情報処理学会論文誌 Vol.57 No.8 1680–1689 (Aug. 2016)

回路分割機能付きJava言語ベース高位合成ツールにおける回路検証機構

松田 和也[1] 三好 健文[2] 竹本 正志[1,3] 船田 悟史[4] 中條 拓伯[1,a)]

受付日 2015年11月19日, 採録日 2016年5月17日

概要：近年，従来の回路設計に用いられてきたHDLに替わり，高位合成ツールの活用に注目が集まっている．しかし，複雑なアルゴリズムをハードウェア化する際に，合成回路が大規模化する場合やシミュレーションに膨大な時間を要する場合がある．そこで，複数FPGAに対する分割実装が用いられるが，FPGAの回路規模やI/Oブロック数による制約が問題となり，検証環境の構築は容易ではない．本研究では，高位合成ツールの合成回路を部分回路に分割し，回路検証用のラッパーを生成することで，部分回路単位での検証を可能とする．高位合成ツールを用いて，FFTを実行するプログラムを合成し，回路分割検証機構により分割した．各部分回路は，シミュレーションおよびFPGA上で動作検証を行い，正常に動作することを確認した．

キーワード：高位合成，回路分割，回路検証

Verification Mechanism for Partitioned Circuits on a Java-based High Level Synthesizer with a Circuit Partiotioning Function

Kazuya Matsuda[1] Takefumi Miyoshi[2] Masashi Takemoto[1,3]
Satoshi Funada[4] Hironori Nakajo[1,a)]

Received: November 19, 2015, Accepted: May 17, 2016

Abstract: In recent years, a high-level synthesis tool has been attracted in designing hardware circuits instead of conventional HDL. However, there exist two issues to implement a complex algorithm into hardware, which brings growing scale of a synthesized circuit and long time for simulation. Therefore, though partitioning a circuit into multiple FPGAs is currently put into practical use, there are two constrains in implementation; the scale and the number of I/O blocks in an FPGA. Thus it is difficult to build a verification environment. In this study, we partition a circuit synthesized by a high-level synthesis tool into some reduced circuits. Moreover, the small circuits are equipped with self-verification function with generating a wrapper for each circuit verification. An FFT circuit which is generated by a high-level synthesis tool is partitioned by our proposed verification mechanism for partitioned circuits. We verify the partitioned circuits in RTL simulation as well as implementation on an FPGA in order to confirm our targeted circuits are correctly operated.

Keywords: high level synthesis, circuit partitioning, test and verification

1. はじめに

近年の集積回路技術の進歩により，LSIに搭載可能な回路規模は増大し，設計開発は複雑化している．それにともない，高位合成ツールによる回路設計が広まり始め，従来のハードウェア記述言語（HDL）による回路設計よりも高い抽象レベルで，複雑なアルゴリズムのハードウェア化が可能となった．現在，C言語をベースとするツールとし

[1] 東京農工大学
Tokyo University of Agriculture and Technology, Koganei, Tokyo 184–8588, Japan
[2] わさらぼ合同会社
Wasalabo LLC., Machida, Tokyo 194–0045, Japan
[3] 株式会社ビート・クラフト
BeatCraft, Inc., Sumida, Tokyo 130–0013, Japan
[4] 株式会社イーツリーズ・ジャパン
e-trees.Japan, Inc., Hachioji, Tokyo 192–0045, Japan
[a)] nakajo@cc.tuat.ac.jp

ては，SystemC [1]，ImpulseC [2]，CyberWorkBench [3] や LegUp [4]，Java 言語ベースには，Lime [5]，JavaRock [6] がある．

この中には，プログラミング言語に対する拡張を行わず，元のソースを入力とするものもあり，対象となる入力プログラムをソフトウェアとして動作させることで，アルゴリズムレベルでの検証が可能である．

高位合成ツールによって生成された回路規模が膨大なものとなり，単一の LSI や FPGA に収まりきらない場合がある．ASIC であればゲート数を増やすことで対応できるが，容量に上限のある FPGA に対しては分割実装を行う必要が生じる．この場合，分割回路間の信号線数が FPGA の I/O ブロック数を超えてしまうという問題に対しては，FPGA の I/O ブロック数制約を満たすという条件のもとで分割回路間の信号線数を最小化したり [7], [8]，1 つの I/O ブロックを複数の信号で共有する手法 [9] が提案されてきた．しかしながら，現状の高位合成ツールには，作成された回路を分割するための機能はあるものの，分割した回路を検証するための機能は備えられていない．

また，分割された回路を複数の FPGA 上に実装する場合，FPGA 間の接続や通信方式を考慮するとともに，マルチ FPGA システムが必要となる．

以上をふまえて，複雑なアルゴリズムを高級言語で記述し，回路を作成する場合には，複数 FPGA にまたがるような大規模回路に対しても，単一 FPGA 上においても容易にテストできるような環境を提供することが望ましいと考える．そこで，本研究では，高級言語による記述の段階で回路分割を行い，その分割回路を個々に動作検証するための機構を提案する．WPCV（Wrapper for Partitioned Circuit Verification）により分割回路を覆うことで，分割回路を独立させて動作させ，単一 FPGA 上で個々の分割回路をテストすることが可能となる．

本論文では，高位合成ツールが合成した回路を分割し，その分割された回路の動作検証のための WPCV を自動的に付加する機構を提案する．2 章において，関連研究について述べ，3 章で高位合成ツール JavaRock を例に合成回路に対する回路分割検証機構の処理について述べる．4 章では，回路分割検証機構の詳細を述べ，5 章で評価を行う．続く 6 章で考察を行った後，7 章で本論文のまとめを行う．

