Vol.2013-SLDM-163 No.13
2013/11/27

# FPGA を対象とした束データ方式による非同期式回路の設計支援ツールセット

滝澤　恵多郎[1,a)]　齋藤　寛[1,b)]

**概要**：本稿では，束データ方式による非同期式回路を FPGA に実装するための設計支援ツールセットを提案する．始めに面積や静的タイミング解析のしやすさを考慮し，プリミティブを用いて制御モジュールを定義する．これらを用いて制御回路を実現する．次に設計制約コマンド生成の自動化，タイミング検証の自動化，タイミング違反時の遅延調整の自動化を行うツールセットを提案する．提案するツールセットと商用の FPGA 設計ツールを使用することにより，FPGA を対象にレイテンシ制約を考慮した束データ方式による非同期式回路設計が容易に行える．実験ではいくつかのベンチマークに対し提案するツールセットを適用し，回路面積，実行時間，消費電力，消費エネルギーの観点から同期式回路との比較を行う．

Keitaro Takizawa[1,a)]　Hiroshi Saito[1,b)]

***Abstract:*** This paper proposes a design support tool set for asynchronous circuits with bundled-data implementation which are implemented on a field programmable gate array (FPGA). First, the control module which is composed of primitives is proposed considering area and ease of static timing analysis. The control circuit is composed of control modules. Next, the tool set which automates generation of delay elements and design constraints, timing verification, and delay adjustment is proposed. By using the proposed tool set with a commercial FPGA design tool, the design of a bundled-data implementation with a latency constraint on an FPGA becomes easy, In the experients, this paper evaluates the synthesized circuits in terms of area, latency, power consumption, and energy consumption for some benchmarks comparing with the synchronous counterparts.

## 1. はじめに

現在の集積回路の大半はグローバルなクロック信号により回路を制御する同期式回路である．しかしながら半導体微細化技術の進歩により，クロックスキューによる同期の失敗やクロックネットワークにおける消費電力の増大などの問題が顕著になってきている．一方，クロック信号を用いずにローカルなハンドシェーク信号で回路を制御する非同期式回路では，クロック信号にまつわる問題が発生しない．また，必要な部分が必要なときのみ動作するため，潜在的に低消費電力・低電磁放射である．

しかしながら，一般的に非同期式回路の設計は同期式回路の設計に比べて難しい．非同期式回路はクロック信号を持たないため，遅延モデルやデータエンコーディング，ハンドシェークプロトコルなどを考慮して設計を行う必要がある．また，予期せぬ信号遷移が誤動作を引き起こす可能性があるため，ハザードフリーであることが要求される．

近年, 組込みシステムの分野で Field Programmable Gate Array (FPGA) の需要が高まっている．これは，設計コストの低さやライフタイムの長さなどに起因する．

商用の FPGA に非同期式回路を実現する研究はこれまでに以下のようなものが行われてきた．Tranchero らは [2] で，商用 FPGA 設計ツールを用いて，FPGA 上に非同期式回路を実現する手法を提案した．この手法では, Simulink[3] ベースの協調設計環境である CodeSimulink [4] から生成された VHDL コードを同期式回路として配置配線まで行い，各論理ブロックの遅延情報を得る．その後，制御を非同期式制御に置き換え，データパスにおける演算の完了を保証するための遅延素子を調整する．遅延素子の遅延が，データパスの遅延にマッチするまで遅延調整と再合成を繰り返す．なお，タイミング解析や遅延調整をサポートするツールは提案されていない．Ho らは [5] で，Xilinx 社や Altera 社の FPGA を対象に，非同期式回路を実現する手法を提案した．この手法では，非同期式回路のなかで最も広く用いられているマラーの C 素子 [6] を 1 つの論理ブロックに実現することによってハザードのないことを示し，Quasi Delay Insensitive[7] モデルによる 4 ビットの加算器に適用した例を報告している．しかし，マラーの C 素子以外に関する言及は乏しく，配置配線やタイミング検証に関する解説もない．

本稿では，Altera 社の FPGA を対象に束データ方式による非同期式回路を実現するための設計支援ツールセットを提案する．提案するツールセットは，商用の FPGA 設計ツールがサポートしていない非同期式回路に対する設計制約生成，タイミング検証，遅延調整の自動化を行う．提案するツールセットは [1] をベースとしている．[1] では ASIC を対象としており，これを Altera 社の FPGA 向けに改良を加える．また，面積や静的タイミング解析のしやすさを考慮し Altera のプリミティブを用いて制御回路を実現する．提案するツールセットと Altera Quartus II を用いることにより，レイテンシ制約を考慮した束データ方式による非同期式回路を FPGA 上に容易に実現することができる．

本稿の構成は以下の通りである．2 節では束データ方式による非同期式回路を，3 節では FPGA について述べる．4 節では提案手法を，5 節では実験に関して延べ，最後に 6 節で結論を述べる．

[1] 会津大学
[a)] m5161141@u-aizu.ac.jp
[b)] hiroshis@u-aizu.ac.jp

## 2. 束データ方式による非同期式回路

### 2.1 回路モデル

束データ方式とは N ビットのデータを表すのに，要求信号 req と応答信号 ack の 2 本を加えた N+2 本の信号線を用いる非同期式回路実装手法の 1 つである．演算の完了は要求信号線上に付加された遅延素子によって保障する．本稿で用いる束データ方式による非同期式回路のモデルを図 1 に示す．モデルの右側はデータパス回路，左側は制御

