

CAD Tools for Early Timing Closure in System-on-a-Chip Design

● Yuzi Kanazawa ● Hiroyuki Higuchi

(Manuscript received October 4, 2005)

This paper introduces two new CAD tools, “March” and “MagusMCP,” developed to partially automate the design of clock signal distribution for timing optimization and the timing constraints for systems-on-a-chip. March synthesizes flexible structures of clock distribution circuits based on flip-flop (FF) grouping and placement information. It also considers the placement and routing resource information of RAM and optimizes the delay on clock paths and clock skew. These features make it easier to satisfy timing constraints. MagusMCP automatically detects multi-cycle/false paths based on analysis that takes circuit logic into account. These tools make it possible to ease timing constraints, thus enabling early timing closure.

1. Introduction

During the design of a system-on-a-chip (SoC), timing closure is achieved by repeatedly estimating the circuit delay and optimizing circuit performance until the design satisfies the given timing constraints. Timing closure has always been an indispensable step in SoC design, but due to the steadily increasing demand for integrated circuits of higher performance and the greater impact of interconnection on timing, it is becoming one of the most difficult and time-consuming tasks to complete. Reducing the cost of timing closure is now a top priority in modern SoC design.

In addition to the demand for higher performance and the significant impact of interconnection on timing, the introduction of complex timing constraints is another major reason for the difficulty posed by timing closure. In modern SoC design, timing constraints are becoming increasingly more complex due to the requirements for low power consumption and the reuse of pre-designed standard modules known as

intellectual properties (IPs). For example, multi-phased clocking schemes such as divided clocks and clock gating techniques are commonly used to reduce power dissipation in inactive modules. In these cases, it is quite difficult for designers to manually maintain consistency between the logic/physical design, clock design, and timing constraints. Thus, the cost of generating and maintaining complex timing constraints accounts for a considerable portion of the total cost for timing closure in modern design.

This paper introduces two new CAD tools developed to handle complex timing constraints and describes the new features of these tools. The first tool is a clock tree synthesis tool called March. The second is a multi-cycle/false path analysis tool called MagusMCP.

The rest of this paper is organized as follows. Section 2 explains the background of timing constraints for timing closure. Sections 3 and 4 introduce the new CAD tools, March and MagusMCP, respectively. Section 5 summarizes our work and presents some future possibilities.

2. Timing closure problem of SoC design

The purpose of timing closure is to ensure that two kinds of constraints are satisfied: the setup and hold constraints (**Figure 1**). The setup constraint requires that a data signal must arrive at the target flip-flop (FF) before the next clock signal. Conversely, the hold constraint requires that a data signal must not arrive at the target FF before the current clock signal. When both constraints are satisfied, the signal originating from the source FF arrives at the target FF after the current clock and before the next clock, thus ensuring the correct signal is latched at the target FF.

2.1 Timing constraints under complex clock structure

The setup and hold constraints depend on the timing at which the clock signal arrives at each FF. In other words, the correct clock arrival time must be ensured to determine the correct timing constraint. The problem is that the correct clock arrival time cannot be determined until the clock tree is built.

In the traditional flow of design, this problem is solved in the following way. First, timing-driven placement is performed by assuming that every FF receives the clock at the same time (i.e., assuming zero skew). Then, the clock

tree is built so as to minimize clock skew. After the clock tree is built, a timing violation check is performed based on the correct clock arrival time. To correct any timing violations that are found, the layout must be refined. If the clock skew is sufficiently small, the portion of design that requires refinement will also be small.

However, this design flow may not always work for existing SoCs that have a complex clock structure involving multiple clock sources, a gated clock, and a divided clock. There are several reasons why such a complex clock structure is necessary. For example, a gated clock and divided clock are needed to reduce power consumption. Multiple clock sources are often used when an SoC has several reused modules, each of which operates at a different frequency. Such a clock structure can cause problems because it negates the assumption of zero skew.

For example, consider clock gating. When a module is idle, disabling the clock signal to that module can reduce the power it consumes. This technique is called clock gating.

