New Concepts of Worst-case Delay Evaluation in Asynchronous VLSI SoC

M. Sokolović, M. Zwolinski, and V. Litovski

Abstract – Estimation of asynchronous circuit performances, such as speed, is one of the major issues that make that design style still less popular than it deserves to be. Evaluating the worst case delays in the paths of an asynchronous circuit using a simple logic simulator would be very useful in overcoming this problem. In this paper a method for timing analysis of the asynchronous non-sequential circuits using a VHDL simulator is presented. With an appropriate extension of the standard logic simulation process, all worst case delays for all paths in a digital circuit with only one run of the simulation can be obtained. High levels of accuracy are achieved using different gate modelling and statistical analysis of the results. Due to the lack of asynchronous benchmark circuits, the method is verified on a set of asynchronous circuit selected by the authors.

I. Introduction

Digital circuit design styles can be classified into two major categories, namely synchronous and asynchronous. A hybrid design style mixes aspects of both categories. The major differences between synchronous and asynchronous circuit lie in the system timing. Synchronous circuits may be simply defined as circuits which are sequenced by one or more globally distributed periodic timing signals called clocks. Any state change that occurs happens on the clock edge, and so system states are predictable. All these issues provide a basis for good design, and simulation tool support. Asynchronous circuits are an inherently larger class of circuits, since they use events to control timing [1], and no clock is used to implement sequencing. Such circuits are also called clockless [2], [3]. Their timings are very difficult to predict and it is the main reason for poor design tool support [4].

Although asynchronous design is still a less travelled road for designers, the benefits of this design style are unanswerable. Such circuits need no clock generation and distribution, which saves a lot of chip area, and they leave the problems related to clock skew behind. Asynchronous circuits are characterized with good modularity and much easier technology migration. 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. Very low EMI occurs during operation, while achieving high noise immunity [4].

There is a lack of motivation for asynchronous circuit techniques since synchronous circuit design styles have large commercial practice and considerable pedigree [1]. For some, 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" [5]. Because of that, some techniques for desynchronization of synchronous circuits have appeared lately [6], [7].

Nevertheless, 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 paths delays in the circuit helps in avoiding early timing problems as well as in circuit performance characterisation. Precise paths delays can be estimated in the final steps of the design process. 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 delay estimation should be performed during the early phases of circuit design.

Simulation is the simplest way to determine the circuit delay. For complex circuits, simulation at the transistor level becomes impractical. To verify the logic function and the timing 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 is the number of inputs. Therefore, the simulation becomes inefficient 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.

Parameter values in electronic circuits' component vary because of the following causes: process and technology; environment and temperature; and specific phenomena within components (such as electromigration). These variations affect the circuit behaviour over time. Because of them, a 100% 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

M. Sokolović and V. Litovski are with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: {miljana, vanco}@elfak.ni.ac.yu
M. Zwolinski is with the Electronics Systems and Devices Group, School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, United Kingdom, E-mail: mz@ecs.soton.ac.uk

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 tolerance onto response tolerance [8].

Timing analysis tools perform the timing analysis of a circuit consisting of primitive gates. They can calculate circuit delays that are the result of parametric variations. The aim of statistical timing analysis is to find how much time might be needed to guarantee that the response of the circuit to any input vector would always be obtained within that time [9].

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, in 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 [10].)

Sampling methods and direct methods can be used instead of worst-case and statistical timing analysis. In direct methods, it is assumed that formulae for mapping the parameter tolerances into the response tolerances are available. These methods are usually limited 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.

Our aim in this paper is to demonstrate the application of a standard logic simulator in asynchronous circuit path delay analysis. Some known methods for timing analysis will be given and compared. A new and very efficient procedure for worst-case delay estimation which can introduce statistical approach and is based on an accurate delay model that takes into account the fanout influence on the total delay will be presented. The procedure is based on netlist modification, multiple simulations and statistical data processing. This required specific modelling of the asynchronous circuits' building blocks. A VHDL implementation of the method follows. The problem in the verification step of the proposed method is the lack of asynchronous benchmark circuits. The method is verified on a set of chosen asynchronous circuit. All results show the excellent performance of the method.

II. Ways to Analyze Timings

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

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

Timing measurements, as already mentioned, can be performed using a circuit simulation, but such an approach is too slow and impractical. To avoid simulations, there are two alternative approaches for delay estimation in logic circuits. The first 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 pdfs (probability density functions) are obtained. The percentage of fabricated dies which meet a required delay, can then be calculated or conversely, the expected performance for a particular yield [11]. The probabilistic nature of the timing behaviour of a circuit imposes the statistical analysis and simulation in the selection of critical path. 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 [10].

