Toward Effective Utilization of Timing Exceptions in Design Optimization

Kwangok Jeong[†], Andrew B. Kahng[†,‡] and Seokhyeong Kang[†]
[†]ECE and [‡]CSE Departments, University of California at San Diego, La Jolla, CA, USA
kjeong@vlsicad.ucsd.edu, abk@cs.ucsd.edu, shkang@vlsicad.ucsd.edu

***Abstract*— Timing exceptions in IC implementation processes, especially timing verification, help reduce pessimism that arises from unnecessary timing constraints by masking non-functional critical paths. Ideally, timing exceptions should always be helpful for quality of results (QOR) metrics such as area or number of timing violations, and for design turnaround time (TAT) metrics such as tool runtime and number of design iterations. We expect this positive impact since timing exceptions reduce the number of constraints that the design optimization must satisfy.**

In this work, we evaluate the impact of timing exceptions on design QOR and TAT, with respect to (1) the forms in which timing exception are declared, (2) the timing criticality of the target paths, (3) the number of applicable exceptions, and (4) the design stages at which timing exceptions are extracted and applied. From our experimental analyses, we observe that applying more exceptions in commercial tool flows does not consistently lead to better QOR, and often only increases runtime unnecessarily. We analyze potential causes of unwanted impacts of timing exceptions, and examine various methods to filter out ineffective timing exceptions.

Our study gives preliminary guidelines for designers and EDA vendors regarding the use of timing exceptions in design optimization processes. Our work, hopefully, lays a foundation for novel design methodologies that can maximize the benefits of timing exceptions.

I. Introduction

As the complexity and size of IC designs increase, exhaustive functional verification using event-based *dynamic* logic simulations has reached its limits. Vector generation considering all feasible operating scenarios is difficult and requires excessive simulation runtime for large designs. As a result, the *static timing analysis* (STA) verification methodology has become ubiquitous for design signoff. STA considers all possible signal transitions in a design, but in a static way. Since STA does not fully consider whether a given signal transition can occur in actual operation of the design, the STA verification methodology can report timing failures on timing paths that are not exercised during actual operations. Thus, timing optimization based on (incremental) STA results will expend unnecessary effort to meet timing requirements for non-functional paths.

To avoid such unnecessary pessimism in STA, *timing exceptions* are used in the design verification stage to filter out violations that correspond to non-functional timing paths. There are two major types of timing exceptions: (1) *false paths* that cannot be sensitized by any input vectors, and (2) *multicycle paths* along which signal propagation does not have to complete within one clock cycle. The former completely ignores timing requirements, while the latter relaxes timing requirements for the specified paths.

False paths in logic circuits have been studied in previous works [1], [2], [3] and [4]. Multicycle paths have also been investigated in the literature [5], [6], [7]. Liou et al. [8] present a false-path-aware statistical timing analysis framework. Higuchi et al. [9] investigate on a multicycle path analysis and detecting method. Yang et al. [10] propose an algorithm to identify multicycle and false paths. However, most of the previous works present methods to identify false paths or multicycle paths, and focus on the use of timing exceptions at timing analysis and verification stages.

Apart from academic investigations, designers have long analyzed timing paths and specified timing exceptions manually. Such manual works are both time-consuming and error-prone. Furthermore, with the increase of system-on-chip (SOC) design size and complexity, timing exceptions can easily take up many thousands of lines in the 'golden' constraint file at signoff. Hence, manual identification and specification of valid timing exceptions is often infeasible. Recent commercial tools such as *Cobalt* [20], *FishTail* [21] and *SpyGlass* [17] address the market for more effective automatic generation and verification of timing exceptions with minimum designer effort. In *SpyGlass* [17], timing exceptions are generated from a list of timing critical paths reported from STA tools, and each critical path is then tested whether true or false. Hence, auto-generated exceptions from commercial tools resemble the timing report format which defines one startpoint, one endpoint, and several intermediate points along a timing path. However, designers define timing exceptions based on the knowledge of functionality, and use simplest forms, omitting detailed points in timing paths, to reduce the effort of describing many timing exceptions manually.

Ideally, adding timing exceptions on the critical path of a design will relax timing constraints and reduce optimization effort, so that runtime and area can be minimized. However, adding more exceptions does not always result in better QOR and TAT since too many exceptions, requiring special care in design optimization, can give rise to negative side effects on optimization runtime and quality. Figure 1 shows the rapid runtime increase with increasing number of timing exceptions – without improving design area, and with small improvement in total negative slack (TNS). The test case is the *AES cipher* circuit in Table V.

Given the above, it is necessary to analyze the benefits of in-

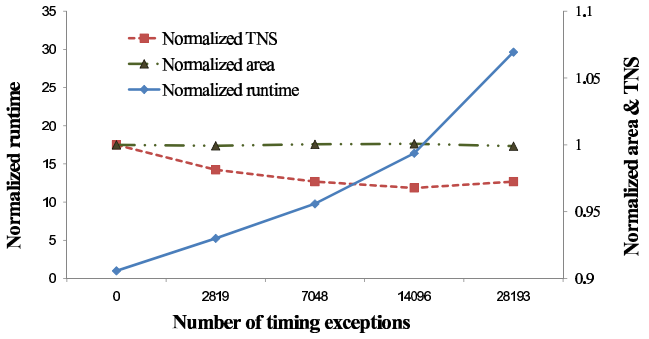

Fig. 1. Normalized runtime, area and TNS for the *AES cipher* testcase after placement and routing, versus the number of timing exceptions applied.

troducing timing exceptions in design optimization processes, and to develop a new design methodology that effectively utilizes timing exceptions. In this paper, we explore the impact of timing exceptions in the design implementation flow through experiments that address three fundamental driving questions:

(1) Do timing exceptions help or hurt in design optimization?
(2) Which exceptions give net benefit when applied?
(3) When can such helpful exceptions be identified with sufficient accuracy?

Our contributions are summarized as follows.

