Mining AC Delay Measurements for Understanding Speed-limiting Paths *

Janine Chen[1,2], Brendon Bolin[1], Li-C. Wang[1],
Jing Zeng[2], Dragoljub (Gagi) Drmanac[1], Michael Mateja[2]
[1]Department of ECE, UC-Santa Barbara; [2]Advanced Micro Devices, Inc.

Abstract

Speed-limiting paths are critical paths that limit the performance of one or more silicon chips. This paper present a data mining methodology for analyzing speed-limiting paths extracted from AC delay test measurements. Based on data collected on 15 packaged silicon units of a four-core microprocessor design, we show that the proposed methodology can efficiently discovered actionable, design-related knowledge that would be difficult to find otherwise.

1 Introduction

In high-performance design, performance optimization does not stop upon the first tape-out. Silicon information is often analyzed to drive further speed and power optimization over multiple design revisions. In each revision, speed-limiting paths, or speed paths, are identified based on silicon samples to guide design optimization.

Typically, speed paths are identified by analyzing results from functional legacy tests. Because the process involves functional debug and diagnosis, it is tedious and demands significant engineering effort. Due to variations, speed paths on different chips can be different. Moreover, a chip may result in multiple speed paths each corresponding to a specific cycle of a specific functional test. Analyzing a collection of chips usually gives a handful of speed paths. For example, a speed-path identification process may analyze hundreds of chips to find tens of speed paths [1].

Once speed paths are found, they are analyzed carefully to uncover the causes. These causes can be used as pointers to identify additional paths as potential speed paths. Then, delays on speed paths and potential speed paths are improved in the revision. To shorten delay of a speed path, designer may take an approach that has nothing to do with the cause, for example by simply replacing some high-Vt (slower) cells with low-Vt (faster) cells on the path.

Uncover the cause of a speed path can be very challenging. To avoid this difficult step, the authors in [2] propose a methodology based on learning the characteristics of speed paths. The goal of learning is to build a model that predicts potential speed paths whose characteristics are similar to the speed paths. The authors in [3] generalize the methodology into a *similarity search* framework by developing better method for learning path characteristics.

Alternatively, (structural) speed paths can be identified by analyzing results from structural AC delay tests such as transition fault test patterns. This involves applying the test patterns at a sequence of frequencies. At the slowest frequency where a failure is first observed, a set of failing flip-flops are collected. For each flip-flop and a failing test pattern, one or more paths can be derived as speed paths.

The process of deriving paths from a failing test pattern is much easier than that in the functional mode. Hence, speed path identification usually does not stop at the slowest failing frequency. Instead, failing flip-flops at the next few frequencies are also collected and analyzed. Analyzing these flip-flops finds potential speed paths whose delays are close to but less than the delays of the speed paths. Because the total number of paths identified can be large, investigating and improving all of them may not be feasible. In this case, understanding the causes for those paths becomes an important alternative so that instead of fixing so many paths individually, one can simply fix the few causes.

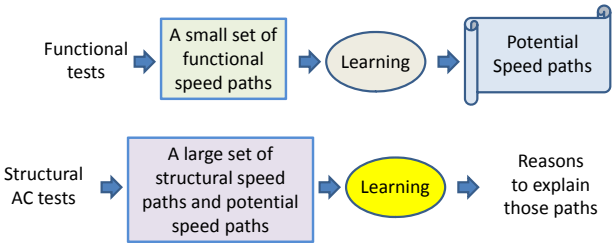

Figure 1. The role of learning in functional vs. structural speed path analysis methodology

Figure 1 illustrates the two distinct scenarios to apply a data learning methodology in speed path analysis. In the first scenario, functional tests are used and the goal of learning is to identify potential speed paths based on learning from a small set of speed paths. The works in [2][3] address this scenario. In the second scenario structural tests are used and the goal of learning is to uncover reasons to explain a large set of speed paths and potential speed paths. This work focuses on the second scenario.

*Work supported in part by National Science Foundation, Grant CCF-0915259 and Semiconductor Research Corporation contract 2007-TJ-1585

978-1-4244-7207-9/10/$26.00

In this work, we present a data mining methodology for speed path analysis. We applied this methodology to analyze speed paths and potential speed paths extracted from AC delay measurements collected based on 15 units of a 4-core microprocessor design. We summarize the experience from two perspectives: (1) development of the data mining flow, and (2) application of the flow with findings.

Rest of the paper is organized as the following. Section 2 gives the background of the work and reviews related work. Section 3 describes the AC delay measurements and some important aspects of the data. Section 4 discusses development of the data mining flow. Section 5 discusses considerations in application of the flow and presents the mining results. Section 6 concludes.

2 Background

In design, static timing analysis (STA) estimates timing slacks for paths, which can be used to produce a rank and select critical paths. This rank serves as a guide for performance optimization. Past studies have shown that it is difficult for STA to predict the actual path rank on silicon [4][5]. This often calls into question how much one can trust STA for power/timing optimization.

For high-performance design, timing analysis can only be relied upon to a certain point. After that, design optimization relies on information extracted from silicon samples. One common practice is to identify actual timing critical paths (speed paths and potential speed paths) on silicon samples. Then, the design is improved by reducing the delays on those paths. In post-silicon design optimization, a design may go through multiple revisions before its performance is deemed satisfactory. Although such an approach has been practiced for years, analysis of the timing critical paths remains a manual intensive process.

The path analysis step can be formulated as solving the following problem. Given two disjoint sets of paths, $S_{critical}$ and $S_{non-critical}$ where the size of $S_{critical}$ is much less than the size of $S_{non-critical}$, the goal is to uncover causes to explain paths in $S_{critical}$. While this is clearly a diagnosis problem, the work in [6] explains why such a diagnosis problem should be treated as a data learning problem. In particular, the learning problem involves two classes of paths, or a binary classification problem.

In the context of speed path analysis, the set $S_{critical}$ consists of speed paths and possibly potential speed paths. The set $S_{non-critical}$ consists of paths where each is activated by at least one test and does not show up as a timing critical path under any test on any sample chip. Usually, the size of $S_{non-critical}$ is much larger than the size of $S_{critical}$.

