Accelerating RTL Simulation by Several Orders of Magnitude Using Clock Suppression

Hannes Muhr
Vienna University of Technology
Institute of Computer Technology
hannes.muhr@tuwien.ac.at

Roland Höller
Vienna University of Applied Sciences
Technikum Wien
roland.hoeller@ieee.org

***Abstract*— In recent years designers of embedded computer systems face a tremendous growth in complexity of their systems. This, together with the fact that the used system clock frequencies rise and that the real time required to see features start up and work correctly in an embedded system also increases, let skyrocket the simulation times of event based simulation engines. Performing these simulations on register transfer level (RTL), however, is crucial to achieve functional verification of embedded computer systems. The acceleration of such event based simulations thus is the aim of the work presented in this paper. To this end a methodology called clock suppression is presented and thoroughly discussed. To underpin the feasibility and performance of this approach, evaluation results of simulation experiments for several designs will be shown.**

I. Introduction

With shrinking feature sizes of semiconductor process technologies today embedded computer systems are often made up of several microprocessors, digital signal processors, memories, and application specific circuitry, which are typically integrated on a single chip forming a system on a chip (SoC). Of those embedded systems an increasing large number is distributed in space and interconnected via a local area network, e.g. in factory automation systems, building automation, or sensor networks.

Not only the design of such complex systems is a challenge, but especially ensuring functional verification is of utmost importance. And it is time consuming, with the cost of undetected design errors tremendously increasing at the same time the later they are found. This is further aggravated by the fact that system clock frequencies of embedded computer systems keep steadily increasing, thus requiring event based simulation engines to simulate a lot more events to progress the same amount of real time for verifying the system's functions. And even worse, the new features of e.g. multi-media applications or higher-level communication protocols require simulations to span longer and longer intervals of real time.

This is why methodologies like hardware-software co-verification or virtual prototypes are in frequent use to speed up and enhance the overall quality of the verification process [1], [2], [3]. But as always, there is a trade-off between simulation speed and the detail in which the models represent the actual physical entities of the system. And often - especially for real-time systems - it is crucial to have reliable timing information available. This is the case e.g. when using an instruction set simulator for a certain processor to execute the software to be verified against not only functional but also timing constraints.

As soon as the refinement process comes to the register transfer level (RTL), where synthesis tools automatically compile the hardware description language code to a certain integrated circuit technology, simulation run times dramatically increase. Even if only parts of the embedded computer system are simulated in that detail.

A gain in simulation performance is possible, if portions of the hardware design that are coded in a hardware description language (typically VHDL or Verilog) on RTL and are run by an event-driven simulation engine, can be turned off[1] during the time they are not needed. This is exactly what the proposed clock suppression (CS) technique, which has been used for the simulation runs under consideration in this work, does.

Generally speaking, the simulation time t_{sim} depends on the number of events N_e to be simulated during a run, the number of activities N_a whose evaluation is initiated by those events, and the rate R at which the simulation engine can evaluate these activities [4]:

$$t_{sim} = \frac{N_e * N_a}{R} \tag{1}$$

Increasing R is accomplished with increasing computing power of the workstations and optimizing the simulation engines. Keeping N_e and N_a low can be achieved by higher abstraction levels for modeling, which is not always possible, as motivated earlier. Consequently the remainder of this paper is devoted to the discussion of how to minimize Equation 1 by means of shrinking N_e and is structured as follows. Section II deals with related scientific work and Section III presents the proposed clock suppression methodology in depth. Section IV shows how clock suppression works using a simple design example and Section V discusses prerequisites of this methodology. Thereafter Section VI eventually presents results of simulation experiments with real life design examples and Section VII finally concludes this paper.

[1] Just like for power saving in power sensitive devices, where clock signals literally are switched off, turning off a clock signal during an RTL simulation saves a lot of simulation events and thus can tremendously speed up the simulation.

