Vol.2015-SLDM-173 No.3
2015/12/1

高位合成による自動パイプライン化を利用したスパイキングニューラルネットワークシミュレーション高速化回路のFPGA実装

川尾 太郎[1,a)]　河野 崇[2,b)]　藤田 昌宏[3,c)]

概要：応用における必要性から、数万ニューロンの実時間シミュレーションを行うための高速化が求められている。本研究では、高位合成ツールを利用し多段パイプライン回路を FPGA 上に実装した。この回路では 256 ニューロンが実時間比 280 倍で動作する。複数チップの接続も考慮して設計されており、7 万ニューロンを実時間シミュレーションできると考えられている。

キーワード：スパイキングニューラルネットワーク，FPGA，高位合成

FPGA Implementation of a Accelerator for Spiking Neural Network Simulation using a High Level Synthesis Tool

Abstract: Realtime simulation of the network with ten thousand or more neurons is required for its application. This paper shows our highly pipelined circuit implemented on a FPGA using a high level synthesis tool. Our accelerator allows 256 neurons to work 280 times faster than real time. It is designed for multi-FPGA implementation and is estimated to execute realtime simulation of 50,000 neurons.

Keywords: Spiking Neural Network, FPGA, High Level Synthesis

1. はじめに

1.1 FPGA を用いた計算高速化

科学技術計算や画像映像処理には実行に時間がかかるものがあり、専用ハードウェアを用いた高速な計算が求められている。専用ハードウェアを用いた計算では、目的とする計算に最適化された演算器を高い並列度で実装することで、計算速度向上が実現される。また、ソフトウェアで実行した場合よりも消費エネルギーが少ないという利点もある。専用ハードウェアの代表例である Application Specific IC (ASIC) は、高い処理能力と電力効率を有する代わりに、開発の手間やコストが莫大であるという問題点を抱えている。一方で、書き換え可能な回路である Field Programmable Gate Array (FPGA) を用いると、チップ製造後に生じる設計変更やバグ修正を原因とする再製造を行う必要はなくなるなど、設計上の困難がいくらか緩和される。FPGA は半導体微細化技術発展により高性能化と大規模化が進んでおり、FPGA を用いた専用回路による計算高速化に注目が集まるようになっている。専用ハードウェアを使わない演算高速化手法としては、General-Purpose computing on GPUs (GPGPU) と呼ばれる、GPU を使う手法などがある。

FPGA 上に構成する回路は RTL 記述で設計するのが一般的である。RTL 設計は一般的なソフトウェア開発と比べて記述量が多く、また、ハードウェアに関する知識を設計者に要求する。これらの問題の解決策の 1 つが高位合成である。高位合成とは、C/C++、Java、それらを拡張した

[1] 東京大学大学院工学系研究科電気系工学専攻
Dept. of Electrical Engineering and Information Systems, The University of Tokyo
[2] 東京大学生産技術研究所
Institute of Industrial Science, The University of Tokyo
[3] 東京大学大規模集積システム設計教育研究センター
VLSI Design and Education Center, The University of Tokyo
[a)] kawao@cad.t.u-tokyo.ac.jp
[b)] kohno@sat.t.u-tokyo.ac.jp
[c)] fujita@ee.t.u-tokyo.ac.jp

言語等の、より抽象的かつ記述量の少ない設計記述から、RTL 設計を得ることである。また、高位合成ツールはパイプライン化やリソースの共有といった種々の最適化をある程度自動で行うことができるため、開発期間とコストの軽減が期待できる。

例えば、津波シミュレーションの高速化研究 [1] では、演算を FPGA 実装することで、ソフトウェア実装に対し約 40 倍の高速化を実現した。

1.2 ニューラルネットワーク

ニューラルネットワークとは、ニューロンとシナプスの機能を数学的に表現したモデルである。ニューロンに相当するノードと、シナプスに相当するエッジで構成される。シナプスの結合強度を変化させること (学習) により、目的とする処理が行えるようになる。曖昧さや多くの雑音を含むデータの処理ができるという特徴があり、パターン認識やデータ分類など、多くの応用で実用されている。

