WISCAD – VLSI Design Automation

Bound-Based Identification of Timing-Violating Paths Under Variability

Lin Xie and Azadeh Davoodi
Dept. of Electrical & Computer Engineering
University of Wisconsin - Madison

WISCAD
VLSI Design Automation Lab
http://wiscad.ece.wisc.edu

Path Identification Under Variability

- Identification is challenging under process and environmental variations
  - Delay of a path varies for each point in variation space
- Useful in different applications
  - At-speed test
  - Post-silicon repair of timing failures
  - Incremental timing-driven optimization

Some Previous Works

- **[Wang et al, TCAD' 04]**
  - Finds *M* paths with highest probability of violating a timing constraint
  - High error for small M and simplified Statistical Static Timing Analysis
- **[Zolotov et al, ICCAD'08]**
  - Finds *M* paths that best "represent" the variation space in which timing violation occurs (Test Quality Metric)
  - Uses branch-and-bound for path pruning
  - Limited number of paths are expected to predict chip failure during testing
- **[Heloue et al, ICCAD'08]**
  - Finds longest paths for each point in the variation space
  - No notion of timing constraint

Contributions

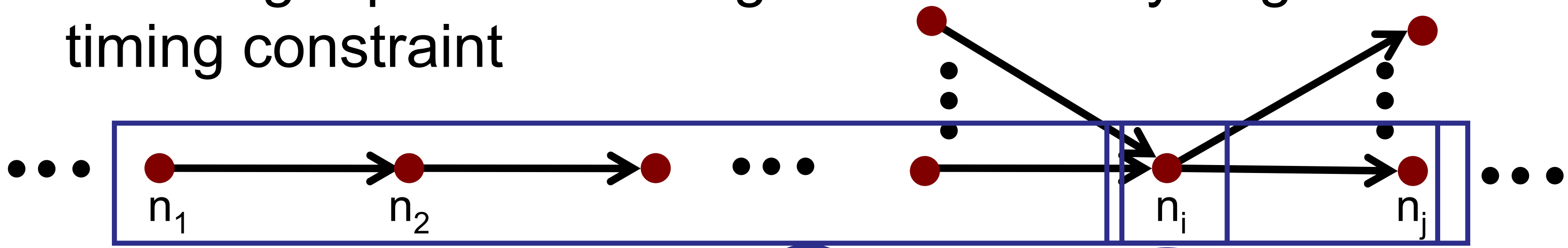
- Analytical bounds for "violation-probability" of a path
  - No assumption on technique used for variation analysis
  - Incremental update (in constant time) if path segment is extended to a larger one
- Demonstrate the use of bounds to find *M* paths with highest "timing-violation probability"
  - Paths found efficiently with high accuracy

Bound-Based Path Extraction

- **Can we identify timing-violating paths efficiently?**
  - Pick up promising nodes/edges to build paths
  - Use lower/upper bounds to prune redundant paths
- **Difficulties**
  - How to evaluate the importance of nodes/edges?
  - How to efficiently and accurately compute the lower/upper bounds of the connected edges?

Violation Probability of A Node/Edge/Path

- Probability that a node/edge/path-segment will be subset of a longer path which might have a delay larger than a timing constraint



Arrival Time of i

Required Arrival Time (includes delay of node i)

Timing Constraint

$$C_{n_i} = \Pr(D_{n_i} \geq D_{tar}) = \Pr(AT_i + RAT_i \geq D_{tar})$$

$$C_{e_{ij}} = \Pr(D_{e_{ij}} \geq D_{tar}) = \Pr(AT_i + RAT_j + d_j \geq D_{tar})$$

$$C_{p_i} = \Pr(D_{p_i} \geq D_{tar}) = \Pr(AT_1 + \sum_{k=2}^{i} d_k + RAT_i \geq D_{tar})$$

- D_{ni}, D_{eij}, D_{pi} represent delay of longest paths going through n_i, e_{ij}, p_i and are all random variables

Problem Statement

- Given a timing-graph with nodes *N* and edges *E*, identify *M* paths with highest violation probabilities (i.e., C_{pi})
- Approach:
  1. Efficiently pre-compute C_{ni} and C_{eij} of all nodes/edges
  2. Find paths using one traversal of timing graph and applying bound-based pruning
     - Use C_{ni} and C_{eij} to efficiently find path violation probabilities and prune paths

