レイテンシ削減を目的とした
フロアプラン指向FPGA向け高位合成手法
に関する研究

Floorplan-driven High-level Synthesis Algorithms for Latency Reduction Targeting FPGA Designs

2019年2月

藤原　晃一

Koichi FUJIWARA

レイテンシ削減を目的とした
フロアプラン指向FPGA向け高位合成手法
に関する研究

Floorplan-driven High-level Synthesis Algorithms for Latency Reduction Targeting FPGA Designs

2019年2月

早稲田大学大学院 基幹理工学研究科
情報理工・情報通信専攻 情報システム設計研究

藤原　晃一
Koichi FUJIWARA

目 次

第1章　序論

1.1　本論文の背景と意義

本節では，本論文での研究背景と本論文の提案内容を述べる．研究背景として，まずFPGA（Field Programmable Gate Array）を対象とした高位合成の必要性を述べる．次に，既存のFPGA向け高位合成手法の研究動向とそれらの課題を整理する．そして，上記で述べた課題に対して，本論文での提案内容を明示する．

1.1.1　FPGAを対象とした高位合成の必要性

本項では，FPGAを対象とした高位合成の必要性を述べる．まずLSI（Large-Scale Integrated circuit）の近年の動向と開発の課題を述べる．次に，開発の課題の解決にアプローチする技術として，FPGAと高位合成についてまとめる．

今日の情報化社会においてLSIやSoC (System-on-a-chip) は，携帯電話などの情報通信機器，PC（Personal Computer），自動車や家電など，人々の身の回りのあらゆるデバイス・場面で利用されており，我々の生活を支える基盤技術の1つである．更には，IoT (Internet of Things) という言葉にも代表されるように，これらは今後より広く深く我々の周囲に浸透していくことが予想され，情報化社会の発展を支える必要不可欠な要素であることは疑いようがない．

LSIの開発では,「生産性の向上（設計コストの削減，設計の高速化）」の要求が非常に高まっている．これは，近年の半導体の微細化技術の進歩によって，1つのチップで実現できるシステムは高性能化している一方で，チップの設計規模・複雑さも著しく増加していき，設計コストの爆発的な増加が深刻となっているためである．更に，社会では情報技術の発展が加速度的に進み，今日の目まぐるしく変化する技術革新・市場変化を捉え，企業がビジネスチャンスを得るためには，早急な新規サービスの市場投入，そのための設計生産性の向上が急務である．

近年，LSI開発の生産性を向上させる技術として,「FPGA」と「高位合成」が注目されている．FPGAとは，従来のASIC（Application Specific Integrated Circuit）と異なり，チップ製造後に構成を設定・変更できる集積回路である．FPGAをLSI基盤として採用することで，ハードウェアによる高速なデータ処理を実現すると同時に，回路の製造後の仕様変更の容易性を確保できる．実際に最近では，通信プロトコル処理 [6] や高精細テレビでの画像処理 [7, 67] などの大量のデータを高速処理する必要がある場面や証券取引 [20] や，SDR（Software Defined Radio）[59] など短期間でアルゴリズムの変更が求められるアプリケーションに対し利用が急速に拡大している．また．宇宙空間で利用されるLSI [70] やDeep Learning

など人工知能への適用 [5] にも注目が集まっており，FPGA の需要は急速に高まっている．

一方，高位合成とは，LSI 自動設計技術の 1 つであり，C 言語など抽象度の高い言語で書かれた対象アプリケーションの直接的な動作の記述（動作レベル記述）から，Verilog などハードウェア記述言語を用いてデータパス（Datapath）とコントローラ（有限状態機械: FSM (Finite State Machine)）のネットリストを表現したレジスタ転送レベル記述（RT（Resister Transfer）レベル記述）を自動的に合成する技術である．LSI 設計に高位合成を用いることで，論理合成よりさらに抽象度の高いレベルから自動設計が実現できるため，従来の RT レベル記述による設計に比べて，記述量の削減・設計誤りの削減に伴う設計の高速化を実現できる．また，LSI 設計に高位合成を利用することでハードウェア特有の知識を抽象化でき，ソフトウェア開発者によるハードウェア設計が可能になる．これによって，人材確保の容易化など開発現場にはメリットも多い．

以上より，生産性を向上させるアプローチとして，LSI 開発に「FPGA」と「高位合成」を利用する場面・必要性は増大している [8,18,37,54,64]．また，現在では商用・研究用のツールも多数存在する [4,9,44,55,72]．しかし，一般的に高位合成によって生成される回路は RT レベル記述を手設計した場合に比べて，性能面に課題が残ってしまう．IoT，AI（Artificial Intelligence）など最新の情報技術で，LSI に要求される処理性能は更なる高まりを見せているため，FPGA 設計に高位合成を利用した上での，処理性能の確保は大きな課題である．

1.1.2 既存の研究動向と課題

本項では，既存の研究動向と課題として，既存の FPGA 向け高位合成の研究動向を整理すると共に，本論文で解決する課題を明らかにする．

FPGA 設計に高位合成を用いて処理性能の高い回路を生成するには，FPGA の持つ特有のアーキテクチャを考慮した高位合成アルゴリズムを利用することが不可欠となる．このため，FPGA を対象とした高位合成手法が盛んに研究されている [10–15,17,19,38–40,42,48–51,56,61–63,66,71,76–78]．

回路の処理性能を示す指標に回路の遅延 (レイテンシ) があり，より処理性能の高い回路を FPGA 上に実現するためには，高位合成によってレイテンシの小さい回路を合成することが重要である．レイテンシとは，その回路が実現するアプリケーションの処理 1 回あたりに要する回路のシステム実行時間を表し，クロック周期とシステム実行に要するクロック数の積で求まる．回路のレイテンシが小さいほど，単位時間当たりの処理数が増え，回路の処理性能が高くなる．一般的に，CPU（Central Processing Unit）によるソフトウェア処理では処理性能の要求が満たされない場面で，FPGA によるハードウェア処理が用いられるため，そこでは FPGA 上に実現される回路に対し，必然的に高い処理性能が要求される．それに伴って，FPGA を対象とした高位合成では，レイテンシの小さい回路を合成することが必要不可欠である．

本論文では，FPGA を対象とした高位合成における以下の 3 つの課題を取り上げる（図 1.1）．1 つ目の課題（課題 1）は「**フロアプランの考慮と MUX のコスト削減の同時実現**」である．FPGA を対象とした高位合成では，モジュールの配置（フロアプラン）を考慮の必要性と，マルチプレクサ（Multiplexer (MUX)）のボトルネックが問題となっており，これらは相互依存の関係が強く同時解決が望まれる．2 つ目の課題（課題 2）は「**高位合成段階に**

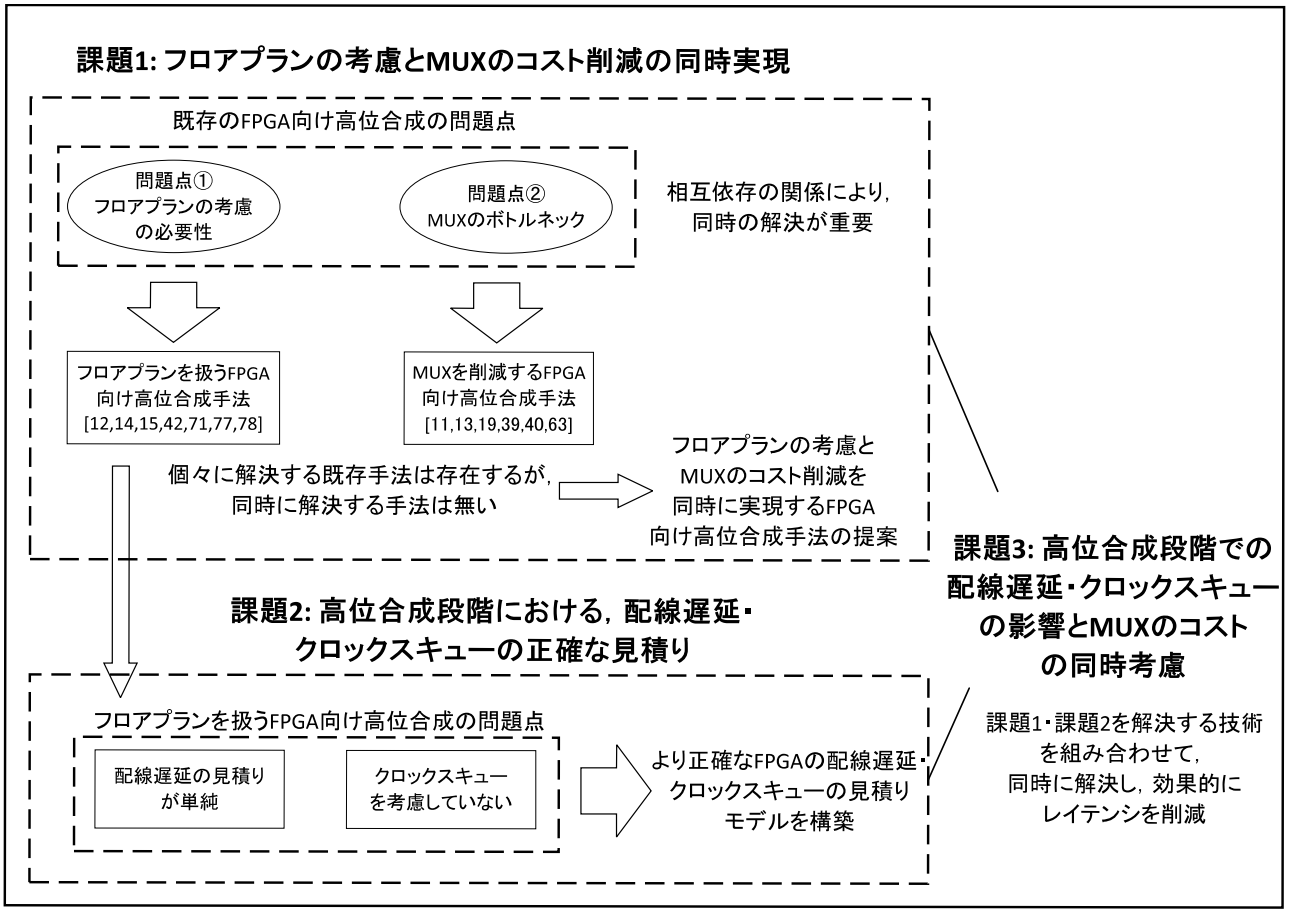


図 1.1: レイテンシ削減を目的とした FPGA 向け高位合成の課題.

おける，配線遅延・クロックスキューの正確な見積り」である．高位合成段階でフロアプランを扱う手法では，よりレイテンシの小さい回路を生成するため，FPGA 上の配線遅延・クロックスキューの正確な見積りの精度が問題となっている．3 つ目の課題（課題 3）は「**高位合成段階での配線遅延・クロックスキューの影響と MUX のコストの同時考慮**」である．上記で挙げた 2 つの課題は，両方共に FPGA を対象とした高位合成でレイテンシの小さい回路を生成する上で重要な課題であり，これらを同時に解決することは有意義である．

課題 1 (フロアプランの考慮と MUX のコスト削減の同時実現)

課題 1 の「フロアプランの考慮と MUX のコスト削減の同時実現」を議論するにあたり，既存の FPGA を対象とした高位合成における問題点を整理する．FPGA を対象とした高位合成では，レイテンシの小さい回路を生成する上で，以下の 2 つの問題点があり，これらを解決することが重要となる．

問題点 1 モジュールの配置（フロアプラン）の考慮の必要性

問題点 2 マルチプレクサ（MUX）のボトルネック

一方で，[10, 17, 38, 49, 56, 61, 62, 66, 76] では，レイテンシの削減へアプローチする技術を持った手法が提案されているが，これらは問題点 1, 2 に対して解決を図る手法ではない．これらの手法では，チェーニング技術，対象アプリケーション向けの特化，ビット幅の考慮など，他の LSI 向け高位合成で用いられていた技術を FPGA 向け高位合成に流用し，レイテンシの削減を図っている．そして，これらの技術によるアプローチは，上記の問題点 1・問題点 2 との関係性が無く，上記の 2 つの問題点を解決する技術との併用が可能と考えられる．従って，本論文ではこれらの手法で解決する問題点およびそのアプローチはスコープ外とする．

問題点 1 に関して，近年の FPGA 設計では LSI プロセスの微細化に伴って，配線部分の伝搬遅延による「モジュール間の**配線遅延**[1]」と「**クロックスキュー**」が回路のクリティカルパス遅延に対して与える影響が増大しており，[42] や [12] で議論されているように，無視できない影響度となっている．FPGA を対象とした高位合成でレイテンシの小さい回路を生成するためには，配線遅延・クロックスキューの影響を含めたタイミング設計が重要となる．回路内での配線遅延・クロックスキューは，FPGA 上の回路の「モジュール配置（フロアプラン）」に依存するが，本来，回路のフロアプランは高位合成以降の FPGA 設計の段階で決定される．従って，配線遅延・クロックスキューの影響を高位合成段階で見積り，これらを含めたタイミング設計を行うためには，FPGA 上のフロアプランを高位合成段階で扱う技術が必要である．問題点 1 の解決へアプローチする，既存のフロアプランを扱う FPGA 向け高位合成手法として，[12, 14, 15, 42, 71, 77, 78] がある．

一方で，問題点 2 に関して，FPGA では [11] で述べられている通り，遅延・面積において MUX のコストが大きく，これのコストを高位合成段階で削減する必要がある．FPGA 設計では，回路のデータパス内の MUX の数・入力数は，高位合成段階において決定される．もし，高位合成段階で MUX のコストを無視したリソース共有によってデータパスを生成した場合，最終的な回路のレイテンシ・面積は大きくなってしまう可能性がある．従って，高位合成段階で MUX のコストを考慮し削減したデータパスを生成することが，レイテンシの小さい回路を得るために重要である．問題点 2 の解決へアプローチする，既存の MUX を削減する FPGA 向け高位合成手法として，[11, 13, 19, 39, 40, 63] がある．

問題点 1・問題点 2 は相互依存の関係が強く，レイテンシの小さな回路生成を目的とする FPGA 向け高位合成において，同時に解決されるべき重要な課題である．FPGA 設計において，データパスの設計とフロアプランには相互依存の関係がある．問題点 1 を解決するフロアプランを扱う FPGA 向け高位合成手法では，高位合成段階で相互依存であるデータパスの設計と FPGA 上のフロアプラン設計を交互に反復して行い，最適解の探索を行う．この時，フロアプランを扱う FPGA 向け高位合成手法のデータパス設計において，問題点 2 の MUX のボトルネックが解決されていない場合，高位合成段階で MUX のコストが大きいデータパスに応じたフロアプラン解しか求められない．その時のレイテンシの削減は，問題点 1 と問題点 2 を同時に解決した場合に比べて，半分未満となることが想像できる．また，問題点 2 のみを解決した手法の場合，下位工程で決定したフロアプラン解に応じたデータパスの改良を行えないため，同様にそのレイテンシの削減の効果が低くなることが予想される．従って，上記の問題点 1・問題点 2 は，レイテンシの小さな回路生成を目的とする FPGA 向け高位合成において，同時に解決を図る必要性が高いと言える．

[1] 本論文では，以降でモジュール間の配線遅延を「配線遅延」と呼ぶ．

上記で述べた通り，問題点 1・問題点 2 それぞれにアプローチする FPGA 向け高位合成手法は存在するが，これらを同時に解決する高位合成手法は我々の知る限り未だ実現されていない．また，既存手法ではそれぞれが独自の目的関数および独自のバインディングアルゴリズムにより目的を達成しており，一概に組み合わるのは難しく，問題点 1・問題点 2 を解決するためには，既存の FPGA 向け高位合成手法とは異なるアプローチが必要となる．

従って，本論文では高位合成段階で回路の FPGA 上でのフロアプランを扱い，MUX のコストを削減したデータパス設計を実現する,「フロアプランの考慮と MUX のコスト削減の同時実現」を 1 つ目の課題に設定する．

課題 2 (高位合成段階における，配線遅延・クロックスキューの正確な見積り)

ここで，上記の課題 2 の「高位合成段階における，配線遅延・クロックスキューの正確な見積り」を議論する．上記での述べた通り，FPGA 向け高位合成では，フロアプランを扱い配線遅延・クロックスキューの影響を正確に見積り，これらを含めたタイミング設計を行うことがレイテンシの小さい回路を生成するために重要となり，これを実現する見積りモデルの構築が必要となる．

フロアプランを扱う FPGA 向け高位合成手法 [12,14,15,42,71,77,78] は,（1）詳細な FPGA アーキテクチャに基づく個別のモジュール配置を扱う手法 [71,77,78]，と（2）抽象化されたモジュールの FPGA 配置を扱う手法 [12,14,15,42] に分かれる.（1）に分類される手法 [71,77,78] は，より正確な配線遅延の見積りが可能になる反面，FPGA には I/O パッドや特殊セル（DSP ブロックや RAM）などを含めた詳細な FPGA アーキテクチャに基づく個別のモジュール配置を扱うことは，高位合成問題を極めて複雑にしてしまう．一方，フロアプランを含めた高位合成問題をシンプルに解くための技術として，(2) に分類される手法 [12,14,15,42] が提案されているが，配線遅延・クロックスキューの見積りに課題が残る．これらの手法は,「フロアプラン指向 FPGA 高位合成手法」と呼ばれ，各々独自の「抽象化されたモジュール」単位で FPGA 上の配置を扱う．この時，FPGA アーキテクチャを抽象化したモデル上での配置情報を想定することで，高位合成問題を簡潔に解くことができる．

フロアプラン指向 FPGA 高位合成手法において，配線遅延・クロックスキューを正確に見積り，これらを含めたタイミング設計を行うためには,「フロアプラン指向 FPGA 高位合成手法向けの配線遅延・クロックスキュー見積りモデル」が必要となる．近年では，Xilinx 社や Altera 社などから，多くの種類の FPGA が提供されている．これらの FPGA はそれぞれの特有のアーキテクチャを持ち，配線遅延・クロックスキューの特性も各々であることから，これらを捉えたフロアプラン指向 FPGA 高位合成手法向けの見積りモデルの構築が必要である．具体的には，FPGA 上の配線遅延見積りにおいて，FPGA 毎に配線遅延特性を持つため，正確に配線遅延を見積るためには，これを考慮することが重要である．また，FPGA 上のクロックスキューの見積りに関しては，FPGA では回路が実装される前にクロック配線が予め構築されており [74]，モジュール配置によってはクロックスキューの影響が深刻となる．従って，FPGA では他の LSI 基盤とは異なり，FPGA のクロック構造を捉え，クロックスキューを見積るモデルが必要である．

しかし，既存のフロアプラン指向 FPGA 高位合成手法において，[14,15] の手法は配線遅延もクロックスキューも見積っていない．また，[42] の手法と [12] の手法では，独自の抽象

化したモジュールの配置から配線遅延を見積っているが，単純な配線遅延の見積りモデルを採用しており，その妥当性は検証されていない．また，これらの手法は，クロックスキューを考慮しておらず，それに伴いクロックスキューの見積りモデルも提案されていない．[65] では下位工程でのクロックスキュー見積りモデルが提案されているが，そのモデルは LSI 基盤の詳細なアーキテクチャ情報に依存するため，抽象化した FPGA チップ上のモジュール配置を扱うフロアプラン指向高位合成に適用するのは難しい．以上より，フロアプラン指向 FPGA 高位合成で正確に配線遅延・クロックスキューを見積るモデルは未だ提案されていない．

一方で，既存の FPGA 開発ツール内の見積りモデルの応用が考えられる．実際に，Xilinx 社の Vivado など既存の FPGA 開発ツールには STA（Static Timing Analysis）ツールが使用されている．これで使用されている独自の見積りモデルは FPGA の配線遅延・クロックスキューを高精度で見積ることが可能だが，入力として FPGA チップ全体の詳細なアーキテクチャ情報およびそれに基づくフロアプラン情報が必要となるため，フロアプラン指向高位合成には使えない．

従って，FPGA を対象としたフロアプラン指向高位合成手法で正確に配線遅延・クロックスキューを見積るために，FPGA の種類による配線・クロック構造の違いを吸収し，抽象化した FPGA 上でのフロアプラン情報に対応した配線遅延・クロックスキューの見積りモデルを構築することが必要となる．以上より，本論文では「高位合成段階における，配線遅延・クロックスキューの正確な見積り」を 2 つ目の課題に設定する．

課題 3 (高位合成段階での配線遅延・クロックスキューの影響と MUX のコストの同時考慮)

課題 3 の「高位合成段階での配線遅延・クロックスキューの影響と MUX のコストの同時考慮」を議論する．レイテンシの小さい回路の生成を目的とする FPGA 向け高位合成では，課題 1・課題 2 を同時に解決することが重要となる．

上記で述べた課題 1，課題 2 を解決する技術を提案した場合を考える．課題 1 を解決する FPGA 向け高位合成手法は，収束性の観点から新たなフロアプラン指向 FPGA 高位合成手法により実現されることが予想される．そして，課題 2 も同時に解決しない時，高位合成段階での配線遅延・クロックスキューの見積り精度は悪く，レイテンシの削減効果は低下することが考えられる．一方，課題 2 を解決する見積りモデルは，それ単独では意味がなく，フロアプラン指向 FPGA 高位合成手法に適用することが必要となる．従って，上記の課題 1，課題 2 を単独で解決する技術を提案した上で，それらを組み合わせて，課題 1・課題 2 を同時に解決する FPGA 向け高位合成手法を提案することが非常に有意義である．

以上より，本論文では課題 1・課題 2 を同時に解決する「高位合成段階での配線遅延・クロックスキューの影響と MUX のコストの同時考慮」を 3 つ目の課題に設定する．

1.1.3 本論文の提案内容

本項では，前項までの議論を踏まえ，本論文内での提案内容について述べる．本論文の提案内容は主に以下の 4 つである．

提案 1 フロアプランの考慮と MUX のコスト削減を同時に実現する FPGA 向け高位合成手法

提案 2 フロアプラン指向 FPGA 高位合成向けの配線遅延・クロックスキューの見積りモデル

提案 3 配線遅延とクロックスキューを考慮したフロアプラン指向 FPGA 高位合成手法

提案 4 フロアプラン指向高位合成を用いた FPGA 実装フロー

本論文では，前項の課題 1 を解決する手法として,「**フロアプランの考慮と MUX のコスト削減を同時に実現する FPGA 向け高位合成手法**」を提案する．提案手法では，高位合成でフロアプランを扱う既存技術である HDR（Huddle-based Distributed-Register）アーキテクチャ[1–3] を FPGA 向け高位合成に適用し，高位合成段階でフロアプランを扱い，配線遅延を見積る．そして，FPGA 上での MUX の遅延・面積を調べ，そこから 2 つのバインディング手法を導き，MUX のコスト削減を図る．FU バインディングでは，配線遅延を考慮しつつ，既にデータパスがある 2 つの FU 同士を積極的に割り当てることで，FU に付加される MUX 数の削減を図る．レジスタバインディングでは，MUX の入力数を制限することで，レジスタに付加される MUX のコストを削減する．この 2 つのバインディング手法により，MUX のコストを削減し，遅延の小さいデータパスを生成する．

次に，前項の課題 2 を解決するために，フロアプラン指向 FPGA 高位合成のための配線遅延/クロックスキュー見積りモデル「**IDEF**（Interconnection-Delay Estimate model for Floorplan-driven high-level synthesis targeting FPGA designs）」/「**CSEF**（Clock-Skew Estimate model for Floorplan-driven high-level synthesis targeting FPGA designs）」を構築する．具体的には，FPGA 設計ツール内の STA ツールを用いた実験を行い，FPGA 上での配線遅延特性とクリックスキュー特性を明らかにする．その実験結果を用いて，IDEF と CSEF の定式化を行う．そして，STA ツールによる FPGA 上の配線遅延/クロックスキューの実測値と IDEF/CSEF の見積り値の比較を行い，見積り精度の評価を行う．

そして，前項の課題 3 を解決する FPGA 向け高位合成手法として,「**配線遅延とクロックスキューを考慮したクリティカルパス最適化 FPGA 高位合成手法**」を提案する．課題 2 に対して提案した IDEF・CSEF を，課題 1 に対して提案したフロアプランの考慮と MUX のコスト削減を同時に実現する FPGA 向け高位合成手法に適用し，改良した手法である．提案手法は，高位合成段階で各パスの配線遅延・クロックスキューをより正確に見積り，それらの影響と MUX の遅延を含むデータパスの遅延を見積り，高位合成段階でクリティカルパスの候補を特定する．そして，FU バインディング・フロアプランの 2 つのフェーズにおいて，集中的に最適化を図る．以上により，提案手法は生成する回路のレイテンシの削減を図る．

最後に，上記で提案したフロアプラン指向 FPGA 高位合成手法を実際に FPGA 上で評価を行うために,「**フロアプラン指向高位合成手法を用いた FPGA 実装フロー**」を確立する．新たに確立する FPGA 実装フローに従って，上記で提案したフロアプラン指向 FPGA 高位合成手法を用いて，FPGA 上にベンチマークアプリケーションを実装し，回路が正常に動作することを確認する．そして，既存手法を用いて実装した回路と処理性能を比較することで，実機レベルで提案手法の優位性を確認する．

1.2 本論文の概要

本節では，本論文の構成と各章での議論内容を述べる．

本論文では，FPGA 上にレイテンシの小さい回路を生成する高位合成を構築するため，配線遅延・クロックスキューの考慮と MUX コストの削減を同時に達成する FPGA 向け高位合成手法を提案する．

本論文は 7 章から構成される．以下に本論文の構成を示す.

第 2 章**「関連研究」**では，既存の FPGA 向け高位合成手法に関する研究のうち，フロアプランを扱う FPGA 向け高位合成手法と MUX コスト削減を図る FPGA 向け高位合成手法に分類されるいくつかの既存手法を紹介する．そして，他の LSI 向けに提案されたフロアプランを扱う高位合成に適したアーキテクチャを紹介する．

まず，フロアプランを扱う FPGA 向け高位合成手法では，(1) 詳細な FPGA アーキテクチャに基づく個別のモジュール配置を扱う手法として，[71,77,78] を紹介し，(2) 抽象化されたモジュールの FPGA 配置を扱う手法として，[12,14,15,42] を紹介する．次に，MUX コスト削減を図る FPGA 向け高位合成手法として，[11,13,19,39,40,63] を上げ，そのうちの例として，Chen らの手法 [11] と Hara らの手法 [39,40] を紹介する．その後，高位合成内でフロアプランを扱う既存技術の 1 つである HDR アーキテクチャ[1–3] を紹介する．

第 3 章**「HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法」**では，FPGA を対象とした高位合成における課題 1「フロアプランの考慮と MUX のコスト削減の同時実現」を解決するため，フロアプランの考慮と MUX のコスト削減を同時に実現する FPGA 向け高位合成手法として，HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法を提案する．具体的には，HDR アーキテクチャを対象とした FPGA 高位合成の問題を定式化し問題を定義する．そして，その問題の解決のため，HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法を提案・評価する．

まず，提案手法のアプローチを決定するために，FPGA 上の回路構成要素を実装・分析し，MUX が遅延・面積においてボトルネックであることを明らかにする．更に，MUX の入力数を様々に変化させて実装し，入力数に応じた MUX のコストの増加傾向を分析し，提案手法の方針を決定する．

その後，HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法を提案する．提案手法は，配線遅延と MUX のボトルネックを同時に考慮した新しい FPGA 向け高位合成手法である．提案手法は，HDR アーキテクチャを採用することで，高位合成段階でフロアプランを扱い，配線遅延の影響を見積る．また，高位合成段階で MUX のコストを削減するため，各 FU（演算器）・レジスタ間のデータ転送を考慮した 2 つのバインディング手法を提案する．

- パス考慮スケジューリング/FU バインディング
- パス考慮レジスタバインディング

パス考慮スケジューリング/FU バインディングでは，配線遅延を含めたデータ転送遅延を考慮しつつ，既にデータパスがある 2 つの FU 同士を積極的に連続する 2 つの演算ノードに割り当てることで MUX 数の削減を図る．パス考慮レジスタバインディングでは，FPGA 実装によって導いた MUX の入力数に伴うコスト増加傾向を基に，入出力先の FU・レジスタを考慮してレジスタに付加する MUX を 4 入力以下に制限する．これによって，回路全体でのレジスタに付随する MUX の遅延・総面積を削減する．最後に，計算機実験により提案手法の有効性を示す.

第 4 章**「フロアプラン指向 FPGA 高位合成向け配線遅延・クロックスキュー見積りモデル」**では，FPGA を対象とした高位合成における課題 2「高位合成段階における，配線遅延・クロックスキューの正確な見積り」を解決するため，フロアプラン指向 FPGA 高位合成のための配線遅延・クロックスキュー見積りモデル「IDEF」と「CSEF」を構築する．まず，FPGA 上における測定を行い，配線遅延/クロックスキュー特性を明らかにする．そして，それらの特性に基づいて定式化を行い，それぞれの見積りモデルを構築する．その後，実測値と比較を行い，各々のモデルの見積り精度を評価する．最後に，第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に両モデルを適用し，評価を行う．

FPGA では，パス遅延に占める配線遅延・クロックスキューの割合が大きい．そのため，フロアプラン指向高位合成では，フロアプランに基づく配線遅延・クロックスキュー見積り精度は，フロアプラン解・生成されるデータパスの質に影響を与えてしまうため，できるかぎり高精度に見積る必要がある．FPGA の配線遅延見積りモデル構築のため，まず，シンプルな測定実験用の回路を用いて，様々なパターンで FPGA の配線遅延を測定し，FPGA の配線遅延特性を明らかにする．そして，測定結果に基づいて FPGA の配線遅延見積りモデル「IDEF」を提案する．次に，FPGA のクロックスキュー見積りモデル構築のため，FPGA 上でのクロックスキューの影響を示した上で，様々なパターンで FPGA のクロックスキューを測定し，FPGA のクロックスキュー特性を明らかにする．そして，クロックスキュー見積もりモデル「CSEF」を提案する．それぞれの見積りモデルの精度は，商用の FPGA 設計ツール内タイミングモデルの結果と比較し確認する．最後に，第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に IDEF・CSEF を適用し，計算機実験により有効性を示す．

第 5 章**「配線遅延とクロックスキューを考慮したクリティカルパス最適化 FPGA 高位合成手法」**では，FPGA を対象とした高位合成における課題 3「高位合成段階での配線遅延・クロックスキューの影響と MUX のコストの同時考慮」を解決するため，第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に前章で提案した IDEF・CSEF を利用し，配線遅延とクロックスキューを考慮したクリティカルパス最適化 FPGA 高位合成手法を提案する．

提案手法は，前章で提案した IDEF・CSEF をフロアプラン指向高位合成全体で利用し，高位合成段階で配線遅延・クロックスキューの影響を含めたデータパスの遅延を見積り，全データパスのうちクリティカルパスおよびその候補となるパスを特定する．そして，データパス生成とフロアプラン両方において，これらを集中的に最適化し回路のレイテンシ—の向上を図る．

データパス生成とフロアプランでクリティカルパス遅延を削減するために，以下のバインディング手法とフロアプラン手法を提案する．

- クリティカルパス指向スケジューリング/FU バインディング
- クリティカルパス指向ハドル合成/フロアプラン

クリティカルパス指向スケジューリング/FU バインディングでは，フロアプラン情報を基に FU バインディングを改良することで，配線遅延とクロックスキューを改善しクリティカルパス遅延の小さいデータパスの生成を図る．また，クリティカルパス指向ハドル合成/フロアプランではフロアプランの改善において配線遅延とクロックスキューを考慮して，クリティカルパスの候補となるパスを集中的に最適化を行う．そして，クリティカルパス遅延の小さいフロアプラン解を目指す．最後に，計算機実験により提案手法の有効性を示す．

第 6 章**「提案手法を用いた FPGA 実装評価」**では，レジスタ分散型アーキテクチャを対象としたフロアプラン指向 FPGA 高位合成を用いた実装方法の確立と実装された回路の評価を行う．

レジスタ分散型アーキテクチャは，レジスタ集中型アーキテクチャに比べて，FU・レジスタ間の配線遅延を小さくできるため，レイテンシの小さい回路を生成するための有効な技術である．しかし，一方でレジスタ数は増加するため，高位合成段階でのタイミング設計はより複雑になる．更に，フロアプラン指向 FPGA 高位合成では，高位合成段階で決定したフロアプラン情報を実装フローにインプットする必要があり，通常の FPGA 実装とは異なるフローとなる．また，実際の FPGA 実装ではアプリケーション回路のみではなく，外部とのデータのやり取りを行うインターフェース回路も必要となる．しかし，既存研究ではフロアプラン指向 FPGA 高位合成を用いた FPGA 実装方法は明らかにされておらず，前章までの議論も配置・配線後の FPGA 設計ツールのレポートを基に議論をしている．

従って，本章では，前章で提案したフロアプラン指向 FPGA 高位合成手法を用いて，FPGA 実装を行い，性能評価を行う．まず，実装環境およびその際のアプリケーション回路に必要となるインターフェース回路について議論する．次に，フロアプラン指向高位合成手法を用いた際の FPGA 実装フローを提案する．そして，第 5 章で提案したフロアプラン指向高位合成手法を用いて，ベンチマークアプリケーションの 1 つである DCT アプリケーションを例に取り，実際に Xilinx Virtex-7 上に FPGA 実装を行い，正常に動作することを確認する．最後に，既存手法を用いて実装した回路と比較して，実機レベルで提案手法によって合成された回路を評価する．

第 7 章**「まとめ」**では，本論文の内容を総括し，今後の課題を検討する．

第2章　関連研究

2.1　本章の概要

本章では，既存のFPGA向け高位合成手法に関する研究のうち，フロアプランを扱うFPGA向け高位合成手法とMUXコスト削減を図るFPGA向け高位合成手法に分類されるいくつかの既存手法を紹介する．そして，他のLSI向けに提案されたフロアプランを扱う高位合成に適したアーキテクチャを紹介する．

まず，フロアプランを扱うFPGA向け高位合成手法では，(1) 詳細なFPGAアーキテクチャに基づく個別のモジュール配置を扱う手法として，[71,77,78] を紹介し，(2) 抽象化されたモジュールのFPGA配置を扱う手法として，[12,14,15,42] を紹介する．次に，MUXコスト削減を図るFPGA向け高位合成手法として，[11,13,19,39,40,63] を上げ，そのうちの例として，Chenらの手法 [11] とHaraらの手法 [39,40] を紹介する．その後，高位合成内でフロアプランを扱う既存技術の1つであるHDRアーキテクチャ[1–3] を紹介する．

2.2　フロアプランを扱うFPGA向け高位合成手法

本節では，既存のフロアプランを扱うFPGA向け高位合成手法を紹介し，これらの課題について議論する．

既存のフロアプランを扱うFPGA向け高位合成手法として，[12,14,15,42,71,77,78] がある．フロアプランを扱うFPGA向け高位合成手法は，(1) 詳細なFPGAアーキテクチャに基づく個別のモジュール配置を扱う手法 [71,77,78]，と (2) 抽象化されたモジュールのFPGA配置を扱う手法 [12,14,15,42] に大別できる．

2.2.1　個別のモジュール配置を扱う手法

本項では上記の (1) 詳細なFPGAアーキテクチャに基づく個別のモジュール配置を扱う手法に分類される既存手法の例として，Xuらの手法 [71] とZhengらの手法 [77,78] を紹介する．これらの手法は，より正確な配線遅延の見積りが可能になる反面，高位合成段階で個別のモジュール配置を扱うことは，配置対象となるモジュール数が多くなり，フロアプランを含めた高位合成問題を極めて複雑にしてしまう課題がある．

Xuらの手法 [71]

Xuらは [71] にてFPGAアーキテクチャに基づくレイアウト情報をフィードバックしたスケジューリング/バインディング手法を提案している．Xuらの手法 [71] の高位合成アルゴリ

ズムを図 2.1 に示す．Xu らの手法の特徴は，以下の 3 点である．

1. 配置・配線を実際に行わないため，合成時間が短い．
2. 反復改良を用いている．
3. スケジューリングとバインディングを同時に実行する．

Xu らの手法 [71] は FPGA のアーキテクチャを想定し，入力で与えられたリソース制約から FU，レジスタをスケジューリング・バインディングより前に見積り，仮想ネットリストを作成する．仮想のネットリストでは，バインディングされる前の状態であるため，全ての FU が全てのレジスタに繋がれている状態を見積っている．仮想ネットリストを用いて，スケジューリング・バインディングを行い，最終的に制約を満たす RTL ネットリスト，あるいはスケジューリング・バインディングが不可能な場合は現状で最良のネットリストを出力する．従来のスケジューリング・バインディングに配置・配線情報を用いる手法では，実際に配置・配線した情報をフィードバックし，再度スケジューリング・バインディングしていた．回路の合成時間において，配置・配線の実行時間が占める割合は大きい．Xu らの手法 [71] は実際に配置・配線を行わない手法であるため，従来手法に比べて回路の合成時間を削減することができる．

Xu らの手法 [71] では，スケジューリング・バインディングより前に，全てのパスの遅延を見積り，遅延情報を基にカットオフ・ポイントと呼ばれる遅延の上限値を設定する．カットオフ・ポイント以上の遅延の配線を遮断し，変更された仮想ネットリストを基に，スケジューリング・バインディングを実行する．スケジューリング・バインディング結果が制約を満たすかどうかを判定し，満たさない場合は RTL ネットリストを調整し，再度制約を満たすか判定する．調整後の RTL ネットリストが制約を満たさない場合は，スケジューリング・バインディングより前に定めたカットオフ・ポイントが原因と考え，カットオフ・ポイントを修正し，同様の処理を再度実行する．Xu らの手法は反復改良によって解を改良する手法であるため，最終的に最良の解が得られる手法である．

Xu らの手法 [71] では，スケジューリング・バインディングを各 CS ごとに同時に行う手法である．スケジューリング・バインディングの部分は，以下の 4 つのフェーズで構成される．

利用できる FU・レジスタを取得 設定されたカットオフ・ポイントを基に，カットオフ・ポイント以上の配線を遮断し，利用できる FU・レジスタを取得する．

リソース情報よりクロック周期を解析 入力として与えられた時間制約，リソース制約よりクロック周期と利用可能なリソースを解析する．

スケジューリンググループを生成 入力の DFG および算出された CS 数を基に，スケジューリンググループと呼ばれる各 CS において，割り当てられる可能性のある演算ノードの集合を生成する．

スケジューリンググループを選択 割り当てる CS において，スケジューリンググループを 1 つ選択し，ノードのスケジューリングおよびバインディングを行う．

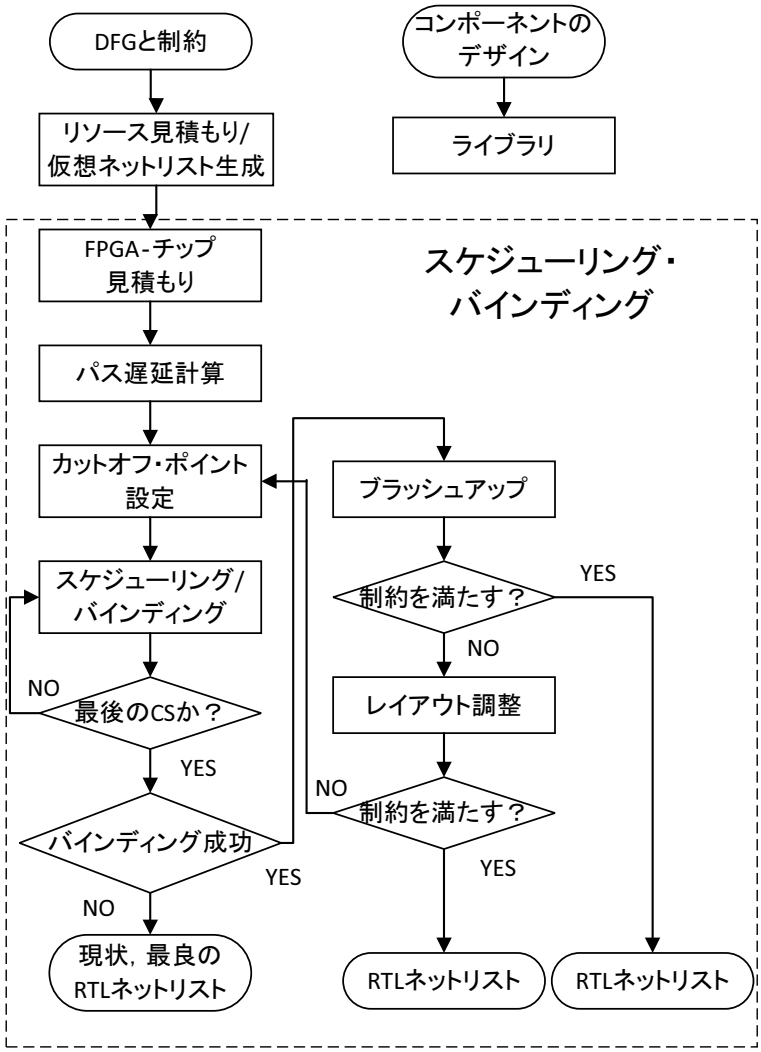


図 2.1: Xu らの手法の高位合成アルゴリズム [71].

Zheng らの手法 [77, 78]

[77, 78] は FPGA の配置配線ツールを結合した高位合成手法を提案している．この手法は，FPGA の配置配線情報に基づいて，モジュール間のデータ転送遅延を見積り，回路の高速化を図る．

Zheng らの手法のアルゴリズムを図 2.2 に示す．これは Altra 社の開発ツールを用いて，高位合成結果を論理合成・配置配線を行う．その結果より，[77] の手法より各パスのサイクル数を算出する．その結果を基に Delay Table を改良し，再度高位合成を行う．反復回数はユーザーの与える制約に依存する．計算機実験の結果，Zheng らの手法は反復しない場合に比べて遅延を最大 22%削減，配置配線を用いない場合に比べて最大 28%削減した．この手法では，配置配線ツールを用いて正確な遅延情報によって最適化を行うことで，高い回路の高速化が実現できている．しかし，実際の FPGA の配置配線ツールを用いるため，1 回のループに要する時間は多く，さらに大規模なアプリケーションを十分に最適化したい場合，必然的に多くのループ回数が必要になり，合成時間が大きく増加すると考えられる．

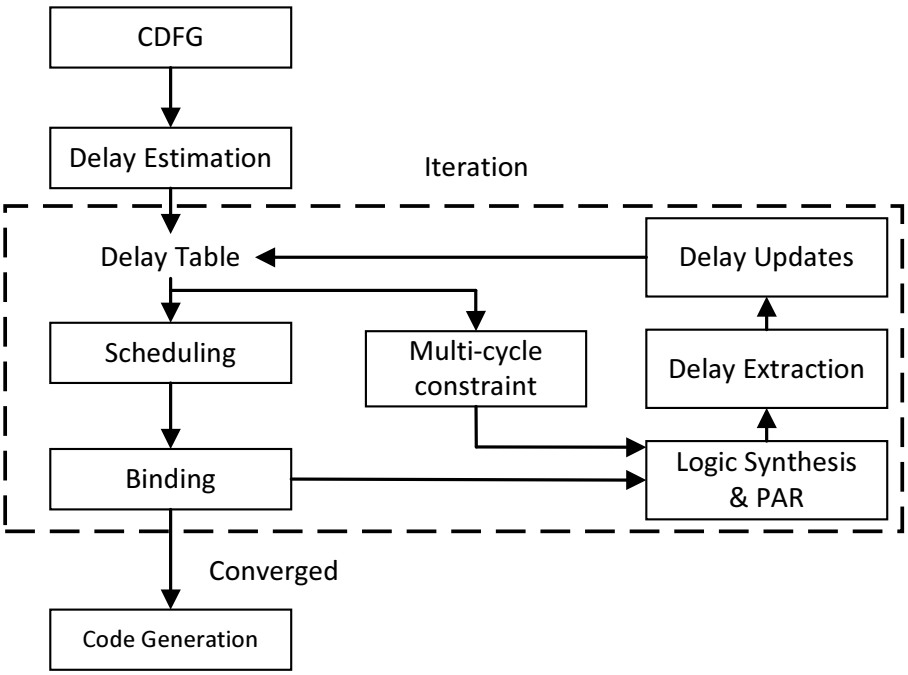


図 2.2: Zheng らの手法 [78].

（1）詳細な FPGA アーキテクチャに基づく個別のモジュール配置を扱う手法に分類される手法は，より正確な配線遅延の見積りが可能になる反面，高位合成段階で個別のモジュール配置を扱うことは，配置対象となるモジュール数が多くなり，高位合成問題を極めて複雑にしてしまう．実際に，Xu らの手法，Zheng らの手法共に，FU・レジスタ単位でのモジュール配置を扱っているが，これでは回路の規模が大きくなった時，それに比例して配置するモジュールも増加してしまう．さらに，FPGA には I/O パッドや特殊セル（DSP ブロックや RAM）など配線遅延に影響を与えるものが多くあり，これらを含めた詳細な FPGA アーキテクチャを想定することは高位合成問題の複雑化を助長し，収束性の課題を招いてしまう．

2.2.2 フロアプラン指向 FPGA 高位合成手法

本項では，フロアプランを含めた高位合成問題をシンプルに解くための技術として，(2) 抽象化されたモジュールの FPGA 配置を扱う手法に分類される既存手法を紹介する．これらは，高位合成問題をシンプルに解ける技術となっている一方で，配線遅延・クロックスキューの見積りに課題が残る．

(2) 抽象化されたモジュールの FPGA 配置を扱う手法は「フロアプラン指向 FPGA 高位合成手法」と呼ばれ，各々の手法で抽象化されたモジュール単位で FPGA 上の配置を扱う．この時，詳細なアーキテクチャを抽象化した FPGA モデル上での配置情報を想定することで，高位合成問題を非常に簡潔に解くことができる．本項ではフロアプラン指向 FPGA 高位合成手法の例として，Cong らの Spreading Score を用いた手法 [14, 15] と RDR アーキテクチャとそれを対象とした高位合成手法 (MCAS) [12]，マクロと呼ばれる単位でフロアプランする Huang らの手法 [42] を紹介する．

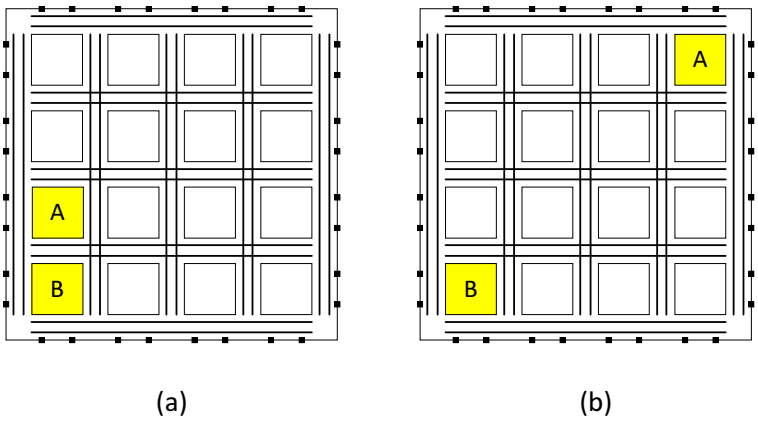


図 2.3: コンポーネントの配置例.

Spreading Score を用いた手法 [14, 15]

Cong らは [14, 15] で Spreading score というチップ上の各コンポーネントの分散を表す値を用いて配線混雑度を考慮した手法を提案している．この手法は，配線混雑度を考慮したフロアプランを行うことで，レイテンシが改善された結果を示しているが，高位合成段階で配線遅延を考慮した手法では無い.

Cong らの手法 [14, 15] では高位合成段階でレイアウトを作成する際に Spreading Score というコンポーネントの分散を示す値を用いる．図 2.3 にコンポーネントの配置例を示す.

図 2.3(a) は，隣接したコンポーネントの配置である．コンポーネント A と B の配線パターンを考えた場合，使用できる配線用チャネルは A と B の間の 1 区画分である．それに対し，図 2.3(b) の離れたコンポーネントの配置例ではコンポーネント A と B の配線において使用できる配線用チャネルは縦横それぞれ 3 区画分存在する．つまり，離れたコンポーネント配置の方が選択できる配線パターンが多いため配線混雑度が低くできる.

