CORE
Metadata, citation and similar papers at core.ac.uk
Provided by IntechOpen

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

25

New Concepts of Asynchronous Circuits Worst-case Delay and Yield Estimation

Miljana Milić and Vančo Litovski
University of Niš, Faculty of Electronic Engineering
Serbia

1. Introduction

Although the benefits of asynchronous design style are undeniable, this style is still a road that designers rather avoid. There are, however, serious advantages of this digital design concept that are making it favourable for many applications. Asynchronous circuits need no clock generation and distribution (Sparso, 2006; Martin & Nystrom 2006), which leaves the problems related to clock skew behind and saves a lot of chip area. Asynchronous circuits are characterized with much easier technology migration and good modularity. Very low EMI occurs during operation, while achieving high noise immunity (Lewis & Brackenbury 2001). Power is consumed only when useful work is done. The absence of the clock itself reduces the power consumption. These issues are very important while designing portable systems where battery size and lifetime are important.

Synchronous circuit design styles have enormous commercial practice and very significant pedigree, and those are the major reasons for the lack of motivation to apply asynchronous circuit techniques (Davis & Nowick, 1997). Nevertheless, the motivation to pursue the study of asynchronous circuits is based on the simple fact that all high-performance ‘‘synchronous” design styles are ‘‘asynchronous in the small” (Cortadella et al. 1999) Because of that, some techniques for desynchronization of synchronous circuits have appeared lately (Cortadella et al. 2006; Andrikos 2007).

Beside their benefits, some problems related to asynchronous circuit design are still waiting to be solved. One of the most important is the estimation of asynchronous circuit performances. That is, determining the delays of the paths in a particular asynchronous circuit. Early evaluation of the path delays in the circuit helps avoiding early timing problems as well as circuit performance characterization (Sokolovic, Litovski & Zwolinski 2009). Precise paths delays, of course, can be estimated only in the final steps of the design process. That is because the delay is extracted from the circuit after layout synthesis. If the delays do not satisfy the required speed of the circuit, the circuit has to be redesigned. The same conclusion stands when timing problems occur. This strongly suggests that new methods are to be offered enabling delay estimation to be performed during the early phases of digital system design. Our aim here is to establish the application of a standard logic simulator in asynchronous circuit path delay analysis and parametric yield estimation.

The simplest way to determine the circuit delay is simulation. At the transistor level complex circuits’ simulation becomes inefficient. To verify the logic function and the timing

Source: Micro Electronic and Mechanical Systems, Book edited by: Kenichi Takahata,
ISBN 978-953-307-027-8, pp. 572, December 2009, INTECH, Croatia, downloaded from SCIYO.COM

specifications of the circuit, logic simulators dealing with gate level descriptions are used. However, the delay of the circuit obtained by a logic simulator depends on the input test vectors. In order to determine the longest and the shortest possible delays in a combinational circuit, it has to be simulated using all 2^n possible input vectors, where n stands for the number of inputs. Therefore, the simulation is not an efficient solution for most circuits. On the other hand, since logic simulation is one of the first design steps, embedding worst-case delay analysis into a logic simulator would ensure early detection of incorrect design solutions.

The causes of the parameter values' variations are following: temperature and environment; technology and process; and specific phenomena within components (such as electromigration). These variations affect the circuit behaviour over time. Because of them, a 100% parametric yield is not achievable since the responses of all the manufactured circuits do not satisfy the required timings. The nature of parameter variations is statistical in the sense that all parameter values are random within a probability interval. As a result, the response values in particular delays are also randomly distributed within an interval that depends on the nature of the mapping of parameter tolerances onto response tolerances (Litovski & Zwolinski 1997).

Timing analysis of a circuit consisting of primitive gates is performed by a timing analysis tools. These tools can calculate circuit delays that are the result of parametric variations. The aim of static timing analysis is to verify that a logic circuit satisfies its timing constraints, i.e. that the logic circuit will function correctly when run at an intended speed (Spence & Soin 1988).

Commercial timing analysis systems are based on statistical timing verification. A problem that often appears in statistical timing analysis is that the longest paths often become false paths. Moreover, when considering DSM (deep-submicron) technologies, almost every path can be considered critical. The critical path of a digital circuit is the longest path or the path with the largest delay between the input and the output of the circuit (Mak et al. 2004).

Direct methods and sampling methods can be used instead of worst-case and statistical timing analysis. Direct methods are using formulae that map the parameter tolerances into the response tolerances. These methods are applied to cases where the parameter variations are small. Nevertheless, this analysis is, although more accurate, still very time consuming, since it requires all gate sensitivities with respect to all possible variations to be calculated.

2. How to analyze timings

The purpose of the timing analysis is to determine the following timing constraints:

- Do the signals arrive at ports in time?
- Do the signals stay long enough at the required state to be useful?
- Will the signals propagate with a proper slope?
- Can the hardware run with a specified speed?
- Are there any paths which need additional analysis and modification?

Timing measurements, as already mentioned, can be performed using a circuit simulation, but such an approach is too slow to be practical. There are two simulation alternatives for delay estimation in logic circuits. The first approach is based on static timing analysis (STA) and statistical static timing analysis (SSTA), while the second includes Monte-Carlo analysis. STA methods evaluate digital circuit timing without simulation. For nanometre manufacturing processes, which have increased parameter variability, a corner-based STA

has become inadequate. To avoid this problem, a statistical approach has been proposed: statistical static timing analysis (SSTA). As a result of SSTA analysis probability density functions (pdfs) are obtained. The percentage of fabricated dies which meet a required delay, can then be calculated or conversely, the expected performance for a particular parametric yield (Maksimović 2000). SSTA aims to determine the distribution of the delay of a design by accounting for actual statistics of process variations.

In practice, however, complete and exact statistical information is not always available or might be difficult to obtain from the foundries. In (Sokolović & Litovski 2005) a method for worst-case estimation of timing-yield that deals with those difficulties is presented. This method is based on distributional robustness theory (DRT). Nevertheless, since finding the worst-case estimate is defined as an optimization problem, it requires comparable simulation and data processing time. The probabilistic nature of the timing behaviour of a circuit, for selection of the critical path, imposes implementation of statistical analysis and simulation. However, even with their clear advantages, developing and using statistical models and methods requires considerable effort. The complexity of the statistical techniques is still significantly high. These can be reasons for avoiding statistical methods, but higher process integration and increasing operational speed also make them inevitable (Mak et al. 2004).

The Monte-Carlo method is based on a large number of circuit simulations (analyses). As a result, it gives the mean and standard deviation of the delay at the output of the circuit. A Monte-Carlo simulation cycle has two steps: a sampling step and an analysis step. In the sampling step, for a given set of parameters (delays of gates in this case) a single random value for every parameter is produced according to the given probability distribution.

An analysis of the circuit must be performed for each new version of the parameter vector. In this way a set of different parameter values (properties) of the circuit's output signals are obtained. The analysis step, in addition, utilizes these sampled values to derive the arrival times of all output signals for the given circuit instance.

The desired accuracy determines convergence criteria of the process i.e. the number of cycles. In fact, there should be no convergence, since the simulation results do not approach to any particular value. This is why the statistical properties of the responses are monitored. Once the mean or variance converges to the desired precision range, the procedure terminates. It takes from a several tens to a few hundreds of Monte-Carlo cycles to achieve convergence of the results. This means that the timing analysis step should be repeated that many times (Lin & Davoodi 2008). If, however, each iteration of Monte-Carlo analysis involves a transistor level simulation of the entire circuit (or the entire circuit path), this approach will have an unacceptable run time (iscas89.html --).

The design and analysis can significantly be accelerated by application of a logic simulator for the timing analysis in the Monte-Carlo analysis. A way for timing analysis with a VHDL logic simulator based on the research in (Maksimović 2000; Maksimović & Litovski 1999; Maksimović & Litovski 2002) will be presented next. It simplifies the delay evaluation procedures and speeds them up. In this way a good base for evaluation of asynchronous circuits' performances is established. This method will now be explained in more detail.