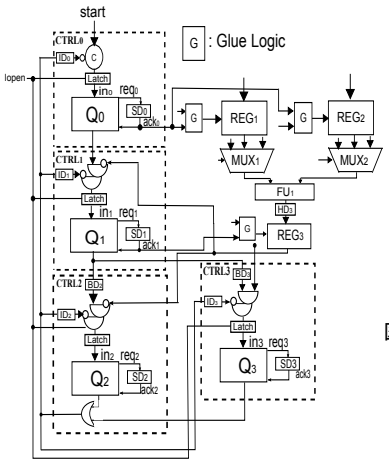
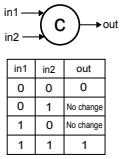



**図 1** 対象とする回路モデル




| in1 | in2 | out |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | No change |
| 1 | 0 | No change |
| 1 | 1 | 1 |

**図 2** 2 入力の C 素子

回路である．データパス回路はレジスタ（$REG$），マルチプレクサ（$MUX$），演算器（$FU$），グルーロジック（$G$）により構成され，同期式回路と同じものを用いる．ホールド制約を保障するための遅延素子 $HD_j(1 \leq j \leq |REG|)$ をレジスタの直前に置く．最初は単に入力信号を出力信号に代入したもので，タイミング検証に応じて調整する．グルーロジックは制御回路からの信号を受け取り，レジスタへの書き込み信号，マルチプレクサの制御信号，演算器の制御信号を生成する．一方，制御回路は，制御モジュール $CTRL_i$（$1 \leq i \leq n$）から構成される．制御モジュール $CTRL_i$ は，Q モジュール $Q_i$ [8]，3 種類の遅延素子 $SD_i$，$BD_i$，$ID_i$，グルーロジックから構成される．$SD_i$ はデータパス回路における演算の終了を保証するために，$BD_i$ は正しい方向への分岐を保証するために，$ID_i$ は制御モジュール $CTRL_i$ の初期化のために伴う制約の保証のために用いる．$SD_i$，$BD_i$，$ID_i$ も最初は入力信号を出力信号に代入したものである．

次に，回路動作について述べる．以降の記述では立ち上がり遷移を $signal+$，立ち下がり遷移を $signal-$ とする．この回路モデルは，外部からの $start+$ で動作を開始する．なお，図 1 の C は C 素子 [6] と呼ばれ，非同期式回路設計で多用される待ち合わせ回路である．図 2 は，2 入力の C 素子のシンボルと真理値表を表す．

制御モジュール $CTRL_i$ は，直前の制御モジュール $CTRL_{i-1}$ から出力された $out_{i-1}+$が $in_i+$となることによって動作を開始する．また，$in_i+$により，データパス回路のマルチプレクサや，演算器の制御信号がグルーロジック $G$ を介して生成される．$in_i+$より，$req_i+ \rightarrow SD_i+ \rightarrow ack_i+$ というパスを通り Q モジュール $Q_i$ に戻る．その後，$req_i- \rightarrow SD_i- \rightarrow ack_i-$ というパスを通り，$ack_i-$ でレジスタにデータが書き込まれる．また，$ack_i-$ で，Q モジュール $Q_i$ は $out_i+$ を生成し，次の制御モジュール $CTRL_{i+1}$ に制御を移す．

最後の制御モジュール $CTRL_n$ が $out_i+$ を生成した後，全ての制御モジュール $CTRL_i$ は，$in_i-$ と $out_i-$を生成する．この時，データパス回路は動作しない．本稿では $in_i-$ から $out_i-$ までの期間を休止相と呼ぶ．なお，Q モジュール $Q_i$ の入力信号 $in_i$ の直前の AND ゲートによって各制御モジュールの休止相はほぼ同時に行うことができる．

### 2.2 タイミング制約

本稿で対象としている束データ方式による非同期式回路は以下の 4 種類のタイミング制約を満たす必要がある．

#### 2.2.1 セットアップ制約

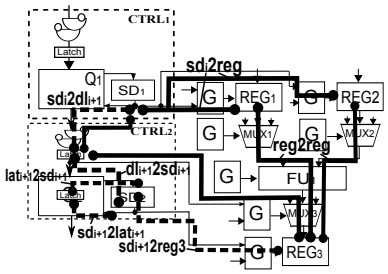



**図 3** セットアップ制約の例

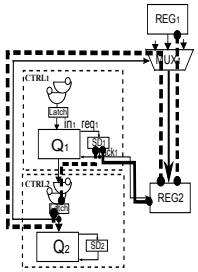



**図 4** ホールド制約の例

レジスタにデータが書き込まれるより一定の時間 (セットアップ時間) 前に書き込まれるデータは安定していなければならない．これをセットアップ制約という．

セットアップ制約の例を図 3 に示す．セットアップパスの始点となるレジスタを制御する $SD_{i-1}$ の出力から制御モジュール $CTRL_i$ の中を通り，セットアップパスの終点となるレジスタのクロックピンに至るパス scp (破線) の最小遅延を $T_{minscp}$，セットアップパスのレジスタを始点となる制御する $SD_{i-1}$ の出力からデータパスを通り，セットアップパスの終点となるレジスタの入力に至るパス sdp (実線) の最大遅延を $T_{maxsdp}$，セットアップ時間を $T_{setup}$，データパスに対するマージンを $sm$ $(sm > 0)$ とすると，セットアップ制約は以下の不等式で表すことができる．

