

# Silicon feedback to improve frequency of high-performance microprocessors - an overview \*

Chandramouli Kashyap<sup>1</sup>, Pouria Bastani<sup>2</sup>, Kip Killpack<sup>1</sup>, Chirayu Amin<sup>1</sup>

<sup>1</sup>Intel Corporation, Hillsboro, OR

<sup>2</sup>University of California - Santa Barbara

## ABSTRACT

In modern high-performance microprocessors designed using advanced process technologies, the frequency of the part is often slower than what the static timing analysis tools predict before tape out. We give an overview of techniques used to observe the failing path on the tester, identify the dominant devices impacting the delay of the path, and learn from the failing path to fix other similar paths in the design. In particular, we describe a Support Vector Machine based approach for learning from speedpaths observed in silicon.

## Categories and Subject Descriptors

B.8.2 [Hardware]: Performance Analysis and Design Aids

## General Terms

Algorithm, Performance, Reliability

## Keywords

Speedpath, Timing analysis, Learning

## 1. INTRODUCTION

A modern high-performance microprocessor contains billions of paths, a subset of which are active in any given clock cycle as the processor is executing the instructions. The frequency of the part is limited by one slowest path, ironically called the speedpath, among the active ones in any cycle. Static timing analysis (STA) is used to determine such frequency limiting paths before tape out of the chip. Abstraction of the many complex low level phenomena is a key requirement for STA tools to work practically. While adequate in the past, these abstractions are increasingly harder to maintain in advanced process technologies. As a result, there are often surprises in silicon where the frequency of the path is observed to be lower than what the STA tool predicted before tape out. Besides, the path identified as the failing one often has large positive slack in the timing reports. As a result, the part is re-spun a few times (called a stepping) to fix timing and other logical bugs before the part is released for high volume manufacturing. Usually just fixing the speedpath that was observed in silicon is not sufficient since in the next stepping, a similar path whose delay was marginally better than the observed one may be the frequency limiter. Therefore, it is highly desirable that we fix as many paths similar to the failing

one before taping out for the next stepping. This way the number of steppings to push the frequency to the desired level can potentially decrease, improving the time to market for the part.

Hence, a systematic methodology for testing, identifying, root causing and learning from the few speedpaths observed from early silicon is required. We present an overview of such a methodology in this paper. We discuss what is needed at the tester level to isolate paths in the next section. In Section 3, once the path has been identified, we describe how we identify the devices that have the most impact on the path delay. In Section 4, we describe a machine learning method called Support Vector Machines (SVM) for learning characteristic features from the observed speedpaths and how it can be applied to fix other similar paths in the design. Section 5 describes feature selection for SVM and some results. We conclude the paper in Section 6.

## 2. SPEEDPATH ISOLATION

A modern microprocessor die has a quarter billion transistors. Even simple test patterns have millions of cycles (or logic vectors). For a given test pattern, there is a single logically sensitized path of transistors that runs slower than all other paths. The goal of speedpath isolation is to identify the slowest path and the cycle when it is slowest. This section gives an overview of the problem of isolating speedpaths in silicon. For more details, refer to [4].

### 2.1 Isolation on the tester

On the tester, a test pattern is run on a single die repeatedly. The voltage is fixed and, for each run, the frequency of the part is increased slightly until the part produces an incorrect result on the bus pins. That gives the frequency of failure for the test but not the clock cycle on which it failed because the offending path might have failed several cycles before the incorrect result appeared on the observable pins. In order to obtain the clock cycle, the part is run at a frequency just above the failing frequency. Subsequently, on a cycle-by-cycle basis, the clock is shrunk. If a critical speedpath is sensitized on the shrunk cycle, it is forced to fail as shown in Figure 1.

