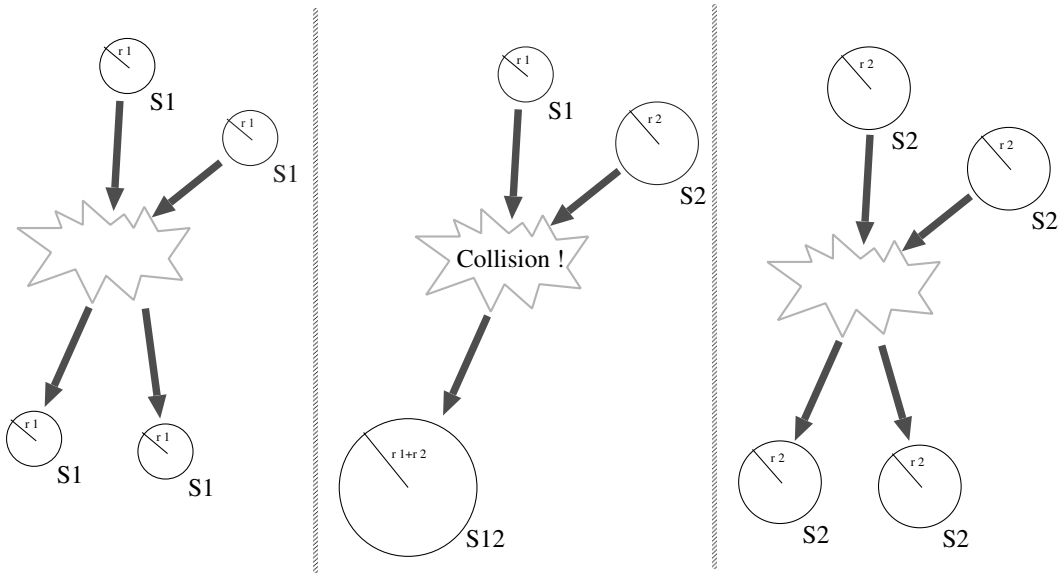



図 2.1: 分子の衝突

$V$ 内でこのような衝突が発生する頻度 (反応の速度) を計算する．

従来の反応速度を導出する方法を以下に述べる．まず，任意の分子の対から反応が起こる分子を選択する．そして，$S_2$ 分子に関して $S_1$ 分子の相対的な運動速度 $v_{12}$ を求める．さらに，微小時間 $\Delta t$ の間に，$S_1$ 分子が $S_2$ 分子に対して移動する分の体積 (Collision Volume : 衝突体積) を計算する．衝突体積 $\Delta V_{\mathrm{coll}}$ は以下の式 2.2 で求められる (図 2.2).

$$\Delta V_{\mathrm{coll}} = \pi r_{12}^2 \cdot v_{12} \Delta t \tag{2.2}$$

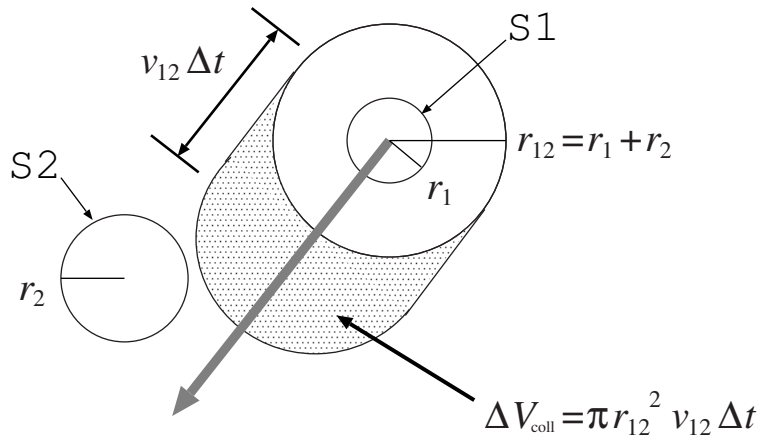



図 2.2: 衝突体積 $\Delta V_{coll}$

もし，時間 $t$ で $S_2$ 分子の中心が $\Delta V_{\mathrm{coll}}$ 内に存在していたとすると，2 つの分子は微小時間 $(t, t+\Delta t)$ で衝突する．

この方法は，微小時間 $\Delta t$ を小さくしていくことで，反応速度の精度は上がることになる．また，極限 $\Delta V_{\mathrm{coll}} \to 0$ なので，衝突体積中に存在する分子は 0 か 1 になってしまう．このように，微小時間で計算していく方法は，より厳密であるとは言い難い．

このことから，以下のように確率を用いた方法を考える．

システムは熱平衡状態であるから，分子は $V$ 内で一様に分布している．そのため，時刻 $t$ において任意の $S_2$ 分子の中心が $\Delta V_{\mathrm{coll}}$ にある確率は，単純に $\Delta V_{\mathrm{coll}}/V$ で与えられる．よって，$S_1$ 分子と $S_2$ 分子の反応速度分布を平均した場合，以下のような式 2.3 が得られる．これは，システム全体の体積に対する衝突体積の割合に等しい．

$$\overline{\left(\frac{V_{\mathtt{coll}}}{V}\right)} = \frac{\pi r_{12}^2 \overline{v_{12}} \Delta t}{V} = S_1\text{-}S_2 \text{ の分子が次の微小時間 } \Delta t \text{ で衝突する確率の平均} \tag{2.3}$$

マクスウェルの速度分布より，相対速度の平均 $\overline{v_{12}}$ は $(8kT/\pi m_{12})^{1/2}$ と等しい．$k$ はボルツマン定数，$T$ は絶対温度，$m_{12}$ は換算質量 $m_1 m_2/(m_1+m_2)$ である．上記の計算は $S_1$ 分子，$S_2$ 分子が $V$ 内にそれぞれ 1 個ずつある場合であるが，$S_1$ 分子の数として $X_1$，$S_2$ 分子の数として $X_2$ が与えられたとすると，式 2.3 は，式 2.4 のように表される．

$$\overline{\left(\frac{V_{\mathtt{coll}}}{V}\right)} = \frac{X_1 X_2 \pi r_{12}^2 \overline{v_{12}} dt}{V} = \text{微小時間 } (t, t+dt) \text{ で } S_1\text{-}S_2 \text{ の衝突が発生する確率} \tag{2.4}$$

式 2.4 により，反応発生数を確定的には計算できないが，有限時間内に $V$ 内で発生する衝突の確率を計算することができる．式 2.4 は決定的な反応速度過程の代わりに，確率的なマルコフ過程を構成しており，過去のシステムの挙動には依存せず，その時点での分子数と “単位時間あたりの衝突確率” によって，熱平衡状態の分子を特徴付けるものである．

#### 反応確率速度定数 — “The Stochastic Reaction Constant”

化学反応の “反応速度” は，“適当な分子の単位時間あたりの衝突確率 = 単位時間あたりの反応発生確率” と表現することができることを前節で述べた．

$V$ 内で $S_1$ 分子と $S_2$ 分子が式 2.5 のように反応すると仮定する．

$$R_1 : \quad S_1 + S_2 \to 2S_2 \tag{2.5}$$

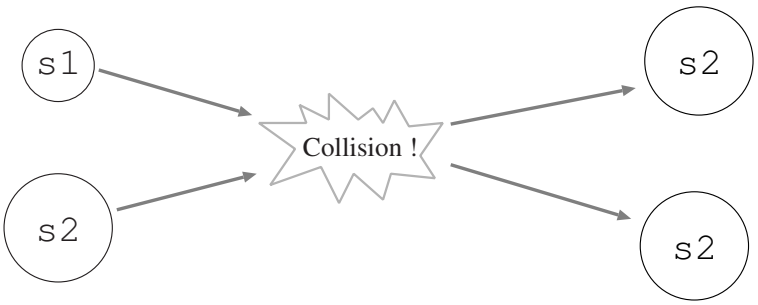



図 2.3: 仮定する反応

式 2.3 より，2 つの分子の物理的性質とシステムの温度だけに依存する定数 $c_1$ を設定する．

$$c_1 dt = \text{微小時間 } dt \text{ で特定の } S_1\text{-}S_2 \text{ 分子対が作用 (反応 } R_1 \text{ が発生) する確率} \tag{2.6}$$

もし，$V$ 内の時間 $t$ において，$S_1$ 分子の分子数が $X_1$，$S_2$ 分子の分子数が $X_2$ 存在するならば，以下の式が成立する．

$$X_1 X_2 c_1 dt = \text{反応 } R_1 \text{ が微小時間 } (t, t+dt) \text{ で } V \text{ 内のどこかで発生する確率} \tag{2.7}$$

式 2.7 を一般化して，熱平衡状態にある体積 $V$ 内の $N$ 種類の $S_i$ 分子が $X_i$ の数だけ存在する混合物について，$M$ 種類の化学反応 $R_\mu$ が相互作用すると仮定すると，分子の物理的性質とシステム温度に依存する式 2.8 で表される $M$ 種類の定数 $c_\mu$ を仮定できる．

$$c_\mu dt = \text{反応 } R_\mu \text{ に関わる分子が無限微小時間 dt で衝突する確率の平均} \tag{2.8}$$

式 2.8 における “平均” とは，その反応 $R_\mu$ に関係する分子の組み合わせの総数に $c_\mu$ を掛けた値が，反応 $R_\mu$ が次の微小時間 $(t, t+dt)$ で $V$ 内のどこかで発生する確率になることを意味している．

#### 反応速度定数と反応確率速度定数の関係

反応確率速度定数 $c_\mu$ は反応速度定数 $k_\mu$ と密接に関係している．式 2.5 のようなある反応 $R_1$ について，以下のような関係がある．

$$k_1 = \frac{V c_1 \langle X_1 X_2 \rangle}{\langle X_1 \rangle \langle X_2 \rangle} \qquad (\langle x \rangle \text{ は } x \text{ の平均値}) \tag{2.9}$$

右辺について，$\langle X_1 X_2 \rangle = \langle X_1 \rangle \langle X_2 \rangle$ と見なせるので，

$$k_1 = V c_1 \tag{2.10}$$

もし反応 $R_1$ が 2 つではなく 3 つの反応分子を持つならば，$V$ の代わりに $V^2$ を使用する．また，もし反応 $R_1$ がただ一つの反応分子を持つ (isomerization : 異性化) ならば，$V = 1$ になる．

次に，反応 $R_1$ の逆反応を以下の式 2.11 から考える．

$$R_2 : \quad 2S_1 \rightarrow S_1 + S_2 \tag{2.11}$$

式 (2.5) から，定数 $c_2$ を決定する ($S_1$ の分子の特定の組が微小時間で衝突し，$R_2$ の反応を発生する確率の平均)．$V$ 内の $S_1$ 分子の組み合わせの数は $X_1(X_1 - 1)/2!$ になる．

$$k_2 = \frac{V c_2 \langle \frac{X_1(X_1-1)}{2!} \rangle}{\langle X_1 \rangle \langle X_1 \rangle} = \frac{V c_2}{2} \tag{2.12}$$

#### 確率的モデルの時間計算

$M$ 種類の相互反応をする $N$ 種類の分子からなるシステムの時間経過を計算する．これは基礎的な仮説である式 2.6 から求めることになる．

### マスター方程式

確率モデルの化学反応システムの時間経過計算は，システムのマスタ方程式を設定し，それを解くことで求められる．マスタ方程式を数学的に解答することは困難で，シミュレーションでは後に述べるマスタ方程式を解くことと同義の処理を行ってシステムの時間経過を計算する．

マスタ方程式の最も重要な要素は主要確率関数 (grand probability function) である．化学反応システムにおいて主要確率関数は式 2.13 で表される．

$$\begin{aligned} P(X_1, X_2, \cdots, X_N; t) \equiv & V \text{ 内の時刻 } t \text{ で，} S_1 \text{ 分子が } X_1 \text{ 存在し，} \\ & \text{かつ } S_2 \text{ 分子が } X_2 \text{ 存在し} \cdots, \\ & \text{かつ } S_N \text{ 分子が } X_N \text{ 存在する確率} \end{aligned} \tag{2.13}$$

式 2.13 は，時刻 $t$ のシステムの状態確率 (stochastic state) を表している．たとえば，$X_i$ に関する $k$ 次の $P$ のモーメント ($k$ 乗平均) は，式 2.14 で与えられる．

$$\begin{aligned} X_i^{(k)}(t) \equiv & \sum_{X_1=0}^{\infty} \cdots \sum_{X_N=0}^{\infty} X_i^k P(X_1, \cdots, X_N; t) \\ & (i = 1, \cdots, N; k = 0, 1, 2, \cdots) \end{aligned} \tag{2.14}$$

$X_i$ に関する $k$ 次の $P$ のモーメントは，“時刻 $t$ で $V$ にある分子数 $X_i$ の平均の $k$ 乗” である．ここでの “平均” とは，2.6 で定義された確率モデルでの平均反応発生確率を使い，同じ初期値 $(X_\nu)$ で，時間 0 から $t$ まで非常に多くのシミュレーションを実行を繰り返したときの $X_i$ の平均のこと意味している．

毎回のシミュレーションで $S_i$ 分子の分子数 $X_i$ は異なるが，これらの値の $k$ 次の累乗の平均値は，シミュレーションを非常に多く実行することで，$X_i^{(k)}$ に収束していく．$k = 1, k = 2$ で表される 1 次モーメント，2 次モーメントは特に有用である．$k = 1$ での $X_i^{(k)}$ は時間 $t$ で $V$ 内に存在する $S_i$ 分子の数の平均を示しており，$k = 2$ では式 2.15 を計算することで，母平均まわりの 2 次モーメント (分散) を求めることができる．

$$\begin{aligned} X_i^{(k)}(t) \equiv & \sum_{X_1=0}^{\infty} \cdots \sum_{X_N=0}^{\infty} X_i^k P(X_1, \cdots, X_N; t) \\ & (i = 1, \cdots, N; k = 0, 1, 2, \cdots) \end{aligned} \tag{2.15}$$

解析的な反応速度の式 2.1 で現れている $X_i(t)$ は，1 次のモーメント $X_i^{(1)}$ を近似しているが，完全な等式になることは少ない．

マスタ方程式は $P(X_1, \cdots, X_N; t)$ で表される時間経過の関数である．$P(X_1, \cdots, X_N; t)$ は，システムが時刻 $t + dt$ で状態 $(X_1, \cdots, X_N)$ になる $1 + M$ 通りの方法があり，それぞれの確率の合計である．この $P$ は，式 2.6 から確率の加法と乗法を使用して求められる．

$$P(X_1, \cdots, X_N; t + dt) = P(X_1, \cdots, X_N; t) \left[ 1 - \sum_{\mu=1}^{M} a_\mu dt \right] + \sum_{\mu=1}^{M} B_\mu dt \tag{2.16}$$

ここで，ある数量 $a_\mu$ を定義する．

$$\begin{aligned} a_\mu dt \equiv & c_\mu dt \times \{ \text{状態 } (X_1, \cdots, X_X) \text{ で反応 } R_\mu \text{ が起こる分子の組み合わせ数} \} \\ = & \text{時刻 } t \text{ で状態 } (X_1, \cdots, X_N) \text{ のシステムで，} \\ & (t, t + dt) \text{ において } R_\mu \text{ 反応が発生する確率} \end{aligned} \tag{2.17}$$

式 2.16 の第 1 項は，システムが時刻 $t$ において状態 $(X_1, \cdots, X_N)$ になり，時刻 $t + dt$ においてその状態 (反応が起こらない) のままでいる確率である．第 2 項の $B_\mu dt$ は，時刻 $t$ でシステムが状態 $(X_1, \cdots, X_N)$ になり，時刻 $t + dt$ で $R_\mu$ の反応を受ける確率である．

式 2.16 からマスタ方程式の導出をすることができる．

$$\frac{\partial}{\partial t} P(X_1, \cdots, X_N; t) = \sum_{\mu=1}^{M} \Big[ B_\mu - a_\mu(X_1, \cdots, X_N; t) \Big] \tag{2.18}$$

特別な場合を除いて，マスタ方程式の記述は簡潔であるが，それを解くことは困難であることが多い．さらに，反応速度方程式とは異なり，独立変数の数と性質が原因で，計算機を使って数値的に解を導くことも容易ではない．全ての反応 $R_\mu$ が単純な単分子反応でなければ，モーメントを導出するための式は常に高次のモーメントを含んでいるため，特定の $X_i$ や $\Delta_i(t)$ のモーメントの時間変化を方程式で表して計算しようとすることは，無限大の時間が掛かってしまう．マスタ方程式は正確で簡潔に表現することはできるが，実際的な数値計算を行う上では有用ではない．そのため，確率モデルでは，結果的にマスタ方程式と同じ解答を得られる方法を用いて計算機でのシミュレーションが可能な方法を用いる．

#### 反応確率分布関数

化学反応システムについて，確率的方法を用いた場合に，時間経過を計算する方法について述べる．

時刻 $t$ で状態 $(X_1, \cdots, X_N)$ のシステムがあるとすると，このシステムの時間経過を計算するには，いつ，どの反応が起こるのかを明らかにする必要がある．

そのため，以下の式 2.19 で定義される関数 $P(\tau, \mu)$ を導入する．

$$P(\tau, \mu) \equiv V \text{ 内の状態 } (X_1, \cdots, X_N) \text{ で，微小時間 } (t + \tau, t + \tau + d\tau) \text{ の間に } R_\mu \text{ の反応が起こる確率} \tag{2.19}$$

関数 $P(\tau, \mu)$ は反応確率分布関数と呼ばれる．$\tau$ は次の反応が起こるまでの時間，$\mu$ は発生する反応の種類を示している．

この化学反応システムにおいて発生しうる反応 $R_\mu$ について，式 2.20 で定義される関数 $h$ を計算する．

$$h_\mu \equiv \text{状態 } (X_1, \cdots, X_N) \text{ において，} R_\mu \text{に関係する分子の組み合わせ数} \qquad (\mu = 1, \cdots, M) \tag{2.20}$$

もし反応 $R_\mu$ が $S_1 + S_2 \to S$ という形式であるならば，$h_\mu = X_1 X_2$ となり，$R_\mu$ が $2S_1 \to S$ という形式ならば，$h_\mu = \frac{1}{2} X_1(X_1 - 1)$ になる．一般的に，$h_\mu$ は変数 $X_1, \cdots, X_N$ の複合関数である．

状態 $(X_1, \cdots, X_N)$ において，時刻 $(t, t + dt)$ で反応が起こらない確率を $P_0(t)$ とする．また，$R_\mu$ が時刻 $(t + \tau, t + \tau + d\tau)$ で発生する確率を $a_\mu$ とすると，$a_\mu$ は式 2.21 で表される．

$$a_\mu d\tau \equiv h_\mu c_\mu d\tau \tag{2.21}$$

よって，$P(\tau, \mu)$ は以下のように定義される．

$$P(\tau, \mu) d\tau = P_0(\tau) \cdot a_\mu d\tau \tag{2.22a}$$

右辺の $P_0(\tau)$ の導出について，$[1-\sum_\nu a_\nu d\tau']$ が状態 $(X_1,\cdots,X_N)$ の時間 $d\tau'$ で反応が発生しない確率である事を利用し，以下の式から $P_0(\tau)$ を得る．

$$P_0(\tau'+d\tau') = P_0(\tau')\cdot\left[1-\sum_{\nu=1}^{M} a_\nu d\tau'\right] \tag{2.22b}$$

$$P_0(\tau) = \exp\left[-\sum_{\nu=1}^{M} a_\nu d\tau'\right] \tag{2.22c}$$

式 2.22c を式 2.22a に代入することで，反応確率密度関数 $P(\tau,\mu)$ を計算することができる．

$$P(\tau,\mu) = \begin{cases} a_\mu \exp(-a_0\tau) & \text{if } 0\leqslant\tau<\infty \text{ and } \mu=1,\cdots,M \\ 0 & \text{otherwise} \end{cases} \tag{2.23}$$

### 2.1.3 確率的シミュレーションアルゴリズム

本節では，化学反応システムにおいて，いつ，どの反応が発生するのかを計算する方法を述べる．

これは，“単位間隔の一様な乱数” から反応確率密度関数 $P(\tau,\mu)$ の乱数を作り出して $\tau,\mu$ を計算することで明らかになる．$(0,1)$ の一様乱数 $r_1$，$r_2$ から，以下の式を用いて $\tau$ と $\mu$ を生成する．

$$\tau = \tfrac{1}{a_0}\ln\left(\tfrac{1}{r_1}\right) \tag{2.24a}$$

$$\sum_{\nu=1}^{\mu-1} a_\nu < r_2 a_0 \leqslant \sum_{\nu=1}^{\mu} a_\nu \qquad (\mu : \text{Integer}) \tag{2.24b}$$

この 2 式より $P_2(\mu) = a_\mu/a_0$ に従う整数の乱数 $\mu$ と，確率密度関数 $P_1(\tau) = a_0\exp(-a_0\tau)$ に従う乱数 $\tau$ が生成され，$P_1(\tau)\cdot P_2(\mu) = P(\tau,\mu)$ を求めることができる．

シミュレーションは Step 1. から Step 3. の繰り返しで進行する (図 2.1.3).