For example, with a given test pattern set, usually a large number of paths are activated by those patterns. Let this set be S. Suppose we let $S_{non-critical}$ be the set $S - S_{critical}$. Since usually a small subset of paths in S would become speed paths and potential speed paths, this would leave a large number of remaining paths in $S_{non-critical}$.

To uncover causes to explain paths in $S_{critical}$, one needs to define a representation for the causes. In classification rule learning [7], for example, causes are represented as *rules*. In the context of speed path analysis, a rule may look like: **if** *COND* is true, **then** the path is in the class of $S_{critical}$. Condition *COND* may correspond to conjunction of multiple path properties. For example, a property may correspond to having more than three MUX cells on a path.

Note that a rule is of the form "**if** *COND* **then** the path $\in S_{critical}$" rather than the form "**if** *COND* **then** the path $\in S_{critical}$ **else** the path $\in S_{non-critical}$" This means that each rule is for describing certain properties on the paths that belong to $S_{critical}$ and is not for predicting which class a path should belong to. Suppose a rule R is found to describe a subset of 9 paths in $S_{critical}$ of 100 paths in total. Suppose 1 path in $S_{non-critical}$ also satisfies the rule. Then, accuracy of the rule is $90\% = \frac{9}{9+1}$. The coverage of the rule is $9\% = \frac{9}{100}$ Note that in this calculation, the total number of paths in $S_{non-critical}$ is never used. This is because a rule has no "else" part and is not for classifying paths into the class represented by $S_{non-critical}$.

Given paths in $S_{critical}$, it is possible that some paths can be explained through the same rule. However, it is unlikely that a single rule can explain all paths. Moreover, one path may be explainable through multiple rules. This type of classification rule learning is called *sub-group discovery* [8] where rule learning involves discovery of a sub-group of paths that can be explained with one rule.

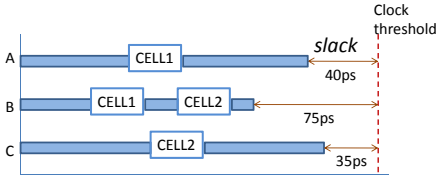

Figure 2. An example to illustrate sub-group discovery

Figure 2 gives a simple example of sub-group discovery. Suppose three speed paths A,B,C are found. In timing analysis, their slacks are estimated as 40ps, 75ps, and 35ps, respectively. Suppose the causes for them to become speed paths are due to inaccurate modeling of two cells, CELL1 and CELL2. With CELL1, suppose the true delay should be 50ps greater that that in the model. With CELL2, the true delay should be 40ps greater. In other words, the measured delay of path A on silicon is 10ps over the threshold. Similarly, path B is 15ps over and path C is 5ps over.

In rule learning, suppose we uncover the two rules: (1) *if* "path containing CELL1" *then* it is a speed path. (2) *if* "path containing CELL2" *then* it is a speed path. We see that rule (1) covers paths A,B and rule (2) covers paths B,C. The set $\{A,B\}$ is the first sub-group and the set $\{B,C\}$ is the second. Notice that these two sub-groups are not disjoint.

The discussion above shows that classification rule learning, in particular sub-group discovery, is at the core of the problem to be solved in the speed path analysis described in this work. With this perspective, intuitively one would think that developing the best learning algorithm to solve the problem should be the focus of the work. In practice, however, developing the best algorithm for the core problem, while important, is insufficient for implementing a useful flow. In practice, a successful analysis not only depends on the learning algorithm used to solve the core problem, but also on the information content of the dataset in use as well as the formulation of the question to be answered. This **data-question-algorithm** interdependence is crucial in the following sense:

- **Data-Question:** The data needs to contain sufficient information for answering the question. The question formulation is constrained by the data.
- **Question-Algorithm:** The question formulation needs to match the capability of data mining algorithm.
- **Data-Algorithm:** The data has to be in a form suitable for the algorithm. For example, if a binary classification algorithm is chosen, the samples in the data need to be divided into two classes. The decision for making the division can affect the effectiveness of the learning.

Because of this interdependence, a speed path analysis task usually cannot be accomplished through a single data learning step by applying a data learning algorithm to a particular dataset. Instead, it has to be an *iterative* data-centric process where in each step, a new dataset is prepared and possibly a new question is asked. In this work, the term "data mining" refers to this iterative process where multiple datasets are analyzed, different questions are asked, and multiple learning algorithms are applied, all toward achieving the high-level objective of understanding the structural timing critical paths observed on the 15 silicon samples.

3 The Data

We collected AC delay measurement data based on 15 microprocessor units from a four-core design, and used a transition fault pattern set. We applied frequency stepping to collect the data on all failing flip-flops' (FFs) first observed failure across all frequencies during the frequency stepping. For each FF, we recorded the failing pattern (or patterns) and its (or their) first failing frequency. We processed this information through a diagnostic tool to extract the paths that were activated by each pattern ending at the FF. Figure 3 illustrates this process.

Each failing FF may correspond to multiple patterns, and each pattern could lead to one or more paths. Moreover, a path might be shared by two patterns. In total the process identified 1,452 structurally critical paths from the top four frequency steps. Each path corresponds to at least one failing FF-pattern pair on at least one of the 60 cores. The total number of failing FF-pattern pairs was 2,422.

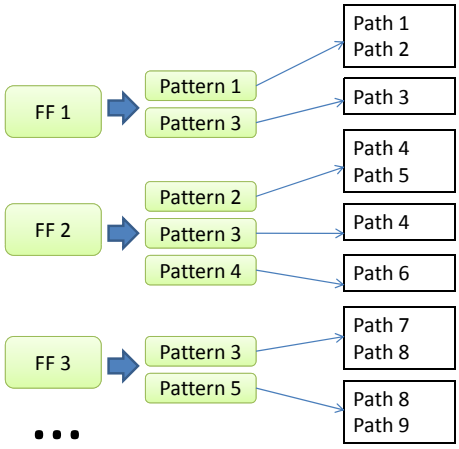

Figure 3. From failing FFs to potential speed paths

To simplify the data, if a FF-pattern pair produced multiple paths, we focused on the most critical path predicted by STA. This process reduced the 1,452 paths to 480 paths. Figure 4 shows the 480 paths based on the frequency steps from which they were extracted. This histogram shows that the number of extracted paths increases rapidly as the test frequency increases from Step 1 to Step 4. Let the clock period in Step 1 be p. For each frequency Step i, the clock period is $p(1-0.036*i)$.

