Synthesis of Clock Gating Logic through Factored Form Matching

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

Abstract— **Clock gating is typically dictated by designers in register transfer level (RTL). Automatic synthesis of clock gating in gate level has been less explored, but is certainly more convenient to designers; it can also complement RTL clock gating by extracting additional gating conditions. The key problem in gate-level clock gating synthesis is to implement gating conditions with minimum amount of additional logic. In this paper, we aim to utilize the existing combinational logic as much as possible. This is done by extracting a factored form (modeled by a factoring tree) of each gating condition, and try to cover the tree by factoring trees of existing combinational logic; the corresponding process is named factored form matching. Experiments demonstrate that the proposed matching achieves 25% reduction in the number of gates to implement gating conditions; this can be compared to prior method using Boolean division, which achieves 10% reduction.**

I. Introduction

Clock gating is a standard practice to reduce clock power consumption. The condition in which clock is gated, called a *gating function*, is typically specified by designers during RTL design stage. A prime example is a load-enable register, illustrated in Fig. 1(a). When EN is 0, the register keeps the current value of Q; otherwise D is loaded at the next clock edge. A gating function, which generates $\overline{\text{EN}}$ in this case, is provided by designers. Clock gating can be applied by using a circuit shown in Fig. 1(b), which is functionally equivalent to Fig. 1(a). A circuitry within the dotted box is called a clock-gating cell (CGC); a latch is needed to remove any glitches in EN when clock is 1.

Another approach to clock gating is to automatically synthesize gating functions from a gate-level netlist [1]–[4]. This is a convenient approach in designer's point of view; it can also complement RTL approach by extracting additional gating functions not specified by designers.

Given a gate-level netlist, the first step toward the synthesis is to extract a gating function f_i of each flip-flop i. This can intuitively be done: if the input d_i and output q_i take the same value, there is no need to load d_i at the next clock edge and f_i can be set to 1:

$$f_i = \overline{d_i \oplus q_i}, \tag{1}$$

where $\oplus$ denotes XOR. A direct implementation of (1) requires an XOR gate and a CGC [5]; this however is possible only when d_i arrives early enough so that the output of CGC becomes stable before the next clock edge.

A more viable approach is to implement f_i as a separate

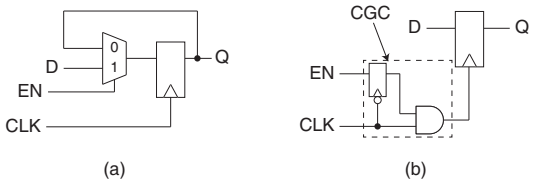

Fig. 1. (a) Load-enable register and (b) clock gating implementation of the same register.

logic[1]. In this situation, two key problems are identified:

- Merge: group f_is so that the corresponding flip-flops in a same group are driven by a single gating function $F = f_1 \wedge f_2 \wedge \cdots \wedge f_n$.
- Simplification: simplify F as much as possible.

The first problem can be solved by various approaches such as greedy [3] or iterative minimum weight perfect matching [6], [7]. The second problem, which is more challenging, is the focus of this paper.

A. Related Work

The on-set of F can be considered as don't-care set; this is because, when clock can be gated ($F = 1$), the functionality of a circuit remains unchanged whether clock is actually gated or not gated. Therefore, F can be approximated by some other function F', which is implemented with less cost of extra logic, as long as the on-set of F' is a subset of that of F; this, however, comes at the cost of reduced gating probability. Several approaches have been proposed for this purpose [1]–[3], e.g. product terms (in sum-of-products form of F) having smaller probability are dropped.

Another direction of simplification is to utilize existing combinational logic to implement parts of gating functions [4], [8]. The idea is depicted in Fig. 2. If F is represented by $DQ+R$, in which D is a Boolean expression at one internal node of a combinational logic, only Q and R can be implemented. As D becomes larger (i.e. it contains more literals), which is desirable, it is less likely to be a divisor of F, which is a drawback of this approach.

B. Contribution

We extend and generalize the idea using Boolean division shown in Fig. 2. Instead of searching for a divisor D, we try to find as many internal nodes as possible (including D) whose

[1]Flip-flop input d_i is represented by a Boolean expression, which may include flip-flop output q_i as one of its variables in sequential circuit; f_i therefore is also represented by a Boolean expression.

978-1-4673-0145-9/12/$31.00 ©2012 IEEE

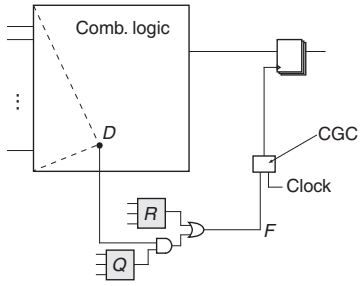

Fig. 2. Utilizing Boolean division, in which internal node is used as a divisor, to implement F.