#### Step 0. 初期化

$M$ 種類の反応定数 $c_1,\cdots,c_M$ と，$N$ 個の初期分子数 $X_1,\cdots X_N$ に初期値を格納する．時刻 $t$ と反応数カウンタ $n$ を 0 に初期化する．乱数生成器を初期化する．

#### Step 1.

この時点での分子の数ごとに，$M$ つの値 $a_1 = h_1c_1,\cdots,a_M = h_Mc_M$ を計算し，$a_0$ に $a_\nu$ の合計値を代入する．

#### Step 2.

$(0,1)$ の一様乱数 $r_1$ と $r_2$ を生成し，$\tau$ と $\mu$ を計算する．

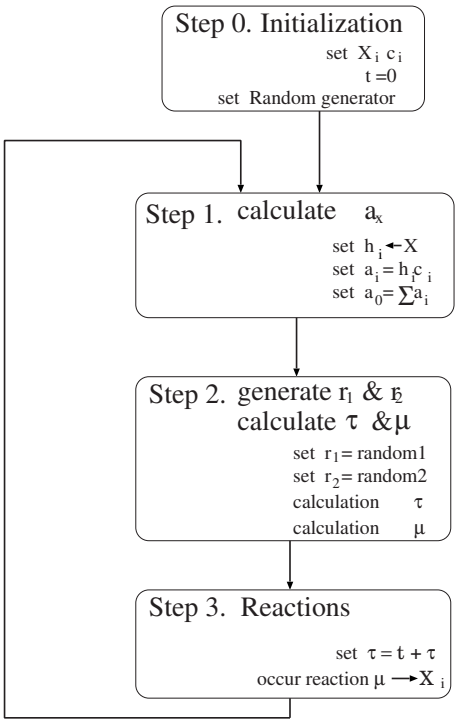



図 2.4: 確率モデルシミュレーションアルゴリズム

**Step 3.**

$\tau$ によって $t$ を増加する．反応 $R_\mu$ が影響を与える分子の数を調節する (例えば，$R_\mu$ が式 2.5 の反応だったならば，$X_2$ が 1 増加し，$X_1$ が 1 減少する)．反応数カウンタ $n$ をインクリメントする．これを Step 1. に戻って繰り返す．

Step 1. から Step 3. の繰り返しの中のどこか，あるいは任意の $t$ または $n$ の間隔において，$(X_1, \cdots, X_N, t)$ の値を出力する．また，$t$ あるいは $n$ が何らかの値に到達したときや，$a_0$ が 0 になったときには計算を終了する．

### 2.1.4 確率モデルの利点と制限

確率的モデルのシミュレーションアルゴリズムの利点と制限を述べる．

このシミュレーションアルゴリズムは，基礎的な仮定 (式 2.6) の形式で記述された化学反応システムについて，正確なシミュレーションが可能である．また，確率モデルのシミュレーションアルゴリズムは，従来の微分方程式の数値解法のように，微小時間 $\Delta t$ ごとに計算を繰り返す方法をとらず，“次の反応までの時間” を計算するため，分子数が急激に変化するようなシステムにも有効である．このシミュレーションアルゴリズムを計算機上で実行した場合，メモリスペースの要求量が少ないことも利点の一つとして挙げられる．$N$ 種類の分子と $M$ 種類の反応では，$N$ 種類の

分子数と $M$ 個の $c_\nu$，$M+1$ 個の $a_\nu$ の変数を格納するメモリスペースが確保できれば良い．このことは，FPGA 上に確率モデルシミュレーションアルゴリズムを実装する上でも大きな利点となる．

また，シミュレーションの過程や結果を読み取ることが容易であることも利点のひとつである．シミュレーションサイクルごとに分子数が数値的に評価できるため，平均，分散，相関などを計算することが容易である．

確率モデルのシミュレーションアルゴリズムは，反応数に比例した計算時間が掛かる．そのため，シミュレーションできる分子の種類，数，反応数は適切な数を指定し，計算時間を適切に抑えなければならない．シミュレーションが可能なシステムは分子が一様分布している状態に限られ，分子数に偏りがある場合は，システムの大きさを変更するなどして対応する必要がある．また，このシミュレーションアルゴリズムから信頼できる結果を得るためには，信頼性のある乱数生成器を使用しなければならない．この信頼性に関する標準を得ることは非常に困難である．さらに，良い統計精度を得るためには，複数回のシミュレーションの評価を取る必要があり，その実行回数は計算時間などのコストとのトレードオフになる．

### 2.1.5 Lotka モデル

化学反応の確率モデルシミュレーションを開発した Gillespie は，確率モデルシミュレーションを適用した例として，4 つのモデルについて言及している．それは，不可逆異性化反応，Lotka 反応，Blusselator，Oregonator の 4 つである．不可逆異性化反応は逆反応の起こらない不可逆変化する分子の反応であり，Gillespie はこの反応のシミュレーションを用いて確率モデルの具体的な検証を行っている．その他の 3 つのモデルはより複雑なモデルとして挙げ，検証している．不可逆異性化反応は，後に述べる Lotka モデルの一部として使われる基本的なモデルである (式 2.25c のみで記述されるモデルが不可逆異性化反応である)．

今回の実装では，Gillespie の述べた “より複雑なモデル” から，Lotka 反応モデルを ReCSiP 上に実装して高速化を図ることを考える．

Lotka 反応モデルは，1920 年に Lotka が以下のような自触媒反応を観察したことから研究が始まった．

$$X_1 + X_2 \xrightarrow{c_1} 2X_2 \tag{2.25a}$$

$$X_2 + X_3 \xrightarrow{c_2} 2X_3 \tag{2.25b}$$

$$X_3 \xrightarrow{c_3} Z \tag{2.25c}$$

このモデルは，Volterra が開発した捕食者-被食者関係に対応する生態系の数学的モデルから研究されているモデルである．Volterra は上記の反応を以下のような微分方程式で記述し，これに対応する反応速度方程式の使用法について研究した．

$$\frac{dX_1}{dt} = c_1 X_1 X_2 - c_2 X_2 X_3 \tag{2.26a}$$

$$\frac{dX_3}{dt} = c_2 X_2 X_3 - c_3 X_3 \tag{2.26b}$$

式 2.25b は捕食種 $X_3$ が被食種 $X_2$ を食べて自分を再生産する様子を表現している．また，式 2.25a は $X_2$ が食料 $X_1$ を食べて自分を再生産する様子を記述している．ここでは $X_1$ はただ消耗されるだけで，増加しないものと仮定している．異性化式 2.25c は $X_3$ の死を表現している．

定常状態は $dX_2/dt = dX_3/dt = 0$ の条件を満たしているシステム状態であることは明らかである．

Lotka 反応モデルは化学反応の単純なモデルの一例であるが，これは以下のような単純なルールに支配される生態系を考えることでどのような反応であるかを容易に理解することができる．ここでの Lotka 反応モデルのシミュレーションに際しては，前述の式 2.25 に加え，$X_2$ の死，および $X_1$ は反応によって減少しない，という条件を追加している．

- 平原
  - 一定の広さの平原があり，草が繁茂している場所が点在している
  - 平原の外に出ることはできず，また平原の中に何かが入ってくることもない
- ウサギ
  - ウサギは，草を見つけるとそれを食べ，繁殖して 2 匹に増える
  - オオカミに食べられたり，一定の確率で自然死する
- オオカミ
  - オオカミは，ウサギを見つけるとそれを食べ，繁殖して 2 匹に増える
  - オオカミは，一定の確率で自然死する

この平原にウサギ 1000 匹，オオカミ 1000 匹を一様に分散させて投入したところ，時間の変化とともにそれぞれの数はどのように変化するか，を計算するモデルが Lotka モデルの単純な形であると考えることができる．一般的に，草がウサギに食べられても減少しないという条件のもとで，適切な $c_\mu$ を設定することで，ウサギとオオカミの数は震動することが知られている．

## 2.2 主な細胞シミュレータ

現在までに開発された，あるいは開発中の細胞シミュレータについての調査結果を述べる．まず，現在では解析モデルでのシミュレータが多く開発されているため，代表的な解析モデル細胞シミュレータである Virtual Cell について，および確率モデルのシミュレータである STOCKS, STOCHSIM について述べる．

### 2.2.1 Virtial Cell

#### 概要

Schaff らが開発した Virtual Cell[5] は単一の細胞の構造とその内部の反応をシミュレーションする汎用的なフレームワークである．Virtual Cell は細胞のメカニズムをテストする総合的なシステムを提供するとともに，細胞内の運動と分布を総合的に扱うフレームワークを提供する．

Virutual Cell は細胞内で発生する個々の反応について，生化学的なデータ，あるいは電気信号などの電磁気学的なデータを微小な画素で表現し，細胞の時間変化を画像で表示する．Virtual Cell

はあらゆる分子のメカニズムを速度方程式や膜輸送の記述で定義し，個々の反応を計算機上に集約させる目的で開発が続けられている．

### Virtual Cell が実行する計算

Virutual Cell は単一の細胞を微小な立方体 (element と呼ばれる) に分割し，微小時間ごとにそれぞれの element ごとに解析的な計算を行う，という方法を取っている．

また，Virtual Cell には以下の特徴がある．

- シミュレーションする問題は偏微分方程式で記述され，変数はシステムの状態を表し，次の微小時間のシステムへの入力にもなる．

  1. 拡散/電位勾配による分子の流れ
  2. 荷電した分子の分布による電束密度
  3. cable 理論によって求められる電流密度
  4. 系から生成される物質
  5. 膜を介した物質の輸送

  上記の項目 1〜4 は全ての element について，項目 5 は膜 (elemet を構成する物質がそれ以外と大きく異なっている部分) のみで定義される．

- 基本的な物理法則の計算式が実装され，シミュレーションごとに再定義する必要が無い
- 変数は element の位置と状態 (分子濃度と電位) の関数で表される

項目 1 の分子の流れ $\overrightarrow{J_i}$ は細胞の挙動の解析に用いられる．$\overrightarrow{J_i}$ は NernstPlank 方程式によって，式 2.27 記述される．

$$\overrightarrow{J_i} = -D_i \nabla C_i - z_i \mu_i C_i \nabla V_i \qquad \mu_i = \frac{D_i F}{RT} \tag{2.27}$$

| | | | | | |
|---|---|---|---|---|---|
| $C_i$ | 濃度 | $D_i$ | 拡散係数 | $x_i$ | イオン価数 |
| $\mu_i$ | 移動度 | $V$ | ポテンシャル | $F$ | ファラデー定数 |
| $R$ | 気体定数 | $T$ | 絶対温度 | | |

式 2.27 の右辺第 1 項は拡散泳動を，第 2 項は電場の流れを示している．第 2 項は急速に反応が起こる場合や，高解像度の空間を解析する場合に使われる．

式 2.27 から，ある微小体積内における分子 $i$ の濃度の時間変化は式 2.28 で表される．

$$\frac{\delta C_i}{\delta t} = -\nabla \cdot \overrightarrow{J_i} + S_i(C_1 \cdot C_2 \cdots C_N) \tag{2.28}$$

$S$ は分子の生成・消費に関わる複数の反応から総合的に決定される反応速度である．

単一の反応では，$S_i$ は速度方程式によって与えられる．

$$A + B \underset{k_{-1}}{\overset{k_1}{\rightleftarrows}} C \tag{2.29a}$$

$$S_C = k_1[A][B] - k_{-1}[C] \tag{2.29b}$$

分子は複数の反応で生成・消費されるので，$S_i$ の値は個々の反応速度の合計を与える．

微小時間ごとに微小体積上に定義される濃度は，分子が element 内で生成・消費される量と，element 間で流動する量との合計である．

項目 2 と 3 は，電子を持つ分子の分布と，移動度ベースの電位を記述する高解像度解析用の冗長的な要素である．項目 5 は膜を通過する分子を制御する法則である．2 つの隣接 element で性質が異なっている部分を膜と定義し，膜内外の分子について，双方向移動が記述されている．

### シミュレーションの流れ

Virutual Cell のシミュレーションの目的は，なんらかの外乱を与えた後の恒常性維持機構の挙動を調べることにある．そのためには濃度や電位などについて適切な初期条件を与える必要がある．定常状態を初期値とすることで，外乱に対する挙動を調べやすくすることができる．初期条件は実験的に概算されているものといないものがある．適当な値を定常状態になるまでシミュレーションすることで定常解を求めることができるが，初期微動を最小化する方法を使用すれば，非線形最適化問題として解くことができるようになる．

Virtual Cell が行うシミュレーションは以下のような流れで進行する．

まず，実験から得られた 3 次元データから，以下の項目の複合で細胞の形状を分析幾何学的に近似する．

- 球体，円柱のような平面プリミティブ
- フラクタルソリッドのような数学的立体モデル
- ポリゴン面 (CAD モデル)

シミュレーションは化学的な現象を分子，次元ごとの element 数，element のサイズ，シミュレーション時間などを定義して柔軟に描き出すことが可能である．分子の運動は，element 内の濃度の変化と，element の境界を横切る分子の流れの合計として描写される．分子の運動は，分子の拡散，イオン化傾向，電位，化学反応，分子の空間分布などの関係を表している．膜は，異なる性質を持つ element の境界で表現される．

シミュレーションの出力になるものは，各 element の時間ごとの分子の濃度，電位，膜上に存在している分子の状態や密度などである．

膜反応モデルと element 内のモデルはモデルの変数や各種パラメータの関数として記述された反応速度のセットで構成されており，これらはデータベースに格納され，必要に応じてモデルオブジェクトライブラリを介して呼び出される．この DB 化により，ユーザは毎回数値モデルに手を加える必要から解放される．

### finite-volume formalism

Virtual Cell は finite-volume formalism と呼ばれる計算方法を利用している．

シミュレーションは計算対象空間を一定サイズの微小な立方体 (element) に区切って計算が行われる．

式 2.30 で分子の流れを表現する (2.5).

$$-\oint_S \vec{J_i} \cdot \vec{dS} = Area_{boundary} \cdot [J_X|_{X+} + J_X|_{X-} + J_Y|_{Y+} + J_Y|_{Y-} + J_Z|_{Z+} + J_Z|_{Z-}]_i \tag{2.30}$$

隣接する element が膜だった場合やシミュレーションの端だったりした場合は，その方向の面で膜輸送の処理など特別な処理を行う．

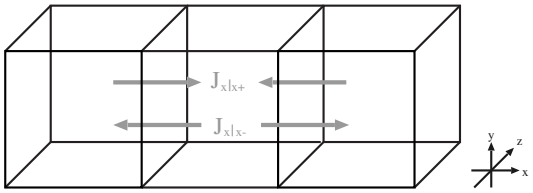



図 2.5: finite-volume formalism

式 2.31 のように，反応速度は微小時間で線形な部分と定数の合計であると近似して計算を行う．

$$S_A \approx -f_{\texttt{A\_linear}}([A],[B],\cdots)\cdot[A] + f_{\texttt{A\_cons tan t}}([C],[D],\cdots) \tag{2.31}$$

未知の特性については，定数や適当なモデルで近似する．

膜反応は，膜上のチャネルが選択的に膜内外の分子を透過させるような処理を行う．チャネルは複数の状態を持ち，膜内外の電位差や分子の濃度によって制御される．状態遷移を化学量論的に記述することによって，膜上の分子密度を記述するだけで膜反応シミュレーションを行えるようになる．

### 実行例 — Calcium wave in neural cell

Virtual Cell の実践テスト例として，カルシウムの生化学レベル，電気生理学レベルでの相互作用のメカニズムが提示されている．カルシウムは細胞内で信号を伝達するセカンドメッセンジャーであり，その機嫌と規則と役割を解析することは大きな研究課題である．

実際の神経細胞におけるカルシウムを媒介とした信号伝達は以下の手順で進行する．

1. イノシトール三リン酸 ($IP_3$) が神経細胞内で高濃度になる
2. 小胞体が $IP_3$ に刺激され，$Ca^{2+}$ を放出する

Virtual Cell で $IP_3$ と $IP_3R$($IP_3$ 受容体) の運動を視覚化する (図 2.6)．これは生物実験的に干渉することが難しい部分である．図 2.6 の上方の神経細胞の画像はカルシウムの濃度を，下方は $IP_3$ の濃度の分布を表している．このような図は任意のシミュレーション時間間隔で出力され，動画として確認することも可能である．このシミュレーションで実験で見られるものと比較して，非常に類似している $Ca^{2+}$ の動きのパターンと速度が得られた．

### Virtual Cell の計算時間

図 2.6 に示された神経細胞のシミュレーションは，反応のモデルと分子の数を制限して適度な 2 次元セルモデルに近似して計算が行われている．計算は 27000 の element と 3 つの非線形偏微分方程式，7 つの非線形常微分方程式から構成される．評価は Silicon Graphics 社の R8000MIPS CPU(300Mflops) のワークステーションを使用した．2 秒間のシミュレーションに 2 日を要している．

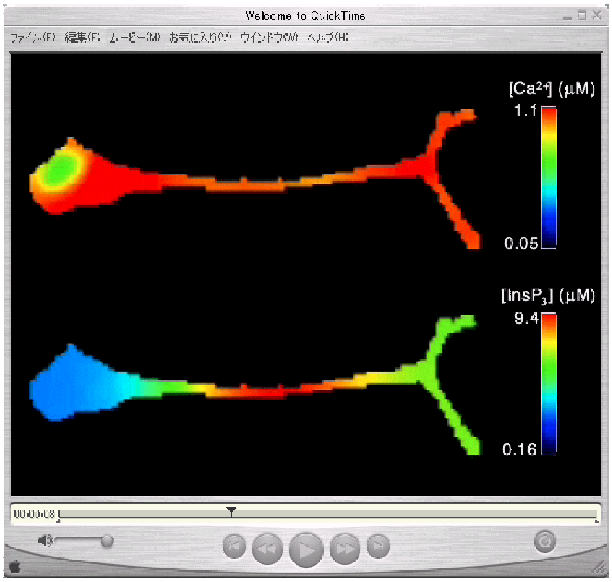



図 2.6: Virtual Cell による神経細胞内のカルシウム濃度と信号の変化

### Virtual Cell の展開

Virtual Cell は現在バージョン 3.1 であり，より多様なモデルに対応し続けている．また，シミュレーションしたデータからリポジトリを構築し，World Wide Web で共有されている．

Virtual Cell は Japa applet で記述され，プロジェクトの Web ページから Virtual Cell を使用することが可能になっている(注 1)．

## 2.2.2 STOCKS

Kierzek は原核生物の遺伝子発現を解析するソフトウェア STOCKS(STOChastic Kinetic Simulation) を開発した．STOCKS は標準 C++で記述されたソフトウェアプログラムである(注 2) STOCKS は化学反応系を Gillespie のアルゴリズムを利用して解析する．

### STOCKS の特徴

STOCKS は Gillespie が提案していた計算方法に加え，細胞内代謝シミュレーションを可能とするために主に以下の 2 つの機能が実装されている．

- **体積増大および細胞分裂に対応**

  STOCKS では，細胞を化学反応システムと見なしてシミュレーションする場合に，シミュレーション中に細胞が生長する場合や，細胞分裂が起こる場合に対応した実装がなされている．

  この機能は以下の項目が前提として仮定されている．

(注 1)http://www.nrcam.uchc.edu/
(注 2)2004 年現在バージョン 1.02．GNU GPL に基づいて公開 http://poczta.ibb.waw.pl/stocks/

- 細胞は一世代で体積が 2 倍になる
- 細胞分裂が発生すると体積は初期値に戻る

これは，Gillespie のアルゴリズムにおいてシステムの体積 $V$ の代わりにシミュレーション時間 $t$ を変数とする関数 $V(t)$ を使うことで実現している．