2. 関連研究

2.1 高位合成ツールに対する回路分割

CyberWorkBench では，C 言語で記述されたアプリケーションプログラムを BDL 形式に変更し，設定ファイルとともに分割ツールに渡される [10]．分割ツールは，プログラムのモジュールや関数をブロックとし，設定ファイルで与えた条件を満たすように分割を行う．FPGA の回路規模制約を満たし，分割回路の性能が最も高くなる部分において，ソースコードは分割され，そのコードは CyberWorkBench に対する入力となり，分割済みの回路が合成される．

また，データパス回路表示ツールを用いた CyberWorkBench の合成回路に対する解析を実施し [11]，有効な分割点候補を特定した後，入力ソースコードの分割を行っている．

なお，文献 [10], [11] では，その通信部分を手動で設計する必要があるため，FPGA 間の通信インタフェースへの対応は将来的な課題としてあげられている．

文献 [12] では，Java プログラムにおけるクラスをベースに回路分割を行い，FPGA の回路規模制約を満たしたうえで，I/O ブロック数の削減が試みられているが，複数 FPGA に分割実装した回路の動作に関する評価は行われていない．

以上のように，現状において高位合成ツールの合成回路に対する回路分割手法の提案は行われているものの，分割した回路が実際にどのように動作するかについての検証は行われていない．その検証のための端的な方法として FPGA 上で分割回路を動作させることがあげられるが，複数 FPGA を装備したシステムは大規模で高価なものとなる．そのため，システム上での FPGA 間の接続や通信方式において，そのインタフェースを含めた開発は設計者にとって大きな負担となり，開発期間に大きく影響を与える．

この問題を解決するためには，高位合成ツールの合成回路を分割した際に，分割回路に対して汎用的なインタフェースを供給し，その分割回路単位での回路検証を可能とすることが有効となると考えた．そこで，本研究では，以下の条件を満たす回路分割検証機構を提案し，実装した．

(1) 高位合成ツールによる合成回路に対する自動分割機能を持つこと
(2) FPGA に対して分割回路を汎用的に実装できること

条件 (1) に関しては，他の回路分割手法においても実現されている．しかしながら，条件 (2) には対応しておらず，この条件を満たすためには，複数 FPGA が相互通信を行い，回路全体を動作させる必要がある．そこで，本研究では，条件 (2) を満たすために，文献 [10], [11], [12] とは異なるアプローチを採用する．すなわち，高位合成ツールが合成した回路の動作に着目し，実行状態にある回路のみを取り出すことで，その時点における合成回路の動作を再現することを考える．このとき，各分割回路に対して WPCV を生成し，分割回路間の信号を隠蔽することで，I/O ブロック数制約を満たす．また，各分割回路は，ラッパーに内包されるため，FPGA 上において分割回路を独立して動作させることが可能である．

また，FPGA の動的部分再構成が注目され，提案手法により分割された回路に対して部分再構成を適用し，単一 FPGA においても大規模回路を動作させる可能性が見えてくる．

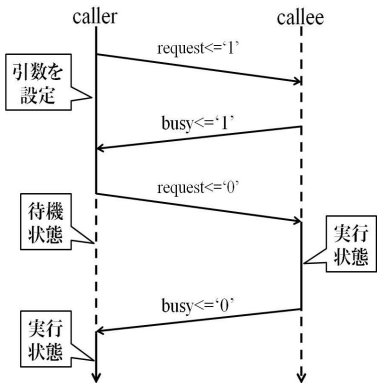


図 1 メソッド呼び出し
Fig. 1 Call of the method.

2.2 高位合成ツール JavaRock

Java ベースの高位合成ツールである JavaRock [13] は，Java 言語で記述されたプログラムから VHDL コードに変換するツールであり，Java 言語に対する拡張を行うことなく，ハードウェア設計を可能にする．

JavaRock では，ソフトウェアとしての動作をハードウェアにトレースするために，Java プログラムにおける逐次処理をステートマシンで実現する．これにより，メソッド全体の処理を管理するステートマシンが生成され，ステートマシンの状態遷移により，処理対象となる Java コードが切り替わる．if 文や while 文をはじめとした制御構文に対してもステートマシンによる制御が行われ，Java プログラムの処理が実現される．

Java プログラムにおけるメソッド呼び出しに対しては，Java 言語のメソッドを VHDL の process 文に変換することで対応する．各メソッドに対して生成されるリクエスト信号やビジー信号を利用することで，実行制御が行われ，メソッドが呼び出される．メソッドを呼び出すための具体的な手順は以下のとおりである（図 1）．

(1) caller 側はリクエスト信号をアサートし，callee 側のメソッド呼び出しをリクエスト
(2) callee 側はリクエスト信号の確認後，ビジー信号をアサート
(3) caller 側はビジー信号の確認後，メソッドの引数をセットし，リクエスト信号をネゲート
(4) callee 側はリクエスト信号の確認後，メソッドを実行
(5) callee 側は処理終了後，ビジー信号をネゲートし，戻り値を出力

現在，JavaRock を活用した研究としては，ハードウェアとソフトウェアの協調設計 [14] や Reconfigurable Android 上での活用 [15]，ロボット制御システムの設計 [16] などがあげられる．しかしながら，今後複雑なアルゴリズムや CFD（Computational Fluid Dynamics）に代表される高性能計算（HPC）への利用をターゲットにすると，合成回路の増大化は避けられず，FPGA で実装するには回路分割検証機構が必要不可欠なものになっていくものと考えられる．

3. JavaRock の合成回路に対する分割処理

JavaRock は，合成回路に対する回路分割をサポートしていなかったため，合成の段階で分割を実施する回路分割検証機構を考案し，実装と評価を進めてきた．ここでは回路分割検証機構による分割処理の概要について述べ，さらに分割回路に与える動作検証のための WPCV について説明する．

