設 計 自 動 化 89－14
(1998. 9. 22)

DCVSLを使用した非同期式細粒度パイプライン・データパスの論理合成

今井 雅
miyabi@hal.rcast.u-tokyo.ac.jp

中村 宏
nakamura@hal.rcast.u-tokyo.ac.jp

南谷 崇
nanya@hal.rcast.u-tokyo.ac.jp

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4-6-1

あらまし DCVSL回路を使用した非同期式細粒度パイプライン・データパスとして、QDIモデルに基づいた回路構成が提案されている。本稿では、より現実的な遅延モデルであるSDIモデルに基づいた非同期式細粒度パイプライン・データパスを構成する手法を提案する。データパスの論理合成では、LUTベースの多段論理合成アルゴリズムを適用する。また、SDIモデルで規定する遅延変動率の比によって細粒度パイプラインの制御回路が異なることを示し、それぞれの遅延モデルの下で構成した回路の遅延及び回路量を比較した結果を示す。

キーワード 非同期式回路、DCVSL、SDIモデル、細粒度パイプライン・データパス

Logic synthesis of finely pipelined asynchronous datapaths using DCVSL cells

Masashi Imai
miyabi@hal.rcast.u-tokyo.ac.jp

Hiroshi Nakamura
nakamura@hal.rcast.u-tokyo.ac.jp

Takashi Nanya
nanya@hal.rcast.u-tokyo.ac.jp

Research Center for Advanced Science and Technology, University of Tokyo
4-6-1, Komaba, Meguro-ku, Tokyo, 153-8904, Japan

Abstract In this note, we present a synthesis method of finely pipelined asynchronous datapaths under the SDI(Scalable-Delay-Insensitive) model using DCVSL cells. For datapath synthesis we use a LUT-based logic synthesis algorithm. We also show that the synthesized control circuits for the pipeline differ according to the value of the relative variation ratio, which is defined in the SDI model. We evaluate the delay and size of synthesized circuits under different delay models.

key words asynchronous circuit, DCVSL, SDI model, finely pipelined asynchronous datapath

1 はじめに

VLSI技術の進歩により年々スイッチング素子の速度が向上している。システム全体へ位相差無く分配されたクロック信号により制御を行う従来の同期式ディジタルシステムでは、配線遅延の相対的な増加によりこのような高速素子の性能を充分に活用できないことが指摘されている [1]。また、VLSI 技術の微細化による配線抵抗の増大により配線遅延は絶対的にも増加しており、位相差のない同一クロックを分配することをますます困難にしている。さらに、同期式システムではある時点で使用されていない回路にもクロック信号が供給される。そのため、無駄な電力が消費されることになり、システム全体の消費電力を本来必要な量だけに押さえることは困難であると言える。

高速素子の性能を有効に活用し、消費電力を低減するための方法の一つに、システムを非同期式で構成する方法がある。非同期式回路は高速性、低消費電力、高い拡張性などの多くの利点を持っており、近年様々な研究がなされている。

我々は実用的レベルの非同期式プロセッサの実現を目標として 32ビットの非同期式プロセッサ TITAC-2[2] を 0.5μm ルールのプロセスで設計・試作し、Dhrystone ベンチマークで 54MIPS の性能を実現した。

非同期式システムの設計を行う際には、ゲートや配線の遅延にどのような遅延があるか仮定して設計を行う。Delay-Insensitive (DI) モデル [3] や Quasi-Delay-Insensitive (QDI) モデル [4] と呼ばれる遅延モデルは遅延変動に対して悲観的なモデルであるため、高いパフォーマンスを得ることが困難であった。そこで、TITAC-2 では遅延に関する仮定として、より現実的な仮定である Scalable-Delay-Insensitive(SDI) モデル [2] を導入し、設計を行った。しかし、SDI モデルの下での組織的な回路設計方法については未だ研究段階である。

本稿では、SDI モデルの下で DCVSL (Differential-Cascode-Voltage-Switch-Logic)[5] を使用した非同期式細粒度パイプライン・データパスを構成する手法を提案する。データパス部の論理合成には、N 入力までの任意の論理関数を実現することが出来る N-LUT(LookUp-Table) ベースの論理合成アルゴリズムを適用する手法を提案する。また、SDI モデルで規定する遅延変動率の比に基づいたパイプライン制御回路の構成方法を示す。そして本稿で提案する論理合成手法をいくつかの遅延モデルの下でベンチマーク回路に適用した例を示し、評価を行う。

