パイプライン化された同期式RTLモデルから非同期式RTLモデルへの変換手法の検討

仙波 翔吾[1,a)]　齋藤 寛[1,b)]

概要：本稿では，パイプライン化された同期式 Register Transfer Level (RTL) モデルから束データ方式による非同期式 RTL モデルへの変換手法を提案する．提案する RTL 変換手法は，同期式 RTL モデルのパイプラインステージを解析し，パイプラインステージ数に応じて，非同期式制御モジュールを割り当てる．制御モジュールの割り当て後，グローバルクロック信号の代わりに，割り当てた制御モジュールで各パイプラインレジスタの制御を行うことで，非同期式 RTL モデルを実現する．実験では，5 つのベンチマーク回路に対して，提案する変換手法を用いて同期式 RTL モデルから非同期式 RTL モデルを生成し，論理シミュレーションによる機能検証を行った．

A Study on a Conversion Method from Pipelined Synchronous RTL Models into Asynchronous RTL Models

Shogo Semba[1,a)]　Hiroshi Saito[1,b)]

Abstract: In this paper, we propose a conversion method from pipelined synchronous Register Transfer Level (RTL) models into asynchronous RTL models with bundled-data implementation. The proposed RTL conversion method analyzes pipeline stages in given synchronous RTL models and assigns asynchronous control modules for each pipeline stage. After assigning the control modules, the proposed RTL conversion method implements asynchronous RTL models by controlling each pipeline register by using the assigned control modules instead of global clock signals. In the experiment, we convert synchronous RTL models into asynchronous RTL models for five benchmark circuits using the proposed RTL conversion method. We also verify the functions of the asynchronous RTL models using logic simulation.

1. はじめに

現在のコンピュータシステムで用いられているデジタル回路は，同期式回路である．同期式回路は，グローバルなクロック信号を用いて回路部品を制御する．しかしながら，半導体微細化技術の進歩により，同期式回路ではクロックネットワークにおける消費電力の増加が問題となっている．これは，高周波数なクロック信号を広範囲に分配することが原因である．また，クロックスキューによる回路部品の同期の失敗も問題である．

一方，非同期式回路は，ローカルなハンドシェイク信号や自己タイミングで回路部品を制御する．非同期式回路は，グローバルなクロック信号がないために，同期式回路と比べて潜在的に低消費電力で低電磁放射といった特徴を持つ．しかしながら，非同期式回路の設計は同期式回路の設計と比べて困難である．例えば，アプリケーションに応じて，適切なデータエンコーディング，ハンドシェイクプロトコル，及び遅延モデルを選択しなければならなく，それらの選択により設計制約や設計手法が異なってくる．また，非同期式回路の設計を支援する Electronic Design Automation (EDA) ツールが少ないといった問題もある．

非同期式回路の設計を容易にするために，同期式 Gate-Level (GL) ネットリストから非同期式 GL ネットリストへの変換手法 (GL 変換) が提案されてきた [1], [2], [3], [4]．しかしながら，GL 変換手法では，同期式 Register Transfer

[1] 会津大学
The University of Aizu, Fukushima 969–8580, Japan
[a)] d8211108@u-aizu.ac.jp
[b)] hiroshis@u-aizu.ac.jp

Level (RTL) モデルに対して論理合成を行うために，非同期式回路の性質を利用した論理最適化ができない．

このような GL 変換手法の問題点を改善するために，我々は同期式 RTL モデルから非同期式 RTL モデルへの変換手法 (RTL 変換) を提案した [5]．RTL 変換手法では，論理合成時に非同期式回路の性質を利用した論理最適化を実現することができる．しかしながら，この RTL 変換手法は，パイプライン化された同期式 RTL モデルを変換することができないといった課題がある．

本稿では，パイプライン化された同期式 RTL モデルから束データ方式による非同期式 RTL モデルへの変換手法を提案する．提案する RTL 変換手法は，[5] を拡張したもので，同期式 RTL モデルのパイプラインステージを解析し，パイプラインステージ数に応じて，非同期式制御モジュールを割り当てる．制御モジュールの割り当て後，グローバルクロック信号の代わりに，割り当てた制御モジュールで各パイプラインレジスタの制御を行うことで，非同期式 RTL モデルを実現する．

本稿の構成は，以下の通りである．2 節では，本稿で用いる束データ方式による非同期式回路モデルについて述べる．3 節では，[5] で提案した RTL 変換手法の概要を述べる．4 節では，パイプライン化された同期式 RTL モデルに対する RTL 変換手法について述べる．5 節では，提案手法に対する実験結果について述べる．最後に，6 節で結論を述べる．

2. 束データ方式による非同期式回路

束データ方式は，非同期式回路のエンコーディング手法の一つである．束データ方式では，N ビットのデータを要求信号 req と応答信号 ack を加えた $N+2$ 本の信号線で表現する．また，データパスの最大遅延より大きな遅延を持った遅延素子にてレジスタへのデータの書き込みタイミングを保証する．そのため，束データ方式による非同期式回路の性能は，遅延素子を含んだ制御回路に依存する．