$$T_{minscp} > T_{maxsdp} * sm + T_{setup} \quad (1)$$

この制約に違反する場合， 遅延素子 $SD_i$ を調整する．

#### 2.2.2 ホールド制約

レジスタにデータが書き込まれた後， 一定の時間（ホールド時間）そのデータは安定していなければならない． これをホールド制約という．

ホールド制約の例を図 4 に示す．図 4 では，$ack_1-$ にて $REG_1$ にデータを書き込んでいる．$ack_1-$ から $REG_1$ の入力データピンまでのデータパス hdp (破線) の最小遅延を $T_{minhdp}$，$ack_1-$ から $REG_1$ のクロックピンまでの制御パス hcp (実線) の最大遅延を $T_{maxhcp}$，ホールド時間を $T_{hold}$，$T_{maxhcp}$ に対するマージンを $hm$ $(hm > 0)$ とすると，ホールド制約は以下の不等式で表せる．

$$T_{minhdp} > T_{maxhcp} * hm + T_{hold} \quad (2)$$

この制約に違反する場合， 遅延素子 $HD_j$ を調整する．

#### 2.2.3 分岐制約

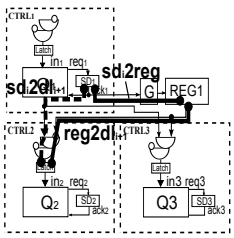



**図 5** 分岐制約の例

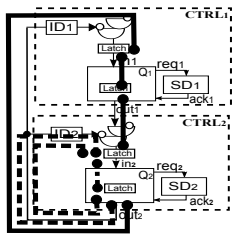



**図 6** 休止相の短縮に伴う制約の例

前段の制御モジュールからの制御信号が到着する前に，分岐判定論理の出力に分岐判定信号が到着していなければならない． これを分岐制約という．

分岐制約の例を図 5 に示す．$CTRL_2$ の後，$REG_3$ の値

によって，$CTRL_3$ か $CTRL_4$ に制御が分岐する．$ack_2-$ より分岐判定 AND ゲートの入力までのパス bcp（破線）の最小遅延を $T_{minbcp}$ とする．一方，$ack_2-$ より，$REG_3$ のクロックピンを経由し，$REG_3$ の出力ピンから分岐判定 AND ゲートの出力までのデータパス bdp (実線) の最大遅延を $T_{maxbdp}$，分岐制約に対するマージンを $bm$ ($bm > 0$) とすると，分岐制約は以下の不等式で表すことができる．

$$T_{minbcp} > T_{maxbdp} * bm \quad (3)$$

この制約に違反する場合， 遅延素子 $BDi$ を調整する．

### 2.2.4 休止相の短縮に伴う制約

使用する回路モデルでは，Q モジュール $Q_i$ の入力信号 $in_i$ の直前に AND ゲートを用いることによって各制御モジュールが並列に休止相を行うことによって，休止相を短縮している，これに伴い，フィードバック信号を生成する制御モジュール $CTRL_n$ の内部を初期化するための最小遅延より，フィードバック信号を受け取る制御モジュール $CTRL_i$ ($1 \leq i \leq n-1$) の内部を初期化して $CTRL_n$ の AND ゲートまでに至るパスの最大遅延のほうが短くなければならない．

休止相の短縮に伴う制約の例を図 6 に示す．$out_2+$ より $CTRL_1$ を通り，$CTRL_2$ の Q モジュール $Q_2$ の直前の AND ゲートの出力ピンまでのパス ifp (図 6 の両端が丸い実線) の最大遅延を $T_{maxifp}$，$out_2+$ より，$CTRL_2$ を通り，$out_2-$ として Q モジュール $Q_2$ の直前の AND ゲートの入力ピンまでのパス ibp (図 6 の破線) の最小遅延を $T_{minibp}$ とすると，休止相短縮に伴う制約は以下の不等式で表すことができる．

$$T_{minibp} > T_{maxifp} \quad (4)$$

この制約に違反する場合， 遅延素子 $IDi$ を調整する．

## 3. FPGA

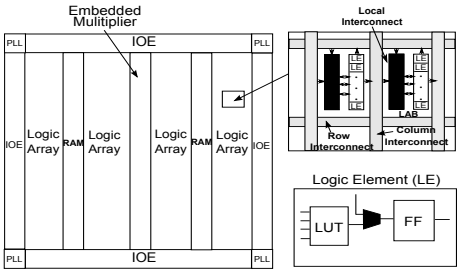



**図 7** Cyclone IV FPGA の構造

FPGA は再構成可能デバイスとも呼ばれ，設計者が回路構成をいつでも書き換えることができる．本稿で用いる Altera 社の FPGA を図 7 に示す．この FPGA は Logic Array，Embedded Multiplier，RAM，IOE，Phase Locked Loop (PLL) などから構成される．Logic Array は 16 個の Logic Element (LE) を持つ Logic Array Block (LAB) から構成される．1 つの LAB は 16 個の LE，同一 LAB 内で LE 間の LE 間の信号を転送する Local Interconnect などから構成される．1 つの LE は Look Up Table (LUT) と呼ばれるプログラム可能な論理，フリップフロップ (FF)，マルチプレクサ (MUX) から構成される．IOE は外部に対する入出力に用いる．Row/Column Interconnect は LAB，IOE，RAM，PLL，Embedded Multiplier に入出力される信号を接続するために用いる．

