LUTアレイ用非同期論理回路の合成法

小西 隆介　小栗 清　伊藤 秀之　永見 康一

NTT光ネットワークシステム研究所

〒239-0847 神奈川県横須賀市光の丘 1-1

E-mail: {ryusuke,oguri,hi,nagami}@exa.onlab.ntt.co.jp

あらまし

非同期式の回路設計は、クロックスキューの問題を回避するために有効であると考えられている。一方、HDL を用いるような高位設計では、非同期モデルより同期モデルの方が理解が容易である。我々はこの溝を埋めるため、同期回路を非同期回路にシステマティックに置き換える方法を提案する。本稿の手法で生成された非同期回路は、処理の順序に関して元の同期回路と同じであることを示す。この方法を用いて、従来の同期モデルに基づく動作記述を、規則正しい構造を持つ LUT アレイ上で、非同期回路として実行することを目指している。

キーワード　非同期回路, 論理合成, ペトリネット, FPGA, PCA

A Synthesis Method for Asynchronous Logic Circuits on Array of LUTs

Ryusuke Konishi　Kiyoshi Oguri　Hideyuki Ito　Kouichi Nagami

NTT Optical Network Systems Laboratories

1-1 Hikarinooka Yokosuka-shi Kanagawa 239-0847 Japan

E-mail: {ryusuke,oguri,hi,nagami}@exa.onlab.ntt.co.jp

Abstract

Asynchronous circuit design has been said to be an effective solution to avoid clock-skew problem. On the other hand, with high-level design methods using HDL, synchronous models are easier understood than asynchronous ones. To bridge this gap, we propose a transformation method from synchronous to asynchronous circuits. We show that the sequence of data-processing of the generated asynchronous circuits, is the same as that of the original synchronous ones. Using this method, we are aiming to execute conventional synchronous description based models as asynchronous circuits implemented on a regular LUT array.

key words　Asynchronous circuits, Logic synthesis, Petri-net, FPGA, PCA

1. はじめに

デバイスの集積度が向上するに伴いクロックスキューの問題が生じている。グローバルクロックを前提とした回路の実装には物理的な限界がある。この問題に対してはクロックを持たない非同期式の回路設計を採用することが有効であると言われている。

しかし非同期式の回路設計は同期式のそれと比べると容易ではない。同期回路と同じように設計論を確立し、高位合成システムを実現することが重要である。既に非同期回路の合成システムが幾つか知られている[2][3]。だがこれらのシステムは、非同期の実行モデルを出発点として非同期回路を生成するアプローチを取っているため、次のような問題をはらんでいる。

- 設計を従来手法から大幅に改める必要がある。
- 並列性が増えるに従って、人間がモデルを理解するのが困難になる。

最初の点は、同期式から非同期式への技術移行が容易でないことを意味する。仮にこれを過渡的な障害と考えるとしても、2 点目は本質的な困難であるように思われる。一方同期式設計では、クロックが与える同時性の意味が、並列計算の理解を容易にしている。

そこで本稿では、同期式の設計手法には手を加えずに、同期回路から非同期回路を生成する方法を提案する。すなわち、同期回路からグローバルクロック信号を削除し、それを等価なタイミング信号に置きかえることによりシステマティックに非同期回路を構成する方法を考える。非同期ハンドシェークを保存量として定義するモデルを用いて、変換された非同期回路の動作が元の同期回路と同じであることを示す。さらに、構成された非同期回路を、均質な構造をもつ LUT アレイ上に実現する方法を述べる。

2. ハンドシェーク

2.1. イベントの伝達方法

非同期回路では、信号の変化でしか情報を伝えることができない。信号の変化を伝える方法としては、パルスを使う方法と信号レベルを交互に変化させる方法がある。パルスによる伝達は直感的であるが、伝達経路が長くなると信号が失われる可能性がある。そこで、レベルを変化させる方法を採用する。

2.2. 保存量を定義するハンドシェークの解釈

前提として、非同期回路における情報の伝達はハンドシェークに基づいて制御されるものとする。信号レベルを交互に変化させてハンドシェークをとる最も基本的な方法を図 1に示す。図中で C と書かれたゲートは Muller-C 素子と呼ばれる非同期回路の基本素子で、同じ方向の変化を待ち合わせる動作をする。