3. Estimating delays with a logic simulator

Our method for path-delay estimation in asynchronous nonsequential digital circuits is based on a robust delay estimation algorithm. It makes sense to analyze the circuit paths

only in one operating sequence. Because of that, and to be able to implement the suggested method to sequential circuits, one needs to pay special attention when dealing with circuits that have feedback loops. When sequential circuits are analysed at the circuit level, the feedback loops, need to be broken, while, for analysis at the gate level, complex models of element descriptions are required. The proposed concept can enable acceleration of Monte-Carlo analysis if it is embedded within the analysis step of the Monte-Carlo loop. The sampling step of the Monte-Carlo analysis is performed in the usual manner.

To perform a timing analysis that is, a delay estimation of all the paths in a circuit using a logic simulator, the logic simulation mechanism needs a small modification (Maksimović 2000). Neutral events that do not change the logic value of the signal in a standard logic simulator are ignored. If the signal description is extended to have a few additional attributes, such as event, delay value, etc. (Mak et al. 2004; Maksimović & Litovski 1999), then a change in any of those attributes will be considered as a non-neutral event. Simultaneous propagation of all input vectors through the circuit is assumed. The values of delay attributes are accumulated along structural paths, starting from the primary inputs and ending at the primary outputs or, if necessary, at any particular node inside a circuit. At the end of this very fast delay estimating process, after only one run of the logic simulator, all delays of both output signal edges are available.

3.1 Models of gates and signals

For each output signal of the circuit, S, four delay values are estimated:

d1mn(S) – the shortest path delay for a rising edge at S,

d0mn(S) – the shortest path delay for a falling edge at S,

d1mx(S) – the longest path delay for a rising edge at S,

d0mx(S) – the longest path delay for a falling edge at S.

In order to evaluate all worst-case path delays to all the signals in the circuit with just one simulation, it is necessary to perform simultaneous simulation of the circuit for all input vectors. To enable this, signals that connect logic gates within the circuit must contain two types of information: events on the signal, and the shortest and the longest path delays to the signal. This information is stored within a signal description in the form of two types of attributes: attributes that contain the delay information, as listed above, and the attributes for triggering the delay calculation processes in a gate. For a signal, S, the four attributes for triggering the calculation are:

arr1mn(S) – rising transition of shortest path arrival flag,

arr0mn(S) – falling transition of shortest path arrival flag,

arr1mx(S) – rising transition of longest path arrival flag,

arr0mx(S) – falling transition of longest path arrival flag.

It should be noticed that the signal now does not contain any logic values, as would be the case in a standard logic simulation.

To process signals described in this way and to perform the delay estimation, the gate model must include two modes: the activation – propagation mode and the delay calculation mode. Moreover, the gate description must contain two separate processes; first to calculate the maximal delay of the falling and rising transitions, and the second to calculate the

minimal delay of the falling and rising transitions. The activation – propagation mode of the model in each of these processes in a gate is sensitive to every change of the signal triggering attribute. After the delay calculation level of the model is activated, it then updates the output signal delay according to the input signal delay attributes and gate delay parameters. When the resulting output delay type (delay attribute of the output signal) is calculated, the output signal changes the particular triggering attribute to trigger processes in the following gates.

```
generic (ifo_izl: integer:= 1;
tpd_hlmn : real := 0.9e-9;
tpd_lhmn : real := 1.0e-9;
tpd_hlmx : real := 0.95e-9;
tpd_lhmx : real := 1.05e-9);
.                    .                    .
p1: process (in1.d0mn, in1.d1mn, in1.arr0mn, in1.arr1mn,
        in2.d0mn, in2.d1mn, in2.arr0mn, in2.arr1mn,
        in3.d0mn, in3.d1mn, in3.arr0mn, in3.arr1mn)
        variable r,p: real;
        variable multipl : real;
begin
        multipl := real(ifo_izl);
        r:= ((multipl*1.0) + (0.03*(gauss_rng)));
        p:= (multipl*0.9 + (0.03*(gauss_rng)));
        if (in1.arr0mn or in2.arr0mn or in3.arr0mn) then
                out1.d1mn   <= min(min(in1.d0mn, in2.d0mn), in3.d0mn) + r;
                out1.arr1mn <= true;
        end if;
        if (in1.arr1mn and in2.arr1mn and in3.arr1mn) then
                out1.d0mn   <= min(min(in1.d1mn, in2.d1mn), in3.d1mn) + p;
                out1.arr0mn<= true;
        end if;
end process p1;

p2: process (in1.d0mx, in1.d1mx, in1.arr0mx, in1.arr1mx,
        in2.d0mx, in2.d1mx, in2.arr0mx, in2.arr1mx,
        in3.d0mx, in3.d1mx, in3.arr0mx, in3.arr1mx)
        variable r,p: real;
        variable multipl : real;
begin
        multipl := real(ifo_izl);
        r:= (multipl*1.05 + (0.03*(gauss_rng)));
        p:= ((multipl*0.95) + (0.03*(gauss_rng)));
        if (in1.arr0mx or in2.arr0mx or in3.arr0mx) then
                out1.d1mx   <= max(max(in1.d0mx, in2.d0mx), in3.d0mx) + r;
                out1.arr1mx <= true;
        end if;
        if (in1.arr1mx and in2.arr1mx and in3.arr1mx) then
                out1.d0mx   <= max(max(in1.d1mx, in2.d1mx), in3.d1mx) + p;
                out1.arr0mx<= true;
        end if;
end process p2;
```

Fig. 1. Process for assigning minimal and maximal delay of the falling and rising edges for a tree input NAND gate.

An example of the process for assigning the minimal and maximal delay of the falling and rising edges for a tree input NAND gate is shown in Fig. 1. The gate inputs are denoted as

in1, in2 and in3, and the output as out1. The gate propagation delays for the rising and falling edges at the output out1 in both processes are denoted by tpd_lhmn/mx (minimal or maximal time propagation delay from low to high) and tpd_hlmn/mx, respectively (minimal or maximal time propagation delay from high to low). Each falling transition at an input of the gate means that one of the falling transition flags at one of the input signals (in1.arr0mn/mx or in2.-arr0mn/mx or in3.arr0mn/mx) becomes ''true". This sets a rising transition flag at the output signal attribute of the gate (out1.-arr1mn/mx) to ''true". This corresponds to an OR function. Simultaneously with setting the output flag, the gate model calculates a new value for the shortest and the longest path delays. The resulting output shortest and longest path-delay attributes for the rising edge is denoted by out1.d1mn/mx and is calculated after taking into account the arriving shortest/longest path delays for all tree gate input signals (in1.d0mn/mx, in2.d0mn/mx, in3.d0mn/mx), the minimal/maximal delay of the rising edge for this gate (a separate function assigns this value) and the function f which depends on the gate fanout value. Conversely, a rising transition flag at any of the gate input signals (in1.arr1mn/mx, in2.arr1mn/mx, in2.arr1mn/mx) produces a falling transition at the output only if a rising transition has previously arrived at all other gate inputs (Spence & Soin 1988; Agarwal et al. 2003). This corresponds to an AND function. The resulting output shortest/longest path-delay attributes for the falling edge, denoted by out1.d0mn/mx, take into account arrived shortest of longest path-delay input signal attributes (in1.d1mn/mx, in2.d1mn/mx, in3.d1mn/mx), the minimal/maximal delay of the falling edge for this gate and the fanout dependent function f. A process denoted with p1 calculates minimal path delays for rising and falling edges while the process denoted with p2 calculates maximal path delays for rising and falling edges. In this figure, it is also shown that the delays of a particular gate can be generated by different random functions which can take into account different input signal slopes, loading capacitances and other parameters that influence the ranges of rising and falling gate delays, tpd_lhmx and tpd_hlmx.

