社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2003－ＳＬＤＭ－112　(17)
2003／11／28

セルコントローラに基づいた非同期制御回路の合成

齋藤　寛†　川鍋　昌紀†　今井　雅†　中村　宏†　南谷　崇†

† 東京大学先端科学技術研究センター　〒153-8904 東京都目黒区駒場 4-16-11
E-mail: †{hiroshi,kawanabe,miyabi,nakamura,nanya}@hal.rcast.u-tokyo.ac.jp

あらまし　本稿では、セルコントローラに基づいた非同期制御回路を合成する手法を提案する。本手法では、設計制約とリソースライブラリを用いて高位合成が行われたあと得られたデータパス回路の各演算に対して、セルコントローラをマップすることによってデータパス回路を制御する制御回路を合成する。マッピングという性質上、制御回路の面積と合成時間は、データパス回路における演算の数に比例した値ということがわかる。更に、高位合成から得られたデータパス回路の遅延情報を利用して、制御回路の面積を削減する手法を提案する。両方の手法を組み合わせることによって、多数の演算を実行するデータパス回路を制御する制御回路を効率よく合成することが可能となる。
キーワード　セルコントローラ、scheduled data flow graph、束データ方式、タイミング解析、concurrent vertex substitution

Synthesis of Asynchronous Control Circuits Based on Cell Controllers

Hiroshi SAITO†, Masaki KAWANABE†, Masashi IMAI†, Hiroshi NAKAMURA†, and Takashi NANYA†

† Research Center for Advanced Science and Technology, The University of Tokyo　Komaba 4-6-1, Meguro-ku, Tokyo, 153-8904 Japan
E-mail: †{hiroshi,kawanabe,miyabi,nakamura,nanya}@hal.rcast.u-tokyo.ac.jp

Abstract　We propose a synthesis method of asynchronous control circuits based on cell controllers. After the high-level synthesis of a datapath circuit with design constraints and a resource library, the control circuit is synthesized by mapping a cell controller for each data operation in the datapath circuit. Because of the nature of mapping, the area and the synthesis time of the control circuit are proportional with respect to the number of data operations in the datapath circuit. In addition, to reduce the area of the control circuit, we present an optimization method by using datapath delay information derived from the high-level synthesis. By combining both methods, we can synthesize control circuits efficiently which control a large number of data operations.
Key words　cell controller, scheduled data flow graph, bundled data implementation, timing analysis, concurrent vertex substitution

1. はじめに

本稿では、セルコントローラに基づいた非同期制御回路の合成手法を提案する。本手法では、設計制約とリソースライブラリを用いて高位合成が行われたあと得られたデータパス回路の各演算に対して、セルコントローラをマップすることによってデータパス回路を制御する制御回路を合成する。マッピングの性質上、制御回路の面積と合成時間は、データパス回路における演算の数に比例した値ということがわかる。それゆえ、多数の演算を実行するデータパス回路を制御する制御回路を現実的な時間で合成できるといった利点がある。

提案された合成手法に加えて、本稿では、制御回路の面積削減のための最適化手法を提案する。具体的には、セルコントローラのマッピングの前に、高位合成によって得られたデータパス回路の遅延情報を利用して、セルコントローラの代用を検討する。データパス回路のクリティカルパス遅延を侵害しない場合に限って代用を行い、制御回路の面積を削減する。

セルコントローラ自体は、Martin の Q-element [1] といった既存のものを利用することも可能であるが、ここではデータパス回路を性能の面で効率良く制御するために、2 つの新しいセルコントローラを提案する。これらのセルコントローラの制御プロトコルは Q-element のものと比べ複雑なものとなっているが、論理回路自体は Petrify [2] や 3D [3] といった既存の論理合成ツールを利用して簡単に得ることができる。

関連研究として、Ku らが提案したセルコントローラを用いた制御回路の合成手法 [4] がある。この手法では同期回路設計を念頭においているが、我々の手法では非同期回路設計を念頭においている。非同期回路設計の領域では、Martin の Q-element を利用した合成手法 [1] が我々の提案するものと類似している。しかしながら、データパス回路を性能の面で効率的に制御するために、本稿では、新たに 2 つのセルコントローラを提案している。また、本稿では、遅延情報を利用した最適化手法を新た