## 4. 提案手法

本稿で提案するツールセットと Altera 社の Quartus II を利用することによって，レイテンシ制約を考慮した束データ方式による非同期式回路を Altera 社の FPGA に容易に実現することが可能となる．なお，他社の FPGA でも，プリミティブや同じようなツールがあれば同様なことができると考えている．

図 8 に設計フローを示す．この設計フローでは Quartus II による合成を 2 回以上行うことを想定している．1 回目の合成（初期合成）結果より，静的タイミング解析（Static Timing Analysis，STA）を行い，遅延素子やレイテンシ制約を維持するための最大遅延制約を生成する．遅延素子や最大遅延制約を含んだ上で 2 回目の合成を行う．仮に全てのタイミング制約が満足するようであれば終了し，さもなければ遅延素子を調整し全てのタイミング制約が満足するまで合成を繰り返す．以下の各節では，設計フローの入力および各ツールセットの説明を行う．

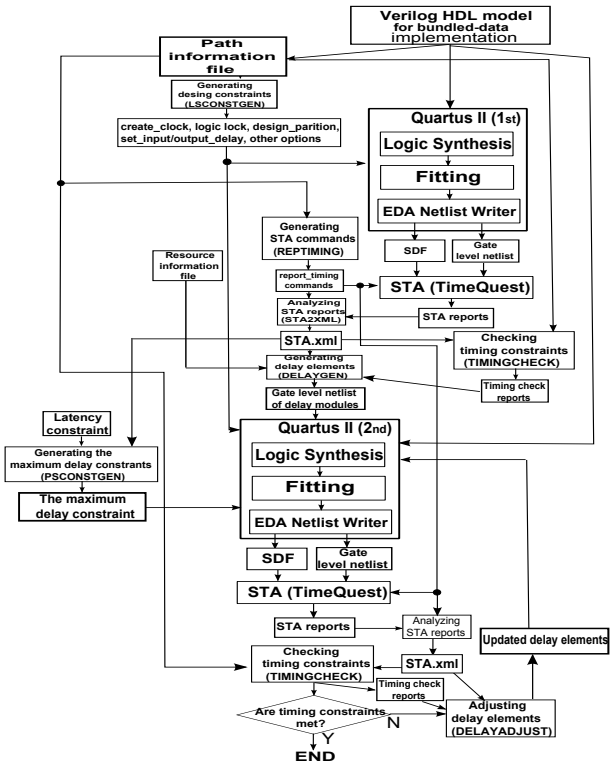



**図 8** 設計フロー

### 4.1 入力

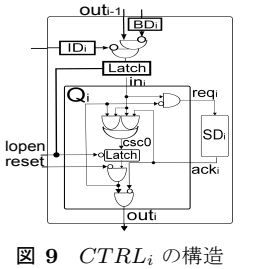



**図 9** $CTRL_i$ の構造

```
module SDi (in, out);
  input in;
  output out/* synthesis keep */;
  wire w0, w1;

  LCELL lc0(.in(in), .out(w0));
  LCELL lc1(.in(w0), .out(w1));
  LCELL lc2(.in(w1), .out(out));
endmodule
```

**図 10** 遅延素子 $SD_i$ の例

この設計フローでは 4 つの入力を必要とする．1 つ目は束データ方式による非同期式回路である．Verilog HDL モデルである．回路部品は部品毎にモジュール化する．

制御モジュール $CTRL_i$ は図 9 のようにモデリングする．オリジナルの Q モジュール [8] には，内部に組み合わせループが存在する．組み合わせループが存在すると，静的タイミング解析ツールが適当なところでループを寸断してしまうため，パス遅延を正しく解析できないといった問題が生じる．そのため，Q モジュールの内部にある C 素子

の出力に lopen という外部からの入力でオープンするラッチを挿入する．Q モジュールの直前の in 信号にも同様の理由でラッチをつける．また，タイミング制約を満たすための $SD_i$，$ID_i$，$BD_i$ は初期合成の段階で制御モジュール内に組み込む．回路構成保持のため，$SDi$ の LCELL の数は最低 1 個は入れておく．$ID_i$，$BD_i$ は最適化されても回路動作に影響はないため，入力信号を単に出力信号に通すモジュールとし，タイミング制約を満たすよう後ほど調整する．

2 つ目は 2.2 節で述べた全てのタイミング制約に関してタイミング検証が必要な全てのパスを表したパス情報ファイルであり，XML 形式で与える．パス情報ファイルの例を表 1 に示す．この例はセットアップパスを表す．*start* と *end* で解析するパスの開始点と終了点の名前を指定する．*source* はソースレジスタを制御する制御モジュールを，*destination* はデスティネーションレジスタを制御する制御モジュールを表す．$T_{io}$ は外部への出力，外部からの入力遅延，*sm* はデータパスに対するマージンを表す．