本研究では、ニューロンの振る舞いを高い正確性を持って再現しつつ、式の複雑性を抑えることでハードウェア実装がしやすいように作られている DSSN モデル [2][3] に基づいたニューラルネットワークを FPGA 上に実装した。同モデルを FPGA 実装した研究 [4][5] と同様のシナプスモデルと学習アルゴリズムを用いている。

1.3 本研究の目的

DSSN モデルのニューラルネットワークシミュレーションを実時間よりも高速に行えるのは、文献 [4] における FPGA 実装では、ニューロンが 1500 個程度の時までである。その実装に学習機能を付け加えた新しい実装 [5] では計算に要する時間はさらに長くなった。扱うことのできるニューロンの個数は応用上重要である。一例として、数万ニューロンを扱うことができれば、脳の一機能がシミュレーション可能になる。本研究では、数万個のニューロンが実時間よりも高速に動作できるようになることを目標とする。そのために、パイプライン化やデータフロー最適化等の手法を用いて文献 [5] で実装されたニューラルネットワークのシミュレーションと学習の高速化する。また、規模大きいネットワークを扱うために必要な、回路大型化やチップ接続に伴う最適化効果の変化等の測定も行う。

2. ニューラルネットワークのモデル

本研究で実装するニューラルネットワークは、**図 1** に示すニューロンが互いにつながり合った構造をしている。シナプスからニューロンへの入力が式 (5)　、ニューロン内部での処理が式 (1–3)、ニューロンからシナプスへの出力が式 (4)　に対応する。ニューラルネットワークの学習が式 (6–7) で表される。これらの式のうち、微分方程式で記述されていたものは、回路実装のために差分方程式化した。

2.1 ニューロンのモデル

本研究では、以下に示す DSSN モデル [2][3] に基づいたニューロンを実装する。スパイクとは活動電位のことで、イオンチャネルの働きによる膜電位の上昇を指す。v, n, q は抽象的な変数である。v が膜電位に相当する変数で、v と n でスパイクを作り、q でその頻度を調整する。

I_0 はバイアス定数、I_{stim} は postsynaptic 電流 (シナプス後電流：ニューロンへの入力電流) を重み付けした和であり、第 2.2 節で述べる式 (5)　で表される。この 2 つの和がニューロンへの刺激となり、それが十分に大きいとスパイクを引き起こす。

φ と τ は時定数であり、その他のパラメータ $v_0, \alpha, a_n, a_p, b_n, b_p, c_n, c_p, k_n, k_p, l_n, l_p, m_n, m_p$ は、ニューロンの動作を調整するためのものである。

入力は I_{stim}、出力は v である。

$$v(t+\Delta t) = v(t) + \Delta t \frac{\varphi}{\tau} \times \begin{cases} \left(a_n(v-b_n)^2 + c_n - n - q + I_0 + I_{stim}\right) & (v < 0) \\ \left(a_p(v-b_p)^2 + c_p - n - q + I_0 + I_{stim}\right) & (v \geq 0) \end{cases} \tag{1}$$

$$n(t+\Delta t) = n(t) + \Delta t \frac{1}{\tau} \times \begin{cases} \left(k_n(v-l_n)^2 + m_n - n\right) & (v < r) \\ \left(k_p(v-l_p)^2 + m_p - n\right) & (v \geq r) \end{cases} \tag{2}$$

$$q(t+\Delta t) = q(t) + \Delta t \frac{\varepsilon}{\tau}\left(v - v_0 - \alpha q\right) \tag{3}$$

2.2 シナプスのモデル

シナプスのモデルは文献 [6] のものを採用した。このモデルは式 (4)　と式 (5)　で表される。

$$I_s(t+dt) = I_s + \Delta t \times \begin{cases} \alpha\left(1 - I_s(t)\right) & ([T] = 1) \\ -\beta I_s(t) & ([T] = 0) \end{cases} \tag{4}$$

式 (4)　は、ニューロンから v を入力され、シナプスに I_s を出力することを表している。I_s は postsynaptic 電流を、$[T]$ は presynaptic 電位 (シナプス前電位：ニューロンからの出力電位) により定まり、伝達物質の放出量を表す。既存研究 [4][5] 及び本研究では、電圧閾値 (0 とする) を超えるか否かで、1 か 0 の値を取るものとしている。α と β は定数である。

