令和 4 年度

修士学位論文

FPGA の論理スライス構造に適したセルフタイム型データ転送制御回路とそのタイミング検証法

Self-Timed Data-Transfer Control Circuit Implementation Suitable for FPGA Logic-Slice Structure and Its Timing Verification

1255102　　尾ノ井 嶺卓

指導教員　　岩田 誠

2023 年 2 月 28 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報システム工学コース

要 旨

FPGA の論理スライス構造に適した セルフタイム型データ転送制御回路と そのタイミング検証法

尾ノ井 嶺卓

IoT（Internet of Things）の普及に伴い、高性能な IoT デバイスを安価かつ柔軟に実現することが求められている。自己タイミングパイプライン (STP) で構成される DDP(Data-Driven Processor) は、データの多重並列処理による高性能と、データ到着箇所のみ動作する省電力性から、IoT デバイス向けのプロセッサとして有望視されている。FPGA(field-programmable gate array) は柔軟に回路を再構成でき，様々な IoT デバイスの性能要求を満たすことができるため、STP を用いた DDP の FPGA 実装が近年注目されている。先行研究では，非同期回路としての STP のデータ転送制御回路に検証回路を付加し，タイミング情報を抽出する検証法が提案されている．しかし，FPGA 回路の配置・配線を設計ツールに任せていたため，データ転送制御回路の最適化や効果的なタイミング検証が困難であった．

本研究では Xilinx 社製 FPGA の論理スライス構造に着目し，それに適したセルフタイム型データ転送制御回路の実装法とそのタイミング検証法を検討した．

結果，提案回路と提案配置によって，C 素子内の全ラッチ遅延時間が 20%〜50%短縮でき，8 ステージ STP の T_f/T_r のばらつきも低減でき，提案手法を適用したタイミング検証法が必要制約を充足することが確認できた．今後は，検証過程の分析と検証コストの定量的な評価が必要である．

キーワード　Internet of Things(IoT), セルフタイム型パイプライン (STP)，FPGA，タイミング検証

Abstract

Self-Timed Data-Transfer Control Circuit Implementation Suitable for FPGA Logic-Slice Structure and Its Timing Verification

Minetaka ONOI

With the proliferation of the Internet of Things (IoT), it is a necessary to realize high-performance IoT devices inexpensively and flexibly. DDP (Data-Driven Processor) realized by self-timed pipelines (STP) is a promising processor for IoT devices because of its high performance and low power consumption, which operates only at the point of data arrival. FPGAs (field-programmable gate arrays) can flexibly reconfigure circuits to meet the performance requirements of various IoT devices, and FPGA implementations of DDPs using STP have recently attracted attention. In previous studies, verification methods have been proposed to extract timing information by adding a verification circuit to the data transfer control circuit of STP as an asynchronous circuit. However, since the placement and wiring of FPGA circuits were left to design tools, it was difficult to optimize the data transfer control circuits and conduct effective timing verification.

In this study, we focused on the logic slice structure of Xilinx FPGAs and investigated how to implement a self-time data transfer control circuit suitable for it and how to verify its timing.

The results showed that the proposed circuit and the proposed layout reduced the total latch delay time in the C element by 20 to 50 percent and also reduced the Tf/Tr

variation in the 8-stage STP, thus confirming that the timing verification method using the proposed method satisfies the required constraints. Future work is needed to analyze the verification process and quantitatively evaluate the verification cost.

key words Internet of Things (IoT), Self-Timed Pipeline (STP), FPGA, Timing Verification

目次

目次

図目次

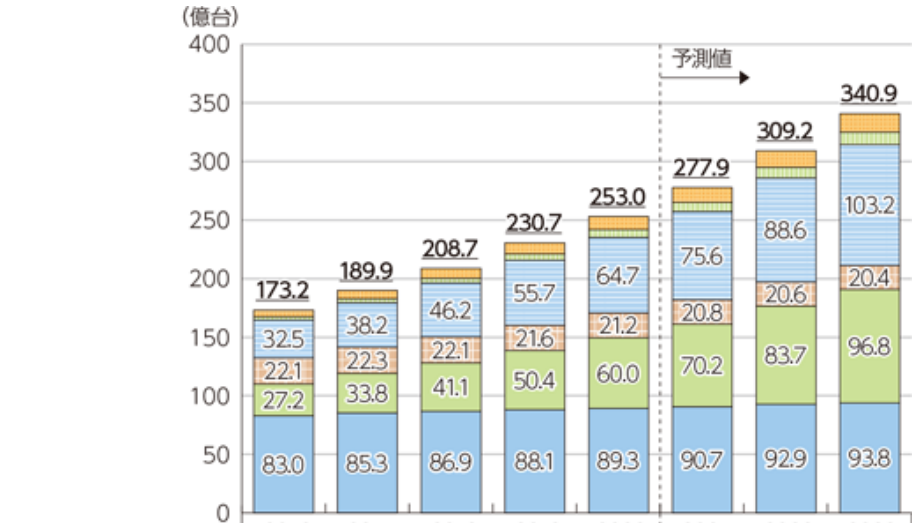
図目次

表目次

第 1 章

序論

近年，IoT デバイスの数は図 1.1 に示す増加の一途をたどっている．総務省の予測では，2023 年には世界中で約 340 億台まで IoT デバイスが増加すると予想されている [1].



	2016	2017	2018	2019	2020	2021	2022	2023
合計	173.2	189.9	208.7	230.7	253.0	277.9	309.2	340.9
自動車・宇宙航空	5.7	7.1	8.5	9.8	11.3	12.7	14.2	15.7
医療	2.6	3.3	4.0	5.1	6.5	7.8	9.2	10.9
産業用途	32.5	38.2	46.2	55.7	64.7	75.6	88.6	103.2
コンピューター	22.1	22.3	22.1	21.6	21.2	20.8	20.6	20.4
コンシューマー	27.2	33.8	41.1	50.4	60.0	70.2	83.7	96.8
通信	83.0	85.3	86.9	88.1	89.3	90.7	92.9	93.8

図 1.1　世界の IoT デバイス数の推移及び予測（文献 [1] より引用）

特に医療・コンシューマ・産業用途・自動車・宇宙航空の分野で成長が著しい．各分野の IoT デバイスの用途はインターネット通信を活用した利用者への付加価値の提供からセンサデータ等の解析による製品品質向上まで多岐にわたる．このような IoT デバイスの使用目

的の多様化に伴い，IoT システム全体の通信帯域の圧迫や処理遅延への対策が要求されており，これらの要求を満たすためにエッジコンピューティングに注目が集まっている [2]．エッジコンピューティングとは，IoT デバイスからデータを収集し，その処理をデータ収集地点に近い IoT デバイスや付近のサーバに任せる技術のことである．従来のクラウドサーバ等でデータを一括処理する場合に比べ，サーバへの負荷が小さく，通信遅延が削減できることから IoT デバイスへのリアルタイムなフィードバックが可能である．一方でクラウドサーバで行う高度な AI 処理などで比較した場合，エッジサーバ遅延が無視できないほど大きい可能性があるため，エッジコンピューティングにおけるエッジ端末で行う処理は，簡便なデータ処理が向いている．また，エッジ端末は山奥やドローンの上など電力供給が乏しい場所に配置を想定する場合，従来よりもさらに低消費電力で動作するエッジデバイスが必要となるため，超消費電力プロセッサについての研究が進められている．

超低消費電力プロセッサによる IoT システムの実現には，アルゴリズム，言語，コンパイラ，アーキテクチャ，実際のハードウェアなどの各レイヤで性能向上や消費電力の低減に取り組むことが重要である．中でもアーキテクチャは，命令セットアーキテクチャ，マイクロアーキテクチャ，論理設計，実装などを含み，CPU の実行命令数，クロック・サイクル時間，命令当たりのクロックサイクル数 (CPI) に影響を及ぼす．現在主流となっているアーキテクチャの実装方式として，ノイマン型処理方式が挙げられる．ノイマン型処理方式はプログラムをメモリ内に数値として格納し，一定タイミングでプログラムを呼び出すことで命令を実行する方式である．同期回路として PC(プログラムカウンタ) によってプログラムが実行される．ノイマン型処理方式を採用している IoT プロセッサの多くはクロックを動作の起点としている．クロック信号とは一定間隔で立ち上がりと立下りを繰り返す信号のことであり，回路全体にグローバルに分配されている．このクロック信号の一定間隔の立ち上がりもしくは立下りに同期して全ての回路が動作する．クロックをタイミング同期に採用している方式をクロック同期方式と呼ぶ．クロック同期方式は，クロックにより各ステージが統一的に動作するため，回路の動作保証が容易である一方で，回路が動作していない期間においてもクロック信号が入力され続けることから，待機電力による余分な電力消費が問題となって

いた．

そこで，プロセッサの各ステージがローカルクロックによって独立して動作するセルフタイム型回路が注目されている．自己同期型回路はグローバルクロックを用いる同期回路と異なり，非同期に動作するが，動作箇所のみ電力を消費するため省電力などの利点がある．この利点を活かすために，セルフタイム型パイプラインを用いたデータ駆動型プロセッサ（DDP）が研究されている．DDP はデータの到着をトリガとして動作するため，セルフタイム型パイプラインと相性が良い．一方でグローバルクロックが存在しないため，タイミング検証のためのタイミング解析パスが膨大かつ複雑であり，設計者の設計コストが同期回路設計と比較して高い [3].

そこで，非同期束データ回路に特化した静的タイミング解析手法が提案されている [4]. 本手法では LCS(Local Clock Sets) を事前に定義しておくことで，自動でタイミング制約を網羅的に検証でき，データ転送を制御する基本的な非同期回路である WCHB（Weak-Condition Half-Buffer）を対象としたタイミング検証の自動化に成功している．しかし，4 相ハンドシェイクによる非同期回路はクリティカルパスにループを含むため，4 相ハンドシェイクのセルフタイム型データ転送制御回路を用いて実現された DDP(データ駆動型プロセッサ) に本手法を適用できない．そこで，DDP に特化したタイミング検証法が提案されている [5]．この先行研究では，疑似同期回路化とタイミング検証ツールにより，タイミング検証の自動化に成功している．しかし，疑似同期回路化による FPGA の性能を最大限活用した回路構成や配置が必要である．

また，ディジタル回路を実装する手段として，ASIC に実装する方法と FPGA（Field Programmable Gate Array）に実装する方法が存在する．ASIC は専用チップを作成することで特化した高性能を得られるが，回路構成変更時には新たにチップを作成する必要があり，経済的・時間的に高コストである．一方で FPGA は，ASIC よりも低性能だが，特有の汎用的な回路構造により柔軟に回路を再構成可能のため，設計コストが高い DDP の実装に適している．そこで本研究では，FSTP を用いた DDP の FPGA 実装に着目した．

そのため本研究では，より Xilinx 社製 FPGA 特有の論理構造に着目し，DDP を応用し

たセンサハブ (DDSH) のデータ転送アーキテクチャである STP を高速かつ省電力に実装し，簡便に動作保証するためのセルフタイム型データ転送制御回路とそのタイミング検証法を検討する．

本稿において，第 2 章では FPGA 開発ツールを提供する Intel 社と Xilinx 社の FPGA 論理構造を比較し，Xilinx 社製 FPGA における STP 実装時の特性の調査から実装に求められる要件を議論する．最後に，要求要件を満たすための方法について検討する．第 3 章では先述した問題点を解決するために，回路実装法とタイミング検証法のそれぞれの観点から最適化を検討する．第 4 章では，検討した回路実装法とタイミング検証法の評価を行う．第 5 章では，本研究のまとめと今後の課題について述べる．

第 2 章

STP を用いた DDP の FPGA 実装時の課題

2.1 緒言

本章では，DDP を構成する STP，および，STP を構成するセルフタイム型データ転送制御回路の転送制御機能や回路構造について詳説し，ハンドシェイク通信時動作を説明する．次に，一般的な商用 FPGA と商用 FPGA 回路設計ツールを用いた回路設計時のタイミング検証法について述べる．最後に，先行研究によって示されたタイミング検証法，その課題について述べる．

2.2 STP（Self-Timed Pipeline）

データ駆動型プロセッサ DDP(Data-Driven Processor) は，被演算パケットの到着をトリガとして動作するプロセッサである．被演算パケットのマッチング機構とローカルクロックによる転送制御により，多重並列性と省電力性を有している．

本研究における DDP は，セルフタイム型パイプライン STP を用いて設計されているものを対象とする．本節では，STP の機能と構造，STP の実現に必要なセットアップ・ホールドなどの制約と STP を応用して実装する DDP について説明する．

2.2.1　STP の機能

STP は，パケットを適切なタイミングで後段に転送する機能を有している．パケットの転送タイミングは転送制御通信にかかる時間を調整することで実現する．

STP は低消費電力という特徴を備えている．これはグローバルクロックを全てのレジスタに挿入し，一定間隔で常時送信する場合と比較して，隣接回路間の転送制御にグローバルクロックのような全体配線を使用しないこと，転送制御していない部分が電力を消費しないことが主因である．

2.2.2　STP の構成

STP は図 2.1 に示すように，データ転送を制御するためのデータ転送制御回路 (C)，到達したデータを格納するためのデータラッチ (DL)，任意の機能を有するファンクションロジック (Logic) および転送制御タイミングを調整する遅延回路 (Ds，Da) を 1 個のパイプラインステージ (Pipeline Stage) にまとめ，複数段連結した構成である．各 C は隣接する C と転送制御のために Send，Ack 信号線で繋がれており，DL とデータ転送を指示するために cp 信号線で繋がれている．