**表 1** パス情報ファイルの例

| start | end | through | src | dst | $T_{io}$ | *sm* |
|---|---|---|---|---|---|---|
| $IN$ | $REG_0$ | | | $CTRL_0$ | 0 | 1.05 |
| $REG_0$ | $REG_0$ | $ADD_0$ | $CTRL_0$ | $CTRL_1$ | 0 | 1.05 |
| $REG_1$ | $REG_2$ | $MUL_0$ | $CTRL_1$ | $CTRL_2$ | 0 | 1.05 |

3 つ目は遅延素子に用いるセルや制御モジュールのピン名を表したリソース情報ファイルである．表 2 はリソース情報ファイルの例を表す．このファイルも XML 形式で準備する．gate-name は遅延素子用セルの名前で，提案手法では Altera FPGA のプリミティブ LCELL を用いる．delay にはセル 1 個の遅延値を指定する．なお，セル 1 個の遅延値は，LCELL1 個分の素子遅延と LCELL 間の配線遅延を足したものとする．配線遅延をたす理由は，LCELL を 1 個増やすことにより配線数も 1 本増えるためである．なお，LCELL 1 個とそれに付随する配線の遅延は，任意の数の LCELL を配置し，静的タイミング解析を行った時の平均値を用いる．pin-name には $SD_i$ の入出力ピン名を指定する．最後はレイテンシ制約とそれにまつわるパラメータである．レイテンシは，*start*+から最後の制御モジュールによって制御されるレジスタまでの制御遅延の最大値と定義し，その値を制約として与える．他には，レイテンシ制約のうちデータパスを制御する時間の割合を表す制御割合 $CR$ $(0 < CR \leq 1)$ と制御回路における時間のうちデータパスにおける処理時間の割合をデータパス割合 $DR$ $(0 < CR \leq 1)$ として与える．また，遅延調整の際，遅延素子の削減をどの程度まで行うかを定めるスレッショルド値 TH を与える．

**表 2** リソース情報ファイルの例

| gate-name | delay | pin-name | |
|---|---|---|---|
| | | in | out |
| $LCELL$ | 0.42 | in | out |

## 4.2 LSCONSTGEN

LSCONSTGEN は，非同期式回路の合成に必要な制約やオプションを Tcl スクリプトとして生成する．まず，束データ方式の非同期式回路の Verilog モデルを論理合成する際，Quartus II では Verilog HDL で定義した回路の階層を破壊して最適化が行われる．しかし，この最適化によってレジスタの削除やレジスタのリネーミングが起こってしまうとパス情報ファイルで示した始点などの名前と整合性が取れなくなる恐れがある．そのため，Quartus II の機能の 1 つである Design Partition を使用し，回路の階層を維持するよう各モジュールに対して *create_partition* 制約を生成する．

次に，各 $SD_i$ の出力がローカルクロックとしてレジスタを制御するので，各 $SD_i$ の出力に *create_clock* 制約を生成する．*create_clock* 制約によって，各制御モジュール $CTRL_i$ によって制御されるデータパスのレジスタ間最大遅延の和がレイテンシ制約を満たすようにする．指定するクロックサイクルタイムの値はレイテンシ制約 * $CR$ * $R_i$ * $DR$ とする．$R_i$ はレイテンシ制約のうち各制御モジュール $CTRL_i$ における処理時間であり，以下の式より求める．

$$R_i = \frac{T_{maxscp_i}}{\sum_{i=1}^{n} T_{maxscp_i}} \tag{5}$$

ここで，$T_{maxscp_i}$ は，$CTRL_i$ において 5 つに分割された制御パスの最大遅延を表す．他の制約として，パス情報ファイルの $T_{io}$ より *set_input_delay*，*set_output_delay* 制約を生成する．モジュール内のロジックが近傍に配置されるよう，各モジュールに Logic Lock Region 制約を生成する．

オプションとして，デザインに変化がなければ論理合成をスキップし，合成時間を短縮するためのスマートコンパイルを適用するオプションやホールド最適化をオフにするオプションを Tcl コマンドとして生成する．ホールド最適化がオンになっていると Quartus II はクロック信号をベースにホールド制約を満たそうとするが，本研究では TIMINGCHECKER にてホールド制約をチェックし，DELAYADJUST にてホールド制約を満足するよう調整する．

## 4.3 REPTIMING

REPTIMING は，入力として準備したパス情報ファイルの各行から，静的タイミング解析 (STA) のためのコマンドを生成する．セットアップ時間やホールド時間を必要とする解析では *report_timing* コマンドを，それ以外の部分の解析は *report_path* コマンドを使用する．これらは Tcl スクリプトとして出力する．

セットアップ制約に関するパスのコマンドの生成を述べる．図 3 のように制御回路のパス scp を，$sd_i2dl_{i+1}$，$dl_{i+1}2sd_{i+1}$，$sd_{i+1}2lat_{i+1}$，$lat_{i+1}2sd_{i+1}$，$sd_{i+1}$2reg の 5 つのパスに分割し，それぞれの最小遅延を解析する *report_path* コマンドを生成する．データパス sdp は始点レジスタを境に $sd_i$2reg，$reg2reg$ の 2 つのパスに分割し，それぞれの最大遅延を解析する *report_timing* コマンドを生成する．