Spreading Score とは，以下の式で表される.

$$\sum_{i=1}^{n} w_i \|p_i\|^2 \tag{2.1}$$

p_i はコンポーネント i の位置ベクトルである．w_i コンポーネントの面積による重みづけの値である．図 2.4 に Spreading Score の例を示す． ここでは各コンポーネントの面積は 1 と仮定する各コンポーネントが近接して配置された図 2.4(a) では Spreading Score 値が $1 \times 1^2 \times 4 = 4$ である．各コンポーネントがより分散して配置された図 2.4(b) では Spreading Score の値が $1 \times 3^2 \times 4 = 36$ となる．以上から，Spreading Score の値が大きいほど，よりコンポーネントが分散して配置されていることがわかる．Cong らの手法 [14, 15] では以下の条件の下，Spreading Score の値を最大化するレイアウトを求める.

$$\begin{aligned} \sum_{i=1}^{n} w_i p_i &= 0 \\ \|p_i - pj\| \leqq l_{ij} \quad & \forall (i,j) \in E \end{aligned} \tag{2.2}$$

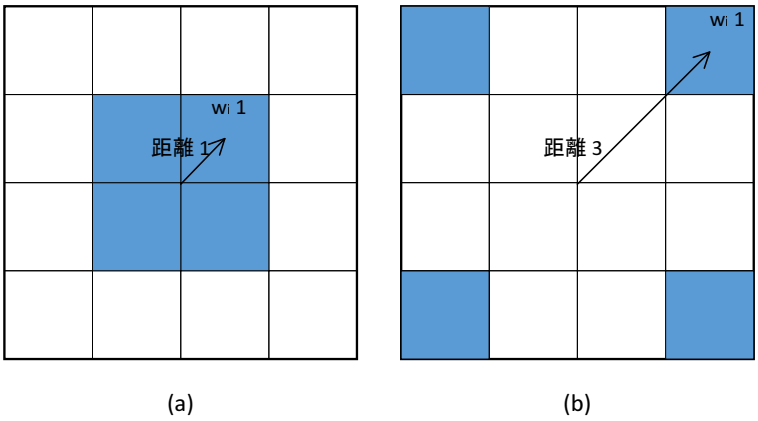


図 2.4: Spreading Score の例.

l_{ij} はコンポーネント i と j における配線長の上限値，E はレイアウトの有向グラフにおける方向エッジの集合である．Cong らは論文内で，コンポーネントの単位を定義しておらず，この手法は複数の FU/レジスタをまとめて 1 つのコンポーネントとした場合にも適用可能である．

Cong らの手法 [14, 15] を用いた結果，[15] では ASIC の設計で Spreading Score を用いて最適化を行っていくことにより，総面積と遅延時間が最適化された．[14] では FPGA での設計で Spreading Score を用いて最適化を行っていくことにより，配線しやすくなる様子が観測できた．この要因と考えられるのは，Spreading Score による最適化によって演算器の共用，FU の共有などの設計が行われたことである．

Cong らの手法 [14, 15] は，配線混雑度を緩和することでクリティカルパスの迂回を避け，高速化が図れることが予想される．しかし，この手法はフロアプランにより配線遅延を見積り，配線遅延の影響を踏まえた高位設計を実現しているわけではない．

RDR アーキテクチャとそれを対象にした高位合成手法 (MCAS)

Cong らは [12] にて，RDR（Regular Distributed-Register）アーキテクチャを提案している．このアーキテクチャは，均一の区画分割により高位合成中の配線遅延の予測を容易化している一方で，一定の大きさに分割するため遅延・面積オーバーヘッドが大きい欠点がある．また，クロックスキューを考慮しておらず，配線遅延の見積りモデルに対しても課題が残る．

RDR アーキテクチャは，FPGA チップを複数の同じ大きさの島に分割する．各島は，演算器とマルチプレクサの集合，レジスタ・ファイル，コントローラで構成される．RDR アーキテクチャはレジスタ分散型アーキテクチャの 1 つであり，レジスタ・ファイルを各島が持つことによって，複数サイクルレジスタ間通信の設計をサポートし，高い規則性を持ったアーキテクチャである．RDR アーキテクチャはチップを均一の区画に分割することで，高位合成中の配線遅延の予測を容易化している．RDR アーキテクチャの構成を図 2.5 に示す.

Local Computational Cluster (LCC) クラスタ化された機能ユニット群である．ここで，機能ユニット群とは，FU および MUX を指す.

Register file 各島専用のローカルレジスタファイルである．LCC に含まれる機能ユニット

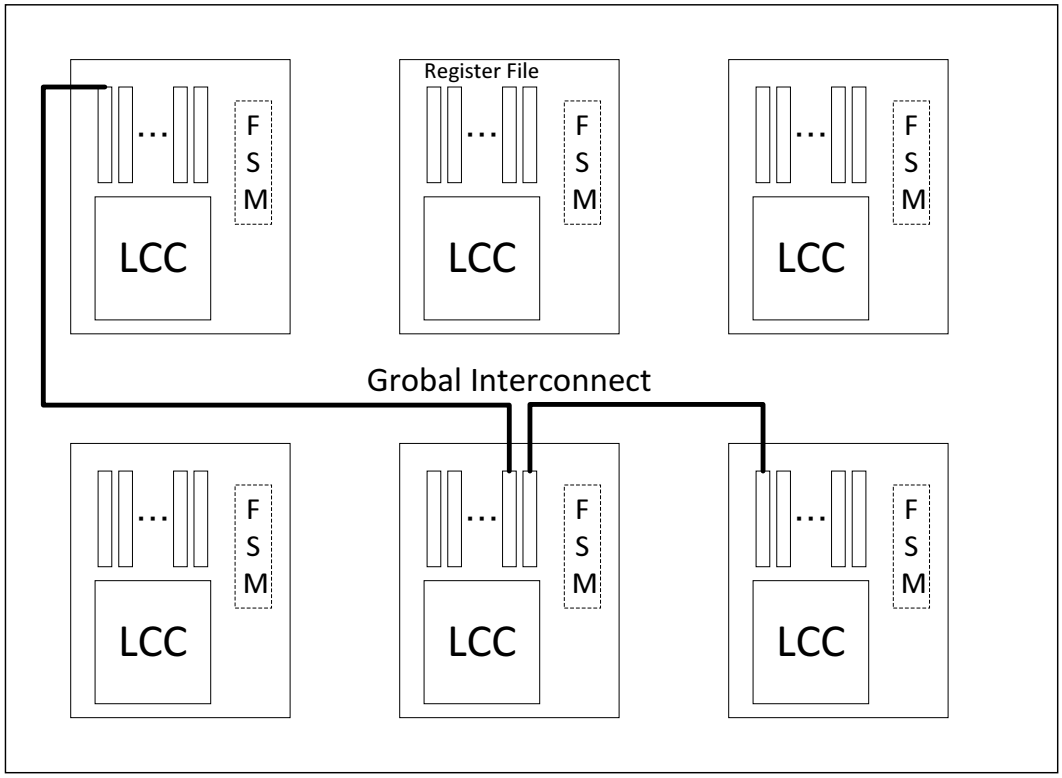


図 2.5: RDR アーキテクチャ[12].

群は同一の島内のレジスタファイルにのみアクセスできる.

Finite State Machine (FSM) 各島専用のコントローラである．同一の島に含まれる LCC とレジスタを制御する.

異なる島に配置された FU 同士でデータをやり取りする場合は，ローカルレジスタファイル間データ転送を用いる.

RDR アーキテクチャは規則的に分割されているため，各島間の配線遅延の見積りが非常に容易になり，さらに区画内のモジュール配置は抽象化され配置の粒度が粗くなる．粒度の粗い区画で分割し区画ごとにフロアプランを行うのは，FPGA に適しているという長所がある．しかし，RDR アーキテクチャはチップを一定の大きさに分割するため遅延・面積オーバーヘッドが大きい欠点がある.

次に図 2.6 に RDR アーキテクチャを対象とした高位合成手法 : MCAS を示す．MCAS は，対象アプリケーションの CDFG (Control-Data Flow Graph)，RDR アーキテクチャの仕様，クロック周期制約を入力として与え，RTL 記述，フロアプラン制約，マルチサイクルパス制約を得る．MCAS の特徴として，以下の特徴が挙げられる.

1. レジスタ分散型アーキテクチャ回路の 1 つである RDR アーキテクチャを対象としている.
2. フロアプラン決定後のデータパス生成を 1 度に制限している.

RDR アーキテクチャは，上記で述べた通り，レジスタ分散型アーキテクチャ回路の 1 つであり，これは一般的にレジスタ集中型アーキテクチャに比べて，FU・レジスタ間の配線遅延

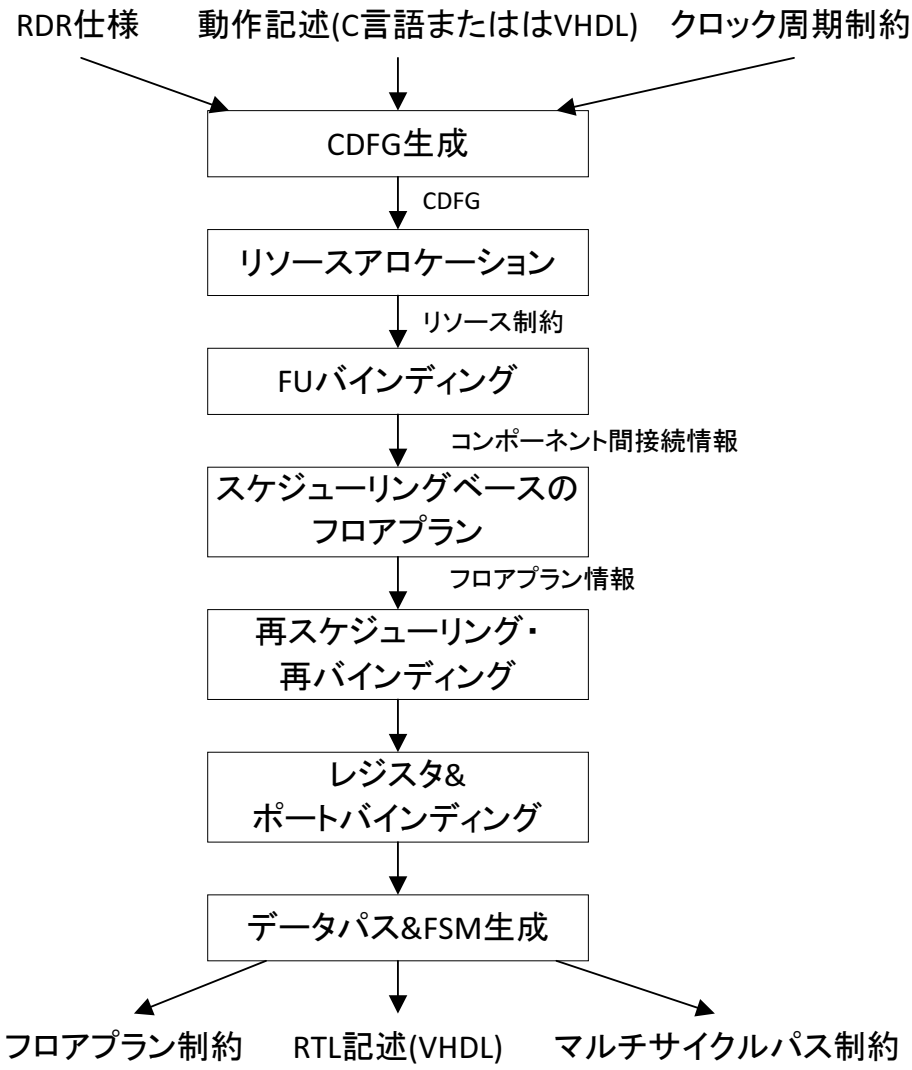


図 2.6: RDR アーキテクチャを対象とした高位合成手法：MCAS [12].

が小さくなるため，レイテンシの小さい回路を合成する上で有利である．また，MCAS は，入力の動作記述に対する CDFG を生成・リソースアロケーション実施後，FU バインディングを行い，フロアプランを決定する．その後，決定したフロアプランを基に 1 度だけ再スケジューリング・バインディングを行い，データパスを決定する．フロアプランを扱う高位合成手法では，第 2.2.1 項の Xu らの手法 [71] や Zheng らの手法 [78] のように，データパスとフロアプランに相互依存の関係があるため，データパス生成とフロアプランの変更を繰り返し，良質な回路を得るために解を反復改良していく．しかし，反復的に繰り返すと収束性によっては合成時間が爆発的に増えてしまう可能性がある．MCAS はこれを防ぐため，フロアプラン決定後 1 度だけ再度データパス生成を行い，最小限の解の改良に留めている．

Huang らの手法 [42]

Huang らは [42] で，スケジューリング・バインディング結果を基にマクロという単位でモジュールを抽象化し，フロアプランする手法を提案している．この手法は，独自のアーキテ

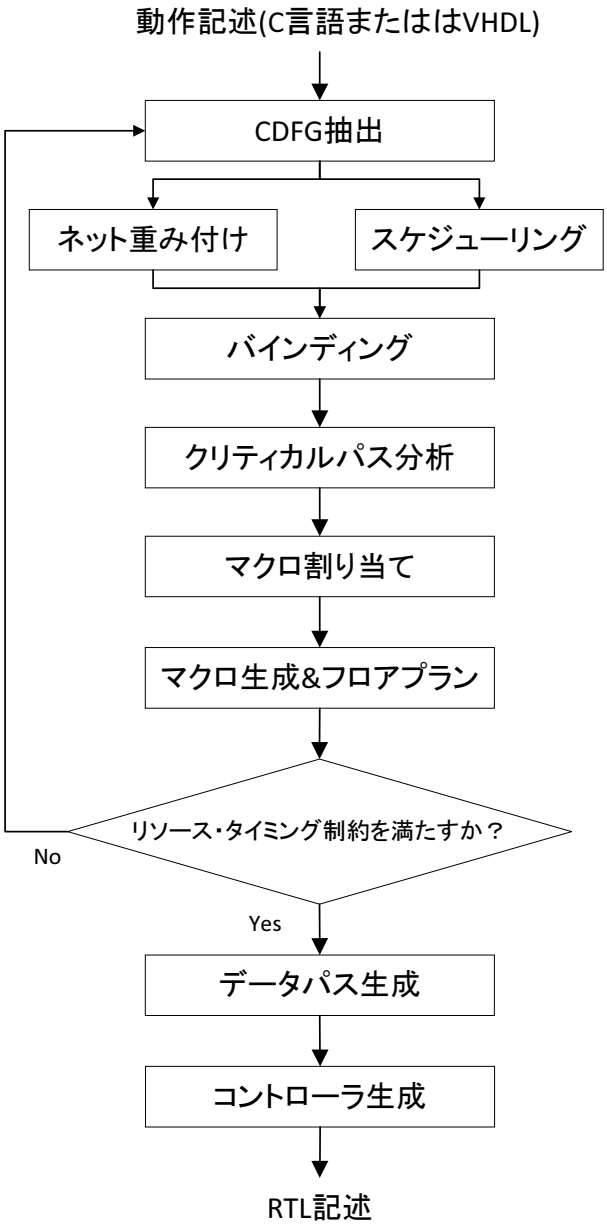


図 2.7: Huang らの高位合成アルゴリズム [42].

クチャを用いらずに一般的な回路アーキテクチャベースに抽象化したモジュール単位のフロアプランを実現している．その一方で，[12] の手法と同様にクロックスキューを考慮しておらず，配線遅延の見積りモデルに対しても課題が残る．

図 2.7 に Huang らの高位合成アルゴリズム，図 2.8 に DFG 上のマクロの例を示す．Huang らの手法の特徴として以下が挙げられる．

1. データパス生成・フロアプランの反復改良を採用している．

2. クリティカルパス上の一部のコンポーネントに限定し，フロアプランを扱っている．

Huang らの手法は，データパス生成・フロアプランの反復改良を行い，リソース・タイミング制約を満たす良質な解を求める．データパス生成では，DFG をスケジューリング・バ

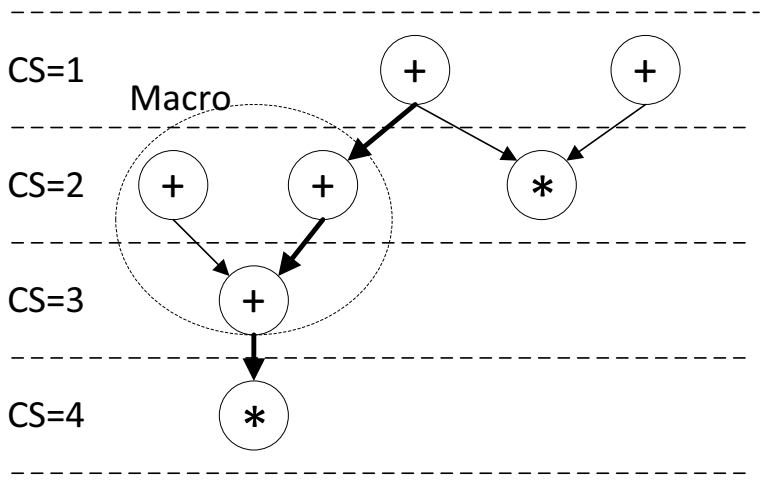


図 2.8: DFG 上のマクロの例 [42].

インディング後，ネットの重みから DFG 上のクリティカルパスを特定する（図 2.8 の太線のデータフロー）．この時，一部の演算ノードを FPGA 上において，近くに配置することで DFG 上のクリティカルパスにおける配線遅延の影響を削減し，レイテンシの削減を図る．この近くに配置される演算ノードの集合をマクロ（図 2.8 の点線で囲まれた部分）と定義している．

フロアプランでは，マクロおよびクリティカルパス上の一部の重みの大きい演算ノードが割り当てられているコンポーネントのみに限定しフロアプランを行う．全てのコンポーネントに対してフロアプランをせず，レイテンシのボトルネックとなるコンポーネントに限定することで，フロアプランおよび高位合成問題を簡素化している．

この手法は，独自のアーキテクチャを用いらずに一般的な回路アーキテクチャベースに抽象化したモジュール単位のフロアプランを実現している長所がある．しかし，一部の FU を除いて FU とレジスタのフロアプランは扱わないため，これらの配置は下位工程で使用する配置・配線ツールに依存してしまう．また，[12] の手法と同様にクロックスキューを考慮しておらず，配線遅延の見積りモデルに対しても課題が残る．

[12,42] などのフロアプラン指向 FPGA 高位合成手法では，独自の抽象化したモジュールおよびそのフロアプランにより，高位合成問題の複雑化を抑え，フロアプランを扱うことを実現している．また，抽象化したモジュールの配置情報から配線遅延を見積っているが，単純な配線遅延の見積りモデルを採用しており，その妥当性は検証されていない．さらに，これらの手法は，クロックスキューを考慮していない．

以上の議論より，新たなフロアプラン指向 FPGA 高位合成で使用される配線遅延・クロックスキューの見積りモデルの提案が必要と考える．フロアプラン指向 FPGA 高位合成で使用される配線遅延・クロックスキューの見積りモデルは高位合成段階でのそれぞれの見積り精度に大きな影響を与える．高位合成段階での見積り誤差はフロアプランの決定に影響を及ぼし，結果として配線遅延・クロックスキューの影響が大きいフロアプラン解を導き，回路のレイテンシの深刻な悪化を招く恐れがある．従って，フロアプラン指向高位合成で用いる配線遅延・クロックスキューの見積りモデルに関して検証し，妥当な精度であるモデルを用

いる必要がある．

2.3 MUX コスト削減を図る FPGA 向け高位合成手法

本節では，既存の MUX コスト削減を図る FPGA 向け高位合成手法を紹介し，これらの課題について議論する．それぞれが独自の目的関数を用いたバインディング手法により，MUX のコスト削減を達成し，レイテンシ改善を実現している．しかし，これらは配線遅延を考慮しておらず，独自のバインディング手法を用いているため，前節で紹介したフロアプラン指向 FPGA 高位合成手法と組み合わせることが難しい．

既存の MUX コストを削減する FPGA 向け高位合成手法として，[11,13,19,39,40,63] がある．[11] は，MUX の総入力数を削減し複数電源電圧技術を併用することで，低消費電力化を図る FPGA 向け高位合成手法を提案している．[39,40] では，FU やレジスタの共有/非共有によって生じる MUX の挿入を遅延・面積に関して定量的に評価し，回路性能の向上を図る FPGA 向け高位合成手法を提案している．[13] では，FPGA アーキテクチャに基づいてリソースバインディングと MUX を最適化する FPGA 向け高位合成手法を提案している．[19] では，FU・レジスタへの入力数最小化を目的とし，FU バインディングバインディングとレジスタバインディングを同時に実行する FPGA 向け高位合成手法を提案している．[63] では，FU の入力部の MUX の大きさを削減し，クリティカルパス遅延を削減する FPGA 向け高位合成手法を提案している．本節では，MUX を削減する FPGA 向け高位合成手法の例として，Chen らの手法 [11] と Hara らの手法 [39,40] を紹介する．

2.3.1 Chen らの手法

Chen らは，[11] で LOPASS と呼ばれる手法を提案している．LOPASS は，FPGA を対象として想定されている．LOPASS では，対象の FPGA より LUT，レジスタ，配線の消費電力の見積り値を計算し，スケジューリング・FU バインディングでは，LUT，レジスタ，配線の消費電力の見積り値を目的関数として用いて，スケジューリング・FU バインディング解を複数電源電圧を考慮して消費電力を最小化するよう SA（Simulated Annealing）法を用いて最適化する．そして，それを反復し解を改良する手法である．また，レジスタバインディングにおいては，レフトエッジによる解を初期解として，MUX の総入力数を最小化するように 2 部グラフマッチング [41] を用いて反復改良する手法である．その後，ポート割り当てを改良し MUX 数の最適化を図る手法である．

LOPASS は，以下の FPGA の特徴を考慮して提案されている．

1. FPGA はチップ全体に分散されてレジスタが提供されている．
2. 入力数の多い MUX は効果的でない．
3. FU やレジスタの数が少ないほど，面積や消費電力が少ないとは限らない．

スケジューリング/FU バインディング

LOPASS のスケジューリング/FU バインディングでは，SA 法を用いて RT レベルでの消費電力見積もり値の最適化を図る手法である．

RT レベルでの消費電力見積もり値 $P_{Dynamic}$，P_{Static} を以下のように定める．

$$P_{Dynamic} = S(P_{LUT} + P_{REG} + P_{LW} + P_{GW}) \tag{2.3}$$

$$P_{Static} = P_{Idle\_LUT} + P_{Static\_LB} + P_{Static\_GB} \tag{2.4}$$

P_{LUT} は全体の LUT の消費電力の見積もり値，P_{REG} は全体の LUT の消費電力の見積もり値，P_{LW} は全体のスライス内の配線の消費電力の見積もり値，P_{GW} は全体のスライス間の配線の消費電力の見積もり値を表す．ここで，P_{LW} と P_{GW} は供給する電源に伴って変動する．ここで，複数電源電圧を用いて消費電力の削減を図る．

LOPASS のスケジューリング/FU バインディングでは，初期解としてスケジューリングはリストスケジューリング，FU バインディングは force-directed アルゴリズムを用いた解を与える．この初期解を基に SA によって，全体の消費電力（$P_{Dynamic}$ と P_{Static} の和）を最小化する解を求める．SA における各反復での処理は以下の 5 種類がある．

再選択 他の FU を選択し，移動させる．

交換 2 つの別々の FU に割り当てられている演算を交換する．

併合 ある FU に割り当てられている全ての演算を別の FU に統合する．

分割 1 つの FU に割り当てられている演算を 2 つの FU に分ける．

混合 2 つの FU を選択し，併合を行った後，分割する．

レジスタバインディング

LOPASS のレジスタバインディングでは，レフトエッジによるレジスタ数最小の解を初期解とし，2 部グラフマッチング [41] によって MUX の総入力数を最小とするレジスタバインディング解を得る．

2 部グラフマッチングでは以下の重み関数を用いて各エッジに重み付けをする．

$$w(v, r) = \alpha_1 x_1(v, r) + \alpha_2 x_2(v, r) + \beta y(v, r) \tag{2.5}$$

$x_1(v, r)$ は変数 v がレジスタ r に割り当てられた時のレジスタの MUX（図 2.9(a)）の入力数，$x_2(v, r)$ は変数 v がレジスタ r に割り当てられた時に増加する FU の MUX（図 2.9(b)）の入力数の数，$y(v, r)$ は変数 v がレジスタ r に割り当てられた時の FU の MUX の総入力数，α, β はパラメータを表す．LOPASS のレジスタバインディングでは，変数割り当てによって生ずる MUX をレジスタの MUX と FU のポートの MUX の 2 つに分けて考えている．

MUX 最適化のポート割り当て手法

LOPASS では，以下のようなポート割り当てを最適化することで，更なる MUX コストの削減を図っている．

図 2.10 に LOPASS のポート割り当てでの動機付けの例を示す．図 2.10(a) と (b) で違う点は，FU のポートの前に MUX がある点とレジスタの前に MUX がある点である．LOPASS では図 2.10(a) と (b) では，(b) の方を良い解と考える．図 2.10(b) は，(a) に比べ，MUX の総入力数が少なく，レジスタ数も少ないためである．LOPASS では，レジスタの前に MUX を作りやすいよう目的関数を定義し，レジスタバインディング解を求めている．

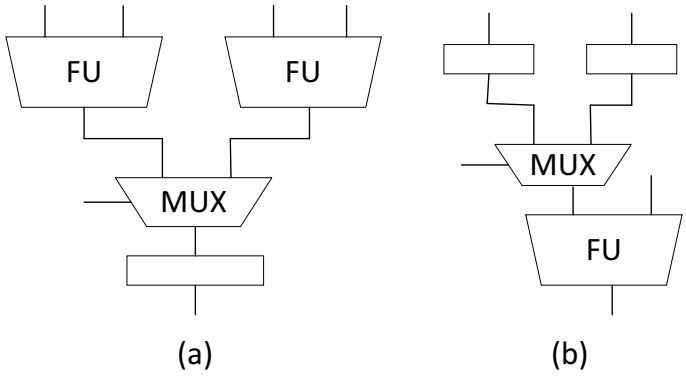


図 2.9: LOPASS での MUX の場合分け.

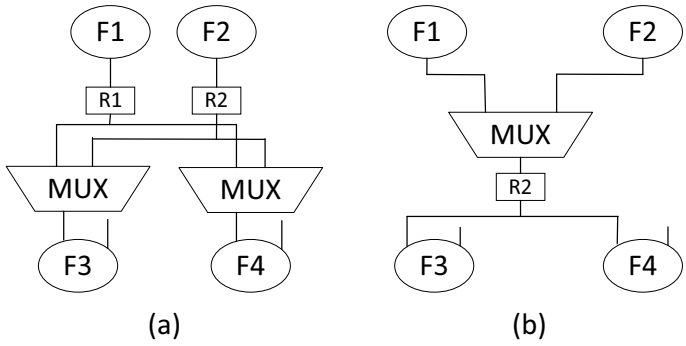


図 2.10: LOPASS での MUX ポート割り当ての動機付けの例.

ポート定義手法

LOPASS では，ポート定義手法も提案している．図 2.11 にポート定義手法の動機付けの例を示す．図 2.11(c)，(d) はそれぞれ図 2.11(a)，(b) の結果において変数の入力される FU のポートを変えることにより，MUX の総入力数を改善した例である．LOPASS では以上のような FU のポート定義により MUX の総入力数を減らせることを示している．

実験結果

LOPASS は，従来手法と比較して，クリティカルパス遅延を 10.6%，消費電力を 61.6%改善された値を示した．

2.3.2 Hara らの手法

Hara らの手法は，レジスタを共有を積極的に行わず，FU に付加される MUX に着目し，MUX のコストの最適化を図る手法である．

動機付け

Hara らの手法の動機付けを図 2.12，図 2.13 に示す．図 2.12，図 2.13 のように FU もしくはレジスタを共有した場合に MUX が必要になる．つまり，できる限りリソースを共有しなけ

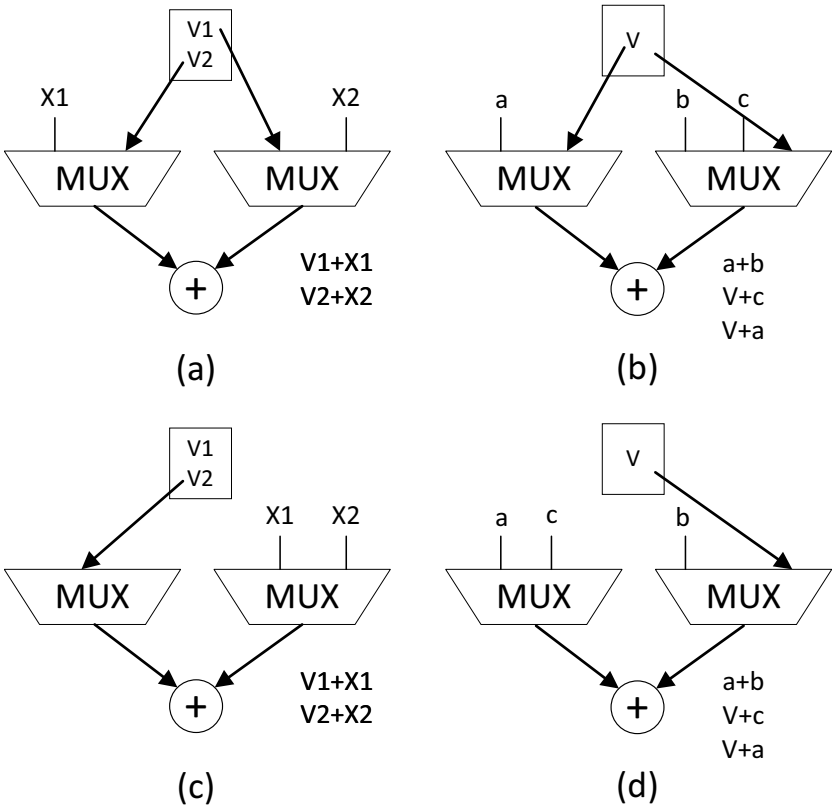


図 2.11: ポート定義手法での動機付けの例.

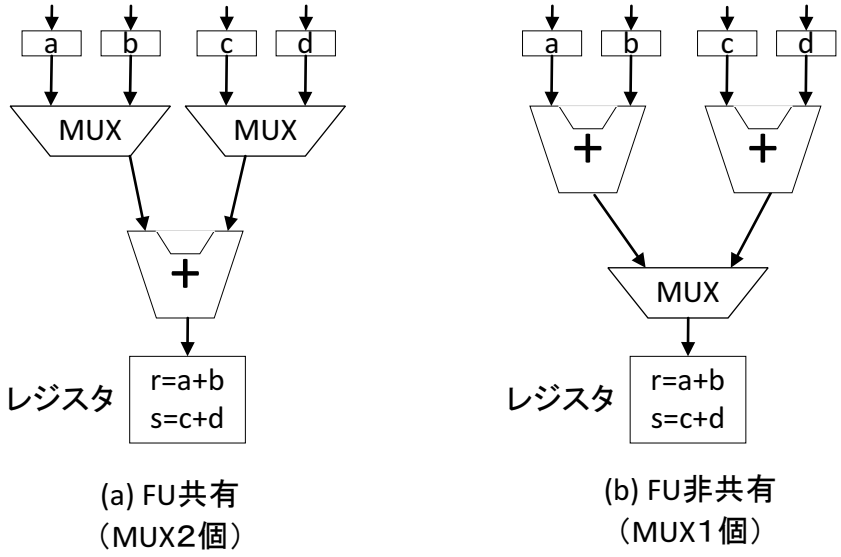


図 2.12: FU の共有/非共有の場合.

れば MUX の数を削減することができる．Hara らの手法はこれを利用した手法である．[39] の中で，以下の理由によりレジスタを共有する必要はないと述べている．

(1) FPGA 上では FU に比べてレジスタは面積が小さい.

(2) レジスタ共有により MUX が挿入され，面積増大する可能性がある.

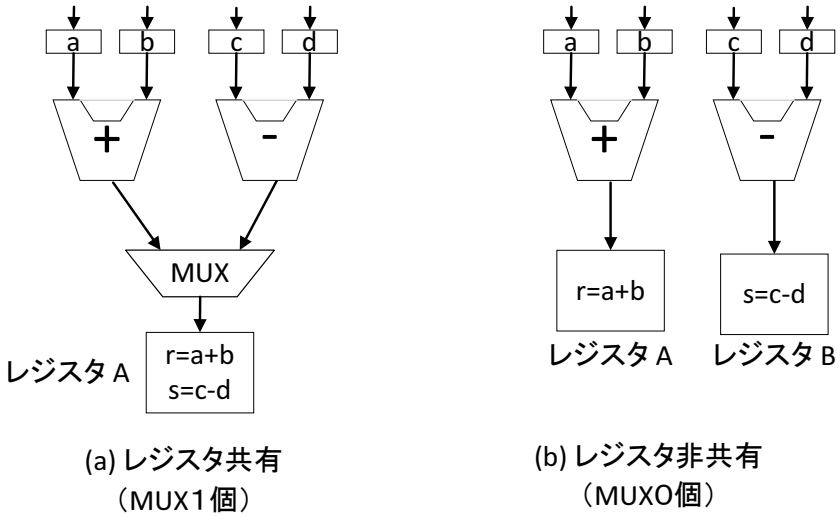


図 2.13: レジスタの共有/非共有.

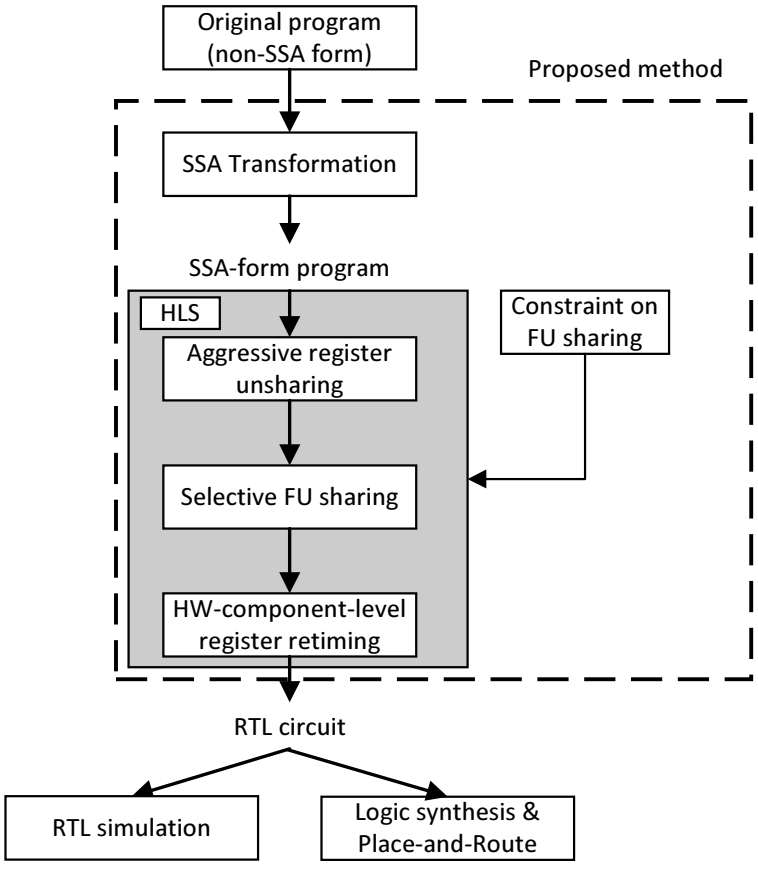


図 2.14: Hara らの手法.

(3) FPGA 上には十分多数のレジスタがあり，不足することはほぼ無い.

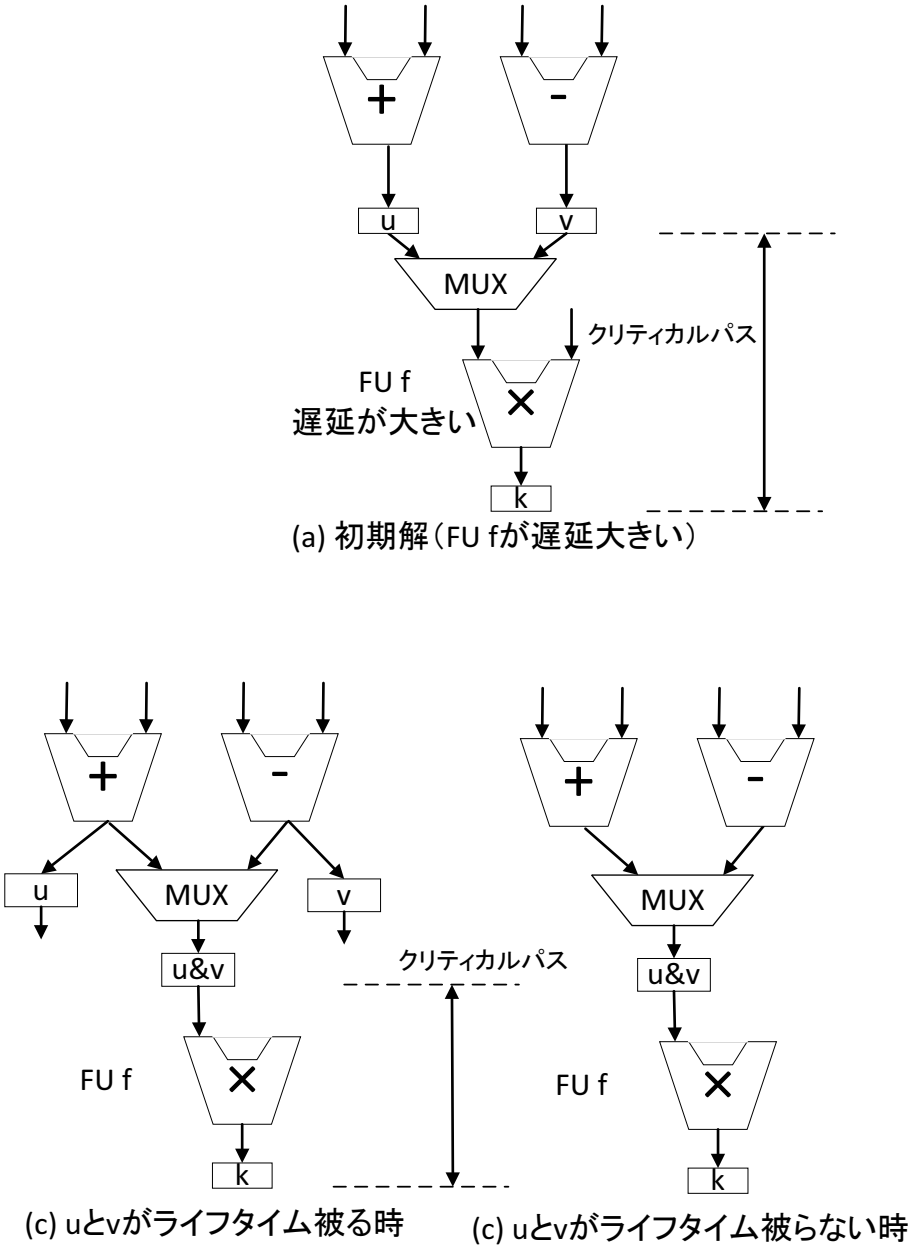


図 2.15: HW-component-level register retiming の例.

アルゴリズム

Hara らの手法のアルゴリズムを図 2.14 に示す．この手法は [52] に加えて Selective FU sharing と HW-component-level register retiming を行う．

まず，動作記述に対して SSA（Static Single Assignment）の形式に変更を行う．次に，Aggressive register unsharing では復号変数（multiple variable）のレジスタ非共有を行う．これは次の Selective FU sharing でより多くの MUX の削減に寄与する．Selective FU sharing では，FU の非共有/共有の変更を行い，面積において最適化を図る．この時，非共有に変更する場合は増加する FU の面積と削減される MUX の面積，共有に変更する場合は削減される FU の面積と増加する MUX の面積を考慮し，判断を行う．つまり，大きい FU の非共有

や小さいFUの共有を避けるための処理である．HW-component-level register retimingの例を図2.15に示す．HW-component-level register retimingでは，遅延の大きいFUが存在するクリティカルパスに対して，レジスタの挿入などを行いMUXの位置をクリティカルパス以外の部分へ移動させる．これによって，クリティカルパス遅延の削減を図る．

実験結果

計算機実験結果より，Haraらの手法は面積オーバーヘッド平均13.8%で遅延を49.5%削減を達成した．これは，[52]が面積オーバーヘッド平均39.36%で遅延63.13%削減したのに比べて，面積オーバーヘッドを大幅に削減し，高い遅延削減を実現していると言える．

以上より，MUX削減を図る手法の例として．Chenらの手法とHaraらの手法を紹介した．これらを含めMUX削減を図る既存手法ではそれぞれが独自の目的関数を用いたバインディングにより，目的を達成している．また，前節で紹介したように，フロアプラン指向FPGA高位合成手法でもフロアプランに基づく配線遅延情報をデータパス生成に用いるため，独自のバインディング手法を用いており，一概にこれらを組み合わせればFPGAのフロアプランを扱った上でMUXのコストを削減を達成できるわけではない．

2.4 HDRアーキテクチャ

本節では，他のLSI向けに提案されたフロアプランを扱う高位合成に適したアーキテクチャを紹介する．まず，Ohchiらが提案したGDR（Generalized Distributed Register）アーキテクチャ[58]を紹介し，長所・短所を議論する．その後，Abeらが提案したHDR（Huddle-based Distributed-Register）アーキテクチャ[1–3]を紹介し，FPGA向け高位合成への適用を議論する．

Ohchiらは[58]でGDRアーキテクチャを提案した．（図2.16）これは，レジスタ分散型アーキテクチャの1つで，高位合成段階でFUのフロアプランを扱い，データ転送遅延がボトルネックとならないFUでレジスタ・コントローラを共有し，ボトルネックとなるFUにはローカルレジスタ・ローカルコントローラを付加するアーキテクチャである．

GDRアーキテクチャは高速かつ小面積という利点を持つが，第2.2節で述べたように，各モジュールごとのフロアプランは高位合成問題を複雑にし，FPGAに利用した時にはチップ上のさまざまな機構と相まって収束性の課題はさらに深刻となる可能性がある．

一方，GDRアーキテクチャと同様にレジスタ分散型アーキテクチャの1つとして第2.2節で紹介したRDRアーキテクチャがある．RDRアーキテクチャの島という区画単位のフロアプランは非常にシンプルで，FPGAに適しているという利点がある．しかし，遅延・面積オーバーヘッドが大きいという欠点がある．

Abeらは[1–3]にて，HDRアーキテクチャを新たに提案した．HDRアーキテクチャはGDRアーキテクチャにハドルというRDRの区画の概念を導入し，各モジュールを抽象化したアーキテクチャである．HDRアーキテクチャでは回路全体をハドル（Huddle）と呼ぶ区画で分割し，ハドル毎にフロアプランを行い，配線遅延を適切に扱う．GDRアーキテクチャと異

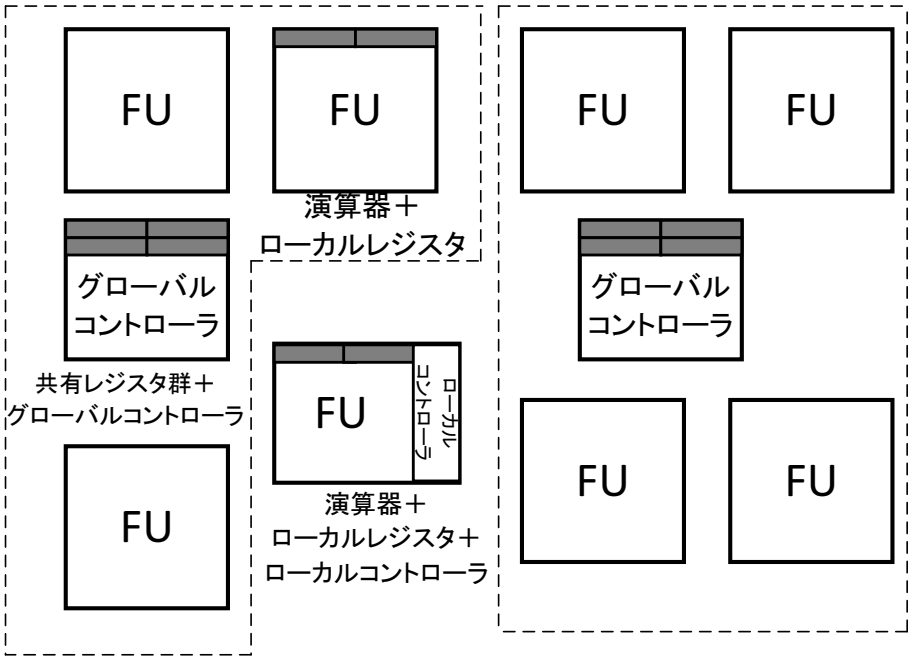


図 2.16: GDR アーキテクチャ[58].

なり，ハドルの中は抽象化され詳細な分割を伴わない．ハドルは配線遅延の影響の無い範囲で任意の矩形を取り，演算器やレジスタ，コントローラ，レベルコンバータを共有する．

HDR アーキテクチャの構成を図 3.2 に示す．ハドルは以下の要素で構成される．

Huddled Local Register（HLR） 各ハドル専用のローカルレジスタとマルチプレクサの集合である．

Huddled Functional Unit（HFU） ハドルに集められた演算器の集合である．ハドル内で処理する演算に必要な演算器を必要数持ち，同一のハドル内の HLR のみにアクセスできる．

Finite State Machine（FSM） 各ハドル専用のコントローラである．同一ハドル内の HFU と HLR を制御する．

Huddled Level Converter（HLC） ハドルに集められたレベルコンバータの集合である．電圧の異なるハドルとデータ転送を行う際，HLC を用いる．

同一ハドル内の HFU でデータを処理する場合，ハドル内の HLR を使うことでデータ転送時間を無視できる．異なるハドルの HFU 同士でデータ通信する場合，HLR 間データ転送を行う．HLR 間データ転送する際，各ハドルの電圧が異なる場合 HLC を用いる．

HDR アーキテクチャは，以下の特徴から FPGA に適したアーキテクチャと考える．HDR アーキテクチャは，ハドル導入によるモジュールの抽象化によって，RDR アーキテクチャと同様に FPGA に適した区画ごとのフロアプランを採用している．また，任意の矩形を取るハドルに対しフロアプランするため，GDR アーキテクチャと同様に小面積で高速なアーキテクチャとなる．また，FPGA ではチップ全体に分散してレジスタが配置されているので，レジスタ分散型アーキテクチャは FPGA に適していると考えられる．以上の点から，HDR

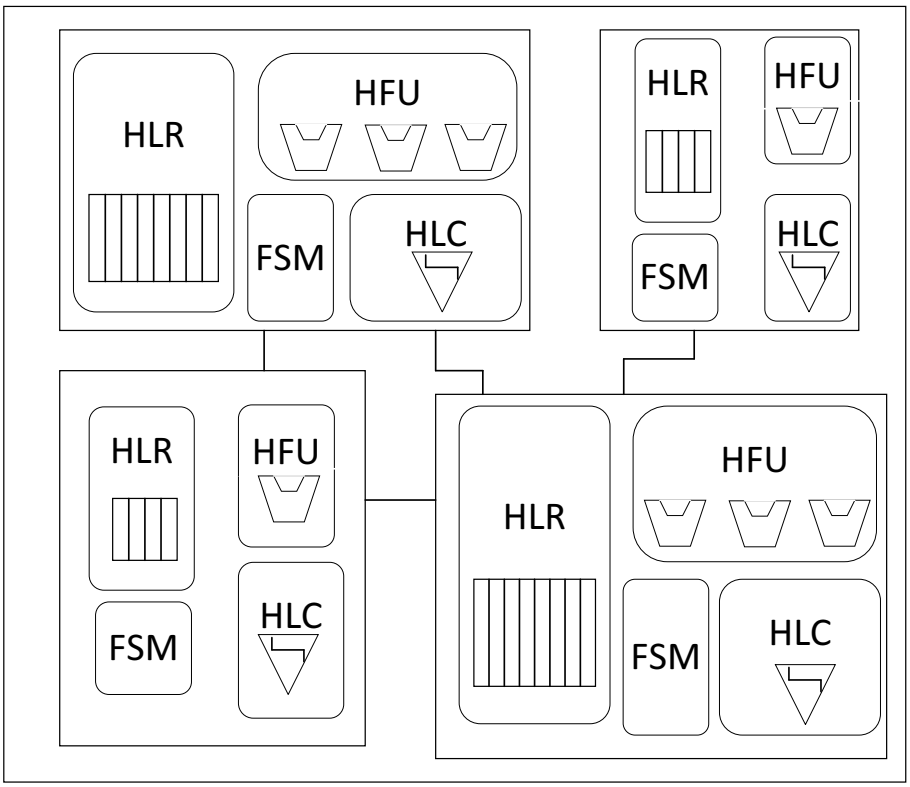


図 2.17: HDR アーキテクチャ[1–3].

アーキテクチャは RDR アーキテクチャと GDR アーキテクチャの利点を併せ持つため，現状非常に FPGA に適したアーキテクチャと考えられる．

しかし，[1–3] で提案された HDR アーキテクチャの高位合成手法は ASIC を対象としており，FPGA 向けの HDR アーキテクチャを対象とした高位合成手法は存在しない．従って，フロアプラン指向 FPGA 高位合成手法に HDR アーキテクチャを採用するためには，新たな高位合成手法の提案が必要となる．

2.5 本章のまとめ

本章では，既存の関連研究のうち，フロアプランを扱う FPGA 向け高位合成手法と MUX コスト削減を図る FPGA 向け高位合成手法に分類されるいくつかの既存手法を紹介した．そして，他の LSI 向けに提案されたフロアプランを扱う高位合成に適したアーキテクチャを紹介した．

表 2.1 に既存の FPGA 向け高位合成手法の分類を示す．既存の FPGA 向け高位合成手法では，フロアプランを扱う手法，MUX コスト削減を図る手法，各々が提案されているが，これらを同時に達成する高位合成手法は実現されていない．フロアプランを扱う FPGA 向け高位合成手法では，フロアプランを含めた高位合成問題を有効に解くために，フロアプラン指向高位合成手法が注目されており，[12,14,15,42] を紹介した．しかし，その中で用いられているフロアプラン指向アーキテクチャや配線遅延・クロックスキューの見積りモデルには，課

表 2.1: 既存の FPGA 向け高位合成手法の分類.