式 (5)　は、ニューロンがシナプスを介して他のニューロンから受ける影響を定式化したものである。シナプスから I_s を入力され、ニューロンに I_{stim} を出力することを表している。W は 1.2 節で述べたシナプスの結合強度のことで、W_{ij} は i 番目のニューロンが j 番目のニューロンから受ける影響の大きさを表す。c は I_{stim} の大きさを調整す

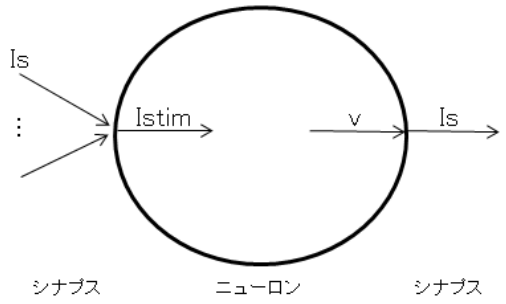

図 1　ニューロンの概要

るパラメータである。

$$I^i_{stim} = c\sum_{j=1}^{N} W_{ij} I^j_s \tag{5}$$

2.3 Hebbian learning

Hebbian learning（ヘブ学習）とは、スパイクのタイミングに基づいた学習である。ニューロン i の発火をニューロン j が引き起こす時、ニューロン j がニューロン i に与える影響 W_{ij} は強化される。このことを表した式が、

$$\Delta W = A_+ \exp\left(\frac{-|\Delta t|}{\tau_+}\right) \tag{6}$$

である。この式に、減衰項を追加すると、

$$\Delta W = A_+ \exp\left(\frac{-|\Delta t|}{\tau_+}\right) - A_- \exp\left(\frac{-|\Delta t|}{\tau_-}\right) \tag{7}$$

となる。$|\Delta t|$ はスパイクのタイミングの違いを表す変数で、シミュレーションにおける離散化された時間の間隔を表す Δt とは異なる。τ は時定数で、A は定数である。

3. 従来の実装

既存研究 [5] における演算の FPGA 実装について説明する。この回路は**図 2** で示すような構造をしており、**図 3** のようなパイプライン処理を行う。使用した FPGA は Virtex 6 XC6VSX315T である。

各パラメータの値は**表 2** のものが用いられ、一部乗算がシフト演算へと置き換えられた結果、実際に行われる乗算は式 (1–4) 中に現れる乗算数より少なくなった。

指数関数演算を実装するとハードウェアを大量に使用してしまうので、指数関数の計算は $|\Delta t|$（スパイク間隔）を入力とする LUT で実現されている。

Accumulator Unit が平均して 4 サイクルに 1 回しかデータを出力しない。図 3 に示すように、DSSN Unit と Silicon Synapse Unit で演算が行われるサイクル数は、シミュレーション 1 ステップのサイクル数の約 3 %である。それにも関わらず、Accumulator Unit と同じ並列度で他の 2 つのユニットを用意しないといけない理由は、Accumulator Unit の出力が、一時期に集中しているからである。

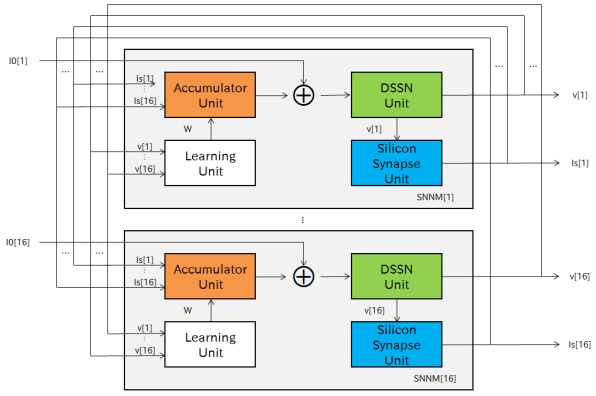

図 2　従来実装 [5] の回路構造

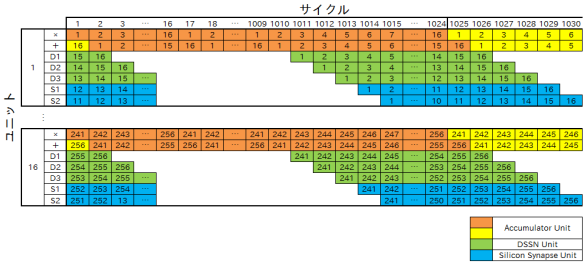

図 3　従来実装 [5] のパイプライン

モジュールが全結合するという構造のため、モジュール間配線がモジュール数の 2 乗に比例し、並列度向上の妨げになっている。また、複数チップを組み合わせるために、配線数の制約を満たしつつ回路全体を複数に分割するのも難しい。

