一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2022-68,RECONF2022-91(2023-01)

一般化並列カウンタ木に基づく 2値化ニューラルネットワークの効率的FPGA実装

谷川　貴弘†　　野田　麦†　　石浦菜岐佐†

† 関西学院大学　〒669–1330 兵庫県三田市学園上ケ原 1

あらまし　2 値化ニューラルネットワーク (BNN) はニューロンの活性化値や入力の重みを 2 値に制限したものであり, コンパクトなハードウェア実装を可能にする. BNN のニューロンを組合わせ回路で実装する場合, ニューロン中のポップカウンタは Wallace 木で構成することにより遅延を抑制できるが, FPGA 実装においては, 3 入力の全加算器を 5 入力以上ある LUT にマッピングするとデバイスの利用効率が低下する. 本稿では, ポップカウンタを一般化並列カウンタ (GPC) に基づく加算木で構成することにより, 回路規模と遅延時間を削減する手法を提案する. また, ポップカウントと閾値の比較をポップカウンタからのキャリーの参照のみで実現することにより, さらに回路規模と遅延時間の削減を図る. 本手法に基づく BNN ニューロンを Xilinx Artix-7 をターゲットに実装した結果, ニューロンへの入力数が 256 のとき, 単純な論理合成に比べてスライス数を 6.3%, クリティカルパス遅延を 8.9% 削減することができた.
キーワード　2 値化ニューラルネットワーク, BNN, 一般化並列カウンタ, FPGA, コンプレッサツリー

Efficient FPGA Implementation of Binarized Neural Networks Based on Generalized Parallel Counter Tree

Takahiro TANIGAWA†, Mugi NODA†, and Nagisa ISHIURA†

† Kwansei Gakuin University,　1 Uegahara, Gakuen, Sanda, Hyogo, 669–1330, Japan

Abstract　Binarized neural networks (BNN) allow compact hardware implementation by binarizing weight values and neuron activations. The critical path delay of a combinational circuit implementing a BNN neuron may be curbed by adopting a Wallace tree of full-adders. However, in FPGA implementation, a 3-input full-adder does not make full use of LUTs of more than 5 inputs. This paper proposes the use of a GPC (generalized parallel counter) based compressor tree in FPGA implementation of a BNN neuron to reduce both the delay and size of the resulting circuit. We further enhance the efficiency of the circuit by reducing the comparison of the popcount and threshould into reference to the carry signal from the compressor tree. The critical path delay and the slice count of our BNN neuron, implemented on a Xilinx Artinx-7 FPGA, were smaller by 8.9% and 6.3%, respectively, compared to those of the circuit produced by simple logic synthesis, at number of inputs 256.
Key words　binarized neural network, BNN, generalized parallel counter, FPGA, compressor tree

1. はじめに

ニューラルネットワークは画像や音声の高い精度での認識を可能にしており, 近年広範な応用に用いられるようになっている.

ニューラルネットによる推論の計算量は大きいため, 携帯端末等 (エッジ) で認識処理が必要な場合, エッジ側で取得したデータをサーバー側に転送して処理を行うことが多い. しかし, 通信データ量やセキュリティの観点から, その処理をエッジ側へ移行させる試みがなされている. ハードウェア量や消費電力の制約の厳しい端末での処理を実現するための一アプローチとして, ニューラルネットによる推論処理のハードウェア実装の研究が行われている [1].

ニューラルネットのハードウェア実装においては, データを固定小数点化するとともに, そのビット数を削減してハードウェア量を抑制することが試みられる. 特に, 値を 2 値にまで制限したものは 2 値化ニューラルネットワーク (Binarized Neural Network; 以下 BNN) [2] と呼ばれる. BNN では積演算を論理ゲート 1 つで計算でき, ハードウェア量を大幅に削減できるため, 近年 BNN のハードウェア実装に関する研究がなされている [3].