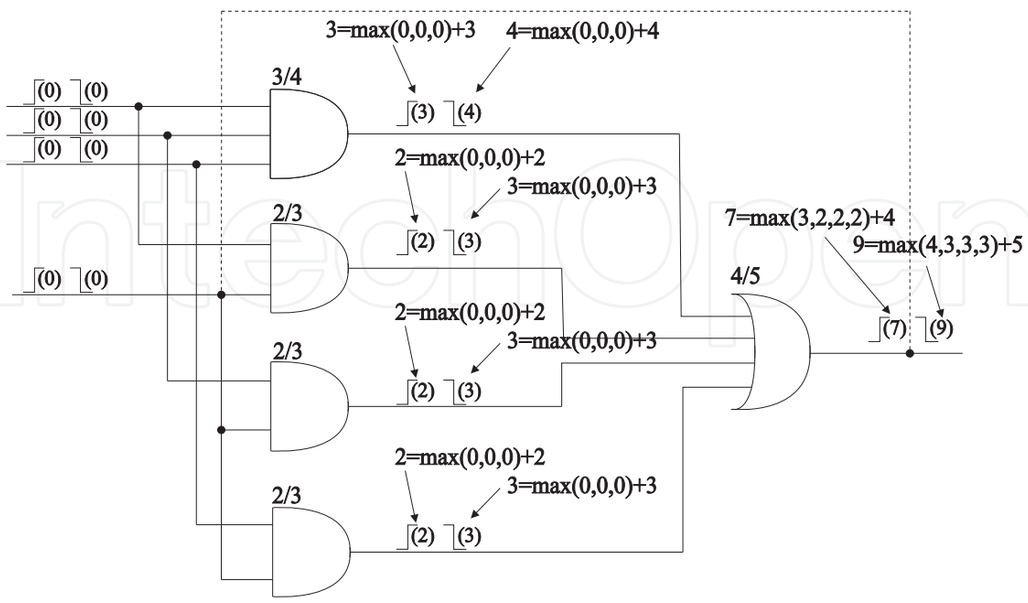

Fig. 2. Illustration of maximal delay estimation method for a possible implementation of a three input C-element.

The basic principle of delay accumulation is described in Fig. 2. The figure describes the maximal delay calculation of all paths in the simple asynchronous circuit which represents one possible implementation of the tree input C-element. Here, both rising and falling transitions are applied to all inputs of the circuit. Both, rising and falling transition delays are updated by each gate. The delay estimation of the circuit stops when all transitions reach the primary outputs. To analyze delays of the paths the feedback line had to broken.

3.2 Dealing with isolated asynchronous sequential elements

It may be of interest to estimate maximal delays of the rising and falling edges for the path that is going through the feedback in this circuit. The method that we suggest can be extended to be able to deal with the elements with loops or feedbacks. In these circuits longest paths will probably be ones that go through the feedback branches and return to the input of the circuit. The approach is similar to one used to generate test vectors in sequential digital circuits described in Ref. (Cheng & Agrawal 1989). Since more sequences must be observed, the circuit is replicated more times, representing each time sequence, and the delay along the paths is estimated for as many sequences as necessary. Fig. 3 shows how the principle would be used to estimate the delay of the longest path for the circuit of a simple asymmetric C-element. Assume that elements' delays in this figure are given in nanoseconds. In the implementation sense, the circuit will not be replicated many times. Instead of complicating the circuitry, only the results of one estimating simulation will be applied to the circuit input signals that are connected to the feedback loop. After that the circuit can be simulated again, with new initial delay parameters. Table 1 shows the results of repeated estimation for tree sequences.

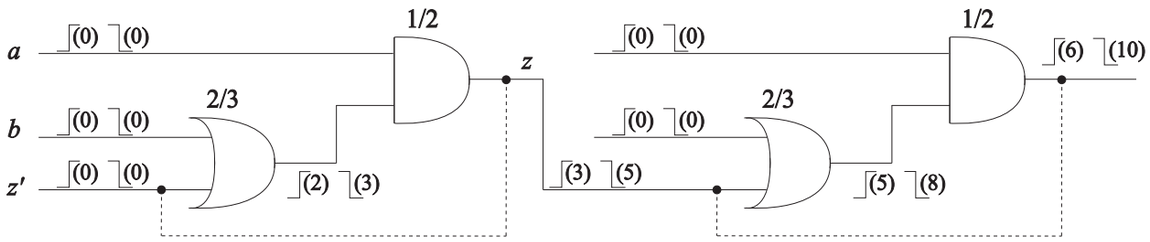

Fig. 3. Illustration of maximal delay estimation method in an isolated sequential element.

time sequence	$t_{mxr}(z)$ [ns]	$t_{mxf}(z)$ [ns]
I	3	5
II	6	10
III	9	15

Table 1. Results of the delay analysis for the asymmetric C gate in tree sequences.

3.3 Assigning the delay to a gate

In order to calculate four different worst-case delays for all paths in one digital circuit, the gate models must contain all four types of delays. It means that each gate in a circuit is characterized with four parameters: the maximal delay of the rising edge, the maximal delay of the falling edge, the minimal delay of the rising edge and the minimal delay of the falling edge. Nevertheless, assigning a particular delay to the gate and calculating the result is a

complex task. There are two delay components in each of gate delay functions (they are denoted with r and p in Fig. 1). One takes into account the fact that we want to use the gate models for statistical worst-case delay estimation, and the second must take care of the netlist of the entire digital circuit, that is the fanout information for each gate in the circuit. This is expressed by Eq. (1)

$$\begin{aligned}&\text{random\_value\_of(tpd\_lhmx/tpd\_hlmx/tpd\_lhmn/tpd\_hlmx)*fanout\_func(output)=}\\&\text{func(tpd\_lhmx/tpd\_hlmx/tpd\_lhmn/tpd\_hlmx)}\end{aligned} \tag{1}$$

where the slash sign (/) represents the OR function.
Considering the first component, we must introduce randomness into the delay assignment process. This is the crucial step which enables Monte-Carlo analysis. The need for statistical delay analysis comes from the variations of the circuit parameters. Therefore, the delay estimation method must also include the influence of parameter variance on minimal and maximal delays of all signals in the circuit. If the delays were modelled as fixed values, the worst-case delay values would not be the real worst-case values due to the process variations. If we consider this fact, we conclude that the solution to this problem can be delay estimation in the usual manner, while all delay ranges in each gate instance are generated randomly. Each time the calculation process is activated in a gate, new worst-case delay values are considered in the signal delay calculations. Since simple models are used and the calculations are still very time-efficient, the simulations can be performed a few hundred times to enable statistical delay analysis. In this way a method for statistical static timing analysis using a standard logic simulator has been developed (SSTA for SLog).
The given delay probability density function determines the delay randomness. Hence, the gate parameters are the mean values of the probability density function for a particular gate type. The gate delay information given as parameters incorporates the real fabrication variations of a particular technology since worst-case delay distributions are characterized with mean and variance values that should be given as the fabrication technology parameters. Each gate randomly generates the maximal and minimal delays of the rising and falling edges with a Gaussian distribution, with the mean and standard deviation defined according to Eq. (2)

$$\varphi(p)=\frac{\exp\left[-(p-\mu_p)^2/\left(2\sigma_p^2\right)\right]}{\sigma_p\sqrt{2\pi}} \tag{2}$$

where μ_p represents the mean value and σ_p^2 is the variance of the random variable p (Litovski & Zwolinski 1997). This function can, of course, be changed if necessary.
The second component of Eq. (1) deals with the real position of the particular gate in the netlist of the entire circuit. It is well known that the delay of the output signal for a single gate depends on the number of gates that are driven by that particular gate. If a gate has to drive two gates, the delay is larger than in the case of driving a single gate. In order to increase the accuracy of the gate delay model and the entire delay estimation algorithm, the fanout information of each gate in the circuit netlist must be included in the delay calculations. To do this, two major modifications must be introduced. One modification affects the logic gate descriptions. The second must be performed on the digital circuit netlist. In this way, the real implementation of the circuit is taken into account. For example,

if one gate output drives two inputs of following gates, it means that all delays of the particular gate will be increased according to the approximation function. Also, the technology has a large impact on the fanout_func(output) value, since the function that gives the fanout dependence of the delay is specific to each technology and each gate type, and would be given by the manufacturer. The VHDL implementation of this idea will be shown later.

