IPSJ Transactions on System LSI Design Methodology Vol. 2 212–221 (Aug. 2009)

Regular Paper

Framework for Parallel Prefix Adder Synthesis Considering Switching Activities ⋆1

Taeko Matsunaga,†1 Shinji Kimura†1
and Yusuke Matsunaga†2

This paper addresses parallel prefix adder synthesis which aims at minimizing the total switching activity under bitwise timing constraints. This problem is treated as synthesis of prefix graphs which represent global structures of parallel prefix adders at technology-independent level. An approach for timing-driven area minimization of prefix graphs has been already proposed which first finds the exact minimum solution on a specific subset of prefix graphs by dynamic programming, then restructures the result for further reduction by removing restrictions on the subset. In this paper, a switching cost of each node of a prefix graph is defined, and an approach to minimize the total switching cost is presented where our area minimization algorithm is extended to be able to calculate the switching cost using Ordered Binary-Decision Diagrams (OBDDs). Furthermore, a heuristic is integrated which estimates the effect of the restructuring phase in the dynamic programming phase, to improve the robustness of our algorithm under severe timing constraints. Through a series of experiments, the proposed approach is shown to be effective especially when timing constraints are not tight and/or there are comparably a large number of nodes with very low switching costs.

1. Introduction

Arithmetic circuits are important circuit elements which could have a large impact on qualities of the whole circuits. Among various arithmetic operations, binary addition is the most fundamental and frequently-used one. A lot of studies have been done and various architectures of adders can be found in many literatures [7]. Among them, parallel prefix adders belong to a class of well-known architectures which generalize carry look-ahead idea for faster carry propagation. In addition to many regular structures [1),6),13)], some algorithms for automatic generation of parallel prefix adders also have been proposed, especially for area minimization [8),14)]. The authors also have proposed an algorithm for parallel prefix adder synthesis which targets area minimization under given bitwise timing constraints at technology independent level [11)]. The problem is captured as a problem to minimize the number of nodes of prefix graphs which represent global structures of parallel prefix adders at technology independent level.

As for a cost to be minimized, power is another important target. Power consumption is becoming one of the major concerns in recent LSI design, and a lot of work has been done for estimation of power, low-power design techniques at various design levels, and acceleration techniques of power estimation [4),12)]. Among various factors, the switching activity is a key factor at logic level, and often estimated and utilized in low power synthesis techniques [3),5)].

In this paper, an approach for power-aware synthesis of prefix graphs is presented. A cost is defined based on the switching activities for nodes in prefix graphs as a factor which affects power consumption, and a switching-cost minimization algorithm is implemented based on our area minimization one with a heuristic which improves the robustness.

As for power minimization of prefix adders, Liu, et al. have proposed an approach to minimize power of prefix adders under timing and area constraints [9)]. Their approach handles a realistic area/timing/power model considering gate and wire capacitance measures, and formulates the power minimization problem under area and timing constraints as an Integer Linear Programming (ILP) problem. It solved the problem exactly though it could be applied to only small adders (8-bit or so) in practically. A divide-and-conquer approach is needed for larger size, which means the exactness will be lost.

In contrast, our approach uses rough measures, so can be more tractable for larger prefix graphs at the expense of accuracy. Our main concern is to construct a framework at technology independent level without any assumption on implementation and leave flexibility for back-end design phases. Other costs such as static power or capacitance-related factors could be integrated if they are defined as the total summation of each node's cost.

†1 Graduate School of Information, Production and Systems, Waseda University
†2 Faculty of Information Science and Electrical Engineering, Graduate School of Kyushu University
⋆1 An original version of this paper was presented at IEEE International Conference on Computer Design, in October 2008 [10)].

The rest of this paper is organized as follows. In Section 2, several definitions related to parallel prefix adders and prefix graphs are described as preliminaries. Then, the basic idea of our area minimization algorithm is briefly reviewed in Section 3. In Section 4, our switching costs on prefix graphs are introduced, and a basic minimization algorithm and a heuristic for specific cases are proposed. Then, experimental results and conclusions are shown in Sections 5 and 6 respectively.

2. Preliminaries

2.1 Parallel Prefix Adder (PPA)

Given two inputs $A = a_n, \cdots, a_1$ and $B = b_n, \cdots, b_1$, n-bit binary addition computes the sum $S = s_n, \cdots, s_1$ and the carry c_n as $s_i = a_i \oplus b_i \oplus c_{i-1}$ and $c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$. In parallel prefix adders, this binary addition is calculated using the bitwise generate and propagate functions and the group generate and propagate functions as basic operations as follows:

- Pre processing: Calculate bitwise generate(g) and propagate(p) functions.
  $g_i = a_i \cdot b_i$
  $p_i = a_i \oplus b_i$
- Prefix computation: Compute $c_i = G_{[i:1]}$ by using group generate(G) and propagate(P) functions.