The Monte Carlo method requires a large number of circuit simulations (analyses), giving the mean and standard deviation of the delay at the output of the circuit as the results. Monte Carlo simulation 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 is chosen according to the given probability distribution.

An analysis of the circuit must be performed for each new parameter value. In this way a set of different parameter values of the circuit output signals are obtained. The analysis step 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 results. Once the mean or variance converges to the desired precision range, the procedure terminates. It takes from a few tens to a few hundreds of Monte Carlo simulations to achieve convergence of the results. This means that the timing analysis step should be repeated that many times [12]. However, since 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 [13].

The use of a logic simulator for the timing analysis in the Monte Carlo analysis can significantly speed up the design and analysis time. A new, simplified way for timing analysis with a VHDL logic simulator 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.

III. Delay Estimation with a Logic Simulator

Our method for path-delay estimation in asynchronous non-sequential 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, the suggested method cannot deal with circuits with feedback loops, unless they are broken. 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 delay estimation of all the paths in a circuit using a logic simulator, that is a timing analysis, a small modification to the logic simulation mechanism is needed [14]. 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 [10], [15], 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, 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.

A. Modelling signals and gates

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 only 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.

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

Fig. 1. Process for assigning the maximal delay of the falling and rising edges for a two input NOR gate.

An example of the process for assigning the maximal delay of the falling and rising edges for a two input NOR gate is shown in Figure 1. The gate inputs are denoted as *in1* and *in2*, and the output as *out1*. The gate propagation delays for the rising and falling edges at the output *out1* are denoted by *tpd_lhmx* (maximal time propagation delay from low to high) and *tpd_hlmx*, respectively. Each rising transition at an input of the gate means that one of the falling transition flags at one of the input signals (in1.arr1mx or in2.arr1mx) becomes "true". This sets a rising transition flag at the output signal attribute of the gate (out1.arr1mx) to "true". This corresponds to an OR function. Simultaneously with setting the output flag, the gate model calculates a new value for the longest path delay. The resulting output longest path delay attribute for the falling edge is denoted by out1.d0mx and is calculated after taking into account the arriving longest path delays for both gate input signals (in1.d1mx, in2.d1mx), the maximal delay of the falling edge for this gate (a separate function assigns this value) and the function *f* which depends on the gate fanout value. Conversely, a falling transition flag at any of the gate input signals (in1.arr0mx, in2.arr0mx) produces a rising transition at the output only if a falling transition has previously arrived at the other gate input [9], [16]. This corresponds to an AND function. The resulting output longest path delay attribute for the falling edge, denoted by out1.d1mx, takes into account arrived longest path delay input signal attributes (in1.d0mx, in2.d0mx), the maximal delay of the rising edge for this gate and the fanout dependent function *f*. A similar process for the shortest path delay estimation is given in process_mn. A delay model of arbitrary complexity can be applied. 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*.

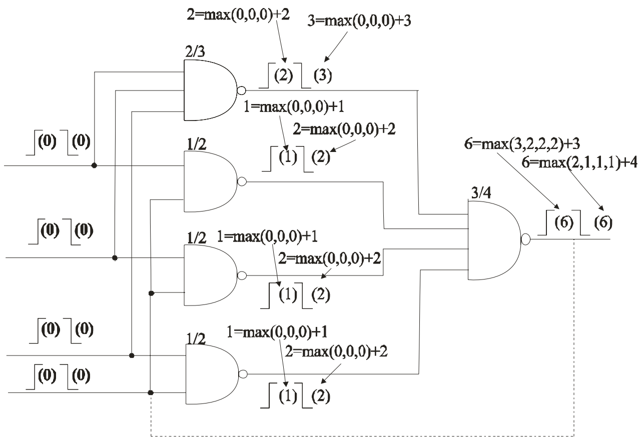


Fig. 2. Illustration of maximal delay estimation method for a 3-input C element.

The basic principle of delay accumulation is described in Figure 2. The figure describes the maximal delay calculation of all paths in the simple asynchronous circuit called C-element. Here, both rising and falling transitions are applied to all inputs of the circuit. Both the 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.

B. Giving 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 Figure 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 the Eq. 1:

```
random_value_of(tpd_lhmx/tpd_hlmx/tpd_lhmn/tpd_hlmx)
*fanout_func(output)
___________________________________________________
=func(tpd_lhmx/tpd_hlmx/tpd_lhmn/tpd_hlmx)              (1)
```

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,

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

where μ_p represents the mean value and σ_p^2 is the variance of the random variable *p* [8]. This function can, of course, be changed if necessary.

