

DDL 素子を用いた非同期式細粒度パイプライン回路の 論理合成用ライブラリ

今井 雅^{1,a)}

概要：本稿では，DDL (Differential Domino Logic) 素子で構成される非同期式細粒度パイプラインデータバス回路を，ツールにより論理合成するためのセルライブラリを提案する．データが2線式符号化され，1ビット毎にタイミング情報を持つ DDL 素子はタイミングマージンが不要なため，ランダム遅延変動が大きくなる微細プロセスでも高性能なシステムを実現し得る．また，DDL セルは同族な関数も全て一つのセルで表すことが出来るため，少ない素子数で大規模な回路を効率的に実現することが出来る．本稿では，3入力までの同族類セルライブラリ及び4入力の一部の関数を加えたセルライブラリを設計し，Synopsys 及び Cadence の論理合成ツールを用いてその有効性を評価した結果を示す．

キーワード：非同期式細粒度パイプライン, DDL 回路, 論理合成

Synthesis Library for Asynchronous Fine-grain Pipeline Circuits using DDL Elements

IMAI MASASHI^{1,a)}

Abstract: This paper presents six DDL cell libraries for technology mapping tools in order to design asynchronous fine-grain pipeline circuits. DDL cells can tolerate any delay variations thanks to the dual-rail encoding and the 4-phase handshaking protocols. Thus, it can achieve high-performance circuits even in the deep-submicron processes in which random delay variations are dominant. The proposed DDL cell libraries only contains 13 physical DDL cells. However, they can represent all the family functions by themselves without redundant gates. In this paper, some evaluation results are shown using the Nangate 45nm process technology and the Synopsys Design Vision and the Cadence RTL Compiler.

Keywords: Asynchronous fine-grain pipeline, DDL circuits, Logic Synthesis

1. はじめに

半導体製造技術の微細化が進むに従い，スイッチング素子や配線など VLSI を構成する要素の微細化・信号遷移速度の高速化が進んでおり，VLSI 一チップに搭載されるトランジスタ数はもはや 40 億以上にもなっている [1, 2]．微細化による配線間隔の減少は隣接配線間容量の増加を招き，クロストークと呼ばれる隣接配線の動的な信号遷移に依存する遅延変動を引き起こしやすくなる．従来は配線が並行して配置される長距離配線でこの影響が大きかったのに対し，微細化が進んで隣接配線間容量が増加すると，組

み合わせ回路内などの近傍領域の配線に関してもクロストークの影響を考慮しなければならなくなってくる．また，DVFS (Dynamic Voltage and Frequency Scaling) と呼ばれる動的電圧・周波数制御や，バッテリーの放電，IR-Drop や Ground bounce 等による電圧幅の変動，電源ノイズなど，様々な要因で供給電圧が変動することにより引き起こされる遅延変動が大きな問題となっている．さらに，HCI (Hot Carrier Injection) や NBTI (Negative Bias Temperature Instability) など，経年劣化による遅延変動も問題となっている．上記のような様々な要因により，構成要素のスイッチングにかかる遅延は大きく変動する．大きな遅延変動に対する回路レベルの対処方式としては，上述の DVFS による動的な電圧・周波数調整が広く用いられている．しかし

¹ 弘前大学 / Hirosaki University

^{a)} miyabi@eit.hirosaki-u.ac.jp

ながら、動的に周波数や電圧を切り替える同期式実装では、高周波数クロック信号自体の消費電力や、タイミング切り替えに伴うオーバーヘッドなどが問題となる。一方、上記以外の回路レベルの対処方式として、クロック信号を用いず、要求-応答ハンドシェイクプロトコルに基づいて動作させる非同期式回路（因果式回路）による実装がある。非同期式回路は事象生起の因果関係に基づき、必要な箇所が必要な時にのみ動作するため、無駄な電力消費が無く、個々の論理ブロックがそれぞれの動作速度で動作するためタイミング切り替えがシームレスに行われ、切り替えに伴うオーバーヘッドを削減することが出来る。その他にも、クロック信号に同期した値の書き換えにより一定周期でピーク電流が流れ、ノイズ源になってしまう同期式回路に対し、非同期式回路ではピーク電流が分散化されることにより、ノイズ発生を小さく抑えることが出来る。このような様々な利点がある非同期式回路ではあるが、非同期式回路設計用セルライブラリや設計支援環境の整備が不十分である。本稿では、非同期式回路設計支援環境の充実を図るため、ダイナミック論理回路の一種である DDL (Differential Domino Logic) 素子を用いた非同期式回路用のセルライブラリ開発とそのライブラリを用いた論理合成手法に関して述べる。