図 1(a) は，本稿で用いる束データ方式による非同期式回路の回路モデルである．この回路モデルは，データパス回路と制御回路から構成される．

データパス回路は，同期式回路と同じで，レジスタ reg_k，マルチプレクサ mux_l，演算器 fu_h から構成される．reg_k でホールド違反が起きた場合，reg_k の入力信号線上に遅延素子 hd_{reg_k} を挿入する．

制御回路は，1 つのパイプラインステージ $stage_i$ $(0 \leq i \leq n-1)$ に対して，1 つの制御モジュール $ctrl_i$ を割り当てることで実現する．$glue_{reg_k}$ は，複数の $stage$ にて書き込みが行われる reg_k を制御する論理である．$glue_{mux_l}$ は，複数の $stage$ にて制御される mux_l を制御する論理である．mux_l の制御信号の遷移にてレジスタのホールド違反が起きた場合，mux_l の制御信号線上に遅延素子 $hd_{mux_{i,l}}$

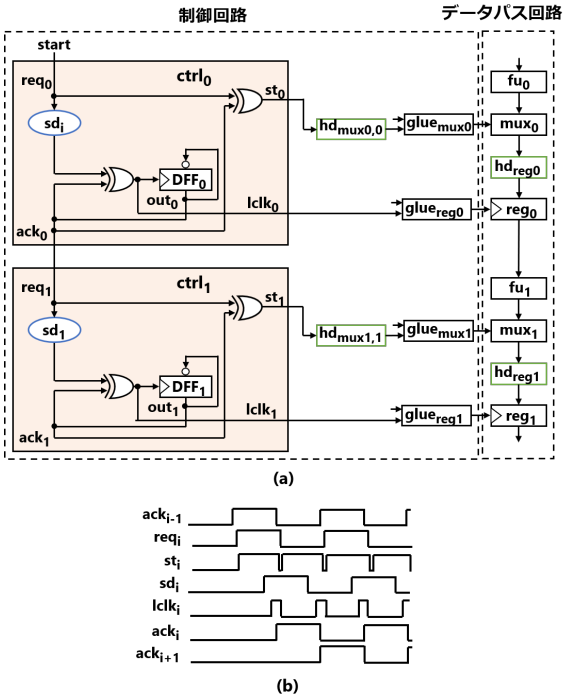

図 1 束データ方式による非同期式回路モデル: (a) 回路構造，(b) 制御モジュール $ctrl_i$ の動作波形．

を挿入する．

$ctrl_i$ は，D フリップフロップ DFF_i，XOR ゲート，遅延素子 sd_i，及び $lclk_i$ を生成する論理から構成される．$ctrl_i$ は，Click エレメント [6] を改良したもので，応答信号を前の制御モジュールに戻すのではなく，次の制御モジュールに対する要求信号として使う．そのため，各 $ctrl_i$ は sd_i による自己タイミングで動作する．sd_i は，レジスタのセットアップ制約を含んだ書き込みタイミングを保証する遅延素子である．この制御モジュールでは，req_i は二相式で，$lclk_i$ は四相式である．二相式は，信号の立ち上がり遷移と立ち下がり遷移に対する区別がなく，両方の遷移で動作する．一方，四相式は，信号の立ち上がり遷移と立ち下がり遷移を区別し，どちらかの遷移で動作する．この回路モデルでは，$lclk_i$ の立ち上がり遷移にてレジスタにデータを書き込む．

本稿で使用する回路モデルは，外部入力信号 $start$ の立ち上がり遷移で動作を開始する．ここからは，信号 $signal$ の立ち上がり遷移を $signal+$，立ち下がり遷移を $signal-$ で表現する．図 1(b) は，制御モジュール $ctrl_i$ の動作波形を表す．$ctrl_i$ は，直前の制御モジュール $ctrl_{i-1}$ からの $ack_{i-1}+$ を受け取り動作を開始する．この信号遷移は，req_i+ となる．その後，req_i+ は，XOR ゲートを通り st_i+ となり，$glue_{mux_l}$ を介して mux_l を制御する．また，req_i+ は，sd_i と XOR ゲートを通り $lclk_i+$ となり，$glue_{reg_k}$ を介して reg_k を制御する．さらに，$lclk_i+$ は reg_k を制御すると同時に DFF_i も制御する．DFF_i は out_i+ を生成し，次の制御モジュール $ctrl_{i+1}$ へ制御を移す．最後に，out_i+

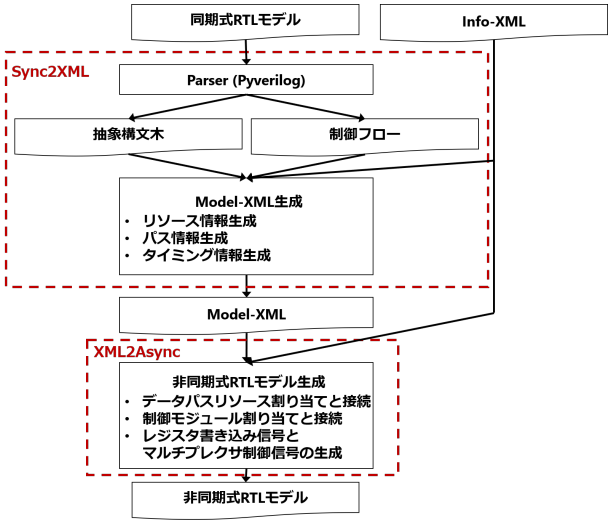

図 2　RTL 変換手法のフロー.