ハードウェア実装における BNN ニューロンは, 積を計算す

る XNOR ゲート, その結果の 1 の数を数えるポップカウンタ (並列カウンタ) および閾値との比較器により構成できる. 組合わせ回路による実装を考えた場合, XNOR 演算は並列に行えるため, ポップカウンタおよび閾値との比較がクリティカルパス遅延のボトルネックとなる.

多入力の加算は, 全加算器で Wallace 木 [4] や Dadda 木 [5] を構成することにより, 遅延を抑制できる. しかし, ルックアップテーブル (LUT) 型の FPGA での実装を想定した場合, LUT の入力数は 5 ~ 7 程度であるため, 3 入力 2 出力の全加算器ではデバイスの利用効率に無駄が生じる. このため, 3 入力 2 出力を 6 入力 3 出力に拡大した加算器や, それをさらに拡張した一般化並列カウンタ (Generalized Parallel Counter; 以下 GPC) の利用が提案されている. GPC は, 並列カウンタにおいて, 重み 1 の入力だけでなく, 重み 2^k の入力を許容したものである. 文献 [6] [7] では乗算等における多入力の加算を対象に, FPGA の回路規模や遅延を最適化する GPC 加算木の構成法を提案している.

本稿では, BNN ニューロンのポップカウンタを GPC 木で構成することによって FPGA 実装における回路規模や遅延時間を削減する手法を提案する. GPC 加算木の構成は, 段数最小かつ回路規模最小のものを整数線形計画法により求める. また, 加算結果の閾値比較を最上位キャリーの参照のみに置き換えることにより, さらに回路規模と遅延時間の削減を図る.

本手法に基づく BNN ニューロンを Xilinx 社の FPGA Artix-7 をターゲットに実装した結果, 単純な論理合成で生成されるニューロンに比べて, ニューロンへの入力数が 256 のとき, スライス数を 6.3%, 遅延時間を 8.9% 削減できた.

2. 2 値化ニューラルネットワークの FPGA 実装

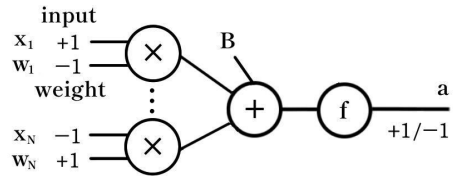
2.1 2 値化ニューラルネットワーク (BNN)

ニューラルネットワークのニューロンの活性化値 a は, 図 1 (a) に示すように, ニューロンの入力数を N, i 番目の入力を x_i, その重みを w_i, バイアスを B, 活性化関数を f とすると, $a = f(B + \sum_{i=1}^{N} x_i \cdot w_i)$ と表現できる. 2 値化ニューラルネットワーク (BNN) は, この x_i と w_i の値を $+1, -1$ に制限したものである [2]. $f(x)$ には, $0 \leq x$ ならば $f(x) = +1$, そうでなければ $f(x) = -1$ となるような f が用いられる.

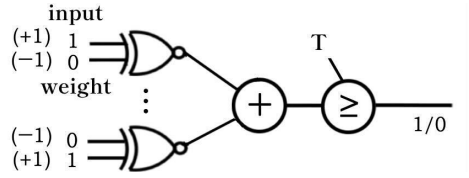
ここで, 値 $+1, -1$ をそれぞれ $0, 1$ で符号化すると, 入力と重みの乗算は排他的論理和否定 (XNOR) 演算に置き換えることができる. すると BNN のニューロンは 図 1 (b) に示すように, XNOR ゲートとその出力の 1 の数を求める加算器 (ポップカウンタ) および閾値 T との大小比較を行う回路で構成できる. 本稿では 図 1 (b) の回路を組合せ回路として FPGA 上に実装することを考える.

2.2 FPGA の LUT とキャリーチェーン

ルックアップテーブル型の FPGA は, 任意の論理関数をプログラムできる LUT (Look-Up Table), フリップフロップ, およびプログラム可能な配線により論理回路を実現する. 近年の FPGA の LUT の入力は 5~7 程度であり, メモリにより真理値表を記憶している. このため, k 入力 1 出力の LUT を, 入力が共通の $k-1$ 入力 2 出力の論理ゲートとして利用できる FPGA も存在する.