Computing Node/Edge Violation Probability

$$C_{n_i} = \Pr(D_{n_i} \geq D_{tar}) = \Pr(AT_i + RAT_i \geq D_{tar})$$

$$C_{e_{ij}} = \Pr(D_{e_{ij}} \geq D_{tar}) = \Pr(AT_i + RAT_j + d_j \geq D_{tar})$$

- Using existing SSTA techniques, we can express the AT_i, RAT_i, d_i using generic quadratic expression such as:

$$AT_i = \sum_{j=1}^{n} c_j (x_j + a_j)^2$$

Process variation

- We can compute all AT_i by one forward SSTA, and all RAT_i by one backward SSTA

Computing Node/Edge Violation Probability

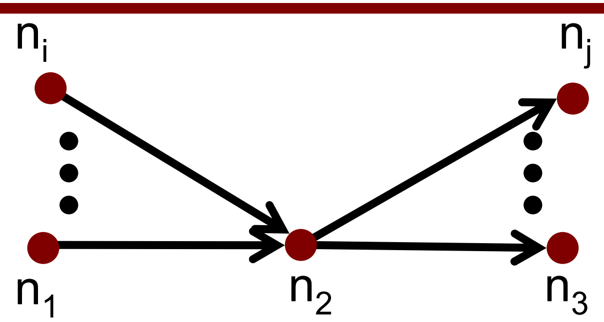
$$C_{n_i} = \Pr(D_{n_i} \geq D_{tar}) = \Pr(AT_i + RAT_i \geq D_{tar})$$

- **To estimate the violation probability efficiently**
  - Use a technique known as Pearson Curve **[Solomon, JASA'78]**
  - Each probability computation involves several 10x20 table-lookups and low-complexity interpretation operations [such as multiplication/addition]
  - Allows working with non-linear (quadratic) SSTA
- **Complexity of node/edge violation probabilities**
  - Two rounds of SSTA for finding all node/edge AT/RAT
  - Constant time at each node to compute the violation probability using Pearson Curve

Approach

1. Efficiently pre-compute C_{ni} and C_{eij} of all nodes/edges
2. Find paths using one traversal of timing graph and applying bound-based pruning
   - Use C_{ni} and C_{eij} to efficiently find violation probability and prune paths
   - Will start from finding bounds for a simple path of two connected edges

Two Connected Edges: Lower Bound



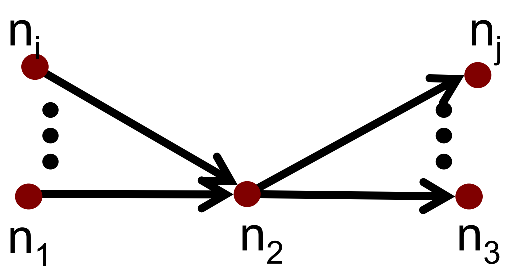
[Lemma] The lower bound for violation of two-connected edges is:

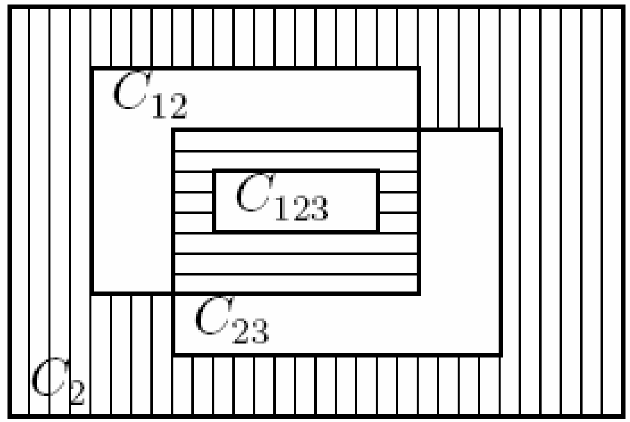
$$C_{123} \geq \boxed{C_{12} + C_{23} - 2C_2 + \Pr(D_2 \geq \max_{\forall j \neq 3}(D_{2j}, D_{tar}))}$$

