2005－ＳＬＤＭ－120（２）
2005／5／19

コンテクストを考慮したparallel prefix adder合成手法

松永多苗子†　　松永　裕介††

† 福岡県産業・科学技術振興財団 福岡知的クラスター研究所　〒814-0001 福岡県福岡市早良区百道浜 3-8-33
†† 九州大学大学院システム情報科学研究院　〒816-8580 福岡県春日市春日公園 6-1
E-mail: †t_matsunaga@fleets.jp, ††matsunaga@c.csce.kyushu-u.ac.jp

あらまし　加算器は，算術演算の中でも最も頻繁に使用される基本的な演算器であり，性能の高い回路を実現するためには，加算器の高速化は重要な課題である．本稿では，加算器のモジュールジェネレータにおいて，その加算器をインスタンス化する際の，各オペランドのビット幅やタイミング制約等，それぞれのコンテクストを考慮して，それらに適した構造をもつ parallel prefix adder を合成する手法を示す．実験結果により，コンテクストを考慮しない場合に比べて 10%程度高速化が可能であることを示すとともに，より品質をあげるための課題について考察する．
キーワード　モジュールジェネレータ，parallel prefix adder，演算器合成，論理合成

An Approach for Context-Oriented Synthesis of Parallel Prefix Adder

Taeko MATSUNAGA† and Yusuke MATSUNAGA††

† FLEETS　3-8-33 Momochihama, Sawara-ku, Fukuoka, 814-0001 JAPAN
†† Kyushu University　6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580 JAPAN
E-mail: †t_matsunaga@fleets.jp, ††matsunaga@c.csce.kyushu-u.ac.jp

Abstract　Binary addition is the most fundamental arithmetic operation, and design of high-quality adders is an important issue to achieve high-performance circuits. In this paper, an approach for context-oriented synthesis of parallel prefix adders is proposed in module generator. Context means the environment where each adder is instantiated in the whole circuit and includes input timing constraint for each bit and bit width of each input operand. Experimental results show that context-oriented approach achieves up to 10% smaller output delay. Issues are also discussed to improve performance more.
Key words　module generator, parallel prefix adder, arithmetic synthesis, logic synthesis

1. はじめに

演算器の設計は，回路全体の品質に影響を与える重要な問題である．中でも加算は算術ユニットの中で最も頻繁に使用される演算であり，加算や減算を実行するだけでなく，より複雑な演算を実行する際に利用されるため，高性能な回路を実現するためには，加算器の高速化は重要な課題である．

加算器の設計に関しては，古くより多くの研究がなされており，様々な特徴をもつアーキテクチャが知られている [1], [2], [3]．それらの多くは，規則的な構造をもち，ビット幅をパラメタとした専用のモジュールジェネレータによって生成し，インスタンス化して回路に組み込むことが可能である．

しかし，実際の回路全体の中で，個々の加算器が実行される状況は，ビット幅が同じであっても，必ずしも同じではない．例えば，その加算器に対するタイミング制約は個々の状況によって異なる．また，n ビットの加算といっても，実は片方の入力のビット幅が小さかったり，オペランドの値の一部，あるいは，全部のビットが定数であることもおこりうる．このような，設計全体の中での個々の演算器がおかれた環境をここではコンテクストと呼ぶ．

[4] では，とくに加算に対する入力タイミング制約がビットごとに均一ではない場合を対象とし，入力到達時刻の差を考慮に入れた parallel prefix adder の構造を生成する手法を提示している．Parallel prefix adder は，carry look-ahead adder のような，キャリー伝搬の高速化方法を一般化した概念を用いており，いくつもの規則的な構造が提案されている．しかし，[4] では，特定の構造を考案するのではなく，与えられた入力到達時刻を考慮した上で，遅延最小となるような構造を，動的計画法によって解くアルゴリズムを提案し，高速な parallel prefix adder として知られている Kogge-Stone の構成と比較して，入力到達時刻を考慮した際の出力遅延が短くなることを示している．

本論文では，入力タイミング制約をコンテクストの一部ととらえ，高性能を目指した加算器の自動生成において，コンテクストに合わせて適切な構成を生成する手法について述べる．[4]で提案された，動的計画法を用いた parallel prefix adder 生成アルゴリズムをもとに，入力タイミング制約だけでなく，ビット幅の異なる加算等，一部のオペランドが定数である場合に対応できるよう拡張する．実験結果を解析するとともに，加算器合成フロー全体の課題についても考察を行なう．

