博 士 論 文

FPGAを用いた生化学シミュレーションに関する研究

2005年度

慶應義塾大学大学院理工学研究科

長名 保範

論文要旨

1953 年に Watson と Crick によって DNA の 2 重らせん構造が明らかにされて以来、生命システムを分子によって秩序的に構成された機械として捉え、その構造を研究することで生命現象の仕組みを理解しようとする研究が盛んになった。その結果として、多くの生物のゲノムが解読されたり、細胞内におけるさまざまな物質の相互作用が解明されてきており、生命のシステムとしての挙動を解明していくことが今後の生命科学の中心的課題である。

これには、実験で得られたデータの分析や、それによって構築された数理モデルのシミュレーションなどで計算機を利用することが必須であり、分析やシミュレーションの高速化のために多くの研究機関が PC クラスタをはじめとする並列システムを導入している。しかし、旧来の並列システムは高価であるため研究者個人のための計算環境として利用することは困難であり、個人レベルで利用可能な小型・低価格かつ高性能な計算エンジンの開発が待たれている。

そこで本研究では、生命科学の研究に用いられるアプリケーションのひとつとして、細胞内の代謝系などの生化学モデルの挙動をシミュレーションする生化学シミュレータに着目し、これを回路を再構成可能な LSI である FPGA (Field Programmable Gate Array) を用いて高速に実行する手法を開発した。本研究の主な目的は、FPGA を用いて生化学シミュレーションを行う場合に基本的な問題となる点を解決して、高速なシミュレーション環境を実現することである。第一の問題点は、モデルに含まれるさまざまな種類の反応速度式を解けなければならない点であり、第二の問題点はホスト CPU と PCI バスに接続した FPGA の通信がボトルネックにならないようにシステムを設計しなければならない点である。

これらの問題点について、本研究で開発した生化学シミュレータ ReCSiP (Reconfigurable Cell SImulation Platform) は、反応経路マップを FPGA 側で処理することにより、PCI バスのボトルネックを回避し、反応速度式を解くモジュールをモデルに応じて組み替えて利用することでさまざまな反応速度式に対応することを可能にした。モジュールの組み合わせによって与えられたモデルに対して柔軟に対応するとともに、モデルごとに最適化された回路を用いて最高のスループットを実現することが可能である。また、FPGA 上に構成される各モジュールはパイプライン化されており、FPGA 上に分散配置されたメモリブロックから並列かつ連続的にデータを供給することで高い性能を実現する。

本論文では研究背景について概説したあと、反応速度式を処理するモジュールや積分を行うモジュールなどのシミュレーションをするための基本的なハードウェアの構成について述べ、それらの性能評価の結果を示す。性能評価の結果として、本研究で開発された手法はパラメータ探索等に有効な計算手段であることと、Xilinx 社の FPGA である XC2VP70 を用いた場合にマイクロプロセッサ (Intel Pentium4 3.2GHz) 比で 10 倍から 50 倍程度の実効性能を得られることが明らかになった。

Abstract

Biochemical simulators are now essential for understanding of cellular systems. Cellular systems are modeled as a network of substances and reactions, and the models are refined in repetition of experiments and simulations. Reaction pathways are expressed as set of ordinal differential equations (ODEs), which have concentrations of molecular species as their variables and kinetic parameters as their coefficients. Since it's not possible to know the complete pathway, concentration and kinetic parameters from experiments, simulations have to be run again and again to find an ideal parameter set to present reasonable behavior.

Parameter optimization process is a time consuming task, and usually takes over 1 week. Current solutions for parameter optimization are parallel systems, such as PC/WS clusters or SMP systems. However, computers are still bottleneck for biologists, because expensive parallel systems can't be the "personal" computing environment for them. To relax this bottleneck, a cheaper and smaller solution for biocomputing is required.

This research introduces an efficient solution for acceleration of biochemical simulations, by using an FPGA as the simulation engine. FPGA can provide optimized hardware and performance, but it's usually difficult to use. Mostly common problems with FPGA-based biochemical simulations are: 1) limited bandwidth of PCI bus between the FPGA and host processor, 2) existence of various rate-law functions, and 3) users without any knowledge in hardware design. The method developed here provides solutions to these problems by modules by 1) describing reaction pathway by pointers among memory blocks on FPGA, 2) modular design of various rate-law functions, and 3) software-friendly design of hardware modules.

This thesis describes the background of this work, basic structure of the FPGA-based biochemical simulator ReCSiP, and the result of its basic performance evaluation. Result of the evaluation showed that ReCSiP is suitable for parameter optimization, by exploiting the potential of its deep-pipelined structure. The peak performance gain reached to 80-fold speedup, and its effective throughput was 10 to 50-fold speedup compared to Intel's Pentium4 processor at 3.2GHz. This result shows that FPGA is now hopeful candidate as the engine for future biocomputing.

目 次

表 目 次

図目次

第1章　緒論

近年の半導体設計・製造技術の進歩により、90nm プロセスや 65nm プロセスによるチップが量産出荷されるなど、面積あたりの回路規模は急速な増大を続けている。これらの微細なプロセスを用いて大規模な ASIC を開発するには高額の開発費と長い開発期間が必要であり、電気的な面においてもアナログ的な要因によるシグナルインテグリティの問題など多くの困難を伴う。しかし、FPGA (Field Programmable Gate Array) をはじめとする可変構造デバイスは微細化に伴う回路容量の増大により、従来の glue logic としての位置付けから、マイクロプロセッサやコントローラ、インタフェイスなどを含めた、システム全体を単一のチップで実現するキーコンポーネントへと変貌を遂げようとしている。

これに伴って、FPGA の科学技術計算分野への応用も大きく様相を変えようとしている。FPGA を用いて高速な計算を実現する研究は、FPGA が商用化された頃から行われているが、回路容量の制約により、整数演算を用いるもの、あるいは固定小数点形式を用いた実数演算に限定されてきた [1][2][3][4]。しかし、Xilinx 社の Virtex-II シリーズ [5][6]、Virtex-4 シリーズ [7] や Altera 社の Stratix シリーズ [8][9] のような、近年の大規模な FPGA は大きなロジック容量とともに、ハードマクロとして乗算器や従来製品よりも豊富なメモリブロックを組み込むなど、複雑なアプリケーションにも対応できる構成になっている。そこで、これらのデバイスによる浮動小数点演算を用いた大きなアプリケーションの研究・開発が活発化しており、分子動力学や天体間の重力問題 [10] などの大規模な問題を FPGA によって高速に処理するシステムが数多く発表されている。

一方、生命科学の分野においては、1953 年の Watson と Crick による DNA の 2 重らせん構造の発見 [11] 以来、生物を分子機械と捉えて、そのメカニズムを解明しようとする研究が盛んになり、多くの生物のゲノムが解読された。また、質量スペクトル分析に代表されるような新しい実験機材・実験手法の開発により、細胞内に存在する各種物質の濃度を、経時的かつ定量的に測定する操作 [12][13] が可能になってきている。これらの実験から得られる大量のデータを計算機上で効率よく処理し、種々の生命現象を計算機上で数理的にモデル化・シミュレーションすることは、生命科学の進歩にとってもはや欠くべからざるプロセスである。

生命科学における計算機の活用分野は非常に広範囲に及んでいる。たとえば DNA やアミノ酸などの配列の相同性検索、分子動力学によるタンパク質の構造と作用の分析や、DNA マイクロアレイや顕微鏡から得られる大量の画像の処理とそこからの知識抽出、遺伝子転写制御や代謝回路のシミュレーションによるシステムの動態解析、さらには各種のモデルや配列などのデータベースにまで及ぶ [14]。

しかしながら、これらのデータ分析・シミュレーションなどのアプリケーションは、いずれも大きく複雑な構造のデータを取り扱うものであり、計算能力の不足が顕著な問題となっている。現在のところ、単一のマイクロプロセッサでは処理能力が不足する場合には、PC クラスタを用いた並列処理、あるいはグリッドコンピューティング [15] などの並列システムによる解決が図られている。しかし、このような従来型の並列計算機による解決では、計算システムの価格や消費電力

等の面から、研究者個人が所有または占有することは困難である。したがって各個人に充分な計算能力が与えられているとは言い難いのが原状であり、これを打開するためには、マイクロプロセッサによる並列処理からの脱却を伴う計算機アーキテクチャ的なブレイクスルーが必要である。

そこで本研究では、生命科学の研究に必須なアプリケーションのひとつとして生化学シミュレータに着目し、これを FPGA により高速化することを試みた。これは、細胞内の生化学反応経路を数理的にモデル化し、細胞内に存在する各種の物質の濃度変化を経時的に求め、これらの反応系の動態をシミュレーションするものである。

生化学シミュレータは 1970 年代から各種開発されており、小さな反応系の動態解析を目的としていた初期の KINSIM[16] や Gepasi[17] などから、近年では細胞全体の動態解析を目指す E-cell[18] や The Virtual Cell[19][20] など、さまざまなものが開発されている。これらのシミュレータは、経時的なシステムの挙動の計算に時間を要するのは無論のこと、実験結果と一致する挙動をモデルに与えるためには広大なパラメータ空間の探索を行うことになり、シミュレーションを高速に実行できる計算環境が必要である。

計算を高速化するには、専用ハードウェアを開発するというアプローチが性能面で有利であるが、これには常に柔軟性と開発コストの問題が伴う。しかし、書き換え可能なデバイスである FPGA を用いて計算を高速化するアクセラレータを開発すれば、専用計算機と同じようにハードウェアで直接問題を処理することによる高速性を実現しながらも、同じハードウェアを用いて他のアルゴリズムによるシミュレーションや、他の問題を高速処理を行うことが可能である。また、生化学シミュレーションというアプリケーションに分野を絞って考えた場合にも、シミュレーション対象のモデルによってハードウェアを最適化し、常に最大限の計算能力を発揮することができる。

本研究で開発された ReCSiP[21][22] はこのような背景に立って設計された FPGA ベースの生化学シミュレータであり、FPGA の柔軟性や、多数のメモリブロックに同時アクセスすることよるデータ並列性、深いパイプライン構成による高スループットな計算処理によって、数十台規模の PC クラスタの代替となりうる計算能力を提供することを目指している。現在までに基本的な方式とハードウェアが開発されており、反応速度式を解く処理の理論的なピーク性能は FPGA (Xilinx 社 Virtex-II Pro XC2VP70) の全面積を使用し、積分に Euler 法を用いた場合で 1 秒あたり 540×10^6 反応、実際に Minimal Mitotic Oscillator モデル [23] をマッピングしてシミュレーションを行う場合に、FPGA の約半分の面積を使って 1 秒あたり 39.6×10^6 反応のシミュレーション能力を発揮できることを確認した。これはマイクロプロセッサ比で約 6.5 倍の速度向上であるが、FPGA の全面積を用いることと、別途提案・実装された実効性能の改善手法により、これをさらに 4 倍の実効性能に改善することができる。これにより、この方式でシミュレーションを行った場合には、一般的なマイクロプロセッサの 10 倍〜50 倍程度の性能が期待できることになる。

本論文の第 2 章では生化学シミュレーションのための基本的な数理モデルと、FPGA を用いた高性能システムについて触れ、第 3 章では ReCSiP の構成について述べる。続いて第 4 章で性能評価と議論を行い、第 5 章で結論を述べる。

第2章　背景・関連研究

本章ではまず、計算機によるモデリング・シミュレーションを実験と組み合わせることで生命現象の理解を目指すシステム生物学のアプローチについて触れ、本研究の対象である生化学シミュレーションが取り扱うモデルがどのようなものであるかについて述べる。続いて、本研究で計算エンジンとして利用する FPGA (Field-Programmable Gate Array) とそれを用いた近年の高性能計算システム、また Bioinformatics への応用について述べる。

2.1　システム生物学

システム生物学とは、システムレベルでの生命の理解を目指すアプローチである [24]。生命現象は遺伝子や代謝のネットワーク、細胞内外のさまざまな構造と、細胞間のシグナル伝達などによって成り立っており、これらの複雑なネットワークと構造をシステムとして理解することがこれからの生物学の中心的な課題であると考えられる。

1953 年の Watson と Crick による DNA の 2 重らせん構造の発見 [11] 以来、分子生物学はタンパク質や核酸でできた分子機械としての生命の構造解明を試み、その結果としていくつもの遺伝子やタンパク質の働きや配列・構造が判明している。しかしそれらの働きに関しては個々の部品に関する、比較的単純な場合についてしかわかっていない。しかし、システムとしての生命現象の理解、その動的な挙動の解析や実験が必要であり、実験から得られる大量のデータを効率よく分析し、仮説とモデルを立て、それに基づいた実験を行い、その結果に基づいて再び仮説やモデルを検証するというアプローチが必要である。

したがって、図 2.1 に示すように、システム生物学では従来からの wet な実験に加えて、数理モデリングと計算機シミュレーションを用いた、dry な実験が重要になる。モデリングとシミュレーションの対象分野は多岐にわたっており、微分方程式や確率モデルを用いた代謝回路の研究、ベイジアンネットワークやペトリネットなどを用いた遺伝子制御ネットワークの研究や、細胞膜

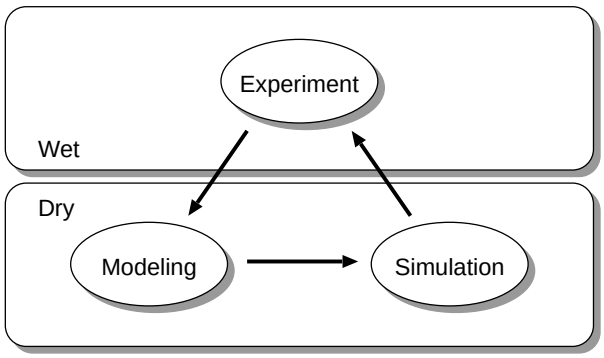


図 2.1　システム生物学のアプローチ

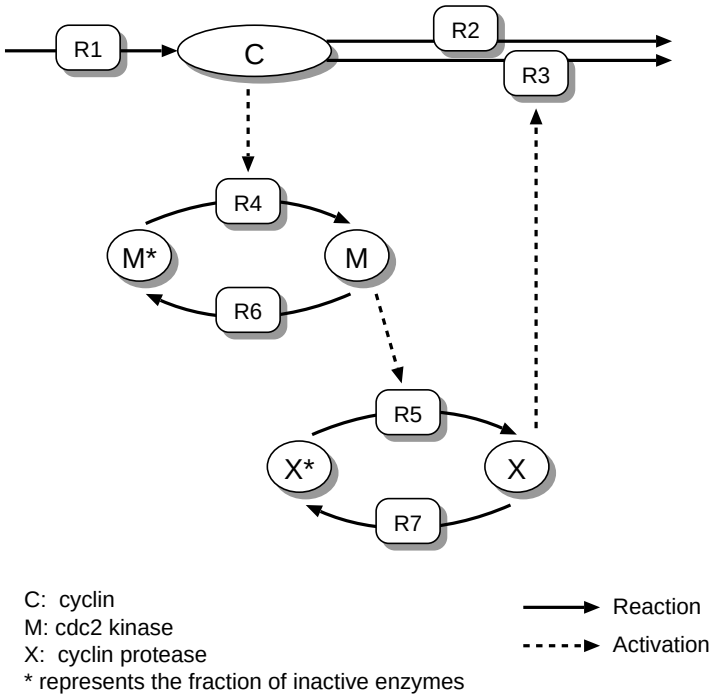


図 2.2 Minimal Mitotic Oscillator モデルの反応経路図

などを力学的にシミュレーションすることによる細胞の機械的な構造の研究などが行われている [14]。いずれも計算コストの大きな問題であり、高性能な計算機が必要とされている。

2.2 生化学反応のモデリングとシミュレーション

前節で述べたように、現代の生命科学においては計算機による様々なシミュレーションが重要な役割を果たすが、計算に要する時間の長さは深刻な問題である。本研究では、代謝回路などの生化学モデルのシミュレーションに着目し、これを高速化することを目的としている。ここでは生化学反応のモデリングとシミュレーションに関して概説する。

2.2.1 生化学反応系モデル

生化学シミュレーションとは、いくつかの化学反応から構成されるシステムが、ある初期状態から出発した場合の、反応系中の物質濃度の経時的な変化を数値的に求める問題である。実験結果をもとにあるシステムのモデルを作り、システム内部のさまざまなパラメータを計算機上で変更することにより、システムの外乱に対する応答の特性などを調べることができる。また、実験とシミュレーションの反復により、より正確なモデルを構築することが可能になり、正確なモデルの構築はより深いレベルでの生命現象の理解を可能にする。

生化学反応系のモデルは、複数の化学反応のモデルの集合体であり、その土台となるのは

- 反応系に含まれる物質のリスト
- 反応系を構成する反応のリスト

の 2 つの要素で構成された反応経路である。図で表せばたとえば図 2.2 (注 1)の例のようになり、物質を表すノードと、反応を表す矢印で構成された図になる。

システムの挙動を知るには反応経路に加えて、各物質の初期濃度と、各反応の反応機構モデル及びパラメータを知る必要があり、これらが与えられれば、反応による物質の増減を数値的に求めることができる。それぞれの物質の消費・生成速度を表す反応速度式は次に述べるように、それぞれの反応機構の反応速度を数理的に表した常微分方程式を用いて記述されるのが一般的であり、システム内の反応を表す式すべてを連立微分方程式として数値的に解くことになる。

反応速度式

化学反応の反応速度 (基質 S の消費速度であり、生成物 P の生成速度) は一般に質量作用則によって表され、一次反応

$$S \xrightarrow{k_1} P \tag{2.1}$$

の反応速度 v は基質 S の濃度を $[S]$ として、

$$v = k_1[S] \tag{2.2}$$

のように記述することができる。二次以上の反応でも同様に、

$$S_1 + S_2 \xrightarrow{k_1} P \tag{2.3}$$

であればふたつの基質の濃度 $[S_1]$、$[S_2]$ を用いて、

$$v = k_1[S_1][S_2] \tag{2.4}$$

のように表現できることが知られている [25]。しかし、細胞内で起きる化学反応には酵素をはじめとして、反応速度を調節するさまざまな因子が複雑に絡み合っており、質量作用則だけですべての反応速度を記述するのは困難である。そのため、各種の反応機構をモデル化した近似式が用いられており、たとえば Michaelis と Menten によるもの [26] が酵素反応のモデルとして有名である。

Michaelis-Menten モデルでは、、基質 S、酵素 E、酵素-基質複合体 ES、生成物 P の反応を式 2.5 のように表現する。k_1、k_2、k_3 は各反応の反応速度定数である。

$$S + E \underset{k_2}{\overset{k_1}{\rightleftarrows}} ES \xrightarrow{k_3} P + E \tag{2.5}$$

この式は、まず酵素 E と基質 S の複合体である ES が形成されて、そこで酵素の作用により生成物 P が生成される、というモデルを表している。これを、個々の反応速度式に展開すると、次の式 2.6〜式 2.8 のようになる。

$$v_1 = k_1[S][E] \tag{2.6}$$

$$v_2 = k_2[ES] \tag{2.7}$$

$$v_3 = k_3[ES] \tag{2.8}$$

(注 1) このモデルに関しては第 2.2.2 節で解説する。

さらに各物質の濃度変化はこれを用いて

$$\frac{d[S]}{dt} = -v_1 + v_2 = -k_1[S][E] + k_2[ES] \tag{2.9}$$

$$\frac{d[P]}{dt} = v_3 = k_3[ES] \tag{2.10}$$

$$\frac{d[E]}{dt} = -v_1 + v_2 + v_3 = -k_1[S][E] + (k_2 + k_3)[ES] \tag{2.11}$$

$$\frac{d[ES]}{dt} = v_1 - v_2 - v_3 = k_1[S][E] - (k_2 + k_3)[ES] \tag{2.12}$$

のように表すことができる。

しかし、実験で観測できるのは $[S]$ や $[P]$ とその変化であり、この式に現れる k_1、k_2、k_3 といった値を求めることは困難である。そこで、実験結果から得られる最大反応速度 V_m のようなパラメータを用いて、単純な $S \rightarrow P$ の反応として表現した式を用いるのが実用的である。

このような式を導出するにあたっては、定常状態付近での挙動を近似した簡略化がよく用いられる。式 2.5 において、定常状態とは $[S]$ や $[P]$ は変化しているが、$[E]$ と $[ES]$ は変化しない状態、すなわち

$$\frac{d[E]}{dt} = -\frac{d[ES]}{dt} = 0 \tag{2.13}$$

が成立している状態である。この場合、式 2.12 の右辺第 1 項と第 2 項は等しくなるので、

$$k_1[S][E] = (k_2 + k_3)[ES] \tag{2.14}$$

これを $[ES]$ についての式に変形して

$$[ES] = \frac{k_1[S][E]}{k_2 + k_3} \tag{2.15}$$

を得る。
ここで、Michaelis-Menten 定数 K_m を

$$K_m = \frac{k_2 + k_3}{k_1} \tag{2.16}$$

と定めると、式 2.15 は

$$[ES] = \frac{[S][E]}{K_m} \tag{2.17}$$

となる。ここで、酵素の総量を E_0 とすると

$$E_0 = [E] + [ES] \tag{2.18}$$

であるから、

$$[E] = E_0 - [ES] \tag{2.19}$$

これを式 2.17 に代入して

$$[ES] = \frac{[S](E_0 - [ES])}{K_m} \tag{2.20}$$

を得る。これを変形すると、

$$[ES]\frac{K_m}{[S]} = E_0 - [ES] \tag{2.21}$$

$$[ES]\left(1+\frac{K_m}{[S]}\right)=E_0 \tag{2.22}$$

$$[ES]=E_0\frac{[S]}{[S]+K_m} \tag{2.23}$$

となり、これを用いると P の生成速度を

$$\frac{d[P]}{dt}=k_3[ES]=k_3E_0\frac{[S]}{[S]+K_m} \tag{2.24}$$

のように書くことができる。E_0 は実験で測定することが困難であるが、k_3E_0 は基質が飽和したときの最大反応速度 V_m と考えることができ、これは実験で測定することができる。これを用いれば、

$$\frac{d[P]}{dt}=V_m\frac{[S]}{[S]+K_m} \tag{2.25}$$

のようにして P の生成速度を得られる。

K_m については、式 2.25 で $[S]=K_m$ とすると

$$\frac{d[P]}{dt}=V_m\frac{K_m}{2K_m}=\frac{V_m}{2} \tag{2.26}$$

と表すことができ、$[S]=K_m$ のとき $d[P]/dt$ が最大反応速度の半分になることがわかる。したがって、K_m は反応速度が $V_m/2$ となるような $[S]$ の値で、これは実験的に知ることができる。

以上のことから、実験で求めることのできる値である K_m、V_m と基質濃度 $[S]$ を用いて、式 2.5 に示す Michaelis-Menten 型の反応速度式を

$$v=\frac{V_m[S]}{K_m+[S]} \tag{2.27}$$

と、いう形で表すことができる。

Michaelis-Menten モデル以外にもさまざまな反応機構の反応速度を表すための式が存在し、反応経路中の反応ひとつひとつについて適切な反応機構・反応速度式とそのパラメータを与えることで、シミュレーションのための生化学モデルを構成することができる。

なお、反応速度式を微分方程式を用いてシステムの挙動を記述する代わりに、各反応の発生確率を用いた確率モデルを使う方式 [27][28][29] もあり、近年注目を集めている。

2.2.2 モデルの例: Minimal Mitotic Oscillator

図 2.2 に示した minimal mitotic oscillator モデル [23] は細胞周期中の cdc2 キナーゼの活性変化に関する実験結果から構築されたモデルである。このモデルでは cyclin は一定の速度で生成され、cyclin が cdc2 kinase を活性化し、cdc2 kinase が cyclin protease を活性化することによって cyclin が分解される。cyclin、cdc2 kinase、cyclin protease の構成するカスケードの終端から cyclin への負のフィードバックによって cyclin、cdc2 kinase、cyclin protease の濃度は図 2.3 のように振動を繰り返す。

図中の C (cyclin)、M (cdc2 kinase)、X (cyclin protease) の濃度をそれぞれ $[C]$、$[M]$、$[X]$ とすると、minimal mitotic oscillator モデルの反応速度式は、次の式 2.28〜2.34 で与えられる。

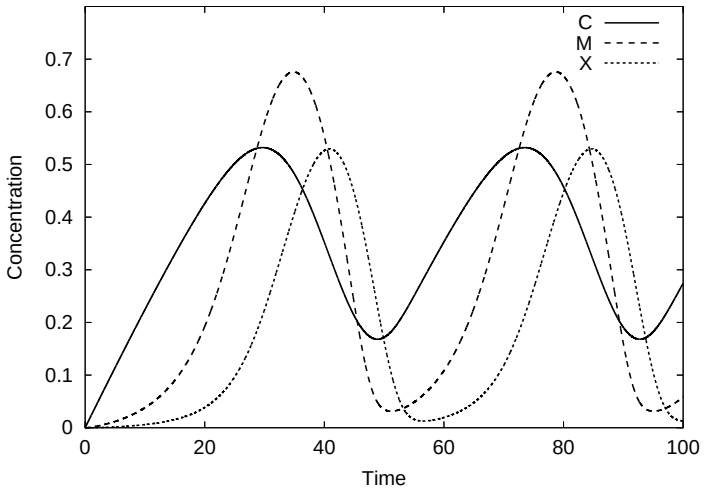


図 2.3　Minimal Mitotic Oscillator モデルの挙動

R_1: cyclin の生成

$$v_1 = V_i \quad (V_i = 0.023) \tag{2.28}$$

R_2: cyclin の分解

$$v_2 = K_d[C] \quad (K_d = 0.00333) \tag{2.29}$$

R_3: cyclin protease による cyclin の分解

$$v_3 = \frac{V_d[C][X]}{K_d + [C]} \quad (K_d = 0.00333,\ V_d = 0.1) \tag{2.30}$$

R_4: cdc2 kinase の活性化

$$v_4 = \frac{V_{m1}[C](1 - [M])}{(1 + K_1 - [M])(K_c + [C])} \quad (K_1 = 0.1,\ K_c = 0.3,\ V_{m1} = 0.5) \tag{2.31}$$

R_5: cyclin protease の活性化

$$v_5 = \frac{V_{m3}[M](1 - [X])}{K_3 + (1 - [X])} \quad (K_3 = 0.1,\ V_{m3} = 0.2) \tag{2.32}$$

R_6: cdc2 kinase の不活性化

$$v_6 = \frac{V_2[M]}{K_2 + [M]} \quad (K_2 = 0.1,\ V_2 = 0.167) \tag{2.33}$$

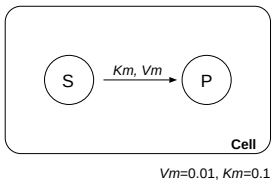
R_7: cyclin protease の不活性化

$$v_7 = \frac{V_4[X]}{K_4 + [X]} \quad (K_4 = 0.1,\ V_4 = 0.1) \tag{2.34}$$

なお、図 2.3 は、時刻 0 において $[C] = [M] = [X] = 0$ とした場合のプロットである。

```
<model name="sample_model">
  <listOfCompartments>
    <compartment name="cell"/>
  </listOfCompartments>
  <listOfSpecies>
    <specie name="S" initialAmount="0.3" compartment="cell"/>
    <specie name="P" initialAmount="0"   compartment="cell"/>
  </listOfSpecies>
  <listOfReactions>
    <reaction name="Reaction1" reversible="false">
      <listOfReactants>
        <specieReference specie="S" />
      </listOfReactants>
      <listOfProducts>
        <specieReference specie="P" />
      </listOfProducts>
      <kineticLaw formula="uui(S,km,vm)">
        <listOfParameters>
          <parameter name="vm" value="0.01" />
          <parameter name="km" value="0.1" />
        </listOfParameters>
      </kineticLaw>
    </reaction>
  </listOfReactions>
</model>
```

a) SBML 記述



Vm=0.01, *Km*=0.1

b) 反応経路

図 2.4　SBML によるモデル記述の例

2.2.3　モデル記述言語

前節で述べたようなモデルを計算機上で取り扱う際には、モデルを設計するためのツールやシミュレータ、分析ツールなどさまざまなソフトウェアを利用することになるが、作業を効率化するためにはこれらのツール間で使える共通のモデル記述言語が必須である。現在、モデル記述言語の標準として SBML (Systems Biology Markup Language)[30] や CellML[31] などの XML ベースの規格が定められ、多くのツールがこれらに対応している。XML を用いることで、互換性を維持しつつ言語の仕様を拡張したり、ツール独自の情報をファイル内に記述したりすることが容易になる。

図 2.4 に SBML によるモデル記述と、その表現する反応経路の図を示す。この図に示されるように、SBML によるモデル定義は、最も基本的な場合、

- モデルの名称 (<model>)
  - 区画のリスト (<listOfCompartments>)
    - 区画の名称 (<compartment>)
      これによって、細胞、細胞質、核などの区画を定義し、物質や反応をそれぞれの区画に配置することができる。区画間の包含関係も記述することができる。
  - 物質のリスト (<listOfSpecies>)
    - 物質の定義 (<specie>)
      物質の名前、初期濃度、配置される区画名などを記述する。
  - 反応のリスト (<listOfReactions>)
    - 反応の定義 (<reaction>)

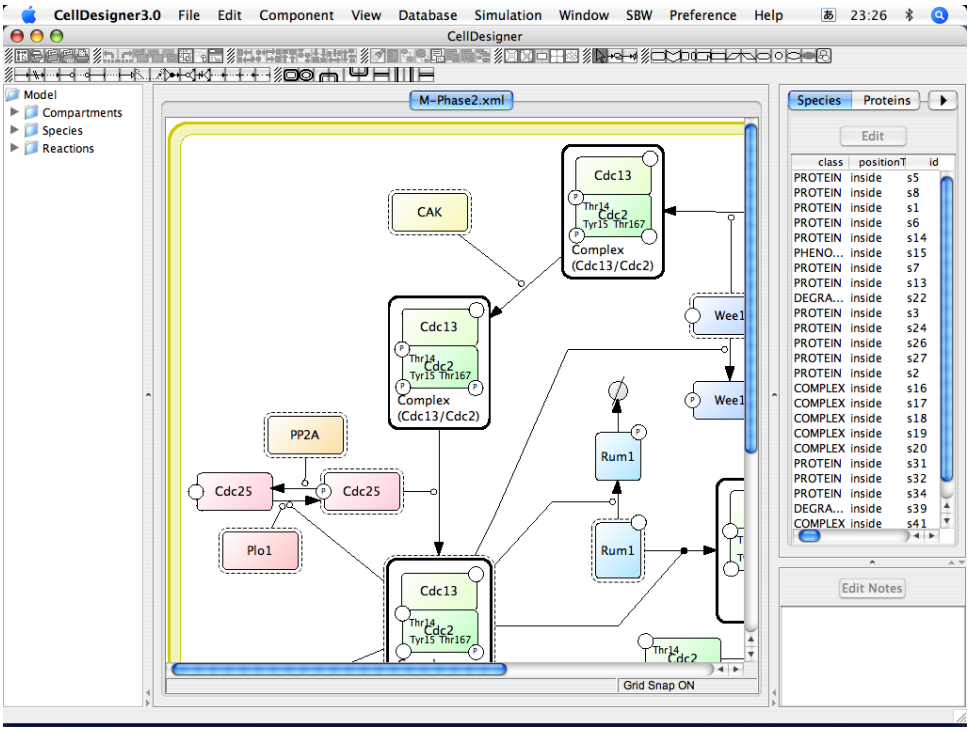


図 2.5　CellDesigner のスクリーンショット

- 反応物のリスト (<listOfReactants>)
  反応で消費される物質を定義する。
- 生成物のリスト (<listOfProducts>)
  反応で生成される物質を定義する。
- 反応速度式 (<KineticLaw>)
  反応速度式とそのパラメータの値 (<parameter>) を定義する。

などの項目を含んでおり、これを用いてシミュレーションを行うために必要な情報をさまざまなツールで共有することができる。ツール固有の情報は、各ブロック内に<annotation>ブロックを設け、そこにツール毎の namespace を用いて記述することができるようになっているほか、モデルを描画する際のレイアウト情報など、多くのツールが付加するようになった情報の記述方法を標準として策定していくなど、規格の更新も行われている。

例として、第 2.2.2 節で解説した minimal mitotic oscillator モデルの SBML 記述を付録 D (101 ページ) に示す。

2.2.4　設計ツール

モデルを記述したり読んだりする際に、XML を直接手作業で記述するのでは作業効率が悪く、大きなモデルを構築するのは困難であるため、CellDesigner[32][33]、JDesigner[34]、PathwayLab[35] などの GUI を用いたツールが多く開発されている。たとえば、図 2.5 は CellDesigner のスクリーンショットであり、ドローイングツールのように画面上に物質や反応を描いていくことで、視覚的に SBML のモデルを作成し、物質の初期濃度や反応速度式などを設定することができる。

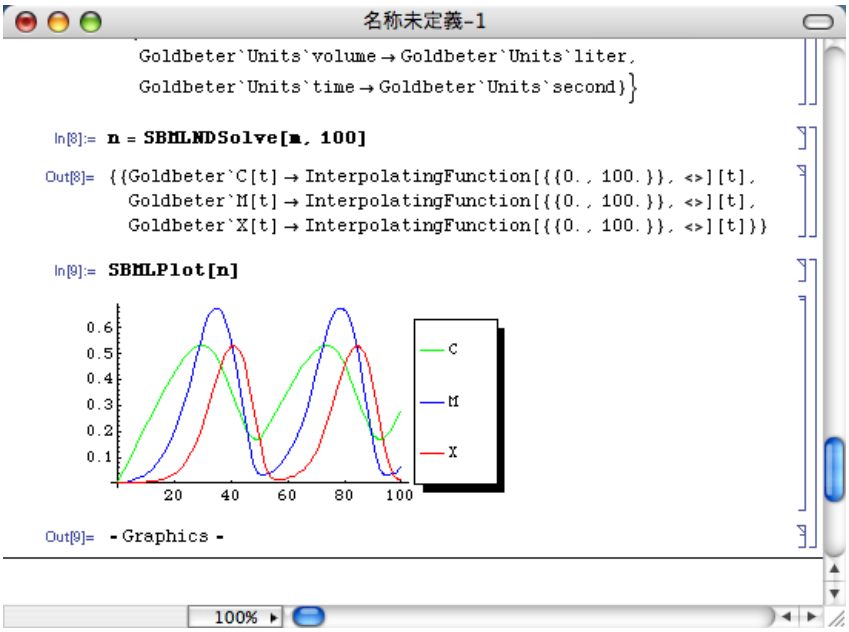


図 2.6　MathSBML のスクリーンショット

この種のツールの多くは、シミュレータや分析ツールなどと連携しており、モデルの作成から分析までを一貫して行えるようになっている。また、最近では大きな反応経路をモデル化する際に、モデルの妥当性を示すために反応系中の反応や物質について出典を記述したり、そこから文献データベースへ接続したりといった機能が活用されるようになってきており、設計ツールは情報を集約するツールとしての役割をも果たすようになりつつある。

2.2.5　シミュレータ

Jarnac[36]、Copasi[37]、MathSBML[38] などのシミュレータも数多く開発されている。これらのシミュレータではいずれも結果を視覚化したり、パラメータを最適化するための独自の特徴的なインタフェイスを持っており、さまざまなシミュレータをうまく使い分けることで目的とする結果を得ることができる。また、設計ツール、シミュレータ、分析ツールなどのソフトウェア間を接続するためのソフトウェア基盤として、SBW (Systems Biology Workbench)[39] が開発されている。

図 2.6 は MathSBML のスクリーンショットである。MathSBML は Wolfram Research 社の汎用科学技術計算ソフトウェアである Mathematica 上で動作するシミュレータで、SBML ファイルを読み込んで連立微分方程式に変換し、シミュレーションをする機能を提供する。

2.2.6　データベース

モデル記述方式の標準化や、関連するツールの整備の結果、反応経路をデータベース化して再利用しようという機運が高まり、KEGG[40][41] や BioModels.net[42] などのデータベースが構築されている。KEGG は独自のモデル記述方式をとっているが、SBML へのトランスレータ [43] も開発されており、SBML 対応の処理系で KEGG のモデルを利用することが可能である。また、SBML

