A Novel Sequential Circuit Optimization with Clock Gating Logic

Yu-Min Kuo Shih-Hung Weng Shih-Chieh Chang
Department of CS, National Tsing Hua University, Hsinchu, Taiwan
{ymkuo, shweng, scchang}@cs.nthu.edu.tw

Abstract—

To save power consumption, it has been shown that the clock signal can be gated without changing the functionality under certain clock-gating conditions. We observe that the clock-gating conditions and the next-state function of a Flip-Flop (FF) are correlated and can be used for sequential optimization. We show that the implementation of the next-state function of any FF can be just an inverter if the clock signal is appropriately gated. By exploiting the flexibility between the clock-gating conditions and the next-state function, we propose an iterative optimization technique to minimize the overall timing.

1. Introduction

A sequential circuit consists of combinational elements to compute next states and sequential elements such as Flip-Flops (FF) to store the current states. When a clock pulse arrives, a circuit re-evaluates the states. Normally, clock signals are delivered to all FFs periodically; however, it has been shown that it is not necessary to deliver a clock pulse to an FF in every clock cycle. Techniques such as clock gating [1][2][3][4][5][6][7][8][9][10][11][12] shut off clock signals when a circuit is in idle state or when FFs need not change their states so as to save the power consumption.

In this paper, we propose novel flexibility for sequential optimization using the concept of clock gating, the novel flexibility of which is completely different from traditional sequential don't cares. The new structure is shown in Figure 1 where the clock of an FF is gated when the clock-gating function is asserted and the next-state function provides the next-state value of the FF. In addition, a *CG Cell* consisting of a latch and an *AND* gate is used to avoid glitches. There should also be logic circuits computing the primary outputs which are omiited in the figure.

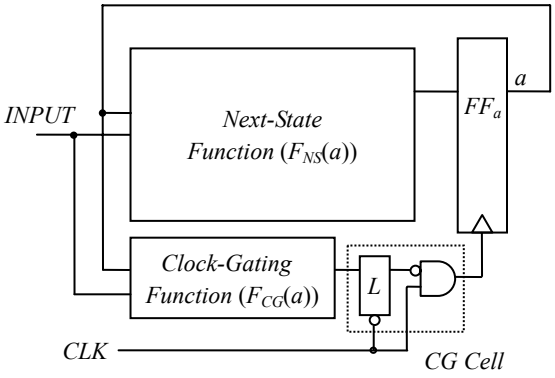

Figure 1: Basic Structure of a Sequential Circuit with the Clock Gating

Current State			Next State		
a	*b*	*c*	*a*	*b*	*c*
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Figure 2: State Transition Table of a 3-Bit Counter

We illustrate how the new architecture works by a 3-bit counter. Figure 2 shows the state transition table of a 3-bit counter where *a* is the 3rd bit and FF_a is the corresponding storage element. When the current state (*a*,*b*,*c*) is (0,0,0), the next state will be (0,0,1) and the state of FF_a will not change its value. According to the state transition table in Figure 2, when the current state (*a*,*b*,*c*) is equal to one of states in S = {(0,0,0), (0,0,1), (0,1,0), (1,0,0), (1,0,1), (1,1,0)}, the state of FF_a does not change. Since FF_a does not change its value for those current states in S, a clock

pulse dose not need to arrive at FF_a and can be gated. We can derive a Boolean function $b'+c'$ to characterize current states in S. Since no clock pulse is delivered to FF_a when the condition $b'+c'$ is true, we can randomly assign the output of the next-state function for those conditions; in other words, we can use $b'+c'$ as the don't-care function to minimize the original next-state function $ab'+ac'+a'bc$. The result of minimization is an inverter, a'.

Still, the Boolean conditions to determine when there should be a clock pulse can be complicated for some sequential circuits. In this paper, we present theoretical foundations and efficient heuristics for building sequential circuits consisting of the clock-gating functions and the next-state functions. Conceptually, our algorithm transforms some combinational logics to the clock-gating function. And in many cases, the transformation can improve the overall efficiency of a circuit. We have performed our experiments on a set of benchmark circuits and obained on the average 13.99% timing improvement in TSMC 0.13 μm library.

The remainder of this paper is organized as follows. Section 2 shows the overall algorithm and the process flow. Section 3 presents the experimental results. Section 4 concludes this paper.