2. 準　　備

2.1 Parallel prefix adder

n ビット加算の入力を $A = a_n, \cdots, a_1, B = b_n, \cdots, b_1$, 出力を和 $S = s_n, \cdots, s_1$, およびキャリー出力 c_n とすると，s_i ，および各ビットのキャリー c_i は以下のように定義される．

$$s_i = a_i \oplus b_i \oplus c_{i-1}$$

$$c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$$

prefix computation とは，n 個の入力 $x_n, x_{n-1}, \cdots, x_1$ と，結合則を満たす任意の演算 $\circ$ が与えられたとき，n 個の出力 y_i を，x_j, $j \leqq i$ となるような入力のみに依存するように生成するものである．

$$y_i = xi \circ x_{i-1} \circ \cdots \circ x_1$$

n ビット加算は，個々のビットに対する generate 関数 (g), propagate 関数 (p)，および，それを複数ビットのグループに拡張した，グループ generate 関数 (G), グループ propagate 関数 (P) を用いて，以下のように prefix computation ととらえることができる．

- 前処理：ビットごとの (g, p) の生成

$$g_i = a_i \cdot b_i$$

$$p_i = a_i \oplus b_i$$

- prefix 処理：(G,P) を用いて $c_i = G_{[i:1]}$ を計算する．

$$P_{[i:j]} = \begin{cases} p_i & \text{if } i = j \\ P_{[i:k]} \cdot P_{[k-1:j]} & \text{if } n \geqq i > j \geqq 1 \end{cases}$$

$$G_{[i:j]} = \begin{cases} g_i & \text{if } i = j \\ G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]} & \text{if } n \geqq i > j \geqq 1 \end{cases}$$

(G, P) に対して，オペレータ $\circ$ を以下のように定義する

$$\begin{aligned}(G,P)_{[i:j]} &= (G,P)_{[i:k]} \circ (G,P)_{[k-1:j]} \\ &= (G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, P_{[i:k]} \cdot P_{[k-1:j]})\end{aligned}$$

- 後処理：各ビットの和 s_i を生成する

$$c_i = G_{[i:1]}$$

$$s_i = p_i \oplus c_{i-1}$$

前処理，後処理は遅延時間が固定であるが，prefix 処理の部分は自由度がある．$\circ$ 演算は結合則がなりたつため，逐次的に計算する必要はなく，様々な順で計算できる．高速計算を行なうためには，如何に並列実行するかが問題となる．

prefix 処理の部分は，$\circ$ をノードとするグラフ (prefix graph) で表現されることが多い．図 1，2 に，代表的な parallel prefix adder の例である，Kogge-Stone parallel prefix adder と Sklansky parallel prefix adder を示す．どちらも ビット幅 n の prefix graph の段数は $log_2 n$ 段で最小であるが，Sklansky の場合は，ファンアウト数が最大で $n/2$ まで変わりうるため、遅延時間への影響が考えられる．

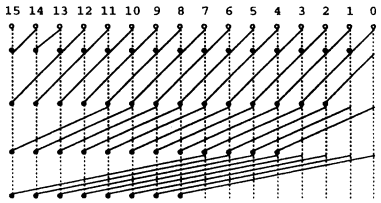

図 1　Kogge-Stone parallel prefix adder

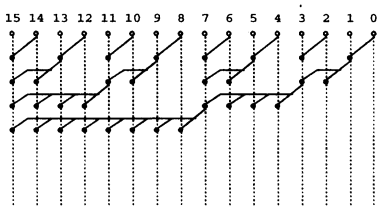

図 2　Sklansky parallel prefix adder

2.2 入力到達時刻の考慮

parallel prefix adder の遅延時間は，一般には prefix graph の (G,P) ノードの段数が目安になる．入力到達時刻が一様である場合には，Kogge-Stone parallel prefix adder が段数最小で，かつ，ファンアウト数も少ないため，高速である．

