Analysis of Circuit Dynamic Behavior with Timed Ternary Decision Diagram

Lu Wan Deming Chen
ECE Department, University of Illinois at Urbana Champaign
{luwan2, dchen}@illinois.edu

ABSTRACT

Modern logic optimization tools tend to optimize circuits in a balanced way so that all primary outputs (POs) have similar delay close to the cycle time. However, certain POs will be exercised more frequently than the rest. Among these critical primary outputs, some may be stabilized very quickly by input vectors, even if their topological delays from primary inputs are very long. Knowing the dynamic behavior of a circuit can help optimize the most commonly activated paths and help engineers understand how resilient a PO is against dynamic environmental variations such as voltage fluctuations. In this paper, we describe a tool to analyze the dynamic behavior of a circuit utilizing probabilistic information. The techniques exploit the use of timed ternary decision diagrams (tTDD) to encode stabilization conditions for POs. To compute probabilities based on a tTDD, we propose false assignment pruning and random variable compaction to preserve probability calculation accuracy. To deal with the scalability issue, this paper proposes a new circuit partitioning heuristic to reduce the inaccuracy introduced by partitioning. Compared to the timed simulation results, our tool has a mean absolute error of 2.5% and a root mean square error of 5.3% on average for ISCAS-85 benchmarks. Compared to a state-of-the-art dynamic behavior analysis tool, our tool is on average 40x faster and can handle circuits that the previous tool cannot.

1. Introduction and Motivation

Traditional circuit design optimizes the static critical paths even when these paths are rarely exercised dynamically. As a result, circuit optimization targets the worst-case conditions to guarantee error-free computation but may also lead to very pessimistic designs. Recently, there are design techniques to achieve higher performance that over-clock the chip to the point where timing errors occur, and then perform error correction either through circuit-level or microarchitecture-level techniques. This approach in general is referred to as *Timing Speculation*.

The idea behind timing speculation and better than worst case (BTW) design is based on the observation that even if two POs have the same static critical path delay, their dynamic behaviors can be very different. For example, in Figure 1, two POs have the same static critical path delay. But one PO (A) has large probability P_s=99% to be stabilized by primary inputs (PI) as early as t. For the other PO (B), the probability of stabilization at time t is only P_s=53%. The difference in stabilization probabilities makes B more *dynamically critical* than A because when the circuit is over-clocked at t, B fails frequently while A may still be able to produce the correct outputs 99% of the times. Knowing such behavior is the key to optimize the circuits for timing speculation.

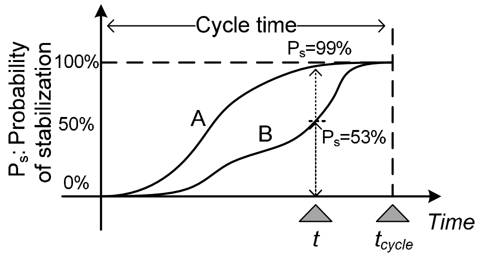


Figure 1. Dynamic behavior curves of primary outputs

Many previous works use Razor logic [1] or other error correcting schemes to enable timing speculation [12][13][16][17][18]. The Blueshift work [13] utilized a commercial design flow to optimize the dynamically critical nodes to achieve higher throughput working with either Razor logic or error-checking processor. In their work, the dynamic behavior is collected through timed simulation, which is very time-consuming. In [17], power-aware slack redistribution was proposed to shift the slack of frequently exercised and near-critical timing paths in a power efficient manner. It requires knowledge of dynamic behavior not only of the whole circuit but also of individual POs, which again was achieved through simulation. To improve microprocessor performance and energy efficiency, Intel's study [12] reduced the timing guard band by using embedded error-detection sequential (EDS) circuits to tolerate dynamic variations. Their work explored path-activation probabilities across various workloads and chose operating points based on the path-delay histogram and path-activation probabilities. DynaTune [16] proposed an analytical approach to compute the dynamic behavior curve of a circuit using a timed characteristic function and BDD. The dynamic behavior is captured in the form of a *behavior curve*, which is similar to the error rate versus clock frequency curve used in [12]. Figure 1 shows an example of behavior curves. A behavior curve is a curve with axis *T* and *P*, where *T* is the operating clock period and *P* is the probability that the circuit (or a primary output – PO) *can* produce correct results within *T*. By varying *T*, one can plot all (*T*, *P*) pairs to get the behavior curve representing the dynamic behavior of a circuit. Guided with this behavior curve, DynaTune optimizes a circuit for higher throughput using dual V_t assignment. The authors of [18] also proposed a BTW synthesis by first characterizing the error probability of a circuit and then used new cost functions considering dynamic behavior of individual gate to do BTW logic decomposition and mapping. Understanding the dynamic behavior of a circuit can help these BTW tools to optimize dynamically critical paths. It can also be used to guide circuit optimization for resilience to environmental variations, such as voltage droop [12], by speeding up dynamically critical POs.

All of the works mentioned above require a mechanism to characterize the dynamic behavior of individual gates, POs, or the whole circuit. Unfortunately, most of the works achieved this through netlist simulation, which is very time consuming. The behavior curves derived by DynaTune give good accuracy but they cannot scale for large circuits because it uses a global BDD to capture the behavior of the entire circuit.

To derive the dynamic behavior curve, we propose: (1) the use of a timed ternary decision diagram (tTDD) to represent stabilization conditions; (2) two tTDD-associated rules – (A) false assignment pruning and (B) random variable compaction – to calculate the dynamic behavior curve of a partitioned sub-circuit; and (3) a novel partitioning heuristic to produce sub-circuits that are suitable for tTDD calculation. To achieve high accuracy during probability calculations, we take care of two types of correlations that can contribute to inaccuracy: (1) a signal's *temporal correlation* and (2) a circuit's *structural correlation*.

The contributions of this work can be summarized as follows:

1) Our algorithm solves the scalability issue of computing the behavior curve through a new partitioning algorithm.
2) For a single partition, when the inputs of the partition are independent of one another, our algorithm can compute the

behavior curve optimally by utilizing false assignment pruning and random variable compaction on a tTDD.

3) Compared to the timed simulation results, our tool has a root-mean-square error of 5.3% and a mean absolute error of 2.5% on average.
4) Compared to DynaTune, we are 40x faster on average.

The remainder of this paper is organized as follows. Section 2 introduces preliminaries. Section 3 introduces encoding stabilization conditions with tTDD. Section 4 shows the overall procedure of computing stabilization probabilities. Section 5 presents two techniques to deal with temporal correlation introduced by tTDD. Section 6 presents a new partitioning heuristic to minimize structural correlation. Section 7 presents experimental results and Section 8 concludes this paper.

2. Preliminaries and definitions

If we observe the switching activities on an output *n* over a large number of clock cycles in the duration of its operation, we can find that in some clock cycles *n* is stabilized very early, and in other clock cycles *n* is stabilized very late. This is because some input vectors applied on the PIs are hard w.r.t. *n*, in that it takes more time to compute. On the other hand, some input vectors are relatively easy w.r.t. *n*. As shown in Figure 1, the cumulative distribution of *n*'s stabilization time can be quantified by consolidating *n*'s scattered activities over its execution life into a single "*probabilistic cycle" –* which captures the probabilistic behavior of the output in a single clock cycle. The probability of *n* being stabilized within a delay t is denoted as $P_s(n,t)$ and is reflected as a point on *n*'s behavior curve with an x-coordinate equal to t.

Given that a dynamic behavior curve can be useful in circuit optimization to achieve the goal of BTW design or dynamic variation resilience, we are interested in deriving the dynamic behavior curve of a circuit quickly without going through time-consuming simulation. *Our problem of interest can be described as: given (1) a circuit; (2) static probabilities of PIs, and (3) a specified time point t, what is the stabilization probability $P_s(n,t)$ that a specified node n can be stabilized within a delay of t after applying input vectors to PIs according to the static probabilities.*

2.1 Definitions

For convenience, we denote **AT** as arrival time, **RAT** as required arrival time, **PI** as primary input, **PO** as primary output, and $\mathbf{T_{clk}}$ as the clock period. In general, we use ***n*** to represent the output of a node in a circuit or a PO. We use the following three definitions to define the term ***stabilization*** in a circuit.