Algorithm *Clock_Gating_Synthesis*
L1 Identify a gating function f_i of each FF i
L2 Compute gating probability $Pr(f_i)$
L3 $\{F_1, F_2, ...\} \leftarrow$ Merge
L4 **for** each F_i **do**
L5 *Factored_Form_Matching*
L6 Implement F_i
L7 **end do**

Fig. 3. Overall flow of clock gating synthesis algorithm.

factored forms match parts of a factored from of F; the process is named factored form matching. Significant reduction in the number of gates to implement gating functions (over division and approximation) is demonstrated.

The remainder of this paper is organized as follows. In Section II, we outline the proposed clock gating synthesis; the proposed scheme for simplification problem is addressed in the following section. Experimental results are reported in Section IV, and we draw conclusions in Section V.

II. Overview of The Approach

The proposed synthesis approach is outlined in Fig. 3. A gating function f_i of each flip-flop i is identified (L1) by using (1). We then compute the gating probability (L2). A simulation-based approach is used for this purpose, which is illustrated in Fig. 4. We assume an imaginary XNOR gate, which receives the input and output of each i as its input; its output corresponds to f_i by its definition. The M number of patterns are applied to the input of a circuit, and the output of each XNOR gate is stored as an M-bit vector, denoted by $\mathbf{v}_i$. Clearly, 1 in $\mathbf{v}_i$ indicates that i can be gated. $Pr(f_i)$ is now easy to compute by counting the number of 1's of $\mathbf{v}_i$ and divide it by M. Computation of $Pr(F)$ for $F = f_1 \wedge f_2 \wedge \cdots \wedge f_n$ is also easy; it is carried out during merge process. We form a new vector by taking a bitwise AND of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, and count the number of 1's of the vector and divide it by M.

The merge process is then performed, which yields a series of merged gating functions (L3). This is based on iterative minimum weight perfect matching [7], except that the merge which is estimated to save power consumption is executed.

Each merged gating function is then submitted to factored form matching (L5) to see which parts of factored form of F_i can be replaced by internal nodes of a combinational logic.

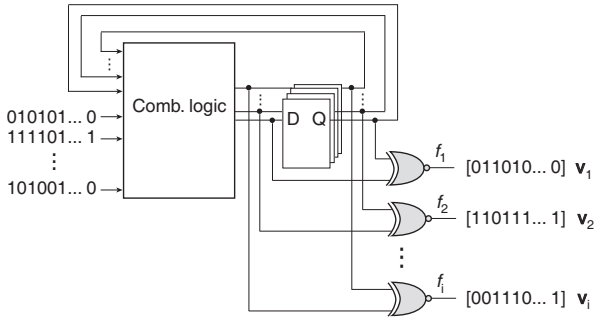

Fig. 4. Computation of gating probability.

The remaining part of F_i is submitted to technology mapping to obtain its gate-level implementation.

III. Factored Form Matching

The result of merge is a set of gating functions $F_1, F_2, \ldots$ (see L3 of Fig. 3). Each gating function is represented as a factored form, or equivalently factoring tree. An example is shown in Fig. 5(a). Our goal is to find internal nodes of a combinational logic, which can be used to implement parts of a gating function. Since an internal node is also represented by a factored form, the problem becomes that of matching the two factored forms.

A. Strong Match

Consider Fig. 5(b), which is a factoring tree of one internal node. It is exactly the same as a sub-tree of F rooted at node n_1, which we call a strong match; there is no need to implement the sub-tree thereby reducing the extra logic to implement F. The two factored forms are, in fact, syntactically equivalent [9], i.e. they represent the same logic function and their factoring trees are isomorphic. For each N_i, detecting whether there is a strong match in F can readily be done.

We should also be able to detect equivalent factored forms (not necessarily syntactically equivalent), e.g. if $N_1 = fd + fe + fg$, its factoring tree will be different from the sub-tree rooted at n_1, even though logic functions are still the same. Detecting equivalent factored forms is more difficult. Fortunately, in SIS [10] in which we implemented the proposed clock gating synthesis, the same logic function is always represented by the same factoring tree.