や CellML などは記述の自由度が高いため、モデルをデータベースに登録するにあたって、再利用性を確保するためにモデルが最低限含むべき情報のガイドラインの策定 [44] も行われている。

KEGG や BioModels.net のような、一般的なデータベースのほかにも線虫に関するデータやモデルを集めた Wormbase[45] や、大腸菌に着目した EcoCyc[46] のように、特定のモデル生物に関する知識の集積をはかるプロジェクトも推進されている。

2.3 FPGA: Field-Programmable Gate Array

前節までで本研究の生命科学的な背景について述べた。本節と次節では、本研究で用いる FPGA の構成と、それを用いた計算機システムの実例について概説する。

FPGA は、商用プログラマブルデバイスとして CPLD (Complex Programmable Logic Device) と並んで多く出荷されているデバイスであり、LUT (Look-Up Table) を用いて細粒度のプログラマブルロジック回路を実現することで、大規模な回路を実装可能な特徴を持つ [47][48]。

2.3.1 FPGA の基本的な構成

FPGA は、LUT をロジック回路の基本要素とするプログラマブルデバイスで、多くの商用 FPGA は 4 入力 1 出力の LUT で構成されている。n 入力 m 出力の LUT は入力 n ビット、出力 m ビットの任意の論理関数を実現するためのテーブルであり、アドレス幅 n ビット、ワード幅 m ビットのメモリであると考えることができる。FPGA ではこのメモリを、SRAM や Flash ROM、Anti-Fuse ROM などで構成し、値を書き込むことで論理関数を表現する。本研究で用いた Xilinx 社の Virtex-II / Virtex-II Pro シリーズは、LUT を SRAM によって構成した SRAM 型 FPGA である。SRAM 型 FPGA は一般的な CMOS 半導体プロセスで製造されるため、最新のプロセスを利用することができ、プロセスの微細化による回路容量の増大の恩恵を最大限に受けることができる。

図 2.7 は、現在の主要な商用 FPGA で採用されている Island-style と呼ばれるアーキテクチャである。このアーキテクチャでは、

- Logic block
- Connection block
- Switch block

の 3 つのブロックで FPGA を構成する。図 2.7(a) は FPGA を構成する基本的なひとつのブロックであり、これを多数並べることで図 2.7(b) のように大きな回路を構成する。Island-style FPGA では構造が均一であるため、ブロック数を変化させることでロジックサイズを変えた製品ラインナップを容易に構成できる。

Logic block は LUT やフリップフロップを含む、プログラマブルロジック自体を実現するためのブロックであり、隣接する connection block への配線 (図中では 1 本線で表現) を持つ。Connection block は、隣接する logic block への配線と、FPGA 中のグローバルな配線 (図中では 3 本線で表現) との間を接続するパストランジスタを用いたプログラマブルスイッチである。縦横のグローバル配線の交点には、やはりプログラマブルスイッチである switch block が設置されており、これが縦横の配線の間の接続を実現している。このように、FPGA はロジックと配線の双方でプログラマビリティを持つことで、柔軟に論理回路を構成することができる。

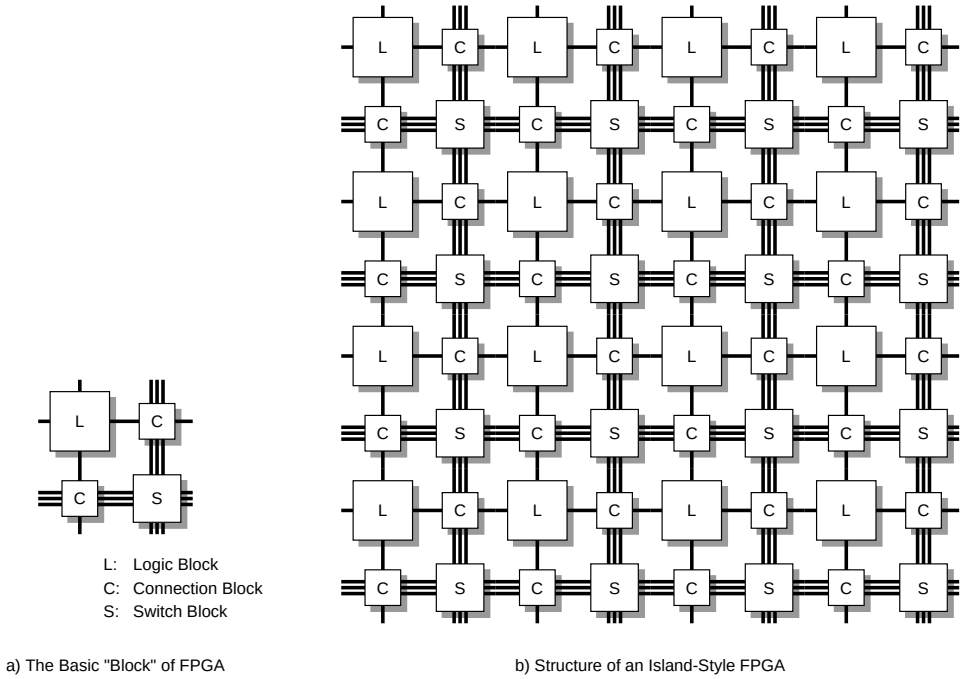


図 2.7 Island-style FPGA の基本ブロックと全体の構造

2.3.2 FPGA アーキテクチャの例: Xilinx 社 Virtex-II シリーズ

ここでは、FPGA のアーキテクチャの一例として、Xilinx 社の Virtex-II アーキテクチャについて述べる。このアーキテクチャは同社の製品ある Virtex-II シリーズ [5] ならびに Virtex-II Pro シリーズ [6] に共通のものである。

CLB の構成

前節で述べた logic block は、LUT だけで構成されるものではなく、実際には図 2.8 に示されるような複雑な構造になっている。図 2.8 (a) は、前節の logic block に相当する CLB (Configurable Logic Block) の構成であり、ひとつの CLB には 4 つの slice と呼ばれるブロックが含まれる。スライスは図 2.8 (b) に表されるような構造になっており、4 入力 1 スライスの LUT とレジスタを各 2 つずつ備えており、LUT を使って組み合わせ回路、レジスタを使って順序回路を構成することができる。LUT とレジスタの間にはマルチプレクサが設置されており、LUT の出力をレジスタを経由せずにそのまま出力したり、スライス内のマルチプレクサを用いて同一スライス内および隣接するスライス内の LUT をカスケード接続し、4 入力以上の論理関数を構成することもできる。

また、図 2.8 (b) では省略されているが、図 2.8 (a) に示すように、各スライスにはいくつかの種類の回路を効率よく構成するための専用配線が用意されている。これには、LUT をシフトレジスタとして用いる際に、複数の LUT やスライスにまたがって長いシフトレジスタを効率よく構成するためのシフト用チェーンである IN/OUT や、全加算器を構成する際にキャリーを効率よく接続するための CIN/COUT などがあり、回路面積の削減や動作周波数の向上などに貢献している。

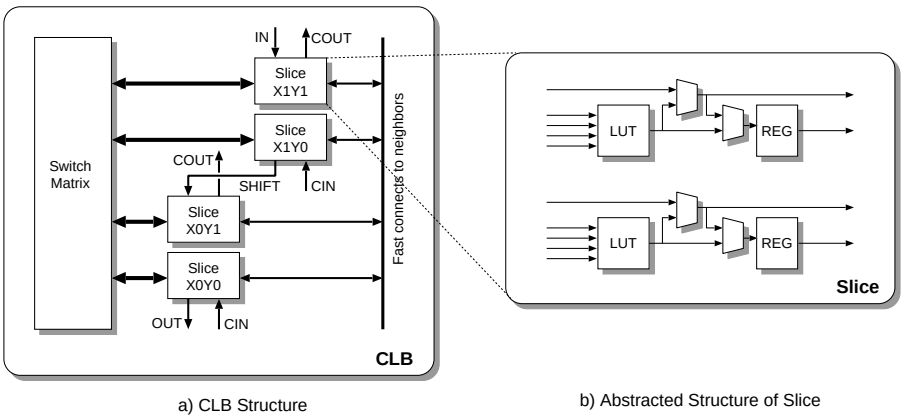


図 2.8 Virtex-II FPGA の CLB の構成

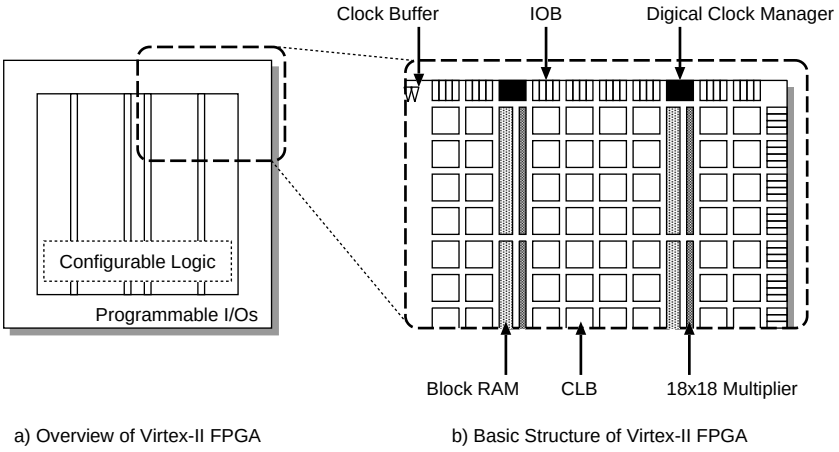


図 2.9 Virtex-II FPGA の構成

Virtex-II FPGA 全体の構成

次に、Virtex-II FPGA 全体の構成を図 2.9 に示す。Virtex-II は、チップ周辺にプログラマブル I/O ブロック (IOB) が配置されている。IOB は CMOS インターフェイス以外にも LVTTL、HSTL、SSTL、LVDS、PCI、AGP など各種の信号規格に対応できるプログラマブルな構成で、DDR 入出力用のレジスタも備えている。

チップ上のロジック部は CLB が碁盤の目状に配置され、その間に接続用の配線が配置される。CLB の間には、縦方向にメモリブロック (BlockRAM) および組み込み乗算器 (18x18 bit) のストライプが数カ所配置される。メモリや乗算器などは、頻繁に使われるが多くの LUT を必要とするため、これらを専用のハードマクロとして組み込むことで回路面積を抑え、また回路の動作周波数を向上させることができる。

Virtex-II Pro はほぼ同様の構成であるが、BlockRAM や組み込み乗算器と同様の発想で、組み込みプロセッサとして PowerPC 405 と、マルチギガビット・トランシーバをハードマクロとして組み込んでいる。これにより、従来外付け部品によって実現されてきた制御用のマイクロプロセッサや高速作動シリアル通信のインタフェイスの 1 チップ化を実現し、ネットワーク機器等の強力

表 2.1　各世代の FPGA のプロセス・LUT 数と電源電圧 (Xilinx 社)

プロセス	シリーズ	型番	LUT 数	電源電圧
350nm	XC4000	XC4085XLA	7,448	3.3V
250nm	XC4000	XC40250XV	20,102	2.5V
220nm	Virtex	XCV1000	27,648	2.5V
180nm	Virtex-E	XCV2000E	43,200	1.8V
150nm	Virtex-II	XC2V8000	104,882	1.5V
130nm	Virtex-IIPro	XC2VP125	125,136	1.5V
90nm	Virtex-4	XC4VLX200	200,448	1.2V

な入出力能力と高速な処理の両方が求められるアプリケーションで力を発揮している。

2.4 FPGA を用いた高性能計算システム

2.4.1 FPGA を用いた計算処理の利点

従来、科学技術計算をハードウェアの力により高速化するには、GRAPE[49] に代表されるような専用計算機を開発する必要があった。専用計算機による計算処理のメリットとしては、

- 必要な演算器をひとつのチップ上に構成し、順番に接続することで、深いパイプラインを構成し、同時に多数の演算器を動かすことにより高いスループットを得ることができる。
- 深いパイプラインを構成した場合にも、マイクロプロセッサのようなパイプラインのストールが発生しないため、高い実効性能が期待できる。
- チップやボード上のレジスタやメモリを柔軟に利用できるため、小さなメモリブロックを多数並列にアクセスすることにより、大きなバンド幅を得ることができる。

といった点が挙げられる。専用計算機の場合には、これらの利点を活かして問題をそのままハードウェア化して計算を行うため、非常に高い性能が期待できる。その一方で、他の問題を解くために利用できないことや、高い開発コストが必要とされることなどが問題である。

これに対して FPGA は、専用ハードウェアのように問題を直接ハードウェアで解くことができるという利点をもちつつ、回路をソフトウェア的に変更できるため、高い性能対コスト比と柔軟性の両立が期待できる (図 2.10)[50]。さらに、FPGA の回路容量は年々拡大しており、表 2.1 に示すように、1990 年代初めにリリースされた 350nm プロセスの FPGA と、2005 年に出荷開始された 90nm プロセスの FPGA とでは、その LUT 数に 25 倍以上の差がある。大容量の FPGA によって浮動小数点演算を含むアプリケーションの実装が可能になり、近年では FPGA をコプロセッサとして搭載した商用計算機も出現している。本章の以下の部分では、FPGA を用いた主要な商用計算機の例と、FPGA の生命科学への応用例について述べる。

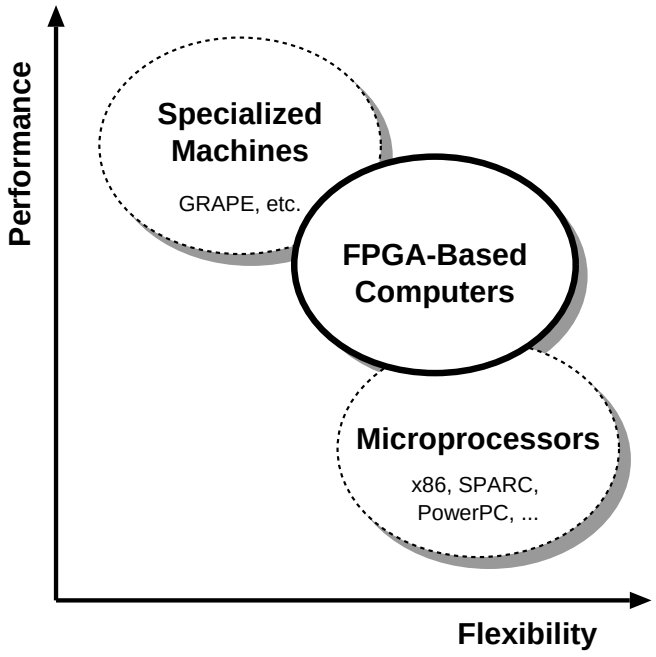


図 2.10　性能と柔軟性のトレードオフ

2.4.2　商用計算機への採用例

RASH (三菱電機)

RASH は三菱電機によって開発されたシステムで、1 枚の CompactPCI ボードに Altera 社の FLEX10K シリーズの FLEX10K100A を 8 チップと 2MB の SRAM を搭載している。CompactPCI のバックプレーンを持つ筐体にこのボードを最大 6 枚格納して 1 ユニットとし、さらにユニット間を Ethernet で接続することで大型のシステムを構成することができる。

RASH のアプリケーションとしては DES の鍵探索 [51] や合成開口レーダーの画像生成 [52] などが実装された。DES では 48 個の FPGA を用いて 300MHz で動作する Alpha プロセッサの 13.8 倍の性能を、レーダーの画像生成では 24 個の FPGA を用いて同規模の DSP ボードの 2 倍の性能をそれぞれ達成している。

RASC (SGI Inc.)

RASC[53] は SGI 社のスーパーコンピュータである Altix シリーズ [54] に組み込むことのできる、FPGA を用いたアクセラレータである。Altix シリーズは Intel 社の 64bit VLIW プロセッサである Itanium2 を要素プロセッサとして用い、2 プロセッサを 1 ノードとして NUMAlink4 と呼ばれる専用の結合網でノード間を結合する構成で、NUMAlink4 は 6.4GB/sec. の転送バンド幅と、スイッチング時間 50nsec. という低レイテンシを実現している。RASC は、2 本の NUMAlink4 を用いて 12.8GB/sec. でシステムに接続するアクセラレータで、PCI-X バスを用いるよりも転送バンド幅や遅延の面で有利な構成としている。

RASC には 2 つの FPGA が搭載されており、演算に用いるのは Xilinx 社の Virtex-II シリーズの XC2V6000 であり、もう一方の FPGA はこの XC2V6000 をコンフィギュレーションするために用

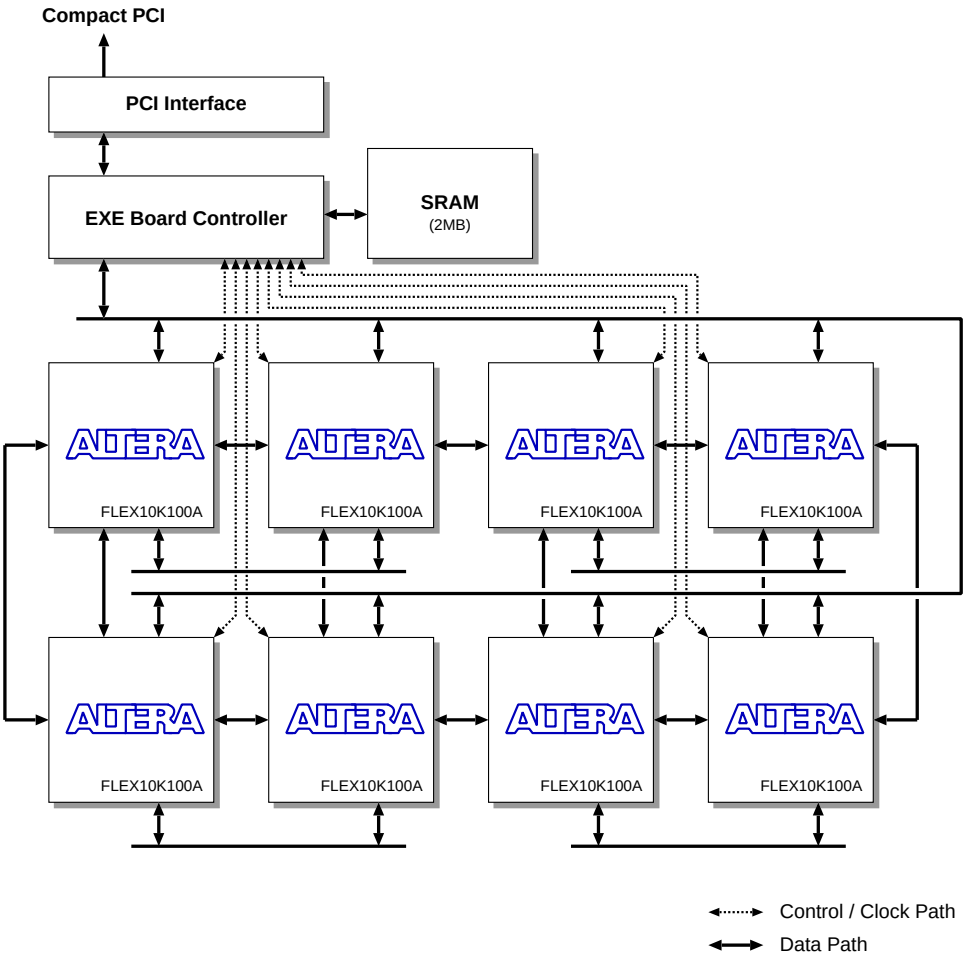


図 2.11 RASH の構成

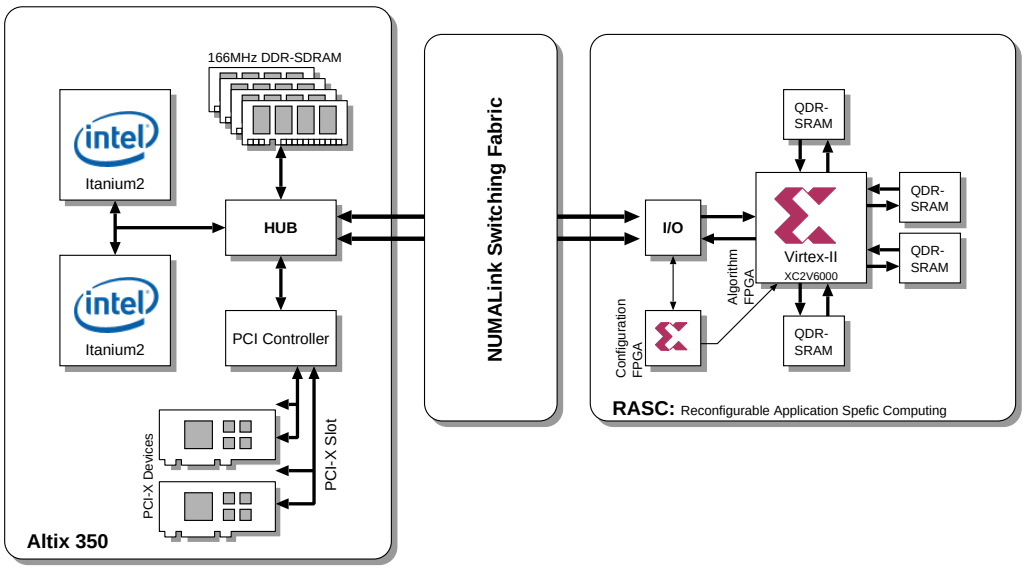


図 2.12 Altix350 / RASC の構成

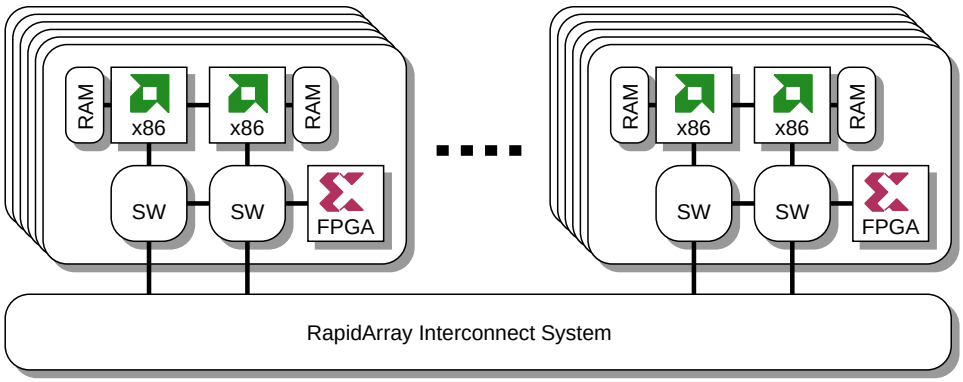


図 2.13　Cray XD-1 の構成

いられる。RASC を用いた文字列のパターンマッチングにおいて、Itanium2 の 100 倍以上の性能向上が報告されている。

Cray XD-1 (Cray Inc.)

XD-1[55] は、AMD Opteron プロセッサをベースにした並列計算機であり、2 プロセッサ構成の基板を 1 つの筐体あたり 6 枚格納したものを 12 台までひとつのキャビネットに納めることができ、さらにそれを相互に接続してシステム規模を拡大することができる (図 2.13)。最大で 1 筐体あたりのプロセッサ数は 12、メモリ容量は 96GB である。

この、マイクロプロセッサを搭載した基板には Xilinx 社の FPGA がひとつ搭載されており、これをコプロセッサとして利用することでさらに高い性能を得ることもできる。この FPGA は PCI バス経由ではなく、プロセッサ結合網に直結されているため、プロセッサ–FPGA 間で高速にデータを転送することができる。この FPGA を用いたソリューションとして、Cray 社から暗号化、FFT、乱数生成、符号化・復号などの処理や、光学、量子力学、塩基配列の検索などのアプリケーションを高速に実行するためのソリューションが提供されている。また、ユーザがこの FPGA に独自のアプリケーションアクセラレータを実装 [56] することも可能である。

SRC-7 (SRC Computers Inc.)

SRC-7[57] は、他の FPGA ベースのシステムと比べるとより汎用的に FPGA を利用するコンセプトで開発されている。SRC-7 には専用のコンパイラが用意されており、FPGA 用の浮動小数点演算ライブラリなどを利用して専用のプログラミング環境でプロセッサ–FPGA 間の負荷分散を実現できるように配慮されている。開発用の言語としては C のほかに Fortran などもサポートされており、広い分野の科学技術計算ユーザをターゲットとしている。

SRC-7 は RASC と同様にマイクロプロセッサを搭載したボードと FPGA を搭載したボードが分離されており、共通のスイッチに接続する設計になっているため、それぞれの数を柔軟に変更することができる (図 2.14)。また、マイクロプロセッサベースのシステムとスイッチを接続するインタフェイスとして、DIMM スロット装着型のインタフェイス (SNAP Interface) を用いることで 6.4GB/sec. と 133MHz PCI-X バス (1GB/sec.) よりも広い通信バンド幅を実現するとともに、汎用のチップセットの利用による低価格化を可能にしている。

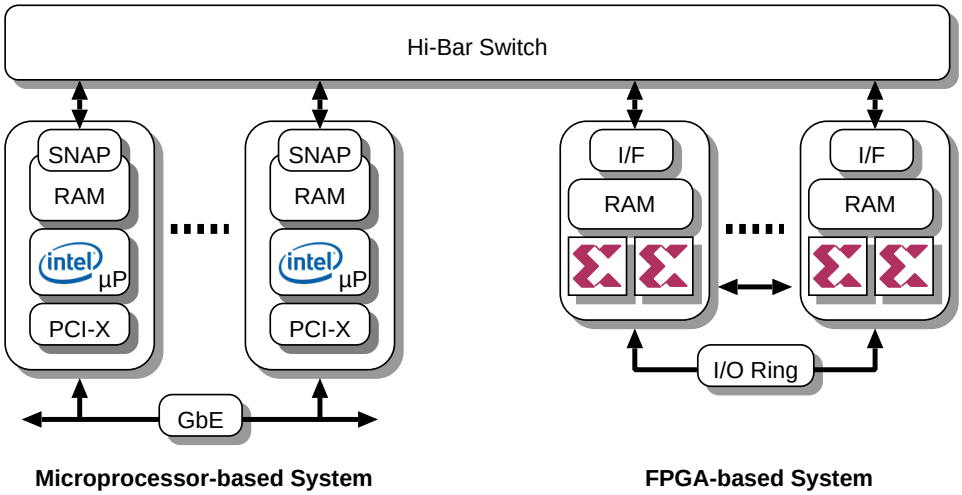


図 2.14　SRC-7 の構成

PROGRAPE-3 (Riken / Tokyo Electron Device)

PROGRAPE-3 (Bioler-3) は、(株) 東京エレクトロンデバイス、千葉大学、理化学研究所などによって開発されたボードであり、Xilinx 社の Virtex-II Pro FPGA (XC2VP70-6) を 4 つ搭載した PCI カードである。PCI カードであるため、通常の PC に組み込んでアクセラレータとして利用することが可能であり、天体間の重力シミュレーション (多体問題) で 236GFLOPS を達成している [10][58]。

2.4.3　生命科学分野での応用例

FPGA が汎用の計算システムに採用されるようになる以前から、生命科学分野向けのアクセラレータとしての FPGA の利用が数多く試みられてきた。ここでは、生命科学における、FPGA の従来からの主要なアプリケーション領域やその実装例を概説する。

配列の相同性検索

新しく解読された遺伝子やアミノ酸の配列と、機能が同定されているものの配列の相同性を検索することで、その機能を推定することができる。したがって、各種生物のゲノムが解読されている現在、膨大なデータベースを対象に配列の類似度や相違点を検索することが生命システムを構成する部品の機能を解明する上で重要な役割を果たしている。

これらの配列検索には、Smith-Waterman アルゴリズム [59] や Pairwise Hidden Markov Model (PHMM)[60] などが用いられるが、いずれも細かなデータの比較の繰り返しを含むため、FPGA を利用した実装が盛んに開発されている。古くは 1990 年代はじめの Splash 2[4] が、CRAY-2 の 330 倍の性能をマークしたことで有名である。このマシンは図 2.16 に示すような構成になっており、ひとつのボードにクロスバで結合した制御用に 1 個 (X0)、演算用に 16 個 (X1〜X16)、合計 17 個の Xilinx XC4010 FPGA を搭載している。複数のボードを使ってリニアシストリックアレイを構成して処理を行ったり、あるいはクロスバから命令を分配することで SIMD 構成の処理を行うこ

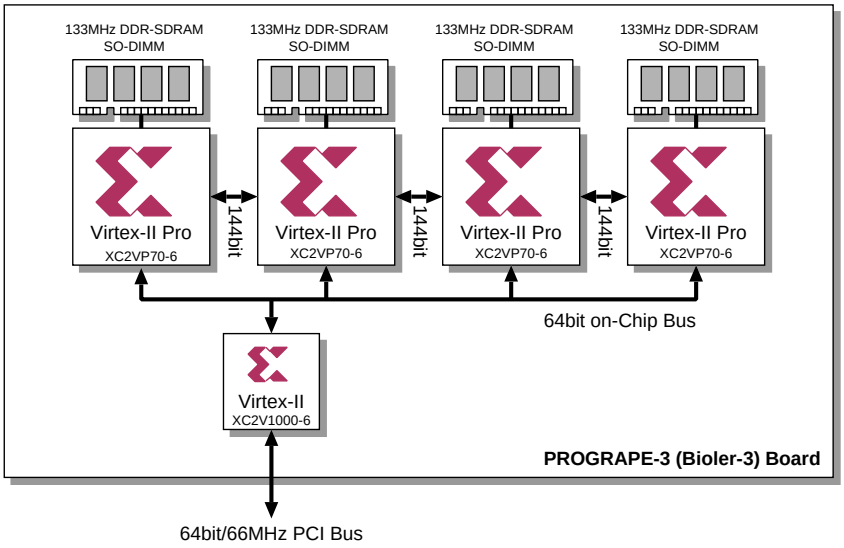


図 2.15　PROGRAPE-3 (Bioler-3) の構成

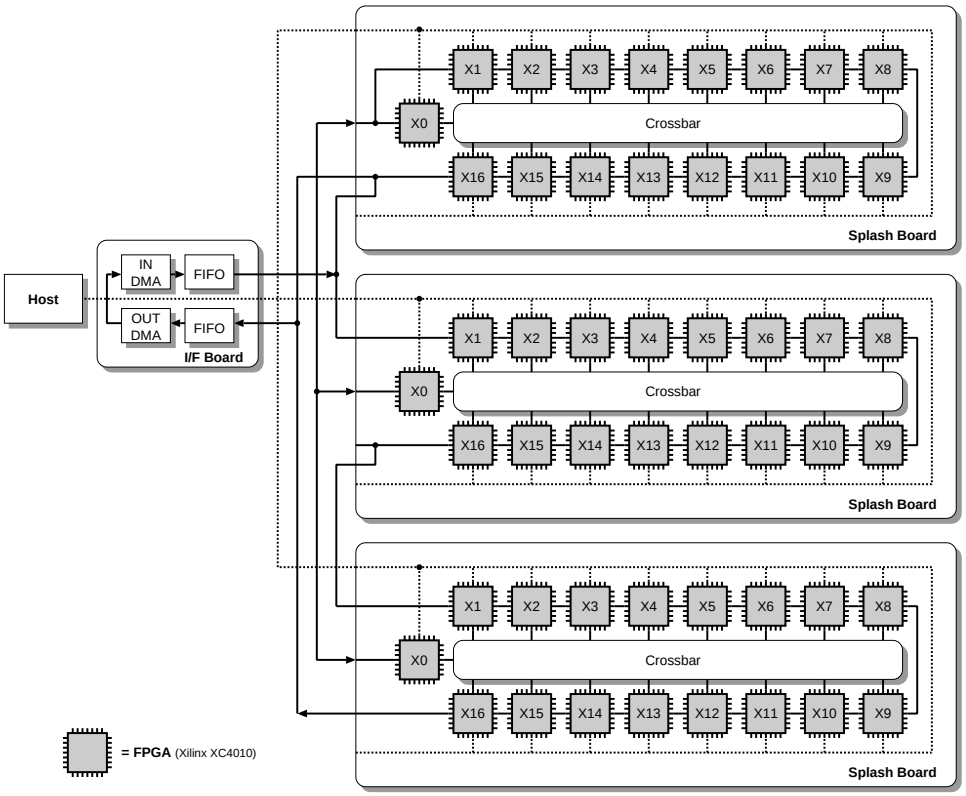


図 2.16　Splash 2 の構成

とができる。Splash 2 はのちに WILDFIRE という名称で Annapolis Micro Systems 社から販売 [61] された。

近年では 1 つの FPGA でマイクロプロセッサに比べて 200 倍程度の性能向上を実現しているものが多いが [62][63]、ゲノムプロジェクトなどで行われる大規模な検索のための多数の FPGA を用いた大型システムの開発も行われている。たとえば、カリフォルニア大学で開発されている BEE2[64] は、5 つの Virtex-II Pro FPGA をひとつのモジュールに搭載した FPGA クラスタシステムであり、1 筐体あたり最大 40 のモジュールを収納し、合計 200 個の FPGA による並列処理を行うことができる。

また、古くから用いられている 2 本の配列を対象とするアルゴリズムだけでなく、3 本以上の配列のアライメントを調べる新しいアルゴリズムなどの実装 [65] なども行われている。

分子動力学シミュレーション

タンパク質の複雑な立体構造とその機能との関係は生命現象においてきわめて重要であるが、その解明にはタンパク質とその他の分子の間の相互作用を計算機上でシミュレーションすることが重要である。分子動力学シミュレーションは生物学以外の分野にも広く適用できるため古くから研究されており、PC クラスタなどの並列システムや MD-GRAPE などの専用計算機を用いた手法 [66] と合わせて、FPGA を用いたアプローチ [67] も多く研究・開発され、Pentium4 をはじめとする最近のマイクロプロセッサを搭載したシステムの 50〜100 倍程度高速な計算を実現している。

画像処理アプリケーション

DNA マイクロアレイや顕微鏡の画像からの知識抽出もシステム生物学において重要な役割を果たしている。特に、顕微鏡画像からの知識抽出は画像フィルタリングの過程を含むため、長い計算時間を要する場合がある。顕微鏡で連続的に画像を取得する場合、オンザフライで処理が終了しないと未処理の画像を蓄積するために膨大なストレージを必要とする可能性があり、このような場合には処理の速度が、システム内のストレージの規模とコストを大きく左右する。

本研究で開発されたシミュレータの動作確認用プラットフォームとして開発された FPGA ボードである ReCSiP-2 Board 上でも、線虫 *C.elegans* の初期胚の細胞系譜を顕微鏡画像から自動構築するシステム [68] のアクセラレータを開発し、ソフトウェアによる処理では画像処理用に 32 台の PC クラスタを利用していたものを、処理速度を落とさずに FPGA ボードを組み込んだ PC 1 台で置き換えることに成功した [69]。

このように、Bioinformatics の分野でも、FPGA を用いて膨大なデータを高速に処理しようという試みは多数行われてきた。しかし、本研究がスタートした 2002 年の時点では、代謝回路などの生化学シミュレーションの高速化に関する取り組みはまだ発表されておらず、本研究はその先駆けであるといえよう。

第3章　ReCSiPの構成と実装

本章ではまず、本研究で開発されたFPGAによる生化学シミュレータReCSiPの設計目標とシミュレーション対象モデルについて述べる。続いてシミュレーションの基本方式とその問題点について触れ、最後にそれに基づいて決定されたシステムの構成について述べる。

3.1　設計目標

未知のパラメータを含む、連立常微分方程式によって記述されたモデルの性質を調べるにはまず、実験データと整合性のある挙動を示すパラメータセットの発見が不可欠である。特に、生きた細胞を測定の対象にする場合には細胞の状態に影響を与えずに測定を行うことが困難であり、実験から得られる限られた数値データからパラメータ推定を行い、実験とシミュレーションを繰り返しながらモデルの精度を高めていくことが重要である。

現在、このパラメータ推定のプロセスにはPCクラスタなどの並列システムで動作する、ソフトウェアベースの生化学シミュレータが用いられている。しかし、これらの並列システムはその価格や機器のサイズ、消費電力などが問題となって各研究者個人の計算システムとして導入することが困難である。