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

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

  where $j < k \leq i$.
- Post processing: Generate each sum bit s_i.
  $c_i = G_{[i:1]}$
  $s_i = p_i \oplus c_{i-1}$

A parallel prefix adder consists of the above three parts (**Fig. 1**). The pre-processing and the post-processing parts have fixed structures, while the prefix computation part has wide flexibility for its configuration depending on how to select k. So, how to execute prefix computation affects the qualities of parallel prefix adders.

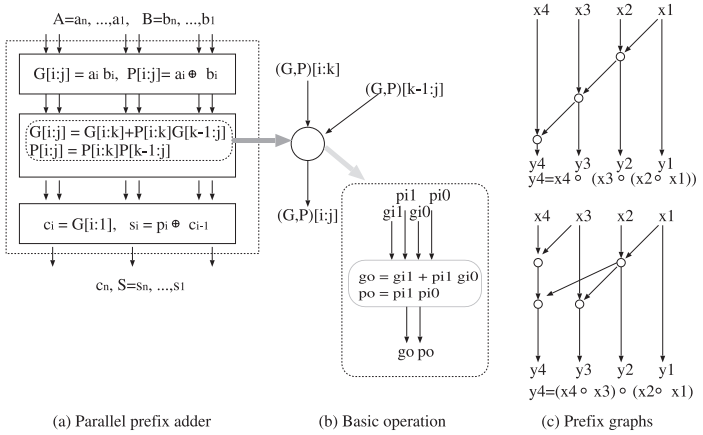


Fig. 1 Parallel prefix adder.

2.2 Prefix Graph

A global structure of a parallel prefix adder at technology-independent level can be represented as a prefix graph whose node corresponds to a basic operation $\circ$ defined as follows.

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

Definition 1 A prefix graph with N inputs is a directed acyclic graph (DAG) whose nodes correspond to associative operations $\circ$ in prefix computation on N inputs, and represents the execution order of each node. An edge from node v_1 to v_2 exists if v_1 is an operand of the operation corresponding to v_2, where a preceding node v_1 of v_2 is referred to as a fanin node of v_2 and a succeeding node v_2 of $v_1$is referred to as a fanout node of v_1.

Figure 1 (c) shows two simple examples of prefix graphs with width 4. **Figure 2** shows the aligned prefix graph corresponding to the bottom graph in Fig. 1 (c), where each node is aligned by the width of input range. Node $v_{i,j}$ which is at i-th row and j-th column ($i \leq j$) represents an intermediate result of prefix computation on i inputs $x_j, x_{j-1}, \cdots, x_{j-i+1}$. For example, a node $v_{2,4}$ in Fig. 2 has two fanin nodes, $v_{1,4}$ and $v_{1,3}$, and represents the result of operation $v_{1,4} \circ v_{1,3}$.

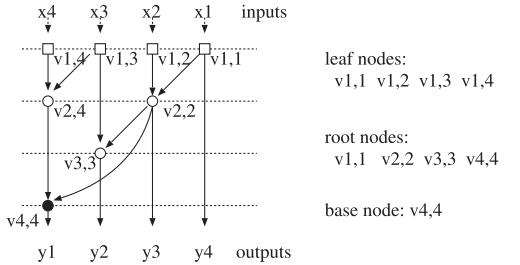


Fig. 2 Aligned prefix graph.

The input ranges of $v_{2,4}$ and $v_{2,2}$ are $[4:3]$ and $[2:1]$ respectively. $v_{4,4}$ is the result of operation $v_{2,4} \circ v_{2,2}$ and its input range is $[4:1]$. Idempotency is not considered, that is, we assume that two ranges do not overlap right now.

In an aligned prefix graph with width N, each node $v_{1,j}$ is called a leaf node and $v_{j,j}$ a root node where $1 \leq j \leq N$. A leaf node and a root node correspond to an input and an output respectively. The root node with the widest width is called a base node, and the other root nodes are called non-base nodes. In what follows, aligned prefix graphs are used as a representation of prefix graphs.

2.3 Area and Timing Measures on Prefix Graph

In this work, area of a prefix graph is measured by the number of nodes except leaf nodes, and referred to as the size of a prefix graph. As for timing measures, two types of node levels are defined.

Definition 2 Arrival level $AL(v)$ and required level $RL(v)$ of a node v in a prefix graph are defined as follows.

$$AL(v) = \max\{AL(v'),\ v' \in FI(v)\} + 1,$$
$$RL(v) = \min\{\min\{RL(v'), v' \in FO(v)\} - 1, RL'(v)\}$$

where $FI(v)$ and $FO(v)$ are sets of fanin and fanout nodes of v respectively, and $RL'(v)$ is the required level already given for itself(∞ if not specified).

For example, area and the arrival level of $y4$ are 3 and 3 in the upper graph of Fig. 1 (c), and 4 and 2 in the lower graph respectively when given arrival levels are 0 for all inputs.