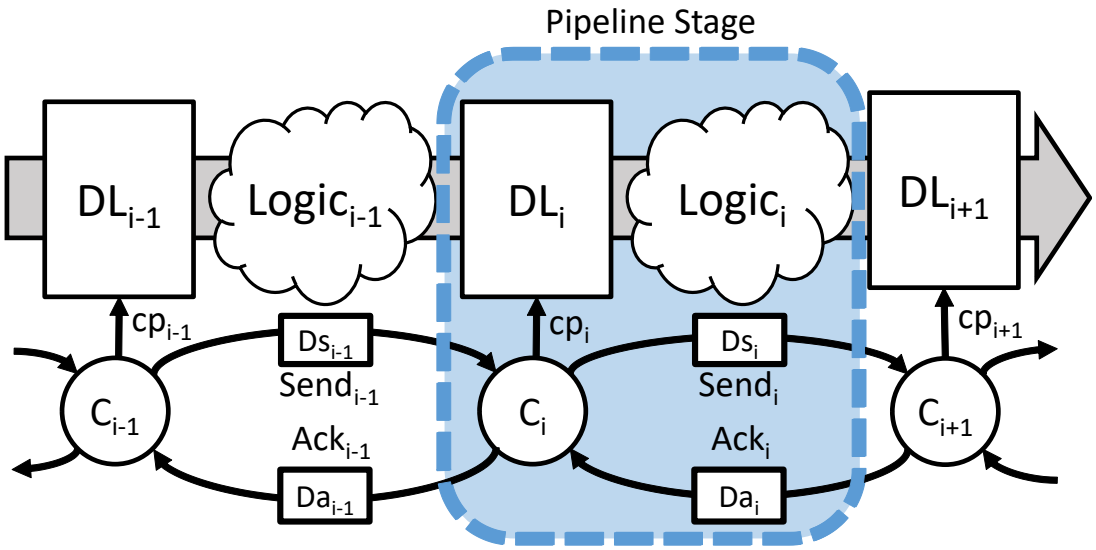


図 2.1　STP の構成

2.2.3 転送制御プロトコル

STP 内の C は隣接する C と 4 相ハンドシェイク通信によるデータ転送制御を行う．図 2.2 に STP の転送制御時タイミングチャートを示す.

cp_i が立ち上がりから cp_{i+1} の立ち上がりまでにかかる時間のことを T_f と呼び，cp_{i+1} の立ち上がりから cp_i の立ち上がりにかかる時間を T_r と呼ぶ．各ステージ DL には，データの到着時間にセットアップ・ホールド制約が存在する．cp_i の立ち上がりが DL_i に伝わるまでに必要なデータが DL_i に到着することを保証するセットアップ時間 Setup time，cp_i の立ち上がりが DL_i に伝わるまでに DL_i 内の転送されるデータが書き換わらないことを保証するホールド時間 Hold time などの時間的制約を守ることで，転送制御の正しい動作を保証している．$Send_{i-1}$ の立ち上がりを受けた C_i の cp_i が立ち上がり，DL_i のデータが転送される．その後，後段素子がもう一段後の C 素子と Send 信号，Ack 信号のやり取りを行った後，cp_i の信号が立ち上がる．この T_f，T_r の時間と cp_i の cp 信号によって DL からデータが転送され，後段の DL に到着するまでの時間を比較し，状態に応じて遅延素子を挿入することで，セットアップ制約・ホールド制約を満たすことができる.

転送制御通信 (ハンドシェイク) を行い，データラッチ解放信号 cp_i を (DL) に出力する．DL は，cp を受け取った時点で前段から出力されているデータを自身に取り込み，後段に転送する．DL から転送されたデータは，Logic を通過し，後段 DL に転送される.

2.2.4 STP の性能

STP の性能は，全てのパイプラインステージのうち，ハンドシェイクにかかる時間が最大のステージの T_f/T_r によって決まる．STP のファンクションロジックでは任意の機能を実現できるが，小規模回路から大規模回路まで幅広く性能を最適化させるためには，遅延回路を除いた T_f/T_r は最小限にしておき，制約の充足には遅延回路を追加して対応することが望ましい.

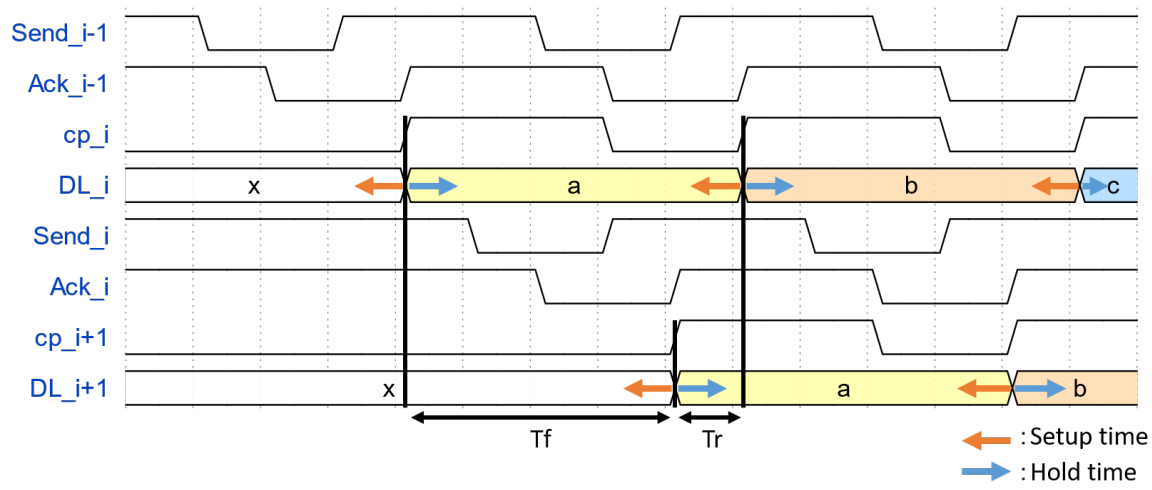


図 2.2　STP の転送制御時タイミングチャート

2.3　セルフタイム型データ転送制御回路 (C 素子)

先述した STP の機能であるハンドシェイクによるローカルクロックを実現するために STP ではセルフタイム型データ転送制御回路 (C 素子) を用いている．また，分流制御やパケット削除などの複雑な STP のパケット転送にはセルフタイム型複合データ転送制御回路 (複合 C 素子) が用いられる．本節では，C 素子の回路構成と各複合 C 素子の役割について説明する．

2.3.1　基本 C 素子

STP の機能を実現するための基本 C（Coincidence）素子について説明する．C 素子は図 2.3 のように 5 入力 NAND を 2 個の SR-latch で挟んだ構造をもつ．Send_in，Ack_out は前段の C 素子から転送制御要求信号を受信し，転送制御許可信号を送信する．Send_out，Ack_in は後段の C 素子に転送制御要求信号を送信し，転送制御許可信号を受信する．基本 C 素子は連結することで，1 対 1 の DL 間の転送制御を行う．しかし，応用に向けて DL にあるパケットを複数 DL に送信することや，複数 DL から送信されたデータを調停するような複雑な転送制御が求められる．そのため，後述する複合データ転送制御回路が用意されている．

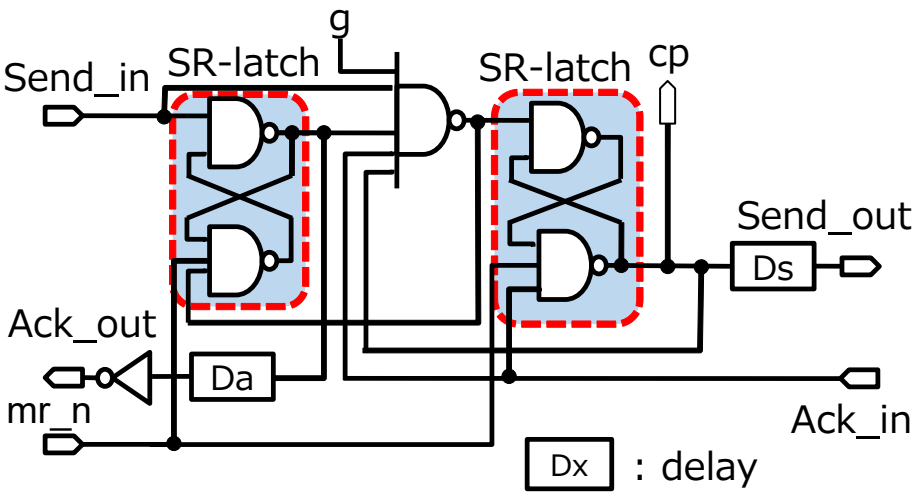


図 2.3　セルフタイム型データ転送制御回路

2.3.2　複合データ転送制御回路

複合データ転送制御回路とは，1 対 1DL の一方向への転送制御に限らず，1 対複数 DL の転送制御や，転送データの無効化（消去），複製などが行える C 素子のことである．

- CB

  CB(C branch) 素子は，1 個の DL のデータを複数の DL から選択的に転送可能な C 素子である．転送先 DL の対象 C 素子に選択的に Send_out 信号を出力するため，基本的な C 素子に NAND ゲート 2 個，後段からの Ack 信号を受ける AND ゲート，出力先を選択するための信号 br，br を格納する DL 追加されている．

- CM

  CM(C merge) 素子は，複数の DL のデータを 1 個の DL で受け取るために，複数のデータ転送制御回路との調停機能を有した C 素子である．

  外部と内部，2 個の入力をそれぞれ受け取るために基本 C 素子を 2 個持ち，それぞれの状態を伝えあうためにアービター（調停回路）を用いて接続されている．また，DL の出力信号を決定するために，セレクタ信号を持つ．

- CX2

  CX2 素子は，複数回パケット転送信号を送信するパケットコピー機能と，後述する CE

と同じ，パケット削除機能を持つ．条件分岐命令などの DDP の演算結果を異なる 2 個の命令で使用する場合に，DL 内データを複数回転送制御するために，回路内に複数個の基本 C 素子を持ち，パケットコピー機能とパケット削除機能の有効・無効を判定するための cpy，exb 信号とそれらの情報を格納するための DL を持つ．

- CE

  CE 素子は，パケットの削除機能を有した複合 C 素子である．前ステージのファンクションロジックから入力される 1 ビットの exb 信号の出力結果によって，パケットを削除するかそのまま転送制御を行うか決定される．

2.4　DDP

DDP は 9 つのパイプラインステージから構成されている．DDP の構成を図 2.4 に示す．

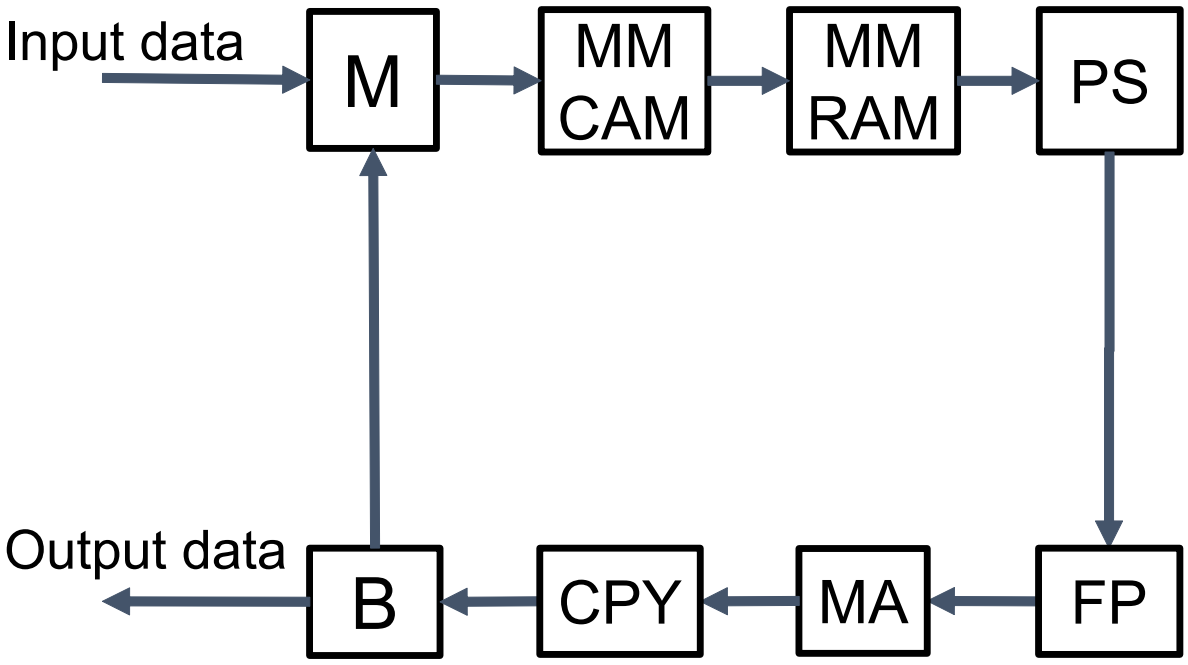


図 2.4　DDP の構成

DDP を構成する各パイプラインステージ内 *Logic* の処理を以下に示す．

- パケット合流機構 M (Merge Unit)

外部から入力されるパケットとパイプラインを周回してきたパケットの合流を調停し，パケットを CST に出力する．

- 定数読み出し機構 CST (Constant Memory)

  定数と読み出しのためのオペレーションコードが格納されたコンスタントメモリという ROM(Read Only Memory) があり，定数命令実行時，パケットの持つオペレーションコードに従ってコンスタントメモリから定数の読み出しを行って，入力パケットに付加し，MM へ出力する．定数命令以外の命令はそのまま MM へ出力する．

- パケット待ち合わせ機構 MMCAM (Matching Memory Content Addressable Memory)

  パケットの待ち合わせのために一時的に入力パケットをレジスタに保持し，MMRAM にデータを格納する．対応するパケットが到着すると MMRAM からデータを読み出し，ALU に出力する．定数命令を実行するパケットの場合は待ち合わせるパケットが存在しないため，そのまま ALU に出力する．

- MMRAM (Matching Memory Random Access Memory)

  CP 信号と同期して，MMCAM からの要求に応じてパケットの読み出しと書き込みを行う．どちらもパケットからのアドレス指定に従う．

- 演算機構 ALU (Arithmetic Logic Unit)

  パケットが保持している命令コードをもとに，各種演算処理を行い，演算結果をパケットに書き込む．

- データメモリ機構 DMEM (Data Memory)

  演算結果などを格納するためのデータメモリと呼ばれる RAM(Random Access Memory) があり，入力パケットがロード命令の場合，データメモリから演算データを読み出し，パケットに付加する．ストア命令の場合，パケット内の演算データをデータメモリに格納する．

- パケット複製機構 COPY (Copy)

  パケット内の複製フラグを読み取り，フラグが立っていればパケットの複製を行い，そ

うでなければ到着したパケットをそのまま PS に出力する．

- 命令フェッチ機構 PS (Program Storage)

  実行する命令コードと宛先ノード番号を格納するプログラムストレージと呼ばれる ROM があり，到着したパケットの保持する情報から実行する命令コードと宛先ノード番号をプログラムストレージから読み出し，パケットの書き換えを行い，B に出力する．

- 分岐機構 B (Branch Unit)

