Cycle Error Correction in Asynchronous Clock Modeling for Cycle-Based Simulation

Junghee Lee and Joonhwan Yi

Telecommunication R&D Center
Samsung Electronics
{junghee77.lee, joonhwan.yi} @ samsung.com

***Abstract*— As the complexity of SoCs is increasing, hardware/software co-verification becomes an important part of system verification. C-level cycle-based simulation could be an efficient methodology for system verification because of its fast simulation speed. The cycle-based simulation has a limitation in using asynchronous clocks that causes inherent cycle errors. In order to reuse the output of a C-level cycle-based simulation for the verification of a lower level model, the C-level model should be cycle-accurate with respect to the lower level model. In this paper a cycle error correction technique is presented for two asynchronous clock models. An example design is devised to show the effectiveness of the proposed method. Our experimental results show that the fast speed of cycle-based simulation can be fully exploited without sacrificing the cycle accuracy.**

I. Introduction

Ever increasing complexity of systems-on-a-chip (SoCs) makes the verification harder and harder [1]. Co-simulation/verification of hardware and software is now widely recognized as an important and viable verification approach [1].

Traditional event-driven simulator has limitations on run-time performance [1]. Cycle-based simulation can be an alternative solution for system level co-verification because of its fast simulation speed. System-level simulation usually does not require detail signal transition information but requires fast simulation speed due to huge amount of simulation data. A cycle-based simulator evaluates signal values once at an active clock edge instead of evaluating whenever a signal transition occurs, which can significantly accelerate the simulation speed.

One of the challenging problems in cycle-based simulation is to handle multiple asynchronous clocks [7] because the simulation is synchronized by a single clock. To our knowledge, all of cycle based simulation researches including [2][3][4] constrain their applications to synchronous circuits. Modern SoCs, however, often have multiple asynchronous clocks. In order to support multiple asynchronous clocks in cycle-based simulation, asynchronous clocks can be approximated as synchronous clocks —, two approximations, namely early- and late-edge models, are introduced in Section III. In this case, inherent clock cycle errors compared to RTL simulation are inevitable. More details are discussed in Section III

The accuracy of asynchronous clock modeling is especially important in verification vector reuse between C-level and register transfer level (RTL) as shown in Fig. 1. Note that C-level modeling usually sacrifices timing accuracy in some degree to achieve the faster simulation speed. Murali [5] proposed a verification framework by using a C simulator that drives stimulus for an RTL circuit. Whereas Murali deals with the signal conversion from C-level to RTL, we are focusing on the clock cycle accuracy of the C-level model compared to RTL model. It is difficult to automate the verification reuse if the C-level model is not cycle-accurate. Kirk [6] suggested a technique to reuse C simulation outputs as a testbench for RTL verification. To overcome the cycle errors between C-level and RTL, both C and RTL models need to have extra handshaking signals that, of course, cause area and pin count overhead. If the model executed on the C simulator is developed as a cycle-accurate level model, the output of the simulator can be directly used without any modification.

A circuit with single clock can be easily modeled cycle accurately in C-level. On the contrary, a circuit with multiple (asynchronous) clocks needs to be modeled carefully to prevent cycle errors at the point of clock domain crossing even though each block is modeled cycle accurately. Cycle errors between C-level and RTL circuit models come from either the inaccurate clock models or simulation mechanisms. In this paper, we address the conditions that cycle errors occur, and propose a technique to correct the cycle errors. Note that synchronizers are assumed to be at the signal paths of clock domain crossing, which is very reasonable and practical assumption. When a signal is transferred across clock domains, there exists a metastability problem [8]. To overcome the metastability, the synchronizers are widely used [11].

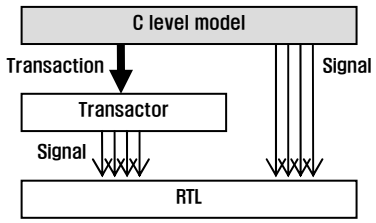

Fig. 1 Using a C level model as testbench of RTL

0-7803-9451-8/06/$20.00 ©2006 IEEE.