本稿の構成は以下の通りである。次節では非同期式細粒度パイプライン回路について述べる。第 3 節では DDL 回路を用いた同族類セルライブラリ設計について述べる。第 4 節では、Nangate 45nm テクノロジを用いて評価した結果を示す。第 5 節でまとめを述べる。

2. QDI モデルに基づいた非同期式細粒度パイプライン

非同期式回路の設計では、回路要素の遅延に関して設ける仮定、すなわち遅延モデルが重要な役割を果たすため、仕様に応じて適切な遅延モデルを選択することが重要な設計要件となる。遅延変動は近傍の回路要素が同じように変動するシステムティック成分と、隣り合った回路要素でさえも異なる特性を示すランダム成分に分けることができ、今後の微細プロセスでは後者のランダム成分が支配的になると予想される。そこで、本稿では、ランダム成分が支配的になり、個々の要素の遅延がばらばらに変動したとしても正しく動作する回路を実現するため、QDI モデル [3] に基づいた回路設計を行う。QDI モデルは、遅延の上限値は有限であるが未知と仮定し、配線の分岐先の信号到達遅延に差はないという等時分岐の仮定を加えたものであり、これまで様々な研究が行われてきた [4-9]。QDI モデルに基づく回路では、1 線式の組み合わせ回路に対してそのクリティカルパスよりも大きな遅延値を持つストロブ信号を用いる束データ方式 [10] を取ることは出来ず、(複数)ビット毎にタイミング情報を持たせる符号化方式を取る必要がある。典型的な符号化方式としては、1 ビットの論理値に対して 2 本の信号線を用いる 2 線式符号と初期状態（休止

状態）を表すスペーサ (0,0) を交互に転送する 2 線 2 相式がある。しかしながら、通常のスタンダードセルを用いて 2 線 2 相式回路を実現すると回路規模が大きくなり、消費電力や面積のオーバーヘッドが大きくなりやすい。そこで、トランジスタレベルの実現方式として、CMOS ダイナミック論理回路の一種である DDL (Differential Domino Logic) 素子を用いた設計が提案されている [4, 8, 9]。

DDL 素子 [11] の基本構成は図 1 に示す通りである。信号線に $_p$ の付加されたものが肯定線、 $_n$ の付加されたものが否定線を表す。初期状態では要求信号 req は 0 であ

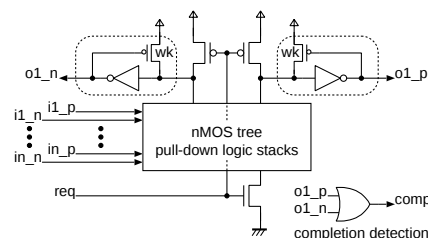


図 1 DDL セルの基本構成

Fig. 1 Basic structure of DDL cell.

り、回路のプリチャージが行われて出力 $o1_p$, $o1_n$ がともに 0、すなわちスペーサ ($o1_p, o1_n$) = (0,0) となる。要求信号 req が 1 になり、nMOS ツリーに対する入力 2 線対 ($i1_p, i1_n$), ..., (in_p, in_n) のうち、論理演算に必要なデータが到着すると、nMOS ツリー内で論理の評価が行われて GND からいずれか一方の出力までのパスが構成される。その結果、出力が符号語 (0,1), (1,0) のいずれかとなる。出力がスペーサか符号語であるかは、図 1 右下に “completion detection” として示す OR ゲートにより判定することが出来る。この出力信号が使用される演算において、信号が符号語になっていなくても演算結果が出力される場合、QDI モデルに基づいた回路実現であることを保証するためには、OR ゲートの出力が 1 になったことを確認する必要がある。一方、XOR 演算のように全ての入力値が揃わなければ出力が確定しない演算では、OR ゲートの出力を用いる必要は無い。このように出力側で符号語が評価されると、ハンドシェイクプロトコルに基づいて要求信号 req が再び 0 となり、回路の初期化が行われて次の入力待つ状態となる。これら一連の動作を繰り返すことでデータ転送が行われる。