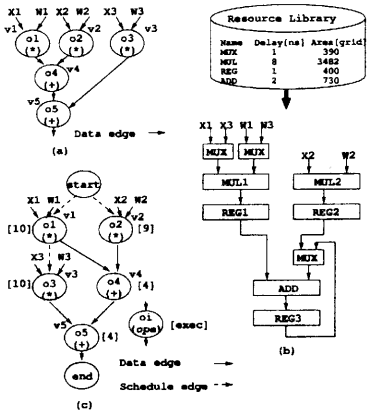

図 1 (a)DFG, (b) データパス回路, (c)SDFG
Fig. 1 (a)DFG, (b)Datapath circuit, (c)SDFG

に提案している。この手法を利用することによって、制御回路全体の最適化が可能となる。文献 [5], [6] の手法も、FSM の各状態に David Cell と呼ばれるセルコントローラをマッピングすることによって制御回路を合成するといった点で、我々の提案する手法と類似している。しかしながら、これらの手法でも、遅延情報を利用した最適化手法に対する検討は行われていない。

本稿の構成は以下の通りである。第 2 章で本稿で利用されるグラフモデルと新たに提案するセルコントローラについてのべる。第 3 章でセルコントローラに基づいた非同期制御回路の合成手法についてのべる。第 4 章で遅延情報を利用した面積最適化手法についてのべる。第 5 章で幾つかのベンチマーク回路を用い提案された手法を評価する。最後に、第 6 章でまとめと今後の展望をのべる。

2. 準　　備

2.1 Data Flow Graph

Data Flow Graph(DFG) は有向グラフで、$G < V, E >$ として表される。ここで、頂点の集合 $V(V = \{v1 \ldots vn\})$ は、データパス回路で実行される演算 $(o1 \ldots on)$ を表し、枝の集合 E は、演算間のデータ依存関係を表す。ここでは枝の集合 E に含まれる枝をデータエッジと呼ぶ。図 1(a) は、DFG の一例を表す。

2.2 非同期回路における高位合成

高位合成は、実行される演算のスケジュールや利用されるリソースを決めることによって、データパス回路の構成を決める作業である。本稿では、非同期高位合成ツール Mercury [7] を使うことによって、このプロセスは既に終了したものと仮定する（すなわち、制御回路合成の対象となるデータパス回路は既に決定している）。Mercury は、入力として DFG とリソースライブラリを受け取ると、面積や性能の効率が良いデータパス回路を自動生成する。クロックに制約される同期回路のスケジューリングとは異なり、非同期回路のスケジューリングは、リソースと入力データの利用可能な状態に制約される。なお、リソースライブラリの各リソースは、面積や遅延がパラメータ化されている。図 1(b) は、図 1(a) の DFG を Mercury に入力し得られたデータパス回路を表す。ここでは、2 つの乗算器と 1 つの加算器が割り当てられている。

2.3 Scheduled Data Flow Graph

Scheduled Data Flow Graph(SDFG) は、DFG の拡張であり、高位合成によって定まった演算のスケジュール情報と遅延情報を新たに含んだものである。SDFG は $G' < V', E', D >$ で表される。頂点の集合 V' は、与えられた DFG における頂点の集合 V に start、end 頂点を加えたものである。start、end 頂点は演算のない参照頂点で、データパス回路における演算の開始と終了を表す。枝の集合 E' は、与えられた DFG の枝の集合 E に、新たにスケジュールエッジを加えたものである。スケジュールエッジは、リソースの共有といったスケジュール情報を表す枝である。遅延の集合 D は、各演算を実行するのに必要とされる最大遅延時間 (exec と表す) の集合である。exec の値は、演算に割り当てられたリソースから概算される。図 1(c) は、図 1(a) の DFG に対して高位合成を行った結果得られた SDFG である。ここでは、start、end 頂点と 3 つのスケジュールエッジが新たに付加されている。また、各頂点には exec の値がアノテートされている。

2.4 セルコントローラ

セルコントローラは、データパス回路で実行される演算やその集合を制御する。Mertin の Q-element [1] のような既存のセルコントローラを利用することが可能であるが、本稿では、データパス回路を性能の面で効率よく制御するために、2 つのセルコントローラ $C1_i$ と $C2_i$ を新たに提案する。$C1_i$ や $C2_i$ におけるインデックス $i(i = 1 \ldots n)$ は、SDFG 上の頂点 vi と対応がとられ、演算 oi を制御する。

$C1_i$ の入出力インターフェースは以下の通りである (図 2(a) を参照)。$C1_i$ は、入力信号 in_i と出力信号 out_i を用いて他のセルコントローラと通信をし、出力信号 req_i と入力信号 ack_i を用いて演算 oi の結果を書き込むレジスタを制御し、出力信号 mux_i を用いて演算 oi で利用されるマルチプレクサを制御する。$C2_i$ は、mux_i がないということを除いて、$C1_i$ と同じである。

