Clock Gating Synthesis of Pulsed-Latch Circuits

Seungwhun Paik, Inhak Han, Sangmin Kim, and Youngsoo Shin, *Senior Member, IEEE*

***Abstract*—Pulsed-latch circuits, in which latches are triggered by a short pulse, can reduce power consumption as well as increasing performance; and they can largely be designed using conventional computer-aided design tools. We explore the automatic synthesis of clock-gating logic for pulsed-latch circuits in which gating is implemented by enabling and disabling several pulse generators. The key problem is to arrange that each group of latches contains physically close latches, so that a short pulse from a pulse generator is delivered safely, and to ensure that the latches in a group have similar Boolean gating conditions because their clock is gated and ungated together. The resulting gating conditions should be implemented using as little extra logic as possible; for this purpose we rely on Boolean division, with an internal node of existing logic being used as the divisor. The proposed clock gating synthesis is assessed in 45-nm technology.**

***Index Terms*—Clock gating, gating function, pulse generator, pulsed-latch.**

I. Introduction

A PULSED-LATCH is a latch driven by a brief clock pulse. The amount of time borrowing, determined by the pulse width, is very small, thereby allowing a pulsed-latch to act as a small, fast flip-flop. The driving pulses come from a pulse generator [2], [3], which we will call a pulser; this receives a normal clock signal with a 50% duty ratio and outputs a clock pulse. Since that pulse is susceptible to distortion, a pulser can only drive a small number of nearby latches; thus a pulser and its latches need to be clustered, an issue which we address later in this paper. Alternatively, a pulser and a latch may be integrated into a single component [4], called a pulsed-flip-flop.

The power benefit of a pulsed-latch circuit is apparent from Fig. 1, which shows the result of an experiment that we performed in 45 nm technology: it demonstrates how the power consumption of sequencing elements is greatly reduced when pulsed-latches are used instead of flip-flops. (Fig. 1 also shows that pulsed-flip-flop circuits provide a somewhat lesser benefit.) Notice that pulsers use a substantial amount of power, and so their use should be minimized, which is an important observation that motivates the work reported in this paper.

Clock gating is a standard way of reducing clocking power. It can be applied either at the register transfer level (RTL) or at the gate level. Using the former approach, registers that are amenable to clock gating, such as load-enabled registers, are identified and clock gating can then be applied fairly simply [5]; but this procedure makes the designer responsible for specifying the candidate registers for clock gating. With the latter approach, the logic that identifies the condition under which a clock can be gated, called the gating logic or gating function, is automatically synthesized [6]–[8]. Gate-level clock gating is more general and can complement the RTL technique, since it can be applied to any registers, including these which are not load-enabled. An important problem in gate-level clock gating is to identify groups of registers within which the registers of each group can be gated together.

A. Motivation and Problem Statement

In pulsed-latch circuits, clock gating is realized by disabling pulsers, without introducing any extra circuitry such as clock-gating cells. (Circuit level details are presented in Section V.) This implies that the gate-level clock-gating synthesis of pulsed-latch circuits, which we will call *pulser gating synthesis*, needs to group latches in such a way that the latches in a group are sufficiently close together to be driven by a single pulser, and also that they can be gated together (by that pulser) as often as possible.

Fig. 2 illustrates the problem. We assume that the location of the latches is known; and f_i denotes the gating function of latch i. When more than one latch is driven by a single pulser, that pulser is enabled and disabled by an aggregate gating function, e.g., if latches 1 and 2 are driven by a single pulser, then $f_1 \wedge f_2 = ab$ is its gating function. Thus it is important to group latches with similar gating functions, so that an aggregate gating function can disable a pulser as often as possible. A counter-example is provided by latches 1 and 3 in Fig. 2: if they are driven by the same pulser, then that pulser will rarely be disabled.

We need to take account of the maximum load capacitance that can be driven by a pulser, so that the pulse it generates is not distorted. This requires us to accept a conservative limit on the load capacitance, which then affects the number of latches that a pulser can drive and the lengths of the wires to connect the pulser to its latches. In Fig. 2, latches 2 and 4 may not be grouped, since they are located too far apart, even though their gating functions are similar.

Manuscript received August 27, 2011; revised November 30, 2011; accepted January 3, 2012. Date of current version June 20, 2012. A preliminary version of this paper [1] was presented at the Asia and South Pacific Design Automation Conference, Yokohama, Japan, January 25–28, 2011. This work was supported in part by the Mid-Career Researcher Program through NRF Grant funded by the MEST (2011-0029087), and by Samsung Electronics. This paper was recommended by Associate Editor I. Bahar.

S. Paik was with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Korea. He is now with Synopsys, Inc., Mountain View, CA 94043 USA (e-mail: seungwhun.paik@synopsys.com).

I. Han, S. Kim, and Y. Shin are with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Korea (e-mail: ihhan@dtlab.kaist.ac.kr; smkim@dtlab.kaist.ac.kr; youngsoo@ee.kaist.ac.kr).

Digital Object Identifier 10.1109/TCAD.2012.2185235

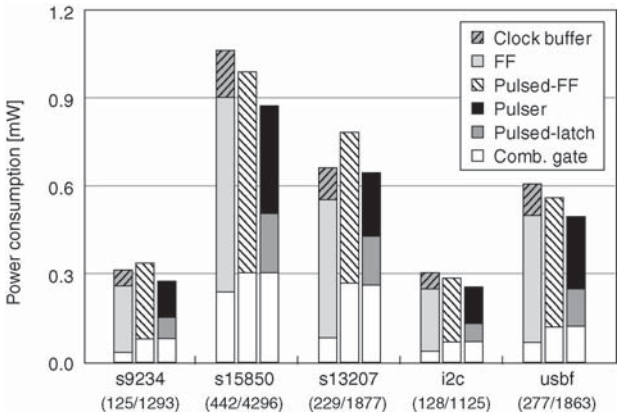

Fig. 1. Power used by equivalent flip-flop (left-hand bars), pulsed-flip-flop (central bars), and pulsed-latch (right-hand bars) circuits. Combinational gates consume more power in pulsed-flip-flop and pulsed-latch circuits due to the extra buffers required to correct their hold-time violations. The numbers inside the parentheses are the number of sequencing elements and combinational gates for pulsed-latch circuits.

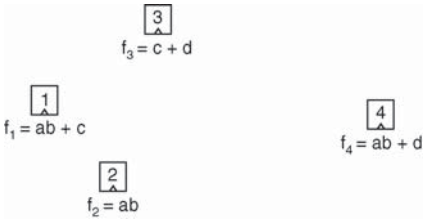

Fig. 2. Illustrative example of the pulser gating synthesis problem.

B. Summary of Contributions

Our main contributions are as follows:

1) design flow for pulser gating synthesis, and its core algorithm Merge (Section IV);
2) formulation of the pulser insertion problem, and a heuristic algorithm to solve it (Section IV-E);
3) extensive experiments in commercial 45 nm technology to assess the synthesis flow (Section V).

The remainder of this paper is organized as follows. In Section II, we review the design of pulsed-latch circuits. Gate-level clock-gating synthesis is addressed in Section III. Pulser gating synthesis flow is presented in Section IV. In Section V, we report experimental results. In Section VI, we review related work, and draw conclusions and finally suggest some direction for future research in Section VII.

II. Pulsed-Latch Circuits

A. Design

A pulsed-latch can be considered to be a fast flip-flop with a long hold time. A typical application-specific integrated circuit (ASIC) design synthesized with flip-flops can simply be transformed to a pulsed-latch version by replacing the flip-flops with latches [9], [10] and adding pulsers, although hold-time violations have to be corrected by inserting delay buffers or resynthesizing combinational circuits [11]. Such a transformation usually benefits both performance (a 10% increase in clock frequency has been reported [11]) and power consumption (possibly a 20% saving [9]), as well as reducing circuit area.