B. Weak Match

Consider Fig. 5(c), a factoring tree of another internal node. It is clear that there is no strong match this time between F and N_2. However, it can be checked that the parts of F marked with dotted line in Fig. 5(a) collectively constitute the same expression as N_2; a corresponding match is called a weak match, which is more difficult to detect than strong match. Notice that g can be ANDed with a in the right sub-tree of F due to distributive law. This means that either a or g in the right sub-tree and $d + e$ in the left sub-tree must have a common factor, $b + c$ colored by gray in Fig. 5(a), as their siblings. It is thus seen that, after this weak match found,

978-1-4673-0145-9/12/$31.00 ©2012 IEEE

TABLE I

TEST CIRCUITS

Circuit	# Gates	# FFs		Avg. gating prob. of gated FFs	After merge	
		Ungated	Gated		# Gating functions	Avg. gating prob. of gated FFs
s1423	1191	46	28	0.89	5	0.61
s5378	1781	12	148	0.91	17	0.61
s13207	1877	69	160	0.91	15	0.56
s15850	4296	121	321	0.94	31	0.62
s35932	15899	321	1407	0.89	141	0.59
s38584	11074	340	898	0.81	91	0.53
b07	431	6	38	0.93	4	0.71
b12	1275	44	75	0.99	13	0.81
b17	30418	1188	126	0.97	17	0.74
ac97	219	13	58	0.97	6	0.71
i2c	1125	40	88	0.98	10	0.84
mem_ctrl	12494	586	465	0.95	44	0.60
systemcaes	13445	533	137	0.97	15	0.80
usbf_top	789	30	68	0.96	9	0.81
wb_dma	6091	88	434	0.95	48	0.60

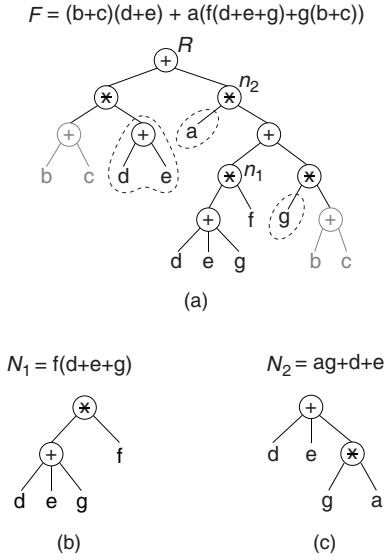

Fig. 5. (a) Factoring tree of a gating function F, (b) factoring tree of an internal node N_1, which is strongly matched in F, and (c) another factoring tree N_2, which is weakly matched in F.

F is implemented by $F = N_2(b+c) + af(d+e+g)$, thereby reducing the complexity of F.

The checking of weak match is done as follows. Each literal l of N_2 is picked one by one. We then search for the same literal l_m in F, which satisfies the following conditions.

1) The parent of l_m is the same Boolean operator as that of l.
2) If the parent of l is OR, all its sibling literals also appear as sibling literals of l_m (e is a sibling literal of d both in F and N_2).
3) If the parent of l is AND and some of its siblings are already checked, the first common predecessor of l_m and the node that matches checked siblings of l is AND (assume that a is checked and $l = g$; in F, n_2 is the first common predecessor of $l_m = g$ and a).

If such l_m exists, it is marked; after all the siblings are marked, their parent is also marked. The nodes within the dotted lines in Fig. 5(a) are such ones.

The next step is to search for the existence of a common factor, such as $b+c$ in Fig. 5(a). For this step, we notice the following proposition.

Proposition 1: The first common predecessor of all marked nodes, denoted by R, is always OR.

We visit each marked nodes one by one, a, g, and $d+e$ in Fig. 5(a). If the parent of marked node is AND, its siblings are checked to see whether they contain any marked nodes, e.g. the sibling of a contains g; such nodes are skipped. Otherwise, the siblings are saved, e.g. $b+c$ is saved for g and for $d+e$. If the saved siblings match in all sub-trees of R, they serve as a common factor.

IV. EXPERIMENTAL RESULTS

A set of test circuits, taken from ISCAS and ITC benchmarks as well as from open cores [11], are listed in Table I. Design Compiler [12] was used for logic synthesis with commercial 32-nm ASIC library. The second column denotes the number of combinational gates. The number of flip-flops that are gated is listed in the fourth column, while the remainder is listed in the third column. Gating probability of each flip-flop was used to decide whether it should be gated; the probability of 0.593 was empirically used in the experiment. Column 5 represents the average gating probability of those in the fourth column.

The result after merge process is shown in the last two columns of Table I. The average gating probability of gated flip-flops necessarily decreases, which is evident by comparing it to the figure in the fifth column.