$C1_i$ の信号における制御プロトコルは以下の通りである。$C1_i$ は、信号 in_i の立ち上がり $(0 \rightarrow 1)$ 遷移である in_i+ が入力されたあと、信号 req_i と mux_i の立ち上がり遷移である req_i+ と mux_i+ を出力することによって、演算 oi を制御する。演算の終了を表す信号 ack_i の立ち上がり遷移である ack_i+ が入力されたあと、続く演算を続くセルコントローラで制御するために、信号 out_i の立ち上がり遷移である out_i+ を出力する。次サイクルにおける演算 oi の制御が信号 in_i の立ち下がり遷移 $(1 \rightarrow 0)$ である in_i- でトリガされる前に、信号 req_i の立ち下がり遷移である req_i- を出力し、信号 ack_i の立ち下がり遷移である ack_i- をまつ。

図 2(b) は、$C1_i$ の制御プロトコルを signal transition graph(STG) [8] で表したものである。STG における頂点は信号遷移を表し、有向枝は信号遷移間の因果関係を表す。信号遷移 in_i+ でトリガされる信号 req_i、ack_i、mux_i の信号遷移は、インデックス 1 でラベルづけされている (例えば $req_i+/1$)。一方、信号遷移 in_i- でトリガされる信号 req_i、ack_i、mux_i の信号遷移は、インデックス 2 でラベルづけされている (例えば $req_i+/2$)。$C1_i$ の論理回路は、図 2(b) の STG を非同期論理合成ツール Petrify [2] に入力することによって合成される。図 2(c) は、Petrify と NEC CB-C10 ライブラリ [9] を用いて合成された $C1_i$ の論理回路を表す。

ここではなぜ提案するセルコントローラが、Martin の Q-element より性能の面で優位に演算を制御できるかを、図 3 で表された 2 つの連続するセルコントローラのタイミング図を例に説明する。図 3 の上のタイミング図は提案するセルコントローラを利用した時の場合であり、下のタイミング図は Martin の Q-element を利用したときの場合である。まず第一に、提案するセルコントローラを利用した場合、2 番目のセルコント

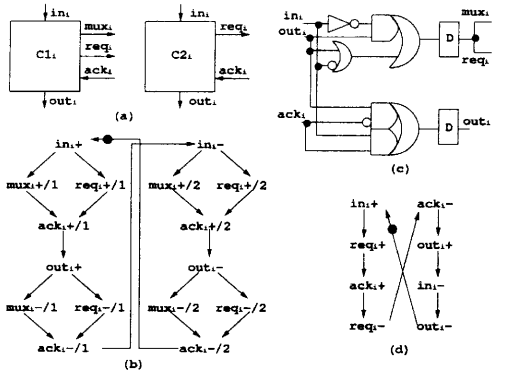

図 2 (a) セルコントローラ, (b)$C1_i$ に対する STG, (c)$C1_i$ の論理回路表現, (d)Q-element に対する STG

Fig. 2 (a)Cell controllers, (b)STG of $C1_i$, (c)Logic implementation of $C1_i$, (d)STG of Q-element

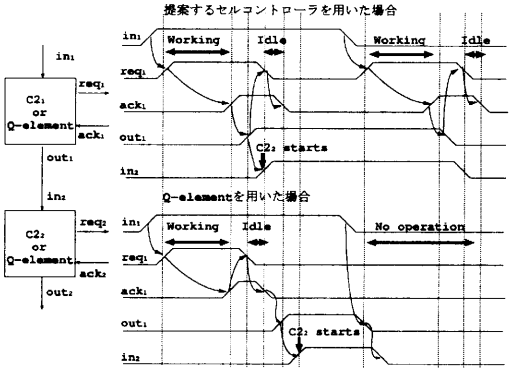

図 3 2 つの連続するセルコントローラのタイミング図

Fig. 3 Timing diagrams of two continuous cell controllers

ローラ ($C2_2$) は、Q-element を利用した場合よりも早く制御が開始される。Q-element を利用した場合、信号遷移 ack_1- が行われない限り 2 番目のセルコントローラをトリガする信号遷移 out_1+ は実行されないが、提案するセルコントローラを利用した場合信号遷移 ack_1+ が行われたあとすぐに実行される。第二に、提案するセルコントローラを利用した場合、次のサイクルにおける演算の制御は in_i- によって開始されるが、Q-element を利用した場合、次の in_i+ まで制御は行われない。

提案するセルコントローラの制御プロトコルは、Q-element の制御プロトコル (図 2(d)) と比べて複雑なものとなっているが、面積に対するオーバーヘッドはわずかである (グリッド数で 10)。