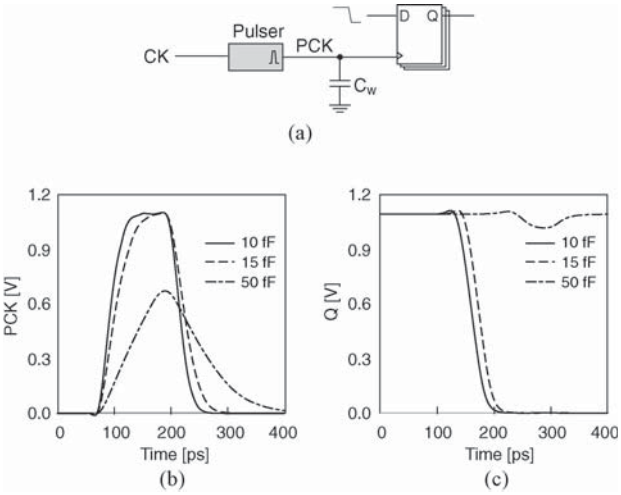

Fig. 3. (a) Experimental circuit to explore timing integrity. (b) Pulse clock PCK. (c) Latch output Q for each PCK of (b).

A key step in the transformation process is to insert pulsers, and this procedure is a significant part of the problem that we address in this paper. It is not straightforward: on one hand, the total number of pulsers should be minimized because of the large amount of power that they consume, as shown in Fig. 1, as well as the large area they occupy; on the other hand, each pulser needs to be connected to a small number of nearby latches, since the pulse that it generates can easily become distorted if it is too heavily loaded.

B. Timing Integrity

Fig. 3(a) shows an experimental circuit designed to illustrate the importance of pulse shape in the operation of pulsed-latch circuits. An example pulser was designed in 45-nm technology to generate a pulse of 110 ps width with a slew of 40 ps, for a load capacitance of 10 fF or smaller. It was connected to ten latches, which provide 6 fF of capacitance. This allows the wire capacitance, denoted by C_w, to be as high as 4 fF.

Fig. 3(b) shows a set of SPICE waveforms of the pulse clock PCK, when the pulser is loaded by 10, 15, and 50 fF, which we achieved by adjusting the value of C_w. The corresponding latch output for a falling latch input is shown in Fig. 3(c). When the pulser is loaded by 10 fF, both PCK and the latch output are nicely generated. At a load capacitance of 15 fF, PCK is slightly distorted; the latches are still able to capture the data but their clock-to-Q delay has increased by 10%, which may cause a timing problem. With a load of 50 fF, PCK is severely distorted and the latches fail to capture the data.

Clearly, as a pulser drives more latches, there is less budget for wire capacitance, which requires a larger number of latches to be closer to the pulser. When the load capacitance of a pulser is 10 fF and more than five latches are connected, then each latch should be located within about five latch cells of the pulser, on average, and that is very close.

III. Clock Gating Synthesis

The first step in gate-level clock gating synthesis is to identify the gating function of each latch. The gating function

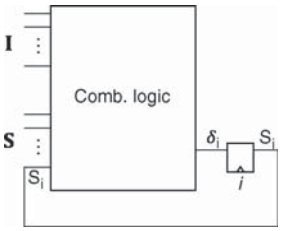

Fig. 4. Generic sequential circuit.

f_i of latch i is given by

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

where δ_i denotes the next state value, which is a function of the circuit inputs $\mathbf{I}$ and the present states $\mathbf{S}$, where $S_i \in \mathbf{S}$ denotes each present state value; this is shown pictorially in Fig. 4. In other words, δ_i is the input to i and S_i is its output; and so, when δ_i and S_i have the same value, there is no need to load δ_i and thus the clock can be gated by setting f_i to 1. Equation (1) may be implemented directly by inserting an XNOR gate [5], but this can only be done when δ_i is evaluated sufficiently early, because the XNOR gate is at the output of δ_i and also decides whether δ_i should be captured.

A. Merge

Implementing each f_i as extra logic is not practical. Therefore, several f_is should be merged into a single gating function F, as follows:

$$F = f_1 \wedge f_2 \wedge \cdots \wedge f_n. \tag{2}$$

It is usually possible to implement this merged gating function using less extra logic. The probability that F is evaluated to 1, denoted by $P(F)$, is called the gating probability and satisfies

$$P(F) \leq \min_i P(f_i) \tag{3}$$

since the on-set of F is the intersection of the on-sets of f_is, i.e., the clock is now gated ($F = 1$) when all n latches can be gated ($f_i = 1$). It is thus essential to merge similar gating functions expressing further intersections of on-sets so that $P(F)$ can be kept high as much as possible.

1) *Effect of Merge:* To understand the effect of merge, we performed an experiment running the Merge algorithm presented in Section IV-C, while we varied the extent of the merge so as to change the number of gating functions produced.

As the merge becomes more extensive, the number of gating functions is reduced, and the number of pulsers drops to the same extent. This causes the average gating probability to decrease, as we would expect from (3), and Fig. 5 confirms this reasoning. The result is higher power consumption in both pulsers and latches, since the clock is now gated less frequently. At the same time, the number of extra gates required to implement gating functions tends to decrease, as we see in Fig. 5, which has an opposite and desirable effect on power consumption. In summary, the power consumption of the combinational logic decreases while that of the pulser and latches increases as the number of gating functions drops. This suggests that there is an optimum number of gating functions.

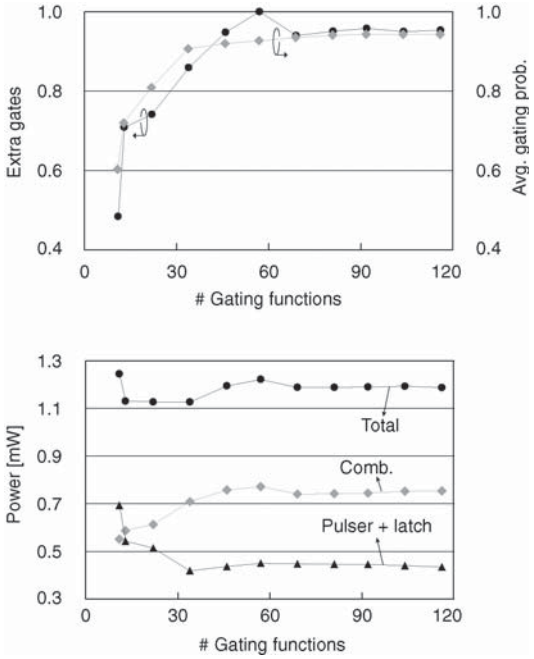

Fig. 5. Number of gating functions produced by a merge affects the number of extra gates for implementing the functions, the average gating probability, and power consumption. These plots are for a simple asynchronous serial controller (sasc) circuit [12].

B. Implementation of Gating Functions

1) *Approximation:* Even when some gating functions have been merged, introducing the extra logic required to implement each F may be still too costly. This can be alleviated by approximating F using the observation that the on-set of F can be considered as a don't-care set, because the functionality of a circuit remains unchanged whether its clock is actually gated or not, provided that the clock can be gated (i.e., $F = 1$). Therefore, F can be approximated by some other function F', which may be implemented with less cost in extra logic, as long as the on-set of F' is a subset of that of F; but this saving comes at the cost of a reduced gating probability.

Several approaches have been proposed [6]–[8] to obtain F'. We report on three methods.

a) Put F into a sum-of-products form, and then determine the probability that each term of the product is evaluated[1] to 1. A term with a very low probability is not useful since its contribution to clock gating is small and therefore it can be declared as the don't-care set [6].

b) Put F into a sum-of-products form and check each variable one by one. If the sum of the probabilities of all the terms that contain that variable is very low, then those terms can be declared as the don't-care set. A

[1]This is approximated by taking the product of the probability of all the literals. If a literal corresponds to a circuit input, its probability must be provided by the designer; if it is the output of a flip-flop, we obtain it by simulation, as discussed in Section IV-B.

