A HILBERT-CURVE BASED DELAY FAULT CHARACERIZATION FRAMEWORK FOR FPGAS

Wenjuan ZHANG

NATIONAL UNIVERSITY OF SINGAPORE

2011

A HILBERT-CURVE BASED DELAY FAULT CHARACERIZATION FRAMEWORK FOR FPGAS

WENJUAN ZHANG

(B.Eng., XI' AN JIAOTONG UNIVERISTY)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPT OF ELECTRICAL AND COMPUTER ENGINEERING

FACULTY OF ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2011

ABSTRACT

With the increasing process variations in advanced technologies, delay defects are gaining a larger impact on Field Programmable Gate Array (FPGA) timing yield. If the delay defect areas can be quickly and accurately located, FPGA timing yield can be improved by avoiding them. Conventional delay testing methods do not take into account the spatial information of variability-induced delay faults, thus cannot accurately locate the delay defects to a well restricted area. Based on the superb locality preserving feature of space-filling curves, we propose a method to locate delay faults and generate a delay variation map (DVM) with scalable resolutions in this thesis. The method uses Hilbert curves to guide the test configurations of FPGAs. It is able to work on FPGAs with regular or arbitrary dimensions. Compared with normal test approaches, our method achieved around 60% increase in delay faults locating resolution.

Keywords:

FPGA, Delay Fault, Delay Fault Characterization, Space-Filling Curves, Hilbert Curve, Timing Yield.

ACKNOWLEDGMENTS

I would like to express my greatest gratitude to my advisor Dr. Ha Yajun for his tremendous help and guidance over the years. Thanks to his insightful directions and constant motivation during the course of my research, I have learnt a great deal and not just about FPGAs. Working with him has been an invaluable experience that I will cherish forever.

I have benefited greatly from many colleagues who contributed to this work. I owe thanks to Chen Xiaolei, Yu Heng, Shakith Devinda Fernando, Loke Wei Ting, Akash Kumar, Wei Ying, Tian Xiaohua, and many more. This work would not have gotten far were it not for their suggestions and observations.

Finally, I am grateful to my parents for standing by me and always being supportive through the ups and downs.

TABLE OF CONTENTs

LIST OF FIGURES

LIST OF TABLES

SUMMARY

Advanced technologies have enabled the increasingly higher density of FPGAs. At the same time, they have also brought forth new challenges such as increased impacts of manufacturing defects and process variations. These variations cause greater uncertainties in circuit timing performance, making it difficult to ensure design quality [1]. The delay of a logic block or a wire segment in FPGAs can vary in a much larger range. Study has shown that variability may cause up to 22% performance penalty in FPGAs [2]. Apart from process variations and manufacturing defects, high performance clocking strategy is also a source of product failure as it makes delay defects more prominent. To guarantee yield, delay defects need to be properly characterized [3].

Efficient testing methods are needed to quickly and accurately detect and locate the delay defect areas. Delay faults are tested by configuring an FPGA into test circuits whose input signals are rising and falling transitions. The results of delay fault testing are used to determine the timing performance of different part of FPGA resources.

Numerous methodologies have been developed to facilitate the FPGA delay fault testing. In [4], the authors proposed a procedure to generate efficient FPGA test configurations. A method to test delay faults in the LUT network of FPGAs by linking them together as a test array was presented in [5]. Application-dependent delay testing was proposed in [6] and [7], which only targets at a subset of the resources. While most of the methods improve the test efficiency for delay faults, the cumulative effect

of delay faults induced by variability remains overlooked. Hence, for a circuit with spatially correlated variations, the affected logic blocks may not be identified correctly as the delay error on each logic block or wire may not be big enough to be detected. These defects will impair the circuit performance if a critical path of the circuit covers a large number of affected logic blocks and wires. Such delay defects may not be located correctly or with a good resolution by the previous approaches, as the spatial correlation of delay variations is not considered.

Based on the superb locality preserving feature of space-filling curves, we propose a method which can quickly and accurately detect the region affected by delay faults in FPGAs. The method generates FPGA test paths based on Hilbert curves, one of the classical space-filling curves. Depending on the number of test points inserted to the curve, different levels of locating resolution can be achieved. Finally, a delay variation map (DVM) will be generated for the target FPGA. The DVM partitions the FPGA into regions with different delay variation levels. The size of the region is scalable depending on the target resolution. Compared with normal test paths, our method significantly improves the speed and accuracy at detecting areas affected by delay defects.

C H A P T E R . 1
I N T R O D U C T I O N

Advanced technologies have enabled the increasingly higher density of larger FPGAs. With reducing transistor sizes, designers are able to pack more functionality onto a single die while increasing the operating frequency. At the same time, decreased dimensions have also brought new challenges such as increased impacts of processes variations. These variations cause increasing uncertainties in design timing performance, making devices more prone to delay faults.

We introduce the whole thesis once over lightly in the remainder of this chapter and concisely describe the application (Section 1.1) contexts, problem definition (Section 1.2), and solution approaches (Section 1.3) of our research. Finally, we conclude this chapter with the main contribution statement in Section 1.4 and the further organization of this thesis in Section 1.5.

1.1 FPGA Delay Fault Characterization

Imperfections of the equipment or the inaccuracies in the fabrication process of VLSI chips create manufacturing defects such as physical flaws, contact open, metallization open and resistive open. Manufacturing defects may cause the devices to fail or worsen their performance. Before the IC chips are shipped to the customers, the manufactures are responsible to perform tests on the chips to ensure that the devices meet their specifications.

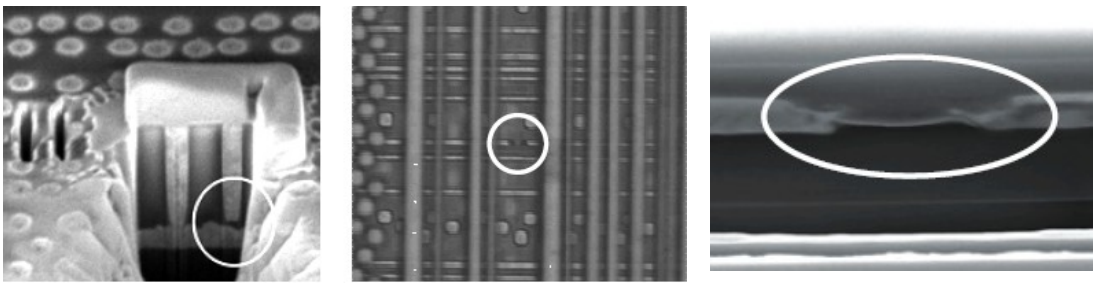


Figure 1.1 Examples of manufacturing defects.

In addition, increased variations in very deep submicron semiconductor processes will result in device parameters with broader distributions. Figure 1.2 shows the power and frequency plot of a batch of Intel processors. The plot clearly shows the spread of power and frequency values.

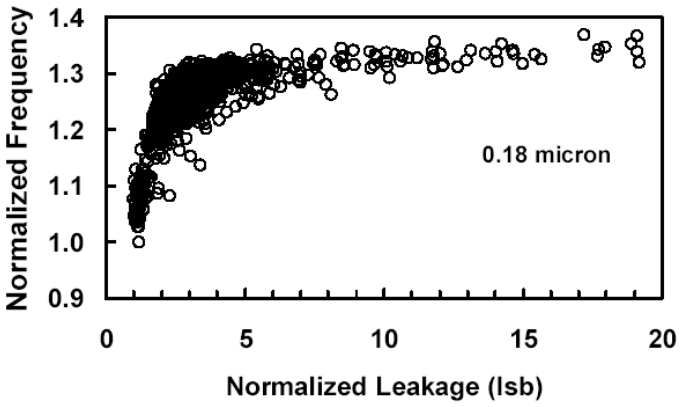


Figure 1.2 The power and frequency plot of a batch of Intel processors

A Field Programmable Gate Array (FPGA) is a state-of-the-art semiconductor device with regularly-structured logic arrays interconnected by a routing network. With the shrinking resistor sizes and aggressive clocking strategy, FPGAs are much more prone to defects in manufacturing process. Device parameters are affected by process

variations, resulting in increased unpredictability in device performance. The delay of a logic block or wire segments in FPGAs can vary in a much larger range in a faulty case. Study has shown that variability may cause up to 22% performance penalty in FPGAs [2].

Increased operating frequencies also have an impact on the timing yield of FPGA. Small delay defects that will not fail a device when operating at lower frequencies will cause timing violations under higher frequencies.

To ensure that manufactured FPGA performs as it should be at its operating frequency, delay fault testing is applied to find and locate delay defects on the FPGA chip.

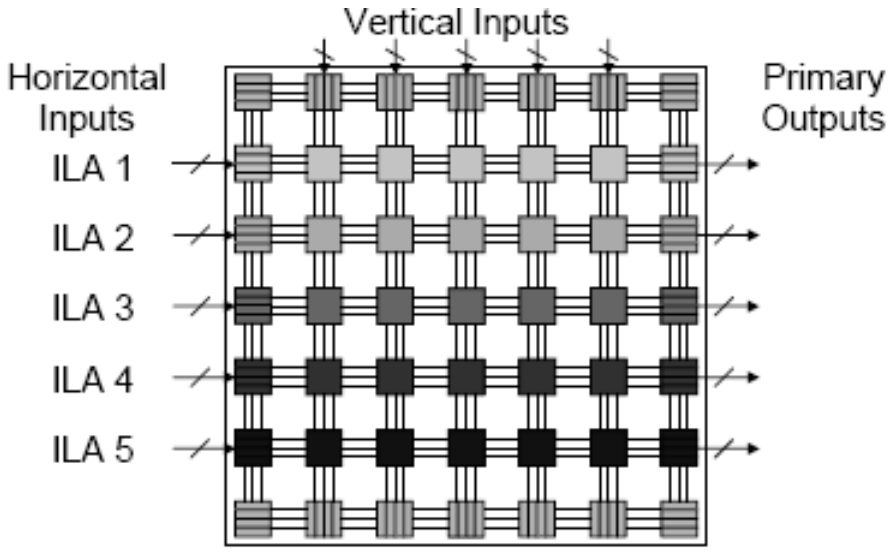


Figure 1.3 Delay testing of FPGAs

Delay fault testing of FPGA is commonly carried out by applying transitions to one end of a FPGA test path and observe the time taken for the transition to the other end. The measured delay value is then compared against the delay of fault-free test path to

determine the existence and severity of delay defects. The standard FPGA delay testing involves three steps:

1. Configure FPGA with a testing design
2. Apply rising/falling transistions
3. Analyze test response

In order to improve the device yield, an efficient delay fault testing method is needed to quickly and accurately detect and locate the delay defect area. The testing method needs to consider the effects of process variations-induced delay defects on device timing performance.

1.2 Problem Definition

Our work presented in this thesis is inspired from the concerns at the increasing impact of process variations in fabricated FPGA devices under deep sub-micron technologies. With the continuous scaling, it becomes harder to control manufactured parameters, this result in larger percentage of parameter variation against nominal values. Moreover, process variations tend to have location-related correlations that are called spatial correlations. Delay defects induced by such correlations are dependent to each other.

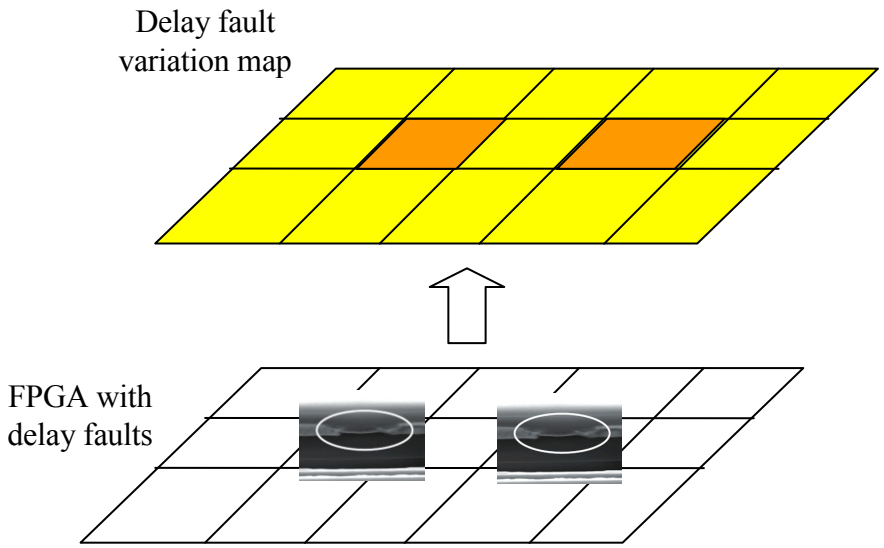


Figure 1.4 Delay fault variation map for an FPGA

Most of existing approaches in delay fault testing uses path-based single-transition propagations to determine the delay of the FPGA device under test (DUT). They usually partition the set of FPGA resources under test into test paths or test arrays, and measure the delay for each of them accordingly. The paths are commonly selected in a straight-forward manner, only taking into account of maximum coverage and minimum testing time, but not the possible spatial correlations between delay defects on the chip.

As the impact of process variations-induced small delay defects continues to increase, traditional delay testing approaches cannot accurately determine and locate resources on FPGA that are affected, as they only use the total accumulated propagation delay along the test paths to find delay faults, without considering the spatial relationships of the delay defects. Thus, the relatively “slower” areas on FPGAs caused by variations may not be accurate mapped by these methods.

Thus, our problem is to develop a delay fault characterization algorithm that accurately locates the "slow" FPGA resources under delay defects, and is able to maximize accumulated small delay errors caused by process variations.

1.3 Solution Approaches

Based on the superb locality preserving feature of space-filling curves, we develop a method which can quickly and accurately detect the region affected by delay faults in FPGAs. The method generates FPGA test paths based on Hilbert curves, one of the classical space-filling curves. Depending on the number of test points inserted to the curve, different levels of locating resolution can be achieved. Finally, a delay variation map (DVM) will be generated for the target FPGA. The DVM partitions the FPGA into regions with different delay variation levels. The size of the region is scalable depending on the target resolution. Compared with normal curves, our method significantly improves the speed and accuracy at detecting areas affected by delay defects.

1.4 Thesis Contributions

Based on the superb locality preserving feature of space-filling curves, we develop a FPGA delay fault characterization method which can quickly and accurately detect the region affected by delay faults, as outlined in the problem definition above. Our main contributions are as follows:

1. We presented a test path generation algorithm based on the geometric tool of space-filling curves. With the superb locality-preserving ability of space-filling curves, the generated test paths are able to capture the accumulated delay faults caused by spatially-correlated variations on FPGA chip. We use the improved

form of space-filling curves which is able to cover area with arbitrary rectangle dimensions, as opposed to classic curves which is To the best of our knowledge, this application of special geometric curves to the problem of FPGA testing is novel, as the curves are commonly used only for image processing or database indexing.

2. Secondly, we developed a test framework based on the path generation method presented above. Our method partitions the FPGA-under-test into suitable test regions, each covered by a test path generated from space-filling curves. We then present the criterion to evaluate test results and how the slower regions are located. Depending on the number of test points inserted to the curve, different levels of locating resolution can be achieved. Compared with normal curves, our method demonstrates significantly better speed and accuracy at detecting areas affected by delay faults.
3. In order to have a complete test framework, we then present another algorithm for a different family of FPGAs. Our original methodology is only able to test FPGA with rectangle dimensions as the classic space-filling curves are designed for a regular continuous space. However, state-of-the-art FPGA devices commonly have embedded on-chip hard IP cores. The shape of testable resources of these FPGAs is no longer a perfect rectangle, but a rectangle with obstacles (black boxes) in it. To tackle delay fault locating for such devices, we develop a drastically modified version of our original algorithm which incorporate the Hamilton curves as guidance for test path allocation. With the modified algorithm, our methodology can test for both regular and irregular shaped FPGAs. We then run experiments to show that the modified algorithm has similar run-time and accuracy as the original algorithm.

1.5 Thesis Organization

The rest of this thesis is organized as follows.