- We evaluate the impacts of timing exceptions on design QOR and TAT, and show that not all timing exceptions are beneficial.
- We analyze the characteristics of timing exceptions and classify effective vs. ineffective timing exceptions.
- We examine various potential metrics to quantify the effectiveness of timing exceptions.
- We give guidelines and a design methodology for use of timing exceptions during optimization.

The remainder of our paper is organized as follows. Section II presents motivational hypotheses on the insertion of timing exceptions in design optimization processes. The hypotheses are analyzed experimentally in Sections III, IV and V. Based on observations from our experiments, in Section VI we propose useful tips and a potential design methodology to use timing exceptions more effectively during optimization. Finally, Section VII summarizes our conclusions.

II. Background and Driving Questions

Timing exceptions are described in *Synopsys Design Constraints* (SDC) format which defines detailed timing requirements for the design, e.g., clock cycle time and waveforms, I/O delays, and timing exceptions. This information can be passed to synthesis, timing-driven placement and routing tools, and final signoff static timing analysis (STA). *Synopsys* also provides a guideline [12] for using the SDC format.

Figure 2 shows a standard RTL-to-layout flow with timing exceptions. Designers usually describe timing exceptions considering functional or architectural attributes of the design. Automatically generated timing exceptions can be added from commercial tools, such as those noted above, which support a path-validation function. However, the number of timing exceptions grows considerably during timing closure iterations, and the complexity of SDC is increased.

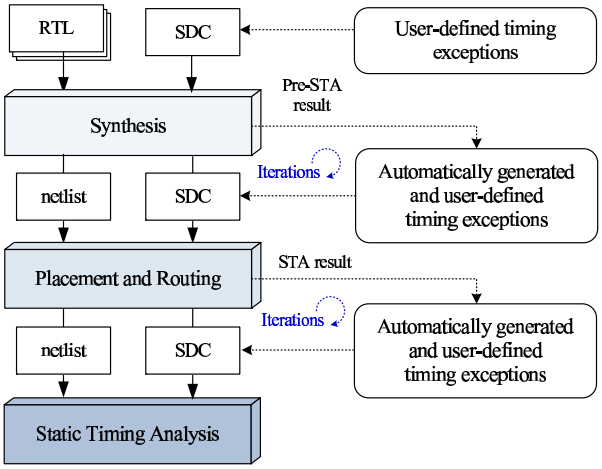

Fig. 2. Standard RTL-to-layout flow with timing exceptions.

In the SDC, timing exceptions for false paths and multicycle paths are defined using the keywords *'set_false_path'* and *'set_multicycle_path'*, respectively. Timing exception definitions can be categorized as shown in Figure 3.

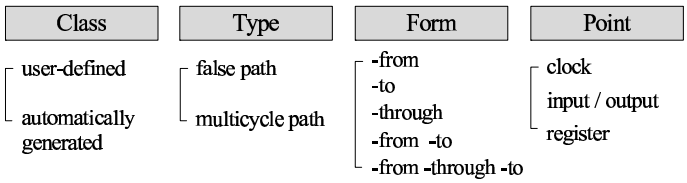

Fig. 3. Categorization of timing exceptions.

In the description of exceptions, paths are lists of "*-from*", "*-through*" and "*-to*" points. Each point can be a clock or input/output port, or a pin of a cell instance including registers (flip-flops) and any combinational cells.[1]

Figure 4 shows an example of timing exceptions in SDC. In this example, we declare three forms of exception – false paths with "*-through*" points, false paths without "*-through*" points, and multicycle paths between two registers. Multicycle paths are usually defined between two registers, between input/output points, or between different clock domains. In general, "*-through*" points are not used with multicycle path definitions.

Users typically define timing exceptions between different clock domains and false paths using only start (*-from*) or end (*-to*) points; these are often called user-defined false paths. Timing exceptions between registers or input/output ports can

[1]A port of hierarchical modules can also be a point for timing exception declaration. However, placement and routing tools are in general based on the assumption of flattened designs, so that hierarchical module ports can disappear during placement and routing. Therefore, it is preferred to use physically meaningful points, such as input/output pins of cells or primary input/output ports, to specify timing exceptions.

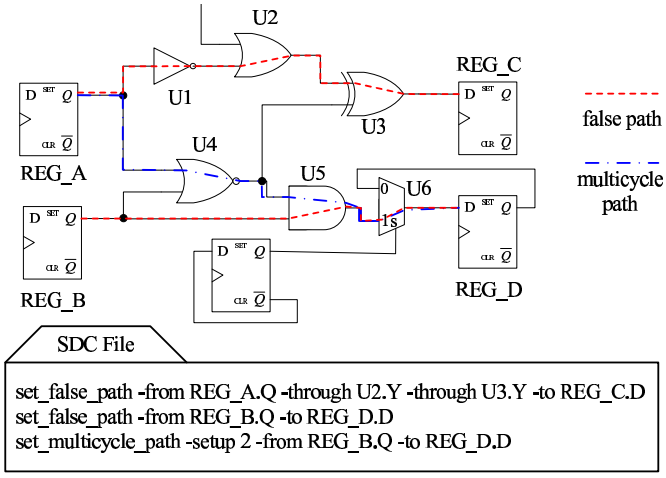

Fig. 4. Descriptions of exceptions according to the format.

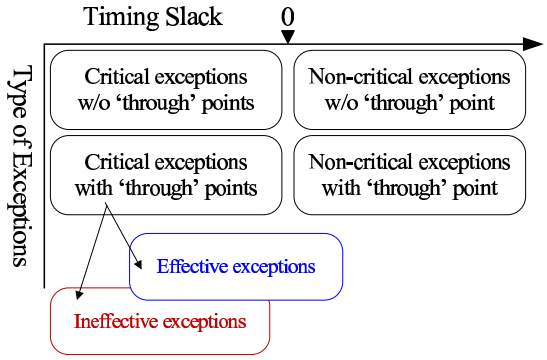

Fig. 5. Exceptions space according to format and criticality.

be additionally extracted and applied using commercial CAD tools. In this work, we focus on whether the generated timing exception is beneficial. We perform experiments on the timing exceptions which are defined between two registers or between a register and an input/output pin.