DDL 素子はその出力部にラッチ構造（図 1 破線で囲った構造）を持つため、ビット毎に記憶を持つ素子として扱うことが出来る。そのため、この記憶部を用いると、入出力の因果関係に基づいてビット毎に細粒度化されたパイプライン構造を取ることが出来る。図 2 上部は DDL 素子を用いて設計された組み合わせ回路全体を、一つの要求信号 (DDL 素子へのプリチャージ信号) により制御する構成である。これをビット毎に細粒度パイプライン化すると、図 2 下部の様な構成になる。図 2 において、“C” が付加された AND ゲートは Muller の C 素子と呼ばれる待ち合わ

せ素子であり、全ての入力が 0 (1) に揃うと出力が 0 (1) になる記憶素子である．図 2 から明らかなように、ビット

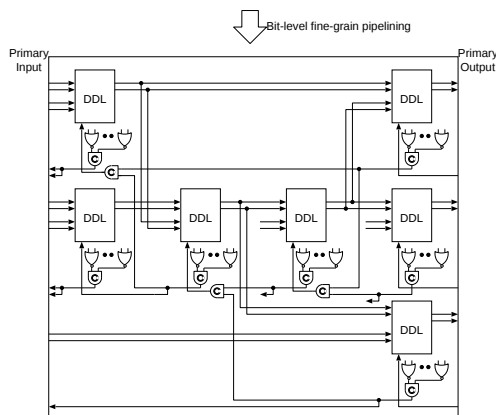
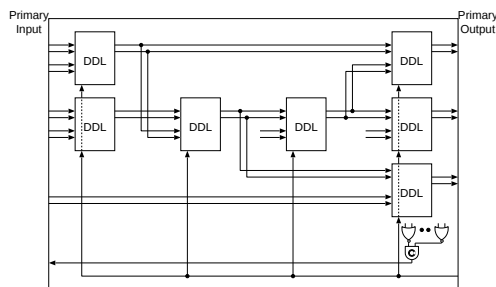


図 2 ビット毎の細粒度化
Fig. 2 Bit-level fine-grain pipelining.

ト毎の細粒度化では、各 DDL 素子に制御回路が付加された構成となるため、回路規模が非常に大きくなる．また、配線に分岐がある場合、分岐した中で最後段の DDL 素子の動作を待ってから次の動作が行われるため、回路構成によっては細粒度化する効果がない場合がある．

システムに対する要求仕様として、高スループットが求められる場合、ビット毎の細粒度パイプライン化ではなく、ステージ数を揃えた形態での細粒度化を行うことが考えられる．図 3 に、図 2 上部の回路構成に対して、ステージ数を揃えて細粒度パイプライン化を行った例を示す．図において、“Buffer DDL” は DDL 素子の出力が使用されている DDL のステージ数がその DDL 素子の属するステージよりも 2 つ以上離れている時に挿入する、バッファ機能を持った DDL 素子である．スループットを上げるためには、プライマリ入力、プライマリ出力に関してもバッファ DDL 素子を挿入する必要がある．図 3 の構成は、1 ステージあたりの DDL 段数が 1 の場合、すなわち、スループットが最も高くなる構成を示している．一方、ステージあたりの DDL 段数を増やすとスループットは低下するが、挿入される Buffer DDL の数及び制御回路の数を削減することが出来るため、面積を小さくすることが出来る．従って、要求仕様を満たす中で 1 ステージの DDL 段数を多くした方がよいと言える．本稿では、複数のセルライブラリを用いた論理合成結果に対して、図 3 に示す細粒度化を行い、面積オーバーヘッドを評価した結果を示す．

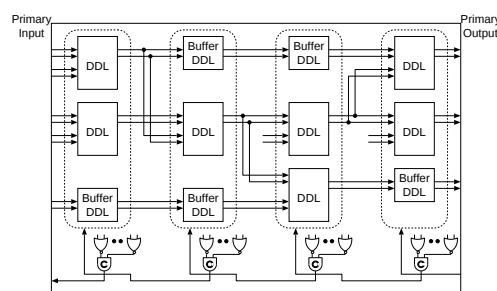


図 3 細粒度パイプライン化
Fig. 3 Fine-grain pipelining.