実際の回路では，入力到達時刻にビットごとのばらつきが起こることがありうる．例えば，Wallace tree 型乗算器において，部分積の最終的な和を計算するキャリー伝搬加算器では，最上位，最下位ビットの到達時刻は速く，中間ビットは遅いという凸型の時刻分布になる．

[5] では，prefix graph 上の局所変換によって，prefix graph の段数やノード数を調整するヒューリスティックが提案された．この枠組では，入力到達時刻にばらつきがある場合にも対応できる柔軟性をもっているが，段数でしか遅延を測る尺度はなく，また，最適性は保証されていない．一方，Liu らは，動的計画法を用いて，入力到達時刻のばらつきを考慮して，遅延時間最小の prefix graph を生成するアルゴリズムを提案した [4]．以下，Liu らのアプローチを示す．

2.2.1 タイミングモデル

- すべての (G,P) ノードは同一の遅延時間 C をもつものとする
- prefix graph の $(G,P)_{[i:j]}$ ノードの出力時刻を $t_{[i:j]}$ とする
- $(G,P)_{[i:j]}$ が $(G,P)_{[i:k]}$,$(G,P)_{[k-1:j]}$ から生成されているとすると

$$t_{[i:j]} = max(t_{[i:k]}, t_{[k-1:j]}) + C$$

a) 動的計画法に基づくアルゴリズム

- prefix graph で，レベル i ごとにその長さ i の GP を生成する．
  - レベルの低い順に $(G,P)_{[i:j]} = (G,P)_{[i:k]} \cdot (G,P)_{[k-1:j]}$ を生成する．j から i の間で分割点 k を動かし，タイミングが最小となる k を見つける．

$$t_{[i:j]} = min_k(max(t_{[i:k]}, t_{[k-1:j]}) + C)$$

- 各レベルで右端の (G,P) が prefix 計算の結果になる．

例えば，図 3 において，$(G,P)_{[3:1]}$ は，$(G,P)_{[3:3]}$ と,$(G,P)_{[2:1]}$ から計算する場合と，$(G,P)_{[3:2]}$ と,$(G,P)_{[1:1]}$ から計算する場合が考えられ，この場合は前者の方が遅延が短いため，$(G,P)_{[3:1]} = (G,P)_{[3:3]} \circ (G,P)_{[2:1]}$ となる．

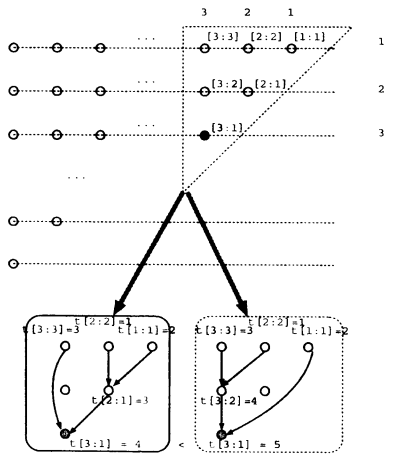

図 3 動的計画法を用いたアプローチ [4]

Fig. 3 An approach using dynamic programming

与えられた複数の入力到達時刻のプロファイルのもとで、Kogge-Stone parallel preix adder と比較して，出力遅延の小さい回路が生成された結果が示されている．

3. 提案手法

本節では，入力到達時刻に加えて，オペランドのビット幅が異なる場合も，同様に枠組で考慮するアプローチを提案する．

3.1 コンテクストを考慮した加算器合成問題

加算の入力を $A = a_n, \cdots, a_1, B = b_m, \cdots, b_1, m \leq n$ としたとき，与えられた入力到達時刻 $Ta = ta_n, \cdots, ta_1, B = tb_m, \cdots, tb_1$ のもとで，遅延時間最小の parallel prefix graph を生成する問題を考える．