Since we had 60 cores, a path might be extracted based on failing one or more cores. Figure 5 shows the number of cores that each path was failing on. It is interesting to observe that paths in Step 1 (most critical) failed almost all cores. Paths in Step 4 might fail from 1 to 60. This can be understood if one considers path delay as a statistical distribution (e.g., Gaussian).

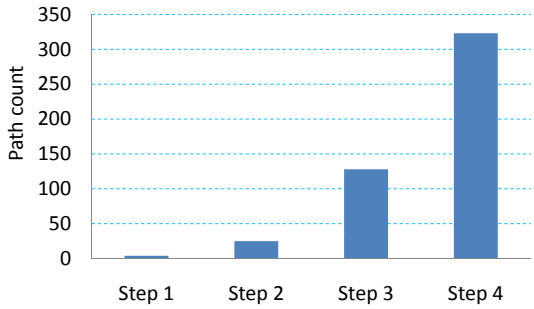

Figure 4. 480 top failing paths collected at different frequency steps

The result indicates that in Step 1, all paths have narrower distributions. In Step 4, by contrast, many paths have wider distributions (i.e., failing on only a few cores). One explanation could be that the delay variation on a more critical path was better control in the design. However, this is unlikely because, as we will explain, most of the paths shown in the figure were not predicted by STA as the top paths. Hence, designers would not naturally focus their optimization efforts on these paths in the design cycle.

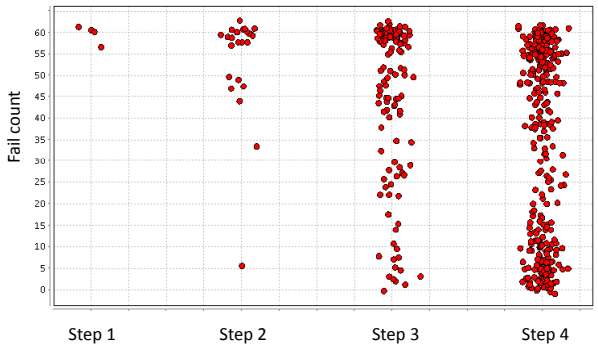

Figure 5. Fail counts across 60 cores for the 480 paths (In such a plot, a small jitter is injected to move the location of each path slightly for better visualization)

Regardless of the reason(s) behind Figure 5, the result was important because it indicated that many of these critical paths were due to some systematic effects causing them to fail on many cores. This justified the use of a data mining approach to uncover causes behind those effects.

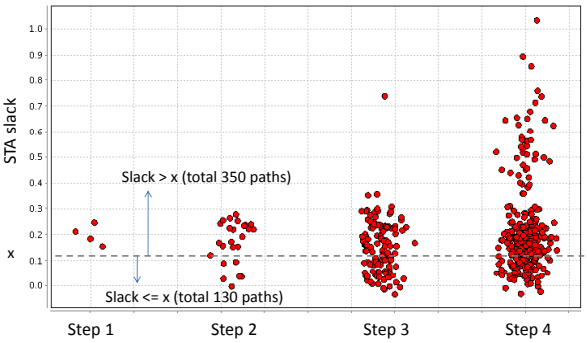

Figure 6. STA timing slacks for the 480 paths

Figure 6 shows that STA-predicted timing slacks were not good indicators for actual path criticality. A slack threshold x ps was conservatively selected to ensure that it was very unlikely for designer to consider any path above the threshold as critical paths. For example, there were about 22k critical paths in the design with slacks $\leq x$ ps.

As shown in the figure, among the 408 top silicon paths, only 130 paths have slacks $\leq x$. Among the total 1452 identified silicon critical paths, 304 paths have predicted STA timing slacks $\leq x$. Hence, it is clear that many silicon critical paths were not identified as critical paths by STA.

To gain further understanding of the nature of the 480 paths, Figure 7 and Figure 8 show statistics of low-Vt cell usage and high-Vt cell usage on those paths. The usage number is calculated based on ratio of the total low-Vt/high-Vt cell delay over the total path delay. This ratio is then normalized between 0 and 1. A lower value shows lower usage, and vice versa. The usage values are shown with different colors.

The design practice used low-Vt cells to accelerate a path and high-Vt cells to slow down a path and save power.

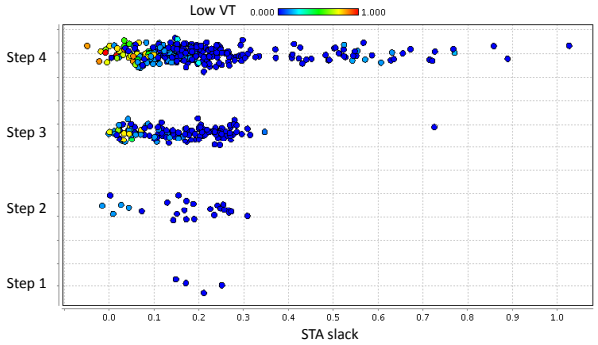

Figure 7. Color plot to show statistics of low-Vt cell usage on the 480 top failing paths

Hence, the two figures demonstrate the criticality of the 480 paths from this design perspective.

In Figure 7, it is interesting to observe that no low-Vt cell was used for majority of the paths. This implies that, in the design phase, these paths were not considered critical; otherwise, designers would have tried to accelerate them.

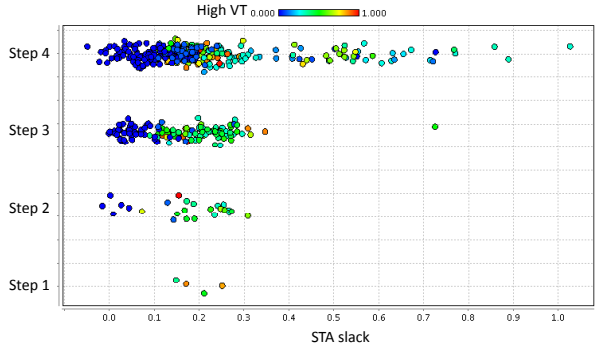

Figure 8. Color plot to show statistics of high-Vt cell usage on the 480 top failing paths

Figure 8 shows that many of the 480 paths contain high-Vt cells. This could mean designers considered these paths so non-critical that they even tried to slow them down to save power. Although STA is a primary approach to check for criticality of a path, it is not the only approach. Hence, results from Figure 7 and Figure 8 have a broader meaning than the result in Figure 6. While the STA result shows it could not predict silicon critical paths well, Figure 7 and Figure 8 show that many of the 480 paths were unexpected to be critical. Because they were unexpected critical paths, it was interesting to study why they became critical on silicon samples. In the following, we describe the data mining methodology employed in the study.

