Pulser Gating: A Clock Gating of Pulsed-Latch Circuits

Sangmin Kim, Inhak Han, Seungwhun Paik, and Youngsoo Shin
Department of Electrical Engineering, KAIST
Daejeon 305-701, Korea

***Abstract*— A pulsed-latch is an ideal sequencing element for low-power ASIC designs due to its smaller capacitance and simple timing model. Clock gating of pulsed-latch circuits can be realized by gating a pulse generator (or pulser), which we call pulser gating. The problem of pulser gating synthesis is formulated for the first time. Given a gate-level netlist with location of latches, we first extract the gating function of each latch; the gating functions are merged to reduce the amount of extra logic while gating probability is not sacrificed too much. We also have to take account of proximity of latches, because a pulser, which is gated by merged gating function, and its latches have to be physically close for safe delivery of pulse. The heuristic algorithm that considers all three factors (similarity of gating functions, literal count to implement gating functions, and proximity of latches) is proposed and assessed in terms of power saving and area using 45-nm technology.**

I. Introduction

A pulsed-latch is a latch driven by a brief clock pulse. The amount of time borrowing that can be exploited is necessarily very small, and is typically ignored in ASIC design to simplify the timing model. Consequently, a pulsed-latch can be approximated as a faster and smaller flip-flop, allowing pulsed-latch ASIC circuits to be designed with standard CAD tools. The major challenge is the generation and delivery of pulse. A normal clock of 50% duty ratio is delivered from a clock source to multiple pulse generators (called pulsers); each pulser [1], [2] then delivers a pulse to more than one latches. Since pulses can easily be distorted, a pulser and its latches have to be physically close to preserve the pulse shape, imposing a constraint on their placement [3].

A simple replacement of flip-flops with pulsed-latches can save appreciable amount of power consumption due to smaller capacitance [4]. This is illustrated in Fig. 1 using industrial 45-nm technology. In the first three circuits, which are FSM controllers, the power consumption (including both switching and leakage) of flip-flops is greatly reduced after replacing them with latches. The pulsers now take a large proportion of total power, but the power consumption from pulsers and latches together is still smaller than that of flip-flops, justifying the replacement. The saving may become smaller when a circuit is dominated by combinational gates such as the last two circuits, which are data-path circuits. Note that the power consumption from combinational gates could also be reduced if increased timing slack after replacement is utilized via gate sizing or re-synthesis, which is not reflected in Fig. 1.

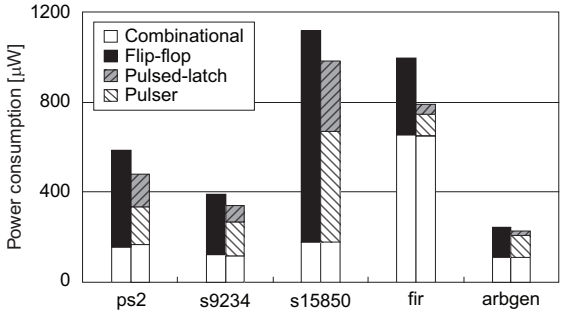

Fig. 1. Power consumption of flip-flop circuits (left bars) and pulsed-latch circuits (right bars).

A. Motivation and Problem Statement

To further reduce power consumption of pulsed-latch circuits, we may consider applying clock gating, which has become a usual practice. The standard clock gating, i.e. clock gating of flip-flop circuits, is illustrated in Fig. 2(a). A gating function determines when a clock is delivered to a group of flip-flops (EN=1) and when it is not (EN=0). A potential glitch from the gating function is removed by using a latch; the AND gate and latch together are conveniently called a clock gating cell. A *clock gating synthesis* derives a list of gating functions such that flip-flops are gated as often as possible, while the amount of extra logic to implement the functions are kept as small as possible. Typical approach to synthesis is to extract a gating function of each individual flip-flop and gradually merge the functions; the key challenge during merge is to find the functions that are similar.