2. Logic Synthesis Using the Clock Gating Function

2.1 Basic definitions and key facts

We first present two simple but very important facts about the relationship between the clock-gating function and the next-state function for a single FF. The facts form the foundations of all following equations and heuristics. Note that the clock of an FF is shut off when the clock-gating function is asserted in Figure 1.

FACT1: When the clock of an FF is gated, the next state of the FF remains the same regardless of whether the next-state function is zero or one. Therefore, the on-set of the clock-gating function can be the don't-care set for the next-state function [13].

FACT2: When the next state and the current state value of the FF are the same, the FF remains its state value regardless of whether the clock is gated or not. Therefore, the conditions when the next-state value is equal to the current state of the FF can be the don't-care set for the clock-gating function.

In the following, we use F_{CG} to denote the clock-gating function and use F_{NS} to denote the next-state function. Let us consider FF_a in Figure 1 where $F_{NS}(a)$ is the next-state function and $F_{CG}(a)$ is the clock-gating function of FF_a. In addition, signal a is the output of FF_a as well as an input of the $F_{NS}(a)$ and $F_{CG}(a)$. To describe the facts precisely, we present them in mathematical form.

FACT1: When $F_{CG}(a) = 1$, $F_{NS}(a)$ can be 0 or 1. Thus, the on-set of $F_{CG}(a)$ is the don't-care set for $F_{NS}(a)$.

FACT2: When $(a \equiv F_{NS}(a)) = 1$, $F_{CG}(a)$ can be 0 or 1. Thus, the on-set of $(a \equiv F_{NS}(a))$ is the don't-care set for $F_{CG}(a)$ where the symbol "≡" represents the Boolean operator XNOR.

2.2 The simplest implementation of F_{NS} and F_{CG}

We can use don't-care conditions in FACT1 and FACT2 to minimize F_{NS} and F_{CG} respectively. In the following, we discuss a very efficient implementation for F_{NS} and F_{CG}. To distinguish between the original next-state function and the newly generated next-state function, we use $F_{ORI\text{-}NS}$ to denote the original implementation of the next-state function without the clock gating.

Without going into a complicated proof, it is easy to show that the simplest implementation of F_{CG} is that $F_{CG} = 0$, because there exists a legal solution that the clock is not gated at all. When $F_{CG} = 0$, according to FACT1, there is no don't care for F_{NS}, so that $F_{NS} = F_{ORI\text{-}NS}$.

Now, we are interested in finding the simplest implementation of F_{NS}. The following theorem shows that $F_{NS}(a)$ after the optimization is a simple literal a'.

Theorem 1: Let the on-set be $F_{ORI\text{-}NS}(a)$ and the don't-care set be $(a \equiv F_{ORI\text{-}NS}(a))$. There exists a don't-care assignment such that the implementation of $F_{NS}(a)$ is a'.

Proof: Omitted.

We have shown that the simplest implementation for F_{CG} is 0 and an implementation for F_{NS} can be just an inverter. However, if the simplest implementation for one of them is chosen, there will be no flexibility for the other

function.

2.3 Heuristic minimization for F_{NS} and F_{CG}

In the traditional design flow, we always choose the simplest implementation of $F_{CG} = 0$. Therefore, the next-state function $F_{ORI\text{-}NS}(a)$ does not have any don't cares. In this section, we would like to explore other alternatives of implementations for F_{NS} and F_{CG}. According to FACT1 and FACT2, both F_{NS} and F_{CG} are correlated. Choosing one implementation may affect the don't-care set of the other.

Our idea is to use an iterative approach to simplify both functions. Before describing our iterative procedure, we rewrite FACT1 and FACT2 by the following two equations.

$$F_{NS}^*(a) <= \{\text{on-set} = F_{NS}(a), \text{dc-set} = F_{CG}(a)\} \quad \text{EQ(1)}$$

$$F_{CG}^*(a) <= \{\text{on-set} = F_{CG}(a), \text{dc-set} = (a \equiv F_{NS}(a))\} \quad \text{EQ(2)}$$

where $F_{NS}^*(a)$ is the simplified next-state function and $F_{CG}^*(a)$ is the simplified clock-gating function.