  パケットが保持する情報から，外部へ出力させるか内部で周回させるかの分流制御を行う．

2.5 FPGA

FPGA（Field Programmable Gate Array）とは，設計者が任意の論理回路構成をプログラムできるような集積回路である．FPGA の種類によって回路構成が異なり，FPGA ベンダごとに使用する EDA ツールも異なる．本研究において，FPGA は Xilinx 社製「zynq-7000」シリーズ，EDA ツールは Xilinx 社製「Vivado」を使用している．本稿では zynq-7000 シリーズと Vivado の性能・機能等を概観し，DDSH のアーキテクチャを詳説し，FPGA に DDSH を実装する際に考えられる課題について検討する．

2.5.1 FPGA の機能

FPGA はハードウェア記述言語によって記述された回路を，FPGA チップを用いて動作させることができる．以下は FPGA を用いて実現できる機能の一例である．

1. ディジタル信号処理（DSP）

   フィルタリング，変調，復調などのリアルタイム信号処理に使用．Fast Fourier Transform(FFT) や Finite Impulse Response(FIR) フィルタリングなど，複雑な演算処理も可能である．

2. 高速データ転送

USB，SPI，I2C などの異なる通信プロトコル間のインターフェースや特定用途向けのカスタム通信プロトコルの実装などができる．

3. プロトタイプ開発

ハードウェア開発のためのプロトタイプ開発に使用することができる．特に ASIC のように特定用途の半導体製造の前段階として，FPGA を用いた設計テストと検証を行うことができる．

2.5.2 FPGA の構成

FPGA には任意の回路を実現するための論理要素が存在する [6]．本研究では，Zynq-7000 シリーズについて取り上げる．Zynq-7000 シリーズは LUT(Look Up Table) 方式を採用している．LUT 方式とは，実現したい回路の入出力を真理値表として，それを構成メモリと呼ばれる SRAM セル群に与えることで任意の回路を実現している．Xilinx 社製 FPGA zynq-7000 シリーズの構成は，プロセッサで処理する PS とプログラマブルなロジックが配置されている PL に分かれている．PS は，Processing System の略称であり，Zynq プロセッサの Arm コア部分のことである．PS には任意の回路を実現する機能はなく，PL はさらに CLB（Configurable logic blocks）ごとに分かれている．CLB は図 2.5 に示すように 2 つのスライスから構成され，各スライスには 6 入力 LUT が 4 つ，セレクタ (MUX) が 3 つ，桁上げ先見加算器 1 つ，FF8 個からなる．各スライスで望みの機能を実現するためにどの回路モジュールを使用するか，回路合成や配置配線最適化の観点からソフトウェア上で選択される．

先行研究では，データ転送制御回路を FPGA に実装し，簡便にタイミング解析を行うため，各データ転送制御回路の論理ゲートを LUT として置換し，転送制御ハンドシェイクに必要な send, ack 信号の出力に遅延回路を挿入し，タイミングレポートの抽出・積算からタイミング検証を行っていた．しかし，その手法は intel 社製 FPGA 用に考案された手法であり，Xilinx 社製 FPGA に応用する場合，論理セル単位でリソースを浪費していた．本研究では Xilinx 社製 FPGA の論理セルに特化したハードウェア記述言語の記法で記述する．

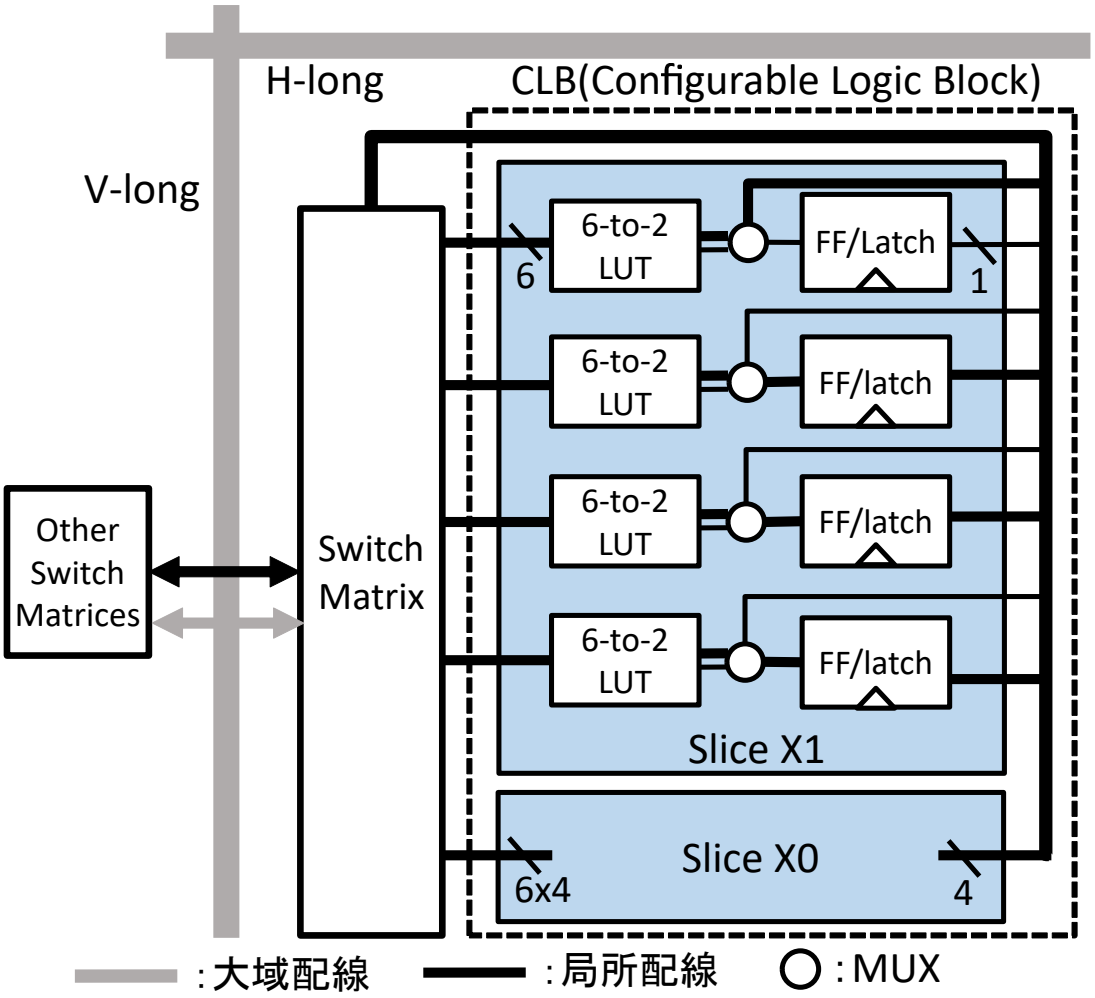


図 2.5　Xilinx 社製 FPGA の論理スライス構造

2.6　タイミング検証

一般的なタイミング解析は，コーナー解析により動作保証する．コーナーとは半導体製造プロセスにおけるチップのばらつき，電圧，温度の大小からなる回路性能の最大，最小遅延のことであり，本研究で対象としている静的タイミング解析では，遅延が min/max になる PVT の条件を Fast/Slow コーナーとよび，それぞれの Setup/hold 制約に対して Fast/Slow コーナーのマルチコーナー解析を行っている．

セルフタイム型データ転送制御を用いた STP には，図 2.6 のようなタイミング制約が存在する．

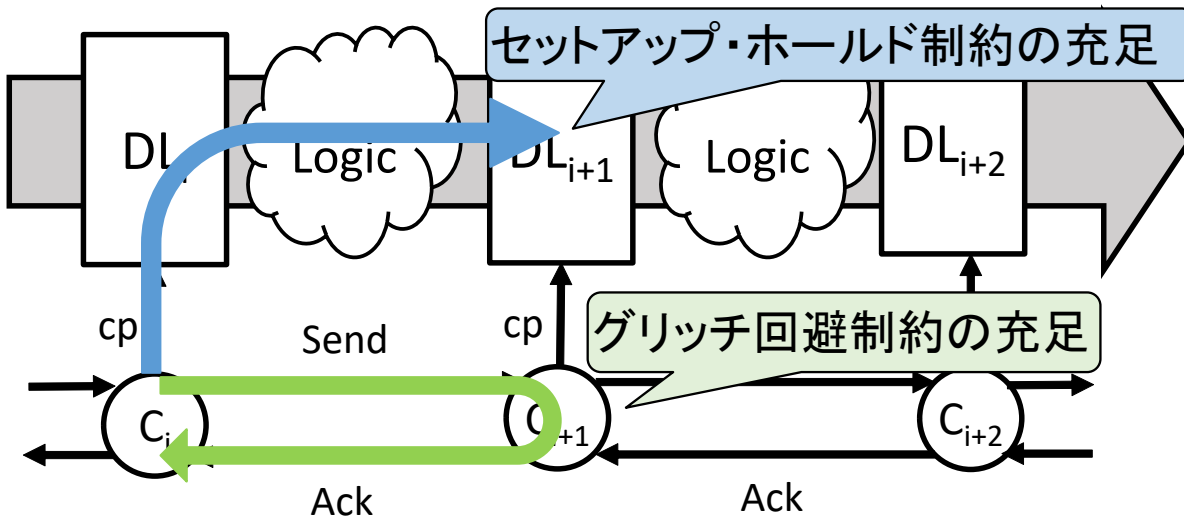


図 2.6　STP の動作保証要件

2.7　先行研究における C 素子構成とタイミング検証

先行研究では，C 素子のレジスタ置換と疑似クロック挿入による疑似同期化を行い，DDP のタイミング情報の抽出と，タイミング検証ツールによるセットアップ・ホールド違反の判定が行われている [5][7][8].

本節では，同期回路と非同期回路のタイミング検証の違いについて，タイミング検証フローを用いて説明する．FPGA におけるタイミング検証フローを図 2.7 に示す．タイミング検証フローは大きく合成系，レイアウト系，実機系の処理と，タイミング制約の充足判定後の遅延量の変更と再コンパイルなどの遅延調整系処理に分けられる．合成系処理では，HDL で記述した回路をゲートレベルに変換し，ネットリストに置換する．その後，レイアウト系では，ネットリストと制約ファイルから，回路の各ゲートを FPGA 論理資源のどこに配置するか決定する．制約ファイル等で配置に指定がない場合，FPGA 回路設計ツールによる自動配置が行われる．その後，同期回路であればクロックを基準としてタイミング情報の抽出を行い，タイミング違反の有無を判定する．非同期回路であれば，疑似クロックを挿入とタイミングレポートの出力を tcl コンソールを用いて行う．

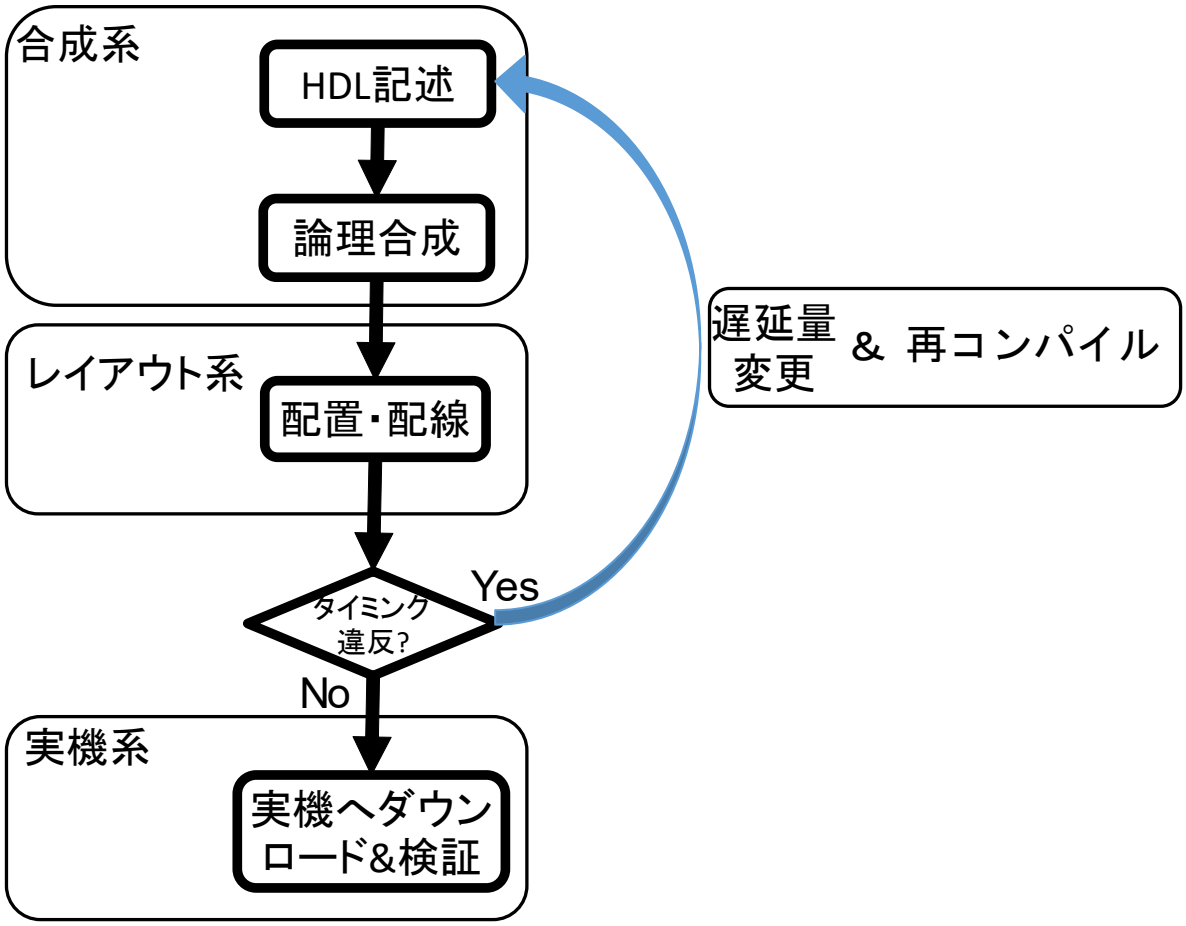


図 2.7　FPGA タイミング検証フロー

2.7.1　C 素子の拡張

転送制御回路（C 素子）の基本構造を図 2.8 に再掲する．通常，Quartus は非同期回路である転送制御回路中の任意の 2 点間のタイミング情報は抽出できない．そこで転送制御には不要なラッチ (Lopen) を追加することで NAND ゲートをレジスタとして認識させタイミング情報を抽出する．また，クリティカルパス遅延の削減のため，NAND ゲートを LUT に置き換える．この Lopen の追加と LUT への置き換えを疑似クロック入力付き LUT への置き換えと呼び，拡張した回路を CCORE と呼ぶ．図 2.9 に CCORE の回路図を示す．LUT1 と LUT2，LUT4 と LUT5 はそれぞれ SR フリップフロップを LUT に置換したため，NAND ゲートの出力と対になったフリップフロップの状態に従った出力を LUT に保存する．