次に，ホールド制約に関するパスのコマンドの生成を述べる．まず，対象となるレジスタ (図 4 の場合，REG2) の入力ピンに至る全てのパスの最小遅延を解析する *report_timing* コマンドを生成する．一方，そのレジスタを制御する制御モジュールからそのレジスタまでの最大遅延を解析するコマンドはセットアップパスのコマンド生成の際にすでに生成されているのでそれを利用する．

次に，分岐制約に関するパスのコマンドの生成を述べる．図 5 のようにデータパス bdp を $sd_i2reg$，$reg2dl_{i+1}$ の 2 つのパスに分割し，$reg2dl_{i+1}$ の最大遅延を解析する *report_timing* コマンドと制御パス $sd_i2dl_{i+1}$ の最小遅延を解析する *report_path* コマンドを生成する．データパス $sdi2reg$ の最大遅延を解析する *report_path* コマンドはセットアップパスのコマンド生成時に既に生成しているので，それを利用する．

最後に，休止相の短縮に伴う制約に関するパスのコマンドの生成を述べる．まず，最後の制御モジュール $CTRL_n$ の $out_n$ 信号から $CTRL_i$ の中にあるラッチまで，$CTRL_i$ の中にあるラッチから Q モジュールの中にあるラッチまでの最大遅延を解析する *report_path* コマンドを生成する．また，最後の制御モジュール $CTRL_n$ の $out_n$ 信号から $CTRL_n$ の中にあるラッチまで，$CTRL_n$ の中にあるラッチから $CTRL_n$ の Q モジュールのなかにあるラッチまでの最小遅延を解析する *report_path* コマンドを生成する．

### 4.4 STA2XML

STA2XML は，TIMINGCHECKER にてタイミング検証を行うため，TimeQuest Timing Analyzer による STA 結果を XML 形式に変換する．

まず，Quartus II にてコンパイル後に REPTIMING で生成された Tcl スクリプトを TimeQuest Timing Analyzer で実行する．次に，得られたパス毎の STA 結果のテキストレポートから，始点，終点，パス遅延，セットアップタイム，ホールドタイムを抜き出し，XML 形式にて出力する．

### 4.5 TIMINGCHECKER

TIMINGCHECKER は，静的タイミング解析結果を集計した XML ファイル，パス情報ファイル，リソース情報ファイルを入力に，合成された回路が全てのタイミング制約を満たしているかを，制約式 (1)，(2)，(3)，(4) の差（左辺 - 右辺）を計算することで検証する．差が正の値の場合，制約を満たしており，差が負の値の場合，タイミング制約違反が発生している．検証結果を Timing Check Report として出力する．

### 4.6 DELAYGEN

DELAYGEN は，各制御モジュール $CTRL_i$ の遅延素子 $SDi$ を生成する．図 8 のフローでは，1 回目の合成後の STA 結果を基に，DELAYGEN を実行する．

まず，パス情報ファイルの情報を基に，制御モジュール $CTRL_i$ 毎にセットアップ制約を分類し，TIMINGCHECKER の結果，制約式の差が負となったもののうち絶対値が最大となるものを求める．次に，その値から制御モジュール $CTRL_i$ における 5 つのパス $sd_i2dl_{i+1}$，$dl_{i+1}2sd_{i+1}$，$sd_{i+1}2lat_{i+1}$，$lat_{i+1}2sd_{i+1}$，$sd_{i+1}$2reg の最小遅延を引き，2 で割った値を $SD_i$ の値とする．ここで 2 で割る理由はレジスタにデータを書き込むまでに遅延素子 $SD_i$ を 2 回通過するためである．DELAYGEN は，$LCELL$1 個分の値で $SD_i$ の値を割って必要な LCELL の個数を求め，Verilog HDL 記述として遅延素子 $SD_i$ を生成する．図 10 は生成された遅延素子 $SD_i$ の例を表す．$SD_i$ の出力ピン名が変化して適切な静的タイミング解析が行えなくなるのを防ぐため，生成される $SD_i$ の出力に synthesis keep オプションをつける．

### 4.7 PSCONSTGEN

PSCONSTGEN は，与えられたレイテンシ制約を満たすように，セットアップ制約に関する各パスに set_max_delay コマンドによる最大遅延制約を生成し，Synopsys Design Constraint (SDC) ファイルで出力する．DELAYGEN 同様，図 8 のフローでは 1 回目の合成後の STA 結果を基に最大遅延制約を生成する．

制御パス scp への最大遅延制約コマンドの生成について述べる．制御パス scp の最小遅延を $T_{minscp}$ とし，図 3 で分割した 5 つの制御パス $sd_i2dl_{i+1}$，$dl_{i+1}2sd_{i+1}$，$sd_{i+1}2lat_{i+1}$，$lat_{i+1}2sd_{i+1}$，$sd_{i+1}$2reg の最小遅延を $T_{minscp}$ で割った値をそれぞれ $\alpha$, $\beta$，$\gamma$，$\delta$, $\epsilon$ ($\alpha+\beta+\gamma+\delta+\epsilon \leq 1$) とする．それぞれのパスの $set\_max\_delay$ コマンドの値は，レイテンシ制約 $* CR * R_i$ の値に，$\alpha$，$\beta$，$\gamma$，$\delta$，$\epsilon$ を掛けたものとする．