Chapter 2 introduces the models and existing detection methods for delay defects in FPGA, presents the theoretical background of the geometric tools we applied to the problem, and gives an overview of our main contributions. We also briefly introduce existing delay testing methods and explain what they are lacking when applied to state-of-the-art FPGAs under VDSM technologies.

Chapter 3 introduces the affine arithmetic timing model and analysis approach used in our framework.

Chapter 4 describes our methodology applied to delay fault characterization of FPGA. Experimental results are given and compared with other possible test methods to show the superior delay fault locating ability of our algorithm.

Finally, **Chapter 5** concludes the thesis. Our contributions are summarized and possible future directions given.

CHAPTER.2
BACKGROUND AND RELATED WORK

To better understand and appreciate our framework, we have listed the various background and related works in this chapter.

In Section 2.1, we introduce the basic FPGA architecture we are considering, the various sources of delay faults in FPGAs, the delay fault models, and the existing approaches of FPGA delay testing. In Section 2.2, we present the general space-filling curves and show the good properties of this class of curves. In Section 2.3, we present the Hilbert type of space-filling curves, which is the right type of space-filling curve that we are going to use in our framework. In Section 2.4, we introduce the differences between our framework and existing approaches.

2.1 Delay Faults in FPGAs

From Chapter 1, we have established that delay fault testing is an essential step to ensure FPGA yield. Delay fault models are needed to properly represent the effect of delay fault. In this section, we introduce the basics of FPGA delay testing and define the models we use to evaluate delay faults.

2.1.1 FPGA Architecture

The Field Programmable Gate Array (FPGA) is a digital integrated circuit consisting of a two-dimensional array of configurable logic blocks (CLBs) and a programmable interconnect network, as shown in Fig 2.1. The logic array is surrounded by input/output

blocks (IOB) that are also programmable [8]. Each of the CLBs is formed by look-up tables, registers and multipliers. Fig 2.2 shows the structure of a FPGA CLB.

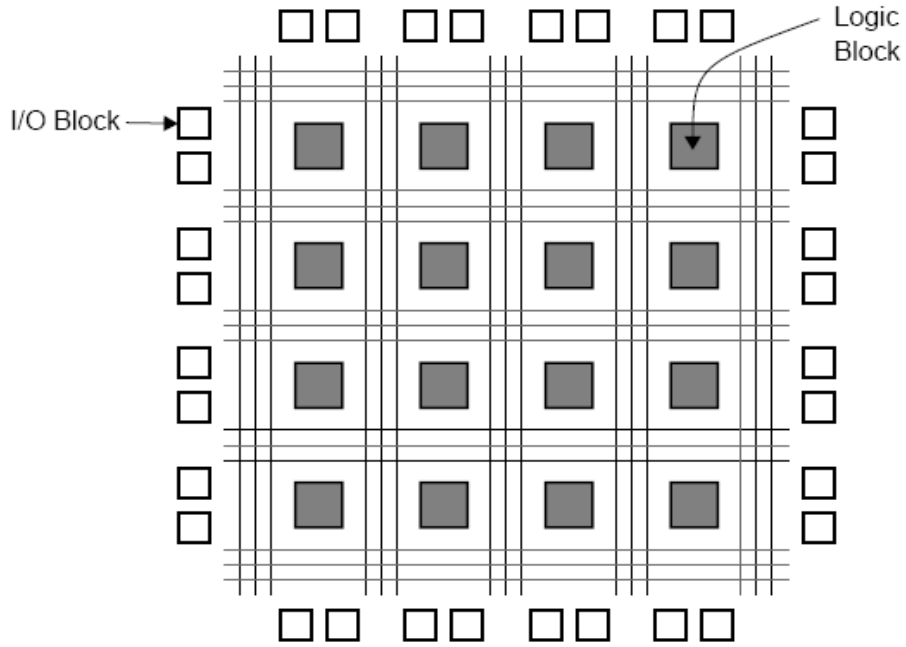


Fig.2.1 General FPGA Architecture

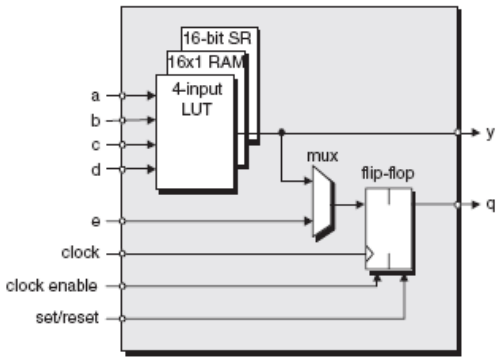


Fig.2.2 FPGA CLB Architecture

FPGAs may be categorized according to process technologies, including antifuse, EPROM, Flash, and Static RAM (SRAM). The majority of commercial FPGAs are SRAM-based devices. They use SRAM configuration cells to store configuration bits for programmable logic, interconnect, and IO blocks. SRAM-based FPGAs have higher density compared with other types of FPGAs, but require an external non-volatile

memory to hold configuration information. Our discussions are focused on SRAM-based FPGAs.

Modern FPGAs often incorporate embedded IP blocks with specific functions, making them more similar to system-on-chip (SoC) [9]. These IP blocks have different functions and complexity, ranging from simple arithmetic unit all the way up to embedded general-purpose microprocessor. Based on the way they are implemented, IP cores are categorized into hard cores and soft cores.

Hard IP cores are predefined and prefabricated blocks with fixed functionality. They may be built within the main fabric of FPGA or as a separate strip to the side of main fabric, as illustrated in Fig 2.3. Xilinx's Virtex-II Pro and Virtex-4 405 PowerPC core are examples of hard cores.

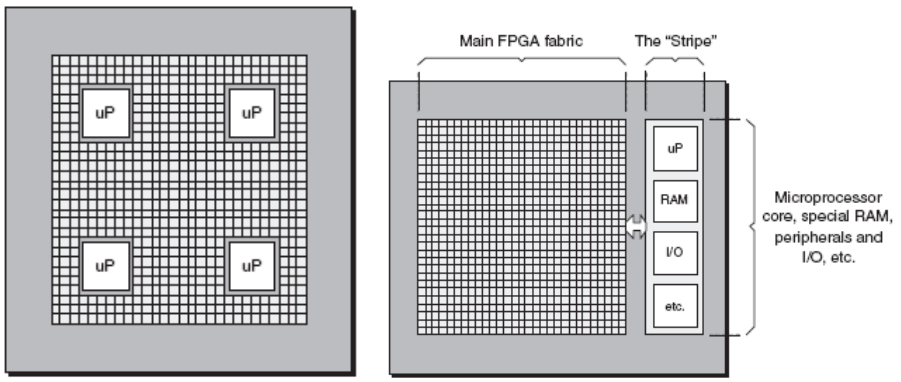


Fig 2.3 FPGA with embedded IP cores built inside/outside main fabric

As opposed to hard cores, soft cores are implemented using the configurable resources on the FPGA chip. They are delivered either in the form of RTL netlist or as placed and routed mapping of CLBs. Soft cores have poorer performance compared to hard cores, but they have the advantage of flexibility and lower cost.

A FPGA needs to be configured with a logic design to perform its function. The design flow of LUT-based FPGA could be briefly summarized as follows:

1. SYNTHESIS the design to gate-level netlist
2. MAP the design to available LUTs on FPGA
3. PACK the LUTs into CLBs
4. PLACE and ROUTE the CLBs to obtain a fully routed physical netlist
5. VERIFICATION of timing, power,etc.

2.1.2 Sources of FPGA Delay Faults

In integrated circuits, a *defect* refers to a physical imperfection or manufacturing flaw that causes a *fault* in the device. Fault is the logical effect of a defect that can lead to a *failure* [10]. Manufacturers need to test FPGAs for defects before they are shipped to ensure that they function correctly.

A *delay fault* is an excessive delay in wire or transistor that causes the total propagation delay to go beyond the given upper-bound. FPGAs with delay defect functions correctly under slow clock, but fails when operated at normal or high speed.

Defects in integrated circuits can be broadly categorized into two types, *bridge* and *open* defects. A bridge or short defect is caused by connection between circuit nodes that were intended to be disconnected. Bridge defects include ohmic bridge, gate oxide short, and transistor punch-through [11]. Ohmic bridge is caused by metal shorting two or more interconnects (Fig 2.4(a)). A gate oxide short is a break in the CMOS transistor oxide that connects the channel or the gate to the drain/source underneath (Fig 2.4(b)).

Transistor punch-through is a short from source to drain that occurs when the drain depletion region expands the whole channel length.

Fig 2.4 Bridge defects in the circuit [11]

Open circuit defects are unintentional breaks or electrical discontinuities in integrated circuits. Causes of open defects can be cracked metal, errors in etching, or faulty mask and fabrication; and they can occur in the transistor or in the interconnects (vias and contacts), as shown in Fig 2.5.

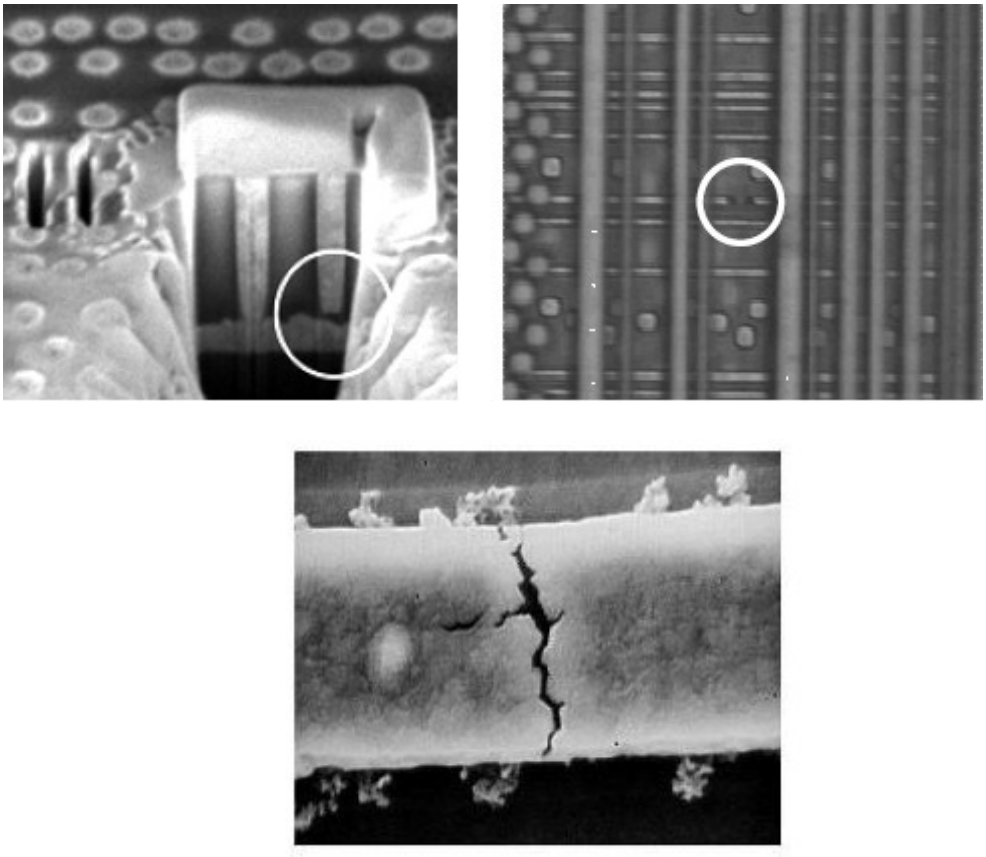


Fig 2.5 Open defects in the circuit [12]

An open defect is called a *resistive open defect* (Fig 2.6) if it is only partially open, meaning a conducting path still remains between the nodes, but with an extra defective load R_{DEF} (Fig 2.7). A resistive open is equivalent to a complete open defect when $R_{DEF} = \infty$.

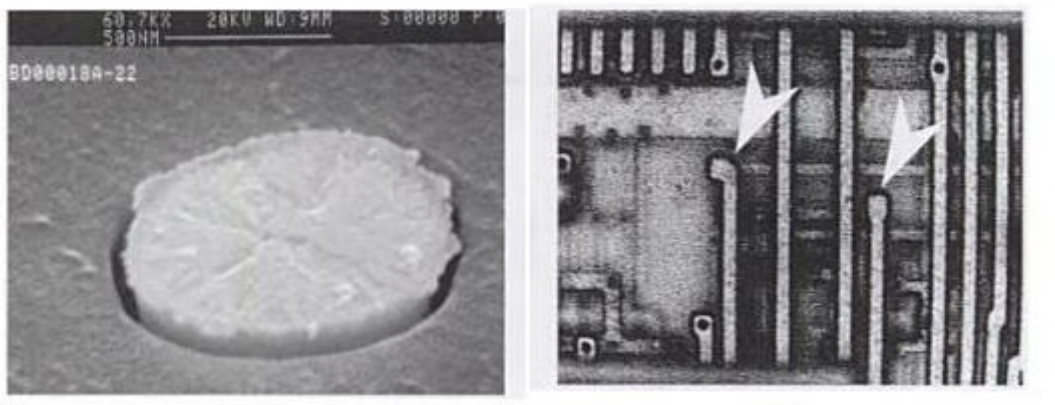


Fig 2.6 Resistive open (a) between via metal and liner, (b) caused by missing vias [11]

Resistive open defect is the source of *delay fault* in the circuit. Delay fault leads to delay defect if it is severe enough to cause a timing failure in the circuit, that is, the extra delay is larger than the slack (the difference between the required time and the arrival time) of a path (see Fig 2.7). As resistive open defects can occur either in wires or transistors, both the routing and logic networks of the FPGAs are affected by them.

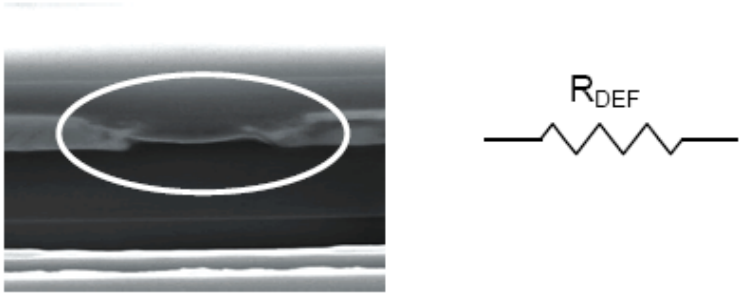


Fig 2.7. A path with delay fault.

2.1.3 Delay Fault Models

In circuits, the time taken by signals to propagate from one memory element to the other, or between a memory element and the input or output of the circuit is called delay. If the signal doesn't arrive at the destination due to a defect, the circuit cannot operate at the designed frequency. The effect of delay defects on signal delay in circuit is model by delay fault models.

If a resistive open defect causes the signal delay to increase so much such that the total combinational delay exceeds the clock period, the circuit will fail when operating at normal frequency, and is said to have a delay fault. Defective circuits are identified by measuring delay values in all parts of the circuits.

The most commonly-used delay fault models include transition delay fault model, gate delay fault model, line delay fault model, path delay fault model and segment delay fault model [13]. Transition, gate and line delay fault models represent defect located at a single gate. Path and segment delay fault models characterize delay defect that expand several gates. However, the number of paths that need to be tested increases exponentially with the number of gates, making exhaustive test impossible. Thus, only the path with the longest propagation delay is tested. This approach is usually applied to ASICs (Application Specific Integrated Circuits). In FPGAs, testing only the longest path is no longer sufficient, as a short line might fail in a configuration if it has delay defect and determines the clock period [14].

The segment delay fault model [15] is a trade-off between path and gate delay fault models. It takes into consideration both slow-to-rise and slow-to-fall delay faults in

FPGA segments. A slow-to-rise delay fault is said to occur if a low-to-high transition fed to the beginning of a segment and does not reach the end of the segment after a given period of time. A slow-to-fall delay fault is be defined similarly.