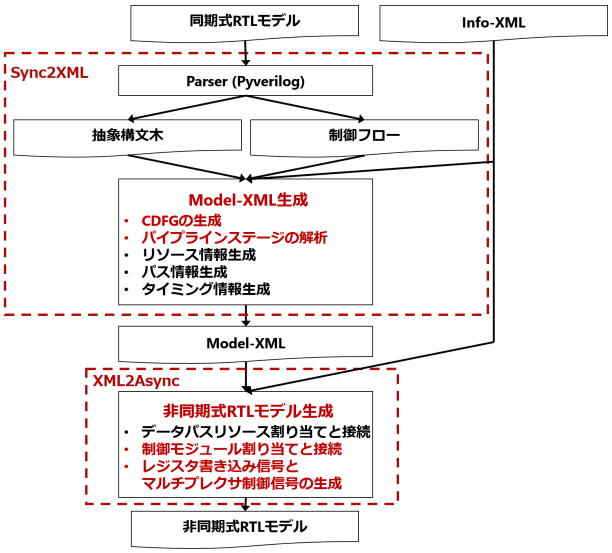

図 3　拡張後の RTL 変換手法のフロー.

を用いて $lclk_i-$ と st_i- を生成する．なお，req_i- における動作も同様である．

3. RTL 変換手法

図 2 は，[5] で提案した RTL 変換手法を表す．本稿における提案手法は，この RTL 変換手法を基にしている．RTL 変換手法は，*Sync2XML* と *XML2Async* と呼ばれる 2 つのパートから構成される．

Sync2XML は，Verilog Hardware Description Language (HDL) で記述された同期式 RTL モデルから *Pyverilog*[7] を用いて抽象構文木と制御フローを生成し，抽象構文木と制御フローから中間表現として eXtensible Markup Language (XML) を生成する．抽象構文木は RTL モデルの構造を表し，制御フローはステートマシンの状態遷移を表す．生成した XML は，Model-XML と呼ぶ．Model-XML は，同期式 RTL モデルのデータパスリソース情報，データパスと制御パスを含んだパス情報，データパスリソースの制御タイミング情報から構成される．また，RTL 変換手法は，Info-XML と呼ばれるパラメータファイルも入力として必要となる．Info-XML は，グローバルクロック信号名や制御モジュールの構成に使用するプリミティブセルなどの情報から構成される．なお，RTL 変換手法は様々な表現による同期式 RTL モデルに対応するため，中間表現として XML を使用している．

XML2Async は，*Sync2XML* にて生成した Model-XML から Verilog HDL で記述された束データ方式による非同期式 RTL モデルを生成する．*XML2Async* は，Model-XML のリソース情報とパス情報を基に，データパスリソースと制御モジュール $ctrl_i$ を割り当てる．また，Model-XML のパス情報を基に，*XML2Async* は各データパスリソースの接続と各制御モジュールの接続を行う．最後に，*XML2Async* は Model-XML のタイミング情報を基に，レジスタ書き込み信号とマルチプレクサ制御信号を生成することで制御回路とデータパス回路を接続し，トップレベルモジュールを生成する．

4. 提案手法

提案手法は，パイプライン化された同期式 RTL モデルを非同期式 RTL モデルに変換するために，[5] で提案した RTL 変換手法に対して，以下の拡張を行う．

- *Sync2XML* での拡張
  - コントロールデータフローグラフ (CDFG) の生成
  - パイプラインステージの解析
- *XML2Async* での拡張
  - パイプライン制御モジュールの割り当てと接続
  - レジスタ書き込み信号とマルチプレクサ制御信号の生成

図 3 は，提案手法のフローを表す．本節では，初めに対象となる同期式 RTL モデルを 4.1 節で解説し，次に拡張の詳細を 4.2 節と 4.3 節で解説する．

4.1 対象となるパイプライン化された同期式 RTL モデル

提案手法は，Verilog HDL で記述された同期式 RTL モデルを変換対象とする．データパス回路は 2 節で述べたように，レジスタ，演算器，マルチプレクサなどで構成される必要がある．一方で，制御回路は同期式 RTL モデル内に 1 つでなければいけない．また，提案手法は，”function”，”task”，”for”，”while”，”wait”，”[sub_z,5’ h0+:32]” などの構文は扱うことができない．これらの構文に対応することは，今後の課題である．

対象となる同期式 RTL モデルには，入力インターバル数による制限はない．入力インターバルは，パイプライン回路に入力データが投入される間隔を表し，入力インターバルによってスループットが変わる．図 4 は，対象となる