The second component of equation (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 logical 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.

C. The path delay estimation algorithm

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 is determined by the required precision and accuracy of the results. Table 1 gives a description of estimation phases for one sample.

The delay values of a particular gate have standard deviation σ, 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 also do not require any kind of stimuli, since they take into account all possible signal transitions. Only initialization is needed for the calculation processes in the entire circuit.

TABLE I
PATH DELAY ESTIMATION ALGORITHM WITH A LOGIC SIMULATOR

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	-Initialize all signals to have zero values of the delay attributes
step 4:	-Initialize the calculation process by setting the primary input signal triggering flags to "true" -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".

IV. IMPLEMENTATION

As mentioned before, 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. Figure 3 show a VHDL implementation of the random number generator with a Gaussian distribution [17].

```
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:=u1*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. 3. Gaussian random function generation implementation.

This function is executed 4 times in each of the gates, 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 consists of only one logic NAND gate. This circuit was simulated 600 times and the results of the randomly generated delays of rising and falling edges at the output of this circuit are shown in Figure 4. In this case, the mean in the distribution is set to 1ns 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.

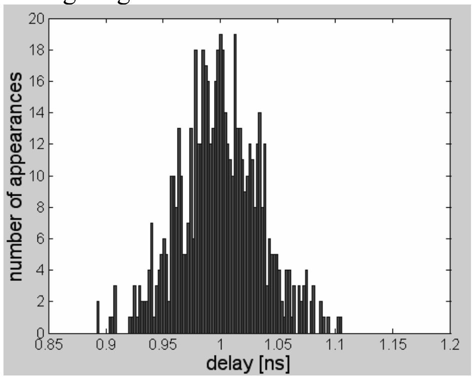


Fig. 3. Histogram of the delays for NAND gate.

VHDL models of primitive logic gates and simple asynchronous elements are kept in a VHDL library. Figure 4, 5, and 6 show VHDL modelling of a D-latch, T-latch and 2-input C-element respectively. It should be noted that latch circuits do not have specific conditions for activating a calculation process, because these circuits have only 1 data input. It should also be emphasized that the T-latch performs output change only when a falling transition happens at its data input. Accordingly, only the delay of the falling edge at its input can influence the delay calculation process further.

```
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. 4. VHDL model of the Dlatch.

```
entity TLatch is
        generic (ifo_izl: integer:= 1;
                tr_en_qmn : real := 1.0e-9;
                tf_en_qmn : real := 0.9e-9;
                tr_en_qmx : real := 1.05e-9;
                tf_en_qmx : real := 0.95e-9);
        port (q : out SDA_std_logic := (0.0, 0.0, false, false, 0.0, 0.0,
        false, false);
                t : in SDA_std_logic := (0.0, 0.0, false, false, 0.0, 0.0, false,
        false));
end TLatch;
architecture only of TLatch is
begin
        p1: process (t.d0mn, t.d1mn, t.arr0mn, t.arr1mn,
        t.d0mx, t.d1mx, t.arr0mx, t.arr1mx)
                variable i, j ,k, l: 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*1.05 + (0.03*(gauss_rng)));
                l:= (f*0.95 + (0.03*(gauss_rng)));
                q.arr1mn <= true;
                q.arr0mn <= true;
                q.d1mn <=  t.d0mn + i ;
                q.d0mn <=  t.d0mn + j ;
                q.arr1mx <= true;
                q.arr0mx <= true;
                q.d1mx <= t.d0mx + k ;
                q.d0mx <= t.d0mx + l ;
        end process;
end only;
```

Fig. 5. VHDL model of the T-Latch.

```
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. 6. VHDL timing processes for the 2-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;
```

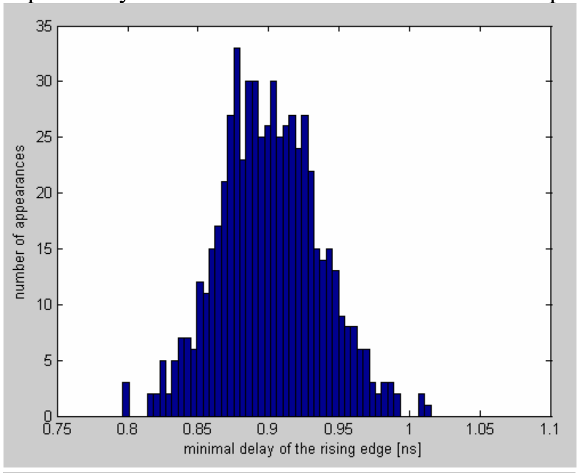
Fig. 7. Testbench process for writing simulation results for minimal delay of the rising edge of all encoder output signals into a file.