2.1.4 Impacts of Process Variations on Delay Faults

Delay is the function of capacitance and resistance; as a result, it is subject to variations in circuit parameters. To characterize delay variations accurately, we need a model which takes into consideration the sources and scale of variability.

Variations in semiconductor circuit manufacturing can cause the circuit parameters to deviate from their nominal values. Examples of variability include changes in environmental factors such as supply voltage and temperature. Physical imperfections can also introduce variability into the fabrication process.

In the past, only the variations between lots, wafers and dice are considered. However, as the technology nodes move further in VDSM, within-die variations will have an increasingly significant impact on circuit characteristics. The within-die variations can be classified as systematic or stochastic variations [16]. Systematic variation sources consist of imperfections in fabrication process, such as mask errors, lithographic off-axis focusing faults, and reticle stepper misalignment errors. Stochastic variation sources include wafer unevenness, non-uniformity in resist thickness, and unsteadiness in lithography [17].

Stochastic variations are not due to imperfections in the fabrication process, but to granularity of material characteristics at VDSM. Such variation sources cannot be

rectified by improvement in processes, they need to be compensated by new device technologies and design technique.

In measurement and experiment of semiconductor circuits, it is important to know the magnitude of the estimated characteristics, in order to determine the required measurement sensitivity. In [18], it was demonstrated that delay is most affected by effective transistor gate length, L_{eff}. The desired total (3σ) tolerance of Leff is $\pm 10\%$, according to the SIA roadmap. However, as the technology moves beyond the 65nm node and to the 45nm node, the relative impact of within-die variations are expected to rise, which requires more control for inter-die variation tolerance.

2.1.5 Existing FPGA Delay Fault Testing Methods

We start by introducing the basics of IC and FPGA delay testing methods, followed by a brief summary of existing FPGA delay testing methods.

Testing for Integrated Circuit

The process of testing is to apply a known input stimulus to the unit-under-test (UUT), in a known state, and evaluate the output response with the expected response [13]. The main criteria for testing are the cost of the test development, the quality of the test set, and the cost for test application.

Testing can be a functional test or a structural test. Functional testing is also called a "design verification test", which is used to verify the UUT behaves as it is supposed to.

Structural testing is performed to verify the topology of the manufactured chip, that is to say, testing for physical defects caused by the manufacturing process. It can also test for delay fault by applying a delay fault model in place of logic model.

Delay Fault Testing

Delay fault testing has become an important part of VLSI testing process in today's deep sub-micron designs, which is more susceptible to process variations, manufacturing defects, and noise. The purpose of delay testing is to discover timing defects and ensure that the design meets the performance specifications. The timing of the circuit has to be carefully evaluated to avoid such errors in the function of the circuit. However, testing of delay defects is significantly different from logic defects, as delay fault testing is two-dimensional, with both timing and logic domains, compared to logic testing which is defined on logic domain only. The actual delay sizes of delay faults are also harder to predict. Due to the influence of variations, the size of delay will no longer be a fixed value, but a random variable instead. These two problems render the traditional testing method insufficient for today's test scenario.

The most well-known technique of delay testing is the oscillation test method. It is concerned with sensitizing a critical path and then test for delay faults. Critical paths are those paths that have the longest propagation delay from primary input to primary output. Therefore, the critical path is the most likely path for a delay fault to cause the circuit to malfunction. To sensitive a path all off-path logic values inputs must be set to non-controlling values [13]

As is mentioned before, path-selection greatly affects the effectiveness of the delay tests. It has been shown that with delay as random variables, the traditional method of critical path selection is not adequate and a new criterion is needed. In [19], a method of delay fault diagnosis based on statistical timing analysis is proposed, which models delay elements as random variables and calculate criticality of the paths by statistical evaluation. However, this method is designed for ASIC only and is not suitable for FPGAs.

FPGA Testing

FPGA testing is performed to ensure the integrity of the components and the interconnections, and that the manufactured board meets the customer requirements. The purpose of testing is to discover defects that may have been introduced during the fabrication process.

FPGA testing can be broadly classified into manufacturing test and user test. The former, performed as part of the manufacturing process, test components and interconnections in the array for faults, such as stuck-ats, shorts and opens. It is also called application-independent testing. Components may also be tested to determine their switching speeds. User tests are intended to detect FPGA faults that occur after a device is programmed for a specific application, thus it is also referred to as application-dependent testing. The faults of interest in this type of testing are only those that can affect the operation of the specific circuit. These consist of stuck-at faults, shorts, opens, and delay faults. Faults to be tested by user tests depend not only on the logic implemented by the circuit, but also on its placement and routing in the FPGA.

FPGA Delay Testing

The objective of FPGA delay testing is to detect delay faults and ensure that the design meets the given performance specifications. Delay faults are triggered and observed by propagating signal transitions through the circuit [13]. This involves application of two test values. The whole process is usually called two-value testing and is the foundation for all kinds of delay testing methodologies.

There are effective well-known methods for testing FPGA logic blocks [20][21]. Additionally, various techniques for testing FPGA routing resources have been proposed [22][23][24], addressing stuck-at, stuck open, and bridging faults. Testing for delay faults in FPGAs has been gaining more attention recently as it is suspected that 58 % of customer-returned semiconductor chips have open defects [24].

We briefly introduce two of the above mentioned proposed methodologies. The added fan-out method tests resistive open defects by adding an extra load or fan-out to the test path. The delay of the path is increased by the load. When the accumulated delay of the test path surpasses that limit, a timing error occurs. Another methodology is the oscillator loop method, which configures the device under test into numerous test arrays. Then a test signal is fed into the input of each test array, and the difference in their propagation delays is measured at the output of the test arrays. The method uses an on-chip oscillator to count the difference and detect a delay fault when the difference surpasses some predefined value.

2.2 Space-Filling Curves

A space-filling curve is a continuous scan that traverses every point in a given region exactly once. Space-filling curves are advantageous to applications which need to consider the spatial coherence of neighbouring points. Hilbert curves belong to the space-filling curves. Of all the space-filling curves, Hilbert curves have a high level of adjacency, meaning adjacent intervals are mapped onto adjacent squares with an edge in common. The next section introduces the Hilbert curve which forms the structural basis of our algorithm.

2.3 Hilbert-Type Space-Filling Curves

2.3.1 Definition of Hilbert Curves

In 1891 D. Hilbert discovered another space-filling curve. Whereas Peano's curve was defined purely analytically, Hilbert's approach was geometric.

A classical Hilbert type space-filling curve is defined as follows: Let $I = \{t \mid 0 \le t \le 1\}$ denote the unit interval and $Q = \{(x, y) \mid 0 \le x \le 1, 0 \le y \le 1\}$ the unit square, for each positive integer n, we partition the interval I into 4^n sub-intervals of length 4^{-n} and the square Q into 4^n sub-squares of side 2^{-n}. A one-to-one correspondence is constructed between the sub-intervals of I and the sub-squares of Q which satisfies the conditions of adjacency and nesting.

Adjacency Condition Adjacent subintervals correspond to adjacent subsquares (with an edge in common).

Nesting Condition If at the n-th partition, the subinterval I_{nk} corresponds to a subsquare Q_{nk} then at the (n+1)-st partition the 4 subintervals of I_{nk} must correspond to the 4 subsquares of Q_{nk} .

The original Hilbert curves can be generated by recursion [25]. The first steps to generate Hilbert curves are shown in Fig. 2.8.

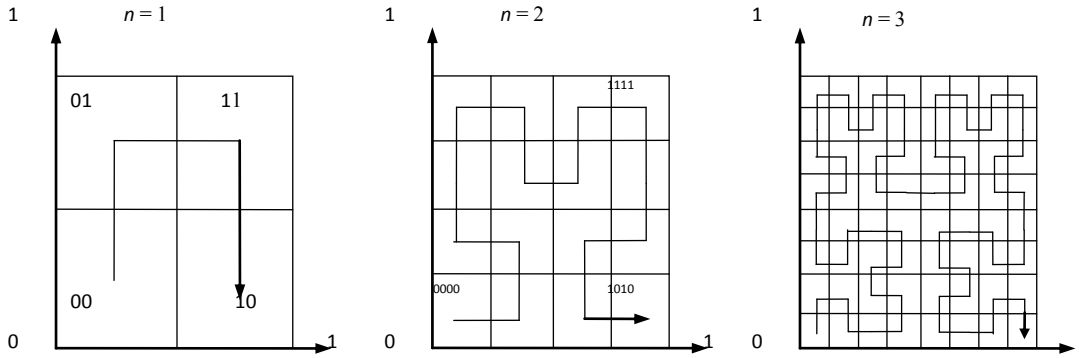


Figure 2.8: The first 3 stages in generating Hilbert curves.

The original Hilbert curve suffers from the problem that it can only handle regions of size $2^n \times 2^n$. To make the Hilbert curve more versatile, the pseudo-Hilbert curve has been developed, which fills rectangle regions with arbitrary dimensions. The pseudo-Hilbert curve covering a (40, 34) area is shown in Fig. 2.9.

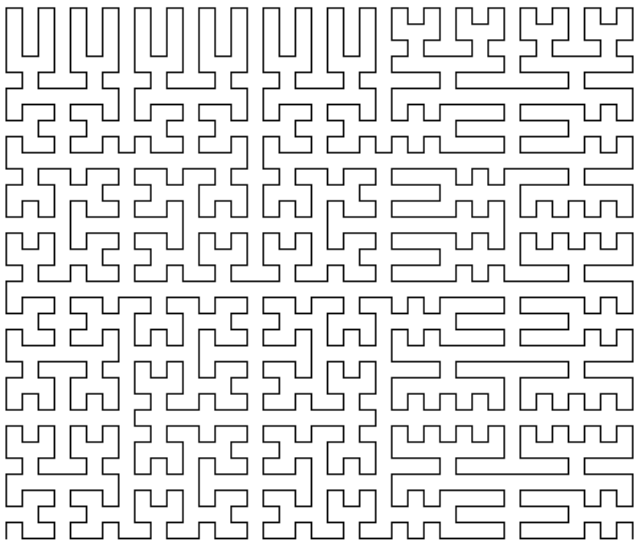


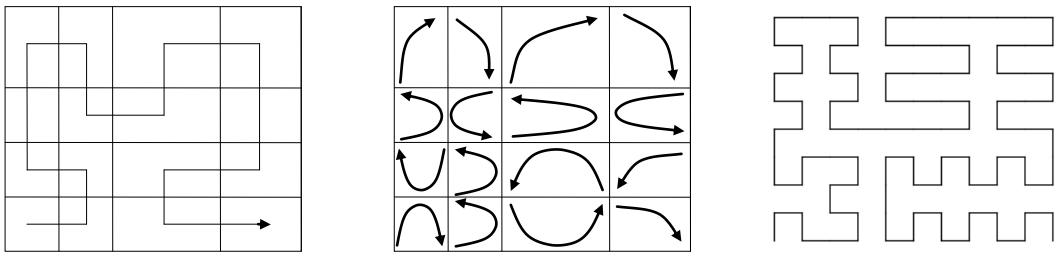
Figure 2.9: An example of pseudo-Hilbert curves.

2.3.2 Methods of Hilbert Curve Generation

The generation of a 2-D space filling curve of successive orders usually follows a recursive framework. The classic Hilbert Curve can be generated in this manner. It can be constructed from a basic unit shape as shown for $n = 1$ in Figure 2.10 The relative position and rotation of each unit shape is defined by its sequential position in the curve generation (see Figure 2.10). As the resolution of the curve increases, more unit shapes are required for its description, but the principle remains same as the original proposition of dividing each part into smaller parts.

Generation techniques have been developed for pseudo-Hilbert curves [26][27][28]. While they use different methods to generate the address of each point, the basic procedure remains the same. At first, the rectangle is split into a set of sub-regions or sub-blocks based on its dimensions. The points within the same sub-region have the same upper address. After that, each sub-region is scanned to determine the lower address of its points. Lastly, the upper and lower parts of the addresses are combined to

obtain the complete address for each point in the matrix. Our algorithm is based on the look-up-table-based, non-recursive approach presented by [27], as it best preserves the adjacency characteristic of Hilbert curve. We modified the algorithm focusing on generating curves for FPGA circuits.



(a) (b) (c)

Figure 2.10: Procedure of pseudo-Hilbert curve generation

2.4 Differences Between Our and Existing Approaches

Numerous methodologies have been previously developed to facilitate the testing of FPGA delay faults. [4] proposed an procedure to generate efficient FPGA test configurations. [5] presented a method to test delay faults in the LUT network of FPGAs by linking them together as a test array. Application-dependent delay testing was presented in [6] and [7], which only targets at a subset of the resources. While most of the methods improve the test efficiency for delay faults, the cumulative effect of delay faults induced by variability is often overlooked. Hence, for a circuit with spatially correlated variations, the affected logic blocks may not be identified correctly as the delay error on each logic block or wire may not be big enough to be detected. These faults will impair the circuit performance if a critical path of the circuit covers a large

number of affected logic blocks and wires. As a result, without considering the spatial information, such delay faults may not be located correctly or with a good resolution using the previous approaches.

The main difference between Cheung's work and ours is that they first uses a 2-D ring oscillator array while ours uses a 1-D curve mapped to a 2-D area. In Cheung's work, the delay map is obtained by measuring the frequency of **EACH** ring oscillator (RO) and plotting it out. This approach is very fine-grained as a RO is the basic testing unit. The spatial correlation is characterized by examine the frequencies of all the ROs.

In our work, the unit of the delay map is the "unit block" which is a resizable region covered by a segment of the space-filling curve. The spatial correlation of delay defect is handled by the inherent spatial adjacency of the curve. Our "delay map" is more **coarse-grained** compared to Prof. Cheung's work and it provides a more general view of the delay distribution of the FPGA. Our work is time-efficient as it requires only one "pass" to generate all the results.

2.5 Summary

Our contribution is a FPGA delay fault characterization tool that detects areas under delay faults quickly and accurately. Hilbert-type space-filling curve has been applied to the problem of test path generation, taking advantage of its excellent locality preserving quality..

CHAPTER.3
FPGA DELAY FAULT CHARACTERIZATION FRAMEWORK

As shown in the previous chapter, delay faults are more and more in the FPGAs fabricated in the advanced technology nodes. It has big impacts on the yield and cost of FPGAs. There is a need to develop a delay fault characterization framework that will address this issue in a comprehensive approach. The framework enables the delay faults to be located quickly in a FPGA by producing a delay variation map with a scalable resolution. This delay fault map creates a brand new horizon for new delay variability-aware EDA software to be supported and developed.

In this Chapter, we first define the general delay fault characterization problem, then the framework components and design flow, followed by the applications of this framework in future variability-aware FPGA design software.

3.1 Delay Fault Characterization Problem Definition

An *FPGA* can be represented by a directed graph $G = (V, E)$. The vertices V denoting the logic blocks and I/O blocks and the edges E are the interconnections. A path in FPGA is a sequence of vertices and edges from a primary input v_0 to a primary output v_n.

A delay fault is defined as an increase or a decrease in the propagation delay of a path, compared to its nominal delay value. A delay fault in an FPGA is subject to increasing

variability, resulting in a spread of delay values. A Gaussian-like distribution is commonly assumed for the delay variations in practice. As introduced in chapter 2, in FPGAs, the delay variations consist of both systematic variations and smaller random variations. Delay distributions tend to be spatially correlated, as logic blocks and wires that lie in close proximity of each other have more common components, resulting in a strong correlation [29].

Delay testing is performed to detect delay faults in a circuit. Delay faults are activated and observed by propagating signal transitions along the test paths. A two-pattern test $<T_1, T_2>$ is commonly used for delay testing. Delay testing in the logic network is done by chaining all the LUTs into a test path and propagating test signals along the path. However, a long test path may not have decent detection ability for small delay faults, as it covers a large area of FPGA and it is difficult to determine the location of delay variations.

To refine the testing resolution, a set of test points can be inserted into a test path. A test point serves both as a controllable point and an observable point. By adding an appropriate number of test points, a long test path can be partitioned into several testable segments or sub-paths, each covering a sub-region of the FPGA plane. Given that we partition the test path into N sub-paths, the FPGA under test is divided into N sub-regions. The number of partitions is user-defined with regard to the granularity of testing. A larger N corresponds to smaller FPGA test regions.