(a) BNN のニューロン



(b) 論理回路による実装

図 1: BNN のニューロンとその回路構成

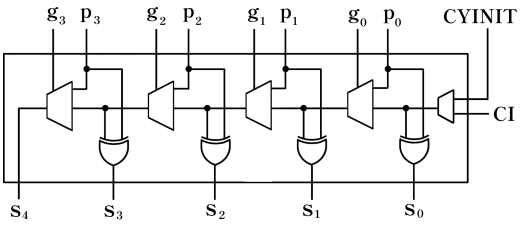


図 2: キャリーチェーンの回路構成例

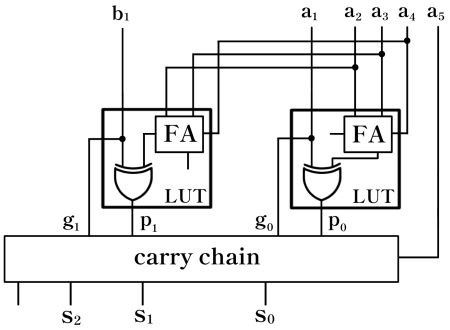


図 3: 一般化並列カウンタ GPC(1,5;3) の FPGA 実装例 [8]

また FPGA の中には加減算を効率よく実装するために, キャリーチェーンを回路として組み込んだものがある. キャリーチェーンは, 2 つの 2 進数の各桁のキャリー生成信号とキャリー伝播信号を入力とし, 2 つの 2 進数の和を計算する回路である. 4 桁のキャリーチェーンの回路例を図 2 に示す. 各桁のキャリー生成信号 g_i とキャリー伝播信号 p_i を入力とし, 各桁の和 s_i を出力する.

LUT, キャリーチェーン, およびフリップフロップにより構成される回路は FPGA 中のスライス (あるいはロジックブロック) と呼ばれるブロックにマッピングされる. スライスは内部に複数の LUT, キャリーチェーン, フリップフロップ, およびそれらの間の接続のための結線やマルチプレクサ等により構成される.

2.3 一般化並列カウンタ (GPC) に基づく加算木

多入力の加算の遅延を抑制する回路構成としては, 全加算器による Wallace 木 [4] や Dadda 木 [5] が知られている. しかし, LUT 型の FPGA での実装を想定した場合, 3 入力 2 出力の全加算器では一般的な LUT の入力数 (5~7) を必ずしも活かしきれない. このため, 6 入力 3 出力に拡張した加算器や, それ