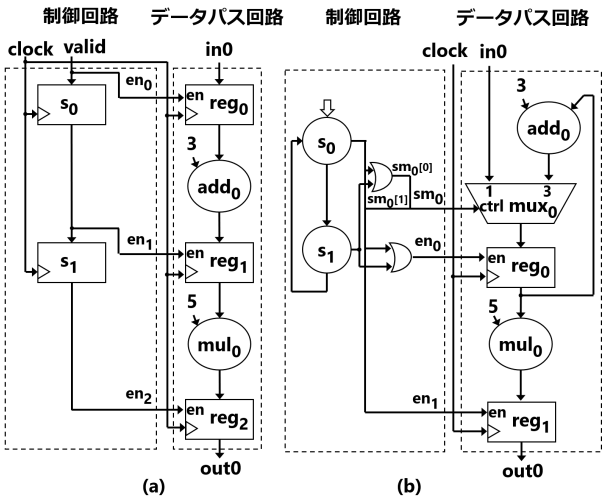

図 4　パイプライン化された同期式 RTL モデルの回路モデル: (a) 入力インターバル 1，(b) 入力インターバル 2.

パイプライン化された同期式 RTL モデルの例を表す．図 4(a) は入力インターバルが 1 の回路モデルを表し，図 4(b) は入力インターバルが 2 の回路モデルを表す．インターバルが 2 の場合は，リソースの共有があるため，マルチプレクサを利用すると共にステートマシンにてマルチプレクサを制御している．

提案手法は，レジスタに対するクロックゲーティングの有り無しは考慮しない．また，提案手法はパイプラインストールが含まれていない同期式 RTL モデルを対象とする．パイプラインストールに対応することは，今後の課題である．

4.2　*Sync2XML* の拡張

非同期式回路では，各パイプラインステージ $stage_i$ を各制御モジュールで制御するため，各 $stage_i$ で制御されるべきレジスタやマルチプレクサを知る必要がある．しかしながら，抽象構文木は RTL モデルの構造のみを表し，制御フローはステートマシンの状態遷移のみを表しているため，抽象構文木と制御フローのいずれかから各 $stage_i$ で制御されるべきレジスタやマルチプレクサを知ることができない．そのため，提案手法は抽象構文木と制御フローから一度 CDFG を生成し，生成した CDFG にて各パイプラインステージを解析する．パイプラインステージの解析後，*Sync2XML* は，データパスリソース情報，データパス情報，前後のパイプラインステージ情報，及び各パイプラインステージで制御されるレジスタやマルチプレクサの制御信号を含んだタイミング情報を生成する．

4.2.1　CDFG の生成

本稿で用いる CDFG は，同期式 RTL モデルの制御フローとデータフローを表したもので，図 5(c) に示す通り，ノードとエッジ，及びパイプラインステージから構成される．ノードはレジスタや演算器といったリソースを表し，

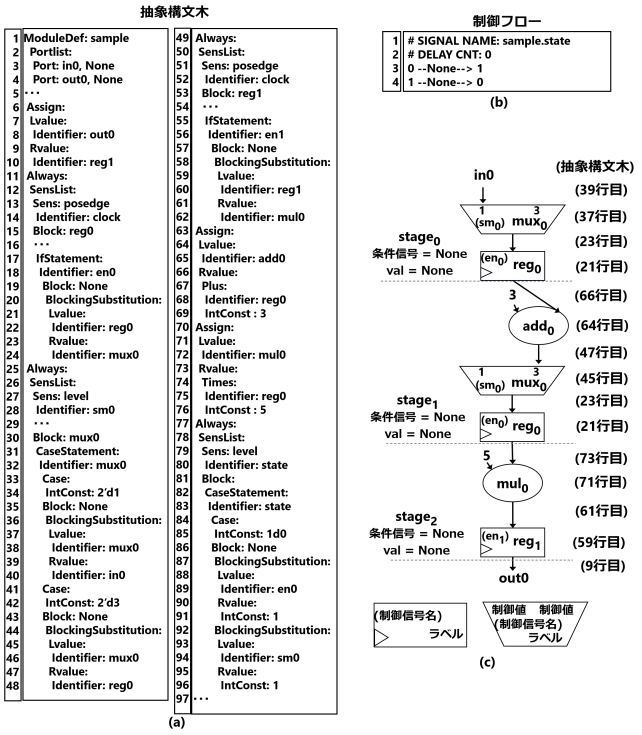

図 5　図 4(b) に対する CDFG の生成例: (a) 抽象構文木，(b) 制御フロー，(c) CDFG.