この回路でハンドシェークは次のように行われる。

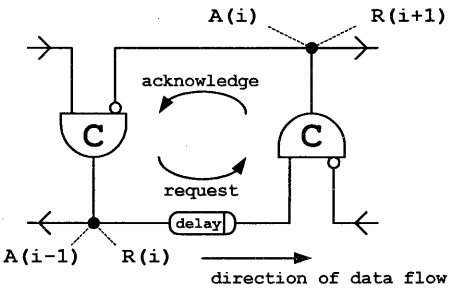

図1: ハンドシェーク回路

1. 左側の Muller-C 素子が右側の Muller-C 素子に要求を信号の変化として伝える。
2. 右側の Muller-C 素子は、変化を通して要求を受け付けたことを応答する。
3. 応答の変化は反転して左側の Muller-C 素子に入る。これにより次の要求が可能になる。
4. 1〜3 を再度繰り返すと信号レベルは元に戻る。

このプロトコルは情報伝達の基本動作としては、直感的に理解するのが難しい。理解を容易にする方法の一つは、保存量を考えることである。そこで我々は、ハンドシェークを次のように解釈する。

要求信号線上の点 R(i)で信号レベルが変化することを、トークンが到来したことに対応づける。そして、応答信号線上の点 A(i)で信号レベルが変化することを、トークンが通過することに対応づける。A(i)と R(i+1)上の変化は同時に起こるので、トークンの数は変化しない。すなわち保存量として扱える。またトークンは同時に複数個重複して存在できない。

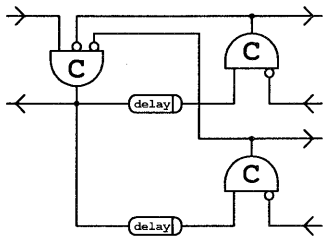

図2: 1 対 2 のハンドシェーク回路

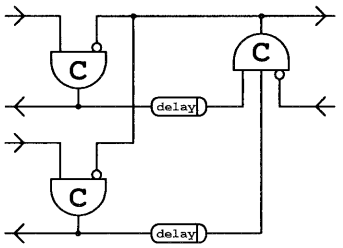

図3: 2 対 1 のハンドシェーク回路

2.3. ハンドシェークの一般形

より一般的なハンドシェークを考える。1 つの送り手に対して 2 つの受け手が存在する場合のハンドシェークを行うための回路は図 2のように構成される。要求信号線と応答信号線が成すループに着目すれば、先程の解釈でトークンを定義できる。

トークンが置ける場所はループに対応して 2 つあることに注意しよう。どちらのループにもトークンが存在しない時に限って、送り手のトークンは 2 つのループに同時に分岐できる。それぞれのループ上のトークンは独立に処理できる。この場合も、経路にそって考えるとトークンの数は変化しない。よってこのトークンは保存量として扱える。

逆に 1 つの受け手に対して 2 つの送り手が存在する場合のハンドシェークを行うための回路は図 3にように構成される。この場合、2 つのループはそれぞれ独立に送り手からのトークンを受け入れることができる。そして両方が揃った時点で、受け手に渡せるようになる。トークンは同様に保存量として扱える。

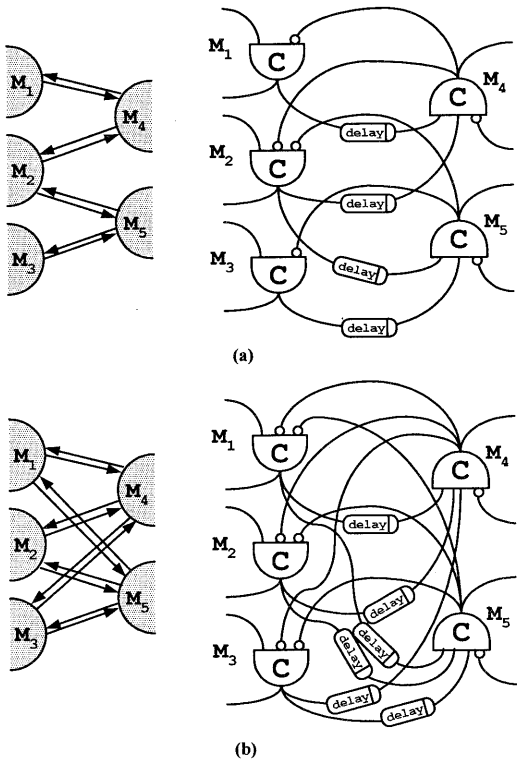

図4: 多対多のハンドシェークパターン

1 つの送り手に対して、3 つ以上の受け手が存在する場合と、1 つの受け手に対して 3 つ以上の送り手が存在する場合も同様である。一方、送り手と受け手がそれぞれ複数ある場合、依存関係に応じてバリエーションが存在する。3 つの送り手(M1,M2,M3)と 2 つの受け手(M4,M5)に対するハンドシェーク回路で、依存関係が異なる例を図 4に示す。図 4 (a)では、依存関係は部分的だが、図 4 (b)では各々の受け手が全ての送り手に依存している。要求信号線と応答信号線が成すループに着目すれば、これらの場合も保存量としてトークンを定義できる。