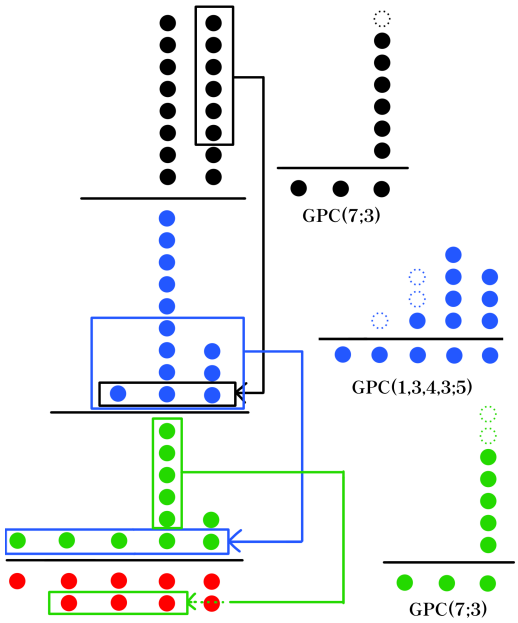


図 4: GPC に基づく加算木

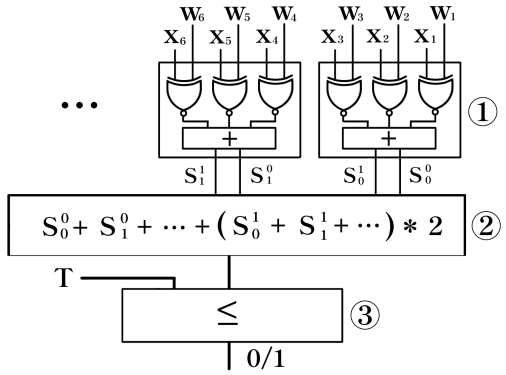


図 5: FPGA 実装における BNN ニューロンの構成

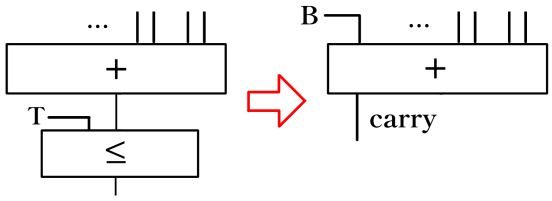


図 6: 閾値比較のキャリー参照への置換

をさらに拡張した一般化並列カウンタ (Generalized Parallel Counter; GPC) の利用が提案されている [6] [7] [8] [9] [10].

全加算器は 1 桁の入力を想定しているのに対し, GPC は重み 1 だけでなく, 重み 2^k の入力を許容して入力の和を求める並列カウンタである. 例えば, GPC(1,5;3) は重み 2 の 1 ビットと重み 1 の 5 ビットの加算結果を 3 ビットで出力するものである. 重み 2 のビットを b_1, 重み 1 のビットを a_1, a_2, a_3, a_4, a_5, 出力を s_2, s_1, s_0 とすると, GPC(1,5;3) は 2 つの LUT とキャリーチェーンを用いて図 3 のように構成できる.

GPC に基づく加算木の構成例を図 4 に示す. ドットは 1 つのビットを表しており, 長方形で囲ったドットが GPC の入出力である. 図では, GPC(7;3), GPC(1,3,4,3;5) の 2 種で入力ビットを 2 つの 2 進数に削減している. その後, 後述の row adder を用いて最終的な和が計算される.

文献 [6] [7] 等では, 2 進数の乗算等における多入力の加算を対象に, スライス効率のよい GPC と, それらを用いて最適な加算木を構成する手法を提案している.

3. 一般化並列カウンタ (GPC) を用いた BNN の効率的 FPGA 実装

3.1 概　　要

本稿では, BNN ニューロンのポップカウンタを GPC 木によって構成することにより, FPGA 実装における回路規模や遅延時間を削減する手法を提案する.

本稿で想定する BNN ニューロンの回路構成を図 5 に示す. まず, LUT を用いて入力と重みの XNOR 演算を k 対まとめて行い (①, 図では k=3), その和を 2 進数 s^0, s^1, …として出力する. 得られた出力の総和を求め (②), 和が閾値 T 以上であれば 1, そうでなければ 0 を出力する (③).

本稿では, ②の加算木を単純に論理合成するのではなく, GPC 木で構成することによって, 回路規模とクリティカルパス遅延の削減を図る. また, GPC 木により計算される和と閾値 T とを比較するのではなく, 図 6 に示すように, 加算木からのキャリー出力を参照するだけで比較結果を求めることにより, さらに回路規模やクリティカルパス遅延の削減を図る.

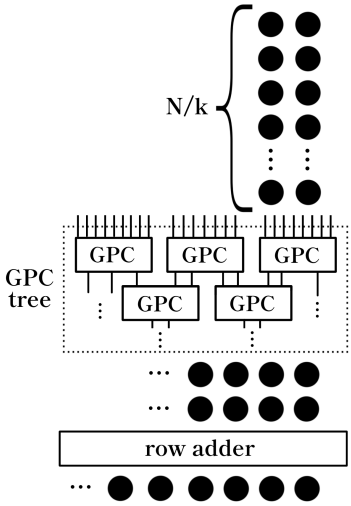


図 7: GPC 加算木の入出力

3.2 ポップカウンタの GPC 木による実装

本稿で提案する GPC に基づく加算木の構成を図 7 に示す. 加算木への入力ビットは N/k 個 (図では k=3) の 2 進数であり, これを GPC 木により 2 つの 2 進数に削減した後, それを加算して最終的な和を求める.