分類		該当する手法	利点	欠点	
フロアプランを扱う手法	個別のモジュール配置を扱う手法	[71, 77, 78]	より正確な配線遅延の見積りが可能	高位合成問題の複雑化	クロックスキューを考慮していない
	フロアプラン指向 FPGA 高位合成手法	[12, 14, 15, 42]	フロアプランを含む高位合成問題の簡略化	配線遅延の見積り精度が低い	
MUX コスト削減を図る手法		[11, 13, 19, 39, 40, 63]	独自のバインディング手法による MUX 削減	配線遅延を考慮していない	

題がある．一方，MUX コスト削減を図る FPGA 向け高位合成手法では [11, 13, 19, 39, 40, 63] が提案されており，本章では Chen らの手法 [11] と Hara らの手法 [39, 40] を例に取り，紹介した．既存研究では，フロアプランを扱う手法，MUX コスト削減を図る手法，各々が提案されているが，既存手法ではそれぞれが独自のバインディング手法により目的を達成しており，一概に組み合わるのは難しい．従って，これらを同時に達成する高位合成手法は私の知る限り提案されていない．

また，FPGA 上のフロアプランを考慮する有効な技術の 1 つとして，HDR アーキテクチャ[1–3] を紹介した．これは，レジスタ分散型アーキテクチャの 1 つであり，任意の矩形を取るハドルに対しフロアプランするため，レイテンシの小さい回路の生成を目的としたフロアプラン指向 FPGA 高位合成手法に有効であると考えられるが，HDR アーキテクチャを対象とした FPGA 向けの高位合成手法は存在しておらず，フロアプラン指向 FPGA 高位合成手法に採用するためには，新たな高位合成手法の提案が必要となる．

第3章　HDRアーキテクチャを対象としたMUX削減FPGA高位合成手法

3.1　本章の概要

本章では[1]，FPGAを対象とした高位合成における課題1「フロアプランの考慮とMUXのコスト削減の同時実現」を解決するため，フロアプランの考慮とMUXのコスト削減を同時に実現するFPGA向け高位合成手法として，HDRアーキテクチャを対象としたMUX削減FPGA高位合成手法を提案する．具体的には，HDRアーキテクチャを対象としたFPGA高位合成の問題を定式化し問題を定義する．そして，その問題の解決のため，HDRアーキテクチャを対象としたMUX削減FPGA高位合成手法を提案・評価する．

まず，提案手法のアプローチを決定するために，FPGA上の回路構成要素を実装・分析し，MUXが遅延・面積においてボトルネックであることを明らかにする．更に，MUXの入力数を様々に変化させて実装し，入力数に応じたMUXのコストの増加傾向を分析し，提案手法の方針を決定する．

その後，HDRアーキテクチャを対象としたMUX削減FPGA高位合成手法を提案する．提案手法は，配線遅延とMUXのボトルネックを同時に考慮した新しいFPGA向け高位合成手法である．提案手法は，HDRアーキテクチャを採用することで，高位合成段階でフロアプランを扱い，配線遅延の影響を見積る．また，高位合成段階でMUXのコストを削減するため，各FU（演算器）・レジスタ間のデータ転送を考慮した2つのバインディング手法を提案する．

- パス考慮スケジューリング/FUバインディング
- パス考慮レジスタバインディング

パス考慮スケジューリング/FUバインディングでは，配線遅延を含めたデータ転送遅延を考慮しつつ，既にデータパスがある2つのFU同士を積極的に連続する2つの演算ノードに割り当てることでMUX数の削減を図る．パス考慮レジスタバインディングでは，FPGA実装によって導いたMUXの入力数に伴うコスト増加傾向を基に，入出力先のFU・レジスタを考慮してレジスタに付加するMUXを4入力以下に制限する．これによって，回路全体でのレジスタに付随するMUXの遅延・総面積を削減する．最後に，計算機実験により提案手法の有効性を示す．

[1]本章の内容は [23–25, 29] による．

3.2 問題定義

本節では，本論文での対象アーキテクチャを定義し，HDR アーキテクチャを対象とした FPGA 高位合成問題を定義する．

3.2.1 対象アーキテクチャ

本項では，本論文で提案する FPGA 向け高位合成手法の対象アーキテクチャを決定する．

高位合成における対象アーキテクチャは，以下の 2 つに大別され [57]，本論文では，対象アーキテクチャとしてレジスタ分散型アーキテクチャを採用する．

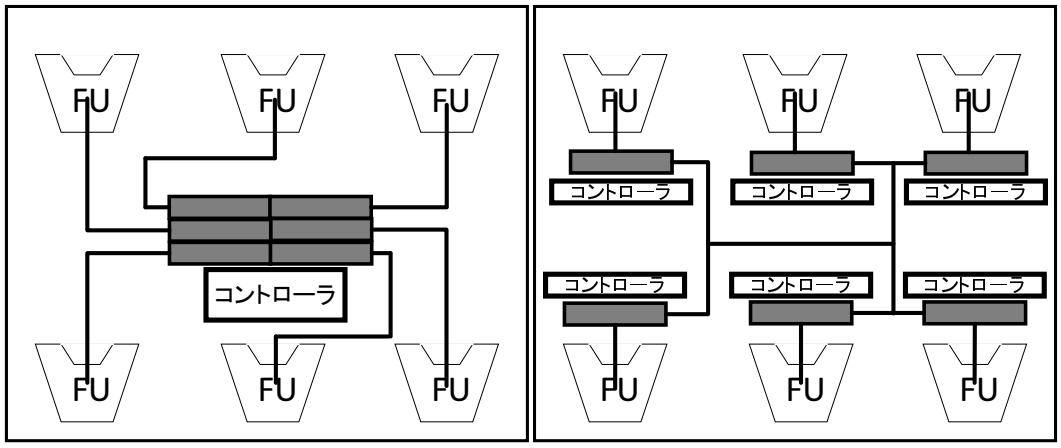
- レジスタ集中型（SR : Shared-Regisuter）アーキテクチャ
- レジスタ分散型（DR : Distributed-Register）アーキテクチャ

図 3.1 に上記 2 つのアーキテクチャを示す．レジスタ集中型アーキテクチャ[57] は，回路全体の FU でレジスタとコントローラを積極的に共有しレジスタ数を少なくするアーキテクチャである．しかし，レジスタ集中型アーキテクチャでは，レジスタと各 FU へ間の距離が異なり，FU・レジスタ間の配線遅延に差が生じる．レジスタ集中型アーキテクチャを対象とした高位合成では，最も大きい配線遅延に合わせてタイミング設計しなければならず，最終的な回路の動作可能クロック周期を大きく見積ることになる．

一方，レジスタ分散型アーキテクチャ[57] は各 FU 毎もしくはいくつかの FU に対して，ローカルレジスタ・ローカルコントローラを持つアーキテクチャである．レジスタ分散型アーキテクチャは，レジスタ集中型アーキテクチャに比べて FU・レジスタ間の配線遅延が小さい．高位合成の対象アーキテクチャをレジスタ分散型アーキテクチャとした場合，高位合成段階で最終的な回路のタイミング設計を予測しやすい．また，FU・レジスタ間の配線遅延の見積りを小さくできるため，最終的な回路の動作可能クロック周期を小さくできる．さらに，FPGA はチップ全体にレジスタが分散されており，よりレジスタ分散型アーキテクチャが適すると考えられる．従って，本論文では，対象アーキテクチャとしてレジスタ分散型アーキテクチャを採用する．

次に，レジスタ分散型アーキテクチャの中から採用する対象アーキテクチャを議論し，本論文で採用する対象アーキテクチャを，HDR アーキテクチャ[1–3] に決定する．本論文のフロアプラン指向 FPGA 高位合成で対象とするレジスタ分散型アーキテクチャとして，第 2.2 節で紹介した RDR（Regular Distributed-Register）アーキテクチャ[12]，第 2.4 節で紹介した GDR（Generalized Distributed Register）アーキテクチャ[58]，HDR（Huddle-based Distributed-Register）アーキテクチャ[1–3] が候補として挙げられる．RDR アーキテクチャは均一な区画に分割されているため，各島間の配線遅延の見積りが非常に容易になり，さらに区画単位のフロアプランは FPGA に適しているという長所がある．しかし，RDR アーキテクチャはチップを一定の大きさに分割するため遅延・面積オーバーヘッドが大きい欠点がある．GDR アーキテクチャは，RDR アーキテクチャに比べて，高速かつ小面積という利点を持つが，第 2.2 節で述べたように，個別モジュール単位のフロアプランは収束性の課題がある．

一方で，HDR アーキテクチャは GDR アーキテクチャにハドルという RDR の区画の概念を導入し，各モジュールを抽象化したアーキテクチャである．ハドル導入によるモジュール



(a) レジスタ共有型アーキテクチャ. (b) レジスタ分散型アーキテクチャ.

図 3.1: 高位合成の対象アーキテクチャ.

の抽象化によって，RDR アーキテクチャと同様に HDR アーキテクチャは FPGA に適した区画ごとのフロアプランを採用している．また，任意の矩形を取るハドルに対しフロアプランするため，GDR アーキテクチャと同様に小面積で高速なアーキテクチャとなる．以上より，現状 HDR アーキテクチャが最もフロアプラン指向 FPGA 高位合成手法に適したレジスタ分散型アーキテクチャと考えられる．従って，本論文では対象アーキテクチャとして，HDR アーキテクチャ[1–3] を採用する．

次に，FPGA 上の HDR アーキテクチャに関して，以下で定義する．HDR アーキテクチャは回路全体をハドル（Huddle）と呼ぶ区画で分割し，ハドル毎にフロアプランを行い，配線遅延を正確に見積もることができる．ハドルは配線遅延の影響の無い範囲で任意の矩形を取り，FU やレジスタ，コントローラを共有する．FPGA 上の HDR アーキテクチャの構成を図 3.2 に示す．本論文では，FPGA 上の HDR アーキテクチャのハドルは以下の要素で構成されるとする．

Huddled Local Register（HLR） 各ハドル専用のローカルレジスタと MUX の集合である．

Huddled Functional Unit（HFU） ハドルに集められた FU の集合である．ハドル内で処理する演算に必要な FU を必要数持ち，同一のハドル内の HLR のみにアクセスできる．

Finite State Machine（FSM） 各ハドル専用のコントローラである．同一ハドル内の HFU と HLR を制御する．

FPGA は論理ブロック，I/O パッド，配線チャネルで構成される．各論理ブロックは複数のスライスで構成され，各スライスは LUT（Look Up Table）とフリップフロップ（FF）で構成される．FPGA 向け HDR アーキテクチャでは，FPGA 上で各ハドルがスライス単位で構成される．HFU は FU と MUX の集合であり，スライスの LUT で構成される．HLR はレジスタと MUX の集合であり，スライスの LUT と FF で構成される．FSM はコントローラ

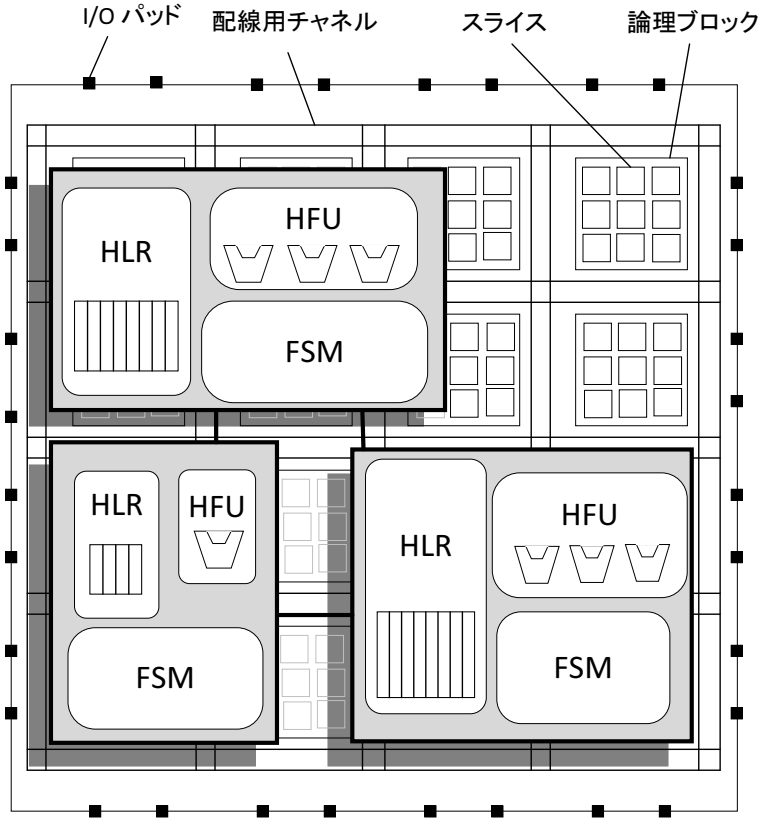


図 3.2: FPGA 上の HDR アーキテクチャの構成.

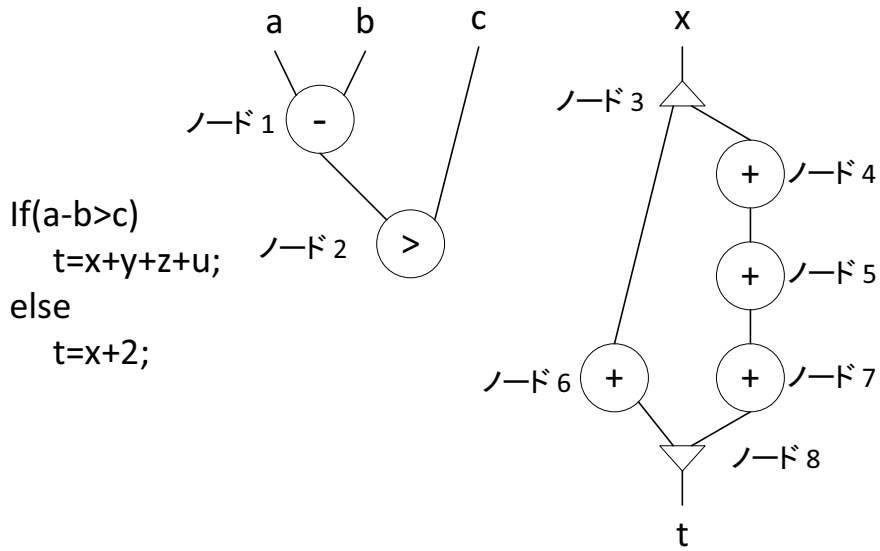


図 3.3: CDFG の例 [69].

であり，スライスの LUT と FF で構成される．各ハドル間は配線チャネルを用いて接続され，外部とのデータの入出力は I/O パッドを介して行われる．

3.2.2 問題の定式化

本稿では，本論文における HDR アーキテクチャを対象とした FPGA 高位合成問題を定式化する．

本論文では，動作記述を有向グラフ $G(N, E)$ に変換したコントロールデータフローグラフ（以下 CDFG）[69] を与える．N は演算ノード N_o と制御ノード N_c からなる．各演算ノード $n_i \in N_o$ は 2 個以下の親ノードを持つとする．E はデータフローエッジ E_d とコントロール依存エッジ E_c からなる．CDFG の例を図 3.3 に示す．図 3.3 において，ノード 1, 2, 4, 5, 6, 7 は演算ノード N_o に含まれ，ノード 3, 8 は制御ノード N_c に含まれる．また，ノード 3 とノード 4 の間のエッジ，ノード 3 とノード 6 の間のエッジ，ノード 6 とノード 8 の間のエッジ，ノード 7 とノード 8 の間のエッジはコントロール依存エッジ E_c に含まれ，それ以外のエッジはデータフローエッジ E_d に含まれる．

各ハドルは，1 個以上の FU を持つ HFU と，HLR と FSM を持つ．ここで，ハドル h_i の HFU の面積を $Area_{FU}(h_i)$，HLR の面積を $Area_{Reg}(h_i)$，FSM の面積を $Area_{FSM}(h_i)$ とする．以上より，ハドル h_i の面積 $Area(h_i)$ は以下の式より求まる．

$$\begin{aligned} Area(h_i) &= W(h_i) \times H(h_i) \\ &= Area_{FU}(h_i) + Area_{Reg}(h_i) + Area_{FSM}(h_i) \end{aligned} \tag{3.1}$$

式 (3.1) において $W(h_i)$ はハドル h_i の幅，$H(h_i)$ はハドル h_i の高さを表す．

FU f_i の遅延[2] を $D_f(f_i)$ で表し，FU f_i の MUX の遅延を $D_{MUX}(f_i)$ で表す．$S_f(f_i)$ は FU f_i の必要ステップ数を表し，クロック周期 T_{clk} より $S_f(f_i) = \lceil (D_f(f_i) + D_{MUX}(f_i))/T_{clk} \rceil$ と計算される．FU f_i を割り当てるハドルを $Hud(f_i)$ で表す．ハドル h_i に割り当てられた FU の集合を $F(h_i)$ で表す．レジスタの遅延を D_{reg} で表す．FU f_i において $T_{clk} \times S_f(f_i)$ より求まる演算時間から演算処理のみに必要な時間を除いた時間を以下の式で表す．

$$Slack(f_i) = T_{clk} \times S_f(f_i) - D_f(f_i) - D_{MUX}(f_i) \geq 0 \tag{3.2}$$

また，ハドル h_i におけるデータ転送に使用できる時間 $Slack(h_i)$ を以下のように定める．

$$Slack(h_i) = \min_{f_i \in F(h_j)} Slack(f_i) \tag{3.3}$$

HDR アーキテクチャの各ハドルの大きさはハドル内の FU が 1 クロック以内に必ず演算と通信が完了する範囲としている．ここで，図 3.4 のように，ハドル h_i のレジスタ r_i，FU f_j がそれぞれハドルの右下，左上に配置された状況を考える．この時，ハドル h_i 内でのレジスタ r_i と FU f_j 間のマンハッタン距離は $W(h_i) + H(h_i)$ である．ここで，ハドル h_i 内における次のようなデータ転送を考える．まず，レジスタ r_i からデータを読み出し，FU f_j へ送る．FU f_j で演算を行った後，データをレジスタ r_i へ送り格納する．この時，以上のようなハドル内での演算は $Slack(h_i)$ 以内に完了しなければならない．$D_w(x)$ は配線長 x スライスのデータ転送に生じる配線遅延とすると，レジスタ r_i と FU f_j 間のデータ転送時間は

[2]FU の遅延とは，その FU が演算 1 回に要する実行時間を表す．

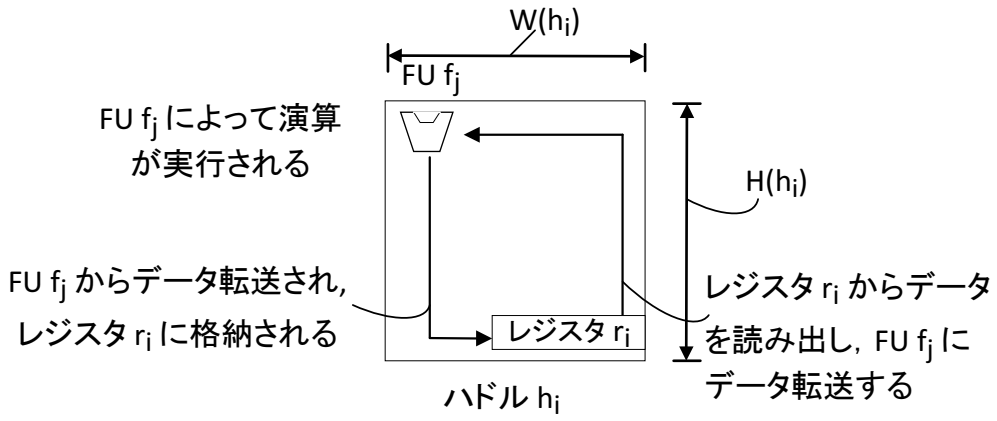


図 3.4: ハドルサイズの例.

$D_w(W(h_i)+H(h_i))$ となる．以上より，$Slack(h_i)$ からハドル h_i のハドルサイズ制約を以下のように定められる．[12]

$$2 \times D_w(W(h_i)+H(h_i)) + D_{reg}(r_i) + D_{MUX}(r_i) \leq Slack(h_i) \tag{3.4}$$

ここで，$D_{reg}(r_i)$ はレジスタ r_i の遅延，$D_{MUX}(r_i)$ はレジスタ r_i の MUX の遅延とする．本章では，FPGA 内の配線にバッファが挿入されているため，配線遅延は距離 x に比例するとし，配線遅延係数 C_d を用いて $D_w(x) = C_d \times x$ で計算する．全てのハドルは式 (3.4) を満たさなければならない.

次に，ハドル間の距離を定義する．ここで，$Dist(h_i, h_j)$ はハドル h_i とハドル h_j 間のスライス単位でのマンハッタン距離を表すとする．さらに，$x(h_i)$ と $y(h_i)$ はハドル h_i の右上の点の x 座標を表すとする．ハドル h_i とハドル h_j 間の距離 $Dist(h_i, h_j)$ は以下の式で求める．

$$Dist(h_i, h_j) = |x(h_i) - x(h_j)| + |y(h_i) - y(h_j)| \tag{3.5}$$

図3.5に2つのハドル間の距離の例を表す．$(x(h_i), y(h_i)) = (20, 20)$，$(x(h_j), y(h_j))$=(100,60) より式 (3.5) に従って，ハドル h_i と h_j の距離 $Dist(h_i, h_j)$ は $|20-100|+|20-60| = 120$ スライスとなる．HDR アーキテクチャでは高位合成段階で FU，レジスタ，コントローラのハドル内の位置を決定しないため，FU やレジスタ間の正確な距離を求めることはできない．本論文では，単純のため各ハドル内の FU，レジスタの位置をハドルの右上の点と扱い，式 (3.5) のようにハドル間の距離をモジュール間の距離と見なす[3].

次に，ハドル間のデータ転送を定義する．ここで，FU $f_i \in F(h_i)$ で生成したデータを FU $f_j \in F(h_j)$ で使用する場合を考える．FU f_i から FU f_j へのデータ転送に要するコントロールステップ（CS）数を id_{f_i,f_j} と表す.

$$Slack(f_i) \geq D_w(Dist(h_i, h_j)) + D_{reg} \tag{3.6}$$

[3]後述の第 3.5 節で，このような距離の評価の下で高位合成し，実際に FPGA 上に配置配線された回路はクロック周期制約を満たしていることから，妥当であると考える.

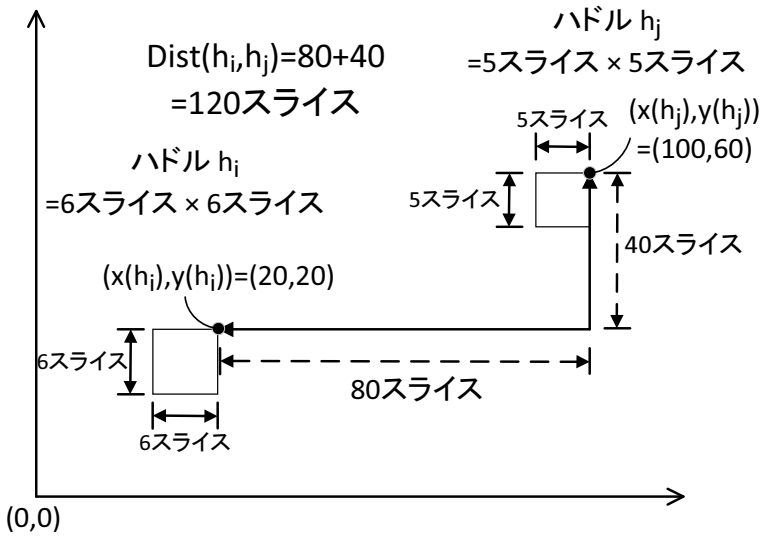


図 3.5: 2 つのハドル間の距離の例.

式 (3.6) が満たされる時，$S_{fu}(f_i)$ ステップ以内に FU f_i の演算およびそのデータをハドル h_j のレジスタに格納できる．従って，この時 $id_{f_i,f_j} = 0$ となる．

$$Slack(f_i) < D_w(Dist(h_i, h_j)) + D_{reg} \tag{3.7}$$

一方，式 (3.7) が満たされる時，$S_{fu}(f_i)$ ステップの間では FU f_i の演算およびそのデータをハドル h_i のレジスタ r_i に格納する．その後，$\lceil D_w(Dist(h_i, h_j))/T_{clk} \rceil$ ステップを要して，レジスタ r_i からハドル h_j のレジスタへ転送する．

HDR アーキテクチャでは，FU f は同一ハドルのレジスタからデータを受け取る．ハドル h_s のレジスタ r_s から FU f を通り，ハドル h_d のレジスタ r_d へデータ転送するパス $path(r_s, r_d)$ の遅延 $D(path(r_s, r_d))$ を以下のように計算する．

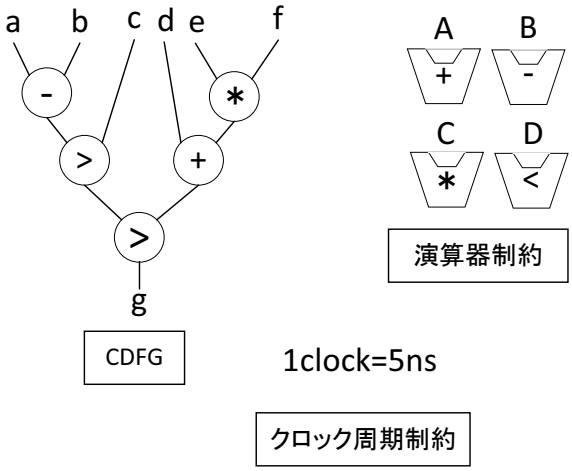
$$D(path(r_s, r_d)) = D_{fu}(f) + D_{mux}(f) + D_{reg}(r_d) + D_{mux}(r_d) + D_w(r_s, r_d) \tag{3.8}$$

レジスタ r_s と r_d 間の配線遅延 $D_w(r_s, r_d)$ は，$D_w(Dist(h_s, h_d))$ と計算する．本論文では，データパス内の全パスのうち，最大の遅延を持つパスをクリティカルパスと定義する．

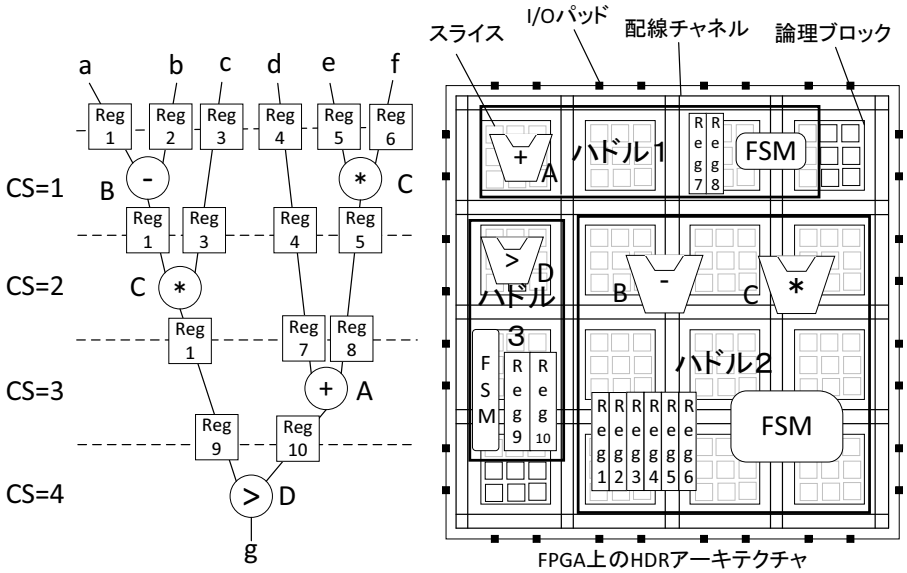
以上より，HDR アーキテクチャを対象とした FPGA 高位合成問題を以下のように定義する．

定義 1 HDR アーキテクチャを対象とした FPGA 高位合成問題とは，CDFG，クロック周期制約，演算器制約が与えられた時，回路のレイテンシを最小化するように CDFG をスケジューリングおよびバインディングし，各 FU をハドルに割り当て，レジスタ・コントローラを合成し，各ハドルを配置することである．レイテンシは 最小クロック周期 × コントロールステップ数（CS 数）で計算する． □

例 1 図 3.6 に HDR アーキテクチャを対象とした FPGA 高位合成問題の例を示す．入力として，CDFG，クロック周期制約，演算器制約を与える．出力として，スケジューリング・バインディング済みの CDFG と FPGA 上の HDR アーキテクチャの回路情報を出力する．FPGA



(a) HDR アーキテクチャを対象とした FPGA 高位合成問題の入力



(b) HDR アーキテクチャを対象とした FPGA 高位合成問題の出力

図 3.6: HDR アーキテクチャを対象とした FPGA 高位合成問題.

上の HDR アーキテクチャの回路情報とは，各ハドル内の FU・レジスタ・コントローラの構成，各ハドルの FPGA 上の位置・大きさ・接続情報である．各ハドルはスライス単位で構成され，ハドル間は配線用チャネルで接続される．外部とのデータの入出力のため，各ハドル内の FU・レジスタ・コントローラは I/O パッドに接続される． □

3.3 動機付け

本節では，前節で定義した問題を解くために，提案手法の方針を決定づけるため，FPGA 上で実験を行う．まず，FPGA 上に実現される回路において面積・遅延のボトルネックとなる構成要素を特定する．この実験より，「MUX の数を削減する.」という 1 つ目の提案手法の方針を導く．次に，FPGA 上で MUX の入力数に対する遅延・LUT 数を測定し，MUX の

表 3.1: FPGA・ASIC の各回路構成要素の遅延相対値と面積相対値.

回路構成要素	FPGA		ASIC	
(FU/MUX/レジスタ)	遅延	LUT 数	遅延	面積
加算器	1.00	1.00	1.00	1.00
減算器	1.00	1.00	1.01	1.21
乗算器	2.84	11.1	2.00	7.15
除算器	19.6	24.1	10.5	9.06
右シフト器	1.47	2.44	0.42	1.02
比較器	1.06	0.38	0.15	0.40
AND	0.55	1.00	0.02	0.24
2 入力 MUX	0.66	1.00	0.03	0.39
レジスタ	0.24	-	0.09	1.08

入力数に応じた遅延・LUT 数の増加傾向を調べる．この実験より,「MUX を 4 入力以下にする.」という 2 つ目の提案手法の方針を導く.

実験 1 (FPGA の回路要素の測定)

表 3.1 に FPGA と ASIC それぞれの加算器の値（遅延・LUT 数・面積）を 1 として，各回路構成要素の相対値を算出した結果を示す．各回路構成要素の入出力は全て 16 ビットとした．実験環境は FPGA に関して，Xilinx 社 ISE Design Suite 14.2 の XST で論理合成した．FPGA ボードは Virtex-6 を想定した．ASIC に関して，Synopsys 社の Design Compiler D-2010.03-SP5 のトポグラフィカルモードを用いて，クロック制約は 5.0ns，ライブラリは STARC 90nm テクノロジを用いて論理合成した.

表 3.1 の結果から FPGA と ASIC における FU の遅延/面積の比較は以下のように求めた．

$$FU \text{ の遅延の比較値} = \frac{\sum(FPGA \text{ の } FU \text{ の遅延})}{\sum(ASIC \text{ の } FU \text{ の遅延})}$$
$$= \frac{1.00 + 1.00 + 1.00 + 2.84 + 19.6 + 1.47 + 1.06 + 0.55}{1.00 + 1.01 + 2.00 + 10.5 + 0.42 + 0.15 + 0.02}$$
$$\approx 1.8 \tag{3.9}$$

$$FU \text{ の } area \text{ の比較値} = \frac{\sum(FPGA \text{ の } FU \text{ の } LUT \text{ 数})}{\sum(ASIC \text{ の } FU \text{ の面積})}$$
$$= \frac{1.00 + 1.00 + 1.00 + 11.1 + 24.1 + 2.44 + 0.38 + 1.00}{1.00 + 1.21 + 7.15 + 9.06 + 1.02 + 0.40 + 0.24}$$
$$\approx 2.0 \tag{3.10}$$

$$MUX \text{ の遅延の比較値} = \frac{FPGA \text{ の } MUX \text{ の遅延}}{ASIC \text{ の } MUX \text{ の遅延}}$$
$$= \frac{0.66}{0.03} \approx 22 \tag{3.11}$$

$$MUX \text{ の面積の比較値} = \frac{FPGA \text{ の } MUX \text{ の } LUT \text{ 数}}{ASIC \text{ の } MUX \text{ の面積}}$$
$$= \frac{1.00}{0.39} \approx 2.6 \tag{3.12}$$

$$\text{レジスタの遅延の比較値} = \frac{FPGA \text{ のレジスタの遅延}}{ASIC \text{ のレジスタの遅延}}$$
$$= \frac{0.24}{0.09} \approx 2.7\ . \tag{3.13}$$

MUX に関して，FPGA では ASIC に比べ面積は約 2.6 倍，遅延は約 22 倍の相対値を示した．FU に関して，FPGA では ASIC に比べて平均で面積は約 1.8 倍，遅延は約 2.0 倍の相対値を示した．レジスタは FF のみから構成され，レジスタの遅延の相対値は 2.7 倍であった．以上の結果より，FPGA の方が ASIC に比べて回路全体に占める MUX の面積・遅延割合が大きくボトルネックである．従って，FPGA では MUX 数を削減する高位合成が必要である．

実験 2 (FPGA の MUX の入力数ごとの測定)

次に，MUX の入力数に対する遅延・LUT 数の測定結果を表 3.2 に示す．実験環境は Xilinx 社の ISE Design Suite 14.2 において XST で論理合成した．入出力は全て 16 ビットとした．Virtex-6 のスライスはそれぞれ 4 つの 6 入力 LUT と 4 つの FF，LUT の出力部にいくつかの 2 入力 MUX（専用 MUX，Dedicated MUX）を持つ．表 3.2 から，2–4 入力の MUX は 16 個の LUT を必要とし，5 入力以上の MUX は 32 個以上の LUT を必要とする．つまり，2–4 入力の MUX は面積が等しく，5 入力以上の MUX の半分以下であることがわかる．また，遅延においても 2–4 入力の MUX はほぼ同等の遅延を持ち，5 入力以上の MUX より小さい．Virtex-6 が 6 入力 LUT を搭載しているため，2–4 入力の MUX の面積が等しく，遅延に関してもほぼ同等である．一方，8 入力の MUX は 5–7 入力の MUX と比べて，小さい遅延・面積を持つ．これは，8 入力の MUX を ISE がより効率的に専用 MUX を用いて合成できるためである．従って，FPGA では 4 入力以下の MUX が遅延・面積において低コストであり，5 入力の MUX の数を削減し，できる限り 4 入力の MUX を作ることが必要である．

表 3.2: FPGA における入力数ごとの MUX の遅延，LUT 数，専用 MUX 数.

MUX	遅延 [ns]	LUT 数	専用 MUX 数 (スライス内の 2 入力 MUX 数)
2 入力 MUX	1.002	16	0
3 入力 MUX	1.184	16	0
4 入力 MUX	1.186	16	0
5 入力 MUX	1.575	32	0
6 入力 MUX	1.650	32	0
7 入力 MUX	1.717	47	1
8 入力 MUX	1.443	33	16
16 入力 MUX	1.611	66	48

以下の例で，FPGA 上の回路における MUX を 4 入力に制限することで，MUX のコストを削減できることを示す.

例 2 MUX の入力数を 4 入力以下に制限する例を図 3.7 に示す．既存手法では 5 入力以上の MUX が付加したレジスタバインディング結果に対し，提案手法ではレジスタを増加させ 4 入力以下の MUX に分割することを考える．図 3.7(a) では，既存手法ではレジスタ 1 つに 5 入力の MUX が付加しているレジスタバインディング結果を想定する．レジスタには a から e の 5 つの変数が割り当てられている．レジスタ数を 1 つ増加させして，計 2 つのレジスタに 5 つの変数を割り当てる．各レジスタに付加する MUX は 3 入力と 2 入力になる．レジスタを増加させ変数の割り当てを変更することで，各レジスタに付加する MUX を 4 入力以下にして同等の機能を保持できる．また，この時 LUT 数の合計は変化しないが，MUX によって生じる遅延は $1.575 - 1.184 = 0.391$ns 削減できる．

5–7 入力の MUX の場合，4 入力以下の MUX の 2 倍以上の LUT 数を必要とする．図 3.7(b) は 6 入力の MUX 2 つの分割例を示す．今，6 入力の MUX に 32 個の LUT を必要とし，4 入力の MUX に 16 個の LUT を必要とする．レジスタ 2 つに 6 入力の MUX 2 つが付加していた場合，レジスタを 3 つに増やし 4 入力の MUX 3 つに分割することで $32 \times 2 - 16 \times 3 = 16$ の LUT を削減できる．さらに，遅延は $1.650 - 1.186 = 0.464$ns 削減できる．

8 入力，16 入力の MUX の場合，4 入力の MUX に比べて，ほぼ入力数に比例した LUT 数を必要とする．つまり，図 3.7(a) のようにレジスタを追加して 8 入力，16 入力の MUX を 4 入力の MUX に分割しても LUT 数の削減は図れない．しかし，8 入力，16 入力の MUX を 4 入力の MUX に分割することで遅延を削減できる．今，8 入力の MUX に 33 個の LUT，遅延を 1.443ns 必要とし，4 入力の MUX に 16 個の LUT，遅延を 1.186ns 必要とする．図 3.7(c) は 8 入力の MUX 1 つの分割例を示す．レジスタ 1 つに 8 入力の MUX が付加していた場合，レジスタを 2 つに増やし 4 入力の MUX 2 つに分割すると，$33 - 16 \times 2 = 1$ と LUT 数はほぼ変わらない．しかし，MUX の遅延は $1.443 - 1.186 = 0.257$ns 削減できる．

このように，既存手法では 5 入力以上の MUX が発生していた場合でも，レジスタの増加を許容することで，MUX を 4 入力以下に制限し，MUX で生じる遅延・LUT 数を削減できることがわかる． □

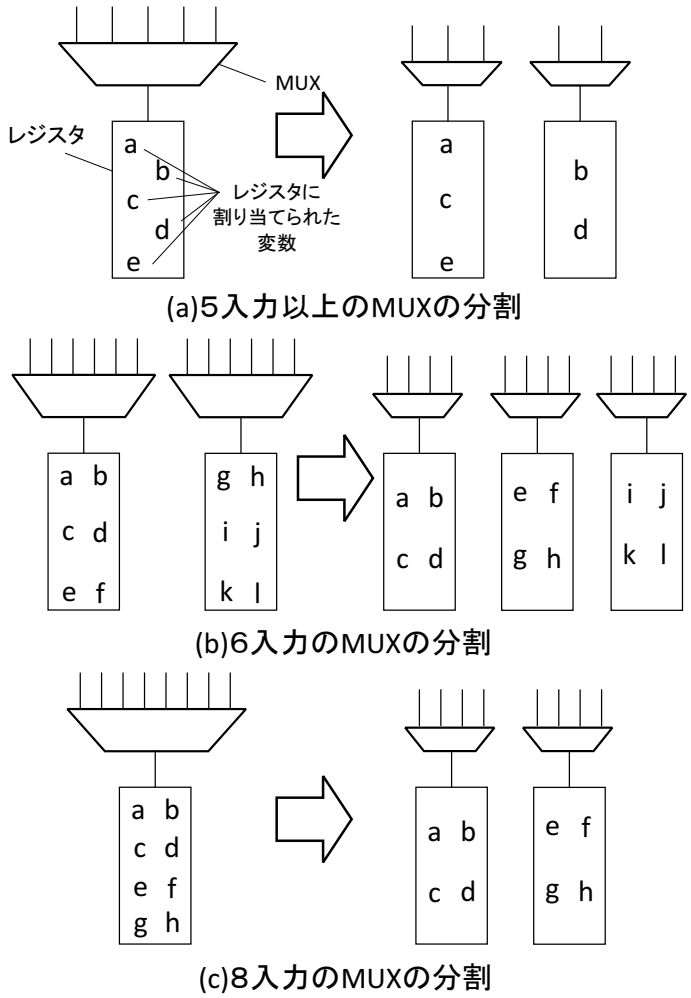


図 3.7: MUX の入力数制限の例.

以上から，FPGA において MUX のコストを削減する高位合成手法において以下のアプローチが有効である．

1. MUX の数を削減する．
2. MUX を 4 入力以下にする．

なお，一般的に他の LSI においても，MUX は MUX 数・入力数が増大するほど遅延・面積が増大する傾向があるため，他の LSI 向けの高位合成手法においても有効なアプローチと考えられる．一方で，上記の FPGA 上での実験結果において，4 入力以下の MUX が同等の遅延・LUT 数であり，5 入力以上の MUX より顕著にコストが小さくなった理由として，Virtex-6 が 6 入力の LUT を搭載しており，4 入力以下の MUX は 6 入力 LUT での構成効率が良いという FPGA アーキテクチャ上の要因が考えられる．従って，上記の MUX のコストを削減する高位合成手法のアプローチは FPGA 設計において，より大きな効果を得られると考えられる．

3.4 提案アルゴリズム

本節では，HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成アルゴリズムを提案する．[2] で提案された HDR アーキテクチャを対象とした高位合成手法 MH^4 に基づいて，提案手法である HDR アーキテクチャを対象とした FPGA 向け高位合成手法を構築する．また，前節で述べた通り，レイテンシの小さい回路を生成するためには，FPGA では MUX のコストが小さいデータパスを生成することが重要である．そのため，提案手法では MUX のコストが小さいデータパスを生成する 2 つの新たなバインディング手法「パス考慮スケジューリング/FU バインディング」,「パス考慮レジスタバインディング」を提案する．以上により構築される提案手法は，配線遅延の考慮と MUX のコストの削減を同時に実現する初の FPGA 向け高位合成手法である．

[2] の手法 MH^4 は，入力として与える各 FU に対しそれぞれハドルを構成し，全てのハドルが重なった状態を初期解とする．このときハドルは 1 つの FU だけを持ち，レジスタやコントローラは持たないとする．初期解をもとにスケジューリング，バインディング，フロアプラン指向ハドル合成を繰り返し，解を改良する．データ転送制約違反のない解を得た場合，最終的な解を出力する．しかし，MH^4 は MUX を大きなコストと考えておらず，バインディングの際に各ハドル毎に FU やレジスタをできる限り共有するようバインディングする．結果として，MUX が増加することが予想される．

HDR アーキテクチャを対象とする FPGA 高位合成として，MUX を大きなコストと考え HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成アルゴリズムを提案する．図 3.8 に HDR アーキテクチャを対象とした MUX 削減高位合成アルゴリズムを示す．提案アルゴリズムでは，入力として CDFG，クロック周期制約，演算器制約を与え，RTL 回路の記述とハドルの構成・配置情報を出力する．まず，ハドルフロアプランでは各演算器に 1 つハドルを用意し，重なった状態を生成する (フェーズ 1)．このハドルを基に，スケジューリング/FU バインディング (フェーズ 2)，レジスタバインディング (フェーズ 3)，コントローラ合成 (フェーズ 4)，フロアプラン指向ハドル合成 (フェーズ 5) を繰り返し，解を改良する．データ転送制約を満たす回路を解として得た場合，実際の面積に調整して結果を出力する (フェーズ 6)．

図 3.9 に提案アルゴリズムの反復処理におけるステップ数とフロアプラン解の関係を示す．i 回目のイタレーションにおいて，フェーズ 2 で得られた CDFG を実行するのに必要なコントロールステップ数を C_i とする．提案手法ではフェーズ 2–5 における反復処理の初期解として，フェーズ 1 でハドルが全て重なりハドル間のデータ転送遅延が 0 である状態を生成する．反復処理の各イタレーションでは，初期解あるいは前のイタレーションでのフロアプラン結果に対し，(フェーズ 2) ステップ数の最小化を図るスケジューリング/FU バインディング，(フェーズ 3) レジスタバインディング，(フェーズ 4) コントローラ合成を行いハドルの構成を決定する．このハドルの構成を基に，(フェーズ 5) ハドルの構成の変更・フロアプランを行う．フェーズ 5 において提案手法は [2] と同様に，timing violation について最適化されたフロアプラン解を得る．ここで，フェーズ 2 で得たスケジューリング結果に対し timing violation が発生するということは，当該イタレーションのフェーズ 2 で得たステップ数を実現するハドルの構成・フロアプランは存在しないと考えられる．そして，次のイタレーショ

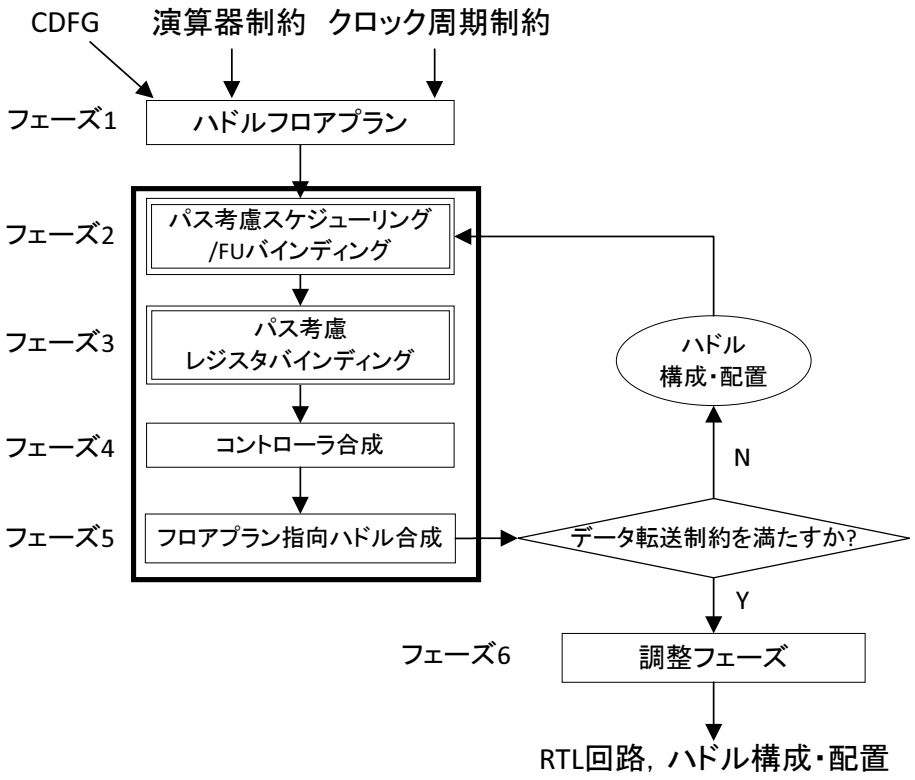


図 3.8: HDR アーキテクチャを対象とした MUX 削減高位合成アルゴリズム.

ンでは前のイタレーションで得たフロアプラン解を入力とするので，次のイタレーションのスケジューリング解は前のイタレーションで得たスケジューリング解よりステップ数が増加することが予想される．従って，図 3.9 の各イタレーションで得られたステップ数にはおおよそ $C_1 \leq C_2 \leq \cdots \leq C_n$ が言える [4].

提案手法はハドル間のデータ転送遅延が 0 でありステップ数の小さい状態を初期解としている．初期解ではデータ転送時間が 0 であるため，多くのタイミング違反があるが，反復処理を繰り返す中で，徐々にタイミング違反を解消していく．そして，そのイタレーションでのスケジューリング解に対してタイミング違反が無いフロアプラン解を得た時，反復処理が終了する．この時得られるスケジューリング解およびフロアプラン解はタイミング違反が無い最小のステップ数にできる限り近い解と期待できる．従って，提案手法はレイテンシが小さい解を見出せると考えられる．

提案アルゴリズムにおいて，回路の MUX のコストは FU バインディング (フェーズ 2) とレジスタバインディング (フェーズ 3) 結果に起因する．各 FU やレジスタ間のデータ転送を考慮し MUX のコストを削減する 2 つのバインディング手法を提案する．

- パス考慮スケジューリング/FU バインディング（フェーズ 2）
- パス考慮レジスタバインディング（フェーズ 3)

パス考慮スケジューリング/FU バインディングでは，すでにデータパスがある 2 つの FU 同士を積極的に連続する 2 つの演算ノードに割り当てることで MUX 数の削減を図る．パス考

[4]Simulated Annealing 手法を用いて確率的にフロアプラン解を求めているため，厳密には $C_1 \leq C_2 \leq \cdots \leq C_n$ が成立するとは限らないが，おおよそこのような性質があると言える．

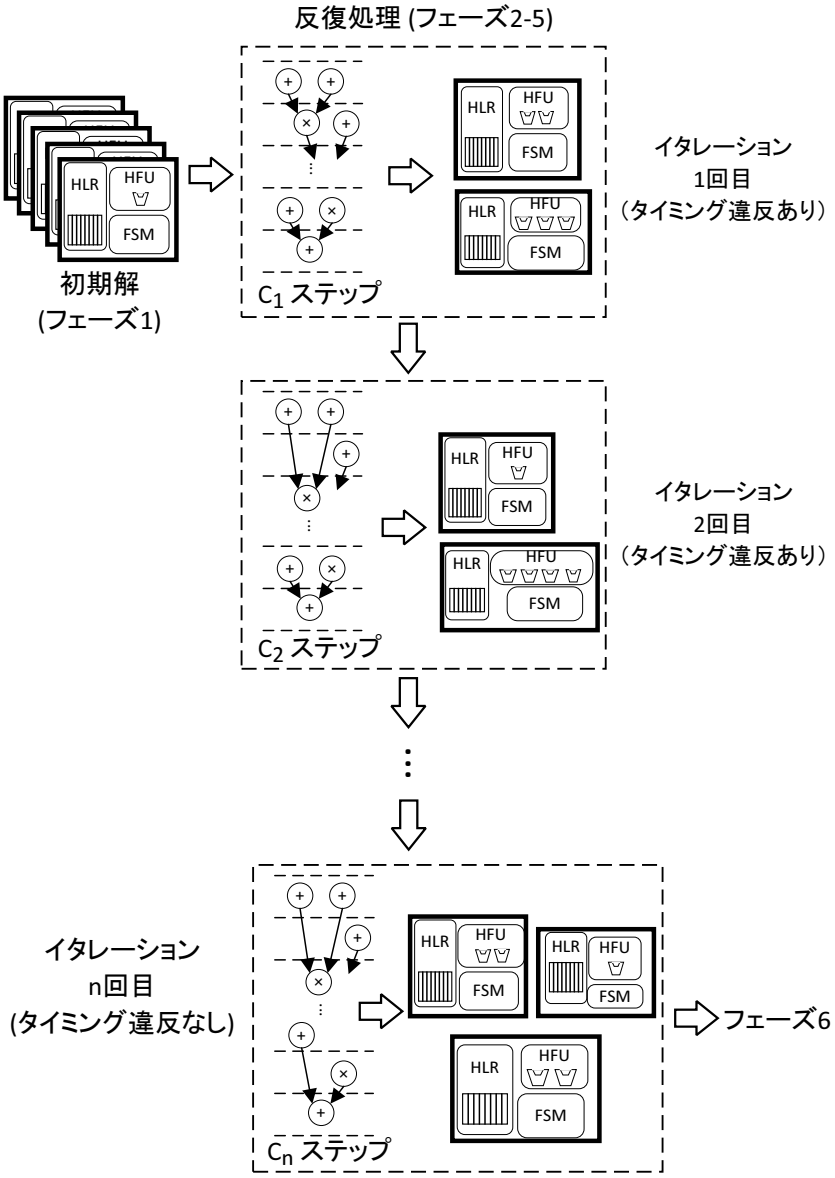


図 3.9: 提案アルゴリズムの反復処理.