3.4 The algorithm for path-delay estimation

For statistical estimation of worst-case delays, that is SSTA using a standard logic simulator – it is necessary to perform a few hundred estimation simulations. The exact number of simulations, thanks to the nature of the Monte-Carlo process, is not influenced by the number of parameters but by the required precision and accuracy of the results.
Table 2 gives a description of estimation phases for one sample.

Input:	-Ranges of delays for rising and falling transition for each gate -circuit netlist -library of circuit elements described to support the timing analysis
Output:	-Ranges of delays for rising and falling transition for all circuit output signal
step 1:	-Set all signals in the circuit to be a composite type consisting of the following attributes: 4 different delay information (maximal and minimal delays of rising edge and maximal and minimal delays of falling edge) 4 different flags for triggering each delay type calculation in a particular gate
step 2:	-Initialize all signals triggering flags to "false" value. Setting them to value "true" starts the calculations.
step 3:	-Initialize all signals to have zero values of the delay attributes
initialization	-Initialize the calculation process by setting the primary input signal triggering flags to "true"
step 4:	-Until all signals and gates are processed (all signal triggering flags should be set to "true") perform the following steps by moving through the topological levels of the circuit: -The delay calculation is activated when all input signals of the gate have triggering flags set to "true". When the delay calculation depends on the logic function of the gate, it is taken into account. For example, each falling transition (flag for falling transition is set to "true" at the input of the gate) at an AND gate input, produces a falling transition at its output (sets the gate output signal triggering flag for falling transition to "true"), but a rising transition at an input is able to produce a rising transition at the output only if the rising transition had previously arrived at all other gate inputs. -In order to complete all attributes of the gate output signal, before activating its output transition flag, the corresponding gate delay should be calculated by processing delays that arrived with input signals of the gate. The particular delay of the chosen gate is also added to the resulting corresponding delay. -The estimation terminates when all triggering flags for all output signals are set to "true".

Table 2. Path-delay estimation algorithm with a logic simulator.

The delay values of a particular gate have standard deviation r, which is in our case set to be 3%. This can be varied if necessary. This value is derived from the parameter tolerances for an integrated circuit fabrication technology.
The circuit is described and simulated at the structural (gate) level, while having available delay ranges values (minimum and maximum delays) of all building blocks for both rising and falling edges. When this estimation process is embedded in a Monte-Carlo loop, the delays for a gate in a circuit will be characterized with a mean and a variance and then randomly chosen in each estimation process. At the start of the simulation, the circuit is excited with both rising and falling transitions at all primary inputs. This is referred to as the initialization phase, where all triggering attributes of all signals at the primary circuit inputs are set to ''true", that is the transitions at all primary inputs are initialized. All these transitions initiate the estimation processes in the gates at the first topological level of the digital circuit. When these processes are completed, the processes in the first topological level gates activate the transitions at their outputs to enable the calculation processes of the gates in the second topological level. As the transitions propagate from primary inputs towards primary outputs, the gate delays are accumulated along the paths, since an activation transition for the gate output signal is possible only if the delay of that gate has been already estimated. Signal attributes for the delay calculation and the calculation activation are used by the processes in the gate models and their values are dynamically updated, while the wave of activation shifts from the input to the outputs of the gates and the entire circuit. Once the circuit calculation activity is exhausted, the shortest and the longest path delays are available in signal attributes d1mn, d1mx, d0mn and d0mx of each output signal in the circuit.
It should be mentioned that these simulations, in addition, do not require any kind of stimuli selection, since they take into account all possible signal transitions. Only initialization is needed for the calculation processes in the entire circuit.

4. VHDL implementation

As already mentioned, the proposed concept is implemented using the VHDL hardware description language and simulator. Matlab was used for processing data obtained after simulations.
In order to have statistical simulation results, a random number generator is needed. Fig. 4 shows a VHDL implementation of the random number generator with a Gaussian distribution (Zwolinski 2004).

```
function gauss_rng return real is
        variable u1, u2, v1, v2, r, q, p: real;
        begin
                loop            u1:=rand;
                                u2:=rand;
                                v1:=u1*2.0 -1.0;
                                v2:=u2*2.0 -1.0;
                                r:=v1*v1 + v2*v2;
                                exit when r<1.0;
                end loop;
                q:=log2(r);
                p:=(sqrt((0.0-2.0)*q/r))*v1;
        return p;
end function gauss_rng;
```

Fig. 4. Gaussian random function generation implementation.

This function is executed 4 times within each gate, once for each delay type. Function rand in this description generates random numbers in the interval [0,1], with a uniform distribution.

In order to verify the efficiency of the applied gate models, we created a simple test circuit that calls the delay random generation function 600 times. The result of this can be used for random delay generation of rising or falling edges at the output of this circuit. The resulting histogram is shown in Fig. 5. In this case, the mean in the distribution is set to 1 ns, while the standard deviation is 3%. The x-axis shows the delays in [ns] units, and the y-axis represents the number of particular delay appearances within the corresponding range.

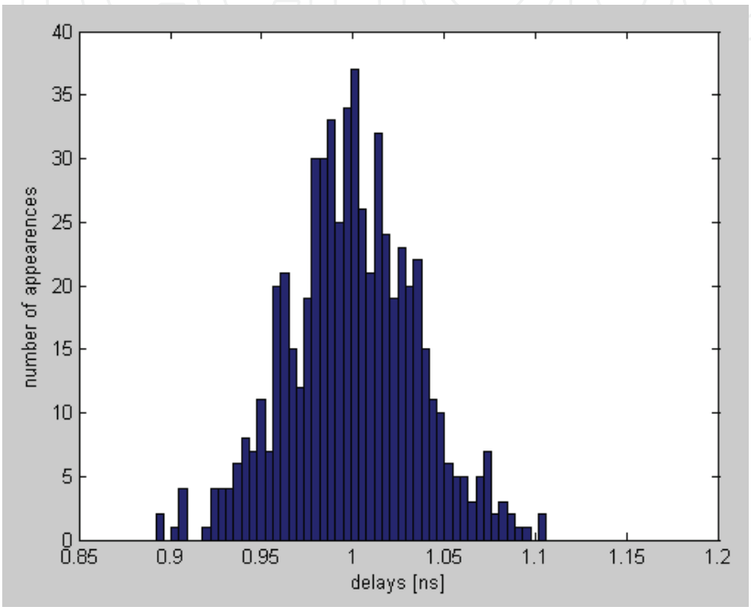

Fig. 5. Histogram of the function for random delays generation.

The shape of the particular delay type distribution function can be of arbitrary complexity. It is even possible to implement the spatial correlations within those random processes. Nevertheless, since we speak about a pre layout analysis no data are available for doing that at this stage.

VHDL models of primitive logic gates and simple asynchronous elements are kept in a VHDL library. Figs. 6–8 show VHDL modelling of a D-latch, RS-latch and two input C-element, respectively. It should be noted that D-latch circuit does not have specific conditions for activating a calculation process, because these circuits have only one data input.