2.4. ペトリネットによるモデル化

非同期の対象をモデル化する平易な方法としてペトリネット(以下 PN)がよく用いられる。そこで、ハンドシェークと PN の対応を考える。

トークンの存在できる場所はループに対応していたので、これを PN のプレースと見る。Muller-C 素子は変化の伝達を制御しているので、PN のトランジションに対応づける。トークンは PN のトークンに対応づける。ただし、ループに同時に存在するトークンの数を 1 個に限るため、トークンがないことを示す相補的なプレースを追加してモデル化する必要がある。図 5 に図 1 の PN を示す。図中 p_i がループに対応するプレースで、p_i'は p_iの相補的プレースである。

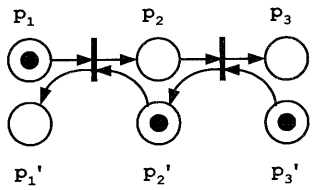

図5: ハンドシェークに対応するペトリネット

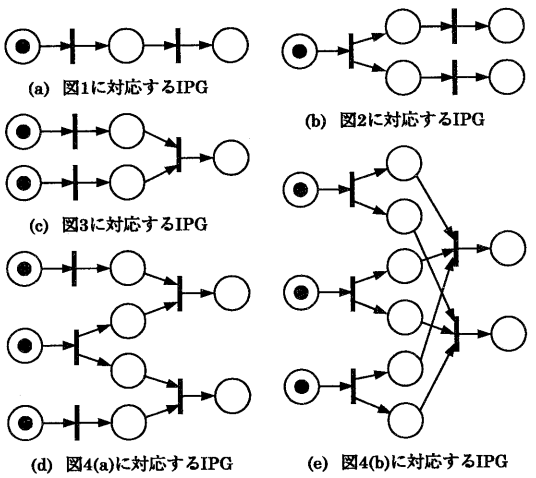

図6: IPG によるハンドシェークのモデル化

このようにして対応付けられた PN では、プレースの入力アークと出力アークはそれぞれ高々1 本しか

ない。このような PN のサブクラスはマークグラフ(Marked Graph)と呼ばれる。

以降では簡単のため、各プレースのトークン数を1に限定する有限容量の PN を用いる。有限容量の PN では「発火後の各出力プレースのトークン数が、有限な容量を越えない」という条件がトランジションの発火条件に課される。全てのプレースについて容量制限が1であるクラスの PN は基本ネットモデルと呼ばれ性質が研究されている[4]。モデル化に用いる、全プレースの制限容量が 1 のマークグラフを以降 IPG (Information Propagation Graph)と呼ぶ。

これまでに現れたハンドシェークに対応する IPG を図6に示す。

3. 同期回路からの非同期回路の合成

3.1. タイミング制御回路の構成

非同期回路を自動的に合成する際の問題の一つは、データパスを正しく制御するタイミングをどのように作り出すかである。STG(Signal Transition Graph)を使うと与えられた仕様(信号遷移の因果関係)を満たすタイミング制御回路を自動的に生成できる[5]。しかし STG による制御回路は一枚岩なので、大規模な非同期回路を合成する際には、制御回路をどう分割し繋ぎ合わせるかボトムアップに考えることになる。

我々は、非同期回路を全体として自動合成するためには、トップダウンのアプローチが望ましいと考えている。このためには単純でかつ必要最小限の部品を選び、それを組み合わせて正しい雛型を構成すれば良い。ここでは単純で見通しのよいモデルに基づいていることが重要である。出来上がった雛型は必ずしも効率が良くないが、これを最適化する作業は分離すべきである。なぜなら最適化はどのようなテクノロジを前提とするかによって大きく変わるし、頻繁に置き換わる部分だからである。

ここでは、タイミング制御回路の雛型の最も基本的な部品として単一の Muller-C 素子を選ぶ。これは Muller-C 素子が IPG のトランジション1個に対応しており、全体構成が IPG と直接対応して考えやすいからである。