慮レジスタバインディングでは，入出力先の FU・レジスタを考慮してレジスタに付加する MUX を 4 入力以下に制限することで MUX の総面積・遅延を削減する．また，パス考慮スケジューリング/FU バインディングでは，前のイタレーションでのハドル構成・配置を基にスライス単位で距離を扱いハドル間のデータ転送遅延を見積る．このデータ転送遅延を考慮してスケジューリング/FU バインディングすることで，最終的な回路の配線遅延を考慮したスケジューリング/FU バインディングを実現している．

フェーズ 1，4，5，6 は [2] と同様である．本節では，新たに提案するパス考慮スケジューリング/FU バインディング（フェーズ 2）とパス考慮レジスタバインディング（フェーズ 3）を議論する．

3.4.1 パス考慮スケジューリング/FU バインディング

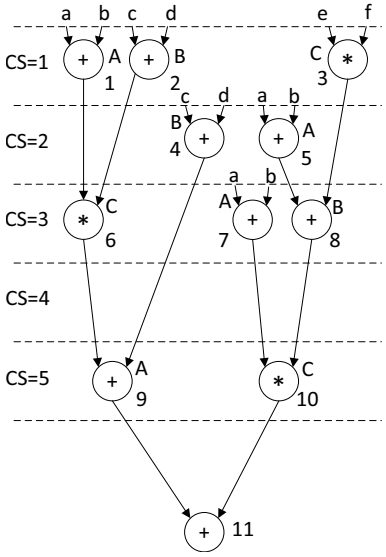
本稿では,（フェーズ 2）パス考慮スケジューリング/FU バインディングのアルゴリズムを例を用いて議論する．まず，以下の例にて FU バインディングにてデータ転送遅延を考慮した上で，合成済みのデータパスを再利用することにより，MUX の入力数を削減できることを示す．その後，パス考慮スケジューリング/FU バインディングのアルゴリズムを図示し，まとめる．

例 3 今，図 3.10 に示すようなスケジューリング/FU バインディングを考える．レジスタとコントローラは提案アルゴリズムの後のフェーズで合成されるため，ここでは考えない．図 3.10(a) は，入力となる CDFG を表す．図 3.10(b) は，スケジューリング/FU バインディングの入力となるハドルの構成・配置情報を表す．FU A, B はハドル 1，FU C はハドル 2 にフロアプランされているとする．ハドル間のデータ転送には 1 CS かかるとする．なお，このハドルの構成・配置情報は 1 つ前のイタレーションで決定したもので，このイタレーションではこの情報をもとに処理を進める．図 3.10(a) のようにコントロールステップ $CS = 5$ までスケジューリング・FU バインディングが完了し，次にノード 11 のスケジューリング/FU バインディングを考える．

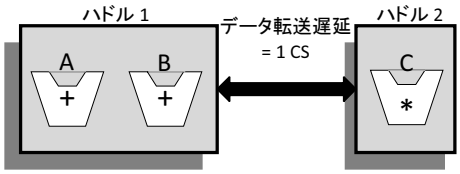
ノード 11 のスケジューリングと FU バインディングを考える．ノード 11 は加算ノードであるので，FU A–C の内，加算器の FU A と B がノード 11 に割り当てられる FU の候補になる．ここで，FU A と B は共にハドル 1 に配置されおり，ハドル 2 に配置されている FU C からのデータ転送に 1 CS かかる．よって，ノード 11 を FU A と B どちらに割り当てたとしても，ノード 11 を $CS = 6$ には割り当てられず，ノード 11 は $CS = 7$ に割り当てられる．図 3.10(c) は，ノード 11 に FU A を割り当てた時のデータパスを表す．ここではレジスタについて議論しないため，各データ転送に 1 つレジスタを設けている．図 3.10(d) は，ノード 11 に FU B を割り当てた時のデータパスを表す.

図 3.10(a) で，ノード 5，ノード 3，ノード 8 に注目する．これらのノードはすでにスケジューリング，FU バインディングが完了しており，ノード 5 は FU A に，ノード 3 は FU C に，ノード 8 は FU B に割り当てられている．そのため，FU A から FU B の左入力に至るデータパス，FU C から FU B の右入力に至るデータパスがすでに存在する．ノード 11 の FU バインディングにおいて FU B をノード 11 に割り当てると，すでに合成済みのデータパスを再利用できるため，図 3.10(d) のように FU B に付加する MUX の入力数が増えない．結果として図 3.10(d) では，図 3.10(c) に比べて MUX の入力数を削減している．FU バインディングではハドルによるデータ転送遅延を考慮した上で，合成済みのデータパスを再利用することにより，MUX の入力数を削減できることがわかる． □

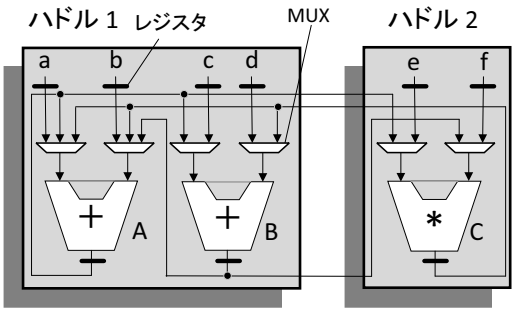
図 3.11 にパス考慮スケジューリング/FU バインディングのアルゴリズムを示す．提案手法は，ハドルによるデータ転送遅延を考慮した上で，合成済みのデータパスを再利用することにより MUX 数の削減を図る．提案手法では，入力として CDFG，演算器制約，クロック周期制約，ハドルの構成，配置情報をもとにデータ転送遅延情報を与えて，演算ノードを実行するコントロールステップ，実行する FU を出力する．ここで，ハドルの構成・配置情報は 1 つ前のイタレーションで得られたものであり，パス考慮スケジューリング/FU バインディ



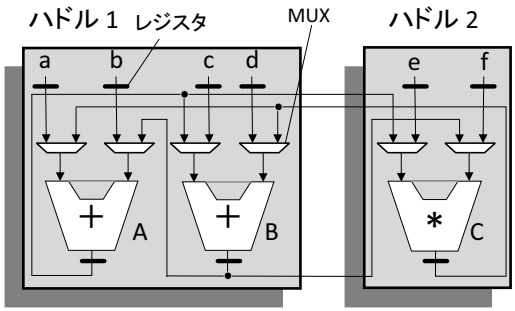
(a) 入力の CDFG.



(b) ハドルの構成・配置情報.



(c) ノード 11 に FU A を割り当てた場合.



(d) ノード 11 に FU B を割り当てた場合.

図 3.10: スケジューリング/FU バインディング例.

ングでは，各 FU が配置されるハドル・ハドルの位置はこれらを流用する．各ハドル内のレジスタとコントローラは後のフェーズで再合成するため，パス考慮スケジューリング/FU バインディングでは考えない．

提案手法では，スケジューリングと FU バインディングを同時に実行する．各ノードの FU を決定する時に，出力先のノードが割り当てられる FU は未決定のため，親ノードに割り当てられた FU のみを考慮して，FU バインディングを行う．提案手法では，FU を選択する際に，親ノードに割り当てられた FU からデータパスがある，かつ最も早い CS に割り当てられる FU を優先的に選択する．親ノードに割り当てられた FU からデータパスがある，かつ最も早い CS に割り当てられる FU が無い場合は，割り当てた際の最終的な全体の CS 数の見積りが最小の FU を選択する．

図 3.11 にパス考慮スケジューリング/FU バインディングアルゴリズムを示し，以下で各ステップを提案する．

Step 1（CS の初期化）

$CS = 0$ よりパス考慮スケジューリング/FU バインディングを実行する．

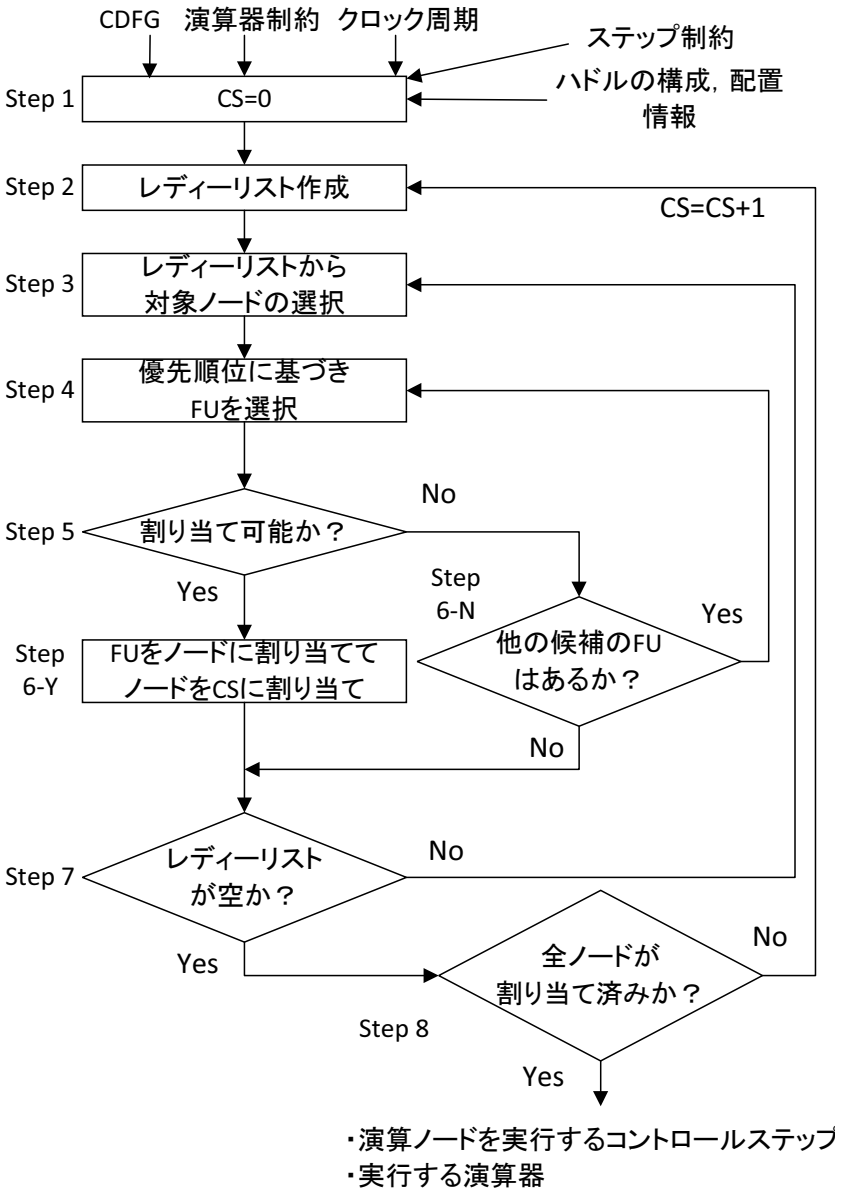


図 3.11: パス考慮スケジューリング/FU バインディング.

Step 2（レディーリスト作成）

$CS = n$ におけるノードのスケジューリングを考える．$CS = n - 1$ までは，リストスケジューリングベースで $CS = n - 1$ までに割り当て可能なノードはスケジューリングされ，スケジューリングされたノード全てにいずれかの FU が割り当てられていると仮定する．データ依存関係を違反しない範囲で，$CS = n$ に割り当てられる可能性のあるノードのリストであるレディーリストを作成する．

Step 3（対象ノードの選択）

レディーリストから優先度 $priority(n_i)$ に従ってノード n_i を選択する．ノード n_i における優先度 $priority(n_i)$ は各ノード n_i において，終端ノードまでのクリティカルパス長を算出

し，クリティカルパス長が長いノードに高い優先度を与える．

$$cp(n_i, fj) = c_j + \max_{n_k \in succ(n_i)} \{\min_{f_l \in F}(id_{f_i,f_l} + cp(n_k, f_l)\} \tag{3.14}$$

$$priority(n_i) = \min_{k \in F} cp(n_i, f_k) \tag{3.15}$$

F は入力で与えられた FU の集合を表す．$cp(n_i, f_j)$ はノード n_i に FU f_j を割り当てた場合の入力の CDFG の終端ノードからノード n_i までのクリティカルパス長を計算したものである．c_j は FU f_j の演算サイクル数，$succ(n_i)$ はノード n_i のすべての子孫ノード，id_{f_j,f_l} は FU f_j から FU f_l へのデータ転送に必要なステップ数を表す．

Step 4–7（FU の割り当て処理）

選択されたノード n_i に対し，割り当てる FU を決定する．入力の CDFG では，各演算ノードの親ノードの数は 2 個以下である．従って，親ノードが 2 個，1 個，0 個の場合を考える．

親ノードが 2 個の場合を考える．ノード n_i の親ノードを n_j と n_k とする．ノード n_j には FU f_j，ノード n_k には FU f_k が割り当てられているとする．FU f_j と f_k からすでにデータパスがある，かつノード n_i の演算に対応する FU の集合を F^1_{path} とする．FU $f_l \in F^1_{path}$ をノード n_i に割り当てて $CS = n$ にスケジューリングする場合，FU f_j と f_k から FU f_l へのデータ転送遅延を満たすか調べる．FU f_j と f_k から FU f_l へのデータ転送遅延を満たす場合，ノード n_i において FU f_l は，新たな MUX を付加せず最も早い CS にスケジューリングできる FU である．FU f_j と f_k から FU f_l へのデータ転送遅延を満たす場合は，FU f_l をノード n_i に割り当て，ノード n_i を $CS = n$ にスケジューリングする．FU f_j と f_k から FU f_l へのデータ転送遅延を満たさない場合は，F^1_{path} の他の FU を同様に調べる．

F^1_{path} の FU 全てが $CS = n$ においてノード n_i に割り当て不可能な場合，FU f_j あるいは f_k のいずれかからすでにデータパスがある，かつノード n_i の演算の種類に対応する FU の集合を F^2_{path} とする．FU $f_m \in F^2_{path}$ をノード n_i に割り当てて $CS = n$ にスケジューリングする場合，FU f_j と f_k から FU f_l へのデータ転送遅延を満たすか調べる．FU f_j と f_k から FU f_m へのデータ転送遅延を満たす場合，ノード n_i において FU f_m は，右または左の入力に新たな MUX を付加せず最も早い CS にスケジューリングできる FU である．FU f_j と f_k から FU f_m へのデータ転送遅延を満たす場合は，FU f_m をノード n_i に割り当て，ノード n_i を $CS = n$ にスケジューリングする．FU f_j と f_k から FU f_m へのデータ転送遅延を満たさない場合は，F^2_{path} の他の FU を同様に調べる．

F^2_{path} の FU 全てが $CS = n$ においてノード n_i に割り当て不可能な場合，ノード n_i の演算の種類に対応する，かつ見積りレイテンシが最小となる FU f_n を選択する．見積りレイテンシとは，あるノード n_i に FU f_n を割り当てた際の最終的に必要となる CS 数の見積り値である．ノード n_i における FU f_n のレイテンシ見積り $lat(n_i, f_n)$ を以下のように定める．

$$lat(n_i, f_n) = cstep(n_i, f_n) + cp(n_i, f_n) \tag{3.16}$$

$cstep(n_i, f_n)$ はノード n_i に FU f_n が割り当て可能になるコントロールステップを表す．見積りレイテンシにより選択された FU f_n をノード n_i に割り当てて $CS = n$ にスケジューリン

グした場合，入力先の FU f_j と f_k から FU f_n へのデータ転送遅延を満たすか調べる．FU f_j と f_k から FU f_n へのデータ転送遅延を満たす場合，FU f_n をノード n_i に割り当て，ノード n_i を $CS = n$ にスケジューリングする．

FU f_j と f_k から FU f_n へのデータ転送遅延を満たさない場合，ノード n_i は $CS = n$ にスケジューリングできないと判断し，レディーリストの次のノードを選択する．

親ノードが 1 個と 0 個の場合も，親ノードが 2 個の場合と同様である．

Step 8（終了判定）

レディーリストが空になった場合，$CS = n$ にスケジューリング可能なノード全てをスケジューリングしたと判断し，$CS = n + 1$ として同様の処理を行う．全てのノードをスケジューリングし終えたら，スケジューリング結果を出力する．

3.4.2 パス考慮レジスタバインディング

本稿では,（フェーズ 3）パス考慮レジスタバインディングのアルゴリズムを示し，例を用いて議論する．まず，レジスタバインディングにおける目的を定義する．その後，パス考慮レジスタバインディングのアルゴリズムを図示し，例を用いて議論する．

ここでは，フェーズ 3 におけるレジスタバインディングの目的を定義する．パス考慮スケジューリング/FU バインディングによって，各ハドルの FU で実行される演算ノードが決定された．この時点でハドル内の構成は FU のみ決定している．この後，各ハドルで変数をレジスタに割り当てる．レジスタバインディングの入力として，スケジューリング/FU バインディング済み CDFG を与える．各ハドル毎にレジスタバインディングを行う．

レジスタバインディングの目的は以下の 2 つである．

(1) 各変数をレジスタに割り当てる．

(2) ライフタイムが排他的な変数を同じレジスタに共有する．

パス考慮レジスタバインディングでは，上記 (1)，(2) に加えて，以下の (3) を目的とし MUX の総面積・遅延の削減を図る．

(3) 各レジスタの入出力先の FU・レジスタを考慮して，レジスタの統合を行い，レジスタに付属する MUX の入力数を 4 入力以下に制限する．

FU f_s で生成され，FU f_t へ伝送される変数 v_i のレジスタバインディングを考える．今，いくつかの変数のレジスタバインディングが完了しているものとする．FU f_s と FU f_t は同一ハドル内に配置されているものとする．既に変数のバインディングされているレジスタ r_i にある変数 v_i をバインディングするとき，レジスタ r_i と FU f_s を接続するデータパスが無い場合，レジスタ r_i に新たな MUX が必要となる．また，レジスタ r_i と FU f_t を接続するデータパスがない場合，FU f_t に新たな MUX が必要となる．提案手法では，[39, 40] と同様に，レジスタより MUX を大きなコストと考える．レジスタの集合 R の内，すでに FU f_s と接続しているレジスタの集合を R_{in}，FU f_t と接続しているレジスタの集合を R_{out} とする．まず $r_j \in R_{in} \cap R_{out}$ となるレジスタ r_j に変数 v_i を優先的に割り当てる．このようなレジスタ r_j に変数 v_i を割り当てた場合，新たな MUX は付加されない．$R_{in} \cap R_{out} = \phi$ の場合，$r_j \in R_{in}$ を満たすレジスタ r_j，$r_j \in R_{out}$ を満たすレジスタ r_j の順に優先して変数 v_i を割り当てる．

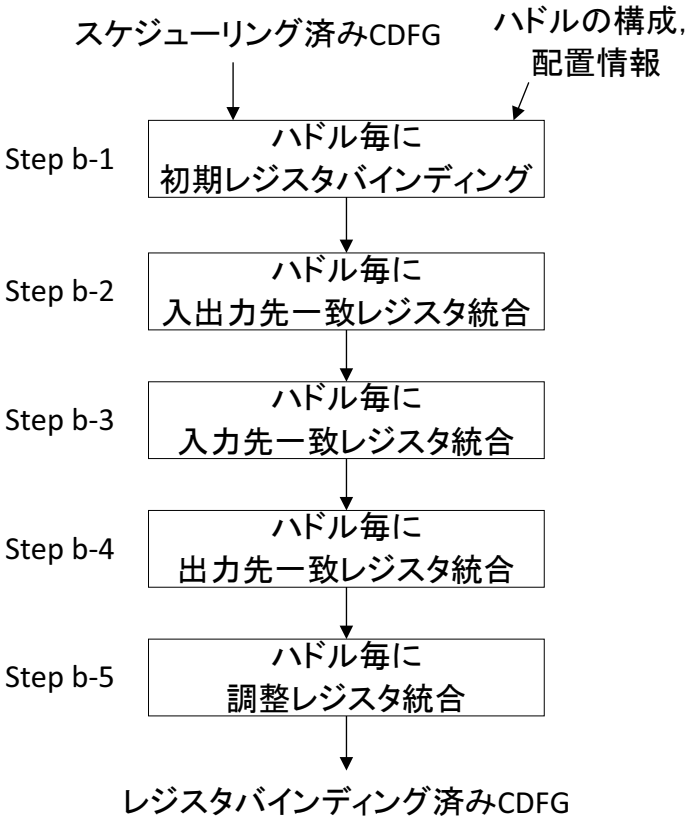
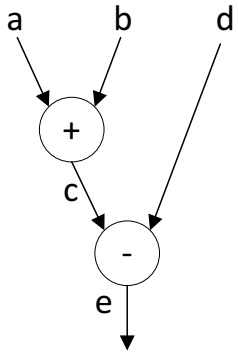


図 3.12: レジスタバインディングアルゴリズム．

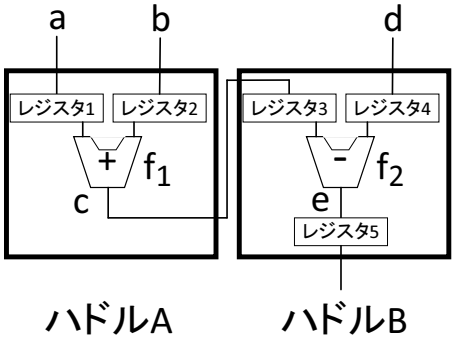
次に，提案手法となるパス考慮レジスタバインディングのアルゴリズムを議論する．提案手法では，1 変数に 1 つレジスタを割り当てた状態を初期解とし，レジスタに付加する MUX の入力数が 4 入力以下になる範囲でレジスタを統合し，レジスタ数・MUX 数を削減していく．

図 3.12 にパス考慮レジスタバインディングアルゴリズムを示す．提案手法では，入力としてスケジューリング済み CDFG，ハドルの構成・配置情報を与え，レジスタバインディング済み CDFG を出力する．提案手法では，レジスタバインディングを各ハドル毎に行う．各ステップでは，ハドル毎に処理を行い全ハドルに対し同様の処理をした後，次のステップへ進む．提案手法は以下の 5 つのステップで構成される．

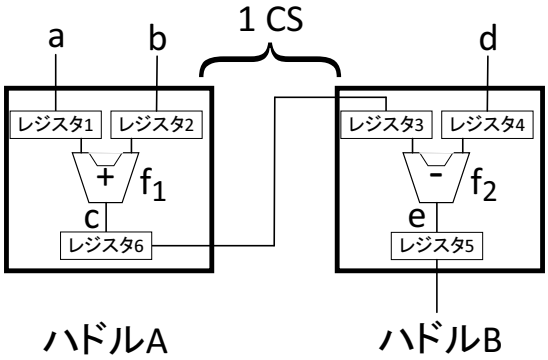
Step b-1 初期レジスタバインディングは，各変数に 1 つレジスタを用意する．初期レジスタバインディングは，1 変数に 1 つレジスタを割り当てた状態である．式 (3.6)，(3.7) に従って，レジスタ間通信する変数は変数の転送先の FU のハドルに 1 つ，変数の転送元の FU のハドルに 1 つレジスタを設ける．図 3.13 に各変数への初期レジスタの割り当て例を示す．図 3.13(a) のような DFG を入力し，FU f_1 がハドル A に，FU f_2 がハドル B に割り当てられているとする．まずハドル内通信の変数 a, b, d, e それぞれにレジスタを 1 つずつ用意する（図 3.13(b)–(e) のレジスタ 1，2，4，5）．次に，ハドル間のデータ転送で使用される変数 c へのレジスタの割り当てを考える．FU f_1 のスラック時間 $Slack(f_1)$ が式 (3.6) を満たす時，図 3.13(b)，(d) のようにハドル B に変数 c のためのレジスタを用意する．$Slack(f_1)$ が式 (3.6) を満たさない時，図 3.13(c)，(e) のようにハドル A と B それぞれにレジスタを 1 つずつ用意



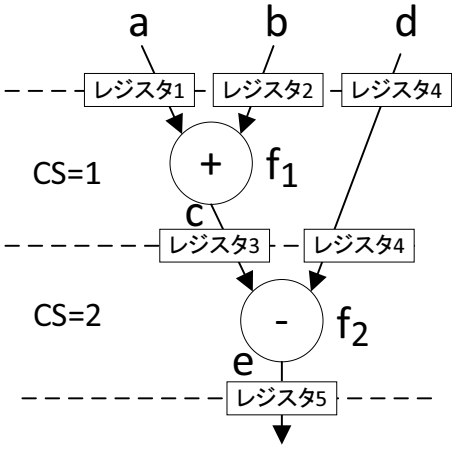
(a) CDFG の例.



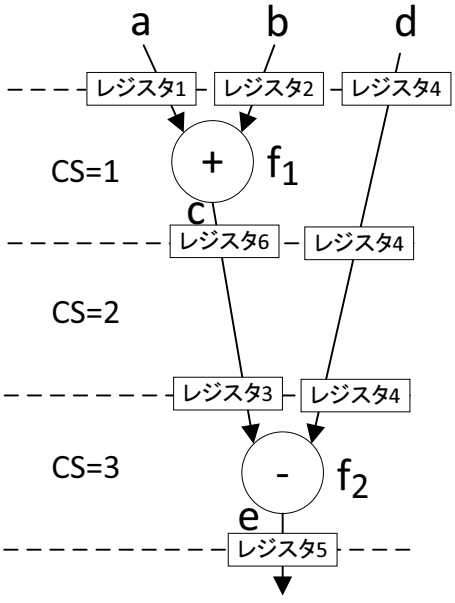
(b) 式 (3.6) を満たすデータパス.



(c) 式 (3.6) を満たさないデータパス.



(d) (b) に対しスケジューリングした CDFG.



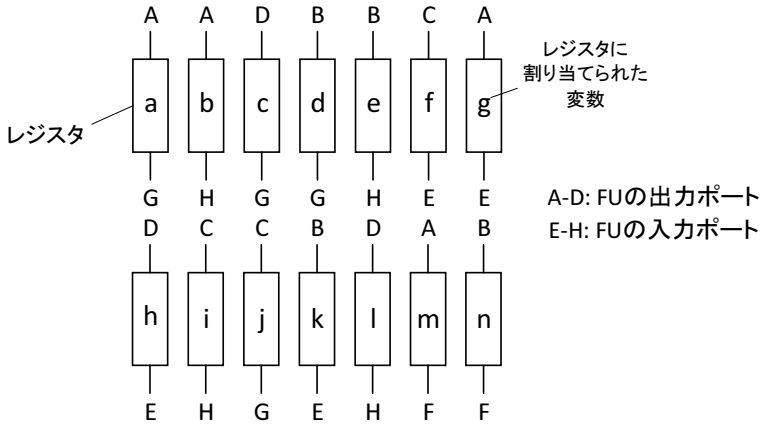
(e) (c) に対しスケジューリングした CDFG.

図 3.13: 各変数への初期レジスタの割り当て例.

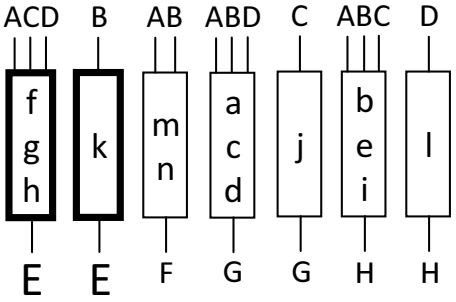
し，レジスタ-レジスタ間転送を行う.

初期レジスタバインディングで用意したレジスタの集合を R とする．$r_i \in R$ の入力先の FU・レジスタの集合を $M_{in}(r_i)$，出力先の FU・レジスタの集合を $M_{out}(r_i)$ とする [5].

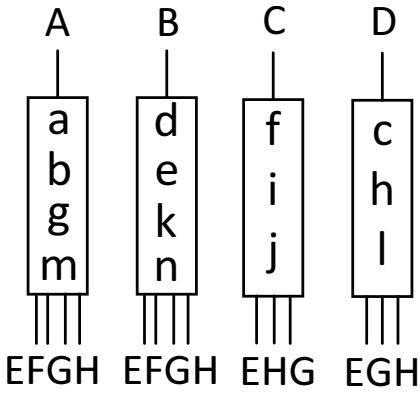
[5]本論文では FU は 1 入力または 2 入力で，1 出力を想定している．$M_{out}(r_i)$ では FU の右入力・左入力を



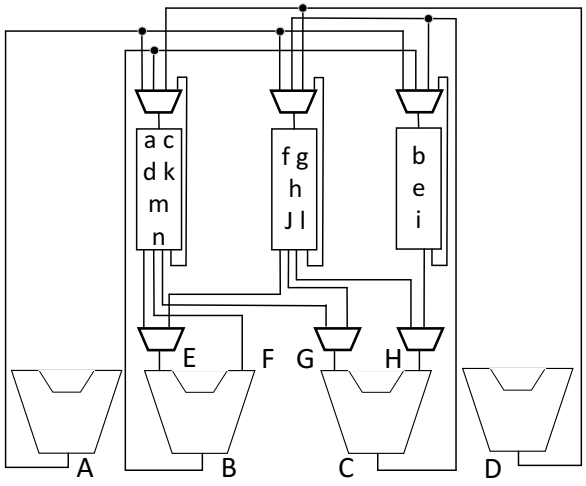
(a) 初期解.



(b) (Step b-4) を先に実行した直後の状態.

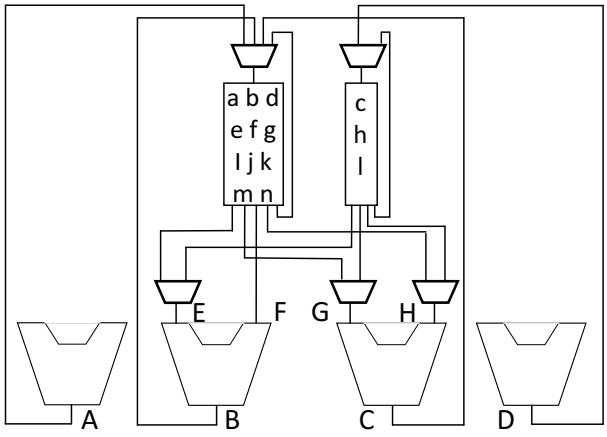


(c) (Step b-3) を先に実行した直後の状態.



レジスタに付随するMUX数 : 3個
FUの入力ポートに付随するMUX数 : 3個
合計 : 6個

(d) (Step b-4)，(Step b-3) の順で実行した場合の最終結果.



レジスタに付随するMUX数 : 2個
FUの入力ポートに付随するMUX数 : 3個
合計 : 5個

(e) (Step b-3)，(Step b-4) の順で実行した場合の最終結果.

図 3.14: レジスタ統合の順序の例.

区別する.

Step b-2 入出力先一致レジスタ統合では，$M_{in}(r_i) \cap M_{in}(r_j) \neq \phi$ かつ $M_{out}(r_i) \cap M_{out}(r_j) \neq \phi$ を満たすレジスタ r_i とレジスタ r_j をについて以下の条件 1，条件 2 を満足するとき統合する．

条件 1 r_i と r_j が同一ハドルに割り当てられている．

条件 2 r_i と r_j に割り当てられている変数のライフタイムに重なりがない．

条件 3 統合後のレジスタ r_k において $Count(M_{in}(r_k)) \leqq 3$ となる．$Count(M_{in}(r_k))$ は $M_{in}(r_k)$ の要素の数を表す．

この時，統合するレジスタと出力先の FU・レジスタに付加された MUX，レジスタ数を削減できる．

Step b-3 入力先一致レジスタ統合では，$M_{in}(r_i) \cap M_{in}(r_j) \neq \phi$ を満たすレジスタ r_i とレジスタ r_j を条件 1，2 を満足する場合，統合する．この時，統合されるレジスタに付加された MUX，レジスタ数を削減できる．

Step b-4 出力先一致レジスタ統合では，$M_{out}(r_i) \cap M_{out}(r_j) \neq \phi$ を満たすレジスタ r_i とレジスタ r_j を条件 1，2 を満足する場合，統合する．この時，出力先の FU・レジスタに付加された MUX は削減し，統合後のレジスタ r_k では MUX の入力数が増えるが 4 入力以下なので，MUX の大きさは変わらない．統合したためレジスタ数が削減できる．

Step b-5 調整レジスタ統合では，残りのレジスタを調べて条件 1，2 を満足するレジスタを統合する．この時，統合後のレジスタ r_k では MUX の入力数が増えるが 4 入力以下なので，MUX の大きさは変わらない．統合したためレジスタ数が削減できる．

データパス考慮レジスタバインディングでは，(Step b-3) 入力先一致レジスタ統合，(Step b-4) 出力先一致レジスタ統合の順に行う．もし，(Step b-4)，(Step b-3) の順に実行した場合，MUX のコストが増大する可能性がある．図 3.14 にレジスタ統合の順序の例を示す．図 3.14(a) のような初期レジスタバインディングされた状況を想定する．各レジスタの上のアルファベットはソース側 FU の出力ポート，下のアルファベットはシンク側 FU の入力ポートを表す．単純のため，この例では全ての変数のライフタイムが重なっていないとする．始め，図 3.14(a) のように 14 個のレジスタがある．データパス考慮レジスタバインディングでは，入力線が自身の戻り線を含め 4 本以下となるようレジスタを統合する．もし，(Step b-4)，(Step b-3) の順に実行した場合，(Step b-4) の直後は図 3.14(b) のようになる．この時，シンク側が E であるレジスタ（図 3.14(b) の太線のレジスタ）を統合することができない．その結果，最終的なレジスタバインディング結果が図 3.14(d) のようになり，レジスタ数が 3 個で MUX 数が 6 個となる．一方，(Step b-3)，(Step b-4) の順に実行した場合，図 3.14(c) のように入力先が同じ変数を全て統合することができる．最終的なレジスタバインディング結果が図 3.14(e) のようになり，レジスタ数が 2 個で MUX 数が 5 個となる．この例が示すように，パス考慮レジスタバインディングでは，(Step b-3) 入力先一致レジスタ統合，(Step b-4) 出力先一致レジスタ統合の順に行う．

次に，上記で示したパス考慮レジスタバインディングのアルゴリズムを例を用いて議論する．**例 4** レジスタバインディングの例を図 3.15 に示す．図 3.15(a) は初期レジスタバインディ

ングした状態を表す．各レジスタの中央のアルファベットは変数，上のアルファベットは各レジスタの入力先の FU・レジスタ，下のアルファベットは各レジスタの出力先の FU・レジスタ，各レジスタ上部の数字はそのレジスタへの他の FU・レジスタからの入力線の数を表す．簡単のため各変数のライフタイムは考えない．

Step b-1 初期レジスタバインディングでは，各ハドルごとに 1 変数につき 1 つのレジスタを用意し，割り当てる．レジスタ間通信する変数は出力先 FU のハドルに 1 つ，入力先 FU のハドルに 1 つレジスタを設ける．

Step b-2 初期レジスタバインディング後，各レジスタと全ての FU・レジスタのデータパスの有無の情報をもとに，入力側・出力側共に同じ FU・レジスタとデータパスのある同じハドル内のレジスタを条件 1，2 を満たす場合，統合していく（図 3.15(b)）．

Step b-3 全てのハドルについて，入力先・出力先が一致するレジスタを統合した後，入力先の FU・レジスタのみが一致するレジスタを条件 1，2 を満たす場合，統合する（図 3.15(c)）．

Step b-4 出力先の FU・レジスタのみ一致するレジスタを条件 1，2 を満たす場合，統合する（図 3.15(d)）．レジスタを統合する際，統合後のレジスタへの他の FU・レジスタからの入力線が 3 本以下になるようにレジスタを統合する．入力線を 3 本以下とするのは，各レジスタには自身の値を再度保持するための戻り線があり，レジスタへの入力線が計 4 本以下にするためである．この段階で，入力先の FU・レジスタ，あるいは出力先の FU・レジスタが一致したレジスタは全て統合される．

Step b-5 残りのレジスタに関して条件 1，2 を満たす場合，他の FU・レジスタからレジスタへの入力線が 3 本以下になるようにレジスタを統合し，空のレジスタを削除する（図 3.15(e)）．こうすることで，レジスタ自身の値を戻すための入力線を含め各レジスタへの入力線は 4 本以下となるため，各レジスタに発生する MUX は最小である 4 入力以下の大きさとなる．図 3.15(e) は以上の処理を行い統合したレジスタバインディング結果である． □

3.5 計算機実験結果と評価

本章では，提案手法の計算機実験結果を示し，既存手法と比べた際の提案手法の有効性を明らかにする．

3.5.1 実験環境

本稿では，計算機実験の方法および環境について述べる．

HDR アーキテクチャを対象とした FPGA 設計フローを図 3.16(a) に示す．まず提案手法を C++言語で実装し，ベンチマークアプリケーションに適用して RTL 記述を出力する．次に，Xilinx 社 ISE Design Suite 14.2 で論理合成を行い，ゲートレベルの回路情報を得る．FPGA ボードは Xilinx 社の Virtex-6 を想定しており，dsp ブロックの使用を禁止とした．次に，ISE

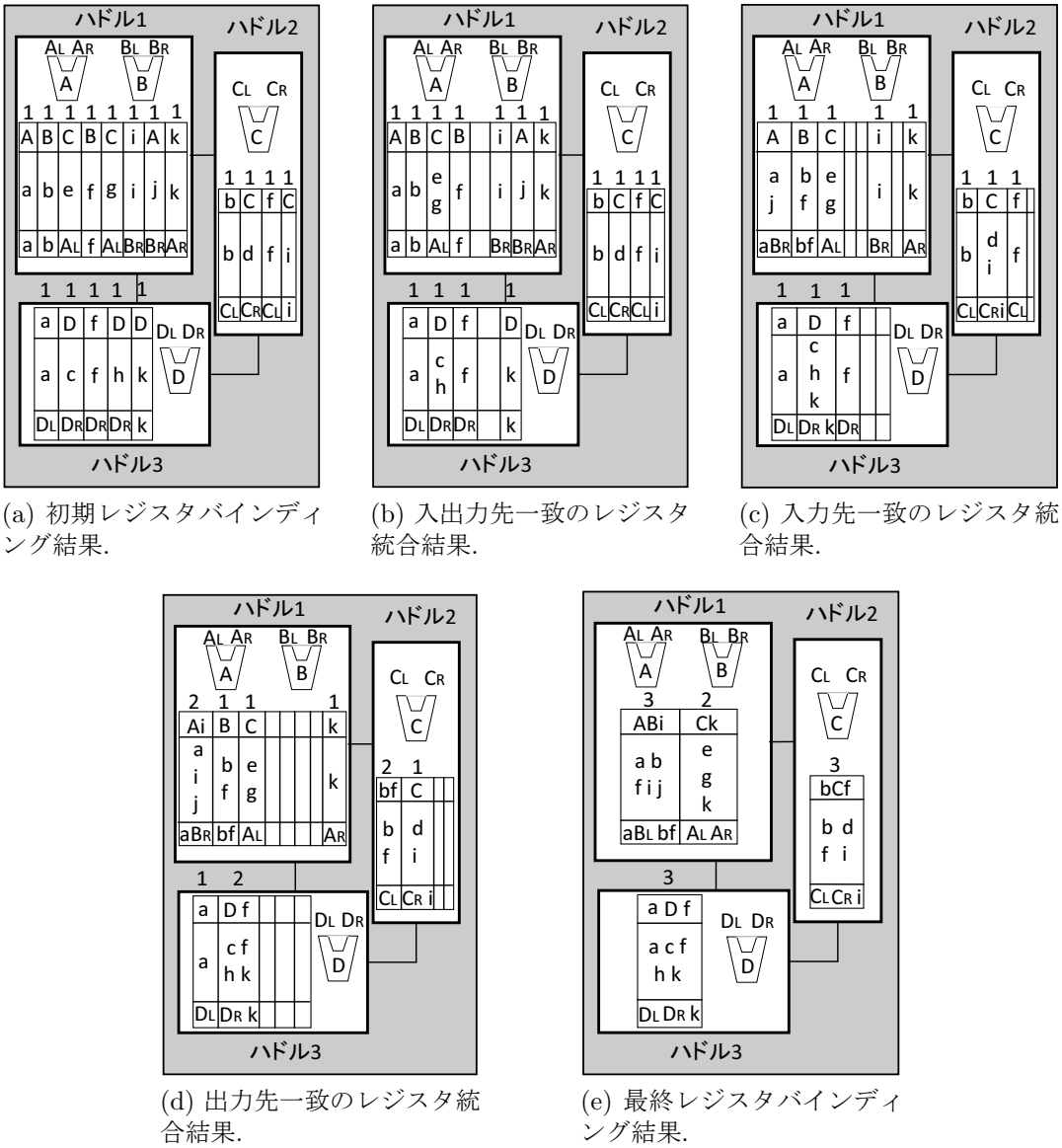


図 3.15: パス考慮レジスタバインディング結果.

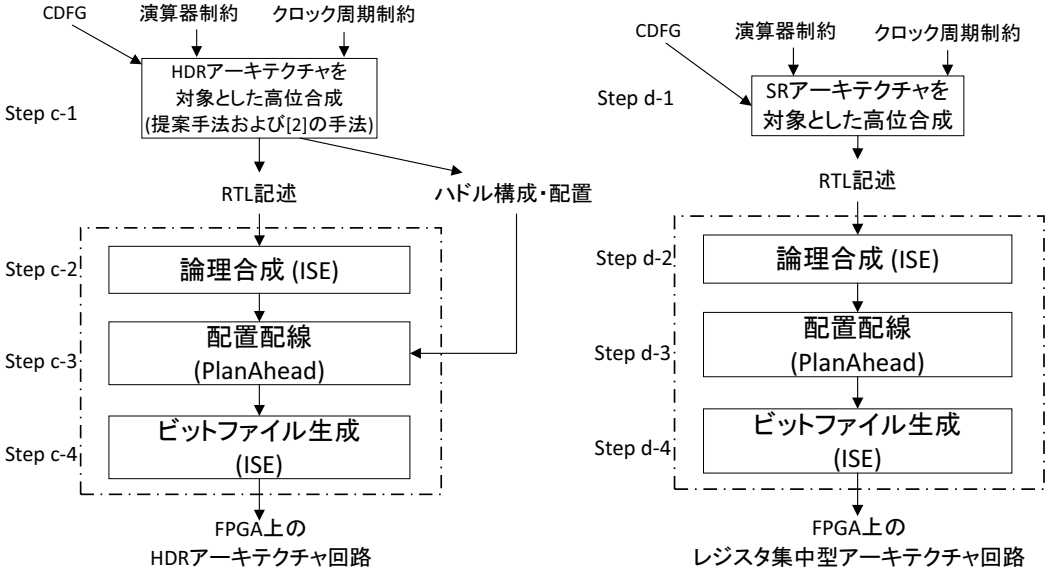
内の PlanAhead で配置・配線を行い，最終的な回路情報を得る．配置配線の際，クロック周期を最小とする合成結果を得ている．最後に，ISE によりビットファイル化し，FPGA 上の HDR アーキテクチャ回路が実装される．

提案手法の実行に使用した計算機環境は，AMD Opteron 2360SE 2.5GHz×2，メモリが 16GB である．論理合成，配置・配線に使用した計算機環境は，Intel Core i5-2520M 2.5GHz ×2，メモリが 4GB である．

対象アプリケーションとして，HAL（ノード数 11），PARKER（ノード数 22，条件分岐あり），DCT（ノード数 48），jacobi（ノード数 52），7 次 FIR フィルタ（ノード数 75），EWF3（ノード数 102），copy（ノード数 378，条件分岐あり）を用いた．実験で用いたライブラリの情報を表 4.12 に示す．FU の扱う値は全て 16 ビットである．各 FU の実行サイクル数は 1 とした．配線遅延は配線長に比例すると仮定し，60 スライス分の距離あたり 1ns とした．

表 3.3: 演算器ライブラリ情報.

FU/レジスタ	スライス数	遅延 [ns]
加算器	4	1.5
減算器	4	1.5
乗算器	38	4.3
除算器	91	29.9
比較器	2	1.6
右シフト器	13	2.2
AND	6	0.83
レジスタ	4	0.37
メモリ書き込み	0	1.8



(a) HDR アーキテクチャの設計フロー.　(b) SR の設計フロー.

図 3.16: FPGA マッピングまでの設計フロー.

比較する従来手法として SR と [2] を用意した．SR はレジスタ集中型アーキテクチャを対象とした高位合成手法である．リストスケジューリングベースのスケジューリング/FU バインディング手法 [57]，レフトエッジアルゴリズム [46] で構成され，フロアプランを考慮しない手法である．SR を用いた FPGA 設計フローを図 3.16(b) に示す．[2] は，提案手法と同様に図 3.16(a) のフローにて実装した．

3.5.2 実験結果

本稿では，計算機実験の結果を示し，それについて考察を行う．

実験結果を表 3.4 に示す．表 3.4 の 1 列目の App. は入力アプリケーション，2 列目はノード数，3 列目は各アプリケーションの FU 制約を表す．例えば，Add × 2 は加算器 2 つ，Sub

表 3.4: 計算機実験結果.

App.	#Nodes	FU constraint	Clock period constraint [ns]	Applied algorithm	#Huddles	Delay [ns]	#Steps	Latency [ns]	#Slices	#LUTs	#FFs	Total synthesis time [s]
hal	11	Add×1, Sub×1, Mul×2, Comp×1	6	SR	-	5.260	6	31.56	136	425	146	131
				[2]	3	4.856	5	24.28	124	365	159	126
				提案手法	2	4.944	5	24.72	107	340	142	135
parker	22	Add×2, Sub×2, Comp×1	3	SR	-	2.151	7	15.06	80	229	256	136
				[2]	3	2.434	8	19.47	69	246	288	254
				提案手法	2	2.264	8	18.11	62	225	270	155
dct	48	Add×4, Mul×4	8	SR	-	5.670	8	45.36	460	1269	355	171
				[2]	5	5.947	9	53.52	369	1187	503	207
				提案手法	4	5.468	8	43.74	321	1101	495	186
jacobi	52	Add×2, Sub×1, Mul×2, Div×2	32	SR	-	21.75	17	369.8	489	1317	297	167
				[2]	4	23.37	17	397.2	373	1289	336	188
				提案手法	4	22.61	17	384.4	340	1249	334	244
fir	75	Add×4, Mul×4, Mem×1	7	SR	-	5.865	15	87.98	492	1449	309	163
				[2]	4	5.769	15	86.54	420	1435	398	210
				提案手法	4	5.267	15	79.01	261	913	432	213
ewf3	102	Add×4, Mul×4	7	SR	-	5.989	40	239.6	604	1659	376	186
				[2]	4	5.190	40	207.6	507	1509	395	279
				提案手法	4	5.066	40	202.6	441	1443	486	239
copy	378	Add×3, Sub×1, Mul×5, Comp×1, Rshift×2, AND×1, Mem×1	12.5	SR	-	6.987	82	572.9	1747	4904	1511	310
				[2]	7	6.884	82	564.5	1591	5048	2429	898
				提案手法	8	6.709	82	550.1	1324	3929	2696	557

× 1 は減算器 1 つを表す．Mul は乗算器，Div は除算器，Comp は比較器，Rshift は右シフト器，Mem はメモリユニットを表す．演算器制約は，[1, 2] に従い設定した．4 列目は各アプリケーションのクロック周期制約を表す．5 列目は使用した手法を表す．今回使用した手法は SR，従来手法 [2]，提案手法である．6 列目は提案手法と [2] のハドル数を表す．7 列目は回路を配置配線した際の最小クロック周期として得られた値を，8 列目はコントロールステップ数を，9 列目は $Delay \times CS$ 数 で求めたレイテンシを表す．10 列目は回路全体のスライス数を，11 列目は回路全体の LUT 数，12 列目は回路全体の FF 数を表す．13 列目は高位合成，論理合成，配置配線で要した計算機時間の合計を表す.

表 3.4 の 7–9 列目の通り，提案手法は従来手法とほぼ同等の CS 数を実現し，クリティカルパス遅延を削減することでレイテンシを削減した．結果として，提案手法は SR と比較して回路のレイテンシを最大 22%，平均 4%削減し，[2] と比較して最大 18%，平均 6%削減した．提案手法と MUX のコストを考慮しない従来の HDR アーキテクチャの手法 [2] をレイテンシにおいて比較すると，hal を除く全てのアプリケーションにおいて提案手法は [2] に比べて遅延削減を実現できている．hal においては，0.088ns の最小クロック周期の差，0.44ns のレイテンシの差と非常に微小であることから，下位工程での配置・配線結果による誤差と考える．従って，本章で提案する MUX のコスト削減の効果は有効であると言える.

一方で，提案手法は SR に比べて，hal, parker, jacobi においてレイテンシを削減できない場合が見られた．hal や parker など小規模なアプリケーションでは生成される回路が小さく，また jacobi では除算器が明らかにクリティカルパスとなる．これらを踏まえると，これらのアプリケーションでは生成された回路におけるクリティカルパスに占める配線遅延の影響が小さく，HDR アーキテクチャのハドルにより冗長な部分がレイテンシを増加してしまったことが原因と考えられる．実際に，提案手法は，fir, ewf3, copy の大規模アプリケーションでは効果的にレイテンシを削減できていることが確認できる.