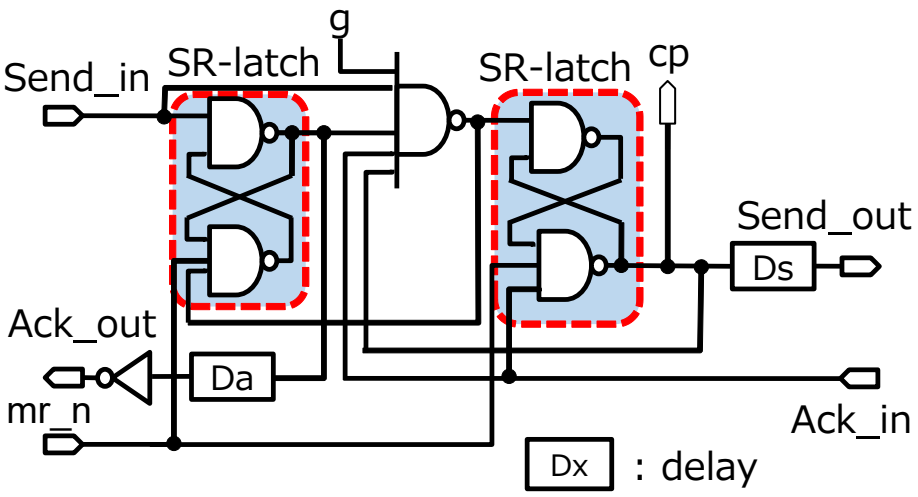


図 2.8　セルフタイム型データ転送制御回路

拡張した回路を STP に搭載し，パケットの転送制御を行う回路を設計ソフトで回路合成することによってタイミング情報を抽出できる．次に先行研究のタイミング情報抽出法と注意点を述べる．

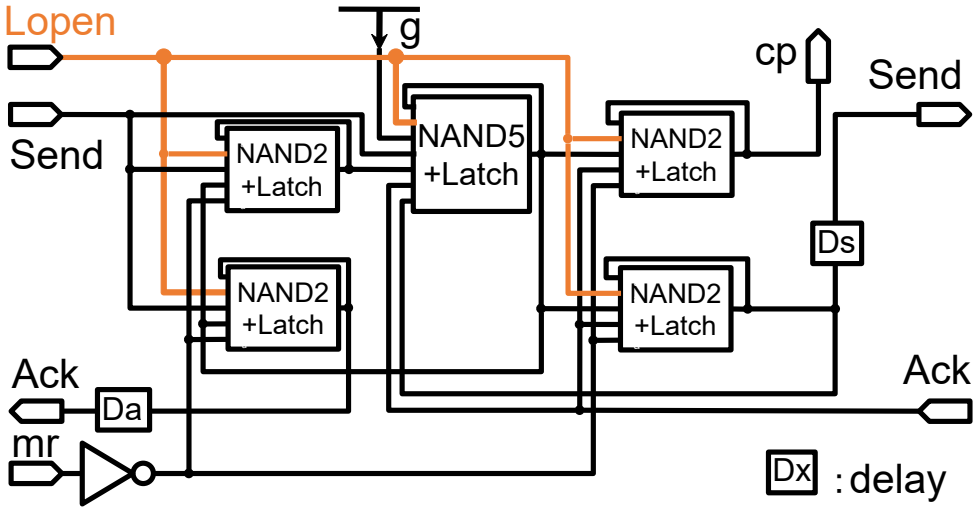


図 2.9　CCORE の回路図

2.7.2　FPGA 設計ツールによる実装

設計した回路の遅延の最適化を行うためには，FPGA 上での配置配線をできるだけ短くするため，LE の配置配線を行う必要がある．ここでは，FPGA への実装法を Intel 社の Quartus Prime Standard Edition 18.0 を用いて説明する．

1. プロジェクト作成

   Quartus でプロジェクトファイルを作成し，設計した回路の合成を行う．

2. パーティション (Partition) 設定

   Quartus には回路面積を最適化する機能があり，必要ないと判定された回路内のモジュールや信号の消去や，設計したモジュールと同じ機能を有する，より回路面積が小さい予期せぬ回路に変換が行われる．それではタイミング解析が正しく行えないため，各モジュールごとにコンパイルが行えるようにパーティションを設定することで，設計ソフトでの回路の自動最適化を防ぐ．

3. リージョン (Region) 設定

   Region とは回路の配置配線を設定するための設計ソフトの機能である．Region の設定により，モジュールごとに配置をまとめることで配線遅延を削減する．

4. タイミング解析

   Quartus で TimingAnalyzer を開く，制約ファイルを用いて Lopen をクロックとして設定し，TCL スクリプトを動作させることでタイミング解析とタイミング情報の抽出を行う．

2.7.3　タイミング検証ツール

タイミング検証ツールは以下の

- パス指定スクリプト (path.json)
- タイミング情報ファイル (TimingReport.txt)

- タイミング判定プログラム (TimingChecker.py)

の 3 つから構成されている．TimingReport.txt は Quartus の TimingAnalyzer で tcl スクリプトを実行することにより自動生成される．PathNum には検証したいパスの数を指定し，Path の内部で具体的な回路を記述する．Data にはパイプラインステージの前段 DL，後段 DL を指定し，LaunchStart，LaunchClock に前段 CCORE の LUT3，LUT4 を指定し，LatchStart，LatchClock に後段 CCORE の LUT3，LUT4 を指定する．T_f，T_r には通過する経路の端点と経路内のレジスタを指定する．

タイミング情報ファイルと経路指定スクリプトを TimingChecker.py で読み込み，path.json で指定したパスに従ってタイミング情報からデータパス時間とデータ到着時間を抽出し，計算してタイミング判定する．

タイミング検証は，ターミナル上で実行可能になっており，TimingChecker.py による判定結果が表示される．

ターミナル上で以下のコマンドを実行し，タイミング制約の充足を確認する．

実行コマンド

```
python TimingChecker.py timingreport.txt path.json
```

出力結果よりタイミング制約が満たされていれば OK を出力し，そうでなければ TIMING NOT MET を出力する．

2.8　従来タイミング検証手法の課題

従来のタイミング検証手法は先述した FPGA タイミング解析フローを辿るが，レイアウト系の配置・配線について，各モジュールの大まかな配置をリージョン指定し，細かな配置をツールの自動配線に任せた手法を取っていた [9]．これにより，回路を凝集して配置することができ，回路全体の配線遅延の削減に成功していた．しかし，リージョン指定のみの指定では，各リージョン内の論理合成結果，配置結果がコンフィグレーションするたびに変化するため，タイミング検証で事前に取得したタイミング情報が再利用できず，全て再取得す

る必要があった．そのため，タイミング検証コストが膨大になっていた．

本研究では，FPGA 構造に適した回路合成を行い，FPGA 構造に適した配置指定を行うことで，STA の結果を部分的に再利用し，最終的に制約の充足に必要な遅延量の見積もりができるタイミング検証の実装を目指す．

2.9　結言

本章では，DDP を構成する STP の特徴である低消費電力性について述べ，アーキテクチャと動作原理であるセルフタイム駆動 4 相ハンドシェイク通信についてタイミングチャートを用いて説明し，セットアップ・ホールド制約について述べた．また，STP を実現するためのセルフタイム型データ転送制御回路の回路構成とセルフタイム型データ転送制御回路の拡張回路であるセルフタイム型複合データ転送制御回路とこれらを用いて実現した DDP の各ステージ構成について述べた．また，これらの回路を実装するハードウェアである FPGA の機能や構成について述べた．次に，これらの回路の正しい動作を保証するためのタイミング検証手法とその課題について述べた．最後に，本研究の方針について述べた．

本章では，設計フローの中でも論理合成・自動配置による検証箇所の変更について述べている．しかし，今後の課題として自動配線の統制について，スイッチマトリックスのスイッチング条件や，配線遅延を含めた遅延回路のミクロな変更の遅延時間に及ぼす影響などを追加で調査することによって，より Xilinx 社製 FPGA に適した実装方法とそれに基づくタイミング検証法を検討する必要がある．

第 3 章

FPGA 論理スライス構造に適した回路実装法とタイミング検証手法

3.1 緒言

本章では Xilinx 社製 FPGA の論理スライス構造について詳説する．次に論理スライス構造に適したセルフタイム型データ転送制御回路の構成法について述べ，論理スライス構造に適したセルフタイム型データ転送制御回路によって構成された STP の論理スライス構造に適した配置法，提案回路と提案配置を組み合わせたタイミング検証法とその利点について述べる．

3.2 Xilinx 社製 FPGA の論理構造

Xilinx 社製 FPGA は一般的な商用 FPGA と同様，論理ブロックから構成されている．Xilinx 社製 FPGA の論理ブロックは CLB（Configurable Logic Block）と呼ばれる．図 3.1 は CLB とその周辺回路である．CLB は 2 個の Slice から構成されており，各 Slice の入出力配線を選択する SM（Switching Matrix）が各 CLB と接続している．Slice は，LUT，FF/Latch，CARRY，MUX から構成されており，FF としてのみ使用可能な 4 個のレジスタと FF/Latch どちらにも使用できる 4 個のレジスタから構成される．本研究では，論理要素や配線要素をまとめて論理スライス構造と呼称している．信号伝搬のためには各 Slice にクロック信号を挿入する必要があるため，スライス使用量が少ないほど，クロック信号によっ

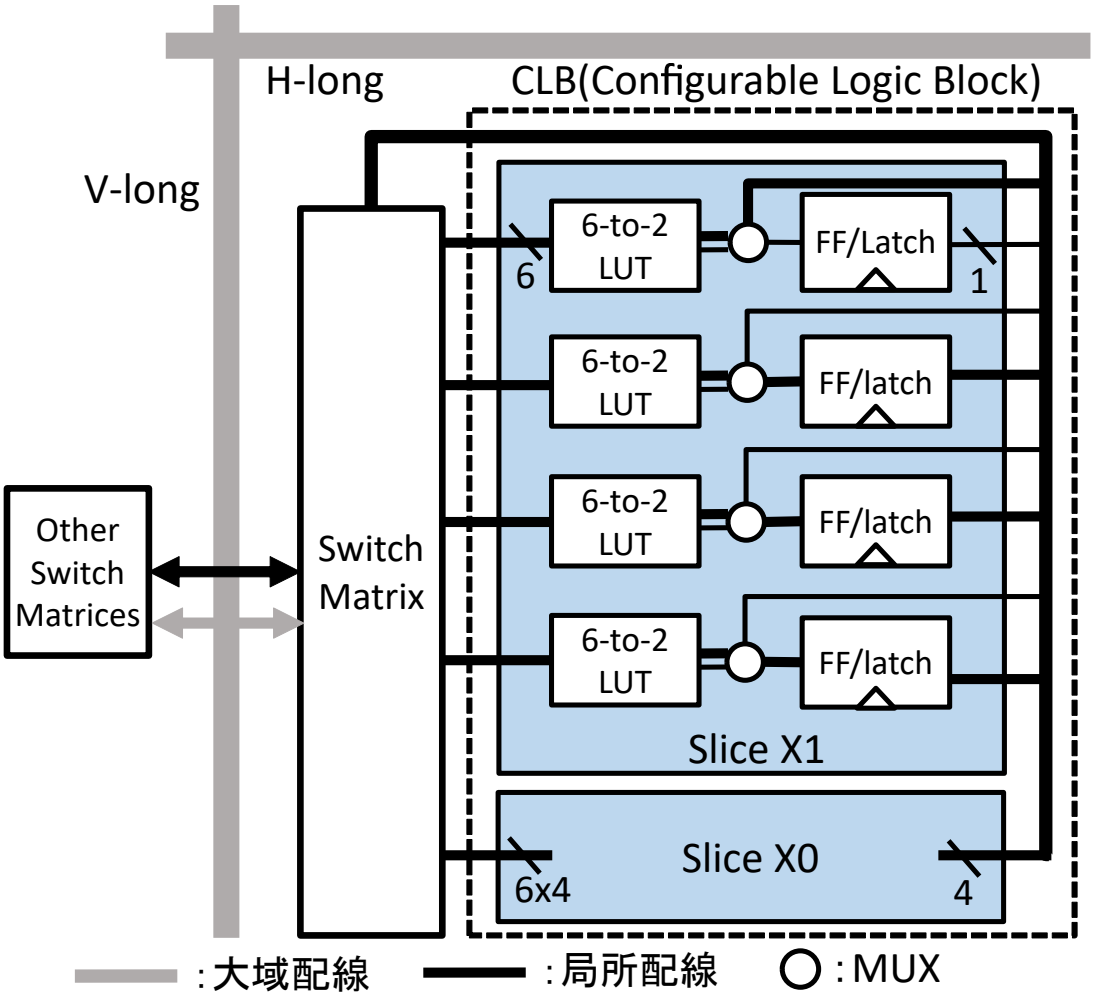


図 3.1　Xilinx 社製 FPGA の論理スライス構造

て生じる電力消費を削減できる．

また先述の通り，Slice 内には LUT と FF/Latch が配置されている．特に配置についての制約を設計ツールで指定しない場合，実装回路のネットリストが使用する LUT と FF/Latch の配置結果はツール内アルゴリズムの結果に依存する．このアルゴリズムは回路設計者には閲覧できないため，ブラックボックス化されている．図 3.1 の LUT + FF/Latch の組み合わせ，つまりどの LUT からどの FF/Latch に回路が配置されるかよって所要配線遅延が異なる．Slice 内部の配線によって LUT と FF/Latch が接続している配置をペア配置と呼ぶ．ペア配置による配線は SM を経由する配線よりも遅延が小さいため，C 素子や STP の実装時にペア配置を優先的に活用することで，回路性能の向上が見込める．