The clock gating of pulsed-latch circuits is shown in Fig. 2(b). The pulser can be typically designed in a way that clock gating capability is embedded. In this setting, clock gating is implemented via pulsers instead of clock gating cells, which shall be called pulser gating. There is a unique challenge in *pulser gating synthesis*: while we find the gating functions to merge, the functions themselves have to be similar (so that they generate the same EN signal as often as possible) as well as their corresponding latches are physically close. The number of pulsers should be minimized in this process because of large proportion of power consumption from pulsers (see Fig. 1).

In our approach to pulser gating synthesis presented in Section III, we receive a gate-level netlist with location of pulsed-latches, which is obtained after initial placement. We

978-1-4244-7514-8/11/$26.00 ©2011 IEEE

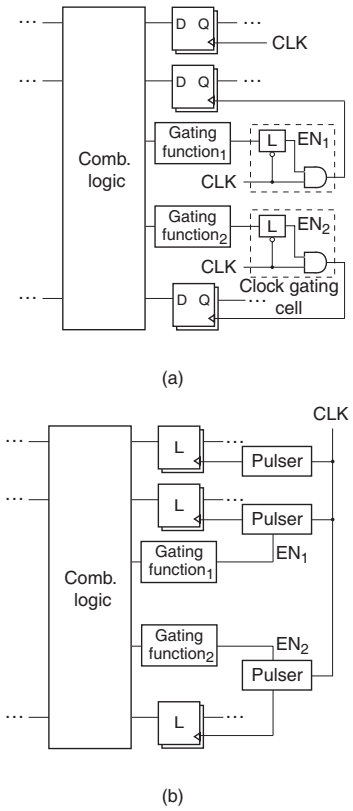

Fig. 2. (a) Clock gating and (b) pulser gating.

want to synthesize gating functions and determine groups of latches with each group driven by a pulser. The objective of synthesis is two-fold: one is to minimize the number of pulsers under load capacitance limit of pulser, and the other is to minimize the extra logic to implement gating functions while we maximize the probability of each gating function to be evaluated to 1 (i.e. EN=0).

B. Organization

In Section II-A, we review the concept of clock gating and its implementation aspect such as extraction of gating function, computation of gating probability, and minimizing the extra logic for gating function. We formulate the problem of pulser gating synthesis in Section II-B, which is one of our contributions. A heuristic algorithm of pulser gating synthesis is addressed in Section III, which constitutes another contribution; the concept of similarity of gating functions and a measure of merit are introduced to develop the algorithm. In Section IV, we assess the circuits generated by the algorithm in terms of average power consumption and area using industrial 45-nm technology. We summarize this paper and look at potential future work in Section V.

II. Problem Formulation

The synthesis of clock gating functions can be performed during FSM design [5] or after gate-level netlist is determined [6], where the latter is our focus.

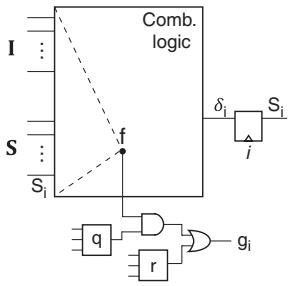

Fig. 3. Computation of a gating function g_i.

A. Clock Gating

The synthesis of gating functions starts from identifying a gating function of individual register (flip-flop or latch). The gating function g_i of a register i is given by

$$g_i = \overline{\delta_i(\mathbf{I}, \mathbf{S}) \oplus S_i}, \tag{1}$$

where δ_i is a next-state function which is a function of circuit inputs $\mathbf{I}$ and present states $\mathbf{S}$, and $S_i \in \mathbf{S}$ denotes a present state bit. In other words, δ_i and S_i correspond to the input and output of i as shown in Fig. 3; when two are the same, there is no need to load the value of δ_i and thus clock can be gated through $g_i = 1$.

For each g_i, we compute the probability that it is evaluated to 1, $P(g_i = 1) = P(g_i)$. This can be done by propagating the signal probability [7] of each circuit input bit of $\mathbf{I}$, which is given by designers, and that of each present state bit of $\mathbf{S}$ through g_i. Note that the signal probability of a present state bit is not given; it can be derived by such method as Picard-Peano iteration [8], which we implemented for the experiment. Clearly, a larger value of $P(g_i)$ is preferred so that clock is gated as often as possible.