II. Related Work

Gravenstein [4] presents several techniques to speed-up Verilog HDL simulation and among them is a similar approach to the one presented herein. It is called emulation of linear function. This solution, however, is not generic and is restricted to the optimization of counters with no sort of input condition and a constant increment of one. The approach described in the work at hand can be considered as a generalization and improvement of the emulation of linear function technique.

Ulrich [5] first introduced the term clock suppression, when sequential clock-driven circuits like transmission gates, flip-flops and registers were temporarily disconnected from their clock source and reconnected as soon as such circuits received new data input. However, its application is in the field of fault simulation, where low level circuit netlists are simulated rather than register-transfer level (RTL) models. Furthermore this approach was not able to suppress data-dependent periodic signals, as opposed to the clock suppression technique presented in this paper.

A state-based prediction of oscillating signals (a third signal state represents an oscillating signal) is introduced in [6], where a clock distribution tree is traversed before simulation begins and the clock inputs along the tree are disconnected as required. This optimization strategy has been integrated in the Creator simulator [7].

A similar approach is shown in [8], where a static analysis of the circuit is done before simulation and the results are used to augment a modified simulator engine. An average performance increase of a factor of 5 is reported.

In [9] Takamine et. al. introduced clock suppression for gate level simulations using a special purpose hardware for simulation.

Optimization strategies on a higher abstraction level are shown in [10] and [11]. In [10] Gezel, a language for synchronous hardware description, is presented, which supports co-simulation including a cycle-skip detection mechanism to optimize the co-simulation interface.

In the field of low power design for synthesis Babighian et. al. [11] introduced automatic clock gating insertion at RTL to eliminate redundant computations performed by temporarily unobservable blocks.

In [12] a method is described to suppress clock events and even aperiodic events without large modifications of the simulator. It is claimed that VHDL programs can be automatically optimized by elimination of insensitive events. However, based on the classifications stated in Park's work [12] an expert VHDL programmer can optimize VHDL models by hand.

All before mentioned papers, however, do not optimize the linear behavior of adders and counters as opposed to the method presented in this work. They therefore all have a limited upper bound of optimization potential and typically do not achieve a shortening of simulation run times in excess of more than a factor of 10.

III. Clock Suppression Methodology

Simulations at RTL or even gate level spend a considerable amount of time for the generation of the clock signal and re-evaluating clocked signals even if they don't need to be processed because no events will happen on them. On gate level about 70 percent of the overall simulation time is spent for the processing of clock events [9].

For example a one second simulation run of a single periodic signal with a frequency of 100 MHz without enabled debugging or tracing support takes 40 seconds to complete on a workstation with a computing performance of more than 1000 MIPS. It is clear that the resulting maximum event rate of 5 Mega Events/s strongly depends on the underlying simulator kernel, but still this is a limiting factor, especially for the simulation of the high clock rates seen in todays embedded computer system designs.

Listing 1 Simple counter in VHDL

```
1   psyn: process (clk, reset)
2   begin
3       if reset = '1' then
4           s <= (others => '0');
5       elsif clk'event and clk = '1' then
6           if enable = '1' then
7               s <= s + conv_unsigned(1, 1);
8           end if;
9       end if;
10  end process psyn;
```

Simple counter structures as shown in Listing 1 are very frequent in RTL designs. Especially in control-centric designs lots of counters and adders are instantiated to generate the required timing for interfaces, keep track with external events, support the internal control flow, and the like.