2 用語の定義

2.1 遅延モデル

論理ゲートや配線における入出力間遅延に関する仮定を遅延モデルという。遅延モデルには、遅延の大きさは有限であるが上限値は未知であるとする DI モデルや、DI モデルに等時分岐の仮定を加えた QDI モデルなどがある。これらのモデルの下では遅延の上限値を未知と仮定しているため、回路の安定を監視するために多くの回路を必要とする。

本稿では、任意の二つの遅延変動率 (遅延の推定値が $d_{estimate}$、実際の遅延が d_{actual} であるとき、$v = d_{actual}/d_{estimate}$ を遅延変動率という) の比 R には上限値 (定数)K が存在し、$1/K < R < K$ が常に成り立つとする SDI モデル [2] の下で回路設計を行う。

SDI モデルを回路に適用するときは、図 1において、t_1、t_2 の共通原因となる分岐 t からの各経路の遅延をそれぞれ d_1、d_2 としたとき、$K \cdot d_1 < d_2$ が成り立つように論理設計及びレイアウトを行う。

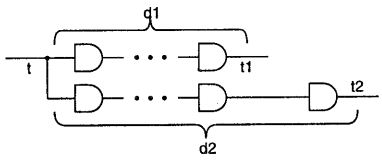

図 1: SDI モデルの下での回路実現

2.2 2線2相式

1ビットの情報に対し、肯定線と否定線の2本の信号線を使用し、$(x,\overline{x})$=(1,0) で論理 "1" を、$(x,\overline{x})$=(0,1) で論理 "0" を表し、これらのいずれかとスペーサと呼ばれる $(x,\overline{x})$=(0,0) を交互に送ることでデータ転送を行う方式を2線2相式と呼ぶ。

2.3 C素子

Muller の C 素子は非同期式回路特有の素子である。全ての入力が "1" になると "1" を出力し、全ての入力が "0" になると "0" を出力し、入力に "0" と "1" が混在している場合には前の出力値を保持する記憶を持った双安定素子である。

2.4 DCVSL回路

DCVSL は CMOS による2線式論理設計の一手法であり、以下の特徴がある。

- 相補的な出力 $(F,\overline{F})$ を持つため、非反転論理のみ実現可能なドミノ論理と比べて回路の自由度が高い。

- 関数実現部がNMOSのみにより構成されているため、入力ゲートキャパシタンスが小さい。
- 2線式論理であり、配線本数が多く複雑となる。

図2にDCVSLの基本構成を示す。

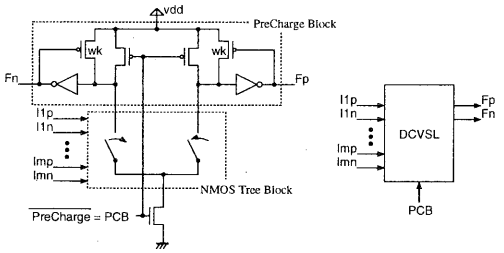

図 2: DCVSLの基本構成

この回路の基本的な動作は以下の通りである。

初期状態 $\overline{PreCharge}$= 0であり、回路がプリチャージされて出力$(F,\overline{F})$=(0,0)となる。

関数評価時 $\overline{PreCharge}$=1となり、NMOSツリーへの入力2線対$(i_1,\overline{i_1}),\cdots,(i_n,\overline{i_n})$に符号語が揃うとNMOSツリー内で$F$側あるいは$\overline{F}$側のどちらかのパスが構成され、出力が(1,0)または(0,1)となる。

3 細粒度パイプライン・データパス

非同期式回路は、ある事象が終了したことを確認して次の事象を開始する事象駆動型論理である。そのため、パイプラインステージや機能ユニットでは、それぞれ処理が完了したことを示す完了信号を生成しなければならない。QDIモデルの下では、完了信号生成のために出力データのビット数Nに対して$\log N$に比例した時間が必要となる。従って、システムの性能上クリティカルなサイクルに完了信号生成のオーバヘッドがかかると全体の性能を制限することになる。