3.2. トークンの色付け

トークンが表す変化は実際には、(1)Low レベルから High レベルへの変化と、(2)High レベルから Low レベルの変化がある。レベルが切れ目なく繋がるためには、IPG のマーキングは以下の条件を満たす必要がある。

- 閉路上のトークンの数は偶数であること
- 始点と終点が同じである経路が複数存在する時、それら経路上のトークン数は、偶数か奇数かどちらかに揃っていること

我々は更に、レベルセンシティブな回路を取り扱えるようにするため、2 種類の変化を区別したい。そこで、トークンに色を導入し、(1)の変化に対応するトークンを黒トークン、(2)の変化に対応するトークンを白トークンと呼ぶ。色付きのトークンを取り扱う一般的な方法として CPN(Coloured Petri-Net)が知られている。ここでは簡単のため、トークンの色付けを次の規則で与える。

1. 経路上は、黒トークンと白トークンは交互に現れることとする(図 7(a))。
2. 複数の経路があるトランジションで合流する場合、それぞれの経路を上流方向に辿って最初に現れるトークンは同じ色とする(図 7(b))。
3. あるトランジションから複数の経路が分岐する場合、それぞれの経路を下流方向に辿って最初に現れるトークンは同じ色とする(図 7(c))。

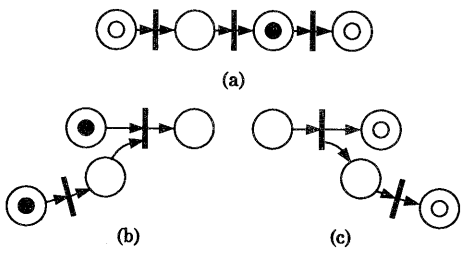

図7: トークンの色付け規則

これらの条件を課すと、有意なマーキングをもつ連結な IPG 上では、あるトークンの色を決めればその他のトークンの色付けは一意に決まる。ここで実行中にトークンの色は変化しないことを前提とすると、同じトランジションで出会うトークンの色は同じになる。発火規則を色に対して拡張する必要はない。また IPG のトークンは経路上で前のトークンを追い越さないので、先の3条件は実行中も持続する。従って、連結なグラフ上では、初期マーキングで少なくとも 1 個のトークンの色を決めることで、初期マーキングから可達な任意のマーキングの色を一般性を損なうことなく決定できる。

3.3. ハンドシェークとデータ転送・加工との関係

データ転送方式には、大きく分けて 2 線エンコーディングを用いるものと束データ方式がある。束データ方式では、データを記憶するラッチをスルー(透過)にする操作と保持にする操作を交互に繰り返すことで、データ転送を行う。保持とは、スルー状態で覚えた有効データを保って出力し続ける状態を指す。

Sutherland のマイクロパイプライン

Sutherland のマイクロパイプライン[6]では、スルーと保持の切り替えをハンドシェーク回路の 2 点

上の信号変化により行う(図 8(a))。この時、変化の方向を対称に扱うラッチ(イベント制御ストレージと呼ばれる)が用いられ、トークンの色に関係なく同じ動作をする。2 つの Muller-C 素子間のプレースにトークンが存在する期間は、変化が図 8(a)の B 点から E 点に至る間である。一方左側のラッチに有効データが存在する期間は、変化が A-B 区間から E-F 区間に至る間である。トークンの存在は離散的であるが、有効データの存在期間は転送のため前後で重複する。プレースにトークンが存在することは有効データが転送・加工中であることを表している(図 8(b))。

