Vol.2015-SLDM-171 No.5
2015/5/14

束データ方式による非同期式回路のFPGA設計支援環境の構築

滝澤　恵多郎[1,a)]　齋藤　寛[1,b)]

概要：本稿では，商用の FPGA に束データ方式による非同期式回路を実装するための設計支援環境を提案する．提案する設計支援環境は, 設計フローと設計支援ツールセットからなる．前者は商用の FPGA 設計支援ツールのフローをベースとし， 束データ方式による非同期式回路として足りないところを拡張したものである．後者は，束データ方式による非同期式回路のための制約生成，タイミング検証，タイミング違反時の遅延調整を自動化する．実験では，提案する設計支援環境を用いて 4 つの回路を合成し，同期式回路と比較して, 非パイプライン回路では平均 27%, パイプライン回路では平均 26% の消費エネルギーを削減することができた．

1. はじめに

近年，設計コストの低さやライフタイムの長さから，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．FPGA は設計者が何度でも回路構造を書き換えることができる再構成可能なデバイスである. FPGA 上にある Look-up table (LUT) に論理を実現し，それらを配線することによって設計者は所望の機能を実現することができる．

今現在市販されている商用 FPGA のほとんどは，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，半導体微細化技術が進むにつれて，クロックスキュー，およびクロックネットワークにおける消費電力の増大が課題となる．

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, 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．クロック信号がない分，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，商用の FPGA 自体が同期式回路を対象としているので，それらの設計支援ツールは，非同期式回路を対象としていない．こうした背景から，これまでに商用の FPGA に非同期式回路を実現するための様々な研究が行われてきた [2], [3], [4], [5]． しかしながらこれらの研究では，設計自動化に関する言及がほとんどない．

筆者らは [1] にて, 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．このツールセットは，非同期式回路のための設計制約生成, タイミング検証, 遅延調整などのプロセスを自動で行う．Altera 社の Quartus II と共に用いることで，束データ方式による非同期式回路を Altera 社の FPGA 上に容易に実現できることを示した．しかしながら [1] では，パイプライン回路が扱えなかった．また，レイテンシ制約を満足するよう，データパスにローカルクロック制約を与えたが，電力削減効果が低かった．

本稿では，[1] で提案した設計支援ツールセットを拡張し, パイプライン回路を扱えるようにする．また，データパスにはローカルクロック制約ではなく，最大遅延制約を与えることで，電力削減の改善を試みる．

2. 束データ方式による非同期式回路

2.1 回路モデル

束データ方式とは非同期式回路実装手法の 1 つで，N ビットのデータを表す為に，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．演算の完了は，要求信号線上にデータパス回路の遅延より大きい遅延素子を付加することによって保証する．

まず, 束データ方式による非同期式回路の構造について述べる. 図 1(a) は非パイプラインモデル, (b) はパイプラインモデルを表す．共に右側はデータパス回路，左側は制御回路である．データパス回路はレジスタ (reg_k) ($1 \leq k \leq e$), 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．hd_k はレジスタ reg_k の直前に置かれた遅延素子で，ホールド制約を保証するために用いる．hd_k は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を生成する．

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 [6]，C 素子 c_i [7], 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．非パイプラインモデルの場合，1 つの $ctrl_i$ で回路の 1 状態を制御する．パイプラインモデルの場

[1] 会津大学
University of Aizu, Aizu-Wakamatsu, Fukushima 965–8580, Japan
[a)] m5161141@u-aizu.ac.jp
[b)] hiroshis@u-aizu.ac.jp