回路分割検証機構では，高位合成ツールに対する入力プログラムを対象とした分割処理を行い，WPCV が付与された分割回路を作成する．入力プログラムを処理単位で分割し，その処理を組み合わせることで，回路分割を可能とする．

本提案手法は，高位合成ツールによって作成される回路の実行状態を把握し，それぞれの実行モジュールの切替え時点に着目することで，他の高位合成ツールにも対応可能となる．

3.1 Java プログラムへの分割

提案手法を適用するためには，Java プログラムを解析し，実行状態にあるモジュールを把握する必要がある．JavaRock の合成回路における実行モジュールの切替えは，インスタンスに対するメソッド呼び出し時に起こる．そこで，インスタンスの呼び出し地点を分割点として回路分割を行うことが考えられる．しかしながら，大規模な Java プログラムにおいて複数のインスタンスが利用されることは一般的なことであるため，単純にインスタンスの呼び出し地点で回路を分割すると，分割数の増大を招くことになる．また，各分割回路が FPGA に実装可能な回路規模に収まるとは限らない．

そこで，提案手法では以下の 3 点を分割点候補とし，Java プログラムへの分割処理を行う．

(1) インスタンスに対するメソッド呼び出し
(2) 同一クラス内におけるメソッド呼び出し
(3) メソッド内部のステートメント（for 文，while 文，if 文，switch 文）

分割点候補 (1) は実行モジュールの切替え地点である．JavaRock の生成回路では，メソッド呼び出し時に実行制御が行われ，モジュール間で通信を行う．この通信時において，実行速度が低速なモジュールがボトルネックとなり，全体の動作速度が低速化する．したがって，各モジュールの動作速度を考慮することが望ましい．

分割点候補 (2) は分割点候補 (1) の場合と同様である．同一モジュール内においてメソッド間の通信が行われ，実行速度が低速なメソッドによりモジュール自体の動作速度が低速化する問題に対応する．

文献 [10], [11] では，プログラムのモジュールや関数を

ブロックとして回路分割を行うため，ブロックの回路規模が単一の FPGA に格納できないほど大きくなる場合には対処できない．この点は，クラスをベースに回路分割を行う文献 [12] も同様である．提案手法では，分割点候補 (3) でメソッド内部を分割し，FPGA に格納できる規模に制限する．

3.2 分割領域の決定

次に，3 つの分割点候補によって分割されたプログラムをブロック（以下では，分割ブロックと呼ぶ）として考え，統合アルゴリズムに適用する．統合アルゴリズムでは，最小単位に分けられた分割ブロックを組み合わせ，分割ブロックのグループを作成する．その結果，プログラムはグループに分けられ，分割領域が決定される．ここでは，分割ブロックの統合結果に対する回路規模を閾値（ユーザが指定した回路規模）以下に制限する．また，処理を行ううえでの分割ブロックにには，そのブロックに付与される WPCV が含まれる．

まず，ステートメントにより区切られた分割ブロックである分割点候補 (3) に対する処理を行う．このとき，処理の対象となる分割ブロックは，同一クラスに所属するものであり，回路規模が最小となる分割ブロックが統合対象とされる．ここでは，回路規模と動作周波数の観点から，統合処理が可能か否かの判断を行う．回路規模による制限は，分割ブロックを FPGA に実装可能な回路規模に収めるための条件である．一方，動作周波数による制限は，分割ブロックが仕様どおりの動作周波数で動作することを規定するための条件である．ボトルネックとなる処理を特定することで，HDL への書き換えや Java ソースコードのアルゴリズムの変更といった選択が可能となる．ある統合対象と連続する分割ブロックに対して，統合した場合の回路規模をおよび動作周波数を算出し，条件を満たす場合は，統合処理を実施する．各クラスの分割ブロックに対する統合の余地がなくなるまで処理は継続する．

続いて，Java プログラム内のメソッド呼び出しによって区切られた分割点候補 (1) および分割点候補 (2) の分割ブロックを処理対象とする．ここでは，以下の手順に従い，統合処理を行う．

Step1：メソッドの呼び出し元および呼び出し先の分割ブロックをまとめる．以下では，分割ブロックを $a_1, \ldots, a_n$ と表現し，分割ブロックの集合を $A(A = a_1, \ldots, a_n)$ と表現する．

Step2：A に属する各分割ブロックの回路規模を比較し，回路規模が最小となる分割ブロック（a_i）を探索する．

Step3：a_i をメソッドの呼び出し元（呼び出し先）とする分割ブロック（a_*）にまとめて a_i' とし，JavaRock で回路を作成する．この回路を論理合成ツールに適用し，a_i' の回路規模と動作周波数を算出する．

Step4：a_i' が回路規模および動作周波数による条件を満たす場合は，A から a_i と a_* を除き，a_i' を A に追加する．条件を満たさない場合は，A から a_i を除き，分割済み分割ブロックの集合 B に追加する．

Step5：A に属する分割ブロックが存在する（$n(A) > 1$）場合は，Step2 へ移行する．条件が満たされない場合は，処理を終了し，B の分割ブロックに回路が分割される．

回路分割では，回路規模だけでなく，I/O ブロック数を考慮することが一般的である．しかしながら，提案手法では動作検証のための WPCV を自動生成することにより，部分回路単体でのテストを可能とするため，回路動作をチェックするために回路全体を動作させる必要はない．したがって，分割回路を実装した複数 FPGA 間の通信は行われず，I/O ブロック数に対する考慮が不要となる．自動生成される WPCV の詳細に関しては次節で説明する．