2 つの 2 進数の和を求める加算器は row adder と呼ばれる. これは FPGA のキャリーチェーンを用いれば, 回路規模と遅延時間の少ないものが構成できる. 例えば図 8 は row adder を 4 桁のキャリーチェーンを用いて構成した例である. 加算される 2 つの 2 進数の i 桁目を a_i, b_i とすると, キャリーチェーンへの入力は $g_i = a_i \cdot b_i$, $p_i = a_i \oplus b_i$ であり, これは 2 入力 2 出力 LUT 1 つで計算できる. 図ではキャリーチェーンを直列に連結しているが, 桁数が多い場合にはキャリーチェーンを木状に接続すればよい.

GPC 加算木は, 文献 [6] [7] [8] [9] [10] 等により示された GPC

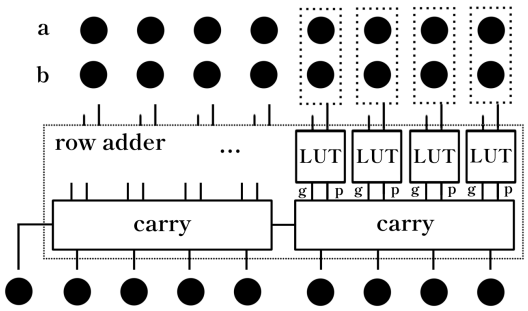


図 8: row adder の構成

```
1  module gpc15_3(input wire[4:0]s0, input wire[0:0]s1, output
       wire[2:0]dst);
2    assign dst = s0[0] + s0[1] + s0[2] + s0[3] + s0[4] + s1[0]*2;
3  endmodule
```

(a) 単純な記述

```
1  module gpc15_3(input wire[4:0]s0, input wire[0:0]s1, output
       wire[2:0]dst);
2    wire[3:0] genes;
3    wire[3:0] props;
4    wire[3:0] sum;
5    wire[3:0] carry;
6    LUT5 #( .INIT(32'h69966996) ) lut0_0_inst (
7      .I0(s0[0]), .I1(s0[1]), .I2(s0[2]), .I3(s0[3]),
8      .O(props[0])
9    );
10   LUT5 #( .INIT(32'hff00ff00) ) lut0_1_inst (
11     .I0(s0[0]), .I1(s0[1]), .I2(s0[2]), .I3(s0[3]),
12     .O(genes[0])
13   );
14   LUT5 #( .INIT(32'h17e817e8) ) lut1_0_inst (
15     .I0(s0[0]), .I1(s0[1]), .I2(s0[2]), .I3(s1[0]),
16     .O(props[1])
17   );
18   LUT5 #( .INIT(32'hff00ff00) ) lut1_1_inst (
19     .I0(s0[0]), .I1(s0[1]), .I2(s0[2]), .I3(s1[0]),
20     .O(genes[1])
21   );
22   CARRY4 CARRY4_inst (
23     .CO(carry), .O(sum),
24     .CI(0), .CYINIT(s0[4]),
25     .DI(genes), .S(props)
26   );
27   assign dst[0] = sum[0];
28   assign dst[1] = sum[1];
29   assign dst[2] = carry[1];
30 endmodule
```

(b) スライスへのマッピングを指定した記述

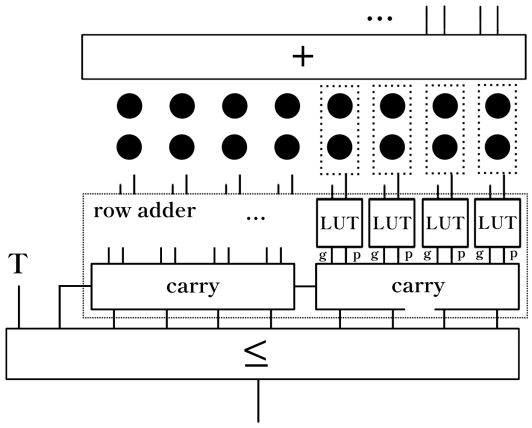
図 9: GPC(1,5;3) の設計記述例

で構成する．本稿では BNN ニューロンを組合せ回路として実装することを想定しているので, 段数が最小となる加算木の構成のうち, スライス数が最小になるものを求める．これは最適化問題を整数線形計画法として定式化し, それをソルバーで解くことにより求める．