This failure propagates and is eventually detected on the external bus as a mismatch between observed and expected data. Once the failing cycle is known by a shrink and subsequent failure on the bus, scan latch data is extracted. The failing scan latches are identified by their incorrect state values following the shrink cycle. This helps to isolate the speedpath to a region of potential speedpaths on the die;

\*This work supported in part by CA Micro/Intel project No. 07-079

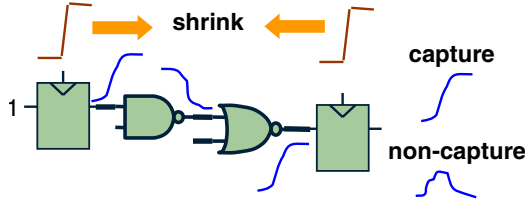


Figure 1: Clock shrink causing non-capture

however, the region may be large. Due to the rarity of scan latches, the non-captured data usually occurs on a non-scan sequential element (latch or flip-flop) and it may take one or more cycles for the faulty latched data to propagate to a scan latch. Thus, one is required to perform further analysis to spatially isolate the speedpath. Both logic and timing analysis is performed for this. Logic analysis corresponds to functional simulation and is not covered in this paper. We describe timing analysis next.

## 2.2 Isolation using timing

The logic analysis mentioned in 2.1 identifies a set of potential sink elements where the critical speedpath terminates. The cardinality of the set is one or more. For the purpose of final speedpath isolation and in order to get the data of interest to feed back to the timing flows, including relevant electrical causal effects, it is necessary to load the logic and tester information in a dynamic timing analysis infrastructure that is significantly different than the *static* timing tools used for design. Typically a cone of logic as depicted in Figure 2 is analyzed. We perform dynamic simulation of the cone using the logic vectors from the test pattern during the shrink cycle. This is in contrast to an STA framework in which worst-case logic sensitization is assumed on each gate in order to bound behavior across all possible logic patterns. In the simulation, as much detail as is possible, such as coupling aggressors, correct voltage and temperature, are used. In the situation when logic analysis identifies more than one potential sink latch, the timing bounds are used to further refine the list. In most cases there is a slow cone of logic and the others are non-critical with their upper bound being less than the lower bound of the slowest cone. Thus timing analysis filters the potential sink list to a single sequential element where the speedpath ends. Once the sink latch is identified, the next step is to identify which gates or devices in the cone cause the failure and which generating sequentials start the path. Logic alone cannot answer this question as there can be many potentially causal transitions in the cone. We outline this in the next section. Figure 2 shows the steps involved in the speedpath isolation.

## 3. IDENTIFYING THE DOMINANT DEVICES IN THE CONE

Consider the logic cone shown in Figure 2. The delay of every gate in the cone is a function of various parameters like  $L_e$ ,  $V_t$ ,  $W$ ,  $V_{cc}$ , Temperature, etc. As a result, the arrival time at the output node,  $A_o$ , is a function of every parameter affecting every gate in the cone along with the logic vector applied to the cone inputs. In this section we derive a linear analytical model that allows us to rank the effects and the gates in the cone that have the most impact

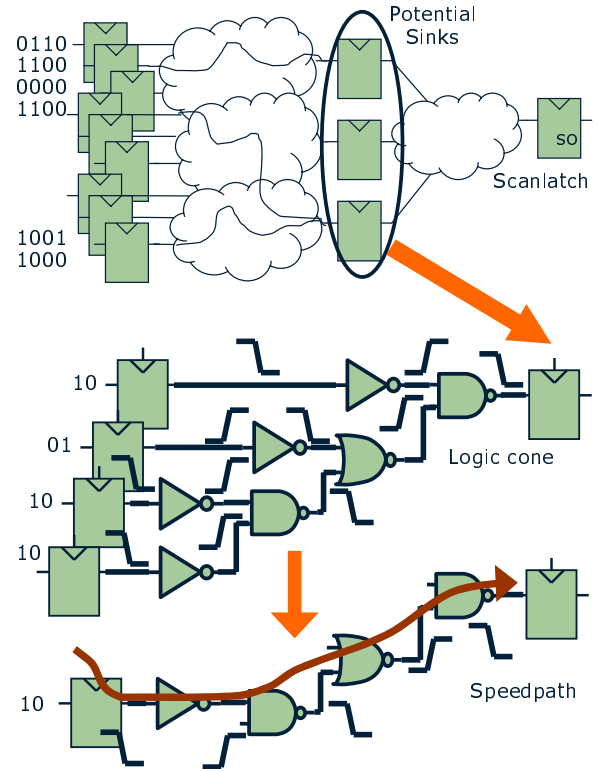


Figure 2: Isolated a speedpath spatially

on  $A_o$ . While it is true that the output arrival time shows a non-linear behavior when the input arrival times are close together, for small changes in the input arrival time the response surface looks almost linear. We find that most of the time the separation between the arrival times is large enough that one arrival time clearly dominates the other and the linearity assumption holds. When the two arrival times are indeed close enough, the variations in arrival times (due to the parameters described) for the individual measured cycle are small enough that over this range the linearity assumption still holds. As we show later in this section, experimental results validate this assumption.