2.4.1 束データ方式によるデータパス回路の制御

本稿では、データパス回路は束データ方式で実装されたものと仮定する (すなわち、1 ビットのデータは 1 本の配線で表現される)。図 4 は、束データ方式によるデータパス回路とセルコントローラのインターフェースを表す。束データ方式では、リソースで費やされる時間とレジスタのセットアップ時間が経過したあと (図 4 の遅延素子 $d1_i$ に相当)、req_i+ が enb_i+ としてレジスタをトリガする。レジスタのホールド時間が経過したあと (図 4 の遅延素子 $d2_i$ に相当)、req_i+ が ack_i+ としてセルコントローラに戻る。なお、遅延 $d1_i$ と $d2_i$ の合計は、SDFG 上の対応する頂点の exec に対応する (図 1(c) を参照)。

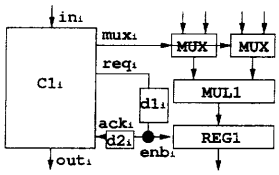

図 4 データパス回路とセルコントローラ間のインターフェイス

Fig. 4 Interface between datapath circuit and cell controller

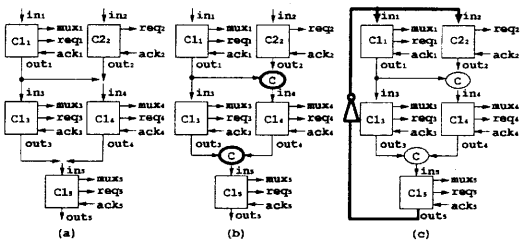

図 5 (a) マップされたセルコントローラ、(b)C 素子の挿入、(c) フィードバックループの挿入

Fig. 5 (a)Mapped cell controllers, (b)Insertion of C-elements, (c)Insertion of a feedback loop

3. セルコントローラに基づいた非同期制御回路の合成

与えられたデータパス回路を制御する非同期制御回路は、以下の手順で合成される。

- セルコントローラのマッピング
- グルーロジックの挿入
- 遅延素子の挿入
- タイミング調整

以下の各節で、手順の詳細を説明する。

3.1 セルコントローラのマッピング

与えられた SDFG 上の start、end 頂点を除く各頂点に、1 つのセルコントローラをマップすることによって制御回路を合成する。マッピングの際、対象となる演算 oi でマルチプレクサが使われる場合、セルコントローラ $C1_i$ がマップされる。一方、マルチプレクサが使われない場合、セルコントローラ $C2_i$ がマップされる。図 5(a) は、図 1(c) の SDFG の各頂点に、セルコントローラをマップした結果得られた制御回路を表す。

マッピングの性質上、制御回路の面積と合成時間は、データパス回路における演算の数に比例した値ということがわかる。それゆえ、多数の演算を実行するデータパス回路を制御する制御回路を現実的な時間で合成できるといった利点がある。しかしながら、たくさんのセルコントローラがマップされた時、制御回路の面積は非常に大きなものとなってしまう。この問題を解くために、次章で面積を削減するための手法を提案する。

3.2 グルーロジックの挿入

グルーロジックは以下の 3 つの状況で挿入される。

a) セルコントローラの同期

与えられた SDFG 上で、2 つ以上のセルコントローラによってトリガされるセルコントローラが存在する可能性がある。そのような場合、セルコントローラ間で同期をとるために、C 素子を挿入する必要がある。C 素子の論理関数は $a = a(b+c)+bc$ で表され、以下のように動作する。入力 b と c が 1 のとき、出力 a は 1。b、c が 0 のとき、a は 0。それ以外では a の値は変化しない。図 1(c) の SDFG を例に、C 素子の挿入を説明する。

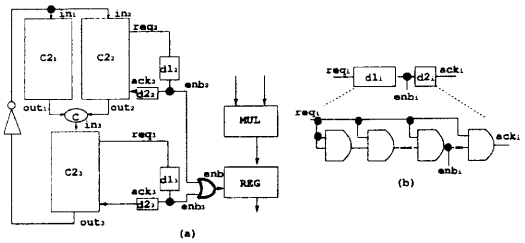

図 6 (a) 共有リソースに対するグルーロジック、(b) 遅延素子
Fig. 6 (a)Glue logics for shared resources, (b)Delay element

ここでは、演算 $o1$ と $o2$ の終了のあと演算 $o4$ が実行される。このような関係を図 5(a) における制御回路で正しく反映するために、$C1_4$ の前に C 素子を挿入し、$C1_1$ の信号 out_1 と $C2_2$ の信号 out_2 の同期をとる。挿入された C 素子から $C1_4$ の入力信号 in_4 を生成する。図 5(b) は、2 つの C 素子が挿入されたあとの制御回路を表す。

b) 次のサイクルを開始するためのトリガ信号生成

次のサイクルを開始するために、フィードバックループとインバータを図 5(c) のように挿入する。

c) 共有リソースへのアクセス

データパス回路におけるリソースの共有によって、複数のセルコントローラによって制御されるリソースが存在する可能性がある。複数のセルコントローラから共有リソースへのイネーブル信号を生成するために、OR ゲートを挿入する。図 6(a) の例で、共有レジスタ REG はセルコントローラ $C2_2$ と $C2_3$ によって制御されるので、REG を制御するイネーブル信号 enb を生成するために OR ゲートを挿入する。