We also have to take account of extra logic gates to implement g_i. To reduce the amount of extra gates, the existing combinational logic can be utilized. Let f be a Boolean expression at an internal node of combinational logic. After we perform algebraic division [9], we get $g_i = fq + r$, where q is a quotient and r is a remainder expression; only q and r are implemented as illustrated in Fig. 3. We thus want to determine f that yields the minimum implementation of q and r; the implementation cost of q and r can be assessed by their total literal count once they are represented in a factored form [9]. To determine such f is not a trivial task due to large number of nodes and the complexity of Boolean manipulation (division and factoring). We will consider a heuristic approach in Section III. Another method to reduce the cost of g_i is approximation [6]. Any element of on-set of g_i can be safely moved to off-set; clock is simply not gated (g_i=0) when it can be. The problem is to select on-set elements in a way that the cost of g_i is minimized while $P(g_i)$ is not sacrificed too much. Approximating g_i is not considered in our approach of pulser gating synthesis; it is left for future investigation.

Since g_i comes at a cost of extra gates, we may want to merge g_is so that they can share the cost. If we merge g_i and

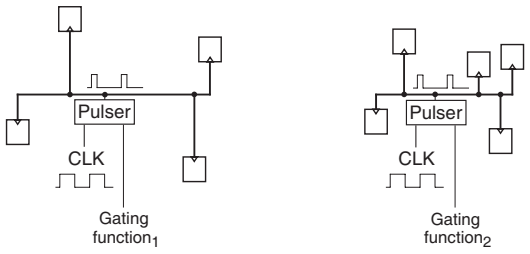

Fig. 4. Steiner tree connection of a group of latches.

g_j, the new gating function becomes $g_i \wedge g_j$; its probability $P(g_i \wedge g_j)$ does not exceed $\min(P(g_i), P(g_j))$, i.e. i and j can be gated only when both can be gated. Therefore, i and j should be selected such that $P(g_i \wedge g_j)$ is kept as high as possible and the number of extra literals to implement $g_i \wedge g_j$ becomes as small as possible. When more than one gating functions are merged, a cluster of corresponding registers is denoted by $\mathbb{C}$ and the merged gating function by G, i.e. $G = \wedge_{i \in \mathbb{C}} g_i$. Its probability is denoted by $P(G)$ and the extra literal count to implement it is denoted by $L(G)$.

B. Pulser Gating Synthesis

For safe delivery of pulse from a pulser to its latches, the load capacitance limit of pulser C_{max} has to be respected. The load capacitance of pulser that drives latches in $\mathbb{C}$, denoted by $C_L(\mathbb{C})$, consists of fanout capacitance and wire capacitance:

$$C_L(\mathbb{C}) = C_{latch} \cdot |\mathbb{C}| + C_{wire} \cdot W(\mathbb{C}), \tag{2}$$

where C_{latch} is clock input capacitance of a latch, C_{wire} is the capacitance per unit length of wire, and $W(\mathbb{C})$ corresponds to wirelength connecting all the latches using Steiner tree, HPWL (half-perimeter bounding box wirelength), and so on[1]. Clearly, the number of latches that a single pulser can drive becomes different depending on wire capacitance, which in turn is determined by how latches are located as illustrated in Fig. 4.

We now state the problem of pulser gating synthesis:

Problem 1 Given a set of latches, each one with its gating function and physical location, the pulser gating synthesis is to derive a set of clusters $\mathbb{C}_i$ of latches, where each cluster is driven by a single pulser. The objective is to maximize $\sum_i P(G_i)$ while we minimize $\sum_i L(G_i)$, such that $C_L(\mathbb{C}_i) \leq C_{max}, \forall i$.