4. 本研究における実装

4.1 FPGA システムの概要

本研究で使用した FPGA システムの概要を**図 4** に示した。FPGA ボードは PCI Express でホストコンピュータに接続されている。ホストコンピュータは Xeon X5660(6 コア、12 スレッド、2.66GHz) と 48GB のメモリを搭載している。FPGA ボードは、Virtex 6 XC6VSX475T と 24GB のメモリを搭載している。

まずホストコンピュータでホストプログラムを起動し、設計データを FPGA に送った後で、入出力データのやり取りを行いながら FPGA 内で演算を行うというのが、処理の流れである。ホストプログラムを起動すると、後の手順は自動で行われる。ホストプログラムは C、FPGA は Java 文法によるデータフローグラフで記述する。

データフローグラフは Maxeler Technologies の高位合成ツール、MaxCompiler [7] で VHDL による記述に変換され、続いて Xilinx のツール群 [8] によって FPGA の設計データが生成される。データフローグラフを RTL 記述に変換する際、コンパイラが自動的に演算器の割り当てや演算段数の設定を行うので、Java 記述の大部分は、プログラ

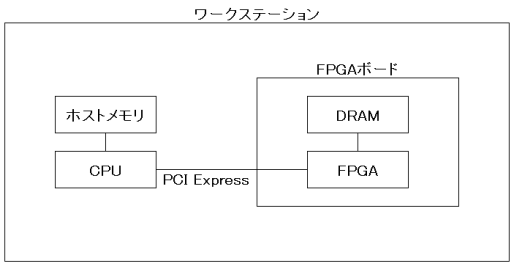

図 4　FPGA システムの概要

ムのように式をそのまま記述したもので良い。

4.2　概要

本研究では、3 変数 DSSN モデル [3] のニューラルネットワークに、従来実装 [5] と同様の学習機能を持たせたものを実装した。各パラメータは表 2 の値にした。

回路の入力は I_s, W の初期値と制御信号で、出力は更新された I_s である。1 から $N(N+12)$ サイクルの間に W が、$N(N-1)+1$ から $N(N+12)$ サイクルの間に $I_s(0)$ が入力される。制御信号は毎サイクル入力され、初期値選択に利用される。v, n, q の初期値は、刺激の入力から十分時間が経った時の値を用いる。差分方程式ユニットの出力はマルチプレクサとつながっており、初期値とユニットの出力の選択が行われる。

W を固定した状態でシミュレーション（式 (1–5) の計算）を 64 ステップ（時刻 $t=0$ から $t=64\Delta t$）行い、その結果を利用し、2 ステップ分の時間で学習を行う。この手順を繰り返し、W の値が一定時間変更されなくなるまで続ける。ニューロン数を N とすると、この回路は 1 ステップを $N+12$ サイクルで処理する。定数部分はパラメータの値で変化することがあるが、以後の説明は 12 として進める。

実装にあたり、回路を式 (5)　の加重平均ユニット・式 (1–4) の差分方程式ユニット・学習部・記憶部の 4 つに分割することにした。

演算のデータフローを表したのが**図 5** である。加重平均ユニットの入力は、W_{ij} と I_s^j の計 $2N$ 個、出力は I_{stim}^i である。入力数が非常に多いので、帯域の都合から、それらの値は回路内の資源に保存される必要がある。また、W_{ij} は $N+12$ サイクルで周期的に呼び出されるので、W_{1j} から W_{Nj} を 1 つの BRAM に保存することができる。

差分方程式ユニットは v, n, q, I_s の 4 変数に対し演算を行う。その結果は BRAM に保存される。I_s と v は他の演算にも利用されるので、他の記憶部にも送られる。学習部とそれに使われる記憶部は、差分方程式ユニットが出力する v を見てスパイクが起きているか、カウンタの値からそれがいつかを判断し、スパイクが起きたタイミングを保存する。学習は、ニューロン i とニューロン j のスパイクタイミングを読み込み、その差の絶対値を引数とする LUT を使って実現されている。