また，大域配線は限られた配線資源であり，大域配線の使用率が増加するとスイッチングコストが生じ，配線遅延が増加する．そのため，局所配線を極力使用するように配置から誘導することで，配線遅延による性能低下を防止できる．

また，遅延回路は自動配置前に使用回路数を増減させることで調整するが，ミクロな配置においては，自動配置後の配線遅延時間の増減が回路数の増減と一致しないことが明らかになっている．

論理スライス構造を有効活用した配置法とタイミング検証法に向けた方針は以下の 3 つである．

- 1 スライス内に配置するために，回路構成を変更

  使用するスライス数を削減するため，従来の C 素子の回路構成を変更し，配置配線を行う．
- ペア配置の積極的な活用

  ペア配置による性能向上のため，制約ファイルと tcl コマンドを用いて，配置の固定を行う．
- 局所配線と大域配線の使い分け

  STP を同クロックリージョン内に等間隔に配置し，n ステージパイプラインを 2 分割し，それぞれを大域配線に割り当てる．

3.3　提案回路実装法

従来回路（図 3.2）を再掲する．先述の通り，従来 C 素子の各 NAND ゲートと同機能を保持したまま，LUT+FF/Latch に置換する場合，LUT を 6 個，FF/Latch を 5 個使用する必要があった．これらを論理スライス構造に配置する際，最小限の Slice 使用の配置の場合，Slice を最低 2 個使用する．そこで，図 3.3 に示すような C 素子の回路構成を提案する．提案回路は SR - latch を Preceded SR-latch，Succeeded SR-latch としてそれぞれ 1 個の LUT+Latch を用いて実現することで，C 素子全体で LUT を 4 個，FF/Latch を 3 個使

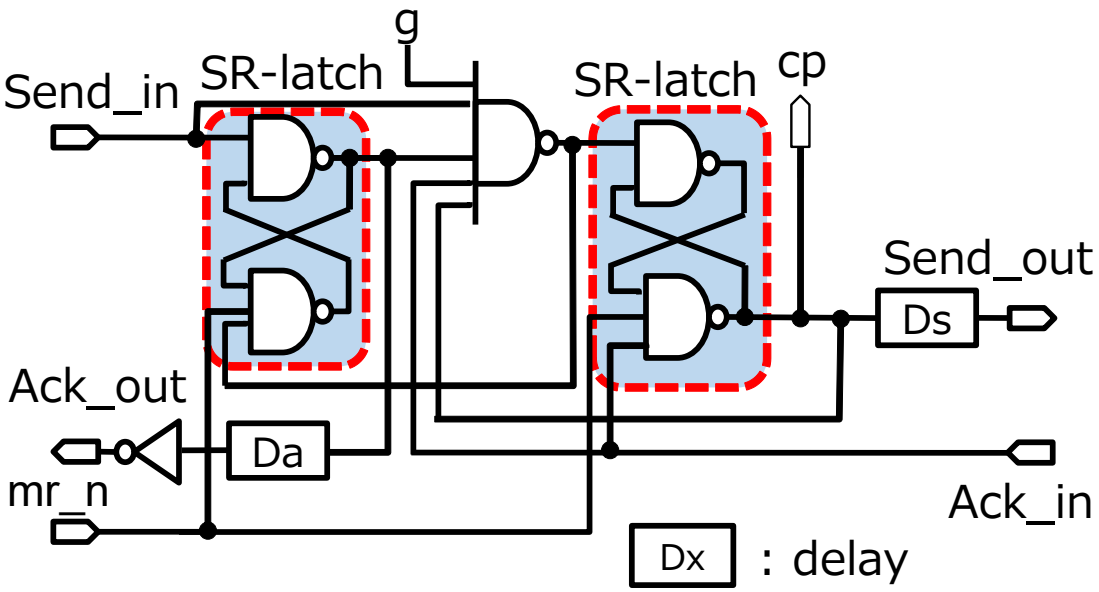


図 3.2　C 素子

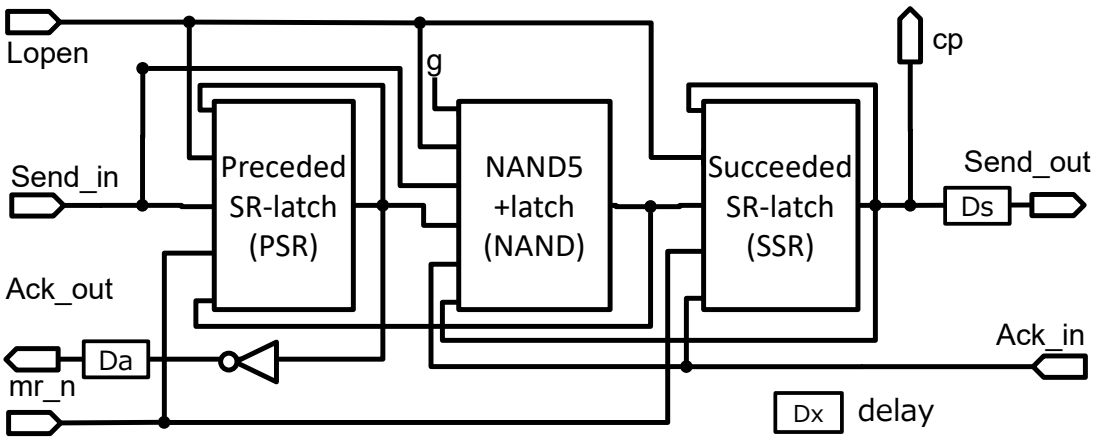


図 3.3　LUT-Latch ペアを有効活用可能な C 素子

用する．そのため，最小限の Slice 使用の配置の場合，Slice を最低 1 個使用する．本回路を実現するために，Xilinx 社製 FPGA にあらかじめ搭載されているデザインエレメントである LDCE プリミティブと LUT4 プリミティブを使用する．これにより，論理合成結果を固定でき，安定したタイミング検証が可能となった．

3.4　提案配置

提案配置を図 3.4 に示す.

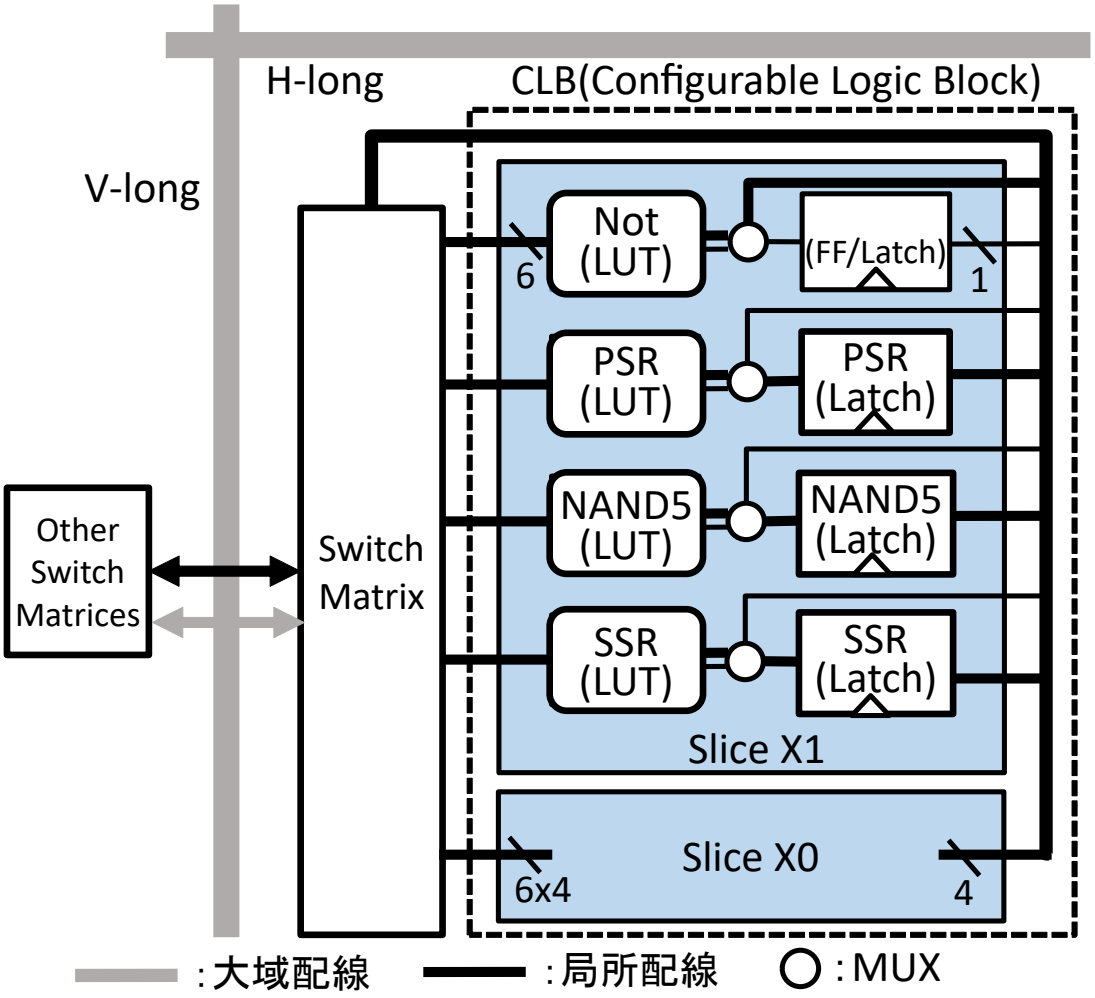


図 3.4　論理スライス構造に適した C 素子の配置法

PSR，NAND5，SSR の LUT+Latch をそれぞれペア配置に配置する．

次に，遅延回路の配置法について説明する．遅延回路は自動配置配線時，LUT 単位のミクロな遅延回路の増減と一致しない．そこで，図 3.5 のように一方向に遅延回路を配置することで，ツールによって選択される配線を統制できる．また，図 3.6 のように，スライス内の 4 個の遅延回路を 1 個の遅延ブロック DELAY とみなし，遅延ブロック単位で遅延量を増減させることで，よりマクロな遅延量の調整が可能となる．図 3.6 の破線は各 CLB を示

している．

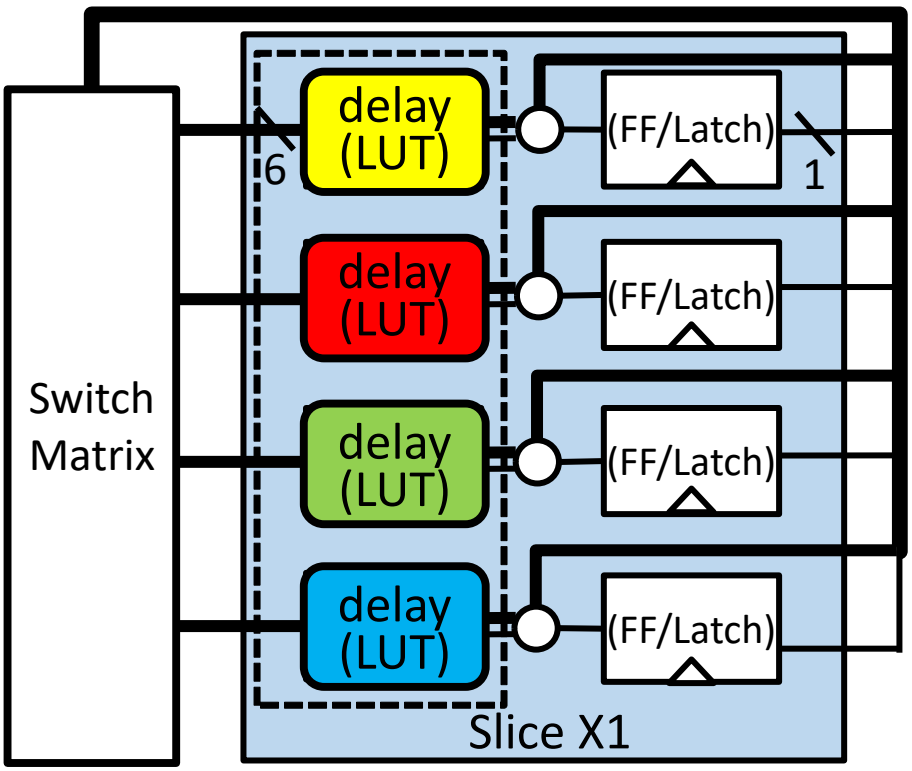


図 3.5　論理スライス構造に適した遅延回路の配置法

3.5　提案タイミング検証法

STP を用いた DDP のタイミング検証法を提案する．基本 C 素子と複合 C 素子を先述した STP と同じ方法で配置し，DL と Logic を基本 C 素子と複合 C 素子からなる STP の周囲に配置し，タイミング検証を行う．