3.3 動作検証用の WPCV

提案手法では，時系列に沿った回路分割を行うため，各部分回路での動作は，本来の Java プログラムの動作と一致する必要がある．本来の Java プログラムは連続しているため，プログラム内での変数情報は共有される．つまり，ある部分回路での処理結果が，次の部分回路の処理に反映される必要がある．

この部分回路間のデータ依存に対するために，WPCV を自動生成し，分割点前後の変数情報に対する整合性を保証する．生成する WPCV の概念図を**図 2** に示す．WPCV の実体は，初期値入力部と変数値比較部により構成される Java プログラムであり，このプログラムを高位合成ツールで合成することで回路に変換される．各部位の役割は以下のとおりである．

初期値入力部：各部分回路に対する初期値を入力する．

変数値比較部：各部分回路に対する処理が終了した時点の変数値が正しいことを確認する．各変数値が Java コンパイル実行後の値と一致する場合は True，不一致の場合は False を出力する．

JavaRock では，入力された Java プログラムをソフトウェアとして動作させることによるアルゴリズムレベル

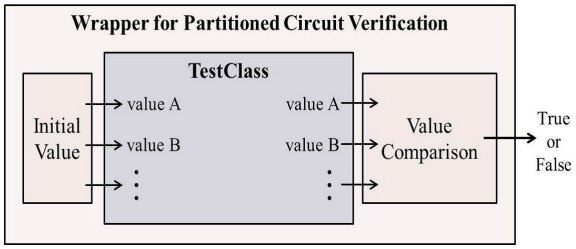


図 2 WPCV の概念図

Fig. 2 Conceptual diagram of the Wrapper for Partitioned Circuit Verification.

でのデバックが可能であるが，合成された回路が正常に動作することを保証するものではない．提案手法では，JavaRock が合成した回路に対して時系列に沿った分割を行い，WPCV を付加することで，各部分回路の処理をシミュレータおよび FPGA を用いてテストできる．また，各部分回路の性能を確認することで，ボトルネックとなる処理をつきとめることができる．

分割した回路に動作検証用の WPCV を与えたことで，部分回路を独立してテストできるため，複数 FPGA 間での通信は不要となり，I/O ブロック数による制約から解放される．通信インタフェースを実装する手間はかからないため，大規模回路のためのテスト環境の構築は容易となる．

4. 回路分割検証機構の構造

本章では，実装および評価を行った回路分割検証機構の詳細について述べる．回路分割検証機構の利用形態を**図 3** に示す．実装した回路分割検証機構では入力ファイルとして，Java ファイル群と設定ファイルを用いる．

設定ファイルでは，使用可能な FPGA デバイス名や FPGA のリソース使用率，分割回路の動作周波数，回路分割検証機構の動作モードを指定する．FPGA のリソース使用量が大きくなると，論理合成時間の長時間化および動作周波数の低速化を招く．そこで，回路分割検証機構では，分割後の各部分回路に対する FPGA のリソース使用量をユーザが任意に指定したリソース使用率以下に制限する．また，分割回路に対する動作周波数は，設計段階で規定された動作周波数以上となるように制限される．Java ファイルには，アプリケーションプログラムが記述されており，JavaRock に適用可能である．したがって，この Java ファイルを JavaRock で合成することで，VHDL ファイルを取得できる．また，回路分割検証機構は，合成モードとテストモードを備える．合成モードでは，入力された Java ファイルを JavaRock に適用し，JavaRock 本来の動作を行う．一方，テストモードでは，JavaRock が合成した回路を部分回路に分割し，動作検証用の WPCV を生成する．

回路分割検証機構は，入力された Java ファイルと設定ファイルをもとに回路分割を行う．分割後，入力プログラムを変換した回路において処理が実行される順番で番号付けを行い，Java ファイルと VHDL ファイルをフォルダ分けして出力する．このとき，Java ファイルに記述されるのは，その時点において実行される分割ソースコードである．VHDL ファイルは，この Java ファイルを JavaRock で合成した際の出力である．また，ユーザ向けレポートが同時に出力される．ユーザレポートでは，各分割回路が使用する FPGA のリソース使用量などの情報をユーザに提供する．

回路分割検証機構の構成を**図 4** に示す．はじめに，入力として与えられた Java ファイルをパーサで解析する．提案手法において，各分割ブロックにおける変数情報の取得は不可欠である．しかしながら，ある時点における変数情報を Java ソースコードから見積もることは困難である．そこで，パーサでは各分割ブロックに対する Java コンパイルを実行し，該当プログラムに対する実行後の変数情報の取得を行う．ここでは，Oracle 社が提供する Java の開発環境 [17] における javac および java コマンドを使用する．取得した Java プログラムの変数情報は，分割回路に対して付与する WPCV で活用され，分割回路の初期値の設定や動作確認が行われる．パーサは解析を行った後，Java の情報をリスト生成部に渡す．

リスト生成部において，Java の情報は整理される．時系列に沿ってリストを生成し，各分割ブロックにおける変数の初期値と終了値を設定する．また，各分割ブロックに対応する入力プログラムの処理や分割ブロック間の接続関係が記載される．作成されたリストは，次の分割プログラムに渡される．

分割プログラムにおいて回路分割が行われるが，Java の情報から JavaRock が生成する回路の規模を見積もることは困難である．そこで，HDL に変換するために JavaRock で合成し，合成後の回路を論理合成ツールに適用する．回路分割検証機構では，Xilinx 社の XST（Xilinx Synthesis Technology）[18] を呼び出す．XST は論理合成結果をレ

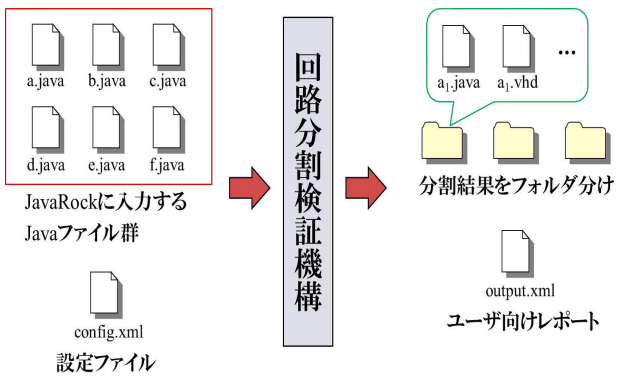


図 3 回路分割検証機構の利用形態

Fig. 3 Utilization form of the test mechanism for partitioned circuits.