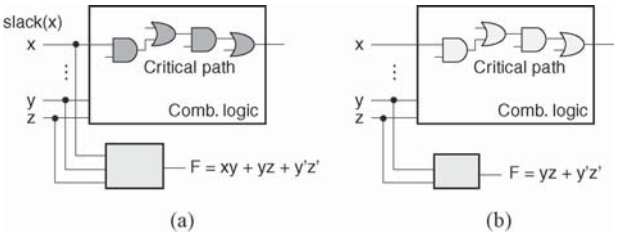

Fig. 6. (a) F increases the delay of a circuit by overloading x, which is on the timing-critical path. (b) Product terms that contain x are removed from F.

gating function with fewer variables is likely to have fewer potential divisors, which reduces the number of Boolean divisions we have to perform (an issue that is to be addressed shortly in Section III-B2).

c) Inserting F into the original netlist with a view to altering its timing behavior due to the change in the load capacitance. Let x be an input that is on the timing-critical path, as illustrated in Fig. 6(a); x is assumed to be one of the variables of F. If the slack at x becomes worse than in the original netlist, then the product terms that contain x are removed, as shown in Fig. 6(b), and declared as off-set.

In our implementation of pulser gating synthesis, shown in Fig. 8, this last method is only applied when F fails after timing analysis.

2) *Division:* An approximated gating function F' is typically implemented as additional logic. We can reduce the amount of extra logic needed to implement F' (or sometimes even F itself) by utilizing existing combinational logic as a Boolean divisor [13].

This idea is illustrated in Fig. 7. Let D be a Boolean expression at some internal node of the combinational logic. If we perform Boolean division [14] using D as a divisor, we get

$$F = D \wedge Q + R \tag{4}$$

where Q and R, respectively, are the expressions corresponding to the quotient and remainder. Then it is only necessary to implement Q and R, in addition to AND and OR gates, which reduces the amount of logic required.

The challenge is to find a D that minimizes the complexity of Q and R. This is not a trivial task, since Boolean division has to be performed for each candidate D, and in essence the Boolean expressions at all internal nodes can be candidates. To reduce the size of this computation, only the nodes within the fanin cone of the latches' inputs are considered as candidates [13], because those nodes are likely to have similar expressions to δ_i [see (1)], which is in turn similar to the gating function; we can also accelerate the computation by performing algebraic division [14] instead of generic Boolean division.

The *Division* algorithm we use has the following steps. We first generate a list of divisors from candidate nodes. These correspond to the Boolean expression at candidate node i, denoted by D_i, and its complement $\overline{D_i}$; we also consider divisors made by combining two divisors, i.e., $D_i + D_j$, $\overline{D_i + D_j}$, $D_i \wedge D_j$, and $\overline{D_i \wedge D_j}$, to expand the pool of potential divisors. Each divisor D in the list is checked to see whether it can divide F or not: for instance, if D has literals that are not in F, algebraic division cannot be performed. But if division is possible, it is performed; and then factored forms of Q and R are obtained, because their literal counts can be used as a measure of complexity. Finally, we select the D that yields the minimum number of literals in Q and R, while allowing some margin to accommodate extra gates such as AND and OR, as shown in Fig. 7. Improvement of the *Division* algorithm is certainly possible, for instance, by performing generic Boolean division or by combining many simple divisors, but the computation-time implications can be daunting.

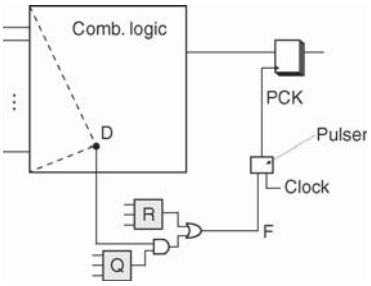

Fig. 7. Boolean division, with an internal node as the divisor, can reduce the extra logic required to implement F.

IV. Pulser Gating Synthesis

A. Overview

The overall flow of pulser gating synthesis is shown in Fig. 8. The input is a gate-level netlist that contains latches, but not yet pulsers. Initial placement [15] of the netlist is performed, so that latch locations are available during the subsequent merge and pulser insertion processes, which are denoted by shaded boxes. A gating function f_i for each latch is determined using (1).

The core of the synthesis process is to merge f_is, thus corresponding latches, to yield a list of merged gating functions Fs; this Merge is the topic of Section IV-C. The Merge is performed in such a way that F obtains a high gating probability $P(F)$. The load capacitance of the pulser allocated to each F must not exceed its load limit. Note that the load capacitance can be computed because latch capacitance is available from the list of latches whose gating functions were merged into F, and the wire capacitance is available by constructing a Steiner tree from the latch locations. Ideally, the amount of extra logic required to implement F should also be taken into account during the merge. But, the complexity of implementation of F, which involves approximation, division, and technology mapping, has so far persuaded us to treat merge and implementation as separate problems.

A trial implementation of each merged gating function is assessed in terms of energy dissipation and circuit timing. For this purpose, F is first submitted to *Division*; technology mapping is subsequently applied to Q and R to derive a gate-level implementation. The energy dissipation of circuit is then

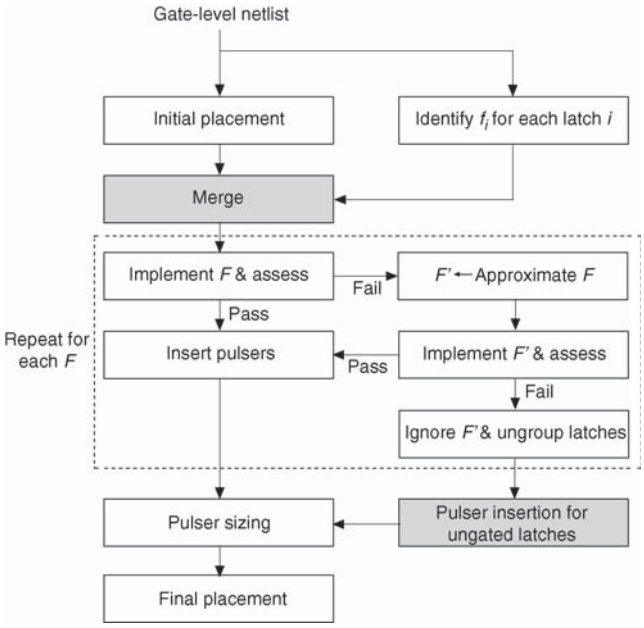

Fig. 8. Overall flow of pulser gating synthesis.

checked to see if F brings about energy savings; and the delay along the timing-critical path is analyzed to see whether this F has caused any increase in latency. If F passes both of these checks, a pulser is inserted into a corresponding group of latches and the appropriate connections are made, including that from the output of F to a pulser. The implementation of a failed F is reversed and we derive an approximate function F'. If F failed due to excessive energy consumption, it is approximated by putting product terms with low probability and less-used variables into a don't-care set (the first two approximation methods presented in Section III-B1); however, if F failed by violating the critical-path delay, then the product terms that contain the variables responsible for the violation are removed from F (the last approximation method of Section III-B1). Next, F' undergoes a process of *Division*, technology mapping, and checking similar to that applied to F. If F' does not pass the checks either, then it is ignored and the corresponding group of latches is dropped from clock gating.

It is possible to repeat the process of merge, approximation, and assessment on dropped latches. We experimented using the fifteen circuits listed in Table I to see whether this actually creates significant room for further clock gating. In just four circuits (s9234, s15850, s35932, and aes_cipher), we found a single additional latch group (with three to four latches) that could now be clock gated. This disappointing outcome strongly suggests that the additional computation time cannot be justified.