3.2 アプローチ

基本的な方針は，[4] と同様であるが，ビット幅の相違を考慮するため，以下の処理を加える．

3.2.1 タイミングモデル

- prefix graph の入力到達時刻

ビット幅が同じ場合，prefix graph のレベル 1 ノードの時刻 $t_{[i:i]}$ は，

$$t_{[i:i]} = max(ta_i, tb_i) + D_{gp},\ D_{gp}\text{は，}gp\text{ ノードの遅延時間}$$

となる．ビット幅が異なる場合，ビット位置 $i(m \leq i \leq n)$ では $b_i = 0$ であるため，gp ノードの論理は，

$$g_i = a_i \cdot b_i = 0$$

$$p_i = a_i \oplus b_i = a_i$$

となるため，$D_{gp} = 0$ とする．このとき，g_i 出力は，値が 0 に固定される．

- (G,P) ノードの遅延時間

$(G,P)_{[i:j]}$ が $(G,P)_{[i:k]}$,$(G,P)_{[k-1:j]}$ から生成されているとする．このとき，

- $G_{[i:k]} \neq 0$ の場合，

$$G_{[i:j]} = G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}$$

$$P_{[i:j]} = P_{[i:k]} \cdot P_{[k-1:j]}$$

- $G_{[i:k]} = 0$ の場合，

$$G_{[i:j]} = P_{[i:k]} \cdot G_{[k-1:j]}$$

$$P_{[i:j]} = P_{[i:k]} \cdot P_{[k-1:j]}$$

- $G_{[i:k]} = G_{[k-1:j]} = 0$ の場合，

$$G_{[i:j]} = 0$$

$$P_{[i:j]} = P_{[i:k]} \cdot P_{[k-1:j]}$$

(出力 G は 0 に固定)

となる．各ノードの遅延時間 D_{GP} は，それぞれの場合により異なるとして、その遅延値は、ノードを論理合成してターゲットライブラリにマッピングした結果の遅延値を用いる．

3.2.2 動的計画法におけるタイブレイク

prefix graph のレベルごとに，各 $(G,P)_{[i:j]}$ に対して，遅延最小となる入力組み合わせを選ぶとき，遅延時間が同じ候補が複数存在する場合がある．prefix graph 上では，どれを選んでも構わないことになるが，実際の回路になると遅延時間に差がでてくる．これは，タイミングモデルとして，ノードの遅延時間のみを考慮しているからで，実際には，クリティカルパス上

表 1　遅延時間の比較

Table 1　Experimental results(delay)

ビット幅 1	ビット幅 2	DP(ns)	DP0(ns)	DP/DP0
16	8	0.90	1.00	0.90
32	16	1.17	1.24	0.94
32	24	1.25	1.23	1.02
64	32	1.46	1.50	0.97
64	48	1.47	1.66	0.89

のノードのファンアウト数が遅延に与えるの影響等が考えられる．そこで，tie break の方法として，同じだったときに，元のを有効（dp1）とするか，新しいのを有効（dp2）とするかの2通りのヒューリスティックを考え，それぞれの場合のファンアウト数や合成後の遅延時間への影響等を調べる．

4. 実験結果

上記の手法で，prefix graph のタイミングモデルの上で最小遅延となる構造を生成し，テクノロジに依存しないレベルのネットリスト記述として出力するモジュールジェネレータを実装した．この結果のネットリスト記述に対して論理合成を実行し，論理簡単化およびテクノロジマッピングを実行し，結果を比較することとする．

実験においては，遅延最適化を目的として，オペランドのビット幅が異なる加算において，コンテクストを考慮した場合としない場合で，論理合成後の回路の遅延時間を比較する．具体的には，ビット幅が異なる加算 5 種

- 16bit-8bit
- 32bit-16bit
- 32bit-24bit
- 64bit-32bit
- 64bit-48bit

に対して，

DP　：幅の違いを考慮して prefix graph を生成

DP0　：同一ビットとして生成して，短い方のビット位置を 0 に固定した場合

の 2 種類の方法で加算器を合成し，遅延時間を比較した．

表 1 に示すように，殆んどの場合で，提案手法の方が，数%から 10%程度高速であるという結果が得られた．

5. 考　　察

今回の実験で一定の効果は確認できたが，単に 0 に固定してインスタンス化した方が結果が良くなる場合も見られた（32 ビットと 24 ビットの加算）．その原因を，prefix graph のタイミングモデルの正当性，および，論理合成の影響，という 2 点から考える．

5.1　タイミングモデルの正当性

prefix graph のタイミングモデルは，ノードの遅延時間のみに依存している．したがって，動的計画法で最小遅延のノードを構築していく際，仮に同じ遅延時間のノード候補があった場合に，そのどれを選んでも結果は変わらないはずであるが，実際には論理合成後の遅延時間には差がでてくる．