4 The Data mining Flow

Figure 9 depicts the data mining flow implemented in this work. The mining process is iterative, consisting of four major steps: question formulation, feature generation, data mining, and investigation. In addition, there are two

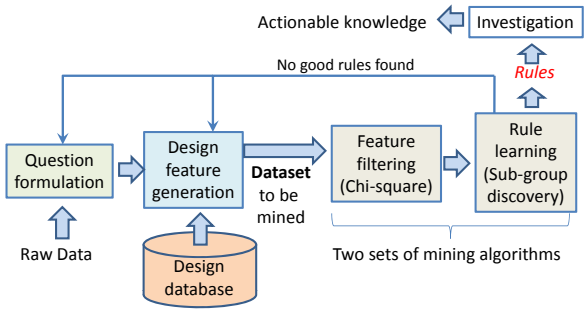

Figure 9. **The iterative data mining process**

sets of mining algorithms in the mining step.

The data mining flow is different from most of the prior works reported [2]-[5], [9]-[14], which usually described data mining as an input-to-output one-way process using one particular mining algorithm.

The process starts with raw data as input. For example, raw data contains the names of speed paths and potential speed paths. Notice that in each iteration, a separate dataset may be created for running the multiple learning algorithms. There are three important perspectives taken toward the data mining process in this work.

First, a raw dataset is large and the data is complex (e.g., with high dimensionality), which prevents effective analysis with simple visualization tools or statistical calculations. Then, it warrants taking a data mining approach. Second, the data might contain noise, so it is important to clean the data before mining it. Third, the emphasis is to discover "knowledge" that is useful in the sense that it can lead to meaningful design action(s). This is in contrast to merely finding statistically significant results, which might or might not have any practical meaning.

The knowledge discovery emphasis implies that the first two steps in the data mining flow (question formulation and design feature generation) are more important than the effectiveness of the mining algorithms in use. In other words, if one asks a right question with a properly prepared dataset, then even with a simple learning algorithm, meaningful results can still be found. In contrast, if one asks a wrong question with the data and/or the dataset is ill prepared, then even with the most sophisticated learning algorithm, it would still be difficult to uncover useful results. In the data mining flow shown in Figure 9, the question and the dataset are refined through the iterative process. Therefore, we see that it is this iterative refinement process that is most crucial to decide the success of a data mining task.

4.1 Question formulation

The raw data described in Section 3 has several dimensions: FF, pattern, path, number of failing cores, STA timing slacks, and a true/false value indicating whether a path is STA critical or not. There are many different perspectives to examine the raw data.

For example, one may ask the question: "What properties are so unique on the 4 paths shown with Step 1 in Figures 5 and 6 above?" To answer this question, the 4 paths are put into the set $S_{critical}$. Then, one needs to decide on the set $S_{non-critical}$. As an example, $S_{non-critical}$ may contain all paths found under frequency steps 2,3,4. This may not be a right question formulation if paths found in step 2 share similar causes with paths in step 1. However, one would only know this after running through the mining and discovering that no meaningful results could be found.

In this work, asking a right question means properly constructing the two sets $S_{critical}$ and $S_{non-critical}$. The question is always of the form "What properties are so unique on the paths in $S_{critical}$ (as comparing to the paths in $S_{non-critical}$)?" In general, asking the right question might not be obvious at the beginning of the data mining process. This is why the process is iterative. As shown in Figure 9, when no good rule can be found, one possible action is to refine the question and re-start the mining process.

In principle, a rule learning algorithm works like the following. The paths in $S_{critical}$ are used to form hypotheses for the causes and the paths in $S_{non-critical}$ are used to filter the unlikely hypotheses [9]. Hence, the more paths in the $S_{non-critical}$ set, the more effective the filtering could be. This means that in general using more paths in the $S_{non-critical}$ set could lead to results with higher statistical significance. Hence, the size of the $S_{non-critical}$ set is an important consideration to formulate the question.

To illustrate further subtleness involved in question formulation, consider the following formulation. Refer to Figure 6 above. Suppose we let $S_{critical}$ be the set of the 350 paths above the dot line. To decide on the $S_{non-critical}$ set, we took the 22k STA critical paths and removed all paths that showed up as silicon critical paths. This gave us 21,589 STA critical paths. We then let $S_{non-critical}$ be the set of these 21,589 paths. With this formulation, essentially we are asking the question: "Why the 350 paths were not predicted as critical paths by STA?"

The first issue with this formulation lies in the use of the 350 paths where some of them may not be the actual paths causing the observed FF failures. Recall in Figure 3 that a failing FF-pattern pair may result in multiple paths. Among them, we did not really know which one cause the observed failure. Hence, treating all 350 paths as silicon critical paths may or may not be entirely correct.

The second issue involves the use of all 21,589 STA paths. Not all of them were activated by the transition fault patterns. Hence, it is not entirely correct to classify all of them as non-critical paths on silicon. It is possible that with a different pattern set, some of them got activated and showed up as silicon critical paths observed in steps 1-4.

The two issues mentioned above indicate that there might be significant uncertainty in the way we classify paths

in the above question formulation. How this uncertainty impacts the learning results could be unclear and difficult to estimate. Because of this difficulty, in principle one should begin the data mining process with question formulations that have as less uncertainty as possible.

For example, among the 21,589 STA paths, we found that 12,248 paths were activated by one or more test patterns. Hence, these 12,248 paths were certain to be STA-critical and silicon-non-critical paths.

4.2 Design feature generation

To uncover causes, one needs to define a hypothesis space for the search. In the data mining methodology, a search space is implicitly defined through a set of design features [5]. Each feature encodes one design aspect that may contribute to the separation of the two classes of paths. Features can be defined based on various types of data in a design database, including circuit netlists, timing library, layouts, timing constraints, wire RC, timing report, etc.