Suppose that there are  $n$  gates in the cone and that each gate depends on  $m$  parameters. For simplicity, we assume that all the parameters are independent. In reality, for correlated parameters Principle Component Analysis can be performed to get a set of uncorrelated parameters. Let  $p_{ij}$  be the  $j$ th parameter of the  $i$ th gate where  $1 \leq i \leq n$  and  $1 \leq j \leq m$ . We assume the parameters are normalized such that  $p_{ij} = (\hat{p}_{ij} - \mu_{ij})/\sigma_{ij}$  where  $\hat{p}_{ij}$  is the original parameter,  $\mu_{ij}$  is the mean of the original parameter and  $\sigma_{ij}$  is the sigma. Assuming small variations of the parameter values around the mean, we can fit a linear model given by

$$A_o = k_0 + \mathbf{c}^T \mathbf{p} \quad (1)$$

Where we use bold upper-case letters to denote matrices and bold lower-case for vectors. Here  $\mathbf{c}$  and  $\mathbf{p}$  are  $mn \times 1$  column vectors. The vector  $\mathbf{c}$  denotes the *sensitivity* of  $A_o$ . The linearity assumption holds since gate delays are fairly linear in  $L$ ,  $V_t$ ,  $W$ , etc. even for a wide range of variations. A disadvantage of the model is that a large number of Monte-Carlo simulations are required to fit the model in 1. A closer

look shows that the arrival time  $A_i$  at the output of a gate  $i$  depends on only a few parameters local to the gate. Specifically,  $A_i$  depends on the arrival times at the inputs to the gate, the input slews, the parameters of the gate itself like  $W$ ,  $L$ ,  $Vt$ , etc. as well as the  $L$ ,  $Vt$ , and  $W$  of the gates in its immediate fanout as they affect the load seen by gate  $i$ . Since the number of parameters that an individual gate depends on is small, only a few simulations are needed to fit  $A_i$  to a linear model. Similarly, we also fit the output slew of the gate to a linear model. Let  $\mathbf{e}$  denote a vector of timing events, where the event could be an arrival time or the slew at the output of a gate. We then have:

$$e_i = k_i + \mathbf{b}_i^T \hat{\mathbf{e}}_i + \mathbf{c}_i^T \mathbf{p}_i \quad (2)$$

Where  $\hat{\mathbf{e}}_i$  is the vector of arrival times and slews at the inputs to the gate  $i$  and  $\mathbf{p}_i$  is the set of local parameters on which gate  $i$  depends. Next we show how  $A_o$  can be computed. With  $k$  PIs on the cone, let  $\mathbf{i}$  be a  $2k \times 1$  vector of arrival times and slews at the input to the cone. Let  $\mathbf{e}$  be the vector of arrival times and slews at all the nodes of the circuit and let  $\mathbf{p}$  be the  $mn \times 1$  vector of parameters of the circuit. The  $2n$  equations of the form 2 can be written as:

$$\mathbf{B}\mathbf{e} + \mathbf{D}\mathbf{i} + \mathbf{C}\mathbf{p} + \mathbf{k} = \mathbf{0} \quad (3)$$

Where  $\mathbf{D}$  is the  $2n \times 2k$  sensitivity matrix with respect to the PIs of the cone and  $\mathbf{C}$  is the  $2n \times mn$  sensitivity matrix of the gate parameters like  $L$ ,  $Vt$ , etc. Equation 3 can be solved for the output arrival time and slew at every node of the circuit:

$$\mathbf{e} = -\mathbf{B}^T(\mathbf{D}\mathbf{i} + \mathbf{C}\mathbf{p} + \mathbf{k}) \quad (4)$$

$A_o$  can be computed by picking off the corresponding entry from  $\mathbf{e}$  which can be symbolically written as:

$$A_o = \mathbf{f}^T \mathbf{e} \quad (5)$$

We refer to the model given by 5 as the local fitting model and the one given by 1 as the global fit model. Figure 3 shows the error using the linear model given in 1 compared to a golden simulation validating the linearity assumption of the model. Figure 4 shows the corresponding errors when using the model given by 5. We observe that the errors are less than 1.5 percent with the local model and also that it requires an order of magnitude fewer simulations than the global model given by 1.