Pulsers are inserted for the latches that are not gated, and this process is addressed in Section IV-E; only latch locations are used in this procedure. After the pulsers have been inserted, whether gated or ungated, it is possible that some end up driving a load capacitance which is much smaller than $C_{\max}$. So the size of each pulser is then reduced to suit its actual load capacitance, to further reduce the overall power consumption.

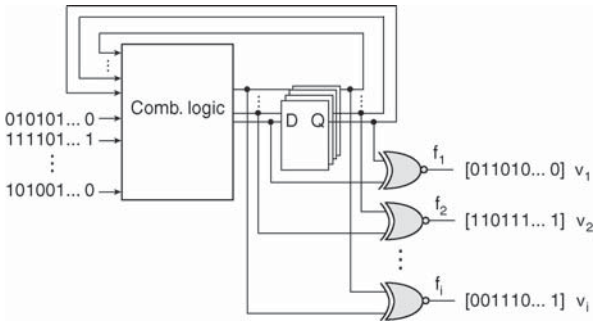

Fig. 9. Computation of a gating vector $\mathbf{v}$.

Final placement is performed either by fixing the location of latches and pulsers and performing placement again, or by using a placement tool that explicitly takes care of the connection between pulsers and latches [16].

B. Computation of Gating Probability

The gating probability $P(F)$ is a key factor during the merge process. Therefore, it is important to be able to compute $P(F)$ efficiently for an arbitrary function F. We use a simulation-based approach in this paper, which is illustrated in Fig. 9. We consider an imaginary XNOR gate which receives the input and output of each latch i as its input; its output corresponds to f_i as defined by (1). N randomly generated patterns are applied to the input of a circuit, and the output of each XNOR gate is stored as a N-bit vector $\mathbf{v}_i$ which we call the gating vector. Clearly, the presence of a logical 1 in $\mathbf{v}_i$ indicates that i can be gated. $P(f_i)$ is now easy to compute by counting the number of logical 1s in $\mathbf{v}_i$ and then dividing the total by N.

It is not difficult to compute $P(F)$ for $F = f_1 \wedge f_2 \wedge \cdots \wedge f_n$. We form a new vector by performing a bitwise AND of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, count the number of logical 1s in the vector, and divide the total by N.

C. Merge Algorithm

The ultimate objective of the merging process is to find the circuit with the minimum power consumption. But the power consumption of a particular solution is hard to estimate in advance, and so we must approach this problem indirectly by maximizing the average gating probability of clusters and minimizing the number of total clusters: these are reasonable replacements for the actual objective function.

The merge problem can be stated as follows.

Problem 1: Given a set of latches, together with the gating function f_i and physical location of each latch, the objective of the merge problem is to find a set of latch clusters in which the latches in each cluster $\mathbb{C}_j$ can be gated and ungated together by a pulser that is controlled by $F_j = \wedge_{i \in \mathbb{C}_j} f_i$, while maximizing the average gating probability avg $P(F_j)$, minimizing the number of clusters, and ensuring that each pulser drives its maximum load $C_{\max}$ or smaller.

This is a multiobjective problem, since maximizing the gating probabilities conflicts with minimizing the number of clusters, as we can see from Fig. 5. We address it with the heuristic algorithm shown in Algorithm 1.

Algorithm 1 *Merge algorithm*

L1 Create a graph $G(V, E, w)$
L2 $F_i \leftarrow f_i$ for each latch i
L3 **while** $E \neq \emptyset$ **do**
L4 Select e_{ij} of maximum $\mathcal{M}_{ij}$
L5 **if** $C(i \cup j) \leq C_{max}$ **then**
L6 $i \leftarrow i \cup j$, $F_i \leftarrow F_i \wedge F_j$
L7 $E \leftarrow E - \{e_{ix}, e_{jy}\}, \forall x \in \mathbb{A}_i - \mathbb{A}_j, y \in \mathbb{A}_j$
L8 Update $w(e_{ix}), \forall x \in \mathbb{A}_i \cap \mathbb{A}_j$
L9 **else** $E \leftarrow E - \{e_{ij}\}$

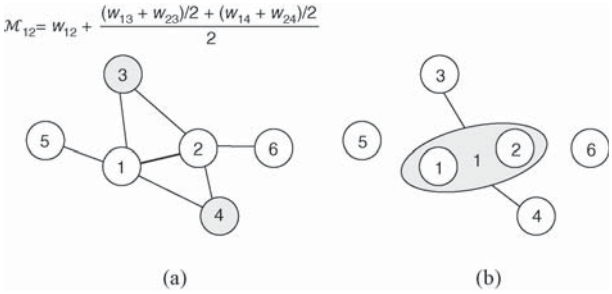

Fig. 10. Graph (a) before and (b) after merging vertices 1 and 2.

We introduce a graph $G(V, E, w)$ in which each vertex $i \in V$ initially corresponds to a single latch; as the algorithm proceeds, latches are grouped and the corresponding vertices coalesce into hyper-vertices. An edge in the graph $e_{ij} \in E$ implies that the gating functions at the vertices may potentially be merged. Each edge is associated with a weight w_{ij}, which is equal to $P(f_i \wedge f_j)$; but if i or j corresponds to a latch cluster, then F_i or F_j replaces f_i or f_j. Initially, every pair of vertices is nominally linked by an edge. However, if merging is likely to violate the $C_{\max}$ condition (i.e., i or j contains too many latches or the latches are too far away), then that edge is dropped from G.

We successively consider each edge e_{ij} and decide whether the two vertices that it connects should be merged (L3–L9). The effectiveness of this approach depends on the order in which edges are selected. A measure of merit $\mathcal{M}_{ij}$ is introduced for this purpose

$$\mathcal{M}_{ij} \triangleq w_{ij} + \frac{\sum_{x \in \mathbb{A}_i \cap \mathbb{A}_j} \left(w_{ix} + w_{jx}\right)/2}{|\mathbb{A}_i \cap \mathbb{A}_j|} \tag{5}$$

where $\mathbb{A}_i$ is the set of vertices that are connected to i. Fig. 10(a) illustrates how $\mathcal{M}_{ij}$ is calculated for an example graph, and Fig. 10(b) shows how this graph is modified after two vertices are merged. The first term, w_{ij} in (5), favors edges with a higher gating probability, meaning that i and j have similar gating functions. The second term corresponds to the average gating probability resulting from the merger of i and j with one of the vertices that are adjacent to both of them; this reflects the potential for a future merger after i and j have become a single hyper-vertex.

If the merger of i and j does not violate the $C_{\max}$ constraint (L5), it is performed and G is modified accordingly. The load capacitance after the merger, denoted by $C(i \cup j)$, is obtained by summing the latch and wire capacitance. The latter is determined from a new Steiner tree spanning all latches; every pulser is assumed to be located on this tree, and so no wire is required to connect a pulser to the tree. The vertices that are adjacent to both i and j, e.g., vertices 3 and 4 in Fig. 10, now share an edge with the merged vertex, and their weights are updated accordingly (L8). All the other edges of i and j are dropped from the graph (L7). Note that G eventually has no edges (L3), and then each hyper-vertex becomes a latch cluster.

D. Assessment of Gating Functions

Latches are grouped by *Merge* without considering the actual implementation of merged gating functions. It is therefore possible that some of these gating functions may cause the power consumption or critical path delay to increase. These possibilities are checked for each F; if the result is negative, F is approximated by a simpler function F', and this is checked in turn; if this result is also negative, then the gating function is dropped and the corresponding latches are ungrouped and declared not to be gated.