III. Synthesis of Pulser Gating

A. Overview

To maximize $P(G_i)$, we should group latches that can be gated together at the same clock cycle as much as possible. To this end, we define a similarity measure between two gating functions g_i and g_j:

$$S(g_i, g_j) \triangleq \frac{P(g_i \wedge g_j)}{P(g_i \vee g_j)}. \tag{3}$$

Clearly $0 \leq S \leq 1$. Both numerator and denominator can be readily computed. While we propagate the signal probability of input bits of $\mathbf{I}$ and that of present state bits of $\mathbf{S}$ to compute $P(g_i)$ and $P(g_j)$, we temporarily insert AND and OR gates with their inputs being g_i, g_j; the probability at the gate outputs yield the value of numerator and denominator, respectively.

Since we gradually build the clusters of latches during the algorithm in Section III-B, we may merge two existing clusters $\mathbb{C}_i$ and $\mathbb{C}_j$ provided that the merge does not violate C_{max}, i.e. we try merging two candidate clusters to derive a new cluster $\mathbb{C}$, obtain a Steiner tree to compute $W(\mathbb{C})$, and compute $C_L(\mathbb{C})$ from (2) which is then compared to C_{max}. The selection of candidates for merge can be based on $S(G_i, G_j)$; for the sake of computational complexity, we approximate $S(G_i, G_j)$ by the average similarity between members of $\mathbb{C}_i$ and $\mathbb{C}_j$:

$$S(G_i, G_j) \triangleq \underset{x \in \mathbb{C}_i, y \in \mathbb{C}_j}{\text{AVG}} S(g_x, g_y). \tag{4}$$

This is convenient to compute because $S(g_x, g_y)$s are already available from (3).

To decide whether we execute the merge of candidate clusters, we assess if the merge really helps. This should be based on the two objectives of Problem 1, namely maximizing $\sum_i P(G_i)$ and minimizing $\sum_i L(G_i)$. We introduce a measure of merit for this purpose:

$$\mathcal{M} \triangleq \sum_i \left(|\mathbb{C}_i| \underset{\substack{x,y \in \mathbb{C}_i \\ x \neq y}}{\text{AVG}} S(g_x, g_y) - \alpha L(G_i) \right). \tag{5}$$

The AVG term corresponds to average similarity between member latches x and y of a cluster $\mathbb{C}_i$, which is well correlated with $P(G_i)$. This is multiplied by the number of latches of a cluster, since we want to group as many similar latches as possible, which effectively helps reduce the number of pulsers. The literal count $L(G_i)$ corresponds to the total literal count of q and r after G_i is divided by f (see Fig. 3), i.e. $G_i = fq + r$. To reduce the complexity of finding f that minimizes $L(G_i)$, we only try the nodes in the fanin cone of the latches of G_i. The weighting factor α is used to balance the two terms, whose value is determined in empirical way.

B. Algorithm

The algorithm *Pulser_Gating_Synthesis* is shown in Fig. 5. In L1, a gating function g_i of individual latch is derived using (1); in L2, this is used to calculate a similarity between each pair of latches $S(g_i, g_j)$ using (3). We create a list of clusters, where each cluster initially contains only one latch (L4).

We introduce a similarity graph $G(V, E, a, s)$ (L3) to capture the similarity information. Each vertex $i \in V$ corresponds to a latch and $e_{ij} \in E$ between i and j has a similarity

[1]We assume that a pulser is located right on the Steiner tree as shown in Fig. 4. We ignore the details of wiring, such as via capacitance and different metal layers, when we build clusters of latches.

Algorithm *Pulser_Gating_Synthesis*

L1 Find a gating function g_i of each latch i
L2 Calculate $S(g_i, g_j)$ of each pair of latches i and j
L3 Create a similarity graph $G(V, E, a, s)$
L4 Create a cluster $\mathbb{C}_i = \{i\}$ for each latch i
L5 **while** $E \neq \emptyset$ **do**
L6 Select a vertex i of maximum attraction
L7 Select a vertex j of maximum similarity with i
L8 **if** $\mathbb{C}_i \cup \mathbb{C}_j$ respects C_{max} and improves $\mathcal{M}$ **then**
L9 $\mathbb{C}_i \leftarrow \mathbb{C}_i \cup \mathbb{C}_j$
L10 Merge i and j into i, and update G
L11 **else** $E \leftarrow E \setminus \{e_{ij}\}$
L12 **for** each cluster $\mathbb{C}_i$ **do**
L13 Assess its power consumption
L14 Drop i from pulser gating if power is not saved
L15 Group the remaining latches, connect to normal pulsers

Fig. 5. *Pulser_Gating_Synthesis* algorithm.