An example of test region partitioning is given in Fig.3.1. In a test session, an 8x8 FPGA is partitioned into 4 test regions by inserting 3 test points into the test paths.

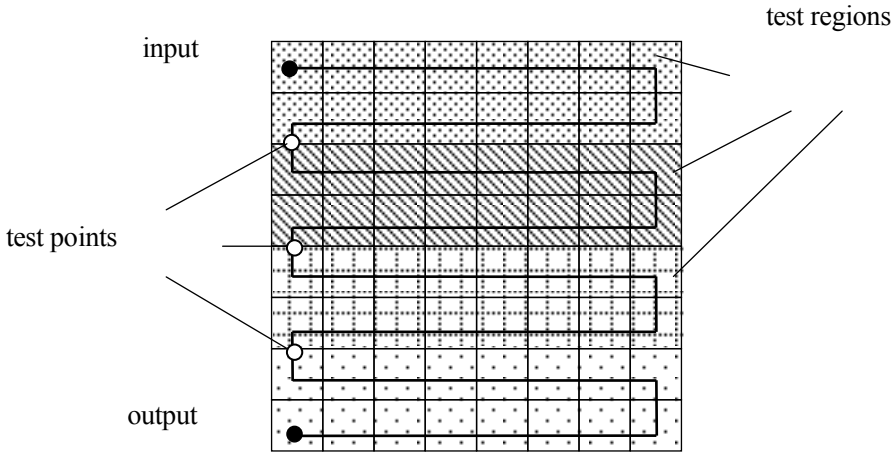


Figure 3.1 Partitioning of the test path of a 8x8 FPGA

After selecting the shape of the test path and the number of test partitions *N*, the FPGA is configured and tested accordingly. The delay values of each test regions are then recorded and analyzed. The differences of the delay values to the nominal values are computed and compared with each other, obtaining the most likely location of the delay fault.

The problem we face is how to select test configurations that maximize the effect of spatially-correlated delay faults. To improve the detection capability, the test path should be partitioned in a specific way such that each sub-path corresponds to a continuous region affected by variations.

3.2 Delay Fault Characterization Framework

To provide a fast and scalable way to generate delay fault variation map, we have developed a novel delay fault characterization framework. The refined flow diagram for this characterization framework is shown in Figure 3.2.

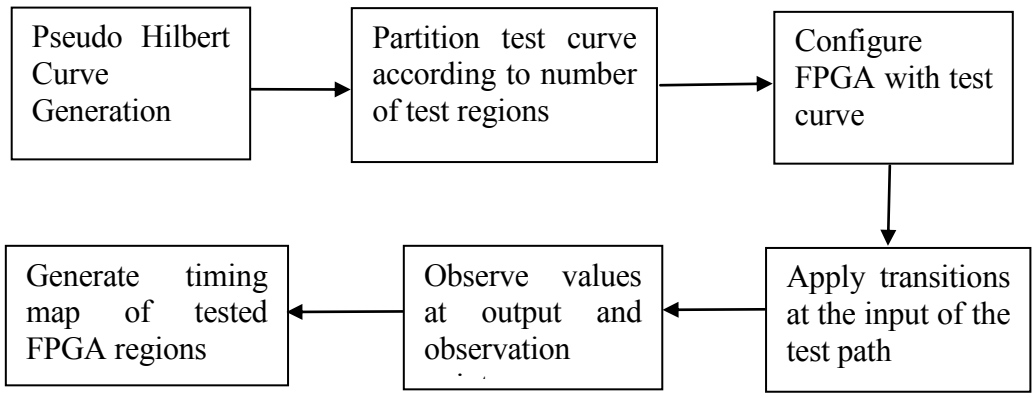


Figure 3.2 Refined Flow Diagram of Our Characterization Framework

In the first step, a pseudo Hilbert curve will be generated. Unlike the original Hilbert curve, the limitation of square shape has been eliminated by the pseudo Hilbert curve, as most FPGAs are of rectangular shape. In addition, the pseudo Hilbert curve generation algorithm will take care if the FPGA contains embedded IP cores or not, to generate the corresponding curves. The detailed description of the pseudo Hilbert algorithm will be the subject of Chapter 4.

In the second step, the pseudo Hilbert curve based test curve will be partitioned depending on resolution of the delay variation map that the users want to achieve. The higher resolution that the users want the DVM to have, the more partitioned test points will be inserted. The distance between each two continuous test points in the pseudo Hilbert curve is usually equal. The larger number of test points will not increase the

testing time, but it only increase the volume of testing data thus needs to have a larger storage space.

In the third step, the design that contains the above partitioned test curve will be captured, synthesized, implemented and configured to the FPGA device for delay variation map generation. After the configuration, the FPGA device will run with such a special design that is created for the characterization purpose. As this design specially created for characterization will be overwritten later in the actual design implementation and deployment, it will never waste the FPGA resource to enable this characterization framework. In other words, the resource overhead for this framework is zero. This is possible because of the reconfigurable feature of FPGAs.

In the fourth step, test inputs with transitions will be applied at the inputs of the test path created with the pseudo Hilbert curve. These inputs with transitions are the usual patterns that are used to test the delay along a path. By changing the frequency of the transition, different degree of delay measurement resolutions can be achieved.

In the fifth step, which is concurrently executed with the fourth step, delay value for the test inputs will be measured at the FPGA outputs and the internal partitioned test points. The positions of the internal partitioned test points are previously determined in the second step. The measured delays in these points will be recorded and compared with the expected ideal delays.

In the sixth step, the delay variation results will be collected from the previous step a delay variation map will be generated. This is a post-processing task and not so sensitive in terms of computation and memory requirements. Based on the different requirements

on the resolution of a delay variation map, different amounts of delay measurement data need to be processed. If a finer resolution of delay variation map should be generated, the measured delay data from more internal test points should be processed, so that the test path will be partitioned into more fine grained segments, and the delay variation in each test path segment will be generated. As a result, with such a higher resolution delay variation map, the user or FPGA design software will be able to have a better idea that where the most affected delay variation regions locate, thus either not to use these regions at all, or not use these regions for critical paths.

Of the six steps in this characterization framework, the generation of a good pseudo Hilbert curve is the key step. As the pseudo Hilbert curve determines how good the locality of the test points can be preserved, the resolution that can be achieved by the characterization framework is determined by the quality of the pseudo Hilbert curves.

We describe the refined flow diagram of pseudo Hilbert curve generation in Figure 3.3. The algorithm of the generation of pseudo Hilbert curves is separated into two independent braches at the very start. One branch is used to consider the pseudo Hilbert curve generation in the presence of embedded IP cores. The other branch is used to consider the pseudo Hilbert curve generation without the presence of embedded IP cores. We will discuss the first branch here with more details, while Chapter 5 will give more details for the case without the presence of embedded IP cores.

In the case that embedded IP cores are presented in an FPGA, unoccupied area on an FPGA will be divided into blocks. These blocks will then be used to create a connection graph. The Hamiltonian path for the graph will be calculated. Following this, Hilbert curve for each block following the Hamilton path will be generated. The sub-paths of the

Hilbert curves for all blocks will be then connected to form a single and complete test curve.

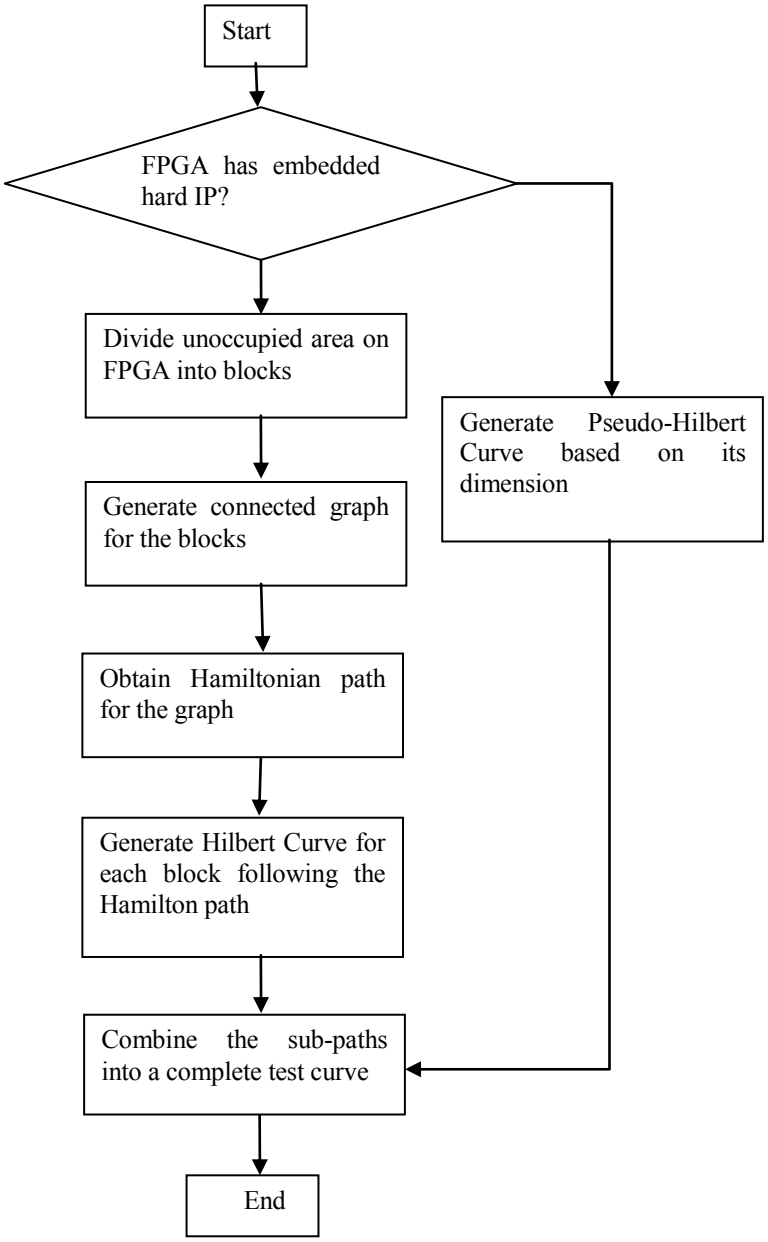


Figure 3.3 Refined Flow Diagram of Pseudo Hilbert Curve Generation

3.3 Applications of the Framework

With the delay fault variation map information obtained from our framework, a new paradigm of FPGA design software will be enabled for the delay variations of FPGAs to be managed. We present the applications of our framework in supporting this variability-aware FPGA design software in this section.

3.3.1 Traditional FPGA Placement and Routing Flow

In the traditional FPGA design flow [30] (Fig 3.5 and Fig 3.6), the software place and route (PAR) mapped design to FPGA with the goal of keeping the longest propagation delay, or the delay of critical path, to the minimum. The PAR procedure is performed regardless of the possibility that the resource used could suffer from manufacturing defects.

However, if the manufactured FPGA has regions affected by delay faults and the critical path happens to pass through these regions (as in Fig 3.4), the resulting design would have poor performance, even if it may operate correctly under a slower clock. Critical path refers to the path with the longest accumulated delay.

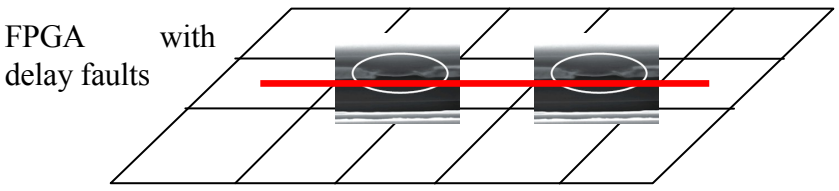


Figure 3.4 A critical path passes the regions with FPGA delay variations

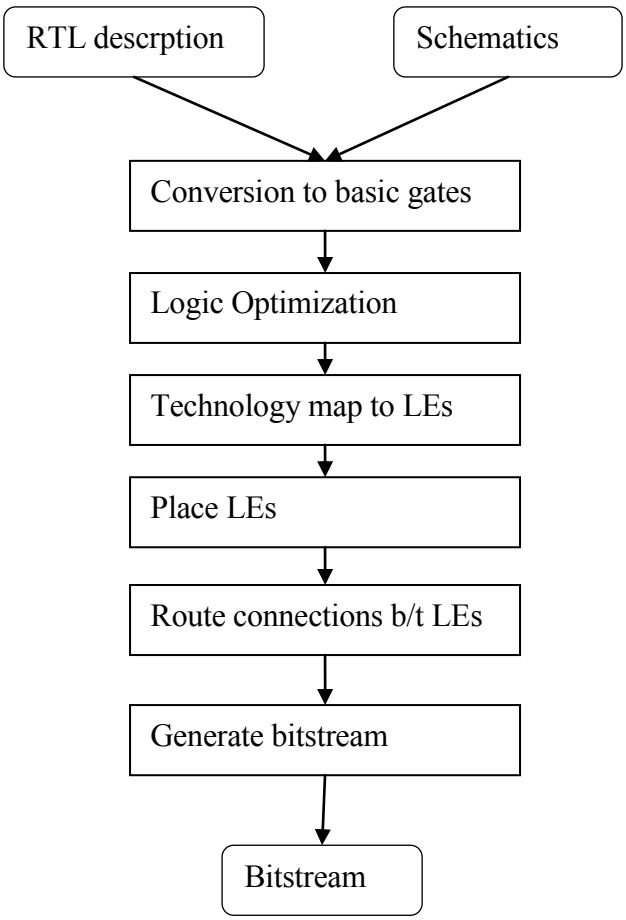


Figure 3.5 Traditional FPGA design flow

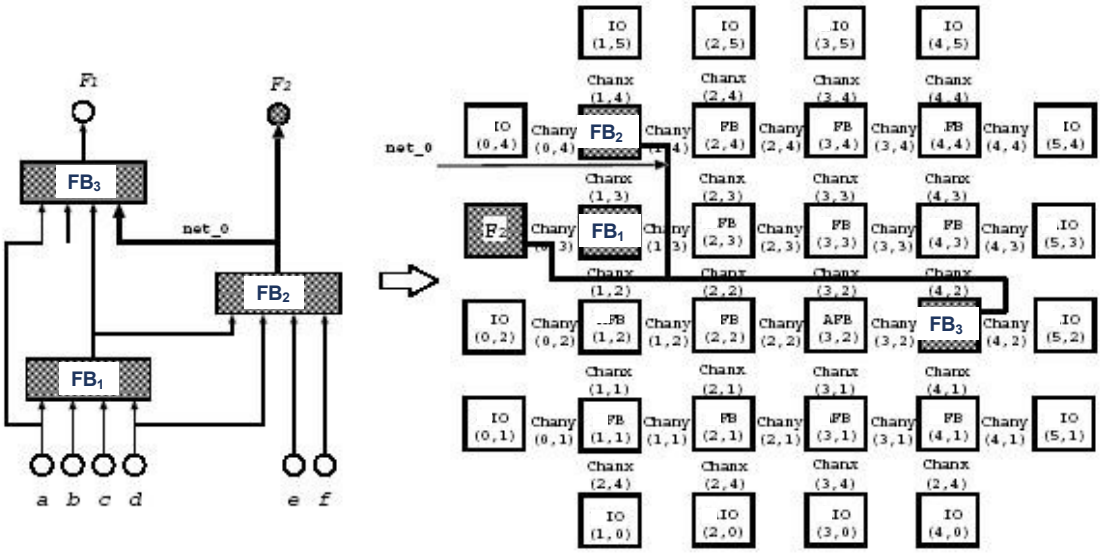


Figure 3.6 Traditional FPGA placement and routing

3.3.2 Variability-Aware FPGA Placement and Routing Flow

In order to improve the timing yield of manufactured FPGAs, our proposed framework generates a delay fault variation map based on results from a one-pass timing test. Test paths are configured based on space-filling curves and test points inserted. The observed signals are used to generate the delay variation map. This map will indicate which areas on the FPGA have poor timing performance, and label them as "slow regions". This information is fed back to the PAR software, and the software re-PAR the design to avoid the "slow regions". The map also specifies the corresponding "fast regions" which can be taken advantage of by giving a higher preference over them. Fig 3.7 shows the original PAR result which is optimized in Fig 3.8 by avoiding the slow regions. The design flow is modified as in Fig 3.9.