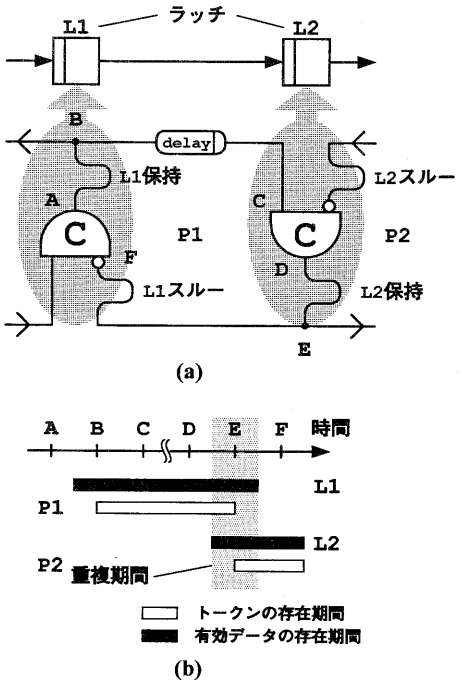

図8: Sutherland のマイクロパイプラインにおけるデータ転送・加工とハンドシェークの関係

RZ 束データ転送

ラッチのスルーと保持を、単に信号レベルに対応させて切り替える方法も考えられる。この考えに基づくデータ転送方式を RZ(Return-to-Zero)束データ方式と呼ぶことにする。図 9(a)に RZ 束データ方式の原理を示す。この場合、ラッチは A-B 区間の制御信号のレベルが Low であればスルー、High であれば保持となる仕様であればよい。トークンと有効データの関係を図 9(b)に示す。この場合、有効データの転送・加工は黒トークンの存在と対応する。

RZ 束データ方式では、同時に有効データを保持できるラッチは 2 段に 1 つとなる。しかし、ラッチはイベント制御ストレージより単純に構成できる。データ幅の大きいデータパスを制御する場合にはラッチの規模が全体に占める割合で大きいので、このことは長所となる。

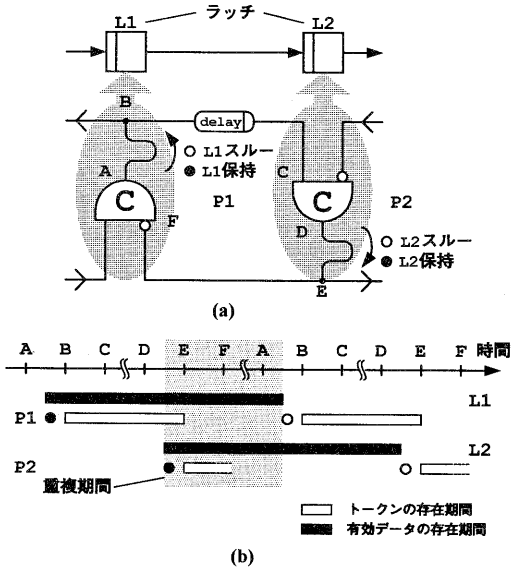

図9: RZ 束データ方式によるデータ転送・加工とハンドシェークの関係

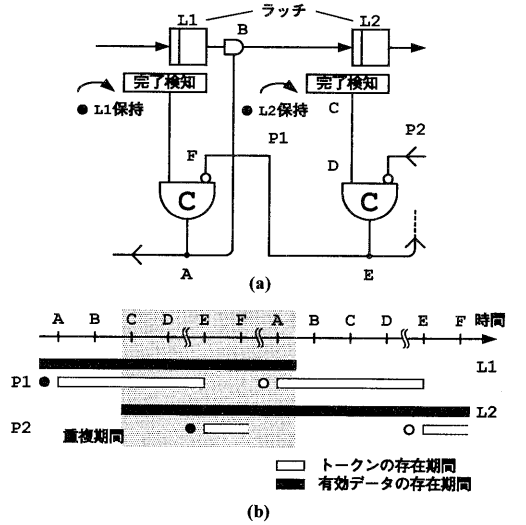

図10: 2 線 2 相式によるデータ転送・加工とハンドシェークの関係

2 線 2 相式データ転送

一方、データ転送・加工を QDI モデル[7]のような厳密な遅延仮定に基づいて行うためには、2 線エン

コーディングを採用する必要がある。実用的な方法として 2 線 2 相式[8]が知られている。2 線 2 相式では、信号レベルを区別して、一方をデータパス上のデータ転送に、他方をクリア制御に用いる。

2 線 2 相式のデータ転送も要求信号がデータパスを経由していると考えれば、これまでと同じハンドシェークの原理で解釈できる。この場合、有効データの転送・加工が黒トークンと対応している(図 10)。

3.4. 同期回路からの変換方法

同期回路は、クロックにより書き込みのタイミングを制御されたレジスタと組み合わせ論理のネットワークである。これを RZ 束データ方式もしくは 2 線 2 相式の非同期回路に変換する方法を与える。同期回路のレジスタは、一般性を損なわずに D-Flip-Flop (以下 D-FF)もしくは書込み許可入力(イネーブル)付きの D-FF であると仮定する。