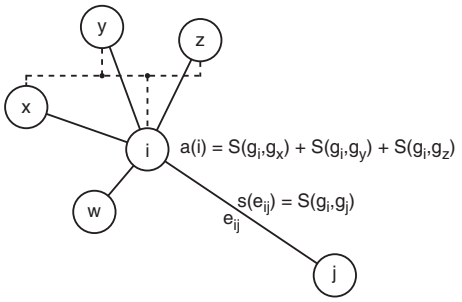

Fig. 6. A similarity graph: edge weight $s(e_{ij})$ and attraction $a(i)$.

value $S(g_i, g_j)$ as its weight $s(e_{ij})$, i.e. $s : e_{ij} \rightarrow S(g_i, g_j)$. This is illustrated in Fig. 6. If merging i and j violates C_{max}, which can be checked by using (2), e_{ij} is dropped. We assign a value of attraction to each vertex, $a(i)$. This is done as follows with Fig. 6 as an example. The vertices that are adjacent to i are sorted in decreasing order of similarity, say in order of x, y, z, w, and j; we determine how many vertices in this order can be merged with i while C_{max} is not violated, which is checked by estimating wirelength and using (2), assume x, y, and z for such vertices; the sum of similarities with these vertices constitutes attraction, i.e. $a(i) = S(g_i, g_x) + S(g_i, g_y) + S(g_i, g_z)$.

To build a cluster of latches that share the same gating function and are thus driven by the same pulser (see Fig. 4), we select a vertex i of maximum attraction (L6). We then consider its adjacent vertex j with maximum similarity (L7). If merging i and j into a single cluster does not violate C_{max} and improves the merit $\mathcal{M}$ (L8), the merging is executed (L9). This is then reflected in G as illustrated in Fig. 7. The vertices that have edges with both i and j now have an edge with merged vertex, which is denoted by i' in Fig. 7(b); x, y, and w are such vertices. The edge between z and j is removed after merging because there is no edge between i and z. The edge weight is updated as shown in Fig. 7(b), which may cause the update of attraction of x, y, and w as well as z.

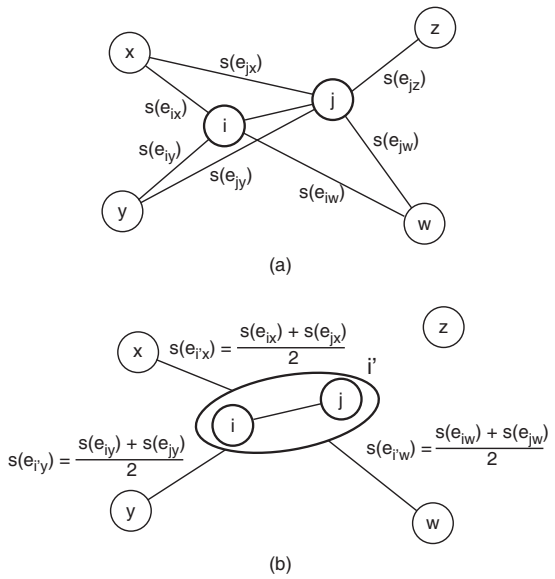

Fig. 7. (a) Before merging and (b) after merging i and j.

The attraction of merged vertex $a(i')$ is also updated; when we compute the Steiner tree of i' and its adjacent vertices, the connection between i and j is always included because they are now treated as a single vertex. If i and j cannot be merged (L11), the edge between them is removed. If we repeat the aforementioned process (L5), all the edges are eventually removed and we are left with a list of clusters of latches.