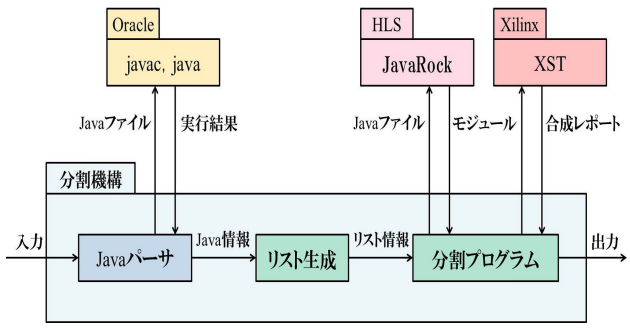


図 4 回路分割検証機構の構成

Fig. 4 Configuration of the test mechanism for partitioned circuits.

ポートとして出力し，レポートには，レジスタ数や LUT 数，BlockRAM 数などの回路規模に関する情報が記載される．このレポートから回路規模の情報を取得し，リストに情報を追加する．そして，分割処理に応じてリストの情報を更新し，適宜 JavaRock や XST に適用することで，回路が分割される．

5. 評価

5.1 評価環境

本実験で用いる FPGA は Xilinx 社の FPGA である XC3S1200E である．JavaRock で合成した回路に対する回路分割を行い，回路規模を XC3S1200E のリソース以下に制限する．XC3S1200E のリソース量は**表 1** に示す．実験に用いた PC のスペックは，Intel Core i5-750 Processor 2.66 GHz，メインメモリは 4 GB，OS は Windows7 である．論理合成には XilinxISE14.4 を使用し，シミュレーションには ISim14.4 を使用した．

実装した回路分割検証機構の評価を行うため，FFT を実行するプログラムを作成した．この FFT のプログラムを JavaRock で合成した回路を評価用回路とする．評価用回路が使用する FPGA のリソースおよび動作周波数は**表 2** に示す．スライスの利用率が 100%を超えているため，評価用回路を 1 つの FPGA に格納することは不可能である．この回路に対して回路分割を行い，回路規模を 75%以下に制限する場合と，回路規模が 50%以下に制限する場合の 2 通りの実験を行った．このとき，分割された回路の動作周波数は 50 MHz 以上とする．

5.2 評価結果

評価用回路に対して，閾値を 75%および 50%に設定し，回路分割を行った結果が**図 5** と**図 6** である．また，それぞれの条件で分割を行った際の分割結果を**表 3** と**表 4** に示す．元となった Java プログラムにおいて実行される処理順に回路が分割され，各部分回路の回路規模は閾値以下に制限される．ここでは，各部分回路に対する統合処理をさらに実行した場合，閾値で設定された回路規模を上回る構成となっている．また，各分割回路の動作周波数は 50 MHz 以上である．

次に，各分割結果に対する動作をシミュレーションおよび実機上で確認する．ここでは，閾値を 50%に設定した際の分割結果である partition1 のシミュレーション上での動作を**図 7** に示す．partition1 は，2 つのモジュールにより構成されており，処理が終了した時点の変数値比較が動作検証のための WPCV により行われる．WPCV に内包される各モジュールのテストが行われ，図 7 中において True が出力された．また，WPCV が出力する信号を XC3S1200E の LED に接続し，実機上での動作確認を行った結果，LED が点灯した．したがって，partition1 に該当する回路はシミュレータと FPGA 上で正常に動作することが確認された．

partition1 以外の分割結果に対しても同様にシミュレーションおよび実機上で動作テストを行った結果，評価用回路全体が正常に動作することが確認された．

表 1 XC3S1200E のリソース量
Table 1 Amount of resources in XC3S1200E.

	個数
スライス数	8,672
レジスタ数	17,344
LUT 数	17,344
Block RAM 数	28

表 2 評価用回路の特徴
Table 2 Features of resources in the target circuit for evaluation.

	評価用回路	FPGA リソース	利用率
スライス数	9,066	8,672	104%
レジスタ数	5,966	17,344	34%
LUT 数	17,047	17,344	98%
動作周波数	49 MHz	-	-

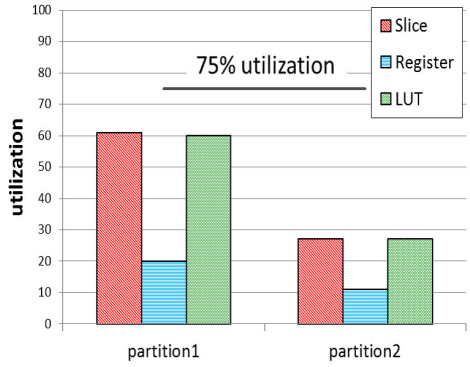


図 5 回路規模を 75%以下に制限した場合の利用率
Fig. 5 Utilization in the case of the limitation on 75% or less.