Assume that arrival levels for input nodes and required levels for output nodes are given as timing constraints. A prefix graph satisfies given timing constraints if $RL(v) \geq AL(v)$ for all nodes v.

3. Prefix Graph Synthesis for Area Minimization

In this section, the basic idea of our timing-constrained area minimization algorithm is summarized [11].

Our first target for PPA synthesis was timing-constrained area minimization problem, which was captured as a problem to search prefix graphs as follows.

Problem 1 Given bitwise input arrival levels and output required levels, generate a prefix graph of the minimum size.

The main idea to tackle this problem is to define a specific subset of the prefix graphs and decompose the whole minimization process into two phases. The first phase, DP phase, finds an exact minimum solution for the subset by dynamic programming. Then further reduction is done by restructuring the resulting prefix graph in the second phase, RS phase.

In DP phase, the minimum solutions can be found for practical bit width within a reasonable time by defining the subset appropriately. We call an element of the subset a restricted prefix graph, which is defined as follows.

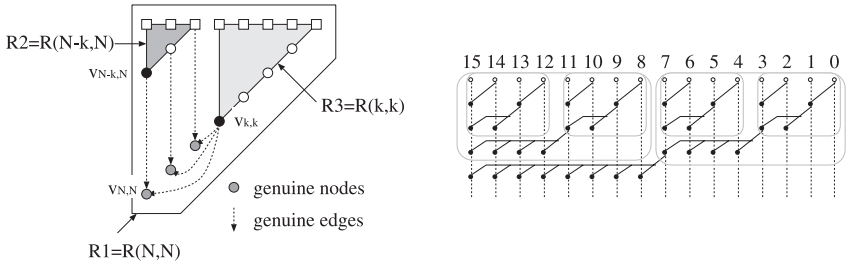
Definition 3 Assume that two fanin nodes of the base node $v_{N,N}$ of a prefix graph with width N are $v_{N-k,N}$ and $v_{k,k}$. A restricted prefix graph is a prefix graph whose structure has the following features as shown in **Fig. 3**:

- Fanin nodes of a root node $v_{N-j,N-j}$ are $v_{k,k}$ and $v_{N-k-j,N-j}$ where $0 \leq j < N-k$.
- Subgraphs whose base nodes are $v_{N-k,N}$ and $v_{k,k}$ also have the above features recursively.

Nodes $v_{N-j,N-j},\ 0 \leq j < N-k$, which are not included in subgraphs, are called genuine nodes of the restricted prefix graph, and their fanin edges are called genuine edges. We denote a set of restricted prefix graphs whose base nodes are $v_{n,p}$ by $R(n,p)$.

Figure 3 (b) shows a famous parallel prefix adder, Sklansky PPA [13], which is considered as an example of a restricted prefix graph. In other words, a restricted prefix graph is a generalized version of Sklansky PPA where a prefix graph can be divided into any two parts with possibly different widths.

A restricted prefix graph R_1 consists of mutually exclusive sub-graphs R_2 and R_3, and genuine nodes. The size of R_1 can be calculated as the sum of the



(a) Definition of Restricted Prefix Graph (b) An example (Sklansky PPA)

Fig. 3 Restricted prefix graph.

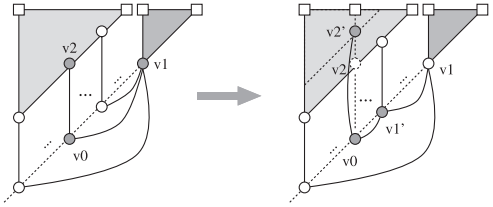


Fig. 4 Restructuring-phase.

sizes of both sub-graphs and the number of genuine nodes. Each sub-graph has the same structure recursively. As a result, the problem which generates a restricted prefix graph with the minimum size under bitwise timing constraints satisfies the principle of optimality. Therefore, a restricted prefix graph with the minimum size can be constructed by solving subproblems with smaller widths and combining the optimum solutions based on dynamic programming.

The resulting prefix graph is restructured by removing imposed restrictions for further area minimization in RS phase. In **Fig. 4**, when a fanin node of v_0 changes from v_1 to $v_{1'}$, another fanin node will be changed from v_2 to $v_{2'}$. Then if $v_{2'}$ is already connected to some node and the number of fanout nodes of the original fanin node v_2 becomes 0, v_2 can be removed.

We expect our two-folded framework based on the exact solutions for the subset can generate globally better structures than ad hoc heuristics. There is no guarantee that the resulting restricted prefix graph might be better seeds for finding better prefix graphs in RS phase. Though effectiveness can not be evaluated theoretically, prefix graphs with smaller sizes are achieved experimentally compared with well-known parallel prefix adders such as Brent-Kung [1], Sklansky [13], and Kogge-Stone [6], as well as those generated by existing heuristics [8),14)], in reasonable execution time.