### 4.8 DELAYADJUST

DELAYADJUST は TIMINGCHECKER にて出力された Timing Check Report を入力にタイミング違反が発生したパスおよび遅延が過剰に入った遅延素子 $SD_i$，$HD_i$，$BD_i$，$ID_i$ に対して遅延調整を行い，Verilog HDL を生成する．

まず，ホールド制約違反に対する遅延調整について述べる．ホールド制約違反の遅延調整は，該当するレジスタのデータ入力ピンの直前に制約式の差の絶対値に相当する $LCELL$ バッファから構成される遅延素子 $HD_i$ を挿入する．この調整はデータパスの遅延に影響するため，セットアップ制約違反の遅延調整より先に行う．

分岐制約違反に対する遅延調整の場合，遅延素子 $BD_i$ に制約式の差の絶対値に相当する $LCELL$ バッファを挿入する．この調整もセットアップ制約における制御パスに影響を与えるので，セットアップ制約違反の遅延調整より先に行う．

セットアップ制約違反に対する遅延調整の場合，各制御モジュール $CTRL_i$ からの $ack_i$ 信号にてデータがレジスタに書き込まれる全てのセットアップ制約違反のうち，制約式の左辺-右辺の絶対値が最大となるものを 2 で割った値に相当する分の $LCELL$ バッファを $SD_i$ に新たに追加する．2 で割る理由はレジスタを制御するまでに遅延素子 $SD_i$ を 2 回通るためである．休止相に関する制約違反の場合，制約式の差の絶対値に相当する分の $LCELL$ バッファからなる遅延素子 $ID_i$ を生成する．

次に遅延が過剰となるパスに対する調整を解説する．各制御モジュール $CTRL_i$ に関連する全てのセットアップ制約において，制約式の左辺-右辺の値が全て正の値の場合，差が最小となるものを 2 で割った値に相当する分の $LCELL$ バッファを $SD_i$ から削除する．$HD_j$，$BD_i$，$ID_i$ は制約式の左辺-右辺が正かつ，スレッショルド TH より大きければ遅延素子の削減を行う．制約式の左辺-右辺の値が正の値の場合，最小となるものに相当する分の $LCELL$ バッファを $HD_j$，$BD_i$，あるいは $ID_i$ から削除する．

## 5. 実験

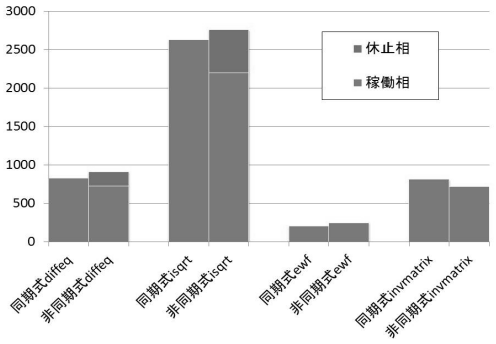



**図 11** 実行時間 [ns]

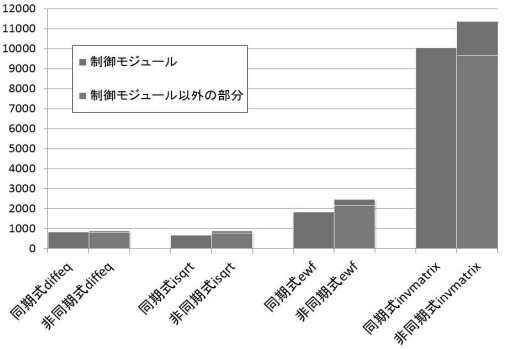



**図 12** 面積 [LE 数]

実験では提案したツールセットと設計フローを微分方程式 (diffeq)，平方根 (isqrt)，elliptic wave フィルタ (ewf)，逆行列積 (invmatrix) に適用し，束データ方式による非同期式回路を合成し，実行時間，消費電力，回路面積，消費エネルギーを評価し，同期式回路と比較を行う．FPGA は，Cyclone IV (EP4CE115F29C7) を用いる．提案するツー

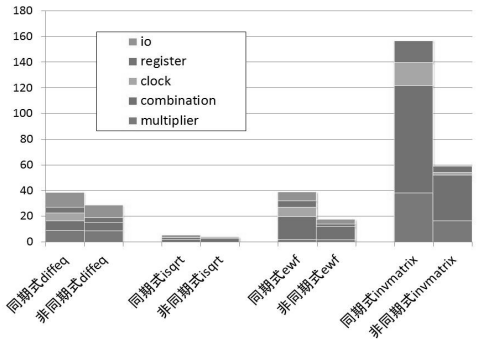



**図 13** 動的消費電力 [mW]

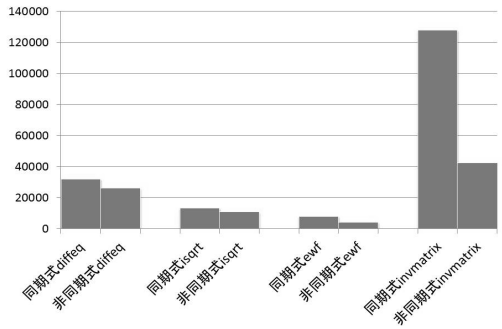



**図 14** 消費エネルギー [pJ]

ルセットは Java と Eclipse で実装した．

始めに比較対象となる同期式回路を合成する．クロック制約 create_clock の値を変え，動作周波数が最高となるものを探索した結果，diffeq，ewf，isqrt，invmatrix のクロックサイクルタイムはそれぞれ 11 ns，12.4 ns，13 ns，18 ns である．