3. 同族類 DDL セルライブラリ

3.1 同族類関数

2 線式符号において、反転論理 (INV) はゲートを用いず配線の入れ換えだけで実現することができるため、ある論理を表す DDL 素子は、以下の 3 つから成る同族変換により得られる論理も論理ゲートを追加すること無く表すことが出来る [9] ．

- 入力信号の反転 (2 値変換の NOT)
- 入力信号の入れ換え (2 値変数の置換)
- 出力信号の反転 (論理関数の NOT)

図 4 は、2 入力演算 ($Y = A * B$) を実現する DDL 素子の回路例と同族変換により得られる関数のバリエーションを全て表したものである．図 4 に示すように、配線とセルを

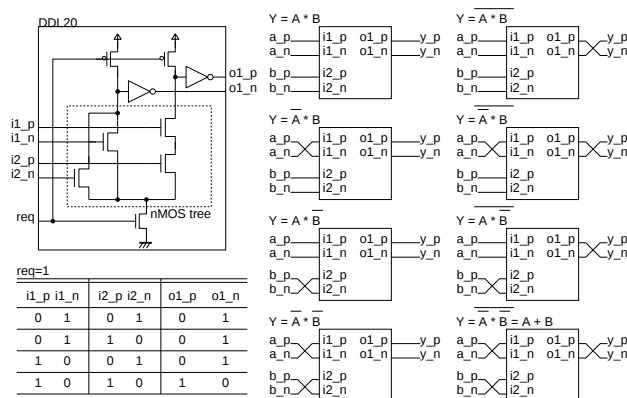


図 4 2 入力 DDL 回路 ($Y = A * B$) の論理バリエーション
Fig. 4 Logic variations of a 2-input DDL cell ($Y = A * B$).

接続する際に入力信号及び出力信号の反転を行うことで、2 入力関数の場合 $2^{2+1} = 2^3 = 8$ 通りの異なる論理を一つの DDL 素子で実現することが出来る．また、入力信号の入れ換えを考えると、 $\overline{A} * B$ と $A * \overline{B}$ ($\overline{\overline{A} * B}$ と $\overline{A * \overline{B}}$) は同じものとみなすことも出来る．

このような同族類関数は、まとめると以下に示す様に 2 入力の場合 2 種類、3 入力の場合 10 種類となる．すなわち、物理的にはこれら 12 種類のセルをライブラリとして用意すれば、3 入力までの論理関数はこの中のいずれか一つのセルで表すことが出来る．また、4 入力の場合、同族類の数は 224 種類となり、これら全てをセルライブラリと

表 1 同族類関数
Table 1 Family function.

2 入力関数	3 入力関数
$A * B$	$A * B * C$
$A \oplus B$	$A \oplus B \oplus C$
	$A * B * C + A * \bar{B} * \bar{C}$
	$A * B * C + \bar{A} * \bar{B} * \bar{C}$
	$A * B + C * A$
	$A * B + \bar{A} * \bar{B} * C$
	$A * B * \bar{C} + A * \bar{B} * C + \bar{A} * B * C$
	$A * B + B * C + C * A$
	$A * B + C * \bar{A}$
	$A * B + C * A + \bar{A} * \bar{B} * \bar{C}$

して用意するのは現実的ではない．そこで、本稿では 4 入力の論理関数として、 $Y = A * B * C * D$ を実現する DDL 素子のみを加えてその効果を評価する．