Since clusters are made on the basis of merit $\mathcal{M}$, some clusters may contribute to power saving while others may not. Therefore, each cluster $\mathbb{C}_i$ is assessed in terms of power saving (L13). For this purpose, a netlist of extra gates to implement G_i (q and r; see Fig. 3) is obtained via technology mapping; its switching power consumption is derived from load capacitance of each gate and switching activity, which we assumed to be 5% in our experiment. The amount of power that can be saved in the clock network is obtained by multiplying gating probability $P(G_i)$ with power consumption of a pulser and latches in $\mathbb{C}_i$. The two power numbers are subtracted; if the result is positive (power consumption increases rather than decreases due to the extra combinational logic), pulser gating is not performed on $\mathbb{C}_i$ (L14); a netlist to implement G_i is removed and the latches of $\mathbb{C}_i$ are un-grouped.

The remaining latches, which are not pulser-gated, should be connected to normal pulsers (L15). We rely on a simple heuristic similar to finding a minimum spanning tree. Fig. 8(a) shows a graph, where each vertex is a latch and edge weight corresponds to Manhattan distance between two latches; the edge that violates C_{max} constraint is dropped, e.g. the distance between a and e is larger than 5. After we select two edges of minimum length of 1, a group of a, b, and c are identified. The three vertices and their edges are removed from the graph, and we continue to identify another group of latches, d, e, and f.

TABLE I

RESULT OF *Pulser_Gating_Synthesis*: THE INCREASE IN THE NUMBER OF PULSERS (ΔPULSERS), THE NUMBER OF EXTRA GATES TO IMPLEMENT GATING FUNCTIONS (# EXTRA GATES), PERCENTAGE OF LATCHES THAT ARE GATED (% GATED LATCHES), AVERAGE GATING PROBABILITY OF ALL LATCHES ($\text{AVG}_i P(G_i)$), AND RUNTIME OF SYNTHESIS ALGORITHM

Name	# Gates	# Latches	# Pulsers	*Pulser_Gating_Synthesis*				
				Δ Pulsers	# Extra gates	% Gated latches	$\text{AVG}_i P(G_i)$	Runtime (min.)
s838	351	32	6	2	0	65.6	0.63	0.3
s1423	1191	74	15	1	24	4.1	0.04	1.3
s5378	1781	160	30	5	417	23.8	0.22	16.6
s9234	1293	125	22	7	276	28.0	0.23	6.8
b04	833	66	13	1	17	7.6	0.07	2.3
b07	431	44	8	1	31	20.5	0.18	0.3
b12	1395	119	21	7	317	31.1	0.29	3.1
i2c	1125	128	22	6	475	39.8	0.30	4.9
pci_ctrl	879	60	12	3	90	51.7	0.51	1.3
sasc	1058	116	21	5	179	30.2	0.26	2.5

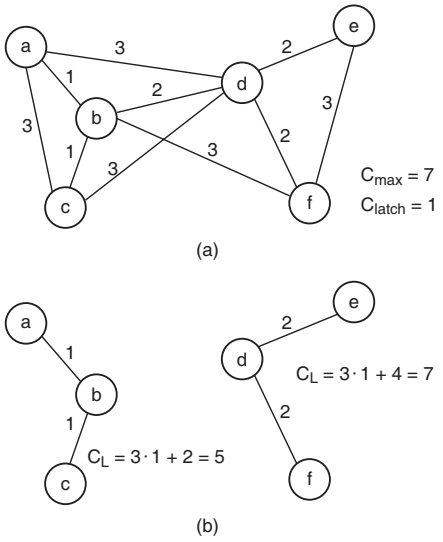

Fig. 8. (a) Latches are denoted by vertices and Manhattan distance between latches by edge weight, and (b) derived clusters of latches.

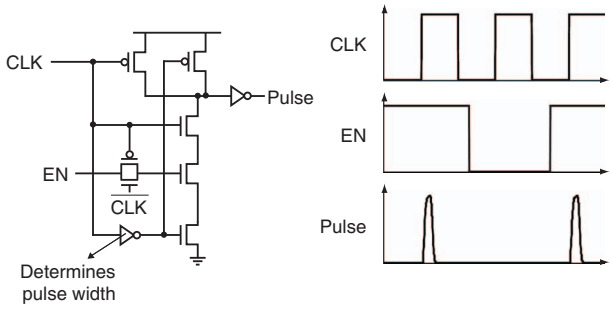

Fig. 9. A pulser [2] and its SPICE waveform.