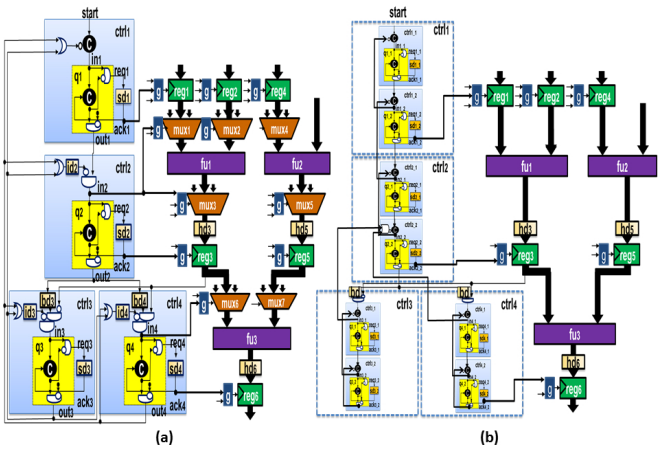

図 1　束データ方式による非同期式回路モデル:(a) 非パイプラインモデル，(b) パイプラインモデル

合，データインターバルが 1 の時と 2 以上の時で異なる．1 の場合，後述する休止相を隠蔽するために 2 つの制御モジュールで 1 パイプラインステージを制御する．2 以上の場合，非パイプラインモデル同様 1 つの制御モジュールで 1 パイプラインステージを制御する．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 $ctrl_i$ の休止相に伴う制約を保証するために用いる．なお，id_i はパイプラインモデルには含めない．C 素子は非同期式回路で多用される待ち合わせ回路である．全ての入力が 0 の時，C 素子の出力は 0 になり，全ての入力が 1 の時，C 素子の出力は 1 になる．それ以外の時，C 素子の出力は変化しない．

次に，回路動作について述べる．以降の記述では信号の立ち上がり遷移を $signal+$，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．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．分岐の場合，データパス回路からの条件信号によって $ctrl_i$ をアクティベートする in_i+が生成されるかそうでないかが決まる．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をグルーロジック g を介して生成する．in_i+は，$req_i+ \rightarrow sd_i+ \rightarrow ack_i+$ という遷移順で Q モジュール q_i に戻る．その後，$req_i- \rightarrow sd_i- \rightarrow ack_i-$ という遷移順で Q モジュール q_i に戻る．また，ack_i-でレジスタにデータが書き込まれる．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．非パイプラインモデルの場合，最後の制御モジュールが out_i+を生成した後，各制御モジュール $ctrl_i$ は，in_i-と out_i-をほぼ同時に生成する．一方パイプラインモデルの場合，各制御モジュールからのフィードバック信号が前の制御モジュールに戻り，次の処理を待機する．なお，各制御モジュールにおいて，in_i+から out_i+はデータパス回路にて処理が行われるが，in_i-から out_i-はデータパス回路にて何も処理が行われない．そのため本稿では前者を稼働相，後者を休止相と呼ぶ．

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．

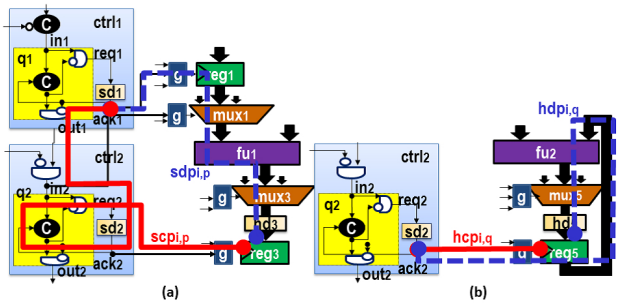

図 2　(a) セットアップ制約に関するパス，(b) ホールド制約に関するパス

最初のセットアップ制約，ホールド制約，分岐制約は非パイプライン回路，パイプライン回路の両方で満足する必要がある．一方，最後の休止相制約は非パイプライン回路だけのものである．

2.2.1　セットアップ制約

ソースレジスタを制御する sd_{i-1} の出力から制御モジュール $ctrl_i$ を通り，ディスティネーションレジスタのクロックピンに至るパス $scp_{i,p}$ (図 2(a) の実線) の最小遅延は，ソースレジスタを制御する sd_{i-1} の出力からデータパスを通り，ディスティネーションレジスタの入力に至るパス $sdp_{i,p}$ (図 2(a) の破線) の最大遅延より大きくなければならない．$scp_{i,p}$ の最小遅延を $t_{minscp_{i,p}}$，$sdp_{i,p}$ の最大遅延を $t_{maxsdp_{i,p}}$ セットアップ時間を $t_{setup_{i,p}}$，データパスに対するマージンを $sm_{i,p}$ $(sm_{i,p} > 0)$ 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．

$$t_{minscp_{i,p}} > t_{maxsdp_{i,p}} * sm_{i,p} + t_{setup_{i,p}} \quad (1)$$

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

ack_i- から reg_k の入力データピンまでのデータパス $hdp_{i,k}$ (図 2(b) の実線) の最小遅延は，ack_i- から reg_k のクロックピンまでの制御パス $hcp_{i,k}$ (図 2(b) の破線) の最大遅延より大きくなければならない．$hdp_{i,k}$ の最小遅延を $t_{minhdp_{i,k}}$，$hcp_{i,k}$ の最大遅延を $t_{maxhcp_{i,k}}$，ホールド時間を $t_{hold_{i,k}}$，$t_{maxhcp_{i,k}}$ に対するマージンを $hm_{i,k}$ $(hm_{i,k} > 0)$ とすると，ホールド制約は以下の不等式で表すことができる．