To simulate the circuit 600 times, a specific VHDL testbench is necessary. Here the netlist is instantiated a few hundred times. Now, for each particular input of all these circuits, and for initialization and for the simulation itself, a specific matrix is formed. All responses to the logic analysis are stored in the matrix. Matrices of input and output signals are defined with variables of type input_mat and output_mat. This is shown in Figure 8. This code contains the description of the process that performs the timing analysis – log_timing1, which is used for determining the minimal delay of the rising edges of the output signals for one asynchronous encoder circuit. This circuit 5 outputs. All results of this analysis are written to a text file (encoder_mn_r.statdel).

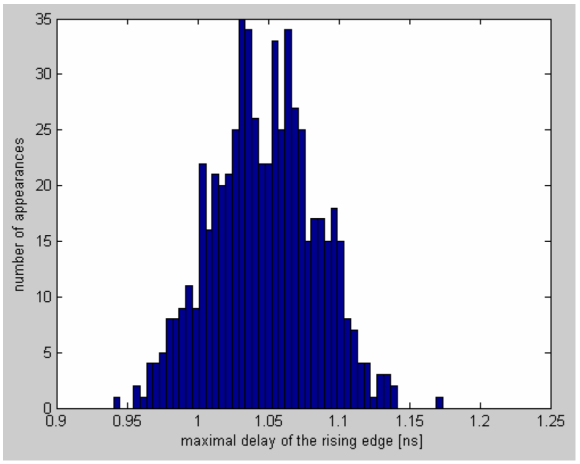
V. Examples

When a circuit is simulated 600 times, a huge amount of data can be expected. Since each gate model consists of four parallel processes, for each of the signal transitions, that gives the equivalent to four parallel simulations during each run of the analysis. In effect, 4*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.

Figure 8 shows the results 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.



a)



b)

Fig. 8. Histogram of the C-element gate.

Table II shows the simulation results of the C-element described at the structural level of abstraction shown in Figure 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.

TABLE II
C GATE - STRUCTURAL

output	delay type	top. level	min/ max	fan- out	statistical	
					mean	dev.
1.	mnr	2	1.9ns	1.9ns	1.860	0.460
	mxr	2	2.0ns	2.0ns	2.035	0.421
	mnf	2	1.9ns	1.9ns	1.861	0.436
	mxf	2	2.0ns	2.0ns	2.035	0.443

Table III shows the simulation results of the four stage asynchronous binary counter consisting of 4 T latches. Table IV gives the timing analysis results for a generalized C-element [18], shown in Figure 9.

TABLE III
T COUNTER

output	delay type	top. 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

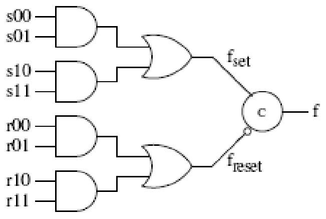


Fig. 9. Generalized C element.

TABLE IV
GENERALIZED C ELEMENT

output	delay type	top. 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

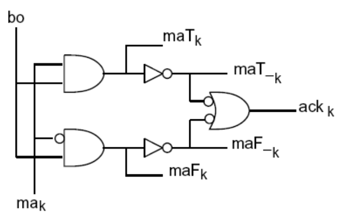


Fig. 10. Address comparator.

A simple asynchronous address comparator unit [19] is shown in Figure 10, and its simulation results are presented in Table V.

TABLE V
ADDRESS COMPARATOR

out.	delay type	topo. 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

Finally, Figure 11 shows one complex asynchronous encoder circuit described in [20], while Table VI gives its timing analysis results.

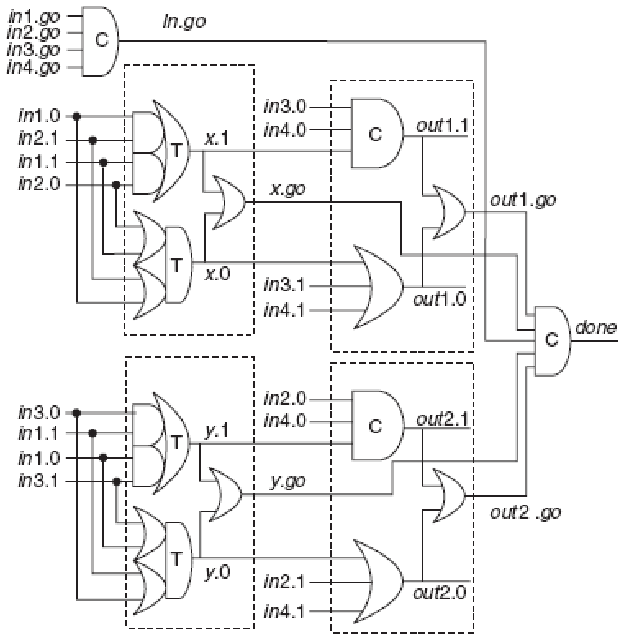


Fig. 11. Encoder circuit.

