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
CPSY2018-112,DC2018-94（2019-03）

ソフトウェア開発者から見たFPGA開発における高位合成の問題点

オセロゲームを題材としたRTL設計と高位合成の比較

濱崎　福平†　　手塚　宏史†　　稲葉　真理†

† 東京大学大学院情報理工学系研究科
〒113-8656 東京都文京区本郷 7-3-1
E-mail: †{hamazaki.fukuhei,hiroshi.tezuka,mary}@ci.i.u-tokyo.ac.jp

あらまし　高位合成によってC言語などのプログラムからハードウェアモジュールを生成することができ，ソフトウェア開発者でもハードウェア設計が可能になりつつある．本稿では，FPGA上のオセロシステムを題材として，ソフトウェア開発者の立場から，Verilog HDLとC言語から合成した回路の比較実験を行った．その結果，高位合成は実行クロック数，ハードウェアリソースの両方でハードウェアに最適化されたRTL設計に及ばなかった．効率的な回路を合成するためには，モジュールインターフェースや回路の並列化など，ハードウェアの知識が必要であることがわかった．

キーワード　FPGA, 高位合成

Problems of High Level Synthesis for software developers

Comparison between RTL and HLS in a FPGA Othello game system

Fukuhei HAMAZAKI†, Hiroshi TEZUKA†, and Mary INABA†

† Graduate School of Information Science and Technology, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
E-mail: †{hamazaki.fukuhei,hiroshi.tezuka,mary}@ci.i.u-tokyo.ac.jp

Abstract　Thanks to high-level synthesis, hardware modules can be generated from programs such as C language, allowing software developers to design hardware as well. In this paper, we compared the circuit synthesized from Verilog HDL and C language from the viewpoint of software developer in a FPGA Othello game system. As a result, high-level synthesis did not reach the RTL design optimized for hardware in both clock cycle and hardware resources. In order to synthesize an efficient circuit, we found that hardware knowledge such as module interface and circuit parallelization is necessary.

Key words　FPGA, high-level synthesis

1. はじめに

プログラミングによって内部を書き換えられる集積回路をFPGA (Field Programmable Gate Array) と呼ぶ．これは逐次処理のマイクロプロセッサとは異なり，高速かつ大量に流れるデータを多チャネル並列でリアルタイムかつ省電力で処理できる．このため，信号処理・ネットワーク・大規模データ処理など様々な分野で応用されている．[1] [2] [3].

FPGAの設計には，ハードウェア記述言語 (Hardware Description Language，以下HDL) を用いてRTL (Register Transfer Level) から回路を記述する．HDLとは半導体チップの回路設計などを行うための人工言語であり，プログラミング言語に似た構文や表記法で，回路に含まれる素子の構成や動作条件，配線等を直接的に記述することができる．

一方で，RTLからのハードウェア設計には，HDLの抽象度の低さゆえ，煩雑なアルゴリズムを実装する場合等に開発に時間を要してしまう他，アルゴリズムの機能検証も困難であるという課題がある．これらを解決するハードウェア設計手法の１つが，高位合成ツールを用いることである．高位合成 (HLS: High Level Synthesis) とはC/C++のようなRTLより高い抽象度のソフトウェアプログラム向けの言語をHDLに変換することで，抽象度の高い記述からハードウェア設計の記述を支援するプロセスである．FPGA開発フローの全体像を図1に示す．高位合成ツールには，Vivado HLS [4] やLegUp [5] のよう

な C/C++/SystemC 等の手続き型言語を設計言語としたものがある．

高位合成ツールを用いることで，RTL から HDL 記述を始める場合に比べて少ない記述量でハードウェア設計が可能になり，開発効率の向上が期待される．しかし，高位合成では元のソースコードに対して多数の最適化ディレクティブを指定しない限り合成後の回路のハードウェアリソースや実行クロック数といった性能が HDL よりも劣ることがある．そのため，HDL 記述と高位合成は多くの場合，実行パフォーマンスと開発効率のトレードオフになる．

本稿では，オセロのアーキテクチャ全体の HDL による実装と，高位合成による NPC モジュールの生成およびアルゴリズム比較実験を通して，ソフトウェア設計者としての視点から RTL 設計と高位合成のハードウェア設計の違いを確認し，既存の高位合成の課題について言及する．

