Method of Statistical Timing Analysis with Uncertainty

Nazeli Melikyan

Abstract **–** Static timing analysis is a critical step in design of digital integrated circuits. Technology and design trends have led to significant increase in environmental and process variations which need to be incorporated in static timing analysis. This paper presents a new static timing analysis technique considering uncertainty. This new method is more efficient as its models arrival times as cumulative density functions and delays as probability functions.

Keywords – static timing analysis, uncertainty, density function.

I. Introduction

Static timing analysis (STA) is critical to the measurement and optimization of the circuit performance before its manufacture.

The timing or performance of the chip is heavily dependent on the manufacturing process variations (e.g. Vt, Length, etc.) and design environment variations (e.g. VDD and temperature variations, noise impact on timing, etc.). As the feature sizes decrease, the ability to control the manufacturing spread or accuracy of a given feature size is also decreasing. Along with increased process variations, the uncertainty caused by design is also increasing. The increase of uncertainty in design is caused by increase of power supply and temperature variations and interconnect loading uncertainty such as coupling noise impact on timing.

Design variations or uncertainty in static timing analysis is typically handled in two broad ways. The first set of techniques handle variations by worst casing the circuit response. In such a scenario, static timing is performed at various design corners (e.g. fast, slow and nominal design corner).

Another method to handle variations in timing is to perform statistical timing analysis [1,2].

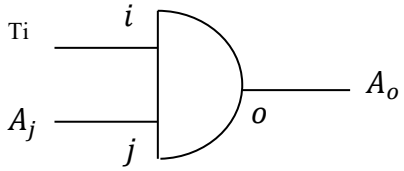
This paper presents a new statistical timing analysis technique. The delay and arrival times in the circuit are modeled as random variables. The arrival times are modeled as Cumulative Probability Distribution Functions and the gate delays are modeled as Probability Density Functions. This leads to efficient expressions for both max and addition operations, the two key functions in both regular and statistical timing analysis.

Nazeli Melikyan is PhD student of Microelectronic Circuits and Systems Chair of National Polytechnic University of Armenia, E-mail: nazeli@synopsys.com.

II. Statistical Timing Analysis

The problem in deterministic static timing analysis is to compute arrival times at the output nodes. Arrival times at the input and delay of the gates are specified as deterministic numbers. In case of statistical timing analysis, the arrival times and delays of the gates are specified as distributions. In general, the distribution of delays of the gates can take any form (i.e. normal, uniform, etc.). The problem in statistical timing analysis is to compute distribution of arrival times at the intermediate nodes and the output nodes. Given the required arrival time and distribution of output arrival times, critical paths and slack distributions can be computed for a given probability or confidence level.

Timing analysis is performed by levelizing the circuit. The arrival time at the input is propagated through the gates at each level till it reaches the output. Propagating the arrival times through a gate is a key function in static timing. Consider a two input gate shown in Figure 1.



D_{io}:Delay from Input Node i to Output node o
D_{jo}:Delay from Input Node j to Output node o

Figure 1: A gate with output o and inputs i and j.

In deterministic static timing analysis, arrival time at output node o is given by:

$$A_o = \max(A_i + D_{io}, A_j + D_{jo}) \qquad (1)$$

Computation of max and addition is straight forward in regular timing analysis. In the proposed approach arrival times are modeled as cumulative density functions and the delays are modeled as probability density functions.

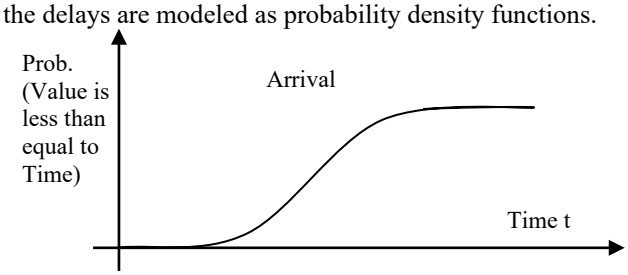


Figure 2: Arrival times are modeled as Cumulative Probability Density Functions.

III. Results

By the proposed method of Statistical Timing Analysis for evaluation the arbitrator circuit has been chosen for the research, which is widely used in electronics (Fig. 3). It is mainly used in asynchronous circuits. During asynchronous requests, the sequence of implementing these requests is selected by means of this circuit. Therefore arbitrator circuit prevents the occurrence of actions at a time when not permitted in the given system. Thus it can be assumed that the arbitrator circuit significantly reduces the probability of having metastability. Preference encoder is used in the arbiter circuit, which enables defining the preferences of requests.