We first introduce basic concepts of cycle based simulation and clock in Section II. Then, we anayze the conditions that cycle errors occur and propose a correction method in Section III. In Section IV, the cycle errors due to simulation mechanisms are discussed. Experimental results are shown in Section V and we conclude the paper in Section VI

II. Preliminary

In cycle-based simulation all signals are evaluated only at an active edge of a reference clock that is called a *simulation clock*. Note that the simulation clock is the only built-in clock in cycle-based simulation. In order to simulate concurrency of hardware, the signal evaluation at an active clock edge is divided into a few phases. For cycle-based simulation, mainly two simulation mechanisms are used.

First, there is a message-passing mechanism [9] where an evaluation is achieved through 3 phases: a phase to communicate messages as scheduled (P1), a phase to update the signal values of resources according to the messages received at P1 (P2), and a phase to communicate messages generated at P2 (P3). Next, there is a communication and update mechanism [10] where an evaluation is achieved through 2 phases: communicate phase and update phase. Comparing to the message-passing mechanism, the *communicate phase* corresponds to P1 and the *update phase* corresponds to P2.

Consider the C-level circuit model C^C that models the RTL circuit C^{RTL}. Assume that signal (or net) S^{RTL} in C^{RTL} corresponds to signal S^C in C^C. When S^{RTL} is triggered by a clock CK^{RTL}, S^C is triggered by a clock CK^C modeling CK^{RTL}. Let $CK^{RTL}(n)$ denote the n-th active edge of the CK^{RTL} and $CK^C(m)$ denote the m-th active edge of CK^C. When $CK^C(m)$ corresponds to $CK^{RTL}(n)$, the relationship between m and n is determined by the clock model as described in Section III.

We take an RTL circuit model as a reference for calculating the cycle error of its C-level model because the physical behavior of a signal crossing clock domains may have non-deterministic cases like metastability [8]. As addressed before, in order to reuse a C-level testbench for RTL verification without any modification, the C-level model should have the same behavior of its corresponding RTL model and should have no cycle error. In order to exploit the fast simulation speed of cycle-based simulation methods, it is desirable to abstract the description in C-level model as much as possible.

Because the only built-in clock in cycle-based simulation is the simulation clock CK^C_{Sim}, a clock generator is needed to generate various clocks using the simulation clock. For example, consider a clock CK^C_k that is n times slower than CK^C_{Sim}. Then, a clock generator counts the number of active edges of CK^C_{Sim}, and generates an active edge of CK^C_k at every k active edges of CK^C_{Sim}. An asynchronous clock whose period is not a multiple of the period of the simulation clock cannot be modeled such a simple way. In the next Section, we explain how to model an asynchronous clock for cycle-based simulation and the inevitable cycle errors due to the clock models. We also propose a technique to correct the cycle errors.

III. Cycle Error due to Asynchronous Clock Models

Asynchronous clocks can be modeled in various ways. Among them, three simplest models are considered here: greatest common divisor (GCD) model, early edge model, and late edge model, see Fig. 2. The number n in the right hand side of an active edge means that the edge is n-th active edge. In the *GCD* (*clock*) *model*, the period of the simulation clock (CK^C_{SimGCD}) becomes the GCD of periods of all clocks in a system. The GCD model is accurate and easy to implement but impedes the simulation speed. Let us assume that there are three clocks CK^{RTL}_{125}, CK^{RTL}_{1024}, and CK^{RTL}_{20000} with periods of 125ns, 1024ns, and 20ms. Then the GCD of the 3 clocks is 1(ns) and thus the period of the simulation clock CK^C_{SimGCD} becomes 1ns. Now, $CK^{RTL}_{125}(n) = CK^C_{SimGCD}(125n)$, $CK^{RTL}_{1024}(n) = CK^C_{SimGCD}(1024n)$, and $CK^{RTL}_{20000}(n) = CK^C_{SimGCD}(20000n)$. To generate CK^{RTL}_{125}, the clock generator makes an active clock edge at every 125 edges of CK^C_{SimGCD}. That is, 124 cycles are used just for counting to know the period of the 125ns clock not for proceeding simulation. Similarly, for CK^{RTL}_{1024}, and CK^{RTL}_{20000}, 1023 and 19999 cycles are wasted in vain.