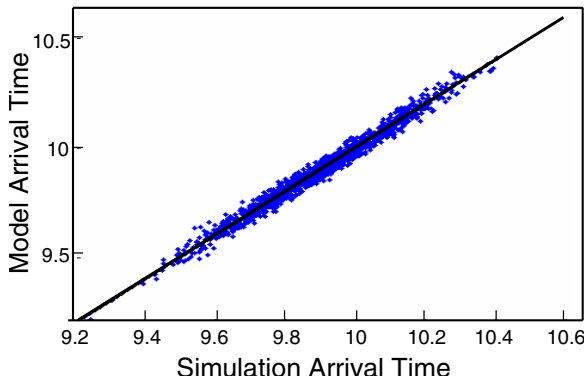


Figure 3: Global Fit Correlation

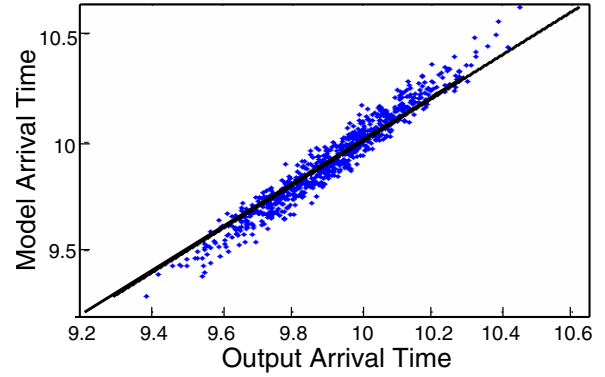


Figure 4: Local Fit Correlation

Using the techniques described in Sections 2 and 3 we identified 56 speedpaths in a 65-um microprocessor. We next describe a machine learning based technique for extracting features out of these paths and identifying other similar paths in the design.

## 4. SUPPORT VECTOR ANALYSIS

We only give a brief description; for details refer to [5]. Suppose that a path  $p$  can be described as a vector  $\vec{v} = (f_1, \dots, f_n)$  of  $n$  numerical values, where each  $f_i$  is called a *feature*.  $\vec{v}$  describes the characteristics of the path. For now, we postpone the discussion of how to generate these features to section 5 later. Given the 56 speedpaths, we assume that we have 56 vectors  $V = \{\vec{v}_1, \dots, \vec{v}_{56}\}$  describing the characteristics of these paths. Our goal in this section is to briefly describe how we can learn from  $V$  and produce a model  $M$  which can be used in the following way [11].

$M(\vec{v})$  for any path  $p$  described by  $\vec{v}$ , gives us a value telling us how likely the path  $p$  may become a speedpath. The "likeliness" is evaluated based on how similar path  $p$  is to one or more of the 56 speedpaths. The similarity between path  $p$  to a speedpath  $p_i$  is evaluated based on a *kernel* function  $k(\vec{v}, \vec{v}_i)$ , which takes the two vectors as inputs and produces a value that measures the degree of the similarity between them.

Since the number of speedpaths used to build the model is small and because the characteristics of these paths are diverse, it is unrealistic to expect we can learn a general model that captures only potential speedpaths without including other good paths. Therefore, in our algorithm we do not try to build a general model. Instead, our strategy is to start by building the most complex model, followed by a sequence of models with gradually reduced model complexity. Then, we develop a method to decide when to stop and use the model at the stopping point as our optimal model.

Suppose that  $M_i$  and  $M_{i+1}$  are built in two consecutive steps where  $M_i$  is more complex than  $M_{i+1}$ . Suppose we use these two models to search the design and identify the top  $k$  paths that have the highest potential to be speedpaths. Let these two sets of paths be  $S_i$  and  $S_{i+1}$ , respectively. We can check to see if  $S_i \approx S_{i+1}$ . If the two sets almost agree with each other, then we have a higher confidence that  $M_i$  and  $M_{i+1}$  are good models to use.