3.2 論理合成用ライブラリ

Synopsys 社の Design Vision 及び Cadence 社の RTL Compiler はデファクトスタンダードとなっている論理合成（テクノロジマッピング）ツールである．これらのツールを用いて論理合成を行うためには、前述の DDL セルライブラリに対応した Liberty フォーマットのファイル（以降 Liberty ファイルと呼ぶ）を作成する必要がある．通常、1 線式回路の場合は一つの論理セルに対して一つの cell 記述がなされた Liberty ファイルが用意される．また、2 線式論理に対応した論理合成ツールは無いため、図 5 の設計フローに示すように、一度 1 線式回路として論理合成した後、自作ツールにより 1 対 1 変換で 2 線化を行う．この際、図 4 に示すように入出力を反転させて接続することで、物理的には同じセルでありながら、異なる論理関数を表すセルとして用いることが出来る．

Liberty ファイルの作成に関しては、2 線式 DDL 素子の場合、同族変換を行って得られる関数それぞれに対応する cell 記述を加えることとなる．物理的には同じ一つの DDL セルであるため、セルの遅延や電力特性を評価するキャラクターライズは各セルに対して 1 度行えば良い．すなわち、表 1 に示したオリジナルの論理関数でキャラクターライズした Liberty ファイルを作成したのち、下記の変換を行うこ

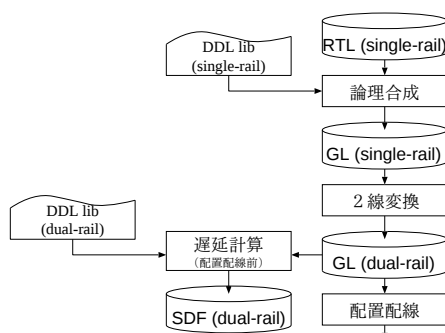


図 5 設計フロー

Fig. 5 Design flow.

とで、同族変換により 2 線式 DDL 素子が表すことの出来る論理関数まで拡張した Liberty ファイルを得ることが出来る．

- パラメータ function で指定される論理関数を変更
- パラメータ sdf_cond 及び when で指定される信号遷移のトリガとなる条件記述を変更
- 出力信号を反転する場合：パラメータ cell_rise と cell_fall, rise_transition と fall_transition, rise_power と fall_power をそれぞれ交換する
- 入力信号を反転する場合：パラメータ timing_sense の unate 関数指定における positive_unate と negative_unate を交換

本稿では、以下の 6 種類の Liberty ファイルを設計し、ツールとして Design Vision 及び RTL Compiler を用いて論理合成を行い、どのような Liberty ファイルが物理的には同じ DDL セルライブラリに対してふさわしいか明らかにする．

- (1) DL1：表 1 に示す 12 種類の論理関数のみ記述した Liberty ファイル（論理関数の数：12 種類）

但し、Liberty ファイルの制約として 2 入力の NOR ゲートが必要なため、2 入力 AND 論理の代わりに 2 入力 NOR 論理関数を用いる．また、INV ゲートも必要とするため、遅延 0 の INV ゲートも含む．

- (2) DL2：DL1 に対して、同族変換を行って得られる論理関数全てを含んだ Liberty ファイル（論理関数の数：176 種類）

この変換では、入力変数の置換によって実現可能な関数（前述の $\bar{A} * B$ と $A * \bar{B}$ のような関係のもの）も全て含むものとする．

- (3) DL3：DL2 に対して、入力変換の置換によって得られる論理関数を取り除いた Liberty ファイル（論理関数の数：90 種類）

Design Vision や RTL Compiler がテクノロジマッピングの際にゲートの入力ポートに信号線を割り当てる時、入力信号の置換で実現される論理セルは冗長なものとして考えられる．ライブラリセルに含まれる論理関数の数によって、論理合成時間などに影響があると考えられるため、その評価を行う．

- (4) DL1+A4：DL1 に対して、4 入力関数 $Y = A * B * C * D$ を追加したもの（論理関数の数：13）
- (5) DL2+A4：DL2 に対して、4 入力関数 $Y = A * B * C * D$ を追加したもの（論理関数の数：208）
- (6) DL3+A4：DL3 に対して、4 入力関数 $Y = A * B * C * D$ を追加したもの（論理関数の数：98）

4. 評価

4.1 評価条件

提案 DDL セルライブラリに関して、Nangate 45nm プロセステクノロジを用いて Cadence 社の Virtuoso により物理設計を行い、RC 抽出した SPICE ファイルを用いて