本研究では、FPGA上にシミュレータの回路を構成し、反応経路や反応速度定数、各分子の濃度などの値をFPGA上のメモリに格納して処理を行う方式について検討し、実装と評価を行った。基本的なシステムの構成を示した図が図3.1であり、処理を高速化するためのFPGAボードをPCIバスに接続し、計算時間を要する処理をPCのマイクロプロセッサからFPGA上の高速化用の計算エンジンにオフロードする。これによって、通常のデスクトップ型のPCを利用しながら、シミュレーション時にはFPGAを用いた高速な処理を実現する。

アクセラレーション用の計算エンジンとしてはFPGAの他に、DSPの利用や専用LSIの開発も考えられる。しかし、生化学モデルは一連の反応速度式で構成される連立常微分方程式として記述されており、これを解くには多くの浮動小数点演算を行う必要があるため、DSPでは充分に性能を出すことが困難である。また、モデルを構成する反応速度式はモデル毎に異なるため、専用LSIではモデルに対する柔軟性に問題があると考えられる。この2点の問題を解決するために、FPGAを用いてモデルに最適な回路を生成し、計算処理を行う方法を選択した。

FPGAを用いたシミュレータの構成を検討するにあたり、次の3点を目標として掲げた。

- スケーラビリティ
  対象のモデルの規模が拡大しても、回路面積が反応数や分子数に比例することなく、FPGAの限られたチップ面積内でシミュレーションを実行可能であること。現在モデル化されているシステムの中でも大きいものに分類されるもの[70][71]はいずれも数百の分子種と反応を含む程度であり、当面は数千の分子種と反応をサポートすることができれば充分であると考えられる。

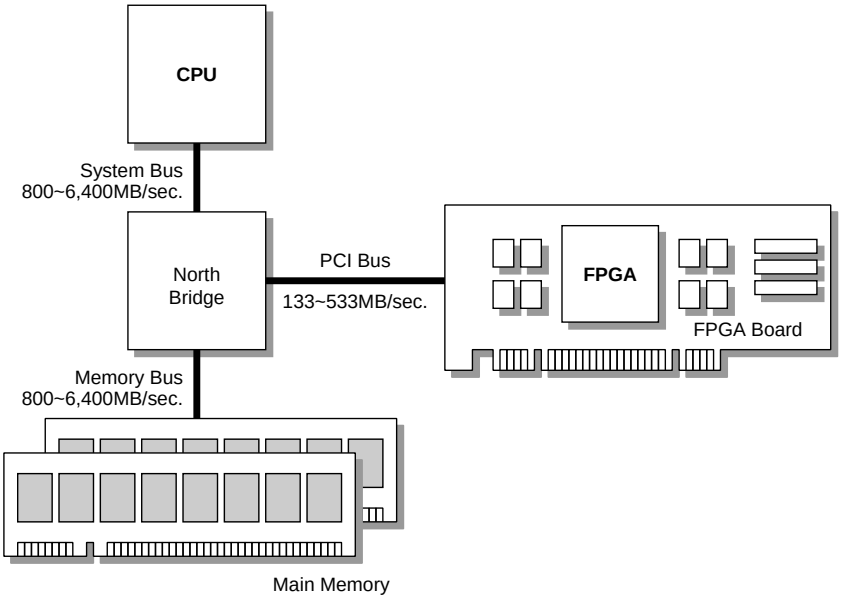


図 3.1　アクセラレータとしての FPGA の利用

- 柔軟性
  さまざまな種類の反応速度式を含む系を取り扱うことができること。FPGA を用いるメリットは、対象モデルに最適なハードウェア構成をとれることにあるが、回路を構成するには CAD ツールを用いて論理合成・配置配線を行わねばならない。これにかかる時間は無視できないため、性能を損なうことなく、一種類の回路でなるべく多くのモデルを取り扱うことができる構成にする。

- 利便性
  FPGA を利用したシミュレータには高速性が望まれるが、それだけではなく他の多くのソフトウェアベースのシミュレータと同様の操作で簡単に利用できることも要求される。このため、モデルの入力やシミュレータの起動にあたっては、標準化されたインタフェイスが容易に利用できるように、ユーザや他のツールとのインタフェイスとなるソフトウェアが開発しやすいようなハードウェアの構成にする必要がある。

性能面では、FPGA を 1 チップ用いた場合の処理能力で、小・中規模の PC クラスタの代替を目標とした。具体的には近年のハイエンド向けプロセッサの主流となっている、Intel の 3GHz 前後のマイクロプロセッサの 10 倍〜100 倍程度である。

3.2　シミュレーション対象モデル

対象とするモデルの定義とシミュレーションの内容であるが、本研究では FPGA による生化学シミュレーションの高速性を確認することが目的であるため、次に挙げるような基本的なものと定めた。

1. シミュレーション開始時に初期濃度と、反応経路中の反応速度式およびそれらの係数を与える。

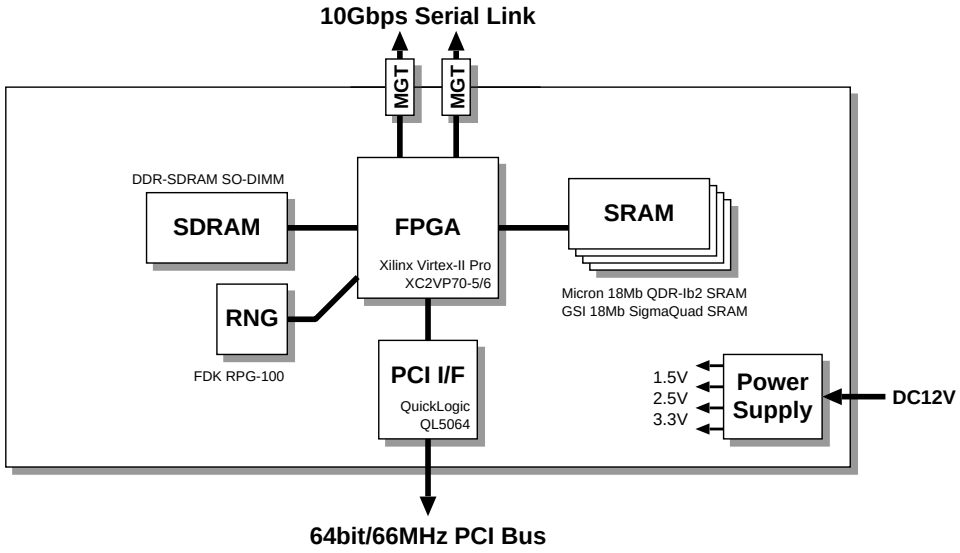


図 3.2　ReCSiP-2 Board の構成

2. これを初期値問題として、一定の時間刻み幅で積分を行う。

したがって、特定の分子種の濃度によって、反応経路がスイッチするような機構、濃度の範囲に制約を設けるような機構などの制御機構は特に設けない。ただし、反応速度式の一部として、あるいは個別の反応としてこれらの制御機構を実装することも可能であるので、これらは本質的なシステムの制約とはならない。

3.3　基本方式

3.3.1　動作プラットフォーム

第 3.1 節で述べたように、本研究では PC の PCI バススロットに FPGA を搭載したボードを挿入して用いる。この FPGA ボードには本研究のリファレンス環境として開発した ReCSiP-2 Board を用いた。

このボードは図 3.2 に示すような構成になっており、計算処理の中核を担うのは Xilinx 社の Virtex-II Pro シリーズの FPGA である XC2VP70 である。これを PCI バスに接続するために 64bit/66MHz の PCI バスに対応した PCI ブリッジである QuickLogic 社 QL5064 を搭載しており、XC2VP70 と QL5064 を接続するバスをローカルバスと呼ぶ。ローカルバスと PCI バスのクロックは分離されており、XC2VP70 および QL5064 のローカルバスインタフェイス部は任意のクロック周波数で動作することができる。

ReCSiP Board には現在 3 種類が存在しており、これらの詳細については付録 A で述べる。なお、ReCSiP-2 Board の XC2VP70 周辺に配置されている QDR-SRAM および DDR-SDRAM は、本研究では使用していない。

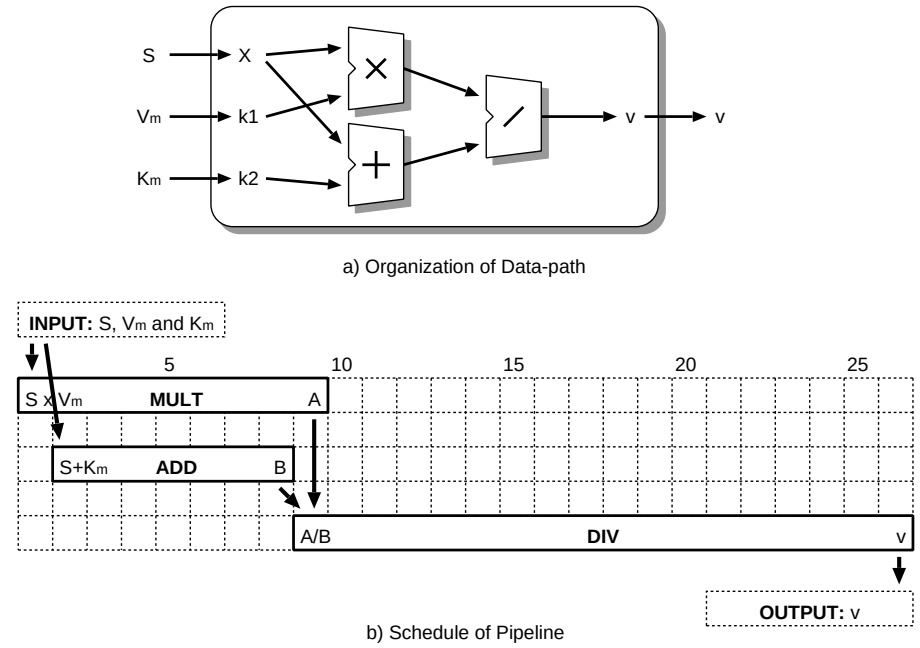


図 3.3　関数のパイプライン処理の例: Irreversible Michaelis-Menten 型の Solver Core

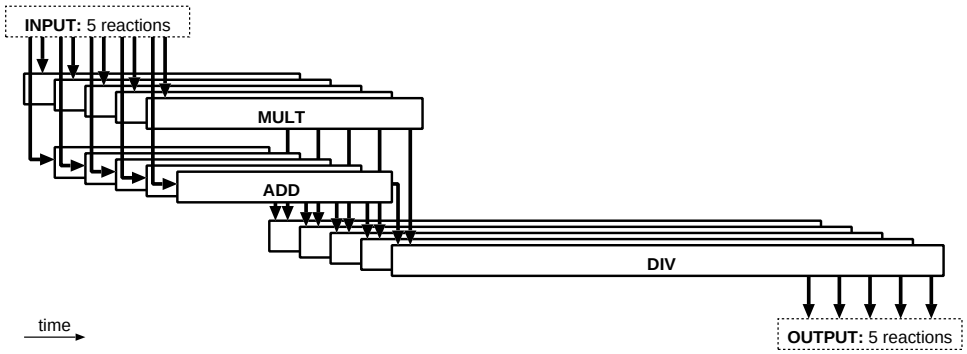


図 3.4　パイプラインを用いた連続処理の例

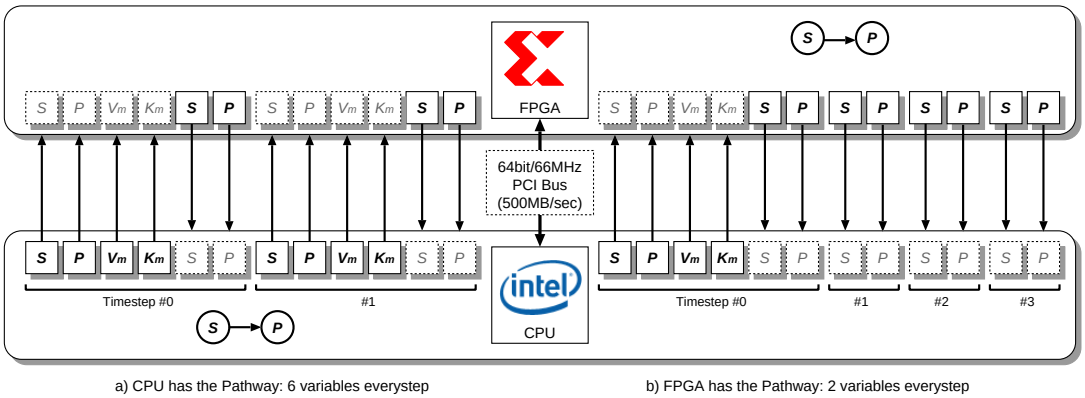


図 3.5　FPGA 上の反応経路処理機構によるデータ転送量の削減

3.3.2　静的パイプライン構成

FPGA などを用いて、対象とする問題専用のハードウェアを構成して計算を行う際にはパイプライン化された演算器の構成が性能に大きく貢献する。

たとえば、図 3.3 は、

$$v = \frac{V_m S}{K_m + S} \tag{3.1}$$

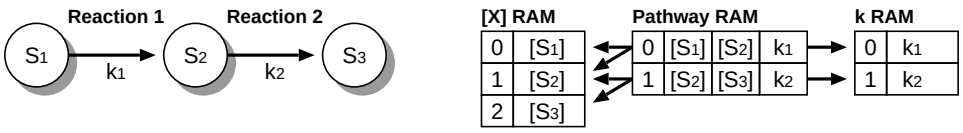
という反応速度式 (irreversible Michaelis-Menten reaction) を解く際の、データパスの構成 (図 3.3 の (a)) と、そのパイプラインスケジュール (図 3.3 の (b)) を示す。このパイプラインは加減算器、乗算器、除算器を各 1 つ持ち、計算の開始から終了までに 26 クロックサイクルを要するが、パイプラインを構成するすべての演算器が完全にパイプライン化されているため、すでにパイプライン中で進行中の処理の完了を待たずに、図 3.4 のように 1 クロックサイクル毎にデータを投入し、連続して結果を得ることができる。

パイプライン構成を細分化して、段数を増やすことは一般に動作周波数、即ち処理能力の向上に貢献する。マイクロプロセッサでは条件分岐命令の影響などでパイプラインの動作が乱れ、パイプラインの細分化による周波数による動作周波数の向上よりもパイプラインの乱れによる処理効率低下のほうが問題になることがあるが、アプリケーションに最適化されたハードウェアでは原則としてパイプラインの動作に乱れは発生しない。したがって、できるだけパイプライン長を長くすることで、演算スループットを向上するような設計方針とした。

3.3.3　反応経路処理機構

PCI バスに接続されたボード上の FPGA を用いてシミュレーションを行う際に、まず考えられるのは、反応速度式を計算するモジュールを FPGA 上に構成し、反応一つ一つに関与する物質の濃度や定数をホスト側 CPU から FPGA に転送し、計算結果を受け取るという方式である。

ここで、変数を 32bit の単精度浮動小数点型とし、式 3.1 の反応速度式を解くことを考えてみる。この反応速度式の計算に必要なのが S、P、V_m、K_m の 4 変数、計算の結果として発生するのが S、P の 2 変数の、合計 6 変数、24bytes である。これを、図 3.5 (a) のように毎回ホスト CPU と FPGA との間で転送した場合、64bit/66MHz の PCI バスの転送能力は約 500MB/sec. であるので、バスの



a) an Example of Reaction Pathway　　b) Corresponding Pathway RAM

図 3.6　PathwayRAM の概念図

理論的な最大転送能力をすべて使えたとしても、1 秒あたりの転送能力は最大で 22.16×10^6 反応分となる。

一方、ソフトウェアで計算した場合(注 1)には、1 秒間に 6.43×10^6 反応を計算することができ、これでは FPGA の計算能力が充分に速くても、PCI バスがボトルネックとなり、4 倍以上の性能向上は不可能であることがわかる。

このボトルネックを解決するには図 3.5 (b) のように、シミュレーション開始時に FPGA 上にデータを一括して転送し、シミュレーションの結果のみを FPGA からホストに転送する方式が考えられる。先ほどの例でいえば、毎回 PCI バスを経由して転送されるのは計算結果である反応後の濃度 [S]、[P] の 2 変数だけであり、データ転送量を 3 分の 1 に削減できる。

これを実現するには、FPGA 上のメモリに物質の濃度や反応速度定数などの他に、反応経路を表現するためのデータを持っておき、それにしたがって FPGA が順次反応速度式を処理していく機構が必要である。

この機構を図示したものが図 3.6 で、図の (a) の反応経路を例とした場合の、FPGA 上でのメモリを利用したモデルの表現が (b) である。この図 3.6 (b) にある、[X] RAM と k RAM はそれぞれ物質の濃度と反応速度定数を格納するメモリであり、中央の Pathway RAM に格納されたポインタ配列によって、各反応の反応速度を計算するのに必要なデータが順次取得される。

FPGA 側で反応経路を処理する場合、図 3.6 (a) に示される反応経路中の物質 S_2 のように、複数の反応に関わる物質の濃度の転送は 1 回だけで済ませることができる。これにより、さらにデータ転送量を削減することが可能である。

3.3.4　浮動小数点演算

FPGA 内で用いる浮動小数点演算器は、単精度浮動小数点形式のものを用いた。これは、倍精度形式では演算器の面積が大きくなりすぎるためである。表 3.1 は Virtex-II FPGA 向けに開発された、商用の浮動小数点演算器の占有スライス数と単精度・倍精度での占有スライス数の比率を示したものであり、倍精度形式では単精度形式の 2 倍から 5 倍程度の違いがあることがわかる。

演算器ライブラリとしては、研究室内で開発されたもの [72] を用いた。このライブラリで用いられているフォーマット (付録 C 参照) は IEEE754 の単精度形式の仮数部を 3bit 拡張して全体で 35bit としたものであり、通常は暗黙のうちに 1 とされている仮数部の最上位の桁を明示的に 1bit 加えた、合計 36bit のビット幅を持つ。Xilinx 社や Altera 社などの商用 FPGA の内部メモリや、近年の同期 SRAM などは、ECC を付加するために 9/18/36bit 単位の幅で設計されており、この ECC ビットをデータ用に用いることで、仮数部の桁数を増加させ、演算精度を向上している。

(注 1)Pentium4 3.2GHz、FreeBSD 5.4、gcc-3.3.4 -O3

表 3.1　各 IP ベンダ提供の Virtex-II/IIPro 向け単精度・倍精度浮動小数点演算器の占有スライス数

Vendor	Precision	Add/Sub	Mult	Div
Xilinx[73]	Single	333	356	702
	Double	701	1238	3036
	Ratio	2.11	3.48	4.32
Nallatech[74][75]	Single	290	126	730
	Double	821	693	3445
	Ratio	2.83	5.50	4.72
Quixilica[76]	Single	371	155	777
	Double	815	923	3127
	Ratio	2.20	5.95	4.02

なお、本研究で開発された方式は取り扱う変数のビット幅や、使用する演算器ライブラリの種類を問わず、パイプライン化された浮動小数点演算器一般に適用できる。

3.4　システム構成

3.4.1　FPGA 上の回路の構成

FPGA 上の回路構成概要を図 3.7 に示す。FPGA 上の回路は、

- シミュレーションを行うための基本単位である Solver
  - 反応速度式を解くモジュール (Solver Core)
  - データを保持し、Solver Core を用いて数値積分を行うモジュール (Integrator)
- Solver 間を接続し、データの交換を行うためのスイッチ
  - Solver 間通信の中核となるクロスバスイッチ (Crossbar)
  - クロスバとスイッチの間のインタフェイスを行うモジュール (Transceiver)
- 各 Solver へのデータ書き込み及び計算結果読み出しを行うためのインタフェイス

から構成される。このうち、1 組の Solver Core と Integrator から成る Solver が、シミュレーション実行のためのもっとも基本的な構成要素である。Solver Core は反応速度式を解くためのモジュールであり、さまざまな反応速度式に合わせたものが実装されている [77]。これを Integrator と組み合わせることで、その Solver Core の解くことのできる反応速度式のみで構成されるモデルのシミュレーションを行うことができる。

モデルに含まれる反応速度式のセットをすべてカバーできる Solver Core が存在しない場合や、複数の Solver を用いた並列処理によって高速化を図る場合には、Solver を通信用のスイッチで結合することによってデータの共有を行う。

たとえば、図 3.8 の反応経路では、角の丸い四角形で表される反応機構 A と、角のある四角形で表される反応機構 B が混在しており、これがひとつの Solver Core で解けない場合には図 3.9 の

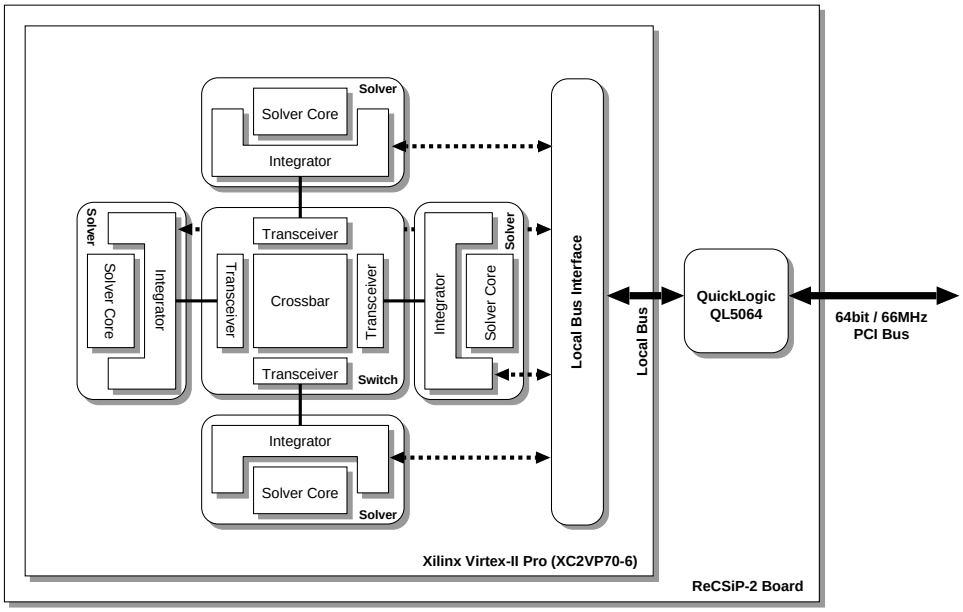


図 3.7　FPGA 上の回路構成

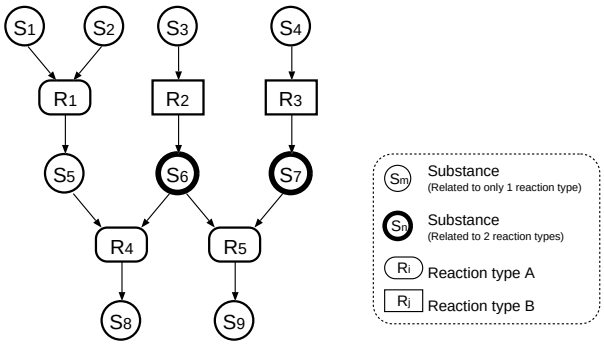


図 3.8　複数の反応機構を含むモデルの例

ように、それぞれの反応機構に対応した Solver Core を持つ 2 つの Solver を用いて処理を行うことになる。このような場合、モデルに含まれる各物質はいずれかの Solver を “Home” として持ち、その Solver 内のメモリにその濃度を保持する。

このシミュレーションの実行スケジュールは図 3.10 のようになり、S_6 と S_7 がふたつの Solver 間で共有される。これらの物質の濃度は Home から使われる Solver へ、Solver 間を接続するスイッチによって転送され、反応速度式を解いて濃度差分が求められる。濃度差分は再びスイッチを介して Home へ送り返され、そこで積分操作が行われる。

インタフェイス部は現在、ReCSiP-2 ボード (付録 A.1.2 を参照) のローカルバスの仕様に合わせたもの [78] が稼働しており、ローカルバスに接続された QuickLogic 社の PCI バスインタフェイス内蔵 FPGA である QuickPCI QL5064[79] を介して 64bit/66MHz の PCI バスに接続される。

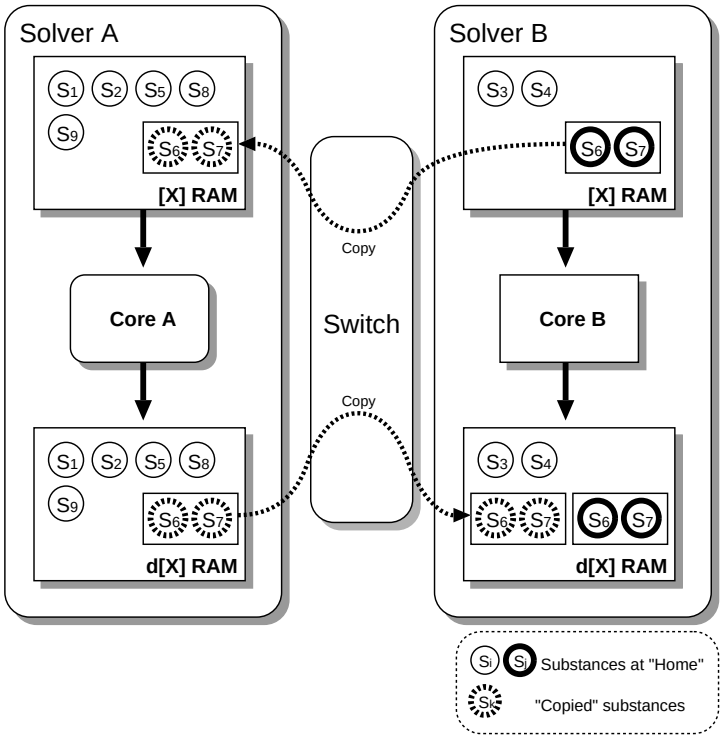


図 3.9　図 3.8 のモデルの Solver へのマッピング

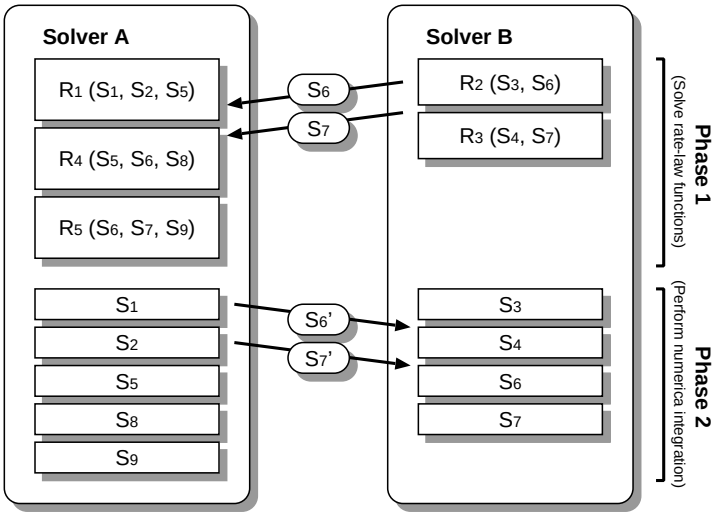


図 3.10　図 3.8 のモデルの実行スケジュールの概観

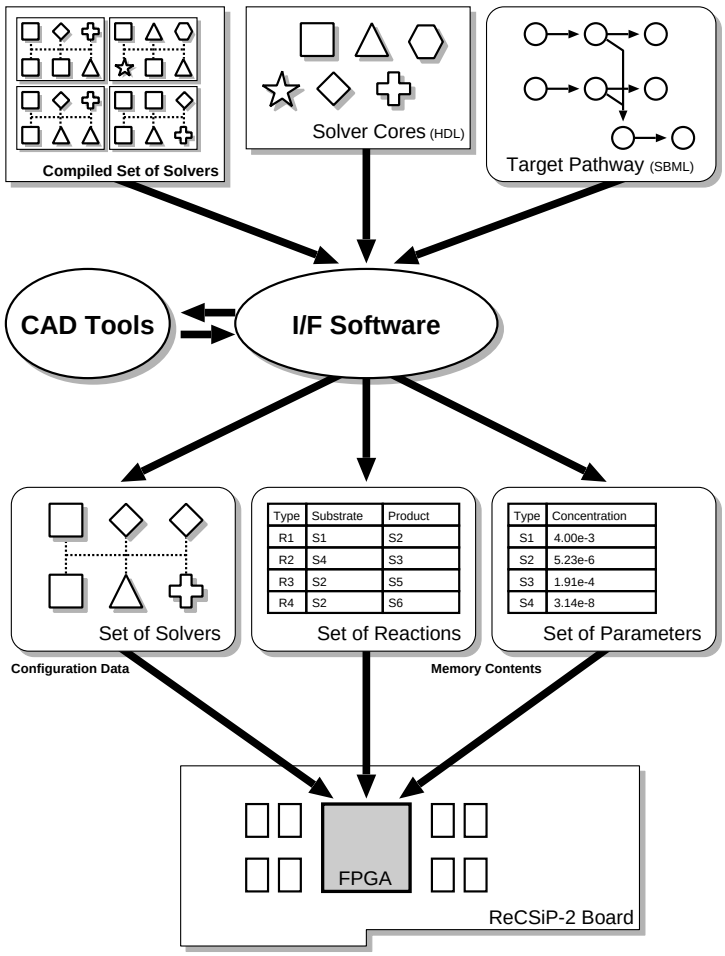


図 3.11　ReCSiP の構成

3.4.2　システム全体の構成

3.4.1 節で述べたような方式でシミュレーションを行うには、与えられたモデルに含まれる反応をすべて解くことのできる Solver Core の組み合わせを含む回路が必要となる。これを自動的に実現するためのシステムの構成を図 3.11 に示す。

回路構成やシミュレーション結果の取得などはインタフェイス部のソフトウェア [80] によって行う。このソフトウェアは入力として SBML によるモデル記述が与えられると、Solver Core のライブラリから必要な Solver Core を取り出して、積分モジュールやスイッチ、ホストインタフェイスの回路などを組み合わせた回路の RTL 記述を構成し、CAD ツールによって論理合成・配置配線を実行することで FPGA の構成情報を作成し、PCI バス経由で FPGA に回路を構成する。CAD ツールによる処理には時間がかかるため、一度配置配線された回路の FPGA 構成情報を保存しておき、それが利用できる場合には回路を新たに論理合成することなく、配置配線済みの回路を利用することで、シミュレーション開始までにかかる時間を短縮することもできる。

インタフェイス部のソフトウェアは、回路の構成に続いて、モデル中の物質と反応の Solver への割り当てを行う。物質と反応の割り当て後には濃度データのメモリへのマッピング、一連の反応を表す Pathway RAM の内容の構成、さらに Solver 間でのデータ転送を行うスイッチの転送スケ

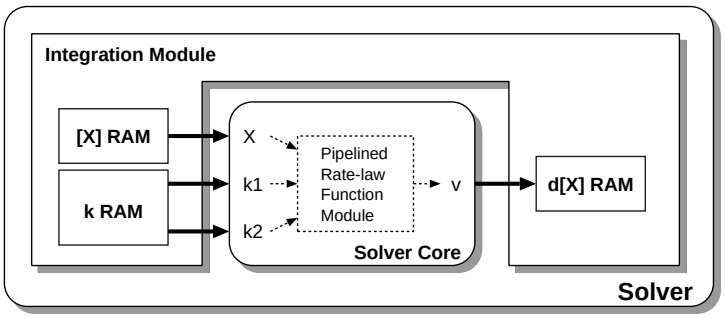


図 3.12　Solver の構成

ジュールが行われる。これらの情報は Solver やスイッチが保持するメモリの初期値として FPGA 上に構成された回路に書き込まれると、これにしたがってシミュレーションが行われる。

シミュレーションの進行中、FPGA からは積分の時間刻み毎に各物質の濃度を取得することができるようになっており、シミュレーション開始後はインタフェイスソフトウェアがこれを取得して時系列の濃度変化として保存する。

本研究では、FPGA 上でシミュレーションを行うのに必要な各モジュールの開発を行った。これらのモジュールは、必要な Solver のセットを容易に組み合わせて利用できるように設計されており、インタフェイス部のソフトウェアで回路を構成する手続きはごく単純である。

3.5　実装

3.5.1　Integrator: 数値積分モジュール

ReCSiP でシミュレーションを行うための基本単位は Solver と呼ばれるモジュールであり、これは図 3.12 に示すように、積分モジュールと Solver Core から構成される。Solver Core は各物質の濃度から反応速度を計算するためのモジュールであり、この入出力は積分モジュールに含まれるメモリブロックに接続されている。本研究ではまず Euler 法に基づく積分モジュールを実装し、これを拡張して Heun 法 (修正 Euler 法) および 4 次の陽的 Runge-Kutta 法による積分モジュールの実装を行った。

Euler 法による積分モジュール

図 3.13 に示される、Euler 法による積分モジュールが ReCSiP の積分モジュールの基本構成となる。この積分モジュールは、

- 反応経路を表す Pathway RAM
- 物質の濃度を格納する [X] RAM
- 反応速度定数などを格納する k RAM
- 時間刻み幅や stoichiometry(1 反応で消費・生成される分子数) を格納する S RAM

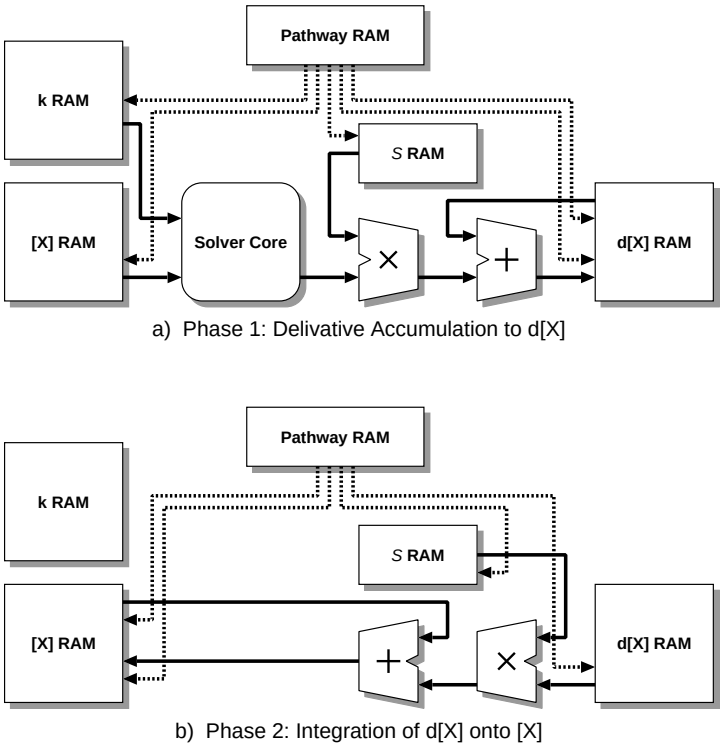


図 3.13　Euler 法による積分モジュールの構成

- 反応によって生ずる濃度変化を格納する d[X] RAM

の、5 つのメモリブロックを含み、これらを用いて、次式で与えられる 1 次の前進解法である Euler 法による数値積分を行う。

$$f' = \Delta t f(x(t)) \tag{3.2}$$

$$x(t + \Delta t) = x(t) + f' \tag{3.3}$$

一連の反応速度式は連立常微分方程式として解かれなければならない。そこで、まず式 3.2 に従ってすべての反応について反応速度を求め、これを d[X] RAM に格納する。Solver Core に入力されるデータは [X] RAM および k RAM から供給されるが、これらのアクセスに際しては、Pathway RAM によって与えられるポインタ配列によりアドレスが指定される。

一連の反応速度式を解く際に、Solver Core から d[X] RAM までの間には乗算器と加算器が備えられている。前者は S RAM から取り出した時間刻み幅、あるいはそれに stoichiometry を掛けた値を取り出して、反応速度と乗算することによって各物質の濃度変化量を求めるために用い、後者は、d[X] RAM に格納される値を、既に d[X] RAM の同じアドレスに存在する値に加算するために使用する。なお、d[X] RAM の当該アドレスへの初めての書き込みの場合には、既存の値に加算する必要はないので、一方のポートに 0 を入力することでこの加算器の動作をキャンセルする。以上が、図 3.13 (a) に示される、積分モジュールの Phase 1 の動作である。