$$t_{minhdp_{i,k}} > t_{maxhcp_{i,k}} * hm_{i,k} + t_{hold_{i,k}} \quad (2)$$

この制約に違反する場合，遅延素子 hd_k を調整する．

2.2.3　分岐制約

分岐判定信号を格納したレジスタ reg_k の値によって，いずれかの制御モジュールに制御が分岐すると仮定する．reg_k を制御する ack_i- から分岐判定 AND ゲートの入力までのパス bcp_i（図 3(a) の破線）の最小遅延は ack_i- より，reg_k のクロックピンを経由し，reg_k 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_i (図 3(a) の実線) の最大遅延より大きくなければならない．bcp_i の最小遅延を t_{minbcp_i}，bdp_i の最大遅延を t_{maxbdp_i}，t_{maxbdp_i} に対するマージンを bm_i $(bm_i > 0)$ とすると，分岐制約は以下の不等式で表すことができる．

$$t_{minbcp_i} > t_{maxbdp_i} * bm_i \quad (3)$$

この制約に違反する場合，遅延素子 bd_i を調整する．

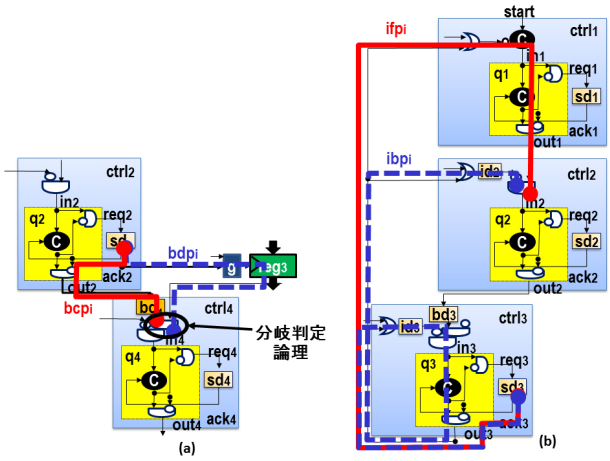

図 3　(a) 分岐制約に関するパス，(b) 休止相制約に関するパス

2.2.4　休止相制約

最後の制御モジュール out_n+からのフィードバック信号で，$ctrl_i$ を通り，out_n-として Q モジュール q_i の直前の AND ゲートの入力ピンまでのパス ibp_i (図 3(b) の破線) の最小遅延は，out_n+より $ctrl_{i-1}$ を通り，$ctrl_i$ の Q モジュール q_i の直前の AND ゲートの出力ピンまでのパス ifp_i (図 3(b) の実線) の最大遅延より大きくなければならない．ifp_i の最大遅延を t_{maxifp_i}，ibp_i の最小遅延を t_{minibp_i}, t_{maxifp_i} に対するマージンを im_i $(im_i > 0)$ とすると，休止相制約は以下の不等式で表すことができる．

$$t_{minibp_i} > t_{maxifp_i} * im_i \qquad (4)$$

この制約に違反する場合， 遅延素子 id_i を調整する．

3.　FPGA

本稿では Altera 社の FPGA 上に非同期式回路を実現する．図 4 は，Altera 社の FPGA Cyclone IV の構造を示す．Altera 社の FPGA は Logic Array，Embedded Multiplier，Random Access Memory (RAM)，Input/Output Elements (IOE)，Phase Locked Loop (PLL) より構成される．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 LE は 1 つの LUT，1 つのフリップフロップ (FF)，マルチプレクサ (MUX) から構成される．LCELL は 4 入力 1 出力のプログラム可能なプリミティブで，ここに論理を書き込む．

Altera 社の FPGA における配線はグローバル配線とローカル配線に分類することができる．グローバル配線は Row/Column Interconnect からなり，LAB，IOE，RAM，PLL，Embedded Multiplier に入出力される信号を接続する．Local Interconnect は Row/Column Interconnect と LAB を接続する．

4.　提案する設計支援環境

提案する設計支援環境は，設計フローと 6 つの設計支援ツールからなる．非パイプラインモデルの場合レイテンシ制約を，パイプラインモデルの場合サイクルタイム制約とデータインターバルを考慮して回路合成を行う．Altera 社の設計ツール Quartus II と 6 つの設計支援ツールによって，束データ方式による非同期式回路の論理合成以降のほとんどを自動化するので，束データ方式による非同期式