To further motivate the strategy of clock suppression in embedded computer system simulations the results of a survey on recently developed designs regarding the utilization of adder structures like the one in Listing 1 are summarized in Table I. It is important to note that the figures for the adder count recognize only once multiple instances of the same adder in the source code. The adder to size ratio in the last column is an indicator for the type of the design, i.e. control vs. data flow dominated ones. For example the design

Design	Description	Adder Count	Size (LUTS)	Adders / 1000 LUTs
D1	PCI UART Controller	123	3405	36.12
D2	DDR2 Controller Test	92	5224	17.61
D3	Telecom Controller	91	5527	16.46
D4	LED Controller	9	633	14.22
D5	Ethernet Switch	137	10426	13.14
D6	8051 Microcontroller	23	3133	7.34
D7	IEEE1588 Clock Core	25	5155	4.85

TABLE I
COMPARING DESIGNS WITH RESPECT TO THEIR ADDER UTILIZATION

D1 with its relatively high adder count has a greater potential for optimization through clock suppression than the design D6.

A. Introduction of Clock Suppression

For a more formal description of the clock suppression methodology we define a synchronous vectored signal s, whose representing value is of type signed or unsigned integer. s depends on a set of synchronous input signals I

$$s = s(I), I = i_1, i_2, ..., i_n. \tag{2}$$

Note, that s itself could be part of I. This property is essential for the following optimization strategy, since it focuses on the linear behavior of counter and accumulator structures as exemplified in line 7 of Listing 1.

We further define a complete set of output conditions OC for s

$$OC(s) = oc_1(s), oc_2(s), ..., oc_n(s), \tag{3}$$

which are utilized as input logic for subsequent signals. An output condition means any signal or condition that depends on the synchronous vectored signal s. These output conditions depend on s and possibly additional other signals. $OC(s)$ has to be complete in that way as every subsequent, on s depending condition must be listet herein.

Events on the clock signal can be suppressed if and only if none of the output conditions is fulfilled ($oc_1(s) = oc_2(2) = ...oc_n(s) = false$). Thus a change of (i.e. an event on) signal s won't cause further events in circuit paths to which s is an input – generated events on s do not propagate.

After times where the clock signal has been suppressed, the reactivation value of s must be calculated. To this end detailed knowledge of the functional behavior of s is necessary. This is why counters and, as a generalization, adders are well suited for this approach. Their behavior is predictable, which offers the following optimization strategy.

The derivation of the time interval Δt, during which CS is applied, starts from the definition of the differential for computing the value of s for a future reactivation.

$$s_{t+\Delta t} = s_t + s'_t \Delta t, \tag{4}$$

with s'_t being the gradient of s at time t and assuming s'_t to be constant within the simulation time interval $[t, t+\Delta t]$, and $s_{t+\Delta t}$ denoting the value of signal s at simulation time $t+\Delta t$. Rewriting of Equation 4 results in an equation for the timeout Δt for continuous systems

$$\Delta t = \frac{(s_{t+\Delta t} - s_t)}{s'_t}, \tag{5}$$

which has to be adapted for discrete systems with a granularity of t_{clk} to

$$\Delta t = [(s_{t+\Delta t} - s_t)\ div\ \Delta s_t]t_{clk}. \tag{6}$$

Here the operator div stands for the integer division without remainder and s'_t is replaced by the increment Δs_t. This timeout is the nearest point in future where the clock signal has to be activated depending on the value of s for a reactivation in the future $s_{t+\Delta t}$. The special case $\Delta s_t = 0$ leads to a division by zero and must be overruled to a $\Delta t = \infty$, i.e. when one summand of an adder function is zero no clock activation is necessary.

The value of s after a period of clock suppression is calculated to

$$s_{t+\Delta t} = s_c - (s_c - s_t)\ mod\ \Delta s_t, \tag{7}$$

where s_c is the value of s for the condition to which the timeout has been calculated. If an input event in I occurred before the timeout has expired, the new value of s will be calculated to

$$s_{t+\Delta t} = s_t + (\Delta t\ div\ t_{clk})\Delta s_t. \tag{8}$$

The clock signal must be reactivated if any of the following events occur:

- a signal from the set of input signals I changes
- an output condition $OC(s)$ changes
- the calculated timeout expires

B. Clock Suppression Algorithm

The pseudocode for the algorithm for clock suppression is shown in Listing 2. Though it looks like C syntax, there is no limitation to the language used despite the fact that it must support event-driven simulation features like sequential statements or waiting on a list of events or conditions (*waitOn*, *waitUntil*), like Verilog, SystemC, or SystemVerilog do.