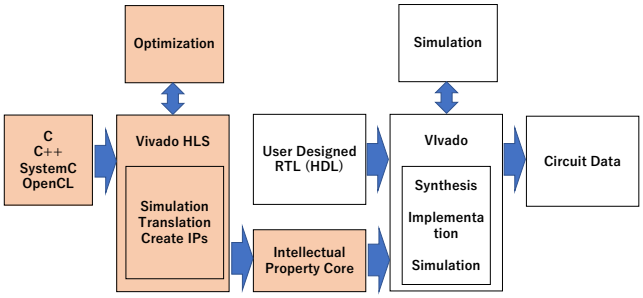

図 1　高位合成を使った FPGA 開発フロー (色で覆った部分が高位合成)

2. 関 連 研 究

FPGA ではデータの高い並列性を利用して，プロセッサによる処理と比べて低消費電力かつ高いパフォーマンスを引き出すことができる．例えば，文献 [7] では津波シミュレーターにおける数値計算の一部で FPGA を用い，CPU を使用した場合との速度比較を行い，23 倍の高速化に成功している．また，文献 [8] では処理の内容によっては FPGA が CPU や GPU と比べて処理性能の他に消費電力の観点でも優れていることが示されている．

ハードウェア開発の大規模化が進むとともに，開発効率を高めるべく高位合成ツールの利用が進んでいる．高位合成ではソフトウェア向けの言語でハードウェアモジュールを設計することができる．FPGA ベンダの一つである Xilinx [9] は，C/C++ 高位合成処理系の VivadoHLS [6] を開発者に提供している．C/C++ 以外の言語を対象した高位合成処理系もあり，Java を設計言語とする Lime [10] や C# の Kiwi [11] などをはじめ，様々な言語で研究が行われている．例えば，文献 [12] では関数型言語である Haskell の組み込み DSL(ドメイン固有言語) を設計言語とした高位合成を行っており，文献 [13] では C の言語仕様に合わせた並列制御の複雑性に着目し，JVM 上で動作する Java プログラムを設計言語とした高位合成を行っている．

開発の効率化という観点から，高位合成を使わないアプローチとして RTL 設計に対する DSL がある．RTL 設計に必要な組合せ回路，レジスタ，配線といった要素を，ホスト言語の持つ型や抽象化システムで構成することでハードウェア設計を可能にしており，Python をホスト言語とする MyHDL [14] や， Scala をホスト言語とする Chisel [15] がある．

3. 高位合成の利点・現状

3.1 高位合成の利点

高位合成によるアプローチによって，既存の言語で記述したプログラムを FPGA 上の回路として移植することができ，開発コストを大きく下げることができる．その他のメリットとしては，ハードウェア素人でもハードウェア設計ができること，アルゴリズムの記述や変更が容易なこと，デバッグが容易なこと，使用言語にもよるが IDE のサポートが充実していることがあげられる．

3.2 現状の問題点

高位合成の問題点として，生成したハードウェアの質が処理系に強く依存する可能性がある．処理系を設計する立場に立ってみても，既存のプログラミング言語で記述されたプログラムからデータ並列性やパイプライン並列性を抽出し，効率良く処理できる回路を生成することは簡単ではない．そのため，良い回路の生成には，最適化ディレクティブや専用の構文や演算を導入して合成を制御することが必要になると言われている．しかしながら，高位合成を利用するユーザからみれば，手軽なハードウェア開発というメリットが損なわれることになる．

4. FPGA を用いたオセロシステムの実装

ソフトウェア開発者から見た RTL 設計と高位合成を明らかにするために，Verilog HDL を用いてオセロシステムを FPGA 上に実装した．本オセロシステムは，NPC モジュールは組み換え可能にし，その内部を Verilog HDL と C 言語で記述して比較実験を行った．

4.1 システム概要

本オセロシステムは，一般的なオセロと同様に，プレイヤーと NPC (Non Player Character: 人工的な指し手) のうち一方が石を置けなくなるか，8 × 8 の盤面が埋まるまで交互に自分の色の石を置いていく．プレイヤーは石を置きたいマスの座標を PC から UART を通じて入力する．ゲームの盤面はリアルタイムにディスプレイに外部出力される．