The clock cycle in the example shown in **Figure 2** is 5 ns. The source FF and target FF receive the clock signal at the same time. The clock path to the target FF has a clock gate. When the “enable” signal at the clock gate is zero, the clock signal is disabled at the target FF. The delay between the clock gate and target FF is

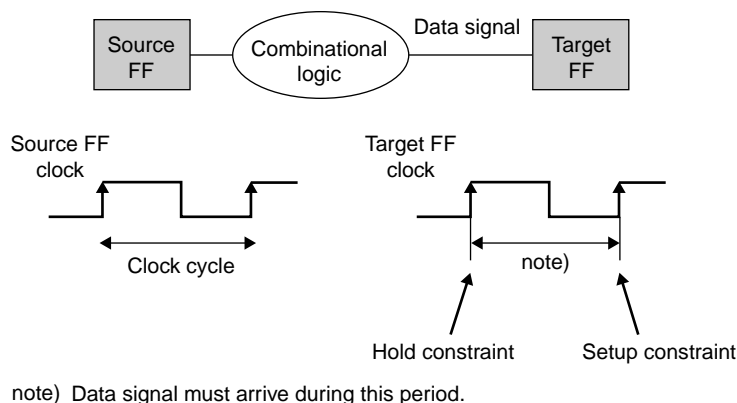
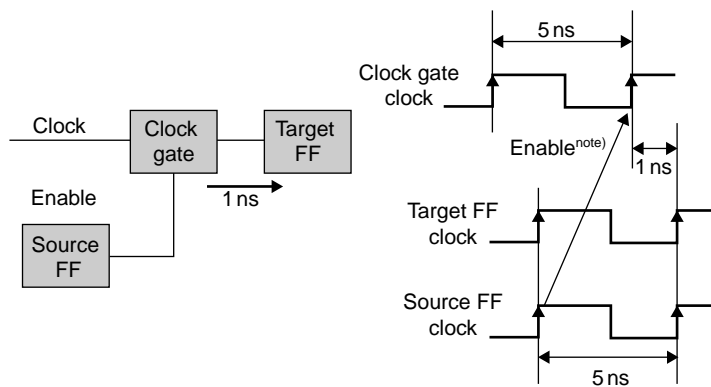


Figure 1
Setup constraint and hold constraint.



note) The Enable signal from the source FF to the clock gate must arrive at the clock gate in 4 ns. Otherwise, the clock signal cannot be controlled.

Figure 2
Setup constraint at clock gate.

1 ns. The enable signal must arrive at the clock gate before the clock, which is 1 ns earlier than the clock arrival time of the target FF. Because the clock cycle is 5 ns, the delay from the source FF to the clock gate must be less than 4 ns.

We cannot assume zero skew in this case. The clock signal first arrives at the clock gate before reaching the target FF. Thus, the target FF and clock gate cannot receive the clock signal at the same time. The designer cannot determine the correct timing constraint until the clock arrival time at each FF is known. Once the clock arrival times are fixed, the designer must repeat timing optimization to correct any new timing errors detected. Moreover, this repetition may take considerable time.

This kind of problem is becoming quite common because it occurs not only for clock gates, but also for divided clock generating circuits.

2.2 Multi-cycle paths and false path extraction

The extraction of multi-cycle paths and false paths is another issue that should be addressed in order to reduce the cost of timing closure.

A multi-cycle path in a sequential circuit is a combinational path that does not have to propagate signals in a single clock cycle. A k -cycle path enables the use of k clock cycles to propagate sig-

nals. An infinite-cycle path — also called a false path — is a path that is never activated given the circuit's functionality and the delay values of the circuit components.

Figure 3 shows an example of a multi-cycle path in a fully synchronous sequential circuit. For this circuit, the details of combinational logic A are omitted. FF1 and FF2 are flip-flops with enable input EN. FF3 and FF4 are flip-flops without an enable input. The paths between FF1 and FF2 are multi-cycle paths, as indicated by the bold lines. This can be recognized by the following analysis. FF3 and FF4 constitute a counter whose state transition is as follows: (00)/ (01)/ (11)/ (10)/ (00). The values of the enable inputs of FF1 and FF2 become 1 when the state of the counter is either (00) or (10). Therefore, the paths between FF1 and FF2 are actually 3-cycle paths.