エッジはリソース間の接続を表す．なお，ノードはリソース名，制御信号名，及び制御信号値を有する．また，レジスタ間はパイプラインステージを表す．さらに，パイプラインステージは，そのパイプラインステージが動作するための条件が存在する場合，条件信号名とその値を有する．

提案手法は，*Sync2XML* にて抽象構文木や制御フローを基に CDFG を生成する．CDFG のノードは，抽象構文木の”Lvalue”や”Instance”より生成する．また，”Lvalue”や”Instance”に記載されている変数名がノードのラベルとなる．一方，生成したノードの”Lvalue”や”Instance”に対する”IfStatement”や”CaseStatement”から生成したノードに対する制御信号名や制御信号値を抽出する．エッジは，抽象構文木の”Rvalue”や”PortArg”より生成する．パイプラインステージは，レジスタの書き込み信号に対応する”Lvalue”と”Rvalue”，及び制御フローの状態遷移を基に決める．また，生成したパイプラインステージ順に，$stage_0$，$stage_1$ とラベルを付ける．さらに，制御フローの状態遷移に対する条件信号からパイプラインステージの条件信号とその値を抽出する．

図 5 は，図 4(b) の同期式 RTL モデルに対する抽象構文木の一部と制御フロー，及び生成した CDFG を表す．ノードの生成に関しては，例えば，抽象構文木の 21 行目における”Lvalue”からレジスタ reg_0 をラベルとしたノードを生成する．また，抽象構文木の 17 行目の”IfStatement”から reg_0 の書き込み信号を en_0 と判断し，reg_0 に対するノードに持たせる．エッジの生成に関しては，例えば，抽象構

文木の 23 行目の”Rvalue”からマルチプレクサ mux_0 から reg_0 へのエッジを生成する．パイプラインステージの決定に関しては，例えば，制御フローの初期状態 (状態 0) で，reg_0 の書き込み信号 en_0(抽象構文木の 88 行目の”Lvalue”) が 1(抽象構文木の 90 行目の”Rvalue”) となるため，入力信号 $in0$ から reg_0 までがパイプラインステージ $stage_0$ となる．また，制御フローにおける状態 0 への状態遷移に対して条件信号 (制御フローの 4 行目の None) がないために，$stage_0$ が動作するための条件信号名とその値はない．図 5(c) は，最終的に得られた CDFG を表す．

4.2.2 パイプラインステージの解析

CDFG の生成後，Model-XML のパイプラインステージ情報とタイミング情報を生成するために，CDFG を基にパイプラインステージ毎に，前後のパイプラインステージ，レジスタ書き込み信号とマルチプレクサ制御信号，及びそれらの値を解析する．

前後のパイプラインステージの解析では，各 $stage_i$ の直前直後に他のパイプラインステージがあるかどうかの解析を行う．CDFG 上で，$stage_i$ のリソースから他のパイプラインステージ $stage_j(j \neq i)$ のリソースへ接続がある場合は，$stage_j$ は $stage_i$ の直後のパイプラインステージとなる．一方，$stage_j$ のリソースから $stage_i$ のリソースへ接続がある場合は，$stage_j$ は $stage_i$ の直前のパイプラインステージとなる．パイプラインステージ間の遷移に条件がある場合は，条件信号とその値を取得する．

レジスタ書き込み信号やマルチプレクサ制御信号の解析では，各 $stage_i$ におけるレジスタ書き込み信号とマルチプレクサ制御信号の値を解析する．CDFG 上の $stage_i$ に reg_k がある場合は，reg_k の書き込み信号は $stage_i$ で 1 になる．一方，CDFG 上の $stage_i$ に mux_l がある場合は，mux_l の入力となっているリソースを選択する制御値が mux_l の制御信号値となる．

解析後，*Sync2XML* は Model-XML のパイプラインステージ情報とタイミング情報を生成する．パイプラインステージ情報の生成では，1 つの $stage_i$ に対して 1 つのパイプラインステージ情報 $\langle ctrl \rangle$ を生成する．また，前後のパイプラインステージの解析から直前のパイプラインステージ情報 $\langle pred \rangle$ と直後のパイプラインステージ情報 $\langle succ \rangle$ を $\langle ctrl \rangle$ 内に生成する．なお，直前のパイプラインステージがない場合は，$\langle pred \rangle$ に外部入力信号 $start$ を与える．さらに，直前直後のパイプラインステージが動作するための条件信号とその値も $\langle pred \rangle$ や $\langle succ \rangle$ 内に $ctrlname$, $ctrlvalue$ として生成する．一方，タイミング情報の生成では，レジスタ書き込み信号とマルチプレクサ制御信号の解析によって得た $stage_i$ 毎の制御信号値からレジスタ書き込み信号情報 $\langle reg \rangle$ とマルチプレクサ制御信号情報 $\langle mux \rangle$ を生成する．なお，マルチプレクサの制御信号値は 2 進数で表記する．

パス情報（パイプラインステージ情報）

```
<ctrlpath>
 <ctrl id="0" name="stage0">
  <pred id="0" name="start" ctrlname="" ctrlvalue=""/>
  <succ id="0" name="stage1" ctrlname="" ctrlvalue=""/>
 </ctrl>
 <ctrl id="1" name="stage1">
  <pred id="0" name="stage0" ctrlname="" ctrlvalue=""/>
  <succ id="0" name="stage2" ctrlname="" ctrlvalue=""/>
 </ctrl>
 <ctrl id="2" name="stage2">
  <pred id="0" name="stage1" ctrlname="" ctrlvalue=""/>
 </ctrl>
</ctrlpath>
```

(a)

タイミング情報

```
<reg id="0" name="en0" stage0="1" stage1="1" stage2="0"/>
<reg id="1" name="en1" stage0="0" stage1="0" stage2="1"/>
<mux id="0" name="sm0" stage0="01" stage1="11" stage2="00"/>
```

(b)

図 6 図 5(c) の CDFG から生成した Model-XML: (a) 制御パス情報，(b) タイミング情報.