4.2 ブロックダイアグラム

図 2 に本オセロシステムのブロックダイヤグラムを示す．本システムは高位合成と入れ替え可能な部分 (NPC セクション) と RTL で設計した部分 (ベースセクション) にわかれている．システムを分割しているのはオセロの持つ戦略の柔軟性を考慮してのことである．オセロは二人零和有限確定完全情報ゲームであり，勝敗は対戦者の戦略によって左右される．対戦者の動きに相当する部分を書き換え可能にすると，対戦者の持つ戦略に幅をきかせることができる．そのため，戦略の部分に相当する NPC モジュールを組み換え可能な設計とした．

ベースセクションでは，PC からの入力情報のシリアルパラレ

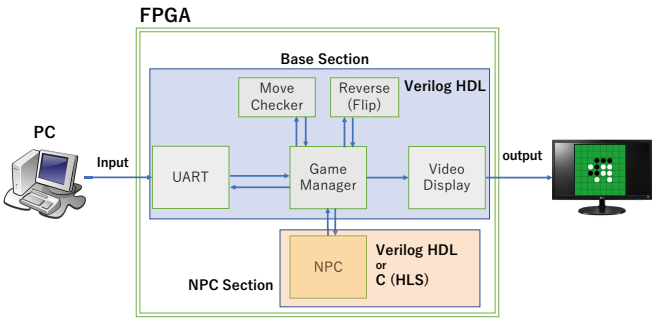

図 2　システム全体のブロック図

ル変換を行う UART モジュール，ゲームの進行状態を管理する GameManager モジュール，置きたい石の場所がルールを満たしているかどうかのチェックを行う MoveChecker モジュール，相手の石を裏返す処理を行う Reverse モジュール，盤面の状態を RGB 信号にして外部ディスプレイに出力する VideoDisplay モジュールの計 5 つのモジュールで構成されている．

NPC セクションでは，NPC モジュールが石を置く位置を決める．このモジュールのみ他の言語で作った回路と入れ替え可能となっている．

4.3　NPC モジュールの動作

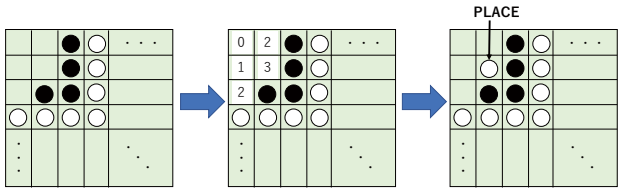

図 3　NPC モジュールの動作

NPC モジュールは，プレイヤーが石を置いた直後の盤面の状態を見て，図 3 のように，プレイヤーの石を最も裏返せる位置の一つに石を置く．このモジュールでは，現在の盤面の状態を受け取り，石を置くことのできる全ての位置に対して相手の石を裏返せる石の個数を求め，その中からそれが最大値を持つときの位置を返す．裏返せる石の個数は，置いた石の位置から盤の端までのマスの数を 8 方向求め，各方向について置いてある石の色を判定することによって求める．

4.4　GameManager モジュールの動作

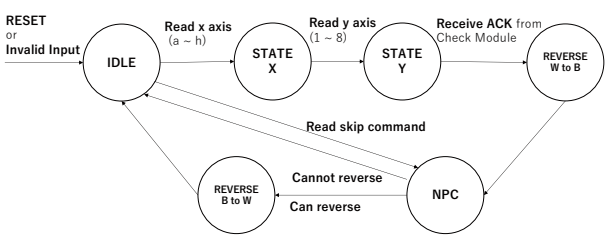

図 4　GameManager の状態遷移

GameManager モジュールは図 4 のような状態遷移に従ってゲームの進行状態を管理する．プレイヤーが石を置いたら，MoveChecker モジュールで置いた石の場所がルールを満たしているか判定を行い，ルールが満たされた場合は Reverse モジュールに信号を送り石を裏返す処理を行う．その後は NPC のターンとなり，NPC モジュールに現在の盤面の状態を送り，NPC が置いた石の座標を受け取る．GameManager モジュールは受け取った座標を Reverse モジュールに渡して再び裏返しの処理を行い，ターンをプレイヤーに戻す．ゲームの基本的な流れはこの手順の繰り返しである．