すべての反応について式 3.2 の計算が終了すると、続いて Phase 2 として式 3.3 の計算を行う (図 3.13 (b)。) Phase 2 では、Phase 1 で用いたのと同じ乗算器と加算器を用い、乗算器で S RAM から取り出した値を d[X] RAM からの値と乗算し、加算器で [X] RAM の値に加算を行う。Phase 2

はさらに 2 段階の処理に分割されており、前半の処理では [X] RAM に対し、d[X] RAM の同じ番地の値を加算する操作がアドレス 0 から、Pathway RAM で指定されたアドレスまで行われ、この間は乗算器の S RAM 側には+1 が入力される。後半の処理では Pathway RAM によって d[X]、S、[X] RAM のアドレスをそれぞれ指定し、d[X] RAM 中の任意の変数に S RAM 中の任意の値を乗算し、[X] RAM 中の任意の変数に加算することができる。Phase 2 前半の処理は、後半と同様に個別にアドレスを記述しても実現できるが、指定したアドレス空間内の変数を自動的に加算する処理を可能にしておくことで、Pathway RAM 中の積分操作の命令数を削減している。

図 3.14 に、1 つの Solver が単独で行う簡単なシミュレーションの例として、その反応速度の計算と積分操作の例を示す。この図では、3.3.2 節の例として挙げた Solver Core(図 3.3) のように、ひとつの反応で基質 S_1 と生成物 S_2 の濃度がそれぞれ変化するものを想定している。

この場合、ひとつの反応で変化する [X] RAM の変数は 2 個あるが、図 3.3 の Solver Core のパイプラインピッチが 1 であるため、Phase 1 では d[X] RAM にひとつの変数しか書き込みをすることができない。この場合、図 3.14 に示すように、Phase 1 では時間刻み幅だけを反応速度に掛けて d[X] RAM に格納し、Phase 2 で d[X] RAM の値にそれぞれの物質の増減分を掛けて [X] RAM の値に加算する、という方法をとることで反応に関与する分子数と Solver Core のパイプラインピッチの関係の自由度を確保している。

Heun 法 (修正 Euler 法) への拡張

Heun 法は修正 Euler 法とも呼ばれ、Euler 法の精度を向上するため、2 回の計算で時間刻みを 1 つ進めるように改良したもので、時間刻み幅を Δt とすると、次式で与えられる。

$$f_1' = \Delta t f(x(t)) \tag{3.4}$$

$$f_2' = \Delta t f(x(t) + f_1') \tag{3.5}$$

$$x_(t + \Delta t) = x(t) + (f_1' + f_2')/2 \tag{3.6}$$

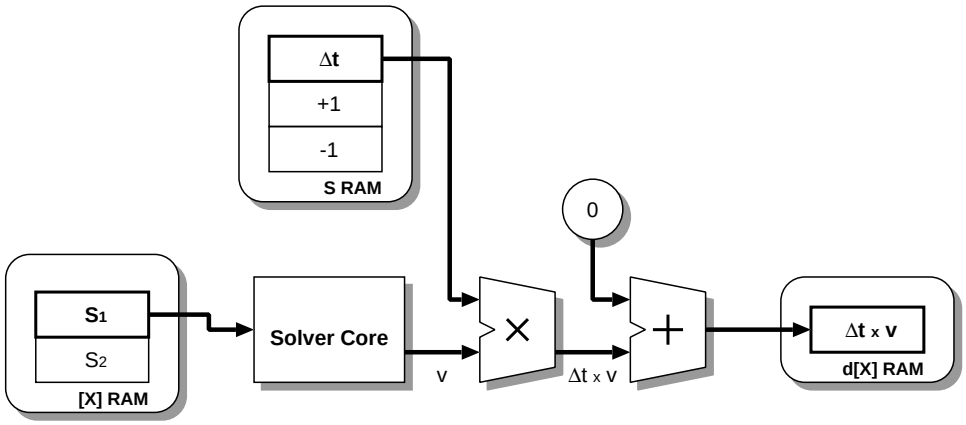
この方法が Euler 法よりも精度的に優れている例として、単振動の方程式

$$v' = kx \tag{3.7}$$

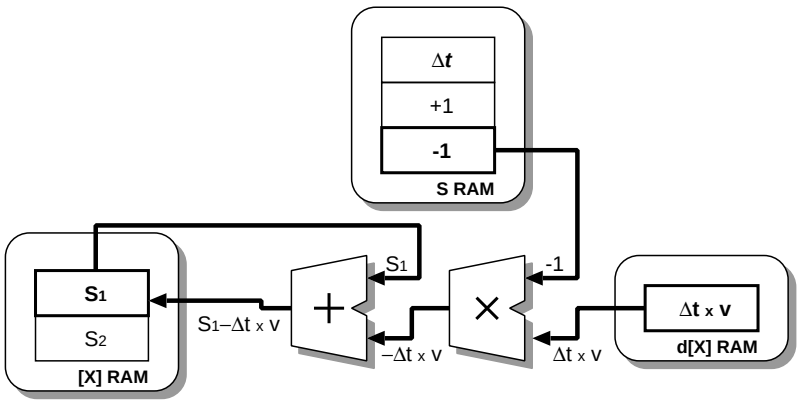
を解いた例を挙げる。この式において v は速度、k はバネ定数で、x は位置である。単振動の場合、位置 x と速度 v をそれぞれ縦軸と横軸にとってプロットすると円を描くことが知られている。バネ定数を 1、x の初期値を 1、v の初期値を 0 として、時間刻みを 0.0001 にとったグラフが図 3.16、0.01 にとったグラフが図 3.17 である。図 3.16 のように、時間刻みが充分に小さい場合には Euler 法でも円になるが、図 3.17 のように時間刻みが大きくなると、Heun 法では円になっているのに対して、Euler 法では結果が発散してしまう様子がわかる。このように Euler 法では、定常的に振動を続けるような場合でも時間刻み幅に充分な注意を払わないと誤差が蓄積してしまう。

このアルゴリズムによる積分モジュール実装を図 3.15 に示す。一連の常微分方程式は、連立微分方程式として解かなければならないので、まずすべての f_1' を求めてから f_2' を求める必要がある。そこで、2 ステップ分のデータを保持するために、[X] RAM と d[X] RAM を 2 セット用意した。

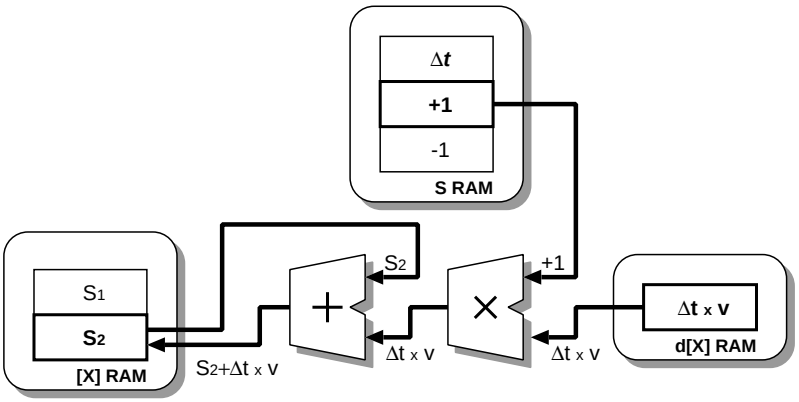
最初のステップの前半のフェーズでは [X] RAM 1 に格納されている、前の時間刻みが終了した時点での [X] から f_1' を計算し、d[X] RAM 1 に格納する。この処理の間に、[X] RAM 1 から Solver Core に送られたデータを [X] RAM 2 にコピーしておき、[X] RAM 1 には、f_1' が加算された仮の



Step 1: $S_1 \rightarrow S_2$ in Phase 1



Step 2: $S_1 \rightarrow S_2$ in Phase 2 (modifying S_1)



Step 3: $S_1 \rightarrow S_2$ in Phase 2 (modifying S_2)

図 3.14　反応速度式の計算と積分操作の例

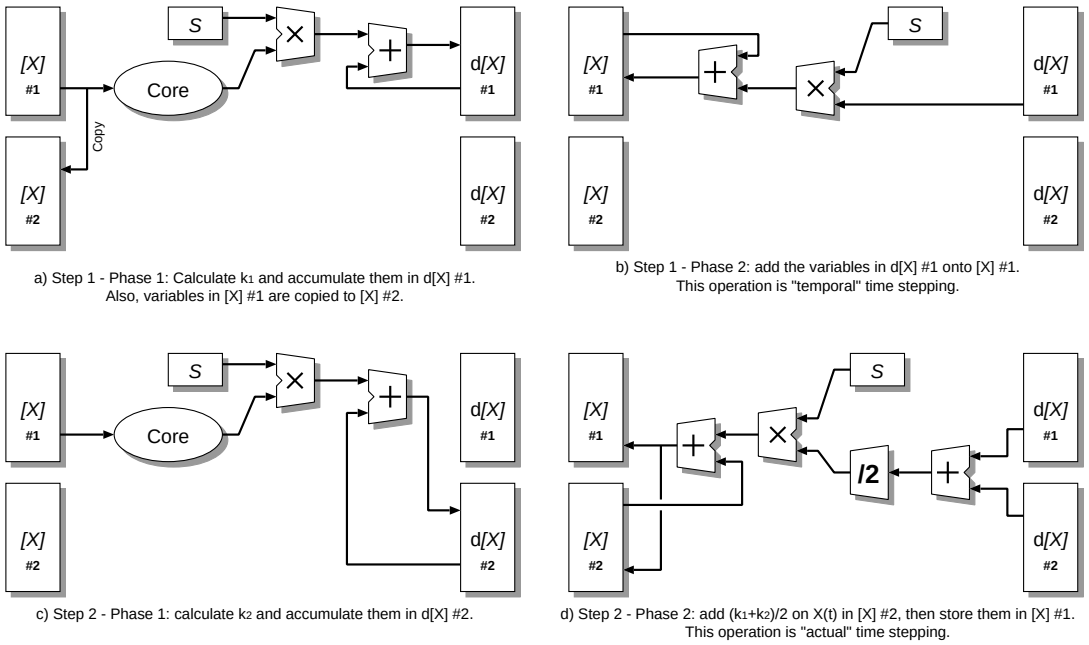
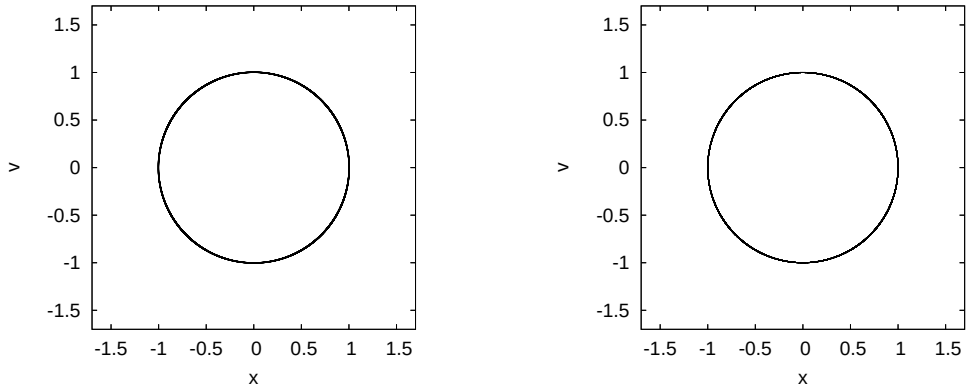
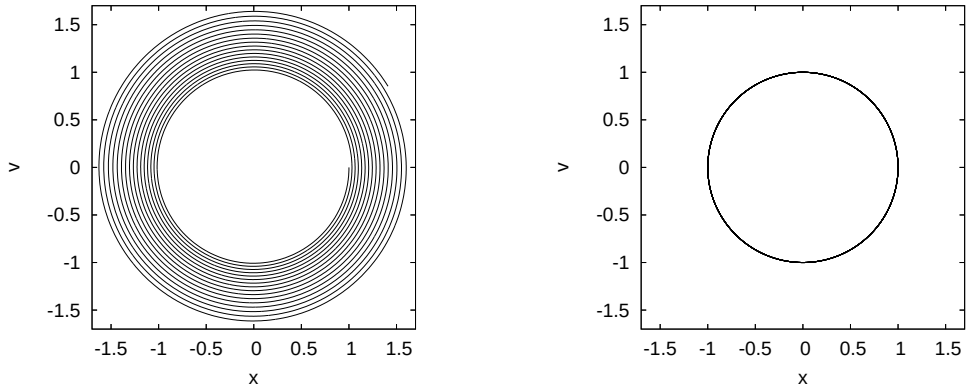


図 3.15　Heun 法による積分モジュールの構成



(a) Euler 法　　(b) Heun 法

図 3.16　時間刻み $\Delta t = 0.0001$ のときの Euler 法、Heun 法による単振動のプロット



(a) Euler 法　　(b) Heun 法

図 3.17　時間刻み $\Delta t = 0.01$ のときの Euler 法、Heun 法による単振動のプロット

値が格納される。次に、これをもとにして f_2' を求め、d[X] RAM 2 に格納する。最後に [X] RAM 2 にコピーされた値と、f_1'、f_2' の平均を加算して、[X] RAM 1 に格納することで 1 時間刻み分の処理を完了する。

この一連の手続きは、2 度目の積分操作に用いるパイプラインが若干長くなるものの、基本的には Euler 法での 1 時間刻み分の処理を 2 回繰り返すのと等しいため、Pathway RAM をはじめとするメモリの内容は Euler 法のものと同じでよく、インタフェイスソフトウェアがコードを生成する際には積分方法を区別する必要がない。

Runge-Kutta 法への拡張

Runge-Kutta 法は、さらに計算精度や安定性を向上するために広く用いられている方法であり、ReCSiP 向けには一般的な 4 次の陽解法を実装した。4 次の陽的 Runge-Kutta 法は次式で与えられる。

$$f_1' = \Delta t f(x(t)) \tag{3.8}$$
$$f_2' = \Delta t f(x(t) + f_1'/2) \tag{3.9}$$
$$f_3' = \Delta t f(x(t) + f_2'/2) \tag{3.10}$$
$$f_4' = \Delta t f(x(t) + f_3') \tag{3.11}$$
$$x(t + \Delta t) = x(t) + (f_1' + 2f_2' + 2f_3' + f_4')/6 \tag{3.12}$$

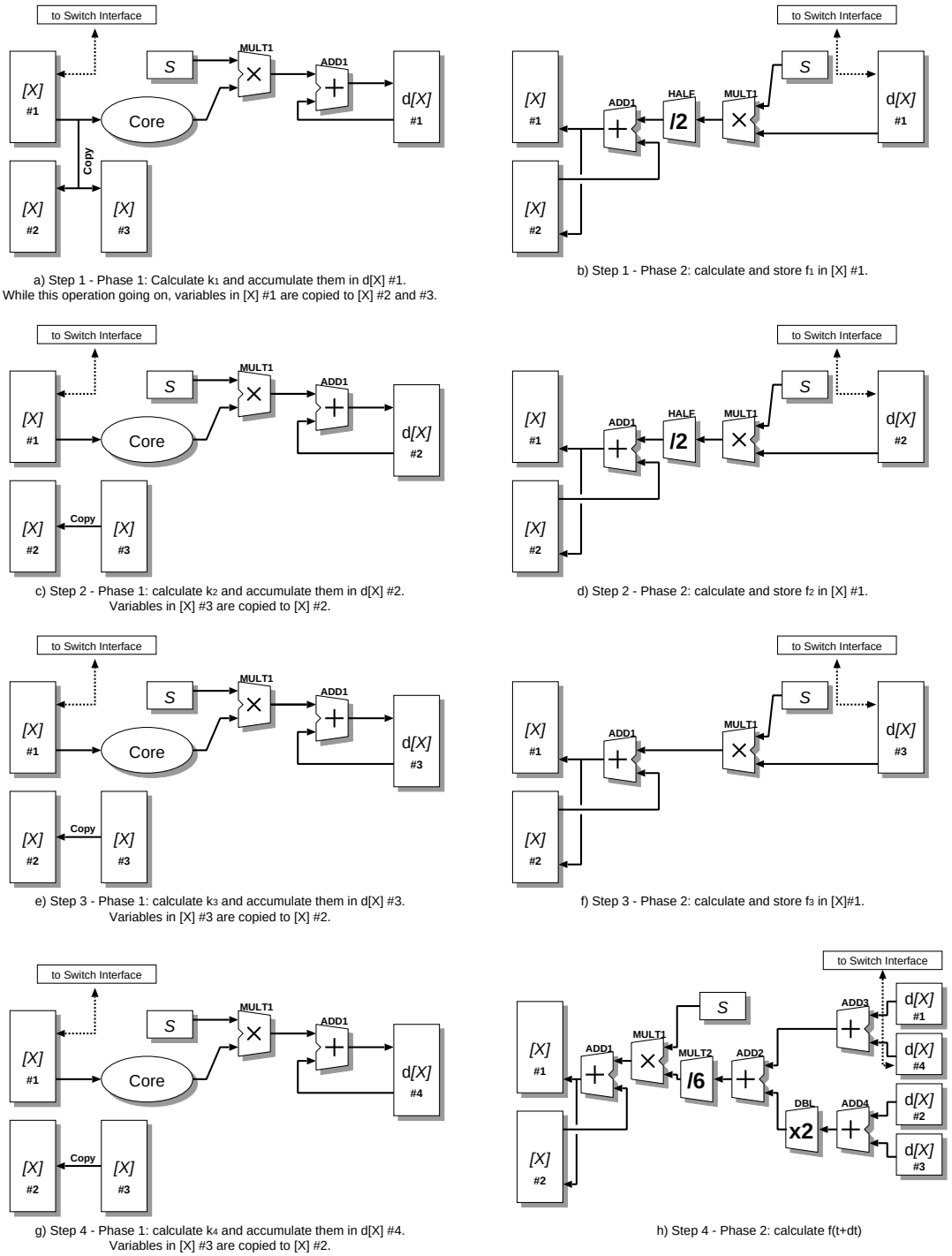
Runge-Kutta 法が Heun 法よりも優れている場合の例としては、0 に向かって収束するような式を解く場合、たとえば

$$dy/dx = -100x \tag{3.13}$$

のような場合が挙げられる。これを y について積分すると

$$y = \exp(-100x) \tag{3.14}$$

となり、0 に向かって収束する関数となる。これを各種の時間刻みで計算してプロットしたものが図 3.19 であり、時間刻みが充分に小さい場合には 3 つのアルゴリズムともにほぼ理論値通りの値



a) Step 1 - Phase 1: Calculate k_1 and accumulate them in d[X] #1.
While this operation going on, variables in [X] #1 are copied to [X] #2 and #3.

b) Step 1 - Phase 2: calculate and store f_1 in [X] #1.

c) Step 2 - Phase 1: calculate k_2 and accumulate them in d[X] #2.
Variables in [X] #3 are copied to [X] #2.

d) Step 2 - Phase 2: calculate and store f_2 in [X] #1.

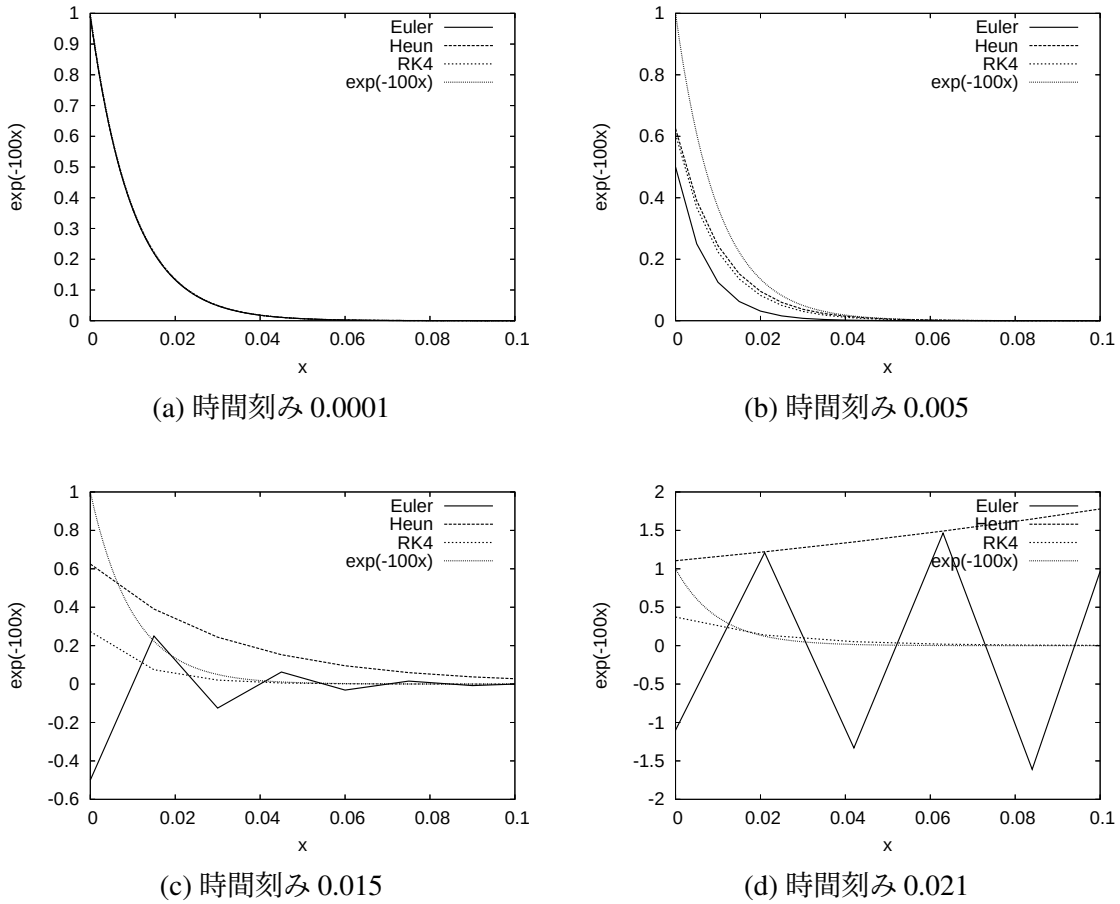
e) Step 3 - Phase 1: calculate k_3 and accumulate them in d[X] #3.
Variables in [X] #3 are copied to [X] #2.

f) Step 3 - Phase 2: calculate and store f_3 in [X]#1.

g) Step 4 - Phase 1: calculate k_4 and accumulate them in d[X] #4.
Variables in [X] #3 are copied to [X] #2.

h) Step 4 - Phase 2: calculate f(t+dt)

図 3.18　Runge-Kutta 法による積分モジュールの構成



(a) 時間刻み 0.0001　(b) 時間刻み 0.005

(c) 時間刻み 0.015　(d) 時間刻み 0.021

図 3.19　Euler 法、Heun 法、Runge-Kutta 法による $\exp(-100x)$ のプロット

になり、刻み幅を大きくしていくと Euler 法は発振、Heun 法も発散してしまい、Runge-Kutta 法だけが 0 に向かって収束するという結果となっている。このように、Runge-Kutta 法は他の陽解法にくらべて安定性に優れており、広く数値解析に使われている。

Runge-Kutta 法も、Heun 法と同様に Euler 法の積分モジュールを拡張し、複数のメモリセットを持たせることで実現できる (図 3.18)。[X] RAM は 3 組あり、ふたつは Heun 法と同じように、残りのひとつは最初の [X] RAM の値 ($x(t)$) を参照するために用いる。4 組の d[X] RAM はそれぞれ、f_1'、f_2'、f_3'、f_4' を保持するために用いられる。

この積分モジュールを用いる場合、1 時間刻みに行う処理は Euler 法での 1 時間刻み分の処理を 4 回繰り返すのと同じ手続きである。したがって、Euler 法、Heun 法で用いるのと同じ Pathway RAM の内容を用いることができ、これら 3 つの積分モジュールのいずれを用いてもインタフェイスソフトウェアによるスケジューリングを同様に行うことができる。

3.5.2　Solver Core: 反応速度式モジュール

ReCSiP では、図 3.3 に示すように、パイプライン化された浮動小数点演算器を組み合わせたモジュール (Solver Core) を用いて反応速度式を計算する。Solver Core の基本的な外部インタフェイ

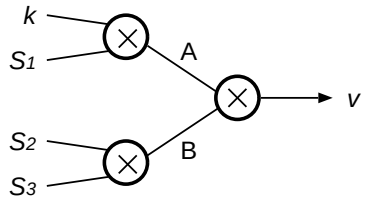


図 3.20　式 3.15 のデータフローグラフ

スは図 3.12 に示すように、入力として X、K1、K2 の 3 つのポート、出力として V という 1 つのポートがあり、それぞれ積分モジュールに接続されている。これらの入出力ポートはすべて 3.3.4 節で述べた浮動小数点フォーマットに合わせて 36 ビットのデータ幅を持つ。

また、これらのデータ入出力に加えて、複数の反応速度式をカバーする Solver で反応速度式を選択するための 4bit の信号 (F) が設けられている。

Solver Core の具体的な構成例について、本節の以下の部分で述べる。

パイプラインの構成方針

Solver Core は複数の浮動小数点演算器から構成されるが、Solver Core 全体を完全にパイプライン化し、毎クロック新たな反応についての反応速度式を解くことができるようにすると、反応速度式に含まれる演算子の数に従って演算器の数が増加してしまう。これは FPGA の有限な回路容量を利用する上で好ましくないため、反応速度式を一度解く間にひとつの演算器を反復利用して演算器の数を減らすことが考えられる。

たとえば、質量作用則を用いると、多数の基質をもつ反応の反応速度を

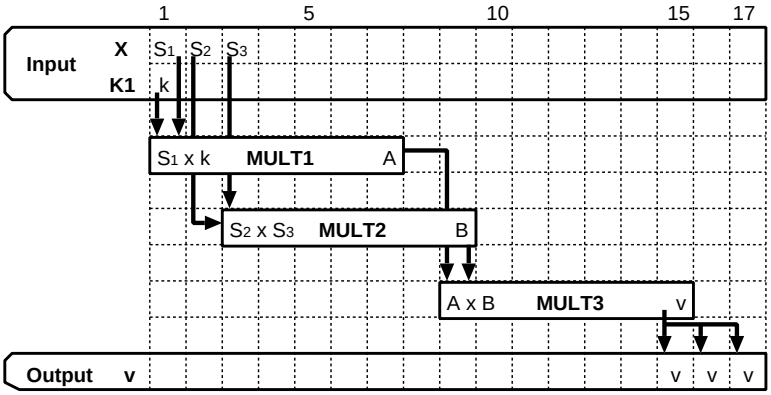
$$v = k \prod_i s_i \tag{3.15}$$

のように書くことができる。これをデータフローグラフにすると、たとえば $i_{\max} = 3$ の場合には図 3.20 のようになり、そのまま演算器を並べてスケジューリングすると図 3.21 (a) のように 3 つの乗算器が必要となる。しかし、このパイプラインスケジュールはデータの入力に 3 クロックを要するため、3 つの乗算器は 3 クロックに 1 度しか使われない。これを、ひとつの乗算器を 3 回反復して利用するような構成にすれば、面積が約 3 分の 1 で済むことになり、この場合のパイプライン構成が図 3.21 (b) である。

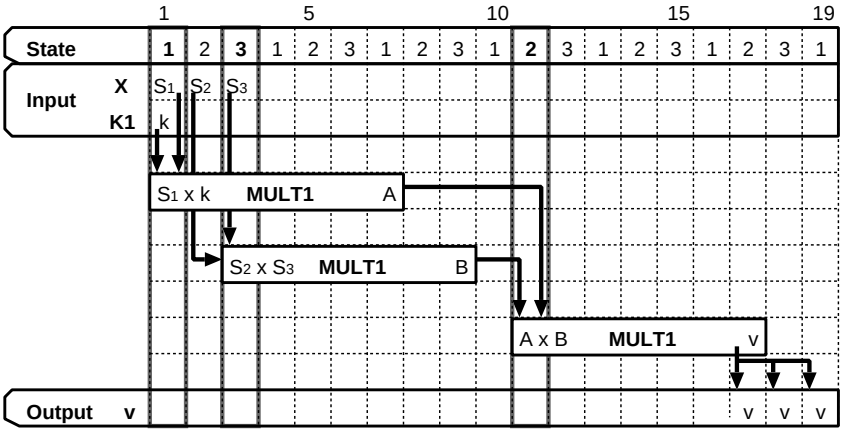
この構成を回路として実装したものが図 3.22 である。回路構成に示すように、入力ポートや演算器の出力を一定のクロックサイクル数遅延させて次の演算器等に入力するにはシフトレジスタを、制御用のステートマシンの状態によって演算器等の入力を切り替える場合にはマルチプレクサを用いる。

図 3.21 の構成では、状態 1、2、3 の 3 状態を、$1 \to 2 \to 3 \to 1$ の順に毎クロック切り替えるステートマシンを用意しておき、乗算器の入力を

- 状態 1: 入力ポート X(変数 S_1) および K1(変数 k)
- 状態 2: 乗算器の出力に接続された 4 段シフトレジスタの出力 (中間変数 A) および 2 段シフトレジスタの出力 (中間変数 B)



a) Non-Shared



b) Shared

図 3.21　図 3.20 に基づくパイプラインスケジュール

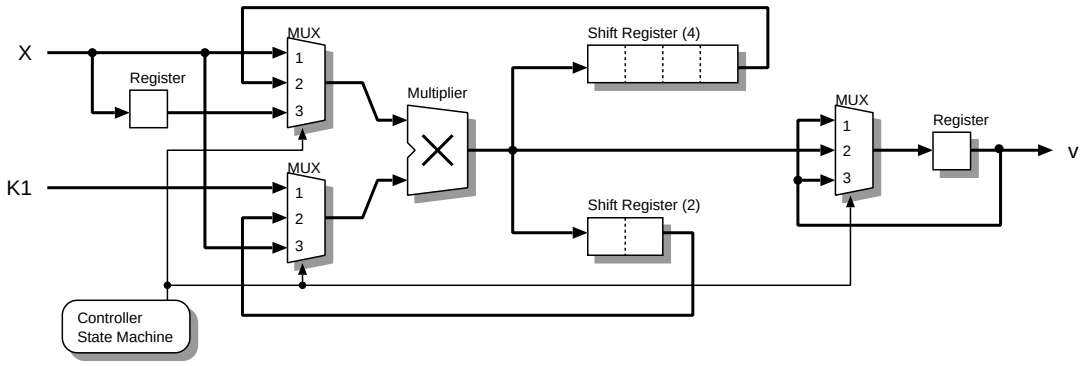


図 3.22　図 3.21 (b) の回路への実装

- 状態 3: 入力ポート X(変数 S_3) および入力ポート X に接続された 1 段のレジスタ (変数 S_2)

のように切り替えることで、ひとつの乗算器を用いて順に計算を行い、回路面積を抑えることができる。

複数の反応速度式を解く Solver Core の例

FPGA 上に同時に構成できる回路のサイズには限界があるため、必然的に Solver の個数にも制限が生じる。そこで、極端に稼働率の低い Solver が存在するとシステム全体の性能低下につながるため、複数の反応速度式を解くことのできる Solver Core を用いて、Solver 間の負荷の均等化を計ることが重要である。

複数の反応速度式に対応した Solver Core を設計する場合には、反応速度式が切り替わるときにパイプラインが乱れないように設計することが望ましい。たとえば、次の 2 つの反応速度式[(注 2)]について考えてみる。

$$v = \frac{V_m S}{K_m + S} \tag{3.16}$$

$$v = \frac{V_f S/K_{mS} - V_r P/K_{mP}}{1 + S/K_{mS} + P/K_{mP}} \tag{3.17}$$

これを加減算器、乗算器、除算器を用いて計算する場合のデータフローとして描いたものが図 3.23 で、(a) が式 3.16 を計算する場合、(b) が式 3.17 を計算する場合である。同じ回路を用いて計算するにはそれぞれのデータフローグラフを極力同じ形に近づけることが望ましいので、(a) の中央下では 0 との加算を行い、乗算器から加減算器を経由して除算器に至る経路が (b) と同じ形になるようにしている。

これを基に、3 種類の演算器を使った Solver Core を構成した場合のパイプラインスケジュールが図 3.24 である。加算器と除算器をそれぞれ 2 つ、乗算器を 1 つ用いており、パイプラインピッチは 2 である。各演算器は反応速度式を 1 回解くのに最大 2 回使われるが、この Solver Core がサポートするふたつの反応速度式のパイプラインスケジュールは、(a) を (b) のサブセットを元に設計し、両者を連続して解く場合にも交互に解く場合にも、演算器の利用が衝突しないようになっている。

このように演算器の利用スケジュールが衝突しないようにパイプラインを設計することで、異なる反応速度式をひとつの Solver Core で連続処理することが可能になる。これによって Solver Core の稼働率が向上し、面積あたりの実効性能向上を実現することができる。

現在、SBML Level 1 Version 2[81] で標準として定められている反応速度式のうち 20 をカバーする 30 余の Solver Core がライブラリ化されている。

3.5.3 スイッチ

複数の Solver を用いる必要がある場合には、図 3.9 のように、複数の Solver をスイッチで接続し、それぞれの Solver に物質と反応を割り当てて計算を行うことになる。それぞれの物質はいずれかの Solver を Home とし、その [X] RAM に濃度が格納されるが、その Home 以外の Solver がこれを必要とする場合には、スイッチによるデータの転送が必要になる。

[(注 2)]式 3.16 は Irreversible Simple Michaelis-Menten で、V_m が最大反応速度、K_m が Michaelis-Menten 定数である。式 3.17 は Uni-Uni Reversible Simple Michaelis-Menten で、V_f が正反応の、V_r が逆反応の最大反応速度であり、K_{mS} が基質の、K_{mP} が生成物の Michaelis-Menten 定数である。

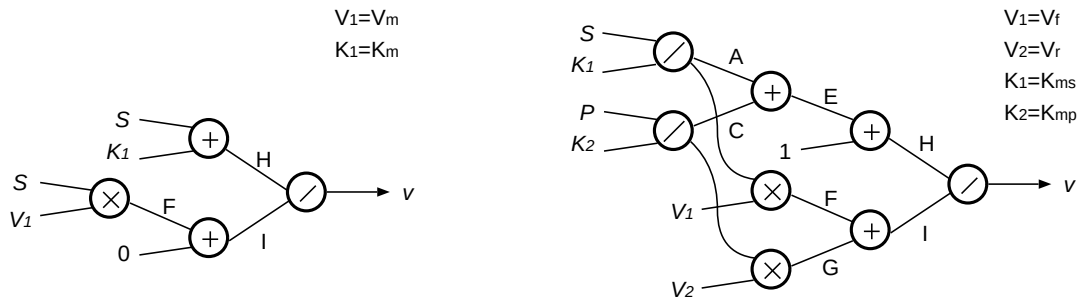


図 3.23　式 3.16 および 3.17 のデータフローグラフ

ここで、データ転送が必要になるケースは、

- Phase 1 では Home となる Solver の [*X*] RAM から、データを必要とする [*X*] RAM へ
- Phase 2 では、Phase 1 の転送先の d[*X*] RAM から、Home の d[*X*] RAM へ

の 2 パターンである。[*X*] RAM や d[*X*] RAM は、FPGA 上の Block RAM を用いて実装されており、Block RAM は dual-port 構成であるため、図 3.25 に示すように、Phase 1 では [*X*] RAM の Solver Core に接続されない側のポートを、Phase 2 では d[*X*] RAM の乗算器に接続されない側のポートを用いて、それぞれ他の Solver 内のメモリとデータ交換を行える。

Solver 間を接続するスイッチの中核はクロスバスイッチであり、そのポートと Solver のメモリを接続するためのインタフェイスとなるモジュールが Transceiver である (図 3.26)。Transceiver は、積分モジュールの Pathway RAM と同様に Code RAM と呼ばれるメモリを用い、メモリのアドレス、クロスバへ与えるデータの送信先などを制御する機構を持っている。また、Solver から読み出したデータを一時的に保持するためのバッファメモリが設けられており、データ送受信の柔軟性を向上している。