タイミング検証は以下の手順を制約が充足するまで繰り返す．

1. タイミング制約に基づくタイミング情報の抽出
2. タイミング制約充足判定
3. 遅延回路の付加

タイミング検証はレジスタに対してクロックを挿入し，タイミング情報を抽出する．タイ

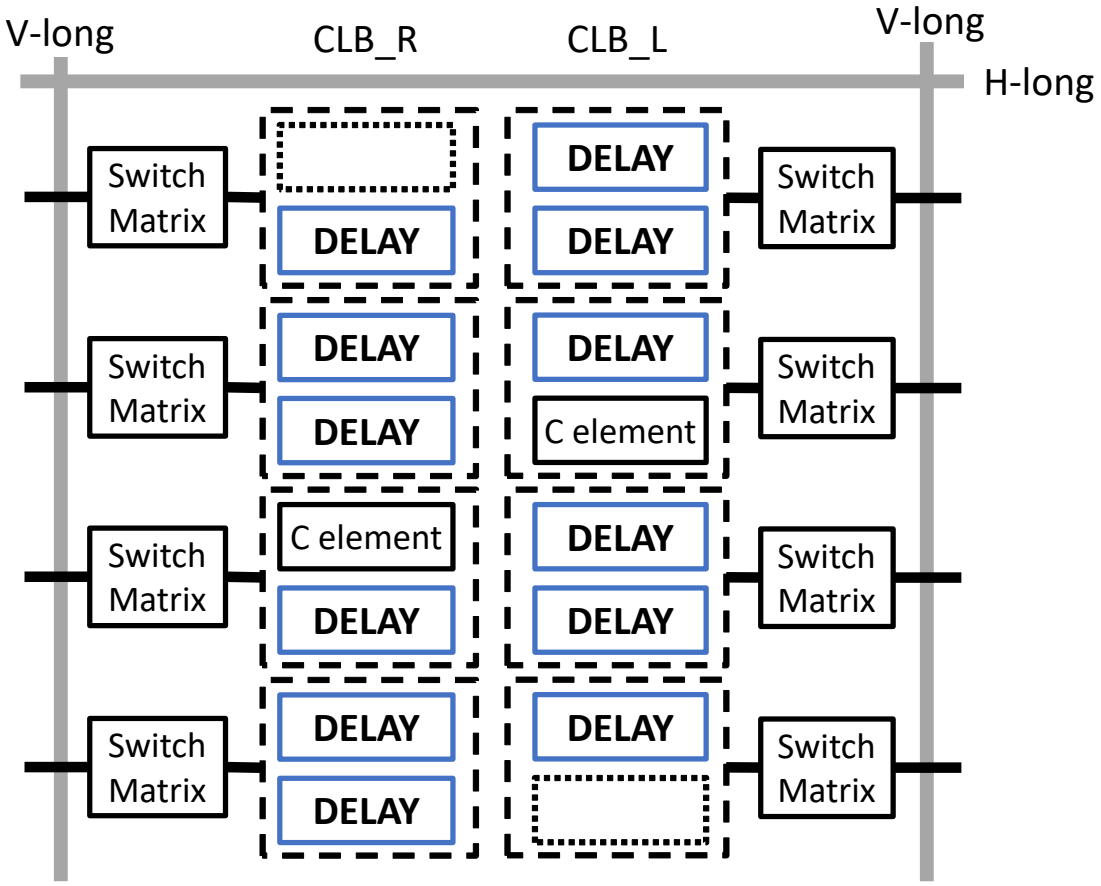


図 3.6　論理スライス構造に適した遅延回路と C 素子配置の詳細

ミング制約の充足を確認するため，スライス番号と使用 LUT，FF/Latch を指定することでレジスタ間遅延時間を抽出している．従来手法では遅延回路を付加するごとに配置配線結果が変わり，結果として全てのタイミング情報を再度抽出する必要があった．そこで，提案タイミング検証法では配置を固定することで，タイミング情報の再抽出箇所を遅延素子周辺のみに制限できる．また，Intel 社製 FPGA から Xilinx 社製 FPGA に変更したことで，疑似クロック挿入や配置の指定を行う制約ファイルが sdc ファイルから xdc ファイルに変更されている．

3.6　結言

本章では，Xilinx 社製 FPGA の論理スライス構造について詳説し，論理スライス構造に適したセルフタイム型データ転送制御回路の構成と論理スライス構造に適したセルフタイム型データ転送制御回路と遅延回路の配置方法について述べた．次に，セルフタイムパイプライン回路の配置法について述べ，そのタイミング検証法について述べた．本研究では，基本 C 素子を 1 スライスに収める回路構成法が提案されているが，その他の C 素子（CB，CE，CM，CX2）などについても，可能な限り少ないスライス数で過密配線を防ぐような回路構成法の検討が残されている．

第 4 章

検証・評価

4.1 緒言

回路実装と提案配置についての評価を行う．はじめにペア配置の配線遅延短縮効果を検証し，提案配置の性能を評価する．次に提案配置の T_f/T_r のばらつきについて述べる．最後に検討済である基本 C 素子による STP のタイミング検証法と複合 C 素子のスライスへの配置法についての評価を行う．

4.2 評価環境

本研究では，以下のツールやハードウェアを使用した．

- 回路設計ツール

  Vivado 2022.1
- FPGA チップ

  Xilinx 社 Zynq7000-Z7020

4.3 評価条件

以下の条件で評価を行う．

- STP データ転送性能
- FPGA 回路リソース使用量

- 設計コスト

4.4　配線遅延傾向の検証

スライス構造の CLB 間配線遅延の傾向を検証するために，基本 C 素子を 2 個直列に接続して実装された 2 ステージ STP に対して，配置による T_f の配線遅延の違いを調査した．表 4.1 に結果を示す．2 段 STP には前段 C 素子，遅延回路，後段 C 素子があり，配置詳細には前段 C 素子からの相対的な後段 C 素子の配置位置について，北（N），西（W）と定義している．N_1 は後段の C 素子と同じ CLB に遅延素子を配置し，N_2 は前段の C 素子と同じ CLB に遅延素子を配置している．また，前段 C 素子，遅延回路，後段 C 素子がそれぞれ別の CLB に配置されている配置は，遅延回路の相対位置を加えた配置を NN と呼ぶ．そして，2 出力 LUT を活用し，前段 C 素子と遅延回路を同スライスに配置することで使用スライスを最小限に抑えた配置を min と定義し，min の後段 C 素子の配置方向によって min_N，min_W と定義した．なお，自動配置は配置が自動で行われ，個別に指定する事項が存在しないため，配置詳細は空欄となっている．これらすべての配置に対して，タイミング検証ツールを用いてタイミング情報の抽出を行い，遅延時間の調査を行った．

配置個別指定かつ使用 LUT を重複させない NN，N_1，N_2，W の配置構成は，自動配置と比較して，配線遅延が小さく，転送性能が向上しているが，使用スライス数を最小限に抑えた min_N，min_W の配置構成は，自動配置よりも性能が低下することが改めて確認できた．

4.5　提案配置の性能評価

LUT + Latch のペア配置の回路性能を評価するため，1 スライスに収めることが可能な回路構成の C 素子について，ツールによる自動配置配線時と図 3.4 のような配置の個別指定配置配線時の提案回路を実装し，タイミング検証ツールを用いて回路内の配線遅延情報を抽出し，比較した．図 4.1 に結果を示す．

表 4.1　2 段直列 STP の性能表 [ns]

配置	auto. place	constrained place					
配置詳細		NN	N_1	N_2	W	min_N	min_W
CLB 数	2	3	2	2	2	2	1
Slice 数	3	3	3	3	3	2	2
Path1	4.316	3.599	2.977	2.869	3.423	4.855	4.741
Path2	1.863	2.188	2.181	2.181	1.709	1.582	1.792
Path3	0.975	0.930	0.930	0.930	0.930	1.243	1.243
Path4	4.982	3.242	2.937	2.959	3.315	4.502	4.567
Path5	0.565	0.930	1.321	1.321	0.930	0.933	1.14
Total	12.701	10.889	10.346	10.26	10.307	13.115	13.483

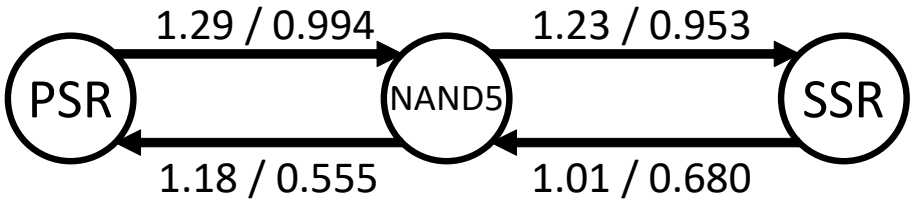


図 4.1　C 素子の内部遅延 (auto[ns] / proposed[ns])

結果，PSR から NAND5 で約 30%，NAND5 から SSR で約 29%，SSR から NAND5 で約 49%，NAND5 から PSR で約 112%の性能向上が確認できた．

4.6　提案配置の T_f/T_r ばらつき評価

提案 STP 配置の T_f/T_r のばらつきを評価するため，C 素子 8 個と遅延回路で構成された 8 段パイプライン回路を設計した．C 素子は基本 C 素子を用いて実装し，遅延回路は LUT プリミティブを 12 個縦列に接続することで実装した．また，配置配置は，ツールによる自動配置と図 4.2 のように隣接する CLB かつ大域配線を 2 つ使用するように環状 STP を配置し，どちらもツールに自動配線を用いて配線している．どちらの回路もタイミング検証

ツールを用いてタイミング情報の抽出を行い，データ転送性能指標である T_f，T_r を比較した．タイミング解析は，Fast/Slow コーナーのマルチコーナー解析である．

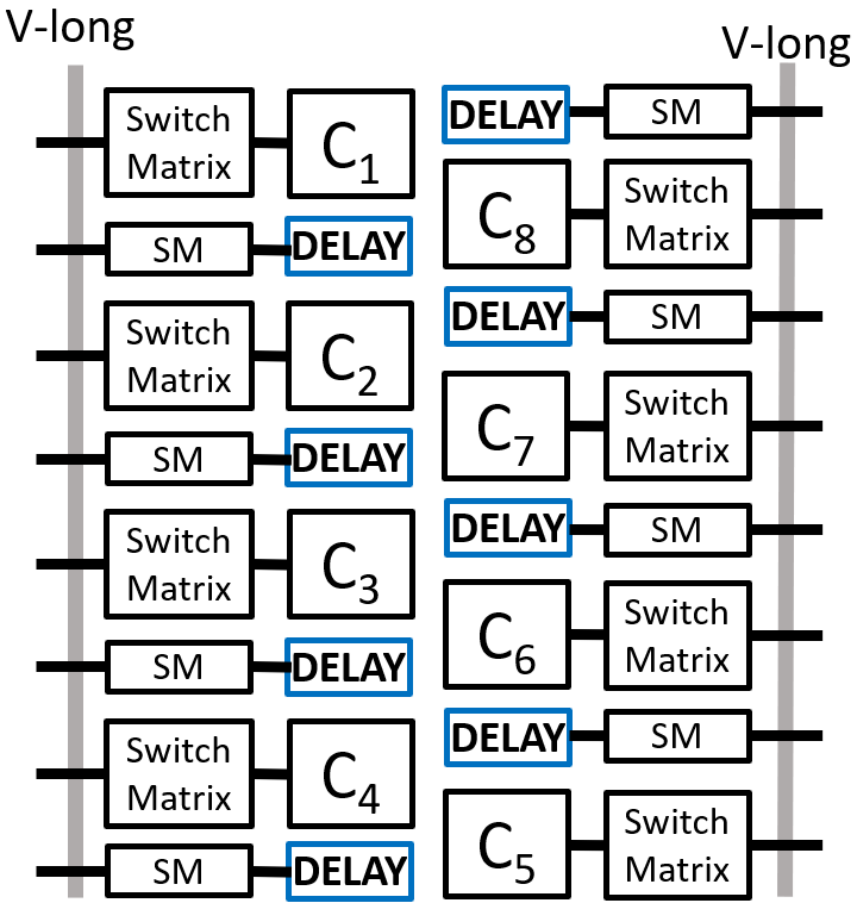


図 4.2　STP 評価用配置

パイプラインステージ全ての C 素子の T_f，T_r の結果を表 4.2 に，それぞれの平均 T_f，T_r と平均値との最大差を表 4.3 に示す．

性能ばらつきの指標である，最大差（Max. Diff.）が T_f においては約 0.06 倍に低下し，T_r においては 0.13 倍に低下している．

また，8 段 STP の T_f，T_r の箱ひげ図を図 4.3 に示す．

以上の結果より，明らかに配線遅延が均質化されており，本研究における提案手法のばらつきの低減効果が確認できた．

表 4.2　8 段環状 STP の性能表 [ns]

	auto. place		proposed place	
	T_f	T_r	T_f	T_r
c1	26.765	10.669	24.124	8.869
c2	34.191	9.659	24.126	8.527
c3	30.92	9.782	24.124	8.527
c4	36.77	13.6	24.49	8.527
c5	34.01	12.919	24.468	8.848
c6	31.887	12.802	24.468	8.983
c7	29.352	10.312	24.593	8.983
c8	30.69	13.19	24.093	8.983

表 4.3　8 段環状 STP の性能ばらつき [ns]

	auto. place		proposed place	
	T_f	T_r	T_f	T_r
Average	31.8	11.6	24.3	8.78
Max. Diff.	5.06	1.98	0.283	0.254

4.7　タイミング検証の評価

本研究において，以下の項目は未着手である．

- タイミング検証の精度検証（SM 内再配線の影響）
- 複合 C 素子を含む STP のタイミング検証法
- DDP 全体のタイミング検証

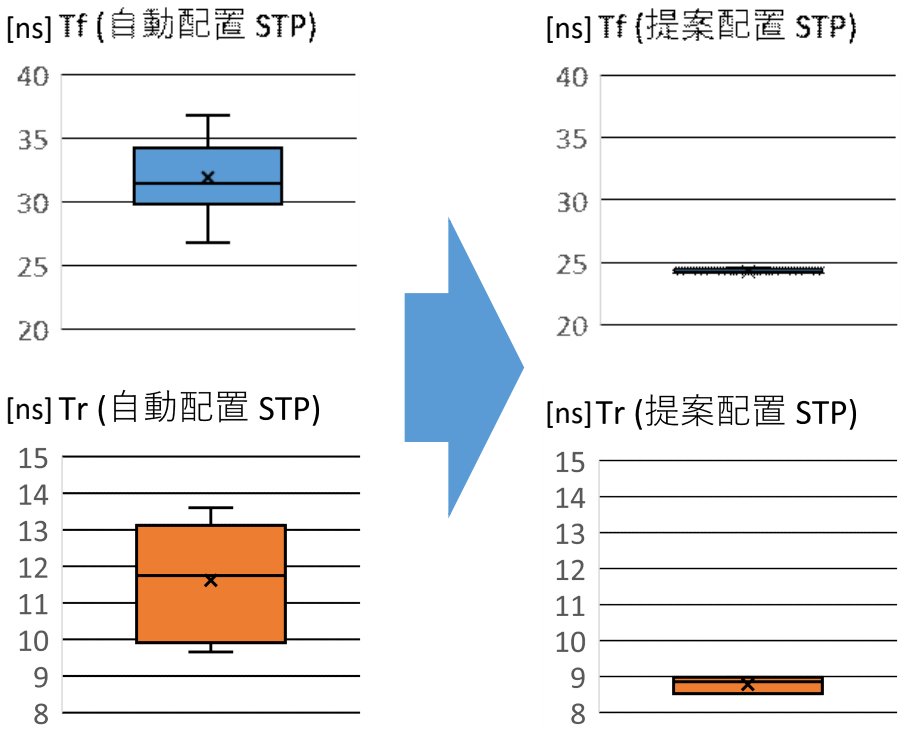


図 4.3　T_f/T_r の評価

4.8　結言

本章では，Xilinx 社製 FPGA チップを対象として，提案手法のセルフタイム型データ転送制御回路を設計し，ツール依存の従来配置手法と提案手法のそれぞれでタイミング情報の抽出を行った．評価指標は，回路内部の配線遅延時間である．提案手法が従来手法よりも 20%から 50%遅延時間を削減できたため，提案手法が性能向上に寄与していることを述べた．また，提案配置による T_f/T_r のばらつきについて 8 ステージ STP を対象としてタイミング検証を行った．結果，T_f/T_r のばらつきが低減できていることを述べた．今後の課題として，タイミング検証に対しての精度検証が不十分であること，セルフタイム型複合データ転送制御回路についてのタイミング検証法が必要であること，DDP 全体のタイミング検証が必要であることが挙げられる．