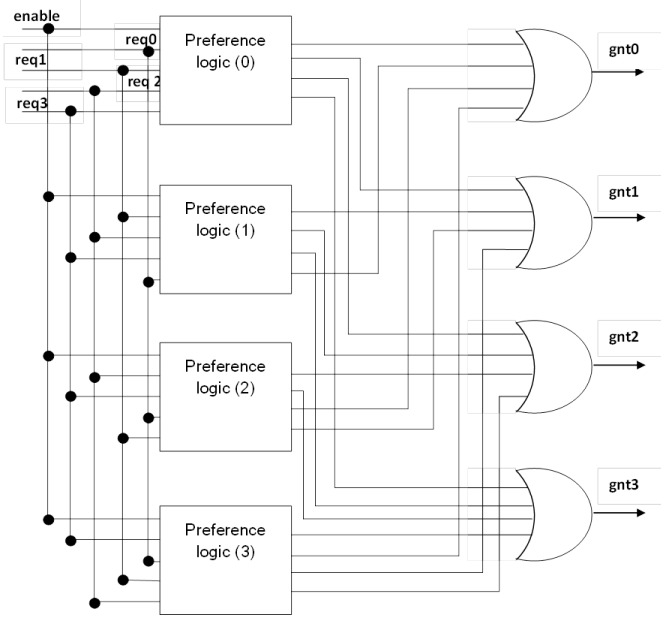


Figure 3: Functional circuit of an arbitor

Behavioral description scheme

Behavioral description has been performed by Verilog hardware description language. Below are behavioral descriptions of an arbitrator.

```
//-----------------------------------------------------
// A four level, round-robin arbiter.
//-----------------------------------------------------
module arbiter (
  clk,
  rst,
  req3,
  req2,
  req1,
  req0,
  gnt3,
  gnt2,
  gnt1,
  gnt0
);
// --------------Port Declaration----------------------
input       clk;
input       rst;
input       req3;
input       req2;
input       req1;
input       req0;
output      gnt3;
output      gnt2;
output      gnt1;
output      gnt0;

//--------------Internal Registers---------------------
wire   [1:0]  gnt     ;
wire          comreq  ;
wire          beg     ;
wire   [1:0]  lgnt    ;
wire          lcomreq ;
reg           lgnt0   ;
reg           lgnt1   ;
reg           lgnt2   ;
reg           lgnt3   ;
reg           lasmask ;
reg           lmask0  ;
reg           lmask1  ;
reg           ledge   ;

//--------------Code Starts Here-----------------------
always @ (posedge clk)
if (rst) begin
  lgnt0 <= 0;
  lgnt1 <= 0;
  lgnt2 <= 0;
  lgnt3 <= 0;
end else begin
  lgnt0 <=(~lcomreq & ~lmask1 & ~lmask0 & ~req3 &
~req2 & ~req1 & req0)
       | (~lcomreq & ~lmask1 &  lmask0 & ~req3 &
~req2 &  req0)
       | (~lcomreq &  lmask1 & ~lmask0 & ~req3 &
req0)
       | (~lcomreq &  lmask1 &  lmask0 & req0  )
       | ( lcomreq & lgnt0 );
  lgnt1 <=(~lcomreq & ~lmask1 & ~lmask0 &  req1)
       | (~lcomreq & ~lmask1 &  lmask0 & ~req3 &
~req2 &  req1 & ~req0)
       | (~lcomreq &  lmask1 & ~lmask0 & ~req3 &  req1
& ~req0)
       | (~lcomreq &  lmask1 &  lmask0 &  req1 & ~req0)
       | ( lcomreq &  lgnt1);
  lgnt2 <=(~lcomreq & ~lmask1 & ~lmask0 &  req2  &
~req1)
       | (~lcomreq & ~lmask1 &  lmask0 &  req2)
       | (~lcomreq &  lmask1 & ~lmask0 & ~req3 &  req2
& ~req1 & ~req0)
```

```
      | (~lcomreq &  lmask1 &  lmask0 &  req2 & ~req1
& ~req0)
      | ( lcomreq &  lgnt2);
  lgnt3 <=(~lcomreq & ~lmask1 & ~lmask0 & req3  &
~req2 & ~req1)
      | (~lcomreq & ~lmask1 &  lmask0 & req3  &
~req2)
      | (~lcomreq &  lmask1 & ~lmask0 & req3)
      | (~lcomreq &  lmask1 &  lmask0 & req3  & ~req2
& ~req1 & ~req0)
      | ( lcomreq & lgnt3);
end

//-----------------------------------------------------
// lasmask state machine.
//-----------------------------------------------------
assign beg = (req3 | req2 | req1 | req0) & ~lcomreq;
always @ (posedge clk)
begin
  lasmask <= (beg & ~ledge & ~lasmask);
  ledge   <= (beg & ~ledge &  lasmask)
       |  (beg &  ledge & ~lasmask);
end


encoder encoder (
                  .lgnt0(lgnt0), .lgnt1(lgnt1),
.lgnt2(lgnt2), .lgnt3(lgnt3),
                  .req0(req0), .req1(req1), .req2(req2),
.req3(req3),
                  .lgnt(lgnt), .lcomreq(lcomreq)
                  );
//-----------------------------------------------------
// lmask register.
//-----------------------------------------------------
always @ (posedge clk )
if( rst ) begin
  lmask1 <= 0;
  lmask0 <= 0;
end else if(lasmask) begin
  lmask1 <= lgnt[1];
  lmask0 <= lgnt[0];
end else begin
  lmask1 <= lmask1;
  lmask0 <= lmask0;
end

assign comreq = lcomreq;
assign gnt    = lgnt;
//-----------------------------------------------------
// Drive the outputs
//-----------------------------------------------------
assign gnt3   = lgnt3;
assign gnt2   = lgnt2;
assign gnt1   = lgnt1;
assign gnt0   = lgnt0;
```

```
endmodule
```