IV. EXPERIMENTAL RESULTS

We carried out experiments on a set of sequential circuits taken from the ISCAS and ITC benchmarks. The circuits extracted from several OpenCores [10] were also used for the experiments; they include i2c, pci_ctrl, and sasc. The test circuits are listed in the first three columns of Table I. Each circuit was synthesized with commercial logic synthesis tool [11] using an industrial 1.1 V, 45-nm technology library. Initial placement was performed using commercial physical design tool [12] to obtain the location of latches; we forced about 70% of the placement region to be occupied by the cells so that the extra gates after pulser gating synthesis can be accommodated during incremental placement. A gate-level netlist together with a DEF file that contains the location of latches are given to *Pulser_Gating_Synthesis*, which was implemented in SIS [13]. A fast SPICE simulator [14] was used to measure the power consumption.

A variety of pulsers have been proposed [1], [2]; we used the one shown in Fig. 9 because of its low power consumption. It was designed to generate a pulse of 110 ps wide, with the load capacitance limit of 10 fF and slew constraint of 40 ps, which was found to be the upper bound at which the safe latching of data was ensured at latches. The SPICE waveforms are also shown in Fig. 9, which shows how gating is performed by a gating function output `EN`. The transmission gate is responsible for filtering out any glitches from `EN` while `CLK`=1.

The number of pulsers in the fourth column of Table I was determined by the method illustrated in Fig. 8, i.e. a heuristic similar to finding a minimum spanning tree. It represents the number of pulsers in the original circuit, where clock gating is not performed. The last five columns of Table I report the result after running *Pulser_Gating_Synthesis*. Fig. 10 shows the power consumption of circuits it generates (right bars), which is normalized to that of initial circuits (left bars), which are not gated.

It can be readily seen that the power saving in Fig. 10 is largely determined by average gating probability $\text{AVG}_i P(G_i)$ shown in Table I. The circuits s838 and pci_ctrl have higher probability, which yields large power saving; the clock is hardly gated in s1423 and b04, which results in almost no saving in power, as it should.

Fig. 10 also indicates that power saving mainly comes from pulsers, which is a understandable consequence of their large load capacitance. The increase in the number of pulsers

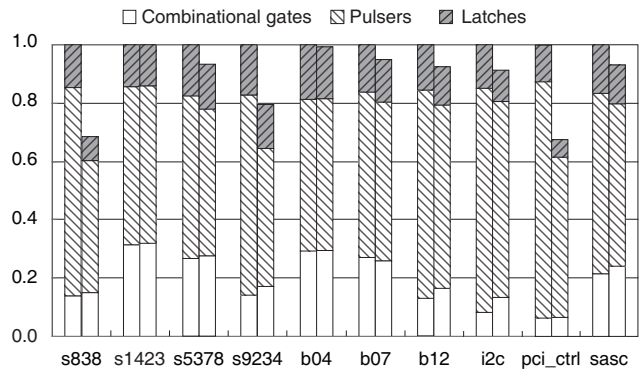

Fig. 10. Power consumption of initial circuits (left bars) and that of circuits after *Pulser_Gating_Synthesis* (right bars).

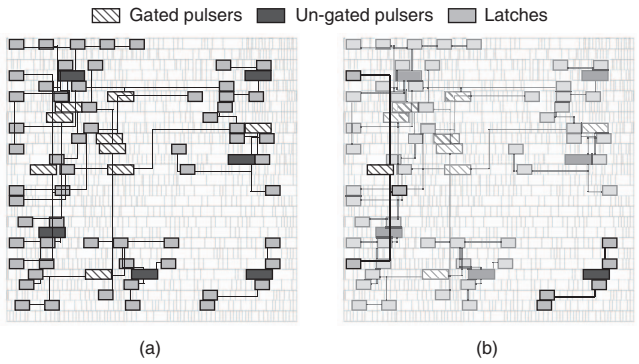

Fig. 11. (a) A layout of circuit pci_ctrl and (b) a sample of gated pulser with its latches is compared to un-gated pulser.