この問題を解決するための一つの方法はデータパスをパイプライン化することである。データパスをパイプライン化することにより、有効なデータの処理と完了信号生成を別のパイプラインステージで同時に実行することができる。さらに、パイプラインの粒度を細かくして有効なデータの処理にかかる遅延を短くすることにより、短いサイクルタイムを実現することができる。これが細粒度パイプライン・データパスである。

図3にQDIモデルに基づくDCVSLを使用した細粒度パイプライン・データパスの基本構成を示す[6]。

この回路では、入力信号線及び出力信号線のデータの有効性を判別するために、全ての2線対毎にOR(NOR)ゲートを使用し、パイプラインの段毎にそれらをまとめて制御信号を生成している。そのため、信号線数が増加すると制御信号の生成に必要な時間が増加する。

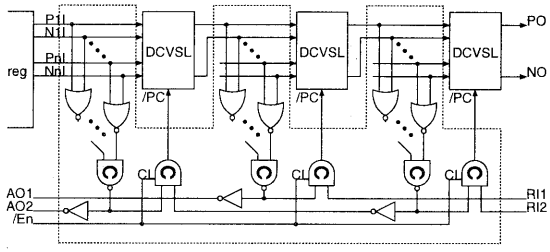

図 3: 細粒度パイプライン・データパスの基本構成

そこで、本稿では図3に示されるような非同期式細粒度パイプライン・データパスをSDIモデルの下で設計し、回路量の削減、高速化を図る。

4 パイプライン制御回路の構成法

QDIモデルに基づくパイプライン・データパスの制御回路は図3において破線で囲った部分、すなわちDCVSL回路以外全てである。この回路はパイプラインの各段毎に、信号線対のデータの有効性を示すために全ての信号線対のOR(NOR)をとり、それらをまとめている。QDIモデルの下ではこのように全ての信号線対の有効性を確認する必要がある。一方、SDIモデルの下では、図1に示されるような信号遷移の共通原因となるものがあり、遅延変動率の比Rが$1/K < R < K$を満たすように構成することが出来れば、必ずしも全ての信号線対のORを取らなくてもそのデータの有効性を保証することが出来る。

図3に示された制御回路において、共通原因を求めると図4に示す共通原因を得ることが出来る。従って、制御信号生成パスの遅延(図4. d_2)とデータパスの遅延(図4. $d_{1p}, d_{1n}, \cdots, d_{np}, d_{nn}$)を比較し、$K\cdot d_{1p} < d_2$, $K\cdot d_{1n} < d_2$, $\cdots$, $K\cdot d_{np} < d_2$, $K\cdot d_{nn} < d_2$を満たすように回路を構成することで遅延変動率の比KのSDIモデルの下での制御回路を実現することが出来る。

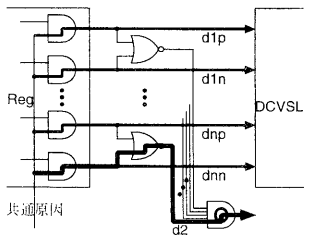

図 4: 制御回路の構成

遅延変動率の比Kが2及び3のSDIモデルの下で構成した制御回路を図5に示す。図5では、ゲート遅延を全て

ユニットディレイとし、配線遅延を0としたときの各パスの遅延を記述している。このとき図5(1)は遅いパスの遅延が2で早いパスの遅延が1であり、$K=2$のSDIモデルの下で構成した回路であると言える。同様に図5(2)は$K=3$のSDIモデルの下で構成した回路である。

このパイプライン制御回路において$K=L$のSDIモデルを構成するときは、2^{L-2}組の2線対のORを取り、それらを2入力C素子でツリー状にまとめることで実現できる。

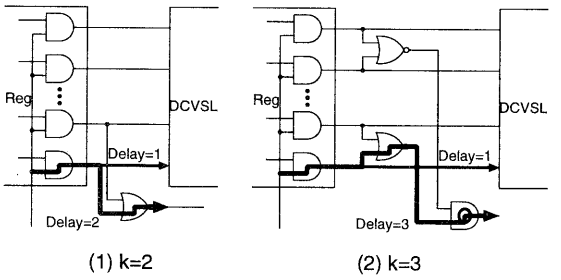

図 5: Kの値による制御回路の変化

図5(1)で示された制御回路をパイプライン・データパスに適用すると図6を得ることが出来る。この回路を採用することにより、制御回路によるオーバヘッドをQDIモデルの下で設計されたものと比較して明らかに小さくすることが出来る。