Our iterative optimization requires initial Boolean functions of F_{NS} and F_{CG}. Since we have the initial next-state function $F_{NS} = F_{ORI\text{-}NS}(a)$, we need to find an appropriate initial clock-gating function $F_{CG} = F_{INI\text{-}CG}$. How $F_{INI\text{-}CG}$ is determined will be described later. Let us assume $F_{INI\text{-}CG}$ is available. According to EQ(1), we can use $F_{NS} = F_{ORI\text{-}NS}$ as the on-set and $F_{CG} = F_{INI\text{-}CG}$ as the don't-care set to obtain a simplified F_{NS}^*. In other words,

$$F_{NS}^*(a) <= \{\text{on-set} = F_{ORI\text{-}NS}(a), \text{dc-set} = F_{INI\text{-}CG}(a)\}.$$

Once the simplified next-state function F_{NS}^* is determined, according to EQ(2), we can use $F_{CG} = F_{INI\text{-}CG}$ as the on-set and $(a \equiv F_{NS}^*(a))$ as the don't-care set to obtain simplified F_{CG}^*.

$$F_{CG}^*(a) <= \{\text{on-set} = F_{INI\text{-}CG}(a), \text{dc-set} = (a \equiv F_{NS}^*(a))\}.$$

After that, we can assign $F_{CG} = F_{CG}^*$ and $F_{NS} = F_{NS}^*$ and then use EQ(1) and EQ(2) to iteratively simplify both functions again.

The above iterative process requires an initial clock-gating function. We now describe our selection for the initial function of F_{CG} called $F_{INI\text{-}CG}$. In our heuristic, we choose either of the following two Boolean functions.

$$F_{INI\text{-}CG}(a) = a' * ((a \equiv F_{ORI\text{-}NS}(a))|_{a=0}). \quad \text{EQ(3)}$$

$$F_{INI\text{-}CG}(a) = a * ((a \equiv F_{ORI\text{-}NS}(a))|_{a=1}). \quad \text{EQ(4)}$$

where the symbol "|" denotes the cofactor operator. In our heuristic, if the number of fanouts of variable a' in F_{NS} is larger than the number of fanouts of variable a, we choose EQ(3); otherwise, we choose EQ(4). It is because EQ(3) has better chance to minimize equations containing a' while EQ(4) is better for a.

The iterative heuristic method can be extended to simplify a whole circuit for the timing optimization. First, we perform a trial run of delay optimization to obtain the critical FFs whose inputs or outputs are in the "long" paths. The long paths can be defined as those paths whose path delay is less than 20% of the delay of a longest path. We then choose several FFs that are the end points of the critical paths and then apply our iterative heuristic method to these FFs for at most k times of iteration loops where k is chosen to be 5 in our experiments.

3. Experimental Results

We implemented the method in Section 2.3 and applied it to iscas89 sequential benchmark circuits. We use TSMC 0.13μm library as the technology libraries and Synopsys Design Compiler® for timing optimization. Table 1 shows the results of TSMC library. Columns 1 and 2 show the name and the number of FFs of a benchmark circuit. Column 3 shows the longest delay after using SIS script.delay. Column 4 shows the longest delay of the whole circuit. Column 5 shows the longest delay of only the next-state functions F_{NS}, and column 6 shows the longest delay of only the clock-gating function F_{CG}. In addition to F_{NS} and F_{CG}, our resulting circuit also consists of logics computing primary outputs. Therefore, the longest delay of the whole circuit in column 4 may be larger than the longest delay of F_{NS} and F_{CG}. Column 7 shows the ratio of timing improvement. Columns 8 and 9 show the area results before and after timing optimization, respectively. Column 10 shows the ratio of area penalty. The run time of our algorithm is shown in the last column. On average, the timing improvement is about 13.99% in the TSMC library. In addition, a circuit also consists of logic to compute primary outputs. If critical paths of a circuit are located in the paths to primary outputs, our timing optimization technique will not be efficient. It is because our technique