When F_i (or F_i') is used for the clock gating of a cluster $\mathbb{C}_i$, consisting of pulser and the latches it drives, then we need to consider three components of energy dissipation, as follows:

$$E_{cg} = E(F_i) + \left(E_p + E_l|\mathbb{C}_i|\right)(1 - P(F_i)). \tag{6}$$

The first component $E(F_i)$ is the energy dissipated by the extra circuitry required to implement F_i (this corresponds to Q and R, as well as the AND and OR gates in Fig. 7); $E(F_i)$ is obtained by simulating the portion of the circuit that contains the extra circuitry with the same vector used to obtain $P(F_i)$. The second and third components are, respectively, the energy dissipated by a pulser, E_p, and the clock energy dissipated by the latches, $E_l|\mathbb{C}_i|$; these dissipations occur when the clock is not gated. The dissipation E_p can be decomposed into internal and load-dependent components

$$E_p = E_{p,\text{int}} + E_{p,\text{load}}. \tag{7}$$

In the original circuit, in which clock gating is not applied to the latches in $\mathbb{C}_i$, only the pulser and latches dissipate energy

$$E_{\text{org}} = \left(\eta E_{p,\text{int}} + E_{p,\text{load}}\right) + E_l|\mathbb{C}_i|. \tag{8}$$

Note that $E_{p,\text{int}}$ is scaled by a factor η. This accounts for the reduction in the number of pulsers that are likely to be used, and the resulting drop in internal energy dissipation by pulsers. This is because grouping is now based on latch locations without regard to gating probability. A reasonable empirical choice of η is $C(\mathbb{C}_i)/C_{\max}$, where $C(\mathbb{C}_i)$ is the load capacitance seen by the pulser that drives $\mathbb{C}_i$.

Energy is saved if $E_{\text{cg}} - E_{\text{org}}$ is determined to be negative. Note that $E(F_i)$ contains both switching and leakage energy components; but only the switching component is considered in determining the energy dissipation of a pulser and its latches, because we assume that clock gating does not greatly affect leakage.

An incremental timing analysis is performed to check that the critical path delay does not increase. In Fig. 7, all the nodes

Algorithm 2 *Pulser_Insertion algorithm*

L1 Create a graph $G_c(V, E, w)$
L2 **while** $E \neq \emptyset$ **do**
L3 Select e_{ij} of minimum w_{ij}
L4 **if** $C(i \cup j) \leq C_{max}$ **then**
L5 $i \leftarrow i \cup j$
L6 $E \leftarrow E - \{e_{ix}, e_{jy}\}, \forall x \in \mathbb{A}_i - \mathbb{A}_j, y \in \mathbb{A}_j$
L7 **else** $E \leftarrow E - \{e_{ij}\}$

that are within the fanout cone of D, as well as Q and R, are submitted to timing analysis.

E. Pulser Insertion for Ungated Latches

Latches are ungrouped and clock gating is not applied if their merged gating function is not acceptable. In the experiments reported in Section V, it turns out that about 50% of latches come into this category. Thus, it is important to minimize the number of pulsers inserted for ungated latches. We can describe this problem more formally.

Problem 2: Given a set of latches, each at a known location, the pulser insertion problem is to minimize the number of latch groups, while keeping the load capacitance of the pulser assigned to each group below $C_{\max}$.

A heuristic similar to *Merge* can readily be applied, without the need to consider gating functions: its pseudocode is shown in Algorithm 2. Again we create a graph (L1); but w_{ij} now corresponds to the wire capacitance that results from grouping i and j, i.e., the wire capacitance of the Steiner tree that spans all the latches contained in i and j. An edge can be deleted if w_{ij} plus the latch capacitance of i and j exceeds $C_{\max}$; this occurs between a and e in Fig. 11(a), where $w_{ae} > 5$. We successively pick an edge with the minimum wire capacitance (L3) and create a hyper-vertex if the merge of the vertices connected by an edge does not violate the $C_{\max}$ constraint. This process is similar to the construction of a minimum spanning tree. The result of applying the *Pulser_Insertion* algorithm to Fig. 11(a) is shown in Fig. 11(b).

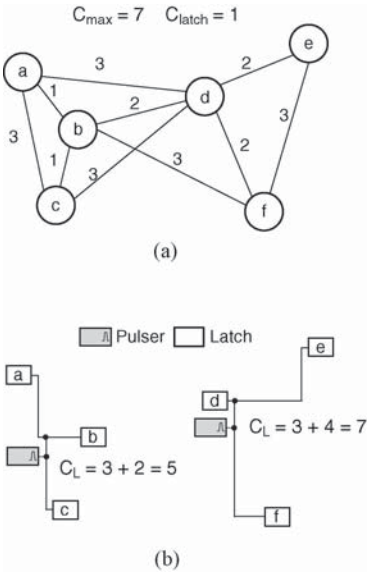

Fig. 11. Inserting pulsers for ungated latches. (a) Example graph G_c. (b) Result achieved by *Pulser_Insertion*.

TABLE I
TEST CIRCUITS

Group	Name	# Gates	# Latches
ISCAS	s838	351	32
	s5378	1781	160
	s9234	1293	125
	s13207	1877	229
	s15850	4296	442
	s35932	15 899	1728
ITC	b11	582	30
	b12	1275	119
OpenCores	aes	13 445	670
	aes_cipher	2609	74
	i2c	1125	128
	pci_ctrl	879	60
	ps2	536	54
	sasc	1058	116
	wb_dma	6091	522

V. EXPERIMENTAL RESULTS

A set of test circuits from well-known ISCAS and ITC benchmarks was prepared, as shown in Table I. We also included some circuits from open cores [12], such as encryption circuits (aes and aes_cipher), communication controllers (i2c, pci_ctrl, and sasc), a keyboard interface unit (ps2), and a DMA channel selector (wb_dma). Each circuit was initially synthesized [17] with flip-flops using an industrial 45 nm ASIC library. All the flip-flops were then replaced by latches. The hold-time violations that occur because the latches will eventually be driven by pulsers with a known pulse width were fixed by inserting buffers [11]. The resulting numbers of combinational gates and latches are reported in the last two columns of Table I.

We designed a new pulser which draws less power while its clock is gated. The circuit and its waveforms are shown in Fig. 12. The clock is gated (so that no pulse is generated) by setting the signal EN to 0, and thus EN is connected to $\overline{F}$. Changing EN effectively enables or disables the whole inverter chain, rather than the AND gate alone [3], thus suppressing the unnecessary power consumption of the inverter chain when EN is 0. Notice that a glitch in EN, while CK is 1, does not affect PCK, unless it appears while PCK is 1, which is unlikely to happen; this is why a conventional clock gating cell is not necessary to filter out glitches. The pulser was designed to drive a load of up to 10 fF with a pulse of width 110 ps and a slew of 40 ps; we verified that this pulser allows the safe capture of data at latches it drives.

The pulser gating synthesis shown in Fig. 8 was implemented in SIS [18]. It has two key components: the merge and the implementation of gating functions through approximation and division. These components of the process are assessed individually in the following sections. We will analyze pulser gating synthesis as a whole in Section V-C.

A. Assessment of Merge

The proposed *Merge* algorithm is a greedy heuristic for Problem 1. This is a multiobjective problem with two conflict-

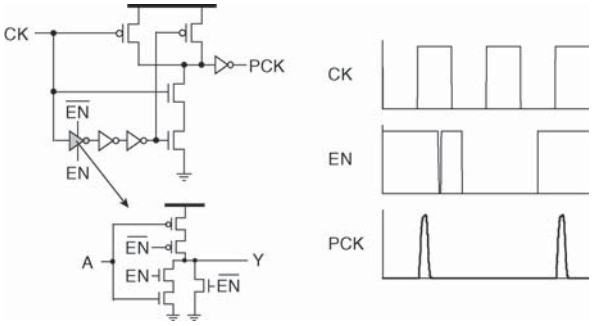

Fig. 12. Pulser and its SPICE waveforms.