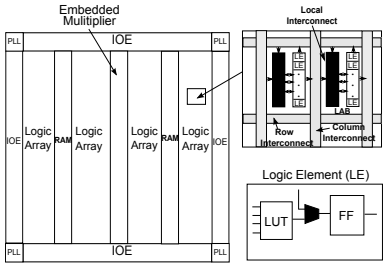

図 4　Cyclone IV FPGA の構造

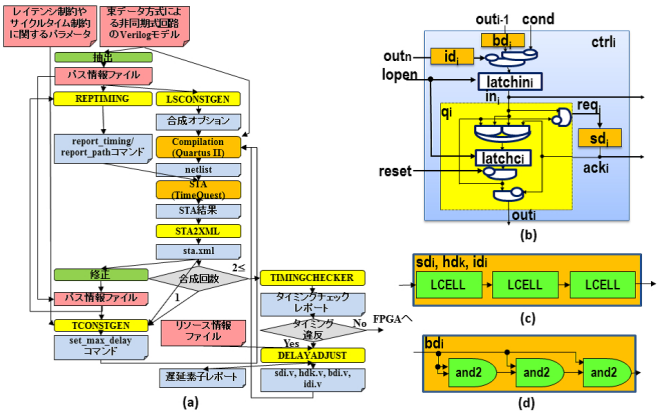

図 5　(a) 設計フロー，(b) 制御モジュール $ctrl_i$ の構造，(c) 遅延素子 sd_i，hd_k，id_i の構造，(d) 遅延素子 bd_i の構造

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，提案する設計支援環境では，一般的なハードウェア記述言語や設計制約を扱うため，他社の FPGA でも同じようなことができると考えている．

4.1　設計フロー

図 5(a) は，本稿で用いる設計フローを表す．この設計フローは，[1] で提案されたものをベースとしている．違いは，パイプライン回路も同じフローで扱えるようにしたことと，レイテンシ制約やサイクルタイム制約を満足するために，データパスに対してローカルクロック制約ではなく，最大遅延制約を用いることである．最大遅延制約を用いる理由は，性能的により良い回路を得ることが実験で分かったためである．

入力は Verilog HDL による束データ方式の非同期式回路，パス情報ファイル，リソース情報ファイル，レイテンシ制約やサイクルタイム制約に関するパラメータの 4 つである．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 回目の合成の前に，LSCONSTGEN を用いて回路合成に関わるオプションを生成する．また，REPTIMING を用いて，パス情報ファイルからパス遅延解析コマンドを生成する．準備の後，Quartus II を用いて 1 回目の合成を行い，静的タイミング解析 (STA) を行う．次に，STA2XML と TIMINGCHECKER を用いてタイミング検証を行い，TCONSTGEN を用いてレイテンシ制約，サイクルタイム制約を可能な限り満足するために各パスに最大遅延制約を生成する．生成後，最大遅延制約を用いて 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 DELAYADJUST を用いて遅延素子を調整し，全てのタイミング制約を満足するまで合成を繰り返す．

start	end	through	src	muxctr	dst	t_{io}	$sm_{i,p}$
in0	reg0				ctrl0	0	1.05
reg0	reg0	fu0			ctrl1	0	1.05
sd1	reg2	fu1	ctrl1	ctrl2	ctrl2	0	1.05

(a)

gate-name	delay	pin-name	
		in	out
LCELL	0.45	in	out
AND2	0.45	in1	out
		in2	

(c)

msd	mhd	mbd	mid
1.0	1.0	1.0	1.0

(b)

L	CT	CR	DR	DI
55	20	0.95	0.95	1

(d)

図 6　(a) セットアップ制約の検証に必要なパス情報，(c) 遅延素子で用いるプリミティブセルの情報，(d) レイテンシ制約やサイクルタイム制約に関するパラメータ