Although in this paper we do not provide a theoretical reason why  $S_i \approx S_{i+1}$  can be a good heuristic to use, we do experimentally validate that it does lead to more meaningful results. Moreover, it is intuitive to see that if  $S_i$  differs from  $S_{i+1}$  much, it cannot be the case that both of them are good.

Even though one of them may be a good model to use, we would not have a clear way to judge which one. Hence, the logical choice is to continue on building  $M_{i+2}$ .

In this work, our heuristic is based on checking the five sets  $S_{i-2}$ ,  $S_{i-1}$ ,  $S_i$ ,  $S_{i+1}$ ,  $S_{i+2}$  for every model  $M_i$  (i.e. two before and two after). We determine the model  $M_i$  such that these five sets agree with each other the most. Then, we consider  $M_i$  our best model.

In support vector analysis, the resulting model complexity can be controlled by how we define the kernel function. As explained before, a kernel defines how to measure the similarity between two path vectors. One common kernel is a radial-based function:

$$k(\vec{v}_i, \vec{v}) = e^{-\frac{\|\vec{v}_i - \vec{v}\|}{\sigma}} \quad (6)$$

Suppose that we have four paths, described by  $\vec{v}_{SV}, \vec{v}_1, \vec{v}_2, \vec{v}_3$ , where each vector consists of two values:  $\vec{v}_{SV} = (1, 0.2)$ ,  $\vec{v}_1 = (1.02, 0.19)$ ,  $\vec{v}_2 = (3, 0.1)$ , and  $\vec{v}_3 = (5, 1.2)$ . Suppose that  $\vec{v}_{SV}$  describes a speedpath. If we were to select potential speedpath(s) from the other three, by observing the values in the vectors, we would pick  $\vec{v}_1$ . Suppose that we build a model using  $\vec{v}_{SV}$  as the only support vector and let its  $\alpha$ , from eq. (6), be 1. Then the model is simply  $k(\vec{v}_{SV}, \cdot)$  and a value of 1 means that the path is most similar to  $\vec{v}_{SV}$ . Table 1 shows the similarity values given by the model based on three different chosen  $\sigma$ 's in the kernel function. Note how sweeping  $\alpha$  changes the similarity metric and can make  $\vec{v}_2$  and  $\vec{v}_3$  become ranked as similar to  $\vec{v}_{SV}$  when  $\alpha$  is equal to 40.

**Table 1: Three different similarity calculations**

|                              | $\sigma = 1/40$ | $\sigma = 1$ | $\sigma = 40$ |
|------------------------------|-----------------|--------------|---------------|
| $k(\vec{v}_{SV}, \vec{v}_1)$ | 0.3011          | 0.9704       | 0.9992        |
| $k(\vec{v}_{SV}, \vec{v}_2)$ | $\approx 0$     | 0.1224       | 0.9488        |
| $k(\vec{v}_{SV}, \vec{v}_3)$ | $\approx 0$     | 0.0067       | 0.9048        |

## 5. FEATURE GENERATION

In this section we explain how a path can be described as a vector. To do so, we first discuss the potential sources of modeling mismatch. Based on these sources we describe how the actual features used in the experiment were selected.

For speedpath prediction it is necessary to analyze situations with potentially *unaccountable* effects. Unaccountable effects refer to special situations where silicon behaves unexpectedly. Typically these effects are not accounted for due to their rarity of occurrence along with the large additional modeling cost. We group them into five categories of effects:

- **Topological effects** are modeling issues related to location, orientation, ordering and configuration. This can include, XY location, layout orientation, neighboring elements, metal layer connectivity, and/or number of via connections.
- **Dynamic effects** are modeling issues based on input patterns. These effects show up during regular operation of a design. Examples are: Multiple input switching, cross coupling noise, dI/dt voltage droop, RC voltage droop, temperature variation.
- **Static effects** are modeling issues that are independent of input patterns. These effects will appear everywhere the static element is placed. An example of this is if a specific cell is mis-modeled. Everywhere

the cell appears it will be mis-modeled independent of the patterns used to activate it. Other examples are during RC extraction, specific shapes or layers can be mis-modeled due to the extraction methodology.

- **Statistical effects** are modeling issues dealing with systematic variations in the physical characteristics of devices and interconnects related to the process. These effects include intra-die and inter-die variations due to lithography, dopant fluctuations and/or chemical mechanical planarization (CMP).
- **Random effects** include any effects that are not based on a statistical system. Examples of these are alpha particles present during manufacturing or seismic activity during CMP. These effects cannot be predicted or modeled.