一方，表 3.4 の 10 列目の通り，提案手法は SR と比較して回路のスライス数を最大 47%，平均 29%削減した．また，[2] と比較して最大 38%，平均 16%削減した．そして，提案手法は全てのアプリケーションにおいて，SR，[2] に比べてスライス数の削減を達成できた．

3.6 本章のまとめ

本章では，FPGA を対象とした高位合成における課題 1「フロアプランの考慮と MUX のコスト削減の同時実現」を解決するため，フロアプランの考慮と MUX のコスト削減を同時に実現する FPGA 向け高位合成手法として，HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法を提案した．まず，FPGA 上に回路の各構成要素を実装・分析し，MUX が FPGA 上でボトルネックであることを明らかにした．そして，MUX の入力数を変化させて実装し，入力数に応じた MUX のコスト増加傾向を分析し，MUX 数の削減・入力数の制限という提案手法の方針を決定した．そして，上記アプローチに従い，HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法を提案した．そして，最後に計算機実験によって既存手法（SR, [2]）と比較した際の提案手法の優位性を明らかにした．

提案手法はレジスタ分散型アーキテクチャの 1 つである HDR アーキテクチャを用いることで，高位合成段階でフロアプランを扱い，モジュール間の配置配線を見積る．また，各 FU (演算器)・レジスタ間のデータ転送を考慮した 2 つの新たな FPGA 向けバインディング手法を用いて，高位合成段階で MUX のコストを削減したデータパスを生成する.「パス考慮スケジューリング/FU バインディング」では，すでにデータパスがある 2 つの FU 同士を積極的に連続する 2 つの演算ノードに割り当てることで MUX 数の削減を図る.「パス考慮レジスタバインディング」では，入出力先の FU・レジスタを考慮してレジスタに付加する MUX を 4 入力以下に制限することで MUX の総面積・遅延を削減する．

計算機実験により，提案手法は従来のレジスタ集中型アーキテクチャを対象とした手法 (SR) と比較して，回路のレイテンシーを最大 22%，平均 4%削減し，スライス数を最大 47%，平均 29%削減することを確認した．また，提案手法は従来の HDR アーキテクチャを対象とした高位合成手法 [2] と比較して，回路のレイテンシーを最大 18%，平均 6%削減し，スライス数を最大 38%，平均 16%削減することを確認した．

第4章　フロアプラン指向FPGA高位合成向け配線遅延・クロックスキュー見積りモデル

4.1　本章の概要

本章では[1]，FPGA を対象とした高位合成における課題 2「高位合成段階における，配線遅延・クロックスキューの正確な見積り」を解決するため，フロアプラン指向 FPGA 高位合成のための配線遅延・クロックスキュー見積りモデル「IDEF」と「CSEF」を構築する．まず，FPGA 上における測定を行い，配線遅延/クロックスキュー特性を明らかにする．そして，それらの特性に基づいて定式化を行い，それぞれの見積りモデルを構築する．その後，実測値と比較を行い，各々のモデルの見積り精度を評価する．最後に，第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に両モデルを適用し，評価を行う．

FPGA では，パス遅延に占める配線遅延・クロックスキューの割合が大きい．そのため，フロアプラン指向高位合成では，フロアプランに基づく配線遅延・クロックスキュー見積り精度は，フロアプラン解・生成されるデータパスの質に影響を与えてしまうため，できるかぎり高精度に見積る必要がある．FPGA の配線遅延見積りモデル構築のため，まず，シンプルな測定実験用の回路を用いて，様々なパターンで FPGA の配線遅延を測定し，FPGA の配線遅延特性を明らかにする．そして，測定結果に基づいて FPGA の配線遅延見積りモデル「IDEF」を提案する．次に，FPGA のクロックスキュー見積りモデル構築のため，FPGA 上でのクロックスキューの影響を示した上で，様々なパターンで FPGA のクロックスキューを測定し，FPGA のクロックスキュー特性を明らかにする．そして，クロックスキュー見積もりモデル「CSEF」を提案する．それぞれの見積りモデルの精度は，商用の FPGA 設計ツール内タイミングモデルの結果と比較し確認する．最後に，第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に IDEF・CSEF を適用し，計算機実験により有効性を示す．

4.2　FPGAの配線遅延特性

本節では，FPGA の配線遅延特性の測定を行う．

[47] で述べられている通り，Virtex-7 [73] など近年主流となっている FPGA は主に 2 つの配線，スライス（論理セル）内の配線とスライス間の配線を持つ．本論文では，スライスの

[1]本章の内容は [32] による．

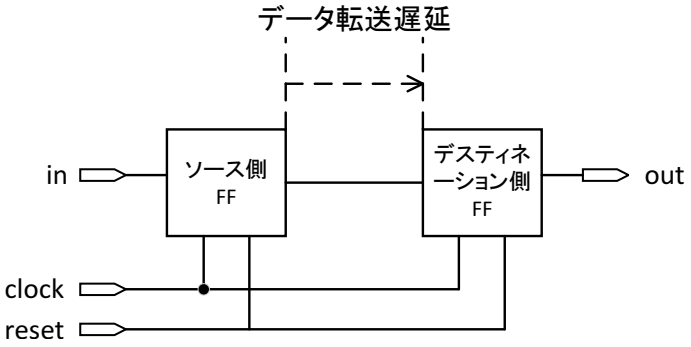


図 4.1: 測定実験用回路.

大きさはチップ全体に比べて微小であり，スライス内の配線遅延も同様に微小であると考え，スライス間の配線遅延のみを想定し議論する．

以上の仮定を確認するため，図 4.1 の回路を用いて Xilinx 社の Virtex-7（xc7vx485tffg1761-2）で予備実験を行った．図 4.2 に Virtex-7 のスライスの構成を示す．Virtex-7 のスライスは 4 つの 6 入力 LUT と 8 つの FF，いくつかの専用ロジック（専用 MUX や XOR ロジック）を持つ．Xilinx 社の Vivado Design Suite 2014.2 を用いて図 4.1 の回路のソース側 FF を図 4.2 の FF1 に，デスティネーション側 FF を図 4.2 の FF2 に配置した．図 4.2 に Vivado による配線結果を示す．データはまず FF1 からスライスの最寄りのスイッチボックスへ送られる．その後，スイッチボックスからスライス内の配線を通って FF2 へ送られる．FF1 から FF2 への遅延を Vivado の STA（Static Timing Analysis）ツールで求めたところ，0.109ns であった．従って，スライス内の配線遅延は 0.109ns 未満であると言える．これは，表 4.2 に後述するスライス間の配線遅延に比べて，十分に小さい．従って，本論文では前述の通り，スライス間の配線遅延のみを想定し議論する．

測定方法

本節での，配線遅延の測定方法について述べる．図 4.3 に Xilinx 社の Virtex-7 (xc7vx485tffg1761-2) の構成を示す [73][2]．図 4.3(a) に示す通り，Virtex-7 は 14 個のクロック領域（Clock region: CR）で構成される．図 4.3(b) の通り，各 CR はいくつかのスライスと特殊セル（DSP ブロックや RAM），I/O パッドで構成される．Virtex-7 はキャリーチェーンのための占有領域を持ち，ここにモジュールを配置することはできない．$S(x, y)$ は Virtex-7 の x 列 y 行目のスライスを表すとし，x, y の範囲は $0 \leq x \leq 221$ and $0 \leq y \leq 349$ である．

スライス $S_A = S(x_A, y_A)$ からスライス $S_B = S(x_B, y_B)$ への配線遅延を以下の手順で測定する．

1. 3 つの入力ピン，1 つの出力ピン，2 つの FF で構成される図 4.1 のデータパス回路を論理合成する．
2. 入力ピン，出力ピンを任意の I/O パッドに配置する．

[2]以下の議論では，近年で最も有名な FPGA の 1 つとして Virtex-7 を想定して議論を進めるが，表 4.4 で示すように他の Xilinx 社の FPGA にも同様の議論ができる．

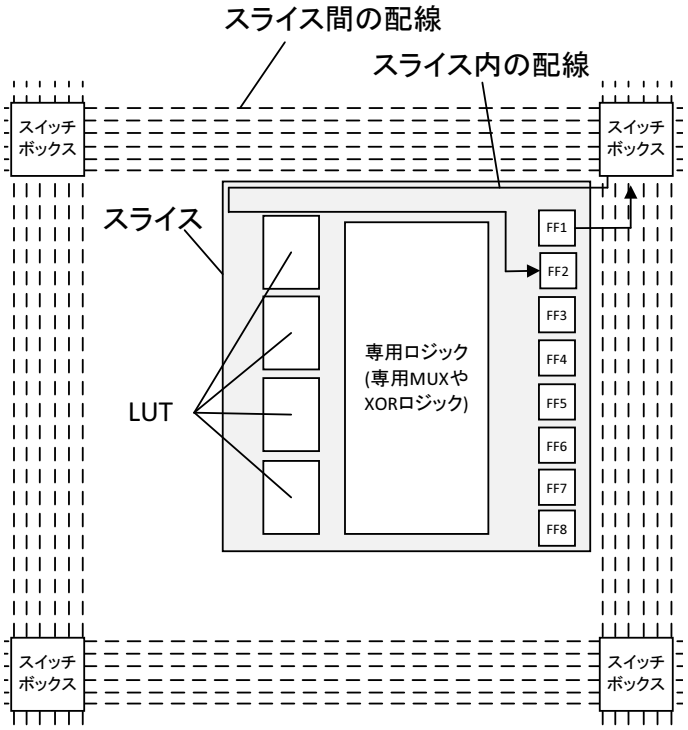


図 4.2: スライス構成とスライス内の配線.

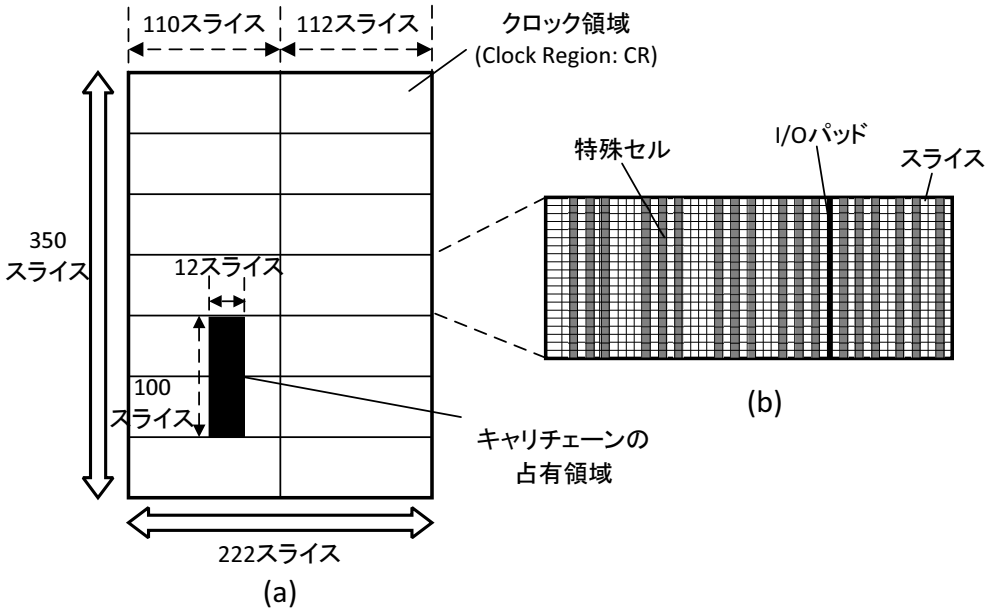
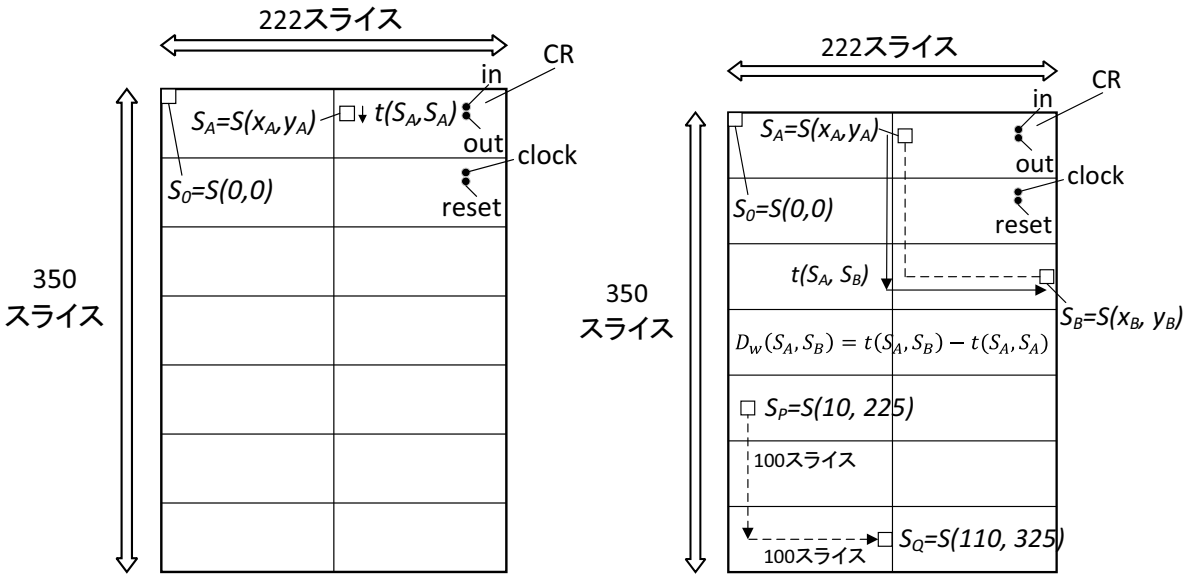


図 4.3: Virtex-7 と論理ブロック構成 [73].

3. ソース側 FF とデスティネーション側 FF をスライス $S_A = S(x_A, y_A)$ へ配置し，ソース側 FF からデスティネーション側 FF へのデータ転送時間 $t(S_A, S_A)$ を Vivado の STA ツールで測定する（図 4.4(a)）.
4. デスティネーション側 FF をスライス $S_B = S(x_B, y_B)$ $(S_A \neq S_B)$ へ移動させ，ステッ



(a) Simulating $t(S_A, S_A)$. (b) Simulating $t(S_A, S_B)$.

図 4.4: Virtex-7 上の配線遅延測定方法.

プ 3 と同様にデータ転送時間 $t(S_A, S_B)$ を測定する（図 4.4(b)）.

5. $D_w(S_A, S_B)$ はスライス $S_A = S(x_A, y_A)$ からスライス $S_B = S(x_B, y_B)$ への配線遅延を表すとし，以下の式で求める.

$$D_m(S_A, S_B) = t(S_A, S_B) - t(S_A, S_A). \tag{4.1}$$

ここで，本論文で図 4.1 のような単純なデータパス回路を用いる妥当性について議論する. 高位合成段階では，個々のモジュールをフロアプランし，かつ FPGA 上の詳細な機構（特殊セルや I/O パッドなど）を扱うことは，高位合成問題の複雑さにより難しい. この問題を解決するため，フロアプラン指向 FPGA 高位合成では例えば HDR アーキテクチャの「ハドル」のような抽象化されたモジュールと簡略化された FPGA 上の配置を扱う. しかし，第 2.4 節で述べた通り HDR アーキテクチャでは，高位合成段階で FU，レジスタ，コントローラのハドル内での詳細な配置は決定されないため，ハドル間の詳細な配線を無視して単純にハドルの位置より配線遅延を見積る. 従って，本論文では図 4.1 のような単純なデータパス回路を用いて FPGA 上の位置に伴う配線遅延特性の測定を行う.

測定結果

測定 1 (FPGA 全体の配線遅延特性)

Virtex-7（SpeedGrade -2）上の 10 本のパスの配線遅延を測定した. ソース側 FF，デスティネーション側 FF を縦もしくは横に 100 スライス離してランダムに配置した. 表 4.1 に測定結果を示す. 表 4.1 の 2–6 行目の通り，FPGA 上の横方向 100 スライスの配線遅延はほぼ同様の値を示し，縦方向 100 スライスの配線遅延に関しても表 4.1 の 7–11 行目より同様の結果が得られた. 以上の結果より，FPGA ではチップ上の配線遅延遅延特性はチップ上の場所に依らず一様であることがわかる. □

表 4.1: 100 スライスの配線遅延測定結果 (**測定 1**).

Placement of source FF S_A	Placement of destination FF S_B	$D_m(S_A, S_B)$ [ns]
$S(0,0)$	$S(100,0)$	1.77
$S(50,100)$	$S(150,100)$	1.80
$S(70,220)$	$S(170,220)$	1.76
$S(110,260)$	$S(210,260)$	1.92
$S(20,320)$	$S(120,320)$	1.88
$S(0,0)$	$S(0,100)$	1.59
$S(20,180)$	$S(20,280)$	1.63
$S(50,50)$	$S(50,150)$	1.68
$S(160,200)$	$S(160,300)$	1.59
$S(220,240)$	$S(220,340)$	1.60

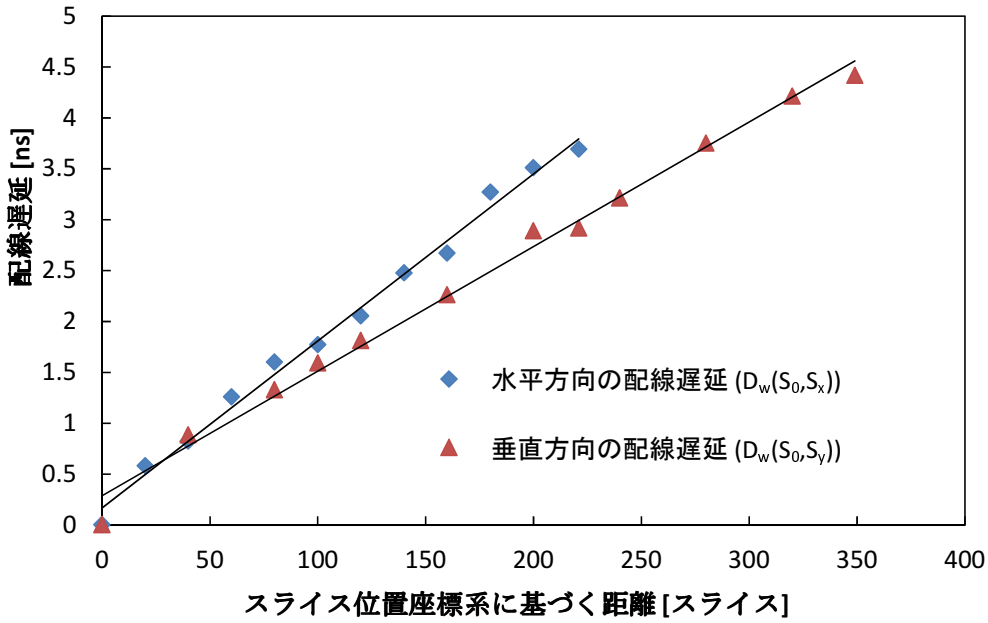


図 4.5: スライス位置座標系に基づく距離と配線遅延の関係 (**測定 2**).

測定 2 (縦と横の配線遅延特性)

Virtex-7 (Speed Grade -2) にて縦と横の配線遅延の測定を行った．スライス $S_0 = S(0,0)$ からスライス $S_x = S(x,0)$ もしくは $S_y = S(0,y)$ の配線遅延を測定した．図 4.5 に縦・横方向の配線遅延 $D_m(S_0, S_x)$ and $D_m(S_0, S_y)$ の測定結果を示す．図 4.5 の通り，FPGA では縦・横の配線遅延はそれぞれ距離に対して比例して増加し，FPGA は縦と横で異なる配線遅延特性を持つことがわかる． □

測定 3 (斜めの配線遅延特性)

斜め方向に配置された 6 本のデータパスを Vertex-7 (Speed Grade -2) 上に実装し，配線

表 4.2: 斜め方向の配線遅延測定結果 (**測定 3**).

Horizontal distance X [slices]	Vertical distance Y [slices]	$D_m(S_0, S_{x,y})$ [ns]	$D_m(S_0, S_x) + D_m(S_0, S_y)$ [ns]
100	120	3.40	3.58
100	240	5.00	4.98
100	349	5.84	6.19
221	120	5.24	5.50
221	240	6.66	6.90
221	349	8.14	8.10

表 4.3: Speed Grade 毎の水平/垂直方向の配線遅延 (**測定 4**).

Speed Grade	$D_m(S_0, S_x)$ [ns]	$D_m(S_0, S_y)$ [ns]
-1	4.06	3.37
-2	3.51	2.89
-3	3.16	2.67

遅延の測定を行った．表 4.2 にスライス $S_0 = S(0,0)$ からスライス $S_{x,y} = S(x,y)$ への斜め方向の配線遅延 $D_m(S_0, S_{x,y})$ の測定結果を示す．表 4.2 の 1, 2 列目はスライス $S_0 = S(0,0)$ のソース側 FF とスライス $S_{x,y} = S(x,y)$ のデスティネーション側 FF 間の横方向，縦方向の距離 $X = L_h(S_0, S_{x,y})$，$Y = L_v(S_0, S_{x,y})$ を表す．4 列目は測定 2 で得られた結果より $D_m(S_0, S_x) + D_m(S_0, S_y)$ の値を示す．表 4.2 の通り，配線遅延 $D_m(S_0, S_{x,y})$ は横方向の配線遅延 $D_m(S_0, S_x)$ と縦方向の $D_m(S_0, S_y)$ の和におおよそ等しいとわかる．□

測定 4 (Speed Grade による配線遅延特性)

Speed Grade を-1, -2, -3，それぞれの時の Virtex-7 で横と縦の配線遅延の測定を行った．表 4.3 に $S_x = S(200,0)$，$S_y = S(0,200)$ の時の横と縦の配線遅延 $D_w(S_0, S_x)$，$D_w(S_0, S_y)$ の測定結果を示す．Speed Grade が-1 の時，横方向の配線遅延は 16%，縦方向の配線遅延は 17%，Speed Grade が-2 の時と比べて増大した．Speed Grade が-3 の時，横方向の配線遅延は 10%，縦方向の配線遅延は 8%，Speed Grade が-2 の時と比べて減少した．以上より，対象 FPGA の Speed Grade によって横と縦の配線遅延特性は変化するとわかる．□

測定 5 (FPGA の種類による配線遅延特性)

Airtex-7 (xc7a200tffg1156-2), Kintex-7 (xc7k480tffg1156-2) と Virtex-7 (xc7vx485tffg1761-2) において，横と縦の配線遅延の測定を行い，測定結果を表 4.4 に示す．表 4.4 の通り，各 FPGA の種類ごとに独自の配線遅延特性を持ち，同距離の縦と横の配線遅延では，18%の差が生じた．つまり，FPGA によって FPGA の縦と横の配線遅延特性を考慮しない場合，その見積り誤差は最大 18%になることになる．□

以上の 5 つの測定結果より，FPGA の種類毎に独自の配線遅延特性を持つことは明らかであり，[12, 42] の配線遅延見積りモデルは FPGA の種類，Speed Grade，縦と横の配線遅延特性を無視しており，大きさな見積り誤差を生じる可能性がある．高位合成段階で FPGA の配線遅延を適切に見積るために，FPGA 毎の配線遅延特性をフロアプラン指向 FPGA 高位合成で考慮できる配線遅延見積りモデルの構築が必要である．

表 4.4: FPGA の種類毎の水平/垂直方向の配線遅延 (**測定 5**).

FPGA family	Horizontal distance [slices]	Vertical distance [slices]	$D_m(S_0, S_x)$ or $D_m(S_0, S_y)$ [ns]	Ratio $D_m(S_0, S_x)/D_m(S_0, S_y)$
Airtex-7	160	0	2.89	2.89/3.00
	0	160	3.00	≈ 0.96
Kintex-7	160	0	2.45	2.45/2.39
	0	160	2.39	≈ 1.03
Virtex-7	160	0	2.67	2.67/2.26
	0	160	2.26	≈ 1.18

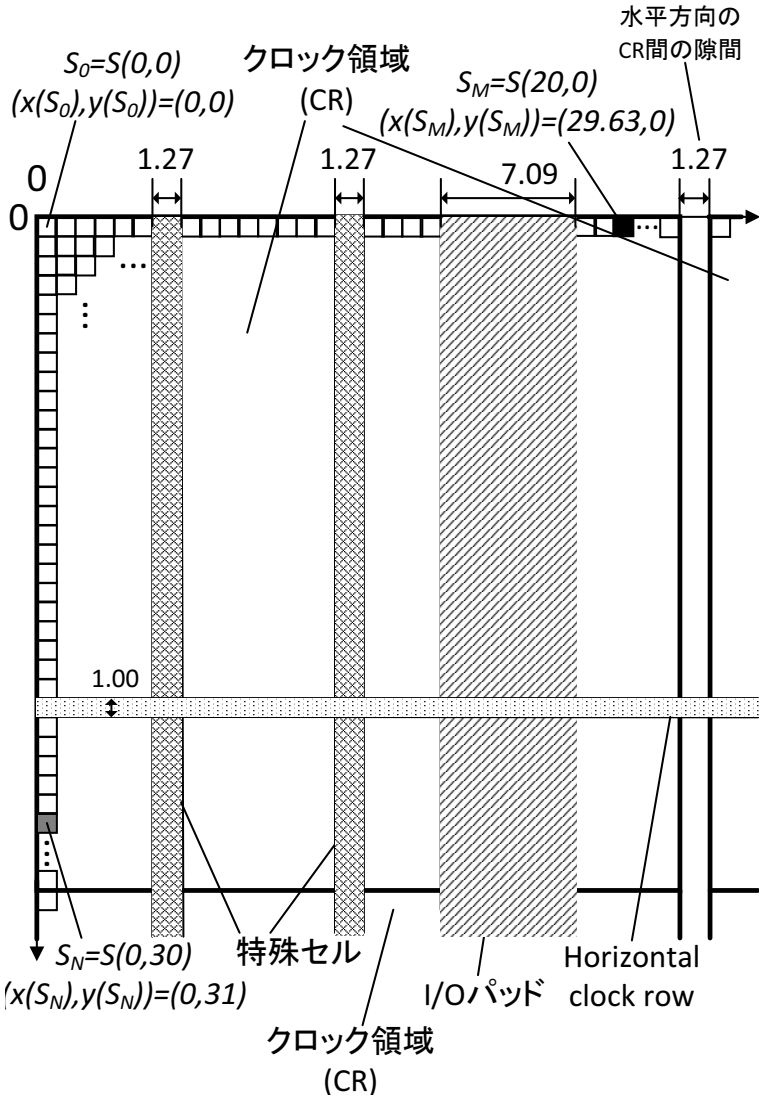


図 4.6: 実際の座標系に基づく Virtex-7 上の座標例.

実際の位置座標系に基づく考察

測定 2 の結果では，FPGA のスライスの位置座標系に基づく結果であった．FPGA の各構成要素（スライス，I/O パッド，特殊セルなど）の大きさを基に FPGA 上の実際の座標に変

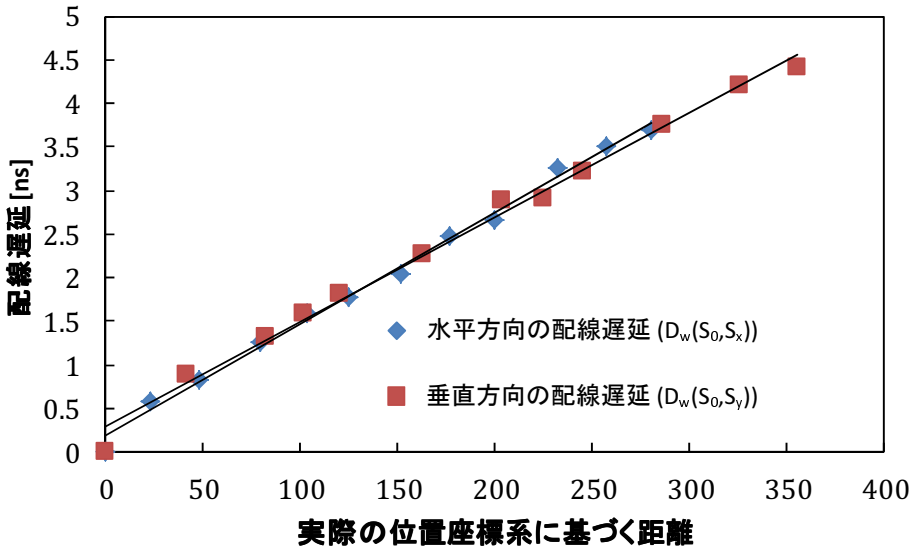


図 4.7: 実際の座標系に基づく距離と配線遅延の関係.

換し，配線遅延特性を評価し議論する．

Virtex-7 では，各スライスは正方形であり，各辺の長さを 1.00 とすると，特殊セルの幅は 1.27，I/O パッドの幅は 7.09，horizontal clock row (図 4.8) の高さは 1.00，各 CR の水平方向の隙間は 1.27 となる（図 4.6）．そして，各スライスの実際の位置座標を求めた．

ここで，$x(S_A)$ と $y(S_A)$ は FPGA チップ上のスライス S_A の左上の点の実際の位置座標を表すとする．スライス S_0 は FPGA チップの左上のスライスとし，S_0 の実際の位置座標を (0,0) とする．図 4.6 のスライス S_M（黒いスライス），スライス S_N（灰色のスライス）の実際の位置座標は以下のように求める．

$$\begin{aligned} x(S_M) &= 1.00 \times 20 + 1.27 \times 2 + 7.09 = 29.63 \\ y(S_M) &= 0 \end{aligned} \tag{4.2}$$

$$\begin{aligned} x(S_N) &= 0 \\ y(S_N) &= 1.00 \times 30 + 1.00 = 31 \end{aligned} \tag{4.3}$$

実際の位置座標系に基づいて再計算した測定 2 の結果を図 4.7 に示す．図 4.7 に示す通り，実際の位置座標では横と縦の配線遅延特性の差はごくわずかとなった．高位合成段階で実際の位置座標系を用いることで，FPGA の横と縦の配線遅延特性は簡略化できることが予想されるが，実際の位置座標系を用いるためには特殊セルや I/O パッドの位置や大きさなど詳細な FPGA アーキテクチャ情報が必要となる．前述した通り，フロアプラン指向 FPGA 高位合成では高位合成問題の複雑さにより，これらを扱うことができない．また，Xilinx 社 Vivado など FPGA 開発ツールはスライス座標系に基づいた配置情報を扱っている．以上より，フロアプラン指向 FPGA 高位合成ではスライス座標系を用いたフロアプラン情報を扱うのが妥当と考える．

また，近年の FPGA では，スライスのみではなく，特殊セルや I/O パッドを必ず有している．従って，スライス座標系に基づいた場合，本章の測定に用いた Xilinx 社の FPGA 以外の FPGA においても，FPGA の横と縦の配線遅延特性の差が存在すると考えられる．

4.3 FPGA の配置配線見積りモデル「IDEF」

前節より，FPGA ではそれぞれの FPGA の種類毎に独自の配線遅延特性を持つ．高位合成段階で FPGA の配線遅延を適切に見積るために，FPGA 毎の配線遅延特性をフロアプラン指向 FPGA 高位合成で考慮できる配線遅延見積りモデルの構築が必要である．本節では，FPGA の配線遅延見積りモデル「IDEF（Interconnection-Delay Estimate model for Floorplan-driven FPGA-HLS)」を定式化する．そして，FPGA 上の配線遅延の実測値と IDEF による見積り値を比較し，十分な見積り精度であることを確認する．

スライス S_A，S_B は，$S(x_A, y_A)$ and $S(x_B, y_B)$ のスライスとする．第 4.2 節で得られた FPGA の配線遅延特性を基に，スライス S_A，S_B 間の配線遅延 $D_e(S_A, S_B)$ は以下の式で見積る．

$$D_e(S_A, S_B) = C_{dh} \times |x_A - x_B| + C_{dv} \times |y_A - y_B|. \tag{4.4}$$

ここで，C_{dh}，C_{dv} は横，縦方向の配線遅延係数を表し，各 FPGA の種類，Speed Grade を考慮し測定 2 の結果に基づいて設定する．例えば，Virtex-7 (Speed Grade -2) では図 4.5 の傾きより，$C_{dh} = 0.0167$，$C_{dv} = 0.0130$ とする．

例 5 図 4.4(b) に示すような Virtex-7 上のスライス $S_P = S(10, 225)$ からスライス $S_Q = S(110, 325)$ へのデータ転送を想定する．Virtex-7 (Speed Grade -2) で，$C_{dh} = 0.0167$，$C_{dv} = 0.0130$ とした．

スライス S_Q と S_P は，水平方向に $|10 - 110| = 100$ スライス，横方向に $|225 - 325| = 100$ スライス離れており，スライス S_Q と S_P 間の配線遅延 $D_e(S_P, S_Q)$ は以下のように求める．

$$\begin{aligned} D_e(S_P, S_Q) =& 0.0167 \times |10 - 110| \\ &+ 0.0130 \times |225 - 325| = 2.97\text{ns} \end{aligned} \tag{4.5}$$

□

測定 6 (IDEF の評価)

IDEF の有効性を評価するため，図 4.1 のデータパス回路を用いて配線遅延の測定を行い，測定した配線遅延と IDEF（式 (4.4)）によって見積った配線遅延を比較した．配線遅延の測定値と見積り値を表 4.5 に示す．表 4.5 の通り，IDEF の見積り誤差は最大 0.20ns，平均 0.10ns であり，後述の 4.6 項で示すベンチマーク回路の遅延や前節の測定 1・測定 3 における配線遅延の測定値と比較して十分に小さい． □

4.4 FPGA のクロックスキュー特性

本節では，FPGA のクロックスキューの影響を測定し，フロアプラン指向 FPGA 高位合成でクロックスキューを考慮する重要性を示す．

まず，FPGA におけるクロックリソースについて述べる．図 4.8 に Xilinx 社の Virtex-7 の構成を示す [74]．クロックリソースとして，チップの上から 150 スライスの位置に 1 つの

表 4.5: 配線遅延の測定値と見積り値の比較 (**測定 6**).

Distance between two slices [(slices,slices)] $(L_h(S_0, S_{x,y}), L_v(S_0, S_{x,y}))$	Simulated delay [ns] $D_m(S_0, S_{x,y})$	Estimated delay by Eq. (4.4) [ns] $D_e(S_0, S_{x,y})$	Estimated error [ns]
(75, 120)	2.97	2.81	0.16
(75, 240)	4.31	4.37	0.06
(75, 349)	5.59	5.79	0.20
(150, 120)	4.01	4.07	0.06
(150, 240)	5.62	5.63	0.01
(150, 349)	7.18	7.04	0.14
(221, 120)	5.24	5.25	0.01
(221, 240)	6.66	6.81	0.15
(221, 349)	8.14	8.23	0.09

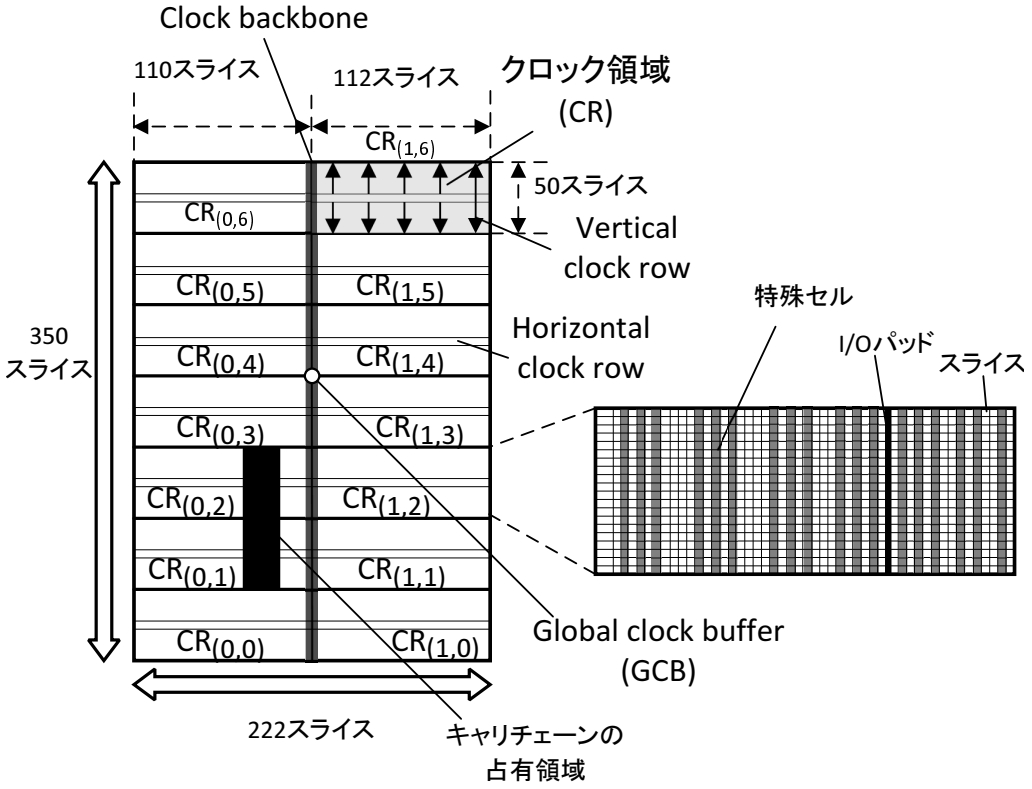


図 4.8: Virtex-7 とクロックネットワーク構成 [74].

Global Clock Buffer（GCB）を持つ．Virtex-7 上の回路へは GCB からクロックが供給される．GCB から発信されたクロック信号は Virtex-7 上を垂直方向の Clock Backbone を通じて各 CR へ供給される．各 CR 内では水平方向に中央を横断する Horizontal Clock Row，垂直方向の Vertical Clock Row を通って，各スライスのフリップフロップへ供給される．以上のように Virtex-7 を初めとする FPGA ではクロックネットワークが回路をチップ上に実装する前にあらかじめ形成されている．従って，フリップフロップがどの CR へ配置されるかによってクロックパスの長さは大きく変わる．

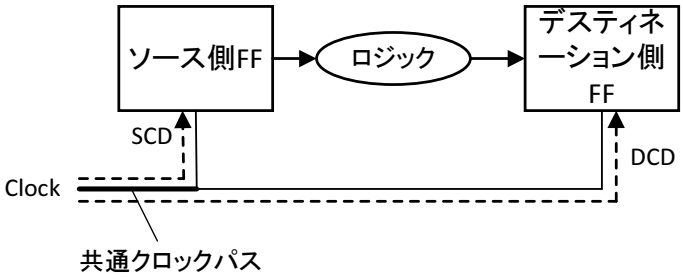


図 4.9: ソース側/デスティネーション側 FF へのクロックパス.

クロックスキュー

クロックスキューの仕組みを図 4.9 に示す．本論文では，クロックスキューはクロックパス長の違いによって生じる同一データパスのソース側 FF とデスティネーション側 FF のクロックの到達する時間差と定義する．[74] で示されているように，FPGA ではクロックネットワークが予め構築されており，各 FF へのクロックパス長は配置に強く依存する．FPGA 設計では，クロックスキューの影響を最悪の場合で見積るために，ソース側の FF へのクロックパスの最大遅延（SCD）とデスティネーション側の FF への最小遅延（DCD）に基づいて計算される．さらに，ソース側とデスティネーション側の FF それぞれへのクロックパスが共通クロックパスを持つ場合，そこに生じる遅延ばらつき（CPR）が考慮される．実際に Xilinx 社 Vivado Design Suite 2014.2 では，以下の式でクロックスキューを計算する．

$$Clock\ Skew = DCD - SCD + CPR. \tag{4.6}$$

Xilinx Vivado Design Suite 2014.2 では，データをデスティネーション側 FF へ正しく格納するために，回路内の全てのパスは以下の式を満たす必要がある．

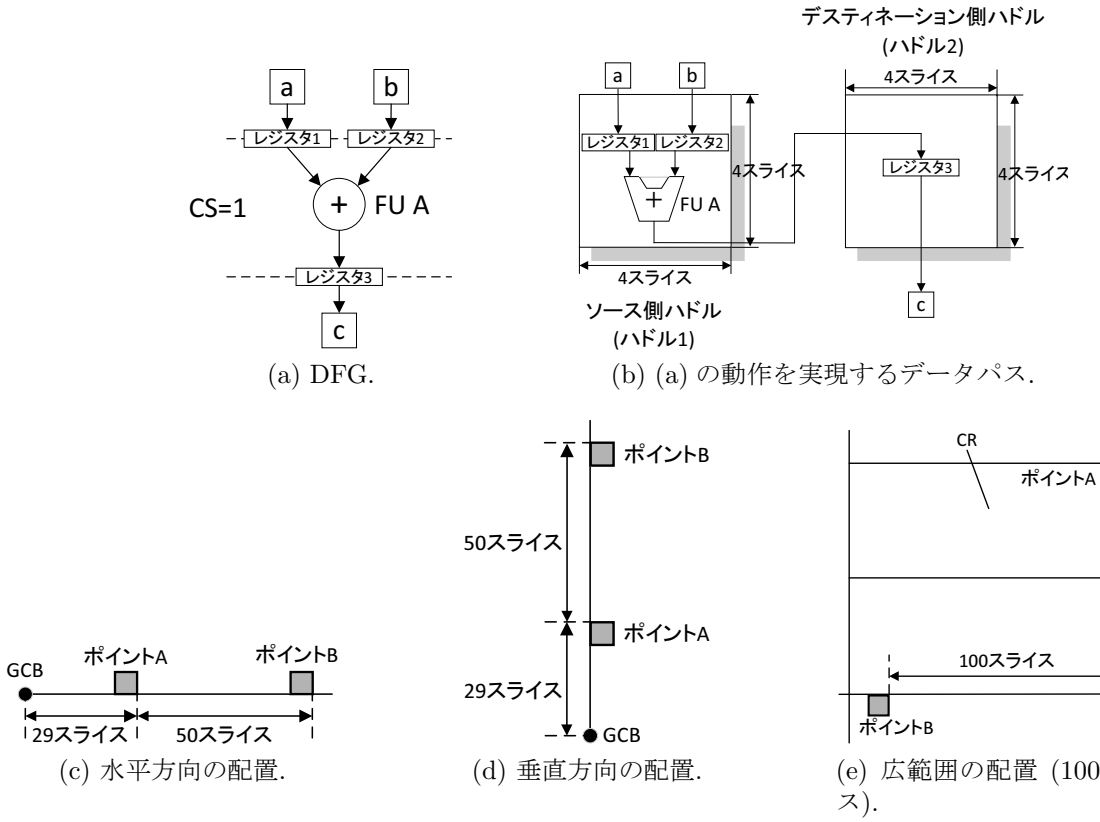
$$\begin{aligned} &\text{ロジック遅延} + \text{配線遅延} - \text{クロックスキュー} \\ &+ \text{クロックのジッター} \leq \text{クロック周期} \end{aligned} \tag{4.7}$$

式 (4.16) においてクロックのジッターは Vivado では FPGA チップ毎に定数で与えられる．また，式（4.16）より，配線遅延とクロックスキューは回路全体の動作周波数に大きな影響を与える可能性があるとわかる．言い換えると，遅延の大きいパスにおける配線遅延とクロックスキューを改善することで，クリティカルパス遅延を削減し回路の動作周波数を向上できる可能性がある．

次に，FPGA でのクロックスキューの影響を確認するために，以下のような実験を行った．

測定 7 (クロックスキューの影響)

クロックスキューの回路の動作クロック周期に与える影響を調べるために Virtex-7 上でクロックスキューの測定を行った．図 4.10(a) に示す DFG を扱うデータパスとして図 4.10(b) を用意した．これは，3 つのレジスタと 1 つの加算器で構成される．扱うデータは全て 16 ビットである．ハドル 1 とハドル 2 を図 4.10(c)-(e) のポイント A またはポイント B に配置して，それぞれクロックスキューの測定を行った結果を表 4.6 に示す．



(a) DFG.

(b) (a) の動作を実現するデータパス.

(c) 水平方向の配置.

(d) 垂直方向の配置.

(e) 広範囲の配置 (100×100 スライス).

図 4.10: 実験環境 (**測定 7**).

表 4.6: Simulated clock skews (**測定 7**).

Placement	Point A	Point B	Minimum *Clock Period* [ns]	*Path Delay* [ns]	*Clock Skew* [ns]	Difference of *Clock Skew* [ns]
Fig. 4.10(c)	ハドル 1	ハドル 2	3.064	3.019	+0.093	0.093 − (−0.195)
	ハドル 2	ハドル 1	3.423	3.114	−0.195	=**0.288**
Fig. 4.10(d)	ハドル 1	ハドル 2	2.745	2.653	+0.008	0.008 − (−0.280)
	ハドル 2	ハドル 1	3.121	2.701	−0.280	=**0.288**
Fig. 4.10(e)	ハドル 1	ハドル 2	5.853	6.197	+0.476	0.476 − (−0.724)
	ハドル 2	ハドル 1	7.037	6.214	−0.724	=**1.200**

表 4.6 よりクロックスキューはソースレジスタがデスティネーションレジスタよりクロックに近い方が，大きな値を示すことがわかる．また，100×100 スライス離れたパスでは，配置によって 1.2ns のクロックスキューの差が生じる可能性があるとわかる． □

以上より，FPGA 設計では大規模な回路においてクロックスキューが動作周波数に大きな影響を与える可能性がある．従って，高位合成段階でフロアプランを扱う際にクロックスキューを考慮し，クロックスキューを改善したフロアプランに決定することは FPGA 上に小遅延な回路を実現するために非常に重要である．

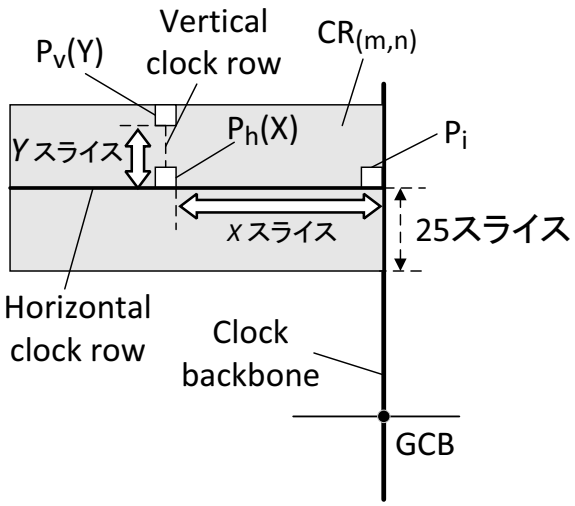


図 4.11: クロック伝播遅延の測定方法.

表 4.7: Clock backbone のクロック伝播遅延の測定結果 (**測定 8**).

Propagation delay [ns]	$CR_{(0,n)}$						
	$CR_{(0,0)}$	$CR_{(0,1)}$	$CR_{(0,2)}$	$CR_{(0,3)}$	$CR_{(0,4)}$	$CR_{(0,5)}$	$CR_{(0,6)}$
Maximum	4.128	3.956	3.784	3.612	3.599	3.759	4.113
Minimum	3.768	3.597	3.427	3.258	3.238	3.368	3.670
Propagation delay [ns]	$CR_{(1,n)}$						
	$CR_{(1,0)}$	$CR_{(1,1)}$	$CR_{(1,2)}$	$CR_{(1,3)}$	$CR_{(1,4)}$	$CR_{(1,5)}$	$CR_{(1,6)}$
Maximum	4.125	3.953	3.781	3.609	3.596	3.756	4.110
Minimum	3.765	3.595	3.425	3.255	3.235	3.365	3.667

4.5 FPGA のクロックスキュー見積りモデル「CSEF」

本節では，FPGA 上でのクロックスキュー特性の測定を行う．そして，測定に基づいて，クロックスキュー見積りモデル「CSEF（Clock-Skew Estimate model for Floorplan-driven FPGA-HLS)」を提案する．その後，FPGA 上のクロックスキューの実測値と CSEF による見積り値を比較し，十分な見積り精度であることを確認する．

式 (4.6) に示すように，クロックスキューは SCD，DCD，CPR に基づいて計算される．以下の実験では Virtex-7 (Speed Grade -2) 上にて図 4.1 の回路を用いて FF の配置を様々に変更し FPGA のクロックスキューを測定していく．論理合成，配置配線には Xilinx 社 Vivado Design Suite 2014.2 を使用し，入出力ピンは任意の I/O パッドに配置した．

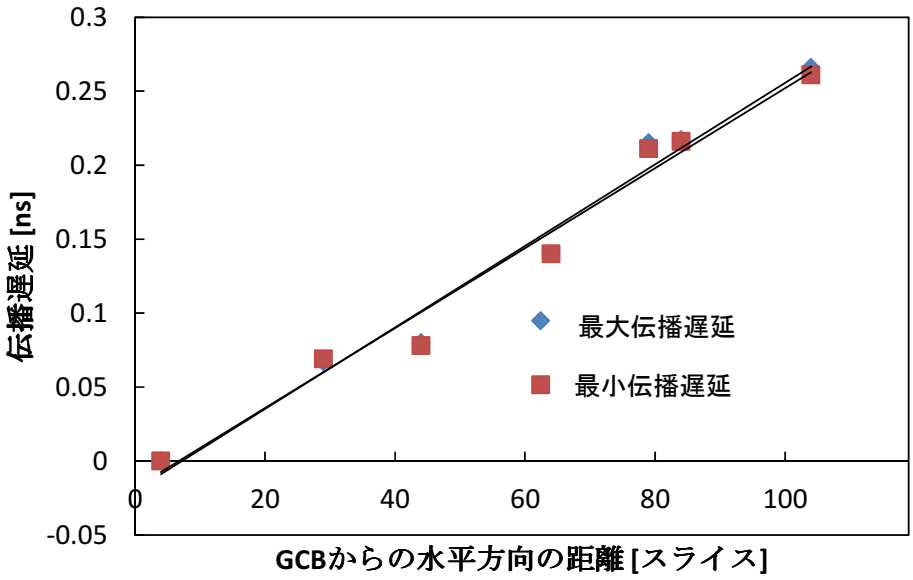


図 4.12: 水平方向の距離と Horizontal clock row のクロック伝播遅延の測定結果 (**測定 9**).