By default the clock is turned on and the simulation remains unaffected by the clock suppression algorithm. If no output condition is satisfied and there exist scheduled timeouts (line 5) a phase of deactivated clock will be entered after the calculation of the future signal value (s_c) (by calling the function *nextEvent*) and the timeout value Δt, which will be infinity if the increment of the adder is zero. In this case the wait statement in line 14 will terminate only on changes in the input signal list or in the output conditions. After the period of clock suppression the new value of s will be forced depending on the presence of a timeout (line 15).

In the activated phase (starting from line 23) an activation signal for the clock is asserted until the occurrence of a $deact$-event, which indicates that no more state changes besides the optimized signal happen. As the checking of this property can be very expensive in terms of computing performance an optimized solution would be the integration of the state change observation into the simulator kernel, where this comes almost for free. Alternatively the deactivation could take place after a certain timeout. While this solution needs some knowledge about the design under test (behavior of the operands of the adder) it could be applied efficiently, especially for huge designs.

Every time the operand of an adder changes, the new value must also be assigned to the local increment variable (*assignIncrement* in line 30).

As the presented algorithm can be clearly implemented in a single process within the testbench but outside the design hierarchy, no modification of the original design code is

Listing 2 Clock Suppression Algorithm

```
1   // Initialization
2   clockActivate = true;
3   Δt = ∞ ;
4   loop{
5       if (not(OC(s)) and Δt <> 0) {
6           // deactivated phase
7           tmp = s;
8           s_c = nextEvent(s);
9           if (Δs_t == 0) {
10              Δt = ∞;
11          } else {
12              Δt = [(s_{t+Δt} − s_t) div Δs_t] · t_clk;
13          }
14          waitOn(I, OC, Δt);
15          if (timedOut()) {
16              tmp = s_c − (s_c − s_t) mod Δs_t;
17          } else {
18              tmp = s_t + ((now − t) div t_clk)Δs_t;
19          }
20          // force s, be aware of multiple drivers
21          s = tmp;
22      } else {
23          // activated phase
24          clkActivate = true;
25          waitUntil(deact);
26          clkActivate = false;
27          Δt = ∞;
28      }
29      // conditional increment assignment
30      Δs_t = assignIncrement(I);
31      t = now;
32  } // loop
```

neccessary. Hence, synthesis can be done on the same code base.

Listing 3 shows the modified clock generation process. The lines 2 – 5 are inserted to prohibit the generation of clock events if the activation signal $clkActivate$ is not asserted.

IV. A simple Example

To give further insight into the methodology and to evaluate the theoretical work, a simple example consisting of two sequential elements with adders, is discussed in this section (see Figure 1). The first adder $C1$ with its input logic en and $INC1$ will be optimized through an external entity CS, instantiated in the testbench. The second adder $C2$ serves as output logic, which is activated on a set of output conditions for $C1$. While any synchronous logic could have been placed instead of $C2$, choosing an parameterisable adder gave the possibility to easily adjust the ratio between turn on and turn off time of the clock signal through modification of the conditions set ($cd1$ and $cd2$ in Figure 2).

A big advantage of the proposed methodology is that the design code is not affected by the optimization, as it takes

Listing 3 PseudoCode for Clock Generation

```
1       loop {
2           if (clkActivate == false) {
3               waitOn(clkActivate);
4               waitFor(t_clk);
5           }
6           clk = 1;
7           waitFor(t_clk/2);
8           clk = 0;
9           waitFor(t_clk/2);
10      }
```