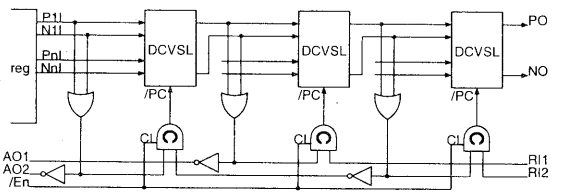

図 6: SDIモデルの下で構成したパイプライン

図6ではDCVSL回路1段をパイプライン1段として回路を構成しているが、DCVSL回路$m(m>1)$段毎にパイプライン1段を構成することも出来る。DCVSL回路2段毎にパイプライン1段を構成した場合の例を図7に示す。DCVSL回路$m(m>1)$段毎にパイプラインを構成することにより、DCVSL回路1段毎に構成したときと比べて制御回路の量を減少させることができる。また、次々段以降へ信号を転送するためのDCVSL回路(これについては後述する)の量を減少させることも出来るため、DCVSL回路を1段毎ではなく多段毎にパイプライン化した方が回路量の面で良い。しかし、DCVSL回路m段を通過するためのレイテンシは比例増加するため、サイクルタイムは増加することになる。従って、DCVSL回路を使用したパイプライン・データパスを作成するときには、実現しなければならないサイクルタイムを決定した後、パイプラインの各段に含まれるDCVSL回路の段数としてそれを実現できる最大値を取れば、最大の性能を得ることが出来る。

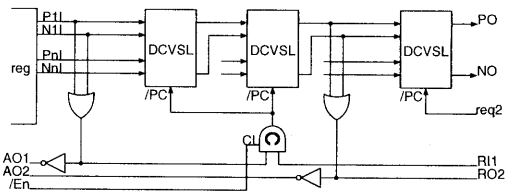

図 7: SDIモデルで構成したパイプライン (DCVSL2段)

5 パイプライン・データパスの構成法

5.1 DCVSLを使用したデータパスの構成

CMOSによる2線式論理設計としては他にパストランジスタによる構成が考えられる。しかし、パストランジスタ論理では各段毎に値を保持するとラッチが必要となり、回路量が増加する。一方、DVSL回路では図2に示すPreCharge Blockで値を保持する機能を持っているためラッチが不要であり、少ない回路量でパイプラインを実現することが出来る。

DCVSLでは、NMOSを使用したツリー構造により任意の関数が実現できる。一般にNMOSを直列接続すると立ち下がり遅延が比例増加する。また、電荷が抜けきらないことで出力が変化しないこともある。そのため、このNMOSツリーの実現においては、NMOSの直列接続段数に4段あるいは5段の制限がかけられる。

一方、DCVSLは2線式論理回路であり、ある信号に対して肯定線と否定線の両方の信号線が入力として必要となる。そのため、入力信号のNOTは肯定線と否定線を交換することで得ることが出来る。また、論理関数自体のNOTも出力の肯定線と否定線を交換することで得ることが出来る。従って、ある論理関数を実現したDCVSL回路は同族変換(2値変数のNOT、2値変数の置換、論理関数のNOT)を有限回行うことによって得られる同族な論理関数も実現していると言える。

互いに同族な論理関数を集めて同族類とした時、2入力の場合2種類、3入力の場合10種類、4入力の場合224種類の同族類しか存在しない。すなわち、これら全ての論理関数のNMOSツリーを用意すれば、$N(N=2\sim4)$入力の任意の論理関数をDCVSLで実現することが出来る。このような同族類ライブラリを用いた論理合成は、既にFPGAのための論理合成で用いられている。

FPGAの中でも、LUT(Look-Up-Table)ベースのFPGAはN入力までの任意のBoolean Functionを実現できるLook-Up Table(N-LUTと記述)を使用したものであ

る。このLook-Up Tableを同族類と対応づけることにより、LUTベースの論理合成アルゴリズムをDCVSLを使用したデータパスの論理合成に利用することが出来る。

N–LUTベースのFPGAの論理合成を行うアルゴリズムとしては様々なものが提案されている。本稿では、最大段数を小さくし、回路量を少なくすることを目的としたアルゴリズムであるDAG–Map[7]アルゴリズムに改良を加えてパイプライン・データパスを構成する。