なお、クロスバスイッチは、複数の宛先にデータを同時に送信することができ、これによって 3 つ以上の Solver で物質を共有する際のデータ転送時間を削減している。

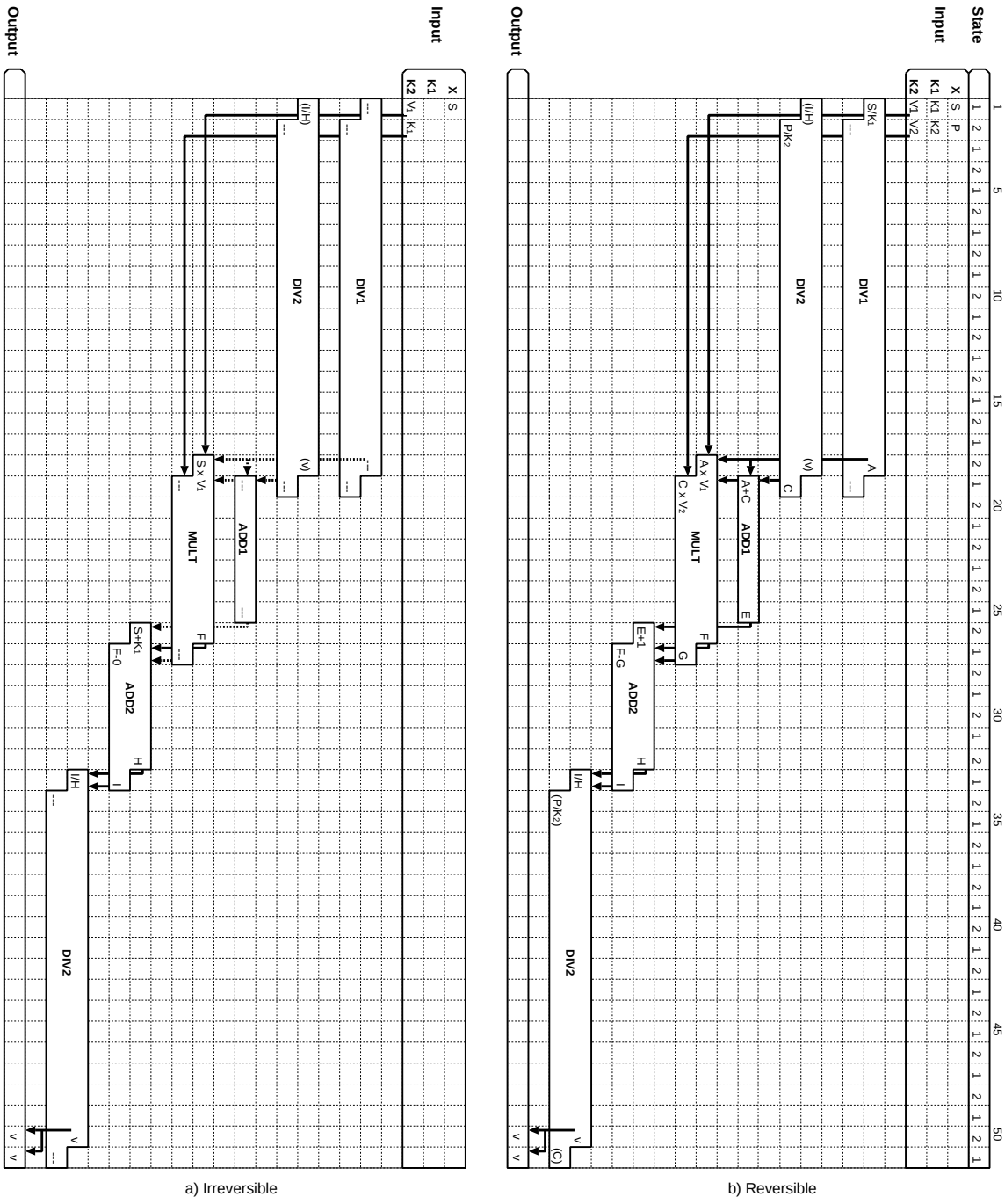
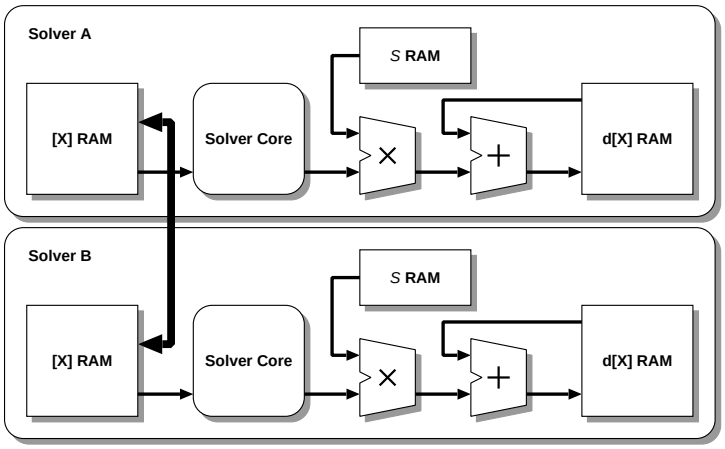
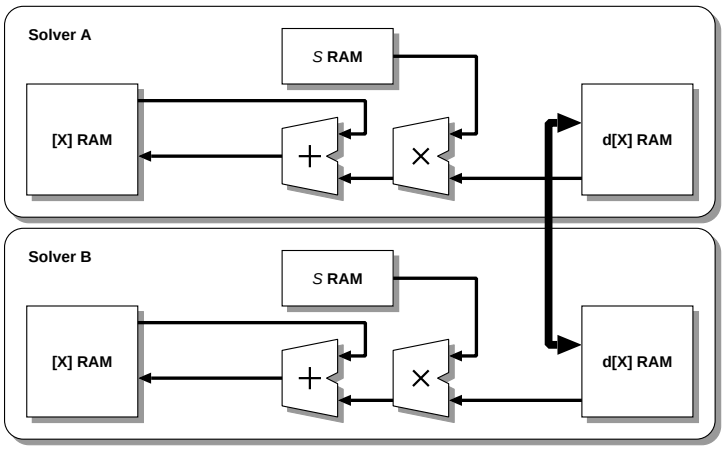


図 3.24　式 3.16 および 3.17 のパイプラインスケジュール



a) Phase 1: Transfer from [X] to [X]



b) Phase 2: Transfer from d[X] to d[X]

図 3.25　Dual-port メモリの空きポートを利用したデータ転送

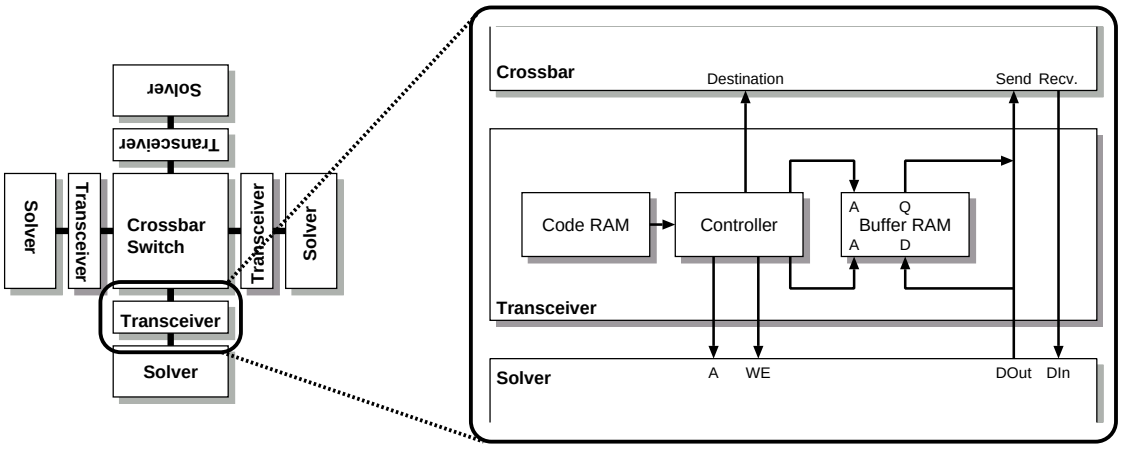


図 3.26　Transceiver の構成

第4章　評価および考察

4.1　回路面積

4.1.1　各モジュールの面積

表 4.1 に各積分モジュールの面積を、表 4.2 および図 4.1 にスイッチの面積を、それぞれ示す。現在実装が完了している Solver Core が平均で 4,185 スライスを占有 [77] することと、ReCSiP のリファレンス環境として開発された ReCSiP-2 Board に搭載されている FPGA である Xilinx XC2VP70 は 33,088 のスライスを持つことから、

- Euler 法では 5〜6 個
- Heun 法では 4〜5 個、
- Runge-Kutta 法では 3〜4 個

程度の Solver をひとつの FPGA 上に搭載できる見積もりとなる。スイッチはクロスバを中心に持つため、ポート数の増加につれて面積が増加する。しかし、実質的に現在の FPGA で利用できる Solver の数は最大でも 6 程度であり、この規模のスイッチは FPGA の全スライス数の 5% 以内に収まる。したがって、90% 以上の面積を演算に用いることができるので、通信機構の追加による面積的なオーバーヘッドは充分に小さいといえる。

4.1.2　浮動小数点演算器とランダムロジックの面積比

ReCSiP の性能向上のための基本コンセプトは、複数の浮動小数点演算器を組み合わせて深いパイプラインを構成し、高いスループットを実現することである。このような場合、並列動作する複数の演算器間で待ち合わせをするためのシフトレジスタや、データパスを切り替えるためのマルチプレクサなどが演算器周辺の回路として必要になるが、演算器の数が高い計算能力の鍵となるため、総面積に対して演算器の占める割合が高いことが理想的である。特に、シフトレジスタは Pathway RAM から読み出された情報を d[X] RAM への書き込みまで保存するなど、数十段に

表 4.1　積分モジュールのリソース使用量

アルゴリズム	Euler	Heun	RK4
スライス	1,469	2,346	4,273
18x18 乗算器	4	4	8
BlockRAM	15	19	25

表 4.2　スイッチのリソース使用量と XC2VP70 全体に占める割合

ポート数	スライス数				BlockRAM	
	全体 (%)		クロスバ (%)		ブロック数 (%)	
3	495	(1.4)	108	(0.3)	12	(0)
4	542	(1.6)	153	(0.5)	16	(4.9)
5	1,020	(3.1)	373	(1.1)	20	(6.1)
6	1,263	(3.8)	463	(1.4)	24	(7.3)
7	1,781	(5.4)	550	(1.7)	28	(8.5)
8	2,570	(7.8)	1,033	(3.1)	32	(9.8)
9	2,973	(9.0)	1,483	(4.5)	36	(11.0)
10	3,102	(9.4)	1,207	(3.6)	40	(12.2)
11	3,888	(11.8)	2,038	(6.2)	44	(13.4)
12	4,381	(13.2)	2,077	(6.3)	48	(14.6)
13	4,539	(13.7)	2,331	(7.0)	52	(15.9)
14	4,967	(15.0)	2,682	(8.1)	56	(17.1)

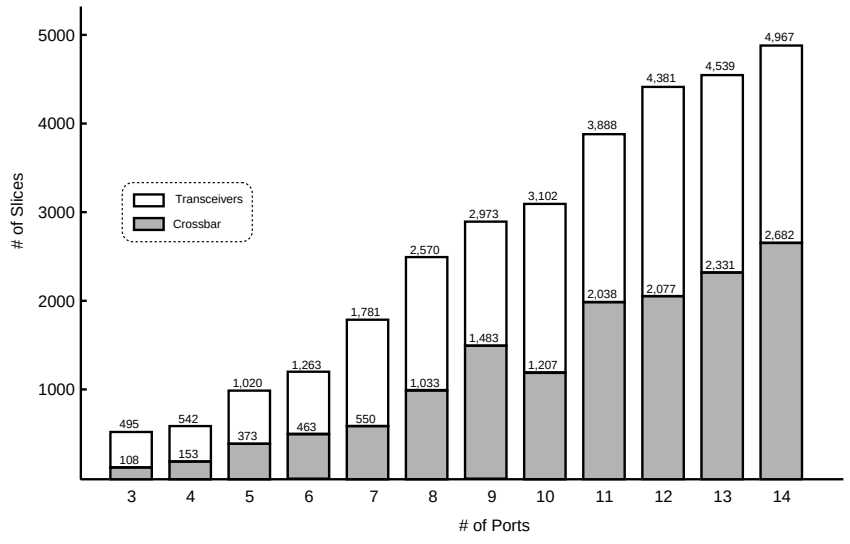


図 4.1　ポート数とスイッチの面積の変化

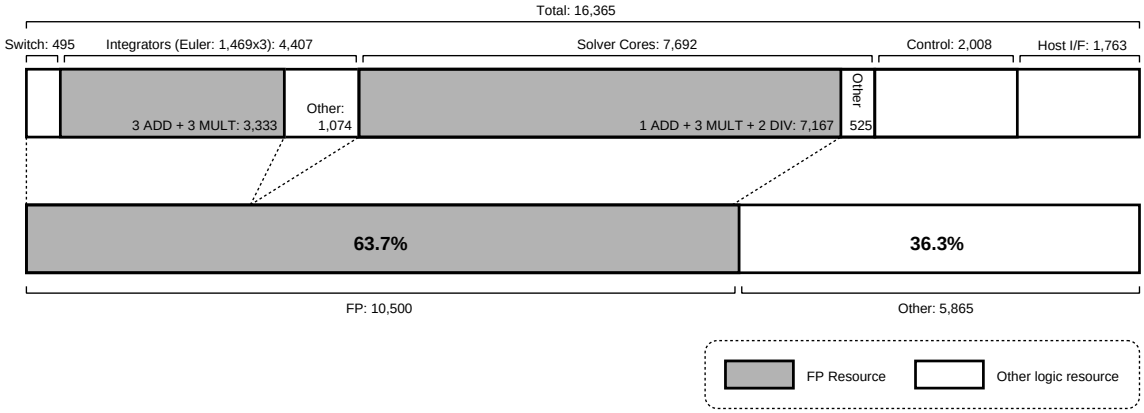


図 4.2　Minimal Mitotic Oscillator モデルをシミュレーションする場合の回路面積プロファイル

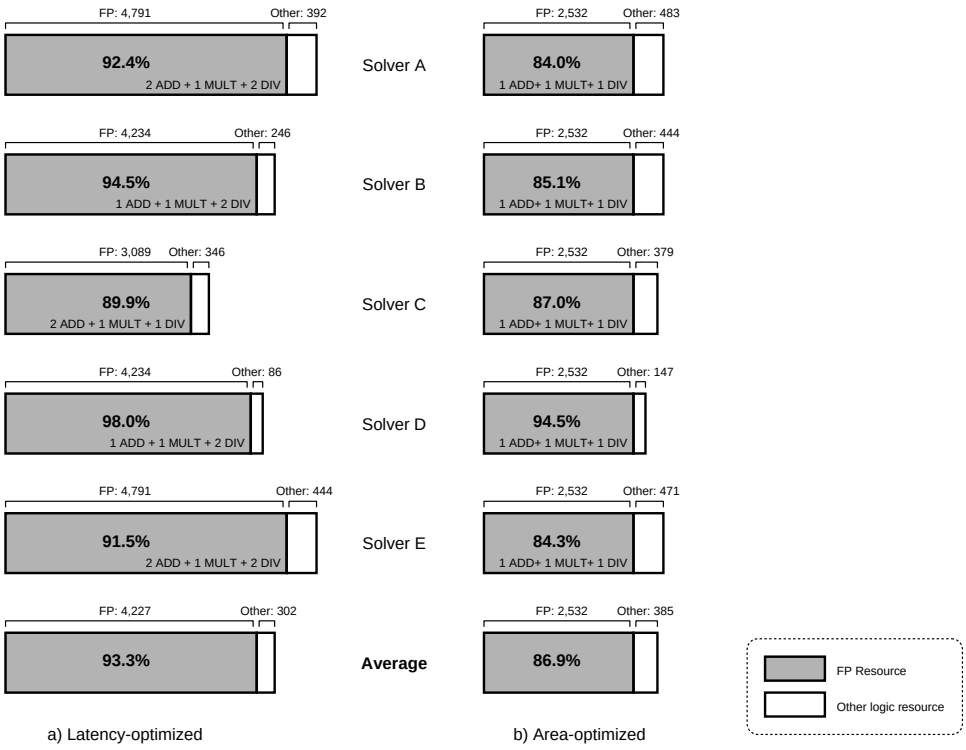


図 4.3　Solver Core の回路面積消費プロファイル

表 4.3　動作周波数とスループット

システム	Euler	Heun	RK4
Pentium4 (gcc3.3.4 -O3) 3.2GHz	6.43	3.14	1.17
ReCSiP (4 solvers) 90MHz	360	180	90
ReCSiP (5 solvers) 90MHz	450	270	N/A
ReCSiP (6 solvers) 90MHz	540	N/A	N/A

(10^6 Reactions/sec.)

及ぶものを多く用いるため、演算器以外の部分がどれだけの面積を占めているかを評価しておくことが重要である。

図 4.2 は、Euler 法で minimal mitotic oscillator モデル (第 2.2.2 節参照) をシミュレーションするための Solver セットを FPGA 上に構成した場合の回路面積プロファイルである。この回路は 3 つの Solver を含み、ホストとのインタフェイス部まで含めて 16,365 スライス (XC2VP70 の 49.5%) である。16,365 スライスのうち、各種の浮動小数点演算器が占めるのは 10,500 スライス (63.7%) となっており、回路の半分以上が演算器で占められている。また、積分モジュールは比較的複雑なデータパスの制御を行うため、Solver Core よりも演算器以外のランダムロジックの割合が多くなっていることがわかる。

一方、図 4.3 は、5 種類の Solver Core の面積プロファイルを示したもの [77] であり、それぞれの Solver Core は数種類ずつの反応速度式を解くことができる。図の左側は演算器の数に制約を与えずにパイプラインの遅延を最小化する設計、右側は演算器の数を極力抑えて面積を最小化する設計である。面積を最小化する場合にはデータ待ち合わせのためのパイプラインや、演算器間のデータパスを切り替えるマルチプレクサなどが増加するため、浮動小数点演算器の面積は遅延最小化設計の場合には 93.3% であったものが 86.9% と、ランダムロジックの割合が増加していることがわかる。これは面積最小化設計において総面積が平均 35% (1612 スライス) 減と大きく減少しているためで、ランダムロジック自体の面積増加率は 80 スライス程度に留まっている。

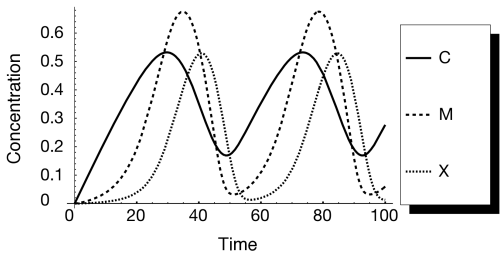
以上の評価結果から、ReCSiP の設計は、演算部と制御部の回路面積の比率的に充分に効率的なものであるといえる。

4.2　計算速度

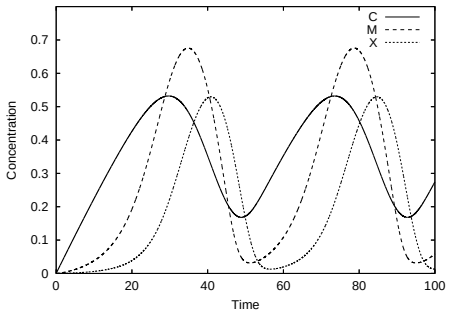
4.2.1　理論ピーク性能

例として Irreversible Michaelis-Menten 型の反応速度式 (式 3.16) を解く処理の ReCSiP(90MHz 動作時) とソフトウェアによる最大スループットの比較を表 4.3 に示す。

この表から、パイプラインを満たすことができる、充分に大きな反応系であれば、汎用マイクロプロセッサの 80 倍程度の性能を発揮できることがわかる。この値は、反応速度式を解くだけの処理であるため、Phase 2 の積分操作を含んでおらず、現在の実装では実際のシミュレーションにおいてこの数字通りのスループットを出すことはできないが、4.3.2 節で述べるような実装方法を用いて Phase 1 と Phase 2 の動作をオーバーラップさせることで、容易に実現可能である。



a) MathSBML によるシミュレーション結果



b) ReCSiP によるシミュレーション結果

図 4.4　Minimal Mitotic Oscillator Model のシミュレーション結果 ($t_{\max} = 100$)

また、ReCSiP による計算は深いパイプラインを用いた処理のため、小さなモデルのシミュレーションではパイプラインの空きが大きくなり、実効性能が低くなることが予想される。しかし、4.3.1 節で述べるように、パラメータ空間を探索するような用途ではパラメータを変えた複数のシミュレーションを同時に実行することで、パイプラインを有効に利用して、より高いスループットを実現できると考えられる。

実際のシミュレーションモデルを用いた実効性能評価については、次節以降で議論する。

4.2.2　モデルを用いた実効性能評価

複数の Solver を使ったシミュレーションのテストケースとして、2.2.2 節で挙げた minimal mitotic oscillator のモデル [23] のシミュレーションを行った。このモデルは 2.2.2 節の式 2.28〜2.34 で表される 7 つの反応から成るフィードバック機構が含まれ、適切な初期濃度および反応速度定数を与えると、4.4 に示すように反応経路中の cyclin (C)、cdc2 kinase (M)、cyclin kinase (X) の濃度が周期的に振動する。

これをシミュレーションするプログラムを C 言語で記述し、3.2GHz の Intel Pentium4 で実行 (gcc-3.4.2 -O3、FreeBSD 5.4) した結果、1 秒あたりにシミュレーション可能な時間刻みは 8.65×10^5 ステップであり、反応数に置き換えると 1 秒あたり 6.05×10^6 反応であった。

一方、90MHz で動作する FPGA 上に 4.1.2 節で面積評価の対象にしたものと同じ、3 つの Solver を用いた回路を構成し、シミュレーションを実行した。この場合には 1 時間刻みあたり 110 クロックを要し、1 秒あたりの時間刻み数は 8.18×10^5、反応数は 5.72×10^6 であった。これはソフトウェアに比べて 5.4% ほど遅い結果である。

このモデルで充分に性能が出ないのは、

1. モデルが小さく、パイプラインにデータを入力する時間よりも、パイプラインの処理完了待ちの時間のほうが長いこと、

2. 3 つの Solver 間で負荷の偏りがあり、すべての Solver とその演算器を有効に利用できていないこと

表 4.4　Phase 1、2 における各 Solver のパイプライン稼働率

Phase	Clock	Solver 0 Active Clk (%)		Solver 1 Active Clk (%)		Solver 2 Active Clk (%)		Average Active Clk (%)	
1	66	2	(3.0)	6	(9.0)	4	(6.1)	4	(6.1)
2	44	4	(9.1)	8	(18.2)	0	(0.0)	4	(9.1)
Total	110	6	(5.5)	14	(12.7)	4	(3.6)	8	(7.3)

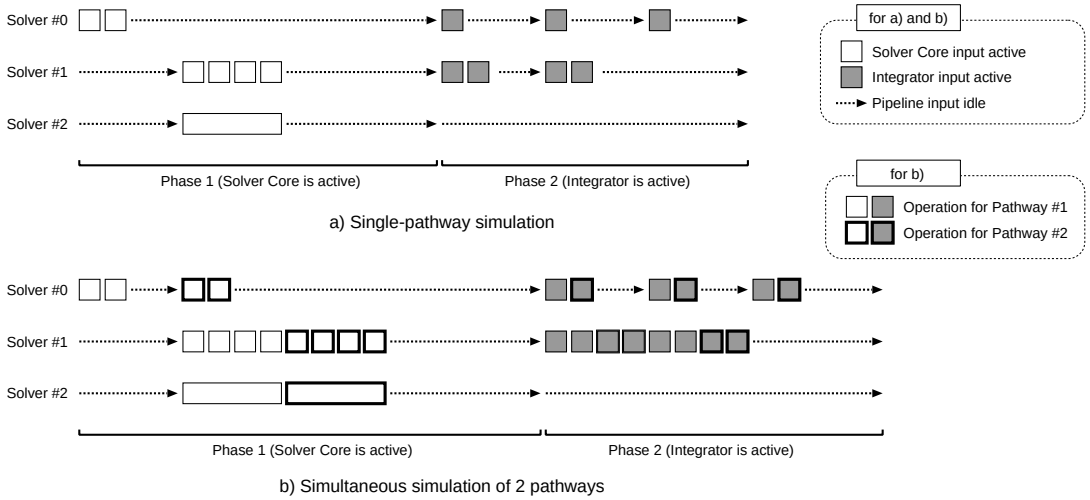


図 4.5　複数の Pathway の同時シミュレーション

の 2 点が原因として挙げられる。数値的にこの状況を表したのが表 4.4 である。この表から、1 時間刻みの処理に所要する 110 クロックのうち、パイプラインにデータが入力されているのは 3 つの Solver を平均すると 8 クロック、7.3% に過ぎないこと (上記の原因 1) と、パイプラインの稼働率がもっとも高い Solver 1 と、もっとも低い Solver 2 で 3.5 倍もの開きがあること (原因 2) がわかる。1 つめの原因に関しては、より大きな系をシミュレーションする場合にはパイプラインの稼働率が向上し、実効性能の向上が期待できるとともに、4.3.1 節で述べるように、モデル内部のパラメータを変化させながら多数のシミュレーションを行いその挙動を調べるような処理において高い性能を発揮できることが予想できる。2 つめの原因については、今回の例で用いているようなひとつの反応速度式専用の Solver Core でなく、複数の反応速度式に対応した Solver Core を用いることで、Solver の稼働率を均等化し、全体としての面積対性能比を向上することができる。

4.3　実効性能向上のためのアプローチ

4.3.1　パラメータ推定への応用

前節では、実際に小さなモデルを ReCSiP で走らせた場合の処理能力について示し、小さな系のシミュレーションを行った場合に、マイクロプロセッサによる処理能力を下回る性能しか発揮

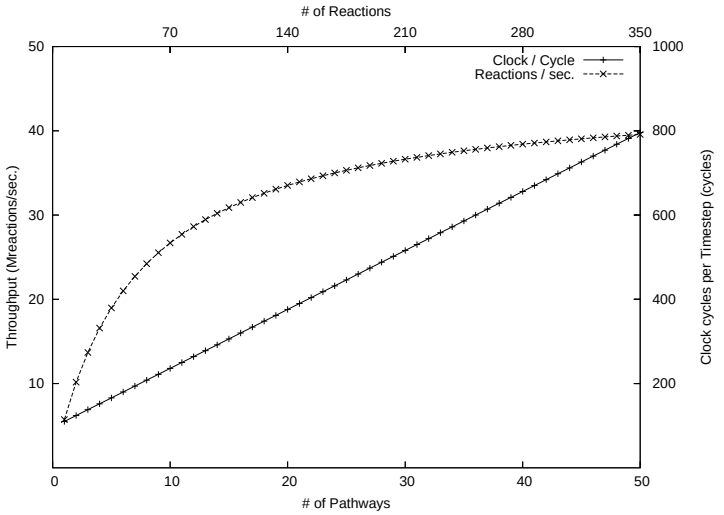


図 4.6　複数の Pathway の同時シミュレーション時のスループット

できない原因について述べた。本節では、パイプライン稼働率を向上することでより高い性能を発揮する ReCSiP の利用方法について述べ、その際の計算スループットを示す。

生化学モデルを構築する際に、実験から得られた結果とモデルの挙動の整合性を取るために、モデル内のパラメータをさまざまに変化させ、実験結果を再現できるようなパラメータセットを探索する必要がある。この作業は多次元のパラメータ空間をスキャンしながらシミュレーションを多数反復して行うことになるため、一般に長い計算時間が必要になる。

たとえば、図 4.7 は、minimal mitotic oscillator モデルにおいて、モデル中のふたつの反応 (R_4: 式 2.31 と R_5: 式 2.32) のパラメータを変更した場合のプロットで、中央の $V_{m1} = 0.5$、$V_{m3} = 0.2$ がもともとの反応速度定数である。V_{m1} は C の影響で M が増える反応の反応速度定数、V_{m3} は M の影響で X が増える反応の反応速度定数で、それぞれのパラメータが C や M が増加した際の M や X の応答速度に影響があることがわかる。このふたつのパラメータが変動した結果として、振動の周期や 3 つの物質の濃度の関係が影響を受けている。

このようにパラメータ推定を行う場合には、同じ反応経路をパラメータだけ変化させながら繰り返しシミュレーションすることになるが、パラメータを変えた複数の反応経路をひとつの大きな反応経路とみなしてシミュレーションすることで、パイプラインの稼働率を向上することができる。

たとえば、図 4.5 の (a) と (b) はそれぞれ、同じ反応経路を 1 つシミュレーションする場合と、2 つシミュレーションする場合の、パイプラインの入力を示している。それぞれ、パイプラインにデータを入力した後に、結果を得て次の処理に移行するためにパイプラインの長さだけの待ち時間が必要になるが、必要な待ち時間は基本的にパイプラインの長さによって決定されるため、反応経路の規模には依存しない。したがって、ひとつの経路だけを処理する (a) のケースよりも、ふたつの経路を処理する (b) のほうが全体の処理時間に占める待ち時間の割合が相対的に小さくなり、本来システムが持っているパイプライン処理の性能をより高く発揮できる。

図 4.6 は、複数の minimal mitotic oscillator の反応経路をひとつにまとめてシミュレーションした場合の、1 時間刻みあたりの所要クロック数と、1 秒間に計算される時間刻み数・反応数を表している。パイプラインの処理終了待ちに必要な待ち時間は一斉にシミュレーションする反応経路

の数にかかわらず一定であるため、全体としてのモデル規模が小さいうちはモデルの拡大に伴って急速にスループットが向上し、規模が大きくなると徐々に飽和する。50 個のシミュレーションタスクを同時に実行する場合の処理速度は 1 秒あたり 39.6×10^6 反応であり、ソフトウェア処理と比較すると約 6.5 倍の性能向上比となる。FPGA の面積を半分しか使っておらず、3 つの Solver のうち 1 つがほとんどの処理を行っていることを考慮すると、充分に良好な性能を発揮できているといえよう。

このように、深いパイプラインを構成して行うシミュレーションは、ある程度大きな系のシミュレーションにおいて力を発揮するが、複数のシミュレーションを一括して行うことで、小さな系の動態を調べるような場合にも同様に効果的であることがわかる。

4.3.2 Phase 1/2 の同時処理による性能向上手法

前節では、原状の積分モジュールを用いて実効性能を向上させる手法について述べたが、本節では積分モジュールの実装を一部変更することで、さらに性能を向上させる手法について考察する。

積分モジュールの動作は、反応速度式を解く Phase 1 と、積分動作を行って時間刻みを進める Phase 2 の 2 段階に分かれており、Phase 2 では Solver Core はまったく使われない。したがって、2 つのまったく独立したシミュレーションで、Phase 1 で用いる回路と Phase 2 で用いる回路を共用することにより、ハードウェア量の増加を小さく抑えながら、Solver Core の稼働率を向上させることができる。

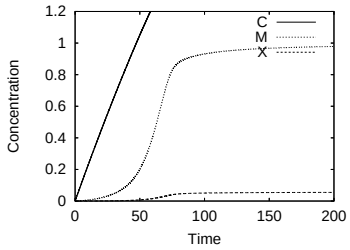
図 4.8 に、Euler 法に基づく積分モジュールにこの手法を適用した場合の図を示す。Phase 1 と Phase 2 を別の回路で処理するようにするとともに、ふたつの独立したシミュレーションタスクに、それぞれの [X] RAM と d[X] RAM を割り当て、一方がタスクが Phase 1 の回路を用いて処理をしている間にもう一方が Phase 2 の回路を用いて処理を実行する。

この手法で Euler 法の積分モジュールを実装し直した結果、追加分の演算器などを含めて 1,630 スライス増加 [82] した。Euler 法の積分モジュール単体としては倍以上の面積増であるが、これはふたつのシミュレーションタスクの動作を制御するための回路が必要となるためである。Euler 法では Phase 1、2 ともに加算器、乗算器を各 1 つ用いるため、この手法を用いるとほぼ倍の面積となるが、Phase 2 の計算がより複雑な Heun 法や Runge-Kutta 法でも、Phase 1 を独立したモジュールとするための拡張には Euler 法の場合と同様に加算器、乗算器各 1 つの追加で済むため、表 4.1 と比べると積分モジュール単体ではそれぞれ 69.4% と 38.1% 程度の面積増で済む。この手法は最大で 2 倍の処理能力が得られることになるため、Solver Core の面積まで考慮に入れると、きわめて良好な面積あたりの処理能力向上比を実現できる手法である。

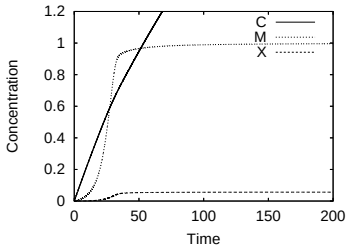
4.4 運用性・拡張性に関する議論

4.4.1 一般的なモデルへの適用

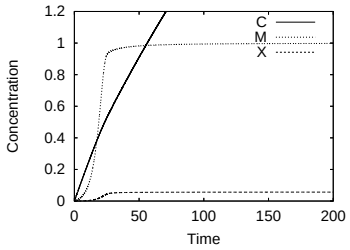
ReCSiP では、与えられたモデルに応じて Solver を組み合わせた回路を構成することで、限られた FPGA の面積内で最大限の並列性を実現し、これによって性能を得ている。しかし、FPGA の面積は有限であり、同時に実装することのできる Solver の数には制限がある。また、HDL の RTL 記述から FPGA の構成情報を得るためには論理合成・配置配線の処理を行う必要があり、これには通常の PC で CAD ツールを実行し、30 分から 1 時間程度を所要するため、一度構成した回