$V(t)$ は式 2.32 で表される式である．$T$ は細胞の一世代の時間である．式 2.32 により，細胞の体積が細胞の一世代の時間で線形に増大していく反応を表現することができる．

$$V(t) = 1 + \frac{t}{T} \tag{2.32}$$

具体的なシミュレーション方法は，毎サイクルの時刻 $t$ から $V(t)$ を再計算し，$c_\mu$ の値を $V(t)$ の値を使って更新していくことで実現する．$t = T$ すなわち $V(t) = 2$ になったとき，細胞の一世代が終わったことになり，細胞分裂が発生する．

細胞分裂が発生すると，体積および $c_\mu$ の値が初期値に戻り，そのときに存在する各分子の数が 2 で除算される．DNA エレメントをモデル化する分子数は除算の前に 2 倍しておき，分裂後も同じ数を保持するようになっている．

このような線形の体積変化で細胞生長，細胞分裂を表現する方法は，原核生物のシステムを対象としたシミュレーションでは妥当な結果が得られている．

- **ランダムプール**

  任意の数の分子が存在する “ランダムプール” を用意しておき，他から小さな影響を大量に受けて数が変動する分子がある場合に，分子の数を変動させる機能が追加されている．STOCKS では，RNA ポリメラーゼのモデリングにこのランダムプールの機能を使用している．

  RNA ポリメラーゼのランダムプールは以下のような方法が実装されている．

  - ランダムプール内の分子は，分子数がサイクルごとに毎回変更される
  - 分子数は $h_\mu$，$a_\mu$ を計算する前に，そのときの分子数の平均と標準偏差を用いて正規分布する乱数を得，その値が設定される
  - 体積の増加とともにプール内の分子数の平均値は増大するが，濃度は一定のままになる

  ランダムプールは RNA ポリメラーゼ以外の分子にも適用でき，また反応分子数の明示的なモデル化が難しい大規模プロセスのバッファにもなると Kierzek は予測している．

  ただし，ランダムプール機能は，多くの場合に妥当な結果が得られているとは言え，数学的に正しさが証明されているわけではないので，ランダムプール機能を使用する場合には注意が必要になる．

### 計算コストへの対策

STOCKS は計算コストが大きく，実行時間は数時間から数日に及ぶため，UNIX オペレーティングシステムのバックグラウンドジョブとして実行することに向いた設計が必要になる．

そのための一つの方法として，入力ファイル，制御ファイル，出力・ログファイルの 3 種類のテキストファイルでシミュレーションを定義している．

- **入力ファイル**
  入力ファイルにはシステムの仕様が格納される．反応物質の構成と初期値，および確率モデルの速度定数が記述されている．同時に，分裂の際のルールも記述されている．

- **制御ファイル**
  制御ファイルにはどの反応物質の数をモニタするのかが記述される．プログラムは制御ファイルに記述されたシミュレーション時間間隔で反応物質の数を記録する．モンテカルロ反復の回数，乱数の種も制御ファイルで設定する．

- **出力・ログファイル**
  出力・ログファイルには出力とログが出力されるディレクトリパスを記述する
  - 出力ディレクトリに書き出されるファイルは軌跡 (Trajectry) ファイルと呼ばれる．軌跡ファイルは 2 カラムのテキストファイルで，制御ファイルに記述された時間間隔で時間の関数として分子の数を記録している．Gnuplot のような作図ソフトウェアで使用しやすい形になっている
  - ジョブが中断された場合に復帰できるように，システム内の全ての反応分子数を格納している

#### 乱数生成器

STOCKS では Marsaglia と Zaman の $2^{144}$ サイクルの乱数発生器を使っている．

#### ユーティリティプログラム

STOCKS にはデータの解析を支援する 4 つのユーティリティプログラムが提供されている．

- **軌道平均算出プログラム**
  モンテカルロ反復により，同じ分子をモニタしているシミュレーションの軌跡ファイルが複数作成される．これらについて，軌道平均算出プログラムは指定した時間間隔で分子数の平均・標準偏差を計算する．

- **分子数加減算プログラム**
  2 つの軌道ファイルから，ある時間における分子数を加減算するプログラムである．いくつかの分子数の合計を知りたい場合に使用する．

- **分子数平均算出プログラム**
  軌道ファイルの所望の部分について，分子数の平均を計算するプログラムである．分子数が定常レベルに達しているときに使用する．

- **軌道-1 次関数比較プログラム**
  軌道ファイルの分子数が描く形について，適当な部分に一次関数を合わせることによって，軌道の挙動を確認するプログラムである．分子数が一定速度で増加または減少しているときに使用する．

### STOCKS の計算時間

Kierzek は STOCKS の提案に際し，2 つの反応について例示している．Kierzek が言及しているように，計算の目的は，反応物質の詳細なモデル構築ではなく，ソフトウェアの性能評価にある．

- **原核生物の転写と翻訳の開始頻度と遺伝子発現の確率変動**
  提案した大腸菌 (E.coli) のタンパク質合成速度レベルと，LacZ 遺伝子の mRNA レベルに関する実験データから Kierzek らが提案した運動モデルをシミュレーションするものである．シミュレーションは 100 個の独立した軌道について，細胞 10 世代の期間 (2100 秒間) の計算を行い，100 回のモンテカルロ反復の結果を蓄積し，タンパク質の時間変動をプロットした．PentiumIII 800MHz プロセッサでこのシミュレーションに約 22 時間掛かっている．

- **E.coli の LacZ，LacY の遺伝子表現およびそのタンパク質の酵素/転送活性**
  STOCKS が酵素反応と少数の反応分子を含むシステムに適用できることをテストするものである．細胞 1 世代 (2100 秒) の 1 つの軌道の計算に，PentiumIII 800MHz プロセッサを用いて約 2.5 時間掛かる．また，単一の軌道について細胞 10 世代のシミュレーションに約 90 時間掛かる．

## 2.2.3 STOCHSIM

STOCHSIM[10] は Gillesie のアルゴリズムの弱点を克服するものとして Shimizu が開発している確率モデルシミュレータである．

### Gillespie のアルゴリズムの弱点克服

従来の解析モデルと同様，Gillespie のアルゴリズムはシステム内の分子の種類と数のみを扱っており，分子の偏在については考慮していない．Gillespie のアルゴリズムが適用できるためには，分子が一様分布していることが前提になっている．これは，異化活性化や親和性による結合など，多重変形するタンパク質がシステム内に存在する場合に問題になる．分子のそれぞれの状態は異なる特性を持っている場合があり，モデルを詳細化していくと，反応数が数百万に上ることがあるかもしれない．解析モデルや Gillespie のアルゴリズムは反応数に比例してシミュレーション時間が増加するため，これらの機能の実装が困難になっている．

STOCHSIM では，同じ分子でも 3 次元構造や性質が異なるものが存在するシステムのシミュレーションへの対応がなされている．

### STOCHSIM の特徴

- **多状態分子の定義**
  STOCHSIM は基本的な単分子反応・2 分子反応でシステムの挙動をシミュレートする．システム内のそれぞれの分子はオブジェクトで定義されている．このオ ブジェクトは “多状態分子 (Multistate molecules)” と呼ばれ，内部に状態を保存しておく分子表現が可能になっている．オブジェクト内部にある状態を示す変数は，状態によって反応発生確率が異なる場合に対応し，反応がもたらす，共有結合修飾や立体構造変化のような分子相互の反応を表現するために使われる．

- **反応確率と計算時間の定義**

  STOCHSIM ではシミュレーション時間を確率モデルで計算せず，ある一定の微小時間間隔ごとに計算を繰り返す，という方法を取っている．シミュレーションの初期化の段階で，全ての反応についての反応確率がユーザ定義による速度定数から計算され，ルックアップテーブルに格納される．シミュレーションの 1 サイクルで進行する時間は，そのサイクルで起こりうる反応について，反応速度を保証できる長さに調節される．

  この基本設定が行われた後は，単分子・2 分子反応がランダムに選択された分子で発生する，という単純な計算の繰り返しでシミュレーションが進展していく．シミュレーションされるシステムには疑似分子と呼ばれる擬似的な分子が定義されている．疑似分子は STOCHSIM でのみ定義される仮想的な分子で，実際に存在するわけではない．STOCHSIM は単分子反応と 2 分子反応の反応を簡潔に記述するために，選択した 2 つの反応分子のうち，片方が疑似分子だった場合を単分子反応であると定義している．疑似分子の数は単分子反応の発生頻度を示すものである．

- **実装**

  STOCHSIM は標準 C++で記述され，Perl/Tk で記述された GUI が提供されている．オリジナルの STOCHSIM シミュレータは Microsoft Windows 用のシミュレータとして開発され，MS/MFC 製品をベースとした GUI が提供されている．UNIX,Linux，MacOS のような他の OS で STOCHSIM を実行するために，Perl で記述された Tk インタフェースが提供されている．

  出力はユーザ定義の時間間隔での分子数や分子の状態であるが，空間の分子の状態や分布を表すグラフィカルスナップショットなども出力することが可能である．

## 計算アルゴリズム

STOCHSIM のシミュレーションは以下のような流れで実行される．時間は非連続の微小時間に量子化される．それぞれの微小時間ごとに，システム内に存在する分子のうち 1 種類のオブジェクトがランダムに選択される (ここで選択する分子は疑似分子ではない)．次に，疑似分子を含めたシステム内に存在する分子の中から 1 つのオブジェクトを選択する．もし，2 つの分子が選択されたならば，対応する分子対で発生する 2 分子反応が発生する．もし 1 つの分子と 1 つの疑似分子が選択されたならば，最初の分子の単分子反応が発生する．

発生した反応に対応する反応確率がルックアップテーブルから読み出される．それから乱数を生成し，反応確率と比較することで反応の発生を判定する．

この結果，反応が発生したならば，分子数の増減，分子の状態の変化など，発生した反応に対応するシステムの更新が行われる．

分子が選択された後に反応が発生する確率は以下のように計算される．

- もし最初に分子 A が選ばれ，次に疑似分子が選ばれた場合，A が単分子反応する確率は，

$$P = \frac{k_1 \times n \times (n + n_0) \times \Delta t}{n_0} \tag{2.33}$$

- もし 2 つの分子が選択された場合，それらが反応する確率は

$$P = \frac{k_2 \times n \times (n + n_0) \times \Delta T}{2 \times N_A \times V} \tag{2.34}$$

記号の意味は以下の通り．

- $n$ システム内の分子の数
- $n_0$ システム内の疑似分子の数
- $k_1$ 単分子反応速度定数 ($s^{-1}$)
- $k_2$ 二分子反応速度定数 ($M^{-1}s^{-1}$)
- $\Delta t$ タイムスケール (分割した微小時間)($s$)
- $N_A$ アボガドロ定数
- $V$ システム体積 ($l$)

疑似分子の数は，単分子反応のうち最も高速な反応の反応確率を，2 分子反応のうち最も高速な反応の反応確率に可能な限り近づけた値が設定される (式 2.35)．式 2.35 の $[x]$ は，$x$ に最も近い自然数に近似することを示している．

$$n_0 = \left[ 2 \times N_A \times V \times \frac{k_{1,\max}}{k_{2,\max}} \right] \tag{2.35}$$

### 多状態分子 — multiple state molecule

STOCHSIM でそれぞれの分子オブジェクトのインスタンスに内部状態を保持している．酵素の活性やタンパク質の信号伝達は，共有結合修飾やリガンドやサブユニットの結合，タンパク質の構造変化のような多数の要素で制御される．これらは分子オブジェクトのインスタンスの内部状態でモデル化されている．内部状態 1 ビットのフラグ列で管理されており，分子の状態を表現する．たとえば，外部のリガンドが膜輸送受容体へ結合するかどうかを決定するフラグなどがある．

### 開発状況

STOCHSIM は 2004 年段階のバーションは 1.4 である．微小時間での計算，多状態分子など，Gillespie のアルゴリズムに比べて現実的なモデル，たとえば熱力学的により現実的なモデルを扱うことができるが，計算時間やメモリ要求はより大きくなっている．開発当初のバージョン 1.0 では均一に混合された溶液に関する反応系を取り扱うことができたが，バージョン 1.2 で空間を 2 次元格子に分割し隣接した場所にある分子との相互反応をシミュレーションすることができるようになっている．これは，細胞内の膜上に密集した受容体などの研究に使用できる．空間の表現方法は現在も拡張が続けられている．

# 第3章

# ReCSiPの構成

本章では、細胞シミュレーションプラットフォーム ReCSiP(Reconfigurable Cell Simulation Platform) の設計方針について述べ、続いてその支援ツール群を含めたシステム構成について述べる。

## 3.1 ReCSiP の設計方針

ReCSiP は，FPGA を搭載した 1 枚のボードを一般的な PC/WS 環境に付加することで，現在はクラスタなどによって実現されている強力な計算能力を実現することを目標としており，全体の設計においては，特に以下の 4 点に重点がおかれている．

**柔軟性** 高い性能とともに，さまざまなアルゴリズムが実装できる柔軟性を備えること

**互換性** ホストとなるシステムをなるべく選ばないようなアーテキクチャであること

**容易性** ハードウェアや低レベルプログラミングに関する知識のないユーザでも簡単に利用できるようなプログラミングインターフェイスを備えること

**可搬性** アクセラレータのハードウェアがない環境でも，ソフトウェアによって代行処理が可能であること

## 3.2 シミュレーション高速化のためのストラテジ

現在，バイオインフォマティクスの分野では BLAST[11] などを用いた遺伝子配列の解析が盛んであり，さまざまな研究機関で大規模な PC クラスタが稼働している．これは，遺伝子配列の解析という問題は主に sequence matching であり，容易に並列化し，かつ高い並列度で実行することが可能であるためである．

Sequence matching を行うための計算機としては，FPGA を用いて Sun のワークステーションに attach する形のアクセラレータである Splash-2[12] が過去に大きな成果を挙げており，分子動力学の分野では，多体問題を解くために開発された専用計算機 GRAPE[13] が，蛋白質の構造解析などで成果を発揮しつつある．

代謝系シミュレーションにおいては，個々の化学反応は互いに独立な現象であるため，問題そのものに並列性は充分にあるものの，

- 複数の反応に関与する物質が多く，その依存関係が極めて複雑であること

- 物質の移動するモデルを扱う場合，計算の途中でシミュレーションする系全体に関するデータが必要となることがある

といった問題があり，クラスタなどを用いた場合には通信がボトルネックになる可能性を含んでいる．この場合に，FPGA などに複数の演算器を載せて並列処理を行えば，通信オーバーヘッドを発生させることなく効率のよい処理を実現することができると考えられる．

しかしながら，FPGA を用いたシステムをシミュレータから利用する場合，ハードウェアに密接したプログラムを書く必要があり，低レベルプログラミングに不慣れなユーザにとっては困難である．また，FPGA は並列処理による高速な演算処理を可能とするが，複雑な処理を行うことは困難であるため，

- ユーザプログラムの開発容易性
- ホストのプログラムと FPGA の処理の適切な切り分けの実現

などを考慮した，ホスト–アクセラレータ協調処理環境を構築する必要がある．ReCSiP は，図 3.1 に示すように，ソフトウェア・ハードウェアの両面でシミュレーションシステム開発に必要なリソースを提供しており，これらを利用することで FPGA を利用したアプリケーションを容易に開発したり，既存のアプリケーションを移植することで高い性能を得ることが可能になる．

このように ReCSiP は細胞シミュレーション (代謝シミュレーション) を主眼において設計されているが，実際にはシミュレータなどの特定のアプリケーションにターゲットを絞ったものではなく，シミュレーションの他にも顕微鏡などの測定機から送られてくる大量のデータをリアルタイムに処理 [14] することができる予定であったり，あるいは一度ディスク上に保存されたデータを高速に処理するシステムを ReCSiP の上に構築することも可能である．

## 3.3 ReCSiP の構成

### 3.3.1 全体像

ReCSiP のシステム構成を図 3.1 に示す．
ReCSiP が基本コンポーネントとして提供するのは，

- ハードウェア資源
  - ReCSiP Board: FPGA を搭載したボード本体
  - ReCSiP Verilog Library: FPGA に演算用のモジュールを実装する際に用いる基本的なモジュール類
- ソフトウェア資源
  - ReCSiP Driver: ReCSiP ボードのデバイスドライバ
  - ReCSiP API: ReCSiP ボードを制御するための基本的なインタフェース
  - ReCSiP 拡張 API 生成ツール: FPGA に搭載したモジュールを制御するコードを書くための補助ツール

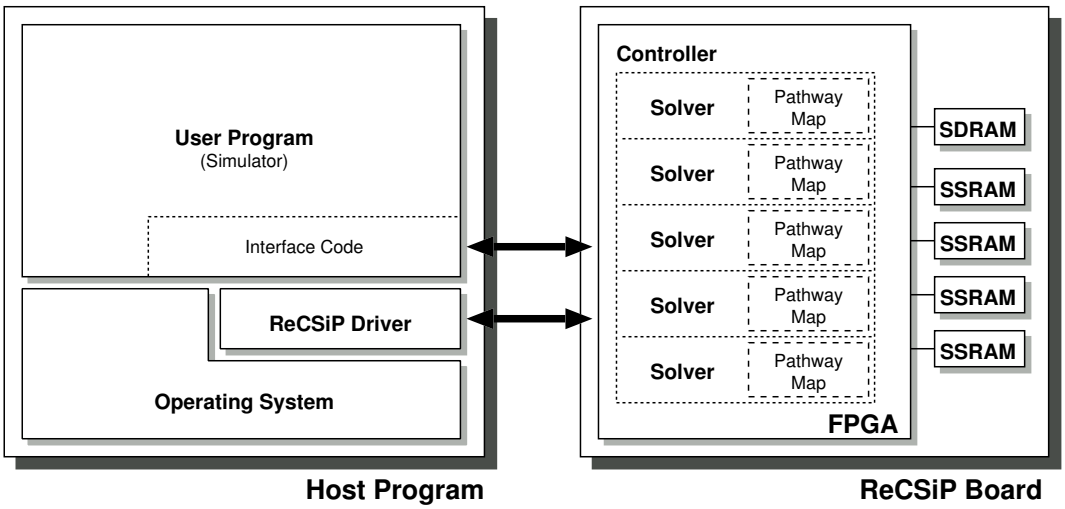



図 3.1: ReCSiP の構成

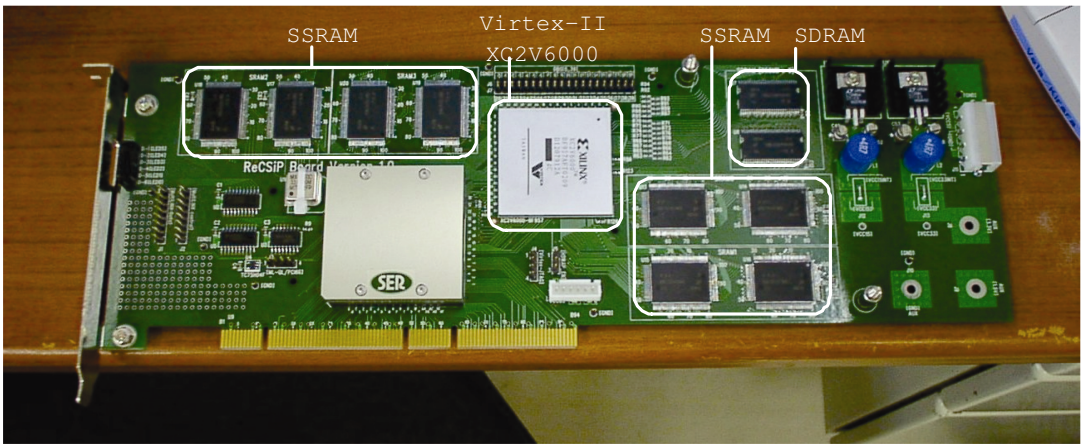



図 3.2: ReCSiP ボード

といったものである．

ユーザは，FPGA 上に目的とする演算モジュール (solver) を実装，あるいは既存の solver を利用し，ReCSiP の提供するソフトウェア資源を利用することで，対象アプリケーションの高速化を実現する．

ReCSiP の提供する個々の基本コンポーネントについては以下で述べる．

### 3.3.2 ReCSiP ボード

ReCSiP ボード (図 3.2) は，FPGA による高速な演算を実現するために，FPGA に PCI インターフェイス，メモリ等を接続したボードである．ボードの構成を図 3.3 に，ボードに搭載されている主要なコンポーネントを表 3.1 に示す．