5.2 データパス生成アルゴリズム

DCVSLを使用した細粒度パイプライン・データパスの論理合成を行うアルゴリズムを以下に示す。アルゴリズムは基本的に3つのフェーズから成り立っている。

初めに、与えられた論理式あるいは回路を入力数が2までの基本ゲート(INV, AND, OR, NAND, NOR, XOR, XNOR)により構成する。これは多入力の素子を排除し、マッピングアルゴリズムを適用させるための前処理を行うものである。

次に、初めのフェーズで得られた2入力までのゲートによる回路に対して、全てのゲートのラベル付けを入力側から行う。得られたラベルは入力からの通過ゲート段数を表している。すなわち、DCVSL回路1段毎に細粒度パイプライン化した場合、そのラベルが何段目のパイプラインになるかを表すことになる。

最後に前のフェーズで付けられたラベルを元に、出力側からN–LUTを適用して回路を構成する。この際、得られたN–LUTをDCVSLに変換するとともにパイプライン制御回路を付加する。DCVSL回路1段毎にパイプラインを構成する場合、各ラベル毎にまとめられたものそれぞれにパイプライン制御回路を付加する。また、DCVSL回路2段毎にパイプライン化するときはこのラベルを元に(0,1),(2,3),(4,5),$\cdots$を組にしてパイプラインを構成することになる。同様にDCVSL回路m段毎にパイプライン1段を構成する場合、$(0,1,\cdots,m\text{–}1),(m,m\text{+}1,\cdots,2m\text{–}1),\cdots$を組にしてパイプラインを構成し、それぞれ制御回路を付加する。制御回路を付加する際には、SDIモデルで規定するKの値に従って監視する2線対の数を決定する。

また、図8に示すように、DCVSLの出力が次々段以降のパイプラインで使用されていた場合、次段にそのDCVSLの出力を伝播するためのDCVSL回路を挿入する。

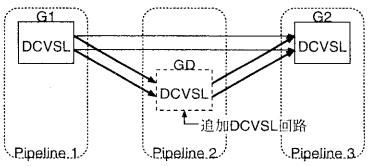

図 8: 信号転送用DCVSL回路を次段に挿入

アルゴリズムの詳細を以下に示す。

1. 2入力までの基本ゲートによる回路に変換する。
   - DMIG(Decompose–Multi-Input–Gate)アルゴリズム[7]を適用し、論理関数を2入力までの基本ゲート(INV, AND, OR, NAND, NOR, X-OR, XNOR)を用いて実現する。
2. 入力からトポロジカルな順でゲートにラベル付けを行う。
   - Primary Input(PI)ノードのラベルを0とする。
   - 注目するゲート(v)の入力の最大ラベルをpとする。このとき、$N_p(v)$をラベルpを持ったゲートのセットとする。
   - $input(N_p(v) \cup \{v\}) \leq N$となる場合$v$のラベルを$p$とし、それ以外のときは$p+1$とする。
   - 各入力に対し、その入力が最後に使用されたものが現在注目しているノードの場合はそのラベルを入力ノードのLastUsedLabel(v)として記憶する。
   - Primary Output(PO)ノードになるまで繰り返す。
3. 出力から順にN-LUTを適用する。
   - 出力から順に、関数を実現するように同じラベルを持ったものをまとめて一つのN-LUTを作成し、DCVSLに変換する。
   - SDIモデルで規定する遅延変動率の比Kの値に従ったパイプライン制御回路を付加する。$K-2$組の2線対のORを取り、それらを2入力C素子でまとめる。
   - DCVSL回路1段毎にパイプライン1段を構成する場合はラベル毎、DCVSL回路m段毎にパイプライン1段を構成する場合は$(0,1,\cdots,m-1),(m,m+1,\cdots,2m-1),\cdots$を組としたパイプラインを構成する。
   - DCVSL回路m段毎にパイプライン化する場合、LastUsedLabel(v) > Label(v) + mのとき、次段のパイプラインに信号伝搬用のDCVSL回路を加える。
   - ゲートの入力が全てPIノードのみになったとき終了。

表 1: 論理合成結果(3入力までの任意の論理関数を実現できるライブラリ使用)