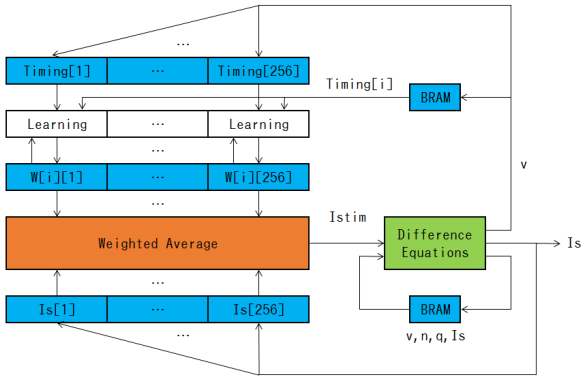

図 5　データフロー

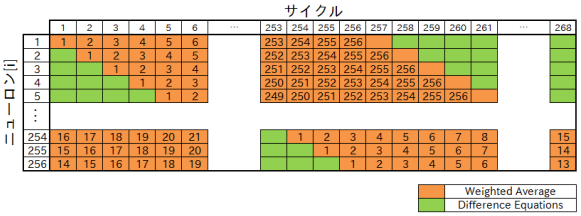

図 6　各サイクルにおける演算

4.3　加重平均の高速化

式 (5)　の計算が、全計算の大半を占めるので、この計算を高速に処理することを第一の目標とする。従来の実装 [5] では、この計算を行う Accumulator Unit が全体の処理速度を低下させていた。パイプライン構造 (図 3) と回路の並列度から、I_{stim} が平均して 4 サイクルに 1 回しか出てこないからである。

本実装では、演算の規模（ニューロン数）に応じて並列度を上げることが容易になるよう、演算器を直線的につなぎ**図 6** のようなパイプラインを処理をしている。I_{stim}^1 を $N+1$ サイクルかけて計算する。その後、11 サイクルかけて I_s^1 を更新し、メモリに書き込む。i が 1 つ大きくなるごとに、これを 1 サイクルずつずらして処理していく。

図を斜め方向に見て、更新対象としてではなく、他の I_s の更新に必要なデータとしての I_s^1 を考える。I_s^1 は、1 から N サイクル目には利用されるため、この間は書き換えてはいけない。この方式では、依存関係から来る書き換え不可能なタイミングを避けるように回路が動作していることがわかる。

演算器の接続が直線的になることと、配線が複雑になり過ぎないことに注意している。さらなる高速処理を目指し、演算器の接続を複雑にし高度な制御を行った処理方法では、ニューロン数（演算器数）が一定値を超えると、動作周波数が大きく低下してしまうということもあった。例外となるのは差分方程式ユニットから記憶部への出力であ

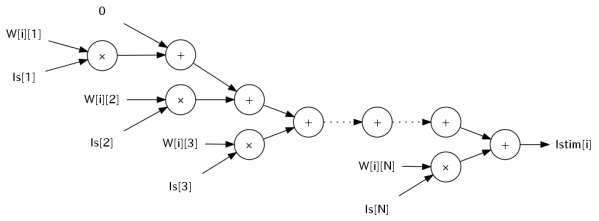

図 7 加重平均計算回路の構造

る。この部分は配線数が非常に多くなっているが、本研究の範囲内ではこれによる性能低下は起きていない。また、第 4.5 節でも述べるように扱えるニューロン数を制約する要因は BRAM の容量であるので、演算器数を演算規模に合わせて増やしても問題はない。

4.4 ビット幅の調整

参考とした論文 [3][5] 間で、時間間隔 Δt やビット幅に差がある。時間間隔 Δt の値は、シミュレーション速度の実時間比に直接影響する。使用するシステムでは、BRAM や DSP のビット幅に一定の単位があり、それを超えるとリソース使用量や演算段数の増加につながってしまう。このことは並列度低下や 1 ステップのサイクル数増による速度低下、扱えるニューロン数減少の原因となるため、変数のビット幅は重要である。