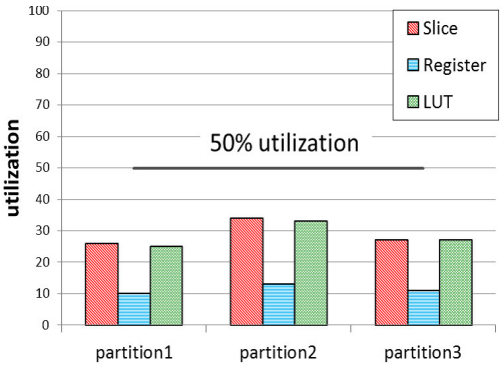


図 6 回路規模を 50%に制限した場合の利用率
Fig. 6 Utilization in the case of the limitation on 50% or less.

表 3 回路規模を 75%以下に制限した場合のリソース量と動作周波数
Table 3 Amount of resources and operation frequency in the case of the limitation on 75% or less.

	スライス数	レジスタ数	LUT 数	動作周波数 [MHz]
partition1	5,362	3,535	10,423	70
partition2	2,425	2,035	4,800	86

表 4 回路規模を 50%以下に制限した場合のリソース量と動作周波数
Table 4 Amount of resources and operation frequency in the case of the limitation on 50% or less.

	スライス数	レジスタ数	LUT 数	動作周波数 [MHz]
partition1	2,300	1,903	4,470	79
partition2	3,019	2,277	5,860	71
partition3	2,425	2,035	4,800	86

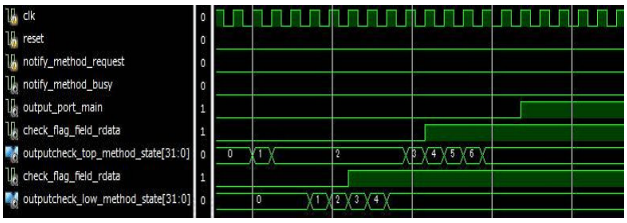


図 7 シミュレーション上での動作
Fig. 7 Operation in the simulation.

6. 考察

6.1 回路規模の増減

以上の結果から，入力された Java プログラムを合成した回路に対する回路分割が行われるといえるが，分割後の回路規模は分割前の回路規模とは異なる値をとる．分割後の回路規模が変化した要因として，以下の 3 点が考えられる．

(1) 回路検証用の WPCV を実装したこと
(2) 分割後の回路に対するリソースの共有が不十分であること
(3) 実行制御機構が削減されたこと

1 点目は，WPCV が分割前の回路には存在しないモジュールであることに起因する．また，内包されるモジュールの複雑化にともない，WPCV 自体が複雑化するため，リソースの消費は大きなものとなる．2 点目は，論理合成ツールによる最適化が十分に行われないことによる．本来ならば，同一機能を有する回路に対しては，リソース共有が行われる．しかしながら，分割後の回路が異なる集合に属した場合，リソース共有は不可能となり，同一機能の回路を複数用意することとなる．そのため，回路規模は増大する．一方，3 点目は，分割回路における実行モジュールの切替えが削減されたことを意味し，リソースは削減される．JavaRock の合成回路における実行制御機構は，リクエスト信号やビジー信号を利用したポーリングにより実現されるため，リソースの消費は大きなものとなる．提案手法では，回路の実行状態を考慮した回路分割を行うことで，実行制御機構は最小限に抑えられる．

以上のように，分割前の回路と比較して回路規模は変化しているが，シミュレータによるシミュレーションは実行可能である．FPGA に実装する場合においても，分割後の回路はより多くの FPGA のリソースを扱うことが可能であるため，回路規模の増大による影響は小さいと考えられる．

シミュレーションを行うことで，回路動作をテストすることが可能であるが，実機上で回路が正常に動作することを保証できない．そこで，FPGA に対する分割回路の実装が行われるが，I/O ブロック数制約を考慮する必要があり，回路全体を動作させることは容易ではない．提案手法では，各部分回路は回路検証用の WPCV に内包されるため，独立して FPGA に実装可能である．したがって，分割回路全体を動作させる必要はなく，通信インタフェースを実装する手間はかからない．そのため，部分回路のテストは容易なものとなる．また，LED や Chipscope を活用し，WPCV の出力信号を観測することで，部分回路に対するデバックは簡単化される．

6.2 WPCV が占有するリソース

本研究では，分割された回路の動作をテストするために，各分割回路ごとに WPCV が与えられる．しかしながら，この WPCV の実装によりリソースは増大する．ここでは，評価で使用した回路規模を 75%以下に制限した分割結果を利用し，WPCV が占有するリソースに関する考察を行う．以下の**表 5** では，WPCV の実装によるリソースの変化を示す．

WPCV では，処理を実行する回路に対する初期値を与え，処理終了時の変数値を比較する．したがって，処理回路において扱われる変数が多いほど，WPCV が占有するリソースは大きくなると考えられる．また，WPCV によって内包される回路構成の複雑さもリソースに影響を与える

表 5 WPCV の回路規模
Table 5 Circuit size of the WPCV.