The overhead caused by the wasted cycles is imposed only on the clock generator. Although the portion of the clock generator in a circuit system is small, the performance degradation due to the GCD model is not because the clock generator is computed at every active edge of the simulation clock. In the example above, even the fastest clock of 125ns period in the system uses only 1/125 of the simulation clock edges. More than 99% cycles are wasted. Furthermore, in real system, the periods of clocks are often not an integer in nanosecond unit. As a result, the period of the simulation clock could be scaled down to the pico second unit if the GCD model is used. This time precision is nearly the same time resolution of the event-driven simulation. Therefore, we cannot exploit the fast simulation speed of cycle-based simulation.

To overcome inefficiency of the GCD model, *early edge* and *late edge models* can be used. In the early (late) edge model, the fastest clock in the system becomes the simulation

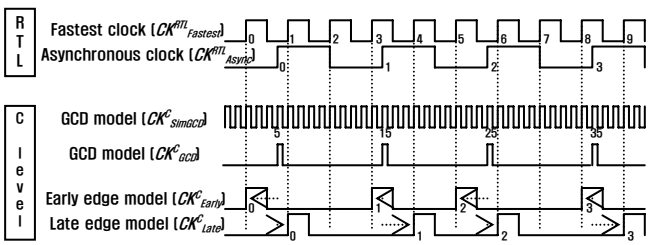

Fig. 2 Candidate asynchronous clock models

clock denoted by CK^{C}_{Early} (CK^{C}_{Late}). Other clocks are synchronized with the simulation clock by forcefully moving their edges to the preceding (following) edges of the simulation clock. The early and the late edge models are illustrated in Fig. 2. Consider the active edge $CK^{RTL}_{Async}(0)$ of an asynchronous clock between $CK^{RTL}_{Fastest}(0)$ and $CK^{RTL}_{Fastest}(1)$ of the fastest clock. If the early (late) edge model is used, $CK^{RTL}_{Async}(0)$ is moved to $CK^{RTL}_{Fastest}(0)$ ($CK^{RTL}_{Fastest}(1)$). The early and late edge models can maximize the simulation performance because they have less idle cycles than the GCD model. In the previous example of CK^{RTL}_{125}, CK^{RTL}_{1024}, and CK^{RTL}_{20000}, the period of the simulation clock becomes 125ns when early or late edge model is used. Then CK^{RTL}_{125} has an active edge every simulation clock cycle. For CK^{RTL}_{1024}, an active cycle is made every 8 or 9 simulation clock cycle thus only 7 or 8 cycles are wasted. Similarly, to generate CK^{RTL}_{20000}, an active edge is made at every 160 simulation clock cycle, thus only 159 cycles are wasted. The number of wasted cycles is drastically reduced. Note that there exists one-to-one correspondence between an RTL clock edge and a C-level clock edge in early and late edge models.

It is favorable to use either the early edge or the late edge model for clocks in order to maximize the simulation performance. But using those models causes inherent cycle errors at the point of clock domain crossing. Cycle errors caused by those asynchronous clock models occur regardless of the simulation mechanism.

The *cycle error* of CK^{C} compared to CK^{RTL} which is denoted by $CE(CK^{C})$ is α if the behavior of an RTL signal S^{RTL} at the edge of $CK^{RTL}(m)$ corresponds to that of a C-level signal S^{C} at $CK^{C}(m\pm\alpha)$. The definition of cycle error can be extended for the GCD model. But the extension is not necessary because there is no cycle error when the GCD model is used.

RTL event-driven simulation traces and their corresponding C-level cycle-based simulation traces with the late edge clock model are depicted in Figures 3,4 and 5. The C-level late edge model of the RTL asynchronous clock CK^{RTL}_{Async} is CK^{C}_{Async}. In Fig. 3, a signal is transferred from the domain of the simulation clock CK^{RTL}_{Sim} to the domain of CK^{RTL}_{Async}. The signal is received at $CK^{RTL}_{Async}(1)$ in RTL, and at $CK^{C}_{Async}(1)$ in C-level. Thus, there is no cycle error. Since $CK^{RTL}_{Async}(n)$ will be mapped to $CK^{C}_{Async}(n)$ that will be always behind the sending clock edge by the late edge model, the cycle error $CE(CK^{C}_{Async})$ is always zero. When a signal is transferred from the domain of CK^{C}_{Async} to the domain of the simulation clock as shown in Fig. 4, the signal is received at $CK^{RTL}_{Sim}(4)$ in RTL and at $CK^{C}_{Sim}(5)$ in C-level. Thus the cycle error $CE(CK^{C}_{Sim})$ is one. But if the sending edge of CK^{RTL}_{Async} is synchronized with an edge $CK^{RTL}_{Sim}(n)$ of the simulation clock, the receiving edge will be $CK^{RTL}_{Sim}(n+1)$ in both RTL and C-level. In this case, $CE(CK^{C}_{Sim})$ is zero. Thus, $CE(CK^{C}_{Sim})$ can be one or zero. Fig. 5 depicts the case that a signal is transferred between asynchronous clocks when the late edge model is applied. In this case $CE(CK^{C}_{AsyncB})$ is one at most.