- Computation of lower bound requires:
  - Pre-computed C_{12}, C_{23}, C_2
  - One statistical Maximum operation, noting D_2, D_{2j} are easily computed by adding the pre-computed ATs and RATs

Proof of Lower Bound (Follows from lemmas 1 and 2)

$$C_{123} \geq C_{12} + C_{23} - 2C_2 + \Pr(D_2 \geq \max_{\forall j \neq 3}(D_{2j}, D_{tar}))$$





[Lemma 1] $C_{123}=C_{12}+C_{23}-C_2+I_1-I_2$

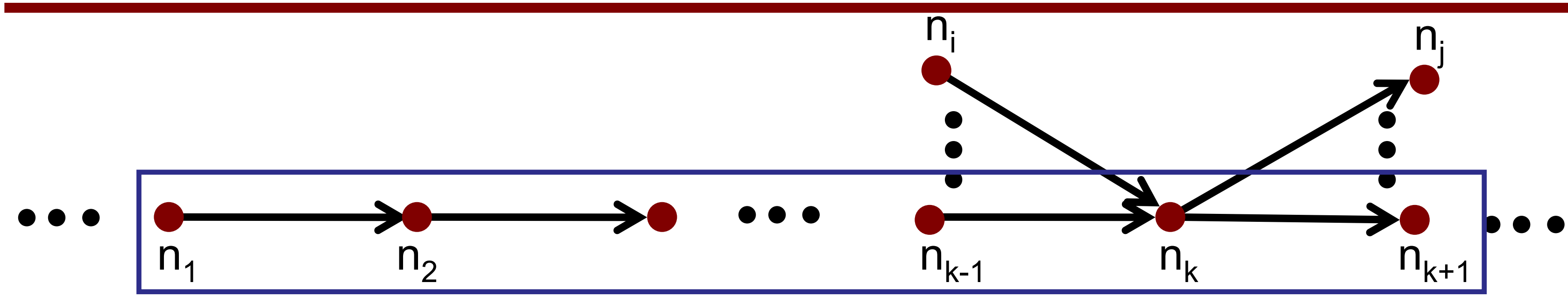
$I_1=\Pr((D_2 \geq D_{tar}) \cap (D_{12} < D_{tar}) \cap (D_{23} < D_{tar}))$

$I_2=\Pr((D_{12} \geq D_{tar}) \cap (D_{23} \geq D_{tar}) \cap (D_{123} < D_{tar}))$

[Lemma 2] $I_1 - I_2 \geq \boxed{\Pr(D_2 \geq \max_{\forall j \neq 3}(D_{2j}, D_{tar})) - C_2}$

$I_1 + I_2 \leq I_3 \rightarrow -I_1 - I_2 \geq -I_3 \rightarrow I_1 - I_2 \geq \boxed{-I_3}$

Extension to Many Connected Edges



$$C_{123\ldots k+1} \geq \sum_{i=1}^{k} C_{i,i+1} + \sum_{i=2}^{k} (-2C_i + \Pr(D_i \geq \max_{\forall j \neq i+1, j \in FO(i)} (D_{i,j}, D_{tar})))$$

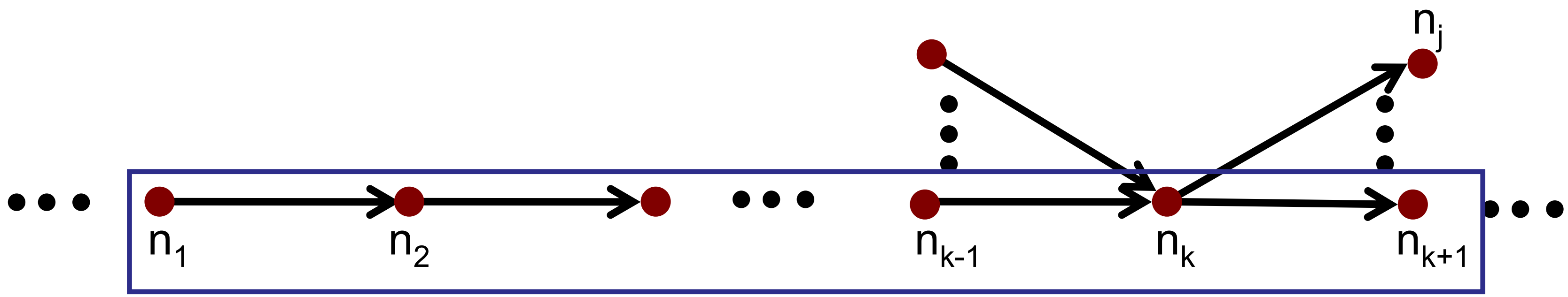
[Lemma] Lower bound L_{k+1} of path-segment ($n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_{k+1}$) is computed bottom-up:

$$L_{k+1} = L_k + C_{k,k+1} - 2C_k + \Pr(D_k \geq \max_{\forall j \neq k+1, j \in FO(k)} (D_{k,j}, D_{tar}))$$

(Proof using induction)

- **Constant time to update given L_k**
  - $C_{k,k+1}$ and C_k pre-computed
  - One statistical maximum operation needed

Upper Bound



[Lemma] Upper bound U_{k+1} is computed bottom-up:

$$U_{k+1} = \min(U_k, C_{k,k+1}) \Rightarrow C_{123...k+1} \leq \min_{\forall i=\{1,2,...,k\}}(C_{i,i+1})$$

- Intuitively, delay of longest path including segment ($n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_{k+1}$) is smaller than delay of longest path including ($n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_k$)

Dynamic Programming Path Extraction

1. Visit nodes in the timing-graph in topological order from primary inputs to primary outputs.

2. At each node n_i, add edge e_{ij} to all the paths P_j stored at fanin n_j of n_i.

3. Merge all paths P_j for each fanin n_j of n_i and remove the inferior paths using the bound-based pruning.

4. At the primary output node, select the top desired number of paths using calculated violation probabilities.

Step 3: Path Pruning

At intermediate node n_i

- Compute the lower/upper bound for the stored paths
- Prune the paths whose upper bound is smaller than the M-th largest lower bound at the visited node

Special case: Since lower bounds are computed bottom-up and depend on previous lower bounds, the error accumulates after a few stages, therefore:

- If the number of paths after pruning is larger than αM ($\alpha > 1$), use actual violation probability to replace the lower bounds for some paths

Step 4: Path Pruning

At primary output nodes:

- Compute actual violation probability of all propagated paths and select M paths with highest violation probability

OR

- Select M paths with the largest upper bound of their path violation probabilities [Faster]
  - Can alternatively use lower bound for selection

Selection of The Most Violating Path (M=1)

1. Define a weighted version of timing-graph

2. Identify and prune edges of the graph which are guaranteed not to be on the most violating path

3. For the remaining (sub)graph, find the most violating path using previous technique for special case of M=1

Selection of The Most Violating Path (M=1)

1. **We add weights to the edges of the timing graph as follows:**

$$w_{ij} = \begin{cases} C_{ij} + \Pr\left(D_i \geq \max\limits_{\forall k \neq j}\left(D_{i,k}, D_{tar}\right)\right) - 2C_i & \forall i \notin PI \\ C_{ij} & \forall i \in PI \end{cases}$$

- For edges connecting to a PI node, the weight is same as (pre-computed) edge violation probability
- For other edges, the weight expression is inspired by the expression of lower bound and requires one statistical Maximum operation per edge

$$C_{123\ldots k+1} \geq \sum_{i=1}^{k} C_{i,i+1} + \sum_{i=2}^{k}(-2C_i + \Pr(D_i \geq \max_{\forall j \neq i+1,\, j \in FO(i)}(D_{i,j}, D_{tar})))$$

[Note]: *The weight of any path in the graph is the lower bound of the violation probability of that path*

Selection of The Most Violating Path (M=1)

2. **Identify and prune edges of the graph which are guaranteed not to be on the most violating path**
   - Find the longest path and compute the summation of its edge weights, *LBmax*
   - *LBmax:* the maximum attainable lower bound

[Lemma] All edges e_{ij} for which $C_{ij} < LB_{max}$ can be removed from the graph and will not be in the most violating path.