Definition 1: given an output *n* and a timing requirement t, we say *n* is ***stabilized*** no later than t if *n* has taken either logic 0 or logic 1 after a delay of t since applying the input vectors at the rising clock edge, and *n* will not change its value thereafter within the same clock cycle.

We use "*n* is stabilized no later than t" and "*n* is stabilized at t" interchangeably. Furthermore, we can distinguish the stable logic values *n* takes at t with the following two definitions:

Definition 2: *n* is ***stabilized-to-1*** at t if it is stabilized no later than t and takes value of logic 1.

Definition 3: *n* is ***stabilized-to-0*** at t if it is stabilized no later than t and takes value of logic 0.

2.2 Introducing behavior graph

A behavior curve captures a node *n*'s stabilization time over its whole execution life into a single probabilistic cycle with a cumulative distribution function $P_s(n,t)$. To compute $P_s(n,t)$ without simulation, we want to do probabilistic reasoning based on behaviors of *n*' inputs. To achieve this, we introduce an auxiliary random variable $\mathbf{X_n(t)}$ defined within the probabilistic cycle w.r.t node *n* and a temporal term t.

$\mathbf{X_n(t)}$: a random variable used to model the status of *n* when *n* is observed after a delay t since applying the input vectors at the clock rising edge at time 0.

$X_n(t)=\{0,1,U\}$: observe *n* after a delay t, $X_n(t)$ will be in one of the three disjoint **statuses** {0,1,U}:

- Status 0: $\mathbf{X_n(t)=0}$ – *n* is stabilized-to-0 at t
- Status 1: $\mathbf{X_n(t)=1}$ – *n* is stabilized-to-1 at t
- Status U: $\mathbf{X_n(t)=U}$ – *n* is un-stable at t

The timing term t in the brackets will be referred as the ***timing tag*** And the statuses {0,1,U} of $X_n(t)$ will be referred as the ***phase tag***. For example, "$X_n(0.5ns)=1$" means an output *n* is stabilized-to-1 at 0.5ns. '1' is its phase tag and '0.5ns' is its timing tag. Note that if *n* is observed to be stabilized to a logic value at t, then it will not change value thereafter in the same cycle.

Moreover, probabilities can be associated with these observed statuses.

- $\boldsymbol{Pr[X_n(t)=0]}$: *the probability of n* being stabilized-to-0 at t
- $\boldsymbol{Pr[X_n(t)=1]}$: *the probability of n* being stabilized-to-1 at t
- $\boldsymbol{Pr[X_n(t)=U]}$: *the probability of n* being un-stable at t

For example, $Pr[X_{n3}(0.5ns)=1]=36.4\%$ means that if we apply a large number of input vectors at the clock rising edge and then observe *n3* at a delay value of 0.5ns, we will find that *n3* is stabilized-to-1 at 0.5ns with a probability of 36.4%.

From the random variable's definition, we have

Equation 1: $Pr[X_n(t)=0] + Pr[X_n(t)=1] + Pr[X_n(t)=U] = 100\%$

And the stabilization probability $P_s(n,t)$ of interest is:

$$P_s(n,t) = Pr[X_n(t)=\{0,1\}]$$

Meaning that the probability of *n* being stabilized at t is just the probability that *n* is either stabilized-to-0 or stabilized-to-1 at t. And given that the statuses {0,1,U} of $X_n(t)$ are disjoint, $P_s(n,t)$ can be calculated as:

Equation 2: $P_s(n,t) = Pr[X_n(t)=0] + Pr[X_n(t)=1]$

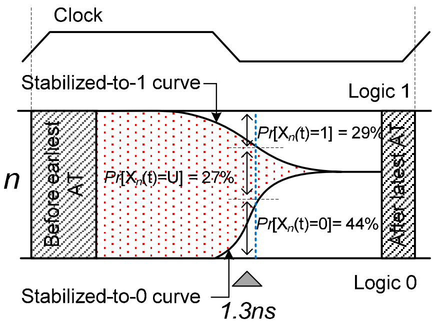


Figure 2: Behavior graph of a node *n*

A pictorial description of probabilities of $X_n(t)$ is shown in Figure 2. We can plot behavior curves for function $S_0(t):=Pr[X_n(t)=0]$ by varying t to get *stabilized-to-0 curve* (lower, curve going up). Similarly, we can plot curves for function $S_1(t):=1-Pr[X_n(t)=1]$ by varying t to get *stabilized-to-1 curve* (upper, curve going down). We call this pair of curves ***behavior graph*** of *n*. For example, if we observe *n* at time t=1.3ns, the probability of *n* being stabilized-to-1 is $Pr[X_n(1.3ns)=1]=29\%$, and the probability of *n* being stabilized-to-0 is $Pr[X_n(1.3ns)=0] =44\%$. With the behavior graph, *n*'s stabilization probability $P_s(n,t)$ can be calculated with Equation 2 as $P_s(n,t)=29\%+44\% =73\%$.

In a behavior graph, two behavior curves of *n* converge as time goes toward *n*'s latest AT. The dotted area between these two curves is *unstable range*. This range represents the probability that *n* has not been stabilized. For example, in Figure 2, $Pr[X_n(1.3ns)=U]=27\%$, meaning that the probability of *n* switching after t is 27%. Note that if the given time point t is less than *n*'s earliest AT, the computation of $P_s(n,t)$ is trivial in that $P_s(n,t)=0\%$ and $Pr[X_n(t)=U]=100\%$. If t is larger than *n*'s latest AT, it is also trivial in that $P_s(n,t)=100\%$ and $Pr[X_n(t)=U])=0\%$.

Assuming we are given behavior graphs associated with a sub-circuit's inputs, we can compute the stabilization probabilities of this sub-circuit's outputs. We will show in the next section how to accomplish this with the help of timed characteristic function and timed inputs.

3. Encode stabilization conditions with tTDD

In this section, we will review the timed characteristic function in section 3.1. In section 3.2, we first introduce the concepts of timed input and timed support set and then present how to use these to encode the stabilization conditions.

3.1 Review of timed characteristic function

The *timed characteristic function* (TCF) is originally used in ATPG to find a test pattern that can sensitize an output at a given time. For convenience, we adopt the *earlier-timed* TCF as proposed in [10]. Similar to most previous work [10][11][16], the TCF is defined in floating mode where input vectors are applied at time t=0, and before time 0, the input vectors are treated as uninitialized.

Definition 4: a ***characteristic function*** *CF(n=val)* is a Boolean function that characterizes the set of input vectors that evaluate an output n to value *val,* where *val* can be logic 0 or 1.

By "characterizes", we mean the CF(n=*val*) evaluates to true for input vectors if and only if these input vectors evaluate n to the desired value *val*. With an additional temporal term t as the RAT of n, we have:

Definition 5: a ***timed characteristic function*** *T(n=val,*t*)* is a Boolean function that characterizes the set of input vectors that stabilizes an output n to value *val no later than time* t.

Given the delay of a cell and the required time t, the TCF of the cell output can be written recursively as TCF functions of its immediate inputs using sensitization criteria [10][11][16].

Take an AND gate "n=AND(a,b)" with cell delay d for example. Given a required time t and the required logic value {0, 1} that n should be stabilized to, the TCFs for n can be written as:

Equation 3: $T(n=1,\text{t}) = T(a=1,\text{t-d}) \wedge T(b=1,\text{t-d})$

Equation 4: $T(n=0,\text{t}) = T(a=0,\text{t-d}) \vee T(b=0,\text{t-d})$

The first formula states that to make n stabilized-to-1 no later than t requires both inputs a and b being stabilized-to-1 at least d time units earlier, 't-d'. The second equation states that n can be stabilized-to-0 no later than t if either input a or b has been stabilized-to-0 no later than (t-d).