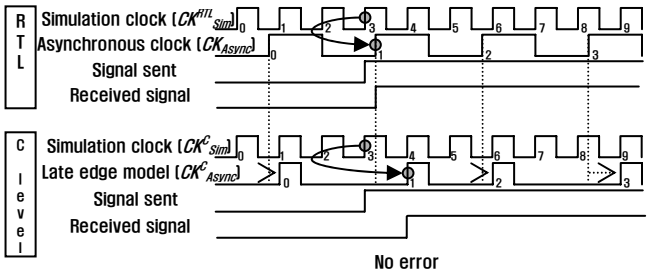

Fig. 3 The late edge model: simulation to asynchronous

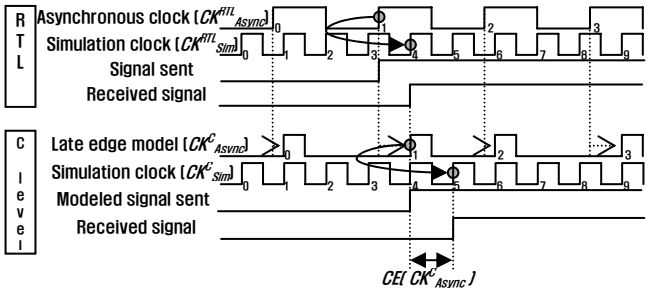

Fig. 4 The late edge model: asynchronous to simulation

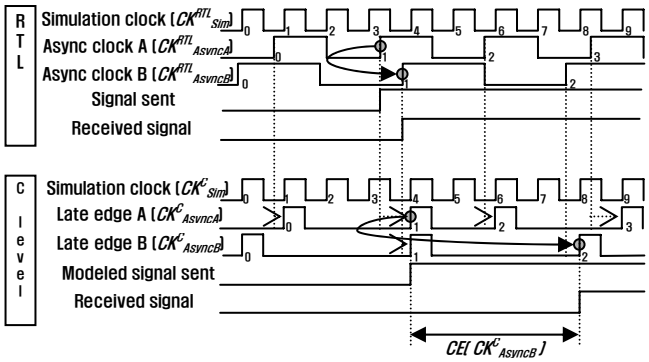

Fig. 5 The late edge model: between asynchronous

Similarly, the cycle errors due to the early edge model for various conditions can be computed. Note that the cycle errors can be formulated by the clock models and the clock domains that send and receive the signals. This is summarized in TABLE I. The cycle error is advent when the edges of the simulation and the asynchronous clocks become identical – that is, they happen at the same time – in C-level but their corresponding RTL edges are not. Now, a cycle error correction method for the late edge clock model is presented. Although only the late edge clock model is considered in this paper, it is easy to extend this discussion for the early edge clock model. As mentioned before, every active clock edge in RTL has corresponding active clock edge in C-level. To correct the cycle errors, the principle is to make signals be transferred at the corresponding edge as illustrated in Fig. 6.