5.　HDL と高位合成の比較実験

本稿では，オセロシステムのモジュール群のうち，NPC モジュールを対象として RTL 設計の場合と高位合成を利用して生成される回路のパフォーマンス比較を行った．合成系が出した数値を参照して回路のクロック周波数，ハードウェアリソース，実行クロック数を評価観点とする．

5.1　実 験 環 境

表 1　FPGA 諸元

ボード名	Nexys 4 DDR
FPGA	XC7A100T-1CSG324C
FF(Flip-Flop)	126800
LUT(LookUp-Table)	63400
ターゲット周波数	100MHz

評価に用いる FPGA ボードは Digilent 社の Nexis4 DDR [16] を，開発ツールは Xilinx 社の Vivado 2018.1 および Vivado HLS 2018.1 [6] を用いた．ターゲットのクロック周波数は 100MHz とした．FPGA ボードの諸元を表 1 に示す．

5.2　ソースコード

NPC モジュールの内容は，与えられた盤面を参照し，最もプレイヤーの石を裏返すことのできるマスの位置を求め，その一つを返すものである．プログラムは，(A) Verilog HDL で記述したもの，(B) 記述スピード重視で C 言語で記述したもの，(C) BitBoard データ構造を使用して C 言語で記述したものの 3 つである．C 言語を利用した (B) と (C) を高位合成の対象とする．

5.2.1　A: Verilog HDL

```
// RdT: 右上方向に裏返せる石の数
if ((B[up1]&&!B[up1+N])&& ... &&(B[up7]&&B[up7+N]))
    RdT <= 6;
else if ((B[up1]&&!B[up1+N])&& ... &&(B[up6]&&B[up6+N]))
    RdT <= 5;
else if ((B[up1]&&!B[up1+N])&& ... &&(B[up5]&&B[up5+N]))
    RdT <= 4;
else if ((B[up1]&&!B[up1+N])&& ... &&(B[up4]&&B[up4+N]))
    RdT <= 3;
else if ((B[up1]&&!B[up1+N])&& ... &&(B[up3]&&B[up3+N]))
    RdT <= 2;
else if ((B[up1]&&!B[up1+N])&&(B[up2]&&B[up2+N]))
    RdT <= 1;
else
    RdT <= 0;
```

図 5　Verilog HDL による記述 (抜粋)

Verilog HDL のコードは，あるマスの座標値を入力とするモ

ジュールを内部に作り，それをインスタンス化している．内部モジュールでは入力座標から盤面端までのマスの数を 8 方向それぞれ求め，図 5 のように各方向の石の配置をあらかじめ作ったテーブルと照合することで裏返せる石の数を算出する．次にそれを 8 方向分足し合わせ，返り値とする．最後に各モジュールの返り値の中で最も大きい値を持つときの座標を求める．インスタンス化ではマスの総数 64 個分を並列に展開することで並列度合いを高めている．

5.2.2 B: 二次元配列を用いた C

```
int boardmap[8][8] =
{
    { 2, 2, 2, 2, 2, 2, 2, 2},
    { 2, 2, 2 ,2, 2, 2, 2, 2},
    { 2, 2, 2, 2, 2, 2, 2, 2},
    { 2, 2, 2, 1, 0, 2, 2, 2},
    { 2, 2, 2, 0, 1, 2, 2, 2},
    { 2, 2, 2, 2, 2, 2, 2, 2},
    { 2, 2, 2, 2, 2, 2, 2, 2},
    { 2, 2, 2, 2, 2, 2, 2, 2},
};// 但し，黒を0，白を1，空きを2とする
```

図 6　二次元配列による盤面表現

```
for (int ix=0; ix<8; ix++){
  for (int iy=0; iy<8; iy++){
    ...
    if (boardmap[ix][iy]==NOTSTONE){
      for (int i = 0; i < 8; i++){
        int checkX = ix + testValueSet[i][0];
        int checkY = iy + testValueSet[i][1];
        ...
        while( 0<=checkX && checkX<=7 && ...
              && boardmap[checkX][checkY]==BLACK ){
          ...
        }
    ...
}
```

図 7　二次元配列を使用した C の記述 (抜粋)