To assess factored form matching (denoted by Matching in Table II), we implemented two additional methods of simplification: using Boolean division [4] explained in Fig. 2; approximation [1], in which product terms (in sum-of-products

978-1-4673-0145-9/12/$31.00 ©2012 IEEE

TABLE II

NUMBER OF EXTRA GATES TO IMPLEMENT GATING FUNCTIONS. 'NO SIMPLIFICATION' SERVES AS A REFERENCE TO COMPUTE DIFFERENCE

Circuit	No simplification	Division		Approximation		Matching	
	# Extra gates	# Extra gates	Diff. (%)	# Extra gates	Diff. (%)	# Extra gates	Diff. (%)
s1423	78	52	-33.3	54	-30.8	32	-59.0
s5378	305	282	-7.5	279	-8.5	264	-13.4
s13207	284	264	-7.0	260	-8.5	232	-18.3
s15850	431	414	-3.9	366	-15.1	367	-14.8
s35932	1401	1401	-	1334	-4.8	1255	-10.4
s38584	1317	1302	-1.1	1277	-3.0	1215	-7.7
b07	30	30	-	30	-	21	-30.0
b12	300	280	-6.7	271	-9.7	236	-21.3
b17	289	272	-5.9	279	-3.5	246	-14.9
ac97	67	44	-34.3	37	-44.8	24	-64.2
i2c	184	168	-8.7	166	-9.8	153	-16.8
mem_ctrl	611	600	-1.8	512	-16.2	486	-20.5
systemcaes	287	253	-11.8	233	-18.8	225	-21.6
usbf_top	158	134	-15.2	86	-45.6	81	-48.7
wb_dma	606	538	-11.2	575	-5.1	516	-14.9
Average			-9.9		-14.9		-25.1

form of a gating function) whose probability is smaller than 0.001 are declared as don't-care set, a new gating function is submitted to two-level minimization followed by implementation in multi-level logic [10]. The number of gates to implement gating functions without any simplification, given in the second column of Table II, serves as a reference to compute the difference of the number of gates (denoted as Diff) of each simplification method.

Division achieves about 10% reduction in the number of gates, but there is wide variation (from 0% to 34%). This is expected since the existence of larger divisor relies on a chance, in particular when algebraic division is applied. Approximation achieves more reduction, but this comes at the cost of reduced gating probability. Average gating probability of gated flip-flops is 0.68 in 'No simplification', while that of Approximation is 0.64; note that Division and Matching do not sacrifice gating probability.

Matching achieves substantial reduction in gate count compared to both division and approximation. The matching takes about 10% more runtime than division method in our current implementation, ranging from 1 to 300 seconds. The majority of runtime is due to the detection of weak matches as can be expected from its complexity. Weak matches account for only 10% of total number of matches; its contribution in gate count reduction (the last column of Table II) however is 25%, which demonstrates its importance in the factored form matching.

V. CONCLUSION

Gate-level clock gating synthesis is not yet in main stream use, but is a promising methodology due to its convenience offered to designers. A new solution for simplification problem of the synthesis has been suggested. Factored form matching has been proposed to utilize existing combinational logic as much as possible; this is an extension of a prior method using Boolean division.

A few future works are identified. Runtime is a prime issue in current implementation of factored form matching. The runtime of matching is 1 to 300 seconds. More efficient implementation is needed for scalability. Circuit timing may change since some internal nodes of a combinational logic are used for gating functions, which alters their load capacitance; this should be checked during matching process. Combining both factored form matching and approximation merits an investigation to achieve more reduction in gate count.

ACKNOWLEDGEMENT

This work was supported by Mid-Career Researcher Program through NRF grant funded by the MEST (2011-0029087).

REFERENCES

[1] 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. on Design Automation of Electronic Systems*, vol. 4, no. 4, pp. 351–375, Oct. 1999.

[2] A. P. Hurst, "Automatic synthesis of clock gating logic with controlled netlist perturbation," in *Proc. Design Automation Conf.*, June 2008, pp. 654–657.

[3] E. Arbel, C. Eisner, and O. Rokhlenko, "Resurrecting infeasible clock-gating functions," in *Proc. Design Automation Conf.*, July 2009, pp. 160–165.

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

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

[6] A. Farrahi, C. Chen, A. Srivastava, G. Téllez, and M. Sarrafzadeh, "Activity-driven clock design," *IEEE Trans. on Computer-Aided Design*, vol. 20, no. 6, pp. 705–714, June 2001.

[7] S. Wimer and I. Koren, "The optimal fan-out of clock network for power minimization by adaptive gating," *IEEE Trans. on VLSI Systems*, 2012.

[8] F. Theeuwen and E. Seelen, "Power reduction through clock gating by symbolic manipulation," in *Proc. Symp. Logic and Architecture Design*, Dec. 1996, pp. 184–191.

[9] G. Hatchtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[10] E. M. Sentovich and et al., "SIS: a system for sequential circuit synthesis," May 1992, Tech. Rep. UCB/ERL M92/41.

[11] "OpenCores," http://www.opencores.org/.

[12] Synopsys, *Design Compiler User Guide*, Sept. 2008.