4. Prefix Graph Synthesis for Switching-Cost Minimization

Our synthesis approach is not restricted to area minimization, and extended to treat other problems whose costs to be minimized can be calculated as the summation of the cost of each node. Switching activity, a factor which affect dynamic power consumption, is an example of those costs. In this section, we propose an approach to minimize a power-related measure at the prefix graph level by utilizing the above synthesis framework.

4.1 A Measure of Power

Dynamic power dissipation of a CMOS gate i is given by

$$DP_i = \frac{1}{2} C_i \cdot V_{dd}^2 \cdot SW_i \cdot f_{clk},$$

where C_i is the physical capacitance at the output of the gate, V_{dd} is the supply voltage, and f_{clk} is the clock frequency. SW_i is referred to as the switching activity, and is the average number of output transitions per $1/f_{clk}$ time, that is, a clock cycle. Dynamic power of the whole circuit is calculated as the sum of DP_i of every gate of the circuit.

Since V_{dd}^2 and f_{clk} are not tractable at logic level, there are two factors, C_i and SW_i to be targeted for power minimization. In this paper, we pick up SW_i as an important factor of power consumption, define a switching cost of a prefix graph node, and focus on the minimization of the total switching cost of all nodes as the target cost to be minimized for power reduction.

There are several approaches to estimate SW_i at logic level [12)]. Calculation of the switching activity depends on many factors such as input-pattern dependency and delay model, and there is a trade off between its accuracy and calculation time. Since technology-dependent factors can not be estimated accurately at the prefix graph level because of the loss of information and the efficiency is crucial in iteratively calculating the costs during synthesis process, we adopt probability-

based estimation under zero delay or non-glitch model [12] where all changes at the inputs propagate through the internal gates instantaneously.

Definition 4 Signal probability $p(i)$ at a node i is the probability that the signal value at the node is one.

We assume that $p(x_i)$ is given for each input x_i. All input signals are assumed to be independent each other (spatial independence). Furthermore, the values of the same input signal in two consecutive clock cycles are also assumed to be independent (temporal independence).

Under these assumptions, the switching activity at an internal node i is calculated as the summation of probabilities that the signal value changes from 0 to 1 and 1 to 0 in consecutive clock cycles as follows:

$$\begin{aligned} SW_i &= p(i) \cdot (1 - p(i)) + (1 - p(i)) \cdot p(i) \\ &= 2 \cdot p(i) \cdot (1 - p(i)) \end{aligned}$$

To calculate the switching cost, signal probabilities should be calculated for all internal nodes. Though they could be obtained by simulation, an OBDD [2]-based approach can be applied effectively under our assumptions.

Assume that the global function of an output of node v is $f(x_1, \cdots x_j)$, where $x_1, \cdots x_j$ are primary inputs. Since $f = x_{j'} f_{x_{j'}} + \bar{x_{j'}} f_{\bar{x_{j'}}}$ for $x_{j'} (1 \leq j' \leq j)$, the signal probability of this function is calculated as follows:

$$\begin{aligned} p(f) &= p(x_{j'}) p(f_{x_{j'}}) + p(\bar{x_{j'}}) p(f_{\bar{x_{j'}}}) \\ &= p(x_{j'}) p(f_{x_{j'}}) + (1 - p(x_{j'})) p(f_{\bar{x_{j'}}}) \end{aligned}$$

where $f_{x_{j'}}$ and $f_{\bar{x_{j'}}}$ are the cofactors of f with respect to $x_{j'}$ and $\bar{x_{j'}}$, respectively. When signal probabilities of all input signals are given, $p(f)$ can be calculated by generating the global function as an OBDD and traversing the OBDD using the above equation.

We now expand the above switching activity to the switching cost of prefix graph nodes.

Definition 5 The switching cost of a prefix graph node v is defined as follows.

$$\begin{aligned} SW(v) &= \alpha \cdot SW(G) + \beta \cdot SW(P) \\ SW(G) &= 2p(G) \cdot (1 - p(G)) \\ SW(P) &= 2p(P) \cdot (1 - p(P)) \end{aligned}$$

As mentioned before, a prefix graph node corresponds to a (G, P) operation and has two distinct outputs G and P corresponding to go and po in Fig. 1 (b). The switching cost of a node is defined as the summation of those of G and P outputs with some factors α and β. Currently they are set to one, though they can be changed to any value.

Our measure of power is rather simple with several assumptions, and would be less accurate than the power model defined in Ref. 9). However, it would be possible to include the effects of static power and glitch if they could be modeled as the summation of node costs, like in their model.

4.2 Switching Cost Minimization Algorithm

Now, the target problem is as follows:

Problem 2 Given bitwise input arrival levels, output required levels, and input signal probabilities, generate a prefix graph whose total switching cost is minimum.

As well as the size of a prefix graph, the total switching cost of a prefix graph is calculated by simply summing up the switching cost of each node. So, the same techniques used in area minimization algorithm are applicable in minimization of switching costs except cost calculation.