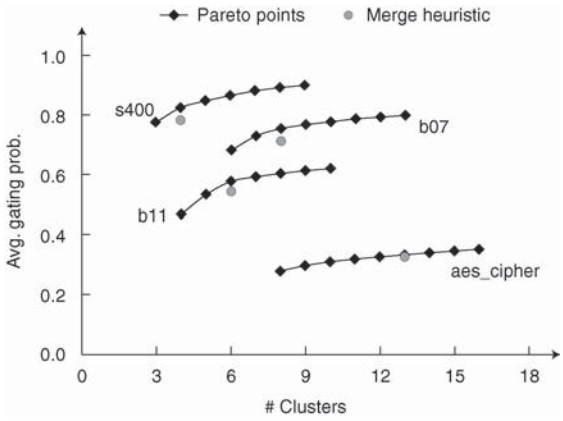

Fig. 13. Pareto points and the solution returned by the *Merge* heuristic.

ing goals: maximizing the average gating probability and minimizing the number of clusters (thus the number of pulsers).

We generated a set of Pareto reference points for very small examples such as b11. Fixing the latch locations, we generated all possible latch clusters by manipulating the graph G introduced in Section IV-C; we first make each latch a cluster; then clusters with two latches are generated by combining two single-latch clusters which share an edge in G, if the new cluster satisfies the $C_{\max}$ constraint. Clusters with more latches are generated in a similar way, until no more new clusters can be formed without violating the $C_{\max}$ constraint.

Each of the resulting N clusters is associated with a decision variable x_i and its gating probability P_i is also determined. We want to choose M clusters with the highest gating probabilities. This choice can be formulated as an integer linear programming (ILP) problem with the objective function

$$\text{Maximize} \quad \sum_{i=0}^{N} P_i x_i \tag{9}$$

which maximizes the average gating probability. We are going to pick only M clusters

$$\sum_{i=0}^{N} x_i = M \tag{10}$$

and each latch has to be included in one and only one cluster

$$\sum_{\forall i \text{ that contains } j} x_i = 1 \qquad \text{for each latch } j. \tag{11}$$

We now vary M and solve the ILP problem for each instance, producing a set of Pareto points; but this computation requires a lot of time and memory, and can therefore be applied to very small circuits. Fig. 13 shows the Pareto points and the solution obtained by the *Merge* algorithm for four example circuits.

For the three circuits, s400, b07, and b11, the solution returned by *Merge* appears to be close to the peak of the curve generated by the Pareto points, which suggests that a reasonable tradeoff has been achieved between the average gating probability and the number of clusters. The curve for aes_cipher is comparatively flat, suggesting that its average gating probability is not strongly affected by the construction of clusters. We found that many latches in aes_cipher have similar gating functions (46% of pairs of the latches have merged gating function with probabilities of 0.7 or more), so the average gating probability does not drop dramatically even if a cluster contains many latches.

TABLE II
NUMBER OF EXTRA GATES TO IMPLEMENT GATING FUNCTIONS WITHOUT AND WITH USING *Division*

Circuit	# Gating Functions	# Extra Gates		
		Without *Division*	With *Division*	Diff. (%)
s838	5	24	14	−41.7
s5378	7	106	106	–
s9234	8	130	103	−20.8
s13207	27	443	423	−4.5
s15850	40	545	517	−5.1
s35932	277	2272	2272	–
b11	3	27	27	–
b12	11	238	230	−3.4
aes	25	355	329	−7.3
aes_cipher	8	86	86	–
i2c	10	181	166	−8.3
pci_ctrl	2	120	85	−29.2
ps2	1	15	15	–
sasc	15	165	165	–
wb_dma	32	591	485	−17.9
Average				−9.2

B. Assessment of Implementation of Gating Functions

The implementation of a gating function F can be followed by two possible procedures (see Fig. 8): either we apply *Division* to F, check if there is an energy saving and the timing-critical path remains intact, and then implement Q and R if F passes these checks; or we can approximate F by F', apply *Division* to F', check F', and implement as before.

To assess the effectiveness of the *Division* algorithm, we collected all the gating functions, either F or F', that passed the check when they were implemented without using *Division*: in this case "Implement" in Fig. 8 becomes the direct implementation of the gating function. The numbers of these gating functions are listed in the second column of Table II; and column 3 contains the number of extra gates required for direct implementation of the gating functions. *Division* is

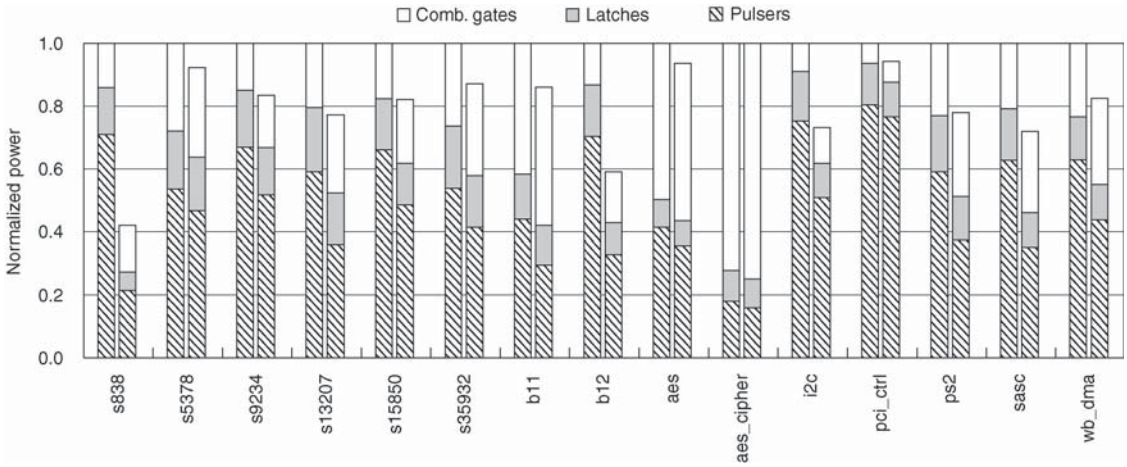

Fig. 14. Power consumption of initial circuits (left-hand bars) and of circuits after pulser gating synthesis (right-hand bars). The reduction in power drawn ranges from 0.1% to 58.0%, with an average of 19.8%.

TABLE III
NUMBER OF EXTRA GATES IN PULSER GATING SYNTHESIS FLOW

Circuit	# Gating Functions	Extra Gates	
		Number	Percentage
s838	6	14	4.0
s5378	7	106	6.0
s9234	8	103	8.0
s13207	27	415	22.1
s15850	40	517	12.0
s35932	277	2272	14.3
b11	3	27	4.6
b12	11	230	18.0
aes	25	329	2.4
aes_cipher	8	86	3.3
i2c	10	166	14.8
pci_ctrl	2	85	9.7
ps2	6	47	8.8
sasc	16	187	17.7
wb_dma	39	634	10.4
Average			10.4

then applied to all the gating functions and the number of extra gates required to implement the quotients and remainders is counted; these totals are shown in column 4. Comparing columns 3 and 4, we see that the average reduction in gate count is 9.2%, but there is a wide variation. Circuits s838, s9234, and pci_ctrl benefit most while there is no improvement in six other circuits. This result is affected by the goodness of the divisors discovered by *Division*. Sufficient divisors were found for all the circuits we tested, even though we used algebraic division; but many of those divisors only contained a small number of literals, and were therefore discarded. The possibility of using generic Boolean division, if computation times can be controlled, merits future investigation.