盤面の情報を図 6 のような int 型の 2 次元配列に格納して，図 7 に示すように盤面の x 方向 y 方向それぞれに for ループを施す．ループの内部で配列の要素を一つずつ参照しながら石の設置の可否を判定し，このとき裏返せる石の個数を計算し，それが最大値を持つときの座標値を返す．

5.2.3 C: BitBoard を用いた C

```
uint64_t black = 0x0000000810000000;
uint64_t white = 0x0000001008000000;
uint64_t blank = ~(black | white);
```

図 8　BitBoard を使用した盤面表現

盤面の情報を BitBoard というデータ構造を用いて表現する．BitBoard とは図 8 のように 64 マスのそれぞれについて石が存在するかを 1 ビットで表すデータ構造である．マスの参照や裏返せるかの判定をビットシフトや論理演算だけで行うこと (ビットパラレル法) ができるので通常のデータ構造よりも処理が高速である上，64 ビットレジスタ一つで石の状態を表せることが可能でリソースの無駄を省くことができる [17].

表 2　最適化ディレクティブ

ディレクティブ	内容
PIPELINE	演算子のパイプライン化
DATAFLOW	関数やループのパイプライン化
UNROLL	for ループの並列展開

5.3 最適化ディレクティブ

(B) と (C) においてそのまま高位合成を行うと，意図通りに並列化が行われずにリソースやクロック数が大きな回路が生成されることがある．このため，Vivado HLS ではプラグマによる最適化ディレクティブを指定することによって回路変換をある程度制御することができる [18]．用いた最適化ディレクティブの種類と内容を表 2 に簡単に示す．

本稿では (B) と (C) に最適化ディレクティブを指定した場合の比較も行う．まず，最適化を指定しない場合をそれぞれ (B1), (C1) とする．(B) では図 7 の 3 つの for ループのうち最も内側のループに PIPELINE 最適化を指定した場合 (B2)，最も外側のループに PIPELINE 最適化を指定した場合 (B3)，同じ箇所に UNROLL 最適化をかけた場合 (B4) を比較する．(C) ではソースコード中に for ループが 1 つしか存在しないため，そのループに PIPELINE 最適化を指定した場合 (C2)，ループ内に DATAFLOW 最適化を指定した場合 (C2) を比較する．

比較対象となるプログラムの一覧を表 3 に示す．

表 3　比較対象のプログラム一覧

	言語	高位合成	データ構造	最適化ディレクティブの種類
A	Verilog HDL	×		
B1	C 言語	○	二次元配列	なし
B2	C 言語	○	二次元配列	PIPELINE(最も内側のループ)
B3	C 言語	○	二次元配列	PIPELINE(最も外側のループ)
B4	C 言語	○	二次元配列	UNROLL(最も外側のループ)
C1	C 言語	○	BitBoard	なし
C2	C 言語	○	BitBoard	PIPELINE
C3	C 言語	○	BitBoard	DATAFLOW

6. 実験結果

表 4　生成回路の比較

対象回路	達成可能周波数 [MHz]	FF	LUT	実行クロック数
A	101.3	3091	1213	20
B1	119.8	327	743	1996
B2	119.8	344	772	749
B3	123.9	10235	18946	1980
B4	117.2	20864	74403	1206
C1	138.7	1767	5547	646
C2	204.1	2350	3291	398
C3	123.9	2795	8893	129

周波数とハードウェアリソース (FF, LUT)，および実行クロック数の比較結果を表 4 に示す．

6.1 最大動作周波数

周波数はいずれも目標周波数を 100MHz としたときの最大動作周波数を示しており，いずれも 100MHz を上回っているため，動作の上では問題ないと言える．特に，(C2) では目標の約 2 倍の周波数が得られており，タイミングに関しては余裕のある回路が生成できている．

6.2 ハードウェアリソース

FF と LUT は，(A) の Verilog HDL の回路が (B1), (B2) の回路の 10 倍近いリソースとなっている．これは，(A) のソースコード内にループ処理が存在しないことによる並列度の高さと，(B1), (B2) がソースコード通りの逐次実行によって同じハードウェアリソースを何度も再利用していることが関係している．