Similarly, the TCFs for an OR gate "n=OR(a,b)" are:

Equation 5: $T(n=1,\text{t}) = T(a=1,\text{t-d}) \vee T(b=1,\text{t-d})$

Equation 6: $T(n=0,\text{t}) = T(a=0,\text{t-d}) \wedge T(b=0,\text{t-d})$

The TCFs for an INV gate "n=INV(a)" are:

Equation 7: $T(n=1,\text{t}) = T(a=0,\text{t-d})$

Equation 8: $T(n=0,\text{t}) = T(a=1,\text{t-d})$

The TCFs of complex cells can be written in the same manner by first decomposing complex cells into AND/OR/INV netlist.

3.2 Encode the stabilization conditions

To calculate the stabilization probability, we need a new data structure to represent the stabilization conditions. First, we introduce the concepts of timed input and timed support set, which are used in our new data structure.

A ***timed input*** v_i(t) of n is an ordinary input v_i coupled with a temporal term t reflecting the timing relation between the input v_i and the output n. The ***support set*** of n, denoted as ***Sup***(n) is the set of the inputs to the circuit rooted at n. Furthermore, n's ***timed support set***, denoted as ***tSup***(n,t), is the set of all timed inputs {v_i(t)} that n's status at time t depends on. The *tSup*(n,t) is determined by back-tracing the fanin-cone of n.

Take circuit in Figure 3 for example. Assume we are working on a sub-circuit rooted at an output n after the whole circuit has been partitioned. We have *Sup*(n)={a,b}. Assume we are interested in n's dynamic behavior at time t. We know that n's status at t is influenced by paths: '$a \rightarrow p \rightarrow n$'(with path delay 3), '$b \rightarrow p \rightarrow n$'(with path delay 3) and '$b \rightarrow n$'(with path delay 1). In other words, n's status at time t is totally determined by a's status at t-3, b's status at t-3 and b's status at t-1. Therefore, *tSup*(n,t)={a(t-3),b(t-1),b(t-3)}. Note that the internal structure of the circuit makes b in *Sup*(n) become multiple correlated timed inputs {b(t-1),b(t-3)} in *tSup*(n,t).

For clarity, an unrealistic delay model is used in this example by assuming each cell has a fixed delay regardless of input pin and fanout load. However, in our experiments, we use a more realistic pin-to-pin delay model by distinguishing: (1) each pin-to-output delay, (2) rise and fall delay, (3) cell's driving strength, and (4) cell's fanout load.

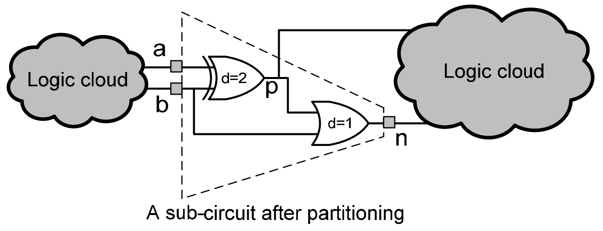


Figure 3: A sub-circuit example

3.2.1 Stabilization conditions for n

TCF enables us to understand n's stabilization conditions at time t by enumerating all sensitization conditions. This enumeration is done by recursive TCF rewriting [10][11]. For example, if we are interested in what makes n in Figure 3 stabilized-to-1 at t, then we can recursively write n's TCF T(n=1,t) through back-tracing n's fanin-cone:

$$T(n=1,\text{t}) = T(p=1,\text{t-1}) \vee T(b=1,\text{t-1})$$
$$= (T(a=1,\text{t-3}) \wedge T(b=0,\text{t-3})) \vee (T(a=0,\text{t-3}) \wedge T(b=1,\text{t-3}) \vee T(b=1,\text{t-1})$$
$$= (a(\text{t-3}) \wedge b'(\text{t-3})) \vee (a'(\text{t-3}) \wedge b(\text{t-3})) \vee b(\text{t-1})$$

The first and the second derivative steps are by OR and XOR sensitization criteria, respectively. The last derivation is based on the fact that the TCF of a circuit input is just the timed input itself.

Furthermore, we can evaluate the temporal term t in a TCF to any desired value. For example, if we are interested in output n's dynamic behavior at 5ns after the rising clock edge, then we can substitute t with 5ns to get an ***evaluated TCF***:

$$T(n=1,5\text{ns})=(a(2\text{ns}) \wedge b'(2\text{ns})) \vee (a'(2\text{ns}) \wedge b(2\text{ns})) \vee b(4\text{ns}).$$

The evaluated TCF compresses information of all feasible input conditions that can stabilize n to logic 1 at 5ns into a single formula. For example, the above formula states that n can be stabilized-to-1 by 5ns if any of the following encoded stabilization conditions is satisfied:

(1) a(2ns)^b'(2ns): a is stabilized-to-1 by 2ns *and* b is stabilized-to-0 by 2ns;
(2) a'(2ns)^b(2ns): a is stabilized-to-0 by 2ns *and* b is stabilized-to-1 by 2ns;
(3) b(4ns): b is stabilized-to-1 at 4ns.

Next, with the stabilization conditions we can build a decision diagram to facilitate the computation of the probability that n can be stabilized-to-1 at t.

3.2.2 tTDD for stabilization conditions

In [16], the original recursive TCF construction procedure is transformed into a different form, in which the CF for each node in a circuit is first constructed as a BDD. Then the sensitization criteria is used to refine the CF. This transformation has the benefit that the timing information is decoupled from the variables in the BDD during construction so that an ordinary BDD can be used for probability computation. However, this necessitates the construction of a *global BDD* using circuit PIs as variables, which may cause a scalability issue. In addition, BDD cannot deal with unstable ranges of the inputs of a subcircuit due to circuit partitioning. To solve this, we need a new data structure that can work on partitioned sub-circuits and can handle unstable ranges.

We first show why BDD cannot handle the unstable range in detail. From the derived TCF, if we construct a ***timed BDD*** (tBDD) from n's timed support set $tSup(n,\mathrm{t})$ by just regarding each unique timed input in it as an ordinary variable and then construct an ordinary BDD from these "ordinary variables" then it cannot handle the U status correctly. This is due to the reason that different from an ordinary BDD the variables used in a tBDD are timed inputs, thus have temporal terms associated with them.

As an example, the tBDD of n's timed characteristic function T(n=1,5ns) is shown in Figure 4(a). Each node V in the tBDD is associated with a timed input v_i(t) marked on the left side. The 1-edge (solid line) branching out of node V represents a stabilization status of the input v_i, e.g. the input v_i is stabilized-to-1 at t. Using auxiliary random variable introduced in section 2.2, this status is denoted as X_{vi}(t)=1. Similarly, the 0-edge (dashed line) of V represents X_{vi}(t)=0.

As mentioned before, a big problem of this tBDD model is that the 'U' status defined in section 2.2 of the input is not represented. For example, from the circuit structure in Figure 3, we know that even if *a is not stabilized at all (the 'U' status), n* can still be stabilized-to-1 as long as input b has been stabilized to logic 1. This stabilization condition is not correctly modeled in the tBDD in Figure 4(a).

The root cause of tBDD's incapability of correctly modeling the 'U' status is that a BDD node has exactly two outgoing edges to explicitly model only two possible assignment {0,1} for a random variable X_v(t). This ignores the effect of the 'U' value of X_v(t) in Equation 1. In fact, with the existence of the unstable range in the behavior graph the stabilization status of each timed input v_i(t) needs to be modeled as a ***three-value*** system. Hence, we introduce ***timed ternary decision diagram*** (tTDD) as a solution.