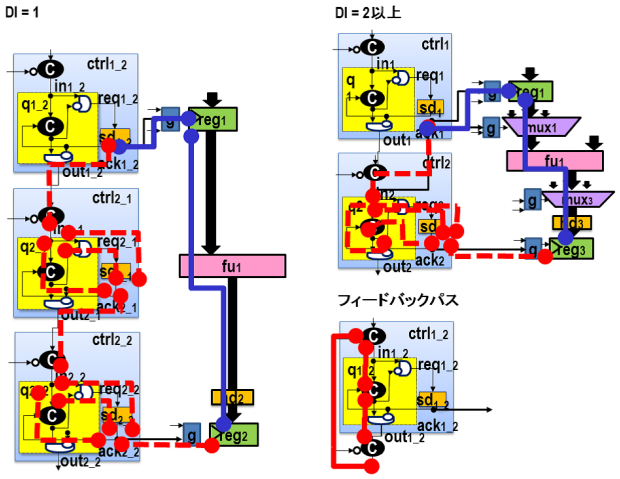

図 7　パイプライン回路におけるセットアップパスとフィードバックパスの分割

4.2　入力

ここでは，提案する設計支援環境に対する 4 つの入力を解説する．

Verilog HDL モデルは，図 1 で示した回路モデルに対して部品毎にモデリングを行い，トップレベルで各部品を呼び出し接続するようモデリングする．制御モジュール $ctrl_i$ は，図 5(b) のように，in_i 信号の直前と Q モジュールの中にラッチを挿入しておく．これは，制御回路内部の組み合わせループを除去し，パス遅延解析を正しく行うためである．遅延素子 sd_i は，最初一つの LCELL で実現しておく．

パス情報ファイルには，セットアップ制約，ホールド制約，分岐制約，休止相に関する制約のパス情報を記述する．必要な情報は，セットアップ制約の場合，図 6(a) で示す通り，パスの始点，終点，中間点，始点を制御する制御モジュール，マルチプレクサを制御する制御モジュール，終点を制御する制御モジュール，最大遅延制約値，マージン $sm_{i,p}$ を記述する．最大遅延制約値は固定値を与えることも，1 回目の STA 結果から自動で決定することもできる．他のタイミング制約に関しても同様である．なお，Quartus II の合成によって，最初に指定したパスの始点，終点，中間点がリネームされることもあるので，STA にて正しいパス遅延が解析できない場合は，これらの値を更新する必要がある．

パス情報ファイルにはタイミング制約に関するパス情報のほかに，遅延素子に対するマージンを記述する．msd は sd_i に対するマージン，mhd は hd_k に対するマージン，mbd は bd_i に対するマージン，mid は id_i に対するマージンである．タイミング検証後の遅延調整において，遅延素子はタイミング制約の右辺の値にこれらのマージンを加えた分持たせる．仮にもしこの値を超えたときは，遅延素子から超えた分に相当するプリミティブセルを削除する．なお，これらのマージンの値が大きければ，タイミング制約は満足しやすくなるが性能を悪くする．

リソース情報ファイルは，遅延素子で用いるプリミティブセルの情報を記述する．プリミティブセルの情報は，図 6(c) のように遅延素子用セルの名前，セル 1 個を配置した時の遅延値，セルの入出力ピン名を指定する．なお，セル 1 個の遅延値は，セル 1 個分の素子遅延とセル間の配線遅延を足したものとする．配線遅延を足す理由は，セルを 1 個増やすことにより配線数も 1 本増えるためである．これらの値は，任意の数のセルを配置し，静的タイミング解析を行った時の平均値を用いる．

最後に，レイテンシ制約やサイクルタイム制約に関するパラメータ（図 6(d)）を準備する．レイテンシは，最初の制御モジュールに対する入力信号遷移 $start+$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，その値をレイテンシ制約 L として与える．レイテンシ制約やサイクルタイム制約のうち，データパス回路を制御する時間の割合を制御割合 CR $(0 < CR \leq 1)$ とし，制御回路における時間のうちデータパス回路における処理時間の割合をデータパス割合 DR $(0 < DR \leq 1)$ として与える．パイプライン回路は，サイクルタイム制約 CT とデータインターバル DI を与えて合成する．サイクルタイムは 1 サイクルあたりの時間，データインターバルは次のデータが入力されるまでのサイクル数を表す．

4.3　提案するツールセット

提案するツールセットは，LSCONSTGEN，REPTIMING，STA2XML，TIMINGCHECKER，TCONSTGEN，DELAYADJUST からなり，束データ方式による非同期式回路の FPGA 実装を支援する．これらは [1] で述べているので，ここでは各ツールの概要，パイプライン回路のための修正と最大遅延制約の生成を中心に解説する．

LSCONSTGEN は，パス情報ファイルを入力とし，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．このスクリプトには，グローバルな信号の使用を避けるためのコマンドなどが含まれる．