While this test flow performs manufacturing test that can significantly improve device performance, it requires a separate bit-stream generation for each individual chip. If the production quantity is large, the cost of the test flow may be too high. However, this test flow is ideal for FPGA cost reduction technologies, such as EasyPath FPGAs of Xilinx. These cost reduction methods usually take the user designs, test and select the devices relative to the specific resources used by user's design. Relatively smaller number of FPGAs is needed to be tested to determine the target placement. The trade-off between performance gain and test cost can be easily adjusted by taking different ratio of FPGA under test.

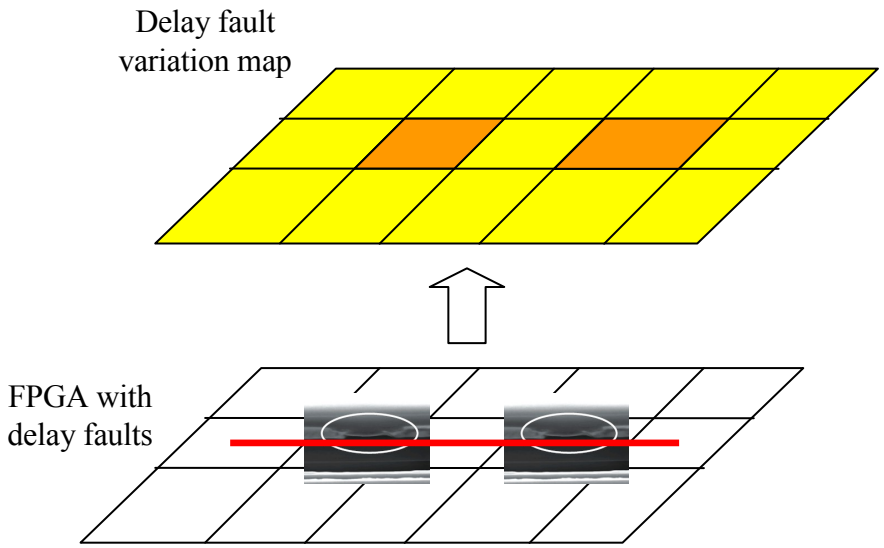


Figure 3.7 Delay fault variation map for an FPGA

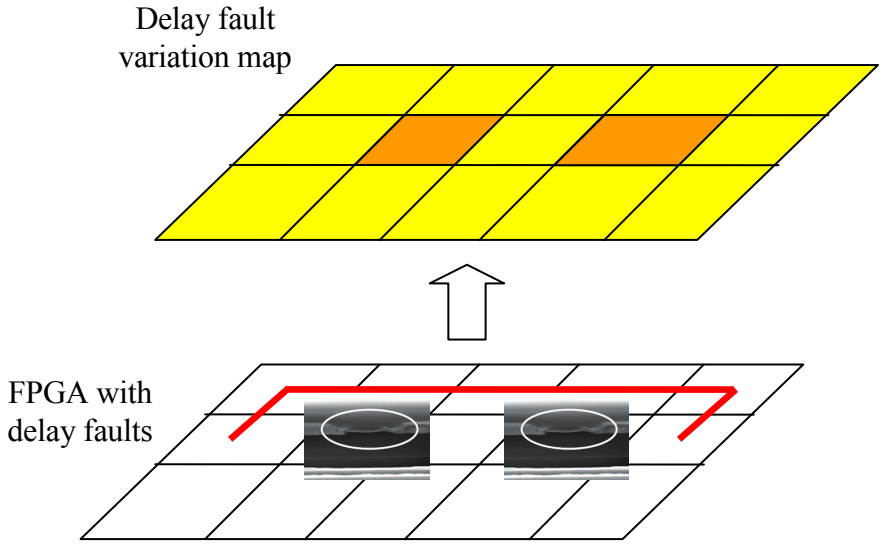


Figure 3.8 A critical path avoids regions with delay variations with the help of DVM

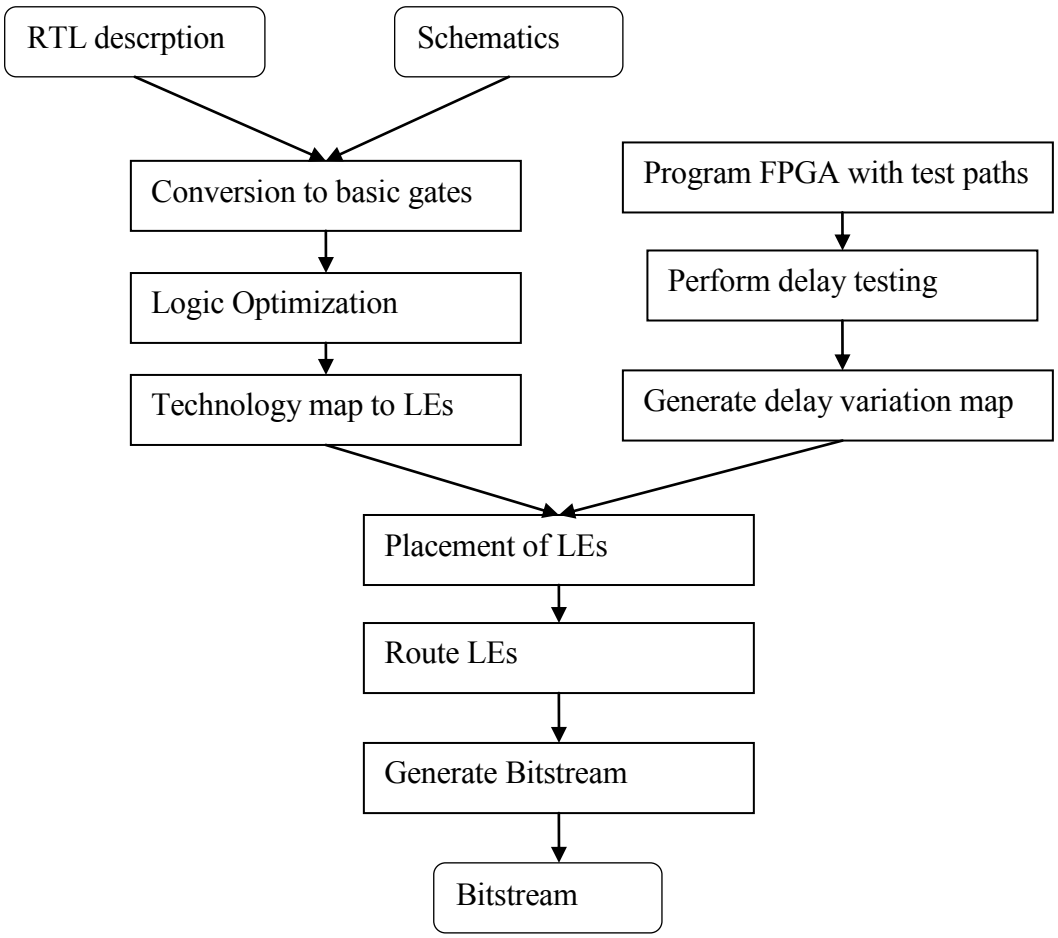


Figure 3.9 Revised FPGA design flow in our framework

3.4 Summary

In this chapter, we have described the step-by-step procedure of our proposed delay fault location methodology. We have also examined its advantages for improving timing yield of FPGA and how to combine it into the current state-of-the-art FPGA design flow.

CHAPTER.4
FPGA TIMING MODEL AND DELAY FAULT CHARACTERIZATION

In this chapter, we present the detailed delay fault characterization algorithm that we have used to extract the delay variation map from FPGAs. We first present an interval arithmetic-based timing model. Then we examine the problem of delay fault characterization and introduce the metrics that we are going to evaluate the different characterization approaches. After that, we describe the original Hilbert curve and introduce its limitation to purely square shapes. Next, we give our algorithm to generate pseudo Hilbert curves to fit the case to FPGAs, which usually take regular shapes. Finally, we explain our experimental setup and give and analyze experimental results.

4.1 FPGA Delay under Process Variations

The main trend in FPGA development has been to increase operating frequency and the number of programmable resources while reducing transistor sizes. This facilitates larger and faster designs with smaller end-system size. However, FPGAs with small and tightly packed transistors are more vulnerable to defects in fabrication process. Small defects that don't cause problems at lower frequencies, fail FPGA at higher frequencies.

Traditionally, the definition of a critical path is based upon the nominal or worst-case timing analysis. In practice, timing analysis often relies on cell characterization where the earliest, latest, and average signal arrival times are estimated for each pin-to-pin pairs of the cell. With these discrete timing values, the delay of a path can be defined as the

accumulated delay on the path. The set of critical paths is then constructed by selecting either a fixed number of the longest paths, or all paths that fall into a pre-defined timing range.

As technology scales down further into the deep submicron regions, the amount of variability in process parameters that have to be accounted for increases significantly. For example, more than 35% variations on the gate length are cited for 90nm processes and they are even larger for 65 nm processes. Process variations can be classified as inter-die variations, which affect the entire chip, and intra-die variations, which are the result of layout-specific variations. The variations are normally with a complex spatial or temporal correlation structure. They create significant timing uncertainty and yield degradation. This growing problem brings the need to build the next generation variability-aware electronic design automation (EDA) tools.

The above observation is especially important for FPGA vendors because they are almost always the first to use the most advanced technologies. For example, Xilinx is the first in the whole semiconductor industry to fabricate their Virtex-2 FPGAs in 130nm, Virtex-4 in 90 nm and Virtex-5 in 65nm processes. As the process shrinks, variations in effective channel length, threshold voltage and gate oxide thickness become more prominent. This will greatly influence the timing performance of FPGAs.

Delay variations resulted from manufacturing process, small defects, and/or signal noise can alter the discrete timing assumptions. The sizes of delay for devices can no longer be considered as discrete values, but as variables with different probabilistic distributions. Consequently, the sets of critical paths may be different from chip to chip. It is then

questionable that testing a set of critical paths selected based upon a traditional discrete timing model can still be effective in the DSM delay testing applications.

The following example shows the problem of traditional nominal timing analysis. Suppose cell characterization gives the mean and standard deviation of the delay random variable for each pin-to-pin delay. For example, the pin-to-pin delay a → e is a random variable with mean 15 and standard deviation 1. By assuming 3σ bound, the minimal and maximal delays are 15 - 3 × 1 and 15 + 3 × 1, respectively. Hence, in a discrete timing model, the pin-to-pin delay can be denoted as {12,15,18}, which represents the earliest, the average, and the latest signal arrival delays. Then, with this discrete delay model, which of the four paths P1, P2, P3, and P4 is the most critical path? Under a worst-case scenario, path P4 is the most critical path because the worst case accumulated delay is 12 + 3 × 3 + 9 + 3 × 3 = 39. The most critical path will be different if 1σbound is used instead of the 3σbound. In this case, P3 will be most critical because 14 + 1 × 2 + 9 + 1 × 2 = 27 represents the worst case. On the other hand, if the critical path is defined based upon the average delays, then PI will be the most critical one because the sum of the average delay values 15 + 10 = 25 is the largest. Fig 4.1 demonstrated this example.

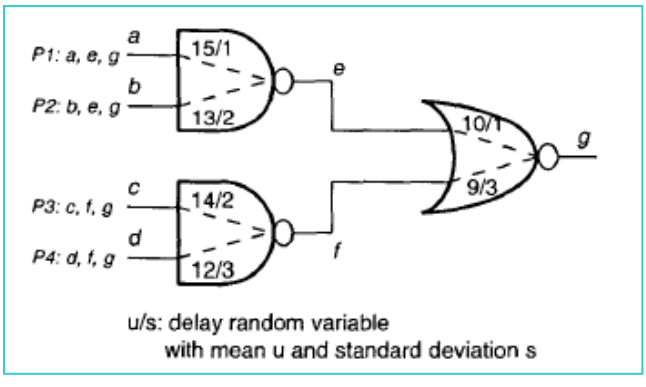


Fig 4.1 Impact of variations on critical path delay

The above example demonstrates the inadequacy of the discrete delay assumptions used to identify the most critical path. If pin-to-pin delays are characterized as random variables, then the delay of each path should be characterized as a joint probability density function (pdf) of all pin-to-pin random variables on the path. For example, the delay of path P1 in the example should be a random variable whose probability distribution is the joint pdf of the two pin-to-pin random variables from a → e (15/11) and e → g (10/11). Note that the calculation of the joint pdf depends on whether the two random variables are correlated or independent.

As a result, if we manufacture 4 chips of the example circuit, the most critical path in these four chip instances can all be different. In other words, any one of the four paths can be the most critical path in a particular chip instance. From this perspective, the definition of a critical path is no longer deterministic. Instead, the most critical path should be defined as the path that has the highest probability of being critical when a large number of the chip instances are produced. To support this definition, a statistical timing evaluation tool is required.

4.2 Interval Arithmetic-based Timing Evaluation

Statistical static timing analysis methods have been a popular research area during recent years. Existing SSTA approaches either assume Gaussian or non-Gaussian distributions. Others may add in consideration for correlation effects. Most of these proposed approaches fall into one of the two categories. The first category is the path-based methodologies [31][32]. Path-based approach statistically estimates the timing performance on critical paths of the circuit. The second category is block-based methodologies [33][34][35], which seeks to perform incremental timing analysis on the

circuit. However, to handle correlations among parameters, most approaches assume Gaussian distribution, which is not entirely the case in digital circuits. Other distributions require extensive computation, either for regression [34] or numerical integration [35]. On the whole, SSTA methods are computationally expensive and not fast enough to provide the variability-aware timing estimation.

We use interval analysis as the mathematic foundation to handle the path selection problem. Two models of interval analysis are initially suggested: interval arithmetic and affine arithmetic. Interval arithmetic (IA) [36] is a surprisingly long-lived branch of range analysis. It makes use of intervals to represent uncertainties in variables. However, it does not consider correlation and dependency between the variables. On the other hand, affine arithmetic (AA) [37][38], which is a novel refinement of interval analysis, can be applied to the problem of circuit timing analysis [39][40] and can preserve correlations among variables. With this motivation in mind, we employ affine arithmetic in proposing a new AA-based timing estimation technique for FPGAs. Correlation and dependencies among process parameters are accounted for. AA is chosen for its low complexity and is distribution-independent property, in contrast to existing SSTA methods.

4.2.1 Basics of Interval Arithmetic and Affine Arithmetic

Interval arithmetic was introduced by R.E. Moore in the 1960s [36]. It is a range-based model for numerical computation where each unknown quantity x being represented in the form $x = [a, b]$, meaning that the "true" value of x lies in the range of a and b with $a \leq b$. Arithmetic operations (add, subtract, multiply, etc) can be applied such that each

computed interval is guaranteed to contain the unknown value of the quantity it represents. The rules to perform IA are as follows:

$$\begin{aligned}
&[a,b]-[c,d]=[a-d,b-c]\\
&[a,b]+[c,d]=[a+c,b+d]\\
&[a,b]\cdot[c,d]=[\min(ac,ad,bc,bd),\max(ac,ad,bc,bd)]\\
&[a,b]/[c,d]=[a,b]\cdot[1/d,1/c]
\end{aligned} \qquad (1)$$

Though IA provides a simple and relatively efficient solution to computational problems, its inability to provide precise data is one of its major setbacks. As IA tends to be too conservative, the computed interval of a quantity may be much wider than the expected range. This is particularly accentuated in long computation chains, where the intervals computed at one stage are the inputs for the next stage. An example to illustrate is when we evaluate the expression x–x where x is in the range [2, 8]. Using IA, we get [2-8, 8-2] = [-6, 6] instead of [0, 0], which is the true range of the expression. Hence, IA is not able to model any form of correlation or dependency between the quantities. To rectify this problem, much research have been carried out [36] but at the expense of additional computations.