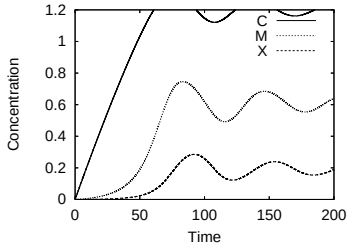
$V_{m1} = 0.1$, $V_{m3} = 0.04$



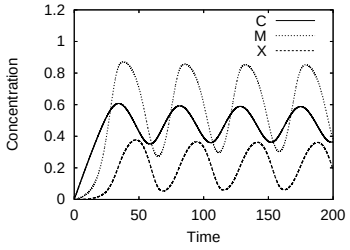
$V_{m1} = 0.5$, $V_{m3} = 0.04$



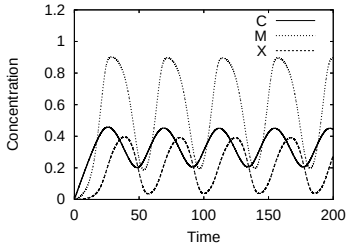
$V_{m1} = 0.9$, $V_{m3} = 0.04$



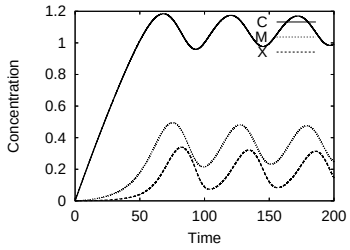
$V_{m1} = 0.1$, $V_{m3} = 0.12$



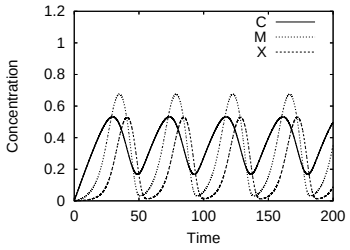
$V_{m1} = 0.5$, $V_{m3} = 0.12$



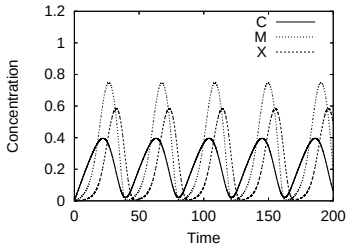
$V_{m1} = 0.9$, $V_{m3} = 0.12$



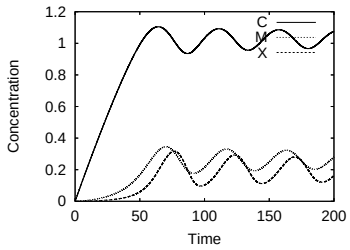
$V_{m1} = 0.1$, $V_{m3} = 0.2$



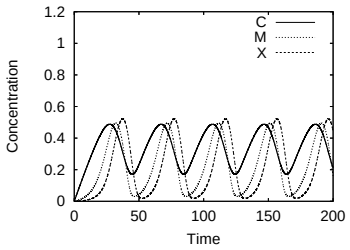
$V_{m1} = 0.5$, $V_{m3} = 0.2$



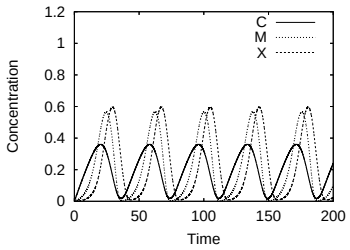
$V_{m1} = 0.9$, $V_{m3} = 0.2$



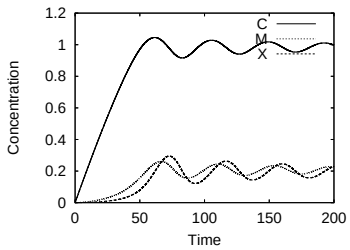
$V_{m1} = 0.1$, $V_{m3} = 0.28$



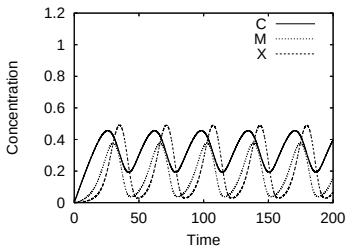
$V_{m1} = 0.5$, $V_{m3} = 0.28$



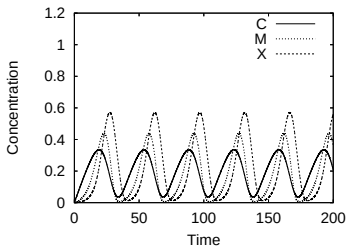
$V_{m1} = 0.9$, $V_{m3} = 0.28$



$V_{m1} = 0.1$, $V_{m3} = 0.36$



$V_{m1} = 0.5$, $V_{m3} = 0.36$



$V_{m1} = 0.9$, $V_{m3} = 0.36$

図 4.7　Minimal Mitotic Oscillator モデルでのパラメータ探索の例

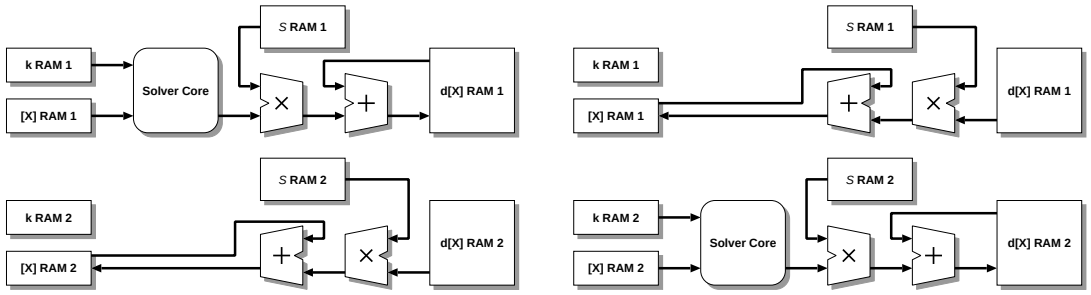


図 4.8 Phase 1 と Phase 2 の同時処理

路をなるべく再利用できることが望ましい。本節では、以上のことをふまえて、ヒト赤血球のシミュレーションモデルを例に挙げて、モデル中に含まれる反応速度式の種類やその割合などを検証する。

ヒト赤血球は比較的単純な構成の細胞として知られており、1989 年に Joshi らによって反応速度式を含む代謝系全体のモデル化が行われている [83][84][85]。Lee らはこれを元に計算機上でシミュレーションを行うためのソフトウェアパッケージを 1992 年に開発しており [86]、松嶋らは各種酵素異常の状態を E-Cell[18] によるシミュレーションで再現することに成功した [87]。

松嶋らによるモデルは 44 の反応で構成されており、ReCSiP でシミュレーションする際の Solver Core のライブラリ構成 [77] に従ってこれらの反応の種類をまとめたものが表 4.5 である。この表から、反応の多くを占めるのは mass action、ordered、ping-pong や Michaelis-Menten といった一般的な反応機構であり、モデル特有の反応機構はそれほど多くないことが伺える。この割合からモデルを構築する際の手法を考えると、基本的にはよく知られた反応機構を用いて各反応をモデル化してゆき、それでは表現できないところに特別に反応速度式を立てる、ということであり、これは一般的なことと考えられる。

このように一般的に広く用いられている反応速度式とそうでないものが混在する場合、一般的に用いられるものに関しては Solver Core が既にライブラリ化されているので、これを組み合わせて対応することができる。さらにこれらの多くは類似の反応速度式をサポートするため、多くのモデルに対しては mass action、ordered/ping-pong、Michaelis-Menten といった、3 つ程度の代表的な Solver Core の組み合わせで対処することができると考えられる。これは現在の FPGA の回路容量と、Solver Core や積分モジュールの回路規模を考慮すると充分に現実的である。

一方、モデル特有の反応機構については、その反応速度式専用の Solver Core を用いる方法と、いくつかの浮動小数点演算器やレジスタに Pathway RAM のようなものを付加して、ソフトウェア的に解く方法が考えられる。前者は現在までの Solver Core ライブラリの構築で得られた知見を生かして自動化することも可能であり、面積や速度のトレードオフを考慮しながら最適な設計を選択できるというメリットがある。後者については現在検討がなされているが、ソフトウェア的に関数を実現できるため、これを mass action、ordered/ping-pong、Michaelis-Menten などの広く使われる Solver Core と組み合わせることで、同一の Solver の組み合わせで多くのモデルのシミュレーションを実現できると考えられる。

モデル特有の反応速度式は、松嶋らのヒト赤血球モデルで 7 種類 8 反応が用いられているように、少ない反応数ながらもそれぞれが別々の反応速度式になることが一般的であると考えられる。処理する反応の数が他に比べて極端に少ない Solver が存在する場合、平均した面積・時間あたり

表 4.5　ヒト赤血球の代謝モデルを構成する反応の Solver への割り当て例

Solver	Reaction Mechanism	# of Reactions
Mass Action	Mass Action	16
Ordered/Ping-Pong	Ordered Uni-Bi	1
	Ordered Bi-Bi	1
	Ordered Bi-Ter	2
	Ping-Pong Bi-Bi	3
Michaelis-Menten	Michaelis-Menten	9
	Uni-Uni	4
Reaction Specific	Reaction Specific (7 types)	8
Total		44

の処理能力が低下するため、モデル特有の反応機構については可能な限り少数の Solver で処理するのがよいと考えられるが、これを実際にどのようなハードウェア構成で、どのようにスケジュールして解くかは今後の検討課題のひとつである。

4.4.2　スイッチの分散化によるシステム規模の拡大

FPGA の回路容量に比例して、単一チップ上に実装可能な Solver の個数は多くなるが、Solver の数、即ち必要なポート数は FPGA の容量に比例して増えるのに対して、クロスバの回路規模はポート数の自乗に比例して拡大する。したがって、Solver の回路規模拡大よりもスイッチの回路規模拡大の方が速く、FPGA の回路容量が大きくなるにつれて演算器のための回路リソースが圧迫される。これを解決するには、小規模のスイッチを相互接続することで多くの Solver を接続する方式が考えられる。スイッチにデータが投入される際の宛先指定は Transceiver によって行われ、Solver 側のスイッチインタフェイスはメモリであるため、dualport 構成のメモリを通信用のバッファとして挟むことにより、スイッチ間を相互接続することができる (図 4.9)。この場合にも、演算と通信のスケジュールはシミュレーション開始前に行われるため、メモリを挟んだふたつのスイッチがそれぞれ、あらかじめ決められたスケジュールに従ってデータの読み書きを行うことで通信を行うことができる。

たとえば表 4.2 にあるとおり、8 ポートのスイッチは 2,570 スライスだが、図 4.10 のように、4 ポートのスイッチ (542 スライス) を 3 つ用いて 8 ポートのスイッチとすると 1,626 スライスとなり、37% の面積減となる。ただし、中央のスイッチにトラフィックが集中する可能性があり、大きな反応経路を多数の Solver に割り当ててシミュレーションする場合には事前のトラフィック解析と、それに基づくスイッチ間の接続トポロジの決定が必要になる。

また、クロスバの代わりに、全結合でない小面積なスイッチに高機能なルーティング機構を付加したルータを用いて FPGA 上のモジュール間を接続する研究 [88] も行われており、演算モジュール間の通信機構に関しては今後も大いに研究の余地が残されている。

これらの手法は複数の FPGA 上に Solver を構成し、それぞれの FPGA 内にスイッチを設置する方法にも適用可能である。ReCSiP では、FPGA 内のロジックを駆動するクロックは 90〜100MHz 前後であるため、スイッチのポートをそのままチップ外の配線に引き出すことも可能である。し

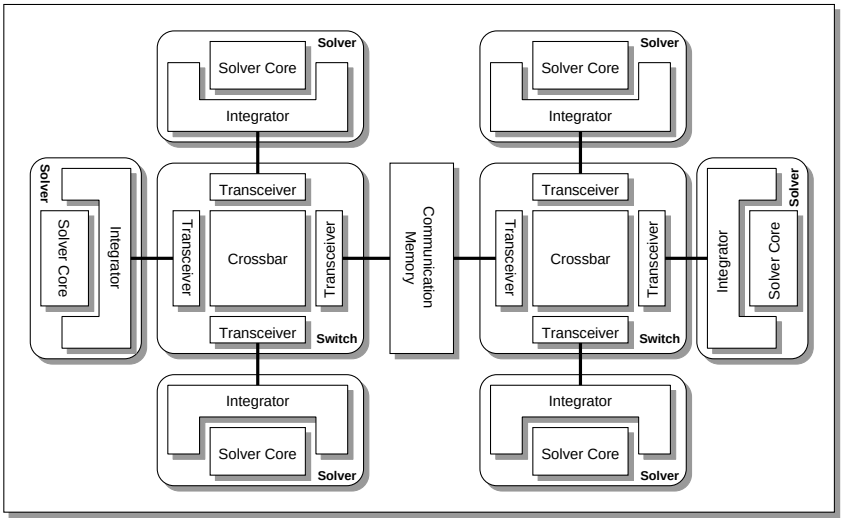


図 4.9　Dualport メモリによるスイッチ間の接続

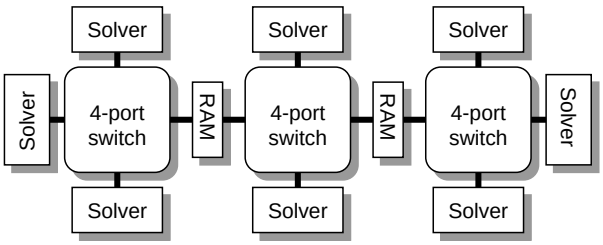


図 4.10　スイッチの組み合わせ例

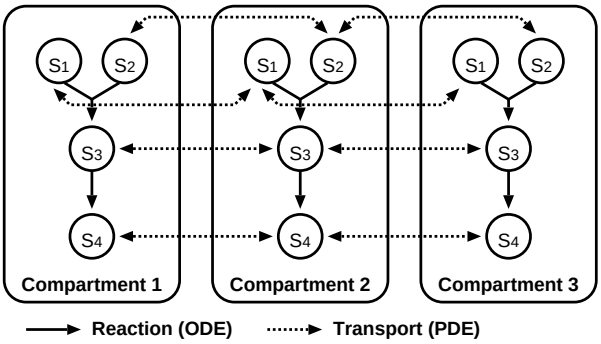


図 4.11　区画内の反応と区画間の移動

たがって、同一の基板上に複数の FPGA を実装する場合には、一方の FPGA が前節で述べた通信用の共有メモリのポートを、他方の FPGA がスイッチのポートを、それぞれボード上の配線に接続することで、異なる FPGA にまたがる構成を実現でき、より多くの Solver による並列処理を行える。

4.5　将来的な可能性に関する議論

4.5.1　複数の区画で構成されるモデルのシミュレーション

これまでで述べてきたようなシミュレーション方式では、対象の反応系が一様に混じり合っていることが前提となるが、実際には細胞は細胞膜で囲まれて隣接する細胞と区切られており、さらには細胞内にも膜で区切られたさまざまな小器官が存在する。したがって、細胞質と核など、膜などで区切られた区画をそれぞれ別のものとして、それぞれの内部で起こる化学反応の速度を計算し、同時に区画間での物質の移動速度を計算することで、より現実的なモデリングとシミュレーションを行うことができる。このような場合、それぞれの区画内では連立常微分方程式によって反応速度を求め、隣接する区画間では濃度勾配や電位勾配を用いた偏微分方程式によって物質の移動速度を求めることができる。

さらに、細胞質のように同一の区画であっても物質の局在を考慮したシミュレーションを行う場合もある。たとえば、The Virtual Cell[19][20] では Finite Volume Method[89] を用いて、2 次元あるいは 3 次元の広がりを持つモデルの取り扱いを実現している。これは、細胞全体を格子状の区画に区切り、核や細胞内小器官を形成する膜を格子に沿わせることで、立体構造を考慮したシミュレーションを実現するものである。Fink らはこれを用いて神経細胞が信号を伝達する際のカルシウムイオンの挙動を再現することに成功している [90]。

このように複数の区画にわかれたシミュレーションを行う場合には、図 4.11 のように処理を行うことになる。異なる区画に存在する同じ分子種は違う分子種としてみなすことで、区画間の物質輸送も反応として取り扱うことができ、全体をひとつの大きな区画として処理することができる。

ただしこの種の問題は、区画数に比例して反応数と必要メモリ量が増大する。細胞質と核のように、数個程度の区画に分けたシミュレーションであれば数倍程度のメモリ消費量の増大で収まるが、The Virtual Cell のように、細かな格子で分ける場合には解像度の 3 乗に比例して必要メモリ量が増大することになるので、FPGA 外部のメモリを活用する実装方法を検討する必要が生じ

る。必要メモリ量と同様に計算量も増大するが、これは前節で述べたような複数の FPGA を用いる方法によって計算負荷を分散することで解決を図ることができる。

第5章　結論

本研究では、FPGA を用いて、微分方程式によってモデリングされた生化学反応経路のシミュレーションを高速化する方式を開発した。これによって、数理モデルを用いて生化学システムのダイナミクスを調べる際に必要となる計算時間の短縮に貢献することができる。

FPGA を用いたシステムを通常の PC に付加して利用する場合、PCI バスなどに接続することになるが、この場合にはホスト PC 側のプロセッサとアクセラレータとなる FPGA の間の通信ボトルネックが問題となる。本研究ではこれを克服するため、反応経路を表現するポインタの配列を用いて FPGA 上の数値積分モジュールが自立して一連の反応を処理し、ホスト側からの指示がなくても途中経過を経時的に出力する方式を開発した。この方式により、メモリ上にポインタ配列を展開することで柔軟に反応経路を記述することができる。また、ポインタ配列で反応経路を表現することにより、反応経路が拡大した場合にも回路面積が増加することなく、FPGA の回路容量の許す範囲で処理することを可能にすることに成功した。

シミュレーションを行うための回路はモジュール化されており、数値積分を行うためのモジュールである Integrator ひとつに反応速度式を計算するための Solver Core と呼ばれるモジュールをひとつ接続して Solver を構成し、これを基本単位としてシミュレータを構成する。それぞれの Solver Core はひとつまたは複数の反応速度式に対応することができ、モデルの含む反応速度式をすべてカバーする Solver Core のセットを用いればシミュレーションを実行することができる。FPGA の面積が余っている場合には、負荷の高い Solver Core を含む Solver を複数構成して並列に処理を実行することで、処理を高速化できる。

また、FPGA 上の回路を構成する際には、HDL で記述されたモジュールを組み合わせるだけで与えられたモデルに最適な回路を生成することが可能なため、SBML などのモデル記述言語から自動的に回路を生成してシミュレーションを行うソフトウェアを構築することもでき、これによって FPGA による高速なシミュレータを容易に利用することができる。このように、モジュール化された Solver を組み合わせて利用することで、柔軟性と高速性の双方を実現できた。

性能評価の結果、Xilinx 社の FPGA である Virtex-II Pro (XC2VP70) を用いて、FPGA 上に構成した演算器がすべて連続して動作可能であれば、各種の積分アルゴリズムにおいて 3.2GHz 動作の Pentium4 の 80 倍程度の性能を発揮することが可能であることがわかった。実際のモデル (Minimal Mitotic Oscillator) をシミュレーションする場合には Solver 間での負荷の差異や、モデルが小さいことに起因するパイプライン稼働率の低さなどから、ひとつの反応系のシミュレーションではマイクロプロセッサに性能面で劣る場合もあるものの、パラメータを変えた複数のモデルのシミュレーションを一斉に実行することにより、実効スループットが 6.5 倍程度改善することが確認された。これは、モデルの作成やチューニングの過程におけるパラメータサーベイのための計算手段として FPGA を用いたアクセラレータが有効であることを示している。また、ふたつのシミュレーションタスクを交互に実行する構成により、実効スループットを最大で 2 倍にする手法も提案し、すでに共同研究者により結果を確認済みである [82]。

現在の実装の枠組みの中でより高い性能を発揮するためには、本論文で示したようにパイプライン稼働率を改善するほかにも、Solver 間の負荷の差異を小さくすることが実効性能向上の鍵となることも挙げた。今後は複数の反応速度式を解くことのできるモジュールの整備や、与えられた関数群を解くモジュールを自動生成するソフトウェアの開発などが必要であると考えられる。

さらなる性能向上を目指すためには、シミュレータの回路構成そのものを改良していくことになるが、これには実際にさまざまなモデルをシミュレーションし、各 Solver やスイッチの負荷状況、各モジュール内のメモリの消費量などについて統計的に評価を行っていくことが必要である。このためには、共同研究者によって現在開発中の、モデルから回路や回路内のメモリに書き込むデータを生成するためのソフトウェアの開発が必須である。

本研究において開発された手法は、多数の浮動小数点演算器を接続して深いパイプラインを構成し、静的なスケジューリングによってパイプラインの乱れを最低限にとどめることで、高い性能を発揮している。また、FPGA 内のメモリブロックを利用し、複数のメモリブロックに並列アクセスを行うことで高いデータ転送バンド幅を実現し、パイプラインに効率的にデータを供給する機構を実現している。

この手法を用いることで FPGA 単体でマイクロプロセッサの数倍から数十倍の性能を発揮できることが確認され、FPGA などの reconfigurable device を計算エンジンとして搭載したデスクトップの計算システムが小規模な PC クラスタなどの並列システムを置き換える可能性があることが示された。FPGA は既に、価格や消費電力あたりの性能が高いことで、次世代の計算システムの中核を担うデバイスとして注目されているが、本研究では多くの浮動小数点演算を含む数値計算を高速に実現したことで、その可能性の一端を実証することができたといえよう。

謝辞

本研究の機会を与えてくださり、6 年間にわたって絶えずご指導いただいた慶應義塾大学理工学部情報工学科 天野 英晴教授に深く感謝いたします。また、お忙しい中草稿の段階から査読をしてくださり、多くの有益アドバイスをいただいた、副査の慶應義塾大学理工学部生命情報学科 岡浩太郎教授、同情報工学科 山本 喜一助教授、斎藤 博昭助教授に深く感謝いたします。

本研究の一部は平成 13 年度から 14 年度までの、農林水産省「イネゲノム・シミュレータプロジェクト (SY-1106)」、および平成 13 年度から 17 年度までの文部科学省科学技術振興調整費「システム生物学者育成プログラム」によるものです。また、FPGA 上の回路の設計にあたっては東京大学大規模集積システム設計教育研究センターを通じ、Cadence 社および Synopsys 社の協力で提供されている設計ソフトウェアを利用させていただきました。

天野研究室 Bio グループの立ち上げ時にご協力くださった安生 健一朗博士 (現在 NEC エレクトロニクス株式会社)、鈴木 紀章博士 (現在日本電気株式会社)、田村友紀氏 (現在ローム株式会社) と、シミュレータの開発にご協力くださった天野研究室 Bio グループの福島 知紀氏 (現在東芝セミコンダクター社)、吉見 真聡氏、岩岡 洋氏、小嶋 利紀氏、西川由理氏に深く感謝いたします。特に、鈴木 紀章博士と吉見 真聡氏には、本研究に不可欠な浮動小数点演算器のライブラリを提供していただきました。また、ReCSiP Board の開発とデバッグには天野研究室 Vision グループの福田 静人氏 (現在 Sony Computer Entertainment Inc.)、金森飛匡氏に多くのご協力をいただきました。

科学技術振興機構 ERATO-SORST 北野共生システムプロジェクトの北野 宏明博士には、プロジェクト立ち上げの際からさまざまな面でご支援をいただきました。また、同プロジェクトの研究員であられる舟橋 啓博士、広井 賀子博士と、長崎大学工学部情報システム工学科の柴田 裕一郎講師、岩永 直樹氏には、共同研究者として多くのご助力をいただきました。

生化学シミュレータに関する研究をスタートする際には慶應義塾大学環境情報学部冨田研究室 E-Cell Project の高橋 恒一博士、Wang Emily 氏に多くのアドバイスをいただきました。また、ReCSiP Board の開発にあたっては、三精システム株式会社 藤代 行康氏、森岡 聡氏、及部 晴康氏、深谷 一義氏と、新和電材株式会社 阿部 伸一郎氏にさまざまな面でご協力をいただきました。

研究費に関する諸事務では、慶應義塾大学理工学部研究支援センターの片桐 素美氏、同学部生命情報学科岡研究室秘書の中野 千秋氏、天野研究室秘書の高橋 朋絵氏、岡部 明子氏、高見 康子氏、安西研究室秘書の永坂 弘子氏、ERATO-SORST 北野共生システムプロジェクトの藤井 昭康氏など、数多くの方々に大変お世話になりました。深く御礼申し上げます。

最後になりましたが、本研究や論文執筆をさまざまな面から支えてくださいました、天野研究室、安西・今井研究室、山崎研究室の皆様、ERATO-SORST 北野共生システムプロジェクトの皆様と、すべての友人、家族に感謝いたします。

2005 年 2 月 矢上キャンパス 26 棟 112B 号室にて

参考文献

[1] Peter M. Athanas and Harvey F. Silverman. Processor reconfiguration through instruction-set metamorphosis. *Computer*, Vol. 26, No. 3, pp. 11–18, Mar. 1993.

[2] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, and S. Ghosh. Prism-i1 compiler and architecture. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 9–16, Apr. 1993.

[3] Maya Gokhale, William Holmes, Andrew Kopser, Sara Lucas, Ronald Minnich, and Douglas Sweely. Building and using a highly parallel programmable logic array. *Computer*, Vol. 4, No. 1, pp. 81–89, Jan. 1991.

[4] 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.

[5] Xilinx Inc. *Virtex-II Platform FPGA User Guide*, v2.0 edition, Mar. 2005.

[6] Xilinx Inc. *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*, v4.0 edition, Mar. 2005.

[7] Xilinx Inc. *Virtex-4 User Guide*, v1.4 edition, Dec. 2005.

[8] Altera Inc. *Stratix Device Handbook*, v3.3 edition, Jul. 2005.

[9] Altera Inc. *StratixII Device Handbook*, v4.0 edition, Dec. 2005.

[10] Naohito Nakasato and Tsuyoshi Hamada. Astrophysical hydrodynamics simulations on a reconfigurable system. In *Proceedings of the 13th IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 279–280, Apr. 2005.

[11] James Watson and Francis Crick. Molecular structure of nucleic acids: a structure for deoxiribose nucleic acid. *Nature*, Vol. 171, pp. 737–738, 1953.

[12] Tomoyoshi Soga, Yuki Ueno, Hisako Naraoka, Yoshiaki Ohashi, Masaru Tomita, and Takaaki Nishioka. Simultaneous determination of anionic intermediates for *bacillus subtilis* metabolic pathways by capillary electrophoresis electrospray ionization mass spectrometry. *Analytical Chemistry*, Vol. 74, pp. 2233–2239, Apr. 2002.

[13] Tomoyoshi Soga, Yuki Ueno, Hisako Naraoka, Keiko matsuda, Masaru Tomita, and Takaaki Nishioka. Pressure-assisted capillary electrophoresis electrospray ionization mass spectrometry for analysis of multivalent anions. *Analytical Chemistry*, Vol. 74, No. 24, pp. 6224–6229, Dec. 2002.

[14] 北野宏明. システムバイオロジー —生命をシステムとして理解する—. 秀潤社, Jul. 2001.

[15] Hideo Matsuda. Development of bio-information environment of the grid. In *Grobus World*, Jan. 2003.

[16] Bruce A. Barshop, Richard F. Wrenn, , and Carl Frieden. 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.

[17] Mendes Pedro. 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.

[18] Masaru Tomita, Kenta Hashimoto, Kouichi Takahashi, Thomas Simon Shimizu, Yuri Matsuzaki, Fumihiko Miyoshi, Kanako Saito, Sakura Tanida, Katsuyuki Yugi, J. Craig Venter, and Clyde A. Hutchison III. E-cell: software environment for whole-cell simulation. *Bioinformatics*, Vol. 15, No. 1, pp. 72–84, Jan. 1999.

[19] James Schaff, Charles C. Fink, Boris Slepchenko, John H. Carson, and Leslie M. Loew. A general computational framework for modeling cellular struc ture and function. *Biophysical Journal*, Vol. 73, pp. 1135–1146, Sep. 1997.

[20] James Schaff and Leslie M. Loew. The virtual cell. In *Proceedings of Pacific Symposium on Biocomputing*, Vol. 4, pp. 228–239, Jan. 1999.

[21] Yasunori Osana, Tomonori Fukushima, Masato Yoshimi, and Hideharu Amano. An FPGA-based acceleration method for metabolic simulation. *IEICE Transactions on Information and Systems*, Vol. E87-D, No. 8, pp. 2029–2037, Aug. 2004.

[22] 長名保範, 吉見真聡, 岩岡洋, 小嶋利紀, 西川由理, 舟橋啓, 広井賀子, 柴田裕一郎, 岩永直樹, 北野宏明, 天野英晴. FPGA を用いた汎用生化学シミュレータ ReCSiP (to appear). 電子情報通信学会論文誌 D, Jun. 2005.

[23] Albert Goldbeter. A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. In *Proceedings of the National Academy of Sciences*, Vol. 88, pp. 9107–9111, Oct. 1991.

[24] Hiroaki Kitano. Prespectives on systems biology. *New Generation Computing*, Vol. 18, No. 1, pp. 199–216, 2000.

[25] David Fell. *Understanding the Control of Metabolism*. Portland Press, 1997.

[26] L.Michaelis and Maud L.Menten. Die kinetik der invertinwirkung. *Biochemische Zeitschrift*, Vol. 49, pp. 333–369, 1913.

[27] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, Vol. 22, pp. 403–434, 1976.

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

[29] Daniel T. Gillespie. Approximate acclerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics*, Vol. 115, No. 4, pp. 1717–1733, Jul. 2001.

[30] Michael Hucka, Andrew Finney, Benjamin Bornstein, Sarah Keating, Bruce Shapiro, Joanne Matthews, Ben Kovitz, Maria Schilstra, Akira Funahashi, John Doyle, and Hiroaki Kitano. Evolving a lingua franca and associated software infrastructure for computational systems biology: The systems biology markup language (sbml) project. *IEE Systems Biology*, Vol. 1, No. 1, pp. 41–53, 2004.

[31] Catherine M. Lloyd, Matt D.B. Halstead, and Poul F. Nielsen. CellML: its future, present and past. *Progress in Biophysics and Molecular Biology*, Vol. 85, pp. 433–450, Jun 2004.

[32] Akira Funahashi, Naoki Tanimura, Mineo Morohashi, and Hiroaki Kitano. CellDesigner: a process diagram editor for gene-regulatory and biochemical networks. *BIOSILICO*, Vol. 1, No. 5, pp. 159–162, Nov. 2003.

[33] The Systems Biology Institute. Celldesigner.org. `http://celldesigner.org/`.

[34] Computational Systems Biology Group at Keck Graduate Institute. JDesigner. `http://sbw.kgi.edu/`.

[35] InNetics AB. Pathwaylab. `http://innetics.com/`.

[36] H.M.Sauro. Jarnac: a system for interactive metabolic analysis. In *The 9th International Bio-ThermoKinetics Meeting*, pp. 221–228. Stellenbosch University Press, 2000.

[37] Copasi. `http://www.copasi.org/`.

[38] Bruce E. Shapiro, Michael Hucka, Andrew Finney, and John Doyle. MathSBML: a package for manipulating SBML-based biological models. *Bioinformatics*, Vol. 20, No. 16, pp. 2829–2831, Apr. 2004.

[39] H. Sauro, M. Hucka, A. Finny, C. Wellock, H. Bolouri, J. Doyle, and H. Kitano. Next generation simulation tools: The systems biology workbench and biospice integration. *Omics*, Vol. 7, No. 4, pp. 355–572, Dec. 2003.

[40] KEGG: Kyoto encyclopedia of genes and genomes. `http://www.genome.ad.jp/kegg/`.

[41] M.Kanehisa, S.Goto, S.Kawashima, and A.Nakaya. The KEGG databases at genomenet. *Nucleic Acids Research*, Vol. 30, No. 1, pp. 42–46, Jan. 2002.

[42] Biomodels. `http://www.biomodels.net/`.

[43] Akiya Jouraku, Akira Funahashi, and Hiroaki Kitano. KEGG2SBML. `http://www.sbml.org/kegg2sbml.html`.

[44] Nicolas Le Novère, Andrew Finney, Michael Hucka, Upinder S Bhalla, Fabien Campagne, Julio Collado-Vides, Edmund J Crampin, Matt Halstead, Edda Klipp, Pedro Mendes, Poul Nielsen, Herbert Sauro, Bruce Shapiro, Jacky L Snoep, Hugh D Spence, and Barry L Wanner. Minimum

information requested in the annotation of biochemical models (MIRIAM). *Nature Biotechnology*, Vol. 23, pp. 1509–1515, Dec. 2005.

[45] Wormbase. http://www.wormbase.org/.

[46] Peter D. Karp, Monica Riley, Milton Saier, Ian T. Paulsen, Julio Collado-Vides, Suzanne M. Paley, Alida Pellegrini-Toole, César Bonavides, and Socorro Gama-Castro. The ecocyc database. *Nucleic Acids Research*, Vol. 30, No. 1, pp. 56–58, Jan. 2002.

[47] Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic. *File-Programmable Gate Arrays*. Kluwer Academic Publishers, 1992.

[48] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-submicron FPGAs*. Kluwer Academic Publishers, 1999.

[49] Atsushi Kawai, Toshiyuki Fukushige, Jun'ichiro Makino, and Makoto Taiji. GRAPE-5: A special-purpose computer for n-body simulations. *Publications of the Astronomical Society of Japan*, Vol. 52, pp. 659–676, Aug. 2000.

[50] Jürgen Becker and Reiner Hartenstein. Configware and morphware going mainstream. *Journal of System Architecture*, Vol. 49, No. 4-6, pp. 127–142, 2003.

[51] 浅見廣愛, 飯田全広, 中島克人, 森伯郎. FPGA ベース並列マシン RASH での DES 暗号解析処理の改良. 情報処理学会論文誌, Vol. 41, No. SIG 5 (HPS 1), pp. 50–57, 2000.

[52] Hiroai Asami, Masaharu Mizuno, Katsuto Nakajima, Masahiro Iida, and Hakuro Mori. Application of SAR image reconstruction processing on an FPGA-based parallel machine "RASH". In *Proceedings of Applied Informatics 2002 (AI2002)*, pp. 65–70, 2002.

[53] SGI Inc. White paper: Extraordinary acceleration of workflows with reconfigurable application-specific computing from sgi. http://www.sgi.com/pdfs/3721.pdf, Nov. 2004.

[54] SGI Inc. SGI Altix family. http://www.sgi.com/products/servers/altix/.

[55] Cray Inc. Cray XD-1 supercomputer. http://www.cray.com/products/xd1/index.html.

[56] Tsuyoshi Hamada and Naohiro Nakasato. PGR: A software package for reconfigurable super-computing. In *Proceedings of the 15th International Conference on Field Programmable Logic and Applications*, pp. 366–373, Aug. 2005.

[57] SRC Computers Inc. SRC-7 reconfigurable general purpose computing system. http://www.srccomp.com/.

[58] Tsuyoshi Hamada, Naohito Nakasato, and Toshikazu Ebisuzaki. A 236 Gflops astrophysical simulation on a reconfigurable super-computer. In *Proceedings of IEEE/ACM SC 2005 Conference*, Nov. 2005.

[59] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, Vol. 147, pp. 195–197, 1981.

[60] R. Durbin, S. Eddy, A. Korgh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[61] Annapolis Micro Systems Inc. Wildfire. `http://www.annapmicro.com/`.

[62] C.W.Yu, K.H.Kwong, K.H.Lee, and P.H.W. Leong. A Smith-Waterman systolic cell. In *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications*, Vol. 2778 of *Lecture notes in Computer Science*, pp. 375–384. Springer Verlag, Sep. 2003.

[63] Stefan Dydel and Piotr Bała. Large scale protein sequence alignment using FPGA reprogrammablelogic devices. In *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications*, Vol. 3203 of *Lecture notes in Computer Science*, pp. 23–32. Springer Verlag, Aug. 2004.

[64] Chen Chang, John Wawrzynek, and Robert W. Brodersen. BEE2: a high-end reconfigurable computing system. *Design & Test of Computers*, Vol. 22, No. 2, pp. 114–125, Mar. 2005.

[65] Yoshiki Yamaguchi, Tsutomu Maruyama, and Akihiko Konagawa. Three-dimensional dynamic programming for homology search. In *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications*, Vol. 3203 of *Lecture notes in Computer Science*, pp. 505–515. Springer Verlag, Aug. 2004.

[66] Yuto Komeiji, Hiroshi Yokoyama, Masami Uebayashi, Makoto Taiji, Toshiyuki Fukushige, Daiichiro Sugimoto, Ryo Takata, Akihiro Shimizu, and Keiji Itsukashi. A high performance system for molecular dynamics simulation of biomolecules using a special-purpose computer. In *Proceedings of Pacific Symposium on Biocomputing*, pp. 472–487, Jan 1996.

[67] Yongfeng Gu, Tom Van Court, and Martin Herbordt. Accelerating molecular dynamics simulations with reconfigurable circuits. In *Proceedings of the 15th International Conference on Field-Programmable Logic and Applications*, pp. 475–480, Aug. 2005.

[68] Shuichi Onami, Shugo Hamahashi, Masao Nagasaki, Satoru Miyano, and Hiroaki Kitano. chapter Automatic acquisition of cell lineage through 4D microscope and analysis of early *C. elegans* embryogenesis, pp. 33–55. Foundations of Systems Biology. MIT Press, 2001.

[69] 福島知紀. FPGA による線虫 *C.elegans* 初期胚の細胞系譜自動構築システムの高速化. 修士論文, 慶應義塾大学大学院理工学研究科, 神奈川県横浜市, Mar. 2005.

[70] Kanae Oda, Yukiko Matsuoka, Akira Funahashi, and Hiroaki Kitano. A comprehensive pathway map of epidermal growth factor receptor signaling. *Molecular Systems Biology*, Vol. msb4100014, , May. 2005.

[71] Kanae Oda, Tomomi Kimura, Yukiko Matsuoka, Akira Funahashi, Masaaki Muramatsu, and Hiroaki Kitano. Molecular interaction map of a macrophage. *AfCS Research Reports*, Vol. 2, No. 4, Aug. 2004.