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 [1]．セットアップ時間やホールド時間を解析する必要があるパスは *report_timing* コマンドを，それ以外のパスは *report_path* コマンドを使用する．コマンド生成の前に STA を正しく行うために，各パスをレジスタ，ラッチ，および遅延素子 sd_i にて分割する．パイプライン回路の場合，データインターバルの値に応じて分割が異なる．パイプライン回路のセットアップパスを例にこの違いを述べる．データインターバル 1 の時は，2 つの制御モジュールで 1 つのパイプラインステージを制御する．そのため，図 7 の左側のように制御回路のパス $scp_{i,p}$ は，$sd_i \rightarrow ctrl_i|latch_{i+1}$，$ctrl_i|latch_{i+1} \rightarrow sd_{i+1}$，$sd_{i+1} \rightarrow q_{i+1}|latch_{i+1}$，$q_{i+1}|latch_{i+1} \rightarrow sd_{i+1}$，$sd_{i+1} \rightarrow ctrl_i|latch_{i+2}$, $ctrl_{i+2}|latch_{i+2} \rightarrow sd_{i+2}$，$sd_{i+2} \rightarrow q_{i+2}|latch_{i+2}$，$q_{i+2}|latch_{i+2} \rightarrow sd_{i+2}$ の 9 つのサブパスに分割し，STA コマンドを生成する．データインターバル 2

以上の時は, 1 つの制御モジュールで 1 つのパイプラインステージを制御する．そのため，非パイプライン回路の時と同様パス $scp_{i,p}$ を，$sd_i \rightarrow ctrl_i|latch_{i+1}$，$ctrl_i|latch_{i+1} \rightarrow sd_{i+1}$，$sd_{i+1} \rightarrow q_{i+1}|latch_{i+1}$，$q_{i+1}|latch_{i+1} \rightarrow sd_{i+1}$，$sd_{i+1} \rightarrow reg_k|clk$ の 5 つのサブパスに分割し（図 7 右上），STA コマンドを生成する．データパス $sdp_{i,p}$ に関しては非パイプライン回路と同様に, sd_i →始点レジスタのクロックピン，レジスタ間，$ctrl_i|latch$ →終点レジスタのそれぞれの最大遅延を解析する STA コマンドを生成する．また，パイプライン回路の場合，フィードバックパスのための *report_path* コマンドを生成する．各制御モジュール $ctrl_i$ 毎に，図 7 の右下のように C 素子でパスを分割し，*report_path* コマンドを生成する．

STA2XML は，TIMINGCHECKER にてタイミング検証を行うため，TimeQuest Timing Analyzer による STA 結果のうち，タイミング検証で利用するパス遅延を抽出し，XML 形式に変換する．

TIMINGCHECKER は，STA2XML によって生成された XML ファイル，パ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を表す．各タイミング制約に関する検証結果をタイミングチェックレポートに出力する．

TCONSTGEN は, パス情報ファイル, STA2XML によって生成された XML ファイル，レイテンシやサイクルタイム制約に関するパラメータを入力とし，与えられたレイテンシ制約やサイクルタイム制約を可能な限り満足するよう，セットアップ制約に関するパス $sdp_{i,p}$ と $scp_{i,p}$ を構成する全てのサブパスに 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．パス情報ファイルで t_{io} に記載がない場合，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．パイプライン回路の場合，あるサブパスの最大遅延制約は，$CT * CR * DR * (w/tp)$ で得られた値となる．ここで，CT はサイクルタイム，CR はサイクルタイムに対する制御時間の割合，DR はサイクルタイムに対するデータパス処理時間の割合，w は STA で求めたサブパスの遅延，tp はサブパスを含んだパス全体の遅延を表す．

DELAYADJUST はリソース情報ファイル，STA2XML にて出力された XML ファイル，遅延素子のセル数を表した遅延素子レポート，TIMINGCHECKER にて出力されたタイミングチェックレポートを入力に，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_k，bd_i，id_i に対して遅延調整を行い，Verilog HDL を生成する．調整はホールド制約，分岐制約，休止相制約，セットアップ制約の順番で行う．これは，ホールド制約と分岐制約による遅延素子 hd_k と bd_i が，セットアップ制約に影響を与えるためである．非パイプライン回路とパイプライン回路でデータインターバルが 2 以上の時は，タイミング違反があった場合，足りなかった分に相当するセルを対応する遅延素子に追加する．逆に余剰の場合，対応する遅延素子の遅延がデータパス遅延の最大値に遅延素子のマージン (msd など) を加えた分になるようセルを削除する．データインターバルが 1 でかつセットアップ制約に関しては，レジスタを制御するために制御信号が 2 つの制御モジュールを通過するので，足りなかった分の半分の遅延に相当するセルを sd_{i_1} と sd_{i_2} に追加する、あるいは余剰な分の半分の遅延に相当するセルを sd_{i_1} と sd_{i_2} から削除する．なお，sd_i，hd_k，id_i は図 5(c) のように LCELL で，bd_i は図 5(d) のように AND2 で構成する．