Affine arithmetic was introduced by Comba and Stolfi in the early 1990s [37]. This model is an improvement of the IA model but with an increased complexity and cost in representing the interval value. However, this extra information gives AA a tighter interval bound and keeps track of the correlations between quantities which IA is unable to provide. In AA, the unknown quantity x is represented by an affine form which is a first-degree polynomial:

$$\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i \qquad (2)$$

x_0 is defined as the central value of the affine form $\hat{x}$. The coefficient x_i are the partial derivatives defined using floating-point numbers. The ε_i are the noise symbols whose values are unknown but assumed to lie in the range [*-1, 1*]. Each noise symbol ε_i represent an independent share of the total uncertainty of the variable x while the corresponding coefficient x_i gives the magnitude of its error.

In AA, the arithmetic operations between the affine forms ensure that the dependencies between the quantities are preserved. These operations are divided into two categories: affine operations and non-affine operations. Examples of affine operations are:

$$\begin{aligned} \hat{x} \pm \hat{y} &= \left(x_0 \pm y_0\right) + \sum_{i=1}^{N} \left(x_i \pm y_i\right)\varepsilon_i \\ \alpha\hat{x} &= \left(\alpha x_0\right) + \sum_{i=1}^{N} \left(\alpha x_i\right)\varepsilon_i \\ \hat{x} \pm \alpha &= \left(x_0 \pm \alpha\right) + \sum_{i=1}^{N} x_i\varepsilon_i \end{aligned} \qquad (3)$$

The non-affine operations include operations for which the representation of the result requires additional noise symbols on top of the same noise symbols of the operands. The result of a non-affine operation can be formulated as follows:

$$\hat{z} = f\left(x_0 + x_1\varepsilon_1 + \ldots + x_N\varepsilon_N, y_0 + y_1\varepsilon_1 + \ldots + y_N\varepsilon_N\right) \qquad (4)$$

The key feature of the AA model is its ability to model correlation between two quantities through sharing of common noise symbols. The magnitude and sign of the dependency is determined from the coefficients assigned to the noise symbols. For example, given two affine forms:

$$\hat{x} = 10 + 2\varepsilon_1 + 1\varepsilon_2 - 1\varepsilon_4$$
$$\hat{y} = 20 - 3\varepsilon_1 + 1\varepsilon_3 + 4\varepsilon_4 \qquad (5)$$

From this data, we observed that *x* lies in [*6, 14*] and y lies in [*12, 28*]. As both *x* and *y* share the same noise symbols ε_1 and ε_4 with non-zero coefficients, they are not entirely independent of each other. Note that the signs of the coefficients are not meaningful in themselves as the sign of ε_i is arbitrary. However, the relative sign of x_i and $_i$ defines the direction of the correlation. Hence, using AA, the pair (*x, y*) is constrained to fall within the dark shaded region as seen in Fig. 1. However, if IA were to be used, this dependency would be lost. In fact, using IA, the pair (*x, y*) is only known to lie in the rectangle shown in light grey in Fig 4.2.

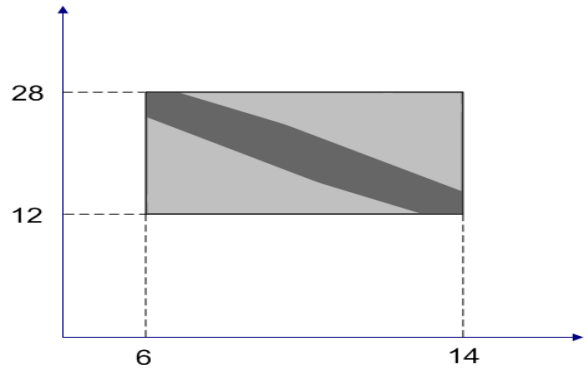


Fig 4.2 Joint range of two partially dependent quantities in Affine Arithmetic

4.2.2 Delay Model

In the traditional deterministic FPGA timing estimation [43], a *directed acyclic graph* representing the circuit structure is utilized. Each node in the graph represents the input

pins and output pins of basic circuit elements. Edges are added between the inputs and outputs of the logic blocks and between pins which the circuit netlist specifies. Each edge is annotated with the delay required to pass through the circuit element. This delay is based on the Elmore delay model. The Elmore delay of a source-sink path is [43]

$$Td = \sum_{i \in Source - \sin k\ path} R_i \cdot C(subtree_i) + T_{d,i} \qquad (6)$$

where $T_{d,i}$ is the intrinsic delay of a buffer if element i is a buffer and 0 otherwise. R_i is the equivalent resistance of element i. $C(subtree_i)$ is the total downstream capacitance of the subtree rooted at element i.

To obtain the delay of a path, the traversal begins at nodes with no incident edges and each is labeled with a signal arriving $T_{arrival}$, of 0. Each node which has incidents edges from the labeled nodes is marked with its arrival time as:

$$T_{arrival}(i) = Max_{\forall j \in fanin(i)} \{T_{arrival}(j) + delay(j,i)\} \qquad (7)$$

where node i is the node being labeled and delay (i, j) is the delay value marked on the edge joining node j to node i. This procedure continues until every node is labeled.

4.2.3 Modeling of Process Variations in Delay Model

In this section, we introduce what are the process variations that we consider in the FPGAs and show how the AA model is incorporated into a traditional deterministic FPGA timing estimator in the tool VPR [44][45] to model intra-chip variations with correlation. The model introduces noise symbols to account for both local and global variations.

We only focus on modeling structural variations and delay variations in our work. In structural variations, 4 components are considered: metal thickness (T), inter-layer dielectric (H), line-width (W) and gate length (L) (See Figure 4.3). These variations do result in adverse changes in the electrical properties which include the resistance (R) and capacitance (C). These electrical parameter variations bring about direct impacts to the performance of the circuit. Hence, an accurate model of interconnect geometry variation is essential for accurate circuit simulation.

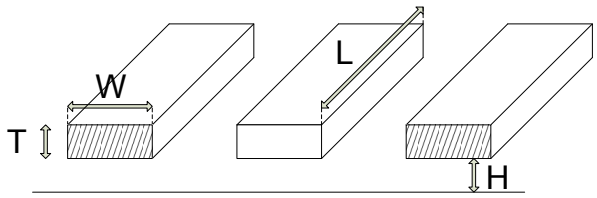


Figure 4.3 Geometry of wiring

In delay variations, variations in the buffer intrinsic delays, sub-block delays and logic delays are taken into consideration. These variations are due to the device variations which are not modeled in VPR. Instead, each of these delays is of a deterministic value defined in the architecture file obtained using SPICE simulations in VPR. These delay variations affect the performance of the circuit a lot.

As AA provides the abilities to model various variations using different noise symbols and at the same time maintain the correlation between these symbols, it is chosen as the ideal model. With the AA model, we are able to accurately track how variations get propagated during the placement and routing using the noise symbols. With the affine

interval obtained, measures to counteract these variations can be then developed to improve the quality of the circuit performance.

To model the structural variations, variations in R and C are used. But initial results show that we are not able to handle correlation at this abstraction level. Instead, a low level implementation is applied to obtain the R and C using W, H, T and L. The formulas used to obtain R and C are indicated in (8) and (9) respectively. Using these equations, noise coefficients of different signs are generated and this creates correlations between the same noise symbols.

$$R = \frac{\rho \cdot L}{W \cdot T} \tag{8}$$

$$C = \varepsilon_{ox} \cdot \left[\frac{W}{H} + 0.77 + 1.06 \left(\frac{W}{H} \right)^{0.25} + 1.06 \left(\frac{T}{H} \right)^{0.5} \right] \tag{9}$$

Although correlation is handled, several extra noise symbols are created while using these equations. This affects the complexity significantly. To reduce the complexity, we sum up the positive and negative coefficients of the extra noise symbols of R and C and assign them into two new noise symbols respectively as demonstrated in (10). All similar parameters will share these two noise symbols and any other extra noise symbols that are generated during the program flow (place and route). To model spatial correlation as well, we have adopted the idea in [46][47], that is, each parameter is reinitialized to contain a unique noise symbol based on its grid position in FPGA. An example is illustrated in Fig 4.4.

$$\begin{aligned} & X = X_0 + x_1 e_1 + x_2 e_2 - x_3 e_3 + x_4 e_4 - x_5 e_5 \\ \Rightarrow\ & X = X_0 + x_1 e_1 + (x_2 + x_4) e_6 - (x_3 + x_5) e_7 \end{aligned} \tag{10}$$

Note: e_1 is the unique noise symbol for R and C, while the rest are generated due to equations (8) and (9) and are global to all.

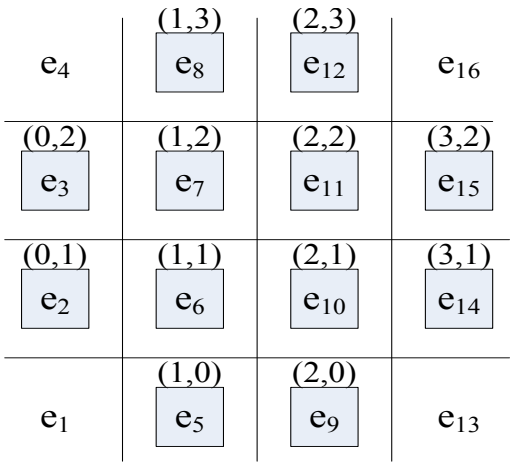


Figure 4.4: The grid-based model to model correlations

In statistical modeling, variables are often modeled as a random variable that is represented using a probability density function (pdf) or a cumulative distribution function (cdf). The MAX and ADD operators [48] are often used to join these variables. Canonical timing modes [49][50] is also proposed to address the correlations through shared parameter variations. Using this model, a block delay and its covariance with another block delay can be evaluated.

4.2.4 Modeling of Process Variation using Affine Arithmetic

Compared with the existing techniques of statistical modeling, our work does take the form of the canonical timing model. However, using the AA model, we are unable to represent a variable using either a pdf or cdf. as AA does not store the distribution of the variable. Although this is a drawback, AA allows quick extraction and good estimation

of the bounds of the variation without the hassle to handle distribution of the variations. Still, when comparing it with the traditional corner-based approach, AA is better in terms of runtime and accuracy. To the best of our knowledge, most SSTA techniques have a complexity ranging from *O(n) to* $O(n^2)$ or have complexities which increase exponentially with number of gates. In AA, the complexity of its arithmetic operators is of *O(m)* where *m* is the max number of noise symbols in any of the operands. And this complexity does not increase exponentially with circuit size. In terms of runtime, our model is more efficient compared to SSTA of the same complexity. This can be proven in terms of the number of operations at a node. In existing SSTA, the SUM and MAX operations are utilized to obtain a variable's distribution. This is computational intensive as the distribution of the variations is tracked. However, in the case of AA, distribution information is not required. Its operations involve only the adding up of the coefficients of the noise symbols and comparing the maximum of the central values. In another words, our proposed technique has a complexity of *O(n)* but with a faster computation of the bounds. Also its complexity is dependent on the nature of the circuit and independent of the circuit's size.

To prove the speed and accuracy of our method, we perform simulations to compare our estimation results with that of Monte Carlo simulations. Our simulations use the following setup. Each FPGA logic block comprises of 2 slices. Each slice has 2 4-inputs LUT and 2 flip-flops. Each logic block's pin is connected to all tracks in the adjacent channel(s) (F_c = W). Each wire segment is of length one and is connected to one wire segment of each of the adjacent channels through disjoint switch boxes (F_s = 3). All wire segments are connected by tri-state buffers. Each benchmark is routed with a maximum of 30 iterations to obtain the minimal number of tracks per channel width. When running

the experiment using W_VPR, the placement cost function is set to use bounding-box calculation, and the router uses the breadth-first search algorithm.

Table 4.1: Parameter and its variation

Parameter	**Variation (%)**
Length (L)	20
Width (W)	5
Height (H)	20
Thickness (T)	5
Sub-block delay (SD)	5
Logic delay (LD)	5
Buffer delay (BD)	5

The table above shows the percentage variations setting for each parameter. The same set of variations is applied to both the affine and Monte Carlo simulation. The values are arbitrary set but more emphasis is on the interconnect variations for their importance in the deep submicron era.

To evaluate the accuracy of our model, we compare it against the MC analysis using both Uniform and Gaussian distribution. Also, we define a metric, looseness (12) to quantify the accuracy of our results. The looseness [51] indicates the ratio of the sizes of the MC interval and the affine interval. The sign means that affine interval is smaller (neg) or larger (pos).

$$\text{lossness} = \frac{\text{AA\_interval}}{\text{MC\_interval}} * 100\% \qquad (12)$$

Table 4.1: Comparison of bounds of critical path (ns)

Circuits	No. of nets	AA model		Gaussian (MC)		Looseness (%)	Mean diff (%)	Uniform (MC)		Looseness (%)	Mean diff (%)
		Range	Mean	Range	Mean			Range	Mean		
apex4	927	[23.5 , 25.9]	24.7	[23.9 , 25.6]	24.8	37.8	-0.2	[23.6 , 25.7]	24.7	11.7	0.2
ex5p	912	[19.8 , 21.9]	20.8	[20.1 , 21.5]	20.8	49.4	0.3	[19.9 , 21.6]	20.7	19.2	0.3
misex3	1019	[20.1 , 22.1]	21.1	[20.4 , 21.8]	21.1	40.1	-0.1	[20.1 , 22.0]	21.1	4.5	0.0
alu4	1029	[23.3 , 25.7]	24.5	[23.3 , 25.5]	24.4	6.9	0.4	[23.4 , 25.6]	24.5	6.1	0.0
s298	1287	[60.9 , 66.3]	63.6	[61.0 , 65.5]	63.2	19.3	0.5	[60.7 , 66.3]	63.5	-3.0	0.1
dsip	1306	[16.1 , 17.5]	16.8	[16.0 , 17.5]	16.8	-7.3	0.1	[16.1 , 17.4]	16.7	9.4	0.3
bigkey	1649	[21.4 , 23.1]	22.2	[21.3 , 23.0]	22.2	1.7	0.3	[21.2 , 23.2]	22.2	-14.4	0.2
des	1794	[23.7 , 26.0]	24.8	[23.9 , 25.7]	24.8	32.5	0.3	[23.7 , 26.0]	24.8	2.5	0.0
Average		-	-	-	-	22.6	0.2	-	-	4.5	0.1

With reference to

Table 4.1, we observe that our AA model has an average looseness of 22.6% and 4.5% for the Uniform and Gaussian distribution using single stream respectively. The large value of looseness is partially due to that AA accounts for the worst case of the simulation. However, worst case scenario is seldom reached in real situations. Hence, the interval obtained in AA is slightly over-pessimistic. Though our AA model gives a large interval, its mean is well-matched to about 0.2% and 0.1% deviation from that obtained in the Uniform and Gaussian distribution respectively. This demonstrates its accuracy in timing estimation.

4.3 Problem Formulation of FPGA Delay Characterization

An *FPGA* can be represented by a directed graph $G = (V, E)$. The vertices V denoting the logic blocks and I/O blocks and the edges E are the interconnections. A path in FPGA is a sequence of vertices and edges from a primary input v_0 to a primary output v_n.

A delay fault is defined as an increase or a decrease in the propagation delay of a path, compared to its nominal delay value. A delay fault in an FPGA is subject to increasing variability, resulting in a spread of delay values. A Gaussian-like distribution is assumed for the delay variations in practice [47]. In FPGAs, the delay variations consist of both systematic variations and smaller random variations. Delay distributions tend to be spatially correlated, as logic blocks and wires that lie in close proximity of each other have more common components, resulting in a strong correlation. [29]

Delay testing is performed to detect delay faults in a circuit. Delay faults are activated and observed by propagating signal transitions along the test paths. A two-pattern test $<T_1, T_2>$ is commonly used for delay testing. Delay testing in the logic network is done by chaining all the LUTs into a test path and propagating test signals along the path. However, a long test path may not have decent detection ability for small delay faults, as it covers a large area of FPGA and it is difficult to determine the location of delay variations.