これらについて、システムと精度の要請を満たす値を選択するため、演算をソフトウェア実装した。このソフトウェアは C 言語で記述され、ソースファイル 3 つとヘッダファイル 5 つで構成されており、論理行数は約 500 である。Δt の値は 3/8000、v, n, q, I_{stim} は 18 ビット、I_s, W は 16 ビットで小数ビット数はどちらも 13 ビットとし、計算精度に関する問題がないことを確認した。

4.5 合成結果と動作結果

以上の内容を Java 文法によるデータフローグラフで記述したところ、演算部分と通信部分の記述ファイルを合わせて、論理行数は約 200 となった。この設計記述をターゲット周波数 200MHz で論理合成、配置配線し、**表 1** のような結果が得られた。

既存研究 [5] と比較するとサイクル数で 4 倍、周波数で 2 倍、計 8 倍の高速化を達成している。サイクル数が 4 分の 1 になったのは、大まかにいうと演算器数を 4 倍にしたからであるが、回路リソースを効率よく配分したため、リソース使用量は 4 倍よりも少なくできた。1 シミュレーションステップの計算に必要な時間と、時間間隔 Δt の比から、実時間の 280 倍高速なシミュレーションが可能である。

4.6 考察

ニューロン数を N とし、演算器の並列度を p とすると、実装した演算の計算量は、N の 2 乗に比例し、演算時間は N^2/p と表現できる。演算器の並列度を演算規模に比例して増やす（$N = p$ とする）ことができる構造を採用したため、処理に要する時間は N に比例するようになっている。動作周波数が回路規模で変化しないことと、リソースに制約がないことを仮定すると、256 ニューロンが実時間比 280 倍高速なので、7 万ニューロンを実時間で扱うことができる。回路の動作周波数が直線的に低下すると仮定すれば、5 万ニューロン程度という計算になる。

表 1 リソース使用量と動作速度

N	256	512	768	総量
LUTs	19,515	58,298	109,443	297,600
FFs	39,934	101,215	168,136	297,600
DSPs	268	524	780	2,016
BRAMs	522	1,034	1,767	2,128

実際には回路資源は有限であり、表 1 が示すように W を保存するのに使う BRAM の容量が制約となり、1FPGA で扱うことのできる最大のニューロン数は 768 程度である。メモリを増やす方法としては、FPGA を複数個利用する方法と、ボートの DRAM（図 4）を利用する方法が挙げられる。前者は必要なメモリが N^2 で増加し、かつチップ当たりのニューロン数（演算器数）が減少するため、それに適した FPGA を用意しないと非効率的である。後者の DRAM は BRAM に対して帯域が狭い。160MHz で 2 バイトのデータを毎サイクル 96 個ずつ送る回路が、用いたシステムで実現できる目安である。この場合、演算器の不足を演算時間で補うため、演算時間は長くなる。何らかの方法で並列度を上げて速度向上を図る必要がある。

いずれにしても、複数の FPGA を活用する必要が出てくる。本研究で用いたシステムにはボード間通信機能があり、200MHz 付近で動作する回路ならば毎サイクル数バイトのデータを送ることができる。本章で示した実装は、データフローが直線的であり、分割に適している。例外は差分方程式ユニットから出力される一部のデータで、多くのメモリにデータを分配している箇所がある。単純な分け方として図 5 を左右に分割して場合、I_s の出力がチップ間をまたぐ場合とそうでない場合に分かれ、演算の規則性がなくなってしまう。また、やりとりするデータ数が増えると現在使用しているシステムの通信の制約を満たせなくなる可能性がある。

FPGA ボードを 2 つ使う実装向けに、データフローを次のように分割する。ここでは $N = 2M$ とする。ボード 1 には $v^i, n^i, q^i, I_s^i (1 \le i \le M)$ と $W_{ij} (1 \le j \le M)$ を保存する。ボード 2 には $v^i, n^i, q^i, I_s^i (M+1 \le i \le 2M)$ と $W_{ij} (M+1 \le j \le 2M)$ を保存する。v, n, q, I_s の更新（式 (1–4) の計算）はそれぞれの値が保存されているボード内で行う。加重平均の計算は式 (8)　のようにボード内の変数のみで計算できる部分和を足したものとし、ボード間通信で渡すのはその部分和のみとする。I_s^i の更新に必要な変