4.5.1 SCD, DCD

本稿では，FPGA 上でのクロックスキューの測定を行い，SCD, DCD の特性を明らかにする．

クロック信号は GCB から Clock backbone, Horizontal clock row, Vertical clock row を通って各 FF に伝搬される．

測定 8 (Clock backbone の遅延)

Clock backbone のクロック遅延の測定を行った．図 4.11 に示すように，ソース側 FF・デスティネーション側 FF をそれぞれの CR の Clock backbone と Horizontal clock row の交点の隣 P_i に配置し，GCB から P_i への遅延を測定した．表 4.7 に各 CR への Clock backbone の最大遅延・最小遅延を示す．

表 4.7 の通り，Clock backbone の遅延は GCB から遠い CR ほど大きく，水平方向に隣り合った CR ではほぼ同じ遅延を示す．この結果より，Clock backbone の最大遅延・最小遅延は FF の配置された CR に強く依存することが言える． □

測定 9 (Horizontal clock row の遅延)

次に，Horizontal clock row の遅延の測定を行った．図 4.11 のように CR $CR_{(m,n)}$ にて，点 P_i から x スライス離れた点 $P_h(X)$ にソース側 FF・デスティネーション側 FF を配置し，GCB から $P_h(X)$ への最大遅延・最小遅延を測定した．そして，測定 8 で得た結果との差を Horizontal clock row の遅延とした．図 4.12 に CR $CR_{(0,4)}$ における距離に伴う Horizontal clock row の遅延を示す．他の CR でも図 4.12 と同様の結果が得られた．図 4.12 より，Horizontal clock row の遅延は距離におおよそ比例し，その増加比率は最大遅延と最小遅延でほぼ同じである． □

測定 10 (Vertical clock row の遅延)

CR $CR_{(m,n)}$ における Vertical clock row の遅延の測定を行った．図 4.11 のように CR $CR_{(m,n)}$

表 4.8: Vertical clock row のクロック伝播遅延の測定結果 (**測定 10**).

Vertical position to horizontal clock row [slices]	Maximum propagation delay [ns]	Minimum propagation delay [ns]
+0	0	0
+5	0.007	0.006
+10	0.012	0.010
+15	0.017	0.013
+20	0.019	0.015
−0	0	0
−5	0.007	0.006
−10	0.012	0.010
−15	0.017	0.013
−20	0.019	0.015

内の点 $P_h(X)$ から縦に Y スライス離れた点 $P_v(Y)$ にソース側 FF・デスティネーション側 FF を配置し，GCB からの最大遅延・最小遅延を測定した．そして，測定 8，9 で得た結果との差を Vertical clock row の遅延とし，CR $CR_{(0,4)}$ での結果を表 4.8 に示す．ここで，Y の値が正の時は Horizontal clock row より上部に，負の時は下部に配置されていることを表す．他の CR についても表 4.8 と同様の結果が得られた．表 4.8 の通り，Vertical clock row の遅延は最大 0.019ns であり，測定 8 や測定 9 の結果と比較して，極めて小さい．従って，GCB から FF へのクロック遅延の見積りでは，Vertical clock row の遅延を無視する． □

測定 8–10 の結果より，GCB から FF へのクロック遅延は Clock backbone と Horizontal clock row の遅延の和で計算するとする．ここで，$CR(r)$ と $X(r)$ はそれぞれレジスタ r が配置された CR，Clock backbone からレジスタ r までの水平距離を表す．ソースレジスタ r_s への SCD $SCD(r_s)$ は以下の式で求める．

$$SCD(r_s) = S_0(CR(r_s)) + C_{dc} \times X(r_s) \tag{4.8}$$

ここで，$S_0(CR_{(m,n)})$ は GCB から CR $CR_{(m,n)}$ への Clock backbone の最大遅延を表す．C_{dc} はクロック遅延係数を表し，測定 9 の結果に基づいて設定される．例えば，Virtex-7 では図 4.12 の傾きより $C_{dc} = 0.00263$ と定める．

同様に，デスティネーションレジスタ r_d への DCD $DCD(r_d)$ は以下の式で求める．

$$DCD(r_d) = D_0(CR(r_d)) + C_{dc} \times X(r_d) \tag{4.9}$$

ここで，$D_0(CR_{(m,n)})$ は GCB から CR $CR_{(m,n)}$ への Clock backbone の最小遅延を表す．

4.5.2 CPR

本稿では，FPGA 上でのクロックスキューの測定を行い，CPR の特性を明らかにする．

表 4.9: 同一 CR 内の CPR の測定結果 (**測定 11**).

Placement of source FF			Placement of destination FF			CPR [ns]
$CR_{(m,n)}$	Horizontal distance from clock backbone [slices]	Vertical position to horizontal clock row [slices]	$CR_{(m,n)}$	Horizontal distance from clock backbone [slices]	Vertical position to horizontal clock row [slices]	
$CR_{(0,4)}$	0	+0	$CR_{(0,4)}$	0	−0	0.320
$CR_{(0,4)}$	20	+0	$CR_{(0,4)}$	20	−0	0.320
$CR_{(0,4)}$	40	+0	$CR_{(0,4)}$	40	−0	0.320
$CR_{(0,4)}$	80	+0	$CR_{(0,4)}$	80	−0	0.320
$CR_{(0,4)}$	20	+20	$CR_{(0,4)}$	80	−10	0.320
$CR_{(0,4)}$	90	+20	$CR_{(0,4)}$	10	−20	0.320
$CR_{(0,4)}$	40	+10	$CR_{(0,4)}$	10	+20	0.320
$CR_{(1,5)}$	20	+20	$CR_{(1,5)}$	80	−10	0.328
$CR_{(1,5)}$	90	+20	$CR_{(1,5)}$	10	−20	0.328
$CR_{(1,5)}$	40	+10	$CR_{(1,5)}$	50	+0	0.328

表 4.10: 異なる CR 間の CPR の測定結果 (**測定 11**).

		Placement of source FF $CR_{(m,n)}$					
		$CR_{(0,3)}$	$CR_{(0,4)}$	$CR_{(0,5)}$	$CR_{(1,3)}$	$CR_{(1,4)}$	$CR_{(1,5)}$
Placement of destination FF $CR_{(p,q)}$	$CR_{(0,3)}$	0.313	0.227	0.227	0.240	0.227	0.227
	$CR_{(0,4)}$	0.227	0.320	0.240	0.227	0.240	0.240
	$CR_{(0,5)}$	0.227	0.240	0.350	0.227	0.240	0.270
	$CR_{(1,3)}$	0.240	0.227	0.227	0.291	0.205	0.205
	$CR_{(1,4)}$	0.227	0.240	0.240	0.205	0.298	0.218
	$CR_{(1,5)}$	0.227	0.240	0.270	0.205	0.218	0.328

ソース側とデスティネーション側の FF それぞれへのクロックパスが共通パスを持つ場合，そこに生じる遅延ばらつき（CPR）が考慮される．

測定 11 (CPR の特性)

CPR の特性を見つけるために，様々な場合で CPR の測定を行った．CPR はソース側 FF とデスティネーション側 FF を FPGA 上に配置し，クロックスキューを Vivado によって求めた際に出力される．

まず，CR $CR_{(m,n)}$ 内にソース側 FF とデスティネーション側 FF をランダムに配置し，CPR の測定を行った．表 4.9 に $CR_{(0,4)}$，$CR_{(1,5)}$ 内での CPR の測定結果を示す．表 4.9 の結果より，CPR は CR 内での位置には依存しないことがわかる．

次に，ソース側 FF とデスティネーション FF を CR $CR_{(m,n)}$，$CR_{(p,q)}$ に配置し，CPR を測定した．$0 \leq m, p \leq 1$，$3 \leq n, q \leq 5$ の範囲での CPR の測定結果を表 4.10 に示す．表 4.10 の結果より，CPR はソース側 FF とデスティネーション FF を配置した CR の組合せに依存することがわかる． □

測定 11 の結果より，本論文では表 4.10 のような CPR 表を作成する．表の各要素 $CT(CR_{(m,n)}, CR_{(m',n')})$ はソースレジスタが CR $CR_{(m,n)}$ に，デスティネーションレジスタが CR $CR_{(m',n')}$ に置かれた時の CPR の値を表す．

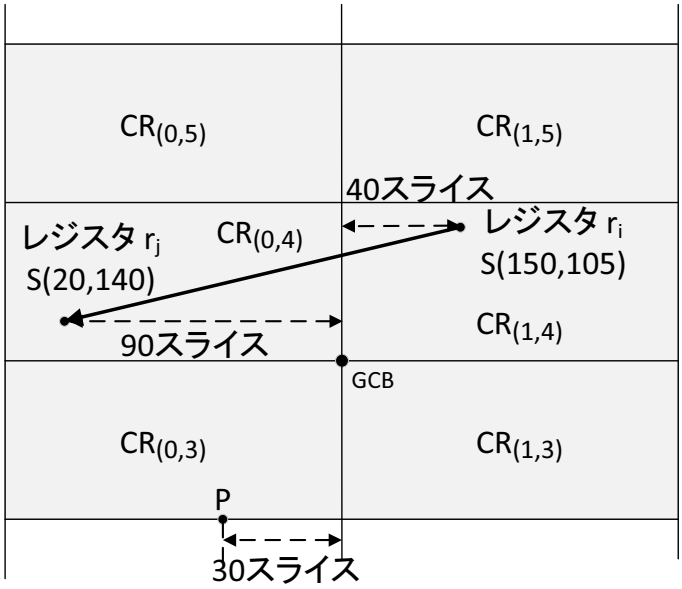


図 4.13: Virtex-7 上のデータパスの例.

4.5.3 CSEF の定式化と評価

本項では，前項までの議論に基づいて，クロックスキュー見積りモデル「CSEF」を定式化する．その後，FPGA 上のクロックスキューの実測値と CSEF による見積り値を比較し，見積り精度の評価を行う．

第 4.5.1 項，第 4.5.2 項の議論より，ソースレジスタ r_s からデスティネーションレジスタ r_d のパス $path(r_s, r_d)$ のクロックスキュー $CSK(path(r_s, r_d))$ は以下の式で求める．

$$\begin{aligned} CSK(path(r_s, r_d)) = DCD(r_d) - SCD(r_s) \\ + CT(CR(r_s), CR(r_d)). \end{aligned} \tag{4.10}$$

ここで，$CR(r_s)$ と $CR(r_d)$ はソースレジスタ r_s とデスティネーションレジスタ r_d が配置された CR を表す.

例 6 図 4.13 のような Virtex-7 上のパス $path(r_i, r_j)$ を想定する．ソースレジスタ r_i は CR $CR_{(1,4)}$ のスライス $S(150, 105)$ に，デスティネーションレジスタ r_d は CR $CR_{(0,4)}$ のスライス $S(20, 140)$ に配置されている．表 4.7 より $SCD(CR_{(1,4)}) = 3.596$，$DCD(CR_{(0,4)}) = 3.238$，表 4.10 より $CT(CR_{(1,4)}, CR_{(0,4)}) = 0.240$ がわかる．$CSK(path(r_i, r_j))$ は以下のように求める．

$$SCD(r_i) = 3.596 + 0.00263 \times 40 = 3.7012 \tag{4.11}$$

$$DCD(r_j) = 3.238 + 0.00263 \times 90 = 3.4747 \tag{4.12}$$

$$\begin{aligned} CSK(path(r_i, r_j)) &= 3.4747 - 3.7012 + 0.240 \\ &= 0.0135\text{ns}. \end{aligned} \tag{4.13}$$

□

表 4.11: クロックスキューの測定値と見積り値の比較 (**測定 12**).

Placement of source FF	Placement of destination FF	Mesured clock skew [ns]	Estimated clock skew by Eq. (4.10) [ns]	Estimated error [ns]
S(185,150)	S(135,150)	−0.195	−0.195	0.000
S(110,125)	S(110,75)	+0.008	−0.013	0.021
S(59,156)	S(160,145)	−0.146	−0.150	0.004
S(130,105)	S(190,140)	+0.157	+0.095	0.062
S(69,75)	S(69,181)	−0.271	−0.274	0.003
S(125,60)	S(89,130)	−0.284	−0.265	0.019
S(79,115)	S(160,55)	+0.069	+0.059	0.010
S(130,91)	S(84,176)	−0.123	−0.098	0.025
S(89,176)	S(120,130)	−0.156	−0.176	0.020
S(180,161)	S(104,120)	−0.362	−0.315	0.047

測定 12 (CSEF の評価) CSEF の有効性を評価するため，Virtex-7 (Speed Grade -2) 上でクロックスキューを測定し，CSEF（式 (4.10)）の見積り値と比較した．表 4.11 にランダムに配置された 10 本のパスでのクロックスキューの測定値と見積り値を示す．表 4.11 の結果より，CSEF の見積り誤差は最大 0.062ns，平均 0.021ns であり，後述の 4.6 節で示すベンチマーク回路の遅延や，前節の測定 7 における 100×100 スライス離れたパスでのクロックスキューの影響の最大値 1.2ns と比較して十分に小さい． □

4.6 IDEF と CSEF のフロアプラン指向 FPGA 高位合成への適用と評価

本節では，第 4.3 節で提案した「IDEF」，第 4.5 節で提案した「CSEF」の効果を検証するために，第 3 章で提案したフロアプラン指向 FPGA 高位合成手法 [29] に適用し，ベンチマークアプリケーションで実験を行う．

4.6.1 フロアプラン指向 FPGA 高位合成手法への適用

本稿では，第 3 章で提案したフロアプラン指向 FPGA 高位合成手法 [29] への IDEF, CSEF の適用方法について議論する．

第 3 章で提案したフロアプラン指向 FPGA 高位合成手法 [29] では，ハドル単位でモジュール配置を扱い，[1,2] に基づき，Simulated Annealing (SA) 法を用いて以下のコスト関数 *cost* に基づき最適化を図る．

$$cost = \alpha \frac{V}{T_{clock}} + \beta \frac{W}{W_{MAX}} + \gamma \frac{A_{BB}}{A_{total}} \tag{4.14}$$

ここで，T_{clock} は高位合成の入力で与えられたクロック周期制約，V はタイミング制約違反の合計値，W_{MAX} は見積った最大の配線長, W は合計の配線長, A_{total} は全ハドルの面積の和で求める総面積，A_{BB} は全てのハドルを包括する最小の矩形の面積を表す．式 (4.14) において，SA 法にて，タイミング制約違反が最も主要な項目であるため，パラメータ α はパラメータ β，γ に比べて，はるかに大きく設定される．従って，SA 法ではまずタイミング制約違反が 0 となるように最適化され，その上で，配線長・面積に関して最適化される．

式 (4.14) では，配線遅延の観点から配線長の最適化を図っているが，これは式 (3.5) に基づいて，ハドル間の縦と横の距離の和で求められる．しかし，第 4.2 節で示したように，FPGA では縦と横の配線遅延係数が異なるため，縦と横を等価値に扱う式 (4.14) は配線遅延最適化の観点でふさわしくない．また，回路のレイテンシに影響を与えるのは，クリティカルパスであるため，高位合成段階でも全てのパスの合計を最適化するのではなく，クリティカルパスおよびその候補となるパスに集中して最適化するのが望ましい．

第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に “IDEF” と “CSEF” を適用し，クリティカルパスおよびその候補となるパスの配線遅延・クロックスキューを最適化するため，以下の新たしいコスト関数 $cost_{new}$ を用いる．

$$cost_{new} = \alpha\frac{V}{T_{clock}} + \beta\frac{ID_n}{T_{clock}} + \gamma\frac{CSK_m}{T_{clock}} + \delta\frac{A_{BB}}{A_{total}}. \tag{4.15}$$

ここで，第 1 項と第 4 項は式 (4.14) の第 1 項と第 3 項と同様である．そして，式 (4.15) でも同様にタイミング制約違反を最も主要な項目と考え，パラメータ α はパラメータ β，γ，δ をはるかに大きく設定した [3]．第 2 項と第 3 項においてクリティカルパスとその候補のパス遅延の最適化を図る．

式 (4.15) の第 2 項，第 3 項について述べる前に，各々のパスの遅延の求め方を議論する．Xilinx Vivado Design Suite 2014.2 では，データをデスティネーション側の FF へ正しく格納するために，回路内の全てのパスは以下の式を満たす必要がある．

$$\begin{aligned}&\text{ロジック遅延} + \text{配線遅延} - \text{クロックスキュー}\\ &\quad + \text{クロックのジッター} \leq \text{クロック周期}\end{aligned} \tag{4.16}$$

式 (4.16) においてクロックのジッターは Vivado では FPGA チップ毎に定数で与えられる．

一方で，HDR アーキテクチャでは高位合成段階で，FU，レジスタ，コントローラのハドル内での位置を決定しない．従って，第 3.2 節で示したように，配線遅延・クロックスキューの見積りは対象のモジュールが所属するハドルの位置に基づいて求められる．ここで，$S(h)$ はハドル h の右上のポイントの FPGA 上でのスライス座標を表すとする．

$path(r_s, r_d)$ は HDR アーキテクチャのソースレジスタ r_s とデスティネーションレジスタ r_d の間に FU f を持つ 1 つのデータパスを表すとする．ここで，ソースレジスタ r_s はソース側ハドル h_s に，デスティネーションレジスタ r_d はデスティネーション側ハドル h_d に割り当てられているとする．HDR アーキテクチャでは，パス $path(r_s, r_d)$ に含まれる FU f は入力となるソースレジスタ r_s と同じハドル h_s に所属する．$D(path(r_s, r_d))$ は，HDR アーキテクチャのパス $path(r_s, r_d)$ の遅延は式 (4.16) を参考に，以下の式で求める．

$$\begin{aligned}D(path(r_s, r_d)) &= D_{fu}(f) + D_{reg} + D_{mux}\\ &\quad + D_e(S(h_s), S(h_d)) - CSK_{HDR}(path(r_s, r_d))\end{aligned} \tag{4.17}$$

ここで，$D_{fu}(f)$，D_{reg}，D_{mux} は FU の遅延，レジスタの遅延，MUX の遅延を表す．$D_e(S(h_s), S(h_d))$ はハドル h_s と h_d 間の見積り配線遅延を表し，IDEF(式 (4.4)) で求められる．式 (4.17) の

[3] [3] に従って，まず $\alpha = 100$，$\delta = 1$ と設定した．そして，式 (4.15) の第 2 項と第 3 項の大きさを揃えるため，$\beta = 1$，$\gamma = -10$ と設定した．

$CSK_{HDR}(path(r_s, r_d))$ は，レジスタ r_s，r_d の位置をそれぞれハドル h_s，h_d で評価した際に，CSEF(式 (4.10)) で求めるクロックスキューの見積り値とする．

例 6 図 4.14 に Virtex-7 上の HDR アーキテクチャのデータパス $path(r_i, r_j)$ を示す．図 4.14 において，MUX は省略されている．ソース側ハドル h_i の右上のスライスはクロック領域 $CR_{(1,4)}$ 内のスライス $S(150, 105)$，デスティネーション側ハドル h_j の右上のスライスはクロック領域 $CR_{(0,4)}$ 内のスライス $S(20, 140)$ であるとする．ソースレジスタ r_i はハドル h_i，デスティネーションレジスタ r_j はハドル r_j に割り当てられ，それぞれの位置は所属するハドルにの位置に基づいて，$S(h_i)$，$S(h_j)$ とする．また，表 4.7 より $SCD(CR_{(1,4)}) = 3.596$，$DCD(CR_{(0,4)}) = 3.238$，表 4.9 より $CT(CR_{(1,4)}, CR_{(0,4)}) = 0.240$ とする．そして，図 4.12 より式 (4.8)，式 (4.9) において $C_{dc} = 0.00263$ と設定する．

ここで，$CSK_{HDR}(path(r_i, r_j))$ は以下の通り，計算される．

$$SCD(r_i) = 3.596 + 0.00263 \times 40 = 3.7012 \tag{4.18}$$

$$DCD(r_j) = 3.238 + 0.00263 \times 90 = 3.4747 \tag{4.19}$$

$$\begin{aligned} CSK_{HDR}(path(r_i, r_j)) &= 3.4747 - 3.7012 + 0.240 \\ &= 0.0135\text{ns}. \end{aligned} \tag{4.20}$$

次に，式 (4.4) において $C_{dh} = 0.0167$，$C_{dv} = 0.0130$ と設定し，$D_e(S(h_i), S(h_j))$ は以下のように計算される．

$$\begin{aligned} D_e(S(h_i),S(h_j)) &= 0.0167 \times |20 - 150| \\ &\quad + 0.0130 \times |140 - 105| = 2.626\text{ns}. \end{aligned} \tag{4.21}$$

そして，$D_{fu}(f) = 1.45$ns，$D_{reg} = 0.45$ns，$D_{mux} = 0.96$ns とした時，$D(path(r_i, r_j))$ は以下のように求められる．

$$\begin{aligned} D(path(r_i, r_j)) &= 1.45 + 0.45 + 0.96 \\ &\quad + 2.626 - 0.0135 = 5.4725\text{ns}. \end{aligned} \tag{4.22}$$

□

次に，式 (4.15) の第 2 項，第 3 項について議論する．式 (4.15) の第 2 項はクリティカルパスおよびクリティカルパスの候補のパスの配線遅延のコストを表す．ID_n は式 (4.4) で計算される，クリティカルパスおよびクリティカルパスの候補のパスの配線遅延の合計を表す．式 (4.15) の第 3 項はクリティカルパスおよびクリティカルパスの候補のパス（最も見積り遅延の大きい n 本のパス）のクロックスキューのコストを表す．CSK_m は式 (4.10) で計算される，クリティカルパスおよびクリティカルパスの候補のパス（最も見積り遅延の大きい m 本のパス）の式 (4.17) のクリティカルパスの見積り値 $CSK_{HDR}(path(r_s, r_d))$ の合計を表す．[4]

第 3 章のフロアプラン指向高位合成手法では，フロアプランフェーズの後，フロアプラン情報をスケジューリング/FU バインディング，レジスタバインディング，コントローラ合成のフェーズにフェードバックする．第 3 章の手法では，単純なモジュール間の距離に基づいてモジュール間のデータ転送遅延を見積もっていたが，IDEF・CSEF を適用したフロアプラン指向高位合成手法では，式 (4.17) を用いてモジュール間のデータ転送遅延を見積る．

[4]式 (4.15) のパラメータ n と m の設定方法は，第 4.6.3 で後述する．

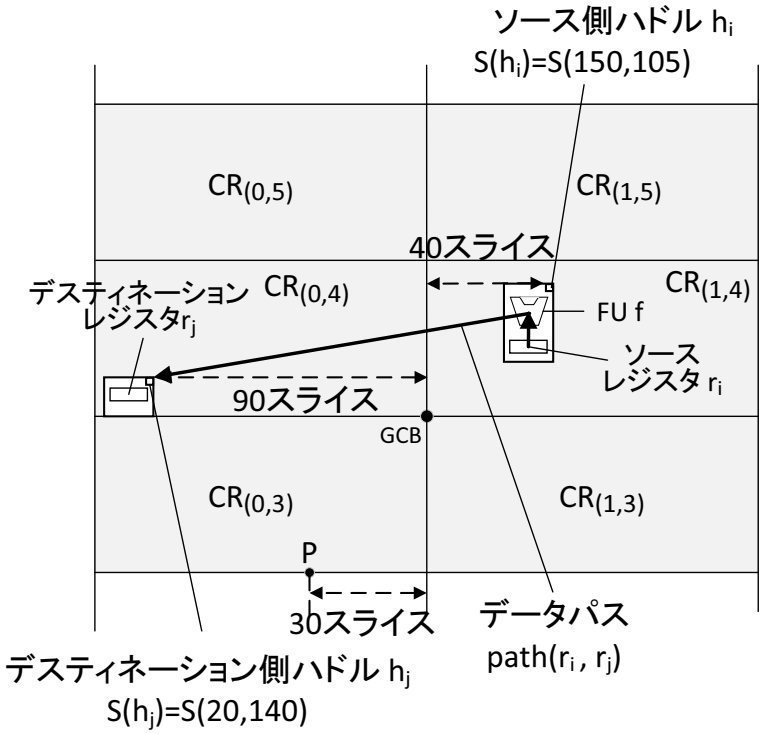


図 4.14: Virtex-7 上の HDR アーキテクチャのデータパスの例.

表 4.12: 高位合成の FU/レジスタ情報.

FUs/register	#Slices	Delay [ns]
Adder	4	1.45
Substracter	4	1.45
Multiplier	36	4.3
Divider	97	28
Comparator	2	1.44
Right shifter	12	2.1
AND	9	0.79
Register	4	0.45
Memory write	1	1.8

4.6.2 実験環境

本稿では，実験方法および実験環境について述べる．

前節で IDEF, CSEF を適用した高位合成アルゴリズムを C++言語で実装し，ベンチマークアプリケーションに適用した．設計フローでは，まず対象の FPGA 上で配線遅延とクロックスキューを測定し，式 (4.4) と式 (4.10) の見積もりモデル式を構築する．本節の実験では，Xilinx 社の Virtex-7 (xc7vx485tffg1761-2) を対象 FPGA とする．IDEF, CSEF を設定し適用した高位合成アルゴリズムで，各々のベンチマークアプリケーションに対し，RTL 記述とハドル配置情報を得る．次に，Xilinx 社 Vivado Design Suite 2014.2 で論理合成を行い，ゲー

トレベルの回路情報を得る．この時，DSP ブロックの使用を禁止した．そして，高位合成で得たハドル配置情報に基づいて，配置・配線を行う．図 4.13 の点 P は，Clock backbone から水平方向に 30 スライス離れた $CR_{(0,3)}$ の底辺の点である．この点 P にそれぞれのアプリケーション回路の左下の点が重なるよう配置した．

Vivado にて論理合成・配置配線を行う際，始め十分に大きいクロック周期制約を与えた．そして，タイミング制約違反が発生するまで，徐々に小さくしていく．タイミング制約違反が発生する直前のクロック周期制約 T_{CLK} にて，実施した時の Vivado の STA レポートで得られる $slack\ time$ より，$(T_{CLK} - slack\ time)$ で計算される値を合成した値をアプリケーション回路の最小クロック周期とした．

最後に，ビットファイル化し，FPGA 上の HDR アーキテクチャ回路が実装される．

対象アプリケーションとして，HAL（ノード数 11），PARKER（ノード数 22，条件分岐あり），DCT（ノード数 48），jacobi（ノード数 52），7 次 FIR フィルタ（ノード数 75），EWF3（ノード数 102），copy（ノード数 378，条件分岐あり）を用いた．実験で用いたライブラリの情報を表 4.12 に示す．表 4.12 の値は，それぞれの回路要素を virtex-7 上に実装し求めた．FU の扱う値は全て 16 ビットである．各 FU の実行サイクル数は 1 とした．提案手法の実行に使用した計算機環境は，AMD Opteron 2360SE 2.5GHz×2，メモリが 16GB である．論理合成，配置・配線に使用した計算機環境は，Intel Core i5-2520M 2.5GHz×2，メモリが 4GB である．

比較する従来手法として SR と第 3 章の手法 [29] を用意した．SR は第 3 章と同様のアルゴリズムで，フロアプランを考慮しない手法である．また，第 3 章の手法 [29] では，同様に 60 スライス分の距離あたり 1ns とした．

4.6.3 Parameter Setting

本稿では，式 (4.15) のパラメータ n，m の適切な値を求めるため，ベンチマークアプリケーションを用いて計算機実験を行う．

HDR アーキテクチャを対象としたフロアプラン指向 FPGA 高位合成手法では，第 3.2 節で述べたように，最初に初期フェーズで各 FU に対して，1 つのハドルを用意する．そして，スケジューリング，バインディング，フロアプラン，ハドルの統合/分割を繰り返していく．この過程で，それぞれのイタレーションでハドルの数は変化する．ここで，初期フェーズで各 FU に用意したハドルの総数を N_{fu} とする．そして，HDR アーキテクチャの全てのデータパスの数 N_p を $N_{fu} \times N_{fu}$ で求める値とする．

各ベンチマークアプリケーションに対し，式 (4.15) のパラメータ n，m の適切な値を求めるため，IDEF, CSEF，両方を適用したフロアプラン指向 FPGA 高位合成手法でパラメータ n，m の値を変化させて，各々のアプリケーション回路を合成した．この時，単純のため，パラメータ n と m の値を同じにした．式 (4.4)，式 (4.8)，式 (4.9) において，$C_{dh} = 0.0167$, $C_{dv} = 0.0130$，$C_{dc} = 0.00263$ と設定した．

表 4.13 にパラメータ n，m の値を変化させて合成した結果を示す．表 4.13 の結果より，それぞれのアプリケーションで最小のレイテンシの回路を得られたパラメータ n，m（表 4.13 の太字箇所）を採用し，n/N_p，m/N_p の値の範囲は 12%–23%となった．以上より，n/N_p, m/N_p の値をおおよそ 20%付近に設定すると，最小のレイテンシの回路が得られることがわ

表 4.13: パラメータ n，m を変えた場合の合成結果.

App. (#Nodes)	FU constraint	Clock period constraint [ns]	N_p	(n,m)	The rate of n/N_p and m/N_p	#Huddles	Delay [ns]	#Steps	Latency [ns]	#Slices	#LUTs	#FFs	Total synthesis time [s]
hal	Add×1, Sub×1,	5	25	**(5,5)**	**20%, 20%**	2	3.947	5	**19.74**	117	416	150	300
(11)	Mul×2,Comp×1			(10,10)	40%, 40%	2	4.034	5	20.17	116	416	150	395
parker	Add×2,Sub×2,	4	25	**(5,5)**	**20%, 20%**	2	1.685	8	**13.48**	81	226	266	374
(22)	Comp×1			(10,10)	40%, 40%	2	1.740	8	13.92	75	226	266	454
	Add×4,			(5,5)	8%, 8%	4	4.504	8	36.03	338	1164	530	439
	Mul×4			**(10,10)**	**16%, 16%**	3	4.490	8	**35.92**	351	1154	515	429
dct		7	64	(15,15)	23%, 23%	4	4.562	8	36.50	354	1160	530	418
(48)				(20,20)	31%, 31%	7	4.552	8	36.18	426	1386	640	452
				(25,25)	39%, 39%	3	4.502	8	36.02	378	1157	515	451
				(30,30)	47%, 47%	4	4.520	8	36.16	342	1158	530	430
	Add×2,			(5,5)	10%, 10%	2	17.48	17	297.2	330	1084	315	387
jacobi	Sub×1,	30	49	**(10,10)**	**20%, 20%**	3	17.47	17	**297.0**	340	1102	352	301
(52)	Mul×2, Div×2			(15,15)	31%, 31%	3	17.53	17	297.9	342	1126	355	442
				(20,20)	41%, 41%	2	17.56	17	298.5	336	1151	354	413
	Add×4,			(5,5)	6%, 6%	3	4.399	15	65.99	331	1012	366	422
	Mul×4,			**(10,10)**	**12%, 12%**	3	4.267	15	**64.01**	318	1037	365	407
	Mem×1			(15,15)	19%, 19%	6	4.480	15	67.20	328	1095	472	467
fir		7	81	(20,20)	25%, 25%	6	4.512	15	67.68	357	1162	491	496
(75)				(25,25)	31%, 31%	6	4.452	15	66.78	361	1164	507	481
				(30,30)	37%, 37%	5	4.343	15	65.15	359	1149	474	509
				(35,35)	43%, 43%	6	4.383	15	65.75	352	1184	508	511
				(40,40)	49%, 49%	3	4.357	15	65.36	314	1052	399	501
	Add×4,			(5,5)	8%, 8%	5	4.233	40	169.3	604	1898	726	428
	Mul×4			(10,10)	16%, 16%	5	4.144	40	165.8	567	1893	727	536
ewf3		7	64	**(15,15)**	**23%, 23%**	4	4.030	40	**161.2**	415	1562	583	495
(102)				(20,20)	31%, 31%	5	4.310	40	172.4	535	1812	707	514
				(25,25)	39%, 39%	3	4.403	40	176.1	494	1560	557	390
				(30,30)	47%, 47%	3	4.375	40	175.0	466	1567	557	547
	Add×3, Sub×1,			(10,10)	5%, 5%	7	5.262	82	431.5	1732	5102	3263	931
	Mul×5, Comp×1,			(20,20)	10%, 10%	10	5.439	82	446.0	2096	6080	3650	1211
	Rshift×2, AND×1,			**(30,30)**	**15%, 15%**	7	5.241	82	**429.8**	1774	5243	3184	1231
copy	Mem×1	8.5	196	(40,40)	20%, 20%	8	5.325	82	436.7	1857	5516	2919	1262
(378)				(50,50)	26%, 26%	9	5.258	82	431.2	1969	5565	3632	1338
				(60,60)	31%, 31%	8	5.329	82	437.0	1832	5254	3494	1544
				(70,70)	36%, 36%	5	5.500	82	451.0	1692	4851	3004	1482
				(80,80)	41%, 41%	5	5.481	82	449.4	1691	4751	2935	1583
				(90,90)	46%, 46%	4	5.443	82	446.3	1647	4442	2486	1681

かる．これは，n/N_p，m/N_p の値が大きすぎる時，ID_n と CSK_m は多くのクリティカルパス以外のデータパスの値を含み，クリティカルパスを最適化しきれないためと考えられる．

4.6.4 計算機実験

表 5.2 に計算機実験結果を示す．1 列目の App. は入力アプリケーション，2 列目は各アプリケーションの FU 制約を表す．例えば，Add × 2 は加算器 2 つ，Sub × 1 は減算器 1 つを表す．Mul は乗算器，Div は除算器，Comp は比較器，Rshift は右シフト器，Mem はメモリユニットを表す．演算器制約は第 3.5 と同様に設定した．3 列目は各アプリケーションのクロック周期制約を表す．4 列目は使用した手法を表す．今回使用した手法は SR，従来手法 [2]，提案手法である．5 列目は提案手法と [2] のハドル数を表す．6 列目は回路を配置配線した際の最小クロック周期として得られた値を，7 列目はコントロールステップ数を，8 列目は $Delay \times CS$ 数 で求めたレイテンシを表す．9 列目は回路全体のスライス数を，10 列目は回路全体の LUT 数，11 列目は回路全体の FF 数を表す．12 列目は高位合成，論理合成，配置配線で要した計算機時間の合計を表す.

表 5.2 の 7 列目より，それぞれのアプリケーションで 5 つの手法はほぼ同じ値となった．一

表 4.14: 5 つの手法比較の計算機実験結果.

App. (#Nodes)	FU constraint	Clock period constraint [ns]	Applied algorithm	#Huddles	Delay [ns]	#Steps	Latency [ns]	#Slices	#LUTs	#FFs	Total synthesis time [s]
hal (11)	Add×1, Sub×1, Mul×2, Comp×1	5	SR	-	3.869	5	19.35	100	380	108	446
			[29]	2	3.956	5	19.78	111	414	150	361
			[29]+IDEF	2	3.946	5	19.73	113	416	150	404
			[29]+CSEF	2	3.954	5	19.77	119	434	166	393
			[29]+IDEF+CSEF	2	3.947	5	19.74	117	416	150	300
parker (22)	Add×2, Sub×2, Comp×1	4	SR	-	1.987	7	13.91	64	186	224	279
			[29]	2	1.791	8	14.33	77	231	266	395
			[29]+IDEF	2	1.739	8	13.91	85	228	266	394
			[29]+CSEF	2	1.740	8	13.92	75	226	266	401
			[29]+IDEF+CSEF	2	1.685	8	13.48	81	226	266	374
dct (48)	Add×4, Mul×4	7	SR	-	4.905	8	39.24	412	1383	345	414
			[29]	4	4.672	8	37.38	373	1168	530	333
			[29]+IDEF	4	4.494	8	35.95	348	1161	529	481
			[29]+CSEF	6	4.541	8	36.33	385	1298	597	481
			[29]+IDEF+CSEF	3	4.490	8	35.92	351	1154	515	429
jacobi (52)	Add×2, Sub×1, Mul×2, Div×2	30	SR	-	18.20	17	309.3	364	1168	271	394
			[29]	3	17.84	17	303.3	337	1113	351	314
			[29]+IDEF	2	17.55	17	298.4	349	1084	315	427
			[29]+IDEF	2	17.62	17	299.6	340	1084	315	309
			[29]+IDEF+CSEF	3	17.47	17	297.0	340	1102	352	301
fir (75)	Add×4, Mul×4, Mem×1	7	SR	-	5.330	15	79.95	435	1535	307	666
			[29]	5	4.723	15	70.85	282	1029	454	405
			[29]+IDEF	3	4.336	15	65.04	321	1066	382	453
			[29]+CSEF	6	4.495	15	67.43	369	1170	508	524
			[29]+IDEF+CSEF	3	4.267	15	64.01	318	1037	365	407
ewf3 (102)	Add×4, Mul×4	7	SR	-	5.175	40	207.0	494	1603	363	576
			[29]	5	4.537	40	181.5	468	1794	691	442
			[29]+IDEF	3	4.372	40	174.9	478	1571	557	493
			[29]+CSEF	3	4.383	40	175.3	472	1547	557	499
			[29]+IDEF+CSEF	4	4.030	40	161.2	415	1562	583	495
copy (378)	Add×3, Sub×1, Mul×5, Comp×1, Rshift×2, AND×1, Mem×1	8.5	SR	-	6.380	82	523.2	1437	5100	1557	878
			[29]	8	5.625	82	461.3	1799	5241	3453	855
			[29]+IDEF	8	5.367	82	440.1	1916	5353	3194	1284
			[29]+CSEF	7	5.451	82	447.0	1777	5032	3335	1262
			[29]+IDEF+CSEF	7	5.241	82	429.8	1774	5243	3184	1231

方で，6 列目より最小クロック周期は各々の手法で異なる．8 列目より，第 3 章の手法 [29] に IDEF のみを適用した手法（ [29]+IDEF）は，SR と比較して回路のレイテンシを最大 19%，平均 9%削減し，第 3 章の手法 [29] と比較して最大 8%，平均 4%削減した．第 3 章の手法 [29] に CSEF のみを適用した手法（ [29]+CSEF）は，SR と比較して回路のレイテンシを最大 16%，平均 8%削減し，第 3 章の手法 [29] と比較して最大 5%，平均 3%削減した．そして，第 3 章の手法 [29] に IDEF と CSEF を適用した手法（ [29]+IDEF+CSEF）は，SR と比較して回路のレイテンシを最大 22%，平均 10%削減し，第 3 章の手法 [29] と比較して最大 11%，平均 6%削減した．

IDEF, CSEF を適用した 3 つの手法では，第 3 章の手法 [29] に比べて全てのアプリケーションでレイテンシを削減できたが，SR と比べていくつかのアプリケーションでレイテンシが増加してしまった．これは，hal や parker のような小規模アプリケーションでは，生成される回路も小さく，クリティカルパスにおける配線遅延・クロックスキューの影響も小さいためと考えられる．実際，[29]+IDEF+CSEF の手法は，fir, ewf3, copy の 3 つの大規模アプリケーションに限ってでは，SR と比較して平均 20%，[29] と比較して平均 9%削減し，より効果的にレイテンシを削減できていることが確認できる．これは，これらのアプリケーションでは，クリティカルパスにおける配線遅延・クロックスキューの影響が大きいためと考える．

また，表 5.2 の 10 列目より，[29]+IDEF+CSEF の手法は第 3 章の手法 [29] とほぼ同程度

のスライス数を実現した．[12]+IDEF の手法，[12]+CSEF の手法は第 3 章の手法 [29] と比べて平均 5%スライス数が増加した．

SR と比較して，[29]+IDEF+CSEF の手法は最大 27%，平均 6%削減したが，一部のアプリケーション（hal, parker, copy）でスライス数が増加した．HDR アーキテクチャはレジスタ分散型アーキテクチャであり，SR に比べてレジスタ数が多いことが，主な理由と考えられる．

以上より，本章で提案した 2 つの見積りモデル “IDEF”，“CSEF” はより高精度な配線遅延・クロックスキューの見積りを実現し，合成される回路のレイテンシ削減に寄与することがわかる．

4.7 提案タイミングモデルの対象 FPGA

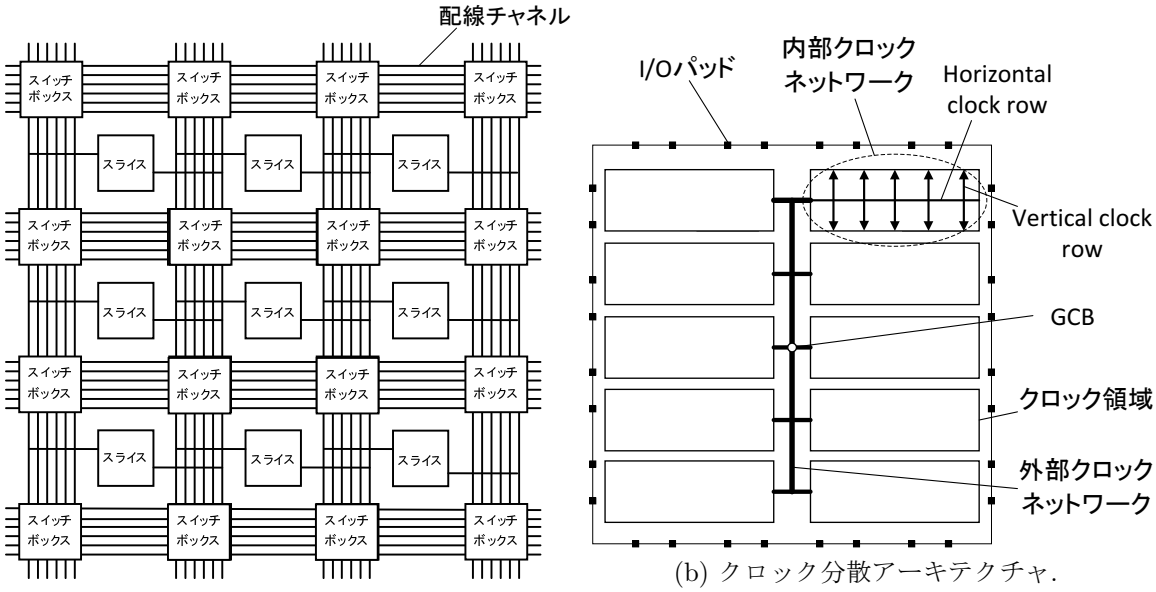
本節では，定性的な議論を行い，「IDEF」,「CSEF」の対象 FPGA アーキテクチャを明らかにする．本章で提案した 2 つのタイミングモデルは，図 6.1(a)，(b) に示すようなクロック分散アーキテクチャを持つアイランド型 FPGA を対象とする．

図 6.1(a) は [60] で述べているアイランド型 FPGA のセクションを示す．アイランド型 FPGA では，スライスの周囲を水平方向・垂直方向の配線チャネルが囲んでいる．全ての配線チャネルは，数本の配線とそれら各々を接続するスイッチボックスで構成される．また，図 6.1(b) に示すようなクロック分散アーキテクチャを持つ FPGA を対象とする．対象の FPGA は，いくつかのクロック領域を持ち，クロックネットワークが予め構築されている．クロックネットワークは，2 つの部分 “内部クロックネットワーク” と “外部クロックネットワーク” で構成される．内部クロックネットワークは，クロック領域内部のクロック配線で，1 つの horizontal clock row と数本の vertical clock row で構成される．外部クロックネットワークは，GCB と各クロック領域の間を結ぶクロック配線である．クロック信号は GCB から各クロック領域に分配される．

IDEF はスライス座標系を利用するため，アイランド型 FPGA に適用することが可能である．また，CSEF は主に次の 2 つの部分で成り立っている：(i) 式 (4.8) と式 (4.9) の第 1 項に現れる外部クロックネットワークの伝搬遅延，(ii) 式 (4.8) と式 (4.9) の第 2 項に現れる内部クロックネットワークの伝搬遅延である．対象の FPGA が図 6.1(b) に示すようなクロック分散機構を持つ時，そのクロックネットワークは CSEF の思想とマッチしており，適用できる．

4.8 本章のまとめ

本章では，FPGA を対象とした高位合成における課題 2「高位合成段階における，配線遅延・クロックスキューの正確な見積り」を解決するため，フロアプラン指向 FPGA 高位合成手法のための FPGA の配線遅延・クロックスキュー見積りモデル「IDEF」と「CSEF」を提案した．まず，FPGA 上で様々なパターンで配線遅延を測定し，FPGA の配線遅延特性を明らかにした．そして，その測定結果を基に，FPGA の配線遅延見積りモデル “IDEF” を構築した．次に，FPGA のクロックスキューの影響を測定し，高位合成段階で考慮すべき要素であることを明らかにした．そして，Xilinx 社 Vivado 内のクロックスキューの計算モデル



(a) アイランド型 FPGA のセクション [60].

(b) クロック分散アーキテクチャ.

図 4.15: 対象のアイランド型 FPGA.

を基に，計算モデル内の各部分の特性を測定から導き，FPGA のクロックスキュー見積りモデル “CSEF” を構築した．

Xilinx 社 Vivado のタイミングモデルと比較して，IDEF は最大誤差 0.20ns，平均誤差 0.10ns，CSEF は最大誤差 0.062ns，平均誤差 0.021ns という見積り精度を達成し，高位合成段階で高精度に見積もれることを確認した．

その後，IDEF・CSEF を第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に適用し，ベンチマークアプリケーションにおいて効果を確認した．見積りモデルを適用した手法は，SR と比較して回路のレイテンシを最大 22 %，平均 10%削減し，第 3 章の手法と比較して最大 11%，平均 6%削減した．また，大規模アプリケーションに限ってでは，SR と比較して平均 20%，第 3 章の手法と比較して平均 9%削減し，より効果的にレイテンシを削減できた．この実験結果より，提案した見積りモデル「IDEF」と「CSEF」をフロアプラン指向 FPGA 高位合成手法に適用することで，よりレイテンシの小さい高性能な回路を合成できることが確認できた．

第5章　配線遅延とクロックスキューを考慮したクリティカルパス最適化FPGA高位合成手法

5.1　本章の概要

本章では[1]，FPGAを対象とした高位合成における課題3「高位合成段階での配線遅延・クロックスキューの影響とMUXのコストの同時考慮」を解決するため，第3章で提案したフロアプラン指向FPGA高位合成手法に前章で提案したIDEF・CSEFを利用し，配線遅延とクロックスキューを考慮したクリティカルパス最適化FPGA高位合成手法を提案する．

提案手法は，前章で提案したIDEF・CSEFをフロアプラン指向高位合成全体で利用し，高位合成段階で配線遅延・クロックスキューの影響を含めたデータパスの遅延を見積り，全データパスのうちクリティカルパスおよびその候補となるパスを特定する．そして，データパス生成とフロアプラン両方において，これらを集中的に最適化し回路のレイテンシ—の向上を図る．

データパス生成とフロアプランでクリティカルパス遅延を削減するために，以下のバインディング手法とフロアプラン手法を提案する．

- クリティカルパス指向スケジューリング/FUバインディング
- クリティカルパス指向ハドル合成/フロアプラン

クリティカルパス指向スケジューリング/FUバインディングでは，フロアプラン情報を基にFUバインディングを改良することで，配線遅延とクロックスキューを改善しクリティカルパス遅延の小さいデータパスの生成を図る．また，クリティカルパス指向ハドル合成/フロアプランではフロアプランの改善において配線遅延とクロックスキューを考慮して，クリティカルパスの候補となるパスを集中的に最適化を行う．そして，クリティカルパス遅延の小さいフロアプラン解を目指す．最後に，計算機実験により提案手法の有効性を示す．

5.2　提案アルゴリズム

本節では，配線遅延とクロックスキューを考慮したクリティカルパス最適化FPGA高位合成手法を提案する．提案手法は，第3章で提案したHDRアーキテクチャを対象としたMUX削減FPGA高位合成手法に対して，IDEF・CSEFを高位合成全体に適用する．そして，高位合成段階で配線遅延・クロックスキューの影響を含めたデータパスの遅延を見積り，全データ

[1]本章の内容は [31,33,34] による．

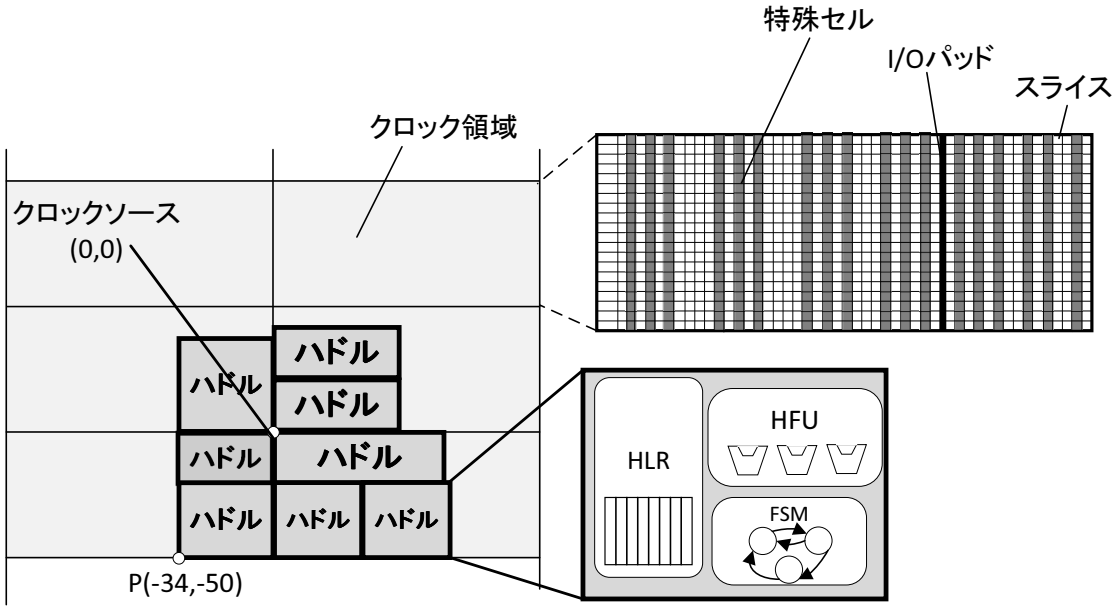


図 5.1: FPGA 上の HDR アーキテクチャ.