A ternary decision diagram (TDD) is similar to BDD with the difference that each node has three possible outgoing branches [14]. Thanks to the TDD's similarity to BDD, a tTDD can be built in an almost identical way as building a tBDD. The tTDD for the example circuit in Figure 3 is shown in Figure 4(d). In this tTDD, the temporal term t is kept as a variable without being evaluated. Each node in the tTDD is associated with a timed input v(t) marked on the left side. For example, V0 is associated with timed input a(t-3).

Given that a tTDD is a three-value system, the stabilization condition of a certain input being unstabilized can now be explicitly modeled with the introduction of the 'U' edge. For example, the problematic stabilization condition for the above tBDD can now be modeled as the path: "V0→V3→T" with a 'U' edge connecting V0 to V3, as shown in Figure 4(d).

With this enhanced tTDD model of the stabilization conditions, we can now calculate the probability $Pr[X_n(\mathrm{t})=1]$ by collecting the probabilities of all stabilization conditions. Note that the probability of n having been stabilized-to-0 at time t can be calculated in the same manner starting with constructing a TCF of n with required stabilization value 0, that is $T(n=0,\mathrm{t})$. We will refer the tTDD of n stabilized-t-0/1 as ***tTDD-0/1***.

Finally, according to Equation 2, the stabilization probability $P_s(n,t)$ is simply the sum of these two probabilities calculated on tTDD-0 and tTDD-1. Therefore, the behavior graph of n can be plotted by evaluating the variable t at different timing points.

4. Stabilization probability calculation

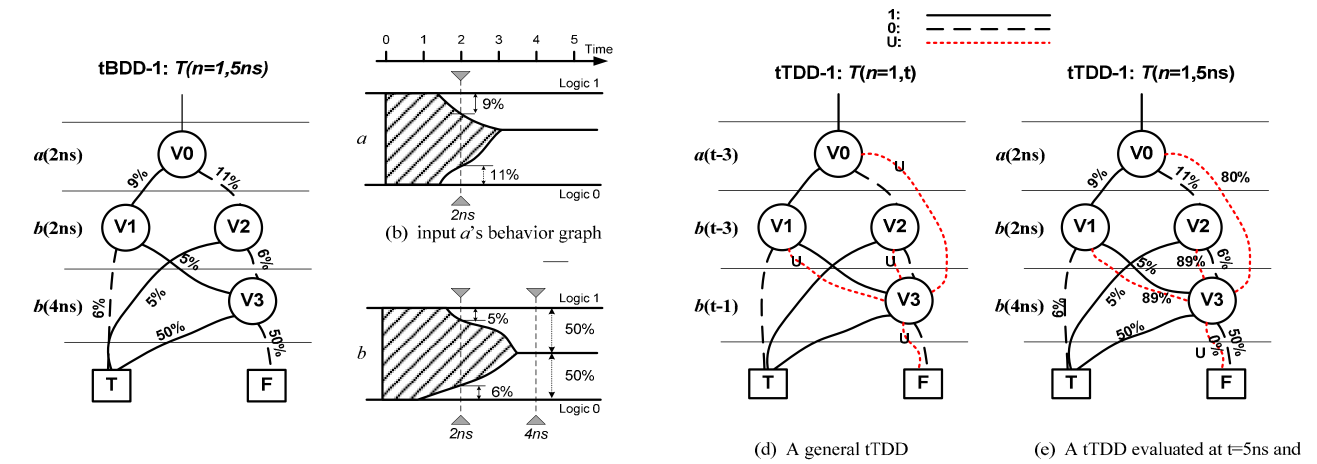
To find a stabilization condition that makes n stabilized-to-1 at a given time t, is equivalent to solving a three-value satisfying assignment problem: we need to find a feasible combination of stabilization status for the timed support set that can evaluate the right hand side (RHS) of the TCF formula to true.

The possible statuses a timed input can take are: {stabilized-to-0, stabilized-to-1, un-stabilized}. This exactly corresponds to the three statuses of our auxiliary random variable X_v(t), as defined in section 2.2. The physical meaning between the tTDD and the auxiliary random variable X_v(t) can be viewed as follows: each tTDD node is associated with a timed input v(t) and has three out-going edges {0,1,U} to reflect the three possible stabilization statuses of its associated input v at time t. The 0-edge taken from this node is equivalent to have input v at 'X_v(t)=0' status, meaning v is stabilized-to-0 at t. Similarly, 1-edge corresponds to 'X_v(t)=1' and U-edge corresponds to 'X_v(t)=U'.

From the TCF, we know that the stabilization status of a circuit output depends on the stabilization statuses of the circuit's inputs. Let us assume v is an input of a circuit rooted at n. Assume that due to existence of multiple timing paths from v to n, v becomes two timed inputs $v(\mathrm{t}_1)$ and $v(\mathrm{t}_2)$ in n's tTDD. As a result, n's stabilization status at a clock cycle c depends on stabilization statues of v at both time t_1 and t_2 in the same clock cycle. We need to point out in this clock cycle c, the stabilization status of $v(\mathrm{t}_1)$ and $v(\mathrm{t}_2)$ are highly correlated. Their temporal correlations are stated as follows:

Lemma 1: $Pr[X_v(\mathrm{t}_1)=1 \cap X_v(\mathrm{t}_2)=Val_2]=0$, if $\mathrm{t}_2 \geq \mathrm{t}_1$ and $Val_2 \in \{0,\mathrm{U}\}$.
$Pr[X_v(\mathrm{t}_1)=0 \cap X_v(\mathrm{t}_2)=Val_2]=0$, if $\mathrm{t}_2 \geq \mathrm{t}_1$ and $Val_2 \in \{1,\mathrm{U}\}$.

Proof: This is in fact the direct result of the definitions of stabilization in section 2.1. If we observe that v has been stabilized to a specific value at time t_1, '$X_v(\mathrm{t}_1)$=1/0', then: (A) v will not switch to a different value in the same cycle c at a later time t_2, '$X_v(\mathrm{t}_2)$=0/1'. (B) It is also impossible to observe that v is still unstable, '$X_v(\mathrm{t}_2)$=U', at a later time $\mathrm{t}_2 \geq \mathrm{t}_1$ in the same cycle c. Note that this lemma holds even for $\mathrm{t}_1=\mathrm{t}_2$ because of the fact that {0,1,U} are inherently disjoint statuses of X_v(t). □



(a) output n's timed BDD

(b) input a's behavior graph

(c) input b's behavior graph

(d) A general tTDD Representing the sensitization conditions for an output n at time t

(e) A tTDD evaluated at t=5ns and with stabilization probabilities annotated

Figure 4: Encoding stabilization conditions with tTDD and probability calculation on tTDD

Using a decision diagram to represent a TCF facilitates probability calculation, in that a satisfying assignment of the TCF is equivalent to a connected path from the root node to the true terminal. In the context of stabilization analysis, each stabilization condition simply corresponds to a path from the tTDD's root node to the true terminal. We call such paths *satisfying paths of tTDD*.

Definition 6: a ***satisfying path* Ψ_j** in a tTDD is a path from root node to the true terminal, where *j* is the path ID. On a satisfying path Ψ_j, the branch taken on each node is its ***satisfying assignment*** of stabilization statues, e.g. {0,1,U}, for the corresponding timed input.

For example, for the circuit in Figure 3, we know to stabilize *n* to 1 at t=5ns one stabilization condition is '*a* is stabilized-to-1 at t=2ns' and '*b* is stabilized-to-0 at t=2ns'. This is represented as a satisfying path "V0→V1→T" in Figure 4(e) after we evaluate the tTDD-1 in Figure 4(d) with t=5ns. This can also be represented equivalently by auxiliary random variables as '$X_a(2ns)=1 \cap X_b(2ns)=0$' (∩ indicates the joint probability of the two random variables). The associated satisfying assignment is: $[X_a(2ns), X_b(2ns)]=[1,0]$ because one 1-edge and one 0-edge are taken. 'V0→V3→T' is another satisfying path with its satisfying assignment: $[X_a(2ns), X_b(4ns)]=[U,1]$. We need to point out that the satisfying paths only exist in a tTDD to encode stabilization conditions and do not reflect any physical paths in the circuit.