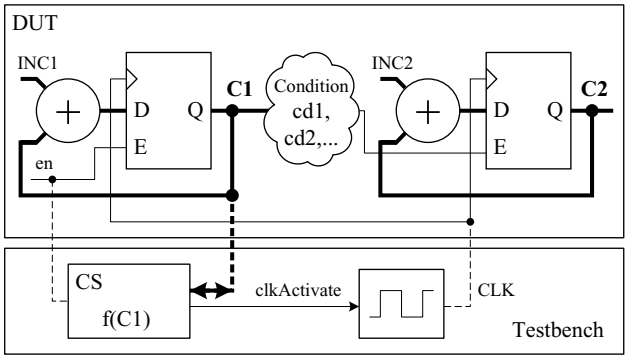

Fig. 1. Two cascaded adders as an example for a linear function with input signals and output conditions.

place only in the testbench. Even entity declarations remain unmodified as indicated in Figure 1.

For verification of the correct behavior of the proposed clock suppression algorithm two identical instances of the test example were simulated, one of them enhanced through clock suppression. Example signal traces are shown in Figure 3. Note, that the optimized version of $C1$ differs from the reference design, when the clock is turned off, but is set to the correct value (marked by an x) before the clock is turned on. The signal $C2$ always has correct values.

The duration of a 20 ms simulation run of both reference and optimized designs are compared in Figure 4. While the simulation performance of the reference design remains nearly

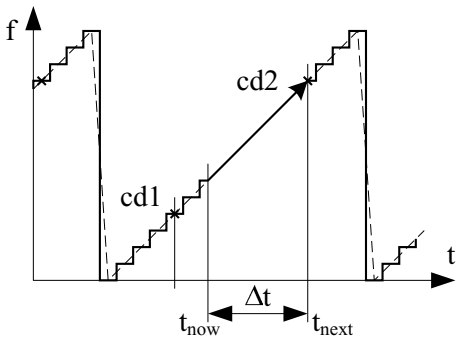

Fig. 2. Output conditions and deactivation of the clock signals between them.

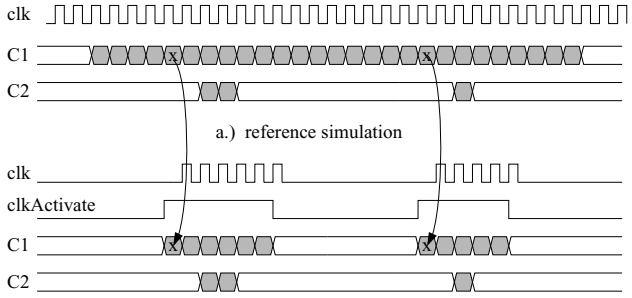

Fig. 3. Sample signal traces

unaffected by the amount of activity on $C2$, the speedups of the optimized version are – as expected – proportional to the turn off time (less activity of $C2$). The break-even point is above 50 percent, i.e. if the turn on time gets equal to the turn off time, the overhead of the clock suppression algorithm will eliminate any speedup. The break even point will shift to the right if the frequency of the turn on - turn off cycles increases. In the next section a more detailed discussion on the prerequisites for the application of clock suppression is given.