Synopsys 社 HSPICE によりセルのキャラクタライズを行った．コーナーモデルとしてはプロセス変動を典型値 (typical), 供給電圧を 1.1V, 温度を 25°C と仮定した．

論理合成を行う対象論理として, 32 ビット加算回路, 32 ビット乗算回路, 及び ISCAS89 ベンチマークの組み合わせ回路 29 種類を用いた．それぞれの回路に対して, 初めに遅延制約無しで論理合成を行い, 面積が最小の回路を求め, そのクリティカルパスの遅延をタイミングレポートなどにより取得する．Design Vision 及び RTL Compiler を用いて論理合成を行うにあたり, Design Vision では compile_ultra コマンドを用い, RTL Compiler では合成オプションとして -effort high を指定した．次に, 入力から出力までの遅延制約として 0.1[ns] からその最小面積回路の遅延値まで, 0.1[ns] 刻みで制約を課し, 複数の論理合成結果を得る．得られた回路に対する遅延や面積, 細粒度パイプライン化によるオーバーヘッドなどを比較する．評価に使用したサーバの仕様は次の通りである: Quad-core Xeon 3.60GHz ×4, 24GB Memory, CentOS 5.7 Linux.

4.2 論理合成結果と遅延面積積・細粒度化の比較

図 6 及び図 7 は, 論理合成対象として, 32 ビット加算回路と ISCAS89 ベンチマーク回路のうち s9234 回路に対して, 上述のタイミング制約を課して得られた論理合成結果である．それぞれ上側が Synopsys Design Vision による論理合成結果, 下側が Cadence RTL Compiler による論理合成結果である．横軸は合成結果のクリティカルパスの遅延, 縦軸は面積を表している．但し, 絶対値は NDA により表示していない．図より, 4 入力 AND 素子を加えた方が全体的に小さい回路を実現できていることがわかる．また, もともとの論理関数が 13 種類だけのライブラリでも性能のよい回路を合成出来ているものがあることがわかる．そこで, 評価基準として “遅延 × 面積” を用いた比較を行う．表 2 は Design Vision を使用して論理合成した結果から, 遅延面積積が最小となる回路を選び出し, DL1 を基準として正規化したものであり, 各回路に対して最も値が小さくなったものを太字で表している．AND4 を追加することで約 30%遅延面積積が小さい回路を得られることがわかる．また, 本稿で提案したセルライブラリ DL1, DL2, DL3 は, Design Vision を使用する場合, いずれも遅延面積積が最小となる回路を設計し得ることがわかる．

同様の評価を RTL Compiler を用いて行った．紙面の都合により比較結果の全ては掲載できないが, 遅延面積積の比の平均値を求めたものを表 3 に示す．RTL Compiler では, 論理関数が多い方がよい合成結果が得られる傾向があり, 13 種類の関数からなる DL1+A4 で遅延面積積が最小となる回路はないという結果が得られた．

また, 得られた論理合成結果に対し, 第 2 節で説明した DDL 回路 1 段毎の細粒度化を行った．ここでは, 細粒度化の評価尺度として, “細粒度パイプラインステージ数 × セル数” を用いる．セル数には “Buffer DDL” の数も含ん

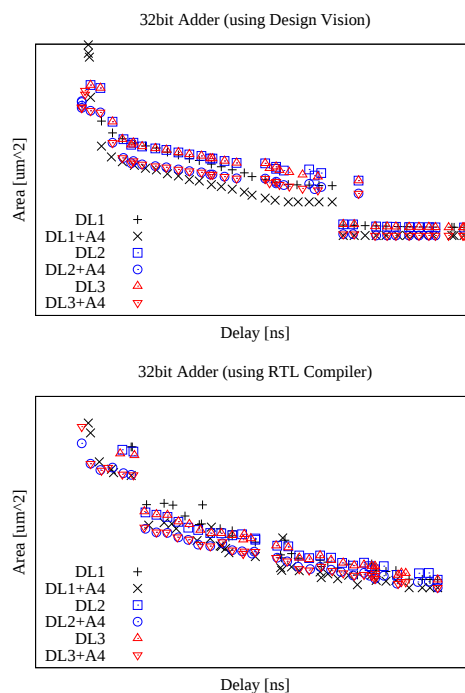


図 6 32 ビット加算回路の論理合成結果

Fig. 6 Synthesis results of 32-bit adder circuits.