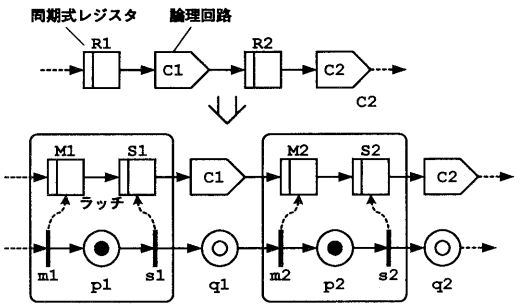

図11: 同期 RTL 網から非同期回路への変換

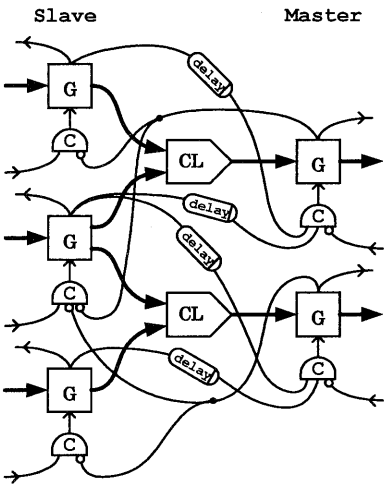

図12: 組合せ論理部分のハンドシェーク回路の構成例

IPG への対応付けを以下のように与える。まず同期式レジスタは 2 段のラッチに展開する。前段をマスタラッチと呼び、後段をスレーブラッチと呼ぶ。それぞれのラッチに対応するトランジションを設け、その間にプレースを設ける(図 11下の長円形内)。

一方、組み合わせ論理は対応する非同期方式の論理回路に変換する。RZ 束データ方式では同期回路と同じものを用いる。2 線 2 相式では、2 線エンコーディングの単調論理に変換する[8]。組み合わせ論理に対応するハンドシェーク回路は、図 12のようにレジスタ間のデータパスの依存関係に応じて構成する。

ハンドシェークは、(1)レジスタ内のマスタラッチとスレーブラッチ間のものと、(2)組み合わせ論理を挟んだレジスタ間のものが生じる。(1)は展開直後、1 対 1 である。(2)はレジスタの依存関係に基づいて構成するので、多対多になりうる。初期化時には、有効データがマスタラッチに格納されていることとする。制御回路の初期状態として、(1)に対応するプレースには黒トークンを置く。(2)に対応するプレースには白トークンを置くこととする。

この段階では全てのプレースにトークンがあるので、IPG が閉路を含んでいると必ずデッドロックになる。論理的に動けるように、閉路上にマージンの空きプレースを 1 つずつ挿入する。閉路だけでなく、通常の経路にも空きプレース挿入してトークンの流れを円滑にしたいと考えるかもしれない。しかし我々は、このような目的のプレースの挿入は最適化の作業として分けて考える。なぜなら、プレースの挿入と性能向上の関係は実現テクノロジに依存しており、プレースの挿入の結果、回路規模が大きくなって、性能が下がる可能性もあるからである。

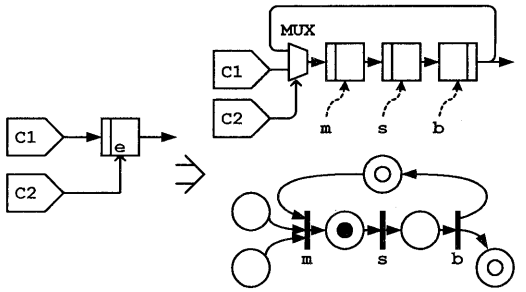

図13: イネーブル付き D-FF の変換

ここまでの変換法では、実はイネーブル付きのレジスタが扱えない。イネーブル付きの D-FF は、クロックが変化する瞬間のイネーブル信号の値により、データを書き込むか、書き込まないかを決めている。イネーブル信号は最終値を採用せねばならず、レベルセンシティブな方法では書込み制御ができない。そこで我々はイネーブル入力付きの D-FF を、自己ループするラッチに展開する(図 13)。デッドロックを防ぐために、自己ループする経路には 3 つのプレースを置く。