TABLE I. The cycle errors due to clock models

Clock model	Clock domain From	Clock domain To	Cycle error	Illustrated in
Late edge model	Sim	Async	$CE(CK^{C}_{Async})$ = zero	Fig. 3
	Async	Sim	$CE(CK^{C}_{Sim})$ = one or zero	Fig. 4
	Async A	Async B	$CE(CK^{C}_{AsyncB})$ = one or zero	Fig. 5
Early edge model	Sim	Async	$CE(CK^{C}_{Async})$ = one or zero	
	Async	Sim	$CE(CK^{C}_{Sim})$ = zero	
	Async A	Async B	$CE(CK^{C}_{AsyncB})$ = one or zero	

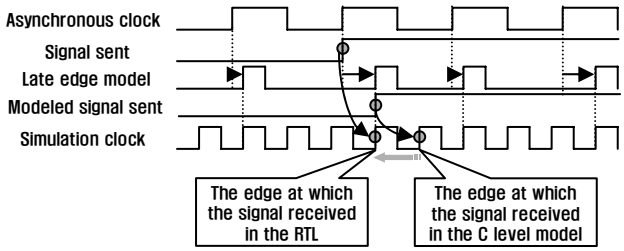

Fig. 6 Basic idea of cycle correction

We assume that synchronizers are inserted in every path crossing clock domains to prevent the malfunction of a circuit due to metastability [8]. Generally synchronizers consisting of two D-type flip-flops (DFFs) are used as shown in Fig. 7. Two DFFs cause two cycle delay at the receiving clock domain. The cycle error can be corrected if the synchronizer is operated as it has only one DFF when an one-cycle error occurs. To do so, a synchronizer like Fig. 8 is needed. The select signal should be zero when the cycle error occurs. The select signal should be provided by a controller. A way to generate the select signal is described in Section IV.

IV. Cycle Error in Communicate & Update Mechanism

The communicate and update mechanism is computationally more efficient than the message-passing mechanism because it utilizes less number of update and communication operations for a signal evaluation. Thus it is desirable to use the communicate and update mechanism to achieve the faster simulation speed in cycle-based simulation. However, unlike the message-passing mechanism, a cycle error may occur unless the modeled circuit has a single clock domain. In this section cycle errors by using the communicate and update mechanism and a correction technique of them are presented.

When the communicate and update mechanism is used for a cycle-based simulation, the communicate phase of all components is executed first to exchange signals among components. Then the update phase is executed to update shared resources. A clock CK^{C}_{Gen} is called a generated clock of a clock CK^{C}_{Sim} if CK^{C}_{Gen} is created by a logic circuit including a set of flipflops activiated by CK^{C}_{Sim}. CK^{C}_{Sim} is

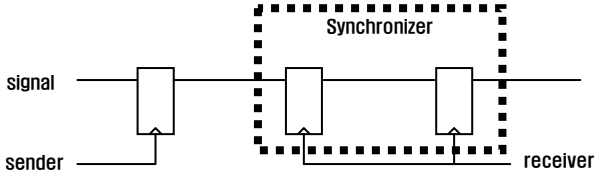

Fig. 7 A normal synchronizer

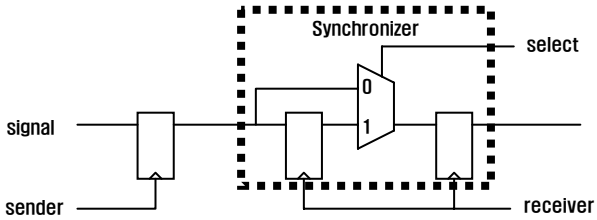

Fig. 8 Synchronizer model for correcting cycle error

called the source clock of CK^{C}_{Gen}. The active edge of CK^{C}_{Gen} is synchronized with either the rising or the falling edge of CK^{C}_{Sim}. Fig. 9 shows a usual logic design that implements CK^{C}_{Gen}, a generated clock of CK^{C}_{Sim} by the factor of two. Fig. 9 illustrates an example that a signal is transferred from the simulation clock CK^{C}_{Sim} to CK^{C}_{Gen}. D0, D1 and D2 are DFFs. $\alpha 0$ and $\alpha 1$ are inputs and $\beta 0$ and $\beta 1$ are outputs of D0 and D1, respectively. $\alpha 2$ is the input of another component receiving sig_c.

Assume that $\alpha 0$ is changing from 0 to 1 at $CK^{C}_{Sim}(0)$ during the communicate phase. A general event-driven simulator will evaluate $\alpha 0$ to 0 due to the delta delay. But a cycle-based simulator evaluates signals only at an active edge. This is an implementation issue. Regardless of the implementation, the cycle errors occur when the communicate and update mechanism is used.