The probability of one satisfying path Ψ_j is the jointed probability of satisfying assignment associated with Ψ_j.

Equation 9: $$Pr[\Psi_j]=Pr[\bigcap_{i=0}^{L}(X_{vi}(t_i)=Val_i)]$$

Where *L* is the length of the path Ψ_j and $(X_{vi}(t_i)=Val_i)$ is the assignment to each random variable along this satisfying path.

With all stabilization-to-1 conditions being encoded in the tTDD-1 data structure, the probability ($Pr[X_n(t)=1]$) of stabilizing *n* to 1 no later than t is just the collection of probabilities of all possible stabilization conditions:

Equation 10: $$Pr[X_n(t)=1]=Pr[\bigcup_{j=0}^{S}\Psi_j]$$

Where Ψ_j is a satisfying path in *n*'s tTDD-1, *S* is the total number of satisfying paths.

Next, we show that the above probability can simply be computed as the sum of probability of every distinct satisfying path.

Theorem 1: $$Pr[\bigcup_{j=0}^{S}\Psi_j]=\sum_{j=0}^{S}Pr[\Psi_j]$$

Proof: we prove this by showing that any two satisfying paths in the tTDD are probabilistically disjointed.

In a tTDD, for any two different path Ψ_i and Ψ_j, i≠j, they must have at least one node shared. Otherwise, if they do not share nodes at all, then the tTDD must have at least two root nodes, which contradicts the fact that a tTDD has only one root node. Secondly, among all their shared nodes there must be at least one shared node, from which Ψ_i and Ψ_j take different branching edges. Otherwise, if Ψ_i and Ψ_j take the same branching edge for every shared node, then it is not difficult to see that this will lead to the conclusion that Ψ_i and Ψ_j are identical, which contradicts our assumption that Ψ_i and Ψ_j are different paths.

Let's denote a particular shared node in the tTDD as *G*, and denote the timed input associated with *G* as *g*(*t*). From node *G*, Ψ_i and Ψ_j take two different branching edges Val_1 and Val_2, respectively. For Ψ_i, the satisfying assignment at node *G* is $X_g(t)=Val_1$; while for Ψ_j the satisfying assignment at node *G* is $X_g(t)=Val_2$.

The joint probability of these two satisfying paths is:

$$Pr[\Psi_i\cap\Psi_j]=Pr[[\bigcap_{i=0}^{Li}(X_{vi}(t_i)=Val_i)]\cap[\bigcap_{j=0}^{Lj}(X_{vj}(t_j)=Val_j)]]$$

We can write it conditionally on the stabilization status of *G*:

$$=Pr[[X_g(t)=Val_1]\cap[X_g(t)=Val_2]]*$$

$$Pr[\{\bigcap_{i=0;vi\neq g}^{Li}(X_{vi}(t_i)=Val_i))\cap(\bigcap_{j=0;vj\neq g}^{Lj}(X_{vj}(t_j)=Val_j)\}\,|$$
$$\{(X_g(t)=Val_1)\cap(X_g(t)=Val_2)\}]$$

Since Val_1 and Val_2 are different statues, one of them must not be 'U' status. Then from Lemma 1, we know:

$$Pr[[X_g(t)=Val_1]\cap[X_g(t)=Val_2]]=0$$

Then the joint probability of two distinct satisfying paths is zero. □

5. Probability of one satisfying path

In this section, we show how to compute the probability of an individual satisfying path. This is done with two steps: (1) evaluate tTDD at a given t and annotate the evaluated tTDD with probabilities, (2) calculate the probability honoring the temporal correlation due to the use of timed inputs.

5.1 Annotate a tTDD with probabilities

Before we start to calculate the probability of a satisfying path, we need to (1) evaluate the tTDD at a specified t to get an evaluated tTDD, and then (2) annotate the probabilities onto the edges of this evaluated tTDD by looking up the corresponding probabilities from the inputs' behavior graphs.

In the example of Figure 3, we are interested in *n*'s dynamic behavior at t=5ns. Then we evaluate the tTDD at t=5ns by substituting all t terms with 5ns.

Because we model the stabilization status of each timed input *v*(t) with random variable $X_v(t)$, and random variable has a value and an associated probability, we need to ***annotate probabilities*** onto the tTDD after it is evaluated. This is done by assigning the corresponding probabilities onto each node's outgoing edges as edge weights. The corresponding probabilities can be looked up from the inputs' behavior graphs.

For example, assume we have the behavior graphs for inputs {*a*,*b*} pre-computed as shown in Figure 4(b) and (c). We first extract the probabilities for *a* being stabilizaed-to-0/1 at time point 2ns from *a*'s behavior graph. Similarly, probabilities for *b* at time points 2ns and 4ns are extracted from *b*'s behavior graph. We then annotate the extracted probabilities onto the evaluated tTDD as the edge weights. The result after annotation is shown in Figure 4(e).

With the probability-annotated tTDD, we can calculate the probability of each satisfying path. To calculate it accurately, we have to take care of the temporal correlations along the satisfying path. This is discussed next.

5.2 Solving the temporal correlation problem

Temporal correlation is introduced by those inputs that have multiple timing paths leading to the output *n*. To motivate why we need to treat it carefully, let us first take a look of the example circuit in Figure 3. Its annotated tTDD-1 is shown in Figure 4(e). One satisfying path is along 'V0→V2→V3→T', with V2 to V3 through the 0-edge. The corresponding satisfying assignment to $[X_a(2ns), X_b(2ns), X_b(4ns)]$ is [0,0,1]. However, careful analysis shows that this is in fact an impossible assignment. This is because after we have '$X_b(2ns)=0$', meaning *b* has been stabilized-to-0 at 2ns, we cannot have *b* flip again and be stabilized-to-1 at t=4ns in the same clock cycle, that is '$X_b(4ns)=1$'. We call such a physically impossible satisfying assignment a ***false assignment.*** To avoid the potential miscalculation of probability due to the existence of false assignments, we introduce the technique ***false assignment pruning***, which eliminates the false assignments in a tTDD from probability calculation.

5.2.1 False assignment pruning

Given a satisfying assignment along the satisfying path Ψ_j contained in the tTDD, denoted as $[X_{v1}(t_1), X_{v2}(t_2), \dots X_{vm}(t_m)]=[Val_1, Val_2, \dots Val_m]$, we want to decide whether it is a false assignment. Since the temporal correlation is due to the existence of multiple timing paths to the output from the same input, we can first group random variables according to inputs, that correspond to the subscripts of $X_{v_i}(t_i)$'s. Grouping $X_{v_i}(t_i)$'s according to their subscripts (i.e., the same input) is called ***grouping*** of random variables. We apply the grouping operation to have random variables corresponding to the same input grouped together into an ***input status group***. Such a group reflects the input stabilization status required for the same input. Recall the concepts of phase tag and timing tag in section 2.2. We prune false assignment by analyzing the phase and timing tag relations among $X_{v_i}(t_i)$'s in each input status group. We have the following theorem.

Theorem 2: Within an input status group, if any of the following conflicting conditions are detected, the entire satisfying assignment is declared as a false assignment.