Table III presents the results of gating function implementation when *Division* is run and then the quotient and remainder are implemented: so that "Implement" in Fig. 8 recovers its expected meaning. Note that the numbers of gating functions in Table III are never smaller than these in Table II, because gating functions are assessed after applying *Division* in the second procedure, whereas they are assessed directly in the first procedure. The numbers of these extra gates and the percentage increase above the initial numbers of combinational gates (column 3 of Table I) are presented in the last two columns of Table III. These results are affected by the size of the gating functions (either F or F') and the applicability of *Division*. For example, there is a 22.1% increase in the number of combinational gates in circuit s13207 because *Division* can rarely be applied (see Table II). But the efficacy of *Division* is not always significant: the increase in gates is only 4.6% for circuit b11, even though *Division* is never applied, because b11 has very small gating functions.

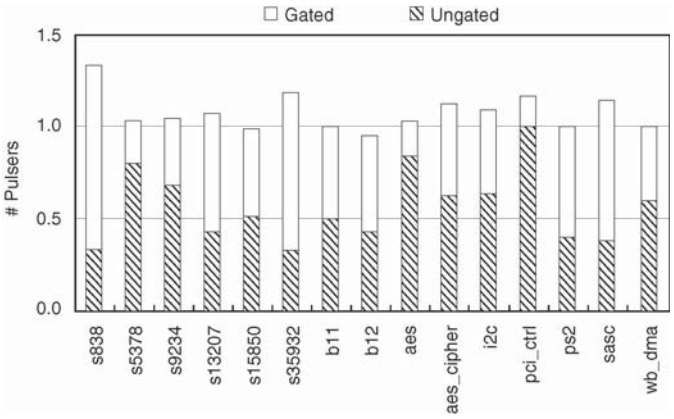

Fig. 15. Number of pulsers after synthesis, normalized to the number obtained by applying *Pulser_Insertion* to the original netlist.

C. Analysis of Pulser Gating Synthesis

The result of pulser gating synthesis is summarized in Table IV. Column 2 contains the number of pulsers to which clock gating is applied; the next column gives the number of ungated pulsers produced by the *Pulser_Insertion* algorithm. Gated and ungated latches are shown in columns 4 and 5. The average gating probability over all the latches is shown in column 6, and the synthesis runtime is reported in the last column. The power consumption of the circuits before

TABLE IV
RESULTS OF PULSER GATING SYNTHESIS

Circuit	# Pulsers		# Latches		Avg. Gating Probability	Runtime (s)
	Gated	Ungated	Gated	Ungated		
s838	6	2	27	5	0.79	12
s5378	7	24	36	124	0.18	79
s9234	8	15	46	79	0.29	73
s13207	27	18	139	90	0.47	92
s15850	40	43	227	215	0.31	442
s35932	277	107	1270	458	0.40	2118
b11	3	3	15	15	0.37	42
b12	11	9	69	50	0.55	139
aes	25	111	114	556	0.16	1041
aes_cipher	8	10	32	42	0.24	62
i2c	10	14	55	73	0.36	68
pci_ctrl	2	12	9	51	0.15	67
ps2	6	4	37	17	0.43	90
sasc	16	8	85	31	0.57	45
wb_dma	39	58	193	329	0.31	807
Average	47%		49%		0.37	

TABLE V
PULSERS WITH DIFFERENT DRIVING CAPABILITIES

Name	$C_{\max}$	Power consumption with various load capacitances (μW)			
		0 fF	4 fF	7 fF	10 fF
Pulser1	4 fF	2.82	6.81	–	–
Pulser2	7 fF	3.93	7.80	11.12	–
Pulser3	10 fF	6.51	9.87	13.11	16.45

TABLE VI
RESULTS OF PULSER SIZING

Circuit	# Pulsers Used			Power Saving (%)	
	Pulser1	Pulser2	Pulser3	Pulsers	Total
s838	3	3	2	−32.1	−15.9
s5378	0	18	13	−20.0	−10.0
s9234	2	8	13	−14.7	−9.4
s13207	1	15	29	−16.6	−7.1
s15850	0	36	47	−15.8	−9.5
s35932	6	128	250	−12.0	−5.9
b11	0	3	3	−12.5	−4.5
b12	1	6	13	−16.7	−10.0
aes	5	81	50	−20.5	−8.3
aes_cipher	3	9	6	−21.0	−3.4
i2c	1	12	11	−20.7	−14.4
pci_ctrl	1	10	3	−27.1	−21.9
ps2	1	3	6	−14.9	−7.0
sasc	0	11	13	−21.0	−10.5
wb_dma	0	45	52	−14.7	−7.8
Average				−18.7	−9.7

and after synthesis is compared in Fig. 14; these figures were obtained by simulating each circuit with a fast transistor-level simulator [19] and applying 1000 input patterns.

There are three factors that determine the efficiency of a synthesis in terms of power consumption.

1) Number of pulsers: as we saw in Fig. 1, pulsers draw a lot of power, and so it is important to use as few as possible. To assess the numbers of pulsers, we applied *Pulser_nsertion* to each original circuit without considering gating functions. We then normalized columns 2 and 3 of Table IV against these results to produce Fig. 15. We can see that the locations of latches with similar gating functions affect the requirement for pulsers: if latches with similar gating functions are dispersed across the layout, the average number of latches that can be driven by each gated pulser decreases; this can be observed in circuits s838, s35932, aes_cipher, and pci_ctrl. Nevertheless, the overall requirement does not seem to be unreasonable.
2) Average gating probability: the average gating probability clearly drops as we try to minimize the number of pulsers. However, Fig. 13 shows that this reduction is unavoidable due to the multiobjective nature of the problem, and it is not very substantial.
3) Extra logic to implement gating functions: even though there is an average increase of 10.4% in the amount of combinational logic, the increase in power consumption that this incurs is marginal, as shown in Fig. 14 (white portion of each bar). This is expected because of the relatively low switching activity (∼5%) of typical combinational logic.

Most of the computation time (∼70%) is taken by *Division* and assessment of the gating functions. Some tuning of these tasks will certainly be possible.

After pulser gating synthesis, the circuit area rises by 9.6% on average; most of this increase is due to gating functions, while the remainder is due to additional pulsers.

D. Analysis of Pulser Sizing

So far, we have assumed that only one size of pulser, in which $C_{\max}$ is 10 fF, is used during pulser gating synthesis, in particular *Merge* and *Pulser_Insertion*. It turns out that, averaged over all the circuits we tested, 53% of pulsers have a load capacitance less than 70% of $C_{\max}$. This opens up the possibility of sizing pulsers in a similar way to the

more usual gate sizing, so as to optimize power consumption, performance, and area.

We designed two smaller pulsers that will generate the same pulse as the 10 fF version (110 ps wide with 40 ps of slew), as long as their maximum load capacitances are not exceeded. Some characteristics of these new pulsers are given in Table V, where they are called Pulser1 and Pulser2; Pulser3 is the original version. The power consumption data given in Table V shows that a smaller pulser does use less power, as we expect.

The result of pulser sizing is shown in Table VI. On average, 51% of the pulsers are now either Pulser1 or Pulser2 types; the latter proved especially useful. This change reduces pulsers power consumption by 18.7%, and the overall power usage by 9.7%.

VI. Related Work

We will briefly review previous work related to gate-level clock gating of pulsed-latch circuits.

A. Pulsed-Latch Methodology

There have been several research efforts to use pulsed-latches instead of flip-flops. It has been shown [9], [10] that the flip-flops in existing designs can be replaced with pulsed-latches to reduce dynamic power consumption; and selective replacement of flip-flops can reduce the timing overhead [20]. In all these schemes, each pulser is buffered to drive many latches; but there is no detailed discussion of the grouping of latches to share a pulser more effectively. We take a different line, and do not place buffers between a pulser and its latches, so that a pulser effectively serves as a leaf-level clock buffer. This strategy avoids the possibility of a pulse disappearing as it propagates through multiple buffers; but now pulser insertion becomes an important problem (which we addressed in Section IV-E).

B. Gate-Level Clock Gating

In gate-level clock gating, gating functions can be automatically synthesized from a netlist [6]–[8]. Since the gating function itself consumes power and area, effectiveness depends on controlling the implementation cost while achieving a high gating probability.