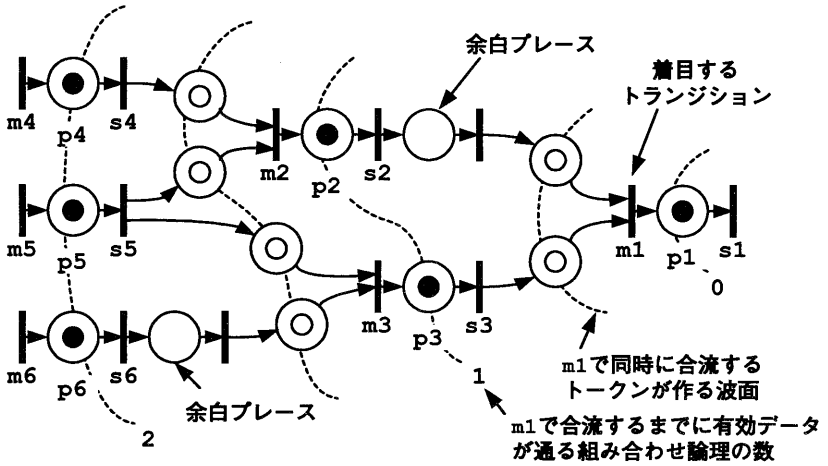

図14: 同時に合流するトークンの波面と有効データが処理される回数

3.5. 同期回路と動作が同じであることの証明

このような方法で同期回路から変換された非同期回路およびその初期状態は、図 14のような初期マーキングをもつ IPG と対応が付く。

今、IPG のマスターラッチに対応するあるトランジションに着目する。説明のため図 14の m1 を選ぶ。トランジション m1 の直後のプレース p1 に黒トークンが存在することは、初期データがレジスタに格納されていることを示している。次の有効データは 1 のラベルのついた波面上の黒トークンが、m1 で合流したときレジスタに格納される。同様に、その次の有効データは 2 のラベルのついた波面上の黒トークンが分岐・合流しながら、m1 で合流したとき格納される。

ここで、m1 で n 回目に合流して p に 1 個の黒トークンを生じさせる黒トークンは、初期マーキングから始めて必ず n 回、組み合わせ論理に対応するプレースを通過している。このことは、レジスタに格納される有効データが、同期回路と全く同じクロック数分、加工されたものであることを表している。言い換えるなら、有効データの処理の順序に関して、変換された非同期回路は元の同期回路と同じ動作をするということである。

例えば、あるモジュールに入力を与えて、3 クロック経過後の出力を別のモジュールが参照するような同期回路があったとする。このとき直感的には、対応する非同期回路において、3 クロック経過した後ということが保証されないように思われる。しかしながら、先ほどの変換法に従うなら、クロックの経過の意味はハンドシェーク回路を通る黒トークンと白トークンの個数として保存されているのである。

4. LUT アレイ上での非同期回路の構成方法

本節では、非同期回路を図 15に示される二次元セルアレイ上に構成する方法を述べる。

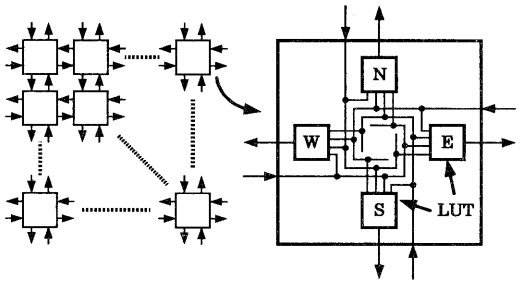

図15: LUT アレイとその基本セルの構造

4.1. LUT アレイの構造と特徴

対象とする基本セルは以下の特徴を有する。

- 東西南北に隣接する基本セルに対し、入力 1bit, 出力 1bit ずつの結線を持つ。
- 内部に 4 つの対称な 4 入力 1 出力の LUT (Look-Up Table)を持つ。
- 論理素子としてレジスタ(ラッチ)を提供しない。

グローバルな配線資源は持たない。その代わり LUT を配線要素として使用することで、遠隔セルとの結線を実現する。この LUT アレイは、単純でかつ非常に均質な構造になっていることが分かる。メモリのように規則正しいこのような構造は、LUT アレイ上の配置配線の問題を単純にするだけでなく、デバイスの集積度を上げるのに効果がある。

4.2. ラッチ及び Muller-C 素子の構成法

組み合わせ論理は LUT を多段に用いて構成される。しかし、ラッチはどのように構成されるのであろ

うか。この答えとして我々は、向かい合った LUT でループを組ませることでラッチを作ることを考えている。例えば、データの記憶に用いるラッチは、図 16 のようにマルチプレクサとして機能する LUT の出力をフィードバックさせて構成する。同様に Muller-C 素子は図 17のように多数決論理として機能する LUT の出力をフィードバックさせることで単純に構成できる。

一方、LUT による構成を前提とすると初期化をどう行うかが問題となる。通常の FPGA ではフリップフロップを初期化するグローバルな信号線が物理的に実装されるが、ここではリセット信号線を LUT アレイ上に配置配線する必要がある。