Given two classes of paths, not all design features are relevant. For example, some cells are not used in the paths in $S_{critical}$. Then, there is no need to define features based on those cells. Producing relevant features is a non-trivial step. It is not easy to develop a complete feature set initially. Hence, the data mining process has to be iterated so the set of features can be refined based on the mining results.

In practice, it is likely to develop features that are correlated. For example, one may define two features A, B and also include a feature to describe its ratio $\frac{A}{B}$. This would not be a problem for data mining algorithms because they do not make assumption about the correlation among features [6][15]. Using too many features, however, could degrade the performance of a rule learning algorithm. Therefore, a feature-filtering step is usually applied to remove irrelevant features before rule learning.

Given n features $\{f_1, \ldots, f_n\}$, for each path i, the values of the features, $x_{i1}, \ldots, x_{in}$ need to be extracted from the design database. Different feature values can have different scales. For example, a feature counts the number of a MUX cell used on a path while another feature captures the maximum load on the path. It is a common practice to normalize feature values into a fixed range before processing the dataset through a data mining algorithm. For example, feature values are normalized into the range $[0, 1]$.

Our experience shows that it is more effective to start with an initial set of high-level features and expand this set with more detailed features based on mining results. For example, one may start with high-level features such as total cell delay, total net delay, and total clock delay. Results might indicate that clock delay is more important than cell delay, and cell delay is more important than net delay. In the next run, one can define a set of detailed features that affect clock delay and perform data mining with only those features. The result might be insufficient. Then, more detailed features can be added by dividing total cell delay with cell types. This iterative process continues until some meaningful rules are discovered.

4.3 Data mining algorithms and their usage

From Figure 9, we see that the mining process requires two sets of algorithms: feature filtering and rule learning.

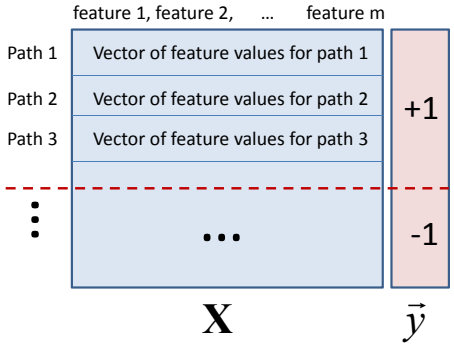

Figure 10. Illustration of a dataset for mining

Once a set of features are generated, each path is encoded as a path vector based on the features. The raw data is processed into a dataset shown in Figure 10. The dataset is denoted as $(\mathbf{X}, \vec{y})$ where $\vec{y}$ contains the labels for the two classes of paths and $\mathbf{X}$ is a table of feature vectors for all paths involved in the analysis.

4.3.1 Filtering unimportant features

The problem of feature ranking was studied in [5], which focused on identifying the most relevant features, In our feature filtering step, the objective is to remove the most irrelevant and redundant features to facilitate the subsequent rule learning step. The focus here is more on the run-time efficiency of rule learning and less on the diagnostic. This runtime efficiency is important because, if rule learning fails to produce a reasonable set of rules, we prefer to know that as early as possible so the process can go back to feature generation or question formulation.

Popular data mining methods that provide the feature ranking capability include, chi-square test [16], linear SVM [17][4], and Gaussian process (GP) [18]. It is important to note that each method evaluates feature importance differently. Hence, it is sensible to run more than one method and compare results. If multiple methods deem a feature unimportant, it is safe to remove it from further consideration.

For efficiency reason, sometimes features are partitioned into groups based on their types in initial analysis. For example, initially cell-based features and interconnect-based features were analyzed separately. Then, the important features were combined to generate another dataset for additional analysis. Our experience shows this strategy can be useful when presented with a large number of diverse features (e.g., hundreds of features extracted from different types of design files).

4.3.2 Rule learning

Given two classes of paths, $S_{critical}$ and $S_{non-critical}$, we are interested in finding rules to describe paths in $S_{critical}$. Hence, a rule is of the form: **if** *COND* then the path is in $S_{critical}$. Because $S_{critical}$ is fixed, to establish a rule is to find the condition *COND*.

In this work, we only consider condition with the following form: $COND := c_1 \wedge \cdots \wedge c_n$. Such a rule is called an n-order rule because it involves n predicates $c_1, \ldots, c_n$. Each predicate is a test on the value of a design feature. A test is of the form "feature value $\in [a,b]$" where one of the values, a, b could be ∞.

In sub-group discovery rule learning, two metrics can be used to assess the merit of a rule. Let i be the number of paths in $S_{critical}$ satisfying a condition *COND* and j be the number of paths in $S_{non-critical}$ satisfying the condition. The accuracy is calculated as $\frac{i}{i+j}$. Also, i divided by the total number of paths in $S_{critical}$ is the rule coverage.

In general, we would like to find rules by maximizing the accuracy and coverage simultaneously. However, it is more likely that one needs to trade coverage for accuracy and vice versa. For example, Figure 11 shows a simple example with two features. In the figure, a condition corresponds to a box in the two-dimensional plot (could be an open-end box if ∞ is used in the range). We see that rule R1 covers 4 paths in $S_{critical}$ with accuracy 100%. R2 covers 7 paths in $S_{critical}$ with accuracy $\frac{7}{8} = 87.5\%$. Also notice that R3 covers 4 paths also covered by R2. Note that since no disjunction operator ($\vee$) is allowed in a condition (only conjunction ($\wedge$) of predicates is allowed), R3 and R2 have to be treated as two separate rules.

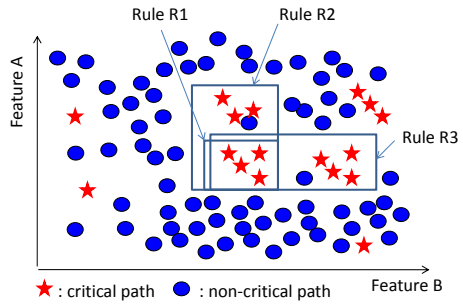

Figure 11. Illustration of rule learning with 2 features

Because of the accuracy-coverage tradeoff, usually in rule learning a user-supplied accuracy threshold is used. Based on such a threshold, then the rule learning finds rules that maximize the coverage. For example, in the example above if the threshold is set at 90%, rule learning would find R1 but not R2. If the threshold is set at 85%, rule learning would find both R2 and R1 and rank R2 higher than R1.