表 2　パラメータの値

パラメータ	値	パラメータ	値
a_n	8.0	a_p	8.0
b_n	0.25	b_p	0.25
c_n	0.5	c_p	0.5
k_n	2.0	k_p	16.0
p_n	$-2^{-2}-2^{-4}$	p_p	$2^{-5}-2^{-2}$
q_n	-0.705795601	q_p	-0.6875
φ	1.0	τ	0.003
r	-0.205357142	I_0	-0.205
c	0.060546875	Δt	0.000375
A_+	2^{-6}	A_-	$2^{-7}+2^{-8}$
τ_+	11.25	τ_-	22.5

数はボード内にあり、かつ、その書き込み先もボード内となっている。。ボードが 3 つ以上ある場合、ボード間のリング状になり、通信は一方向に行うだけでよい。よって、全体の演算を複数に分割し、ボード間通信は 1 ボードあたり送受信 1 つずつとすることができる。

$$\sum_{j=1}^{2M} W_{ij} I_s^j = \underbrace{\sum_{j=1}^{M} W_{ij} I_s^j}_{\text{ボード 1}} + \underbrace{\sum_{j=M+1}^{2M} W_{ij} I_s^j}_{\text{ボード 2}} \qquad (8)$$

最後に、使用しているシステムが 4 ボードをリング状につなぐことができることを踏まえ、2 つのケースにおいて 4 ボードで可能なシミュレーションの規模と範囲を計算する。W を BRAM に保存すると 1512 ニューロンを 140 倍高速にシミュレーションできる。W を DRAN に保存し、1 ボードあたり演算器が 96 個あると、約 5000 ニューロンの実時間シミュレーションができる。

5. 結論

ニューロン数の多いネットワークの実時間シミュレーションを行うため、演算の規模に応じて並列度を上げることが容易な構造の回路を設計した。その結果、280 倍の速さで 256 ニューロンの挙動を再現することができた。用いた最適化手法がそのまま適用できるのならば、数万ニューロンのネットワークを実時間で扱うことができる。

その規模の演算を行うには、FPGA チップを複数使い、演算並列度向上や必要なメモリ容量の確保をする必要がある。FPGA 間の通信帯域は内部と比べて狭いが、その制限下でも演算を行うことができるような回路分割方法を考案した。実際には、メモリの容量や帯域の制約が厳しく、膨大な数のチップを用意しない限りは、規模拡大や速度向上の程度はあまり大きくないだろう。

今後の課題としては第 4.6 節で示した方式で、複数の FPGA を用いた実装をすることが挙げられる。加えて、そういった実装に適した FPGA システムやネットワークモデルについての検討・考察を行うことも求められる。

参考文献

[1] 谷田英生, 福井啓, 吉田浩章, 藤田昌宏, “FPGA と GPGPU を利用した計算高速化・効率化に関する研究,” 情報処理学会研究報告,2012.
[2] Kohno,T.,and Aihara, K.(2007) “Digital spiking silicon neuron: concept and behaviors in GJ-coupled network,” in Proceedings of International Symposium on Artificial Life and Robotics, Beppu, OS3-OS6.
[3] 小林航, “デジタル演算回路による 3 変数シリコンニューロンの設計,” 東京大学大学院工学系研究科修士論文, 2012.
[4] Jing Li, Yuichi Katori and Takashi Kohno, “An FPGA-based silicon neuronal network with selectable excitability silicon neurons,” in frontiers in NEUROSCIENCE, 2012, Volume 6, Article 183.
[5] Jing Li, Yuichi Katori and Takashi Kohno, “Hebbian Learning in FPGA Silicon Neuronal Network,” in Proceedings of the 1st IEEE/IIAE International Conference on Intelligent Systems and Image Processing 2013.
[6] Destexhe, A., Mainen, Z. F., and Sejnowski, T. J.(1998). “Kinetic models of synaptic transmission,” in Methods in Neuronal Modeling, eds C. Koch and I. Segev (Cambridge, FL:MIT Press),1-25.
[7] MaxCompiler | Maxeler Technologies, `http://www.maxeler.com/products/software/maxcompiler/`.
[8] ISE Design Suite, `http://japan.xilinx.com/products/design-tools/ise-design-suite/index.htm`.