Figure 4 shows a classic example of a false path. As you can see, the circuit has four combinational logic blocks (A, B, C, and D), for which the longest delays are 3 ns, 1 ns, 3 ns, and 1 ns, respectively. The longest path through pin D0 of MUX1 and pin D0 of MUX2 is false because control inputs C of MUX1 and MUX2 never have values of 0 at the same time (assuming the NOT gate has zero delay).

A circuit with many clock gates can have several multi-cycle paths. When a clock gate blocks

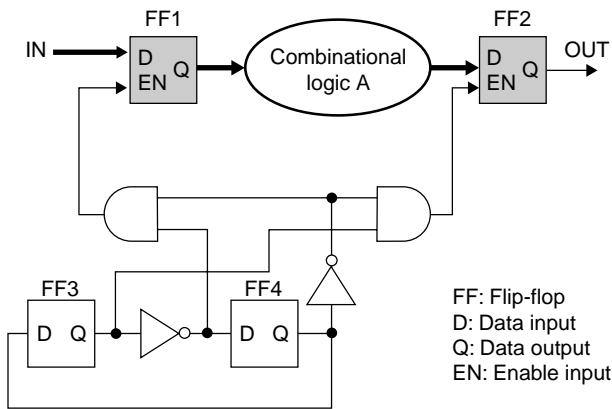


Figure 3
Example of multi-cycle path.

the clock signal, a clock-driven FF does not latch the data signal. This means the data signal need not arrive in the clock cycle. Such a circuit may have many multi-cycle paths, of which even the designer may not be aware. Detecting multi-cycle paths is important to reduce the design time. Without knowing which path is a multi-cycle path, we may needlessly spend much time optimizing its delay.

A complex clock structure, multi-cycle paths, and false paths make it difficult to obtain the correct timing constraint. Consequently, the cost of timing closure increases. To solve this problem, we have developed a clock tree synthesis tool called March and a multi-cycle/false path analysis tool called MagusMCP.

3. Clock tree synthesis tool: March

3.1 Basic idea

The increasing complexity of clock trees makes the design process even more difficult under the current flow of design where zero skew is assumed. As discussed in Section 2.1, the timing constraint of a path to the enable signal of a clock gate depends on the clock path delay from the gate to the FF. Existing clock tree synthesis tools require repeated trial-and-error testing to obtain a good result for such a circuit.

We believe that to resolve this problem, two

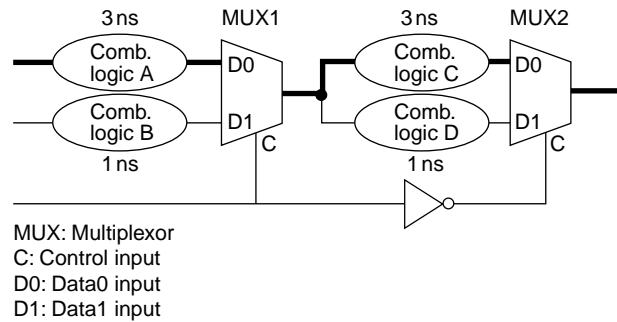


Figure 4
Example of false path.

aspects of clock tree synthesis tools must be improved. One is having an interface to handle the complex clock structure. The other is accurately estimating the clock arrival time during an early phase of layout so the designer can estimate the correct timing constraint before placement and avoid repeated timing optimization. Based on this idea, we developed a clock tree synthesis tool called March that has the following features:

- 1) March has an interface to identify FF groups. FFs included in the same group are expected to receive the clock signal at the same time. When two FFs are included in different groups, the skew between the FFs will be ignored. March allows a pin on a clock path to be included in such a group (Figure 5). The user may include the pin of a clock gate or a divided clock generating module and then use March to synchronize the group that includes a clock gate or clock generating module. Conventional clock tree synthesis tools control skew for each FF clock pin at the end of the clock path. Thus, the ability to control skew at a pin on the clock path serves to simplify the clock tree synthesis process.
- 2) March considers the placement and routing resource information of RAM. RAM uses a varying number of metal layers. When RAM uses all the metal layers, no wires can be routed over it. Conversely, wires can be routed over RAM that does not use the upper