パスのうちクリティカルパスおよびその候補となるパスを特定する．特定したクリティカルパスおよびその候補となるパスを最適化するため，新たなバインディング手法「クリティカルパス指向スケジューリング/FU バインディング」と，新たなフロアプラン手法「クリティカルパス指向ハドル合成/フロアプラン」を提案する．提案手法は，データパス生成とフロアプラン両方において，これらを集中的に最適化し回路のレイテンシ―の向上を図る．

第3章で提案した HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法では，一般的な見積りモデルを用いて FPGA の配線遅延のみを考慮しており，第 4.2 節で述べた FPGA の配線遅延特性を考慮しておらず，配線遅延の見積もり誤差を多く含んでしまう可能性がある．また，同様に FPGA のクロックスキューを考慮しておらず，第 4.4 節で述べたようにクロックスキューによってクリティカルパス遅延が増大し，レイテンシが悪化する恐れがある．また，第3章の手法は全パスを等価に扱い最適化を図っており，クリティカルパスに着目していない．このため，クリティカルパス遅延を削減できず同様の結果を招く可能性がある．そこで，本章では HDR アーキテクチャを対象とした配線遅延とクロックスキューを考慮するクリティカルパス最適化 FPGA 高位合成手法を提案する．

5.2.1 対象アーキテクチャとパス遅延の見積り

本項では，IDEF と CSEF を用いた高位合成段階でのパス遅延の見積りを定式化する．

本章では，第3章と同様に HDR アーキテクチャ[1–3] を対象とする．(図 5.1)

また，本章ではレジスタ r_s から FU f を通りレジスタ r_d へデータ転送するパス $path(r_s, r_d)$ の遅延 $D(path(r_s, r_d))$ を 4.6.1 項の式 (4.17) と同様に以下のように求める．

$$\begin{aligned} D(path(r_s, r_d)) &= D_{fu}(f) + D_{reg} + D_{mux} \\ &\quad + D_e(S(h_s), S(h_d)) - CSK_{HDR}(path(r_s, r_d)) \end{aligned} \tag{5.1}$$

ここで，$D_{fu}(f)$，D_{reg}，D_{mux}はFUの遅延，レジスタの遅延，MUXの遅延を表す．$D_e(S(h_s), S(h_d))$はハドルh_sとh_d間の見積り配線遅延を表し，IDEF(式(4.4))で求められる．式(4.17)の$CSK_{HDR}(path(r_s, r_d))$は，レジスタ$r_s$，$r_d$の位置をそれぞれハドル$h_s$，$h_d$で評価した際に，CSEF(式(4.10))で求めるクロックスキューの見積り値とする．各レジスタのハドル内での位置は第3章と同様にそれぞれのハドルの右上と仮定する．

そして，本章では上記の設定の下，第3章と同様に以下の問題の解決を図る．

定義2 HDRアーキテクチャを対象としたFPGA高位合成問題とは，CDFG，クロック周期制約，演算器制約が与えられた時，回路のレイテンシを最小化するようにCDFGをスケジューリングおよびバインディングし，各FUをハドルに割り当て，レジスタ・コントローラを合成し，各ハドルを配置することである．レイテンシは最小クロック周期 × コントロールステップ数（CS数）で計算する． □

5.2.2 全体フロー

本項では，提案アルゴリズムの全体フローを示し，議論する．

図5.2に提案アルゴリズムの全体フローを示す．提案手法は配線遅延とクロックスキューを大きなコストとして考慮する．フロー全体を通して配線遅延とクロックスキューを含めた各パス遅延を見積り，パス遅延の大きいクリティカルパスの候補となるパスを特定する．データパス生成とフロアプランの両方において，これらに対して集中的に最適化を図ることで，クリティカルパス遅延を削減し，回路のレイテンシ削減を図る．データパス生成とフロアプランでクリティカルパスの配線遅延とクロックスキューを改善するために，以下のスケジューリング/FUバインディング手法とフロアプラン手法を新たに提案する．

条件1 （フェーズ2）クリティカルパス指向スケジューリング/FUバインディング

条件2 （フェーズ4）クリティカルパス指向ハドル合成/フロアプラン

クリティカルパス指向スケジューリング/FUバインディングでは，まずモジュール配置情報を基に，配線遅延を考慮した上で総CS数を最小化するように入力のCDFGをスケジューリングおよび演算をFUへ割り当てて初期解を得る．その後，各ハドル間の配線遅延とクロックスキューを見積り各パスの遅延を基に，クリティカルパスになりそうな演算ノードに対して，FUバインディングの変更により配線遅延とクロックスキューを改善し，クリティカルパス遅延を削減する．最終的に，CS数を最小化した上で，クリティカルパス遅延の削減を図る．

クリティカルパス指向ハドル合成/フロアプランでは，モジュール配置より配線遅延とクロックスキューを見積り，パス遅延の大きいクリティカルパスの候補となるパスを特定する．そして，SA（Simulated Annealing）法によりこれらのパスを集中的に最適化し，クリティカルパス遅延の小さいフロアプラン解を実現する．これら2つの手法によってクリティカルパス遅延を削減し，回路の最小クロック周期の削減を図る．

提案アルゴリズムでは，入力としてCDFG，クロック周期制約，演算器制約，対象FPGAの情報を与え，RTL回路の記述とハドルの構成・配置情報を出力する．第4章のように対象のFPGA上で配線遅延・クロックスキューを測定し，IDEF (式(4.4))，CSEF (式(4.10)) に

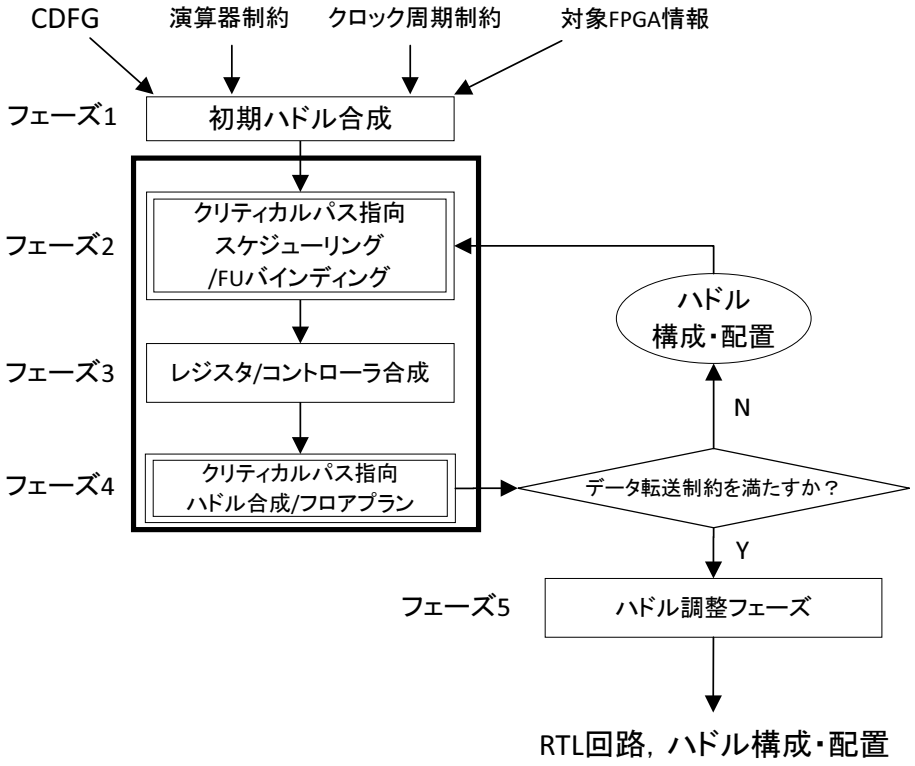


図 5.2: 提案アルゴリズム.

必要な値を対象FPGA情報として与える．まず，初期ハドル合成では1つのハドルに演算器が1つ割り当てられ，全てのハドルが重なった状態を生成する (フェーズ1)．これを基に，スケジューリング/FUバインディング (フェーズ2)，レジスタ/コントローラ合成 (フェーズ3)，ハドル合成/フロアプラン (フェーズ4) を繰り返し，データ転送制約を満たす回路を得るまで解を改良する．反復改良を終了したらFUを持たないハドルを削除し，それぞれのハドルを調整して結果を出力する (フェーズ5).

各イタレーションでは，初期ハドル合成 (フェーズ1) または前のイタレーションのハドル合成/ハドルフロアプラン (フェーズ4) で得られたハドルの構成・配置結果を基に，スケジューリング/FUバインディング (フェーズ2) とレジスタ/コントローラ合成 (フェーズ3) によってデータパスが生成される．スケジューリング/FUバインディング (フェーズ2) においてCDFGの各ノードをスケジューリング，およびFUに割り当てる．ここで，ハドル間のデータ転送が決定される．レジスタ/コントローラ合成 (フェーズ3) では，フェーズ2で得られたFU間のデータ転送に対して各ハドルにレジスタを用意し，各MUXのコントローラを生成する．従ってデータパス生成において，各ハドル間のデータ転送を決定するフェーズ2が配線遅延とクロックスキューの影響を左右する．一方，モジュール配置を変更するハドル合成/フロアプラン (フェーズ4) は各パスの配線遅延とクロックスキューに大きな影響を持つ．以上より，提案アルゴリズムではフェーズ2，4において配線遅延とクロックスキューを考慮しクリティカルパス遅延を削減する新たな手法を提案する．フェーズ1，3，5は第3章の手法と同様である．

5.2.3 クリティカルパス指向スケジューリング/FUバインディング

本項では，クリティカルパス指向スケジューリング/FUバインディングのアルゴリズムを示し，例を用いて議論する．

スケジューリング/FUバインディング問題は，入力としてCDFG，演算器制約，クロック周期制約，データ転送遅延情報を与え，各演算ノードを実行するCSと実行されるFUを出力する．その際の第1目的は総CS数の最小化，第2目的はクリティカルパス遅延の最小化である．データ転送遅延情報は，フェーズ1または前のイタレーションのフェーズ4で得られるモジュール配置を基に，各ハドル間のデータ転送に要するステップ数[2]と，各ハドル間の配線遅延・クロックスキューの見積り値を与える．

クリティカルパス指向スケジューリング/FUバインディングでは，FU間のデータ転送遅延を考慮してスケジューリングを決定する必要があるため，スケジューリングとFUバインディングを同時に実行する．提案手法はまずCS数の最小化を図ったスケジューリング/FUバインディングにより初期解を得る．そして，この初期解を基に，クリティカルパスになりそうな演算ノードに対してFUバインディングを改良することで，配線遅延とクロックスキューを改善し，最終的にクリティカルパス遅延を削減する．FUバインディングの改良では，対象ノードを割り当てたFUの転送先のFU，つまり各子ノードを割り当てたFUに対してFUバインディングの変更を検討する．以下の条件を満たすFUが他に存在する時，子ノードのFUバインディングを変更する．

1. FUに子ノードを割り当てた時，配線遅延とクロックスキューの影響が改善される．
2. FUに子ノードを割り当てた時，タイミング制約違反が起きない．

FUバインディングの改良を解が収束するまで繰り返す．

図5.3にクリティカルパス指向スケジューリング/FUバインディングのアルゴリズムを示す．提案手法はStep 1でCS数の小さいスケジューリング/FUバインディング解を生成し，Step 2以降でクリティカルパス遅延の削減を図る．以下で，各ステップについて示す．

Step 1（初期スケジューリング/FUバインディング）

FU間のデータ転送遅延を考慮してCS数の最小化を図るスケジューリング/FUバインディングを行い，各演算ノードが実行されるCSとそれを実現するFUバインディングを初期解として得る．本稿では第3章のパス考慮スケジューリング/FUバインディングを用いる．

Step 2（対象ノードの選択）

Step 1で得られた初期スケジューリング/FUバインディングされた演算ノードの中から，各ノードn_iの優先度$priority(n_i)$に従ってクリティカルパスになりそうな改善ノード群N_{cand}を抽出する．遅延の大きいFUを含むパスがクリティカルパスになりやすいと考え，ノードn_iの優先度$priority(n_i)$は以下のように求める．

$$priority(n_i) = D(Fu(n_k))/S(Fu(n_k)) \tag{5.2}$$

ここで，$Fu(n_k)$はノードn_kが割り当てられているFUを表し，$S(f_i)$はFU f_iの演算サイクル数を表す．提案手法では，優先度$priority(n_i)$の大きいN個のノード[3]を改善ノード群

[2]算出方法は，第5.2.4項に後述する．

[3]Nは第4.6.3項の議論に基づいて，全ノード数の20%付近に設定し，後述の第5.3で実験を行う．

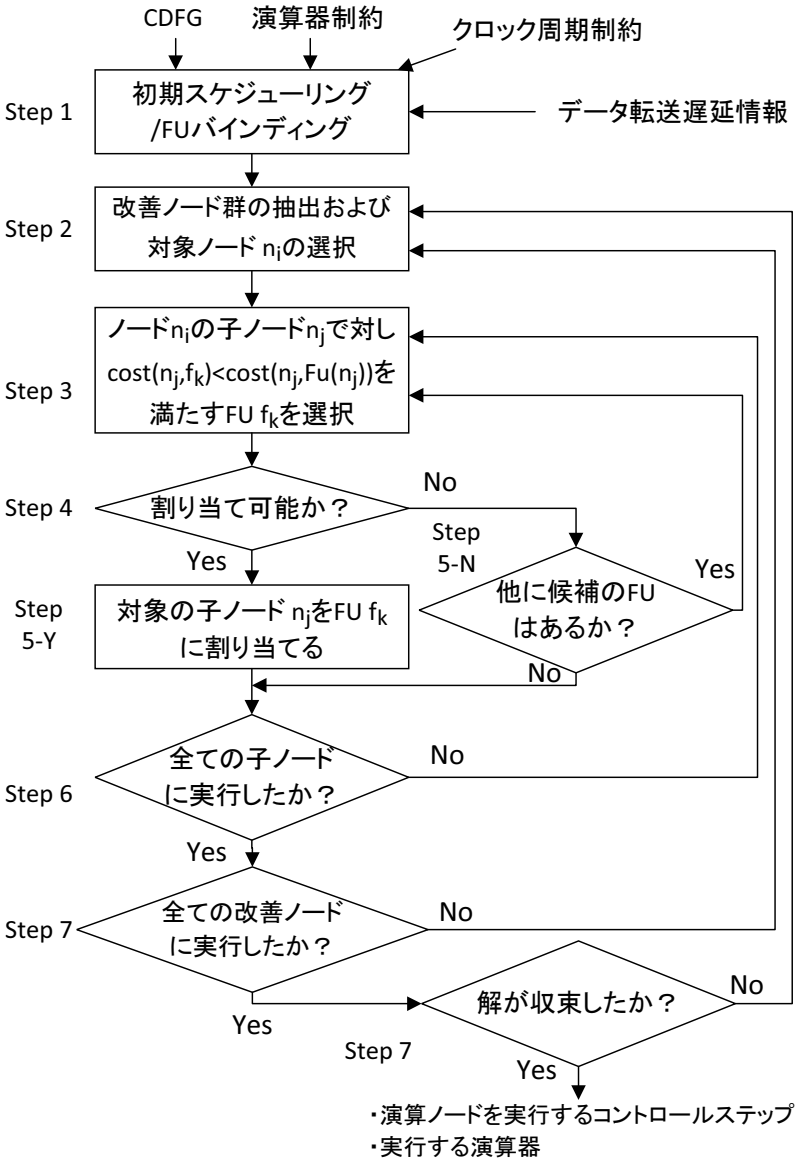


図 5.3: クリティカルパス指向スケジューリング/FU バインディング.

N_{cand} とする.

改善ノード群 N_{cand} から，パス遅延を削減する対象ノード n_i を選択する．CDFG のデータ依存に従って，改善ノード群 N_{cand} のうち早い CS にスケジューリングされたノードから選択する.

Step 3–5（FU バインディングの改良）

Step 2 で選択されたノード n_i に対して，パス遅延の削減を図る．HDR アーキテクチャでは各 FU は同じハドル内のレジスタからデータを受け取るようレジスタバインディングされる．提案手法では，各 FU の転送先の FU，つまり各子ノードを割り当てた FU に対して FU の割り当てを変更し，パスのデスティネーションレジスタを変更する．これによって，そのパスの配線遅延とクロックスキューの改善を図る．ノード n_i の子ノード n_j において，ノー

ド n_j を FU f_k に割り当てた時のコスト $cost(n_j, f_k)$ を以下のように求める．

$$cost(n_j, f_k) = \max_{n_l \in par(n_j)} \{D(path(R(Fu(n_l)), R(f_k)))\} \tag{5.3}$$

ここで，$par(n_j)$ はノード n_j の親ノード群を表し，$R(f_k)$ は FU f_k と同じハドルのレジスタとする．

ノード n_j の演算と一致する FU の集合を $F_{op}(n_j)$ とする．$F_{op}(n_j)$ のうち，$cost(n_j, f_k) < cost(n_j, Fu(n_j))$ を満たす FU の集合を $F_{cand}(n_j) \in F_{op}(n_j)$ とする．$F_{cand}(n_j)$ のうち，$cost(n_i, f_k)$ が最小となる FU $f_k \in F_{cand}(n_j)$ を選択する．

ノード n_j を FU f_k に割り当てた時，親ノードの FU から FU f_k へのデータ転送遅延と FU f_k から子ノードの FU へのデータ転送遅延を踏まえて，Step 1 で求めたスケジューリングを満たすか調べる．満たす場合は，ノード n_j を FU f_k に割り当てる．満たさない場合は，$F_{op}(n_j)$ の中で FU f_k を除いて $cost(n_j, f_l)$ が最小となる FU $f_l \in F_{cand}(n_j)$ を選択する．そして，FU f_k と同様にデータ転送遅延が Step 1 で求めたスケジューリングを満たすか調べる．FU バインディングが変更される，または $F_{cand}(n_j)$ の全ての FU に対して Step 3–5 を実行するまで繰り返す．

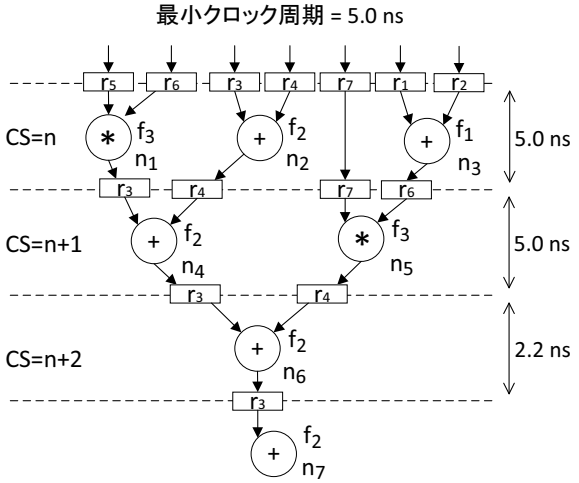
Step 6–8（終了判定） Step 3–5 をノード n_i の子ノード全てに対して実行する．実行し終えたら，N_{cand} 内の次に早い CS に割り当てられたノード n_l を選択し，Step 3–5 を実行する．N_{cand} 内の全てのノードに Step 3–5 を実行したら，実行前と後の解を比較する．もし，同一の解ならば収束したとみなしてスケジューリング/FU バインディング解を出力する．同一でない場合は，再度 Step2 から繰り返す．

例 1 図 5.4 に FU バインディング改良の例を示す．図 5.4(a) は Step 1 で得られた初期スケジューリング/FU バインディングの一部を表す．図 5.4(c) はハドル配置を表し，FU f_1，f_2，f_3 はそれぞれハドル h_1，h_2，h_3 に割り当てられているとする．また，レジスタ r_1，r_2 はハドル h_1，レジスタ r_3，r_4 はハドル h_2，レジスタ r_5，r_6，r_7 はハドル h_3 に割り当てられているとする．$D(f_1) = D(f_2) = 2$ns, $D(f_3) = 4$ns とし，単純のため各演算は 1CS で実行されハドル間のデータ転送は 0CS で完了するとする．図 5.4(d) はそれぞれハドル間のパスに生じる配線遅延/クロックスキューを表す．

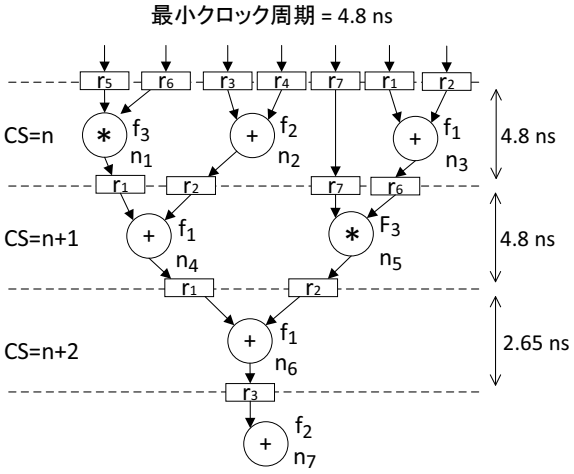
ここで，図 5.4(a) への FU バインディングの改良を考える．まず，式 (5.2) に従って，改善ノード群 $N_{cand} = \{n_1, n_5\}$ を抽出する．次に，N_{cand} のうち早い CS に割り当てられているノード n_1 へのパス遅延削減を図る．ノード n_1 の子ノードは n_4 であり，加算のノードである．ノード n_4 を割り当て可能な FU の候補は FU f_1，f_2 があり，現在 FU f_2 が割り当てられている．ここで，$D_{reg} = 0.2$ns，$D_{mux} = 0.5$ とし，それぞれの FU にノード n_4 を割り当てたときのコスト $cost(n_4, f_i)$ は式 (5.3) より以下の通りになる．

$$\begin{aligned} cost(n_4, f_1) &= 4.8 \\ cost(n_4, f_2) &= 5.0 \end{aligned} \tag{5.4}$$

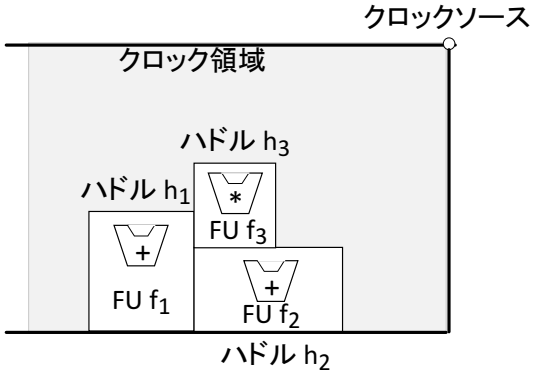
以上より，ノード n_4 を FU f_1 に割り当てた方がコストが小さく，条件 1 に合致する．次に，ノード n_4 を FU f_1 に割り当てた場合を考える．ノード n_4 の親ノード n_1，n_2 はそれぞれ



(a) 初期スケジューリング/FU バインディング解.



(b) FU バインディング改良後のスケジューリング/FU バインディング解.



(c) ハドルの構成と配置.

		デスティネーション ハドル		
		h_1	h_2	h_3
ソース ハドル	h_1	0.00/0.00	0.30/-0.15	0.20/-0.10
	h_2	0.30/0.10	0.00/0.00	0.25/0.05
	h_3	0.20/0.10	0.25/-0.05	0.00/0.00

(d) ハドル間の配線遅延/クロックスキューの値.

図 5.4: FU バインディング改良の例.

FU f_3，f_2 へ割り当てられている．FU f_3，f_2 から FU f_1 へのデータ転送遅延は 0CS であり，これは図 5.4(a) のスケジューリングを満たす．一方，ノード n_4 の子ノード n_6 は FU f_2 に割り当てられている．FU f_1 から FU f_2 へのデータ転送は 0CS かかり，これは図 5.4(a) のスケジューリングを満たす．従って，ノード n_4 を FU f_1 に割り当てたとしても，図 5.4(a) のスケジューリングを満たすので，条件 2 に合致する．よって，ノード n_4 を FU f_1 へ割り当てを変更する．

ノード n_5 に対しても同様に処理し，子ノード n_6 を FU f_1 に割り当てを変更する．FU バインディングを改良した後のスケジューリング/FU バインディング解が図 5.4(b) になる．FU バインディング改善前は図 5.4(a) の通り，クリティカルパス遅延および最小クロック周期は 5.0ns であった．しかし，FU バインディング改善後は FU バインディング図 5.4(b) の通り，クリティカルパス遅延および最小クロック周期は 4.8ns となり，0.2ns 削減できている． □

以上の例より，パス遅延の見積りが大きくクリティカルパスになりそうな演算に対して，FU 間の配線遅延とクロックスキューを考慮して FU バインディングを改善することで，クリティカルパス遅延を削減できるとわかる．

5.2.4 クリティカルパス指向ハドル合成/フロアプラン

本項では，IDEF・CSEFを用いて改良されたフロアプラン手法であるクリティカルパス指向ハドル合成/フロアプランについて議論する．

クリティカルパス指向ハドル合成/フロアプランでは，[2]と同様にFUの移動によりハドルの構成と配置の変更を統合的に扱う．フロアプランにおいてデータ構造としてSequence-pair [53]を用いる．フェーズ1あるいは前のイタレーションでのハドルの配置・構成を初期解としてSA法によりフロアプランを最適化する．SA法では以下の5つのmoveを考える．

move 1 2つのハドルを選択し，縦の位置（Γ_+）を入れ替える．

move 2 2つのハドルを選択し，横の位置（Γ_-）を入れ替える．

move 3 2つのハドルを選択し，縦と横の位置（Γ_+, Γ_-）を入れ替える．

move 4 1つのハドルを選択し，縦と横のアスペクト比を変更する．

move 5 FU f_iを選択し，ハドル$h_i(= Hud(f_i))$からハドルh_jへ移動する．

ここで，$Hud(f_i)$とはFU f_iが割り当てられているハドルを表す．

第4.6.3項では，IDEF・CSEFを適用した新たなコスト関数を示したが，このコスト関数では，配線遅延・クロックスキューのコストを各々独立した項で設け，最適化していた．しかし，提案手法ではクリティカルパス遅延の削減が主たる目的であるので，パス遅延そのものをコストとして設け最適化することが，より効果的と考えられる．そこで，提案手法では式(5.1)により，各パスの遅延を見積もる．この見積りを基にクリティカルパスになりそうなパスを高位合成段階で特定し，集中的に最適化することでクリティカルパス遅延の小さいフロアプラン解を実現する．提案手法ではSAのコスト関数$cost_{fl}$を以下のように定義する．

$$cost_{fl} = \alpha \frac{V}{T_{clock}} + \beta \frac{DP_n}{T_{clock}} + \gamma \frac{A_{BB}}{A_{total}} \tag{5.5}$$

ここで，α，β，γはパラメータを表す．式（5.5）は第1項がタイミング制約違反，第2項がクリティカルパス遅延，第3項が面積を表し，それらの和で求められる．Vはタイミング制約違反の総合計，T_{clock}はクロック周期，DP_nは式(5.1)による値が大きいN本のパスの遅延の和[4]，A_{BB}は最小矩形の面積，A_{total}は全ハドルの面積の合計である．

提案手法では，タイミング制約違反が最も最適化すべき項目と考え，αをβ，γに比べて大きく設定する[5]．従って，タイミング制約違反が0になった後に，クリティカルパス遅延と面積に関して最適化される．

また，フェーズ4では各パスのデータ転送に要するCS数を求め，フェーズ2へフィードバックする．FU fを含むパス$path(r_s, r_d)$に要するデータ転送ステップ数$T_r(path(r_s, r_d))$は式(5.1)より以下のように求める．

$$T_r(path(r_s, r_d)) = \begin{cases} 0 \ (T_{clock} \times S(f) \geq D(path(r_s, r_d))) \\ D(path(r_s, r_d))/T_{clock} \\ \quad (T_{clock} \times S(f) < D(path(r_s, r_d))) \end{cases} \tag{5.6}$$

[4] Nの値を，前項と同様に第4.6.3項の議論に基づいて，全体のパス数の20%程度に設定し次節で実験する．

[5] 本章では$\alpha = 100$，$\beta = 1$，$\gamma = 1$と設定した．

表 5.1: 計算機実験結果.

App. (#Nodes)	FU constraint	Clock period constraint [ns]	Applied algorithm	#Huddles	critical path delay [ns]	#Steps	Latency [ns]	#Slices	#LUTs	#FFs	Total synthesis time [s]
hal (11)	Add×1, Sub×1, Mul×2, Comp×1	5	SR	-	3.869	5	19.35	100	380	108	446
		5	[2]	2	4.185	5	20.93	125	435	135	354
		5	[29]	2	4.050	5	20.25	117	427	151	368
		5	提案手法	1	3.869	5	19.35	100	380	108	412
parker (22)	Add×2, Sub×2, Comp×1	4	SR	-	1.987	7	13.91	64	186	224	279
		4	[2]	2	2.185	8	17.48	85	235	263	356
		4	[29]	2	1.578	7	11.05	75	214	246	370
		4	提案手法	2	1.548	7	10.84	75	209	231	363
dct (48)	Add×4, Mul×4	7	SR	-	4.905	8	39.24	412	1383	345	414
		7.5	[2]	6	4.437	8	35.50	438	1489	623	478
		7	[29]	5	4.434	8	35.47	366	1178	544	427
		7	提案手法	5	4.349	8	35.08	362	1196	544	392
jacobi (52)	Add×2, Sub×1, Mul×2, Div×2	30	SR	-	18.20	17	309.3	364	1168	271	394
		31	[2]	3	17.69	17	300.6	374	1212	345	432
		30	[29]	3	17.54	17	298.1	340	1135	354	454
		30	提案手法	3	17.46	17	296.8	338	1124	355	415
fir (75)	Add×4, Mul×4, Mem×1	7	SR	-	5.330	15	79.95	435	1535	307	666
		7	[2]	4	5.192	15	77.88	445	1482	420	485
		7	[29]	4	4.468	15	67.02	329	1062	400	475
		7	提案手法	3	4.035	15	60.53	332	1063	365	439
ewf3 (102)	Add×4, Mul×4	7	SR	-	5.175	40	207.0	494	1603	363	576
		7	[2]	2	4.820	40	192.8	510	1769	646	490
		7	[29]	4	4.390	40	175.6	500	1640	616	463
		7	提案手法	4	3.931	40	157.2	488	1605	567	472
copy (378)	Add×3, Sub×1, Mul×5, Comp×1, Rshift×2, AND×1, Mem×1	8.5	SR	-	6.380	82	523.2	1437	5100	1557	878
		10.5	[2]	9	6.227	82	510.6	2098	6608	3261	780
		8.5	[29]	10	5.718	82	468.9	1939	5867	3778	920
		8.5	提案手法	8	5.201	82	426.5	1950	5620	3327	1707

ここで，$S(f)$はFU fの演算サイクル数を表す．全パスのデータ転送ステップ数$T_r(path(r_s, r_d))$を基にハドル間のデータ転送遅延表 DT を作成し，次のイタレーションへフィードバックする．DT の各要素 $DT(h_i, h_j)$ はハドル h_i とハドル h_j 間のデータ転送に要するCS数を表し，ハドル h_i とハドル h_j 間の全てのパスの中で最大のデータ転送ステップ数とする．

フェーズ5におけるハドルフロアプランの調整においても，式(5.5)と同様のコスト関数を用いる．

5.3 計算機実験結果と評価

本節では，計算機実験を行い，提案手法の優位性を明らかにする．

5.3.1 実験環境

本項では，計算機実験の環境について述べる．

提案手法をC++言語を用いて計算機上に実装し，第3章と同様の7つのアプリケーションに適用した．実験方法は，まず提案手法によってRTL記述を出力する．次に，Xilinx社Vivado2014.2で論理合成を行い，ゲートレベルの回路情報を得る．本実験では対象FPGAはXilinx社Virtex-7 (xc7vx485tffg1761-2) とし，dspブロックの使用を禁止とした．また，この

時RTLファイルの階層をまたいで最適化するflatten_hierarchyを禁止とした．次に，Vivadoで配置・配線を行い，最終的な回路情報を得る．図5.1のように，回路の左下の点をクロックソース(0,0)から下に50スライス，左に34スライス離れた点P(-34,-50)に合わせて配置した．最後に，VivadoによりBITファイル化し，FPGA上のHDRアーキテクチャ回路が実装される．各手法の高位合成の実行に使用した計算機環境は，AMD Opteron 2360SE 2.5GHz×2，メモリが16GBである．論理合成，配置・配線に使用した計算機環境は，Intel Core i5-2520M 2.5GHz×2，メモリが4GBである．

演算器/レジスタ情報，IDEF・CSEFのパラメータは第4.6節と同様である．比較する既存手法として，SR, [2]の手法, 第3章の手法 [29]を用意する．SRはレジスタ集中型アーキテクチャを採用しフロアプランを考慮しない手法である．[2]はMUXのコストを削減しないフロアプラン指向高位合成手法である．[29]はクロックスキューを考慮せず，かつクリティカルパスを特定しないフロアプラン指向FPGA高位合成手法である．従来手法では，配線遅延は配線長に比例すると仮定し，60スライス分の距離あたり1nsとした．これらも提案手法と同様にFPGA上に実装し，比較した．

5.3.2 実験結果

Virtex-7における実験結果を表5.1に示す．提案手法はSRと比較して，CS数を増加させずにレイテンシを最大24%，平均15%削減し，[2]と比較して，レイテンシを最大38%，平均15%削減した．さらに，3章の手法 [29]と比較して，CS数とスライス数をほぼ同程度にした上で，レイテンシを最大10%，平均6%削減した．この結果は，第4.6節における [29]+IDEF+CSEFの [29]に対する最大削減率より1%少ないが，これは測定位置などによる誤差と考える．一方，SRに対しては，[29]+IDEF+CSEFよりレイテンシ削減率が高いので，[29]+IDEF+CSEFに比べて，手法の優位性があると考える．

さらに，中でも大規模なfir，ewf3，copyの3つのアプリケーションでは，SRと比較して平均22%，[2]と比較して平均19%，[29]と比較して平均10%削減し，効果的にレイテンシーを削減できている．また，提案手法は従来手法3つと比較して，全てのアプリケーションにおいて同じもしくは削減されたレイテンシーの回路を実現した．

スライス数はSRと比較して，最大24%削減したが平均ではほぼ同程度である．一方，[2]と比較してスライス数を最大26%, 平均14%削減し，[29]と比較して最大2%，平均1%削減している．

次に，表5.2に表5.1における各アプリケーションの回路のクリティカルパス遅延の内訳を示す．表5.2の通り，提案手法はparkerを除く全てのアプリケーションで，従来手法3つに比べて，配線遅延を削減することでクリティカルパス遅延の削減を実現している．また，[2]のewf3やcopy, [29]のcopyのようにクロックスキューを大幅に悪化させずクリティカルパス遅延の削減を実現できている．

以上より，提案手法は高位合成段階で配線遅延とクロックスキューを考慮し，クリティカルパスの候補を特定し，それらを最適化することでクリティカルパス遅延およびレイテンシーを削減した．また，大規模なアプリケーションでは配線遅延とクロックスキューの影響が大きくなり，より効果的にレイテンシーを削減できる．

表 5.2: アプリケーション毎のクリティカルパスの内訳.

App.	Applied algorithm	Logic delay [ns]	Interconnection delay [ns]	Clock skew [ns]	Clock uncertainty [ns]	Critical-path delay [ns]
hal	SR	1.642	2.142	-0.050	0.035	3.869
	[2]	1.769	2.355	-0.046	0.035	4.185
	[29]	1.645	2.326	-0.044	0.035	4.050
	提案手法	1.642	2.142	-0.050	0.035	3.869
parker	SR	0.804	1.123	-0.025	0.035	1.987
	[2]	1.266	0.864	-0.020	0.035	2.185
	[29]	0.809	0.708	-0.026	0.035	1.578
	提案手法	0.755	0.710	-0.048	0.035	1.548
dct	SR	1.873	2.947	-0.05	0.035	4.905
	[2]	1.827	2.544	-0.031	0.035	4.437
	[29]	1.676	2.671	-0.052	0.035	4.434
	提案手法	1.711	2.556	-0.047	0.035	4.349
jacpbi	SR	10.364	7.747	-0.051	0.035	18.197
	[2]	10.057	7.542	-0.051	0.035	17.685
	[29]	10.015	7.440	-0.047	0.035	17.537
	提案手法	10.161	7.212	-0.049	0.035	17.457
fir	SR	1.746	3.501	-0.048	0.035	5.330
	[2]	2.073	3.041	-0.043	0.035	5.192
	[29]	1.767	2.627	-0.039	0.035	4.468
	提案手法	1.809	2.150	-0.041	0.035	4.035
ewf3	SR	1.773	3.31	-0.057	0.035	5.175
	[2]	1.727	2.934	-0.124	0.035	4.820
	[29]	1.693	2.607	-0.055	0.035	4.390
	提案手法	1.704	2.167	-0.025	0.035	3.931
copy	SR	1.908	4.394	-0.043	0.035	6.38
	[2]	1.752	4.313	-0.127	0.035	6.227
	[29]	1.678	3.822	-0.183	0.035	5.718
	提案手法	1.666	3.466	-0.034	0.035	5.201

5.3.3 大規模アプリケーションでの追加実験

提案手法の大規模アプリケーションでの効果を検証するため，更に3つのアプリケーションを用意し，提案手法を適用する．適用したアプリケーションは，Mesaと呼ばれる画像処理の内，Matrix Multiplication（Mesa_MM, 109ノード），Smooth Triangle（Mesa_ST, 197ノード），Matrix invertion（Mesa_MI, 333ノード）である [22].

実験環境は，第5.3.2節での実験と同様の環境を用いた．比較する手法として，SRと第3章の手法 [29] にIDEFのみを適用した手法 [28] を利用した.

各手法での合成回路結果を表5.3に示す．提案手法はSRと比較して，レイテンシを最大27%，平均18%削減し，スライスを最大11%，平均5%削減した．提案手法は [28] と比較して，スライス数を同程度にした上で，レイテンシを最大14%，平均8%削減した．また, 表5.4

表 5.3: 追加アプリケーションの合成回路結果.

App. (#Nodes)	FU constraint	Clock period constraint	Applied algorithm	#Huddles	Minimum clock period [ns]	#Steps	Latency [ns]	#Slices	#LUTs	#FFs	Total synthesis time [s]
Mesa_MM (109)	Add×4	7	SR	-	5.556	15	83.34	567	1919	495	349
	Mul×4	7	[28]	5	5.070	15	76.05	494	1650	744	443
	Mem×2	7	提案手法	5	4.709	15	70.64	503	1526	653	600
Mesa_ST (197)	Add×4, Sub×1	7.5	SR	-	5.758	29	167.0	791	2864	732	382
	Mul×4	7.5	[28]	5	5.477	29	158.8	861	2562	1255	497
	Mem×2	7.5	提案手法	8	5.065	29	146.9	845	2463	1452	835
Mesa_MI (333)	Add×4, Sub×1	7.5	SR	-	6.612	42	277.7	1178	4134	970	364
	Mul×4	7.5	[28]	7	5.597	42	235.1	1030	2933	1812	518
	Mem×2	7.5	提案手法	7	4.805	42	201.8	1049	2818	1794	700

表 5.4: 追加アプリケーション回路のクリティカルパス遅延の内訳.

App. (#Nodes)	Applied algorithm	Logic delay [ns]	Interconnection delay [ns]	Clock skew [ns]	Clock uncertainty [ns]	Critical-path delay [ns]
Mesa_MM (109)	SR	1.911	3.566	-0.044	0.035	5.556
	[28]	1.814	3.105	-0.116	0.035	5.070
	提案手法	1.690	2.941	-0.043	0.035	4.709
Mesa_ST (197)	SR	1.807	3.867	-0.049	0.035	5.758
	[28]	1.669	3.319	-0.150	0.035	5.477
	提案手法	1.818	3.093	-0.119	0.035	5.065
Mesa_MI (333)	SR	2.119	4.343	-0.115	0.035	6.612
	[28]	1.814	3.572	-0.176	0.035	5.597
	提案手法	1.712	3.018	-0.040	0.035	4.805

に表3.4における各アプリケーションの回路のクリティカルパス遅延の内訳を示す．表5.4の通り，提案手法は [28] に比べて，クロックスキューの悪化を抑制しつつ配線遅延を削減することで，クリティカルパス遅延を削減できたことがわかる．

5.4 本章のまとめ

本章では，FPGAを対象とした高位合成における課題3「高位合成段階での配線遅延・クロックスキューの影響とMUXのコストの同時考慮」を解決するため，第3章で提案したフロアプラン指向FPGA高位合成手法に前章で提案したIDEF・CSEFを利用し，配線遅延とクロックスキューを考慮したクリティカルパス最適化FPGA高位合成手法を提案した．提案手法はモジュール配置を基にIDEFとCSEFを用いて，モジュール間の配線遅延とクロックスキューを見積る．さらに，高位合成段階で配線遅延とクロックスキューを含めた各パスの遅延を見積り，見積り遅延の大きいクリティカルパスの候補となるパスを特定する．そして，データパス生成とフロアプラン両方において，これらを集中的に最適化し回路のレイテンシーの向上を図る．クリティカルパス指向スケジューリング/FUバインディングでは，フロアプラン情報を基にFUバインディングを改良することで，配線遅延とクロックスキューを改善しクリティカルパス遅延の小さいデータパスの生成を図る．また，クリティカルパス指向ハドル合成/フロアプランではフロアプランの改善において配線遅延とクロックスキューを

考慮して，クリティカルパスになりそうなパスを集中的に最適化を行う．そして，クリティカルパス遅延の小さいフロアプラン解を目指す．

計算機実験により，第3章で用いた7つのベンチマークアプリケーションにおいて，提案手法はSRと比較して，回路のレイテンシを最大24%，平均15%削減し，スライス数を最大24%削減し，平均ではほぼ同程度であることを確認した．また，提案手法は従来のHDRアーキテクチャを対象とした高位合成手法 [2] と比較して，回路のレイテンシを最大38%，平均15%削減し，スライス数を最大26%，平均14%削減することを確認した．提案手法は，第3章で提案したIDEFとCSEFを用いらずクリティカルパスを特定しない手法と比較して，スライス数を同程度に保った上で回路のレイテンシを最大10%，平均6%削減することを確認した．

特に，提案手法は配線遅延。クロックスキューの影響が大きくなりやすい大規模アプリケーションでレイテンシの削減効果が大きく，SRと比較して平均22%，[2] と比較して平均19%，[29] と比較して平均10%削減できることを確認した．さらに，大規模な3つの追加アプリケーションで追加実験を行い，SRと比較して，レイテンシを最大27%，平均18%削減し，第3章の手法にIDEFのみを適用した手法と比べて，レイテンシを最大14%，平均8%削減することを確認した．また，提案手法はCSEFを用いてクロックスキューを考慮しているため，一部の大規模アプリケーションで，クロックスキューを改善した上でレイテンシ削減を実現していることが確認できた．

第6章　フロアプラン指向高位合成手法を用いたFPGA実装と評価

6.1　本章の概要

本章では[1]，レジスタ分散型アーキテクチャを対象としたフロアプラン指向FPGA高位合成を用いた実装方法の確立と実装された回路の評価を行う．

レジスタ分散型アーキテクチャは，レジスタ集中型アーキテクチャに比べて，FU・レジスタ間の配線遅延を小さくできるため，レイテンシの小さい回路を生成するための有効な技術である．しかし，一方でレジスタ数は増加するため，高位合成段階でのタイミング設計はより複雑になる．更に，フロアプラン指向FPGA高位合成では，高位合成段階で決定したフロアプラン情報を実装フローにインプットする必要があり，通常のFPGA実装とは異なるフローとなる．また，実際のFPGA実装ではアプリケーション回路のみではなく，外部とのデータのやり取りを行うインターフェース回路も必要となる．しかし，既存研究ではフロアプラン指向FPGA高位合成を用いたFPGA実装方法は明らかにされておらず，前章までの議論も配置・配線後のFPGA設計ツールのレポートを基に議論をしている．

従って，本章では，前章で提案したフロアプラン指向FPGA高位合成手法を用いて，FPGA実装を行い，性能評価を行う．まず，実装環境およびその際のアプリケーション回路に必要となるインターフェース回路について議論する．次に，フロアプラン指向高位合成手法を用いた際のFPGA実装フローを提案する．そして，第5章で提案したフロアプラン指向高位合成手法を用いて，ベンチマークアプリケーションの1つであるDCTアプリケーションを例に取り，実際にXilinx Virtex-7上にFPGA実装を行い，正常に動作することを確認する．最後に，既存手法を用いて実装した回路と比較して，実機レベルで提案手法によって合成された回路を評価する．

6.2　実装環境と実装回路の構成

6.2.1　実装環境

本項では，FPGA実装に用いる環境について述べる．

本章では，FPGA実装のためにFPGA評価ボードTB-7V-2000T-LSIのVietex-7（XC7V2000T-2FLG1925）[73]を使用する．FPGAボードの全体図を図6.1に示す．当FPGAは，DDR3 SDRAMがオンチップメモリとして搭載されており，これをBRAMとして用いる．また，[43]

[1]本章の内容は[32,36]による．

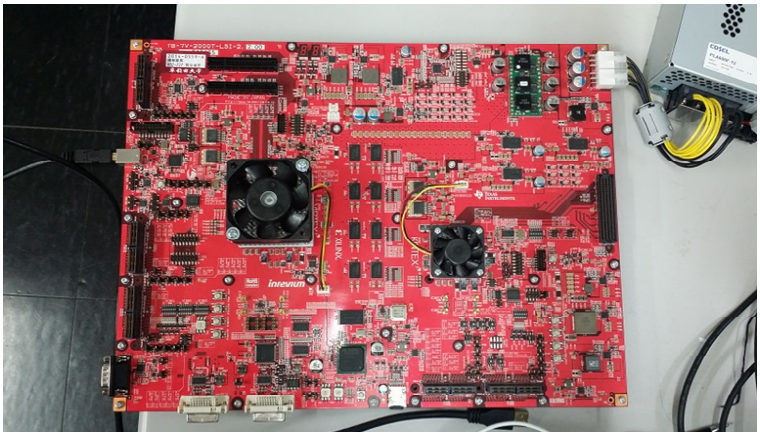


図 6.1: FPGA 評価ボード TB-7V-2000T-LSI.

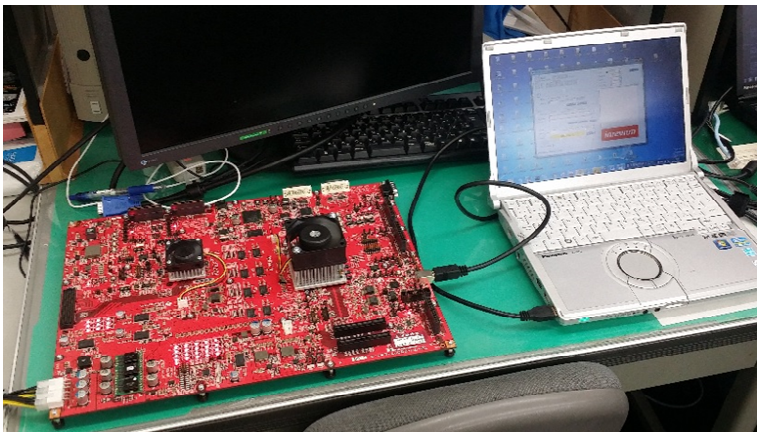


図 6.2: FPGA 実装環境の様子.

と同様に，FPGA ボードの FX3USB コントローラと外部 PC のデータ通信のために，Cypress 社の CyAPI を使用し，FPGA のバス規格は AXI（Advanced eXtensible Interface）である．開発ツールは Xilinx 社の Vivado Design Suite 2014.2 [68] を使用する．実際の FPGA 実装環境の様子を図 6.2 に示す.

6.2.2 実装回路の構成

本項では，FPGA の実装回路の構成を示し，実装に必要な 3 つのインターフェース回路について議論する．

図 6.3 に FPGA 上の実装回路の構成を図 6.3 に示す．実装回路は大きく分けて HDR アーキテクチャのアプリケーション回路，3 つのインターフェース回路（“USB 通信用インターフェース”，“メモリ読み書き用インターフェース”，“クロック/リセット生成回路”），オンチップメモリ（BRAM）の 5 つで構成される．

HDR アーキテクチャ回路は第 5 章の高位合成手法を用いて合成したアプリケーションの演算処理を行う回路である．USB 通信用インターフェースは PC より USB データ通信によって BRAM のアドレス（0x00–0x54 番地）と読み書き命令を受け取り，メモリ読み書き用イ

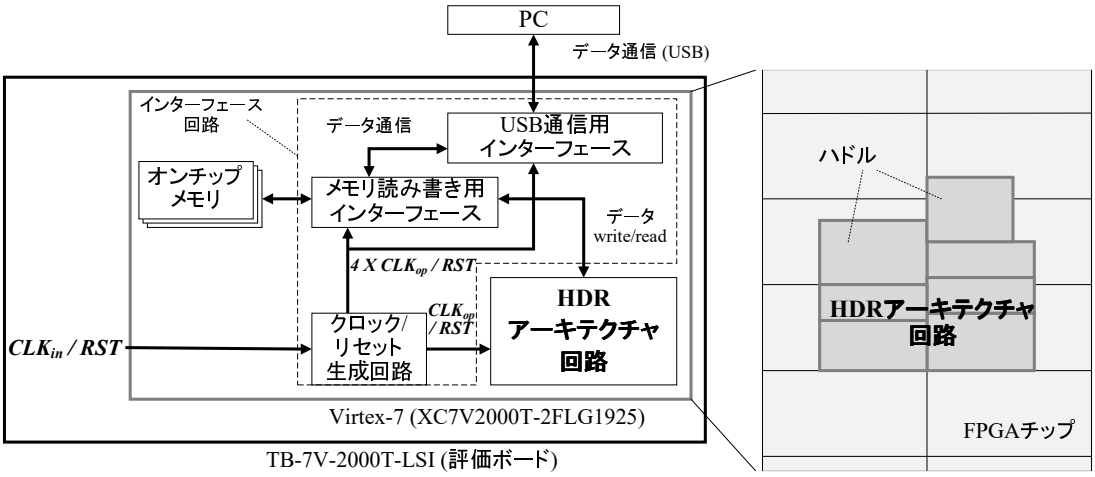


図 6.3: FPGA 上の実装回路の構成.