また，(B3)，(B4)，(C2)，(C3) は最適化ディレクティブを指定する前と比べてハードウェアリソースが増大している．これはループの内部の演算回路を並列展開したことによる増大だと考えられる．(B3), (B4) では増大の程度が特に大きいが，外側のループで並列化した場合，その内側のループ内の演算まで展開されたことでの回路規模の膨大化によるものだと考えられる．さらに，(B) のソースコードでは変数の型を考慮していないため，ビット幅を不必要に取りすぎていることもリソースを増加させている一因だと考えられる．(B4) では合成後の FF がボードの FF リソース (63400) をオーバーしているので，回路をボードに載せることができない．

6.3 実行クロック数

結果の実行クロック数を見ると，(A) の Verilog HDL がほかの高位合成のどの結果よりも小さなクロック数であり，優れたパフォーマンスを出していることがわかる．これは，Verilog HDL では 8 方向に同時にパターンマッチングを試すモジュールを 64 個同時にインスタンス化することで並列度合いを高めているためだと考えられる．

一方，高位合成で生成された回路の実行クロック数が大きくなった原因として，一つはパイプラインの各ステージの処理時間の乱れが考えられる．パイプライン化したい処理のステージのうち，あるステージの処理時間が長いと，パイプライン処理全体に乱れが生じるため，結果として遅延につながる．これは，外側のループでパイプラインを指定した場合に特に影響が大きくなっている．もう一つはモジュールインターフェースが考えられる．二次元配列を入力としたコードの合成では，メモリをインターフェースとして入力だけで 64 サイクル以上要する回路が生成された．

6.4 考　　察

(B) と比較して (C) は実行クロック数において高いパフォーマンスを出している．これは，少ないリソースで盤面状態を表現できるデータ構造をしていること，ビットシフトによる演算処理を多用していること，ループ処理が B と比べて少ないことなどが考えられる．

最適化ディレクティブを指定して合成された回路は，最適化ディレクティブを指定しない場合と比べてリソースとクロック数のトレードオフになった．並列度が上がるとそれだけ演算器やレジスタが必要になるからである．しかし，(B3), (B4) のような最適化はリソース，クロック数共に悪くなることから，トレードオフにはならず，ループの無闇なパイプラインは必ずしも良い結果となることは限らないことも明らかになった．パイプラインのステージの処理時間にボトルネックが生じているため，パイプライン化時に各ステージの終了時間を一定時間で完了できるような処理の調整が必要である．

適切な最適化をするには，処理内部のリソースの細かな割り振り，パイプラインのステージ数やレイテンシの指定，合成語のモジュールインターフェースの指定など，ハードウェアに踏み入った最適化が要求されている．これは，ハードウェア経験の少ないソフトウェア開発者が直面する重要な課題と言えるだろう．また，本稿の (C) のコードのように可読性を犠牲にして演算処理に特化したコードと最適化をもってしても，RTL 設計した回路に実行クロック数で負けてしまうことがあることも記憶に留めておきたい．

7. ま と め

本稿ではソフトウェア開発者から見た RTL 設計と高位合成の違いを確認し，高位合成の問題点を明らかにするため FPGA 上にオセロシステムのアーキテクチャを RTL から実装した．それを題材にして，単体で取り換え可能なモジュールとして設計した NPC モジュールを対象に Verilog HDL と C 言語で生成された回路の性能比較を行った．C 言語はスピード重視で簡単に記述したソースコードと，リソースや計算負荷を考慮したデータ構造を用いたソースコードを 2 種類用意し，最適化ディレクティブを数パターンに分けて指定した．しかし，高位合成では全ての場合において Verilog HDL で記述した回路の性能に実行クロック数で及ばない結果となった．

本稿の実験でディレクティブを複数パターン試すことにより，最適化ディレクティブを適切にかけることで回路のパフォーマンスを上げられることがわかったが，より良いパフォーマンスを引き出すための最適なディレクティブをかけるためには今回使用した以外の最適化ディレクティブの知識やモジュールインターフェースなどハードウェア関連の知識を抑えておく必要がある．