In our synthesis framework, any nodes on the same row of an aligned prefix graph have the same global functions, and they can be generated before DP phase. Switching costs of all nodes can be calculated also in advance based on given input signal probabilities. So, timing-constrained switching cost minimization algorithm can be implemented efficiently by expanding our area minimization algorithm as follows:

(1) Generate global functions of all possible nodes and calculate switching costs under given signal probabilities using OBDD-based approach.
(2) DP Phase: Find the restricted prefix graph with the minimum total switching cost by dynamic programming. Since switching costs of all nodes have been already calculated, switching costs of prefix graphs can be calculated by simply summing up already calculated costs of subgraphs and genuine nodes.
(3) RS Phase: Start from the above result, and re-structure for further reduction of switching costs. Gains by removing nodes are calculated as the total

switching costs of removable nodes.

Because of the pre-computation of node costs, execution time is comparable to those of area minimization algorithm, and can be applied for practical bit widths in reasonable time.

4.3 A Heuristic for Robustness

We modified our synthesis framework to be able to address switching cost minimization, and executed several preliminary experiments. Our new approach resulted in lower switching costs compared to existing heuristics for area minimization and regular parallel prefix adders such as Sklansky and Brent-Kung PPAs. However, we also found that it sometimes generated prefix graphs with higher switching costs than those generated by area minimization algorithm. Those situations often occurred under severe maximum delay constraints, that is, output required levels were set to the possible minimum levels. In these cases, though restricted prefix graphs with the minimum cost were obtained in DP phase, RS Phase could not remove much nodes since there was no or few opportunities to restructure it without conflicting timing constraints. All nodes are considered to have the same cost in area minimization, but switching cost of each node is not same, and there can be several prefix graphs where the differences of total costs are very subtle. We observe that this subtle difference sometimes could prevent a good seed for RS phase from being generated.

Our approach for this problem is to roughly estimate the gain of restructuring in RS Phase and modify the costs in DP Phase. RS phase calculates the gain, that is, the number of removable nodes by changing the diagonal fanin nodes to each possible genuine node, selects the best candidate for restructuring, and changes fanins. It repeats this procedures until no restructuring occurs. A node is removable when it has no fanout nodes.

As shown in **Fig. 5**, if a node of a restricted prefix graph is not a base node, it has at most one fanont node and the fanout node is also a non-base node. When a node is a base node, it can have more than one fanout node and can not be removed. So, only non-base nodes relate to the gain calculation. In our approach, non-base nodes in the same column are recorded, and used to calculate the gain in DP phase.

This heuristic only treats simplified restructuring where a fanin node of

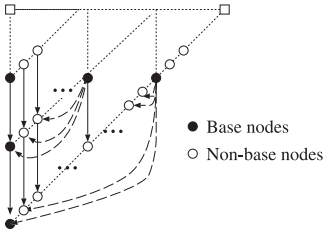


Fig. 5 Fanout nodes of base nodes and non-base nodes.

$v_{n-j,n-j}$ can be changed only to $v_{n-j-1,n-j-1}$, so it might be less accurate for more complicated restructuring. We basically apply this heuristic only where timing constraints are severe and there are few possible alternatives which satisfy timing constraints. Though this approach is an ad hoc heuristic, it could be expected to enhance the robustness of our approach.

5. Experiments

We integrated the above heuristic to synthesis framework, and executed it under various combinations of input signal probabilities and timing constraints. We also implemented and executed existing heuristics [8),14)] for comparison. All experiments were done on Pentium 4, 2.4 GHz machine.

5.1 Case1: PPA in Wallace-tree Multipliers

The first set of experiments is for 24-bit and 39-bit PPAs used in 16*16 and 24*24-bit Wallace-tree multipliers respectively. Input timing constraints for PPAs were extracted from the structures of both multipliers. Signal probabilities of primary inputs of PPAs were estimated from the structures of multipliers under the condition where signal probabilities of all inputs of multipliers were equal to 0.5.

Table 1 shows the total switching costs for 24-bit and 39-bit prefix graphs under several output required levels from the possible minimum levels to more relaxed levels shown in the second column. The third to the seventh columns show the total switching costs of the resulting prefix graphs generated by area minimization algorithm (ND), switching-cost minimization algorithm (SW1), switching-cost minimization with a heuristic (SW2), and existing

Table 1 Total switching costs for PPAs in Wallace-tree multipliers.