TABLE VI
ENCODER

out.	delay type	topo. 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 VII 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 1.14GHz with 1GB RAM.

TABLE VII
SIMULATION RUN TIMES AND ALLOCATED MEMORY

circuit	CPU time [s]	allocated memory [kB]
C element	6.4	19.734
T counter	7.7	20.277
Addr. comp	8.2	25.734
Gen C elem	10.5	40.443
Encoder	28.5	92.583

VI. CONCLUSION

A new concept for asynchronous circuit delay analysis was presented in this paper. The estimation method was incorporated into Monte-Carlo analysis, so that the obtained results of this analysis represent statistical worst-case delay ranges. The method generates and exploits information about the fanout of each gate implemented in a complex digital system and was implemented in VHDL and verified on some particular asynchronous circuits.

REFERENCES

[1] A. Davis, and S Nowick, "An Introduction to Asynchronous Circuits Design", *Technical Report* UUCS-97-013, Computer Science Department, University of Utah, September 1997.

[2] J. Sparso: "Asynchronous Circuit Design – A Tutorial", Technical University of Denmark April 2006.

[3] A. Martin, and M Nystrom, "Asynchronous Techniques for System-on-Chip Design", *Proceedings of the IEEE*, vol. 94, issue 6, June 2006, pp. 1089 - 1120.

[4] M. Lewis and L. Brackenbury: "CADRE: A Low-power, Low-EMI DSP Architecture for Digital Mobile Phones", *VLSI Design*, vol. 3, issue 12, 2001, pp. 333-348.

[5] J. Cortadella et al.: "Synthesis of asynchronous control circuits with automatically generated relative timing assumptions", *Proceedings of the 1999 IEEE/ACM international conference on CAD*, San Jose, California pp. 324-331.

[6] J. Cortadella et al.: "Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications", *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 25, no. 10. pp. 1904-1921, October 2006.

[7] N. Andrikos et al.: "A fully-automated desynchronization flow for synchronous circuits", *Proceedings of the 44th annual conference on Design automation*, San Diego, California, pp. 982-985.

[8] V. Litovski and M. Zwolinski.: "VLSI circuit simulation and optimization", *Chapman and Hall*, London, 1997.

[9] R. Spence and R. Soin.: "Tolerance design of Electronic circuits", *Addison-Wesley Publ. Comp.* Wokingham, England, 1988.

[10] T. Mak, et al.: "New Challenges in Delay Testing of Nanometer, Multigigahertz Designs", *Design & Test of Computers*, vol. 21, issue 3, May-June 2004, pp. 241-248.

[11] A. Agarwal, ed al "Circuit Optimization using Statistical Static Timing Analysis", *Proc. of the 42nd Annual Conf. on Design Automation*, San Diego, California, 2005, pp. 321-324.

[12] http://courses.ece.uiuc.edu/ece543/iscas89.html

[13] A. Agarwal, et al.: "Statistical delay computation considering spatial correlations", *Proc. of the* ASP-DAC, 2003, Kitakyushu, Japan, 2003, pp. 271 – 276.321-324.

[14] D. Maksimović, et al.: "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.

[15] D. Maksimović, and V. Litovski: "Tuning Logic Simulators for Timing Analysis", *Electronic Letters*, vol. 35, no. 10, May 1999, pp. 800-802.

[16] D. Maksimović, and V. Litovski: "Logic Simulation Methods for Longest Path Delay Estimation", *IEE Proc. Computers and Digital Technique*, vol. 149, no. 2, March 2002, pp. 53-59.

[17] M. Zwolinski: "Digital System Design with VHDL", *Prentice Hall*, London, UK, 2004.

[18] C. Myers: "Asynchronous Circuit Design", Unversity of Utah, John Wiley & Sons, Inc. May, 2001.

[19] C. Myers and M. Alain "The Design of Asynchronous Memory Management Unit" *Technical Report* CS-TR-93-30, California Institute of Technology, April 1993.

[20] A. Kondratyev and K Lwin: "Design of Asynchronous Circuits Using Synchronous CAD Tools", *IEEE Design & Test*, vol. 19, issue 4, July 2002, pp 107 – 117.