Considering RS-latch circuit, the situation is far more complicated. It is assumed that this circuit has two inputs, R and S, and two outputs Q and its complement NQ. At the beginning of the description, it can be noticed that this circuit requires 18 different generics. There are 16 delay generics that are related to minimal and maximal delays through the latch, of four possible input–output combinations (R–Q, R–NQ, S–Q, and S–NQ), and to both possible signal transitions. The first and the second generics consider fanout values for the first and the second output. As at the other gate descriptions, these two values are initially set to one. In the latch instantiation, and after the circuit netlist is being processed, these values become their actual values that correspond to their actual topological position

```
entity DLatch is
        generic (ifo_izl: integer:= 1;
                tr_en_qmn : real := 1.0e-9;
                tf_en_qmn : real := 0.9e-9;
                tsu_d_enmn : real := 0.45e-9;
                tr_en_qmx : real := 1.05e-9;
                tf_en_qmx : real := 0.95e-9;
                tsu_d_enmx : real := 0.55e-9);
        port (q : out SDA_std_logic := (0.0, 0.0, false, false, 0.0, 0.0, false,
        false);
        d, en: in SDA_std_logic := (0.0, 0.0, false, false, 0.0, 0.0, false,
        false));
        end DLatch;
architecture only of DLatch is
begin
        p1: process  (en.d0mn, en.d1mn, en.arr0mn, en.arr1mn,
        en.d0mx, en.d1mx, en.arr0mx, en.arr1mx)
                variable i, j, k ,l, m, n: real;
                variable multipl : real;
        begin
                multipl := real(ifo_izl);
                f<=fanout_func(multipl)
                i:= (f*1.0 + (0.03*(gauss_rng)));
                j:= (f*0.9 + (0.03*(gauss_rng)));
                k:= (f*0.45 + (0.03*(gauss_rng)));
                l:= (f*1.05 + (0.03*(gauss_rng)));
                m:= (f*0.5 + (0.03*(gauss_rng)));
                n:= (f*0.55 + (0.03*(gauss_rng)));

                q.arr1mn <= true;
                q.arr0mn <= true;
                q.d1mn <= en.d1mn + i + k;
                q.d0mn <= en.d1mn + j + k;
                q.arr1mx <= true;
                q.arr0mx <= true;
                q.d1mx <= en.d1mx + l + n;
                q.d0mx <= en.d1mx + m + n;
        end process;
end only;
```

Fig. 6. VHDL model of the D-latch.

and implementation in the entire circuit. There are two processes in the circuit description, for determining all maximal (process p1, shown in Fig. 7a) and all minimal delay types (process p2, described in Fig. 7b). For example, the rising transition at the output Q happens if both falling transition at input R and rising transition at input S have arrived. The maximal delay of the rising transition at output Q is then calculated as a maximum between the maximal delays of the falling transition at R and rising transition S, which arrived at those inputs, plus the maximal of new, randomly generated maximal delays of the rising transition delays between S–Q and R–Q ports. The same conditions must be established for achieving the falling transition at the output NQ, while the maximal delays of the falling edge at this output are calculated using other values. The falling transition at the output Q happens if both rising transition at input R and falling transition at input S have arrived. The maximal delay of the falling transition at output Q is then calculated as a maximum between the maximal delays of the rising transition at R and falling transition S, which arrived at those inputs, plus the maximal of new, randomly generated maximal delays of the falling transition delays between S–Q and R–Q ports. Similar process stands for determining the minimal delay types.

```
entity RSLatch is
      generic (ifo_izl_1: integer:= 1;
           ifo_izl_2: integer:= 1;
           tr_rq_mn : real := 1.0e-9;
           tf_rq_mn : real := 0.9e-9;
           tr_rq_mx : real := 1.05e-9;
           tf_rq_mx : real := 0.95e-9;
           tr_rnq_mn : real := 1.0e-9;
           tf_rnq_mn : real := 0.9e-9;
           tr_rnq_mx : real := 1.05e-9;
           tf_rnq_mx : real := 0.95e-9;
           tr_sq_mn : real := 1.0e-9;
           tf_sq_mn : real := 0.9e-9;
           tr_sq_mx : real := 1.05e-9;
           tf_sq_mx : real := 0.95e-9;
           tr_snq_mn : real := 1.0e-9;
           tf_snq_mn : real := 0.9e-9;
           tr_snq_mx : real := 1.05e-9;
           tf_snq_mx : real := 0.95e-9);
      port (q, nq : out SDA_std_logic := (0.0, 0.0, false, false, 0.0, 0.0, false, false);
           r, s : in SDA_std_logic := (0.0, 0.0, false, false, 0.0, 0.0, false, false));
end RSLatch;
architecture only of RSLatch is
begin
      p1: process (r.d0mx, r.d1mx, r.arr0mx, r.arr1mx, s.d0mx, s.d1mx, s.arr0mx, s.arr1mx)
           variable i, j ,k, l, m, n, o, p : real;
           variable multipl1, mulitipl2 : real;
      begin
           multipl1 := real(ifo_izl1);
           multipl2 := real(ifo_izl2);
           f1<=fanout_func(multipl1);
           f2<=fanout_func(multipl2);

           i:= (f1* tr_rq_mx + (0.03*(gauss_rng)));
           j:= (f1* tr_sq_mx + (0.03*(gauss_rng)));
           k:= (f2* tf_rnq_mx + (0.03*(gauss_rng)));
           l:= (f2* tf_snq_mx + (0.03*(gauss_rng)));
           if (r.arr0mx and s.arr1mx) then
                q.d1mx<=max(r.d0mx,s.d1mx)+max(i, j);
                q.arr1mx <= true;
                nq.d0mx<=max(r.d0mx,s.d1mx)+max(k, l);
                nq.arr0mx <= true;
           end if;

           m:= (f1* tf_rq_mx + (0.03*(gauss_rng)));
           n:= (f1* tf_sq_mx + (0.03*(gauss_rng)));
           o:= (f2* tr_rnq_mx + (0.03*(gauss_rng)));
           p:= (f2* tr_snq_mx + (0.03*(gauss_rng)));
           if (r.arr1mx and s.arr0mx) then
                q.d0mx<=max(r.d1mx,s.d0mx)+max(m, n);
                q.arr0mx <= true;
                nq.d1mx<=max(r.d1mx,s.d0mx)+max(o, p);
                nq.arr1mx <= true;
           end if;
      end process;
```

Fig. 7. VHDL model of the RS-Latch.

```
p2: process (r.d0mn, r.d1mn, r.arr0mn, r.arr1mn, s.d0mn, s.d1mn, s.arr0mn, s.arr1mn)
    variable i, j ,k, l, m, n, o, p : real;
    variable multipl1, mulitipl2 : real;
begin
    multipl1 := real(ifo_izl1);
    multipl2 := real(ifo_izl2);
    f1<=fanout_func(multipl1);
    f2<=fanout_func(multipl2);

    i:= (f1* tr_rq_mn + (0.03*(gauss_rng)));
    j:= (f1* tr_sq_mn + (0.03*(gauss_rng)));
    k:= (f2* tf_rnq_mn + (0.03*(gauss_rng)));
    l:= (f2* tf_snq_mn + (0.03*(gauss_rng)));
    if (r.arr0mn and s.arr1mn) then
        q.d1mn<=min(r.d0mn,s.d1mn)+min(i, j);
        q.arr1mn <= true;
        nq.d0mn<=min(r.d0mn,s.d1mn)+min(k, l);
        nq.arr0mn <= true;
    end if;

    m:= (f1* tf_rq_mn + (0.03*(gauss_rng)));
    n:= (f1* tf_sq_mn + (0.03*(gauss_rng)));
    o:= (f2* tr_rnq_mn + (0.03*(gauss_rng)));
    p:= (f2* tr_snq_mn + (0.03*(gauss_rng)));
    if (r.arr1mn and s.arr0mn) then
        q.d0mn<=min(r.d1mn,s.d0mn)+min(m, n);
        q.arr0mn <= true;
        nq.d1mn<=min(r.d1mn,s.d0mn)+min(o, p);
        nq.arr1mn <= true;
    end if;
end process;
end only;
```

Fig. 7. (continued)

To simulate the circuit few hundred times, a specific VHDL testbench is necessary. This is shown in Fig. 9. Now, for each particular input of the circuit (instantiated few hundred times), for the responses to the logic analysis, and for initialization and for the simulation itself, a specific matrices are formed. The process that performs the timing analysis – log_timing1, is used for determining the minimal delay of the rising edges of the output signals for one asynchronous encoder circuit with five outputs. The results of the analysis are written to a text file.