In this work, based on our domain knowledge and application scenario we choose the following features:

1. Active device type in a path due to logic sensitization. These are based on four different flavors of N and P MOS transistors.
2. The predicted arc or stage delay for a particular cell in the path. This is a function of input transition time, output load and net resistance and capacitance.
3. Percent of total path delay due to the nets as opposed to cells.  $\%Net = NetDelay / (TotalPathDelay)$
4. Dynamic switching activity in the region.
5. Cross coupling aggressor impact.
6. Temperature variation, using Infrared Emission Microscopy to obtain a temperature map during functional operation [8].

With these feature definitions every path can now be described as a vector. In the next subsection we describe the results of applying our proposed methodology to identify potential speedpaths.

## 5.1 Results

A large design can have millions or even billions of different paths and we may not be interested in searching all of them. In this work, we select 19,000 paths that were sensitized during functional testing. Our goal is therefore to select potential speedpaths from this list.

Using a radial-based kernel and the heuristic described in section 4, we learned a sequence of models and determined the optimal model complexity. In the heuristic each path set  $S_i$  is the 50 most potential speedpaths resulting from model  $M_i$ . Figure 5, shows the results from the most complex model consisting of 56 support vectors to the least complex model consisting of 8 support vectors. The optimal model occurs at 24. Where 48 out of 50 paths are shared by the five models  $M_{22}$ ,  $M_{23}$ ,  $M_{24}$ ,  $M_{25}$ ,  $M_{26}$ .

It is interesting to note that with more than 26 support vectors (higher model complexity) there is little agreement between learned models. In other words, above 26 the model is unstable in the sense that a small change in complexity results in a large change in how the model determines the potential speedpaths. As complexity falls to between 23 and 19 support vectors there is no agreement between models, showing that the model becomes unstable again. Although model complexity below 19 support vectors tends to produce stability, those models are not chosen because: 1) the number of shared paths is much smaller than the 48 given by the optimal model, and 2) as mentioned before we prefer

a more specific (higher complexity) model than a general one (lower complexity). It is interesting to note that if we only compare three models during our complexity heuristic,  $M_{i-1}$ ,  $M_i$ ,  $M_{i+1}$ , the results are very similar to Figure 5, with 24 support vectors again being optimal.

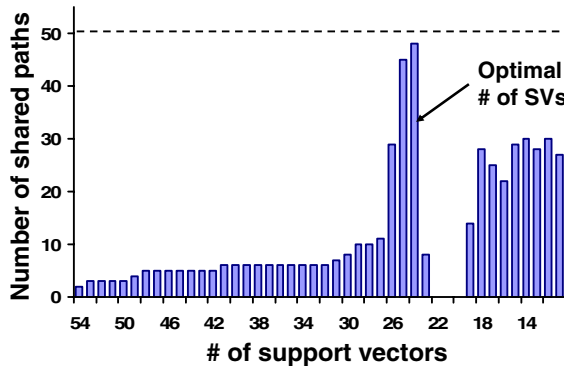


Figure 5: Sweeping model complexity

Using the optimal learned model  $M_{24}$ , we search all the 19,000 paths. For each path vector  $\vec{v}$  we compute its similarity  $M_{24}(\vec{v})$  and we sort all the similarity values and the result is shown in Figure 6. To select potential speedpaths for further improvement it is very easy to define a similarity threshold based on the knee of the curve. For example in Figure 6 we show such a threshold. From the figure we can see that there are 6 paths that clearly stand out from the rest. Furthermore, when zooming in on the right of Figure 6, we observe 7 additional paths that stay above the threshold. Using our searching methodology, we identified 13 paths that we deemed to be potential speedpaths.