bit	LV	ND	SW1	SW2	LO	DO	ND/SW	LO/SW	DO/SW
24	19	31.22	31.16	30.75	31.68	31.68	1.02	1.03	1.03
	20	28.98	28.90	28.90	29.98	31.88	1.00	1.04	1.10
	21	28.97	28.10	28.10	29.46	29.46	1.03	1.05	1.05
	22	28.89	27.42	27.42	29.34	29.92	1.05	1.07	1.09
	23	28.75	26.88	26.88	29.10	29.10	1.07	1.08	1.08
	24	28.50	26.52	26.52	28.84	28.84	1.07	1.09	1.09
	25	28.20	25.95	25.95	28.47	28.47	1.09	1.10	1.10
39	21	57.26	58.06	56.63	58.72	61.75	1.01	1.04	1.09
	22	51.64	52.40	51.88	53.94	60.03	1.00	1.04	1.16
	23	50.98	50.70	51.63	52.81	57.69	1.01	1.04	1.14
	24	50.14	49.46	49.46	52.02	54.62	1.01	1.05	1.10
	25	50.13	48.65	48.65	51.83	53.16	1.03	1.07	1.09
	26	50.02	47.86	47.86	51.45	53.22	1.05	1.07	1.11
	27	49.79	47.15	47.15	51.09	51.79	1.06	1.08	1.10

heuristics (LO [14] and DO [8]). The last three columns indicate ratios of ND, LO, and DO to SW, where SW is the smaller value between SW1 and SW2 and the results of our switching cost minimization algorithm.

As shown in Table 1, SW resulted in the lowest costs in all cases. The reduction ratio grows as output required levels increase. SW2 seems to be effective for tighter timing constraints and not for relaxed ones as expected.

Execution time of SW1 and SW2 are around 20 seconds and 25 seconds respectively for 39-bit with level 27 (the largest case of the above setting).

5.2 Case2: 32-bit Adders with Various Timing Constraints and Signal Probabilities

The second set of experiments is for observing the impact of signal probabilities and input timing constraints in our synthesis framework. We set the following three types of input timing constraints for 32-bit PPAs as shown in **Fig. 6**:

(1) 32C: convex pattern (the upper left of Fig. 6)
(2) 32C_l0: late inputs for upper 16 bits (the upper right of Fig. 6)
(3) 32C_u0: late inputs for lower 16 bits (the bottom of Fig. 6)

As for signal probabilities, the following two types are considered:

(1) uni: all inputs have the same signal probabilities (0.5)
(2) zero: signal probabilities are extracted under the situations where the adder sequentially calculates the total summation of 16-bit data.

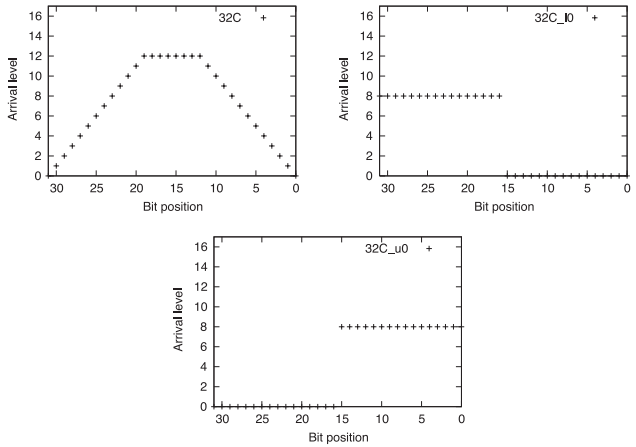


Fig. 6 Three types of input profiles for 32 bit adder.

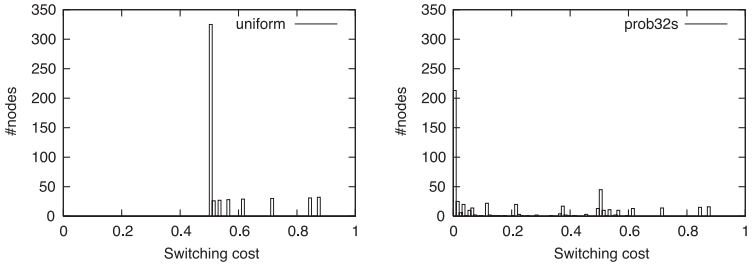


Fig. 7 Switching cost of each node.

The latter case is an example where there are many nodes with very low costs. **Figure 7** shows the distributions of switching costs for all the possible nodes in a 32-bit prefix graph. The horizontal axis indicates switching cost per one node and the vertical axis shows the number of nodes with the cost. The left graph is for "uni" case, and the right one is for "zero" case. Compared to "uni" case, there are found many nodes with 0 or nearly 0 costs in "zero" case.