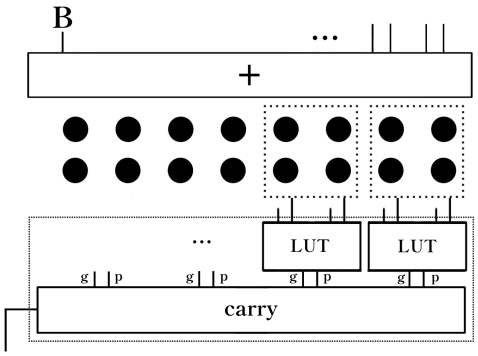
GPC のスライスへのマッピングはユーザが直接記述することにより, 意図しないスライス数の増大を防ぐことができる．Xilinx 社の Artix-7 を想定した GPC(1,5;3) の VerilogHDL 記述の例を図 9 (module gpc15_3) に示す．機能的には 図 9 (a) のように書けるが, これを 図 9 (b) のように書く．6–13 行目と 14–21 行目でそれぞれ入力を共通とする 5 入力 2 出力 LUT の使用とその真理値表を指定している．22–26 行目ではキャリーチェーンの使用を指定している．

3.3 閾 値 比 較

加算結果と閾値を単純に比較する場合の回路構成は 図 10 (a) のようになる．GPC 木により得た 2 つの 2 進数を row adder で加算し, それを閾値 T と比較している．これに対し本稿では,



(a) 従来の閾値比較



(b) キャリー参照による閾値比較

図 10: 閾値比較

和のキャリー参照のみで比較結果を求める．これを 図 10 (b) に示す．加算木に $B = 2^b - T$ で求められるバイアス B (ただし b は $N + T < 2^b$ となる最小の整数) を入力する．これにより, 入力ビットの和と閾値の大小比較を最上位ビットからのキャリー参照のみによって行うことができる．row adder によって和の全桁を求める必要がなくなるため, g 信号と p 信号は 2 桁分まとめて計算でき, キャリーロジックも 8 桁につき 1 つで済む．(6 入力 LUT を 2 つ用いて 3 桁分の g 信号と p 信号をまとめることもできる.) これにより, 計算に必要な LUT とキャリーチェーンの数およびキャリー伝播遅延を削減することができる．

3.4 パラメータ埋め込み型ニューロン

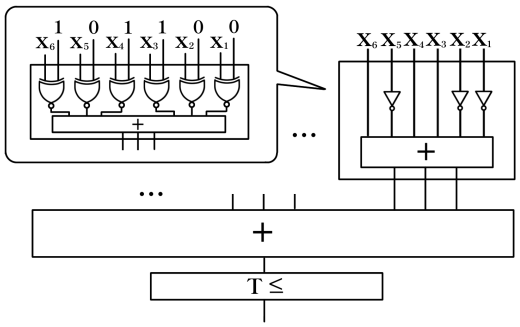
同一の重みや閾値を用いて繰り返し推論を行う場合には, 重みのパラメータはニューロンの外部 (メモリ等) から供給するのではなく回路に埋め込むことができ, これによって回路規模の削減を図れる．

BNN のパラメータ埋め込み型ニューロンの構成を 図 11 (a) に示す．w_i と x_i の XNOR 演算を, w_i が 1 の場合には x_i に, w_i が 0 の場合には $\bar{x}_i$ に置換できる．例えば LUT を用いて 6 入力分の加算を行う場合, 図のように, 加算木に入力する 2 進数は 3 桁になる．また, 閾値から計算されるバイアス B は 図 11 (b) のように, 0 であるビットを無視して GPC 加算木に入力できる．