図 6(a) と (b) は, 図 5(c) の CDFG を基に生成した Model-XML のパイプラインステージ情報とタイミング情報を表す．パイプラインステージ情報の生成では，*Sync2XML* は 3 つの $\langle ctrl \rangle$ を生成する．$stege_0$ では，$\langle pred \rangle$ に外部入力信号 $start$ を与え，$\langle succ \rangle$ に $stege_1$ を与える．また，$stege_1$ が動作するための条件信号名とその値はないため，$\langle succ \rangle$ の $ctrlname$ と $ctrlvalue$ は空白としている．他のパイプラインステージに対する直前直後のパイプラインステージ情報も同様である．タイミング情報の生成では，CDFG 上の $stage_0$ と $stage_1$ に reg_0 があるため，reg_0 に対する書き込み信号 en_0 に対してレジスタ書き込み信号情報 $\langle reg \rangle$ を生成すると共に，$stage_0$ と $stage_1$ で書き込み信号を 1 とする．他の制御信号も同様である．

4.3 *XML2Async* の拡張

XML2Async は，生成した Model-XML を基に，パイプライン制御モジュールの割り当てと接続，及びレジスタ書き込み信号とマルチプレクサ制御信号の生成を行う．

4.3.1 パイプライン制御モジュールの割り当てと接続

XML2Async は，*Sync2XML* にて生成した Model-XML の $\langle ctrl \rangle$ 毎に，制御モジュール $ctrl_i$ を割り当てる．また，$\langle ctrl \rangle$ 内にある $\langle pred \rangle$ と $\langle succ \rangle$ を基に，各制御モジュールを接続する．

図 7(a) は，図 6(a) における Model-XML の $\langle ctrl \rangle$ を基に割り当てられた制御モジュールとその接続関係を表す．*XML2Async* は，3 つの $\langle ctrl \rangle$ に対して，制御モジュール $ctrl_0$，$ctrl_1$，及び $ctrl_2$ を割り当てる．$ctrl_0$ に対応する $\langle ctrl \rangle$ の $\langle pred \rangle$ と $\langle succ \rangle$ より，外部入力信号 $start$ と次の制御モジュール $ctrl_1$ を接続する．他の制御モジュール間の接続も同様である．

4.3.2 レジスタ書き込み信号とマルチプレクサ制御信号の生成

XML2Async は，*Sync2XML* にて生成した Model-XML

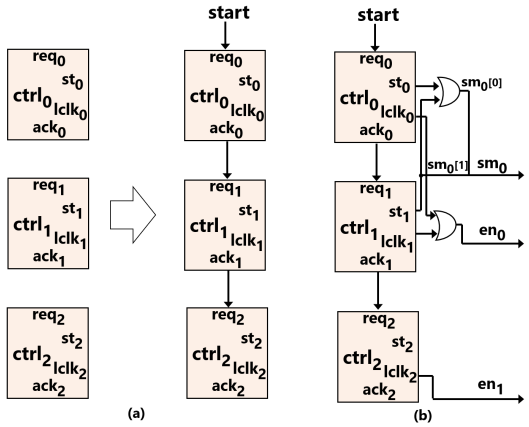

図 7　制御回路: (a) 制御モジュールの割り当てと接続，(b) レジスタ書き込み信号とマルチプレクサ制御信号の生成.

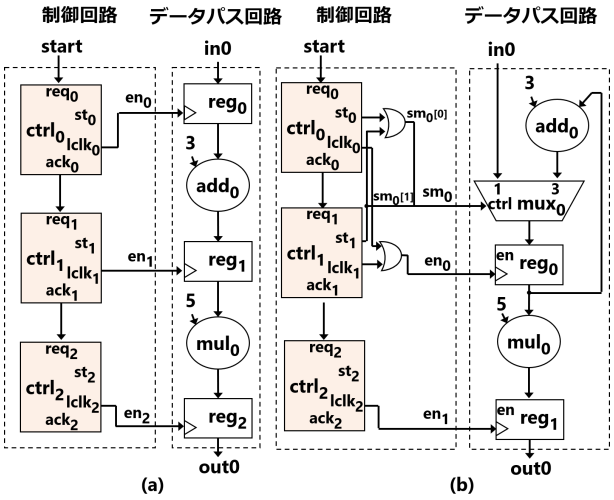

図 8　非同期式 RTL モデル: (a) 図 4(a) に対する非同期式モデル，(b) 図 4(b) に対する非同期式モデル.