Table 2, **Table 3**, and **Table 4** show the total switching costs of the resulting prefix graphs under the above three input profiles respectively. Regardless of input profiles, the advantage of our approach increases as output required levels increase. Among the three input profiles, our approach is most effective for 32C_l0

Table 2 Total switching costs under timing constraints "32C".

prob	LV	ND	SW	LO	DO	ND/SW	LO/SW	DO/SW
uni	16	60.53	60.07	65.47	65.47	1.01	1.09	1.09
	17	54.40	54.34	60.18	60.18	1.00	1.11	1.11
	18	53.99	53.36	56.49	61.09	1.01	1.06	1.14
	19	53.14	52.31	54.90	56.83	1.02	1.05	1.09
	20	52.64	51.75	53.07	54.40	1.02	1.03	1.05
zero	16	35.05	34.55	35.43	35.43	1.01	1.03	1.03
	17	34.02	32.96	34.84	34.84	1.03	1.06	1.06
	18	33.82	32.38	34.12	35.28	1.04	1.05	1.09
	19	33.82	32.11	34.03	34.49	1.05	1.06	1.07
	20	33.82	31.94	33.70	34.03	1.06	1.06	1.07

Table 3 Total switching costs under timing constraints "32C-l0".

prob	LV	ND	SW	LO	DO	ND/SW	LO/SW	DO/SW
uni	13	57.75	57.25	58.40	58.40	1.01	1.02	1.02
	14	57.00	55.41	57.13	57.13	1.03	1.03	1.03
	15	56.44	54.52	56.43	56.43	1.04	1.04	1.04
	16	55.83	53.70	55.62	55.62	1.04	1.04	1.04
	17	55.12	52.86	54.92	54.92	1.04	1.04	1.04
zero	13	38.45	36.03	38.60	38.60	1.07	1.07	1.07
	14	38.29	34.56	38.46	38.46	1.11	1.11	1.11
	15	38.27	33.32	38.33	38.33	1.15	1.15	1.15
	16	38.19	32.37	37.99	37.99	1.18	1.17	1.17
	17	37.98	32.10	37.78	37.78	1.18	1.18	1.18

Table 4 Total switching costs under timing constraints "32C-u0".

prob	LV	ND	SW	LO	DO	ND/SW	LO/SW	DO/SW
uni	13	63.77	63.54	63.77	63.77	1.00	1.00	1.00
	14	60.32	60.16	60.76	62.59	1.00	1.01	1.04
	15	59.73	58.92	59.98	59.98	1.01	1.02	1.02
	16	58.88	57.91	59.34	60.05	1.02	1.02	1.04
	17	58.38	56.87	58.84	58.84	1.03	1.03	1.03
zero	13	41.11	40.54	41.11	41.11	1.01	1.01	1.01
	14	39.63	38.79	40.09	40.93	1.02	1.03	1.05
	15	38.66	37.35	39.78	39.78	1.04	1.06	1.06
	16	38.66	36.69	39.62	40.33	1.05	1.08	1.10
	17	38.66	35.64	39.62	39.62	1.08	1.11	1.11

Table 5 The number of nodes of resulting prefix graphs (32C-l0).

prob	LV	ND	SW	LO	DO
uni	13	49	50	50	50
	14	48	48	48	48
	15	47	47	47	47
	16	46	46	46	46
	17	45	45	45	45
zero	13	49	61	50	50
	14	48	64	48	48
	15	47	58	47	47
	16	46	56	46	46
	17	45	51	45	45

where timing constraints seem to be loose since upper bits require the results of lower bits so it is preferable that arrival levels of lower bits are earlier than those of upper bits. So, as in the case of late output required levels, relaxed timing constraints may give opportunities to reduce switching costs.

As for input signal probabilities, our approach is more effective where there are many low-cost nodes. Since the total switching cost is the summation of the costs of all nodes, area minimization algorithm also leads to reduction of the switching cost. There is high correlation between the number of nodes and the total switching cost especially when there is less difference among the switching cost of each node. Meanwhile, in case that there are many nodes with very low costs, such as "zero" case, reducing area might not lead to reduction of the total switching cost as shown in Table 3 and **Table 5**, where the numbers of nodes in SW are larger than those in ND in several cases, though the switching costs are lower. So, the correlation might become lower, which means direct minimization of the total switching cost could be superior.

6. Conclusions

In this paper, our area minimization framework for prefix graph synthesis was expanded to treat switching cost minimization, and improved to integrate estimation of restructuring effects for robustness. Relation between various types of signal probability and switching costs of nodes are also surveyed through several experiments. Switching-cost minimization algorithm is effective especially where there are a comparably large number of nodes with very low switching costs and timing constraints are not tight.

The effectiveness of our switching-cost minimization algorithm depends on features of signal probabilities and timing constraints. It is important to carefully

observe the effectiveness under distinct situations.

Our switching cost is a rather rough measure, so more accurate measures should be considered. In those cases, our framework can be applied as long as the target cost to be minimized can be calculated as the summation of all node costs, and similar observation on effectiveness as mentioned above may be applicable.

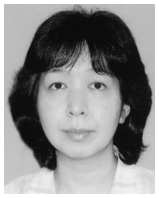
Acknowledgments This research was supported in part by Waseda University Global COE Program "International Research and Education Center for Ambient SoC" sponsored by MEXT, Japan, and JST CREST Project.