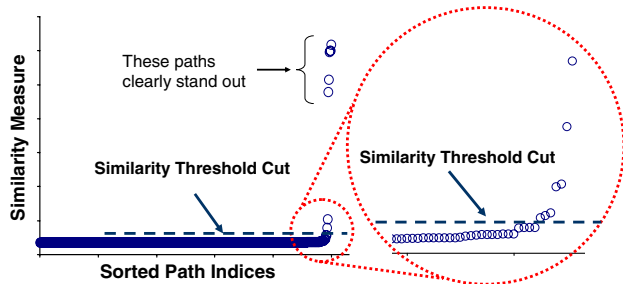


Figure 6: Result of support vector similarity ranking

After learning the results from our potential root cause analysis, we inspected the characteristics of the 13 paths identified by our support vector methodology. Not only did the paths share similar characteristics to one or more speedpaths, but also if we were to select potential speedpaths manually, after the root cause analysis these 13 paths would have been selected.

In particular, all the paths had either identical or very similar combination of active device types as one or more of the speedpaths. In addition to the active devices: 1) Six paths showed high delay pushout due to coupling noise. The amount of coupling noise pushout is equal to or greater than the calculated pushout for 21 of the 56 speedpaths. 2) Four paths had a large percent of total path delay due to the net delay as opposed to cell delay. The net delay was greater than 9.5% of the total path delay which is similar to 9 of the 56 speedpaths. 3) Two paths showed high amounts of

switching activity. The level of the switching activity for these paths is similar to 15 of the 56 speedpaths. Note that 2 of the 13 paths selected had both a large pushout and a large net delay.

Of the thirteen paths that were selected, three of them did not show any potential root causes. However, each of these paths has similar active device types, net delays, coupling noise pushout and switching activity to at least one speedpath. It is important to note that for these speedpaths no potential root causes were also found. This further emphasizes our earlier point that even without knowing the root cause information, our methodology can search the design and select potential speedpaths.

## 6. CONCLUSIONS

We have shown a new framework for analyzing real silicon speedpaths in order to obtain data to feed back into our timing flows. This data is the reality behind the abstracted and isolated models we often use in EDA. This paper outlined in brief a silicon speedpath isolation technique, causality analysis using sensitivities and a machine learning based framework for extracting features and identifying other similar paths in the design. We would like to encourage the research community to think about this problem and expand the approaches that will finally answer the fundamental question: how good are our flows in really predicting silicon?

## 7. ACKNOWLEDGMENTS

We acknowledge the following people at Intel for their contributions to our speedpath identification and root cause analysis in the form of guidance, discussions and implementation: Suriyaprakash Natarajan, Praveen Parvathala, Arun Krishnamachary, Eli Chiprout and Li C. Wang.

## 8. REFERENCES

- [1] L. Lee, L. Wang, P. Parvathala, TM Mak, "On Silicon-Based Speed Path Identification," *Proc. VTS* 2005.
- [2] P. Bastani, B. Lee, L. Wang, M. Abadir, "Analyzing the risk of timing modeling based on path delay test," *Proc. ITC*, 2007.
- [3] Li-C. Wang, P. Bastani, M. Abadir, "Design-silicon timing correlation – a data mining perspective," *Proc. DAC*, 2007.
- [4] K. Killpack, C. Kashyap, E. Chiprout, "Silicon Speedpath Measurement and Feedback into EDA flows," *Proc. DAC*, 2007.
- [5] P. Bastani, Kip Killpack, Li-C. Wang, Eli Chiprout, "Speedpath prediction based on learning from a small set of examples," *Proc. DAC*, 2008.
- [6] B. Gottlieb, et al, "Silicon Debug: What Do You Do When Your ASIC Does Not Work as Fast as Expected?" *Proc. DAC*, 2004.
- [7] A. Agarwal, D. Blaauw, and F. Dartu, "Statistical Gate Delay Model Considering Multiple Input Switching," *Proc. DAC*, 2004.
- [8] D. Barton, P. Tangyonyong, J. Soden, A. Liang, F. Low, et al, "Infrared Light Emission From Semiconductor Devices," *ISTFA Proc.* 1996.
- [9] P. Larsson and C. Svensson, "Noise in digital dynamic CMOS circuits," *IEEE J. Solid-State Circuits* June 1994, pp. 655-663.
- [10] Sanjay Pant, Eli Chiprout, "Power Grid Physics and Implications for CAD," *Proc. DAC* 2006.
- [11] N. Cristianini, and J. Shawe-Taylor, "An Introduction to Support Vector Machine," *Cambridge University Press*, 2002.
- [12] Vladimir Vapnik, "The nature of Statistical Learning Theory," 2nd editions, *Springer*, 1999.