The values are transferred to $\alpha 0$ and $\alpha 1$ during the communication phase of CK^{C}_{Sim} (0). Now, $\alpha 0$ and $\alpha 1$ become 1 and 0, respectively. During the update phase of CK^{C}_{Sim} (0), $\beta 0$ and $\beta 1$ are updated to 1 and 0, respectively. In the

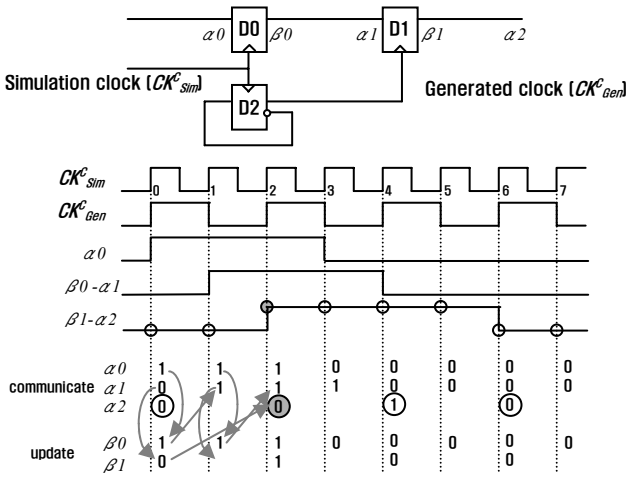

Fig. 9 Communicate and update example

communication phase of CK^{C}_{Sim} (1), $\alpha 0$ and $\alpha 1$ become 1 and 1, respectively. Then, in the update phase of CK^{C}_{Sim} (1), only $\beta 0$ is updated to 1 because D1 is not active at that time. Note that $\beta 0$ and $\beta 1$ are 1 and 0, respectively. During the communication phase of CK^{C}_{Sim}(2), $\alpha 0$, $\alpha 1$, and $\alpha 2$ become 1, 1, and 0, respectively. In the update phase, both $\beta 0$ and $\beta 1$ are updated to 1. Note that $\alpha 2$ will remain as 0 although $\alpha 2$ is supposed to be 1 unless the output value 1 of D1 is transferred to $\alpha 2$ by a communication phase. This errorneous value of $\alpha 2$ will last till the next active edge of CK^{C}_{Gen}. This is because the updated value is transferred to $\alpha 2$ during the communicate phase of the next active edge. In the message-passing mechanism [9], this error is eliminated by the last communication phase after the update phase during an evaluation.

This erroneous behavior due to the communicate and update mechanism occurs when signals are transferred between asynchronous clocks. If the receiving clock is synchronous with the sending clock and they have different periods, one extra communicate phase should be enforced after the update phase. However, if the receiving clock is asynchronous with the sending clock, the method using a synchronizer proposed in Section III can be used to correct the cycle errors.

Fig. 10 shows a flowchart of the function *GenFlag* which generates the enable flags, `Enable_A` and `Enable_B`, for asynchronous clocks and the select signal of the mux shown in Fig. 8, `Select_A,B` and `Select_B,A`, for synchronizers. The communicate and the update phases will be executed only when the enable flag is set. This function is included in the clock generator and called every edge of the simulation clock.

When *GenFlag* function is called, it first calculates the next active edge time `NextActiveEdge_Clk` of every clock in the system except for the simulation clock. Each clock *Clk* has its own active edge counter `CntEdge_Clk`. For example, CK^{RTL}_{Clk}(`CntEdge_clk`) denotes the `CntEdge_clk`-th active edge of *Clk*. `NextActiveEdge_Clk` is the time of the next active edge of *Clk* relative to the simulation clock. Reference of `NextActiveEdge_Clk` is the period of the simulation clock. If the ceiling of the next active edge time equals to that of the simulation clock, the enable flag `Enable_Clk` is set. If we use the flooring instead of the ceiling, we can implement the early edge model instead of the late edge model. For every pair of clock *A* and clock *B* there are two select signals of the mux shown in Fig. 8. `Select_A,B` is for the synchronizers from clock *A* domain to clock *B* domain. `Select_B,A` is for those from clock *B* domain to clock *A* domain. Fig. 11 shows an example illustrating function *GenFlag*.