1. $X_{vi}(t_1)=1 \cap X_{vi}(t_2)=0$;
2. $X_{vi}(t_1)=1 \cap X_{vi}(t_2)=U$, with $t_1 \leq t_2$;
3. $X_{vi}(t_1)=0 \cap X_{vi}(t_2)=U$, with $t_1 \leq t_2$;

Proof: Without loss of generality, we assume $t_1 \leq t2$ for condition 1. For $t1>t2$, we can just reverse t_1 and t_2. Then the above three conflicting conditions can only occur with a probability of zero as stated in Lemma 1:

1. $Pr[X_{vi}(t_1)=1 \cap X_{vi}(t_2)=0] = 0$; if $t_1 \leq t_2$;
2. $Pr[X_{vi}(t_1)=1 \cap X_{vi}(t_2)=U] = 0$, if $t_1 \leq t_2$;
3. $Pr[X_{vi}(t_1)=0 \cap X_{vi}(t_2)=U] = 0$, if $t_1 \leq t_2$;

Therefore, if any of the above conflicting conditions are observed in an input status group, the corresponding assignment results in zero probability. □

Consider the example false assignment mentioned in the beginning of this sub-section. The grouping operation results in two input status groups: a group associated with *a*, that is '$[X_a(2ns)]=[0]$' and a group associated with *b*, that is '$[X_b(2ns), X_b(4ns)]=[0,1]$'. Checking conflicting rules for each group, we find group $[X_b(2ns), X_b(4ns)]$ hits the first conflicting condition. Then the whole assignment is declared as a false assignment and is excluded.

After pruning away false assignments, we can calculate the probabilities of the remaining true satisfying assignments.

5.2.2 Random variable compaction in a group

After the false paths are pruned, we need to calculate the probability $Pr[\Psi_j]$ of each remaining satisfying paths and then sum them up. This section provides a way to calculate an individual $Pr[\Psi_j]$. The overall procedure is as follows. We first apply grouping. We then work internally in each input status group and calculate the probability of each such group with a technique called ***random variable compaction*** consisting of two steps: unifying the timing tags and unifying the phase tags. Then we work externally on all groups in Ψ_j to calculate $Pr[\Psi_j]$. We introduce details next.

Within each group, there can be multiple random variables with different timing tags and phase tags. To calculate the probabilities of each input status group, we need first to unify the timing tags by applying the following rules:

(1) ***Unify the timing tags*** in a group. For each phase tag, if there are multiple random variables with different timing tags in the group, we compact them into a single random variable with a unified timing tag according to Theorem 3.

Theorem 3:

1. $Pr[X_{vi}(t_1)=1 \cap X_{vi}(t_2)=1 \ldots \cap X_{vi}(t_m)=1] = Pr[X_{vi}(\min(t_1,t_2 \ldots t_m))=1]$
2. $Pr[X_{vi}(t_1)=0 \cap X_{vi}(t_2)=0 \ldots \cap X_{vi}(t_m)=0] = Pr[X_{vi}(\min(t_1,t_2 \ldots t_m))=0]$
3. $Pr[X_{vi}(t_1)=U \cap X_{vi}(t_2)=U \ldots \cap X_{vi}(t_m)=U] = Pr[X_{vi}(\max(t_1,t_2 \ldots t_m))=U]$

Proof: We prove 3.1. The other two can be proven in a similar way. Without loss of generality, we assume the timing tags are ordered s.t. $t_1 \leq t_2 \leq \ldots \leq t_m$. We can write it conditionally as:

$Pr[X_{vi}(t_1)=1 \cap X_{vi}(t_2)=1 \ldots \cap X_{vi}(t_m)=1]$
$= Pr[X_{vi}(t_2)=1 \ldots \cap X_{vi}(t_m)=1 | X_{vi}(t_1)=1] * Pr[X_{vi}(t_1)=1]$
$= 1 * Pr[X_{vi}(t_1)=1]$;

The last step is by definition: if v_i is stabilized-to-1 at t_1, then it will not switch and is still stabilized-to-1 later at $t_2, t_3, \ldots t_m$ in the same clock cycle. □

Note that after this step, it is guaranteed that for any phase tag, there is at most one random variable associated with it in a group.

For example in Figure 4(e), one satisfying path is 'V0→V1→V3→T', with V1 going to V3 through the 1-edge. The corresponding satisfying assignment is: '$[X_a(2ns),X_b(2ns),X_b(4ns)]=[1,1,1]$'. After grouping, input *b*'s status group is '$[X_b(2ns),X_b(4ns)]=[1,1]$'. According to Theorem 3.1:

$Pr[X_b(2ns)=1 \cap X_b(4ns)=1]$
$=Pr[X_b(\min(2ns,4ns))=1]=Pr[X_b(2ns)=1]= 5\%$.

(2) ***Unify the phase tags*** in a group. If any group still has more than one random variable, then we compact the random variables with different phase tags into a closed form according to Theorem 4.

Theorem 4:

1. $Pr[X_{vi}(t_1)=U \cap X_{vi}(t_2)=1] = Pr[X_{vi}(t_2)=1] - Pr[X_{vi}(t_1)=1]$, if $t_1<t_2$
2. $Pr[X_{vi}(t_1)=U \cap X_{vi}(t_2)=0] = Pr[X_{vi}(t_2)=0] - Pr[X_{vi}(t_1)=0]$, if $t_1<t_2$

Proof: We prove 4.1. 4.2 can be proven in a similar way.

For $t_1<t_2$, we can have:

$Pr[X_{vi}(t_2)=1 \cap X_{vi}(t_1)=U]$
$= Pr[X_{vi}(t_2)=1] - Pr[X_{vi}(t_2)=1 \cap X_{vi}(t_1)=0] - Pr[X_{vi}(t_2)=1 \cap X_{vi}(t_1)=1]$
$= Pr[X_{vi}(t_2)=1] - 0 - Pr[X_{vi}(t_2)=1 \cap X_{vi}(t_1)=1]$ (Lemma 1)
$= Pr[X_{vi}(t_2)=1] - Pr[X_{vi}(t_1)=1]$; (Theorem 3.1) □

For example in Figure 4(e), one satisfying path is 'V0→V2→V3→T', with V2 to V3 through the U-edge. The corresponding satisfying assignment is: '$[X_a(2ns),X_b(2ns),X_b(4ns)]=[0,U,1]$'. After grouping, input *b*'s status group is '$[X_b(2ns),X_b(4ns)]=[U,1]$'. According to Theorem 4.1:

$Pr[X_b(2ns)=U \cap X_b(4ns)=1]$
$=Pr[X_b(4ns)=1]-Pr[X_b(2ns)=1]=50\%-5\%=45\%$

5.3 Put everything together to get $Pr[\Psi_i]$

We can verify that theorems 2, 3 and 4 capture all possible combinations of the random variables that can exist along a satisfying path. After grouping, Theorem 3 unifies different timing tags for each phase tag in each group. After this, a group has no more than three random variables with each having a distinct phase tag. If only one random variable exists in a group, no special processing is needed. If two random variables co-exist in a group, all their possible phase and timing combinations are covered and processed according to row 3-5 of Table 1. If all three phases still co-exist in a group, row 6 of Table 1 is applied to prune away this false assignment (based on Theorem 2.1). This verifies that all possible combinations are covered by corresponding theorems.

After random variable compaction, the probability of each group is represented in a closed form and can be evaluated to a real number. The joint probability of different groups can then be expressed as the product of the probability of each individual group, similar to most previous work [1][2][3][4][5][6]. To maintain good accuracy, the inputs associated with each group need to be kept as independent as possible. This is accomplished by our circuit partitioning algorithm.

Table 1: Possible combinations of random variables in an input status group

Phases			Timing		
0	1	U	t1<t2	t1=t2	t1>t2
—	●	●	Theorem 2.2	Theorem 2.2	Theorem 4.1
●	—	●	Theorem 2.3	Theorem 2.3	Theorem 4.2
●	●	—	Theorem 2.1	Theorem 2.1	Theorem 2.1
●	●	●	Theorem 2.1	Theorem 2.1	Theorem 2.1

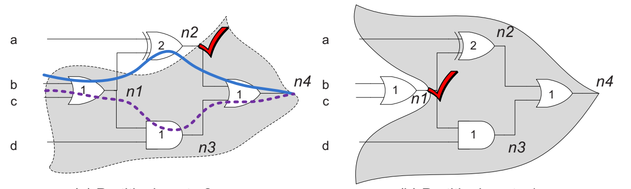
6. Circuit Partitioning

The goal of circuit partitioning is to deal with the circuit scalability issue and enable the tTDD-based probability calculation to handle large circuits in reasonable runtime. Given that the worst case complexity of a TDD is of O(3^n/n) [14], we desire to limit the size of the TDD constructed from the partitioned sub-circuit. Meanwhile, the partitioning algorithm should also produce sub-circuits that are minimally affected by the structural correlation introduced by reconvergent nets in the circuit.