In the example, suppose R2 is found first. Then, the 7 paths in the box are already covered. In order to continuously find rule R3, the 4 paths in the box of R1 have to be kept. However, we could not simply keep the 7 paths covered by R2 because that would lead us back to R2 again. In sub-group discovery, a weight is assigned to each path based on how many times it has been covered by the rules found so far. Then, rule coverage is calculated based on these weights. For example, suppose initially all paths have weight 1. Each coverage results in dividing the weight by 2. Then, after R2 is found, the coverage of R3 is no longer 8 but 6 because 4 paths are already covered by R2 and their weights would be 0.5 in calculating the coverage for R3.

In general, a rule learning algorithm consists of three methods [7][8]: (1) a method to generate a set of rule candidates, (2) a method to assess the merit of a rule, and (3) a method to weight covered samples in sub-group discovery. Methods (2) and (3) are discussed above. Below we discuss generation of rule candidates.

Suppose we are only interested in 2-order rules. This means that a condition is of the form: "$f_1 \in [a_1,b_1] \wedge f_2 \in [a_2,b_2]$ where f_1, f_2 are two features and $[a_1,b_1], [a_2,b_2]$ are two ranges. Suppose features have numerical values. Then, in theory there can be infinite number of choices for the ranges. In practice, this can be resolved by discretizing each feature value with pre-assigned ranges [9]. After discretization, the choices for the ranges become limited.

Suppose each feature value is discretized into k ranges. With m features, the total number of possible 2-order rules is limited by $k^2 \frac{m(m-1)}{2!}$. The number of 3-order rules is limited by $k^3 \frac{m(m-1)(m-2)}{3!}$. We see that this number grows exponentially on the order. Therefore, exhaustive search of all rules is infeasible.

In practice, rules involving more than three features are hard to explain. Hence, the search usually stops at 2nd or 3rd order. By limiting the search to such low-order rules, exhaustive search becomes possible.

Focusing on producing only low-order rules is a result of the knowledge discovery emphasis discussed at the beginning of Section 4. In other words, we are not interested in complex rules, even if they are statistically significant. We are interested only in finding simple rules that can be interpreted with domain knowledge and with a reasonable amount of effort be translated into actionable design knowledge (in the investigation step in Figure 9).

By limiting to low-order rules, generation of rule candidates becomes a question of how to properly discretize feature values. Many discretization algorithms were proposed in the past [19]. Experience shows that cluster analysis based discretization usually gives the best result [9].

5 Application and Findings

Recall from Section 3 that we collected 1,452 paths through 4 frequency steps. Since each failing FF-pattern pair might activate multiple paths, there was significant uncertainty in the data associated with those 1,452 paths. In

other words, we could not be sure which path was really the silicon-critical path causing the failure when multiple paths were activated by a test pattern. To reduce such uncertainty, we created two reduced-path sets, as illustrated in Figure 12. The set of the 480 top paths is discussed above in Section 3. For each FF-pattern pair, instead of collecting all paths activated by the pattern, we collected only the most critical path predicted by STA. This gave the 480 top paths. However, as discussed before, not all paths in this list are certain to be true silicon-critical path.

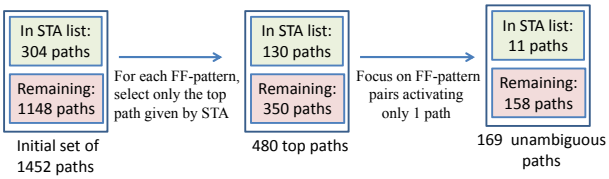

Figure 12. Three sets of observed silicon paths

To further reduce the uncertainty in the data, we focused on FF-pattern pairs that activated a unique path. This resulted in 169 paths. For these 169 paths, we were certain that they were silicon-critical paths.

In Figure 12, each set of paths are divided into two subsets, one with paths whose STA timing slacks $\leq x$ ps (see discussion associated with Figure 6 above) and the other with paths whose STA timing slacks $> x$. For example, among the 169 silicon-critical paths, only 11 paths had their timing slacks $\leq x$ ps. Hence, the other 158 paths were not considered by STA as critical paths. In the following, we focus our discussion on analyzing the 169 paths. The processes to analyze data in other scenarios are similar.

Recall from the discussion at the end of Section 4.1 that there were 12,248 STA-critical and silicon-non-critical paths. These paths were activated by one or more test patterns and did not show up as one of the 1452 silicon critical paths. To formulate the question, we let $S_{critical}$ be the set of the 158 silicon-critical paths and $S_{non-critical}$ be the set of the 12,248 STA critical and silicon-non-critical paths. Notice that with this formulation, all the uncertainties in the path lists as discussed in Section 4.1 are removed.

5.1 Feature generation

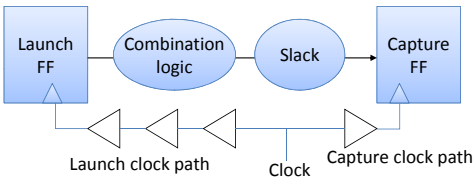

Figure 13. A path view for feature definition

Figure 13 illustrates the structural view of a path. Based on this view, features can be defined to describe various components of a path, including the launch FF, capture FF, the combinational logic path, the launch clock path, and the capture clock path.

In this work, we utilized seven categories of features:

- Basic information (three features): for example, whether the path is a half-cycle path, transition directions on NMOS and PMOS devices, etc.
- Timing statistics (19 features): delay information from STA timing report.
- Usage of Vt devices (16 features): counts of various Vt devices used on the path, i.e. high, low, regular, etc.
- Cell type (30 features): features of the usage of various important cell types.
- RC related (13 features): features capturing the load information on the path.
- MUX related (4 features): special features focusing on the usage of some MUX cells.
- Location (7 features): describing location of a path.

The total number of features is 92. However, note that some of these features were created after multiple iterations of the data mining process. Among the 92 features, 16 features were identified early in the mining process to be highly correlated to other features. This left with 76 features.

5.2 Filtering features

In each run, we ranked features based on the chi-square test ranking method and the linear SVM ranking method, and compared the results. During multiple iterations of the mining process, 33 features were deemed irrelevant or redundant and discarded in the study.