Below is the behavioral description of preference encoder, used in the arbitrator.

```
module encoder
(lgnt0,lgnt1,lgnt2,lgnt3,req0,req1,req2,req3,
lcomreq, lgnt);
input         req3;
input         req2;
input         req1;
input         req0;
input         lgnt0    ;
input         lgnt1    ;
input         lgnt2    ;
input         lgnt3    ;
output                    lcomreq;
output  [1:0] lgnt;
//-----------------------------------------------------
// comreq logic.
//-----------------------------------------------------
assign lcomreq = ( req3 & lgnt3 )
               | ( req2 & lgnt2 )
               | ( req1 & lgnt1 )
               | ( req0 & lgnt0 );

//-----------------------------------------------------
// Encoder logic.
//-----------------------------------------------------
assign  lgnt =  {(lgnt3 | lgnt2),(lgnt3 | lgnt1)};

endmodule
```

Circuit simulation has been implemented by means of VCS tool. Figure 4 presents the diagrams of the obtained signals.

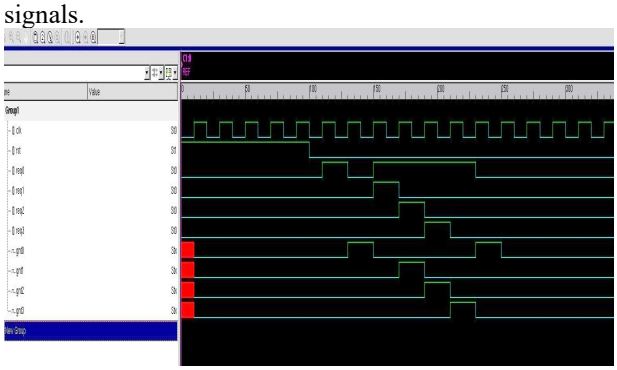


Figure 4: Signals diagram of an arbiter

Statistical static timing analysis of the circuit

The circuit has been synthesized by all contemporary low power design methods. All the circuits have been synthesized by SAED32 nm technology library, which enables the use of all low power design methods. As the

main operating power supply for all circuits, 0,95V was chosen. The same space limit of 150μm^2 has been set for all circuits, which allowed to examine the circuits the same way in terms of the surface. Synchro signal frequency was chosen 50MHz.
By means of statistical static timing analysis method, the total statistical delay of elements from req0 input to gnt0 output has been computed for all circuits.
Below are all the applied methods and obtained circuits, static and dynamic power consumption and the total statistical delay of elements from req0 input to gnt0 output has been computed for all circuits.