To refine the testing resolution, a set of test points can be inserted into a test path. A test point serves both as a controllable point and an observable point. By adding an appropriate number of test points, a long test path can be partitioned into several testable segments or sub-paths, each covering a sub-region of the FPGA plane. Given that we partition the test path into N sub-paths, the FPGA under test is divided into N sub-regions. The number of partitions is user-defined with regard to the granularity of testing. A larger N corresponds to smaller FPGA test regions.

An example of test region partitioning is given in Fig. 4.5. In a test session, an 8x8 FPGA is partitioned into 4 test regions by inserting 3 test points into the test paths.

After selecting the shape of the test path and the number of test partitions *N*, the FPGA is configured and tested accordingly. The delay values of each test regions are then recorded and analyzed. The differences of the delay values to the nominal values are computed and compared with each other, obtaining the most likely location of the delay fault.

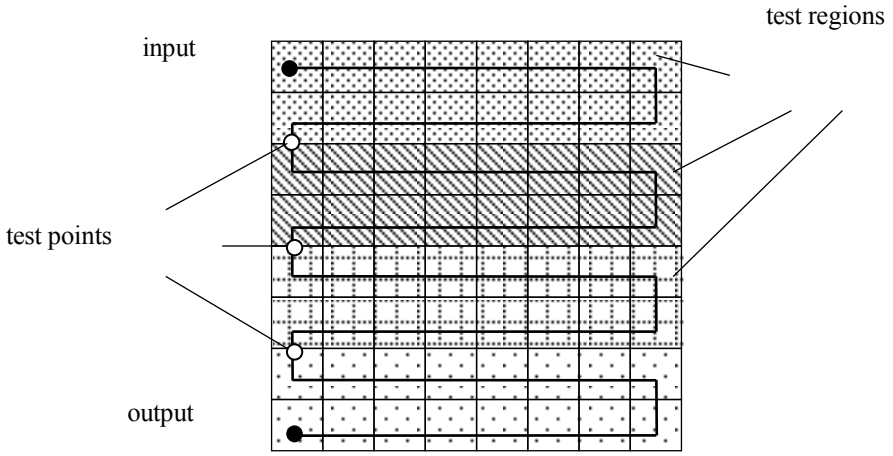


Figure 4.5 Partitioning of the test path of an FPGA

The problem we face is how to select test configurations that maximize the effect of spatially-correlated delay faults. To improve the detection capability, the test path should be partitioned in a specific way such that each sub-path corresponds to a continuous region affected by delay variations.

4.4 Locality Preserving Hilbert Curves

In order to have a test configuration which maximizes the effect of spatially-correlated delay variation, we use Hilbert curve, a type of classical space-filling curves, as the basic geometric shape of our test paths. Fig 4.6 shows the test path generated using our Hilbert curve-based algorithm. Compared with Fig 4.5, test paths generated by this technique significantly improve the locality of each test regions. Also, it corresponds much better with the grid-based model for spatial correlation (Fig 4.4), which means it captures the spatially-correlated delay variations much better than normal curves.

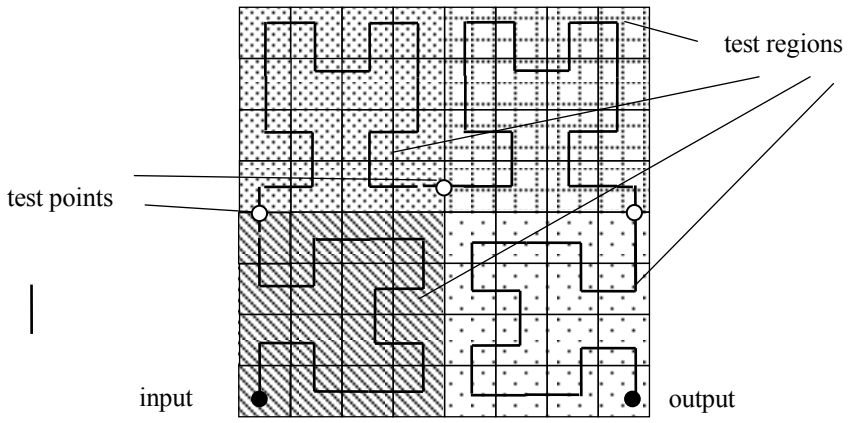


Figure 4.6 Partitioning of the test path of a 8x8 FPGA with a Hilbert curve

4.5 Original Hilbert Curve Generation Algorithm

The generation of a 2D space filling curve of successive orders usually follows a recursive framework which involves “rotating” and “sub-dividing” the basic curve unit. The classic Hilbert Curve can be generated in this manner. It can be constructed from a basic unit shape as shown for $n = 1$ in Figure 4.7. The relative position and rotation of

each unit shape is defined by its sequential position in the curve generation (see Figure 4.7). As the resolution of the curve increases, more unit shapes are required for its description, but the principle remains same as the original proposition of dividing each part into smaller parts.

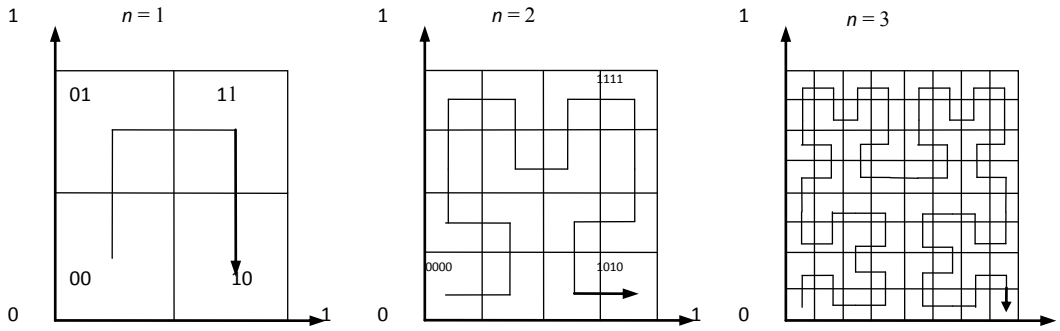


Figure 4.7: First 3 stages in generating Hilbert curves

The first steps to generate Hilbert curves are shown in Figure 4.3. Although this method is simple, elegant and suitable for computer automation, it suffers from the vital problem that it is only able to generate curves for $2^n \times 2^n$ square regions. Most state-of-the-art FPGAs have flexible dimension. Accordingly, we apply pseudo Hilbert curve, which is a modified version of the original Hilbert curve to our methodology.

4.6 Pseudo Hilbert Curve Generation Algorithm

The method presented in this work focuses on generation of special delay testing arrays to detect timing errors in the FPGA logic architecture. Guided by Hilbert curves, our test algorithm covers all the delay faults located in the logic network, and locates the faults within a confined FPGA region. The size of region is determined by the resolution of the test.

The pseudo code of our test methodology is given in the figure below:

Algorithm 1
Define: ***m***, ***n*** *= dimensions of FPGA logic matrix*
N *= number of test partitions*

```
Determine_path_set(m, n, N);
Return if_all_LUTs_are_covered;

Generate_Hilbert (m, n);
If (Pn = 1)
; // Only one test region
Else
   Partition_FPGA_plane(m, n, Pn);
   Return Subregion_set;

Foreach (subregion) do {
   Calculate_cumulative_delay_error();
} while (! All_regions_are_processed();)

Return Accumlated_delay_error;
```

Fig 4.8 Pseudo code for Overall Delay Fault Variation Calculation Algorithm

The resolution of the test is determined by *N*. It is a user-defined parameter which shows the granularity of the test requirements, i.e., the size of area in which the delay error is confined to.

The algorithm begins by validating the input parameters and generating a pseudo-Hilbert curve based on the dimensions of the logic matrix. The matrix is then partitioned into *N* test regions, each corresponding to a continuous sub-interval of the curve. The accumulated delay error in each test region is calculated by adding the delay error of each point within the same region, and the overall results are then analyzed to determine the location and severity of the delay faults.

The procedure and pseudo-code to generate Hilbert curves is shown in the following figure. First, we split the FPGA logic matrix into a set of sub-regions by selecting the appropriate splitting time. Assuming $m \geq n$, the splitting time M is calculated by $M = \log_2(n/2)$, resulting in 4^M sub-regions. The sequence of the sub-regions when mapped to the curve is the same as that of an original Hilbert curve with $n = M$. Such a curve is generated to find the upper address for each region.

Algorithm 2
Generate_Hilbert (m, n);
Define: ***m, n*** *= dimensions of FPGA logic matrix*

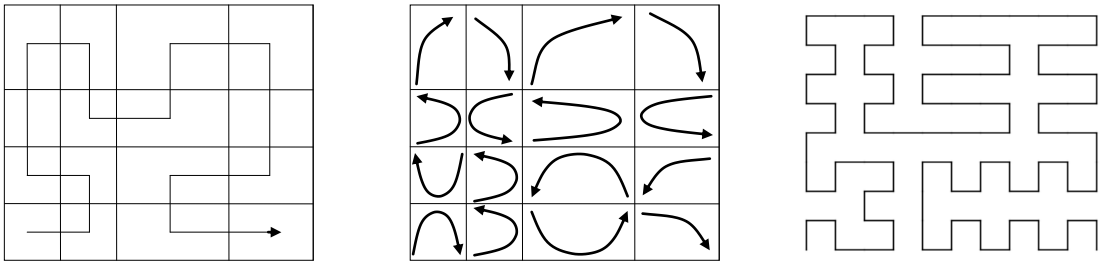
*//*Stage 1: Generate Upper Address for subregions
Else
Partition_Rectangle(m, n);
Return Subregion_dimensions;
//Stage 2:Scan each subregion
Foreach (subregion) do {
Determine_inner_scanning_procedure();
If (horizontal) Scan_Horizontal();
Else Scan_Vertical();
Generate_inner_address();
} while (! All_regions_are_processed();)

Combine_addresses();
Return
List_of_address;

Fig 4.9 Pseudo code for Pseudo Hilbert Curve Generation

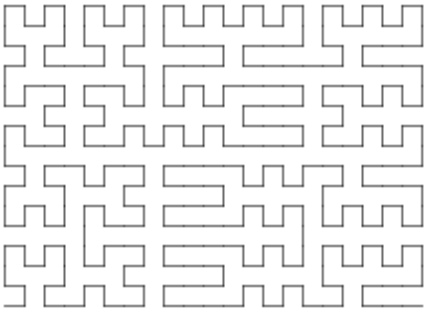
In the next stage, each of the sub-regions is scanned to calculate the lower address of the points in it. Based on different orientations, we use two types of scanning procedure: *scan_vertical* and *scan_horizontal*. The scanning direction is determined by the dimension of the sub-region (even or odd), the entry point, and the exit point. After the appropriate scan procedure is selected, the lower parts of the address for all the points are generated and then combined with the upper part. Hence, we obtain the complete address for each point in the matrix.

To demonstrate the algorithm, Fig. 6 shows how the Hilbert curve is generated for a (11, 9) region. As the splitting time is 2 for (m = 11, n = 9), the whole region is split into 16 sub-regions, as shown in (a) (the curve shows the direction of basic Hilbert curve for n = M = 2). In (b), the entry point and exit point for each sub-region is obtained, and hence the internal scanning directions. Finally, (c) shows the complete curve by combining the addresses obtained in (a) and (b). The algorithm has a complexity of ($4^M mn$). Examples of generated curves are shown in Fig 4.11.

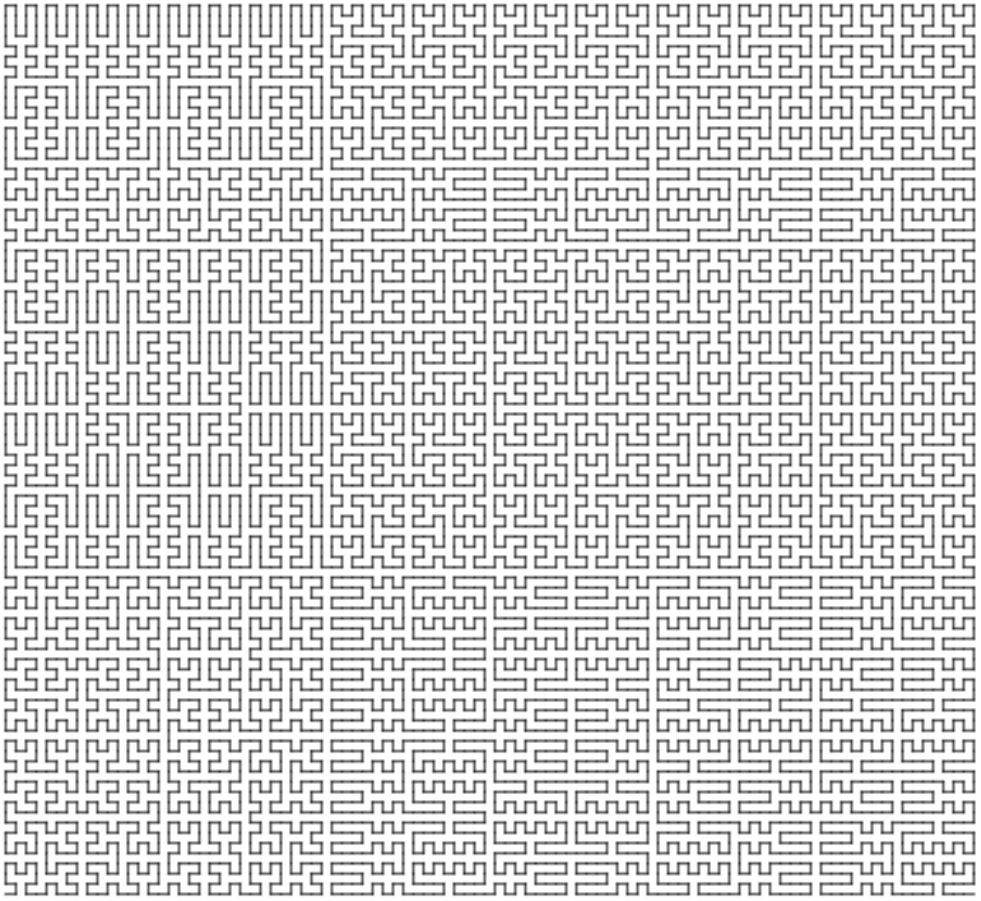


(a) (b) (c)

Figure 4.10 Procedure of Pseudo Hilbert Curve Generation



(a)



(b)

Fig 4.11 Examples of Generated Pseudo Hilbert Curves:
(a) 22 × 16, (b) 96 × 88

4.7 Experimental Results and Analysis

We will describe the experimental results that we have obtained to validate our delay fault characterization framework and provide a thorough analysis of the results.

To compare our framework with other state-of-the-art work, we have implemented both our Hilbert curve based delay fault characterization algorithm and a snake curve based algorithm. The same batch of delay faults were generated and injected into the same FPGA model, and one delay fault map has been generated with our Hilbert approach, and the other delay fault map has been generated with the Snake approach. We want to show that the delay fault map generated with our Hilbert map not only has a much better resolution to locate the delay faults within a FPGA, but also it provides a scalable approach to trade-off the number of test points and the delay fault map resolution.

Two metrics are defined to evaluate our method: the detected region $R_{detected}$ and the detection gain G. The former determines the region which is the most affected by the faults and the latter shows the likelihood for a delay fault to reside in a region. For our experiments, we select the region with the most increase in propagation delays to be the detected region. Let $\boldsymbol{R}$ be the N test regions and $\boldsymbol{d}$ be the accumulated delay error in a test region.