Our work is motivated by three driving questions.

(1) Do timing exceptions help or hurt? On the one hand, added exceptions help because constraints are removed from optimization. That is, exceptions reduce the number of timing problems that designers must be concerned with. False or multicycle paths reduce the number of negative-slack paths, so that timing closure is more easily achieved. In addition, reduced constraints prevent excessive optimization such as upsizing of cells on non-functional timing paths; this leads to reduction of design area.

On the other hand, added exceptions hurt because they add complexity in optimization and extra care-abouts in timing analysis and optimization processes. Moreover, from our experiments it appears that the EDA industry, by convention, preserves exceptions on nodes in a circuit throughout the optimization process. This preservation of exceptions on nodes prevents restructuring and reduces the design solution space: feasible solutions with restructuring are excluded from consideration.

Intuitively, there must be a tradeoff between setting timing exceptions and discarding timing exceptions. In Section III, we examine the impact of timing exceptions on the design optimization process in detail.

(2) Which exceptions give net benefit when inserted? Timing exceptions can be classified not only by their format but also by the timing criticality of the associated paths. Timing criticality implies negative timing slack on paths specified by the exceptions before the timing exceptions are applied. Hence, the 'exception space' has two dimensions as shown in Figure 5. Within this exception space, we examine which exceptions are effective versus ineffective for timing optimization. In Section IV, we seek to identify beneficial exceptions in this exception space.

(3) When should exceptions be identified and applied? There are several stages in the implementation flow – synthesis, placement, clock tree synthesis (CTS), and routing stages. Between stages, or within each stage, timing optimization can be executed several times. Our initial intuition regarding this question is that (i) higher benefit can be obtained when timing exceptions are extracted as late as possible, since more accurate timing information is available in later design stages with the convergence of spatial embedding and parasitics; on the other hand, (ii) higher benefit can be obtained when timing exceptions are applied as early as possible, since we can maximize the benefit of timing exceptions before performing excessive timing optimizations.

Figure 6 illustrates feasible scenarios to extract and apply exceptions. *DC*, *PLACE*, *PLACE – OPT*, *CTS*, *CTS – OPT*, and *ROUTE* respectively denote synthesis (using *Synopsys Design Compiler*), placement, timing optimization after placement, clock tree synthesis, timing optimization after clock tree synthesis, and routing. Exceptions extracted after the *DC* stage can be applied before the *PLACE*, *PLACE-OPT*, *CTS* and *CTS-OPT* stages. Exceptions extracted after the *PLACE-OPT* stage can be applied in advance to the following stages. However, if we apply exceptions at a stage which is far from the stage at which the exceptions were extracted, most of the exceptions with "through" points would not be feasible since the name or structure of a given point (in the exception specification) can be changed during optimization. In Section V, we experimentally determine when exceptions should be extracted and applied to achieve the best QOR results. Our experiments extract timing exceptions from each stage – after placement (*PLACE*), placement optimization (*PLACE-OPT*), CTS, CTS optimization (*CTS-OPT*) and routing (*ROUTE*) – and apply the exceptions to the next stage so as to identify the most beneficial design stages for extraction and application of exceptions.

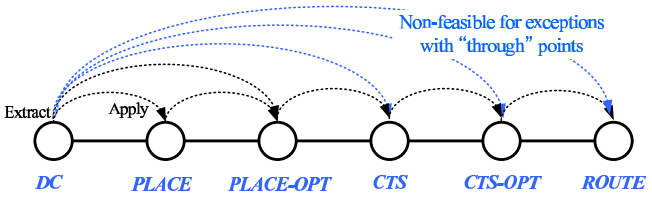

Fig. 6. Experiment design for extraction stage and application stage of timing exceptions.

III. Impact of Timing Exceptions in Optimization

The first driving question above elicits the hypothesis that added timing exceptions help remove overconstraints, but can increase optimization complexity. To prove this hypothesis, we perform experiments with the two types (false and multicycle paths) of exceptions. In addition, we study the impact of "through" points, which may prevent circuit restructuring and degrade timing results.

A. Difference of Timing Exceptions

We use a 4-bit ripple carry adder (RCA) as shown in Figure 7 to analyze the impact of timing exception types. A full adder circuit is shown in Figure 8.

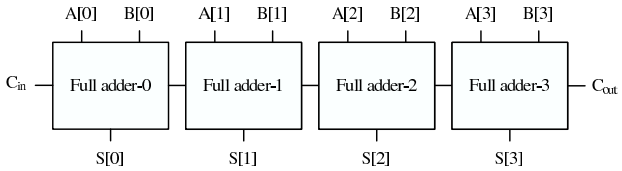

Fig. 7. 4-bit ripple carry adder.

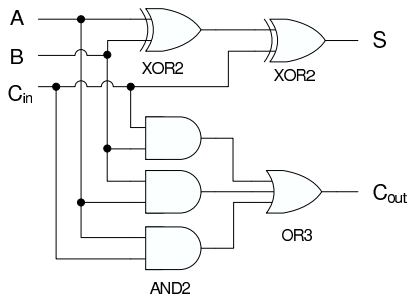

Fig. 8. Full-adder circuit in RCA.

We choose three timing paths arbitrarily, i.e., from $A[0]$ to C_{out}, from $B[0]$ to C_{out}, and from C_{in} to C_{out}, and define timing exceptions on these paths in different ways: (1) false paths (FP) with "through" points, (2) false paths without "through" points, (3) multicycle paths (MCP) with "through" points, and (4) multicycle paths without "through" points.

Applying these different types of exceptions, we analyze quality of optimization in three different commercial tools, i.e., *Synopsys DesignCompiler (DC)* [14], *Cadence SOC Encounter (SOCE)* [18], and *Synopsys Astro (ASTRO)* [16]. Table I shows the timing and area results from each tool. For *DC*, we measure timing and area after performing incremental timing optimization. For *SOCE* and *ASTRO*, we measure timing and area after performing placement and routing with timing optimizations.