[72] 吉見真聡, 長名保範, 岩岡洋, 柴田裕一郎, 岩永直樹, 天野英晴. FPGA を用いた科学技術計算向け浮動小数点算術演算器の 設計. 第 26 回研究会 1-7, パルテノン研究会, May. 2005.

[73] Xilinx Inc. *Floating-Point Operator v1.0 Product Specification*, 2005.

[74] Nallatech Inc. *Single Precision Floating Point Cores Product Brief*, 2002.

[75] Nallatech Inc. *Double Precision Floating Point Cores Product Brief*, 2004.

[76] QinetiQ Inc. *Quixilica Floating Point FPGA Cores*, 2004.

[77] Naoki Iwanaga, Yuichiro Shibata, Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Tomonori Fukushima, Hideharu Amano, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano, and Kiyoshi Oguri. Efficient Scheculing of Rate Law Functions for ODE-based Multimodel Biochemical Simulation on an FPGA. In *Proceedings of the 15th Field Programmable Logic and its applications(FPL) 2005*, pp. 666–669, Tampere, Finland, August 2005. IEEE.

[78] 小嶋利紀. ReCSiP におけるホストインターフェイスの実装と評価. 卒業論文, 慶應義塾大学理工学部情報工学科, 神奈川県横浜市, Mar. 2006.

[79] QuickLogic Inc. *QuickPCI QL5064 User Manual*, rev.e edition, Oct. 2004.

[80] 岩岡洋, 長名保範, 吉見真聡, 小嶋利紀, 西川由理, 舟橋啓, 広井賀子, 柴田裕一郎, 岩永直樹, 北野宏明, 天野英晴. FPGA を用いた生化学シミュレータ向け SBML 処理系の構築. RECONF2005 40, 電子情報通信学会, Sep. 2005.

[81] Michael Hucka, Andrew Finney, Herbert Sauro, and Hamid Bolouri. *Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Basic Model Definitions*. Systems Biology Workbench Development Group, ERATO Kitano Symbiotic Systems Project, MC 107-81 California Institute of Technology, Pasadena, CA 91125, USA, version 2 edition, Aug. 2003.

[82] 西川由理, 長名保範, 吉見真聡, 岩岡洋, 小嶋利紀, 舟橋啓, 広井賀子, 柴田裕一郎, 岩永直樹, 北野宏明, 天野英晴. FPGA を用いた生化学シミュレータ ReCSiP における数値積分機構の性能改善手法. VLD2005 106, IEICE, Jan. 2006.

[83] Abhay Joshi and Verbhard O. Palsson. Metabolic dynamics in the human red cell. part i – a comprehensive kinetic model. *Journal of Theoretical Biology*, Vol. 141, pp. 515–528, 1989.

[84] Abhay Joshi and Verbhard O. Palsson. Metabolic dynamics in the human red cell. part ii – interactions with the environment. *Journal of Theoretical Biology*, Vol. 141, pp. 529–545, 1989.

[85] Abhay Joshi and Verbhard O. Palsson. Metabolic dynamics in the human red cell. part iii – metabolic reaction rates. *Journal of Theoretical Biology*, Vol. 142, pp. 41–68, 1990.

[86] I-Der lee and Bernhard O. Palsson. A macintosh software package for simulation of human red blood cell metabolism. *Computer Methods and Programs in Biomedicine*, Vol. 38, pp. 195–226, 1992.

[87] 松嶋亮. E-cell システムによるヒト赤血球細胞のシミュレーションと各種病態解析. 修士論文, 慶應義塾大学大学院政策メディア研究科, 1999.

[88] 吉見真聡, 長名保範, 岩岡洋, 西川由理, 小嶋利紀, 舟橋啓, 広井賀子, 柴田裕一郎, 岩永直樹, 北野宏明, 天野英晴. データ転送網を用いた確率モデル生化学シミュレータの FPGA への実装の検討. VLD2005 105, 電子情報通信学会, Jan. 2006.

[89] Patankar. *Numerical Solution of Heat Transfer and Fluid Flow*. Taylor and Francis, 1980.

[90] Charles C. Fink, Boris Slepchenko, Ion I. Moraru, James Watras, James C. Schaff, and Leslie M. Loew. An image-based model of calcium waves in differentiated neuroblastoma cells. *Biophysical Journal*, Vol. 79, No. 1, pp. 163–183, Jul. 2000.

論文目録

本研究に関する論文

公刊論文

1. 長名 保範, 吉見 真聡, 岩岡 洋, 小嶋 利紀, 西川 由理, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “FPGA を用いた汎用生化学シミュレータ ReCSiP (to appear).”, 電子情報通信学会論文誌 D. Jun. 2006.

2. Yasunori Osana, Tomonori Fukushima, Masato Yoshimi, and Hideharu Amano: “An FPGA-Based Acceleration Method for Metabolic Simulation.”, IEICE Trans. on Information and Systems, E87-D (8). Aug. 2004. pp. 2029-2037.

国際会議

1. Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano: “The Design of Scalable Stochastic Biochemical Simulator on FPGA.”, Proceedings of 2005 IEEE International Conference on Field Programmable Technology, Signapore, Dec. 2005, pp. 339-340.

2. Yasunori Osana, Yow Iwaoka, Tomonori Fukushima, Masato Yoshimi, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano, and Hideharu Amano: “A Framework for ODE-Based Multimodel Biochemical Simulations on an FPGA.”, Proceedings of the 15th Field Programmable Logic and its applications(FPL) 2005, Tampere, Finland. August 2005. pp. 574–577.

3. Naoki Iwanaga, Yuichiro Shibata, Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Tomonori Fukushima, Hideharu Amano, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano, and Kiyoshi Oguri: “Efficient Scheduling of Rate Law Functions for ODE-based Multimodel Biochemical Simulation on an FPGA.”, Proceedings of the 15th Field Programmable Logic and its applications(FPL) 2005, Tampere, Finland. August 2005. pp. 666–669.

4. Yasunori Osana, Tomonori Fukushima, Masato Yoshimi, Yow Iwaoka, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hiroaki Kitano, and Hideharu Amano: “An FPGA-Based, Multi-model Simulation Method for Biochemical Systems.”, Proccedings of the 19th International Parallel and Distributed Processing Symposium / Reconfigurable Architecture Workshop, Denver, Colorado, USA. Apr 2005.

5. Masato Yoshimi, Yasunori Osana, Tomonori Fukushima, and Hideharu Amano: “Stochastic Simulation for Biochemical Reactions on FPGA.”, Proceedings of the 14th International Conference on Field Programmable Logic and Applications, Aug. 2004. pp. 105-114.

6. Yasunori Osana, Tomonori Fukushima, and Hideharu Amano: “ReCSiP: a ReConfigurable Cell SImulation Platform - Accelerating Biological Applications with FPGA -.”, Proceedings of the 9th Asia South Pacific Design Automation Conference, Jan. 2004. pp. 731-733.

7. Yasunori Osana, Tomonori Fukushima, and Hideharu Amano: “Implementation of ReCSiP: a Reconfigurable Cell SImulation Platform.”, Proceedings of the 13th International Conference on Field Programmable Logic and Applications, Sep. 2003. pp. 766-775.

8. Yasunori Osana, Hironori Nakajo, Noriaki Suzuki, Tomonori Tamura, and Hideharu Amano: “Performance Evaluation of a Parallel I/O Mechanism on a Massively Parallel Processing System JUMP-1.”, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Jun. 2001. pp. 1179–1185.

講演

1. 長名 保範: “FPGA を用いた生化学シミュレータ ReCSiP.”, 新和電材カスタマ・セミナー, 新和電材株式会社, 東京. Nov. 2005.

2. 長名 保範: “Reconfigurable Hardware の Bioinformatics への応用.”, 文部科学省振興調整費「システム生物学者育成プログラム」シンポジウム, 慶應義塾大学理工学部, 横浜. Sep. 2004.

研究会ほか

1. 長名 保範, 岩永 直樹, 吉見 真聡, 岩岡 洋, 小嶋 利紀, 西川 由理, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 北野 宏明, 天野 英晴: “FPGA を用いた生化学シミュレータ ReCSiP におけるハードウェアリソース消費に関する考察.”, 電子情報通信学会. VLD2005-107. Jan. 2006.

2. 西川 由理, 長名 保範, 吉見 真聡, 岩岡 洋, 小嶋 利紀, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “FPGA を用いた生化学シミュレータ ReCSiP における数値積分機構の性能改善手法.”, 電子情報通信学会. VLD2005-106. Jan. 2006.

3. 吉見 真聡, 長名 保範, 岩岡 洋, 西川 由理, 小嶋 利紀, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “データ転送網を用いた確率モデル生化学シミュレータの FPGA への実装.”, 電子情報通信学会. VLD2005-105. Jan. 2006.

4. 長名 保範, 吉見 真聡, 天野 英晴: “可変構造型デバイスを用いた神経回路網シミュレータの実現に関する検討.”, 情報処理学会. ARC158-18. Nov. 2005.

5. Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano, and Hideharu Amano,: “A System Design of Accelerating Biochemical Simulations on Programmable Hardware.”, The 10th Workshop on Software Platforms for Systems Biology, Harvard Medical School, Boston, MA, Oct. 2005.

6. 長名 保範, 吉見 真聡, 岩岡 洋, 小嶋 利紀, 西川 由理, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “FPGA を用いた生化学シミュレータ ReCSiP のシミュレーション制御機構.”, 電子情報通信学会. RECONF2005-39. Sep. 2005.

7. 岩岡 洋, 長名 保範, 吉見 真聡, 小嶋 利紀, 西川 由理, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “FPGA を用いた生化学シミュレータ向け SBML 処理系の構築.”, 電子情報通信学会. RECONF2005-40. Sep. 2005.

8. 吉見 真聡, 長名 保範, 岩岡 洋, 柴田 裕一郎, 岩永 直樹, 天野 英晴: “FPGA を用いた科学技術計算向け浮動小数点算術演算器の設計.”, パルテノン研究会. 第 26 回研究会-1-7. May. 2005.

9. 長名 保範, 吉見 真聡, 岩岡 洋, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “FPGA を用いた生化学シミュレータ ReCSiP 向けの数値積分機構の実装と評価.”, 電子情報通信学会. RECONF2005-7. May. 2005.

10. 岩永 直樹, 柴田 裕一郎, 吉見 真聡, 長名 保範, 岩岡 洋, 福島 知紀, 天野 英晴, 舟橋 啓, 広井 賀子, 北野 宏明, 小栗 清: “FPGA を用いた生化学シミュレータにおける反応速度関数スケジューリングに関する検討.”, 電子情報通信学会. RECONF2005-8. May. 2005.

11. 吉見 真聡, 長名 保範, 岩岡 洋, 福島 知紀, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “ヒープ木を用いた確率モデル生化学シミュレータの FPGA への実装.”, 電子情報通信学会. RECONF2005-9. May. 2005.

12. 長名 保範, 福島 知紀, 吉見 真聡, 岩岡 洋, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “FPGA を用いた生化学シミュレータ用の SBML 処理系の構築.”, 情報処理学会. ARC161-3. Jan. 2005.

13. 岩岡 洋, 長名 保範, 福島 知紀, 吉見 真聡, 舟橋 啓, 広井 賀子, 柴田 裕一郎, 岩永 直樹, 北野 宏明, 天野 英晴: “SBML 対応細胞シミュレータ環境の構築.”, 情報処理学会. VLD2004-109. Jan. 2005.

14. 長名 保範, 福島 知紀, 吉見 真聡, 天野 英晴: “FPGA を用いた細胞内代謝系のマルチモデルシミュレーション.”, 電子情報通信学会. 第 4 回リコンフィギャラブルシステム研究会-8. Sep. 2004.

15. 吉見 真聡, 長名 保範, 福島 知紀, 天野 英晴: “確率モデルを用いた化学反応シミュレーションの FPGA によ る高速化.”, 電子情報通信学会. 第 4 回リコンフィギャラブルシステム研究会-33. Sep. 2004.

16. 福島 知紀, 長名 保範, 田村 友紀, 濱橋 秀互, 大浪 修一, 天野 英晴: “細胞系譜構築システムの FPGA 利用による高速化.”, 電子情報通信学会. 第 2 回リコンフィギャラブルシステム研究会-14. Nov. 2003.

17. 長名 保範, 福島 知紀, 吉見 真聡, 天野 英晴: “FPGA による細胞内代謝系のシミュレーション.”, 電子情報通信学会. 第 1 回リコンフィギャラブルシステム研究会-7. Sep. 2003.

18. 長名 保範, 福島 知紀, 天野 英晴: “FPGA を用いた細胞シミュレーションシステムの構築.”, 電子情報通信学会. CPSY2002-93. Jan. 2003.

19. 長名 保範, 安生 健一朗, 天野 英晴: “Reconfigurable Hardware による細胞シミュレータの高速 化手法.”, 情報処理学会. ARC148-8. May. 2002.

その他の論文

研究会ほか

1. Yasunori Osana,: “CellDesigner–SBML Layout Extension Annotation Converter.”, The 10th Workshop on Software Platforms for Systems Biology, Harvard Medical School, Boston, MA, Oct. 2005.

2. 額田 匡則, 鈴木 紀章, 天野 英晴, 西村 克信, 田村 友紀, 長名 保範, 小西 将人, 五島 正裕, 富田 眞治: “超並列計算機 JUMP-1 のマルチキャスト機構による性能向上.”, 電子情報通信学会. CPSY2001-49. Aug. 2001.

3. 長名 保範, 中條 拓伯, 鈴木 紀章, 田村 友紀, 天野 英晴: “超並列計算機 JUMP-1 の並列入出力機構の評価.”, 電子情報通信学会. CPSY2001-48. Aug. 2001.

付 録A　ReCSiP Board概略

ReCSiP Board は、ReCSiP の動作確認のためのリファレンス環境として開発された基板であり、Xilinx 社製の FPGA を搭載した PCI ボードである。ReCSiP-1、2、2.1 の 3 種類のボードが開発されており、現在は ReCSiP-2/2.1 Board を用いてシミュレータが稼働している。

図 A.1 に各ボードの概要を示し、以下で各ボードの構成について述べる。

A.1　各ボードの構成部品

A.1.1　ReCSiP-1 Board

ReCSiP-1 Board(図 A.2) は、Virtex-II (XC2V6000) と、同期 SRAM を搭載した基板であり、PCI インタフェイス部の動作確認など、後継基板の開発のためのノウハウ蓄積に貢献した。QL5064 の開発と並行して基板の設計が行われたため、QL5064 が BGA ソケットによる実装になっており、Virtex-II のコンフィギュレーションも MultiLINX cable と QL5064 の双方から行えるようになっている。

メモリは SRAM、DRAM ともに、2 チップ 1 組でそれぞれ 36bit、32bit のデータ幅で FPGA に接続されており、SRAM は合計 4 組、DRAM は 1 組の構成である。

主要部品

- FPGA: Xilinx Virtex-II (XC2V6000-4BF957C)
- SRAM: Micron 18Mb DCD SyncBurst SRAM (MT58L1ML18D: 1Mx18) x 8
- DRAM: Micron MT48LC16M16 166MHz SDR SDRAM (16Mx16) x 2

ジャンパ設定

ReCSiP-1 Board のジャンパ・コネクタ類の設定を表 A.1 に示す。通常は PCI バス、QL5064 経由で Virtex-II の configuration を行い、PCI バスの 66MHz 動作を許可するため、これらを設定するジャンパ J3 はオープンでよい。ロジックアナライザ等で信号を観測する場合のために、J6、J7 で合計 40 本のユーザ利用可能なピンヘッダが用意されている。

A.1.2　ReCSiP-2 Board

ReCSiP-2 Board (図 A.3) は、ReCSiP-1 Board で得られたノウハウを応用して設計された基板で、Virtex-II Pro と QDR-SRAM への変更が行われ、FDK 社の物理乱数生成チップ (RNG) が搭載されている。SRAM は QDR になって、DDR でのデータ転送になったが、データ幅は変わらず 36bit

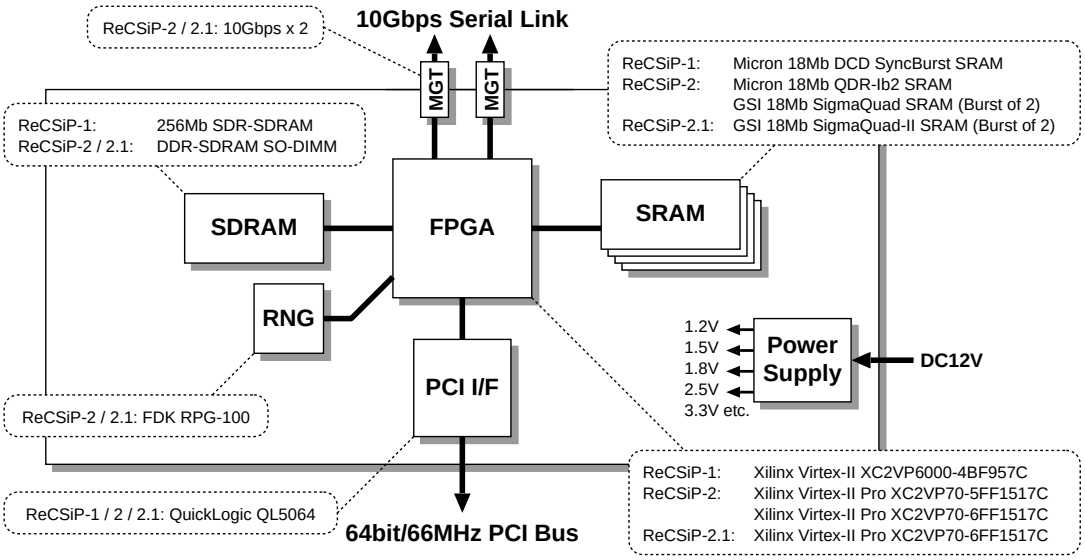


図 A.1 ボードの概要

図 A.2 ReCSiP-1 Board

表 A.1　ReCSiP-1 Board のジャンパ・コネクタ

部品番号	ピン番号	機能
J1	1-8	MultiLINX ケーブルで FPGA の構成情報を転送するためのジャンパ。1 番ピンから順に D0-D7 を接続
J2	1-7	MultiLINX ケーブルの制御系信号用。1 番ピンから順に BUSY、WRITE、CS、INIT、PROG、DONE、CCLK を接続する。8 番ピンは未使用
J3	1-2	Open: QL5064 経由で Virtex-II を configuration Close: MultiLINX ケーブル経由で Virtex-II を configuration
J3	3-4	Open: PCI バスの 66MHz 動作を許可 Close: PCI バスを常に 33MHz で動作させる
J4	1-4	Virtex-II の JTAG 信号線。1 番ピンから順に TMS、TDO、TDI、TCK
J6 J7	1-20 1-20	デバッグ用に、Virtex-II に直結されており、ユーザが自由に利用可能
J15	1-6	MultiLINX ケーブル等に電源を供給するためのコネクタ。1、2 番ピンが 3.3V、5、6 番ピンが 5V で、3、4 番ピンが GND

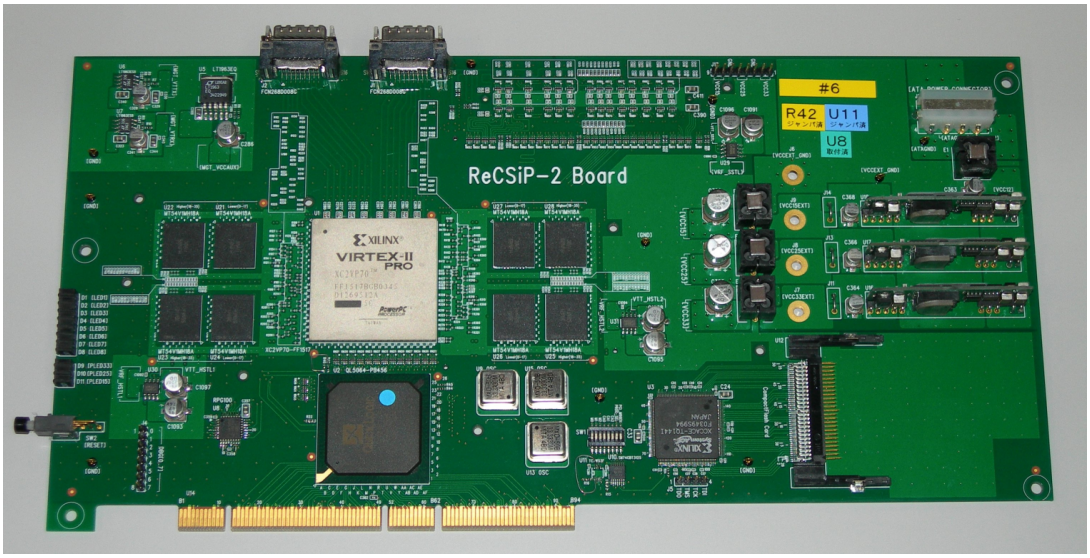


図 A.3　ReCSiP-2 Board

表 A.2 ReCSiP-2 Board のジャンパ・スイッチ設定

部品番号	ピン番号	機能
SW1	1-3	Virtex-II Pro の configuration mode を設定する信号。1 番ピンから順に M0、M1、M2 に接続されている。スイッチはプルアップされており、on で low、off で high になる
	4-6	SystemACE CF の configuration address を設定する信号。4 番ピンから順に CA0、CA1、CA2 に接続されている。スイッチはプルダウンされており、on で high、off で low になる
	7	On: QL5064 から configuration Off: SystemACE CF から configuration
	8	On: PCI バスを常に 33MHz で動作させる Off: PCI バスの 66MHz 動作を許可
J4	1-8	ユーザ用のデバッグ端子
J5	1-4	Virtex-II Pro および SystemACE CF の JTAG チェイン。1 番ピンから順に TDO、TMS、TCK、TDI
J12	1-8	JTAG ケーブルなどに接続するための電源ピン。1、2 番ピンが 3.3V、4、5 番ピンが 2.5V、7、8 番ピンが 1.5V で、3、6 番ピンが GND

である。DRAM は直接基板には実装せず、モジュールを基板裏面の DDR-SDRAM SO-DIMM 用のソケットに挿す形とした。

XC2VP70-5 と Micron 製の SRAM (166MHz QDR) を搭載した基板と、XC2VP70-6 と GSI Technology 製の SRAM (200MHz QDR) を搭載した基板の 2 種類が存在するが、両者は互換品であり、利用方法に差異は存在しない。SRAM は ReCSiP-1 Board 同様に 2 チップひと組、36bit 幅で FPGA に接続されており、合計 4 セットとなる。

主要部品

- FPGA: Xilinx Virtex-II Pro (XC2VP70-5FF1517C / 6FF1517C)
- SRAM: Micron 18Mb QDR-Ib2 SRAM (MT54V1MH18A) / GSI Technology 18Mb SigmaQuad SRAM (GS8180QV18) x 8
- DRAM: 200pin DDR-SDRAM SO-DIMM Socket x 1
- RNG: FDK RPG-100

スイッチ設定

表 A.2 が、ReCSiP-2 Board のジャンパ及びスイッチの一覧である。Slave SelectMAP モードでは {M0, M1, M2} = {0, 1, 1} であるため、通常 SW1 は 1 番と 7 番のみ ON にして動作させることになる。

表 A.3　ReCSiP-2 Board の MGT コネクタのピン配置

レーン	0	1	2	3
TX P	S15	S11	S7	S3
TX N	S16	S12	S8	S4
RX P	S14	S10	S6	S2
RX N	S13	S9	S5	S1

マルチギガビットトランシーバ

Virtex-II Pro の MGT (Multi Gigabit Transceiver) に接続されているコネクタは J1、J2 であり、それぞれ 4 レーンの双方向差動チャネルを持つ。J1/J2 は Infiniband 4X 等で用いられている HSSDC コネクタであるが、基板上の配線の関係上、ピン配置は独自のもの (表 A.3) となっている。

差動シリアル信号線の受信側にはコンデンサが挿入されており、2.5Gbps 前後での利用を想定した AC カップリングになっている。

オシレータ

ReCSiP-2 Board 上には次に挙げる 3 つのオシレータソケットが存在する。

- U15: ローカルバスクロック。QL5064 と Virtex-II Pro の両方に接続される。33MHz〜66MHz の範囲で動作させることが可能
- U9: ユーザクロック: Virtex-II Pro 内部のユーザロジックをローカルバスよりも高い周波数で動作させたい場合などに利用することが可能。不要な場合はオシレータを挿入しなくてもよい
- U13: SystemACE CF 用クロック: 通常は 33MHz のオシレータを挿入する

これらの他に、MGT 用のリファレンスクロックとして U4 (EPSON EG-2121: 125.000MHz LVDS) が搭載されている。

A.1.3　ReCSiP-2.1 Board

ReCSiP-2.1 Board (図 A.4) は、ReCSiP-2 Board の FPGA のピン配置などの不具合を修正し、さらに SRAM を QDR-II SRAM 相当品に変更した基板である。搭載している SRAM は 18Mb であるが、アドレス線は 36/72/144Mb 対応分まで結線されており、チップを交換すれば容量を拡大することができる設計になっている。

主要部品

- FPGA: Xilinx Virtex-II Pro (XC2VP70-6FF1517C)
- SRAM: GSI Technology 18Mb SigmaQuad-II SRAM (GS8182Q18) x 8

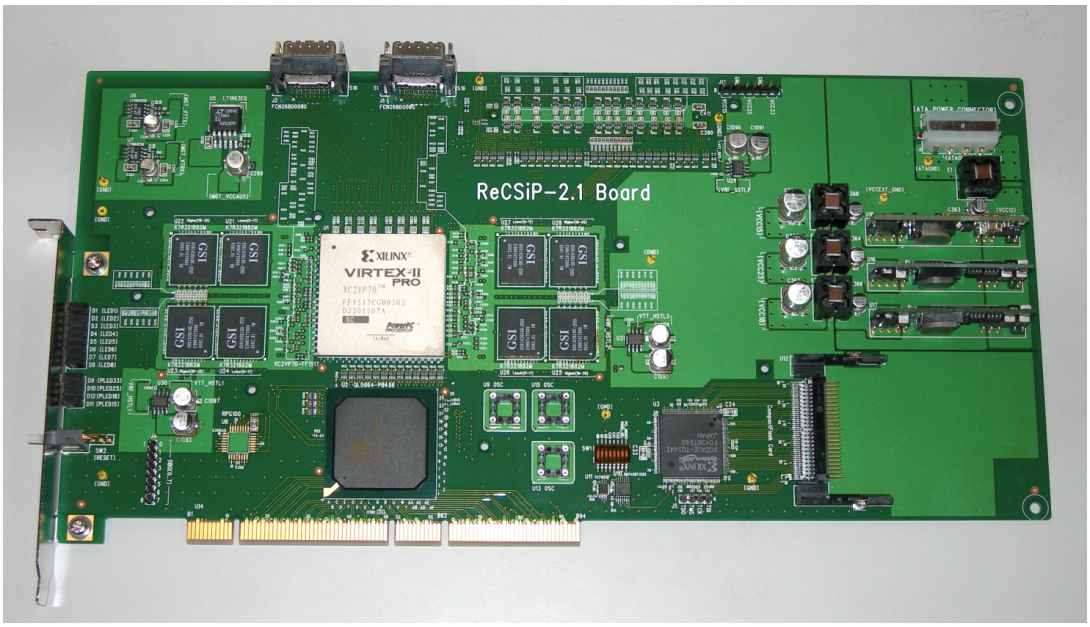


図 A.4 ReCSiP-2.1 Board

表 A.4 ReCSiP Board の PCI vendor、product、revision コード

	Vendor	Product	Rev.	FPGA	SRAM
ReCSiP-1 Board	0x3614	0x3086	0x01	XC2V6000-4	Micron SyncBurst
ReCSiP-2 Board			0x02	XC2VP70-5	Micron QDR-I
			0x03	XC2VP70-6	GSI SigmaQuad
ReCSiP-2.1 Board			0x04	XC2VP70-6	GSI SigmaQuad-II

- DRAM: 200pin DDR-SDRAM SO-DIMM Socket x 1
- RNG: FDK RPG-100

ReCSiP-2.1 Board は、一部の部品と FPGA の SRAM 関係のピンアサインが変更になった他は ReCSiP-2 Board と同等である。

A.2 PCI インタフェイス

各ボードの PCI インタフェイス部には QuickLogic QL5064 を使用しており、3.3V/5V、32bit/64bit、33MHz/66MHz の各モードで、マスタ/スレーブとして動作可能である。Vendor、product、revision の各コードを表 A.4 に、ベースアドレスレジスタ (BAR) の設定を表 A.5 に、それぞれ示す。ベースアドレスレジスタは 3 つ使用されており、BAR0 には QL5064 自体のシステムレジスタ空間、BAR1 にはローカルバスのアドレス空間、BAR2 には QL5064 から Virtex-II/II Pro をコンフィギュレーションするためのコントローラのアドレス空間が、それぞれマップされている。

表 A.5　PCI インタフェイスのベースアドレスレジスタ設定

BAR	Address width (Size)	Type	役割
0	8 (256B)	Memory	QL5064 システムレジスタ
1	27 (128MB)	Memory	ReCSiP Board ローカルバス
2	8 (256B)	I/O	ReCSiP Board FPGA コンフィギュレーション

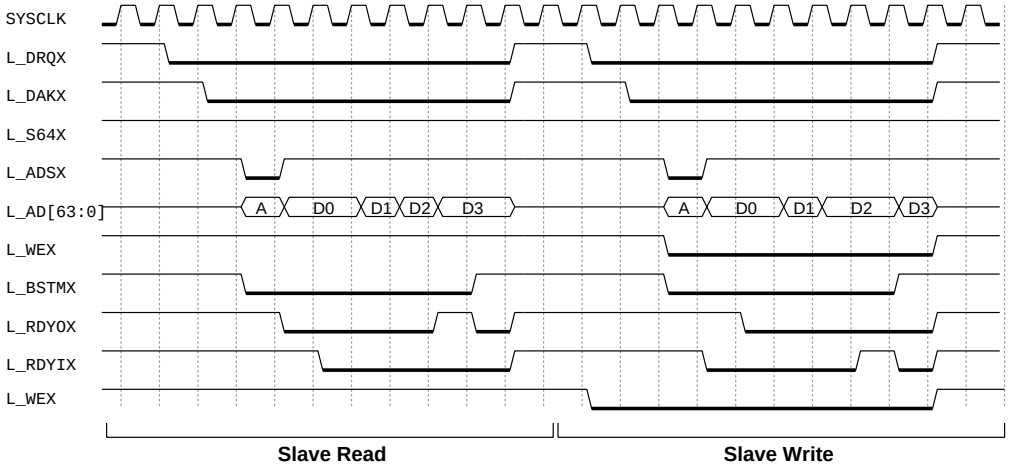


図 A.5　ローカルバスの Slave Read/Write 動作

A.3　ローカルバス

A.3.1　概要

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 A.6 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して動作させることが可能である。

なお、ローカルバスは 64bit 幅であるが、ターゲットアクセス時のアドレス空間は 27bit である。

A.3.2　スレーブアクセス

ローカルバスの slave read/write 動作におけるタイミングを図 A.5 に示す。

スレーブアクセスでは QL5064 が PCI バスのスレーブ、ローカルバスのマスタとして動作し、Virtex-II/II Pro がローカルバスのスレーブとなる。27bit のアドレス空間はホスト CPU の物理メモリアドレス空間内にマッピングされており、このアドレス空間に対して read/write を行うとアドレスの下位 27 ビットをローカルバスのアドレスとして Virtex-II/II Pro に対するアクセスが行われる。

このアクセスモードではまず QL5064 が L_DRQX を assert し、これに対して Virtex-II/II Pro が L_DAKX を assert することで転送が開始される。転送の始めにはまず ADSX と AD を用いてアドレ

表 A.6 ローカルバスの信号線

信号名	方向	論理	能書き
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	スレーブ転送要求信号
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。スレーブアクセス時は出力 (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、スレーブアクセス時は出力、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。スレーブ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。スレーブ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

＊「方向」は QL5064 からみた信号の入出力方向

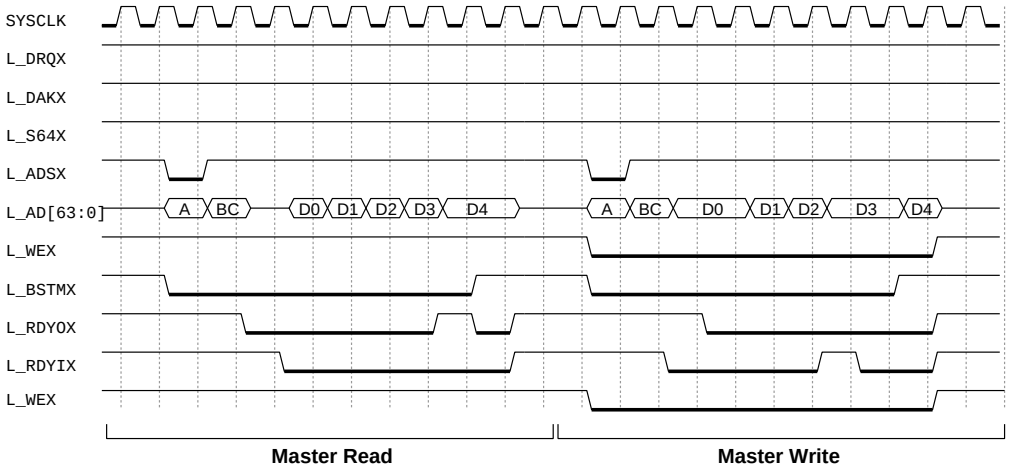


図 A.6　ローカルバスの Master Read/Write 動作

スが転送され、その後は L_RDYIX、L_RDYOX(注 1)を用いてハンドシェイクを行いながら、連続したアドレス空間に 8 バイトずつデータが転送される。アドレスは 8 バイト境界に基づくため、一連の転送の最初と最後のワードが 8 バイト境界を埋めきれない場合、L_BEX を用いてバイト単位での転送になることがあるが、一連の転送の最初と最後のワード以外で L_BEX が利用されることはない。転送サイクルは L_DRQX が negate された時点で転送は終了となる。

A.3.3　マスタアクセス

PCI バスが他のデバイスの DMA によって使用されていなければ QL5064 が PCI バスのマスタ、ローカルバスのスレーブに、Virtex-II/II Pro がローカルバスのマスタとなってデータ転送を行うことができる。他のデバイスによる DMA が行われているかどうかは L_DBSYX を用いて確認することができ、この信号が High であればマスタ転送を開始することができる。

マスタ転送モードは Virtex-II/II Pro 側が L_ADSX を assert し、L_AD のビット [31:0] に転送先のアドレスを出力することで開始される。この場合の転送先アドレスは、ホスト側の 32bit の物理アドレスである。アドレスを出力した次のクロックサイクルでは L_AD のビット [23:0] に転送するバイト数を出力する。転送バイト数を出力した後はスレーブ転送時と同様に L_RDYOX、L_RDYIX を用いてハンドシェイクを行いながらデータを転送する。

なお、マスタアクセスを行う場合には、PCI-localbus ブリッジの FIFO が適切に動作するように FIFO Almost Full の閾値を初期化する必要がある。このレジスタはベースアドレスレジスタ 0 のオフセット 0x68 にあり、64bit の 0 を書き込むこで初期化が行われる。初期化動作は一度だけ行えばよいので、デバイスドライバのロード時などに実行するのが適切である。

表 A.7 Virtex-II/II Pro コンフィギュレーション機構の外部信号

信号名	方向	論理	機能
PROG	out	負	FPGA に対する configuration 要求信号
INIT	in	負	FPGA の内部 RAM クリア信号。Low のとき FPGA は初期化動作中で、High になるとコンフィギュレーション可能
CCLK	out	なし	Configuration 用のクロック信号。ローカルバスのクロックを 2 分周したものが使われ、コンフィギュレーション関係の信号はこれに同期して動作する
D[0:7]	out	正	コンフィギュレーション用のデータ線。D0 が最上位ビットになる
CS	out	負	FPGA へのチップセレクト信号。CS が Low になった次の CCLK 立ち上がりからデータが取り込まれる
WRITE	out	負	D の方向を示す。この信号が Low のときコンフィギュレーションデータの書き込みとなり、High のときは読み出しとなるが、読み出しはこのコンフィギュレーション機構ではサポートされない
BUSY	in	正	FPGA 側がコンフィギュレーションデータの処理で busy になったことを示す。CCLK が 50MHz を超えない場合には無視してよい
DONE	in	正	コンフィギュレーション中は Low になり、終了すると High になる