の $\langle reg \rangle$ 毎にレジスタ書き込み信号を生成し，$\langle mux \rangle$ 毎にマルチプレクサ制御信号を生成する．レジスタ書き込み信号の生成では，$\langle reg \rangle$ 内の $name$ を基に信号名を決定し，$stage_i$ が 1 であるところを基に $ctrl_i$ からの $lclk_i$ 信号の OR をとることにより，レジスタ書き込み信号を生成する．マルチプレクサ制御信号の生成では，$\langle mux \rangle$ 内の $name$ を基に信号名を決定し，$stage_i$ が 1 であるところを基に $ctrl_i$ からの st_i 信号の OR をとることにより，マルチプレクサ制御信号を生成する．

図 7(b) は，図 6(b) における Model-XML の $\langle reg \rangle$ と $\langle mux \rangle$ を基に生成したレジスタ書き込み信号とマルチプレクサ制御信号を表す．*XML2Async* は，2 つの $\langle reg \rangle$ に対してレジスタ書き込み信号 en_0 と en_1 を生成し，1 つの $\langle mux \rangle$ に対してマルチプレクサ制御信号 sm_0 を生成する．en_0 に対応する $\langle reg \rangle$ の $stage_0$ と $stage_1$ の値より，$ctrl_0$ と $ctrl_1$ からの $lclk$ 信号の OR をとることで en_0 を生成する．同様に，レジスタ書き込み信号 en_1 とマルチプレクサ制御信号 sm_0 を生成する．

生成した制御信号とデータパス回路の接続にて，変換された非同期式 RTL モデルを得る．図 8(a)，(b) はそれぞれ，図 4(a)，(b) の同期式 RTL モデルを提案手法にて変換した非同期式 RTL モデルを表す．

5. 実験結果

実験では，5 つの同期式 RTL モデルを提案手法を用いて非同期式 RTL モデルへと変換した．また，提案手法を Java にて実装し，Windows 10 マシン (Intel Core i7-8700 の 3.2GHz の CPU と 16GB メモリ) にて実験を行った．

実験で用いた同期式 RTL モデルは，SystemC モデルから Cadence 社の Stratus HLS 18.1 を用いて高位合成を行って得たもので，微分方程式を解く回路 (DIFFEQ)，楕円フィルター (EWF)，ニューロン数を 32 に変更したマルチレイヤーパーセプトロン (MLP)[8]，Tiny Encryption アルゴリズム (TEA)[9]，及び Advanced Encryption Standard (AES)[10] である．なお，クロックゲーティングのオプションを適用した上で，高位合成を行った．ライブラリは，eShuttle の 65nm プロセスを用いた．

表 1 は，変換結果を表す．$Name$，II，CT，$Stage$，及び $Sverilog$ は，同期式 RTL モデルの名前，入力インターバル数，クロックサイクルタイム，パイプラインステージ数，同期式 RTL モデルの Verilog HDL における行数を表す．一方，AST，$Model\text{-}XML$，$Averilog$，$Time$ は，抽象構文木の行数，生成された Model-XML の行数，非同期式 RTL モデルの Verilog HDL における行数，及び変換時間を表す．なお，クロックサイクルタイムは，タイミング違反のない最速のクロックサイクルタイムである．表 1 より，変換時間は同期式 RTL モデルのパイプラインステージ数や抽象構文木の行数に依存することがわかる．

提案手法により変換された非同期式 RTL モデルの機能の正しさを示すために，入力データ 100 回分を含んだ任意のテストベンチを準備し，Synopsys 社の VCS Q-2020.03-SP1 を用いて論理シミュレーションを行った．なお，レジスタに対するデータの書き込みを確認するために，遅延素子の遅延は正確な値でなければならない．そのため，非同期式回路を一旦 Cadence 社の Genus 18.1 を用いて論理合成し，Standard Delay Format (SDF) を生成した．生成した SDF ファイルを用いてシミュレーションを行うことで機能検証を行った．シミュレーション後，非同期式 RTL モデルの全ての出力信号値が，同期式 RTL モデルの出力信号値と同じであることを確認した．

提案手法により変換された非同期式 RTL モデル (async) の品質を確認するために，DIFFEQ，EWF，及び MLP に対して，[11] の設計フローを基に論理設計までを行った．その際，同期式回路 (sync) と同等の性能を得るために，クロックサイクルタイムを基に，パスに対して最大遅延制約やパイプラインステージ毎にローカルクロック制約を生成して

表 1　変換結果.

Name	*II*	*CT* [ps]	*Stage*	*Sverilog* [行数]	*AST* [行数]	*Model-XML* [行数]	*Averilog* [行数]	*Time* [秒]
DIFFEQ	1	1,200	4	164	765	110	359	2.5
	2	1,200	3	127	398	88	380	1.9
EWF	1	1,200	9	668	3,202	429	1,345	2.6
	7	1,200	8	390	1,957	313	1,131	2.4
MLP	1	400	20	16,925	94,130	22,272	36,668	322.9
TEA	1	600	383	20,379	97,688	12,902	42,055	179.7
AES	1	600	41	130,270	947,246	191,781	139,104	3081.2

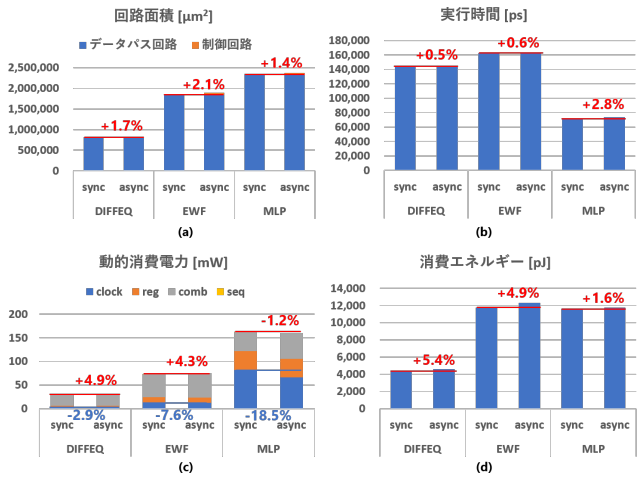

図 9　回路評価: (a) 回路面積, (b) 実行時間, (c) 動的消費電力, (d) 消費エネルギー.