ンターフェースから BRAM に読み書きを行うフラグ，データ，アドレスを相互通信し，得られたデータを PC へ送信する．メモリ読み書き用インターフェースは USB 通信用インターフェースまたは HDR アーキテクチャ回路より受け取ったデータを指定された BRAM へ書き込む．また，BRAM から指定されたアドレスのデータを読み出し，USB 通信用インターフェースまたは HDR アーキテクチャ回路へ渡す.

クロック/リセット生成回路は FPGA ボードから受け取る 19.933ns ≈ 50MHz のクロック信号 CLK_{in} を変調して任意のクロック周期に設定する [74]．HDR アーキテクチャ回路へのクロック周期 CLK_v (ns) は以下の式で計算される．

$$CLK_v = CLK_{in} \times \frac{\alpha}{\beta} \tag{6.1}$$

ここで，パラメータ α, β は以下の条件を満たす必要がある．

条件 1 $\alpha \in \mathbb{N}$ and $\beta \in \mathbb{N}$.

条件 2 $\beta \leq 64$[2].

本稿では，USB 通信用インターフェース，メモリ読み書き用インターフェース，クロック/リセット生成回路は [43] と同様の回路を用いている．

6.3 フロアプラン指向高位合成を用いた FPGA 実装フロー

本節では，フロアプラン指向高位合成を用いた FPGA 実装フローを定義する．提案フローは，前節で定義したインターフェース回路の組み込みと，高位合成段階で決定したフロアプラン情報の下位工程へのインプットを考慮している．

[16] では，高位合成を用いた Xilinx FPGA への実装方法が述べられているが，これはフロアプランを扱わないレジスタ集中型アーキテクチャを対象とした高位合成を用いた方法となっている．レジスタ分散型アーキテクチャを対象としたフロアプラン指向高位合成では，

[2] β が 64 より大きい時，Vivado による配置・配線が失敗してしまう事が経験的にわかっている．

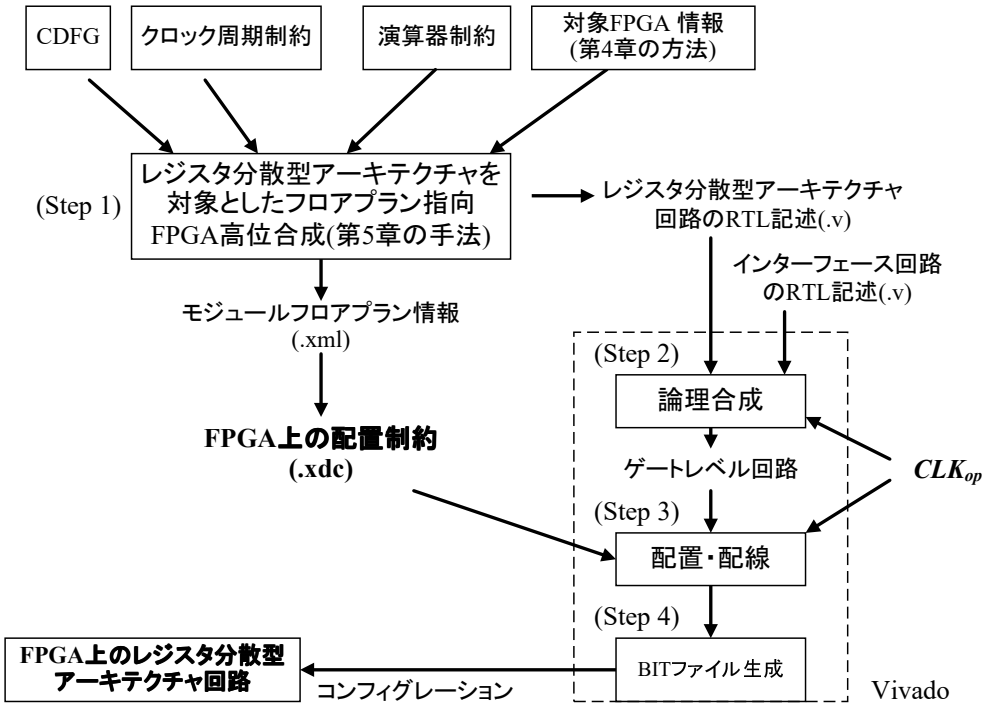


図 6.4: レジスタ分散型アーキテクチャを対象としたフロアプラン指向高位合成を用いた FPGA 実装フロー.

高位合成段階で抽象化したモジュールのフロアプランが決定されるため，下位の FPGA 設計フローにおいて適切にインプットする必要がある.

従って，[16] の方法をそのまま用いることはできず，本節では，新たにレジスタ分散型アーキテクチャを対象としたフロアプラン指向高位合成を用いた FPGA 実装フローを提案する. 図 6.4 にレジスタ分散型アーキテクチャを対象としたフロアプラン指向高位合成を用いた FPGA 実装フローを示す.

まず，第 5 章の手法の入力として，対象アプリケーションの動作記述を変換した CDFG, クロック周期制約，演算器制約，第 4 章の方法で取得した FPGA 上の配線遅延・クロックスキュー情報を与え，Verilog で記述された “レジスタ分散型アーキテクチャ回路の RTL 記述” と XML 形式の “モジュールフロアプラン情報” を得る（Step 1）. 次に，Step 1 で得た RTL 記述と 3 つのインターフェース回路の RTL 記述を合わせて，論理合成を実行し，ゲートレベル回路を得る（Step 2）. この時，式 (6.1) で得られる FPGA 上で動作させるクロック周期 CLK_{op} を与える. 動作クロック周期 CLK_{op} は Step 1 で与えたクロック周期制約と同じ，もしくはわずかに小さい値に小さく設定する. 式 (6.1) のパラメータ α, β は以下のように設定する.

Step 1 で与えたクロック周期制約と同じ動作クロック周期 CLK_{op} を得られる $\frac{\alpha}{\beta}$ の値を以下の式で求める.

$$\frac{\alpha}{\beta} = CLK_{op} \div CLK_{in}, \tag{6.2}$$

ここで，FPGA ボードから与えられるクロック周期 CLK_{in} を 20 ns と近似して扱う. そし

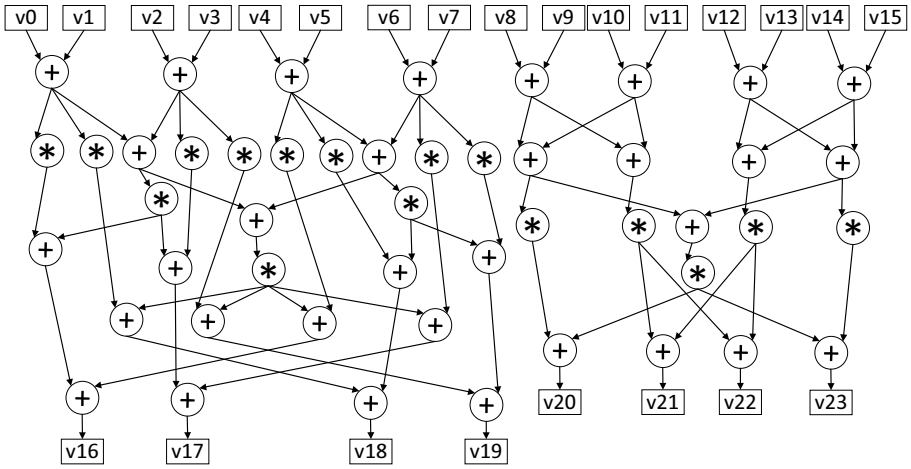


図 6.5: DCT アプリケーションの DFG.

て，式 (6.2) で得られる $\frac{\alpha}{\beta}$ と条件 1, 2 を満たすパラメータ α, β を設定する [3][4].

そして，Step 2 で得られたゲートレベル回路に対し，配置・配線を行う．この時，Step 1 で得られたモジュールフロアプラン情報を Vivado への配置制約ファイル（.xdc）に変換し，Vivado に与えて対象 FPGA 上への配置・配線を行う（Step 3）．最後に，BIT ファイルを生成し，対象 FPGA にコンフィグレーションし，FPGA 上のレジスタ分散型アーキテクチャ回路を得る（Step 4）.

6.4 実装結果と評価

本節では，前節で述べた実装フローを用いてベンチマークアプリケーションの FPGA 実装を行う．そして，既存の SR の手法を用いた場合と比較を行い，前章での提案手法を実機レベルで評価する．また，Vivado の合成レポートと同様の性能評価結果が FPGA 上で得られることを確認し，第 3–5 章での計算機実験結果の裏付けとする.

前節で述べた実装フローにて，第 5.3 節で用いたベンチマークアプリケーションのうち，DCT アプリケーション（ノード数 48 個，条件分岐無し，図 6.5）を例に取り，Virtex-7 上に FPGA 実装した．DCT アプリケーションの DFG 内の乗算は全て 1 つの入力しかもたないため，2 倍の定数乗算とした．また，性能比較のため，第 3.5 節と同様の SR の手法を用いて DCT アプリケーションを Virtex-7 上に FPGA 実装した.

実装した HDR/SR アーキテクチャ回路の実装結果を表 6.1 に示す．それぞれの手法では，入力で与えるクロック周期制約を徐々に小さくしていき，解が得られたクロック周期のうちレイテンシが最小となる合成結果を採用した.

[3]後述の第 6.4 節では，パラメータ β を 20–30 の範囲に設定した.
[4]パラメータ α, β は，クロック/リセット生成回路の RTL 記述にて明示的に与える.

表 6.1: DCT アプリケーションの実装結果.

	Clock period constraint in HLS [ns]	Parameters (α, β)	Operation clock period CLK_{op} [ns]	#Steps	Latency [ns]	#Slices	#LUTs	#FFs	Total impl. time [s]	Operation time on Xilinx Virtex-7 [ns]
SR アーキテクチャ回路	6.4	(8, 25)	6.378	10	63.78	1268	2943	3394	1179	65.87
HDR アーキテクチャ回路	5.0	(5, 20)	4.983	10	49.83	1303	2779	3233	710	51.92

6.4.1 動作検証

実装した DCT アプリケーション回路が正常に動作することを確認するため，以下の手順で検証を行った.

検証手順 1 PC より HDR/SR アーキテクチャ回路で使用するデータ（図 6.5 の $v0$–$v15$）を BRAM へ格納する.

検証手順 2 PC よりスタート信号を送る.

検証手順 3 BRAM から HDR アーキテクチャ回路内のレジスタへデータが送られる.

検証手順 4 データがレジスタに格納されたら HDR アーキテクチャ回路内での処理を開始する.

検証手順 5 処理が完了したら，レジスタ内の演算結果を BRAM へ格納する.

検証手順 6 BRAM から PC へ演算結果（図 6.5 の $v16$–$v23$）を送り，入力に対する理論値と比較する.

上記の手順にて動作検証を行い，表 6.1 の 4 列目に示す動作クロック周期 CLK_{op} にて正常に演算処理ができることを確認した.

6.4.2 性能測定

実装した回路の Virtex-7 上の処理性能を以下の手順で測定した.

測定手順 1 Virtex-7 上に SR-based circuit, あるいは DR-based circuit による DCT アプリケーション回路をコンフィグレーションする.

測定手順 2 外部 PC より演算の入力データとスタート信号を送信する.

測定手順 3 DCT アプリケーション回路を 1,000,000,000 回実行する.

測定手順 4 手順 3 の時間を測定し，回路の動作回数 (1,000,000,000 回) より，1 回あたりの実行時間を求める.

上記手順により，測定した HDR/SR アーキテクチャ回路の Virtex-7 上でのレイテンシを表 6.1 の 11 列目に示す.

表 6.1 の 11 列目より，実 FPGA チップ上でのレイテンシに関して，HDR アーキテクチャ回路が SR アーキテクチャ回路より $65.87 - 51.92 = 13.95$ ns，割合にして約 21%削減した. これは表 6.1 の 6 列目 Vivado 合成レポートに基づく実行時間の差　$63.78 - 49.83 = 13.95$ ns

と等しく，削減割合もほぼ同等の結果が得られている．このことから，前章までの Vivado 合成レポートに基づく性能評価は，実 FPGA 上でもほぼ同等の結果が得られると考えられる．

また，表 6.1 の 11 列目 “Operation time on Xilinx Virtex-7” の値が表 6.1 の 6 列目 “Latency” の値に比べて 2.09 ns 大きくなっているのは，“Operation time on Xilinx Virtex-7” の値が，外部 PC から FPGA チップへのデータ転送遅延を含む値であるためと考えられる．

また，本章での FPGA 実装に用いた DCT アプリケーションは文献 [1-3] および第 3–5 章の計算機実験で用いられている 7 つのベンチマークアプリケーションの中でも，中規模かつ [21,45,54,75] でも用いられている一般的なアプリケージョンであり，DCT アプリケーションでの実装結果は他のベンチマークアプリケーションにも通じる結果と考えられる．また，近年の商用ツールでは解の収束性の観点からレジスタ集中型アーキテクチャを採用していると考えられ，本節の実験結果において SR に対し第 5 章の手法の優位性を示せたことにより，商用ツールとの優位性を確認できたと考える．

6.5 本章のまとめ

本章では，レジスタ分散型アーキテクチャを対象としたフロアプラン指向 FPGA 高位合成を用いた実装方法の確立と実装された回路の評価を行った．

まず，実装環境およびその際のアプリケーション回路に必要となるインターフェース回路について述べ，その後，フロアプラン指向高位合成手法を用いた際の FPGA 実装フローを提案した．そして，第 5 章で提案したフロアプラン指向高位合成手法を用いて，ベンチマークアプリケーションの 1 つである DCT アプリケーションを例に取り，実際に Xilinx Virtex-7 上に FPGA 実装を行った．Virtex-7 上で実装した HDR アーキテクチャ回路は，正常に動作することを確認した上で，SR アーキテクチャ回路と比べてレイテンシを約 21%削減することを確認した．これは，Vivado 合成結果と同等の結果であり，前章までの提案手法の評価は実 FPGA チップ上でも成り立つと裏付ける結果と考えられる．

第7章　結論

本論文では，FPGA の利用拡大を背景にして，レイテンシ削減を目的としたフロアプラン指向 FPGA 向け高位合成手法を提案した．

第 2 章**「FPGA を対象とした高位合成の研究動向」**では，既存の関連研究のうち，フロアプランを扱う FPGA 向け高位合成手法と MUX コスト削減を図る FPGA 向け高位合成手法に分類されるいくつかの既存手法を紹介した．そして，他の LSI 向けに提案されたフロアプランを扱う高位合成に適したアーキテクチャを紹介した．

既存の FPGA 向け高位合成手法では，フロアプランを扱う手法，MUX コスト削減を図る手法，各々が提案されているが，これらを同時に達成する高位合成手法は実現されていない．フロアプランを扱う FPGA 向け高位合成手法では，フロアプランを含めた高位合成問題を有効に解くために，フロアプラン指向高位合成手法が注目されており，[12,14,15,42] を紹介した．しかし，その中で用いられているフロアプラン指向アーキテクチャや配線遅延・クロックスキューの見積りモデルには，課題がある．一方，MUX コスト削減を図る FPGA 向け高位合成手法では [11,13,19,39,40,63] が提案されており，本章では Chen らの手法 [11] と Hara らの手法 [39,40] を例に取り，紹介した．既存研究では，フロアプランを扱う手法，MUX コスト削減を図る手法，各々が提案されているが，既存手法ではそれぞれが独自のバインディング手法により目的を達成しており，一概に組み合わるのは難しい．従って，これらを同時に達成する高位合成手法は私の知る限り提案されていない．

また，FPGA 上のフロアプランを考慮する有効な技術の 1 つとして，HDR アーキテクチャ[1–3] を紹介した．これは，レジスタ分散型アーキテクチャの 1 つであり，任意の矩形を取るハドルに対しフロアプランするため，レイテンシの小さい回路の生成を目的としたフロアプラン指向 FPGA 高位合成手法に有効であると考えられるが，HDR アーキテクチャを対象とした FPGA 向けの高位合成手法は存在しておらず，フロアプラン指向 FPGA 高位合成手法に採用するためには，新たな高位合成手法の提案が必要となる．

第 3 章**「HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法」**では，FPGA を対象とした高位合成における課題 1「フロアプランの考慮と MUX のコスト削減の同時実現」を解決するため，フロアプランの考慮と MUX のコスト削減を同時に実現する FPGA 向け高位合成手法として，HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法を提案した．まず，FPGA 上に回路の各構成要素を実装・分析し，MUX が FPGA 上でボトルネックであることを明らかにした．そして，MUX の入力数を変化させて実装し，入力数に応じた MUX のコスト増加傾向を分析し，MUX 数の削減・入力数の制限という提案手法の方針を決定した．そして，上記アプローチに従い，HDR アーキテクチャを対象とした MUX 削減 FPGA 高位合成手法を提案した．そして，最後に計算機実験によって既存手法（SR, [2]）と比較した際の提案手法の優位性を明らかにした．

提案手法はレジスタ分散型アーキテクチャの 1 つである HDR アーキテクチャを用いることで，高位合成段階でフロアプランを扱い，モジュール間の配置配線を見積る．また，各 FU (演算器)・レジスタ間のデータ転送を考慮した 2 つの新たな FPGA 向けバインディング手法を用いて，高位合成段階で MUX のコストを削減したデータパスを生成する.「パス考慮スケジューリング/FU バインディング」では，すでにデータパスがある 2 つの FU 同士を積極的に連続する 2 つの演算ノードに割り当てることで MUX 数の削減を図る.「パス考慮レジスタバインディング」では，入出力先の FU・レジスタを考慮してレジスタに付加する MUX を 4 入力以下に制限することで MUX の総面積・遅延を削減する.

計算機実験により，提案手法は従来のレジスタ集中型アーキテクチャを対象とした手法 (SR) と比較して，回路のレイテンシーを最大 22%，平均 4%削減し，スライス数を最大 47%，平均 29%削減することを確認した．また，提案手法は従来の HDR アーキテクチャを対象とした高位合成手法 [2] と比較して，回路のレイテンシーを最大 18%，平均 6%削減し，スライス数を最大 38%，平均 16%削減することを確認した.

第 4 章**「フロアプラン指向 FPGA 高位合成向け配線遅延・クロックスキュー見積りモデル」**では，FPGA を対象とした高位合成における課題 2「高位合成段階における，配線遅延・クロックスキューの正確な見積り」を解決するため，フロアプラン指向 FPGA 高位合成手法のための FPGA の配線遅延・クロックスキュー見積りモデル「IDEF」と「CSEF」を提案した．まず，FPGA 上で様々なパターンで配線遅延を測定し，FPGA の配線遅延特性を明らかにした．そして，その測定結果を基に，FPGA の配線遅延見積りモデル “IDEF” を構築した．次に，FPGA のクロックスキューの影響を測定し，高位合成段階で考慮すべき要素であることを明らかにした．そして，Xilinx 社 Vivado 内のクロックスキューの計算モデルを基に，計算モデル内の各部分の特性を測定から導き，FPGA のクロックスキュー見積りモデル “CSEF” を構築した.

Xilinx 社 Vivado のタイミングモデルと比較して，IDEF は最大誤差 0.20ns，平均誤差 0.10ns，CSEF は最大誤差 0.062ns，平均誤差 0.021ns という見積り精度を達成し，高位合成段階で高精度に見積もれることを確認した.

その後，IDEF・CSEF を第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に適用し，ベンチマークアプリケーションにおいて効果を確認した．見積りモデルを適用した手法は，SR と比較して回路のレイテンシを最大 22 %，平均 10%削減し，第 3 章の手法と比較して最大 11%，平均 6%削減した．また，大規模アプリケーションに限ってでは，SR と比較して平均 20%，第 3 章の手法と比較して平均 9%削減し，より効果的にレイテンシを削減できた．この実験結果より，提案した見積りモデル「IDEF」と「CSEF」をフロアプラン指向 FPGA 高位合成手法に適用することで，よりレイテンシの小さい高性能な回路を合成できることが確認できた.

第 5 章**「配線遅延とクロックスキューを考慮したクリティカルパス最適化 FPGA 高位合成手法」**では，FPGA を対象とした高位合成における課題 3「高位合成段階での配線遅延・クロックスキューの影響と MUX のコストの同時考慮」を解決するため，第 3 章で提案したフロアプラン指向 FPGA 高位合成手法に前章で提案した IDEF・CSEF を利用し，配線遅延とクロックスキューを考慮したクリティカルパス最適化 FPGA 高位合成手法を提案した．提案手法はモジュール配置を基に IDEF と CSEF を用いて，モジュール間の配線遅延とクロック

スキューを見積る．さらに，高位合成段階で配線遅延とクロックスキューを含めた各パスの遅延を見積り，見積り遅延の大きいクリティカルパスの候補となるパスを特定する．そして，データパス生成とフロアプラン両方において，これらを集中的に最適化し回路のレイテンシ—の向上を図る．クリティカルパス指向スケジューリング/FU バインディングでは，フロアプラン情報を基に FU バインディングを改良することで，配線遅延とクロックスキューを改善しクリティカルパス遅延の小さいデータパスの生成を図る．また，クリティカルパス指向ハドル合成/フロアプランではフロアプランの改善において配線遅延とクロックスキューを考慮して，クリティカルパスになりそうなパスを集中的に最適化を行う．そして，クリティカルパス遅延の小さいフロアプラン解を目指す．

計算機実験により，第 3 章で用いた 7 つのベンチマークアプリケーションにおいて，提案手法は SR と比較して，回路のレイテンシを最大 24%，平均 15%削減し，スライス数を最大 24%削減し，平均ではほぼ同程度であることを確認した．また，提案手法は従来の HDR アーキテクチャを対象とした高位合成手法 [2] と比較して，回路のレイテンシを最大 38%，平均 15%削減し，スライス数を最大 26%，平均 14%削減することを確認した．提案手法は， 第 3 章で提案した IDEF と CSEF を用いらずクリティカルパスを特定しない手法と比較して，スライス数を同程度に保った上で回路のレイテンシを最大 10%，平均 6%削減することを確認した．

特に，提案手法は配線遅延。クロックスキューの影響が大きくなりやすい大規模アプリケーションでレイテンシの削減効果が大きく，SR と比較して平均 22%，[2] と比較して平均 19%，[29] と比較して平均 10%削減できることを確認した．さらに，大規模な 3 つの追加アプリケーションで追加実験を行い，SR と比較して，レイテンシを最大 27%，平均 18%削減し，第 3 章の手法に IDEF のみを適用した手法と比べて，レイテンシを最大 14%，平均 8%削減することを確認した．また，提案手法は CSEF を用いてクロックスキューを考慮しているため，一部の大規模アプリケーションで，クロックスキューを改善した上でレイテンシ削減を実現していることが確認できた．

第 6 章**「フロアプラン指向高位合成手法を用いた FPGA 実装と評価」**では，レジスタ分散型アーキテクチャを対象としたフロアプラン指向 FPGA 高位合成を用いた実装方法の確立と実装された回路の評価を行った．

まず，実装環境およびその際のアプリケーション回路に必要となるインターフェース回路について述べ，その後，フロアプラン指向高位合成手法を用いた際の FPGA 実装フローを提案した．そして，第 5 章で提案したフロアプラン指向高位合成手法を用いて，ベンチマークアプリケーションの 1 つである DCT アプリケーションを例に取り，実際に Xilinx Virtex-7 上に FPGA 実装を行った．Virtex-7 上で実装した HDR アーキテクチャ回路は，正常に動作することを確認した上で，SR アーキテクチャ回路と比べてレイテンシを約 21%削減することを確認した．これは，Vivado 合成結果と同等の結果であり，前章までの提案手法の評価は実 FPGA チップ上でも成り立つと裏付ける結果と考えられる．

最後に，今後の課題として「詳細な FPGA リソースの位置の考慮」が挙げられる．本論文で議論したフロアプラン指向 FPGA 高位合成手法では，解の収束性の観点から，詳細な FPGA アーキテクチャを抽象化し，スライスの位置座標のみのフロアプランを扱っていた．しかし，レイテンシの小さい回路を生成する上で効果のある FPGA リソースを FPGA 高位

合成で考慮し，その位置を含めたフロアプランを扱うことでより回路の高速化を図ることができると考える．考慮すべき FPGA リソースの例として，配線リソースと専用演算セルが挙げられる．

FPGA の配線リソースを考慮することで．フロアプラン指向 FPGA 高位合成であつかうモジュール間の配線経路を予測することができ，それにより配線混雑度を考慮することが可能になる．実際の FPGA 上の回路での回路配置密度によっては，配線混雑度は回路の配線遅延に大きな影響を与え，IDEF の見積もり誤差を悪化させる恐れがある．従って．高位合成段階で FPGA の配線リソースを考慮し，モジュール間のグローバル配線を扱うことは，今後の課題の 1 つである．

また，FPGA では DSP ブロックなどの専用演算セルが固定配置されており，これらの使用は回路の高速化を図る上で重要なリソースである．本論文ではこれらを考慮していないが，今後はこれらの配置を取り入れたフロアプラン指向 FPGA 高位合成手法の構築も考えていきたい．

謝辞

本研究は，筆者が早稲田大学大学院基幹理工学研究科博士後期課程在学中に，同大学大学院基幹理工学研究科戸川望教授の指導のもとに行ったものである．本論文を結ぶにあたり，博士後期課程進学以前から 6 年間にも亘り，日常の研究活動ならびに定期ゼミ等において，多大なる御指導，御助言を授かった戸川望教授に深く感謝いたします。同じく常日頃からご指導いただきました同大学大学院基幹理工学研究科柳澤政生教授に心より感謝いたします。柳澤政生教授には，定期ゼミ等の場において，筆者の研究方針・内容に対し鋭いご指摘を頂き，本研究を躍進させることができました．また，柳澤政生教授には本論文の執筆に際して熱心で的確なご教示と共に，多くのご指摘を頂き，本論文をより良い内容にできました．同じくお忙しい中でありながら，多岐にわたるご指摘を頂いた同大学大学院基幹理工学研究科山名早人教授に深く感謝いたします．ご指摘のおかげで，議論の不足点を補うことができ，本論文の質を高めることができました．そして，同大学大学院基幹理工学研究科史又華教授には戸川望教授，柳澤政生教授と共に，定期ゼミの場で研究方針にアドバイス頂き，本研究の推進にご助力いただきました．

本研究の過程では研究室への配属以来，多くの方に終始，適切な御指導ならびに御助言を頂きました，特に，同じ高位合成に関する研究に励む本学講師の川村一志氏には，本研究テーマに対し，広い知見からのご助言，多大なるご支援を頂き，心より感謝申し上げる次第です．同じく本学講師の多和田雅師氏には研究室でのサーバ管理方法や計算機上での実験環境ならびにその他研究活動全般に関して適切な御指導ならびに御助言を頂きました．また，本学大学院基幹理工学研究科情報理工学専攻博士課程修了生の阿部晋矢氏には，研究当初の右も左も分らぬ頃，丁寧にご指導いただき感謝いたします．同じ研究室の同期かつ高位合成に関する研究に励む同博士課程修了生の寺田晃太朗氏には研究内容に関して適切なご助言を頂きました上記の方々の含め，周囲の皆様に御指導を頂いたこと全ては本論文に関する研究活動ならびに現在の研究活動の重要な礎となっています．

同じ研究室の同期である五十嵐啓太氏，吉田慎之介氏には，博士後期課程以前から私生活の面で大変お世話になり，大いに感謝します．お二方のご支援のおかげで，本論文の成果を挙げることができました．また，日本電気株式会社の遠藤健司氏，十楚航氏，西倉実里氏には，博士後期課程進学後，筆者が同企業での勤めと本学での研究活動を両立に苦心する中，多くのご支援いただき感謝いたします．

最後に，本論文に関する研究活動全般にわたり，支援していただいた戸川研究室秘書の渡部周子氏，戸川研究室の皆様，柳澤研究室の皆様，暖かく見守ってくれた家族，心の支えになってくれた全ての友人，日本電気株式会社の同僚，自分の関わった全ての人に深く感謝いたします．

参考文献

[1] S. Abe, M. Yanagisawa, and N. Togawa, "Energy-efficient high-level synthesis for HDR architectures," *IPSJ Trans. on System LSI Design Methodology*, vol. 5, pp.106–117, 2012.

[2] S. Abe, Y. Shi, M. Yanagisawa, and N. Togawa, "MH^4: multiple-supply-voltages aware high-level synthesis for high-integrated and high-frequency circuits for HDR architectures," *IEICE Electronics Express*, vol. 9, no. 17, pp. 1414–1422, 2012.

[3] S. Abe, Y. Shi, K. Usami, M. Yanagisawa and N. Togawa, "Floorplan driven architecture and high-level synthesis algorithm for dynamic multiple supply voltages," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E96-A, No.12, pp. 2597–2611, 2013.

[4] Accellera System Initiative, http://www.accellera.org/downloads/standards/systemc.

[5] B. M. H. Alhafidh, A. I. Daood, M. M. Alawad and W. Allen, "FPGA hardware implementation of smart home autonomous system based on deep learning," in *Proc. of International Conference on Internet of Things 2018*, pp. 121–133, 2018.

[6] R. Ammendola, A. Biagioni, O. Frezza, G. Lamanna, F. Lo Cicero, A. Lonardo, M. Martinelli, P. S. Paolucci, E. Pastorelli, L. Pontisso, D. Rossetti, F. Simula, M. Sozzi, L. Tosoratto and P. Vicini, "NaNet: design of FPGA-based network interface cards for real-time trigger and data acquisition systems in HEP experiments," in *Proc. of 2015 IEEE Nuclear Science Symposium and Medical Imaging Conference*, 2015.

[7] H. Azgin, A. C. Mert, E. Kalali and I. Hamzaoglu, "An efficient FPGA implementation of HEVC intra prediction," in *Proc. of 2018 IEEE International Conference on Consumer Electronics*, 2018.

[8] A. Boutros, B. Grady, M. Abbas and Paul Chow, "Build fast, trade fast: FPGA-based high-frequency trading using high-level synthesis," in *Proc. of 2017 International Conference on ReConFigurable Computing and FPGAs*, 2017.

[9] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T.Czajkowski, S. D. Brown and J. H. Anderson, "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems," *ACM Trans. on Embedded Compututing Systems*, vol. 13, no. 2, pp. 24:1–24:27, 2013.

[10] A. Canis, J. H. Anderson, and S. D. Brown, "Multi-pumping for resource reduction in FPGA high-level synthesis," in *Proc. of the Conference on Design, Automation and Test in Europe*, pp. 194–197, 2013.

[11] D. Chen, J. Cong, Y. Fan, and L. Wan, "LOPASS: a low-power architectural synthesis system for FPGAs with interconnect estimation and optimization," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 18, no. 4, pp. 564–577, 2010.

[12] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and synthesis for on-chip multicycle communication," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 550–564, 2004.

[13] J. Cong, Y. Fan, and J. Xu, "Simultaneous resource binding and interconnection optimization based on a distributed register-file microarchitecture," *ACM Trans. on Design Automation of Electronic Systems*, vol. 14, no. 35, 2009.

[14] J. Cong and B. Liu, "A metric for layout-frinedly microarchitecture optimization in high-level syntesis," in *Proc. of Design Automation Conference*, pp. 1239–1244, 2012.

[15] J. Cong, B. Liu, G. Luo, and R. Prabhakar, "Towards layout-friendly high-level synthesis," in *Proc. of International Symposium on Physical Design*, pp. 165–172, 2012.

[16] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers and Z. Zhang, "High-level synthesis for FPGAs: from prototyping to deployment," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[17] J. Cong, P. Zhang, and Y. Zou, "Optimizing memory hierarchy allocation with loop transformations for high-level synthesis," in *Proc. of the 49th Annual Design Automation Conference*, pp. 1233–1238, 2012.

[18] S. Dai, Y. Zhou1, H. Zhang, E. Ustun, E. F. Y. Young, Z. Zhang, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *Proc. of The 26th IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2018.

[19] U. Dhawan, S. Sinha, S.-K. Lam, and T. Srikanthan, "Extended compatibility path based hardware binding algorithm for area-time efficient designs," in *Proc. of 2010 2nd Asia Symposium on Quality Electronic Design*, pp. 151–156, 2010.

[20] M. Dvorak and J. Korenek, "Low latency book handling in FPGA for high frequency trading," in *Proc. of 17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, 2014

[21] T. S. Elias, and P. B. Dhanusha, "Area efficient fully parallel distributed arithmetic architecture for one-dimensional discrete cosine transform," in *Proc. of2014 International Conference on Control, Instrumentation, Communication and Computational Technologies*, pp. 294–299, 2014.

[22] ExPRESS-Benchmarks, http://express.ece.ucsb.edu/benchmark/, Jan 4, 2016.

[23] 藤原晃一, 阿部晋矢, 川村一志，柳澤政生，戸川望，"フロアプランを考慮したマルチプレクサ入力数制限FPGA向け高位合成手法," 信学技報 VLD2014-41, pp. 219–224, 2014.

[24] 藤原晃一, 阿部晋矢, 川村一志，柳澤政生，戸川望，“フロアプランを考慮したマルチプレクサ削減FPGA高位合成手法,” 情報処理学会DAシンポジウム2014論文集, pp. 109–114, 2014.

[25] K. Fujiwara, S. Abe, K. Kawamura, M. Yanagisawa, and N. Togawa, “A floorplan-aware high-level synthesis algorithm for multiplexer reduction targeting FPGA designs,” in *Proc. of 2014 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 244–247, 2014.

[26] 藤原晃一, 柳澤政生，戸川望，“FPGAの配線遅延特性を利用したフロアプラン指向高位合成手法,” 信学技報 VLD2014-85, pp. 99–104, 2014.

[27] 藤原晃一, 柳澤政生，戸川望，“FPGA向けフロアプラン指向高位合成手法のための配線遅延モデリング,” 電子情報通信学会 2015年総合大会 基礎・境界講演論文集, p. 80, 2015.

[28] K. Fujiwara, M. Yanagisawa, and N. Togawa, “A floorplan-driven high-level synthesis algorithm utilizing interconnection delay characteristics in FPGA designs,” in *Proc. of The 19th Workshop on Synthesis and System Integration of Mixed Information Technologies*, pp. 224–225, 2015.

[29] K. Fujiwara, K. Kawamura, S. Abe, M. Yanagisawa, and N. Togawa, “A floorplan-driven high-level synthesis algorithm for multiplexer reduction targeting FPGA designs,” *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, No.07, pp.1392–1405, 2015.

[30] K. Fujiwara, K. Kawamura, M. Yanagisawa, and N. Togawa, “Clock skew estimate modeling for FPGA high-level synthesis and its application,” in *Proc. of The IEEE 11th International Conference on ASIC*, 2015.

[31] 藤原晃一, 川村一志，柳澤政生，戸川望，“配線遅延とクロックスキューを利用したフロアプラン指向FPGA高位合成手法,” 信学技報 VLD2015-54, pp. 99–104, 2015.

[32] 藤原晃一, 川村一志，五十嵐啓太，柳澤政生，戸川望，“フロアプラン指向高位合成を用いたレジスタ分散型アーキテクチャ回路のFPGA実装,” 信学技報VLD2015-127, pp. 93–98, 2016.

[33] 藤原晃一, 川村一志，柳澤政生，戸川望，“クリティカルパス最適化フロアプラン指向FPGA高位合成手法のアプリケーション適用評価,” 電子情報通信学会 2016年総合大会 基礎・境界講演論文集, p. 79, 2016.

[34] K. Fujiwara, K. Kawamura, M. Yanagisawa, and N. Togawa, “A high-level synthesis algorithm for FPGA designs optimizing critical path with interconnection-delay and clock-skew consideration,” in *Proc. of 2016 International Symposium on VLSI Design, Automation and Test*, 2016.

[35] K. Fujiwara, K. Kawamura, M. Yanagisawa, and N. Togawa, “Interconnection-delay and clock-skew estimate modelings for floorplan-driven high-level synthesis targeting

FPGA designs," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99-A, No.07, pp.1294–1310, 2016.

[36] K. Fujiwara, K. Kawamura, M. Yanagisawa, and N. Togawa, "An FPGA implementation method based on distributed-register architectures," *IPSJ Trans. on System LSI Design Methodology*, vol. 12, 2019.

[37] F. A. Ghani, E. Kalali and I. Hamzaoglu, "FPGA implementations of HEVC sub-pixel interpolation using high-level synthesis," in *Proc. of 2016 11th International Conference on Design & Technology of Integrated Systems in Nanoscale Era*, 2016.

[38] S. Hadjis, A. Canis, R. Sobue, Y. Hara-Azumi, H. Tomiyama, and J. Anderson, "Profiling-driven multi-cycling in FPGA high-level synthesis," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition*, pp. 31–36, 2015.

[39] Y. Hara-Azumi, T. Matsuba, H. Tomiyama, S. Honda, and H. Takada, "Selective resource sharing with RT-level retiming for clock enhancement in high-level synthesis," in *Proc. of 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference Software and Systems*, pp. 1534–1540, 2012.

[40] Y. Hara-Azumi, T. Matsuba, H. Tomiyama, S. Honda, and H. Takada, "Quantitative evaluation of resource sharing in high-level synthesis using realistic benchmarks," *IPSJ Trans. on System LSI Design Methodology*, vol.6, pp.122–126, 2013.

[41] C.Y. Huang, Y.S. Chen, Y.L. Lin, and Y.C. Hsu, "Data Path Allocation Based on Biparite Weighted Matching," in *Proc. of Design Automation Conference*, pp.499–504, 1990.

[42] R. Huang and R. Vemuri, "Forward-looking macro generation and relational placement during high level synthesis to FPGAs," in *Proc. of the 18th International Parallel and Distributed Processing Symposium*, pp.139-144, 2004.

[43] K. Igarashi, M. Yanagisawa, and N. Togawa, "Image synthesis circuit design using selector-logic-based alpha blending and its FPGA implementation," in *Proc. of The IEEE 11th International Conference on ASIC*, 2015.

[44] Intel Corporation, https://www.intel.co.jp/content/www/jp/ja/software/programmable/quartus-prime/hls-compiler.html.

[45] K. R. Kiran, C. A. Kumar, and S. Kumar, "Design and analysis of a novel high speed adder based hardware efficient discrete cosine transform (DCT)," in *Proc. of 2015 Fifth International Conference on Advances in Computing and Communications*, pp. 169–173, 2015.

[46] F. J. Kurdahi, and A. C. Parker, "REAL: A program for register allocation," in *Proc. of the 24th ACM/IEEE Design Automation Conference*, pp. 210–215, 1987.

[47] G. Lemieux, E. Lee, M. Tom and A. Yu, "Directional and single-driver wires in FPGA interconnect,"in *Proc. of The 2004 IEEE International Conference on Field-Programmable Technology*, pp. 41–48, 2004.

[48] G. Lhairech-Lebreton, P. Coussy, D. Heller, and E. Martin, "Bitwidth-aware high-level synthesis for designing low-power DSP applications," in *Proc. of 2010 17th IEEE International Conference on Electronics Circuits*, pp. 531–534, 2010.

[49] C. Li, Y. Bi, Y. Benezeth, D. Ginhac and F. Yang, "High-level synthesis for FPGAs: code optimization strategies for real-time image processing," *Journal of Real-Time Image Processing*, vol. 14, no. 3, pp. 701–712, 2018.

[50] P. Li, P. Zhang, L. N. Pouchet, and J. Cong, "Resource-aware throughput optimization for high-level synthesis," in *Proc. of the 2015 ACM/SIGDA International Symposium on Field-programmable gate arrays*, pp. 1–10, 2014.

[51] S. Li, A. Li, Y. Liu, Y. Xie, H. Yang, "Nonvolatie memory allocation and hierarchy optimization for high-level synthesis," in *Proc. of 2015 20th Asia and South Pacific Design Automation Conference*, pp. 166–171, 2015.

[52] T. Matsuba, Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Aggressive register unsharing based on SSA transformation for clock enhancement in high-level synthesis," in *Proc. of International Symposium on Electronic Design, Test & Applications*, pp. 87–92, 2010.

[53] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1518–1524, 1996.

[54] B. Mohamed, A. Elsayed, O. Amin, E. Khafagy, M. Abdelrasoul1, A. Shalaby, M. S. Sayed, "High-level synthesis hardware implementation and verification of HEVC DCT on SoC-FPGA," in *Proc. of 2017 13th International Computer Engineering Conference (ICENCO)*, 2017.

[55] NEC Corporation, https://jpn.nec.com/cyberworkbench/index.html.

[56] D. H. Noronha, B. Salehpour and S. J. E. Wilton, "LeFlow: enabling flexible FPGA high-level synthesis of tensorflow deep neural networks," arXiv preprint arXiv:1807.05317, 2018.

[57] A. Ohchi, N. Togawa, M. Yanagisawa and T. Ohtsuki, "Floorplan-driven high-level synthesis for distributed/shared-register architectures," *IPSJ Trans. on System LSI Design Methodology*, vol. 1, pp. 78–90, 2008.

[58] A. Ohchi, N. Togawa, M. Yanagisawa and T. Ohtsuki, "Floorplan-aware high-level synthesis for generalized distributed-register architectures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92-A, no. 12, pp. 3169–3179, 2009.

[59] G. S. Oudraogo, M. Gautier, and O. Sentieys, "Frame-based modeling for automatic synthesis of FPGA-software defined radio," in *Proc. of 2014 9th International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, pp. 341–346, 2014.

[60] M. Palczewski, "Plane parallel a maze router and its application to FPGAs," in *Proc. of the 29th ACM/IEEE Design Automation Conference*, pp. 691–697, 1992.

[61] B. C. Schafer, "Enabling high-level synthesis resource sharing design space exploration in FPGAs through automatic internal bitwidth adjustments," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 97–105, 2017.

[62] A. Shastri, G. Stitt, and E. Riccio, "A scheduling and binding heuristic for high-level synthesis of fault-tolerant FPGA applications," in *Proc. of the 26th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp. 202–209, 2015.

[63] S. Sinha, U. Dhawan, S.-K. Lam, and T. Srikanthan, "A Novel binding algorithm to reduce critical path delay during high level synthesis," in *Proc. of 2011 IEEE Computer Society Annual Symposium on VLSI*, pp. 278–283, 2011.

[64] P. Sjovall, J. Virtanen, J. Vanne, T. D. Hamalainen, "High-level synthesis design flow for HEVC intra encoder on SoC-FPGA," in *Proc. of 2015 Euromicro Conference on Digital System Design*, pp. 49–56, 2015.

[65] A. Takahashi, W. Takahashi, and Y. Kajitani, "Clock-routing driven layout methodlogy for Semi-Synchronous Circuit Design," in *Proc. of the 1997 IEEE/ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pp. 63-66, 1997.

[66] M. Tan, S. Dai, U. Guipta, and Z. Zhang, "Mapping-aware constrained scheduling for LUT-Based FPGAs," in *Proc. of the 2015 ACM/SIGDA International Symposium on Field-programmable gate arrays*, pp. 190–199, 2015.

[67] K. Vissers, S. Neuendorffer, and J. Noguera, "Building real-time HDTV applications in FPGAs using processors, AXI interfaces and high level synthesis tools," in *Proc. of the Conference on Design, Automation and Test in Europe*, pp. 848–850, 2011.

[68] Vivado Design Suite, http://www.xilinx.com/products/design-tools/vivado/index.htm.

[69] K. Wakabayashi and T. Yoshimura, "A resource sharing and control synthesis method for conditional branches," in *Proc. of ICCAD '89*, pp. 62–65, 1989.

[70] M. J. Wirthlin, "FPGAs operating in a radiation environment: lessons learned from FPGAs in space," *Journal of Instrumentation*, vol. 8, no. 02,p. C02020, 2013.

[71] M. Xu and F. J. Kurdahi, "Layout-driven high level synthesis for FPGA based architecture," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition*, pp. 446–450, 1998.

[72] Xilinx Inc., http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[73] Xilinx User Guide, "7 Series FPGAs configuration," UG470, June 24, 2015.

[74] Xilinx User Guide, "7 seriens FPGAs clocking resources," UG472, Jun 12, 2015.

[75] R. Ye, T.Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proc. of 2013 IEEE/ACM International Conference on Computer-Aided Designs*, pp. 48–54, 2013.

[76] R. Zhao, M. Tan, S. Daim, and Z. Zhang, "Area-efficient pipelining for FPGA-targeted high-level synthesis," in *Proc. of 52th ACM/IEEE Design Automation Conference*, 2015.

[77] H. Zheng, S. T. Gurumani, L. Yang, D.Chen, and K. Rupnow, "High-level synthesis with behavioral level multi-cycle path analysis," in *Proc. of 2013 IEEE 23rd International Conference on Field Programmable Logic and Applications*, pp. 1–8, 2013.

[78] H. Zheng, S. T. Gurumani, K. Rupnow and D.Chen, "Fast and effective placement and routing directed high-level synthesis for FPGAs," in *Proc. of the 2014 ACM/SIGDA International Symposium on Field-programmable gate arrays*, pp. 200–209, 2014.

本論文に関する発表業績

論文誌

⟨1⟩ ◯ **K. Fujiwara**, K. Kawamura, M. Yanagisawa, and N. Togawa, "An FPGA implementation method based on distributed-register architectures," *IPSJ Trans. on System LSI Design Methodology*, vol. 12, Feb. 2019. (掲載決定)

⟨2⟩ ◯ **K. Fujiwara**, K. Kawamura, M. Yanagisawa, and N. Togawa, "Interconnection-delay and clock-skew estimate modelings for floorplan-driven high-level synthesis targeting FPGA designs," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99-A, No.07, pp.1294–1310, Jul. 2016.

⟨3⟩ ◯ **K. Fujiwara**, K. Kawamura, S. Abe, M. Yanagisawa, and N. Togawa, "A floorplan-driven high-level synthesis algorithm for multiplexer reduction targeting FPGA designs," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, No.07, pp.1392–1405, Jul. 2015.

国際会議（査読付）

⟨1⟩ ◯ **K. Fujiwara**, K. Kawamura, M. Yanagisawa, and N. Togawa, "A high-level synthesis algorithm for FPGA designs optimizing critical path with interconnection-delay and clock-skew consideration," in *Proc. of 2016 IEEE International Symposium on VLSI Design, Automation and Test*, Hsinchu, Taiwan, Apr. 2016.

⟨2⟩ ◯ **K. Fujiwara**, K. Kawamura, M. Yanagisawa, and N. Togawa, "Clock skew estimate modeling for FPGA high-level synthesis and its application," in *Proc. of The IEEE 11th International Conference on ASIC*, Chengdu, China, Nov. 2015.

⟨3⟩ ◯ **K. Fujiwara**, M. Yanagisawa, and N. Togawa, "A floorplan-driven high-level synthesis algorithm utilizing interconnection delay characteristics in FPGA designs," in *Proc. of The 19th Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI 2015)*, pp. 224–225, Yilan, Taiwan, Mar. 2015.

⟨4⟩ ◯ **K. Fujiwara**, S. Abe, K. Kawamura, M. Yanagisawa, and N. Togawa, "A floorplan-aware high-level synthesis algorithm for multiplexer reduction targeting FPGA designs," in *Proc. of 2014 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 244–247, Ishigaki, Japan, Nov. 2014.

国内学会（査読付）

⟨1⟩ **藤原晃一**, 阿部晋矢, 川村一志，柳澤政生，戸川望，"フロアプランを考慮したマルチプレクサ削減 FPGA 高位合成手法," 情報処理学会 DA シンポジウム 2014 論文集, vol. 2014, pp. 109–114, 下呂市, Aug. 2014.

国内学会（査読無し）

⟨1⟩ **藤原晃一**, 川村一志，柳澤政生，戸川望，“クリティカルパス最適化フロアプラン指向 FPGA 高位合成手法のアプリケーション適用評価,” 電子情報通信学会 2016 年総合大会 基礎・境界講演論文集, p. 79, 福岡市, Mar. 2016.

⟨2⟩ **藤原晃一**, 川村一志，五十嵐啓太，柳澤政生，戸川望，“フロアプラン指向高位合成を用いたレジスタ分散型アーキテクチャ回路の FPGA 実装,” 信学技報 VLD2015-127, pp. 93–98, 那覇市, Mar. 2016.

⟨3⟩ **藤原晃一**, 川村一志，柳澤政生，戸川望，“配線遅延とクロックスキューを利用したフロアプラン指向 FPGA 高位合成手法,” 信学技報 VLD2015-54, pp. 99–104, 長崎市, Dec. 2015.

⟨4⟩ **藤原晃一**, 柳澤政生，戸川望，“FPGA 向けフロアプラン指向高位合成手法のための配線遅延モデリング,” 電子情報通信学会 2015 年総合大会 基礎・境界講演論文集, p. 80, 草津市, Mar. 2015.

⟨5⟩ **藤原晃一**, 柳澤政生，戸川望，“FPGA の配線遅延特性を利用したフロアプラン指向高位合成手法,” 信学技報 VLD2014-85, pp. 99–104, 別府市, Nov. 2014.

⟨6⟩ **藤原晃一**, 阿部晋矢, 川村一志，柳澤政生，戸川望，“フロアプランを考慮したマルチプレクサ入力数制限 FPGA 向け高位合成手法,” 信学技報 VLD2014-41, pp. 219–224, 札幌市, Jul. 2014.

受賞

⟨1⟩ 情報処理学会 SLDM 研究会 2015 年度 優秀発表学生賞，情報処理学会 DA シンポジウム 2016.

⟨2⟩ 早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻　専攻賞

⟨3⟩ デザインガイア・ポスター賞，デザインガイア 2015.

⟨4⟩ DA シンポジウム 2015 アルゴリズムデザインコンテスト 優秀賞（学生部門），情報処理学会 DA シンポジウム 2015.

⟨5⟩ 情報処理学会 SLDM 研究会 2014 年度 優秀発表学生賞，情報処理学会 DA シンポジウム 2015.