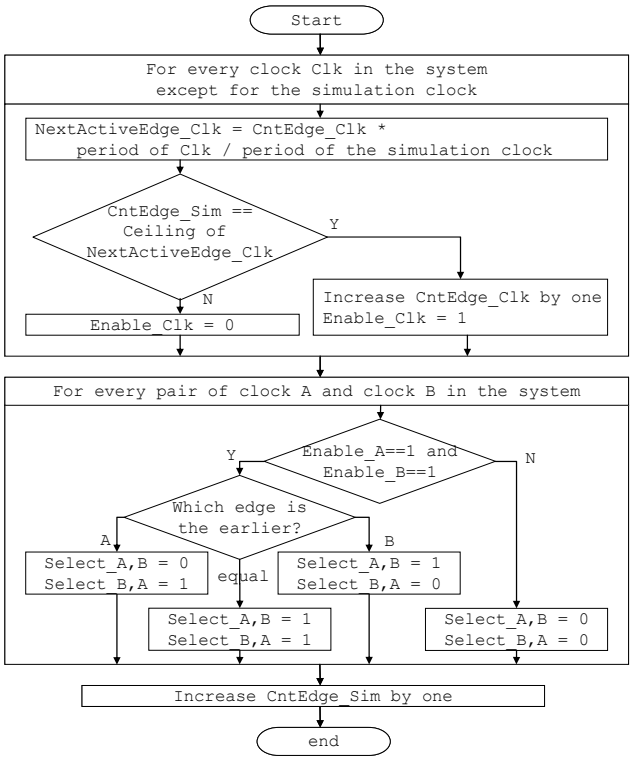

Fig. 10 Flowchart of the *GenFlag*

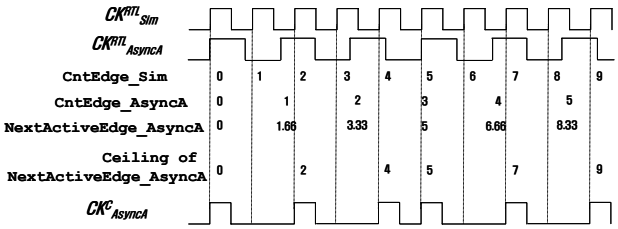

Fig. 11 An example of implementing the late edge model

After the next edge time is caculated, then it generates select signals. For every pair of clocks in the system, *GenFlag* determines which edge is earlier if both clock edges are active. To determine which edge is the earlier, it uses difference between `NextActiveEdge_Clk` and ceiling of it. The larger the difference is, the earlier the edge is.

V. Experiment

An example circuit shown in Fig. 12 is implemented to demonstrate the effectiveness of the proposed method.

The example system consists of three clock domains: CK^{C}_{Sim}, CK^{C}_{AsyncA}, and CK^{C}_{AsyncB}. Clock CK^{C}_{Sim} is the fastest clock and thus the simulation clock. Clocks CK^{C}_{AsyncA} and CK^{C}_{AsyncB} are asynchronous clocks. Each clock domain receives signals from other clock domains through a two-DFF synchronizer. Signals to be transferred to other clock domains are stable for at least one period of the receiving clock so that it is assured that the signal is transferred to the other clock domain.

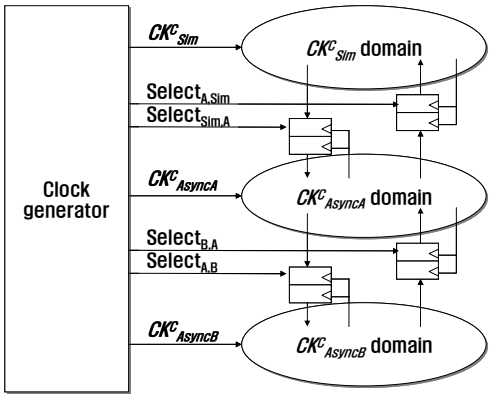

Fig. 12 An example system

When a request from CK^C_{AsyncA} domain goes to the CK^C_{Sim} domain, the CK^C_{Sim} domain returns a data to the CK^C_{AsyncA} domain with a certain delay. Then the received data is passed to the CK^C_{AsyncB} domain. The CK^C_{AsyncB} domain accumulates the received data and passed it to the CK^C_{AsyncA} domain with a certain delay. That operation is executed 1000 times in this example. The periods of CK^C_{Sim}, CK^C_{AsyncA}, and CK^C_{AsyncB} are 40ns, 110ns, and 250ns, respectively.