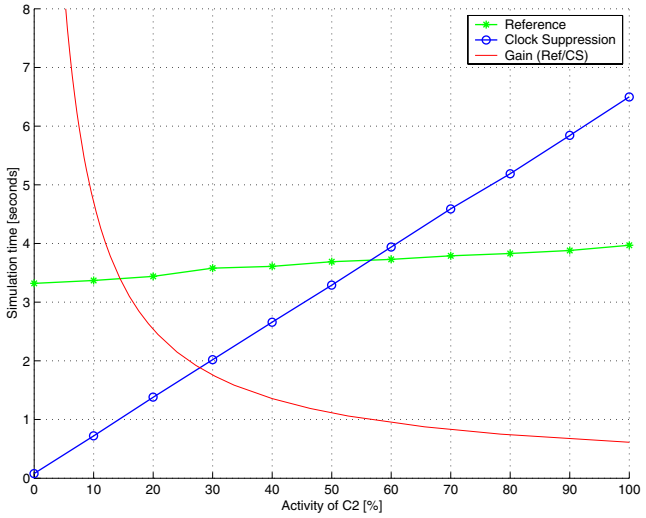

Fig. 4. Gain of clock suppression

V. Prerequisites and Remarks

Though the presented approach has a great potential for optimization, some restrictions limit its application in verification environments. As mentioned before, this optimization methodology makes use of specific properties of signals, e.g. partly constant behavior of the operands of accumulators. For example traditional accumulators known from CPU's normally don't have a partly constant summand and are therefore not suitable for this approach.

The following two conditions must be satisfied to accelerate a simulation:

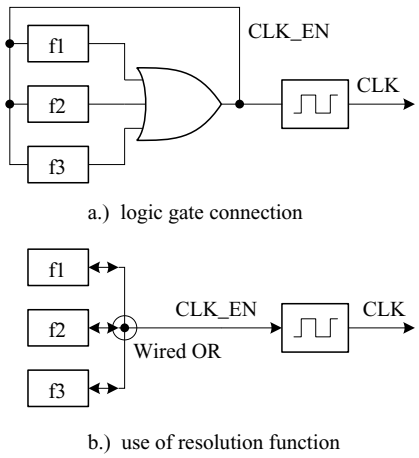

Fig. 5. Grouping of clock suppression modules

- The average event rate of the input signal list $I(s)$ has to be low compared to the clock frequency:

$$f_{clk} = \frac{1}{t_{clk}} \gg \overline{f_I} \tag{9}$$

- The average event rate of activations due to satisfied output conditions has to be low compared to the clock frequency:

$$f_{clk} = \frac{1}{t_{clk}} \gg \overline{f_{OC}} \tag{10}$$

To even suppress any clock event within a given time period, thus allowing jumps in time, the whole logic driven by that clock must be supplied with clock suppression. Hence, for hierarchical designs a mechanism for synchronizing two or more clock suppression modules within a clock generation process is necessary. Figure 5 shows two methods. Either an explicit logic OR function or the implicit resolution function of multi-driver signals (std_logic in VHDL) can be used to for this purpose. The latter has the advantage of easy extensibility and eliminates the need for a feedback of the activation signal, because the activation signal needs to be readable by all other clock suppression modules in order to set the correct state of a deactivated module if another module needs clock activation.

To prevent the designer from modifying the source code, forcing and reading of signals over component hierarchies has been used in the previous section. Both Verilog and SystemC have natively built-in support for setting and reading signals from foreign entities. The current VHDL standard forbids this type of action, but simulators usually can deal with it[2].

VI. Results

To underpin the feasibility and performance of the presented clock suppression technique, the results of three different simulation runs of a distributed embedded computer system consisting of a 4-port Ethernet switch and four computers with network interface cards are summarized. Results for a 300

[2]e.g. for Modelsim the Foreign Language Interface (FLI) as well as the SignalSpy feature are available.

Design, abstraction level (BFM ...Bus Functional Model) (RTL ...Register Transfer Level)	Simulation Time [sec]		Speedup
	without CS (estimated)	using CS	
D 7, BFM (Switch + 4 NICs)	9,0 . 10^4	8,0	1,1 . 10^4
D 7, BFM (4 NICs) + RTL (Switch)	2,7 . 10^8	7,8 . 10^3	3,5 . 10^4
D 7, RTL (Switch + NIC)	4,5 . 10^8	3,0 . 10^4	1,5 . 10^4

TABLE II

COMPARISION OF SIMULATION PERFORMANCE FOR A 300 SECONDS SIMULATION RUN WITH AND WITHOUT CLOCK SUPPRESSION (CS) FOR DESIGN 7 FROM TABLE I

seconds simulation run on a Pentium 4 CPU with 3.4 GHz and 1 GB main memory running a 32-bit Linux operating system are listed in Table II.