表 1 非同期式回路のデータ

回路	CT [ns]	L [ns]	$ctrl_i$	調整回数
diffeqa	12	60	5	17
ewfa	14	210	15	18
imatrixa	19	684	36	29
idcta	20	300	15	28
diffeqa1	12	-	10	13
diffeqa2	12	-	5	15
ewfa1	12	-	30	17
ewfa2	12	-	15	16
imatrixa1	20	-	72	28
imatrixa2	20	-	36	28

5. 実験

実験では，提案する設計支援環境を用いて微分方程式 (diffeq)，elliptic wave フィルタ (ewf)，逆行列積 (imatrix)，逆離散コサイン変換 (idct) を合成する．合成後， 回路面積，実行時間，消費電力，消費エネルギーを評価し，同期式回路と比較を行う．非パイプライン回路の実験では, 全ての回路を合成する. 同期式回路は回路名の末尾を s, 非同期式回路は回路名の末尾を a とする．パイプライン回路の実験では, diffeq, ewf, imatrix の 3 つを合成する．データインターバル DI=1，2 の同期式回路の回路名の末尾を s1，s2，非同期式回路の回路名の末尾を a1，a2 とする．回路合成には Altera Quartus II Ver13.1 sp1 を用い, ターゲットデバイスは Cyclone IV (EP4CE115F29C7) とする．STA には TimeQuest Timing Analyzer を, 論理シミュレーションには Mentor ModelSim-Altera 10.1b を，電力解析には PowerPlay Power Analyzer を用いる．提案する設計支援ツールセットは 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* の値を変更し, 動作周波数が最高となるものを探索する．これは，性能を同期式回路と同等にすることで，非同期式回路にした時の電力削減効果を知るためである．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 $CR(= 0.95)$，データパス割合 $DR(= 0.95)$ を準備し，束データ方式による非同期式回路を合成する．非パイプライン回路のためのレイテンシ制約は，同期式回路のクロックサイクルタイムとサイクル数の積とし，パイプライン回路のためのサイクルタイム制約は, 同期式回路のクロックサイクルタイムとする．TCONSTGEN を用いて各パスの *set_max_delay* を生成し，Quartus II にて回路を合成する．なお調整は，遅延素子遅延に対するマージン msd，mhd，mbd，mid を 1 ns から始め，10 回の調整でタイミング制約が収束しないものはマージンを増やしていく．diffeq と ewf は回路規模が小さいので 0.5 ns ずつ，imatrix と idct は回路規模が大きいので 1.0 ns ずつ増やしていく．表 1 は，非同期式回路のサイクルタイム (CT) とレイテンシ制約値 (L)，制御モジュールの数、および調整回数を表す．