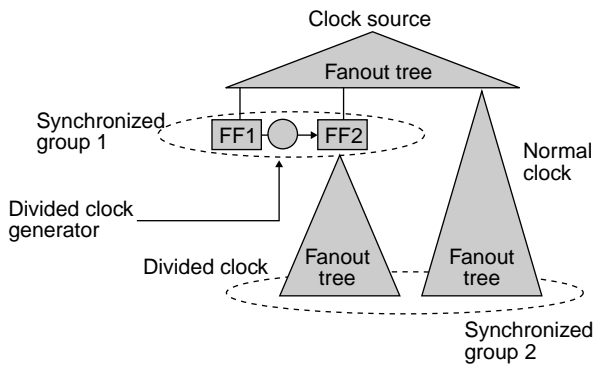


Figure 5
Example of synchronized groups.

layers. A clock tree synthesis tool that ignores such information cannot determine whether a net can be routed around or over RAM. As a result, the clock net routing estimated by such a tool is incorrect. Having both the correct placement and routing resource information are important in estimating the delay from the clock source to each FF.

- 3) March optimizes not only the clock skew, but also the delay from the clock source to each FF. Thus, a smaller delay on the clock path reduces the amount of estimation error and improves the predictability of clock arrival time.

3.2 Experimental results

To determine how much more accurate the estimated clock arrival time can be based on the idea described in Section 3.1, we applied March to a design with 139 RAMs and 173 k FFs. The delay between the clock source and each FF is 8 ns for a conventional clock synthesis tool and 5 ns for March. The error in clock arrival time estimation by conventional clock synthesis is more than 1000 ps due to routing resource estimation error and a longer delay on the clock path, while the estimation error of March is less than 200 ps.

Thus, given the more accurate estimation of clock arrival time, March is instrumental in solving the timing closure problem caused by

increased clock complexity.

4. Multi-cycle/false path analysis tool: MagusMCP

This section introduces the MagusMCP multi-cycle/false path analysis tool. This tool automatically detects multi-cycle paths and false paths by analyzing circuit functionality based on the features described in References 1) and 2). Information about the multi-cycle paths and false paths detected is added to the timing constraints and then used for more accurate static timing analysis (STA) and early timing closure. This section reviews the implication-based analysis proposed in References 1) and 2). The new contributions reported in this paper in terms of multi-cycle/false path analysis deal with extending the techniques for false path analysis, along with the results of the impact of multi-cycle/false path detection on early timing closure being reported.

4.1 Implication-based analysis for multi-cycle/false path detection

4.1.1 Multi-cycle path analysis

One of the major obstacles to multi-cycle path analysis is the excessive computation time, which is mainly due to the exponential expansion of paths in circuits. To accelerate multi-cycle path analysis, we focused on multi-cycle FF pairs instead of multi-cycle paths. A pair of multi-cycle FFs (FF_i, FF_j) is an ordered pair of FFs such that every path from FF_i to FF_j is a multi-cycle path. In Figure 3, (FF_1, FF_2) represent a pair of multi-cycle FFs. To check whether a given FF pair is a multi-cycle FF pair, we only need to examine the transition conditions of the source FF and sink FF. Specifically, we need to confirm the following condition, which we call the Multi-Cycle (MC) condition:

$$FF_i(t) \neq FF_i(t+1) \Rightarrow FF_j(t+1) = FF_j(t+2),$$

where $FF_i(t)$ denotes the value of FF_i at time t .