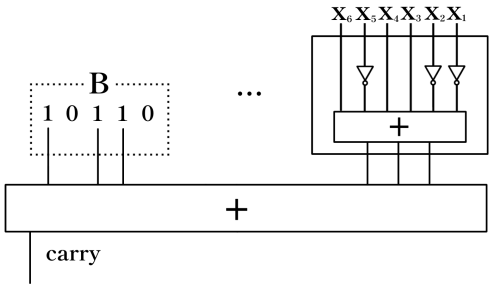
4. 実装と実験

提案手法に基づく BNN ニューロンを Verilog HDL により設計した．

ターゲットとした FPGA は Xilinx Artix-7 (xc7a100tcsg324-



(a) パラメータ埋めこみ型 BNN ニューロン



(b) キャリー参照による閾値比較

図 11: パラメータ埋めこみ

表 1: 実装に用いた GPC

GPC [文献]			
(1;1)		(1,4,0,6;5)	[9]
(3;2)		(1,3,2,5;5)	[6]
(7;3)	[9]	(1,3,4,3;5)	[6]
(1,5;3)	[9]	(2,1,3,5;5)	[10]
(2,3;3)		(1,3,5;4)	[6]
(6,2,3;5)	[9]	(2,2,3;4)	
(6,0,6;5)	[9]	(2,0,7;4)	
(6,1,5;5)	[9]	(2,1,5;4)	
(1,4,1,5;5)	[9]		

3) である. LUT は 6 入力 1 出力であり, 入力が共通であれば 5 入力 2 出力の LUT としても使用できる. 1 スライスには 4 つの LUT と 4 ビットのキャリーチェーン 1 つが含まれている. 論理合成には Vivado 2020.2 を用いた.

図 5 の①および図 11 の XNOR 演算部には 6 入力 LUT を用いた. 図 10 (b) の g 信号と p 信号の計算には 5 入力 2 出力 LUT を用いた.

加算木の構築には表 1 に示す GPC を用いた. これらは全て, 4 つ以内の LUT と 1 つのキャリーチェーンで構成でき, 1 スライスで実装できる. GPC 加算木は, スライスの段数が最小となる構成のうち, スライス数が最小となるものを整数線形計画法により求めた.(注1) ソルバーには CPLEX22.1.0 を用いた. 以下の全ての実験において, 最適解は 1 秒以内に求めることができた. 実験は Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz で行った.

論理合成と配置配線の結果を表 2 に示す. N は BNN ニューロンの入力数であり, それぞれのスライス数とクリティカルパス遅延を示している. 表中の “naive” は加算木と閾値比較を単純に論理合成したものであり, “GPC-tree” は加算木を GPC により構成したもの, “GPC-tree+bias”は閾値比較をキャリー参照に置き換えたものである.

(注1) : 加算木構築のために用いたプログラムは, https://github.com/void-hoge/cmpgen で公開している.

表 2: 論理合成結果 (パラメータ埋め込みなし)

N		32	64	128	256
スライス数	naive	28	69	177	176
	GPC-tree	26	41	79	158
	GPC-tree+bias	20	40	77	165
遅延 (ns)	naive	9.51	13.87	21.07	14.09
	GPC-tree	10.47	12.44	12.70	13.15
	GPC-tree+bias	9.29	11.09	11.53	12.84

Synthesizer: Xilinx Vivado (2020.2)
Target: Xilinx Artix-7(xc7a100tcsg324-3)

表 3: 論理合成結果 (パラメータ埋め込みあり)

N		32	64	128	256
スライス数	naive	10	59	108	107
	GPC-tree	16	25	54	97
	GPC-tree+bias	11	15	43	95
遅延 (ns)	naive	6.87	10.26	16.58	12.39
	GPC-tree	8.50	9.41	11.37	12.17
	GPC-tree+bias	8.86	9.30	11.18	11.71

Synthesizer: Xilinx Vivado (2020.2)
Target: Xilinx Artix-7(xc7a100tcsg324-3)

“naive” と “GPC-tree” を比較すると, すべての N においてスライス数が削減され, N=256 においては 10.2% の削減が達成できた. N=64 以上においては遅延時間も削減され, N=256 においては 6.7% の削減となった. また, “naive” と “GPC-tree+bias” を比較すると, すべての N においてスライス数と遅延時間が削減され, N=256 においてはそれぞれ 6.3%, 8.9% の削減が達成できた.