3.3 遅延素子の挿入

束データ方式で、レジスタにデータを正しく書き込むタイミングを保証するために、各 req_i に対して遅延素子を挿入する。図 6(b) のように, 遅延素子は 2 入力の AND ゲートのカスケードから構成される。リソースで費やされる時間とレジスタのセットアップ時間が経過したあと、req_i+ が enb_i+ としてレジスタをトリガする。レジスタのホールド時間が経過したあと、req_i+ が ack_i+ としてセルコントローラに戻る。SDFG 上の各頂点にアノテートされた exec の値より、各遅延素子に必要な AND ゲートの数を概算することができる。なお、AND ゲートのカスケードといった性質上、立ち下がり遷移 req_i- は、AND ゲート 1 つ分の遅延時間で ack_i- に伝達される。

3.4 タイミング調整

セルコントローラの制御プロトコルによると、図 6(a) のような連続した 2 つのセルコントローラ $C2_2$ と $C2_3$ が、ある共有リソース (この場合 REG) を同時に制御してしまう可能性がある。すなわち、$C2_2$ が信号遷移 $req_2/1-$ を出力し $ack_2/1-$ を待っている最中に、$C2_3$ が信号遷移 $req_3+/1$ を出力する可能性がある。共有レジスタ REG のイネーブル信号 enb は OR ゲートから生成されているので、$req_3+/1$ が先に実行されてしまうと、あとに続く $req_2/1-$ によって enb の信号遷移 $enb+$ を無効にしてしまうおそれがある。このような動作はハザードと呼ばれ誤動作を導くので、ハザードのない回路を実現しなければならない。ハザードをなくすためには、$C2_2$ において $ack_2-/1$ が入力されたあとに、$C2_3$ において $req_3+/1$ が出力されることを保証しなければならない。このような動作タイミングを保証するために、$C2_2$ と $C2_3$ の間にバッファのカスケードからなる遅延素子を挿入する。

実際にハザードの可能性があるかどうかは、以下のタイミング制約をチェックすることによって確認することができる。

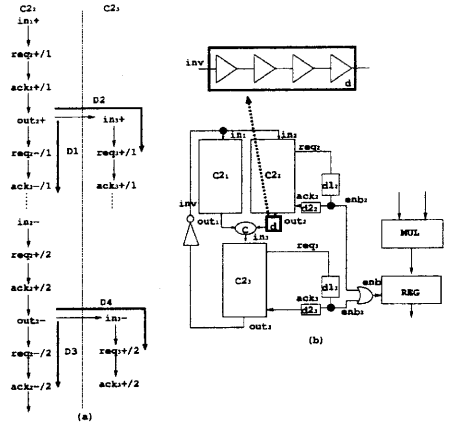

図 7 (a)STG から得られたトレース、(b) 遅延素子の挿入
Fig. 7 (a)Trace derived from STGs, (b)Insertion of a delay element

- タイミング制約 1: $D2 > D1$

ここで、$D2$ は $C2_2$ における out_2+ の出力から $C2_3$ における $req_3+/1$ の出力までの時間を表し、$D1$ は $C2_2$ における out_2+ の出力から $C2_2$ における $ack_2-/1$ の入力までの時間を表す (図 7(a) の STG からえられたトレースを参照)。なお、$D1$ と $D2$ は、与えられたデータパス回路と合成された制御回路に対する論理回路をタイミング解析することによって得ることができる。同様に、

- タイミング制約 2: $D4 > D3$

ここで、$D4$ は $C2_2$ における out_2- の出力から $C2_3$ における $req_3+/2$ の出力までの時間を表し、$D3$ は $C2_2$ における out_2- の出力から $C2_2$ における $ack_2-/2$ の入力までの時間を表す。

信号遷移 $req_3+/j(j=1,2)$ は、信号遷移 in_3+ と in_3- の両方でトリガされるので、タイミング制約 1 と 2 の両方が満たされる必要がある。仮にもし 2 つの制約のうちいづれかが満たされないときは、図 7(b) のように、$C2_2$ と $C2_3$ の間にバッファのカスケードからなる遅延素子を入れてタイミングを調整する必要がある。