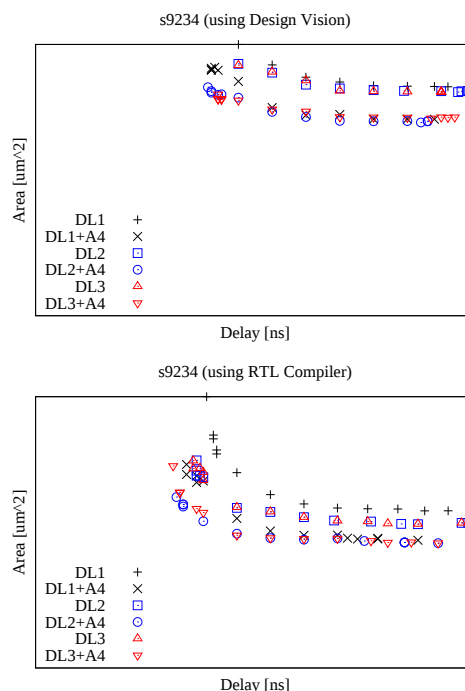


図 7 s9234 回路の論理合成結果

Fig. 7 Synthesis results of ISCAS89-s9234 circuits.

であり, ほぼ面積に比例した値となる．表 4 は対象論理関数の複数の合成結果に対して評価尺度が最も良い (最も小さい) 回路を選択し, DL1 を基準として正規化し, 平均を取ったものである．細粒度化においても, AND4 ゲートを含んだ方が高い性能を得ることが出来ることがわかる．また, Design Vision では冗長な関数を除いたライブラリを用いた方がよく, RTL Compiler では, 関数の種類が多いライブラリを用いた方がよい傾向があることがわかる．

表 2 遅延面積積 最小回路 (Design Vision 使用)

Table 2 Minimum “delay*area” circuits (using Design Vision).

回路名	DL1	DL2	DL3	AND4 追加		
				DL1	DL2	DL3
add32	1.000	0.988	0.987	0.871	0.748	0.748
mul32	1.000	0.967	0.972	0.845	0.871	0.877
s27	1.000	0.865	0.865	0.617	0.701	0.701
s208.1	1.000	1.047	1.058	0.663	0.646	0.634
s298	1.000	1.268	0.937	0.771	0.755	0.680
s344	1.000	0.954	0.954	0.664	0.765	0.765
s349	1.000	0.954	0.954	0.664	0.765	0.765
s382	1.000	1.082	1.083	0.844	0.795	0.792
s386	1.000	1.043	1.019	0.741	0.729	0.739
s400	1.000	1.072	1.045	0.729	0.780	0.771
s420.1	1.000	0.931	0.994	0.617	0.545	0.587
s444	1.000	1.022	1.013	0.802	0.894	0.854
s510	1.000	0.944	0.944	0.656	0.554	0.549
s526	1.000	1.062	1.051	0.697	0.659	0.644
s641	1.000	0.874	0.875	0.571	0.571	0.607
s713	1.000	0.874	0.875	0.571	0.571	0.607
s820	1.000	0.969	0.890	0.645	0.618	0.569
s832	1.000	0.974	0.920	0.651	0.588	0.593
s953	1.000	0.941	0.957	0.647	0.612	0.566
s1196	1.000	0.915	0.929	0.688	0.647	0.654
s1238	1.000	0.916	0.937	0.602	0.570	0.596
s1423	1.000	0.991	0.965	0.783	0.792	0.732
s1488	1.000	0.899	0.908	0.651	0.605	0.603
s1494	1.000	0.869	0.852	0.625	0.564	0.602
s5378	1.000	0.949	0.935	0.711	0.636	0.635
s9234	1.000	0.928	0.923	0.784	0.713	0.718
s13207	1.000	0.998	1.095	0.747	0.730	0.779
s15850	1.000	0.955	0.967	0.730	0.718	0.706
s35932	1.000	0.930	0.930	0.764	0.714	0.714
s38417	1.000	0.972	0.973	0.741	0.751	0.732
s38584	1.000	0.970	0.976	0.732	0.751	0.759
平均	1.000	0.972	0.961	0.704	0.689	0.686

表 3 遅延面積積 最小回路 (RTL Compiler 使用)

Table 3 Minimum “delay*area” circuits (using RTL Compiler).