A number of switching power estimation and testability analysis works have attempted to address the structural correlation issue introduced by reconvergent nets during signal probability calculation. Proposed methods to preserve signal dependency during signal probability computation include the use of Bayesian networks [7], conditional independence [8], and Boolean approximation [9]. However, these are all limited to a zero-delay model. Other proposed approaches to reduce the signal correlation include limited depth reconvergent path analysis [4] and a method to find the independent terms in the probability polynomial at super-gate by leveraging the concept of graph dominator [5][6]. Unfortunately, these methods are only applicable when the signal probability is represented by polynomials. When a BDD is used to compute the signal probability, a partitioning heuristic was proposed [15] to minimize the number of nodes shared between the partitioned sub-circuit and the remaining circuit. We denote this method as Kapoor's heuristic.

6.1 Partitioning preliminary

Kapoor's heuristic minimizes the structural correlation of the partition's input as well as controls the size of a BDD by enforcing that the size of each partition has no more than K inputs. Given that we introduced timed inputs, we extend Kapoor's heuristic by controlling the number of timed inputs. In our new heuristic, we also take care of the effect observed in [4] that the structural correlation between two nodes reduces as they move away from their common source node.



(a) Partitioning at n2 (b) Partitioning at n1

Figure 5: Partitioning example of a small circuit

Definition 7: A ***K-feasible fanin cone*** of a node *n* is a sub-circuit rooted at *n* with a timed support set of cardinality no more than K.

In other words, a K-feasible fanin cone of *n* has at most K unique timing paths from the inputs to the output *n*. As an example, we know the fanin cone of *n4* in Figure 5(a) before partitioning is 6-feasible but not 5-feasible: *tSup*(*n4,t*)={*a*(t-3),*b*(t-4),*c*(t-4),*b*(t-3),*c*(t-3),*d*(t-2)} with a cardinality of 6. For a non-K-feasible fanin cone, cut points need to be chosen to make it K-feasible.

Definition 8: An internal node *m* in the fanin cone rooted at *n* ***dominates*** a timed input *v*(t) in *n*'s timed support set if all timing paths associated with *v*(t) go through *m* to reach *n*.

Definition 9: The ***dominator factor*** *D*(*m*) of an internal node *m* w.r.t the output *n* is the total number of timed inputs that are dominated by *m*.

These definitions are directly extended from the concept of a graph dominator. In Figure 5(a) w.r.t the output *n4*, *n1* dominates {*b*(t-4),*c*(t-4),*b*(t-3),*c*(t-3)}; therefore, *D*(*n1*)=4. And *n2* dominates {*a*(t-3),*b*(t-4),*c*(t-4)}. Thus, *D*(*n2*)=3.

Definition 10: a timed input *v*(t) ***leaks*** from *m* in the fanin cone rooted at *n* if *m* dominates some *v*(t') with t'≠t but does not dominate *v*(t).

In Figure 5(a), timed variable *b*(t-4) is dominated by *n2* because the only timing path associated with it goes through *n2*, as shown with a solid path from *b* to *n4*. Similarly, *c*(t-4) is also dominated by *n2*. On the other hand, {*b*(t-3),*c*(t-3)} leak from *n2* because the timing paths associated with them do not go through *n2*, shown with the dashed path.

Definition 11: The ***leaking factor*** *L*(m) of an internal node *m* w.r.t. the output *n* is the total number of timed inputs that leak from *m*.

For example, no timed input leaks from *n1* and two leaks from *n2*. So *L*(*n1*)=0 and *L*(*n2*)=2.

6.2 Partitioning cost function

Kapoor's heuristic has a limitation that may affect the quality of the partitioning. In that, if a node is not K-feasible, Kapoor's heuristic limits the searching of cut point within the node's immediate fanins. For *n4* in our example, only *n2* and *n3* will be investigated. And *n2* is cut to form a new partition as marked with the check sign in Figure 5(a), resulting in a 5-feasible fanin cone of *n* with *tSup*(*n4*,t)={*n2*(t-1),*b*(t-3),*c*(t-3),*d*(t-2)}. Note that *b,c* are still correlated with *n2* because they are transitive fanins of *n2*. This structural correlation may introduce inaccuracy in the tTDD probability calculation.

Our partitioning algorithm extends the searching for potential cut points to all nodes in *n*'s current non-K-feasible fanin cone and assigns a cost to each potential cut point *m* according to:

$$Cost(m) = \frac{L(m)+1}{D(m)*R(m)^{\lambda}}$$

Where *L*(m) and *D*(m) are the leaking factor and dominating factor of node *m*, respectively, and *R*(m) is the max distance from *m* to *n* in term of logic levels. The parameter λ is used to control the effect of *R*(m). The node with the lowest cost is chosen to form a new fanin cone of *n*. This procedure repeats until *n* is K-feasible.

The intuition behind our cost function includes: (1) The numerator in the cost function favors those nodes that have less timed inputs leaking from *m*. Because we do not want those leaked inputs to introduce new structural correlation for their descendent nodes. (2) The denominator *D*(m) biases our choice toward those nodes that dominate more timed inputs, as suggested in [5][6]. (3) At the same time, we want to keep the cut point far away from *n* with the introduction of *R*(*m*). If we can have the cut point far away from the output, then the new correlation that can potentially be introduced by the cut point will diminish more as the distance from *m* to *n* becomes longer, as suggested in [4].

For example, for λ=0.5 our cost function will compute as: cost(*n1*)= 1/(4*2^0.5)=0.18, cost(*n2*)=3/(3*1^0.5)=1 and cost(*n3*)=1. So *n1* is chosen as the partitioning point, as shown with the check mark in Figure 5(b). The new 5-feasible timed support set of *n4* is *tSup*(*n4*,t)={*a*(t-3),*n1*(t-3),*n1*(t-2),*d*(t-2)}. This is actually the optimal partition to minimize structural correlation due to reconvergent net from *n1*. Note that the two temporally correlated terms, *n1*(t-3) and *n1*(t-2) introduced by cutting at *n1*, will be taken care of by theorems in section 5.2.

6.3 Overall partitioning algorithm

The overall partitioning is performed as follows: first a PI's timed support set is initialized to itself. Then the nodes are processed from PI to PO in a topological order. During processing, a node *n* will first inherit all timed support sets from its immediate fanin nodes. If the fanin cone of *n* is not K-feasible, partitioning points are chosen as described in section 6.2 until the fanin cone becomes K-feasible. Since the complexity of computing the cost function is bounded by a specified constant K, the overall partitioning algorithm runs very fast as shown in our experimental results. We set K to 6.

7. Experimental results

The proposed Partition+tTDD algorithm (*tTDD-P*) is implemented in C on top of SIS[20]. We use ISCAS'85 benchmarks, which are widely used to study the performance of BDD-based applications, to test the accuracy and performance of our algorithm. These benchmarks are first compiled into Verilog netlists using Synopsys *Design Compiler* (DC) ver.2007-SP3 with a subset of TSMC65nm cells that are supported in our current tTDD-P implementation. These netlists are then fed into our tool to derive behavior curves for each node. The POs' behavior curves are compared with the golden results generated from simulation point by point. The details of the timing model and golden results are described as follows:

Timing model: To enable comparison with DynaTune which only supports a *simple timing model*, we first did experiments for both DynaTune and tTDD-P for the same simple timing model where cells of the same type have the same delay. The golden results for this test group are also from simulation with the simple timing model.