5. Examples

After completing Monte-Carlo sampling and simulation steps, a huge amount of data can be expected. Since each gate model consists of four parallel processes, for each of the signal transitions, which gives the equivalent to four parallel simulations during each run of the analysis. For example, when the circuit was run through 600 analyses, then the equivalent of $4 \cdot 600 = 2400$ simulations are performed per output. For a circuit that has a small number of outputs, the resulting statistical data can be presented in the form of a histogram.

Fig. 10 shows the results (histogram) obtained for C-element, described as a logic gate. This is the easiest form of statistical representation of the simulation results. It is adequate only for circuits with a small number of outputs.

Considering particular delays of the gate outputs the, the corresponding yield can be easily determined by counting the number of the delays that fall into the acceptable range, and

```
p1: process (in1.d0mn, in1.d1mn, in1.arr0mn, in1.arr1mn,
            in2.d0mn, in2.d1mn, in2.arr0mn, in2.arr1mn)
            variable r,p: real;
            variable multipl : real;
begin
            multipl := real(ifo_izl);
            f<=fanout_func(multipl)
            r:= ((f*1.0) + (0.03*(gauss_rng)));
            p:= ((f*0.9 + (0.03*(gauss_rng)));
            if (in1.arr0mn and in2.arr0mn ) then
                        out1.d0mn   <= min(in1.d0mn, in2.d0mn) + r;
                        out1.arr0mn <= true;
            end if;
            if (in1.arr1mn and in2.arr1mn) then
                        out1.d1mn   <= min(in1.d1mn, in2.d1mn) + p;
                        out1.arr1mn<= true;
            end if;
end process p1;

p2: process (in1.d0mx, in1.d1mx, in1.arr0mx, in1.arr1mx,
            in2.d0mx, in2.d1mx, in2.arr0mx, in2.arr1mx)
            variable r,p: real;
            variable multipl : real;
begin
            multipl := real(ifo_izl);
            r:= (multipl*0.95 + (0.03*(gauss_rng)));
            p:= ((multipl*1.05) + (0.03*(gauss_rng)));
            if (in1.arr0mx and in2.arr0mx) then
                        out1.d0mx   <= max(in1.d0mx, in2.d0mx) + r;
                        out1.arr0mx <= true;
            end if;
            if (in1.arr1mx and in2.arr1mx) then
                        out1.d1mx   <= max(in1.d1mx, in2.d1mx) + p;
                        out1.arr1mx<= true;
            end if;
end process p2;
```

Fig. 8. VHDL timing processes for the two input C-element.

```
initialization: for J in 1 to 600 generate
  encoder_inst: encoder port map (inp => inputs(J), outp => outputs(J));
  inputs(J) <= (others => (0.0, 0.0, true, true, 0.0, 0.0, true, true));
end generate;
log_timing1: process
            use std.textio.all;
            file log: text open write_mode is "encoder_mn_r.statdel";
            variable line_1: line;
            variable I, J:integer;
            variable izlaz: SDA_std_logic_vector(0 to 4);
            variable delay_mn_r : real;
begin
            wait for 1 ps;
            for J in 1 to 600 loop
                        for I in 0 to 4 loop
                                    izlaz := outputs(J);
                                    delay_mn_r:= izlaz(I).d1mn;
                                    write (line_1, delay_mn_r, left, 15);
                        end loop;
                        writeline (log, line_1);
            end loop;
            wait;
end process log_timing1;
```

Fig. 9. Testbench process for writing simulation results for minimal delay of the rising edge of all encoder output signals into a file

dividing it by a total number of simulated circuits. Nevertheless, since the yield is contributed not only with the speed of the circuit but also based on the defects present during the chip manufacturing. Many of those defects are timing defects. Therefore, this methodology can be indirectly used to estimate the yield loss caused by the number of chips that are not operating with needed performance, or are having timing defects due to the fabrication process tolerances.

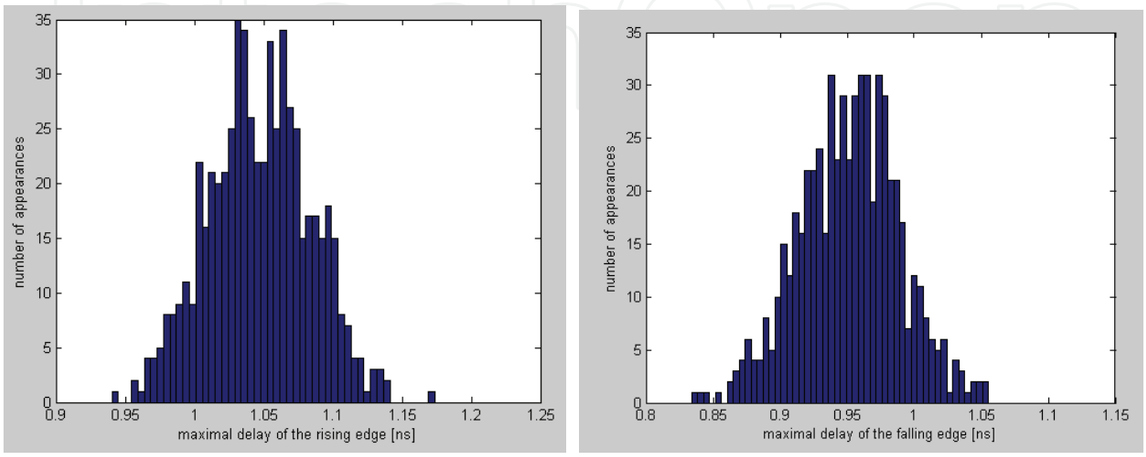

Fig. 10. Histograms of the C-element maximal delays.

Table 3 shows the simulation results of the C-element described at the structural level of abstraction shown in Fig. 2. The first column of the table shows the output number, the second shows the delay type for that output, and the third column gives the topological level of the particular delay type. The next two columns give the worst-case delay estimation results excluding randomness of the delay value, without and with the fanouts of each gate. In this case all fanouts are equal to one, giving the same values in these two columns. The last column shows the results of the statistical analysis of the results. It gives the mean value and the deviation value of the particular delay type.

output	delay type	topol. level	min/ max	fan-out	statistical	
					mean	dev.
1.	mnr	2	1.9ns	1.9ns	1.854	0.455
	mxr	2	2.0ns	2.0ns	2.041	0.418
	mnf	2	1.9ns	1.9ns	1.859	0.441
	mxf	2	2.0ns	2.0ns	2.033	0.439

Table 3. C gate – structural.

Table 4 shows the simulation results of the four stage asynchronous binary counter consisting of 4 T latches. Table 5 gives the timing analysis results for a generalized C-element (Myers 2001), shown in Fig. 11. A simple asynchronous address comparator unit (Myers & Alain 1993) is shown in Fig. 12, and its simulation results are presented in Table 6.

output	delay type	topol. level	min/ max	fan- out	statistical	
					mean	dev.
1.	mnr	4	3.7ns	3.7ns	3.704	0.705
	mxr	4	3.9ns	3.9ns	3.898	0.071
	mnf	4	3.6ns	3.6ns	3.599	0.681
	mxf	4	3.8ns	3.8ns	3.799	0.687

Table 4. T counter.