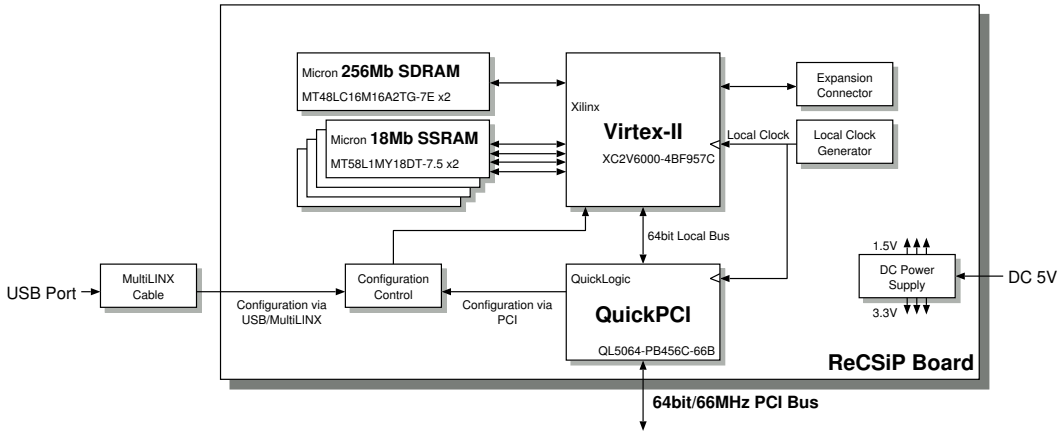



図 3.3: ReCSiP ボードの構成

表 3.1: ReCSiP ボードの構成部品

| | メーカー | シリーズ | 型番 |
|---|---|---|---|
| Core FPGA | Xilinx | Virtex-II | (XC2V6000-4BF957C) |
| PCI Interface | QuickLogic | QuickPCI | (QL5064-PB456C-66B) |
| Memory (SRAM) | Micron | SyncBurst SRAM | (MT58L1MY18DT-7.5) |
| Memory (DRAM) | Micron | Synchronous DRAM | (MT48LC16M16A2TG-7E) |

**Core FPGA**

コアとなる FPGA には，Xilinx 社の Virtex-II シリーズの製品である，XC2V6000-4BF957C を採用している．ボード上のほぼすべてのコンポーネントは Virtex-II に接続されており，Virtex-II は計算処理を行うとともに，ボード上のコンポーネントの制御を行う．

ボードに搭載されている Virtex-II は，6M ゲート相当で 8,448 の CLB (Configurable Logic Blocks) と 2,592kbits の内部 RAM，684 のユーザ I/O ピンを持っており，大規模な演算ロジックやコントローラを構成することが可能である．

**PCI インターフェイス**

PCI インターフェイスは，QuickLogic 社の FPGA である QL5064-PB456C-66B が用いられている．この LSI は，

- PCI インターフェイスロジックと，ユーザがプログラム可能で高速な動作が可能なアンチヒューズ型 FPGA を 1 チップに統合
- PCI インターフェイスはマスタ DMA 転送が可能で，32bit/64bit, 33MHz/66MHz, 3.3V/5V の各規格に対応
- PCI インターフェイスと FPGA 部分のクロックは分離可能

表 3.2: ローカルバスの信号線

| | | | |
|---|---|---|---|
| SYSCLK | in | なし | ローカルバスクロック．基板上のオシレータから QL5064 と Virtex-II のクロックピンに供給される |
| L_RSTX | out | 負 | システムリセット信号．PCI バスのリセット信号に連動して動作 |
| L_DRQX | out | 負 | ターゲット転送要求信号 |
| L_DAKX | in | 負 | L_DRQX に対する acknowledge |
| L_ADSX | i/o | 負 | 転送サイクルの開始とアドレスの転送を示すアドレスストローブ．ターゲットアクセス時は出力 (QL5064→Virtex-II) で，マスタアクセス時は入力 |
| L_WEX | i/o | なし/負 | Write 動作を示す信号で，ターゲットアクセス時は出力，マスタアクセス時は入力 |
| L_BSTMX | i/o | 負 | バースト転送を示す |
| L_RDYOX | out | 負 | QL5064 がデータ転送可能であることを示す |
| L_RDYIX | in | 負 | Virtex-II がデータ転送可能であることを示す |
| L_AD[63:0] | i/o | 正 | アドレス/データ信号線．ターゲットアクセス時は最初に L_ADSX と同時にアドレスが転送され，続いてデータが転送される．マスタアクセス時は L_ADSX と同時にアドレスが，続いて転送バイト数が転送され，続いてデータが転送される |
| L_BEX[7:0] | i/o | なし/負 | バイトイネーブル信号．ターゲットアクセス時には出力，マスタアクセス時には入力 |
| L_INTX | in | 負 | 割り込み信号 |
| L_S64X | in | 負 | コントロールレジスタ選択信号 |
| L_DBSYX | out | 負 | PCI bus の状態を示す．Low の場合には PCI ライトマスタアクセスは行えない |

といった特徴を備えており，ReCSiP ボードのカードエッジにある PCI バスコネクタに接続されている．このコネクタは 64bit/66MHz，3.3/5V 対応になっており，66MHz のバスに接続した場合にはボード上のジャンパで 33MHz 動作を強制することもできる．

### ローカルバス

QL5064 の FPGA 部分にはローカルバスを制御するためのロジックが書き込まれ，PCI バスとローカルバスの間のブリッジとして動作する．ローカルバスは表 3.2 に示す信号線で QL5064 と Virtex-II の間を接続しており，PCI バスとはクロックが切り離されている．ローカルバスのクロックはボード上の水晶発振器から供給されており，任意の周波数のものに交換することが可能である．

ローカルバスの target read/write 動作におけるタイミングを図 3.4 に示す．なお，ローカルバスのアドレス空間は 27bit である．

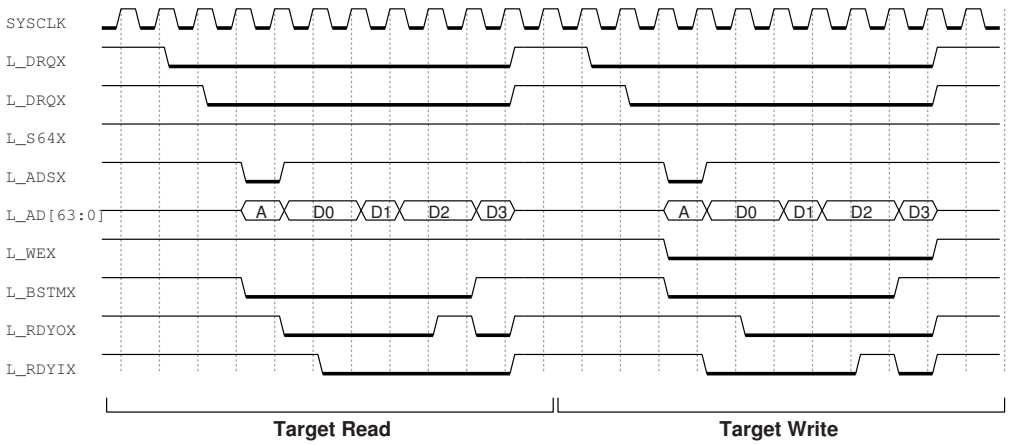



図 3.4: ローカルバスの read/write 動作

**RAM**

メモリは 18Mb の同期 SRAM である，Micron 社の MT58L1MY18DT-7.5 を 2 つずつ 4 セット(注 1)と，256Mb の同期 DRAM である，Micron 社の MT48LC16M16A2TG-7E を 2 つ実装している．

各メモリのデータ幅は 16bit であり，2 つを 1 セットで並列に Virtex-II に接続することで 32bit のデータ幅にしている．SRAM の各セットの容量は 4MB，計 16MB であり，DRAM は 64MB である．

各メモリへ供給するクロックは，Virtex-II 上の DLL(Delay Locked Loop)) により，水晶発振器から供給されるクロックを分周または逓倍して供給することができる．クロックスキューを低減するため，ボード上のクロック配線から DLL にフィードバックを行うことも可能である．

SRAM は制御が簡単で柔軟なアクセスパターンに対応できるため，シミュレーションの中間データなどを格納しておき，SDRAM は計算結果などを保存するために用いられる．

**電源**

PCI バスから供給される電源は最大 25W と定められている [15] が，SRAM の最大消費電力は 1 チップあたり 1.7W で，8 チップでは 13.5W となる．これに FPGA の消費電力が加わると電圧低下などの危険があるため，ボード上にレギュレータを搭載し，電源を供給している．

FPGA が用いる 1.5V の電源を供給するのは Linear Technology 社の LT1585A-1.5 で，最大出力は 5A である．メモリ等の 3.3V の電源は LT1584-3.3 が供給し，最大出力は 7A である．それぞれのレギュレータへの電力は，PC に内蔵される一般的な ATA 電源コネクタから供給される．

### 3.3.3 Verilog ライブラリ

ReCSiP では，ReCSiP ボード上の Virtex-II に回路を実装する際に必要となる，いくつかのモジュールを標準で提供する．ここでは，これらの実装について述べる．

(注 1)最初の 1 枚の基板に限り，8Mb の Micron MT58L512L18DS-7.5 であり，容量は半分となる．

表 3.3: 標準ローカルバスインターフェイスのレジスタ割当

| アドレス | 方向 | 内容 |
|---|---|---|
| 0x0000_0000 | W | マスタ動作時のボード側物理アドレス |
| 0x0000_0008 | W | マスタ動作時のホスト側物理アドレス |
| 0x0000_0010 | W | write 転送長さ (bytes) |
| 0x0000_0018 | W | read 転送長さ (bytes) |
| 0x0000_0020 | R | 割り込み制御レジスタ |
| 0x0000_0028 | W | ユーザ回路リセット |

#### 標準ローカルバスインターフェイス

QL5064 とのハンドシェイクを行い，ホストとのデータ転送を行うのがこのインターフェイスモジュールである．標準で実装されている制御レジスタを表 3.3 に示す．

マスタ動作時のボード側／ホスト側物理アドレスを設定し，write 転送長さ，あるいは read 転送長さのレジスタに転送長を設定すると，転送長設定レジスタへの書き込みがトリガとなって，QL5064 経由でマスタモード転送を開始する．

割り込み制御レジスタは，ReCSiP ボードが PCI バスの INTA#をアサートして割り込みを発生する場合に用いる．Virtex-II 側では，INTA#をアサートする前にこのレジスタを non-zero な値にしておき，ホストからレジスタに対するリードアクセスが発生するまで保持する．ReCSiP ボードが INTA#をアサートしていない間は，このレジスタの値は 0 に保持される．

ユーザ回路リセットレジスタへの書き込みアクセスにより，ユーザ回路リセット信号が発生するようになっており，Virtex-II 内のすべてのモジュールは，このユーザ回路リセット信号を参照しなければならない．. リセット動作時はユーザロジックとローカルバスインターフェイスからクロック制御系の DLL にリセット信号が出たのちに，DLL による位相制御がロックしたことを確認してから，ユーザロジックへのリセットが解除される．これにより，DLL を利用するユーザは，DLL のステータスを気にせずにロジックを設計することが可能になる．

また，空いているアドレス空間にユーザが任意のレジスタを実装することも容易である．空きアドレス空間は，これらの標準レジスタの後から，表 3.4 に示すメモリ類の前までであるが，システムの拡張を考慮して 0x50 より下のアドレス空間は予約されている．

#### BlockRAM コントローラ

多くのユーザは 32bit 幅のメモリを必要とするが，ローカルバスの幅は 64bit である．したがって，メモリに書き込む際には 64bit から 32bit への変換を要する．Virtex-II の内部 RAM (BlockRAM) を用いる場合に，この変換を行うモジュールが BlockRAM コントローラである．

このコントローラには，1 つの 64bit インターフェイスと，2 つの 32bit インターフェイスがあり，8bit 幅の dual port な BlockRAM が 4 つ接続される．BlockRAM 自体には byte write enable の機構がないため，8bit 幅の RAM を複数用いることで byte write enable に対応している．64bit インターフェイスは，ローカルバスインターフェイスに接続され，32bit インターフェイスはユーザロジックに接続される．

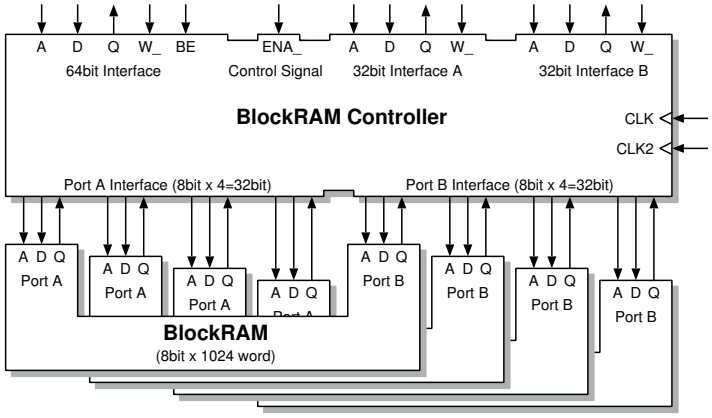



図 3.5: BlockRAM コントローラ

表 3.4: 標準ローカルバスインターフェイスのメモリアドレス割当

| アドレス | 内容 |
|---|---|
| 0x0200_0000 | SRAM0 |
| 0x0280_0000 | SRAM1 |
| 0x0300_0000 | SRAM2 |
| 0x0380_0000 | SRAM3 |
| 0x0400_0000 | DRAM |

ふたつの 32bit インターフェイスは，BlockRAM と同じタイミングで使用することができる．32bit インターフェイスの片方 (A) は 64bit インターフェイスと排他利用であり，もう一方 (B) は常にアクセスが可能である．BlockRAM 同様に，32bit インターフェイス A と 32 ビットインターフェイス B，または 64bit インターフェイスと 32bit インターフェイス B は dual port RAM として動作し，同時アクセスが可能である．

64bit インターフェイスからの read/write を 32bit に変換するため，64bit インターフェイスが enable されている間，BlockRAM のポート A にはシステムクロックの 2 倍の周波数のクロックが供給され，BlockRAM コントローラがデータ幅変換を行う．

### SSRAM/SDRAM コントローラ

SSRAM/SDRAM のローカルバス上のアドレスを表 3.4 に示す．これらのコントローラは現在デバッグ中である．

### 浮動小数点演算器

ReCSiP では，Verilog-HDL で記述された浮動小数点演算器として，単精度の加減算器と乗算器を用意している．表 3.5 に示すように比較的コンパクトかつ高速に作られており，Virtex-II 用の

表 3.5: 浮動小数点演算器の仕様

| | 加減算器 | 乗算器 |
|---|---|---|
| パイプライン | 5 段 | 8 段 |
| 動作周波数 | 83MHz | 84MHz |
| 使用スライス数 | 384 (1.13%) | 1125 (3.33%) |
| フォーマット | IEEE-754 準拠 単精度浮動小数点 | |

表 3.6: 市販 IP コアの浮動小数点演算器の仕様

| | 加減算器 | 乗算器 |
|---|---|---|
| パイプライン | 4 段 | 7 段 |
| 動作周波数 | 58MHz | 74MHz |
| 使用スライス数 | 605 (1.79%) | 742 (2.20%) |
| フォーマット | IEEE-754 準拠 単精度浮動小数点 | |

IP コアとして Digital Core Design 社が提供しているもの (表 3.6) と比較しても遜色ないものであることがわかる.

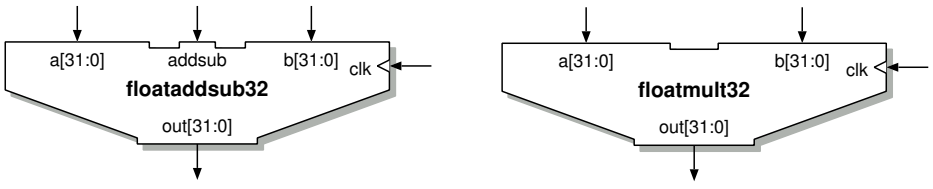
### 3.3.4 ソフトウェア

ReCSiP が提供するソフトウェアは,

1. 基本的なボードとの通信機能を提供するドライバ
2. ユーザプログラムからドライバを利用するための API
3. FPGA に実装した回路とのインターフェイスを作成するための拡張 API

から構成される.

ドライバは ReCSiP ボードに割り当てられた 128MB のアドレス空間に対して読み書きを行うためのインターフェイスを提供しており，次に示す API を利用してアクセスすることが可能である.




(a) 加減算器 (b) 乗算器

図 3.6: ReCSiP の浮動小数点演算器

**ReCSiP API**

- デバイスの使用宣言／開放
  - `int  rc_open();  // device open`
  - `void rc_close(); // device close`

  ReCSiP ボードを使用する場合には `rc_open()` によって使用を宣言する．複数のプロセスによるボードの利用は許されず，既に使用しているプロセスが存在している場合にはエラーが返される．ReCSiP ボードは `rc_close()` によって解放される．

- PIO による 32bit 転送
  - `long rc_single_read(unsigned long device_addr);`
  - `void rc_single_write(unsigned long device_addr, long data);`

  ReCSiP ボードのローカルアドレスを指定して 32bit の read/write 操作を行う．

- PIO によるブロック転送
  - `void rc_block_read(void *user_addr, unsigned long device_addr, size_t size);`
  - `void rc_block_write(unsigned long device_addr, void *user_addr, size_t size);`

  ホスト側のポインタと ReCSiP ボードのローカルアドレスを指定してその間でブロック転送を行う．write 操作時には PCI ブリッジによって自動的にバーストサイクルが発生することがある．

- バスマスタ転送
  - `void rc_master_read(void *user_addr, unsigned long device_addr, size_t size);`
  - `void rc_master_write(unsigned long device_addr, void *user_addr, size_t size);`

  使い方は PIO によるブロック転送と同じであるが，これらの関数を用いた場合には ReCSiP ボード上の QL5064 の DMA によってデータが転送される．

- メモリ空間へのマップ
  - `void *rc_mmap(unsigned long device_addr, size_t size, int prot);`
  - `int   rc_munmap(void *ptr, size_t size);`

  `rc_mmap` は，ReCSiP ボードのローカルアドレスとサイズを指定し，ユーザプログラムのメモリ空間にマップし，`rc_munmap` は，これを解除する．

- その他

- `void rc_wait();`
- `void rc_load(char *file);`
- `void rc_reset();`

これらは ReCSiP ボードからの割り込み待ち，Virtex-II のコンフィギュレーション，および Virtex-II 上のユーザ回路のリセット操作を行うための関数である．

### 拡張 API

API を直接使用する場合，ユーザは Virtex-II 上に存在する資源が，ReCSiP ボード上のローカルアドレス空間にどのようにマッピングされているかを常に意識しなければならない．

拡張 API は，Virtex-II 上に構成されたレジスタやメモリのアドレスマップから，それらをポインタとして名前ベースの操作で扱えるようなヘッダファイルを生成するプログラムによって提供され，ユーザはこのヘッダファイルを呼び出すことで `mmap()` などの操作を意識せずに，ボード上のリソースにアクセスすることが可能である．

# 第4章

# 設計と実装

## 4.1 変数の定義

Gillespie のシミュレーションアルゴリズムを設計・実装するにあたり，以下のように変数を定義する．

$X_\nu$ : システム内に存在する $S_\nu$ 分子の数
$h_\mu$ : 反応 $R_\mu$ にかかわる分子の組み合わせ数
$c_\nu$ : システム温度と分子の物理的性質に依存する確率反応速度定数
$a_\mu$ : 各反応の発生確率を示す指標
$a_0$ : 発生反応と時間経過計算の基礎的な値
$\tau$ : 次の反応が発生するまでの時間
$\mu$ : 次に発生する反応番号

## 4.2 シミュレーションの流れ

Gillespie のアルゴリズムを用いて，Lotka 反応モデルのシミュレーションを行うモジュールを ReCSiP 上に実装した．

Lotka 反応モデルは，初期値 $X_\nu$ $(\nu = 1, \cdots, 4)$ と $c_\nu$ $(\nu = 0, \cdots, 3)$ を設定し，$(0, 1)$ の一様乱数 $r_1$，$r_2$ を毎回生成した上で 1 反応ごとに以下のステップの計算を行うことでシミュレーションが進行する．

#### Step 1. $h_\nu$ の計算

シミュレーションのサイクルの実行時に，その時点の分子の数をもとに，$h_\nu$ を計算する．

$$h_1 = X_1 \cdot X_2 \tag{4.1a}$$
$$h_2 = X_1 \cdot X_3 \tag{4.1b}$$
$$h_3 = X_3 \tag{4.1c}$$
$$h_4 = X_4 \tag{4.1d}$$