Circuit Synthesis by Classical Method
In this case the Design Compiler synthesis tool was given the characterized library for 0,95V.

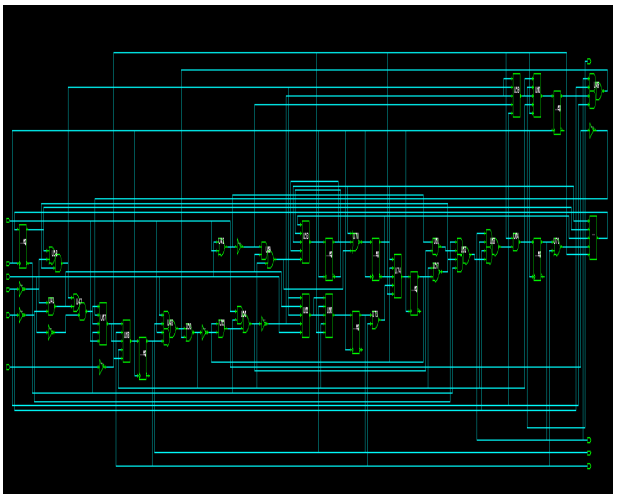


Figure 5: Synthesized circuit of an arbiter by classical method

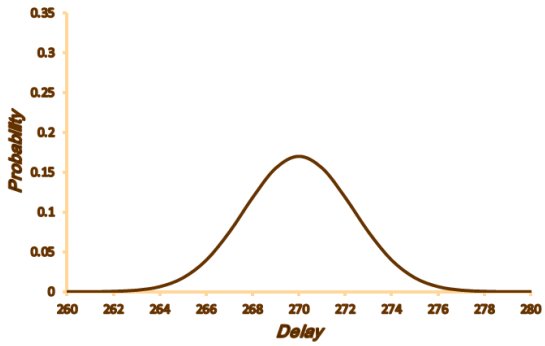


Figure 6: Statistical delay of an arbiter from req0 input to gnt0 output in case of classical method

In this case the circuit consumes 32,3μW power, average value of statistical delay from req0 input to gnt0 output is 270ps, and the standard deviation is 2,345ps.

By scaling of circuit synthesis voltage
In this case, voltage scaling has been implemented, i.e. Design Compiler synthesis tool was given the characterized library for 0,7V.

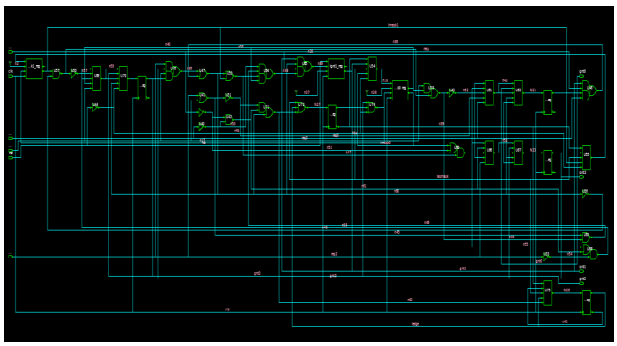


Figure 7: Synthesized circuit of an arbiter by voltage scaling

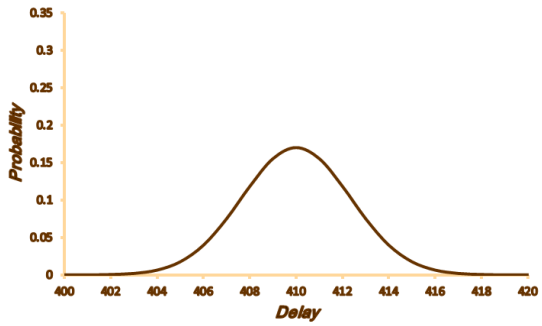


Figure 8: Statistical delay of an arbiter from req0 input to gnt0 output in case of voltage scaling method

In this case the circuit consumes 13.4μW power, average value of statistical delay from req0 input to gnt0 output is 410ps, and the standard deviation is 2.3ps.

The research is supported by SCS MES RA, within the frames of joint Armenian-Belarusian research project No: 13РБ-045.

References

1. Orshansky M., Nassif S. Design for Manufacturability and Statistical Design. - Springer, 2007. - 330 p.
2. Wong B., Mittal A., Cao Y., Starr Greg W. Nano-CMOS Circuit and Physical Design. - Wiley-Interscience, 2004. - 416 p.