Moreover, we also did experiments for tTDD-P with a *detailed timing model* where the timing information is directly extracted from design compiler's timing engine. We use the DC's "write_sdf" command to extract the timing in *Standard Delay Format* (SDF), which distinguishes: (1) each pin-to-pin delay, (2) the rise and fall delay, (3) the cell's driving strength, and (4) the cell's fanout load. This SDF file is then converted into a delay table to be used in our tool.

Timed simulation: We use Cadence NCsim 6.2-S007 as our simulator. First, the SDF file extracted from DC is back-annotated to the netlist. Input vectors are randomly generated according to specified probabilities. In our experiments we set the input static probability P=0.5. The timed simulation is carried out in floating mode[19], under which the internal gates and nets take value "X" before a single input vector is applied to settle down their values.

Golden result: To determine POs' dynamic activities during the simulation, we write a *Verilog Programming Interfacing* (VPI) program using the callback mechanism to record the timestamp of each PO's last switching activity in every cycle. After the simulation of one million cycles, these timestamps are collected to form the PO's dynamic behavior curve.

7.1 Accuracy comparison

Dynamic behavior curves derived from our algorithm and the original DynaTune algorithm [16] are both compared with the golden behavior curves generated from simulation. To quantify the accuracy, we use Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) measures. Only the data points between the earliest AT and latest AT on these dynamic behavior curves are compared, because the data points beyond this range are trivial.

Table 2: Dynamic behavior curve accuracy comparison

	DynaTune		tTDD-P			
	Simple		Simple		Detailed	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
C17	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%
C432	1.7%	3.6%	1.7%	3.7%	3.0%	5.9%
C499	1.4%	3.3%	1.4%	3.5%	1.0%	3.1%
C880	0.6%	2.3%	0.8%	2.4%	2.1%	5.2%
C1355	2.1%	4.9%	2.3%	6.0%	1.3%	2.4%
C1908	1.6%	3.1%	1.7%	3.3%	1.9%	3.7%
C2670	1.5%	4.7%	1.1%	4.0%	2.1%	5.7%
C3540	Failed	Failed	6.1%	11.1%	6.3%	11.2%
C5315	2.8%	4.9%	2.6%	4.7%	4.2%	6.5%
C6288	Failed	Failed	1.9%	5.2%	3.9%	9.9%
C7552	Failed	Failed	2.2%	5.3%	2.0%	4.6%
Ave.	**1.5%**	**3.4%**	**2.0%**	**4.5%**	**2.5%**	**5.3%**

The computed results of DynaTune (simple) and tTDD-P (simple) are compared to simulations using the simple timing model. The computed results of tTDD-P (detailed) are compared to simulations with a detailed timing model. From Table 2, we notice that with the simple timing model, the accuracy of both DynaTune and our algorithm are very high, and tTDD-P is just slightly worse than DynaTune. Our algorithm with the detailed timing model still maintains very good accuracy with a MAE of 2.5% and a RMSE of 5.3%. Note that three circuits cannot be finished by DynaTune due to running out of memory.

7.2 Runtime comparison

The experiments are done on a 2.8GHz Intel Xeon processor with 4GB of memory. The runtimes for different configurations are shown in Table 3. With the detailed timing model, NCSim takes tens of minutes to finish one million cycles of simulation. DynaTune can run into malloc failure due to BDD size explosion. tTDD-P can finish computation in less than 2 minutes for every circuit. Compared to DynaTune with a simple timing model, tTDD-P with detailed timing model still achieves an average speedup of 40x.

Table 3: Runtime comparison

	Ncsim Detail	Dyna-Tune (Simple)	tTDD-P (Detailed)			Speedup over DynaTune
			Partition	tTDD	Total	
C17	1m3s	0.01s	0s	0.01s	0.01s	1
C432	8m38s	34.3s	0.06s	2.54s	2.6s	13
C499	8m30s	326.9s	0.05s	5.36s	5.41s	60
C880	9m59s	41.9s	0.06s	3.35s	3.41s	12
C1355	9m7s	643.1s	0.04s	7.21s	7.25s	89
C1908	9m54s	977.61s	0.1s	8.28s	8.38s	117
C2670	38m53s	184.1s	0.15s	6.66s	6.81s	27
C3540	22m28s	Failed	1.13s	74.95s	76.08s	N/A
C5315	49m46s	84.2s	0.48s	15.97s	16.45s	5
C6288	66m32s	Failed	0.97s	101.3s	102.27s	N/A
C7552	63m53s	Failed	0.56s	20.09s	20.65s	N/A
Average speedup						41x

8. Conclusion

In this work, we proposed a method to analyze circuit-level dynamic behavior with a new data structure tTDD. False assignment pruning and random variable compaction are proposed to take care of temporal correlation. In addition, a new partitioning heuristic is used to deal with structural correlation and the scalability issue.

9. Acknowledgement

This work is partially supported by an SRC grant 2007-1592 and an NSF grant CCF 07-02501.

10. REFERENCES

[1] D. Ernst, et al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation.", *Annual Intl. Symp. on MICRO,* 2003.

[2] F. N. Najm,. "Transition density, a stochastic measure of activity in digital circuits.", In *Proceedings of* the DAC, 1991.

[3] C. Tsui, et al., "Efficient estimation of dynamic power consumption under a real delay model.", In *Proceedings of* the ICCAD, 1993.

[4] J. C. Costa, et al., "Switching activity estimation using limited depth reconvergent path analysis.", In *Proceedings of* ISLPED, 1997.

[5] S. C. Seth, et al., "PREDICT: Probabilistic Estimation of Digital Circuit Testability.", In *Proceedings of* the FTCS, 1985.

[6] D. Cheng. "Power Estimation of Digital CMOS Circuits and the Application to Logic Synthesis for Low Power.", PhD thesis, UCSB, 1995.

[7] S. Bhanja, et al., "Dependency preserving probabilistic modeling of switching activity using bayesian networks.", In *Proceedings of* the DAC, 2001.

[8] R. Marculescu, et al., "Probabilistic modeling of dependencies during switching activity analysis.", *IEEE Transactions on* CADICS, 1998.

[9] T. Uchino, et al., "Switching activity analysis using Boolean approximation method.", In *Proceedings of* ICCAD, 1995.

[10] Y. Kuo, et al., "Efficient Boolean characteristic function for fast timed ATPG.", In *Proceedings of* ICCAD, 2006.

[11] G. Silva, et al., "Satisfiability models and algorithms for circuit delay computation.", ACM Trans. DAES, 2002.

[12] K. Bowman, et al., "Circuit techniques for dynamic variation tolerance.", In *Proceedings of* DAC, 2009.

[13] B. Greskamp, et al., "Blueshift: Designing processors for timing speculation from the ground up.", In *Proceedings of* HPCA, 2009.

[14] T. Sasao, "Ternary Decision Diagrams: Survey.", In *Proceedings of* ISMVL, 1997.

[15] B. Kapoor, "Improving the accuracy of circuit activity measurement.", In *Proceedings of* DAC, 1994.

[16] L. Wan and D. Chen, "DynaTune: Circuit-Level Optimization for Timing. Speculation Considering Dynamic Path Behavior.", In *Proceeding of* ICCAD, 2009.

[17] A. B. Kahng, et al., "Slack Redistribution for Graceful Degradation Under Voltage Overscaling. ", In *Proceedings of* ASP-DAC, 2010 .

[18] J. Cong, et al., "Logic Synthesis for Better Than Worst-case Designs.", In *Proceedings of* VLSI-DAT, April 2009.

[19] S. Devadas, et al. "Computation of floating mode delay in combinational circuits: practice and implementation," *IEEE Transactions on* CADICS, 1993

[20] SIS unofficial release 1.3: *http://embedded.eecs.berkeley.edu/Alumni/pchong/sis.html*