		DCVSL1段毎		DCVSL2段毎	
Benchmark	最大段数	セル数	Cycle Time(ns)	セル数	Cycle Time(ns)
5xp1	6	81	3.51	69	4.91
9sym	7	48	3.53	38	4.80
alu4	16	635	3.60	599	5.22
apex1	13	1042	3.62	978	5.58
apex2	15	243	3.62	219	5.35
bw	6	129	3.55	113	4.97
clip	9	91	3.50	78	4.92
con1	4	23	3.48	16	4.87
duke2	9	267	3.61	238	5.34
e64	32	143	3.57	111	5.32
misex1	5	54	3.55	43	4.93
misex2	7	87	3.56	72	4.73
rd53	5	27	3.49	20	4.61
rd73	5	34	3.51	27	4.56
rd84	7	45	3.52	36	4.79
sao2	9	120	3.59	106	5.20
vg2	9	76	3.55	64	5.08

表 2: 論理合成結果(4入力までの任意の論理関数を実現できるライブラリ使用)

		DCVSL1段毎		DCVSL2段毎	
Benchmark	最大段数	セル数	Cycle Time(ns)	セル数	Cycle Time(ns)
5xp1	4	58	3.58	46	4.98
9sym	6	35	3.54	27	4.82
alu4	13	456	3.72	408	5.36
apex1	9	730	3.78	687	5.80
apex2	9	177	3.66	155	5.46
bw	3	97	3.60	87	5.01
clip	6	69	3.58	55	5.03
con1	3	18	3.49	13	4.89
duke2	6	213	3.72	180	5.42
e64	17	80	3.64	59	5.36
misex1	3	41	3.55	30	4.95
misex2	3	56	3.58	42	4.78
rd53	4	20	3.52	15	4.66
rd73	4	25	3.52	20	4.56
rd84	5	35	3.54	27	4.82
sao2	5	87	3.61	74	5.28
vg2	7	66	3.58	52	5.16

6 論理合成結果とその評価

論理合成用ベンチマーク回路 LGSynth89[7][8] に前述のアルゴリズムを適用してDCVSLによるパイプライン・データパスを構成した結果を表1、表2、表3 に示す。

各表において、回路の遅延を求めるときはゲート遅延としてNEC CBC10ライブラリの遅延を使用している。平均サイクルタイムは、パイプラインへのデータ供給及びパイプラインから出力されたデータの消費がそれぞれ1(ns)で処理されるものとし、2^{16}回のランダムな入力信号パターンを入力したときの平均サイクルタイムである。

また、制御信号の分配にはファンアウト数16の制限を設けている。16以上のDCVSL回路に同一の制御信号を分配するときはバッファを通し、全ての回路がフラットに信号が伝達されるようにしている。

表1は3入力までの任意の論理関数を実現できるDCVSLライブラリを使用して$K=2$のSDIモデルのもとで回路を構成した時の値である。DCVSL回路の最大段数、及び1段毎に細粒度パイプライン化、2段毎に細粒度パイプライン化したときそれぞれの回路量と平均サイクルタイムを示している。表2は同様に4入力までの任意の論理関数を実現できるDCVSL ライブラリを使用して論理合成を行った時の結果である。

表1、表2より、DCVSL回路1段毎にパイプラインを構成した場合はDCVSL回路2段毎にパイプラインを構成した時よりも回路量が増加していることが確認できる。また、各パイプラインに含まれるDCVSL の段数を1段から

表 3: 論理合成結果(3入力ライブラリ使用、DCVSL回路1段毎のパイプライン)

		SDI ($K=2$)		SDI ($K=3$)		QDI	
Benchmark	最大段数	セル数	Cycle Time(ns)	セル数	Cycle Time(ns)	セル数	Cycle Time(ns)
5xp1	6	81	3.51	95	3.94	113	4.59
9sym	7	48	3.53	62	3.97	74	4.37
alu4	16	635	3.60	683	4.06	788	4.98
apex1	13	1042	3.62	1130	4.32	1785	5.28
apex2	15	243	3.62	285	4.22	400	4.82
bw	6	129	3.55	145	4.12	188	4.58
clip	9	91	3.50	110	4.08	137	4.39
con1	4	23	3.48	31	3.98	35	4.23
duke2	9	267	3.61	287	4.08	430	4.62
e64	32	143	3.57	210	4.10	228	4.52
misex1	5	54	3.55	64	3.98	72	4.14
misex2	7	87	3.56	102	4.04	123	4.48
rd53	5	27	3.49	37	3.89	44	4.03
rd73	5	34	3.51	44	3.98	48	4.22
rd84	7	45	3.52	62	4.00	68	4.21
sao2	9	120	3.59	140	4.08	183	4.62
vg2	9	76	3.55	98	3.98	110	4.32