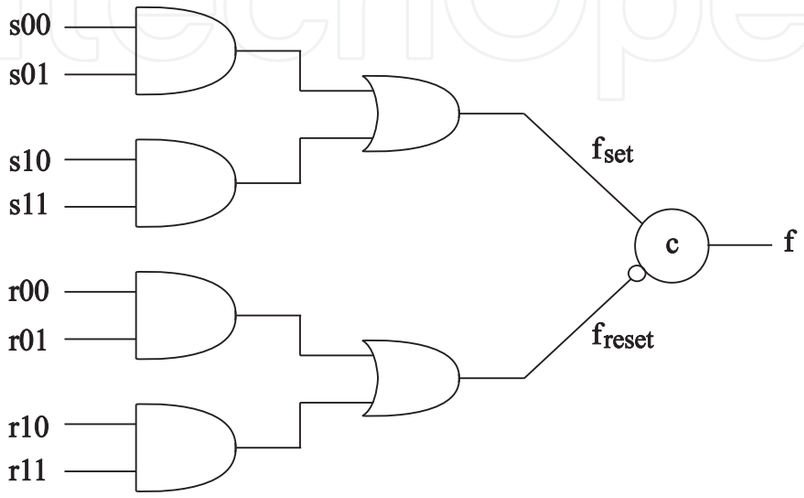

Fig. 11. Generalized C-element.

output	delay type	topol. level	min/ max	fan- out	statistical	
					mean	dev.
1.	mnr	3	2.7ns	2.7ns	2.682	0.058
	mxr	4	4.2ns	4.2ns	4.217	0.071
	mnf	3	3ns	3ns	2.981	0.058
	mxf	4	3.8ns	3.8ns	3.816	0.070

Table 5. Generalized C-element.

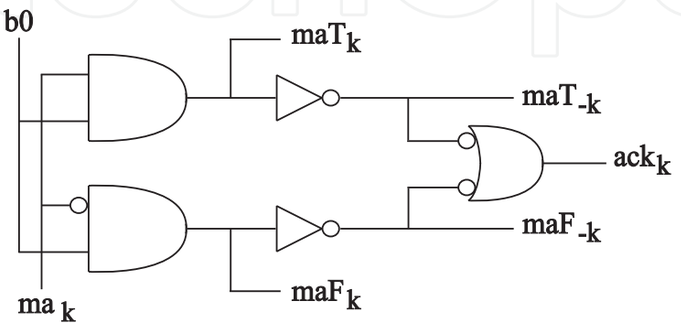

Fig. 12. Address comparator.

out.	delay type	topol. level	min/ max	fan-out	statistical	
					mean	dev.
1.	mnr	1	0.9ns	0.9ns	0.900	0.035
	mxr	1	0.95ns	0.95ns	0.954	0.036
	mnf	1	1.0ns	1ns	1.000	0.036
	mxf	1	1.05ns	1.05ns	1.051	0.035
2.	mnr	2	2.0ns	2ns	1.999	0.050
	mxr	2	2.1ns	2.1ns	2.101	0.050
	mnf	2	1.8ns	1.8ns	1.801	0.053
	mxf	2	1.9ns	1.9ns	1.905	0.049
3.	mnr	3	2.8ns	2.8ns	2.773	0.055
	mxr	4	4.0ns	4ns	4.000	0.072
	mnf	3	2.9ns	2.9ns	2.898	0.060
	mxf	4	4.0ns	4ns	4.000	0.072
4.	mnr	2	2.0ns	2ns	1.999	0.052
	mxr	3	3.05ns	3.05ns	3.051	0.060
	mnf	2	1.8ns	1.8ns	1.802	0.051
	mxf	3	2.95ns	2.95ns	2.954	0.064
5.	mnr	1	0.9ns	0.9ns	0.900	0.036
	mxr	2	2.0ns	2ns	2.002	0.053
	mnf	1	1.0ns	1ns	1.006	0.035
	mxf	2	2.0ns	2ns	2.002	0.049

Table 6. Address comparator.

Finally, Fig. 13 shows one complex asynchronous encoder circuit described in (Kondratyev & Lwin 2002), while table 7 gives its timing analysis results.

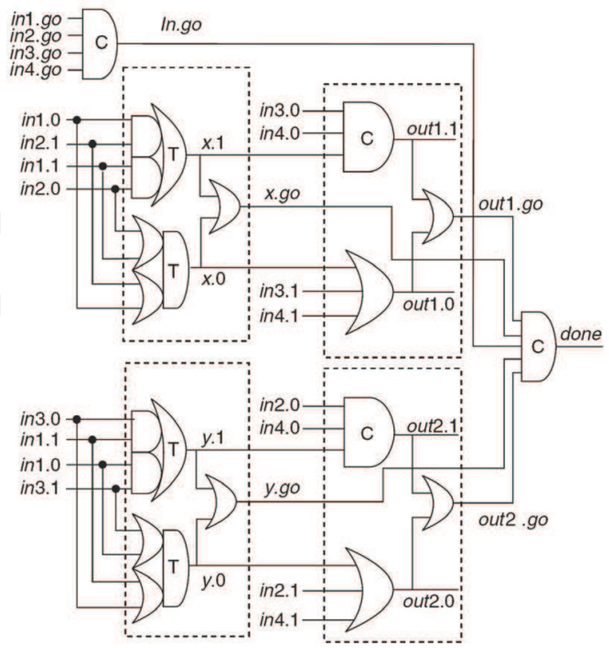

Fig. 13. Encoder circuit

Table 8 gives the simulation run times and the corresponding allocated memory for all these circuits. These results are for 600 timing simulations per circuit, achieved on an AMD Athlon processor at 1.14 GHz with 1 GB RAM.

out.	delay type	topol. level	min/ max	fan-out	statistical mean	statistical dev.
1.	mnr	1	0.90ns	0.9ns	0.902	0.036
	mxr	3	2.85ns	3.8ns	3.823	0.055
	mnf	1	1.00ns	1.0ns	0.999	0.036
	mxf	3	3.15ns	4.2ns	4.218	0.060
2.	mnr	1	0.9ns	0.9ns	0.900	0.038
	mxr	3	2.95ns	3.9ns	3.917	0.060
	mnf	1	1.00ns	1.0ns	0.998	0.035
	mxf	3	3.05ns	4.1ns	4.121	0.067
3.	mnr	2	1.80ns	1.8ns	1.802	0.049
	mxr	5	4.95ns	5.9ns	5.957	0.066
	mnf	2	2.00ns	2.0ns	1.999	0.049
	mxf	5	5.15ns	6.2ns	6.263	0.066
4.	mnr	1	0.90ns	0.9ns	0.901	0.036
	mxr	3	2.85ns	3.8ns	3.819	0.058
	mnf	1	1.00ns	1.0ns	1.001	0.037
	mxf	3	3.15ns	4.2ns	4.222	0.055
5.	mnr	1	0.90ns	0.9ns	0.897	0.037
	mxr	3	2.95ns	3.9ns	3.920	0.060
	mnf	1	1.00ns	1.0ns	0.999	0.036
	mxf	3	3.05ns	4.1ns	4.120	0.069

Table 7. Encoder.

circuit	CPU time [s]	allocated memory [kB]
C element	6.3	19.697
T counter	7.7	20.277
Addr. comp	8.2	25.734
Gen C elem	10.5	40.443
Encoder	28.5	92.583

Table 8. Simulation run times and allocated memory.

We have chosen to compare our method with Monte-Carlo analysis based on the application of standard logic simulations. The merits for comparison are estimation times, complexities of the approach (number of simulator runs or simulations), required resources. Considering the quality of the results, it is very difficult to compare these two approaches, since classical Monte-Carlo analysis requires a huge amount of time for data processing, and obtaining it in a form that can be comparable with our method. The Table 9 gives the results of comparison.

circuit	allocated memory [kB]	Estimation time [s] (CPU time)	Number of simulations
C element	444	1	16
T counter	375	2	2
Addr. comp	446	3	4
Gen C elem	452	4	255
Encoder	480	436	1048576

Table 9. Allocated memory, simulation run times and number of simulations for Monte-Carlo analysis.