Depending on the actual configuration and the detail of modeling, a speedup factor of up to four orders of magnitude has been reached. In this case the simulation of the RTL design of an Ethernet switch together with four connected nodes including RTL designs of the network interface cards and the interconnecting bus system has been brought down from several days to minutes without giving up any detail of the simulated models. The evaluation of other designs, like the ones mentioned in Table I, is subject to ongoing research.

VII. CONCLUSION AND FUTURE WORK

This paper revisited the clock suppression technique used to significantly speed up simulation runs, which are mandatory to verify the functionality of nowadays embedded computer systems. After formally introducing the problem, results of applying clock suppression to real life simulations have been presented. They deliver a convincing impression of the potentials of the presented clock suppression methodology. And it is important to note that all this comes without the necessity to modify neither the event simulation engine itself nor the design, but by using existing means that state of the art hardware description and modeling languages and simulators offer.

To even squeeze the last performance out of this technique, it is planed for the near future to add the presented clock suppression algorithm to the simulation kernel of an event based simulation engine.

ACKNOWLEDGEMENTS

This work was partly funded by the Austrian FHplus research initiative within the scope of the DECS (Design Methods for Embedded Control Systems) research project, grant number 811414, and by the Austrian Research Promotion Agency (FFG) under the grant number 810092 (IMAGINE). Further support was received from Oregano Systems Corp.

REFERENCES

[1] L. Sèmèria and A. Ghosh, "Methodology for Hardware/software Co-verification in C/C++," in *Proceedings of the Asia and South Pacific Design Automation Conference*. New York, USA: ACM Press, 2000, pp. 405–408.

[2] F. Fummi, M. Loghi, S. Martini, M. Monguzzi, G. Perbellini, and M. Poncino, "Virtual Hardware Prototyping through Timed Hardware-Software Co-simulation," in *Proc. Design Automation and Test in Europe Conf. (DATE 05)*, 2005, pp. 798–803.

[3] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "SystemC Cosimulation and Emulation of Multiprocessor SoC Designs," *IEEE Computer*, pp. 53–59, April 2003.

[4] M. Gravenstein, "Modeling techniques to support system level simulation and a top-down development methodology," in *Proceedings of the International Verilog HDL Conference*, Santa Clara, CA, USA, Mar. 1994, pp. 43–50.

[5] E. Ulrich, "A design verification methodology based on concurrent simulation and clock suppression," in *Proceedings of the Design Automation Conference*, Miami Beach, FL, USA, June 1983, pp. 709–712.

[6] T. Weber and F. Somenzi, "Periodic signal suppression in a concurrent fault simulator," in *Proceedings of the European Design Automation Conference*, Amsterdam, NL, Feb. 1991, pp. 565–569.

[7] S. Gai and P. Montessoro, "Creator: New advanced concepts in concurrent simulation," *IEEE Transactions on CAD/ICAS*, vol. 13, no. 6, pp. 786–795, June 1994.

[8] P. Razdan, G. P. Bischoff, and E. G. Ulrich, "Clock suppression techniques for synchronous circuits," *IEEE Transactions on CAD/ICAS*, vol. 12, no. 10, pp. 1547–1556, Oct. 1993.

[9] Y. Takamine, S. Miyamoto, S. Nagashima, M. Miyoshi, and S. Kawabe, "Clock even suppression algorithm of velvet and its application to s-820 development," in *Proceedings of the 25th Design Automation Conference*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1988, pp. 716–719.

[10] P. Schaumont and I. Verbauwhede, "Interactive cosimulation with partial evaluation," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 2004*, Paris, France, Feb. 2004, pp. 642–647.

[11] P. Babighian, L. Benini, and E. Macii, "A scalable algorithm for rtl insertion of gated clocks based on odcs computation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 29–42, Jan. 2005.

[12] K. I. Park and K. H. Park, "Event suppression by optimizing VHDL programs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 682–691, Aug 1998.