表 A.8　コンフィギュレーション機構の制御レジスタ

アドレス	レジスタ名	ビット	名称	機能
0x00 (write)	Control	31:1	Reserved	予約
		0	CFR	Configuration request: 1 を書くことでコンフィギュレーション、0 を書くことでコンフィギュレーション機構をリセット
0x08 (read)	Status	31:8	Reserved	予約
		7	REQ	CFR の状態を読み出し。CFR はコンフィギュレーション終了で 0 になる
		6:4	Reserved	予約
		3	BUSY	FPGA に接続されている BUSY 信号
		2	DONE	FPGA に接続されている DONE 信号
		1	INIT	FPGA に接続されている INIT 信号
		0	CFA	Configuration acknowledge: FPGA のコンフィギュレーション許可ビット。CFR に 1 を書き込んだあと、CFA が 1 になってからコンフィギュレーションデータレジスタにデータを書き込む
0x08 (write)	Configuration Data	31:8	Reserved	予約
		7:0	Data	FPGA のコンフィギュレーションデータを書込み

A.4 Virtex-II/II Pro コンフィギュレーション機構

QL5064 にはローカルバス–PCI バスのブリッジの他に Virtex-II/II Pro をコンフィギュレーションするための機構も実装されており、ベースアドレスレジスタ 2 の I/O 空間にマップされている。コンフィギュレーションは Slave SelectMAP モードで行うようになっており、表 A.7 に示すような信号線で Virtex-II/II Pro に接続されている。

コンフィギュレーションを行う場合、ホスト側のソフトウェアでは

1. コントロールレジスタ (0x00) の CFR(ビット 0) に 1 を書き込んでコンフィギュレーション動作を要求
2. ステータスレジスタ (0x08) の CFA(ビット 1) が 1 になるまで待機
3. データレジスタ (0x08 のビット 7:0) に 1 バイトずつデータを書き込み
4. ステータスレジスタ (0x08) の DONE(ビット 2) が 1 であれば正常終了

という手順になる。このコンフィギュレーション機構を経由して FPGA に書き込む構成情報は Parallel PROM からコンフィギュレーションを行う場合の、PROM の内容と同じものでよいので、Xilinx のツールが生成する bit ファイルから、

```
% promgen -u 0000 example.bit -p mcs -x 32M -o example.mcs
```

のように MCS86 形式のファイルを生成することでコンフィギュレーションに必要なデータを得ることができる。

(注 1)L_RDYOX、L_RDYIX は、受け手→送り手の順で準備完了を通知するための信号であり、read と write で、QL5064 と Virtex-II/II Pro のどちらが先に assert するかが異なる

付 録B　各モジュール仕様概略

B.1　積分モジュール

B.1.1　構成

図 B.1 に Euler 法積分モジュール (Phase 1) の詳細な構成を示す。Pathway RAM から読み出されたポインタ配列は、[*X*] RAM および *k* RAM にはそのままアドレスとして与えられるが、その他のメモリに関しては Solver Core やそれに続く演算器の動作による遅延に合わせてアドレスを入力する必要があるため、Pathway RAM の出力の一部はタイミング調節用のシフトレジスタに入力される。

シフトレジスタは 2 系統あり、一方は Pathway RAM の出力を伝達するためのもの、もう一方は Solver Core の出力が有効な演算結果であることを示すためのものである。後者は積分モジュール内部の制御部によって入力が与えられ、d[*X*] RAM の書き込み制御に用いられる。この 2 系統のシフトレジスタはそれぞれ、Solver Core と同じ長さの外付けのものと、積分モジュール内部の固定長のもので構成されており、Solver Core と同じ遅延のものを Solver Core とあわせて積分モジュールに接続することで、積分モジュールの構成を変更することなく、さまざまな遅延を持つ Solver Core に対応することができる。

Pathway RAM の出力を伝達するシフトレジスタのうち、Solver Core に内蔵されているもの (idt Pathway SR) は、d[*X*] RAM の値を読み出したり、加算器の入力を制御するマルチプレクサの選択信号を与えるために、シフトレジスタの途中の段から値が読み出される。

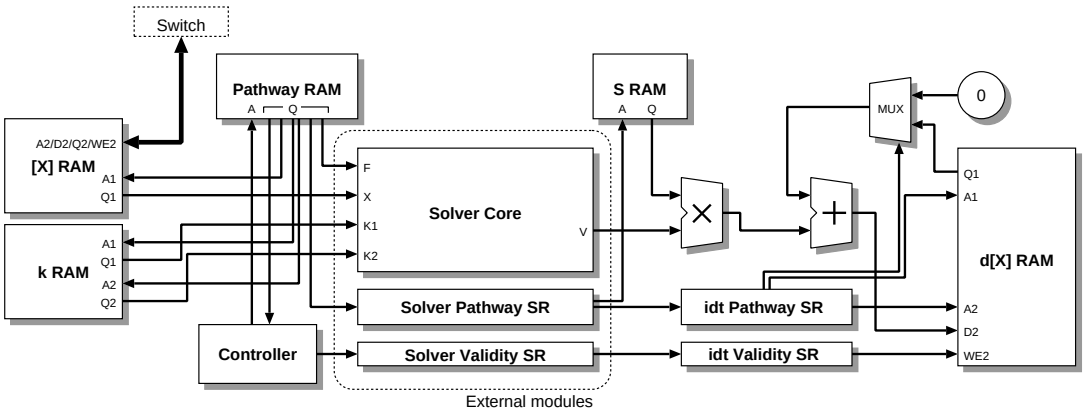


図 B.1　積分モジュールの構成 (Euler、Phase 1)

表 B.1 Pathway RAM のフォーマット

位置	幅	名称	役割
53	1	Special	0: 通常の演算処理
			1: 制御命令
52	1	DX Clean	0: 計算結果を d[X] RAM に加算
			1: 計算結果を d[X] RAM に直接書き込み
51:48	4	Func	Special=0 のとき: Solver Core に与える反応速度式選択信号
			Special=1 のとき、制御命令コード
47:40	8	S Address	S RAM の読み出しアドレス
39:30	10	DX Address	Phase 1: dX RAM の書き込みアドレス
			Phase 2: dX RAM の読み出しアドレス
29:20	10	X Address	Phase 1: X RAM の読み出しアドレス
			Phase 2: X RAM の書き込みアドレス
19:10	10	K1 Address	k RAM の読み出しアドレス (Port 1)
9:0	10	K2 Address	Special=0 のとき: k RAM の読み出しアドレス (Port 2)
			Special=1 のとき: 制御命令オペランド

B.1.2 Pathway RAM の仕様

表 B.1 に、Pathway RAM のワードの各ビットの役割を示す。1 ワードは 54 ビット幅であり、各メモリへ与えるアドレスのほかに、処理の終了やアイドルサイクルなどを指示するための制御命令 (表 B.2) を含む。Pathway RAM は毎クロック、0 番地から順に読み出されるが、制御命令によって一時的に停止したり、0 番地に戻したりすることができる。

B.1.3 動作

表 B.3 に、Pathway RAM の記述例を示す。この記述例では、

- S1、S2 のふたつの物質
- R1、R2 のふたつの反応
  - R1 では S1 の濃度だけが変化
  - R2 では S1、S2 ともに濃度が変化

する反応経路を表している (簡単にするために、k RAM は 1 ポートだけ表記し、S RAM は省略してある)。この Pathway RAM の記述に基づく積分モジュールの動作を図示したのが図 B.2 である。

積分モジュールが動作を開始すると、Pathway RAM は 0 番地から読み出される。読み出された Pathway RAM の値に基づいて [X] RAM と k RAM がアクセスされ、そこから読み出された値が Solver Core とそれに続く乗算器・加算器の構成するパイプラインに投入される。このパイプラインの遅延は一定であり、加算器から取り出された値は Pathway RAM によって指定された d[X] RAM のアドレスに書き込まれる。Pathway RAM からひとつめの END 命令が読み出された時点で

表 B.2　Pathway RAM の命令コード

[51:48]	名称	動作
0x00	NOP	3 クロックサイクル以上のアイドルサイクルが必要な場合に用いる。サイクル数は Pathway RAM のビット [9:0] で指定し、最大 1024 サイクルまで指定可能
0x01	END	Phase 1、Phase2 の終了を示す。0 番地から順に読み出して最初に出てくる END が Phase 1 の終了であり、このワードの [9:0] で Phase 2 の自動加算処理の終了アドレスを指定する。次に出てくる END が Phase 2 の終了を示し、このワードの [9:0] で指定されたクロックサイクル数だけ待機したのちに Phase 1 に戻り、Pathway RAM の 0 番地から処理を開始する
0x02	XtoK	[*X*] RAM から *k* RAM への転送を行う。このワードの [29:20] で読み出し元の [*X*] RAM のアドレスを指定し、次のワードの [9:0] で書き込み先の *k* RAM のアドレスを指定する。書き込み先アドレスの指定には XtoK を連続して用いるか、NOP1 を用いる
0x03	NOP1	1 クロックサイクルのアイドルサイクルを発生する。2 クロックサイクル以下のアイドルサイクルが必要な場合や、XtoK による転送の最後に書き込みアドレスを指定するのに用いる

Phase 1 での [*X*] RAM および *k* RAM の読み出しは終了するが、Solver Core 以降のパイプラインの処理が完了して d[*X*] RAM に結果が書き込まれるまでが Phase 1 であるため、自動的にパイプラインの遅延分のクロックサイクル数分の待ち時間が確保される。Phase 1 を終了する END 命令のワードでは、ビット [9:0] を用いて、Phase 2 の最初に行われる自動加算処理の終了アドレスを指定する。

続いて Phase 2 の処理が開始されるが、Phase 2 の最初のステップは Pathway RAM によらない積分動作 (自動加算) であり、アドレス 0 から、Phase 1 を終了する END 命令の引数で与えられたアドレスまで、d[*X*] RAM の値を直接 [*X*] RAM に加算する。この処理では *S* RAM のアドレスを指定することはできず、乗算器の *S* RAM 側のポートには常に 1 が入力される。この処理が終了すると、Pathway RAM による加算動作が行われ、ここでは d[*X*]、*S*、[*X*] RAM のアドレスをそれぞれ Pathway RAM によって指定することができる。Phase 2 が終了すると、Pathway RAM のアドレスは 0 に戻って次の時間刻みの Phase 1 の処理が開始されるが、Phase 2 の最後にも Phase 1 のときと同様にパイプライン遅延分の待ち時間が必要であり、これが自動的に挿入される。複数の Solver を並列動作させる場合などで追加の待ち時間が必要な場合には、Phase 2 を終了する END 命令のワードの、[9:0] により、追加の待ち時間のクロックサイクル数を指定することができる。

B.1.4　Solver Core インタフェイス

積分モジュールの、Solver Core インタフェイス部の信号を表 B.4 に示す。Solver Core インタフェイス部には Solver Core のほかに、積分モジュールの制御のために Solver Core と同じ遅延サイクル

表 B.3　Pathway RAM の記述例

Address	X	K1	dX	説明
0	00 (S1)	10	00	反応 R1
1	01 (S2)	11	01	反応 R2
2	END 1			Phase 1 終了 (アドレス 1 まで自動加算)
3	00 (S1)		01	物質 S1 の濃度変化への R2 の寄与分
4	END 0			Phase 2 終了 (待ちサイクルなし)

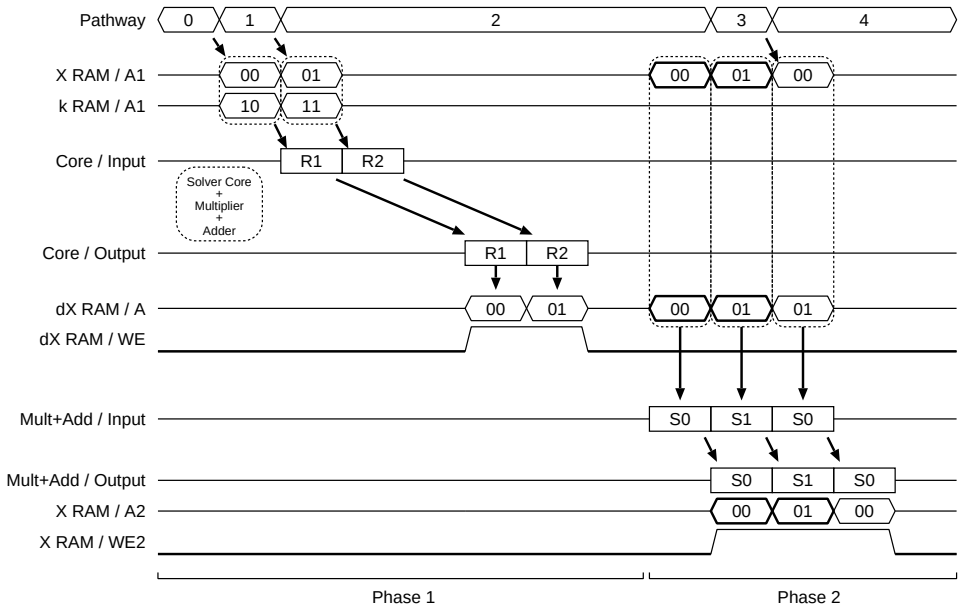


図 B.2　表 B.3 の Pathway RAM を用いた場合の積分モジュールの動作

表 B.4　Solver Core インタフェイス部の信号

信号名	幅	方向	機能
CORE_RST	1	out	Solver Core のリセット信号。積分モジュールに与えられるリセット信号とは独立に、積分モジュールによって制御される
CORE_F	4	out	Solver Core の反応速度式選択信号
CORE_X	36	in	[X] RAM から Solver Core への入力
CORE_K1	36	in	k RAM から Solver Core への入力 (Port 1)
CORE_K2	36	in	k RAM から Solver Core への入力 (Port 2)
CORE_V	36	in	Solver Core の出力
CORE_LATENCY	8	in	Solver Core のパイプライン遅延サイクル数を入力
SPSR_A	54	out	Solver Pathway SR の入力側を接続
SPSR_Q	54	in	Solver Pathway SR の出力側を接続
SVSR_A	1	out	Solver Validity SR の入力側を接続
SVSR_Q	1	in	Solver Validity SR の出力側を接続

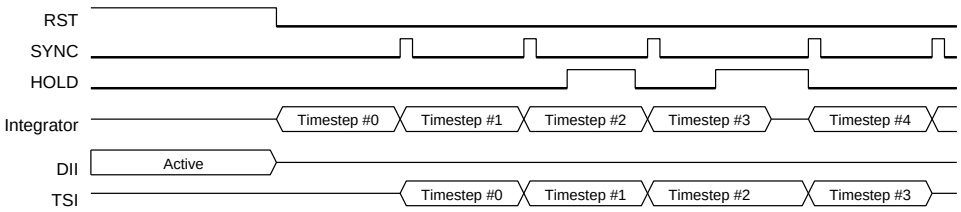


図 B.3　積分モジュールの外部制御信号と動作

数を持つシフトレジスタをふたつ接続する必要がある。これらのシフトレジスタは、Pathway RAM の内容を積分モジュールの Solver Core より後ろの部分に伝達するためのもの (Solver Pathway SR: 54bit 幅) と、Solver Core の出力が有効なものであることを示すためのもの (Solver Validity SR: 1bit 幅) であり、それぞれ SPSR_、SVSR_ではじまる名称の信号線に接続される。

Solver Core には、CORE_ではじまる名称の信号線を接続する。Solver Core のリセットは積分モジュールが行うことになっており、Solver Core は CORE_RST が Low になった後の、最初のクロック立ち上がりから入力データの取り込みと計算処理を開始する。

B.1.5　外部インタフェイス

積分モジュールの動作を制御するための制御信号とその動作を図 B.3 および表 B.5 に示す。積分モジュールに含まれるメモリはシミュレーション開始時にデータ注入用のインタフェイス (Data Injection Interface: DII) を用いて初期化される。この間、リセット信号 (RST) は 1 に保たれなければならない。

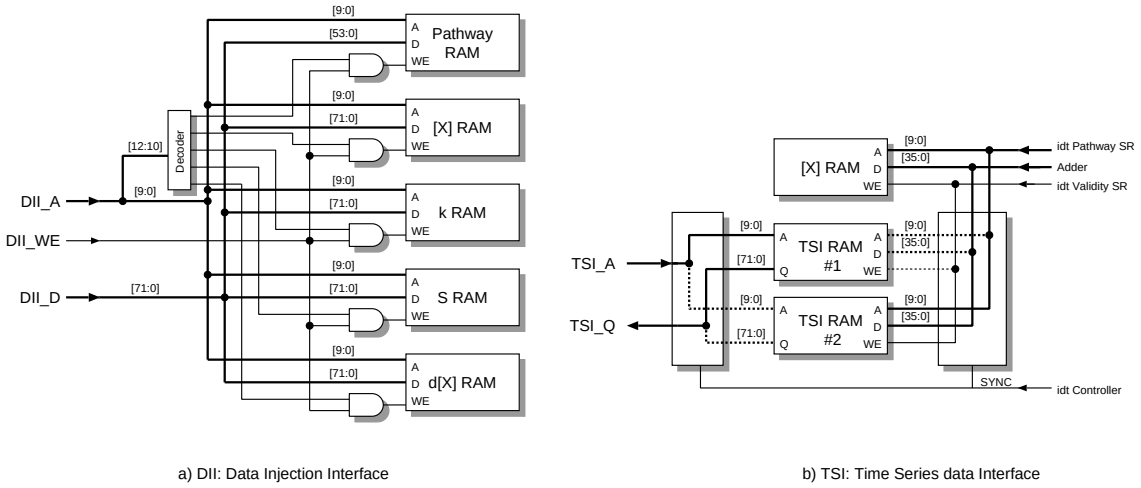
リセット信号が 0 になると、積分モジュールが動作を開始され、時間刻みが進むごとに 1 クロックサイクルの間、SYNC が assert されることによって処理の進行状況を外部に通知する。時間刻

表 B.5　積分モジュールの外部制御信号

信号名	方向	機能
RST	in	1: 回路のリセットと DII を用いた初期化動作 0: シミュレーション実行
SYNC	out	1 時間刻み分の処理の終了を示す
HOLD	in	0: 現在の時間刻みの処理を終了後、続行 1: 現在の時間刻みの処理を終了後、一時停止
DII_*	in	初期化時のメモリ初期化のためのインタフェイス (表 B.6)
TSI_*	in/out	時間刻み毎のデータ読み出しのためのインタフェイス (表 B.6)

表 B.6　積分モジュールのデータ入出力インタフェイスの信号

信号名	幅	方向	ビット	機能
DII_A	13	in	12:10	書き込むメモリを選択。0=[X] RAM、1=d[X] RAM、2=k RAM、3=S RAM、4=Pathway RAM
			9:0	書き込む先のメモリアドレス
DII_D	72	in	71:36	Pathway RAM 以外: アドレス A[9:0]+1 に書き込むデータ
			35:0	Pathway RAM 以外: アドレス A[9:0] に書き込むデータ
			53:0	Pathway RAM: アドレス A[9:0] に書き込むデータ
DII_Q	72	out	71:0	未使用
DII_WE	1	in	—	A[12:10] で選択したメモリへの書き込み許可
TSI_A	10	in	9:0	読み出すメモリアドレス
TSI_Q	72	out	71:36	A[9:0]+1 から読み出されたデータ
			35:0	A[9:0] から読み出されたデータ



a) DII: Data Injection Interface　　b) TSI: Time Series data Interface

図 B.4　積分モジュールのデータ入出力インタフェイスの構成

表 B.7　クロスバスイッチの各ポートの信号

名称	幅	方向	役割
IN	36	in	クロスバへの入力データ
OUT	36	out	クロスバからの出力データ
DST	14	in	IN に入力されるデータの送信先

み毎の途中経過は、時系列データ取得用のインタフェイス (Time Series data Interface: TSI) から読み出すことができ、ここには [X] RAM のコピーである TSI RAM が接続されている。TSI RAM は図 B.4 (b) に示すように 2 セット用意されており、この 2 つを入れ替えながら片方は [X] RAM への書き込み信号をそのまま接続してデータのコピーを行い、もう一方をデータ取得用インタフェイスに接続することで、処理の進行中にひとつ前の時間刻みのデータを取得することができるようになっている。ホストへの転送が間に合わない場合に、一時的に処理の進行を停止するための信号として HOLD が用意されており、これが assert されている場合は 1 時間刻み分の処理の終了後に積分モジュールが HOLD が negate されるまで一時停止する。これによって、ホストインタフェイス部の転送能力に合わせて処理を行うことが可能である。

DII は書き込み専用、TSI は読み出し専用となっており、[X]、k、S、d[X] などのメモリは、連続した 2 ワードを同時に読み書きするようになっており、64bit のローカルバスへの接続に適した構成となっている。DII、TSI に関連する信号は表 B.6 に示す通りである。

B.2 Crossbar

スイッチの中核となるのがクロスバスイッチ (以下クロスバと表記) である。ReCSiP では、シミュレーションの開始前にすべての Solver での処理が静的にスケジュールされており、スイッチを用いた Solver 間の通信も同様に静的にスケジュールされるため、データの送受信時の衝突などは発生しない。これにより、簡潔な構成のクロスバを用いることを可能にしている。

表 B.7 はクロスバの各ポートの構成である。送信先を指定するための信号である DST は 14 ビット幅であり、クロスバは最大で 14 のポートを持つことができる。DST の各ビットはクロスバの各ポートに対応しており、1 になっているポートに IN に入力された値が送信される。ここで複数のビットを 1 にすれば宛先を指定することも可能である。各ポートの IN から入力された値は、1 クロック遅れて宛先ポートの OUT から出力される。

なお、データを送信しないときには DST のすべてのビットを 0 にしておく。また、送信先がぶつからなければ、複数の送信元からのデータ転送を同時に行うことができる。

B.3 Transceiver

B.3.1 構成

クロスバ単体では通信を制御する能力を持たないため、クロスバスイッチの各ポートに接続して、予め与えられたスケジュールに基づいて Solver との仲介を行うモジュールが Transceiver であ

表 B.8 Transceiver Code RAM のフォーマット

位置	幅	名称	役割
35	1	Special	0 のとき通常のデータ転送、1 のとき制御命令
34	1	Write Enable	Solver に与える書き込み許可信号
33:24	10	Address	Solver に与えるアドレス信号
23	1	Buffer Select	0: Solver の出力ポートの信号をクロスバに出力
			1: バッファメモリの出力をクロスバに出力
22	1	Buffer Write Enable	バッファメモリの書き込み許可信号
21:18	4	Buffer Write Address	バッファメモリの書き込み側アドレス
17:14	4	Buffer Read Address	バッファメモリの読み出し側アドレス
13:0	14	Crossbar Destination	クロスバスイッチに与えるデータ送り先
13:10	4	Instruction	Special=1 のとき、制御命令コード
9:0	10	Command Parameter	Special=1 のとき、制御命令オペランド

る。Solver のスイッチ側インタフェイスは [*X*] RAM または d[*X*] RAM そのもののポートであるため、Transceiver はこれらのメモリを読み書きして、クロスバとの送信あるいは受信を行う。

Transceiver は、図 3.26 (46 ページ) に示すように、Solver 側のインタフェイスとスイッチ側のインタフェイスに加えて、送信データを一時的に蓄積するためのバッファメモリを持ち、これらを一括して Transceiver Code RAM (以下 Code RAM と表記) に格納されたポインタと制御ビット列の配列が制御する。Code RAM の各ワードは 36 ビット幅であり、各ビットは表 B.8 のような意味を持つ。

バッファメモリを用いない場合、Solver の出力はクロスバの入力に、クロスバの出力は Solver の入力に直結され、Solver のアドレス線 (A) および書き込み許可信号 (WE) には、Code RAM から読み出された値の当該ビットが接続される。Solver 内のメモリの読み出しは 1 クロックの遅延を伴うため、クロスバの送信先を示す DST は Code RAM から読み出された後、1 クロックの遅延を経てクロスバに与えられる。これにより、送信するデータのアドレスと宛先を Code RAM の同一ワードに記述することができる。

バッファメモリは LUT を利用した分散メモリによって実装されており、16 ワード分の容量を持つ。このメモリは書き込みポートと読み出しポートを別々に持っており、Transceiver 内部では書き込みポートが Solver 側、読み出しポートがクロスバ側に接続される。Solver から読み出されたデータはそのままクロスバに渡されるが、同時にバッファメモリに書き込むことも可能になっている。Code RAM のビット 23 がクロスバへ出力するデータの選択信号になっており、これを 1 にすることでバッファメモリの出力をクロスバの入力に接続することができる。バッファメモリの読み出し時遅延も 1 クロックであるため、バッファメモリ中の値を送信する場合は、バッファメモリのアドレスとクロスバに与える宛先を同一ワードに記述する。

なお、積分モジュールと異なり、スイッチには Phase 1/2 の区別はない。

制御命令は表 B.9 に示す通りである。

表 B.9　Transceiver Code RAM の制御命令コード

[13:10]	名称	動作
0x00	END	1 時間刻み分の処理の終了を示す
0x01	NOP	[9:0] で指定したクロックサイクル数の間、Code RAM からのフェッチ動作を停止

表 B.10　Transceiver の Solver/クロスバインタフェイス信号

信号	幅	方向	機能
A	10	out	Solver 内のメモリのアドレス
D	36	out	Solver 内のメモリに書き込むデータ
Q	36	in	Solver 内のメモリから読み出されたデータ
WE	1	out	Solver 内のメモリの書き込み許可信号
IN	36	in	クロスバからの受信ポート (クロスバの OUT に接続)
OUT	36	out	クロスバへの送信ポート (クロスバの IN に接続)
DST	14	out	データの送信先 (クロスバの DST に接続)

B.3.2　Solver およびクロスバへのインタフェイス

表 B.10 は Transceiver の Solver およびクロスバへのインタフェイス部の信号を示す。これらの信号はそれぞれ、B.1.5 節 (積分モジュールの外部インタフェイス) および B.2 節 (クロスバ) で述べられてるものに対応する。

B.3.3　初期化インタフェイス

シミュレーションを行うには積分モジュールの Pathway RAM をはじめとするメモリを初期化するのと同様に、Transceiver 内部の Code RAM を初期化する必要があるため、Transceiver にも Code RAM にデータを書き込むための初期化インタフェイスが備えられており、表 B.11 が初期化インタフェイスの信号一覧である。このインタフェイスは積分モジュールの DII と同様に書き込み専用であり、リセット信号が High になっている間アクティブである。

表 B.11　Transceiver の初期化インタフェイス信号

信号	幅	方向	機能
DII_A	11	in	Code RAM の書き込み先アドレス
DII_D	36	in	Code RAM に書き込むデータ
DII_WE	1	in	Code RAM の書き込み許可

付 録C　浮動小数点フォーマット

ここでは、現在 ReCSiP の実装に用いられている浮動小数点演算器が扱う浮動小数点形式変数の表現形式について述べる。この形式は、IEEE-754 に定められている単精度浮動小数点形式に基づいたものである。

C.1　IEEE-754形式

IEEE-754 に定められている、単精度浮動小数点形式は図 C.1 (a) に示すように、1 ビットの符号、8 ビットの指数部と 23 ビットの仮数部の合計 32 ビットから構成されるフォーマットである。仮数部には 127 のオフセットが加算されているため、符号を s、仮数部を e、指数部を m とすると、この 32 ビットで表される実数は

$$x = (-1)^s \times 2^{e-127} \times m$$

となる。仮数部の最上位ビットは常に 1 であることになっており、この 1 は省略される。

浮動小数点演算の結果、仮数部がシフトされて右側にビットが押し出される場合があり、この場合には仮数部が 23 ビットに収まるように近似値を求める丸め処理を行う必要があり、このために Guard、Round、Sticky の 3 つの追加ビットが定義されている。これらのビットの初期値は 0 で、Guard と Round にはそれぞれ 23 ビットからはみ出した 1 ビット目、2 ビット目のビットが、Sticky ビットには Round ビットよりも右側のビットすべての論理和をとったものが格納される。

この 3 つのビットを使った丸めモードは表 C.1 のように 4 つ定義されており、演算処理のあとで指定された丸めモードにしたがって追加ビットが丸められ、繰り上げが発生した場合は仮数部に加算が行われる。

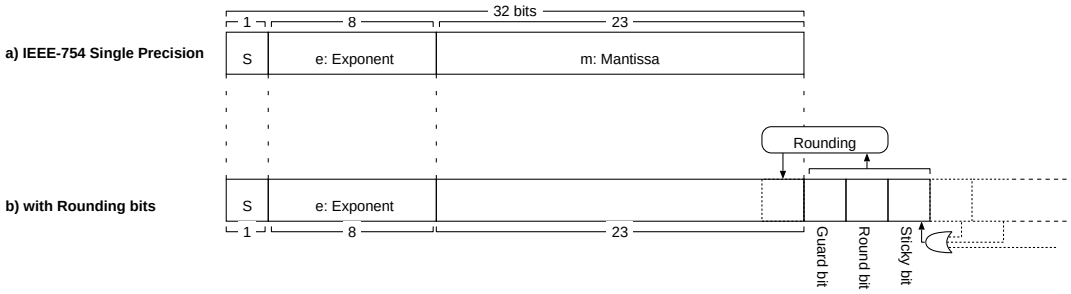


図 C.1　IEEE754 標準フォーマット (単精度) と丸め操作

表 C.1　IEEE-754 に定められている丸め処理

丸めモード	動作
RN: Round to Nearest	表現可能な最も近い値に丸める
RZ: Round toward Zero	0 の方向へ丸める
RP: Round upward	+ inf 方向へ丸める
RM: Round downward	− inf 方向へ丸める

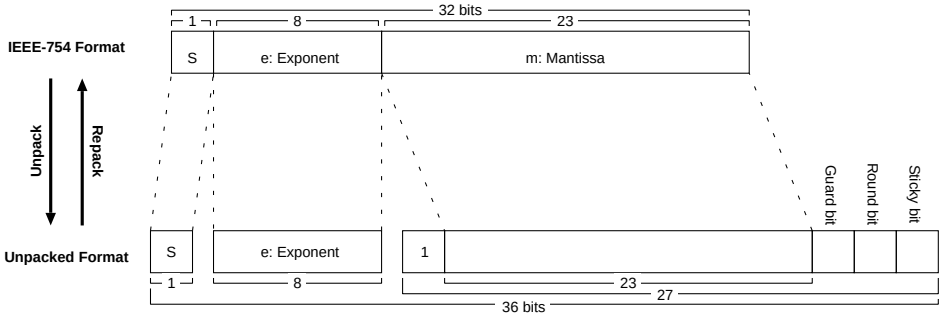


図 C.2　Unpack/Repack 操作

C.2 Unpacked 形式

浮動小数点演算を繰り返すと、丸め操作による桁落ちの繰り返しにより誤差が蓄積する場合がある。これを避けるためには仮数部の桁数を大きくとることが対策になりうるが、これはメモリ消費量の増加とともに、演算器の規模に影響する。そこで ReCSiP では、32bit 幅のデータに ECC の 4bit が付加されたメモリで用いることのできる 36bit のデータ幅を用いている。単体の SRAM や、FPGA 内部のメモリなど多くのメモリが ECC の利用を考慮して 18bit あるいは 36bit のデータ幅を持っており、36bit 幅の変数は利用しやすいものとなっている。

図 C.2 に、IEEE-754 形式と ReCSiP で用いている独自の Unpacked 形式の関係を示す。Unpacked 形式は IEEE-754 形式の仮数部を 4 ビット拡張し、1 ビットは暗黙のうちに 1 とされている仮数部の最上位ビットを明示的にするために、残りの 3 ビットは Guard、Round、Sticky ビットをそのまま保持するために、それぞれ用いられる。

ReCSiP で用いる浮動小数点演算器はこの Unpacked 形式を入出力ともに用い、メモリ上でもこの形式で値を保持する。ホスト側では IEEE-754 形式で値を取り扱うため、FPGA 上のローカルバスインタフェイス内部で形式の変換を行う。IEEE-754 形式から Unpacked 形式への変換操作を unpack、その逆を repack と呼び、後者で丸めを行う。FPGA 内部では丸め処理を最後まで行わないようにすることで、拡張された仮数部の効果を最大限に活かすことができる。

付 録D　ベンチマークに用いたモデルのSBML記述

D.1　Minimal Mitotic Oscillator モデル

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by SBML API 2.0(a17.0) -->
<sbml xmlns="http://www.sbml.org/sbml/level1"
      xmlns:celldesigner="http://www.sbml.org/2001/ns/celldesigner"
      level="1" version="2">
  <model name="Goldbeter">
    <listOfCompartments>
      <compartment name="default"/>
      <compartment name="cytoplasm" outside="default" volume="1.0"/>
    </listOfCompartments>
    <listOfSpecies>
      <species boundaryCondition="true" compartment="cytoplasm"
               initialAmount="0.0" name="EmptySet"/>
      <species compartment="cytoplasm" initialAmount="0.0" name="C"/>
      <species compartment="cytoplasm" initialAmount="0.0" name="M"/>
      <species compartment="cytoplasm" initialAmount="0.0" name="X"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction name="Reaction1" reversible="false">
        <listOfReactants>
          <speciesReference species="EmptySet"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="C"/>
        </listOfProducts>
        <kineticLaw formula="vi">
          <listOfParameters>
            <parameter name="vi" value="0.023"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
      <reaction name="Reaction2" reversible="false">
        <listOfReactants>
          <speciesReference species="C"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="EmptySet"/>
        </listOfProducts>
        <kineticLaw formula="mass(C,kd)">
          <listOfParameters>
            <parameter name="kd" value="0.00333"/>
          </listOfParameters>
```

```
      </kineticLaw>
    </reaction>
    <reaction name="Reaction3" reversible="false">
      <listOfReactants>
        <speciesReference species="C"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="EmptySet"/>
      </listOfProducts>
      <kineticLaw formula="uui(C, X*vd, kd)">
        <listOfParameters>
          <parameter name="kd" value="0.00333"/>
          <parameter name="vd" value="0.1"/>
        </listOfParameters>
      </kineticLaw>
    </reaction>
    <reaction name="Reaction4" reversible="false">
      <listOfReactants>
        <speciesReference species="EmptySet"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="M"/>
      </listOfProducts>
      <kineticLaw formula="(1-M)*VM1*C/(1+K1-M)*(Kc+C)">
        <listOfParameters>
          <parameter name="K1" value="0.1"/>
          <parameter name="Kc" value="0.3"/>
          <parameter name="VM1" value="0.5"/>
        </listOfParameters>
      </kineticLaw>
    </reaction>
    <reaction name="Reaction5" reversible="false">
      <listOfReactants>
        <speciesReference species="EmptySet"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="X"/>
      </listOfProducts>
      <kineticLaw formula="uui((1-X), M*VM3, K3)">
        <listOfParameters>
          <parameter name="K3" value="0.1"/>
          <parameter name="VM3" value="0.2"/>
        </listOfParameters>
      </kineticLaw>
    </reaction>
    <reaction name="Reaction6" reversible="false">
      <listOfReactants>
        <speciesReference species="M"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="EmptySet"/>
      </listOfProducts>
      <kineticLaw formula="uui(M,V2,K2)">
        <listOfParameters>
          <parameter name="K2" value="0.1"/>
```

```
            <parameter name="V2" value="0.167"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
      <reaction name="Reaction7" reversible="false">
        <listOfReactants>
          <speciesReference species="X"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="EmptySet"/>
        </listOfProducts>
        <kineticLaw formula="uui(X,V4,K4)">
          <listOfParameters>
            <parameter name="K4" value="0.1"/>
            <parameter name="V4" value="0.1"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
    </listOfReactions>
  </model>
</sbml>
```