TABLE I

Worst negative slack (*ns*) and area (um^2) of 4-bit ripple carry adder. FP and MCP denote false path and multicycle path, respectively.

		Without exceptions	FP w/ 'through'	FP w/o 'through'	MCP w/ 'through'	MCP w/o 'through'
DC	WNS	-0.040	-0.540	0.000	-0.540	0.000
	Area	244.8	166.3	196.9	166.3	196.9
SOCE	WNS	-0.059	-0.337	-0.035	-0.337	-0.035
	Area	226.4	201.6	233.3	201.6	233.3
Astro	WNS	-0.253	-0.290	-0.136	-0.290	-0.136
	Area	212.4	192.6	209.2	192.6	209.2

We also compare two different forms of timing exceptions – compact and equivalent complex form. Figure 9 shows three compact forms (upper) and their equivalent complex forms (lower). We use the *AES cipher* circuit (Table V) as a testcase. We take 20 start ("*-from*") points from the top-k critical paths, and we make equivalent complex-form exceptions by specifying all corresponding end ("*-to*") points. We define compact and complex-form exceptions for "*-to*" and "*-through*" cases in the same way.

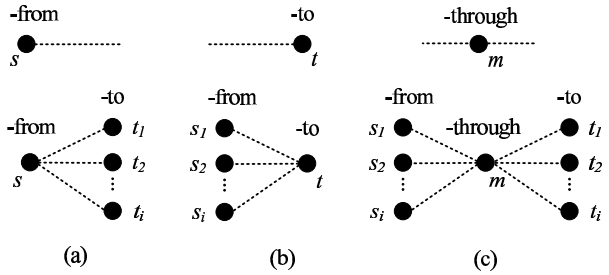

Fig. 9. Three compact forms and their equivalent forms in timing exceptions.

Table II shows optimization results after applying exceptions. The compact forms of timing exceptions lead to better timing and TAT results than complex forms. TNS is degraded with use of "through" exceptions in case (c).

TABLE II

Timing Slack and Area results after applying compact and equivalent complex form of timing exceptions.

Form	# FP	Before optimization		After optimization		
		WNS	TNS	WNS	TNS	Runtime
w/o FP	0	-0.401	-71.966	-0.242	-53.359	0:16:14
(a) compact	20	-0.356	-69.268	-0.234	-49.911	0:18:38
(a) complex	390	-0.356	-69.268	-0.235	-53.145	0:24:45
(b) compact	20	-0.319	-65.019	-0.238	-48.905	0:15:06
(b) complex	818	-0.319	-65.019	-0.236	-53.191	0:24:07
(c) compact	20	-0.383	-70.861	-0.257	-54.357	0:14:32
(c) complex	476	-0.383	-70.861	-0.240	-55.111	0:19:27

From the experimental results, we observe the following.

(1) Comparison of the results between "without exceptions" and "FP w/o through" or "MCP w/o through": timing

is improved and area is reduced. This is because three critical paths are removed due to timing exceptions.

(2) Comparison of the results between FP and MCP: FP and MCP have the same impacts on timing and area for all three optimization tools, *DC*, *SOCE* and *ASTRO*.

(3) Comparison of the results between "w/ through" and "w/o through": we observe that, in both FP and MCP cases, "w/ through" cases show significant degradation of timing, compared to "w/o through" cases.

(4) Comparison of the results between compact form and equivalent complex form, we observe that compact form is more beneficial than complex form considering timing slack and runtime.

The first and second observations are intuitive. However, the third and fourth observations need further analysis. In the following subsection, we analyze the third observation in detail.

B. Experiments on 'through' Points

From Table I, we also observe that the area from "w/ through" is significantly smaller than that from "w/o through". We can conclude that "through" points prevent aggressive optimization. The reason for timing degradation and smaller area due to "through" points may stem from the fact that optimization tools try to preserve the points given by timing exceptions. Hence, aggressive timing optimizations that sacrifice area, such as logic restructuring, are prevented for the points specified by timing exceptions. Figure 10 compares the resulting circuits after optimization with "through" points in (A) and without "through" points in (B). During optimizations without "through" points, cells in a critical path (from C_{in} to C_{out}) have been changed from $AND2-OR3$ to $AOI21-INV$. However, with "through" points, original circuit topology is maintained.

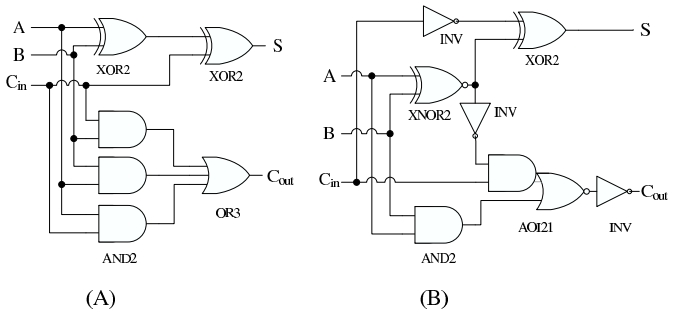

Fig. 10. Full-adder circuit after optimization with "through" points in (A) and without "through" points in (B).

It is difficult to analyze the impact of timing exceptions in large, real designs due to the complexity and uncontrollability of the number of false paths. Thus, we design a scalable artificial circuit as shown in Figure 11.

The circuit consists of cascaded unit circuits, with the unit circuit having one select port and four input-output pairs. The unit circuit contains two false paths, so that we can control the number of false paths easily by cascading the unit circuit.