回路名	DL1	DL2	DL3	AND4 追加		
				DL1	DL2	DL3
平均	1.000	0.848	0.865	0.741	0.649	0.670

表 4 ステージ数 × セル数 最小回路

Table 4 Minimum “stage number * cell number” circuits.

	DL1	DL2	DL3	AND4 追加		
				DL1	DL2	DL3
Design Vision	1.000	0.955	0.930	0.890	0.830	0.827
RTL Compiler	1.000	0.903	0.931	0.829	0.789	0.808

論理合成にかかった時間を正規化してまとめたものを表 5 に示す。同じ物理ライブラリに対してそれが表すことの出来る論理関数まで拡張したことにより、論理合成時間は最大約 1.5 倍まで長く必要とするが、この評価で用いた回路ではほぼ全て 1 分未満で論理合成が終了しているため、許容範囲と考えられる。

表 5 論理合成時間

Table 5 Synthesis time.

	DL1	DL2	DL3	AND4 追加		
				DL1	DL2	DL3
Design Vision	1.000	1.516	1.533	1.111	1.771	1.476
RTL Compiler	1.000	1.356	1.232	0.948	1.309	1.177

5. まとめ

本稿では、ランダム遅延変動に対して高耐性な QDI モデルに基づく非同期式回路設計において、DDL 素子を用いた細粒度非同期式パイプライン回路を実現するためのライブラリとして、物理的には 13 種類のごく小さいセルライブラリを提案した。論理合成用ライブラリとして、DDL 素子が同族変換により表すことの出来る関数まで拡張した Liberty ファイルを提案し、使用するツールに合わせて用いることで高性能 VLSI システムを実現できることを示した。

謝辞

本研究の一部は東京大学大規模集積システム設計教育研究センターを通し、シノプシス株式会社、日本ケイデンス、メンター株式会社の協力で行われたものである。

参考文献

- [1] ITRS. International technology roadmap for semiconductors, the 2013 edition, 2013.
- [2] Stefan Rusu, et al, Ivytown: A 22nm 15-core enterprise xeon processor family. *Proc. ISSCC2014*, pages 102–103, Feb. 2014.
- [3] Alain J. Martin. Synthesis of asynchronous VLSI circuits. In J. Straunstrup, editor, *Formal Methods for VLSI Design*, chapter 6, pages 237–283. North-Holland, 1990.
- [4] 今井雅, 中村宏, 南谷崇. DCVSL を使用した非同期式細粒度パイプライン・データバスの論理合成. 電子情報通信学会技術報告, CPSY98:47–54, Sep. 1998.
- [5] Recep O. Ozdag and Peter A. Beerel. High-speed QDI asynchronous pipelines. In *Proc. ASYNC02*, pages 13–22, April 2002.
- [6] Marcos Ferretti and Peter A. Beerel. Single-track asynchronous pipeline templates using 1-of-N encoding. In *Proc. DATE02*, pages 1008–1015, March 2002.
- [7] B. R. Sheikh and R. Manohar. Energy-efficient pipeline templates for high-performance asynchronous circuits. *ACM Journal on Emerging Technologies in Computing Systems*, 7(4), Nov. 2011.
- [8] Basit Riaz Sheikh and Rajit Manohar. An asynchronous floating-point multiplier. *Proc. ASYNC2012*, pages 89–96, May 2012.
- [9] 今井雅, 五十嵐大将, 工藤三四郎. DDL セルライブラリを用いた非同期式回路設計支援環境の構築. 電子情報通信学会技術報告, CPSY2014-2:3–8, Apr. 2014.
- [10] Scott Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, January 1995.
- [11] Kerry Bernstein, et al., *High Speed CMOS Design Styles*. IBM Microelectronics/Kluwer Academic Publishers, 1998.