第 5 章

結論

5.1 まとめ

近年，IoT(Internet of Things) 技術の普及に伴い，IoT デバイスが増加の一途をたどっている．2023 年には世界中で約 340 億台まで IoT デバイスが増加すると予想されている．IoT デバイスの用途は多様化してきているが，IoT システム全体の通信帯域の圧迫や処理遅延が問題となっており，対策が要求されている．そこで，これらの要求を満たすエッジコンピューティング技術に注目が集まっている．

エッジコンピューティングではデータの保存と処理はネットワークのエッジで行われ，通常はデータソースに近い場所にあるデバイスで実行される．IoT データ処理をエッジコンピューティングを用いた実現とクラウドコンピューティングを用いた実現を比較した場合，エッジコンピューティングを用いる方がサーバへの負荷が小さく，通信遅延が削減できることから，エッジコンピューティングは IoT データ処理に向いている．また，エッジコンピューティングを行うエッジ端末は，電源を持たない現場で使用されることを想定し，超低消費電力で動作することが必要である．そのため，超低消費電力プロセッサの研究が進められている．超低消費電力プロセッサを実現するための技術として，セルフタイム型パイプライン (STP) を用いた非同期データ駆動型プロセッサ (DDP) が挙げられる．DDP は従来のクロック同期で動作するプロセッサと異なり，データの到着をトリガとして動作するため，データが到着していない時の消費電力を最小限にすることができる．また，一度設計したプロセッサの構成の変更や用途に応じた機能の追加など広く応用に適した回路を提供するため，ハードウェアの設計柔軟性が求められている．そこで，FPGA(Field Gate Programmable

Array) を用いたリコンフィギュラブルな実装が進められている．よって，STP によって実現された DDP の FPGA 実装が急務である．非同期回路である STP 回路は同期回路と比較して実装時のタイミング検証の難易度が高いという課題がある．特に STP 回路にはセットアップ・ホールド制約の充足とグリッチ回避制約の充足という 2 つの動作保証要件があり，これらを満たすためのタイミング検証法が求められていた．

従来手法では，タイミング検証を含めた設計フローを自動化する試みがある．しかし，設計フローの配置配線工程をツールに依存していたため，タイミング制約の充足のための遅延回路の負荷後，論理合成と配置配線の結果が変化するため，静的タイミング解析の結果が再利用できないという問題があった．

そこで，本研究では，FPGA 構造に適した回路合成を行い，FPGA 構造に適した配置配線を行うことで，STP の結果を部分的に再利用でき，制約の充足に必要な遅延量の見積もりが可能なタイミング検証を可能にするため，Xilinx 社製 FPGA の論理スライス構造の解析を行い，ペア配置とスライス間・CLB 間の配線遅延などの性能低下要因を発見し，性能低下を防ぐ回路構成を検討した．また，配線構造にも着目し，STP 回路の大域配線の占有を防ぐ配置法を検討した．最後に，回路の論理合成結果や配置配線結果を固定し，タイミング解析の結果を遅延回路付加部分のみ再利用することで検証可能なタイミング検証法を検討した．

そして，提案手法を用いて実装したデータ転送制御回路の従来配置と提案配置を比較し，データ転送制御回路単体で内部の配線遅延が 20%から 50%低減しており，性能の向上が確認できた．また，STP 回路を提案配置で実現した場合，性能指標である T_f/T_r のばらつきが低減した．これにより各ステージのデータ転送性能が一定になることで，性能が向上しているといえる．

5.2　今後の課題

今後の課題としては，スライス構造のより詳細な理解と適した配置法の検討が挙げられる．本研究の提案配置において，1 スライスに収めるために個別指定による配置を行っていたが，スライス内の LUT と FF/Latch は縦に 4 つ並列配置されているが，それぞれの LUT と FF/Latch の配置順序による遅延時間の違いは考慮された配置になっていない．そのため FF/Latch 間の配線遅延を全パターン調査し，最もばらつきが少なく，性能向上が見込める配置を検討する必要がある．

現在，精度検証のために，より詳細な配置による配線遅延時間の違いについて検討が行われている．表 5.1 では CLB 毎に間を空けて配置する場合と同クロックリージョン内で配置できる最大の幅である 5CLB 毎に間を空けて回路を配置する場合でそれぞれ遅延時間を抽出したが，提案手法と比較して遅延時間が延び，ばらつきについても大きくなっていた．

表 5.1　8 段環状 STP 配置の 1CLB 毎と 5CLB 毎（最大面積）で配置した場合の T_f/T_r [ns]

	CLB 毎に配置		5CLB 毎に配置	
	T_f	T_r	T_f	T_r
c1	24.488	9.246	26.525	9.038
c2	24.494	9.776	27.062	9.627
c3	24.494	9.776	26.612	10
c4	24.872	10.335	27.208	10.215
c5	25.157	9.712	28.613	10.65
c6	25.153	9.224	28.665	10.213
c7	25.157	9.224	28.665	10.267
c8	25.264	9.224	27.327	10.184

一方で，遅延回路を過密に配線し，配線遅延の増加によって遅延回路の個数を減らす手法についても検討している．表 5.2 の Ds，Da を同 LUT に配置した 8 段 STP は，T_f と T_r

で配線遅延の増加に違いがみられた．また，ばらつきも提案手法と比較して増加した．これは，過密配線によるスイッチマトリックスの変化が影響している．

また，C 素子，遅延回路をそれぞれ点対称に配置する．FPGA 上の大域配線を 2 分割し，それぞれに N/2 ステージ配置する場合の左側と右側で，C 素子と遅延回路の配置を上下反転させた．結果として，一部の配線遅延が減少したが，配線遅延全体のばらつきが増加した．

表 5.2　8 段環状 STP 配置の過密配線時（最小面積）と点対称配置時の T_f/T_r [ns]

	Ds，Da を同 LUT に配置		点対称に配置	
	T_f	T_r	T_f	T_r
c1	36.353	11.135	24.376	8.97
c2	36.485	11.196	24.378	9.572
c3	36.363	11.318	24.376	9.572
c4	34.348	11.931	24.488	10.131
c5	32.713	11.976	22.587	9.473
c6	32.148	14.241	22.587	8.915
c7	32.345	13.741	22.585	8.915
c8	34.638	13.811	22.808	8.983

次の課題としては，タイミング検証法の精度検証が挙げられる．本研究で提案したタイミング検証法は，回路構成変更箇所以外のタイミング情報が変化しない前提に成り立っている．しかし，現実的には，一部の回路構成を変更した結果，周辺回路のスイッチマトリックスの配線が変わってしまうことは容易に考えられる．そこで，本タイミング検証法を DDP などに適用したのちに，遅延素子挿入個数の変更などで，周辺回路のタイミング情報がどの程度変化するか調査し，配線遅延の変化が許容範囲外の場合に新たなタイミング検証法を検討する必要がある．

さらに，複数データ転送制御回路を含む STP のタイミング検証法の確立が挙げられる．本提案手法は，基本的なデータ転送制御回路のみを環状接続した STP についてのタイミン

グ検証法であるが，複合データ転送制御回路の拡張回路の拡張制約を含めたタイミング検証が出来ていない．そのため，拡張回路の詳細な配置を決定し，先行研究によって導出されている複合データ転送制御回路のタイミング制約を用いたタイミング検証法を確立する必要がある [10]．現在，各複合データ転送制御回路の回路構成法について，検討が終了している．図 5.1 は CB 回路と CE 回路の構成法である．どちらも C 素子の外に 1 つの DL を持ち，Send 信号の出力先が拡張されている．そこで，拡張回路部分を 2 スライスで実現することで，最小構成回路となる．

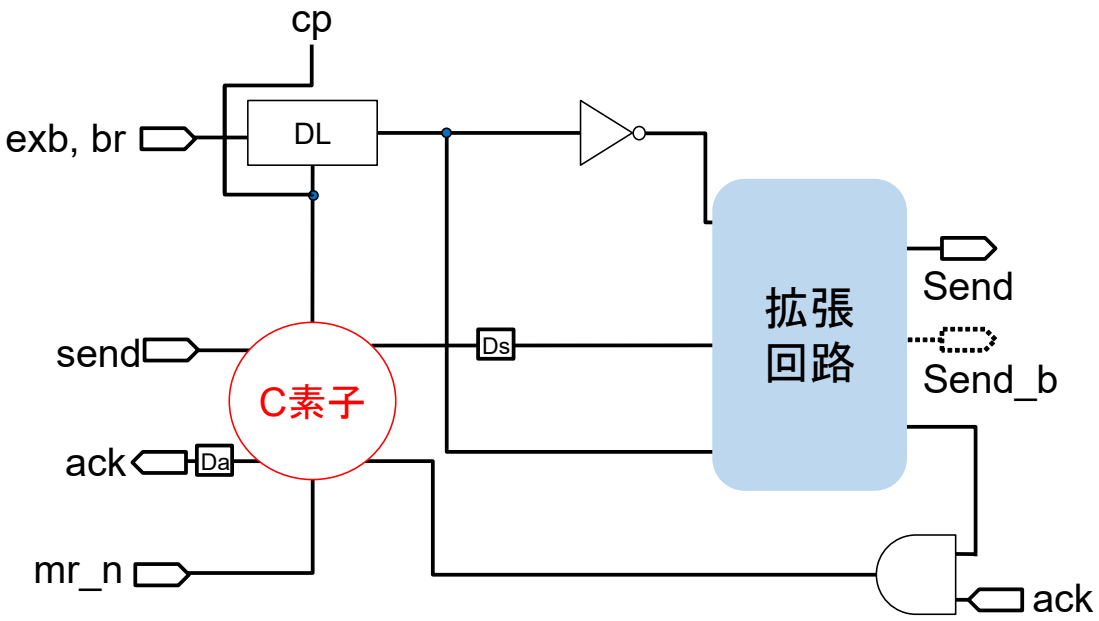


図 5.1　CB，CE 回路構成

図 5.2 の配置は，拡張回路部分と ARB をそれぞれ 2 スライスで実現し，NOT 回路を他 LUT に統合することで，最小構成となる．

図 5.3 の配置は，CX2 内の基本 C 素子で挟まれた拡張回路部分とそれ以外の拡張回路部分をそれぞれ 1 スライスで実現し，NOT 回路を他 LUT に統合することで，最小構成となる．