表 2　tie break のヒューリスティックの差による遅延時間の比較

ビット幅 1	ビット幅 2	dp1(ns)	dp2(ns)
16	16	1.03	0.99
32	32	1.38	1.24
64	64	1.60	1.54
128	128	1.90	1.77
16	8	0.91	0.90
32	16	1.22	1.17
32	24	1.33	1.25
64	32	1.43	1.46
64	48	1.62	1.47

今回のプログラムにおいては，タイブレイクのために前述の 2 つのヒューリスティックを導入し，実験を行なった．結果を表 2 に示す．

実験結果をみると，同じビット同士の加算の場合，明らかに dp2 の方が結果が良かった．両者の合成結果は，dp1 が Sklansky parallel prefix adder, dp2 は Kogge-Stone parallel prefix adder と同じ構造になっており，dp1 の方では最大 n/2 個のファンアウトをもつノードが存在した．一方，ビット幅が異なる場合には，両者の差異は小さくなり，逆転する場合もおきている．prefix graph のファンアウト数を調べたところ，dp2 の場合に大幅な増加がみられており，これが影響していると考えられる．ビット幅が異なる場合には，入力到達時刻が一定ではなくなり，構造が不規則になるため，同一ビットの時のような傾向はみられなくなったものと考えられる．

タイブレイクについては，ランダムに選ぶ，ファンアウト数の増加が少ない方を選ぶなど，いくつかの候補は考えられるが，効果は場合により，本質的な解決にはなっていない．

なお前節の実験結果は，dp2 を用いている．

5.2　論理合成の影響

論理合成の結果は，最適化オプションや与える制約によって大きく変化しうる．その影響をみるため，与える制約，および，最適化のオプションを変えて比較を行なった．表 3,4 において，1 列目はビット幅，2 列目は最大遅延制約値を示す．- は，制約を与えずに合成した場合である．

prefix graph から生成されたネットリスト記述は，gp, GP ノードを verilog の module として定義し，それらをインスタンス化した接続記述である（図 4）．論理合成で最適化を実行する際，モジュール境界を保持させる場合と，取り払う場合がありうる．表で dp2 は境界を保持した場合であり，dp2U は境界を取り払った場合 (ungroup) である．

表 3,4 をみる限り，ungroup した方が (dp-2U) 探索空間が広がるため，全般的に結果はよい．同一ビット幅の加算の場合，概ね，dp-2U の方が最小遅延時間が小さくなり，同じ制約を満たす範囲では面積が小さくなっている．ただし，64, 128 ビットに対しては，最小遅延がむしろ大きくなっている．一方，ビット幅が異なる場合は，全体的に dp2-U の方が高速で小さい回路が得られている．

```
module gp_gen ( input a,
                input b,
                output g,
                output p);
  assign g = a & b;
  assign p = a ^ b;
endmodule // gp_gen


module GP_black ( input gi1,
                input gi0,
                input pi1,
                input pi0,
                output go,
                output po);
  assign go = gi1|pi1 & gi0;
  assign po = pi1 & pi0;
endmodule // GP_black

module GP_black2 (
                input gi0,
                input pi1,
                input pi0,
                output go,
                output po);
  assign go = pi1 & gi0;
  assign po = pi1 & pi0;
endmodule // GP_black2
...

module prefix_DP1_16_8(
  input [15:0] a,
  input [7:0] b,
  output [16:0] z );
...
  gp_gen gp0(.a(a[0]),.b(b[0]),.g(w_g0_0),.p(w_p0_0));
  gp_gen gp1(.a(a[1]),.b(b[1]),.g(w_g0_1),.p(w_p0_1));
  gp_gen gp2(.a(a[2]),.b(b[2]),.g(w_g0_2),.p(w_p0_2));
...
  GP_black GP_b1_1(.gi1(w_g0_1), .gi0(w_g0_0),
       .pi1(w_p0_1), .pi0(w_p0_0),
       .go(w_g1_1), .po(w_p1_1) );
  GP_black GP_b1_3(.gi1(w_g0_3), .gi0(w_g0_2),
       .pi1(w_p0_3), .pi0(w_p0_2),
       .go(w_g1_3), .po(w_p1_3) );
  GP_black GP_b1_5(.gi1(w_g0_5), .gi0(w_g0_4),
       .pi1(w_p0_5), .pi0(w_p0_4),
       .go(w_g1_5), .po(w_p1_5) );
...
```