いる．DIFFEQ，EWF，MLP の論理合成で用いたクロックサイクルタイムは，それぞれ $1,400ps$，$1,500ps$，$600ps$ である．この値は，タイミング違反のない最速のクロックサイクルタイムである．論理設計後，Genus による面積レポートにて回路面積の評価を行い，論理シミュレーションにて実行時間の評価を行った．また，論理シミュレーション時に Switching Activity Interchange Format (SAIF) を生成し，Synopsys 社の Prime Time Q-2019.12-SP3 を用いて動的消費電力の評価を行った．さらに，実行時間と動的消費電力の積により消費エネルギーの評価を行った．

図 9 は，回路面積，実行時間，動的消費電力，及び消費エネルギーを表す．非同期式回路の回路面積は，平均で 1.7%増加した．これは，制御回路の挿入によるものである．また，非同期式回路の実行時間は，平均で 1.3%増加した．非同期式回路の DIFFEQ，EWF，MLP のサイクルタイムはそれぞれ $1,407ps$，$1,510ps$，$617ps$ であり，同期式回路のサイクルタイムより僅かに大きいことが実行時間増加の原因である．サイクルタイムの増加の原因は，最大遅延制約やローカルクロック制約を与えることで同期式回路と同等のクリティカルパス遅延を得ることができたが，クリティカルパス遅延を超えるように遅延素子を準備したためである．一方，非同期式回路の動的消費電力は，平均で 2.6%増加した．これは，グローバルなクロック信号を使わないことによるクロック電力の削減率より，制御モジュールの挿入による組み合わせ回路における電力の増加率の方が大きかったためである．組み合わせ回路の動的消費電力を削減することは，今後の課題である．最後に，非同期式回路の消費エネルギーは，平均で 3.9%増加した．これは，実行時間の増加と動的消費電力の増加によるものである．

6. 結論

本稿では，パイプライン化された同期式 RTL モデルから束データ方式による非同期式 RTL モデルへの変換手法を提案した．実験では，5 つのパイプライン化された同期式 RTL モデルから非同期式 RTL モデルへ変換した．

今後は，ストール処理を含んだパイプライン回路への対応を検討する．また，組み合わせ回路における動的消費電力削減手法を検討する．

謝辞　本研究は，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し，シノプシス，eShuttle の協力で行われたものである．また，本研究は，JSPS 特別研究員 DC2 の特別研究員奨励費 20J11724 と JSPS 科研費 18K11221 の助成による．

参考文献

[1] J. Cortadella et al., ”Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications”, IEEE TCAD, vol. 25, pp. 1904–1921, 2006.
[2] N. Andrikos et al., ”A Fully-Automated Desynchronization Flow for Synchronous Circuits”, Proc. DAC, pp. 982–985, 2007.
[3] A. Branover et al., ”Asynchronous Design By Conversion: Converting Synchronous Circuits into Asynchronous Ones”, Proc. DATE, pp. 870–875, 2004.
[4] J. Oberg et al., ”Automatic Synthesis of Asynchronous Circuits from Synchronous RTL Description”, Proc. NORCHIP, pp. 200–205, 2005.
[5] S. Semba and H. Saito, ”Conversion from Synchronous RTL Models to Asynchronous RTL Models”, IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, vol. E102-A, No. 7, pp. 904–913, 2019.
[6] Ad Peeters et al., ”Click Elements: An Implementation Style for Data-Driven Compilation”, Proc. ASYNC, pp. 3–14, 2010.

[7] S. Takamaeda-Yamazaki, "Pyverilog: A Python-based Hardware Design Processing Toolkit for Verilog HDL", Proc. ARC, Lecture Notes in Computer Science, Vol.9040/2015, pp.451–460, 2015.

[8] Y. Umuroglu et al., "FINN: A Framework for Fast, Scalable Binarizrd Neural Nwtwork Inference", Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp. 65–74, 2017.

[9] D. Wheeler and R. Needham, "TEA, a Tiny Encryption Algorithm", Fast Software Encryption, 2nd International Workshop Proceedings, Springer-Verlag, pp. 97–110, 1995.

[10] Yuko Hara et al., "Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis", Journal of Information Processing, vol. 17, pp. 242–254, 2009.

[11] S. Semba and H. Saito, "Comparison of RTL Conversion and GL Conversion from Synchronous Circuits to Asynchronous Circuits", Proc. ISCAS, pp. 1–4, 2019.