これらの複合データ転送制御回路の配置方針を図 5.4，図 5.5，図 5.6 に示す．

CB，CE は図のように拡張回路部分+基本 C 素子外部の DL を 1 個の CLB に配置し，

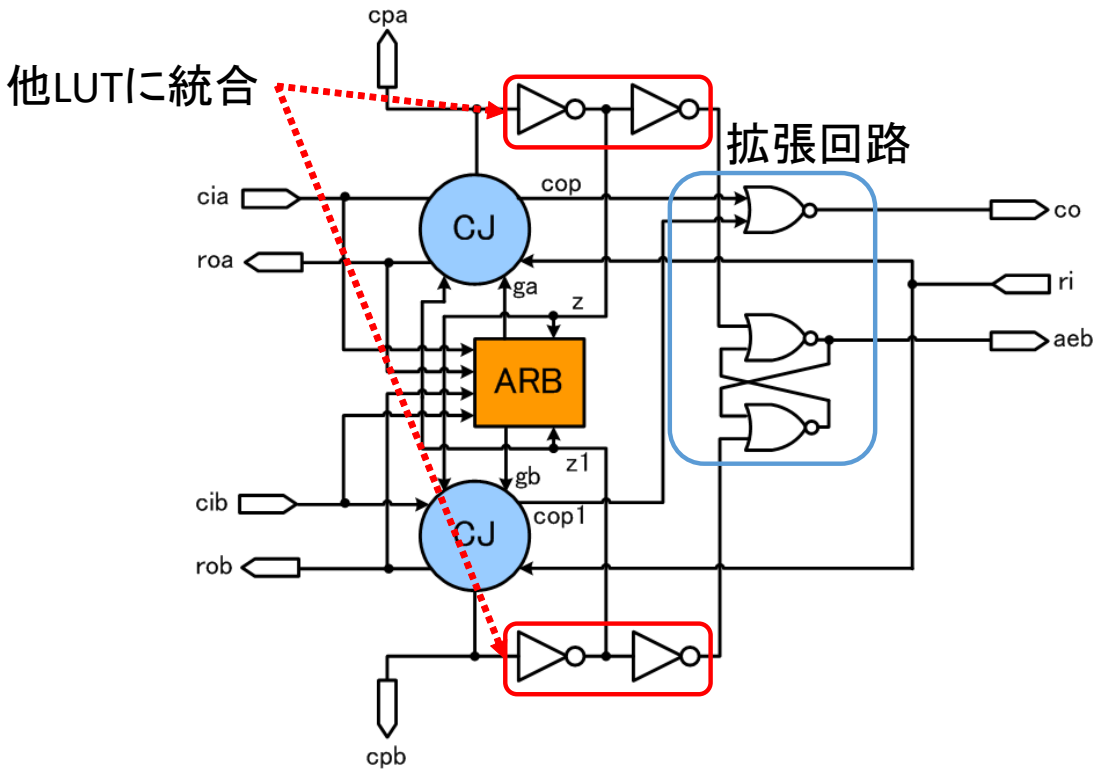


図 5.2　CM 回路構成

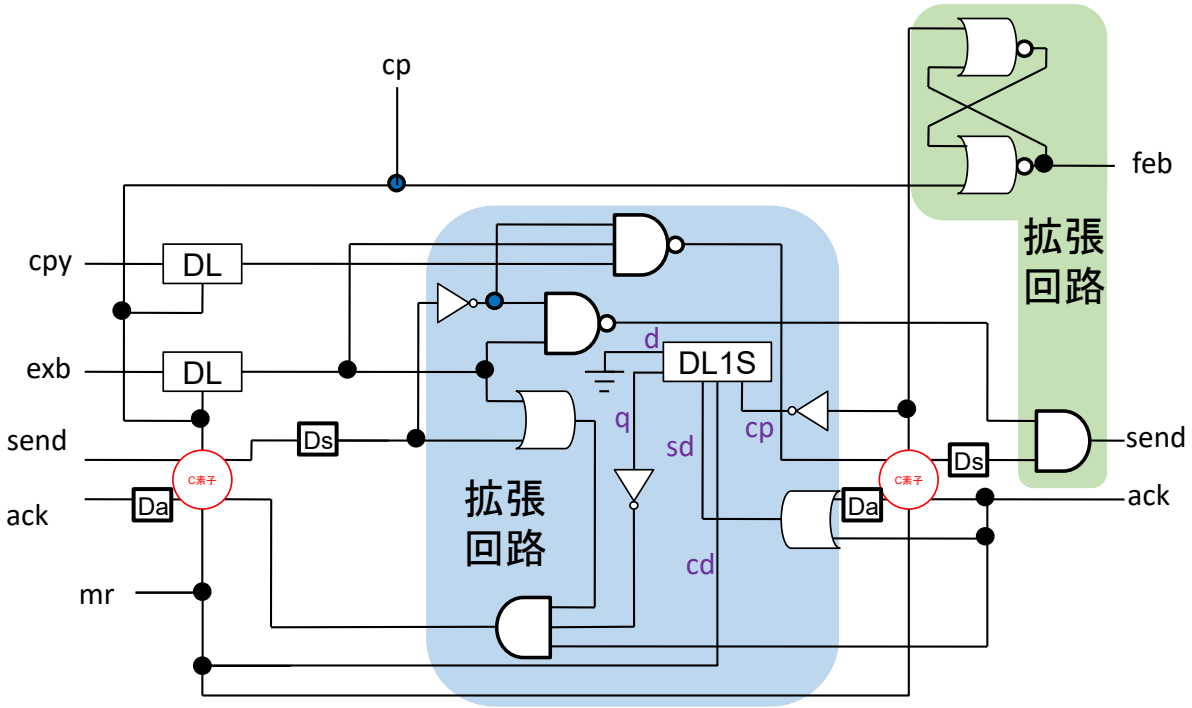


図 5.3　CX2 回路構成

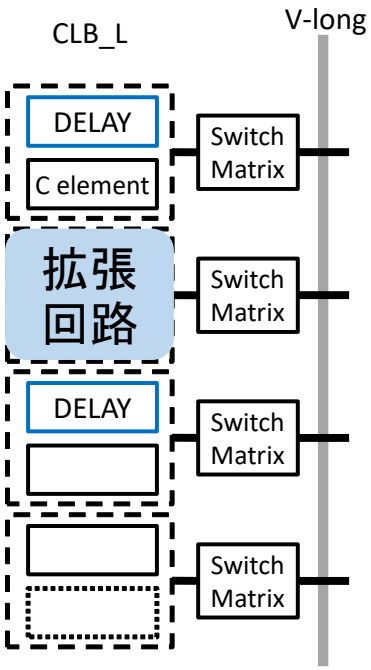


図 5.4　CB，CE 回路配置

データフローに沿って配置する．

CX2，CM についても同様に，データフローに沿って拡張回路部分を CLB に配置することで，回路構成や配置配線による遅延の影響が少ない回路が実現でき，精密なタイミング検証が可能となる．

次の課題として，DDP 全体のタイミング検証が挙げられる．本研究では研究日程の都合上，DDP 全体を通してタイミング検証し，本提案手法の実用性を確かめることができなかった．そのため，DDP のタイミング制約を Xilinx 社製 FPGA を用いて検証し，実際の回路実装で通用する手法かどうか評価することが残されている．DDP の FPGA 実装時の配置については，本研究の提案の骨子であるペア配置の活用と，大域配線の分割による過密配線の防止を踏まえて，図 5.7 のように配置する．内部の C，CE，CB，CM，CX2 については，提案手法のように STP として配置し，その外周のスライスに DDP の各ステージのファンクションロジックや DL を配置する．これにより，DDP のデータフローに沿った配置が可能となり，性能に与える配線遅延の影響を抑えることができる．

最後に，これらの回路構成法やタイミング検証法は従来手法とファイル形式や使用する

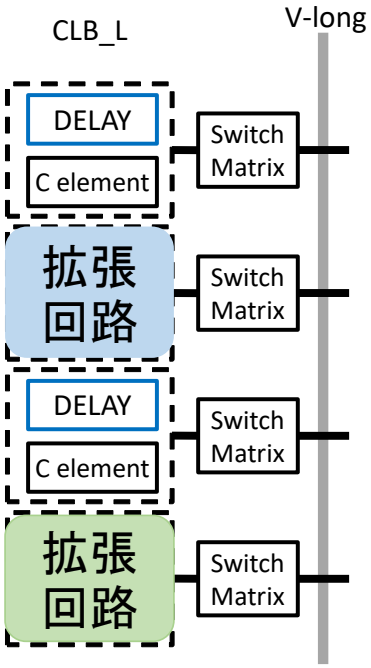


図 5.5　CX2 回路配置

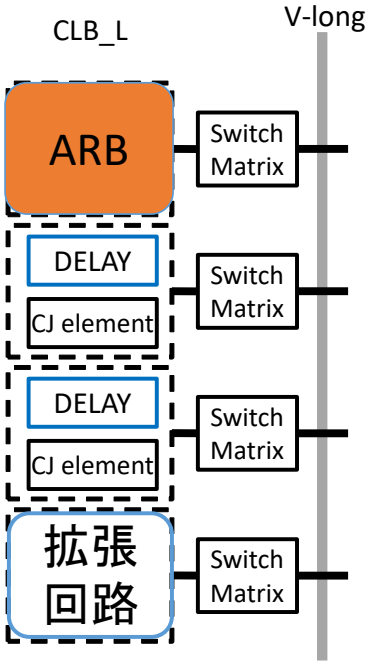


図 5.6　CM 回路配置

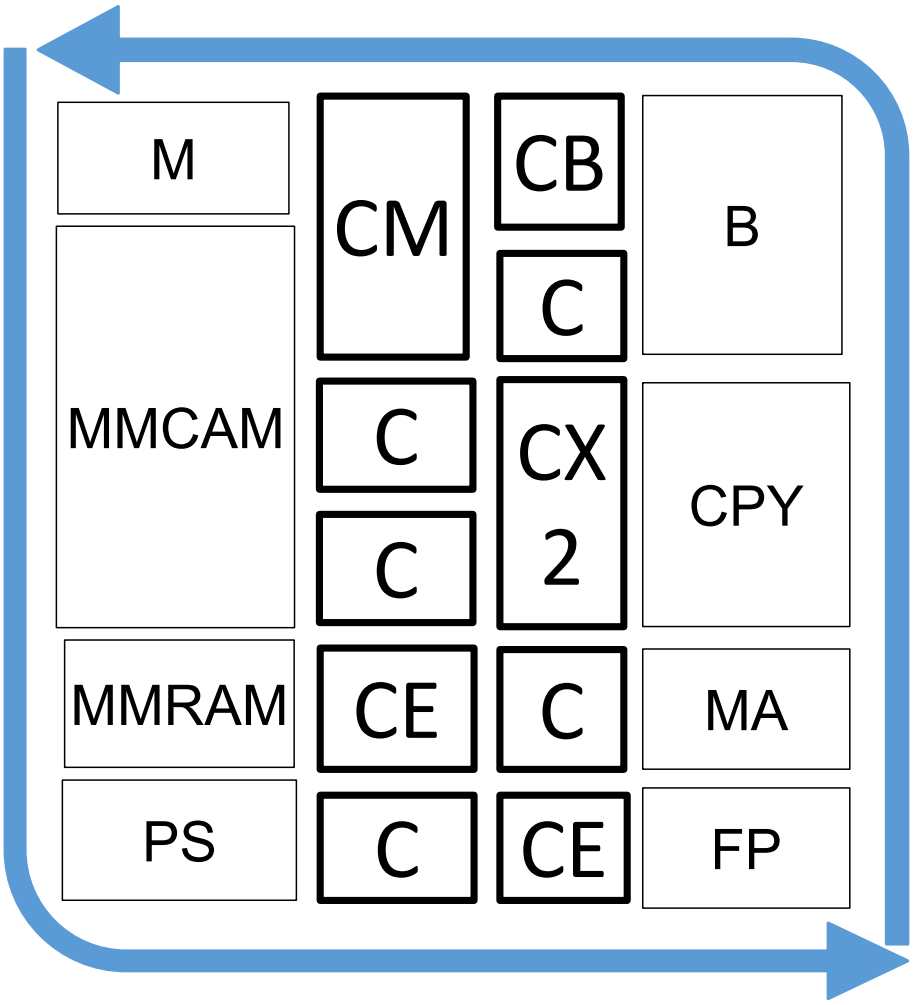


図 5.7　DDP 内データ転送制御回路のタイミング検証時配置

FPGA 回路設計ツールなどが異なる．そのため，従来手法で提案されている回路設計及びタイミング検証の自動化手法を検討する必要がある．

謝辞

本研究を進めるにあたって，親身にご指導いただきました岩田誠教授に深く感謝申し上げます．御多忙な中で研究の進め方や研究者・技術者としての考え方など，多くのことを教えていただき，本研究を進めることができました．学部生 3 年次の研究室配属から今日に至るまで，研究のみならず，技術者としての振る舞いを教えていただき，また就職活動からコロナ禍のオンライン留学への挑戦など様々なことで背中を押してくださるなど，大変お世話になりました．ここに感謝の意を表します.

御多忙な中，本研究の副査を務めてくださり，研究の改善点や疑問点を丁寧にご指摘くださった栗原徹教授，松崎公紀教授に深く感謝申し上げます.

博士課程の学生として様々なご指導をいただきました張震氏，本研究の技術支援を頂きました Tamnuwat Valeeprakhon 氏に深く感謝申し上げます。

また，研究室に配属されてから同期として，共に 4 年間切磋琢磨させていただきました，岡野秀平氏，筧拓也氏に心より感謝いたします。

研究室の後輩として，日頃から研究活動のご支援とご協力をいただきました，修士 1 年の高橋龍一氏，学部 4 年の植本陸氏，松坂拓海氏，坂口白磨氏，計屋和希氏，短い期間でしたが，共に研究に邁進した学部 3 年の石橋璃貴氏，市ノ木一希氏，伊藤雅俊氏，岡村健勝氏，椎葉啓介氏，山下拓巳氏に心より感謝申し上げます。

最後になりますが，産まれてから今日に至るまで経済的にご支援くださった両親をはじめ，精神的にサポートを頂きました友人である芳我信乃介氏，木原直哉氏，日頃からご支援いただきました関係者の皆様に心より御礼申し上げます.

参考文献

[1] 総務省 ”令和 3 年度情報通信白書 補論 デジタル経済の進展と ICT 経済の動向,” https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r03/pdf/n0100000_hc.pdf, 2022 年 1 月 6 日参照

[2] W. Yu et al., ”A Survey on the Edge Computing for the Internet of Things,” in IEEE Access, vol. 6, pp. 6900-6919, 2018, doi: 10.1109/ACCESS.2017.2778504.

[3] Davis, Al, and Steven M. Nowick. ”An introduction to asynchronous circuit design.” The Encyclopedia of Computer Science and Technology 38 (1997): 1-58.

[4] G. Gimenez, A. Cherkaoui, G. Cogniard and L. Fesquet, ”Static Timing Analysis of Asynchronous Bundled-Data Circuits,” 2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2018, pp. 110-118, doi: 10.1109/ASYNC.2018.00036.

[5] 長野寛司, ”FPGA を対象としたデータ駆動型プロセッサの設計自動化フローの検討,” 修士学位論文, Feb. 2021.

[6] 天野英晴編, ”FPGA の原理と構成,” オーム社, pp.35-37, Apr. 2016.

[7] 吉川千里, 三宮秀次, 岩田誠, 佐藤聡, 西川博昭,”FPGA 向き自己同期型パイプライン回路構成法,” 情報処理学会研究報告システム・アーキテクチャ（ARC）,2021-ARC-243(25), pp.1-7, Jan. 2021.

[8] S. Yoshikawa, S. Sannomiya, M. Iwata and H. Nishikawa, ”Pipeline Stage Level Simulation Method for Self-Timed Data-Driven Processor on FPGA,” 2020 8th International Electrical Engineering Congress (iEECON), 2020, pp. 1-5, doi: 10.1109/iEECON48109.2020.229515.

[9] 井上聡, ”データ駆動型プロセッサの FPGA 向き回路最適化手法の検討～データ転送制御回路に着目した最適化～”, ”修士学位論文”, Feb. 2022.

[10] 尾ノ井嶺卓, ”セルフタイム型複合データ転送制御回路の FPGA 実装用タイミング検証法,” 学士学位論文, Feb. 2021.

付録 A

提案回路構成の回路記述（PSR のみ）

```
(* dont_touch = "true" *)
module C_Slice (G,
                MR,
                SENDIN,
                ACKIN,
                SENDOUT,
                ACKOUT,
                CP,
                LOPEN
                );

input wire  G;
input wire  MR;
input wire  SENDIN;
input wire  ACKIN;
input wire  LOPEN;
```

```
output wire SENDOUT;
output wire ACKOUT;
output wire CP;
wire w1, w2;

//PSR means preceded SR-FF
(* dont_touch = "true" *) PSR p1(.l(LOPEN), .s(SENDIN),
.r(w2), .mr_n(MR), .q(w1));
assign ACKOUT = ~w1;
assign CP = SENDOUT;
endmodule

module PSR(
    l,
    s,
    r,
    mr_n,
    q
);

input  wire l;
input  wire s;
input  wire r;
input  wire mr_n;
```

```
output wire q;
wire O;

LUT4 #(
    .INIT(16'h8a08)  // Specify LUT Contents
) LUT4_inst (
    .O(O),   // LUT general output
    .I0(mr_n), // LUT input
    .I1(r), // LUT input
    .I2(s), // LUT input
    .I3(O)  // LUT input
);

LDCE #(
    .INIT(1'b0) // Initial value of latch (1'b0 or 1'b1)
) LDCE_inst (
    .Q(q),      // Data output
    .CLR(1'b0),  // Asynchronous clear/reset input
    .D(O),      // Data input
    .G(l),      // Gate input
    .GE(1'b1)     // Gate enable input
);
endmodule
```