4. 遅延情報を利用した面積最適化手法

この章では、SDFG 上の各頂点にアノテートされた演算を実行するのに必要な遅延時間 (exec) を利用して、制御回路の面積を最適化する手法をのべる。

4.1 SDFG 上でのタイミング解析

最適化手法をのべるまえに、SDFG 上でのタイミング解析についてのべる。

4.1.1 開始/終了時間の計算

SDFG 上の各頂点に対して、開始 (start)/終了 (end) 時間を計算する。start は、対応する演算がデータパス回路で実行を開始する時間を表し、end は、対応する演算がデータパス回路で実行を終了する時間を表す。クロックで制約されない非同期回路では、SDFG 上の各頂点に対して開始/終了時間を計算することによって、データパス回路における演算の実行時間を解析することが可能となる。

各頂点に対する start と end は、start 頂点から end 頂点に向けて以下のように計算される。start 頂点の直後の頂点集合に

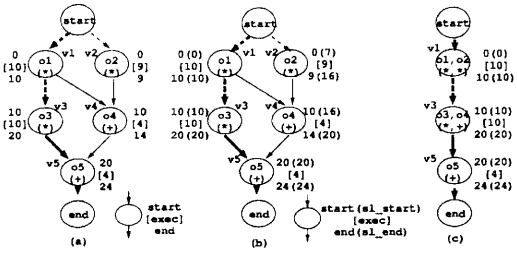

図 8 (a) 開始/終了時間の計算、(b)Slack の計算、(c)Concurrent vertex substitution の結果

Fig. 8 (a)Start and end time calculation, (b)Slack calculation, (c)Result of concurrent vertex substitution

属する頂点の start は 0 である。それ以外の各頂点の start は、その直前の頂点集合のうち end がもっとも大きいものの end の値と同じ値になる。一方、各頂点の end は、その頂点における start と exec の値を足し合わせたものである。図 8(a) は、図 1(c) の SDFG 上の各頂点に対して開始/終了時間を計算した結果を表す。

開始/終了時間を計算することによって、データパス回路上のクリティカルパスを形成する演算を特定することができる。図 8(a) の例では、頂点 $v1$、$v3$、$v5$ に対応する演算がクリティカルパスを形成している (図 8(a) のなかで、クリティカルパスは太線で表されている)。

4.1.2 Slack の計算

開始/終了時間の計算のあと、SDFG 上の各頂点に slack(遅延余裕度) を導入する。slack は、各頂点の start と end のそれぞれに定義される。start に対する slack(sl_start) は、開始時間に対する遅延余裕度を表し、end に対する slack(sl_end) は、終了時間に対する遅延余裕度を表す。したがって、各頂点が有効な slack(sl_start > start) をもつならば、その頂点の開始時間を start と sl_start の範囲内で遅らせても、データパス回路内のクリティカルパス遅延を遅らせることはない。なお、有効な slack をもった頂点とは、クリティカルパス上にない頂点に限定される。

Slack は、開始/終了時間の計算とは逆に、end 頂点から start 頂点に向けて計算される。end 頂点の直前の頂点集合に属する頂点の sl_end は、それらの end のうち、もっとも値が大きいものと同じ値になる。それ以外の頂点の sl_end は、直後の頂点集合に属する頂点の sl_start のうち、もっとも値が小さいものと同じ値になる。一方、各頂点の sl_start は、sl_end から exec を引いた値になる。図 8(b) は、図 8(a) の SDFG 上の各頂点に対して slack を計算した結果を表す。この例では、頂点 $v2$ と $v4$ が有効な slack を持つ。

4.2 Concurrent Vertex Substitution

Concurrent vertex substitution は、SDFG 上で解析された遅延情報をもとに、頂点の代用を行うことによって、頂点の数を削減する手法である。マップされるセルコントローラの数は SDFG 上の頂点の数に比例するので、頂点の数の削減は制御回路の面積削減に直結する。なお、頂点の代用は、データパス回路におけるクリティカルパス遅延を遅らせないように行われる。

4.2.1 代用の条件

SDFG 上の平行な頂点 vi と $vj(i,j = 1 \ldots n, i \neq j)$ に対して、vi は vj に対する以下の代用条件を満たすときにかぎり vj を代用することができる。代用条件は、頂点の代用がデータパス回路におけるクリティカルパス遅延を遅らせないことを保証

```
入力:  ある SDFG
出力:  削減された SDFG

procedure concurrent_vertex_substitution
  頂点集合 V(V ⊂ V') を頂点集合 C と NC に分割する
  (C はクリティカルパス上にある頂点の集合、NC はそうでない頂点の集合)
  C と NC のそれぞれで、start の小さい順に並べ替えをおこなう
  C と NC の順で、V を更新する
  while V は空でない
    V から頂点 v を選択し削除する
    以下の条件を満たす頂点集合 VC(VC ⊂ V) を探索する
    1)VC に含まれる頂点 vc は、v と順序付けがされていない
    2)VC に含まれる頂点 vc に対して、v は代用条件を満たす
    while VC は空でない
      VC から頂点 vc を選択し削除する
      頂点 vc を頂点 v で代用する
      V から頂点 vc を削除する
      開始/終了、slack を再計算する
    end while
  end while
end procedure
```