**Step 2. $a_\nu$ および $a0$ の計算**

Step 1 で求められた $h_\nu$ と初期設定されている $c_\nu$ を掛け，$a_\nu$ を計算する．さらに，$a_\nu$ を全て加算し，$a_0$ を算出する．

$$a_1 = h_1 \cdot c_1 \tag{4.2a}$$
$$a_2 = h_2 \cdot c_2 \tag{4.2b}$$
$$a_3 = h_3 \cdot c_3 \tag{4.2c}$$
$$a_4 = h_4 \cdot c_4 \tag{4.2d}$$
$$a_0 = \sum_{i=1}^{4} a_i \tag{4.2e}$$

**Step 3. $\tau$ の計算**

$a_0$ と $(0, 1)$ の一様乱数 $r_1$ から，次の反応が起こる時間 $\tau$ を計算する．

$$\tau = \frac{1}{a_0} \ln\left(\frac{1}{r_1}\right) \tag{4.3}$$

**Step 4. $\mu$ の計算**

$a_\nu$ と $a_0$ を比較し，“次に発生する反応” を示す $\mu$ を求める．

$$\sum_{\mu=1}^{\mu-1} a_\nu < r_2 a_0 \leqslant \sum_{\nu=1}^{\mu} a_\nu \tag{4.4a}$$

**Step 5. 反応結果のフィードバック**

求められた $\mu$ の値から，対応する反応を実行し，分子数を増減する．Lotka モデルの場合は，式 4.5 で表される式が反応で起こる分子数の変化を示している．

$$\begin{cases} X_2 + 1 & \text{if } \mu = 0 \\ (X_1 - 1) \end{cases} \tag{4.5a}$$

$$\begin{cases} X_2 - 1 & \text{if } \mu = 1 \\ X_3 + 1 \end{cases} \tag{4.5b}$$

$$\begin{cases} X_3 - 1 & \text{if } \mu = 2 \\ X_4 + 1 \end{cases} \tag{4.5c}$$

$$\begin{cases} X_2 - 1 & \text{if } \mu = 3 \\ X_4 + 1 \end{cases} \tag{4.5d}$$

Step 1 から 5 の一連の処理がシミュレーションの 1 サイクルになる．実装における Lotka モジュールにおいて，この処理を実行する．

## 4.3 実装

### 4.3.1 実装モジュール

実装した Gillespie のシミュレーションアルゴリズムの実装にあたり，基本的な演算器以外に実装したモジュールの名称と機能を定義する．

| | | |
|---|---|---|
| Lotka 反応セットモジュール | : | 今回の実装の最上位モジュール<br>Lotka 反応モデルモジュールを 2 つ搭載し，メモリへの入出力を持つ |
| Lotka 反応モデルモジュール | : | Lotka 反応モデルシミュレータ<br>Gillespie のアルゴリズムを用いてシミュレーションを実行する<br>各種演算器と反応実行モジュールを内部に持つ |
| 反応実行モジュール | : | Lotka 反応の分子数変更モジュール<br>式 4.5 に従い，入力された反応により分子数の増減を行う |
| 対数テーブル | : | 32 ビット幅 15 ビットエントリのデュアルポートテーブル<br>$(0, 1)$ の乱数 $r_1$ より計算された $\ln(1/r_1)$ の値が格納される |
| 変数保存テーブル | : | 32 ビット幅 37 エントリのデュアルポートテーブル<br>シミュレーション中の分子数を格納する |

Gillespie のアルゴリズムを用いて確率モデルシミュレーションを実装するために，必要となる基本的な演算モジュールは以下のものが挙げられる．

- 32 ビット 整数 加減算器
- 32 ビット 整数 乗算器
- 単精度 浮動小数点加減算器
- 単精度 浮動小数点乗算器
- 整数-浮動小数点実数変換器
- 浮動小数点比較器
- 一様乱数生成器
- 対数格納用テーブル
- 変数用テーブル

### 4.3.2 浮動小数点実数の算術演算器

Gillespie のアルゴリズムでは，(0, 1) の乱数が使用され，対数計算を含めた有効数字の大きな実数が使用できることが望まれる．そのため，浮動小数点実数を扱えることが必要になる．実装する浮動小数点実数の形式は，IEEE754 に準拠した単精度浮動小数である．

シミュレーションアルゴリズムに使用する演算器は，ReCSiP が標準的に提供している浮動小数点加減算器を使用している．

現行の ReCSiP 標準の単精度浮動小数点乗算器は，整数乗算部分が通常の RTL で記述されており，Vertex-II の特有の機能である “18 × 18 ビットの乗算ブロック (embedded multiplier)” を使用していない．そのため，使用面積が大きくなってしまっている．これを解決するために，Vertex-II の乗算ブロックを使って整数演算器および浮動小数点乗算器の整数乗算器部分を実装した．乗算ブロックは 64 ビットまでの任意ビットの整数乗算器に使用することが可能である．整数乗算器は Xilinx 社提供のツール ISE CORE Generator 6.1.03i を使用して自動的に生成した．

実装した浮動小数点乗算器の性能を表 4.1 に示す．表 4.1 のスライス数，乗算ブロック数は，ReCSiP に搭載されている FPGA である Vertex-II XC2V6000 の全スライスに対する割合を示している．ReCSiP で標準的に提供されていた浮動小数点乗算器と比較して，約 1/7 の面積の面積と高速化を実現できていることがわかる．

表 4.1: 浮動小数点乗算器の比較

| | ReCSiP 標準 | 乗算ブロック使用 |
|---|---|---|
| パイプライン | 8 段 | 7 段 |
| 動作周波数 | 84 MHz | 110 MHz |
| 使用スライス数 | 1125 (3.33%) | 159 (0.471%) |
| 使用乗算ブロック | 0 (0.00%) | 4 (2.78%) |
| フォーマット | IEEE-754 準拠 単精度浮動小数点 | |

組み込み乗算器を使用した浮動小数点乗算器は，図 4.3.2 のような過程を経て処理を行っている．

計算の過程は以下の通り．

- **phase 1.**
  - 乗数 $a$，被乗数 $b$ の符号ビット (最上位ビット) の排他的論理和を取る
  - $a$，$b$ の指数ビット (31〜24 ビット目の 8 ビット) を整数加算する (出力は桁上げ含め 9 ビット)
  - $a$，$b$ の仮数ビット (23〜1 ビット目の 23 ビット) を組み込み乗算器を使用して整数乗算する (4 段パイプライン)
- **phase 2〜4.**
  整数乗算の計算を待つ間，符号部はビット数 ×6 段，指数部はビット数 ×5 段の長さのシフトレジスタに格納し，次の入力に対応する
- **phase 5.**

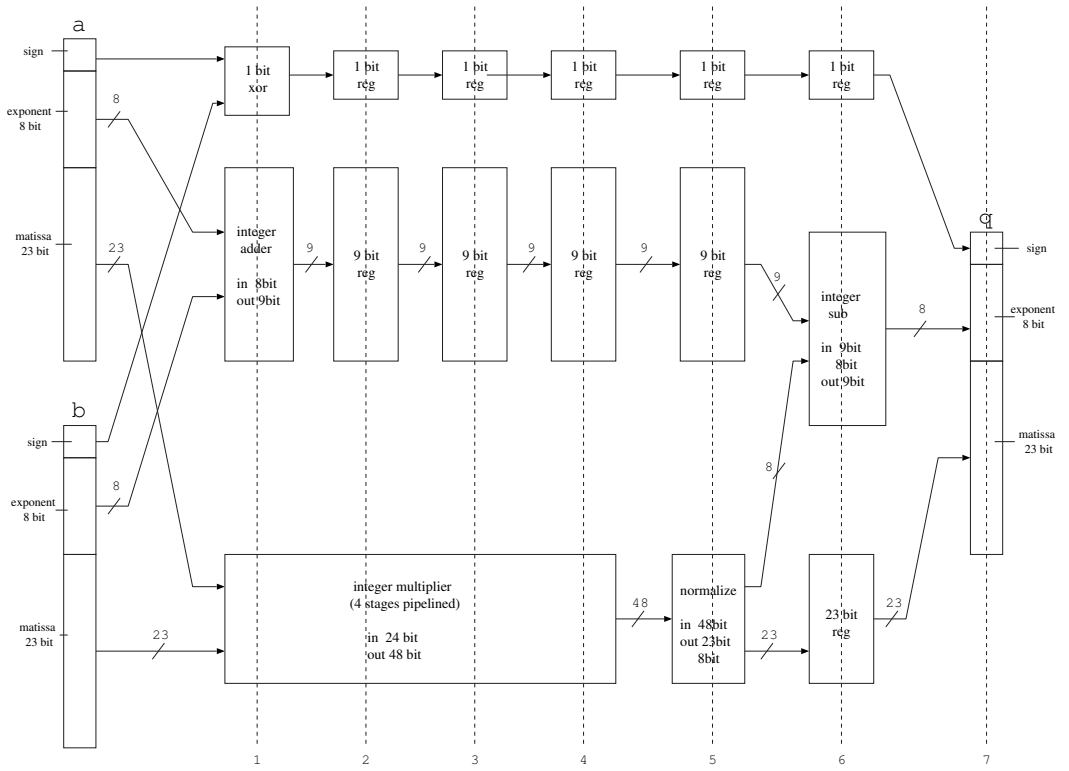



図 4.1: 組み込み乗算器を使用した浮動小数点乗算器

  - 仮数部の整数乗算の結果を正規化し，$1.x \times 2^y$ の形にし，$x$ の上位 23 ビットを出力する
  - $y$ の値が 1 であったら 127 を，0 ならば 126 を次の指数部の整数減算器の入力とする

- **phase 6.**
  指数部の値から，正規化モジュール (normalize) から入力された値を減算し，乗算結果の指数レジスタ (8 ビット) に格納する

- **phase 7.**
  符号ビット，指数ビット，仮数ビットを乗算結果として出力する

phase 5. において，127 または 126 を指数ビットから減算する理由は，浮動小数点の指数部はイクセス表現を取っており，+127 を 0 とみなしているため，phase 1. での指数ビットで整数加算された余分なイクセス表現部を取り除く必要があるからである．

### 4.3.3 一様乱数生成器

一様乱数生成器として，32 ビットの線形フィードバックシフトレジスタ (LFSR:Linear Feedback Shift Register) を用いた M 系列乱数発生器を実装した．

LFSR は排他的論理和で帰還をかけたシフトレジスタによるカウンタである．構成するビット数を $n$ とすると，$2^n - 1$ までの疑似乱数を生成することができる．確率モデルシミュレータでは，15 ビットと 23 ビットの一様乱数が必要になるため，32 ビットの LFSR を 2 つ生成し，それぞれ

上位の 15 ビットを取るモジュール rgen1 と，23 ビットを取るモジュール rgen2 を実装した．この動作を図 4.2 に示す.

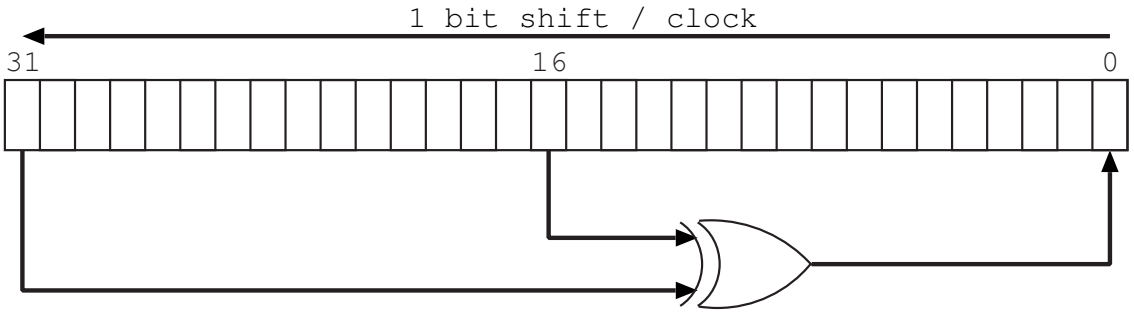



図 4.2: 32 ビット LFSR による疑似乱数生成モジュール

このモジュールは 1 クロックごとに以下のような動作を行う.

- 32 ビットのレジスタのうち，31 ビット目と 16 ビット目を取りだし，排他的論理和を取る
- レジスタ全体を 1 ビット右シフトし，0 ビット目に排他的論理和を取った結果のビットを格納する
- 最上位ビットから必要な分のビット列を出力する

$(0, 1)$ の浮動小数型実数の乱数を得るためには，23 ビットの疑似乱数 $M$ を生成し，その上位に 001111111 という 9 ビットを付加し，32 ビットの浮動小数を生成する．付加した 9 ビットのうち，最上位は符号ビット (0 は正)，続く 8 ビットは浮動小数の指数部 ($01111111_{(2)} = 127_{(10)}$ はイクセス表現で 127 が加算されているため指数部は 0 となる) を意味している．この過程で生成された乱数値 $r_2$ は，23 ビットの疑似乱数を $M$ とすると，

$$r_2 = 2^0 \times 1.M_{(2)} = 1.M_{(2)} \tag{4.6}$$

である．これは，小数点以下 23 ビットの $(1, 2)$ の一様乱数と見なすことができる．そのため，浮動小数点減算器を使い，$r_2$ から 1.0 を減ずることで，$(0, 1)$ の実数型一様乱数を得る.

### 4.3.4 対数テーブル

対数を使った計算に用いるテーブルは Vertex-II 上に搭載されている BlockRAM(XC2V6000 の使用可能サイズは 288KB) に実装した．このテーブルには $(0, 1)$ の一様乱数 $r_1$ を使用した $1/\ln(1/r_1)$ の値が格納されているが，便宜上これを “対数テーブル” と呼ぶことにする．BlockRAM はデュアルポートで使用し，確率モデルシミュレータを 2 つ FPGA 上に生成し，1 つのテーブルから同時に 2 つ値を読み出せる構成とした．また，現時点では対数テーブルのデータをシミュレータ側から書き換える必要が無いため，この対数テーブルは読み出し専用で構成している.

テーブルは 32 ビットのデータを 15 ビットエントリ ($2^{15} = 32768$ 個) 格納できるサイズで構成し，テーブルの中に書き込む値は，0 から 1 の間をエントリの数 (15 ビットエントリなので 0 から 1 の間を $2^{15} = 32768$ 個に分割) に等分し，それぞれの値を $r_1$ とし，その $r_1$ から $1/\ln(1/r_1)$ を計算し，32 ビット浮動小数点表現で格納している．シミュレータでは，LFSR で 15 ビットの乱数を生

成し，その値をアドレスとしてテーブルから 32 ビットのデータを引いてくることで，$1/\ln(1/r_1)$ の浮動小数点型実数を得ることになる．

対数テーブルは，Vertex-II の BlockRAM 上に 32 ビット幅 13 ビットエントリのテーブルを 4 つ構成し，15 ビット乱数の上位 2 ビットから値を取り出すテーブルを選択するモジュールを実装した．テーブルを 4 つに分割した理由は，ISE で BlockRAM を構成する場合に，巨大なテーブルにデータを書き込む場合に非常に長時間かかることと，将来の対数テーブル書き換えに対応するためである．

### 4.3.5 Lotka 反応モデルの実装

Gillespie のアルゴリズムを用いて Lotka 反応モデルのシミュレーションを実行する Lotka 反応モデルモジュールを実装した．このモジュール 1 サイクルのデータフローは図 4.3 のように進行する．

Lotka 反応モデルモジュールの 1 サイクルは，以下に示す 7 つのフェーズで実行される (phase 0. は初期化の段階)．各フェーズの (Step X) は，4.2 節での Step に対応している．

**phase 0.**

シミュレーションの実行前に各種初期化を行う．

- 反応に関わる分子の物理的性質とシステムの温度のみに依存する $c_\mu$ の初期値を設定する
- 乱数種 2 つを乱数生成器用に設定する
- 各分子の初期分子数を設定する．パイプライン化した場合，パイプラインの数だけ入力を繰り返す

**phase 1.**

Step 1. : シミュレーションサイクル第一段階．サイクル開始時の分子数から，反応ごとに関わる分子の組み合わせを示す値 $h_\nu$ を計算する．

$$\begin{aligned} h_1 &= X_1 \cdot X_2 \\ h_2 &= X_1 \cdot X_3 \\ h_3 &= X_3 \\ h_4 &= X_4 \end{aligned}$$

- 変数保存テーブルから読み出された分子数から，分子の衝突で反応が発生する可能性がある分子，または単一の分子で反応が起こる可能性がある分子を選択する
- 異種の分子の場合，反応に関わる分子の組み合わせの数を整数乗算により計算し，整数を浮動小数点実数に変換する
- 衝突無しで反応する可能性のある分子 ($X_2$，$X_3$) も分子数を浮動小数実数に変換する

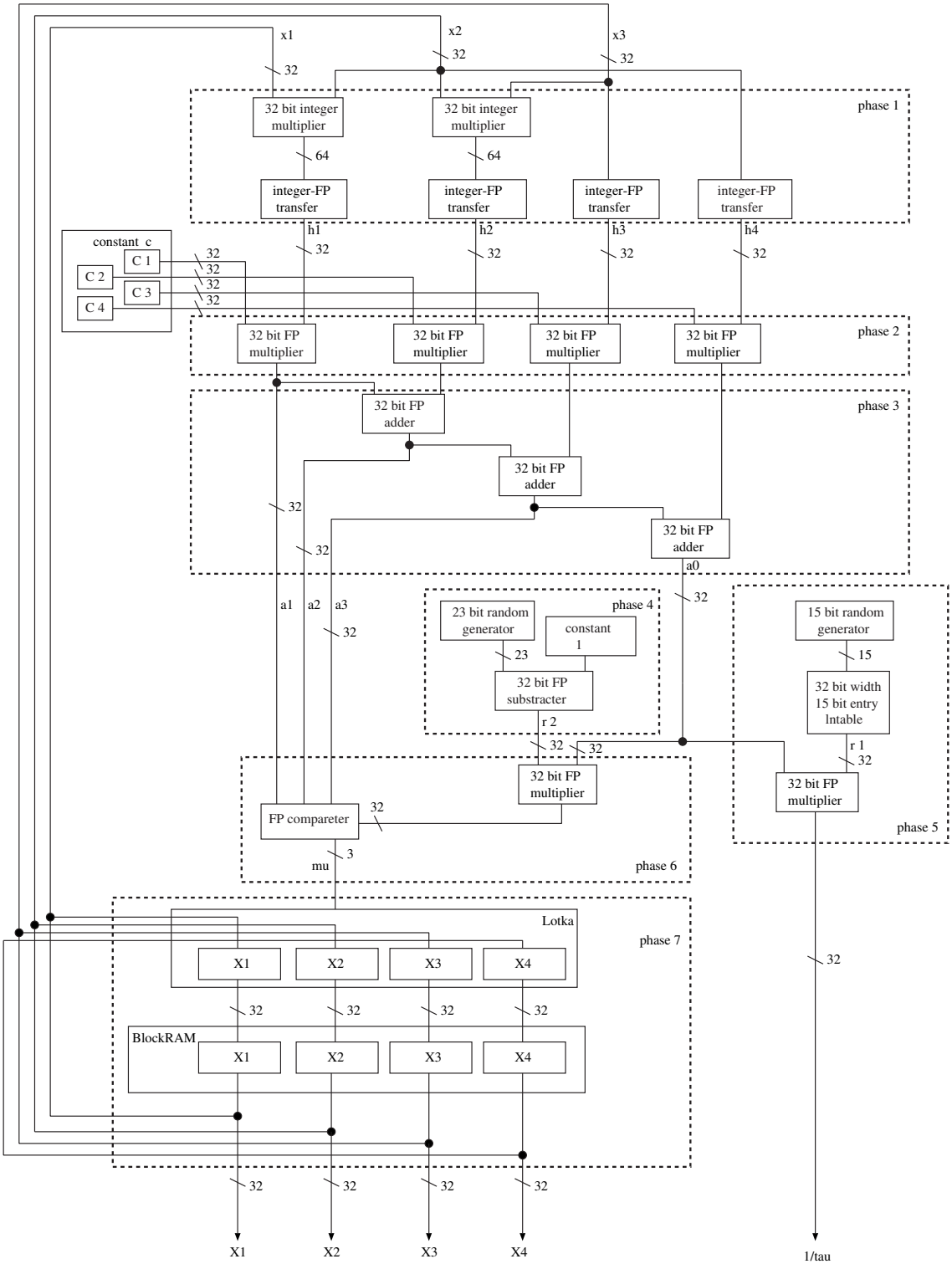



図 4.3: Lotka 反応モデルの確率モデルシミュレーションの計算過程

**phase 2.**

Step 2. 前半 : $a_\nu$ を計算する.

$$\begin{aligned} a_1 &= h_1 \cdot c_1 \\ a_2 &= h_2 \cdot c_2 \\ a_3 &= h_3 \cdot c_3 \\ a_4 &= h_4 \cdot c_4 \end{aligned}$$

- システム温度と分子の物理的性質で決定される固定値 $c_\mu$ と，phase 1. で計算した分子の組み合わせ数 $h_\nu$ を乗算し，$a_\nu$ を求める

**phase 3.**

Step 2. 後半 : phase 2. で求められた $a_\nu$ を全て加算することで，次の反応までの時間 $\tau$，次に起こる反応 $\mu$ を計算するための要素となる $a_0$ を計算する.

$$a_0 = \sum_{i=1}^{4} a_i$$

- phase 2. の結果を順番に加算する (最後の加算の結果が $a0$ になる). $a_1$ から $a_3$ まで順番に加算した結果を phase 6. で使用する. そのため，反応が 1 種類増えるごとに，シミュレーションの使用する加算器が 1 つずつ増え，この加算の部分の長さが増加することになる.