5.3 Rule learning and results

Based on the 43 remaining features, we ran sub-group discovery rule learning and discovered a number of 2-order rules. Table 1 summarizes the top five rules ordered by their coverage. For example, 68 paths in $S_{critical}$ and 1 path in $S_{non-critical}$ satisfy rule #1. Hence, for rule #1 the coverage is $\frac{68}{158} = 43\%$ and the accuracy is $\frac{68}{69} = 98.55\%$.

Rule#	Rule	$S_{critical}$	$S_{non-critical}$
#1	CMAC $\in [8,14] \wedge$ CCT $\in [6,7]$	68	1
#2	CMAC $\in [8,14] \wedge$ PS $\in [247,\infty]$	47	0
#3	VRC $\in [0.565,0.571] \wedge$ PS $\in [247,\infty]$	47	0
#4	CMAC $\in [8,14] \wedge$ VRC $\in [0.565,0.571]$	46	0
#5	CMAC $\in [8,14] \wedge$ VRD $\in [0.520,0.545]$	46	0

Table 1. Top five rules from rule learning

Figure 14 shows a color plot to visualize rule #1. The plot shows the coverage by the 12,248 STA paths in the space defined by the two features, CMAC and CCT. The 68

class +1 paths can be hardly seen; therefore, the bottom-left corner of the plot is magnified in Figure 15.

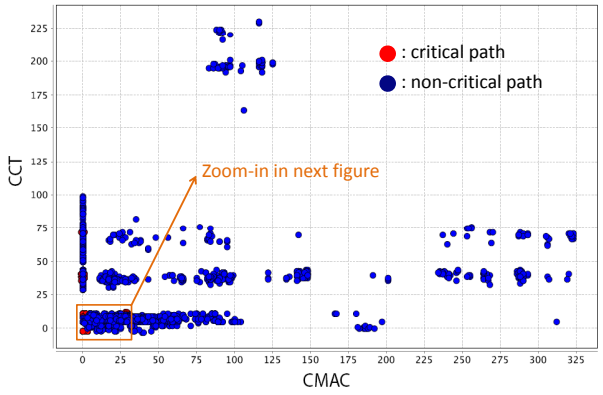

Figure 14. Visualization for Rule #1

Figure 15 shows the boxed area covered by Rule #1 in Figure 14. Because many dots overlap on top of one other, the area shows fewer than 68 red dots. By observing the plot, one may identify two additional boxes as potential rules. The two boxes are also drawn in the plot. After checking them, we found that rule 1a covered 8 paths in $S_{critical}$ and 6 paths in $S_{non-critical}$. It is not a good rule due to low accuracy. Rule 1b covered 4 paths in $S_{critical}$ and 0 path in $S_{non-critical}$. It was not reported as one of the top rules due to its low coverage.

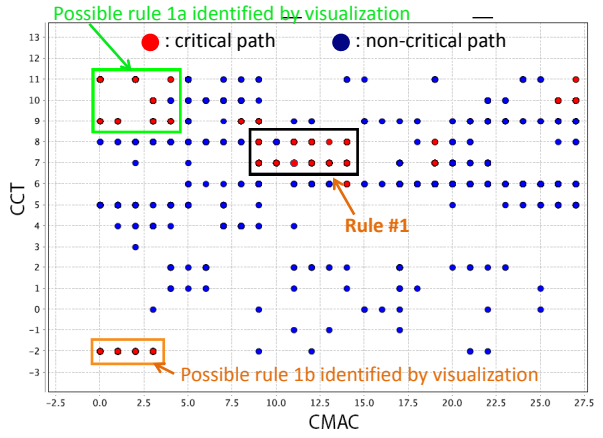

Figure 15. Zoom-in view from Figure 14

5.4 First Finding

It is important to note that a rule could be a behavioral reflection of a cause and not the cause itself. Hence, in most cases a rule should be treated as an indicator that guides further investigation for the cause. This is especially true when multiple rules are found to share a predicate and explain the same set of paths. In such a situation, it is likely that the multiple rules are reflecting the same cause. In the five rules listed above, we see that four of them share the same predicate "CMAC $\in [8,14]$." This motivated a careful examination on the CMAC feature and the four rules.

First, we found that all paths covered by rules #2,4,5 are also covered by rule #1. After examining the source of the CMAC feature, we discovered that all 68 paths covered by rule #1 utilized a particular custom cell type (call it Custom Cell 1) for implementing the capture FF. With further investigation, we discovered that this cell was also used in many of the STA critical paths. However, when the Custom Cell 1 was used on those STA paths, its CMAC values were always different from the CMAC values reported on the cell instances used on the 68 paths. More specifically, after the investigation, we could create a new rule (call it R1) with condition "capture FF = Custom Cell 1 $\wedge$ CMAC $\leq$ 14 and this rule would have covered all the 68 paths covered by rules #1,2,4,5 together. The reason that this rule was not found by the rule learning was because we did not include the usage of Custom Cell 1 as a feature.

5.5 Second Finding

After explaining the 68 paths, we repeated the analysis on the remaining 158-68=90 paths, i.e. $S_{critical}$ containing the 90 paths with $S_{non-critical}$ unchanged. Table 2 summarizes the top four rules we found.

Rule#	Rule	$S_{critical}$	$S_{non-critical}$
#1	CID $\in [102,148]\wedge$ TS $\in [378,404]$	26	0
#2	CBC $\in [0,\infty]\wedge$ TS $\in [378,404]$	25	0
#3	CBC $\in [0,\infty]\wedge$ CFD $\in [38,39]$	24	0
#4	CFD $\in [38,39]\wedge$ TS $\in [378,404]$	24	0

Table 2. Top four rules from rule learning

Notice that among the four rules, three share the predicate "TS $\in [378,404]$." Investigating the feature and rules #1,3,4 led us to discover that 25 paths covered by rules #1,2 all utilized another particular custom cell type (call it Custom Cell 2) to implement their capture FF. Although CMAC feature did not appear in any of the top four rules, based on the first finding above, we discovered another rule (call it R2) with condition "capture FF = Custom Cell 2 $\wedge$ CMAC $\leq$ 18. We discovered that this new rule could cover the 25 paths covered by rules #1,2 with no inclusion of any path from $S_{non-critical}$. Notice that the predicate on *CMAC* in R2 is different from that in R1.