図 9 Concurrent vertex substitution アルゴリズム

Fig. 9 Algorithm of concurrent vertex substitution

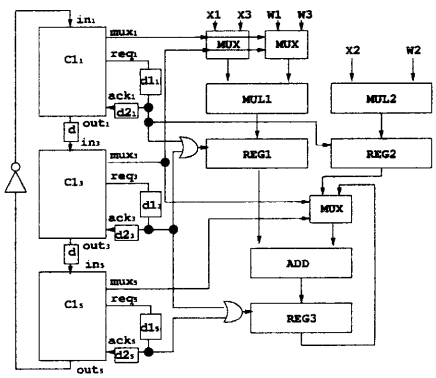

図 10 図 1(a) の DFG に対応した回路構成

Fig. 10 Circuit structure for the DFG in Figure 1(a)

したものである。代用条件は以下の通りである。

- vj(start) $\leq$ vi(start) $\leq$ vj(sl_start)
- vj(end) $\leq$ vi(end) $\leq$ vj(sl_end)
- vj(exec) $\leq$ vi(exec)

ここで、$vi(time)$ は頂点 vi の start、end、exec、sl_start、sl_end のいずれかに該当する。最初の条件は、vi の開始時間が vj の start と sl_start の範囲内にあるということである。2 番目の条件は、vi の終了時間が vj の end と sl_end の範囲内にあるということである。最後の条件は、vi の exec が vj の exec より大きいということである。もし、vi が以上の条件を満たすなら、vi は vj を代用することができる。さもなければ、vj の代用は行われない。

例として、図 8(b) の SDFG で、頂点 $v1$ は $v2$ を代用できるかを調べる。頂点 $v1$ と $v2$ は同じ開始時間をもち、$v1$ の終了時間は $v2$ の end と sl_end の範囲 $(9 < 10 < 16)$ にあり、$v1$ の exec は $v2$ の exec より大きい $(9 < 10)$ ということがわかる。したがって、図 8(c) のように、$v1$ は $v2$ を代用することができる。この結果、頂点 $v1$ に対応するセルコントローラは演算 $o1$ と $o2$ の両方を制御することになる。

4.2.2 アルゴリズム

図 9 は、concurrent vertex substitution アルゴリズムを表す。アルゴリズムの概要は以下の通りである。まず第一に、頂点の代用を行う順番を決めるために、頂点集合 $V(V$ は SDFG 上の頂点のうち、start、end 頂点を除いた頂点集合である) をクリティカルパス上にある頂点の集合 C とそうでない集合 NC

表 1　ベンチマーク回路の面積
Table 1　Area of benchmark circuits

name	DP					without sharing						with sharing					
	+	*	R	M	total	ver.	cell	glue	delay	timing	total	ver.	cell	glue	delay	timing	total
FIR3	1	2	3	3	10064	5	384	36	816	8	1244	3	240	16	516	16	792
IIR2	2	2	4	3	11194	8	608	68	1236	16	1928	5	384	28	738	16	1166
FIR5	1	2	4	6	11796	9	720	90	1470	32	2312	5	400	50	636	32	1118
LMS4	2	2	6	7	14390	17	1360	150	2742	48	4300	9	720	100	1470	32	2322
AR	2	2	9	8	17782	28	2240	324	4608	120	7292	10	800	154	1824	64	2842

にわける。C と NC のそれぞれに対して、start の小さい順に頂点の並び替えを行う。その後、C と NC の順で、頂点集合 V を更新する。次に、V が空になるまで、V の先頭から頂点 v を選択/削除し、以下の作業をおこなう。まず、以下の 2 つの条件を満たす頂点の集合 VC を探す。

- VC に含まれる頂点 vc は、v と順序付けされていない (vc と v の間に枝からなるパスは存在しない)
- VC に含まれる頂点 vc にたいして、v は代用条件を満たす