今回比較したモジュールは回路規模としては小さなものだったが，対象とするアルゴリズムや規模が変われば違った結果になることも考えられる．また，手元の資源量によって求められるパフォーマンスが違う可能性もあるので，その辺りについて今後比較実験を行っていきたい．また，高位合成のさらなる最適化の探求と同時に，ソフトウェア開発者の立場からより記述しやすい HDL の検討をすべく，Scala ベースの DSL である Chisel などについて調査も進めていきたい．

8. 謝　　辞

本研究にあたって，数々の助言をいただいた西野兼治氏，塩谷亮太准教授に深く感謝申し上げます．

文　献

[1] A. Putman, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao and D. Burger, "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services," *Proc. ACM/IEEE 41st Annual International Symposium on Computer Architecture (ISCA),* Minneapolis, MN, USA, IEEE, pp. 13–24 (online) , DOI: 10.1109/ISCA.2014.6853195(2014).

[2] J. Ouyang, S. Lin, S. Jiang, Z. Hou and Y. Wang, "Software-defined Flash for Web-scale Internet Storage Systems," *Proc. 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS),* Salt Lake City, Utah, USA, ACM, pp. 471–484 (online) , DOI: 10.1145/2541940.2541959(2014).

[3] Z. Istvan, G. Alonso, M. Blott and K. Vissers, "A Hash Table for Line-Rate Data Processing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS),* Vol. 8, No. 2, pp.13:1–13.15 (online), DOI: 10.1145/2629582 (2015).

[4] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers and Z. Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 30, No. 4, pp. 473–491 (online), DOI: 10.1109/TCAD.2011.2110592 (2011).

[5] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown and J. H. Anderson "LegUp: An Open-Source High-Level Synthesis Tool for FPGA-Based Processor/Accelerator Systems," *ACM Transactions on Embedded Computing Systems (TECS),* Vol. 13. No. 2, pp. 24:1–24:27 (online), DOI: 10.1145/2514740 (2013).

[6] Xilinx: Vivado High-Level Synthesis, https://japan.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[7] 福井啓，藤田昌宏情報処理学会九州支部火の国シンポジウム 2017，A4-4，March 2017.(2017) "高位合成ツールを用いたハードウェアアルゴリズムの最適化," 情報処理学会研究報告 Vol.2011-SLDM-150 No.10 (2011).

[8] Accelerating floating-point fitness functions in evolutionary algorithms: A FPGA-CPU-GPU performance comparison

[9] Xilinx, https://japan.xilinx.com

[10] J. Auerbach, D. F. Bacon, P. Cheng and R. Rabbah, "Lime: a Java-compatible and synthesizable language for heterogeneous architectures," *Proceedings of the ACM international conference on Object oriented programming systems languages and applications, OOPSLA '10,* New York, NY, USA, ACM, pp. 89–108 (online), DOI: http://doi.acm.org/10.1145/1869459.1869469 (2010).

[11] D. Greaves and S. Singh, "Designing application specific circuits with concurrent C# programs," *Formal Methods and Models for Codesign (MEMOCODE),* 2010 8th IEEE/ACM International Conference on, pp. 21–30 (online), DOI: 10.1109/MEMCOD.2010.5558627 (2010).

[12] 福田寛介，尼崎太樹，飯田全広，久我守弘，末吉敏則 "並列デザインパターンを用いた関数型言語による高位合成," 情報処理学会九州支部火の国シンポジウム 2017, A4–4, March (2017).

[13] 三好健文，船田悟史 "FPGA 向け高位合成言語としての Java の活用手法の検討," 情報処理学会第 53 回プログラミング・シンポジウム, pp59–68 (2012).

[14] MyHDL, http://www.myhdl.org

[15] Bachrach, Jonathan, Vo, Huy, Richards, Brian, Lee, Yunsup, Waterman, Andrew, Avizienis, Rimas, Wawrzynek, John, Asanovic and Krste "Chisel: Constructing Hardware in a Scala Embedded Language," *Proceedings of the 49th Annual Design Automation Conference,* DAC '12, New York, NY, USA, ACM, pp. 1216–1225 (2012).

[16] Nexys4 DDR, https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start

[17] ビットボード，https://ja.wikipedia.org/wiki/ビットボード

[18] Vivado Design Suite ユーザーガイド 高位合成, https://www.xilinx.com/support/documentation/sw_manuals_j/xilinx2017_4/ug902-vivado-high-level-synthesis.pdf