(Δ Pulsers) is relatively large in such circuits as s9234, b12, and i2c; the latches that have larger similarity value are not localized in these circuits, which calls for more pulsers. Adjusting the placement may yield better clustering of latches and less number of pulsers, which merits future investigation. The runtime of the synthesis algorithm is reported in the last column of Table I.

Compared to un-gated circuits, the area, which is the sum of the areas of all the cells in each design, increases by 11% on average. This is largely due to the extra gates to implement gating functions (column 6 of Table I), even though their contribution to the increase of power consumption is very small due to their low switching activity (around 5%). Fig. 11(a) shows a layout of circuit pci_ctrl after performing *Pulser_Gating_Synthesis*. Gated pulsers are marked using diagonal patterns, un-gated pulsers in black, and latches in gray. The local clock network, which delivers a clock pulse from a pulser to the latches driven by it, is also shown. In Fig. 11(b), a sample of gated pulser that drives 3 latches is highlighted; another sample of un-gated pulser that drives 5 latches is also highlighted for comparison. The long distance among latches with high similarity makes the gated pulser include a smaller number of latches compared to the un-gated pulser.

V. Conclusion

The problem of pulser gating synthesis has been formulated. Gating functions of latches are merged to reduce the amount of extra logic while gating probability is not sacrificed too much, which is also the objective of conventional clock gating synthesis. Only the latches that are physically close can be the candidates of this merge in pulser gating synthesis, which makes the problem challenging. The heuristic algorithm, which considers the similarity of gating functions and extra literals for their implementation, has been proposed to solve this new problem.

Deriving a gating function of individual latch can be improved in several directions, e.g. by detecting and using ODC (observability don't cares) and by approximating the function. Each pulser is controlled by its own gating function to simplify the problem; sharing a gating function between more than one pulsers will help reduce the number of extra gates. Combined pulser gating synthesis and placement will be an ideal strategy; how to manage the complexity of both will be a key to the integrated approach.

Acknowledgment

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD, Basic Research Promotion Fund) (KRF-2008-331-D00406).

References

[1] S. Kozu *et al.*, "A 100 MHz 0.4W RISC processor with 200 MHz multiply-adder, using pulse-register technique," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 1996, pp. 140–141.
[2] S. Naffziger *et al.*, "The implementation of the Itanium 2 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1448–1460, Nov. 2002.
[3] Y. Chuang, S. Kim, Y. Shin, and Y. Chang, "Pulsed-latch aware placement for timing-integrity optimization," in *Proc. Design Automation Conf.*, June 2010, pp. 280–285.
[4] S. Shibatani and A. Li, "Pulse-latch approach reduces dynamic power," July 2006, EE Times.
[5] L. Benini and G. De Micheli, "Automatic synthesis of low-power gated-clock finite-state machines," *IEEE Trans. on Computer-Aided Design*, vol. 15, no. 6, pp. 630–643, June 1996.
[6] E. Arbel, C. Eisner, and O. Rokhlenko, "Resurrecting infeasible clock-gating functions," in *Proc. Design Automation Conf.*, July 2009, pp. 160–165.
[7] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Riccó, "Estimate of signal probability in combinational logic networks," in *Proc. European Test Conf.*, Apr. 1989, pp. 132–138.
[8] C. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M. Despain, and B. Lin, "Power estimation methods for sequential logic circuits,," *IEEE Trans. on VLSI Systems*, vol. 3, no. 3, pp. 404–416, Sept. 1995.
[9] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. on Computer-Aided Design*, vol. 6, no. 6, pp. 1062–1081, Nov. 1987.
[10] OpenCores. [Online]. Available: http://www.opencores.org/
[11] Synopsys, "Design Compiler User Guide," Sept. 2008.
[12] ——, "IC Compiler User Guide," Dec. 2008.
[13] E. Sentovich *et al.*, "SIS: a system for sequential circuit synthesis," May 1992, Tech. Rep. UCB/ERL M92/41.
[14] Synopsys, "NanoSim User Guide," Sept. 2008.