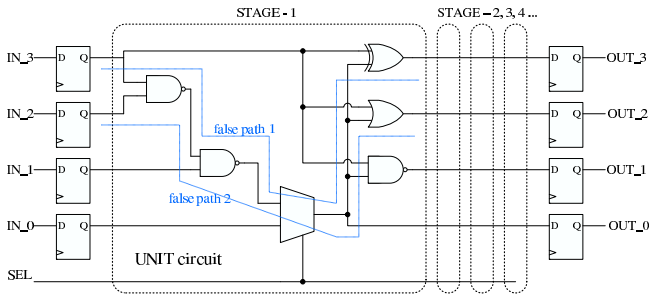

Fig. 11. Artificial circuit which is scalable by cascading a unit circuit.

To calculate the total number of paths, we count all incoming paths at the four output nodes of each stage. Initially, for 1-stage circuit, the numbers of incoming paths for output nodes are N_{1,OUT_3} = 5, N_{1,OUT_2} = 5, N_{1,OUT_1} = 5, and N_{1,OUT_0} = 5, respectively. Summing up the number of incoming paths of each output node gives total number of paths in the 1-stage circuit. The number of paths in n-stage circuits can be calculated recursively as:

$$N_{n,total} = 8N_{n-1,OUT_3} + 4N_{n-1,OUT_2} + 4N_{n-1,OUT_1} + 4N_{n-1,OUT_0}$$

where the number of incoming paths at the output nodes of $n-1$ (N_{n-1,OUT_3},...N_{n-1,OUT_0}) is also calculated recursively. The coefficients, i.e., 8 and 4, in the equation indicate the number of branches from the output nodes of $n-1$-stage circuit.

The number of true paths in the circuit is then calculated using the same method after removing false paths in the circuit, and finally, the number of false paths is calculated by subtracting the number of true paths from the total number of paths.

We apply different clock periods for each circuit, since the length of combinational paths is different and proportional to the number of stages. Table III summarizes the number of all paths and false paths in each artificial circuit with different stages.

TABLE III
THE NUMBER OF PATHS AND TARGET FREQUENCY IN THE ARTIFICIAL CIRCUIT.

	stage–1	stage–2	stage–4	stage–8
# of all paths	20	100	2,500	1,562,500
# of false paths	2	20	916	941,636
clock period (ns)	0.4	0.8	1.6	3.2

We perform placement and routing (P&R) for five cases, i.e., without false paths, with 25% of all false paths, with 50% of all false paths, with all false paths, and with multicycle path (MCP) exceptions only. In this experiment, we do not use "through" points. 25% and 50% false paths are selected according to the ascending order of timing slack values, i.e., top 25% and 50% of critical paths are selected. For P&R, we use a traditional timing-driven implementation flow with *Cadence SOC Encounter*, and a signoff timing analysis flow

with *Synopsys PrimeTime vB-2008.12-SP2* [15]. In the artificial circuits, real multicycle paths do not exist. So, we define four multicycle paths arbitrarily, i.e., from *IN_2* and *IN_3* to *OUT_2* and *OUT_3* among 16 register-to-register paths. After placement and routing (P&R), we measure worst negative slack (WNS), total negative slack (TNS), area, and runtime (TAT).

Table IV summarizes timing, area, and runtime with respect to the number of timing exceptions applied.

TABLE IV
QOR RESULTS AFTER P&R WITH SOC ENCOUNTER. FP DENOTES FALSE PATH TIMING EXCEPTIONS ARE APPLIED.

SOCE	QOR	0% FP	25% FP	50% FP	100% FP	MCP
stage-1	WNS (ns)	-0.314	-0.311	-0.311	-0.305	-0.262
	TNS (ns)	-1.100	-1.111	-1.111	-1.094	-0.948
	Area	254.7	268.1	268.1	252.6	280.8
stage-2	WNS (ns)	-0.233	-0.226	-0.189	-0.219	-0.182
	TNS (ns)	-0.786	-0.838	-0.721	-0.761	-0.676
	Area	352.8	293.5	311.2	288.6	320.3
stage-4	WNS (ns)	-0.042	-0.138	-0.132	-0.094	0.000
	TNS (ns)	-0.089	-0.449	-0.466	-0.288	0.000
	Area	460.8	360.6	370.4	378.9	452.3
stage-8	WNS (ns)	0.002	-0.459	-0.325	-0.310	0.000
	TNS (ns)	0.000	-1.729	-1.241	-1.185	0.000
	Area	581.4	503.1	505.9	512.3	601.9
	TAT	**0:03:37**	**0:24:38**	**1:09:25**	**6:53:33**	**0:02:51**

From the results, we observe that timing exceptions help reduce timing violations for small numbers of stages, stage-1 and stage-2, compared to the results for no exceptions; however, the improvement is quite small and not consistent. For large numbers of stages, stage-4 and stage-8, the timing violations with false paths are worse than those without false paths. In addition, for the stage-8 case, runtime increases excessively without improving timing compared to the case of no exceptions. When we apply MCP exceptions, QOR is improved.

Analysis of the circuit topology after optimization gives the same conclusion as with the RCA example: when many false paths are used, so that many "through" points are specified, restructuring is limited and timing is degraded. *SOCE* performs restructuring when timing exceptions without "through" points are applied, as shown in Column 3 (0% FP), and Column 7 (MCP) cases. Figure 12 shows the unit circuit as restructured by *SOCE*.

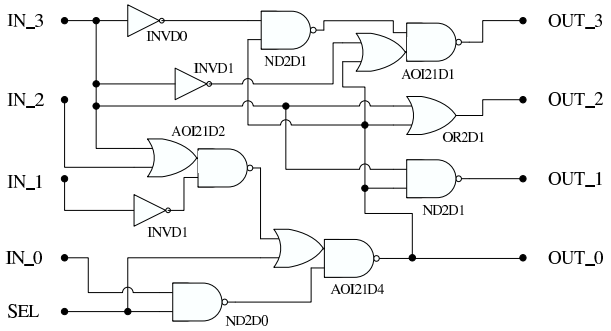

Fig. 12. Restructured unit circuit

In summary, timing exceptions without "through" points reduce timing slack and do not hurt optimization processes. By contrast, timing exceptions with "through" points may help reduce timing slack, but they limit the optimization (restructuring) solution space, so that overall optimization quality is degraded for the paths specified by such timing exceptions.

IV. BENEFICIAL TIMING EXCEPTIONS

Since a larger number of timing exceptions with "through" points degrade the optimization quality, we wish to reduce the number of exceptions with "through" points. In other words, we have to audit exceptions to obtain beneficial timing exceptions.

A. Critical Timing Exceptions

First, we examine the impact of critical exceptions (on negative timing slack paths) and non-critical exceptions (on positive slack paths) in the exception space (Figure 5). To compare the impact of critical and non-critical exceptions, we perform experiments with four different kinds of exceptions – critical false paths, critical multicycle paths, non-critical false paths and non-critical multicycle paths. We select 10,000 exceptions and apply them in the incremental optimization stage of *Synopsys DesignCompiler (DC) Y-2006.06-SP5* [14] We use multicycle paths without "through" points and false paths with "through" points. Figure 13 shows experimental results when we apply top-10%, 25%, 50% and 100% of exceptions in descending order of criticality (timing slack). Figure 13 shows the worst negative slack after the incremental optimization with different number of timing exceptions. From the figure, we observe that WNS is not improved by non-critical exceptions, and in fact remains the same as when no exceptions are applied (0% cases in Figure 13). However, critical false paths and multicycle paths improve timing slack. Critical timing exceptions without "through" points (Critical MCP) effectively reduce the worst timing slack, but critical timing exceptions with "through" points (Critical FP) do not significantly improve timing slack.

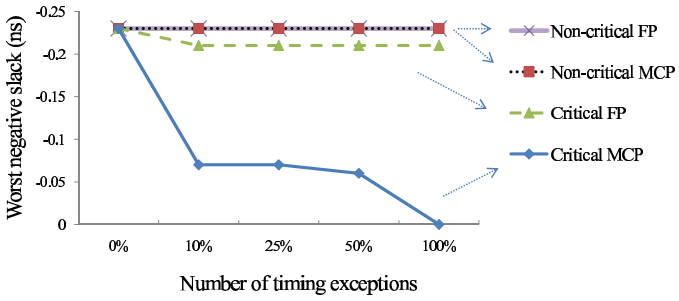

Fig. 13. Timing results (WNS) after applying critical and non-critical exceptions.

B. Effective Timing Exceptions

We can use critical exceptions rather than non-critical exceptions. However, not all critical exceptions are beneficial: the critical exceptions can be either *effective* or *ineffective* for

design timing optimization. Ineffective exceptions need to be avoided in design optimization.

Effective timing exceptions. To be beneficial in design optimization, timing exceptions need to (1) turn negative slack to positive, (2) improve large negative slack to the point where it can be turned to positive with a simple sizing optimization, or (3) improve timing slack of non-critical paths as well. The first condition enables direct improvement of timing quality without any extra optimization cost. The second condition allows improvement of timing quality without using aggressive optimization, so that the limitation on restructuring due to timing exceptions may not affect the final timing quality. The third condition is expected to reduce design area.

Ineffective timing exceptions. Some timing exceptions do not help to reduce timing slack. In Figure 14, whether or not we define path A as a false path, the timing slack for each timing point (all input/output pins of cells, or primary ports) of the path will not change if the path B is a true path and has tighter timing requirement (worse timing slack) than path A. Therefore, adding a timing exception for path A will only prevent restructuring that may be required for path B to be optimized. Even if timing exceptions improve timing slack, they can be regarded as *ineffective* if their timing improvement is small and the timing slack is still critical, so that aggressive timing optimization is required. In this case, timing degradation from restrictions on optimization due to timing exceptions can outweigh any timing improvement from the timing exceptions themselves.

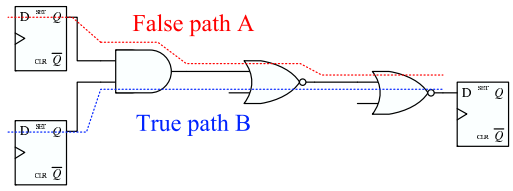

Fig. 14. Timing exception on Path A is an ineffective false path due to the presence of true path B.

To filter out ineffective timing exceptions from the generated set of exceptions, we propose the following metrics – *MaxImp*, *SumImp*, *AvgImp* and *EndImp* – to quantify the effectiveness of timing exceptions. Formally,

$$MaxImp = Max_{c \in p}(s'_c - s_c) \geq \alpha_{max} \quad (1)$$

$$SumImp = \sum_{c \in p}(s'_c - s_c) \geq \alpha_{sum} \quad (2)$$

$$AvgImp = \frac{\sum_{c \in p}(s'_c - s_c)}{n_{c \in p}} \geq \alpha_{avg} \quad (3)$$

$$EndImp = s'_{c_{end}} \geq \alpha_{end} \quad (4)$$

where p is a path that is specified by a false path, c is a timing point belonging to the path p, and c_{end} is an endpoint of the path p. Let s_c and s'_c respectively denote the timing slack of the timing point c before and after defining path p as a false path. Let α_{max}, α_{sum}, α_{avg} and α_{end} be threshold values for each metric. It is difficult to quantify the timing benefit from restructuring until we evaluate all possible technology mappings during timing optimization. Hence, we introduce a threshold value that trades off the effectiveness of false paths against the benefit that may be obtained if restructuring occurs.

- *MaxImp* prunes any ineffective path that does not give a slack improvement greater than $alpha_{max}$ at any node in the path. The underlying assumption of the *MaxImp* filter is that if no cells in a path have improvements larger than a given small threshold, all timing arcs in the path may be overlapped with true critical paths as shown in Figure 14.
- *SumImp* (*AvgImp*) checks the sum (average) of slack improvements of all timing arcs in a timing path. Underlying assumptions of these filters are that if the total slack improvement along a path is less than a given small threshold, the path may be still critical after being specified as a false path. Hence, the path still requires aggressive optimization but the optimization will be limited if it is specified as a false path.
- *EndImp* checks the timing slack improvement at the endpoint of a path. If an endpoint slack of an exception is too small or not improved, the path can be considered as an ineffective exception for the same reason as with *SumImp* or *AvgImp*.

We apply selected false paths from the above four metrics to our testcases. Table V summarizes the area and timing information of our testcases, implemented without using timing exceptions.

TABLE V
TESTCASES FOR EXPERIMENTS.

Module	Description	Cell count	Area (um^2)	TNS
AES	AES Cipher	26,000	75,000	-53.36
JPEG	JPEG Encoder	70,000	180,000	-93.54
LSU	Load Store Unit	26,000	112,000	-44.97
EXU	Integer Execution Unit	22,000	76,000	-36.16

We extract 10,000 false paths from our testcase with *SpyGlass-TXV v4.2* [17]. From the 10,000 false paths, we obtain the various number of effective exceptions with the proposed metrics using Tcl-script applicable to *Synopsys PrimeTime* [15]. We apply the selected exceptions at the placement optimization stage of *Cadence SOC Encounter (SOCE)* [18], and we observe timing improvements after timing optimization.

Table VI summarizes worst negative slack (WNS), total negative slack (TNS), and optimization runtime of our testcases after applying different number of exceptions. From the results, we observe that runtime increases according to the number of exceptions. Runtime with 10,000 false paths is up to 16 times larger than that without false paths. Even though the optimization runtime increases, the timing slack does not change accordingly in any of our metrics and testcases. In

addition, even though there is a improvement, it is difficult to distinguish the benefits of timing exceptions from the 'inherent noise' in IC implementation tools [11], since the improvement is quite small (e.g., maximum WNS and TNS improvements of *AES* testcase against the "0" exceptions are only 12*ps* and 2.05*ns*, respectively).

V. Design Stages to Extract and Apply Timing Exceptions

We find which design stages are most beneficial to extract and to apply timing exceptions. To maximize the benefit from timing exceptions, we may extract exceptions in very early design stages, e.g., right after synthesis, and use the exceptions in the later design stages. However, there is a problem with timing mismatch between design stages. Figures 15, 16, 17 and 18 compare timing slack of topmost critical paths (specified between registers) between different design stages. Figure 15 shows timing correlation between synthesis and placement, and Figure 16 shows timing correlation between placement and CTS. Figures 17 and 18 show timing correlation between placement and CTS and between CTS and routing, respectively. We observe that timing between synthesis and placement stages is not well correlated, so that the timing exceptions based on the synthesized netlists may no longer be valid for the later design stages. However, after placement, the timing correlation between stages is improved to more than 0.7.

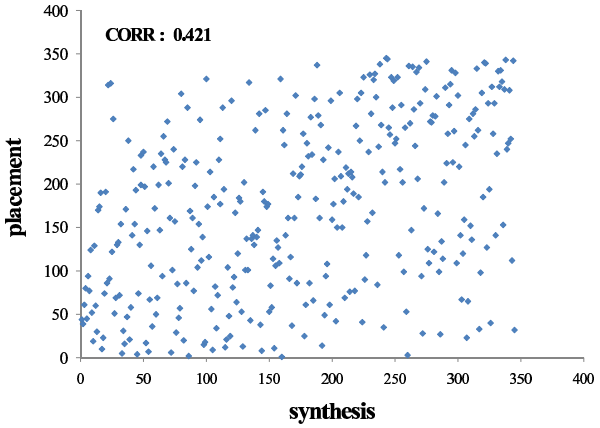

Fig. 15. Timing correlation between synthesis and placement.

We select 1,000 false paths which have the largest improvements with respect to the *MaxImp* metric at each stage. We apply these exceptions to the next stages, and examine QOR (WNS, TNS and Area) after placement and routing.

Table VII summarizes the QOR improvements when timing exceptions are extracted and applied at different design stages. We observe that total negative slack (TNS) is improved, when exceptions are extracted after placement or the first post-placement optimization (Place-opt) stages.

To improve the results further, it may better to extract exceptions at each stage and apply them in the next stage. However, extracting timing exceptions at every stage requires

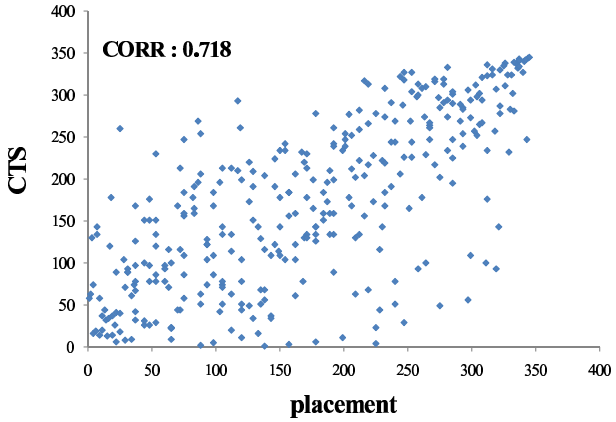

Fig. 16. Timing correlation between placement and CTS.

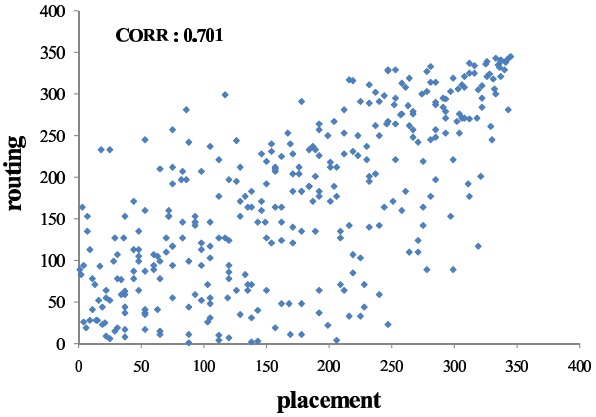

Fig. 17. Timing correlation between placement and routing.