One way to reduce the implementation cost is to reuse the existing logic. Internal signals can be used as a Boolean divisor D to implement a gating function F given by $F = D \wedge Q + R$ [13]. If some loss of gating probability can be tolerated, a gating function can be approximated by summing the internal nodes that are strictly contained in that function [7]; this only requires an extra OR gate. There are several other approaches to approximating F. If the probability of a particular product term is low, it can be declared as a don't-care set, and this reduces the size of the synthesized gating function [6]. If some variables only make a small contribution to the gating probability, they can be removed from the support of gating function [6]. When gating functions are described using a binary decision diagram (BDD), there is advantage in eliminating a portion of the BDD nodes below some gating probability threshold [21].

Merging gating functions across multiple flip-flops is also an effective way of reducing the implementation cost. This has been explored in the context of counters by means of a simple grouping of neighboring bits [22]. A more general algorithm [13] randomly selects a gating function and greedily merges other gating functions in pursuit of a heuristic objective function. A similarity measure, which takes account of both logical similarity and physical distance, has also been proposed to guide merging [8]. Merging gating functions is also a key problem in clock gating of pulsed-latches; but existing algorithms cannot be directly applied to our approach because of the $C_{\max}$ constraint, which is why we have had to develop a new heuristic algorithm.

VII. Conclusion

Gate-level clock gating synthesis can discover clock gating conditions that are not specified by designers. We explored gate-level clock gating synthesis of pulsed-latch circuits, in which a key problem is to group physically close latches that have similar gating functions. We showed that Boolean division can reduce the amount of extra logic required to implement gating functions, and the same approach can also be applied to general gate-level clock gating synthesis.

We can identify some of the further work needed to develop this concept. Currently, merge is performed without considering the complexity of the resulting gating functions; some way of quickly estimating that complexity would be of considerable value. Also, each gating function is currently implemented separately; sharing logic between several gating functions might well be advantageous. Finally, we noted that pulser sizing is currently performed after optimization; in a future investigation we might consider combining pulser sizing and pulser gating synthesis.

References

[1] S. Kim, I. Han, S. Paik, and Y. Shin, "Pulser gating: A clock gating of pulsed-latch circuits," in *Proc. Asia South Pacific Des. Autom. Conf.*, Jan. 2011, pp. 190–195.

[2] S. Kozu, M. Daito, Y. Sugiyama, H. Suzuki, H. Morita, M. Nomura, K. Nadehara, S. Ishibuchi, M. Tokuda, Y. Inoue, T. Nakayama, H. Harigai, and Y. Yano, "A 100 MHz, 0.4 W RISC processor with 200 MHz multiply adder, using pulse-register technique," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 1996, pp. 140–141.

[3] S. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T. Sullivan, and T. Grutkowski, "The implementation of the Itanium 2 microprocessor," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1448–1460, Nov. 2002.

[4] H. Partovi, R. Burd, U. Salim, F. Weber, L. DiGregorio, and D. Draper, "Flow-through latch and edge-triggered flip-flop hybrid elements," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 1996, pp. 138–139.

[5] *Power Compiler User Guide*, Synopsys, Inc., Mountain View, CA, Dec. 2010.

[6] L. Benini, G. D. Micheli, E. Macii, M. Poncino, and R. Scarsi, "Symbolic synthesis of clock-gating logic for power optimization of synchronous controllers," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, no. 4, pp. 351–375, Oct. 1999.

[7] A. Hurst, "Automatic synthesis of clock gating logic with controlled netlist perturbation," in *Proc. Des. Autom. Conf.*, Jun. 2008, pp. 654–657.

[8] E. Arbel, C. Eisner, and O. Rokhlenko, "Resurrecting infeasible clock-gating functions," in *Proc. Des. Autom. Conf.*, Jul. 2009, pp. 160–165.

[9] S. Shibatani and A. Li, "Pulse-latch approach reduces dynamic power," *EE Times*, Jul. 2006.

[10] H. Li, M. Chen, and K. Ho, "System and method of replacing flip-flops with pulsed latches in circuit designs," U.S. Patent 7 694 242 B1, Apr. 2010.
[11] Y. Shin and S. Paik, "Pulsed-latch circuits: A new dimension in ASIC design," *IEEE Des. Test Comput.*, vol. 28, no. 6, pp. 50–57, Nov. 2011.
[12] OpenCores [Online]. Available: http://www.opencores.org
[13] F. Theeuwen and E. Seelen, "Power reduction through clock gating by symbolic manipulation," in *Proc. IFIP Int. Workshop Logic Architecture Synthesis*, 1996, pp. 184–191.
[14] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Comput.-Aided Des.*, vol. 6, no. 6, pp. 1062–1081, Nov. 1987.
[15] *IC Compiler Implementation User Guide*, Synopsys, Inc., Mountain View, CA, Mar. 2010.
[16] Y. Chuang, S. Kim, Y. Shin, and Y. Chang, "Pulsed-latch-aware placement for timing-integrity optimization," in *Proc. Des. Autom. Conf.*, Jun. 2010, pp. 280–285.
[17] *Design Compiler User Guide*, Synopsys, Inc., Mountain View, CA, Sep. 2008.
[18] E. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Univ. California, Berkeley, CA, Tech. Rep. UCB/ERL M92/41, May 1992.
[19] *NanoSim User Guide*, Synopsys, Inc., Mountain View, CA, Jun. 2009.
[20] T. Baumann, D. Schmitt-Landsiedel, and C. Pacha, "Architectural assessment of design techniques to improve speed and robustness in embedded microprocessors," in *Proc. Des. Autom. Conf.*, Jul. 2009, pp. 947–950.
[21] U. Hinsberger, R. Kolla, and T. Kunjan, "Approximative representation of Boolean functions by size controllable ROBDDs," Instit. Inform., Univ. Würzburg, Würzburg, Germany, Tech. Rep. 182, Sep. 1997.
[22] P. Parra, A. Acosta, and M. Valencia, "Selective clock-gating for low power/low noise synchronous counters," in *Proc. 12th Int. Workshop Integr. Circuit Des.*, 2002, pp. 448–457.

Seungwhun Paik received the B.S. and Ph.D. degrees in electrical engineering from the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 2006 and 2011, respectively.

He joined Synopsys, Inc., Mountain View, CA, in 2011, where he has been working on a timing analysis tool. His current research interests include timing analysis and optimization of very large scale integration circuits.

Dr. Paik has been a member of the Technical Program Committee of ASP-DAC.

Inhak Han received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2010 and 2012, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, KAIST.

His current research interests include timing and thermal analysis for very large scale integration circuits and low-power design using clock gating and pulsed-latch.

Sangmin Kim received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2008 and 2010, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, KAIST.

His current research interests include computer-aided design for low-power design and pulsed-latch application-specific integrated circuit design.

Youngsoo Shin (M'00–SM'05) received the B.S., M.S., and Ph.D. degrees in electronics engineering from Seoul National University, Seoul, Korea.

From 2000 to 2001, he was a Research Associate with the University of Tokyo, Tokyo, Japan, and from 2001 to 2004 he was a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY. He joined the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 2004, where he is currently an Associate Professor. His current research interests include computer-aided design with emphasis on low-power design and design tools, high-level synthesis, sequential synthesis, and programmable logic.

Dr. Shin has received several awards, including the Best Paper Award at the 2005 International Symposium on Quality Electronic Design and the 2002 IP Excellence Award from Japan. He has been a member of the technical program committees and organizing committees of many technical conferences, including DAC, ICCAD, ISLPED, ASP-DAC, CASES, ISVLSI, and ISCAS. He is a member of the ACM SIGDA Low Power Technical Committee and is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and the *ACM Transactions on Design Automation of Electronic Systems*.