We simulated the sample circuit on a cycle-based simulator MaxSim [10]. In MaxSim the communicate phase and the update phase are implemented as function calls. Each component implements the communicate function and the update function and the clock generator calls the communicate functions and the update functions of components at every active clock edge. To correct cycle errors the clock generator generates select signals to let synchronizers know whether they should operate as one DFF or two DFFs.

We performed experiments to verify the cycle error correction technique and measure the accuracy and the performance. TABLE II shows the cycle accuracy of various models: RTL, the GCD model, the late edge model with cycle error correction, and the late edge model without cycle error correction. The accuracy of the late edge model with cycle error correction is 100% while the one without cycle error correction shows only 87.2% accuracy.

TABLE II Cycle accuracy of each model

	Total clock cycle used for the complete simulation	Cycle difference from RTL	Cycle accuracy
RTL	93654		
GCD	93654	0	100.0 %
Late edge model with correction	93654	0	100.0 %
Late edge model without correction	105669	12015	87.2 %

TABLE III Simulation speed of each model

	Cycle/sec	Ratio
GCD	410702	100.0 %
Late edge model with correction	588638	143.3 %
Late edge model without correction	688227	167.6 %

TABLE III shows the simulation speed of each model on a PC with a 2.53 GHz Pentium 4 processor with 512 MB memories. As can be seen, the late edge model can be implemented 1.4 times faster then the GCD model without sacrificing the cycle accuracy. Although the late edge model without cycle error correction can be 1.6 times faster than GCD, it has drawback due to the cycle inaccuracy.

VI. Conclusion

A cycle error correction technique for a cycle-based simulation with asynchronous clocks is proposed. A two-flipflop synchronizer is assumed to be inserted on every clock domain crossing path. Three clock models for cycle-based simulation are introduced: greatest common divisor (GCD), early edge, and late edge models. Although the early and the late edge clock models are more efficient for faster simulation speed, they pose inherent cycle errors with respect to register-transfer level. It is demonstrated that a 100% cycle accurate cycle-based simulation is possible with the early and late edge clock models by using the proposed technique.

References

[1] Lisa Guerra *et al.*, "Cycle and Phase Accurate DSP Modeling and Integration for HW/SW Co-Verification," *Proc. of Design Automation Conference*, pp.964-969, 1999

[2] G. Cabodi *et al.*, "Exploiting timed transition relations in sequential cycle-based simulation of embedded systems," *Proc. of Computers and Digital Techniques*, pp.305-312, 2000

[3] B.H. Yaran, B.H., D. Rahmati, and A.S. Zebardast, "Applying cycle-based simulation technique to VITAL as a VHDL gate level standard," *Proc. of Canadian Conference on Electrical and Computer Engineering*, pp.1076-1084, 2001

[4] L. Ghasemzadeh and Z. Navabi, "A fast cycle-based approach for synthesizable RT level VHDL simulation," *Proc. of International Conference on Microelectronics*, pp.281-284, 2000

[5] Murali Kudlugi, Soha Hassoun, Charles Selvidge, and Duaine Pryor, "A Transaction-Based Unified Simulation/Emulation Architecture for Functional Verification," *Proc. of Design Automation Conference*, pp.623-628, 2001

[6] Kirk Ober, "Doing Behavioral Design the Right Way Minimizes Verification," *Proc. of Design and Verification Conference and Exhibition*, 2004

[7] K. Olukotun, M. Heinrich, and D. Ofelt, "Digital system simulation: methodologies and examples," *Proc. of Design Automation Conference,* pp.658-663, 1998

[8] T.J. Gabara, G.J. Cyr, and C.E. Stroud, "Metastability of CMOS master/slave flip-flops," *Proc. of Custom Integrated Circuits Conference*, pp.29.4.1-29.4.6, 1991

[9] G. Maturana *et al.*, "Incas: a cycle accurate model of UltraSPARC," *Proc. of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp.103-135, 1995

[10] MaxSim designer's guide, AXYS design automation, Inc., 2004

[11] J. Walker and A. Cantoni, "A new synchronizer design," *Proc. of IEEE Transactions on Computers*, pp 1308-1311, 1996