次に，束データ方式による非同期式回路モデル，パス情報ファイル，リソース情報ファイル，レイテンシ制約，制御割合 $CR$，データパス割合 $DR$ を準備し，束データ方式による非同期式回路を合成する．レイテンシ制約は，同期式回路のクロックサイクルタイムとサイクル数をベースに求め，同期式回路と同等の性能を持った束データ方式による非同期式を得ることを目標とし，消費電力や消費エネルギーの差を確認する．diffeq，ewf，isqrt，invmatrix のレイテンシ制約はそれぞれ 55 ns，240 ns，104 ns，882 ns となった．各ベンチマークとも $CR$，$DR$ の値はタイミングの収束性を考慮しそれぞれ，0.95，0.9 とした．

図 11 はシミュレーションによる実行時間を表す．任意のテストベンチを基に，Modelsim-Altera にてシミュレーションした時の実行時間である．diffeq，isqrt では約 9%，ewf では約 8%，非同期式回路の性能低下が見られた．これらは条件を満たすまで繰り返し処理を行うため，休止相が隠蔽できないのが主たる理由である．このオーバヘッドを削減するために，休止相に対するパスにも制約を与えることが考えられる．一方，invmatrix では約 13%の性能向上が見られた．invmatrix は状態数が多く，乗算が多いため，制御モジュール毎に遅延素子を調整することができる非同期式回路の方が有利になった．

図 12 は，合成された回路の面積を表す．面積の値は Quartus II Ver.12.1 Subscription Edition のレポートから得たものである．diffeq では約 7%，isqrt では約 33%，ewf では約 34%，invmatrix では約 13%の面積増加が見られた．これは制御回路やデータパス回路における遅延調整の LCELL 増加が起因する．非同期式制御回路の LE 数は全体の 9%から 15%である．特に ewf や isqrt はホールド制約を満足するために $HD_j$ の面積が多くなった．（ewf は約 13%，isqrt は約 12%）

図 13 シミュレーション時に生成したトグル情報を基に PowerPlay Power Analyzer にて評価した動的消費電力を表す．diffeq では約 26%，isqrt では約 22%，ewf では約 55%，invmatrix では約 63%の電力削減が見られた．小規模な回路より大規模な回路で非同期式制御による効果が出ている傾向が見られ，特に非同期式 ewf の combination や register ではそれぞれ約 41%，約 47%，非同期式 invmatrix の multiplier や combination ではいずれも約 58%の電力が削減されていた．

図 14 は消費エネルギーを表す．diffeq, isqrt では約 19%，ewf では約 46%，invmatrix では約 66%のエネルギー削減効果が見られた．invmatrix では実行時間や消費電力での優位性がそのまま消費エネルギーの削減につながったと見られる．

## 6. 結論

本稿では， FPGA を対象とした束データ方式による非同期式回路の設計支援ツールセットを提案した．提案するツールセットは, 束データ方式による非同期式回路のための制約生成の自動化，タイミング検証の自動化，タイミング違反時の遅延調整の自動化を行う．実験を通じて，提案するツールセットを用いることにより，FPGA を対象とした束データ方式による非同期式回路の設計が容易に行えることが確認でき，また同期式回路と比べ消費エネルギーの少ない回路を得ることができた．

今後はフロアプランニングの検討，パイプライン回路を扱えるようなツールセットの拡張を行う．

**謝辞** 本研究は，アルテラ社ユニバーシティプログラムの協力で行われたものである．また，本研究は科研費若手研究 (B) 24700051 の助成による．最後に研究を遂行するに当たって助言を下さった飯塚成氏に感謝申し上げたい．

### 参考文献

[1] M. Iizuka, "A Tool Set for the Design of Asynchronous Circuits with Bundled-data Implementation", *International Conference on Computer Design*, Oct. 2011, pp.78-83.

[2] M. Tranchero and L. M. Reyneri "Exploiting synchronous placement for asynchronous circuits onto commercial FPGAs", *Field Programmable Logic and Applications, pp.622- 625,* Aug. 31 2009.

[3] The Mathworks. Simulink on-line documentation. [Online]: http://www.mathworks.com/products/simulink/

[4] L. Reyneri, F. Cucinotta, A. Serra, and L. Lavagno, "A hardware/software co-design ow and ip library based on Simulink", *DAC*, June 2001.

[5] Q. T. Ho, J-B. Rigaud, L. Fesquet, M. Renaudim and R. Rolland, "Implementing Asynchronous Circuits on LUT Based FPGAs", *Proc. FDL*, pp36-46, 2002.

[6] D. E. Muller and W. S. Bartky, "A theory of asynchronous circuits", *Proc. International Symposium on the Theory of Switching*, pp204-243, Apr. 1959.

[7] N. Sretasereekul and T. Nanya, "Eliminating Isochronic-Fork Constraints in Quasi-Delay-Insensitive Circuits", *ASP-DAC*, pp437-442, Jan, 2001.

[8] F. U. Rosenberger, C. E. Molnar, T. J. Chaney and T. P. Fang, "Q-Modules: Internally Clocked Delay Insensitive Modules", *IEEE Transaction of Computer*, vol. C-37, no. 9, pp. 1005-1018, 1988.