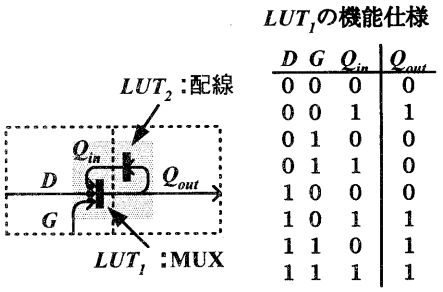

図16: ラッチの構成法

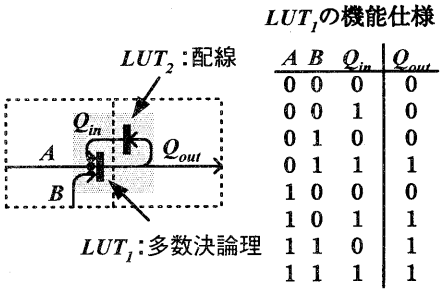

図17: Muller-C 素子の構成法

4.3. 非同期データ転送・加工の構成方式について

これらの部品を組み合わせれば、前節の方法で同期回路から変換された非同期回路を LUT アレイ上に構成できる。この構造上に非同期回路を構成する方法として、我々は束データ方式が最適であると考えている。2 線 2 相式ではデータパスの配線幅(つまり基本セルの列数)が倍になる上、完了信号を集約する回路の構成が現実的でない。一方この構造では、構成する論理に関係なく、LUT のファンイン、ファンアウトが同じで、LUT の通過遅延はほぼ一定となるよう設計できる。遅延時間は通過する LUT の段数として正確に見積もれるので、束データ方式の設計で満たすべき遅延条件は配置配線時に考慮できる。あえて厳密な遅延仮定を適用する必要はないのである。具体的なデータ転送方式としては、ラッチが簡単に構成できるという理由により RZ 束データ方式を採用する。

5. まとめ

同期回路を、クロックが与える実行の同期の意味を保存したまま非同期回路へ変換する方法を示した。通常の非同期回路の構成法と異なるのは、選択・競合などの制御要素を含まず、分岐・合流のみにより構成している点にある。この方法は、非同期式の回路合成を従来の同期式設計の枠組みに統合する一つの方法を与えている。我々は特に、LUT の多段接続だけの世界で、同期式の動作モデルで記述された回路を実行することに興味がある。その成果は、自律的再構成が可能な計算機構であるプラスティック・セル・アーキテクチャ(Plastic Cell Architecture)[9]を、非同期方式で実現するために適用される予定である。

謝辞

同期回路を変形して非同期回路を得ることができる可能性については、NTT DoCoMo の石井 健司氏との議論に端を発する。また得られた非同期回路が元の同期回路と順序に関し同じ動作をすることについては、データサイトの中島 彰社長の示唆を受けた。両氏に感謝いたします。

参考文献

[1] 小栗 清: "VLSI アーキテクチャと設計自動化技術の将来," 信学技報 VLD98-40 (1998-09)

[2] K. van Berkel, et al: "The VLSI programming language Tangram and its translation into handshake circuits," EDAC, pp. 384-389, 1991

[3] V. Akella, et al: "SHILPA: A high-level synthesis system for self-timed circuits," In Proc. ICCAD, pp. 587-591, IEEE Computer Society Press, 1992

[4] L. Bernardinello, et al: "A Survey of Basic Net Models and Modular Net Classes", Advanced in Petri Nets 1992, LNCS Vol. 609, Springer-Verlag, 1992

[5] L. Lavagno, et al: "Algorithms for synthesis of hazard-free asynchronous circuits," In Proc. DAC, pp. 302-308, 1991

[6] I. E. Sutherland: "Micropipelines," Communications of the ACM, Vol. 32, no.6, pp. 720-738, June, 1989

[7] A. J. Martin: "The Limitations to Delay-Insensitivity in Asynchronous Circuits," Advanced Research in VLSI (6th MIT Conf.), pp. 263-278, William J. Dally, 1990

[8] I. David, et al: "An Efficient Implementation of Boolean Functions as Self-Timed Circuits," IEEE Trans. on Computers, Vol. 41, No. 1, pp. 2-11, January, 1992

[9] K.Nagami, et al: "Plastic Cell Architecture: Towards Reconfigurable Computing for General-Purpose," Proc. of FCCM, 1998.