図 8 は非パイプラインの合成結果を，図 9 はパイプライン回路の合成結果を表す．回路面積は LE 数，実行時間は

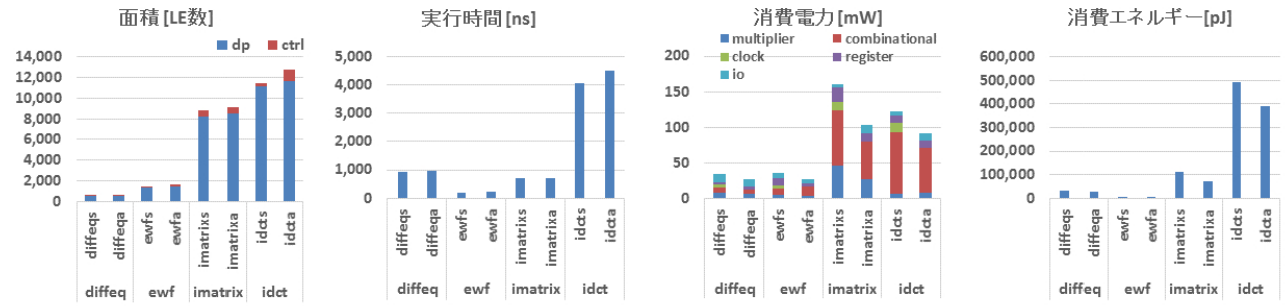

図 8　非パイプライン回路の結果

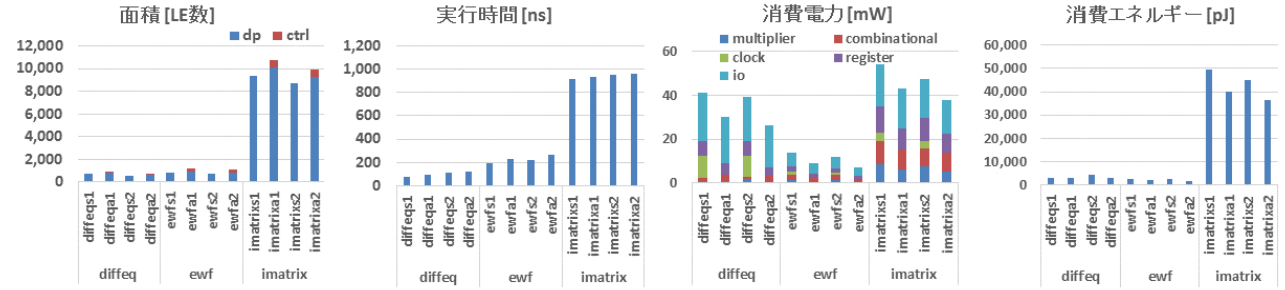

図 9　パイプライン回路の結果

任意のテストパターンによるシミュレーション時間，消費電力はシミュレーション結果を基にした電力消費，消費エネルギーは実行時間と消費電力の積である．

面積に関しては, 同期式回路と比較して非パイプライン回路は平均 11%, パイプライン回路は平均 12%の面積増加が見られた．面積増加の原因は遅延素子を含んだ非同期式制御回路の LE 数に起因し，制御モジュールの数に比例する．実行時間に関しては, 同期式回路と比較して非パイプライン回路は平均 1%, パイプライン回路は平均 8%のオーバヘッドが見られた．*set_max_delay* 制約の制約値よりパス遅延が大きなパスがあったのが原因である．消費電力に関しては, 同期式回路と比較して非パイプライン回路は平均 28%, パイプライン回路は平均 27%の電力削減が見られた．clock と Register Cell 部分の消費電力の減少が主な理由である．消費エネルギーに関しては, 同期式回路と比較して非パイプライン回路は平均 27%, パイプライン回路は平均 26%の削減が見られた．実行時間のオーバヘッドがそれほど大きくなく, 消費電力の削減効果が大きかったことが消費エネルギーの削減に繋がった．

[1] では，データパスに *create_clock* コマンドを用いてローカルクロック制約を割り当てて非パイプライン回路の diffeq と ewf を合成したが，その時の消費エネルギーは 26,108 pJ と 7,043 pJ であった．今回 *set_max_delay* コマンドを用いた結果，25,700 pJ と 5,891 pJ と共に消費エネルギーを削減することができた．

6. 結論

本稿では，商用の FPGA に対する束データ方式による非同期式回路の設計支援環境を提案した．提案する設計支援環境は, 設計フローと設計支援ツールセットからなる．提案する設計支援環境を用いることで設計者は，商用の FPGA 上に束データ方式による非同期式回路を容易に実現することができる．実験では，提案する設計支援環境を用いて 4 つの回路を合成した．同期式回路と比較して, 非パイプライン回路では平均 27%, パイプライン回路では平均 26% の消費エネルギーを削減することができた．

今後は消費エネルギーを削減するために，フロアプランニングの導入を検討する．FPGA においては配線遅延が大きいため，配線遅延の抑制が不可欠である．配線遅延が抑制できれば、配線リソースの数やその電力消費も抑えることが期待できる．

謝辞　本研究は，文科省科研費若手研究 (B)24700051 の助成によって行われたものである．

参考文献

[1] K. Takizawa, S. Hosaka, and H. Saito, ”A Design Support Tool Set for Asynchronous Circuits with Bundled-data Implementation on FPGAs”, Proc. FPL, pp.232–235, 2014.

[2] M. Tranchero, and L. M. Reyneri, ”Exploiting synchronous placement for asynchronous circuits onto commercial FPGAs”, Proc. FPL, pp.622–625, 2009.

[3] M. Tranchero, and L. M. Reyneri, “Implementation of Self-Timed Circuits onto FPGAs Using Commercial Tools”, Proc. DSD, pp.373–380, 2008.

[4] Q. T. Ho, J-B. Rigaud, L. Fesquet, M. Renaudim, and R. Rolland, ”Implementing Asynchronous Circuits on LUT Based FPGAs”, Proc. FDL, pp.36–46, 2002.

[5] R. U. R Mocho, G. H. Sarton, R. P. Ribas, and A. I. Reis, ”Asynchronous Circuit Design on Reconfigurable Devices”, Proc. SBCCI, pp.20–25, 2006.

[6] 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.

[7] D. E. Muller and W. S. Bartky, ”A theory of asynchronous circuits”, Proc. International Symposium on the Theory of Switching, pp.204–243, Apr 1959.