Table 1: Results using Synopsys Design Compiler® with TSMC 0.13 μm library

Circuit	# FFs	Timing (ns)					Area (μm^2)			Runtime (sec.)
		Original	Optimized			Improvement (%)	Original	Optimized	Overhead (%)	
			Whole	F_{NS}	F_{CG}					
s27	3	0.11	0.10	0.10	0.10	9.09	156.16	151.07	-3.26	7.1
s820	5	0.88	0.66	0.63	0.61	25.00	4877.13	5302.35	8.73	58.0
s832	5	0.89	0.69	0.63	0.61	22.47	5103.24	5229.24	2.48	57.9
s1494	6	1.38	1.22	1.19	1.01	11.59	3829.33	4114.50	7.45	22.3
s510	6	0.75	0.67	0.46	0.67	10.67	3951.76	3805.40	-3.70	47.9
s208	8	0.61	0.55	0.02	0.25	9.84	1111.02	758.42	-31.74	37.2
s344	15	1.17	0.91	0.89	0.83	22.22	909.81	1351.13	48.51	60.9
s349	15	1.14	0.90	0.89	0.78	21.05	906.41	1378.29	52.06	71.3
s641	19	0.83	0.75	0.67	0.59	9.64	6879.00	7770.47	12.96	89.1
s526	21	0.59	0.53	0.48	0.42	10.17	3206.65	2780.87	-13.28	95.3
s526n	21	0.59	0.53	0.48	0.42	10.17	3206.65	2780.87	-13.28	95.2
s1423	71	3.61	3.02	2.68	2.18	16.34	3846.31	4056.79	5.47	1341.3
s9234	211	1.94	1.87	1.83	1.73	3.61	11124.76	12092.28	8.70	961.3
AVG						13.99			6.24	

can only reduce the delay to FFs.

4. Conclusions

In this paper, we first propose the flexibility provided by the concept of the clock gating. Then, we present the theoretical facts and efficient heuristics to optimize a sequential circuit consisting of the clock-gating functions and the next-state functions. On average, the timing of sequential benchmark circuits can be reduced about 13.99% in the TSMC library.

REFERENCES

[1] P. Babighian, L. Benini, and E. Macii, "A Scalable Algorithm for RTL Insertion of Gated Clocks Based on ODCs Computation," *IEEE Trans. on CAD*, vol. 24, no. 1, Jan 2005.

[2] M. Alidina, J. Monteiro, S. Devadas, and A. Ghosh, "Precomputation-Based Sequential Logic Optimization for Low Power," *Proc. of ICCAD*, pp. 74-81, 1994.

[3] L. Benini, and G. De Micheli, "Automatic Synthesis of Low-Power Gated-Clock Finite-State Machines," *IEEE Trans. on CAD*, vol. 15, no. 6, Jun. 1996.

[4] M. Müch, B. Wurth, R. Mehra, J. Sproch, and N. When, "Automating RT-Level Operand Isolation to Minimize Power Consumption in Datapaths," *Proc. of DATE*, pp. 624-633, 2000.

[5] V. Tiwari, S. Malik, and P. Ashar, "Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design," *Proc. of ISPLED*, pp. 221-226, 1995.

[6] H. Kapadia, L. Benini, and G. De Micheli, "Reducing Switching Activity on Datapath Buses with Control-Signal Gating," *IEEE J. of Solid-State Circuits*, vol. 34, no. 3, March 1999.

[7] M. Onishi, A. Yamada, H. Noda, and T. Kambe, "A Method of Redundant Clocking Detection and Power Reduction at RT Level Design," *Proc. of ISLPED*, pp. 131-136, 1997.

[8] L. Benini, G. De 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 Electronic Systems*, vol. 4, no. 4, pp. 351-375, 1999.

[9] G. Lakshminarayana, A. Raghunathan, K. S. Khouri, N. K. Jha, and S. Dey, "Common-Case Computation: A High-Level Technique for Power and Performance Optimization," *Proc. of DAC*, pp 56-61, 1999.

[10] Y. Luo, J. Yu, J. Yang, and L. Bhuyan, "Low Power Network Processor Design Using Clock Gating," *Proc. of DAC*, pp. 13-17, 2005.

[11] H. M. Jacobson, "Improved Clock-Gating through Transparent Pipelining," *Proc. of ISLPED*, pp. 26-31, 2004.

[12] N. Banerjee, K. Roy, H. Mahmoodi, and S. Bhunia, "Low Power Synthesis of Dynamic Logic Circuits Using Fine-Grained Clock Gating," *Proc. of DATE*, pp. 6-10, 2006.

[13] A. P. Hurst, "Automatic Synthesis of Clock Gating Logic with Controlled Netlist Perturbation," *Proc. of DAC*, pp. 654-657, 2008