Implication-based analysis can be illustrat-

ed by using the circuit shown in Figure 3. Consider the FF pair (FF1, FF2) in this figure. The combinational logic part of the circuit is first expanded into two timeframes as shown in **Figure 6** to show the relationship between values at time t , $t+1$, and $t+2$. In Figure 6, the leftmost column, middle column, and rightmost column of the FFs are at time t , time $t+1$, and time $t+2$, respectively. The combinational circuit between the FFs at time t and FFs at time $t+1$ is at time t . The combinational circuit between the FFs at time $t+1$ and FFs at time $t+2$ is at time $t+1$. Each dotted-line arrow indicates an implication relationship, which means that the value at the head implies the value at the tail. To check whether a pair of FFs is a multi-cycle path, we must check whether the pair satisfies the MC condition for both rise and fall transitions at FF1. Only the case of rise transition is explained here; that is, $FF1(t) = 0$ and $FF1(t+1) = 1$. Note that these values are assigned before the implication procedure. In Figure 6, these values are circled and the other values are the results of the implication procedure. Because FF2 has an implied value 0 on enable input EN at time $t+1$, the MC condition holds true for rise transition at FF1.

This paper focuses on circuit functionality. Taking into account circuit delay and hazards in

functional multi-cycle path analysis is discussed in Reference 1).

4.1.2 False path analysis

Theoretically, a false path is considered a special case with regard to multi-cycle paths. When a pair of FFs is a false FF pair, the analysis described in Section 4.1.1 above always detects such a pair as a multi-cycle FF pair. However, false FF pairs are rare in practice. Therefore, path-based false path analysis is also required to detect many false paths. In MagusMCP, false path analysis is also based on implication. For a given path, logic values corresponding to the activating condition of each gate on the path are first assigned, and then the implication procedure is invoked. If a contradiction is found, we can conclude that the path is false because it has no activation vector.

For example, consider the path drawn as a bold line in Figure 4. The activation conditions for gates MUX1 and MUX2 are $MUX1.C = 0$ and $MUX2.C = 0$, respectively. However, this implies a conflict at the NOT gate.

Taking into account the circuit delay in functional false path analysis is discussed, for example, in Reference 3).

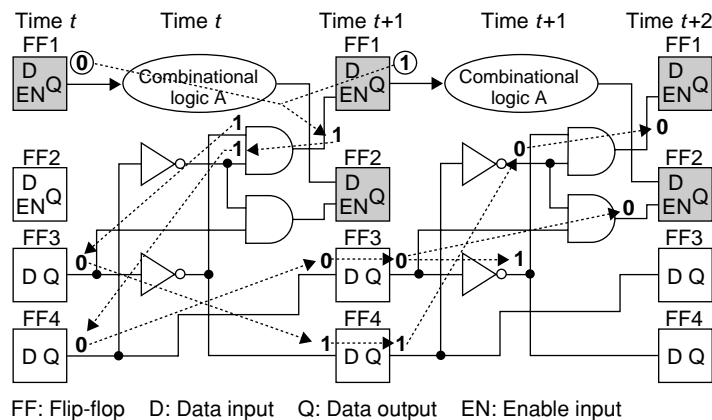


Figure 6
Example of implication procedure.

4.1.3 Automatic recognition of complicated clocking

Actual designs typically include complicated clocking. For multi-cycle path analysis, recognizing clock behavior is essential for accurate and efficient analysis. MagusMCP automatically recognizes a clock definition from the timing constraints and transforms complicated multi-phased or gated clocks into fully synchronous sequential circuits.²⁾

4.2 Experimental results in an industrial setting

We have applied MagusMCP to actual designs. **Table 1** lists the results of applying multi-cycle path analysis to an industrial design. In the table, the “Number of cells” column indicates the number of cells in the circuit. The “Number of FFs” column indicates the number of FFs. The “Number of target critical FF pairs” column is the number of target-critical timing FF pairs specified in an input file in the form of an STA timing report. The “Number of MC pairs” indicates the number of multi-cycle FF pairs detected by the analysis. “CPU time(s)” indicates the CPU time in seconds required for multi-cycle path analysis.

Table 1 indicates that more than 2% of the target-critical timing FF pairs are multi-cycle paths. The CPU time is very short. Because the

example design was in the last stage of layout, multi-cycle path detection is helpful, even though not very many FF pairs are recognized as multi-cycle paths. It should also be mentioned that just one multi-cycle FF pair may help ease the timing constraints when the pair has several paths between them.