表 3 はパラメータを埋め込んだ BNN の合成結果である. GPC 加算木を用いることにより, N=64 以上でスライス数や遅延時間が削減され, N=256 においては, スライス数が 9.3%, 遅延時間が 1.8% 削減された. 閾値比較の改良によってもスライス数が削減され, N=256 においては遅延時間も最大で 5.5% 削減された.

5. むすび

本稿では, 一般化並列カウンタ (GPC) に基づいて 2 値化ニューラルネットワーク (BNN) のニューロンの加算木を構成することにより, 効率的に FPGA 実装する手法を提案した. 本手法に基づく BNN ニューロンを FPGA に実装した結果, 単純な論理合成と比べて, 入力数 256 のとき回路規模を 6.3%, クリティカルパス遅延を 8.9% 削減できた.

提案手法は BNN のみならず固定小数点を用いるニューラルネットワークの FPGA 実装にも応用できると考える. また本稿では, ニューロンの全入力を並列に処理する組合せ回路実装について述べたが, 入力を複数回に分けて処理する並列/直列ハ

イブリッド実装や, パイプライン実装への適用が今後の課題として挙げられる.

謝　　辞

本研究にあたり, ご助言を頂きました京都高度技術研究所の神原弘之氏, 立命館大学の冨山宏之教授, 元立命館大学の中谷嵩之氏に感謝いたします. また, 西明寺亮太氏はじめ, 本研究にご協力, ご討議頂いた関西学院大学石浦研究室の諸氏に感謝致します.

文　　献

[1] 田中 愛久, 黒柳 奨, 岩田 彰: “FPGA のためのニューラルネットワークのハードウェア化手法,” 信学技報, NC2000-179, pp. 175–182 (Mar. 2001).

[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio: “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1,” *Computing Research Repository (CoRR)*, pp. 1–11 (Mar. 2016).

[3] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr: “Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC,” in *Proc. International Conference on Field-Programmable Technology (FPT 2017)*, pp. 77–84 (May 2017).

[4] S. C. Wallace.: “A suggestion for a fast multiplier,” in *IEEE Trans. Electronic Computer*, vol. EC-13, issue 1 (Feb. 1964).

[5] L. Dadda.: “Some schemes for parallel multipliers,” in *Associazione Elettrotecnica ed Elettronica Italiana*, vol. 34, pp. 349–356 (May 1965).

[6] Y. Yuan, L. Tu, K. Huang, X. Zhang, T. Zhang, D. Chen, and Z. Wang: “Area optimized synthesis of compressor trees on Xilinx FPGAs using generalized parallel counters,” in *IEEE Access*, pp. 134815–134827 (Sept. 2019).

[7] M. Kumm and J. Kappauf: “Advanced compressor tree synthesis for FPGAs,” in *IEEE Trans. Computers*, pp. 1078–1091 (Jan. 2018).

[8] M. Kumm and P. Zipf: “Efficient high speed compression trees on Xilinx FPGAs,” in *Proc. Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, pp. 1–13 (Jan. 2014).

[9] T. B. Preußer: “Generic and universal parallel matrix summation with a flexible compression goal for Xilinx FPGAs,” in *Proc. International Conference on Field Programmable Logic and Application*, pp. 1–7 (Sept. 2017).

[10] M. Kumm and P. Zipf: “Pipelined compressor tree optimization using integer linear programming,” in*Proc. International Comference on Filed Programmable Logic and Applications (FPL2014)* pp. 1–8 (Sept. 2014).