After we found that rule R1 could explain 68 paths and rule R2 could explain 25 paths, we suspected the trend might continue. Among the 158 paths, there were 118 paths each utilizing a particular custom cell to implement its capture FF. There were totally six types of these custom cells. Rules R1 and R2 accounted for two types. There were four other types remaining. Hence, we created four other rules

R3,R4,R5, and R6 similar to R1,R2 to account for the remaining four types of custom cell.

Figure 16 visualizes the six rules in a two-dimensional plot. We see that for rules R1-R5 (involving cell types 1-5), each can separate the two classes of paths nicely with a proper choice of the cut-off value for CMAC feature. This value is 14 for R1 and 18 for R2 as mentioned above. Rules R3-R5 further explained 22 paths. Hence, in total, $68 + 25 + 22 = 115$ paths were explained with rules R1-R5. This left with $158 - 115 = 43$ paths unexplained.

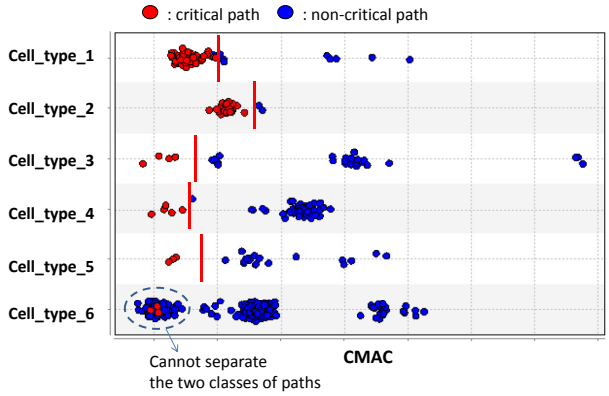

Figure 16. Custom cell type vs. CMAC feature

From Figure 16, we see that rule learning could not find a single predicate based on CMAC feature for all 115 paths because different cut-off points were required for different types of custom cell. Because custom cell types were not included as features, rule learning could not find rules R1-R5 directly. However, rule learning discovered other rules as good indicators that led to the finding of R1-R5.

5.6 Other Results

For the remaining 43 paths, we repeated the analysis and found no 2-order rules with good coverage and accuracy. Multiple 3-order rules were found with 100% accuracy. Those rules covered 16 of the 43 paths. At the end, 27 paths could not be explained through any simple rules. This indicated that the feature set was not enough to encode the the reasons behind those paths.

Recall that among the 169 silicon critical paths, 11 were in the STA path list. We also ran the experiment with $S_{critical}$ being the set of the 11 paths. However, we did not find any simple rules with good accuracy.

It is interesting to note that we repeated all the experiments discussed above by changing the set $S_{non-critical}$ to be the set of the 21,589 STA critical paths (see discussion in Section 4.1 about this path list). The findings were similar to those discussed above.

6 Conclusion

This work presents a data mining methodology for analyzing timing critical paths observed on silicon samples. We applied this methodology to analyze timing critical paths observed on 15 4-core microprocessor units. Among the 169 silicon critical paths to be analyzed, 115 of them could be reasonably explained through simple design rules R1-R5. These rules could easily be translated into design actions for fixing the 115 paths. Although the data mining process did not find those R1-R5 rules directly, other rules were identified, providing good guidance to find rules R1-R5. Meaningful rules such as R1-R5 would be difficult to find without the data mining methodology.

Although the data mining methodology is presented for analyzing timing critical paths, it is general enough to be applied in analyzing many other types of data. To enable the analysis, all it needs are two classes of samples and a set of features to describe the characteristics of interest on those samples. In future work, we plan to explore other applications of the proposed data mining methodology.

References

[1] K. Killpack, C. Kashyap, and E. Chiprout. "Silicon Speedpath Measurement and Feedback into EDA flows," *Proc. Design Automation Conference*, 2007.

[2] P. Bastani, et al., "Speedpath Prediction Based on Learning from a Small Set of Examples," DAC, 2008.

[3] N. Callegari et al., Feature based similarity search with application to speedpath analysis. ITC, 2009

[4] Li-C. Wang et al., "Design-Silicon Timing Correlation — A Data Mining Perspective," DAC, 2007, pp. 390-395.

[5] P. Bastani et al., "Diagnosis of design-silicon timing mismatch with feature encoding and importance ranking - the methodology explained," ITC, 2008.

[6] Li-C. Wang, "Data Learning Based Diagnosis," *ASP-DAC*, 2010.

[7] Peter Clark and Tim Niblett. "The CN2 induction algorithm", *Machine Learning*, Vol. 3, No. 4, 1989, pp. 261-283.

[8] N. Lavrač et al., Rule induction for subgroup discovery with CN2-SD, *Journal of Machine Learning Research*, Vol 5, 2004.

[9] N. Callegari et al., "Speedpath Analysis Based on Hypothesis Pruning and Ranking," DAC, 2009.

[10] Dragoljub (Gagi) Drmanac, et al., "Minimizing Outlier Delay Test Cost in the Presence of Systematic Variability," *International Test Conference*, 2009.

[11] Janine Chen, et al., "Data Learning techniques and Methodology for Fmax Prediction," *ITC*, 2009.

[12] Leendert M. Huisman, Maroun Kassab, Leah Pastel, "Data Mining Integrated Circuit Fails with Fail Commonalities," *ITC*, 2004.

[13] Pietro Babighian, Gila Kamhi, Moshe Vardi, "PowerQuest: Trace Driven Data Mining for Power Optimization," *DATE*, 2007.

[14] Sholom M. Weiss, et. al, "Rule-based data mining for yield improvement in semiconductor manufacturing," *Applied Intelligence*, 2009.

[15] N. Callegari et al., Classification Rule Learning using Subgroup Discovery of Cross-Domain Attributes Responsible for Design-Silicon Mismatch, DAC, 2010, pp. 374-379.

[16] D. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[17] B. Schlkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2001.

[18] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[19] Grzymala-Busse, J. W. "Discretization of numerical attributes", In *Handbook of Data Mining and Knowledge Discovery*, ed. by W. Klsgen and J. Zytkow, Oxford University Press, 2002, pp. 218225.