図 4　モジュールジェネレータの出力

Fig. 4　An output description of module generator

表 3　論理合成の影響（ビット幅が等しい場合）

bit 幅	制約	dp2(area)	dp2(delay)	dp2U(area)	dp2U(delay)
16	0.0	12614	1.08	13697	0.99
	3.0	5017	2.80	3606	2.87
	-	4717	2.69	2229	7.21
32	0.0	30597	1.29	26980	1.24
	3.0	12430	2.99	10218	3.00
	5.0	12205	3.30	9124	4.75
	-	11445	3.12	5985	6.94
64	0.0	72599	1.47	61194	1.54
	3.0	30764	3.00	26473	3.00
	5.0	28776	3.85	22170	4.99
	-	26928	3.71	14723	8.71
128	0.0	164442	1.69	136559	1.77
	3.0	83778	3.00	68279	3.00
	5.0	66394	4.45	52663	5.00
	-	61948	4.58	34905	9.71

表 4　論理合成の影響（ビット幅が異なる場合）

bit 幅	制約	dp2(area)	dp2(delay)	dp2U(area)	dp2U(delay)
16/8	0.0	10863	1.14	8260	0.90
	3.0	4746	2.78	2673	2.89
	-	4481	2.89	1590	5.27
32/16	0.0	28063	1.34	20696	1.17
	3.0	12315	3.00	7165	2.99
	5.0	12165	3.41	6756	4.27
	-	11416	3.62	4562	5.35
32/24	0.0	28155	1.31	26415	1.25
	3.0	12131	3.00	8640	2.99
	5.0	11912	3.26	7903	4.42
	-	11186	3.34	5253	6.31
64/32	0.0	65117	1.57	44444	1.46
	3.0	29905	3.00	17366	3.00
	5.0	28776	4.16	15638	4.76
	-	27037	4.93	11514	5.91
64/48	0.0	67778	1.57	61321	1.47
	3.0	30527	3.00	22383	3.00
	5.0	28615	3.80	19210	4.98
	-	26807	3.97	13173	7.33

コンテクストを考慮した parallel prefix adder では，全体として構造が不規則になること，回路構成の基本要素となる G, P 等の論理は，フルアダーのような演算器用の特殊なセルではなく，通常のコントロールロジックを扱うのと同じ枠組で扱える，という特徴があるため，論理合成の技術が有効に使える領域であると考えられる．しかし，その反面，論理合成によって回路構造が大幅に変わるとなると，prefix graph でのタイミングモデルが実際とずれがでてくる可能性がある．

高性能演算器合成を目指すためには，prefix graph の生成と論理合成を切り離すのではなく，prefix graph 生成の際に，ファンアウトの影響やテクノロジマッピング等も考慮に入れるアプローチの検討が重要だと思われる．

6. おわりに

本稿では，入力到達時刻やオペランドのビット幅等のコンテクストに合わせて，高性能な parallel prefix adder を生成するモジュールジェネレータを実装し，実験により効果を確認した．より高性能を目指すためには，ファンアウトの影響の考慮，論理最適化やテクノロジマッピング技術の融合についての検討が重要である．

謝辞 本研究は，福岡地域の文部科学省知的クラスター創成事業の支援による．

文　献

[1] James E. Stine, "Digital Computer Arithmetic Datapah Design Using Verilog HDL", Kluwer Academic Publishers

[2] Israel Koren, "Computer Arithmetic Algorithms", A.K.Peters Ltd.

[3] Neil H.E. Weste, David Harris,"CMOS VLSI Design: A Circuits and Systems Perspective", Addison Wesley

[4] Jianhua Liu, Shuo Zhou, Haikun Zhu, and Chung-Kuan Cheng, "An Algorithmic Approach for Generic Parallel Adders", in Proceedings of ICCAD'03, pp.734-740, Nov.2003.

[5] R.Zimmermann, "Non-heuristic optimization and synthesis of parallel-prefix adders", in Proceedings of International Workshop on Logic and Architecture Synthesis, Dec.1996, pp.123-132.