**phase 4.**

Step 3. : $(0, 1)$ の単精度浮動小数型一様乱数 $r_2$ を計算する.

- 一様乱数生成モジュールが出力する 23 ビットの乱数の上位に $(00111111)_{(2)}$ を付加し，$(1, 2)$ の浮動小数点実数の一様乱数を得る
- 前項の一様乱数から 1.0(浮動小数点型 $3F800000_{(16)}$) を減算し，$(0, 1)$ の浮動小数点実数の一様乱数を得る

**phase 5.**

Step 3. : 15 ビットの乱数から対数テーブルを引いて $1/\ln(1/r_1)$ の値を得 ($r_1$ は $(0, 1)$ の単精度浮動小数点型実数)，次の反応が起こるまでの時間の逆数 $1/\tau$ を計算する.

$$\frac{1}{\tau} = \frac{a_0}{\ln(1/r_1)}$$

- 一様乱数生成モジュールが出力する 15 ビットの乱数から，その値をアドレスと見なし，対数格納テーブルから値を読み出す
- $a_0$ の値と対数テーブルから得られた値を乗算し，$1/\tau$ を得る

**phase 6.**

Step 4. : phase 2. で求めた $a_\nu$ と phase 3 で求めた $a_0$ を比較し，次に発生する反応 $\mu$ を求める．

$$\sum_{\mu=1}^{\mu-1} a_\nu < r_2 a_0 \leqslant \sum_{\nu=1}^{\mu} a_\nu$$

- $a_0$ の値と phase 4. で生成した (0, 1) の実数を乗算する
- 浮動小数点比較器でその値が $0, a_1, a_2, a_3$ のどの間にあるのかを求め，$(0, a_1]$ にあるならば $\mu = 0$，$(a_1, a_2]$ にあるならば $\mu = 1$ というように，$\mu$ を求める

**phase 7.**

Step 5. : 発生した反応 $\mu$ に応じて，phase 1. で使用した分子数 $X_\nu$ を増減する．ここで，phase 5. の $1/\tau$ と分子数 $X_2, X_3$ を外部メモリに出力する．

$$\begin{array}{lll} X_2 + 1 & & \texttt{(if } \mu = 0\texttt{)} \\ X_2 - 1 \quad \texttt{and} \quad X_3 + 1 & & \texttt{(if } \mu = 1\texttt{)} \\ X_3 - 1 \quad \texttt{and} \quad X_4 + 1 & & \texttt{(if } \mu = 2\texttt{)} \\ X_2 - 1 \quad \texttt{and} \quad X_4 + 1 & & \texttt{(if } \mu = 3\texttt{)} \end{array}$$

- 変数保存テーブルから分子数を読み出し，phase 6. で得られた $\mu$ をもとに，分子数を加減算する
- 結果の分子数を次サイクル用に phase 1. へ出力する
- 結果の $1/\tau$ と分子数を外部に出力する

実装したモジュールごとの処理の流れを示すと，図 4.4 のようになる．

使用している整数・浮動小数演算器はパイプライン化されているため，1 クロックごとに連続的に値を投入することが可能である．しかし，浮動小数点加減算は，出力の結果を次の入力で使うため，単体の演算器に入力値を選択するレジスタが必要になり，結果が出るまで 6 クロック (入力時にセレクタが追加され 1 クロック増える) 要している．

各演算モジュールにより，シミュレーション 1 サイクル (反応が発生する間隔) に必要なクロック数は，図 4.4 より，40 クロックとなる．

各演算モジュールを 1 つずつ使用してこの Lotka 反応モジュールを構成すると，使用率が最も高いのは浮動小数点乗算器で，1 つのシミュレーション計算に 1 サイクル 40 クロックのうち，6 クロック分使用している．この構成で単純に並列化すると，40/6 = 6.7 より最大で 6 つの並列化が可能となる．しかし，この方法はモジュールに投入する値を制御する回路が複雑になってしまう．また，浮動小数点乗算器以外のモジュールの使用率が低く，シミュレータをさらに複数用意して共有することは，回路がさらに複雑になり，信号線を引き回すために起こるオーバヘッドの増加や動作周波数の下落，設計の複雑化などのデメリットが考えられる．

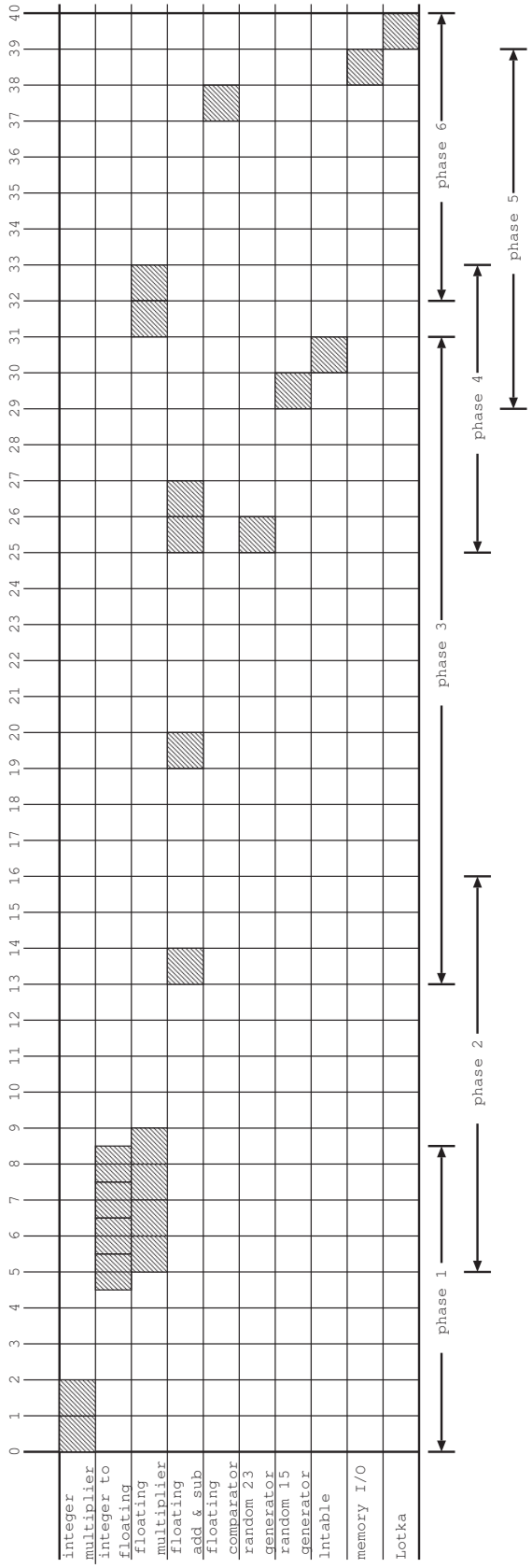



図 4.4: シミュレーション 1 つが 1 サイクル中でモジュールを使用する部分

図 4.3 から，phase 1. と phase 2. における整数乗算および浮動小数点乗算，整数-浮動小数変換器はそれぞれ並列動作させることができることがわかる．また，実装したすべてのモジュールはパイプライン動作し，かつ入力動作が 1 クロックで完了することから，演算数分の演算器を使用し，1 サイクルにつき 1 クロックだけモジュールを使用するようにそれぞれのモジュールを接続し，Lotka 反応モデルモジュールをパイプライン化する．表 4.2 に，パイプライン Lotka モデルモジュールに必要なモジュールのうち，複数使用するモジュールとその数を示す．

各モジュールは面積が比較的小さく，Vertex-II に十分収めることができ，また設計と信号線の制御の容易性の面からもこの設計は妥当であると言える．

表 4.2: パイプライン化 Lotka 反応モジュールに必要な演算器

| | |
|---|---|
| 32 bit 整数乗算器 | 2 |
| 32 bit 整数-浮動小数変換器 | 4 |
| 単精度 浮動小数点加減算器 | 4 |
| 単精度 浮動小数点乗算器 | 6 |

各乱数生成器，浮動小数点比較器，反応実行モジュールはパイプライン化が可能である．このパイプライン化により，あるモジュールの出力が次のモジュールの入力に直接信号を接続することができるため，浮動小数点加減算器の入力時にセレクタが不必要になり，図 4.4 よりも 1 サイクルに掛かるクロック数が少なくなっている．37 クロック 1 サイクルのパイプライン Lotka 反応モデルモジュールの 1 つのシミュレーション計算に使用する演算モジュールとその処理の流れを図 4.5 に示す．

パイプライン化により，それぞれの演算モジュールは 1 サイクルにつき 1 クロック使用されるため，データを 1 クロックずつずらして入力することで，1 つの Lotka 反応モデルモジュールは 37 個のシミュレーションを同時に計算することが可能である．

### 4.3.6 変数保存テーブル

Lotka 反応モデルモジュールでは，それぞれのシミュレーションサイクルの最初で現時点の分子数が入力され，サイクルの最後で次の反応までの時間 $\tau$ と，反応の結果の分子数が出力される．そのため，実装したパイプライン化 Lotka 反応モジュールでは，反応前の分子数を保存しておく必要がある．実装した Lotka 反応モデルは 4 つの分子からなるモデルであり，37 段パイプラインのため 32 ビット幅で深さ 37 の変数格納用テーブルが 4 つ必要になる．このテーブルは出力された分子数を対応する場所に書き込み，その時点の入力に対応する分子数を Lotka 反応モジュールに読み出す，という動作をするため，Vertex-II に搭載されているデュアルポートの BlockRAM を使用して構成した．変数保存テーブルを構成するために必要な BlockRAM のサイズは，

$$32[\text{bit}] \times 37[\text{pipeline}] \times 4[\text{molecules}] = 4736[\text{bit}] = 592[\text{Byte}] \tag{4.7}$$

となる．この変数保存テーブルの 2 つのポートの一方をポート A，他方をポート B とすると，ポート A は読み出し専用，ポート B は書き込み専用に使用し，それぞれアドレスを 0 から 36 までの値をインクリメントしながらそれぞれのポート A で読み出し，ポート B で書き込みを行う．浮動小数点比較器が $\mu$ の値を出力してから，メモリから分子数をポート A から読み出し，反応実行モ

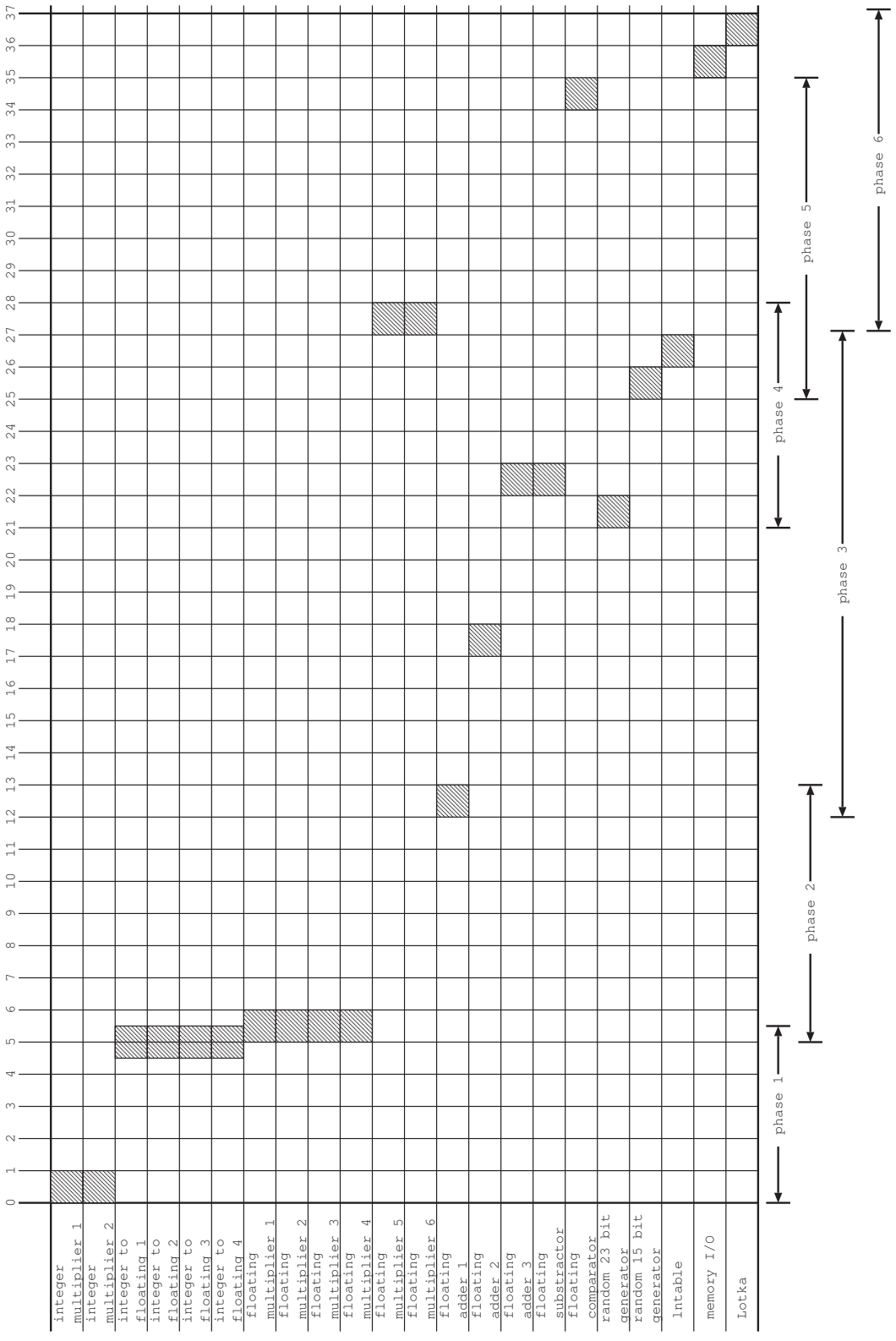



図 4.5: 37 段パイプラインの Lotka 反応モデルモジュールが使用する部分

ジュールを通して反応による分子数の増減を行った上で，読み出したアドレスをポート B で書き換えるため，2 つのポートのアドレスが一致することはなく，RAW ハザード，WAR ハザードは発生しない．

### 4.3.7 入出力モジュール

Lotka 反応を ReCSiP 上で動作させる上で，現実的な実装方法を述べる．

対数テーブルはデュアルポートで構成されており，1 クロックで 2 つの値を読み出すことができる．Lotka 反応モデルモジュールの面積は Vertex-II 全体に比べて小さいことが考えられる．BlockRAM の総量に比べて対数テーブルの占める割合は大きいため，Lotka 反応モジュールを 2 つ 1 組で構成し，BlockRAM を効率的に使用することにする．

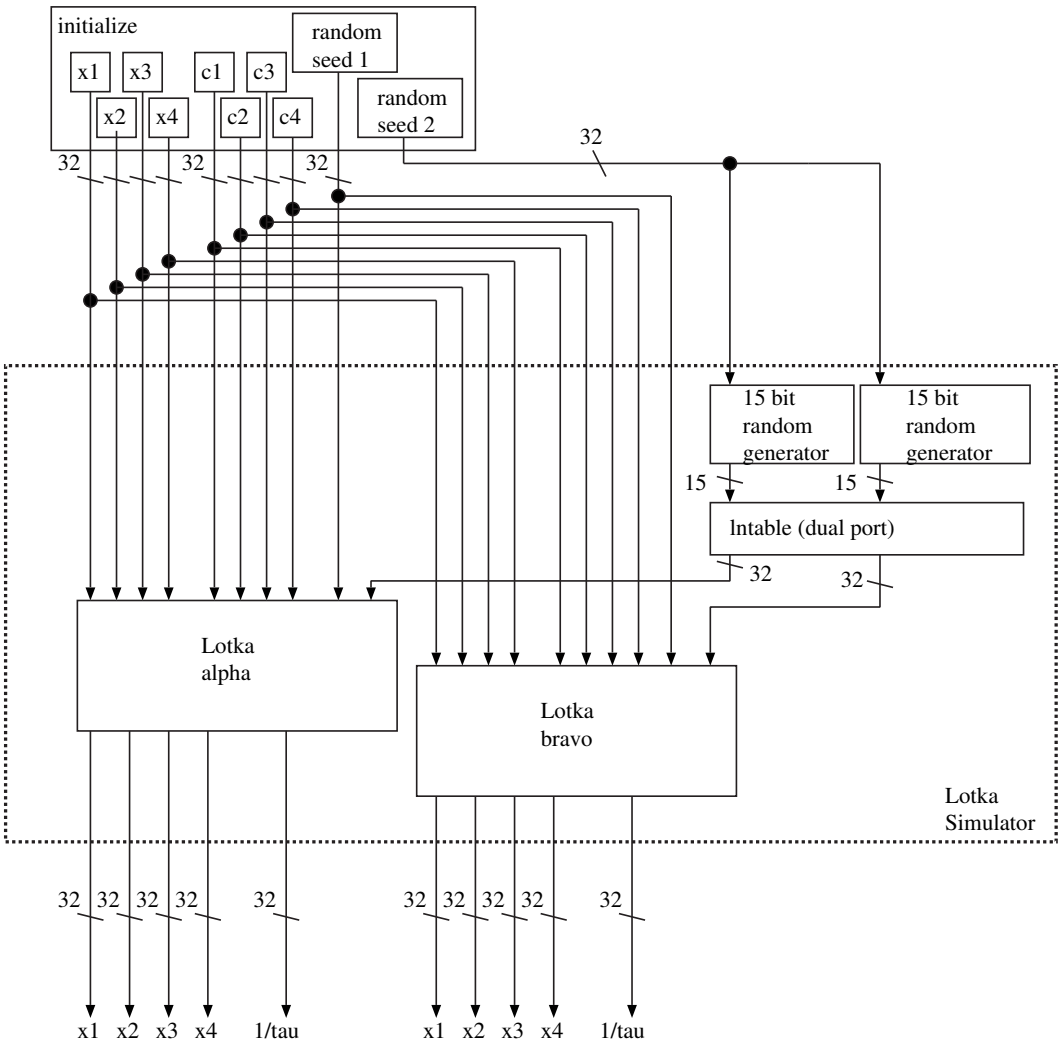



図 4.6: Lotka 反応モデルモジュールを 2 つ使用したシミュレーションモジュール

以降は 2 つの Lotka 反応モジュールを 1 組として，Lotka 反応セットモジュールとして扱う．Lotka 反応セットモジュールの入力は，リセット信号が無効になった後，初期化値として 4 つの分子の数の値 ($x_1$〜$x_4$) の値と 2 つの乱数種を 32bit の値で与える必要がある．入力値は 1 組を 1 クロック

ずつ 37 クロックに渡って入力する．38 クロック目から 1 クロックごとに出力として反応後の分子の値の組 ($x_1$〜$x_4$) が Lotka 反応モデルモジュール alpha と bravo より合計 2 組ずつ出力される．Lotka 反応モデルモジュールの 1 つの反応は 37 クロックごとに結果を出力する．37 段パイプラインで全ての演算器を 100% 使用するため，1 クロックごとに分子数と $1/\tau$ が出力される．毎回の出力の組とともに，前回の反応からその反応が発生するまでの時間の逆数 $1/\tau$ が単精度浮動小数点として出力される．