Beside the columns that show simulation run times and the corresponding allocated memory for this approach, the last column in the table shows how many Monte-Carlo standard simulations are performed. These simulations are performed for all possible that for complex circuit the number of possible input combinations can be high enough to enable statistical processing of the results. But, with a simple circuits, where the number of inputs is not large enough (if all possible combinations of inputs signals is less than few hundreds), statistical results processing is impossible. The major improvement achieved with the proposed method is that all possible input vector combinations are analysed with only one run of the simulator, that is, with only one timing analysis cycle. Only one run of the simulator is needed for obtaining all possible worst-case delays.
The proposed method was previously verified on a set of combinational ISCAS benchmark circuit (Sokolović & Litovski 2005; Lin & Davoodi 2008; iscas89.html --), with up to 9000 gates, and no bottleneck resulted.

6. Conclusion

A new concept for asynchronous circuit delay analysis was presented in this paper. The method is based on a specific signal and gate modelling. Signals in the circuit do not carry the information about the logic value of the signal, but the information about the delay and about the arrival of the information instead. Specific attributes of the signals are introduced in order to implement that property. The method generates and exploits information about the fanout of each gate implemented in a complex digital system. It has the ability to deal with sequential elements containing loops and feedbacks. It was implemented in VHDL and verified on some particular asynchronous circuits. This way of delay modelling allows for the delay to become a random variable so that the delay estimation method was incorporated into Monte-Carlo analysis to produce statistical worst-case delay distributions. The obtained delay distributions of the output signals can be used for any kind of timing analysis which will consider the simulated circuit as a whole. This also includes the for example incremental timing analysis. Having in mind the number and the complexity of the gate processes, and the number of simulations required to achieve high accuracy, the simulation times for each gate, as well as the allocated memory show high efficiency of the proposed method, comparing to very intensive and complex Monte-Carlo analysis.

7. References

Agarwal, A. et al. (2003). Statistical delay computation considering spatial correlations. *Proc. of the ASP-DAC 2003*, pp. 271 – 276, ISBN: 0-7803-7659-5, Kitakyushu, Japan, January 2003.

Agarwal, A. et al. (2005). Circuit Optimization using Statistical Static Timing Analysis. *Proc. of the 42nd Annual Conf. on Design Automation*, pp. 321-324, ISBN: 1-59593-058-2, San Diego, California, 2005.

Andrikos, N. (2007). A fully-automated desynchronization flow for synchronous circuits. *Proceedings of the 44th annual conference on Design automation*, pp. 982-985, San Diego, California, Juns 2007, ISSN: 978-1-59593-627-1.

Cheng, T. & Agrawal, V. (1989). *Unified Methods for VLSI Simulation and Test Generation*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1989, ISBN: 978-0-7923-9025-1.

Cortadella, J. et al. (1999). Synthesis of asynchronous control circuits with automatically generated relative timing assumptions. *Proceedings of the 1999 IEEE/ACM international conference on CAD*, pp. 324-331, ISBN:0-7803-5832-5, 1999, IEEE Press Piscataway, NJ, San Jose, California.

Cortadella, J. et al. (2006). Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications. *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 25, No. 10, October 2006, pp. 1904-1921, ISSN: 0278-0070. http://courses.ece.uiuc.edu/ece543/iscas89.html

Davis, A. & Nowick, S. (1997). *An Introduction to Asynchronous Circuits Design*, Technical Report UUCS-97-013. Septemper 1997, Computer Science Department, University of Utah.

Kondratyev, A. & Lwin, K. (2002). Design of Asynchronous Circuits Using Synchronous CAD Tools. *IEEE Design & Test*, Vol. 19, Issue 4, July 2002, pp. 107 – 117, ISSN: 0740-7475.

Lewis, M. & Brackenbury, L. (2001). CADRE: A Low-power, Low-EMI DSP Architecture for Digital Mobile Phones. *VLSI Design, special issue on low power system design*, Vol. 3, Issue 12, September 2001, pp. 333-348.

Lin, X. & Davoodi, A. (2008). Robust Estimation of Timing Yield with Partial Statistical Information on Process Variations. *Proc. of A Quality Electronic Design, ISQED 2008*, pp. 156 – 161, ISBN: 978-0-7695-3117-5, San Jose, California, USA, March 2008.

Litovski, V. & Zwolinski, M. (1997). *VLSI circuit simulation and optimization*. Chapman and Hall, London, ISBN: 0412638606, 1997.

Maksimović, D. (2000). *Logic Simulation – Estimation of the Worst-case characteristics of the Designed Digital Circuits*. PhD thesis, Faculty of Electronic Engineering, University of Niš, Serbia, June 2000.

Maksimović, D. & Litovski, V. (1999). Tuning Logic Simulators for Timing Analysis. *Electronic Letters*, Vol. 35, No. 10, May 1999, pp. 800-802, ISSN: 0013-5194.

Maksimović, D. & Litovski, V. (2002). Logic Simulation Methods for Longest Path Delay Estimation. *IEE Proc. Computers and Digital Technique*, Vol. 149, No. 2, March 2002, pp. 53-59, ISSN: 1350-2387.

Mak, T.M. et al. (2004). New Challenges in Delay Testing of Nanometer, Multigigahertz Designs. *Design & Test of Computers*, Vol. 21, Issue 3, May-June 2004, pp. 241-248, ISSN: 0740-7475.

Martin, A. & Nystrom, M. (2006). Asynchronous Techniques for System-on-Chip Design. *Proceedings of the IEEE*, pp. 1089 – 1120, Vol. 94, Issue 6, June 2006, ISSN: 0018-9219.

Myers, C. (2001). *Asynchronous Circuit Design*. University of Utah, John Wiley & Sons, Inc. May, 2001, ISBN: 0-471-41543-X.

Myers, C. & Alain, M. (1993). *The Design of Asynchronous Memory Management Unit. Technical Report CS-TR-93-30*, California Institute of Technology, April 1993.

Sokolovic, M. & Litovski, V. (2005). Using VHDL Simulator to Estimate Logic Path Delays in Combinational and Embedded Sequential Circuits, *Proceedings of IEEE Region 8 EUROCON 2005*. Conference, pp. 547-550, ISBN: 1-4244-0049-X, November 2005, Belgrade.

Sokolovic, M., Litovski, V. & Zwolinski, M. (2009) New concepts of worst-case delay and yield estimation in asynchronous VLSI circuits. *Microelectronics Reliability*, Vol. 49, Issue 2, pp. 186-198. ISSN 0026-2714.

Sparso, J. (2006). *Asynchronous Circuit Design – A Tutorial.* April 2006, Technical University of Denmark.

Spence, R. & Soin, R. (1988). *Tolerance design of Electronic circuits*. Addison-Wesley Publ. Comp. Wokingham, England, ISBN: 978-1-86094-040-8, 1988.

Zwolinski, M. (2004). *Digital System Design with VHDL*, Prentice Hall, London, UK, 2004, ISBN 0-13-039985-X.

IntechOpen

Micro Electronic and Mechanical Systems

Edited by Kenichi Takahata

ISBN 978-953-307-027-8

Hard cover, 386 pages

Publisher InTech

Published online 01, December, 2009

Published in print edition December, 2009

This book discusses key aspects of MEMS technology areas, organized in twenty-seven chapters that present the latest research developments in micro electronic and mechanical systems. The book addresses a wide range of fundamental and practical issues related to MEMS, advanced metal-oxide-semiconductor (MOS) and complementary MOS (CMOS) devices, SoC technology, integrated circuit testing and verification, and other important topics in the field. Several chapters cover state-of-the-art microfabrication techniques and materials as enabling technologies for the microsystems. Reliability issues concerning both electronic and mechanical aspects of these devices and systems are also addressed in various chapters.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Miljana Milić and Vančo Litovski (2009). New Concepts of Asynchronous Circuits Worst-Case Delay and Yield Estimation, Micro Electronic and Mechanical Systems, Kenichi Takahata (Ed.), ISBN: 978-953-307-027-8, InTech, Available from: http://www.intechopen.com/books/micro-electronic-and-mechanical-systems/new-concepts-of-asynchronous-circuits-worst-case-delay-and-yield-estimation

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2009 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License, which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen