A Fast and Highly Accurate Path Delay Emulation Framework for Logic-Emulation of Timing Speculation

Shuou Nomura, Karthikeyan Sankaralingam, Ranganathan Sankaralingam[1]
University of Wisconsin-Madison, Madison, USA
[1]Cadence Design Systems Inc., San Jose, USA
Email: {nomura, karu}@cs.wisc.edu, srangana@cadence.com

Abstract

This paper proposes a novel path-delay fault emulation technique called Replay. We specifically show it allows FPGA emulation of digital ICs that adopt timing-speculation techniques. For each flip-flop, Replay builds a timing-error predictor based on timing-speculation's aggressive clock period. We use a heuristic which replicates the combination logic and uses path delays to determine which paths will be excited based on the aggressive clock period. The timing-error prediction accuracy is more than 99% for a set of real workloads on the OpenRISC processor and the FPGA emulation speed shows practically no slowdown. We also demonstrate that Replay can evaluate the impact of voltage-drop timing-faults. This fast and accurate timing-error prediction enables practical emulation of timing-speculation and quantitative analysis early in the design-cycle.

1. Introduction

Due to non-ideal technology scaling and aging effects VLSI circuits and systems designers must embed considerable design guardband to prevent any timing failure for future microprocessor designs. To minimize this design guardband, *timing-speculation* (TS) techniques have been recently proposed [1]-[5]. They draw on the insight that signal arrival time to timing elements (flip-flops) strongly depends on input patterns applied to circuits and worst-case conditions rarely occur. Thus, they allow aggressive adaptive voltage and frequency scaling optimized for the common-case execution and use a mechanism that can detect/correct rare timing errors that occur during worst-case conditions.

To understand the usefulness and quantify the impact of TS in real processors, a practical methodgy is required for emulating TS at design-time. For a given design and its representative workload, we must estimate how often the worst-case condition occurs and how effectively TS operates the circuit in the common-case condition.

Considering a digitial circuit as made up of individual flip-flops and combinational logic, we require:

> *A technique that predicts on a cycle-by-cycle basis, whether TS results in an error on any flip-flop.*

However, conventional static timing analysis (STA) techniques do not support such analysis. Statistical static timing analysis (SSTA) techniques are more accurate and estimate the distribution of the worst-case timing across different manufactured chips. They cannot evaluate TS techniques, since they do not consider the input data dependency of signal arrival time. Finally, functional simulation and emulation is not aware of timing information and delay-aware annotated functional simulation is very slow.

Thus, we need a different framework that allows analysis for common-case behavior to evaluate the effectiveness of TS for various input patterns and environmental parameters like supply voltage and temperature. Such a technique must satisfy the following three criteria.

1) *Accuracy*: It must be accurate in estimating the potential timing errors due to TS. Under- or over-estimation of TS errors will result in inaccurate assessment of the performance and power cost associated with TS.
2) *Speed*: It must allow fast analysis for various input patterns and environmental parameters.
3) *Flexibility*: It must be flexibile and allow varying of the TS boundary (aggressive clock period) to quantify the impact of how aggressively timing-speculation can be applied.

FPGA emulation is fast and allows representative workloads to be analyzed for large designs. It is already used for different types of fault analysis including gate-level modeling for single event upsets, stuck-at faults, and transition faults. Stuck-at faults, for example, can be emulated by simply fixing a node to VDD or VSS, and the area complexity is acceptable for full-design emulation [6]. To make TS practial, our approach is to develop support for TS evaluation in FPGA emulation. We call this technique Replay and it satisfies the three criteria described above.

Our basic idea is to create a representative logic block (a predictor) that captures and emulates the timing behavior of the original logic when running at an aggressive TS clock period. Our technique employs a prediction heuristic derived from the following observations. First, every input node of a gate can be considered to hold the value from the previous cycle until new inputs arrive in a cycle resulting

978-1-4244-4867-8/09/$25.00 ©2010 IEEE

in computation. Here, a predictor must essentially determine when to use a previous or new value to propagate it to the subsequent logic stages. *If all path-delays[7] through an input node are below the TS boundary, then it will evaluate to the correct new value. Else, it will hold on to its previous value* (In the interest of simplicity, we have described the node as "holding" the previous value. In reality, the node triggers computation based on the previous value). In our Replay algorithm, we attach the correct value for such non-violated input nodes of logic gates, while others are computed from the last cycle's start-point values.

Replay improves upon techniques that have been proposed to provide high accuracy and speed. Compared to startpoint-based approximation [8], which ignores all internal circuits, Replay is an order of magnitude more accurate. Compared to path-excitation detection [9], it is several orders of magnitude lower in area complexity.

While the focus is on making the analysis for TS practical, our Replay approach provides a general mechanism for emulating path delays with combinational logic in FPGA. It can be used for studying timing-errors caused by voltage-drop, wear-out etc. as well.

In this paper, we present the algorithm, describe its integration into the CAD tool-flow, and show results from our prototype implementation of the technique. Our results show that Replay provides more than 99% accuracy in predicting timing errors in TS for a set of real workloads running on the OpenRISC processor. Meanwhile, that of the conventional startpoint-based approximation is about 40%. Replay also allows fast FPGA emulation of timing-fault behavior; it is 300,000× faster than delay-annotated gate-level simulation.

The remainder of this paper is organized as follows. Section 2 reviews the fundamentals of TS and Section 3 describes TS emulation. Section 4 presents our Replay techniques, Section 5 compares to previous approaches, and Section 6 presents quantitative evaluation. Section 7 shows extensions beyond fine-grained TS, and Section 8 concludes.

2. Primer on Timing-Speculation

The motivation of TS is to remove or reduce design guardband required for conservative design using the worst-case conditions in determining the clock period. Static techniques like frequency binning and per-die voltage assignment and control can eliminate the margin when the maximum frequency is not necessary. However, they cannot eliminate safety guardband, and cannot exploit arrival time dependency on data, environmental parameters, and aging effects, which are growing in importance.

Figure 1(a) shows the basic concept of TS. It uses an aggressive clock period namely TS boundary, which is

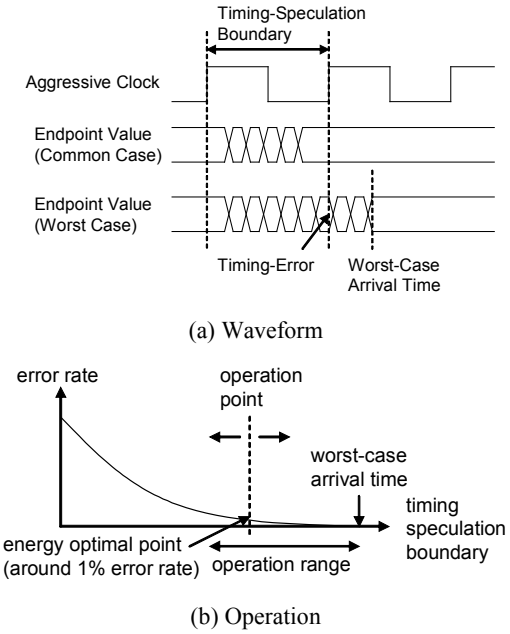


Figure 1. Concept of timing speculation.

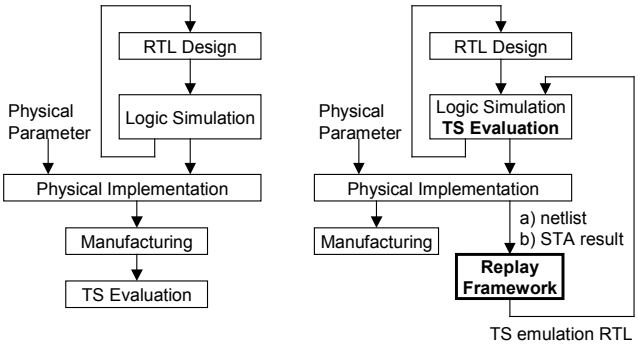
shorter than worst-case arrival time. In the common case, the arrival time is shorter than TS boundary, and it does not cause any timing errors. In case the arrival time is longer than the TS boundary for a given input, an error detection mechanism detects the timing-error, and the recover phase re-executes it with a longer clock-period.

The Razor flip-flop implements TS at fine-grain [1] while the Paceline architecture implements a similar idea at coarse-grained processor level [4].

Figure 1(b) shows how to determine the TS boundary. There is an optimal point in terms of energy consumption. If the TS boundary is too long, it cannot eliminate design guardband. If it is too short, the error rate is high, and the performance and power overhead associated with recovery becomes high. In TS techniques, some feedback scheme (hardware only or with software help) observes the error rate and adjusts the TS boundary to set the operation point at the optimal error rate.

However, a fundamental problem is that the TS boundary corresponding to the optimal error-rate depends on the circuit and applications. Thus far, researchers have estimated this for individual datapath elements like adders and multipliers. Others have built prototype systems that show 50% power reduction [2] or 20% frequency increase [3] can be achieved. These quantitative results are specific to prototype designs and provide no guidance on how the benefit translates to other designs.

Figure 2(a) shows the state-of-art in the CAD tool-flow in how designers can evaluate and use TS. Only after manufacture is it possible to quantitatively evaluate improvements from TS. Estimating TS error rates on a HDL design or netlist can guide designers early in the



(a) conventional (b) proposed

Figure 2. Design flow of timing-speculation.

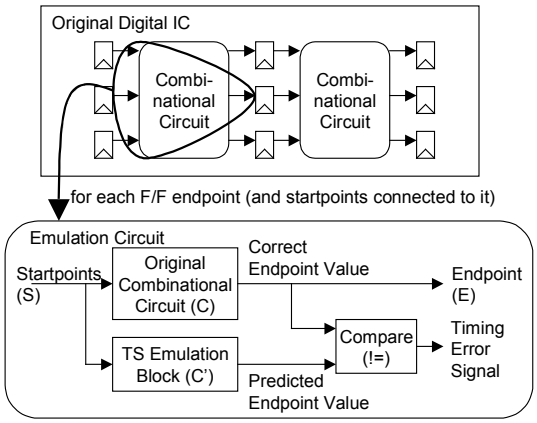


Figure 3. Emulation of timing-speculation.

design cycle. We propose a tool-flow shown in Figure 2(b). Here, TS emulation is done on a netlist and designers can iterate to improve its benefit or continually understand its benefits with design changes. *Note that, TS emulation logic is never part of the manufacturing or physical design flow.* The TS emulation logic is similar to verification assertions but is inserted automatically. The ability to estimate these benefits before manufacturing can greatly aid in the practical deployment of TS.

3. Emulation of Timing-Speculation

Figure 3 shows an overview of our TS emulation (TSE) framework that can simulate TS at RTL level. The two high-level inputs to such a framework are the RTL of the design and a clock-frequency that represents the aggressive TS clock. The output is simulation of the circuit and cycle-by-cycle trace of when errors occur due to TS. In the figure, a digital circuit is abstracted as a set of flip-flops with combinational logic. Consider a single startpoint (S), endpoint (E), and the logic in between (C). One way to implement TSE is to develop a sparse representation of C, say C' such that for all inputs, the output of C' matches the logical value that would be available when the circuit C operates at the aggressive TS clock. We call this block the *TS emulation block (or TSE block)*. In the figure, we show a detector unit that compares C' to C every cycle and reports when a TS error has occurred. When this reports true, a TS error has occurred, when it reports false, no TS error has occurred. The emulator combined with this simple comparator gives us the *TS predictor*.

The goal of this paper is to develop an algorithm that can produce such representative circuits given the RTL and TS clock as input. In a TSE framework, the three important criteria are accuracy, speed, and flexibility. Accuracy measures how closely to real TS behavior the TS predictor behaves. In evaluating a TSE framework, we have four cases to consider: true positives, true negatives, false positives and false negatives. These can be determined by comparing the TSE framework to a reference TS timing detector (either using a post-manufacture prototype or detailed delay-aware simulation). True positive and true negatives are cases when the TS predictor reports true (or false) and the reference TS timing-error detector reports true (or false) respectively. False positives are defined as cases when the TS predictor reports a timing error (true) while in the reference TS hardware no timing error occurs (false). False negatives occur when the TS predictor misses a timing error and reports false, while TS hardware detects this error with a true. The goal of a TS predictor is to minimize both false positives and false negatives.

To build effective TS predictors, we draw inspiration from prior approaches from path-delay testing and path-delay emulation [10][11]. In path-delay testing, the circuit is analyzed with timing information, and the goal is to develop inputs to the circuit that sensitize different timing paths, such that functional correctness is violated if there is a timing error. For TS prediction, we use this same insight, but create a single representative circuit that captures the behavior of the circuit up to the aggressive TS clock boundary instead of creating a set of test inputs. This is more complex than path-delay analysis alone because to develop an effective predictor we need to keep track of values from the "previous" cycle to correctly emulate a race behavior between the arrival times of values and the fast TS clock.

4. Replay Framework

This section describes our Replay algorithm and how it can be implemented in a logic synthesis, static timing analysis, and FPGA emulation framework.

4.1. Algorithm

Figure 4 shows an abstract circuit and its corresponding Replay generated TSE block to illustrate our basic insight. Our key insight is that the longest path-delay represents the delay of all paths through an input-node of a gate and an endpoint. We explain the algorithm by describing how it constructs the TSE block.

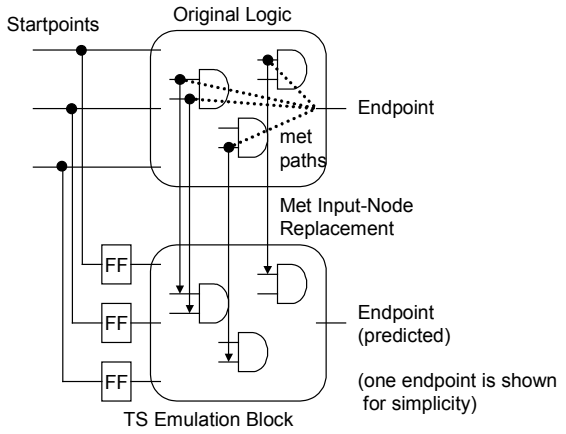


Figure 4. Replay algorithm

1) We perform STA on the original gate-net to determine path delays for different endpoints.

2) For each endpoint, the original combinational logic is replicated to form the logic part of the TSE block.

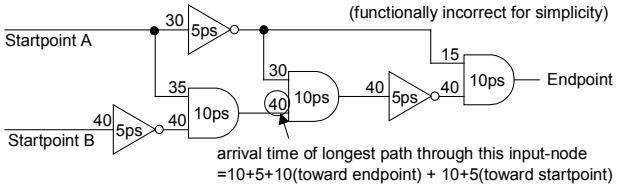
3) The startpoint signals of the logic part of TSE block are the previous cycle's startpoint signals, resulting in a logically redundant module that is one cycle behind the original.

4) Based on the provided TS boundary and the path-delay analysis, we modify the input nodes of some gates in the TSE block. If the arrival times of all paths through an input node and the endpoint are earlier than the given TS boundary, the node is tied to the original. Here, the endpoint value of the TSE block is the predicted speculation value.

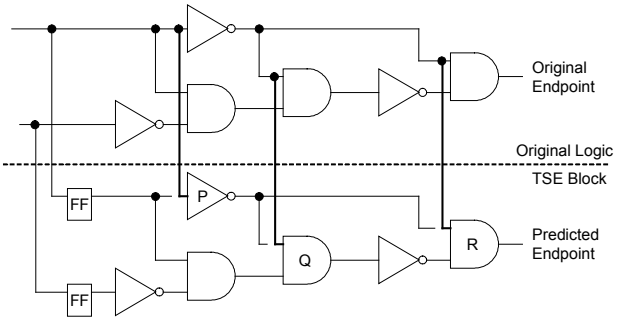
Figure 5(a) shows an example circuit with the delay of each gate and the maximum path delay of each input node of the gates. This path delay is the sum of delays of gates that feed into this input and delays of gates driven by that gate. Figure 5(b) shows the circuit with the TSE block when the TS boundary is 32ps. Here, input nodes of three gates (P, Q, and R) are replaced because their maximum delays are less than 32ps, and the replaced gates emulate logical masking behavior for the prediction of the endpoint value, in contrast to the startpoint-based approximation approach as shown in later.

The core advantage of this algorithm is that it can represent cumulative mask probability. For the example in Figure 5, the probability that the endpoint is not masked by met paths is the product of non-mask probability in gate Q and that in gate R. Replay simulates this behavior accurately as the actual circuit works, and this lowers false-positive rate, which is important for TS emulation as described above.

Note that most gates are common among TSE blocks, and a logic synthesis for FPGA emulation often removes the redundant circuits although the number of TSE blocks is the same as that of endpoints. Thus, our technique results



(a) Original gate-net with timing information



(b) Prediction circuit when timing-speculation boundary is 32ps

Figure 5. Example of Replay algorithm.

in only a small growth in logic complexity as shown in Section 6.5.

Replay was motivated by designing an emulation framework for TS. What we developed in the end is a general and universal technique for timing-aware emulation with combinational logic.

4.2. Implementation

Figure 6 shows how our Replay algorithm can be integrated with a standard CAD tool flow. Logic synthesis generates the gate-net on which we run a timing analysis step to determine the maximum arrival times for every combination of endpoints and input nodes of logic-gates. These delays are input to the Replay TSE Generator, which outputs an RTL that predicts the endpoint value based on the given TS boundary. For each gate, it compares the path delay on each port and TS boundary to decide if that

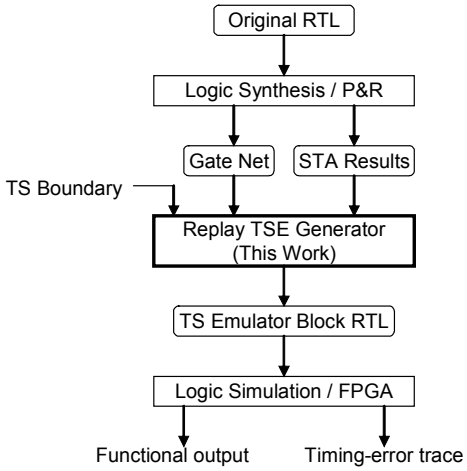


Figure 6. Tool flow of Replay framework.

port must be connected to the original circuit or left as is. The delay analysis through STA and the generation of the predictor logic modules are performed only once for a given TS boundary. In a typical CAD flow, we expect this generator to be embedded in an STA tool.

During functional emulation, the violation of the TS boundary will be determined only through the logical value of the TSE block for applied input patterns. No individual gate delay calculations are required during emulation.

Flexibility: Replay can generate prediction RTL for any arbitrary TS boundary without any restrictions. Hence, it can satisfy one of the three required criteria of *flexibility* shown in Section 1.

4.3. Computational Complexity of STA

We discuss below the computational complexity of embedding this technique into an STA tool. Here we are considering the complexity of generating the TSE block. The actual logic simulation or emulation of the netlist with the TSE block shows practically no slowdowns. Consider a circuit with *e* endpoints. For every endpoint, the fan-in tree (*n* nodes) can be represented as a directed acyclic graph rooted at start-points. We consider the reverse topological sorted list of nodes of this graph which will start with the end-point (which can be constructed with *O(n)* time complexity). This list is serially traversed, and for each node we increment the path-delay of all nodes in its transitive fan-out by the node's gate-delay. This phase is also *O(n)* time complexity. This gives the required path delays. The total time complexity is *O(n*e)*.

5. Comparison with Past Approaches

Figure 7 shows the startpoint-based approximation [8], which is the only other proposed solution that can realize full-design path-delay fault emulation on FPGA. The main idea of this technique is that timing errors occur at an endpoint only when the value of at least one violated startpoint changes (the violated startpoint means the startpoint that contains at least one timing-violated path to the endpoint). It implements value change detector for every violated startpoint, and the endpoint holds last cycle's value when any of startpoint values change. The complexity is low enough for full-design emulation because the number of violated startpoints and endpoints only affects the complexity.

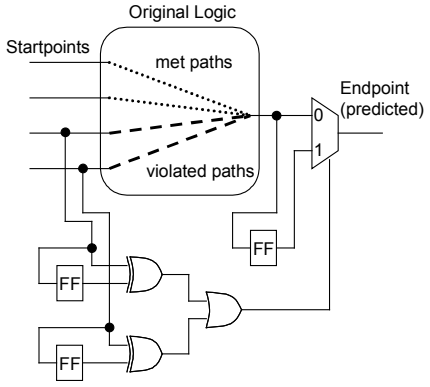


Figure 7. Startpoint-based approximation [8].

However, the accuracy of the startpoint-based approximation is low because of two reasons. First, the error injection scheme on endpoint of holding last cycle's value is a rough approximation. This scheme only works when there is no value change at an endpoint before TS boundary, and it is too optimistic because general waveform at an endpoint contains many value changes before TS boundary like Figure 1(a). The second reason is that it ignores the internal path structure. Generally, the timing-error rate significantly depends on how frequently the timing-violated paths are logically masked by timing-met paths. In contrast to our Replay technique, the startpoint-based approximation cannot take this mask effect into account. For the example of Figure 5, since the maximum path delays of both startpoints are more than TS boundary of 32ps (35ps for startpoint A and 40ps for startpoint B), the value changes of startpoint "A" cannot propagate to the predicted endpoint in one cycle. As a result, the prediction accuracy is not practical for TS emulation because it contains a lot of false-positives as shown in our evaluation results in Section 6.

Figure 8 shows the other approach of path-excitation detection [9], which was originally proposed for evaluating delay-testing coverage. It prepares path excitation condition detectors in RTL for violated paths, and realizes very high accuracy.

However, the implementation area on FPGA is unrealistic for full-design emulation because it is proportional to the number of paths to be detected. The experimental results in [9] shows that the emulation of 200 paths requires 2× and that of 400 paths requires 3× area resources compared to the original. This limitation is acceptable for the grading of delay-test because it only needs to analyze some longest paths. In contrast, for the emulation of TS, it is necessary to emulate the timing-fault behavior of all paths whose arrival time is later than the given TS boundary, which can be even 0.5× of the worst-case arrival time. Generally, the number of such paths is exponential. In our

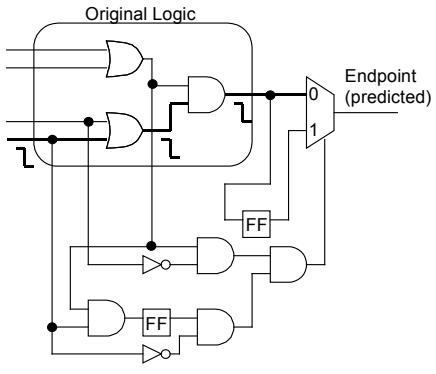


Figure 8. Path-excitation detection [9].

experiment, a 16-bit multiplier has 175 billions of paths. Therefore, this approach becomes impractical for circuit analysis employing TS techniques.

Meanwhile, the worst-case area of the Replay TSE block is less than 6× of the original as shown in the evaluation results in Section 6.5. Thus, it is acceptable for the FPGA-based fault-emulation purpose. Note that the TSE block does not introduce any area penalty to the design since it is never synthesized into manufactured hardware.

6. Evaluation

In this section, we evaluate Replay on two of the three required criteria of *accuracy* and *speed*. Section 4.2 discussed *flexibility*. Speed is evaluated in Section 6.5, and other sections are for accuracy evaluation. Section 6.1 describes the experimental framework and metrics. Section 6.2 evaluates the basic characteristics of the Replay algorithm using the ISCAS85 benchmark circuits. In Section 6.3, we evaluate with OpenRISC running full applications. In Section 6.4, we describe a detailed performance characterization of Replay.

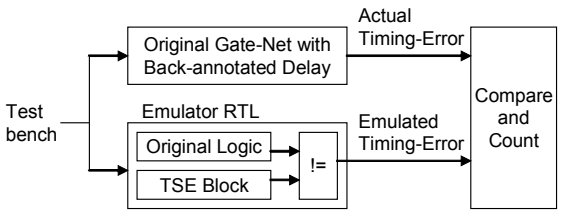
6.1. Experimental Framework

Figure 9(a) shows our experimental prototype framework. In addition to the emulator RTL, we perform detailed back-annotated gate delay simulation to determine accurately when timing-errors occur. We check the timing-error signal from the original gate-net with back-annotated delay and that from the emulator RTL. The actual timing-error signal asserts when the arrival time of a cycle is later than the given TS boundary. The emulated timing-error signal asserts if a predicted value of TSE block is different from the correct value of original logic. We evaluated every endpoint for which the worst-case arrival time is later than the given TS boundary.

Figure 9(b) shows the metrics related to accuracy, which are evaluated for each timing-violated endpoint. Error rate is the ratio of cycles for which the error signal of original gate-net asserts. Accuracy means the ratio of cycles for which the *emulated error signal* is the same as the *actual error signal*. We also show false positive and false negative separately. To study flexibility, we look at many circuits and different TS boundaries.

Experimental setup: We used the Synopsys 90nm technology library, Synopsys Design Compiler for the synthesis and the static timing analysis, and Synopsys VCS for simulation. The delay information for the gate-net simulation (.sdf file) was generated by Design Compiler. The timing constraint was tight enough to synthesize all circuits at their maximum operating frequency.

The processing resources for evaluating the OpenRISC design are measured on a 2GHz CPU and 64bit Linux. The processing time of the TSE RTL generator itself is very short (about 7sec). Since this experiment used



(a) Validation block diagram.

	Emulated Error	Emulated Not Error
Actual Error	A	B
Actual Not Error	C	D

Actual Timing Error Rate = (A + B) / (A + B + C + D)
Accuracy = (A + D) / (A + B + C + D)
False Positive Rate = C / (A + B + C + D)
False Negative Rate = B / (A + B + C + D)

(b) Metrics definition.

Figure 9. Evaluation overview.

report_timing command, the STA and report analysis consumed about 5 hours. An STA tool implementing the algorithm should consume less time as described in Section 4.3. The maximum memory usage is about 500MB.

6.2 ISCAS85 Benchmark Circuits

We evaluate the basic characteristics of Replay compared to startpoint-based approximation for the ISCAS85 benchmark circuits [12]. To study aggressive and conservative TS, we show the results for TS boundary set to 0.8× and 0.9× of the worst-case arrival time. The input data are random for every startpoint, and the length of the test vector is 64Kcycles. We evaluate the average of the metrics for all endpoints of each benchmark circuit, and we evaluate the geometrical mean of the metrics for the endpoints of overall benchmark circuits in each timing-error rate range (the metric value of 0 is excluded for the calculation of geometrical mean).

Table 1 shows the evaluation results for each benchmark circuit. Here we report error rate for each end-point. Compared to the conventional startpoint-based approximation, Replay shows significant improvements on the accuracy (about 2×) and the false-positive rate (about 10×) for most benchmark circuits. The accuracy is relatively low in the c3540 and c6288 benchmarks, which are binary-coded decimal arithmetic ALU and a 16-bit multiplier, respectively; both are complex data-path circuits, in which Replay prediction likely misses because of its approximation as described in Section 6.4. Even in these cases, Replay outperforms startpoint-based approximation.

Table 2 shows a summary across all endpoints classified according to error-rate. The results for this table are obtained by running experiments with 0.8× and 0.9× TS boundaries. Very high error rates are not important because a real system would not operate at this point. The range of less than 1% is the typical operating point. In this

Table 1. Evaluation results with ISCAS85 benchmark circuits (average of endpoints).

Benchmark	Timing-Error Rate (%)		Accuracy (%)				False-Positive Rate(%)				False-Negative Rate(%)			
			Replay		Startpoint-based [8]		Replay		Startpoint-based [8]		Replay		Startpoint-based [8]	
TS boundary	**0.8×**	**0.9×**	**0.8×**	**0.9×**	**0.8×**	**0.9×**	**0.8×**	**0.9×**	**0.8×**	**0.9×**	**0.8×**	**0.9×**	**0.8×**	**0.9×**
c432	6.4	4.5	94.6	99.2	53.7	56.5	4.1	0.7	42.2	40.4	1.3	0.1	4.1	3.1
c499	0.3	0.0	99.3	99.0	50.0	50.0	0.7	1.0	49.8	50.0	0.1	0.0	0.2	0.0
c880	9.9	3.1	90.9	94.8	59.9	57.0	8.1	5.1	36.6	41.8	1.0	0.1	3.5	1.3
c1355	0.4	0.0	99.4	99.1	50.0	50.0	0.5	0.9	49.8	49.9	0.1	0.0	0.2	0.0
c1908	0.2	0.0	99.5	99.1	51.2	52.8	0.4	0.9	48.7	47.2	0.1	0.0	0.1	0.0
c2670	3.2	0.8	89.5	86.1	74.3	66.9	9.9	13.7	24.3	32.7	0.6	0.2	1.3	0.4
c3540	15.9	2.6	75.0	77.0	61.8	55.2	17.7	21.8	30.5	43.6	7.4	1.2	7.7	1.2
c5315	1.0	0.2	96.9	94.4	60.4	58.9	2.8	5.5	39.2	41.0	0.3	0.1	0.5	0.1
c6288	33.6	4.7	70.3	80.6	51.9	52.3	11.6	16.6	31.3	45.4	18.1	2.9	16.8	2.3
c7552	6.2	1.1	90.5	92.6	55.9	57.1	7.5	6.9	41.0	42.4	2.0	0.5	3.0	0.6

Table 2. Results for each timing-error rate range (geometrical mean of endpoints).

Range of Timing-Error Rate (x) (%)	Accuracy (%)		False-Pos. (%)		False-Neg. (%)	
	Rep	**[8]**	**Rep**	**[8]**	**Rep**	**[8]**
10 < x <= 100	64.8	53.7	15.7	29.1	8.69	12.6
1 < x <= 10	87.3	54.7	5.04	41.0	0.71	1.82
0.1 < x <= 1	97.0	52.2	0.84	46.6	0.06	0.18
0.01 < x <= 0.1	98.7	52.4	0.84	43.9	0.01	0.02
0 < x <= 0.01	98.6	53.0	0.70	46.1	0.00	0.00
x = 0	97.9	61.2	0.37	30.8	-	-
ALL Range	92.4	54.1	1.33	40.7	0.11	0.20
TS Important Range (x <= 1)	**97.8**	**54.0**	**0.71**	**42.3**	**0.03**	**0.05**

range, accuracy is almost 2× better, the false-positive rate is 60× better, and false negative rate is 2× better than startpoint-based.

6.3. OpenRISC with Full Application

To study Replay's effectiveness in a full design, we applied it to the OpenRISC processor: a 32bit MIPS-like, 5-stage RISC processor [13], which contains 4345 startpoints and 2745 endpoints. We ran three full applications namely: H.264 video decoding, G.721 speech decoding, and JPEG image decoding. For the emulation of TS, we evaluate the cycle-based metrics of OpenRISC when the TS boundary is changed from 0.5× to 0.9× of the worst-case arrival time. For the cycle-based metrics, if timing-error occurs at any one endpoint, the timing-error occurs at the cycle. Therefore, the false-positive detections of each individual endpoint can cause serious degradation of cycle-based error detection accuracy.

Table 3 shows Replay's accuracy results for this design. We do not show false-negative rate because most of them are nearly 0. For the TS boundary range of 0.6× to 0.9×, where error-rates are 1%-3%, which is the practical range for TS, *Replay performs at more than 99% accuracy*

Our evaluation of the ISCAS85 circuits was a stress test of the technique, since we varied inputs randomly every cycle.

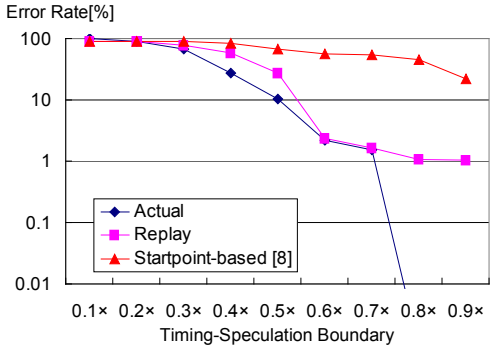
Table 3. OpenRISC evaluation results (cycle-based).

App.	TS Boundary	Error Rate (%)	Accuracy (%)		False-Positive Rate (%)	
			Rep	**[8]**	**Rep**	**[8]**
H.264 Dec. (28M cycles)	0.5×	10.3	81.4	42.9	17.9	57.1
	0.6×	2.2	99.8	46.9	0.2	53.1
	0.7×	1.5	99.9	47.8	0.1	52.2
	0.8×	0.0	98.9	54.0	1.1	46.0
	0.9×	0	98.9	77.6	1.1	22.4
	g.m. of 0.6-0.9		99.4	55.4	0.4	41.1
G.721 Dec. (3M cycles)	0.5×	17.0	87.8	28.7	10.8	71.3
	0.6×	2.3	99.8	18.2	0.1	81.8
	0.7×	1.5	99.6	17.4	0.4	82.6
	0.8×	0	98.8	22.3	1.2	77.7
	0.9×	0	98.8	59.3	1.2	40.7
	g.m. of 0.6-0.9		99.3	25.4	0.5	68.0
JPEG Dec. (6M cycles)	0.5×	15.9	83.2	38.0	16.0	62.0
	0.6×	2.6	99.6	34.7	0.4	65.3
	0.7×	1.6	99.5	33.8	0.5	66.2
	0.8×	0.0	98.9	37.9	1.1	62.1
	0.9×	0	98.9	71.3	1.1	28.7
	g.m. of 0.6-0.9		99.2	42.2	0.7	52.7

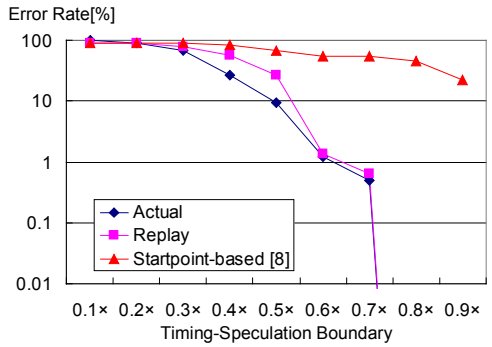
Hence, the accuracy of Replay was lower for those circuits than the full OpenRISC design, even though the latter is more complex.

6.4. Source of Misprediction

We now look at Replay's behavior in detail for one application to understand how well our heuristic works. Figure 10(a) shows the actual and estimated error rate of H.264 decoder when TS boundary is changed from 0.1× to 0.9×. Replay clearly outperforms startpoint-based estimation and is very close to the real error rate for 0.1× to 0.7×. However, for the TS boundary of 0.8× and 0.9×, Replay's estimated error rate is much higher than the actual error rate. This is because of false-positives in the 32-bit multiplier. To confirm this, we exclude 64 endpoints of the 32bit multiplier for the evaluation of cycle-based metrics. Figure 10(b) shows the result when these false-positive problems are removed.



(a) Overall



(b) Exclude 64 endpoints of 32bit multiplier

Figure 10. Actual and emulated timing-error rate

We now describe why the multiplier results in high false positives. The prediction miss is due to the approximate heuristic assumption that the longest path-delay represents the delay of all paths through an input-node of a gate and an endpoint. If the arrival times of such paths are widely distributed as shown in complex circuits like multiplier, the assumption can likely be false.

Figure 11 shows an example circuit and the worst-case arrival time for each input nodes of gates. Replay does not replace any nodes when TS boundary is 25ps, and no signal change of startpoints can propagate to endpoint in one cycle. However, if the signal (A, B, C, D) changes from (1, 1, 1, 1) to (0, 1, 1, 1), the node X changes from 1 to 0 at 10ps and the endpoint changes from 1 to 0 at 20ps. Therefore, in this case, the change of startpoint A can propagate to endpoint in one cycle, and the prediction of Replay is not correct.

In future work, we will address heuristics to reduce these miss rates further. A solution that does not use worst-case

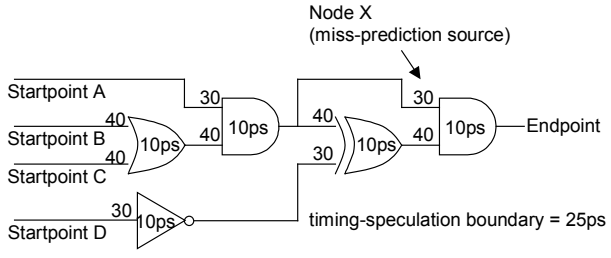


Figure 11. Source of misprediction (example).

Table 4. Area of predictor (relative to the original).

Benchmark	TS Boundary				
	0.5×	0.6×	0.7×	0.8×	0.9×
c432	2.3	2.3	2.8	2.6	1.8
c499	1.7	1.7	1.7	2.0	2.3
c880	1.2	1.6	1.5	1.4	1.4
c1355	1.7	1.7	1.8	2.4	2.8
c1908	1.8	2.1	2.9	2.7	2.2
c2670	2.1	2.0	2.1	2.1	1.8
c3540	2.6	2.6	2.6	2.6	2.1
c5315	1.8	1.8	1.8	1.8	1.6
c6288	2.5	3.1	3.8	4.2	4.1
c7552	2.0	2.1	2.3	2.3	2.2
OpenRISC	3.1	3.6	4.0	4.9	5.5

Table 5. Summary of FPGA implementation.

FPGA	Xilinx Virtex-5 XC5VLX110T-1
Design Tool	Xilinx Platform Studio EDK 10.1.03
Design Components	OpenRISC (predictor RTL), UART, microblaze, PLB, DRAM controller
Area	13,471 out of 17,280 (occupied slices)
Operating Frequency	30MHz (predictor RTL) 120MHz (microblaze, PLB)

delay but uses typical-case delay as a representative of path-delay through an input-node of gates and the endpoint is promising. For the example of Figure 11, the average delay from startpoints to node X for every possible value change patterns of startpoint A, B, and C is about 14ps. If Replay uses this value for the representative path-delay of node X instead of the worst-case delay of 20ps, the replacement occurs because 14ps+10ps is less than the TS boundary, and the prediction works correctly in the above example case.

6.5. Logic Complexity and Speed

Table 4 shows the re-synthesized area of TSE block RTL (including the original logic) relative to the original RTL. The area is less than 6× for all benchmarks. For logic simulation, this resulted in practically no slowdowns and it is practical for FPGA emulation.

Table 5 shows the characteristics of our FPGA implementation of the OpenRISC processor with TSE at a TS boundary of 0.75×. The operating frequency of Replay TS emulator is 30MHz, while that of the original design is 40MHz, resulting in 25% emulation slowdown. It is 300,000× faster than gate-level simulation with back-annotated delay, which executes at 100Hz.

7. Beyond Fine-Grained TS

In this section, we discuss some examples of how Replay can be used for purposes beyond fine-grained TS. We discuss two coarse-grained TS techniques, their emulation requirements, and how they can be modeled using the Replay framework.

7.1. Emulation of Coarse-Grained TS

Architecture-level TS [4][5]: This type of TS does not detect timing-errors on individual flip-flops. Instead, architecture-level techniques are used to detect timing-errors. Hence, in contrast to cycle-level TS [1]-[3], the emulation needs to propagate timing-errors that occur at the circuit-level to successive pipeline stages in the circuit.

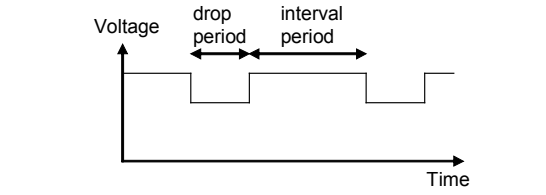
Error-tolerant TS: Some class of applications such as video decoding, image processing, and graphics do not necessarily need 100% correctness in the application output. For such applications, relaxed fault-tolerance allowing acceptable errors has been proposed [14]-[16]. We study the effect of timing errors on such applications hence we refer to this study as error-tolerant TS. We need an emulator that can analyze the impact of such errors on application quality such as PSNR of degraded images.

These TS techniques can be studied using the Replay framework as their needs are mostly similar to fine-grained TS. They need fast simulation speed and the capability to emulate a number of timing-error patterns. While fine-grained TS requires a per-cycle timing error trace, here we need to propagate timing-errors in the circuit to understand the long-term impact on the program. With a simple modification to how the Replay TSE block is used, we can model such fault propagation.

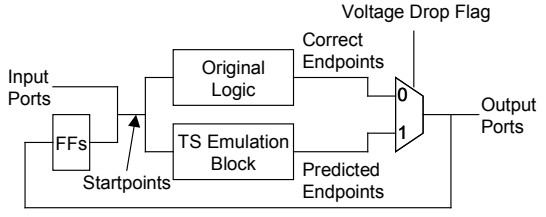
7.2. Fault Modeling

For this case study, we consider voltage-drop faults. The behavior of voltage-drop fault changes along the time-domain. It causes global delay increase only when the supply voltage drops because of effects such as rush-current or noise.

Replay can emulate this type of fault by selectively injecting faults. Figure 12(a) shows the modeling of timing faults due to external voltage drop. This is modeled by setting an aggressive TS boundary in voltage drop periods. In our analysis, to emulate the impact of the voltage drop on the path delays, the TS boundary is set to 0.75× of the worst-case arrival time in the drop period. This is a simple model which we use for our case study, more sophisticated models can be incorporated into our framework.



(a) Voltage-drop fault model



(b) Implementation

Figure 12. Fault implementation.

Figure 12(b) shows the block diagram of the fault implementation. The combinational logic of the predictor RTL code outputs the correct values of endpoints for the interval period and the predicted values for the drop period. The external voltage-drop flag signal selects which endpoint values are used for the output ports and the input nodes of F/Fs. This final selection multiplexer is the only difference to our previous fine-grained TS system shown in Figure 3. It replaces the *comparator*, which outputs a timing-error signal.

7.3. FPGA Experiment

For evaluation, we consider our H.264 decoder executing on the OpenRISC processor when voltage-drop fault is injected, and we analyzed how the output image is degraded compared to the original.

Figure 13 shows the experimental results when the drop period and the interval period are changed. The vertical axis is the length of interval period, and the horizontal axis is the length of drop period. Each point in the graph represents one full execution of the program (QCIF 3frames), demonstrating the speed of our framework. Each point is labeled Pass, Correct, Ignorable, Acceptable, Fail, or Crash.

Emulating the architecture mask effect, Replay shows the impact on architecture-level TS systems. Figure 13 contains a lot of "correct" points. For such patterns, timing-errors are masked and not shown in the application-level output and represent how an architecture-level TS system will behave.

For the emulation of error-tolerant TS, Replay can analyze the impact of timing-errors on application-level quality. In Figure 13, there are a number of "ignorable" and

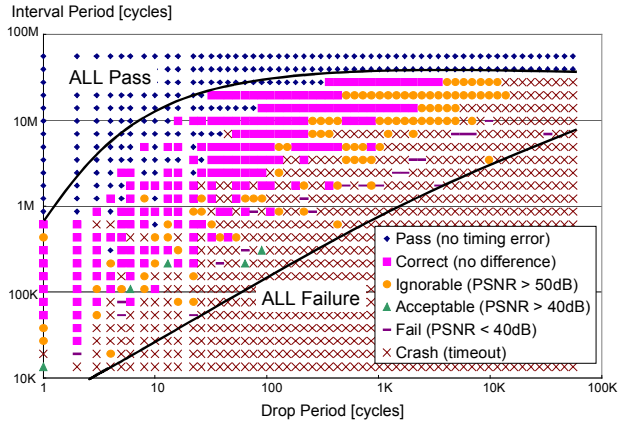


Figure 13. FPGA Experiment results.

"acceptable" points, for which PSNR of the output image is more than 50dB and more than 40dB of the original output, respectively. These outputs are of acceptable application quality.

These results show that the Replay is flexible and can be generalized to other timing fault modeling needs. Replay can also emulate wear-out faults by adjusting delay of each gate before STA. For example, the delay increase caused by hot-carrier injection (HCI) is proportional to how many times the input signal has switched, and it can be estimated by gate-level simulation of typical workload in advance.

8. Conclusion

We have developed a novel path-delay fault emulation framework called Replay. We have specifically applied it for evaluating the effectiveness of timing-speculation techniques. The generated endpoint-value predictor RTL code acts as a functional simulator of circuits employing timing-speculation techniques. Replay satisfies the following three required criteria.

1) *Accuracy*: Replay is more than 99% accurate for the timing-error prediction of a set of real workloads on the OpenRISC processor, while conventional startpoint-based approximation is about 40% accurate.
2) *Speed*: Replay algorithm produces a simple timing emulation logic block which is amenable to FPGA emulation. As a result, with FPGA acceleration Replay is 300,000× faster than gate-level simulation with back-annotated delay.
3) *Flexibility*: Replay framework can generate predictor RTL for arbitrary TS boundaries.

Moreover, for the analysis of coarse-grained timing-speculation techniques, Replay can emulate how timing-errors caused by path-delay faults propagate circuits and result in failures in system-level. In addition, Replay can emulate any type of timing-faults that can be represented as change in gate delay.

The accurate, fast, and flexible path-delay fault emulation framework enables practical evaluation of the effectiveness of timing-speculation techniques.

Acknowledgement

We thank the anonymous reviewers and the Vertical group for comments and UW CSL for their assistance. Many thanks to Nam Sung Kim and Kewal K. Saluja for several discussions that helped refine this work. Support for this research was provided by NSF CAREER award CCF-0845751 and Toshiba Corporation.

9. Reference

[1] D. Ernst, et al. "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," *Annual International Symposium on Microarchitecture*, Dec. 2003.

[2] D. Bull, et al. "A Power-Efficient 32b ARM ISA Processor Using Timing-error Detection and Correction for Transient-error Tolerance and Adaptation to PVT Variation, " *International Solid-State Circuit Conference*, Feb. 2010.

[3] J. W. Tschanz, et al. "A 45nm Resilient and Adaptive Microprocessor Core for Dynamic Variation Tolerance," *International Solid-State Circuit Conference*, Feb. 2010.

[4] B. Greskamp and J. Torrellas, "Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking," *International Conference on Parallel Architecture and Compilation Techniques*, Sep. 2007.

[5] K. Sundaramoorthy, et al., "Slipstream processors: Improving both performance and fault tolerance," *International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[6] S. Hwang, et al., "Sequential Circuit Fault Simulation Using Logic Emulation," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 8, pp. 724-736, August 1998.

[7] G. Smith, "Model for Delay Faults based upon Paths," *IEEE International Test Conference*, Oct. 1985.

[8] A. Pellegrini, et al., "CrashTest: A Fast High-Fidelity FPGA-Based Resiliency Analysis Framework," *IEEE Custom Integrated Circuit Conference*, Sep. 2008.

[9] P. Bernardi, et al., "Hardware-Accelerated Path-Delay Fault Grading of Functional Test Programs for Processor-based Systems," *GLSVLSI*, Mar. 2007.

[10] L. Wang, et al, "System on Chip Test Architectures," *Morgan Kaufmann Publishers*, 2007.

[11] M. Bushnell and V. Agrawal, "Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits," *Kluwer Academic Publishers*, 2000.

[12] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinatorial Benchmark Circuits and a Target translator in Fortran," *International Symposium on Circuits and Systems*, Special Session on ATPG and Fault Simulation, Jun. 1985

[13] "OpenRISC", http://www.opencores.org/

[14] X. Li, et al., "Application-Level Correctness and Its Impact on Fault Tolerance," *International Symposium of High Performance Computer Architecture*, Jan. 2007.

[15] L. Leem, et al., "ERSA: Error Resilient System Architecture for Probabilistic Applications," *Design, Automation and Test In Europe*, Mar. 2010.

[16] M. Kruijf, et al., "Relax: An Architectural Framework for Software Recovery of Hardware Faults," *International Symposium on Computer Architecture*, Jun. 2010.