現行の設計では浮動小数点の除算器が実装されておらず，$\tau$ の値はホスト側で計算しなければならないため，複数サイクルの反応をまとめてメモリに出力する，という方法が使用できない．そのため，1 サイクルごとの結果をメモリに記録する必要がある．また，結果の出力において，結果を書き出す SRAM が 4 つであることから，分子数 $x_2, x_3$ を 16 ビットで併せて 32 ビット，32 ビットの $1/\tau$ を 1 つの SRAM に書き出し，シミュレーション結果を検討する．

# 第5章

# 評価

## 5.1 ソフトウェアでの計算時間

Lotka 反応セットモジュールを Vertex-II 上で実行する上で，現在あるソフトウェアシミュレーションと比較した際の性能向上について述べる．ソフトウェアで Gillespie の確率モデルシミュレーションアルゴリズムの Lotka 反応モデルについてシミュレーションを行った場合の処理能力について評価を行った．

ソフトウェアによって Lotka 反応モデルの確率的シミュレーションを行うプログラムを C 言語で記述し，50 万反応 (50 万サイクル) のシミュレーションを 1 回実行させた場合に要する時間を表 5.1 に示す．また，1 秒当たりに実行できるサイクル数 (反応数) を表に含めた．処理時間はプログラムを 10000 回実行し，平均を取ることで算出した．1 秒当たりに実行できるサイクル数 $S$ [Mcycle/sec] は，以下の式から算出した．

$$S = \frac{0.5[\text{Mcycle}]}{\text{処理時間 } [\mu\text{sec}]} \tag{5.1}$$

表 5.1: ソフトウェアでの処理能力

| プロセッサ | メモリ | 環境 | 処理時間 [$\mu$sec] | cycle/sec |
|---|---|---|---|---|
| Xeon 2.8GHz Dual(HT off) | 4GB | Linux2.4.21+gcc2.95.3 | 214219 | 2.33M |
| Pentium4 2.53GHz | 1GB | Linux2.4.21+gcc2.95.3 | 244607 | 2.04M |
| PentiumIII 800MHz | 512MB | Linux2.4.21+gcc2.95.3 | 512386 | 0.98M |
| AthlonXP 2800+ 2.08GHz | 2GB | FreeBSD 4.8+gcc2.95.3 | 172226 | 2.90M |
| Pentium4 2.6GHz | 1GB | FreeBSD 4.8+gcc2.95.3 | 192496 | 2.59M |
| Celeron 450MHz | 512MB | FreeBSD 4.8+gcc2.95.3 | 740459 | 0.68M |
| UltraSPARCIIIcu 1.2GHz | 4GB | Solaris8+gcc2.95.3 | 555907 | 0.90M |
| UltraSPARCII 300MHz | 2GB | Solaris8+gcc2.95.3 | 1847605 | 0.27M |

ソフトウェアで反応を 50 万サイクルシミュレーションした結果は，以下のようになる．比較対象としたソフトウェアプログラムでは，C 標準ライブラリを用いて乱数を生成している．図 5.1 に，$x$ 軸方向に時間，$y$ 軸方向に分子数を取り，$X_2$(被食者) と $X_3$(捕食者) の数の変化をプロットしている．$x$ 軸の単位は $c_\nu$ の単位によって変わるため，明示していない．一般的な化学反応の場合，

単位は [sec] である．図 5.1 のように，Lotka 反応モデルでは 2 つの分子数が時間の進行とともに振動する．

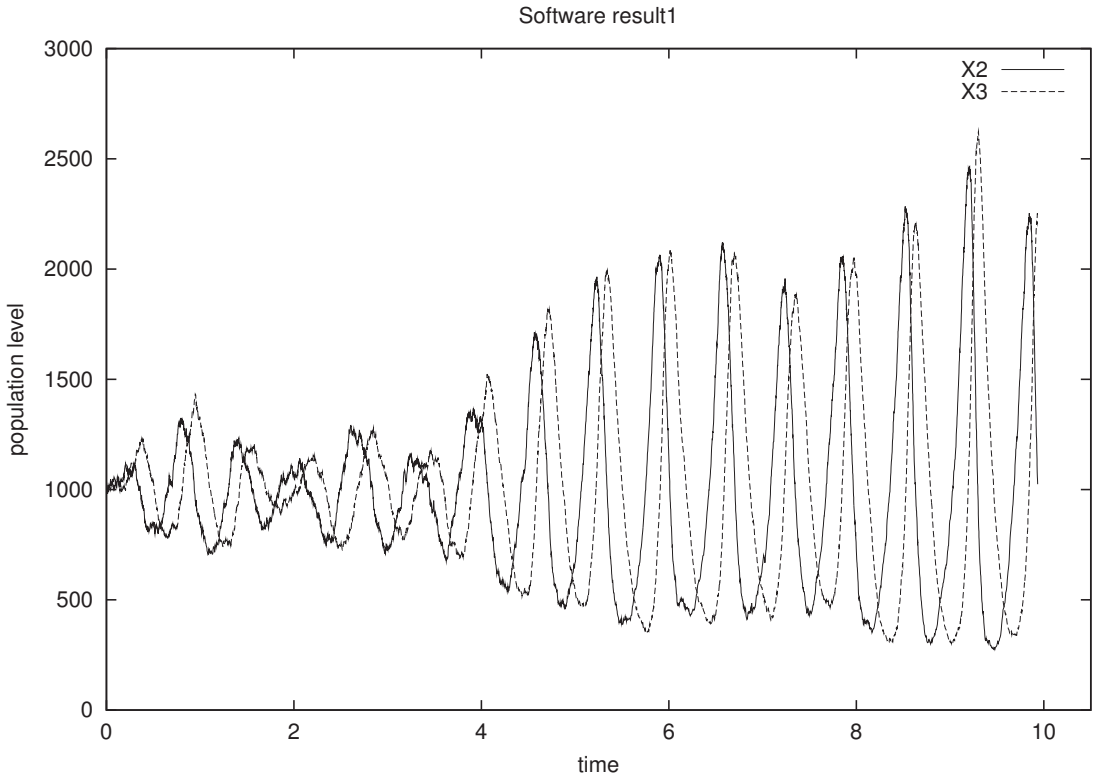



図 5.1: ソフトウェアで出力される分子数変化

このソフトウェアプログラムについて，乱数の種を変えたときにどうなるかを述べる．ソフトウェアプログラムを乱数の種を変えて実行した時のグラフを図 5.2 に示す．

乱数種を変えるとシミュレーション結果にも変化が現れることがわかる．Gillespie のアルゴリズムは確率モデルを用いて化学反応モデルのシミュレーションをするものであるため，サイクルごとに異なる乱数が与えられるならば，異なる結果が現れる．

シミュレーション結果は起こる反応に変化が現れるだけでなく，シミュレーションの時間も変化する．図 5.1 と図 5.2 は同じ反応を 50 万サイクルの間シミュレーションを実行した結果であるが，50 万回反応が発生するまでの時間は異なっている．

図 5.3 と図 5.4 で 50 万サイクルにかかる時間に違いが発生している．最初のシミュレーションをシミュレーション A，次のシミュレーションをシミュレーション B とすると，50 万サイクル経過の時間について，A は 9.936，B は 10.143 である．このように，次に反応が発生するまでの時間 $\tau$ も乱数を用いて計算するため，シミュレーションを特定サイクル実行しても，シミュレーション時間が異なることがわかる．

## 5.2 Vertex-II への実装結果

現在，Gillespie のアルゴリズムを用いて基本的な Lotka 反応モデルのシミュレートを行うために，4.3 章で述べた設計・実装法を用いて Lotka 反応セットモジュールの論理合成と Vertex-II への

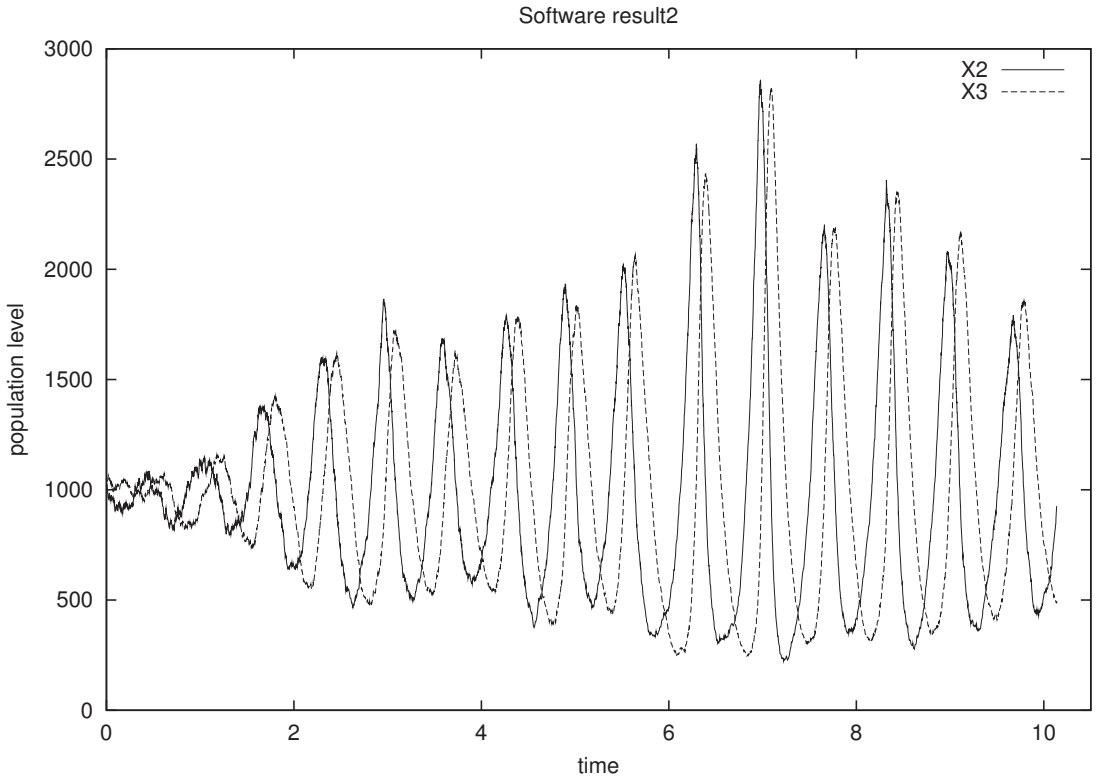



図 5.2: ソフトウェアの乱数種を変えたときの変化

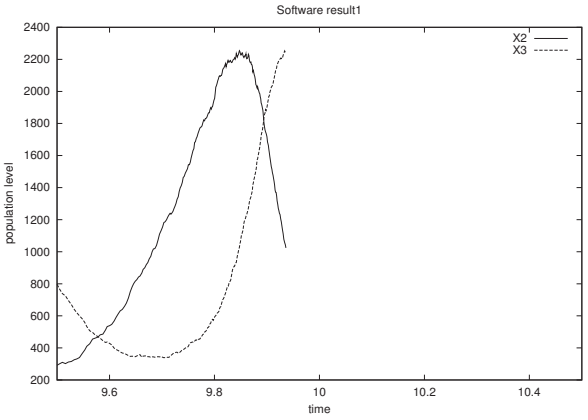



図 5.3: Software result1 の末端

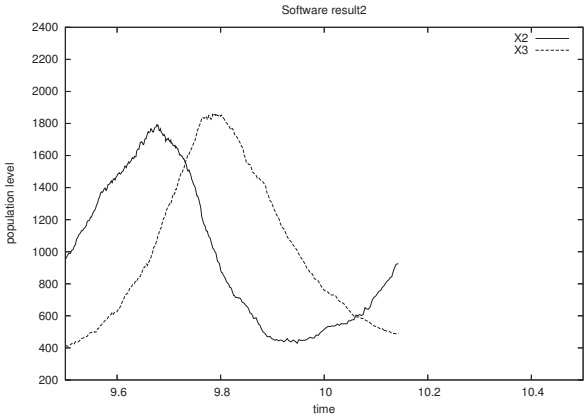



図 5.4: Software result2 の末端

配置配線の結果を述べる．

HDL の記述は Verilog-HDL を使用し，論理合成/配置配線は Xilinx 社の ISE6.1.03i を用いて行った．

Lotka 反応セットモジュールに入出力モジュールを加えた状態で論理合成/配置配線を行った場合のリソース使用量および動作周波数を表 (5.2) に示す．

表 5.2: 入出力付き Lotka 反応セットの合成結果

| スライス数 | 組み込み乗算器 | ブロック RAM | 動作周波数 |
|---|---|---|---|
| 8376 (24.79%) | 60 (41.67%) | 66 (45.83%) | 78 [MHz] |

この Lotka 反応セットモジュールは 37 クロックを 1 サイクルとして，1 クロックごとに分子数 $X_2$，$X_3$ および反応発生までの時間の逆数 $1/\tau$ を 2 組ずつ出力する．このことから，Lotka 反応モデルについてのシミュレーションを $37 \times 2 = 74$ 個並列に実行することが可能である．

Lotka 反応セットに含まれている各モジュールの合成結果を以下の表 (5.3) に示す．

表 5.3: 各モジュールの合成結果

| モジュール | スライス数 | 組込乗算器 | BlockRAM | 動作周波数 | パイプライン |
|---|---|---|---|---|---|
| 単精度浮動小数点加減算器 | 384 (1.13%) | 0 (0%) | 0 (0%) | 83 [MHz] | 5 段 |
| 単精度浮動小数点乗算器 | 159 (0.471%) | 4 (2.78%) | 0 (0%) | 110 [MHz] | 7 段 |
| 整数乗算器 | 106 (0.313%) | 3 (2.08%) | 0 (0%) | 78 [MHz] | 4 段 |
| Lotka 反応処理モジュール | 97 (0.287%) | 0 (0%) | 0 (0%) | 97 [MHz] | 1 段 |
| 一様乱数生成器 (15 ビット) | 16 (0.0473%) | 0 (0%) | 0 (0%) | 94 [MHz] | 1 段 |
| 一様乱数生成器 (23 ビット) | 16 (0.0473%) | 0 (0%) | 0 (0%) | 87 [MHz] | 1 段 |
| 対数テーブル | 348 (1.03%) | 0 (0%) | 60 (41.67%) | 79 [MHz] | 1 段 |
| 単一の Lotka 反応モジュール | 3974 (11.76%) | 30 (20.83%) | 4 (2.78%) | 80 [MHz] | 37 段 |
| 入出力付き Lotka 反応セット | 8376 (24.79%) | 60 (41.67%) | 66 (45.83%) | 78[MHz] | 37 段 |

## 5.3 シミュレーション結果

Lotka 反応セットモジュールを Verilog-HDL で記述し，シミュレーションを行った結果を以下に述べる．シミュレーションは synopsys 社 vcs を用いて行い，シミュレーションログファイルを出力した．シミュレーションログを perl5.6.1 で 74 のファイルに分割し，Gnuplot を用いてグラフ化した．シミュレーションログで出力されるシミュレーション結果は $X_2, X_3$ の分子数，およびその反応までの時間の逆数 $1/\tau$ であるため，ログファイルをシミュレーションごとのファイルに分割する際に逆数を取ることで $\tau$ を求め．サイクルごとに $\tau$ を加算し，その分子数はシミュレーション時間でいつの状態なのかを計算している．初期の分子数 $X_\nu$ と，システム温度と分子の物理的性質のみで決定する確率反応速度定数 $c_\nu$ は以下の値を使用している．

$$c_1 = 0.0002 \quad c_2 = 0.01 \quad c_3 = 10 \quad c_4 = 10$$
$$X_1 = 100000 \quad X_2 = 1000 \quad X_3 = 1000 \quad X_4 = 0$$

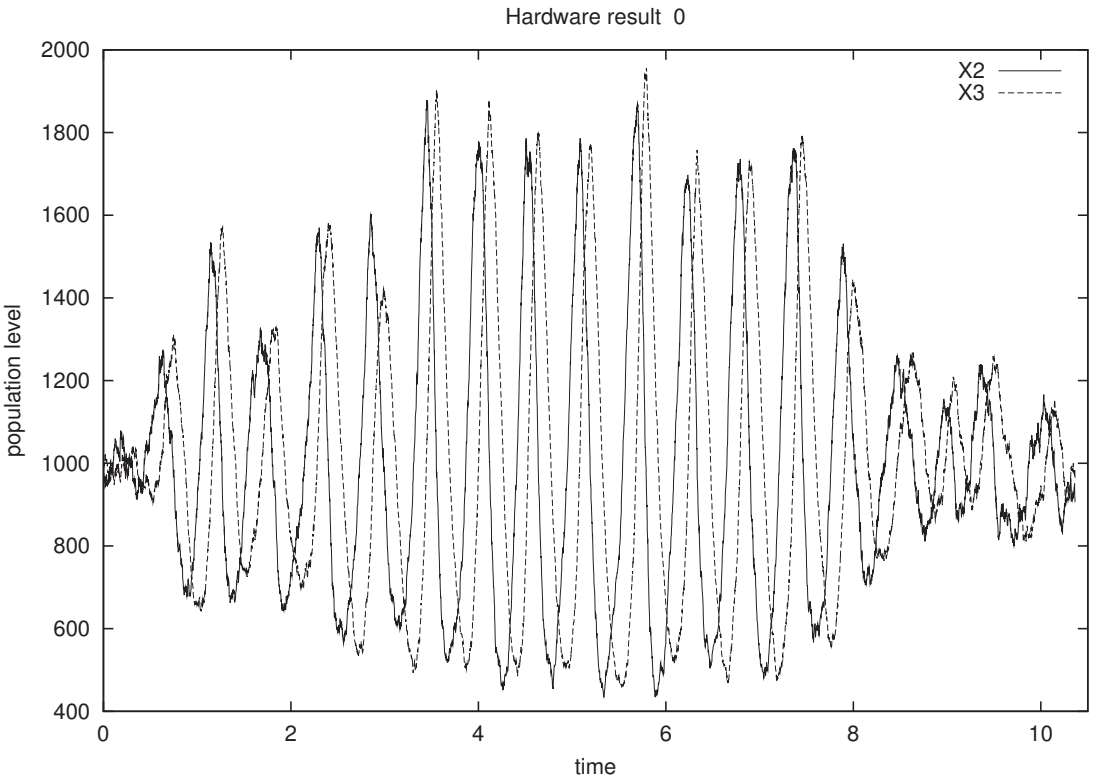



図 5.5: 実装した Lotka 反応セットモジュールのシミュレーション結果

図 5.5 は Lotka 反応モジュール A のうちの 1 つについて，50 万サイクルを実行したときのグラフである．実装した Lotka 反応セットモジュールは，これと同時に 74 個のシミュレーションを行っている．それらの結果の一部を図 5.6 に示す．