3. **For remaining subgraph (which we should is of significantly smaller size) apply previous technique for M=1 to find most violating path**

Simulation Results

- Benchmarks: ISCAS'85 suite
- Technology: 90nm TSMC Library
- Process variations in channel length and zero-bias threshold voltage
  - 42 independent random variables
  - 21 independent Vt variables and 21 independent Leff variables for different regions specified by a 3-level hierarchical grid-model
  - Assume process variations have Gaussian distribution with standard deviation of 7% of their mean

Simulation Results

- Monte Carlo simulation to compute node and edge violation probabilities (pre-possessing setup)
- Considered finding paths for small values of M which have been shown to be more prone to error
- For comparison we applied Monte Carlo simulation to exactly find M paths with highest violation probability (search among all paths)

Path Extraction (*M*=200)

	Case I (M = 200)				
	Ave(C_p-LB_p)	Ave(UB_p-C_p)	Ave(C_{MC})-Ave(C_p)	Ave(C_p)	Runtime (SEC)
C432	0.0170	0.0001	0.0000	0.3823	3.89
C499	0.0001	0.0253	0.0142	0.2722	17.49
C880	0.0006	0.0006	0.0000	0.3516	1.15
C1355	0.0001	0.0253	0.0142	0.2722	16.51
C1908	0.0001	0.0347	0.0173	0.2567	14.15
C2670	0.0001	0.0102	0.0009	0.2615	1.02
C3540	0.0004	0.0000	0.0000	0.1948	9.17
C5315	0.0021	0.0001	0.0000	0.1878	3.25
C7552	0.0095	0.0280	0.0126	0.1201	7.43
AVE	0.0033	0.0138	0.0066		

We find the top M paths using MC simulation (search among all paths) and report different between average violation probability of the M paths found using MC and the M paths using our approach

Accuracy of the bounds: Average difference of actual violation probability (C_p) found from MC simulation with computed lower/upper bound is very small

Path Extraction (*M*=100)

	Case I (M = 100)				
	Ave(Cp-LBp)	Ave(UBp-Cp)	Ave(Cp)	Runtime (M=100)	Runtime (M=200)
C432	0.0298	0.0002	0.3960	1.26	3.89
C499	0.0002	0.0154	0.2962	7.81	17.49
C880	0.0005	0.0005	0.3776	0.44	1.15
C1355	0.0002	0.0154	0.2962	7.71	16.51
C1908	0.0002	0.0002	0.2953	2.67	14.15
C2670	0.0002	0.0029	0.3028	0.68	1.02
C3540	0.0008	0.0047	0.2138	3.09	9.17
C5315	0.0013	0.0099	0.2072	1.49	3.25
C7552	0.0000	0.0100	0.2039	2.49	7.43
AVE	0.0037	0.0066			

- Our algorithm still has very low error compared to Monte Carlo simulation results. The runtime is in seconds.

Graph Pruning (M = 1)

BENCH	LB_{max}	Pruning %	max(C_{pi})-I	max(C_{pi})-II
C432	0.4327	86.50	0.4327	0.4327
C499	0.3923	97.21	0.3923	0.3923
C880	0.4758	95.75	0.4758	0.4758
C1355	0.3923	97.21	0.3923	0.3923
C1908	0.3964	95.68	0.3964	0.3964
C2670	0.3911	97.89	0.3911	0.3911
C3540	0.3375	97.22	0.3375	0.3375
C5315	0.3375	97.58	0.3449	0.3449
C6288	0.3895	97.15	0.3895	0.3895
C7552	0.4086	98.95	0.4086	0.4086

We compared two cases: maximum Cpi among all the paths (case I) and maximum Cpi among the paths going through the edge with maximum edge violation probability (case II)

[Observation] The path with the highest timing violation probability goes through the edge with the highest edge violation probability.

- Pruning done using LBmax, corresponding to path with maximum lower bound
- Simple graph pruning can prune 96.12% edges on average

Summary and Conclusions

- Main contribution is in obtaining lower and upper bounds for a path segment
  - Need constant time for incremental update
- Showed application of bounds to find top M violating paths
  - Bounds were used for pruning in a dynamic programming framework
- Discussed simplified solution for graph pruning if the most violating path should be found
- Overall, bounds can be useful in other formulations of the problem and in other (non-dynamic programming) frameworks