もし頂点集合 VC が存在するなら、頂点 v によって VC の各頂点 vc を代用し、SDFG 上の各頂点に対して開始/終了時間、slack を再計算する。

例: 図 8(b) の SDFG に対して、concurrent vertex substitution アルゴリズムを適用する。頂点集合 V は、V={$v1$,$v2$,$v3$,$v4$,$v5$} である。まず、V を C={$v1$,$v3$,$v5$} と NC={$v2$,$v4$} にわける。C と NC のそれぞれに対して、start の小さい順に頂点の並び替えを行ったあと、C と NC の順で V を更新する (V={$v1$,$v3$,$v5$,$v2$,$v4$})。次に、V が空になるまで頂点の代用を試みる。まず始めに、頂点 $v1$ が選択される (V={$v3$,$v5$,$v2$,$v4$})。$v1$ に対する頂点集合 VC は、VC={$v2$} である。したがって、$v1$ は $v2$ を代用する。同様に、$v3$ が選択されたとき、$v4$ が代用される。結果として、図 8(c) で表された SDFG が得られる。

図 8(b) の SDFG に対して concurrent vertex substitution アルゴリズムが適用した結果、図 10 のような回路構成がえられる。ここでは、3 つのセルコントローラだけで制御回路が構成される。結果として、アルゴリズムの適用前 (図 5(c)) と比べ、セルコントローラ 2 つ分の面積が削減されたことになる。

5. 実験結果

提案された合成手法を 5 つのベンチマーク回路 (FIR3, IIR2, FIR5, LMS4, and AR) に適用し、面積コストを評価した。

実験に先だって、データパス回路におけるリソースを Synopsis 社の Design Compiler [10] と NEC 社の CB-C10 ライブラリ [9] を用いて合成した。また、これらのリソースを利用し、データパス回路を合成した。表 1 の "DP" における列の集合は、データパス回路に割り当てられたリソースの数 ("+" - 加算器、"+" - 乗算器、"R" - レジスタ、"M" - マルチプレクサ) と面積 ("total" - グリッド数) を表す。

提案された合成手法によってえられた制御回路の面積を表 1 の "without sharing" に表す。ここで、列 "ver." は、SDFG における頂点の数を表し、列 "cell"、"glue"、"delay"、"timing" はそれぞれセルコントローラ、グルーロジック、遅延素子、タイミング調整で挿入された遅延素子それぞれにかかった面積をグリッド数で表したものである。列 "total" は、制御回路全体の面積を表す。結果から、制御回路の面積は SDFG における頂点の数に比例することがわかる。

Concurrent vertex substitution アルゴリズムを適用した結果を表 1 の "with sharing" に表す。各列は、"without sharing" と同じように、制御回路の各部にかかる面積と、制御回路全体にかかる面積を示している。Concurrent vertex substitution によって、たくさんの SDFG 上の頂点が削除された (列 "ver." を参照)。結果として、制御回路の面積も大幅に削減されたことがわかる。

6. まとめ

本稿では、セルコントローラに基づいた非同期制御回路の合成手法を提案した。提案された手法を用いることによって、たくさんの演算を実行するデータパス回路を制御する制御回路を、効率よく合成することが可能となった。

合成手法に加えて、制御回路の面積を最適化する手法を提案した。データパス回路の各演算にかかる実行時間を利用することによって、マップされるセルコントローラの数を抑えることができた。結果として、制御回路全体の面積も効率よく削減することができた。

今後の課題として、提案された手法をツール化する予定である。また、分岐やループといった制御を含む仕様に対しても、効率よく制御回路を合成できるよう提案手法を拡張する予定である。

文　献

[1] A. J. Martin. Synthesis of Asynchronous VLSI Circuits. In *Formal Methods for VLSI Design*, Chapter 6, pages 237–283, North-Holland, 1990.

[2] J. Cortadella, M. Kishinevsky, A.Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.

[3] K. Yun. Synthesis of Asynchronous Controllers for Heterogeneous Systems. *PhD thesis*, Stanford University, August 1994.

[4] D. C. Ku and G. De Micheli. High Level Synthesis of ASICs Under Timing and Synchronization Constraints *Kluwer Academic Publishers*, 1992.

[5] L. A. Hollaar. Direct Implementation of Asynchronous Control Units. *IEEE Transactions on Computers*, C-31, 12:1133-1141, Dec. 1982.

[6] C. P. Sotiriou. Implementing Asynchronous Circuits using a Conventional EDA Tool-Flow. In *Proc. of Design Automation Conference*, pages 415–418, Jun. 2002.

[7] B. M. Bachman, H. Zheng, and C. J. Myers. Architectural Synthesis of Timed Asynchronous Systems. In *Proc. of IEEE International Conference on Conputer Design*, Oct. 1999.

[8] T. A. Chu. Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications. *PhD thesis*, MIT Laboratory for Computer Science, June 1987.

[9] NEC Corporation. CB-C10 Family 0.25-μm CMOS Cell-Based IC (2.5V) Library. July 1997.

[10] Synopsis Co. http://www.synopsis.com