Table 2 lists the results of applying false path analysis to another industrial design and evaluating the impact of generated false paths on negative slack. The slack at a pin is the difference between its required time and the arrival time. Negative slack means that the arrival time is longer than the required time, thus indicating a timing error. A negative slack path has negative slack at the end of the path. In Table 2, the “Number of target critical paths” column indicates the number of target critical paths specified in an input file in the format of an STA timing report. The “Number of false paths” column indicates the number of false paths detected. The “negative slack without FP” and “negative slack with FP” columns indicate, respectively, the negative slack values of the most critical path calculated by STA with and without the false paths detected by MagusMCP being specified. The “CPU time(s)” column indicates the CPU time required for false path analysis.

Table 2 indicates that more than 10% of the target-critical timing paths are false paths. The

Table 1
Results of multi-cycle path analysis.

Circuit	Number of cells	Number of FFs	Number of target critical FF pairs	Number of MC pairs	CPU time (s)
CKT-A	277k	47k	690	17	267

Table 2
Results of false path analysis.

Circuit	Number of cells	Number of FFs	Number of target critical paths	Number of false paths	Negative slack (ps)		CPU time (s)
					without FP	with FP	
CKT-B	582k	73k	39881	4376	-1674	-1535	1731

amount of CPU time needed to analyze about 40000 paths is not so long. Regarding the impact of generated false paths on negative slack, false path generation improves negative slack by more than 10%. During this experiment, we also observed that the most critical path is quite likely to be false. These results suggest that automatic false path generation can significantly improve negative slack and accelerate timing closure.

5. Conclusion

We have introduced two new CAD tools for early timing closure in SoC design. The March clock tree synthesis tool can take into account complex timing constraints from early stages of design, thus enabling clock path delay to be accurately estimated from the beginning of layout. The MagusMCP multi-cycle/false path analysis tool automatically detects timing exception paths and improves negative slack for early timing closure. Experimental results reveal that more than 2% of target-critical timing FF pairs are multi-cycle paths and that false path generation improves negative slack by more than 10%.

References

- 1) H. Higuchi: An Implication-based Method to Detect Multi-Cycle Paths in Large Sequential Circuits. 39th ACM/IEEE Design Automation Conference, June 2002, p.164-169.
- 2) H. Higuchi and Y. Matsunaga: Enhancing the Performance of Multi-Cycle Path Analysis in an Industrial Setting. Asia South Pacific Design Automation Conference, January 2004, p.192-197.
- 3) D. Brand and V. S. Iyenger: Timing Analysis using Functional Analysis. *IEEE Transactions on Computers*, **37**, 10, p.1309-1314 (1988).



Yuzi Kanazawa, Fujitsu Laboratories Ltd.

Mr. Kanazawa received the B.S. degree in Mathematical Engineering and Information Physics, and the M.S. degree in Information Engineering from the University of Tokyo, Japan in 1988 and 1990, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1990 and has since been engaged in research and development of operating systems and

CAD for digital systems. He is a member of the Information Processing Society of Japan (IPSJ).

E-mail: ykanazawa@jp.fujitsu.com



Hiroyuki Higuchi, Fujitsu Laboratories Ltd.

Mr. Higuchi received the B.S. and M.S. degrees in Information Science from Kyoto University, Japan in 1991 and 1992, respectively, and obtained a Ph.D. degree in Computer Science and Communication Engineering from Kyushu University, Fukuoka, Japan in 2005. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1993 and has since

been engaged in research and development of CAD for digital systems. From August 1998 to August 1999, he was a visiting researcher at the University of Colorado in Boulder where he worked with the department of Electrical Engineering and Computer Science. He is a member of the Association for Computing Machinery (ACM); Institute of Electrical and Electronics Engineers (IEEE); Institute of Electronics, Information and Communication Engineers (IEICE); and Information Processing Society of Japan (IPSJ).

E-mail: h-higuchi@jp.fujitsu.com