	項目	WPCV なし		WPCV あり		リソースの増加量
partition1	スライス数	4,074	46%	5,362	61%	+15%
	レジスタ数	2,905	16%	3,535	20%	+4%
	LUT 数	7,605	43%	10,423	60%	+17%
partition2	スライス数	1,563	18%	2,425	27%	+9%
	レジスタ数	1,263	7%	2,035	11%	+4%
	LUT 数	2,970	17%	4,800	27%	+10%

と考えられる．回路が複数のモジュールによって構成されるならば，WPCV はそれぞれのモジュールに対する動作確認を行う必要が生じるため，WPCV は複雑化する．そのため，WPCV によるリソース消費は大きなものとなる．以上のことから，分割回路に実装される WPCV の構成は，その回路固有のものとなり，分割回路に与えられる WPCV は異なる．

近年では，集積回路技術の進歩により搭載可能な回路規模は増大している．そのため，今回の評価では Spartan-3E を用いたが，近年の容量の大きい FPGA を用いれば，WPCV によるリソース消費の増加の影響はそれほど大きくはないと考えている．

6.3 部分回路への分割

FPGA を用いた回路検証において，回路の動作に不具合が発生した場合，問題個所を特定することは困難である．FPGA 上で動作する信号を可視化できないため，シミュレーションを用いて回路の動作を確認する必要がある．しかしながら，大規模回路のシミュレーションに要する時間は膨大であり，実用的ではない．

一方，提案手法では，部分回路単位での検証が可能であり，動作不良を起こしている回路の切り分けが容易である．WPCV の出力信号を FPGA の LED に接続することで，部分回路レベルでの可視化が可能である．また，提案手法では，FPGA のリソース使用率による分割が可能であるため，不具合のある回路をさらに分割し，問題箇所を特定できる．したがって，分割した部分回路に対するシミュレーションを行うだけでよく，回路のテストに要する時間は短縮されると考えられる．

また，部分回路への分割により，ボトルネックとなる処理を探索できる．該当部分に対する HDL への書き換えや Java ソースコードのアルゴリズムの変更をユーザは選択でき，FPGA の処理能力を生かすことが可能である．

6.4 分割後の動作周波数

提案手法によって，分割された回路に対する動作周波数は表 3 と表 4 のとおりである．これらの表より，分割前の回路との比較において，分割後の回路における動作周波数が向上することが観測される．分割前の回路は，論理合成ツールの最適化によりリソースが共有されるため，回路構成が複雑である．一方，分割後の回路は，割り当てられたリソースを占有することが可能である．また，JavaRock の合成回路における実行制御機構が削減されたため，回路構成は単純化する．以上により，クロック周期が削減されたと考えられる．

6.5 分割範囲の指定

回路分割検証機構による分割処理では，分割後の回路規模を見積もるために，論理合成を実行する必要がある．そのため，複雑な Java プログラムを JavaRock で合成した場合に，実行時間は長大化すると考えられる．この問題に対応するために，回路分割検証機構では分割範囲を指定するためのアノテーションを提供する．アノテーションにより指定された範囲を対象に分割処理が行われるため，回路全体に対して実行した場合と比較して，実行時間は短縮される．また，設計の段階で，仕様変更が行われたクラスやメソッドが関係する処理を指定し，検証を行うといった柔軟な対応が可能となる．

6.6 他の高位合成ツールに対する分割手法への適用

現在，高位合成ツールを対象として提案されている回路分割手法 [10], [11], [12] では，回路を分割することにとどまっているため，複数 FPGA に分割実装された回路の評価は行われていない．これらの回路分割手法が対象とする CyberWorkBench や JavaRock のような高位合成ツールでは，入力となるプログラムをソフトウェアとして動作させることが可能である．そのため，C 言語や Java 言語の入力プログラムを実行することで，作成される回路のアルゴリズムレベルでの動作確認を行うことができる．しかしながら，高位合成ツールにより生成した回路を分割する際には，プログラムの分割位置にまたがって共有される変数に対する考慮は不可欠である．それゆえに，分割回路の動作を検証するうえで，単純なアルゴリズムレベルでの動作確認だけでは不十分である．

以上をふまえて，これらの分割手法に対して本研究を適用することが有効であると考える．つまり，高位合成ツー

ルに対する分割手法を活用して最適な分割点の探索を行い，その分割点で分割した回路に対する動作検証を実施する．本研究では，分割した時点での変数情報を考慮した回路分割を行うため，分割回路間の共有変数を含めたテストを行うことが可能である．したがって，本研究を活用し，十分に分割回路のテストを行った後に，複数 FPGA 実装を行うことで，大規模回路の実装に要する負荷は削減されると考えられる．

7. まとめ

本論文では，高位合成ツールによる合成回路の検証を容易にする回路分割検証機構について述べた．回路分割検証機構により分割された Java プログラムを JavaRock により合成し，回路検証用の WPCV を自動生成することで，部分回路単位でのテストを可能とした．実験では，Java で記述した FFT プログラムを JavaRock で合成した回路を評価対象とし，指定した FPGA のリソース使用量が 50%および 75%に収まるように回路を分割した．また，各分割回路をシミュレータと FPGA 上で動作させ，すべての分割回路が正常に動作することを確認した．

今後の課題としては，提案手法を他の高位合成ツールに適用し，有効性を確認することがあげられる．提案手法を適用可能な高位合成ツールの条件として，以下の 2 点が考えられる．

(1) 入力プログラムをソフトウェアとして実行できること
(2) 高位合成ツールが作成した回路に実行制御が存在すること

条件を満たす高位合成ツールとして，JavaRock-Thrash [19] と Synthesijer [20] があげられ，現在，これらに提案手法の適用を検討している．

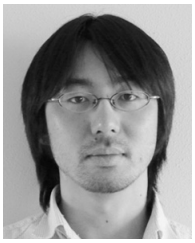
謝辞 本研究の一部は，文部科学省特別経費「持続可能社会にむけた知的情報空間技術の創出」および JSPS 科研費基盤研究（C）25330067 および 16K00078 による支援を得た．ここに記して感謝する．