References

1) Brent, R.P. and Kung, H.T.: A Regular Layout for Parallel Adders, *IEEE Trans. Computers*, Vol.31, No.3, pp.260–264 (1982).
2) Bryant, R.: Graph-based Algorithms for Boolean Function Manipulation, *IEEE Trans. Computers*, Vol.35, No.8, pp.677–691 (1986).
3) Chen, B. and Nedelchev, I.: Power Compiler: A Gate Level Power Optimization and Synthesis System, *IEEE International Conference on Computer Design*, pp.74–79 (1997).
4) Coburn, J., Ravi, S. and Raghunathan, A.: Power Emulation: A New Paradigm for Power Estimation, *DAC*, pp.700–705 (2005).
5) Iman, S. and Pedram, M.: Multi-Level Network Optimization for Low Power, *IEEE International Conference on Computer Aided Design*, pp.372–377 (1994).
6) Kogge, P.M. and Stone, H.S.: A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations, *IEEE Trans. Computers*, Vol.22, No.8, pp.786–793 (1973).
7) Koren, I.: *Computer Arithmetic Algorithms*, A K Peters, Ltd. (2002).
8) Liu, J., Zhou, S., Zhu, H. and Cheng, C.-K.: An Algorithmic Approach for Generic Parallel Adders, *ICCAD*, pp.734–730 (2003).
9) Liu, J., Zhu, Y., Zhu, H., Cheng, C.-K. and Lillis, J.: Optimum Prefix Adders in a Comprehensive Area, Timing and Power Design Space, *ASPDAC*, January, pp.609–615 (2007).
10) Matsunaga, T., Kimura, S., and Matsunaga, Y.: Synthesis of Parallel Prefix Adders Considering Switching Activities, *IEEE International Conference on Computer Design*, pp.404–409 (2008).
11) Matsunaga, T. and Matsunaga, Y.: Timing-Constrained Area Minimization Algorithm for Parallel Prefix Adders, *IEICE Trans. Fundamentals*, Vol.E90-A, No.12, pp.2770–2777 (2007).
12) Pedram, M.: Power Minimization in IC Design: Principles and Applications, *ACM Trans. Design Automation of Electronic Systems*, Vol.1, No.1, pp.3–56 (1996).
13) Sklansky, J.: Conditional Sum Addition Logic, *IRE Trans. Electron. Comput.*, Vol.9, No.6, pp.226–231 (1960).
14) Zimmermann, R.: Non-Heuristic Optimization and Synthesis of Parallel-Prefix Adders, *International Workshop on Logic and Architecture Synthesis*, pp.123–132 (1996).

(Received November 17, 2008)
(Revised February 20, 2009)
(Accepted April 13, 2009)
(Released August 14, 2009)

(Recommended by Associate Editor: *Hiroshi Saito*)



Taeko Matsunaga received Bachelor of Liberal Arts in Pure and Applied Sciences from the University of Tokyo in 1982. She joined Fujitsu Laboratories in Kawasaki, Japan, in 1982 and she had been involved in research and development of the CAD for digital systems. In 2001, she joined Logic Research Co. Ltd. in Fukuoka, Japan, then moved to Institute of Systems & Information Technologies/Kyushu as an industrial researcher. In November 2003, she joined Fukuoka Laboratory for Emerging & Enabling Technology of SoC (FLEETS), and had been engaged in EDA projects until March 2007. She is currently working toward Ph.D. degree at Graduate School of Information, Production and Systems, Waseda University. Her research interests include arithmetic synthesis, logic synthesis, and high level synthesis. She is a member of IEEE, ACM and IPSJ.

Shinji Kimura received the B.E., M.E. and Dr.of Eng. degrees in information science from Kyoto University, Kyoto, Japan in 1982, 1984, and 1989, respectively. He was an assistant professor at Kobe University from 1985, was an associate professor at Nara Institute of Science and Technology from 1993, and has been a professor of Waseda University since 2002. He was a visiting scientist at Carnegie Mellon University from 1989 to 1990, and was a visiting scholar of Stanford University from 2000 to 2001. He is interested in the formal and timing verification of logic circuits, the hardware/software codesign methodologies, reconfigurable hardware, and the low-power design. He is a member of the Information Processing Society of Japan and the IEEE Computer Society.

Yusuke Matsunaga received the B.E., M.E. and Ph.D. degrees in Electronics and Communications Engineering from Waseda University, Tokyo, Japan, in 1985, 1987, and 1997, respectively. He joined Fujitsu Laboratories in Kawasaki, Japan, in 1987 and he had been involved in research and development of the CAD for digital systems. From October 1991 to November 1992, he was a visiting Industrial Fellow at the University of California, Berkeley, in the department of Electrical Engineering and Computer Sciences. In 2001, he joined the faculty at Kyushu University. He is currently an associate professor of Department of Computer Science and Communication Engineering. His research interests include logic synthesis, formal verification, high-level synthesis and automatic test patterns generations. He is a member of IEEE, ACM and IPSJ.