2段にしても、サイクルタイムは2倍にはならず、1.5倍程度で押さえられることがわかる。これは、DCVSLの段数が増えると有効なデータを出力するまでの遅延は比例増加するが、プリチャージ(初期化)は一斉に行われるためであると考えられる。

また、3入力の任意関数を実現できるライブラリを使用したときの方が4入力の任意関数ライブラリを使用したときよりもほとんどサイクルタイムが短くなっている。これは4段のNMOSツリーよりも3段のNMOSツリーで構成されたDCVSL回路の方が高速であることと、制御信号を各DCVSL回路に分配するときにバッファを必要とする数が3入力任意関数ライブラリの方が少ないためと考えられる。

表3は3入力までの任意関数を実現できるライブラリが利用可能であるとして、遅延モデルとしてQDIモデルの下で構成した場合、SDIモデル($K=2$)の下で構成した場合、SDIモデル($K=3$)の下で構成した場合それぞれのセル数とサイクルタイムを求めたものである。また、各パイプラインに含まれるDCVSLの段数は1段として構成している。

遅延モデルの違いによる回路特性を求めた表3より、SDIモデルの下で構成したときには、サイクルタイムがQDIモデルの下で構成したときの約70%~95%となることが確認できる。また、SDIモデルの下で構成した方が大幅に回路量を削減できることが確認できる。

7 まとめ

SDIモデルの下で、DCVSL回路を利用した非同期式細粒度パイプライン・データパスの論理合成を行う手法を提案した。また、論理合成用ベンチマーク回路にアルゴリズムを適用して細粒度パイプライン・データパスを構成し、評価を行った。

SDIモデルに基づいたパイプライン・データパスの制御回路を示し、Kの値によって異なる制御回路とその構成方法を示した。また、QDIモデルの下で構成した場合との比較を行い、SDIモデルの下で構成した方がセル数が少なく、サイクルタイムが大幅に小さくなることを確認した。

このアルゴリズムを現在開発中の高速版非同期式プロセッサTITAC-3の設計に適用する。

謝辞

本研究の一部は文部省科学研究費補助金基盤研究(B)(2)09480049、及び(株)半導体理工学研究センターとの共同研究によるものである。

参考文献

[1] 南谷崇. 非同期式マイクロプロセッサの動向. 情報処理, Vol. 39, No. 3, pp. 181–186, March 1998.

[2] Akihiro Takamura, Masashi Kuwako, Masashi Ima, Taro Fujii, Motokazu Ozawa, Izumi Fukasaku, Yoichiro Ueno, and Takashi Nanya. TITAC-2: An asynchronous 32-bit microprocessor based on scalable-delay-insensitive model. In *Proc. International Conf. Computer Design (ICCD)*, pp. 288–294, October 1997.

[3] Jan Tijmen Udding. A formal model for defining and classifying delay-insensitive circuits. *Distributed*

Computing, Vol. Vol.1, pp. 197–204, No.4 1986.

[4] Alain J. Martin. Synthesis of asynchronous VLSI circuits. In J. Staunstrup, editor, *FORMAL METHODS FOR VLSI DESIGN*, chapter 6, pp. 237–283. Elsevier Science Publishers B. V., 1990.

[5] William R. Griffin Lawrence G. Heller. Cascode voltage switch logic: A differential cmos logic family. *IEEE International Solid-State Circuits Conference*, Vol. Digest of Technical Papers, pp. 16–17, February 1984.

[6] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nystroem, Paul Penzes, Robert Southworth, and Uri Cummings. The design of an asynchronous MIPS R3000 microprocessor. In *Advanced Research in VLSI*, pp. 164–181, September 1997.

[7] Kuang-Chien Chen, Jason Cong, Yuzheng Ding, Andrew B. Kahng, and Peter Trajmar. DAG-map: Graph-based fpga technology mapping for delay optimization. *IEEE Design and Test of Computers*, pp. 7–20, Sep. 1992.

[8] ftp://ftp.mcnc.org/pub/benchmark/