参考文献

[1] SystemC, available from ⟨http://www.systemcjapan.com/⟩ (accessed 2016-06-01).
[2] Impulse accelerated technology, available from ⟨http://www.impulseaccelerated.com/⟩ (accessed 2016-06-01).
[3] CyberWorkBench, available from ⟨http://jpn.nec.com/cyberworkbench/⟩ (accessed 2016-06-01).
[4] Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J.H., Brown, S. and Czajkowski, T.: LegUp: High-level synthesis for FPGA-based processor/accelerator systems, *Proc. 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp.33–36 (2011).
[5] Auerbach, J., Bacon, D.F., Cheng, P. and Rabbah, R.: Lime: A Java-compatible and synthesizable language for heterogeneous architectures, *Proc. ACM International Conference on Object Oriented Programming Systems Languages and Applications*, pp.89–108 (2010).
[6] JavaRock, available from ⟨http://javarock.sourceforge.net/⟩ (accessed 2016-06-01).
[7] Kernighan, B.W. and Lin, S.: An Efficient Heuristic Procedure for Partitioning Graphs, *Bell System Technical Hournal*, Vol.49, No.2, pp.291–308 (1970).
[8] Fiduccia, C.M. and Mattheyses, R.M.: A linear-time heuristic for improving network partitions, *DAC '82 Proc. 19th Design Automation Conference*, pp.175–181 (1982).
[9] Babb, J., Tessier, R. and Agarwal, A.: Virtual wires: Overcoming pin limitations in FPGA-based logic emulators, *Proc. IEEE Workshop on FPGA-based Custom Computing Machines*, pp.142–151 (1993).
[10] Kugami, D., Miyajima, T. and Amano, H.: A Circuit Division Method for High-Level Synthesis on Multi-FPGA Systems, *Proc. 2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pp.156–161 (2013).
[11] 國上太旗，宮島敬明，天野英晴：高位合成を用いたストリーム処理におけるマルチ FPGA システム向け回路分割手法の提案，電子情報通信学会技術研究報告，Vol.113, No.324, pp.53–58 (2013).
[12] 松田和也，三好健文，船田悟史，中條拓伯：Java 言語ベース高位合成ツールを用いた回路分割方式の実装と評価，情報処理学会論文誌，Vol.56, No.8, pp.1582–1592 (2015).
[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，第 53 回プログラミング・シンポジウム予稿集，Vol.2012, pp.59–68 (2012).
[14] 三好健文，船田悟史：JavaRock を用いた HW/SW の協調設計の検討，電子情報通信学会技術研究報告 AI，Vol.112, pp.119–124 (2012).
[15] 榎戸健二，三好健文，小池恵介，船田悟史，藤波香織，中條拓伯：高位合成系 JavaRock による Reconfigurable Android におけるハードウェア・アクセラレーション，情報処理学会論文誌「組込みシステム工学」特集号，Vol.55, No.2, pp.1027–1036 (2014).
[16] 植竹大地，大川　猛，三好健文，横田隆史，大津金光：高位合成ツール JavaRock による倒立振子制御処理の高速化，電子情報通信学会技術研究報告 Reconfigurable Systems，Vol.113, pp.55–60 (2013).
[17] Oracle, available from ⟨http://www.oracle.com/⟩ (accessed 2016-06-01).
[18] Xilinx, available from ⟨http://japan.xilinx.com/⟩ (accessed 2016-06-01).
[19] 小池恵介，三好健文，船田悟史，中條拓伯：Java 言語ベース高位合成ツール JavaRock-Thrash の開発，組込みシステムシンポジウム 2013 論文集，Vol.2013, pp.41–48 (2013).
[20] 三好健文：企業における FPGA アプリケーション研究開発の一例，電子情報通信学会技術研究報告 Reconfigurable Systems，Vol.114, pp.51–56 (2014).

松田 和也

2014 年東京農工大学工学部情報工学科卒業．2016 年同大学大学院工学府博士前期課程情報工学専攻修了．現在，三菱電機勤務．



三好 健文 （正会員）

2007 年東京工業大・物理情報システム専攻修了．博士（工学）．同年東京大学情報理工学系研究科特任助教．東京工業大学情報理工学研究科産学官連携研究員，電気通信大学大学院情報システム学研究科助教を経て，2012 年株式会社イーツリーズ・ジャパンに入社，2014 年わさらぼ合同会社を設立，現在に至る．FPGA や GPU を活用したアクセラレータ，HW/SW 協調設計に関する研究・開発およびコンパイラ技術に関する研究に従事．2011 年より Synthesijer の開発を開始，オープンソースで公開中．

竹本 正志 （正会員）

2000 年株式会社ビート・クラフトを設立し，代表取締役に就任．以降，同社にてコンシューマエレクトロニクス用ソフトウェアを開発．2006 年からはハードウェアの設計も手がけ，ソフトウェア・エンジニアのためのハードウェアプラットフォームの開発を続けている．現在，東京農工大学大学院工学府博士後期課程電子・情報工学専攻在学中．

船田 悟史

1999 年東京工業大学情報理工学研究科計算工学専攻博士前期課程修了．2000 年有限会社イーツリーズ・ジャパンの設立に参画，取締役専務就任，FPGA による超高速 HTTP キャッシュサーバを開発．2003 年東京工業大学情報理工学研究科計算工学専攻博士後期課程単位取得退学．2008 年株式会社イーツリーズ・ジャパン代表取締役社長に就任．現在，FPGA を活用したアクセラレータに関する研究開発に従事．

中條 拓伯 （正会員）

1985 年神戸大学工学部電気工学科卒業．1987 年同大学大学院工学研究科電子工学専攻修了．1989 年同大学工学部助手を経て，1998 年より 1 年間 Illinois 大学 Urbana-Champaign 校 Center for Supercomputing Research and Development（CSRD）にて Visiting Research Assistant Professor を経て，1999 年より東京農工大学大学院准教授．プロセッサアーキテクチャ，組込みシステム，リコンフィギャラブルコンピューティングに関する研究に従事．電子情報通信学会，IEEE-CS，ACM 各会員．博士（工学）．