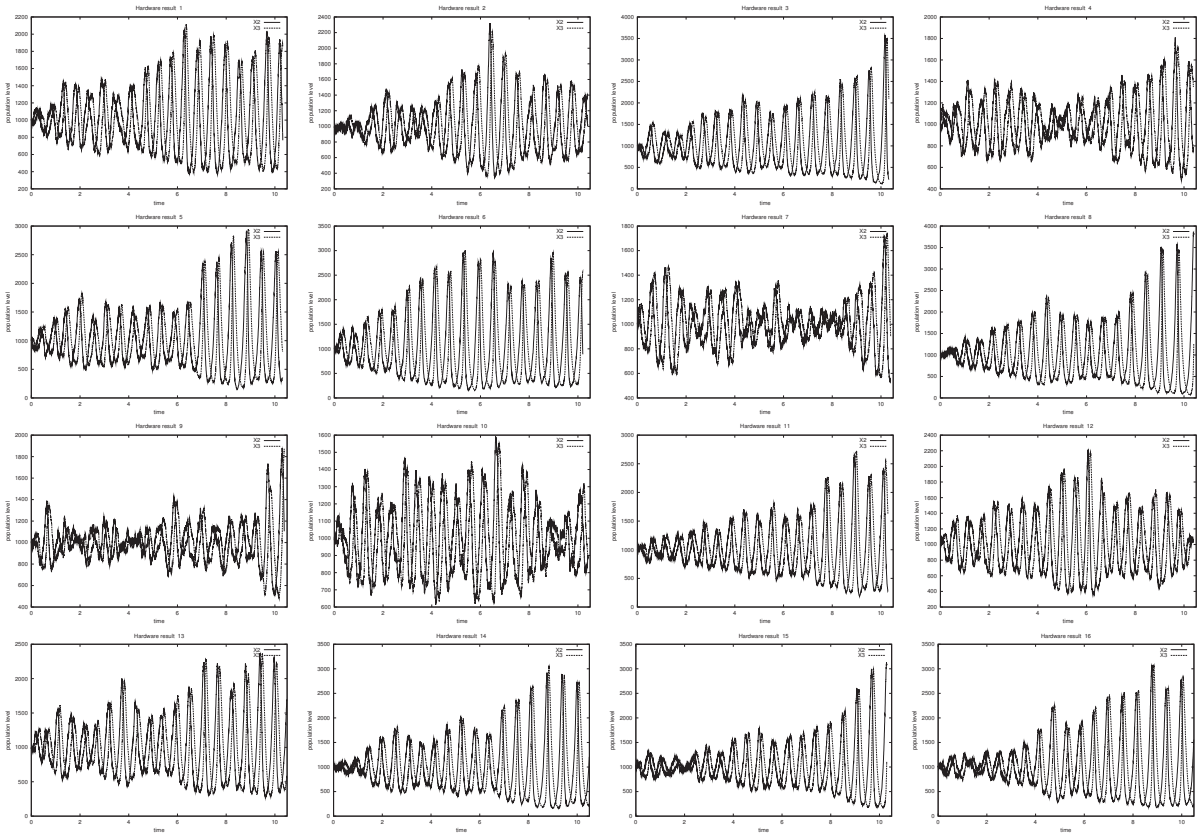



図 5.6: 並列動作する Lotka 反応シミュレーションの一部

### 5.3.1 ハードウェアとソフトウェアの動作比較

Lotka 反応セットモジュールが正常に動作し，それとソフトウェアによるシミュレーション結果が一致することを確認する．

図 5.1 や図 5.5 より，ソフトウェア，ハードウェアともに，$X_2$，$X_3$ の数が振動していることがわかる．しかし，ソフトウェアでは標準ライブラリ，ハードウェアでは M 系列と，乱数生成規則が異なり，また，ハードウェアの浮動小数点演算器には丸め機能が無い．それらが出力される結果にどのような影響を及ぼすかを述べる．

ハードウェアとソフトウェアで乱数生成規則が異なるため，初期値や乱数種に同じ値を設定してシミュレーションを実行しても，少数の試行の結果だけでは，同じ形状の波が形成されることはない．乱数生成規則の有効性の有無については，今後検証を行っていくこととし，ここではハードウェアで実装したシミュレータが正しい動作をしていることを述べる．

このことを確認するため，ソフトウェア側にハードウェアと同等の 32 ビット M 系列乱数生成関数を実装した．シミュレーションにおける $\tau$ は，$a_0$ と乗算した後に出力され，以降のシミュレーションで再び使用されることはない．そのため，$\tau$ による誤差の蓄積は無い．

しかし，(0, 1) の実数から求められる $\mu$ は，発生する $R_\mu$ の決定に関わり，反応によって分子数が変更され，それが次の $a_0$ の決定に使用されるため，ここで誤差が発生した場合，反応の進展に合わせて誤差が伝搬してしまう．丸め誤差の有無による誤差伝搬はここに現れる．

10 万サイクルの Lotka 反応モデルについて，乱数生成規則を M 系列とし，乱数種に同じ値を与え，使用する乱数列を等しくした上で，ソフトウェアとハードウェアの両実装を比較した．

Lotka 反応モデルの 10 万サイクルの反応について，ソフトウェアによるシミュレーション結果を図 5.7 に，ハードウェアによるシミュレーション結果を図 5.8 に示す．

ハードウェアのシミュレータには浮動小数点除算器が実装されていないため，Perl を使用し，シミュレーションログから $1/\tau$ の値を取り出し，逆数を取ることで $\tau$ の値を計算している．

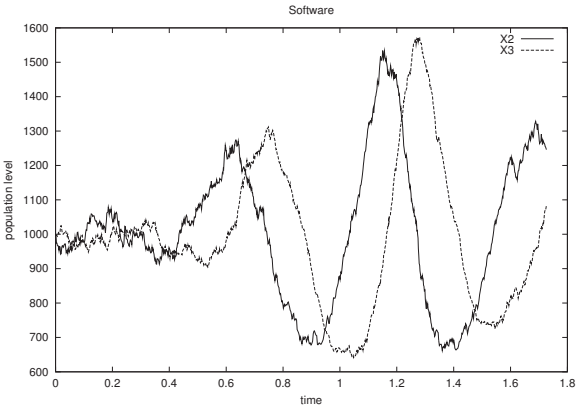



図 5.7: ソフトウェアによる実行結果

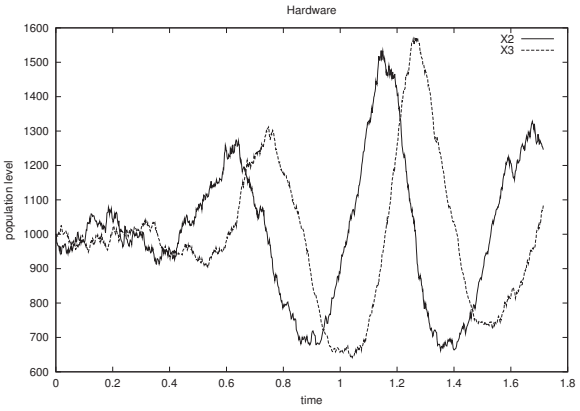



図 5.8: ハードウェアによる実行結果

ソフトウェア，ハードウェアのシミュレーションログから，発生する反応を示す $\mu$ の値はシミュレーションを通して同一であり，与えられる乱数列が同一の場合，丸め誤差による発生反応の変化，およびその誤差伝搬は生じていない．反応発生時間 $\tau$ は，対数テーブル内のデータは C で作成されたことから，対数テーブルからデータを読み出す乱数生成器側が同じであるので，ハードウェア，ソフトウェアともに同じ値が出力されている．$\tau$ は $a_0$ との乗算およびログ出力後の除算により，出力される値に小さな違いがある．しかし，$\tau$ は誤差伝搬しないため，シミュレーションを通して結果が大きく異なることはない．

このことから，ソフトウェアで実行するシミュレーションとハードウェアで実行するシミュレーションに大きな誤差は発生していないことが確認できた．

シミュレーションの結果が正しいことを判断するためには，これらのシミュレーションを多数繰り返し，平均や分散を取って調査する必要がある．

## 5.4 性能評価

実装した Lotka 反応セットは，37 クロックで 1 つの反応を処理することが可能である．Lotka 反応セットモジュールを 78[MHz] で動作させたときのシミュレーション 1 つあたりの処理時間は，78[MHz]/37 = 2.11M[cycle/sec] となり，Pentiun4 2.53GHz とほぼ同等の処理速度を実現できる．

また，Lotka 反応セットモジュールはパイプライン化されており，37 個のシミュレーションを同時に行うことが可能である．さらに，内部には Lotka 反応モデルモジュールが 2 つ構成されて

おり，同時に動作させることができる．つまり，Lotka 反応セットモジュールは $37 \times 2 = 74$ 個のシミュレーションを同時に処理することができる．そのため，Lotka 反応セットモジュールの全てを使って処理を行った場合，$74 \times 2.11 = 156.14$M[cycle/sec] の処理速度を実現できる．これは，ソフトウェアで実行したときにもっとも高速だった AthlonXP 2800+の 53.84 倍である．

# 第6章

# 結論

## 6.1 確率モデルを FPGA でシミュレーションする有効性

本研究では，FPGA を用いた生物学的計算向けのプラットフォームである ReCSiP をターゲットとして，Gillespie のアルゴリズムを用いた確率モデルによる代謝シミュレーションコアの開発を行った．Gillespie が例示している確率モデルから Lotka 反応モデルをシミュレーションする回路を実装し，その評価を行った．その結果，Lotka 反応モデルについて AthlonXP2800+の 53.84 倍の速度でシミュレーションが可能であることがわかった．生物学計算をはじめとしたモンテカルロシミュレーションは大きな並列性を持つため，ReCSiP ボードのような FPGA による並列化，高速化の有効性を示すことができた．

## 6.2 今後の課題

### 6.2.1 除算器の実装による並列化

面積，BlockRAM，組み込み乗算器に余裕があるため，Lotka 反応セットモジュールを ReCSiP 上を 2 つ構成することも可能である．現行の実装では Lotka モジュール 1 つにつき 2 つの SRAM へ結果を出力しているため，Lotka 反応セットモジュールが 1 つに限られてしまう．この解決法として，浮動小数点除算器を実装することで出力を間引くことが考えられる．結果格納メモリへの入出力を取り除くことによって，組み込み乗算器の数の上限まで，Lotka 反応セットを 2 つ ReCSiP 上に実装することが可能である．このため，最大で $37 \times 2 \times 2 = 148$ 個のシミュレータを同時に実行することが可能になる．

また，浮動小数点除算器により，Gillespie の例示している Lotka 反応以外の “複雑な反応モデル” も実装でき，より幅広いモデルのシミュレータを構築することが可能になる．

### 6.2.2 精度向上

#### 対数テーブル

現行のシミュレータでは，対数テーブルが 32 ビット幅 15 ビットエントリであり，格納データ数が $2^{15} = 32768$ 個しかなく，次の反応までの時間 $\tau$ の計算が雑なものになっていることは明らかである．最大で $2^{15}$ の周期の実数の対数は乱数として用いるには不十分である．この問題に対して，ホスト PC との協調処理を行って解決を図る方法が考えられる．

すなわち，$\tau = \ln(1/r_1)/a0$ のうち，$1/\ln(1/r_1)$ を計算する役割をホスト PC が担当する処理を行う．このことによって，ハードウェアによる乱数生成に比べ，ホスト PC 側のソフトウェアで乱数生成アルゴリズムを使用し，信頼性の高い乱数を得ることが可能になる．実装方法は，以下のように対数テーブルを複数に分割し，読み出しを行うテーブル，値の更新を行うテーブルを構成することが考えられる (図 6.1)．読み出し，更新のテーブルを一定クロックごとに切り替えるセレクタを付加することで，擬似的に (0, 1) の 32 ビット浮動小数点乱数を得ることが可能になる．この場合，ホスト PC から書き込まれる値もランダムになるため，15 ビットの乱数生成器は不要になり，また浮動小数点減算器も不要になる．

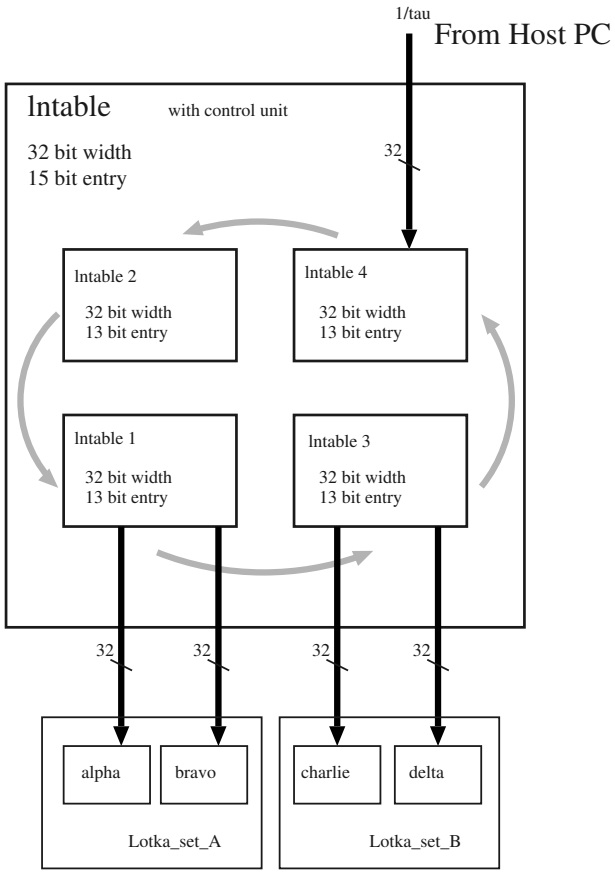



図 6.1: ホストと協調動作する対数テーブル

### 乱数生成器の改良

乱数生成器は現在 32 ビットの LFSR による M 系列疑似乱数生成器である．この乱数生成期は最も長いものでも周期が $2^{32} - 1$ であり，複雑なモデルの数百万反応を超えるシミュレーションを視野に入れると，現行の乱数生成器では対応できないことが容易に予想できる．開発中の ReCSiP2 では自然乱数生成器が搭載される予定だが，自然乱数生成器は動作周波数が低く，全ての乱数の生成には利用できない．そのため，自然乱数生成器を使って乱数の種を生成し，そこから疑似乱数を作っていくことが考えられる．ハードウェアでの乱数生成は M 系列を用いたものが主であるが，この場合も，LFSR を 90 ビット程度の長いものを使用し，その LFSR を複数並べ，それらか

ら 8 ビットずつ読み出し，それらを組み合わせて 32 ビットの長さの乱数を生成する，といった方法で，乱数の周期性を伸ばす手法を取り，十分な検証を行う必要がある．図 6.2 にその一例を示す．これは，90 ビットの LFSR を 4 つ使用し，毎クロック最上位 8 ビットを取り出し，それらを組み合わせることで 32 ビットのビット列を生成するモジュールである．

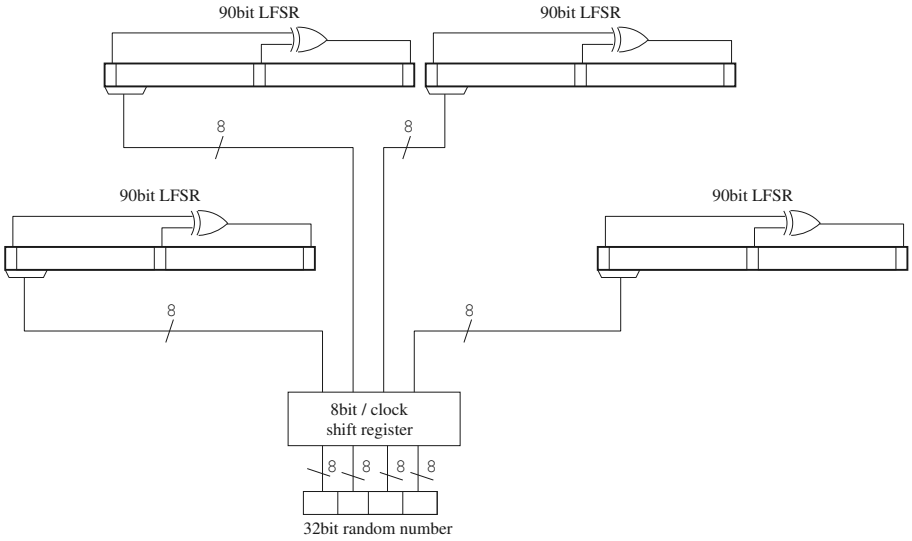



図 6.2: より周期の長い乱数生成器の一例

### 6.2.3 モデル自動生成

Lotka 反応モデルを実装したことで，化学反応モデルのモンテカルロシミュレーションは ReCSiP のような FPGA を搭載したデバイスで大きな効果が期待できることがわかった．Gillespie は Lotka 反応モデル以外にも複雑な化学反応モデルを例示しているし，STOCKS のようなシミュレータは入力ファイルを記述することで細胞のシミュレーションに対応している．

Gillespie のアルゴリズムのシミュレーションコアは，モデルごとの反応処理モジュールに加え，面積的に多くを占有する算術演算器が以下の数だけ必要になることがわかっている．反応分子の組み合わせ数を $x$ とすると，$h_\nu$ を決定するための整数乗算器を $x$，$h_\nu$ と確率速度定数 $c_\nu$ を乗じるための浮動小数点乗算器を $\nu$，それらを累積加算するための浮動小数点加算器が $x-1$ 必要になる．

ReCSiP の Vertex-II XC2V6000 に，反応処理モジュールおよびこの数の算術演算器が搭載できる範囲内であれば，Lotka 反応セットモジュールで示したように各サイクルをパイプライン化し，動作中は全てのモジュールの使用率が 100% でモジュールを有効に使用することが可能であり，さらに大きな並列性を得ることが期待できる．搭載できないほど大きな処理の場合は，演算器や使用モジュール数を制限した上で，それらのスケジューリングを行い，並列化を図る予定である．

モデルに合わせたシミュレーションモジュールを自動生成することから始め，将来的には，SBML(Systems Biology Markup Language) で記述されたモデルから確率モデルシミュレータの Verilog コードを生成できるように展開する予定である．

# 謝辞

本研究の機会を与えてくださり，また常にご指導，ご助言をいただきました
天野英晴教授
に，深く感謝いたします．毎回のアドバイスが構想の発展に繋がり，研究を続けるための大きなモチベーションになりました．

配属後の研究初期段階から多くの面でお世話になり，また論文チェックも行ってくださった
長名 保範氏
に感謝致します．実装の楽しさを学びました．

実装の良否について多くの助言をいただき，また論文の査読を行ってくださった
福島 知紀氏
福田 静人氏
に、感謝致します．たくさんの画期的なアイデアを発想できました．

研究を進めるにあたり，生物学分野，ソフトウェア工学分野からの多くの有益な知識を頂きました，現情報工学科斉藤博昭研究室，生命情報学科榊原研究室の
松井 洋氏
に感謝致します．

日々共に作業し，励ましてくださった
住吉 正人氏
黒瀧 俊輔氏
に感謝します．深夜に寂しくて泣き出さなかったのはお二人がいてくれたおかげです．

日々の研究に際し，多くの助言と，生活のアドバイスをいただいた
天野研究室の全てのみなさま
に感謝します．この１年間で知識，サバイバビリティともに大きく成長できました．

家族から，祖母の
川上 榮子氏
に感謝します．彼女の有形無形の支援が無ければ，私はここまで来られませんでした．ありがとう．

今まで私と関わり，ここまで引っ張ってくれた全ての皆様に限りない感謝を．
ありがとうございました．

# 参考文献


[1] Bruce A. Barshop, *et al.* Analysis of numerical methods for computer simulation of kinetic processes: Development of kinsim — a flexible, portable system. *Analytical Biochemistry*, Vol. 130, pp. 134–145, 1983.

[2] Mendes P. Gepasi: a software package for modelling the dynamics, steady states and control of biochemical and other systems. *Computer Applications in the Biosciences*, Vol. 9, No. 5, pp. 563–571, Oct. 1993.

[3] I Goryanin, *et al.* Mathematical simulation and analysis of cellular metabolism and regulation. *Bioinformatics*, Vol. 15, No. 9, pp. 749–758, Sep. 1999.

[4] Masaru Tomita, *et al.* E-cell: software environment for whole-cell simulation. *Bioinformatics*, Vol. 15, No. 1, pp. 72–84, Jan. 1999.

[5] J. Schaff, *et al.* A general computational framework for modeling cellular struc ture and function. *Biophysical Journal*, Vol. 73, pp. 1135–1146, Sep. 1997.

[6] Daniel T. Gillespie, *et al.* Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, Vol. 81, No. 25, pp. 2340–2461, Dec. 1977.

[7] Andrzej M. Kierzek. Stocks: Stochastic kinetic simulations of biochemical systems with gillespie algorithm. *Bioinformatics*, Vol. 18, No. 3, pp. 470–481, Mar. 2002.

[8] Nicolas Le Novère, *et al.* Stochsim: modelling of stochastic biomolecular processes. *Bioinformatics*, Vol. 17, No. 6, pp. 575–576, Jun. 2001.

[9] 長名保範ほか. Reconfigurable hardware による細胞シミュレータの高速化手法. ARC2002 148-8, 情報処理学会, May. 2002.

[10] Carl Firth, Nicolas Le Novère, and Tom Shimizu. *STOCHSIM,the stochastic simulator*. ftp://ftp.cds.caltech.edu/pub/dbray/, 2001.

[11] Stephen F, *et al.* Basic local alignment search tool. *Journal of Molecular Biology*, Vol. 215, pp. 403–410, Oct. 1990.

[12] Jeffrey M. Arnold, Duncan A. Buell, and Elaine G. Davis. Splash 2. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pp. 316–322. ACM Press, 1992.

[13] Ebisuzaki T, *et al.* Grape: special purpose computer for simulations of many-body systems. In *Proceedings of the twenty-sixth Hawaii International Conference on System Science*, Vol. 1, pp. 134–143. IEEE, Jan. 1993.

[14] 田村友紀ほか. 細胞系譜構築システムの reconfigurable system による高速化. VLD2001 122, 信学報, Jan. 2002.

[15] PCI Special Interest Group. *PCI Local Bus Specification Revision 2.3*, Oct. 2001.