The detected region is

$$R_{\text{detected}} = \{R_i, d_i = \max(d_1, d_2, ..., d_N)\} \quad (1)$$

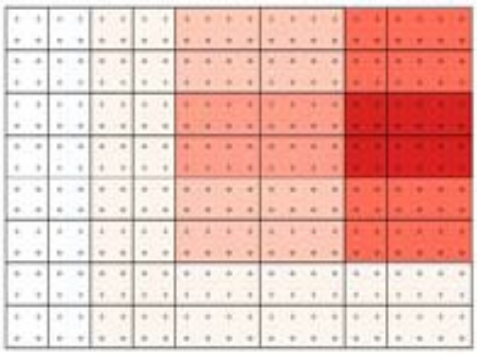
The detection gain is expressed as

$$G = \frac{1}{N}\sum_{i=1}^{N}\frac{d_{\text{detected}}}{d_i} \tag{2}$$

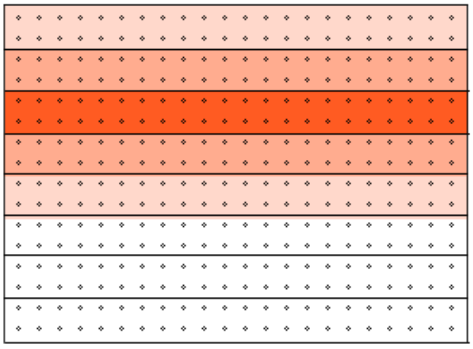
It is the average difference in accumulated delay error between the detected region and other regions, in other words, it shows how the delay error is concentrated to the detected region and its neighbouring regions.

To demonstrate the algorithm, Fig.4.12 shows how the Hilbert curve is generated for a (11, 9) region. As the splitting time is 2 for (m = 11, n = 9), the whole region is split into 16 sub-regions, as shown in (*a*) (the curve shows the direction of basic Hilbert curve for $n = M = 2$). In (*b*), the entry point and exit point for each sub-region is obtained, and hence the internal scanning directions. Finally, (*c*) shows the complete curve by combining the addresses obtained in (*a*) and (*b*). The algorithm has a complexity of $O(4^M mn)$.

For comparison, a snake (zigzag) curve is used as its structure is straightforward and it can handle arbitrary rectangle dimensions, as shown in Fig. 4.11. A snake curve simply traverses through all the point in a rectangle from one side to the other. FPGA dimensions are taken from the Xilinx devices datasheet. We use the same setting for both curves and compare the results obtained.



(a) Hilbert curve generated delay fault map



(b) Snake curve generated delay fault map

Fig 4.12 Comparison of delay fault map generated by different curves

Table 4.3 shows the detection gains for various FPGA dimensions and error sizes. For ease of comparison, the gain value in the table is log$G$ as the original value is very big. The error magnitudes are computed as different percentages of nominal propagation delay. We also computed the increase in detection gain of Hilbert curve with snake curve. From the result, we can see that the pseudo-Hilbert curves achieve substantial larger gains over common curves. This increase is stable over different FPGA sizes and error magnitudes.

Fault Size	1%			5%			8%		
Curve Type / FPGA size	Snake	Hilbert	Increase	Snake	Hilbert	Increase	Snake	Hilbert	Increase
22*16	96.81	200.98	107.60%	43.51	90.36	107.68%	26.86	55.86	107.97%
32*24	113.29	209.84	85.23%	36.86	68.90	86.91%	21.65	40.80	88.48%
40*34	184.13	302.14	64.09%	54.48	89.13	63.60%	35.97	58.71	63.22%
56*46	134.99	212.35	57.31%	49.61	78.40	58.03%	30.62	48.39	58.03%
64*56	150.98	238.30	57.82%	54.04	85.57	58.38%	32.54	51.74	58.99%
88*70	187.82	301.79	60.68%	48.55	78.04	60.74%	30.35	48.92	61.19%
104*82	127.61	204.65	60.37%	41.88	66.97	59.95%	29.64	47.39	59.88%
120*94	169.39	282.15	66.57%	47.29	79.17	67.41%	29.19	48.84	67.33%

Table 4.3 Comparison of detection Gain (log G) between Pseudo Hilbert Curves and snake curves, and the increase in percentage

Also, compared with normal curves, Pseudo-Hilbert curves can restrict delay defect affected resources into more practical geometric regions, as shown in Fig 4.12. The detection gain of FPGA test regions are indicated by the corresponding gradients.

4.8 Summary

In this chapter 4, we have presented an interval arithmetic-based timing model and explained our overall delay fault locating methodology. Algorithms for generating test curves are given in detail and experiments are performed. The results prove that our algorithm has considerably better delay fault locating ability, compared with test performed with normal test curves.

C H A P T E R . 5
C O N C L U S I O N

This thesis presents a framework to characterize FPGA delay faults with space-filling curve-based configuration paths. The algorithm maximizes the effect of spatially-correlated delay variations and thus is able to quickly and accurately locate the delay faults with high resolution. Experimental results show that our method significantly outperforms paths generated from other curves.

Contributions

We present a test methodology for FPGA, targeted at detecting and locating FPGA delay faults. The novelty of our work lies in the application of space-filling curves as the underlying geometric basis for our test framework. Based on space-filling curves which have superb locality-preserving characteristics, we generate test arrays for FPGAs that divide the device into test regions. The test signals are then sent to the input of test arrays, and results measured to determine the severity of timing errors. By selecting different test parameters, the test method can achieve different locating resolutions. The test method is able to work on FPGAs with arbitrary dimensions. Experiments are run to show the validity of our algorithm. Compared with normal test curves, our space-filling curve-based test arrays achieve around 60% increase in delay fault locating accuracy.

Future Work

For our future work on the FPGA delay locating framework, we would like to:

- Extend our framework to examine FPGAs with embedded hard IP cores, such as microprocessor, signal processing blocks. We have obtained some initial results with such FPGA structure by combining Hamiltonian Curve with Pseudo Hilbert Curve. A Hamiltonian curve or path traverses all the points in a circuit graph once and exactly once.

  For FPGAs with hard IP cores, our method first divides the "free" area on FPGA into rectangular slices of different sizes. Then a graph is generated by taking each rectangular slice as a vertex. We then find the Hamiltonian path of this graph, and do a 2-phase test path generation for the whole FPGA. We have gained some results in proving the existence of such path; the future work is to combine it with the Hilbert curve-based path generation.

- Integrate our path generator with timing analyzer algorithms to have a more complete EDA tool. Up until now, our method only targets delay fault characterizing for FPGAs. In order to make better utilization of the delay information obtained, we need to incorporate the test procedure with different stages of EDA flow. For example, after getting the different levels of timing errors of each FPGA test region, we can do a re-PAR to improve the timing performance of the design.

BIBLIOGRAPHY

[1] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, *"Statistical timing analysis: from basic principles to state of the art,"* IEEE Trans Comput.-Aided Design Integr. Circuits Syst., vol. 27, no. 4, pp. 589–607, April 2008.

[2] P. Sedcole and P. Y. K. Cheung, *"Within-die delay variability in 90nm FPGAs and beyond,"* Proc. IEEE International Conference on Field-Programmable Technology, June 2006, pp. 97–104.

[3] Liao, H.-C. Liou, J.-J. Peng, Y.-L. Huang, C.-T. Wu, C.-W., *"Delay Defect Coverage for FPGA Test Configurations Based on Statistical Evaluation",* International Symposium on VLSI Technology Systems and Applications, 2005.

[4] Y. Peng, J. Liou, C. Huang, and C. Wu, *"An application-independent delay testing methodology for island-style FPGA,"* 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp.478-486, 2004.

[5] P. Girard, O. Heron, S. Provossoudovitch, and M. Renovell, *"Manufacturing-oriented testing of delay faults in the logic architecture of symmetrical FPGAs,"* IEEE Proc. ETS, pp. 52-57, May 2004.

[6] M. B. Tahoori and S. Mitra, *"Application-dependent delay testing of FPGAs,"* IEEE Trans Comput.-Aided Design Integr. Circuits Syst., vol. 26, no. 3, pp. 553–563, March 2007.

[7] A. Krasniewski, *"Application-dependent testing of FPGA delay faults,"* 25th Euromicro Conference, vol. 1, pp.1260, 1999.

[8] Ian Kuon, Russell Tessier, and Jonathan Rose. *FPGA Architecture*, Now Publishers Inc, 2008.

[9] Clive Maxfield, *The design warrior's guide to FPGAs: devices, tools and flows*, Elsevier, 2004.

[10] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Cheng, *Electronic design automation: synthesis, verification, and test*, Morgan Kaufmann, 2009.

[11] Jaume Segura, Charles F. Hawkins, *CMOS electronics: how it works, how it fails*, Wiley-IEEE, 2004.

[12] **http://www.ibm.com/**

[13] Krstic Angela, Cheng Kwang-Ting (Tim), *Delay Fault Testing*, Kluwer Academic Publishers, Boston 1998.

[14] Y.-L. Peng, J.-J. Liou, C.-T. Huang, C.-W. Wu, "*An Application-Independent Delay Testing Methodology for Island-Style FPGA"*, Proc. of 19th IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT), pp. 478-486, 2004.

[15] Keerthi Heragu, Janak H.Patel, Vishwani D. Agrawal, "*Segment Delay Faults: A new Fault Model"*, Proc. of the 14th IEEE VLSI Test Symposium (VTS '96), pp.32-39, 1998.

[16] Y. Cao, P. Gupta, A. B. Kahng, D. Sylvester, and J. Yang, *"Design sensitivities to variability: extrapolations and assessments in nanometer VLSI,"* in Proc. IEEE International ASIC/SOC Conference, 2002.

[17] A. Asenov, S. Kaya, and J. H. Davies, *"Intrinsic threshold voltage fluctuationsin decananometer MOSFETs due to local oxide thickness variations,"* IEEE Trans. Electron Devices, vol. 49, no. 6, pp. 112–119, Jun. 2002.

[18] S. R. Nassif, "*Design for variability in DSM technologies*," in Proc. IEEE International Symposium on Quality Electronic Design, 2000.

[19] S. Bhardwaj, S. B. Vrudhula, and D. Blaauw, *"Timing analysis under uncertainty,"* ICCAD'03, pp. 615–620, Nov 2003.

[20] M. Abramovici and C. Stroud, *"BIST-based detection and diagnosis of multiple faults in FPGAs,"* in Proc. IEEE Int. Test Conf., 2000, pp. 785–794.]

[21] A. Doumar and H. Ito, *"Testing the logic cells and interconnect resources for FPGAs,"* in Proc. Asian Test Conf., 1999, pp. 369–374.

[22] W. K. Huang, X. T. Chen, and F. Lombardi, *"On the diagnosis of programmable interconnect systems: Theory and application,"* in Proc. VLSI Test Symp., 1996, pp. 204–209.

[23] M. B. Tahoori and S. Mitra, *"Automatic configuration generation for FPGA interconnect testing,"* in Proc. VLSI Test Symp., 2003, pp.134–139.

[24] D. Das and N. A. Touba, *"A low cost approach for detecting, locating, and avoiding interconnect faults in FPGA-based reconfigurable systems,"* in Proc. Int. Conf. VLSI Des., 1999, pp. 266–269.

[25] G. Breinholt and C. Schierz, "*Generating Hilbert's space-filling curve by recursion,"* ACM Trans Mathematical Software, vol. 24, no. 2, pp. 184-189, June 1998.

[26] S. Kamata and Y. Bandoh, *"An address generator of a pseudo-Hilbert scan in a rectangle region,"* Proc. ICIP, vol. 1, pp.707, 1997.

[27] J. Zhang, S. Kamata, and Y. Ueshige, *"A pseudo-Hilbert scan algorithm for artibrarily-sized rectangle region,"* Proc. IWICPAS 2006, LNCS 4153, pp. 290-299.

[28] K. Chung, Y. Huang, and Y. Liu, *"Efficient algorithms for coding Hilbert curve of arbitrary-sized image and application to window query,"* Inf. Sci. 177, 10, pp. 2130-2151, May 2007.

[29] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda, *"Statistical delay computation considering spatial correlations,"* Proc. ASP-DAC, pp. 271-276, Januray 2003.

[30] J. R. V. Betz and A. Marquart. *"Architecture and CAD for Deep-Submicron FPGAs"*. Kluwer Academic Publishers, 1999.

[31] X. Li, J. Le, M. Celik and L. T. Pileggi. *"Defining Statistical Sensitivity for Timing Optimization of Logic Circuits with Large-Scale Process and Environmental Variations"*. In Proc. of ICCAD, 2005.

[32] H. Mangassarian and M. Anis. *"On Statistical Timing Analysis with Inter- and Intra-die Variations"*. In Proc. of DATE, 2005.

[33] A. Devgan and C. Kashyap. *Block-based Static Timing Analysis with Uncertainty*. In Proc. IEEE/ACM ICCAD, pages 607–614, 2003.

[34] Vishal Khandelwal and Ankur Srivastava, *A General Framework for Accurate Statistical Timing Analysis Considering Correlations*, ACM/IEEE DAC, 2005

[35] H. Chang et al. *Parameterized Block-Based Statistical Timing Analysis with Non-Gaussian Parameters, Nonlinear Delay Functions*. In Proc. ACM/IEEE DAC, pages 71–76, 2005.

[36] R. E. Moore, *Interval Analysis*, Prentice-Hall, 1966

[37] J. Stolfi and L. H. de Figueiredo, *Self-validated Numerical Methods and Applications, Brazilian Mathematics Colloquium Monograph*, IMPA, Rio de Janeiro, Brazil, 1997

[38] J. L. D. Comba and J. Stol, *"Affine arithmetic and its applications to computer graphics"*, In Proceedings of VI SIBGRAPI (Brazilian Symposium on Computer Graphics and Image Processing), pages 9-18, 1993.

[39] C. L. Harkness and D. P. Lopresti, *"Interval methods for modeling uncertainty in RC timing analysis",* TCAD, 1992

[40] J.D. Ma, et al, *"Fast Interval-Valued Statistical Interconnect Modeling and Reduction,"* Proc. ACM ISPD, Apr. 2005.

[41] Cirillo, A., Femia, N., Spagnuolo, G. *"An Interval Mathematics Approach to Tolerance Analysis of Switching Converters",* In Proc. of IEEE PESC 1996.

[42] Femia, N., Spagnuolo, G., *"Identification of DC-DC Switching Converters Characteristics for Control Systems Design Using Interval Mathematics",* In Proc. of 1996 IEEE Workshop on Computers in Power Electronics, Portland, August 1996.

[43] R. Hitchcock, G. Smith and D. Cheng, *"Timing analysis of computer hardware",* IBM Journal of Research and Development, Jab 1983, pp100-105.

[44] T. Okamoto and J. Cong, *"Buffer steiner tree construction with wire sizing for interconnect layout optimization"*, ICCAD, 1996 pp. 44-49.

[45] V Betz, J Rose, *"VPR: A New Packing Placement and Routing Tool for FPGA Research,"* Int. Workshop on Field-Programmable Logic and Application, pp. 213 - 222, 1997.

[46] H. Chang and S. S. Sapatnekar. *"Statistical Timing Analysis Considering Spatial Correlations Using a Single PERT-like Traversal"*. In Procs of ICCAD, 2003.

[47] J. Singh and S. Sapatnekar. *"Statistical Timing Analysis with Correlated Non-Gaussian Parameters using Independent Component Analysis".* In Procs of DAC, 2006

[48] A. Agarwal, V. Zolotov, and D. Blaauw, *"Statistical timing analysis using bounds and selective enumeration,"* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 22, pp. 1243 –1260, Sept 2003.

[49] C. Visweswariah, K. Ravindran, and K. Kalafala, *"First-order parameterized block-based statistical timing analysis,"* TAU'04, Feb 2004.

[50] A. Agarwal, D. Blaauw, and V. Zolotov, *"Statistical timing analysis for intra-die process variations with spatial correlations,"* Computer Aided Design, 2003 International Conference on. ICCAD-2003, pp. 900 - 907, Nov 2003.

[51] A. Singhee, C.F. Fang, J.D. Ma, R.A. Rutenbar, *Probabilistic interval-valued computation: toward a practical surrogate for statistics inside CAD tools*, IEEE/ACM DAC 2006.