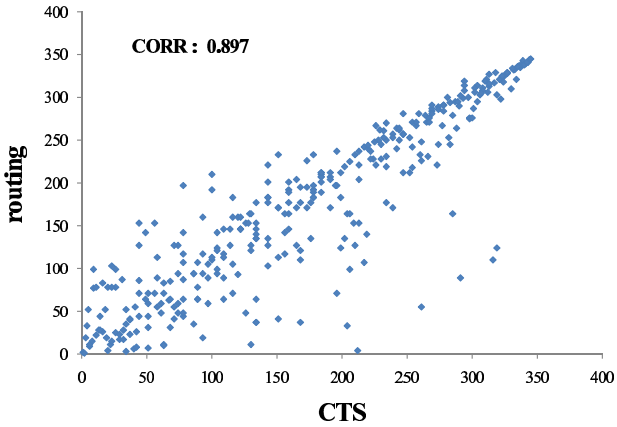

Fig. 18. Timing correlation between CTS and routing.

TABLE VI

QOR RESULTS OF TEST MODULES AFTER APPLYING DIFFERENT NUMBER OF EFFECTIVE EXCEPTIONS.

Filter	# of FP	AES			JPEG			LSU			EXU		
		WNS	TNS	Runtime	WNS	TNS	Runtime	WNS	TNS	Runtime	WNS	TNS	Runtime
MAX	10,000	-0.244	-52.56	1:27:29	-0.207	-111.89	1:53:56	-0.091	-37.99	0:25:30	-0.147	-41.61	0:54:15
	2,000	-0.245	-53.33	0:16:12	-0.171	-93.70	0:37:28	-0.098	-43.49	0:06:00	-0.133	-40.14	0:09:21
	1,000	-0.268	-53.62	0:11:30	-0.174	-94.07	0:23:23	-0.094	-39.96	0:05:57	-0.169	-42.30	0:04:32
	500	-0.251	-52.91	0:10:31	-0.185	-95.35	0:20:59	-0.088	-42.19	0:04:24	-0.128	-37.35	0:03:39
	100	-0.251	-53.09	0:11:01	-0.181	-94.08	0:19:02	-0.088	-40.99	0:03:27	-0.159	-35.48	0:02:49
	50	-0.247	-53.29	0:09:05	-0.184	-93.53	0:18:39	-0.097	-44.78	0:02:50	-0.148	-39.90	0:02:38
	0	-0.242	-53.36	0:09:32	-0.174	-93.54	0:17:39	-0.095	-44.97	0:02:46	-0.127	-36.16	0:03:24
SUM	10,000	-0.244	-52.56	1:27:29	-0.207	-111.89	1:53:56	-0.091	-37.99	0:25:30	-0.147	-41.61	0:54:15
	2,000	-0.231	-52.62	0:22:14	-0.170	-93.62	0:26:27	-0.112	-45.32	0:05:26	-0.141	-38.21	0:07:41
	1,000	-0.230	-51.31	0:19:29	-0.178	-96.16	0:22:33	-0.101	-41.50	0:03:54	-0.142	-41.09	0:05:28
	500	-0.244	-52.22	0:17:41	-0.191	-96.37	0:19:51	-0.094	-43.31	0:03:43	-0.123	-36.74	0:04:54
	100	-0.245	-52.94	0:09:33	-0.175	-94.73	0:18:37	-0.093	-44.68	0:03:04	-0.164	-37.04	0:03:04
	50	-0.247	-54.56	0:09:32	-0.175	-94.73	0:18:31	-0.093	-44.68	0:02:54	-0.121	-38.42	0:02:56
	0	-0.242	-53.36	0:09:32	-0.174	-93.54	0:17:39	-0.095	-44.97	0:02:46	-0.127	-36.16	0:03:24
AVG	10,000	-0.244	-52.56	1:27:29	-0.207	-111.89	1:53:56	-0.091	-37.99	0:25:30	-0.147	-41.61	0:54:15
	2,000	-0.251	-53.28	0:15:20	-0.170	-90.63	0:28:53	-0.104	-44.33	0:05:11	-0.127	-37.65	0:07:35
	1,000	-0.255	-54.22	0:12:58	-0.177	-96.78	0:23:03	-0.094	-44.33	0:03:37	-0.127	-38.59	0:05:20
	500	-0.237	-52.08	0:16:40	-0.191	-99.00	0:20:09	-0.089	-44.60	0:03:18	-0.161	-38.37	0:04:31
	100	-0.236	-53.17	0:14:28	-0.175	-94.73	0:18:36	-0.092	-46.35	0:03:01	-0.142	-42.00	0:03:25
	50	-0.241	-52.88	0:10:11	-0.175	-94.73	0:18:30	-0.095	-43.88	0:03:01	-0.144	-38.84	0:02:48
	0	-0.242	-53.36	0:09:32	-0.174	-93.54	0:17:39	-0.095	-44.97	0:02:46	-0.127	-36.16	0:03:24
END	10,000	-0.244	-52.56	1:27:29	-0.207	-111.89	1:53:56	-0.091	-37.99	0:25:30	-0.147	-41.61	0:54:15
	2,000	-0.246	-54.46	0:14:45	-0.178	-94.13	0:22:28	-0.094	-43.83	0:06:02	-0.139	-40.64	0:06:24
	1,000	-0.257	-52.16	0:18:42	-0.184	-98.25	0:14:41	-0.103	-41.26	0:03:52	-0.150	-37.46	0:04:21
	500	-0.237	-51.44	0:11:25	-0.164	-92.65	0:19:57	-0.090	-43.72	0:03:13	-0.134	-36.59	0:04:03
	100	-0.240	-53.67	0:10:39	-0.184	-98.57	0:13:14	-0.097	-44.42	0:03:01	-0.118	-29.83	0:05:20
	50	-0.251	-52.46	0:09:24	-0.176	-94.00	0:13:07	-0.091	-41.38	0:02:58	-0.142	-39.50	0:03:37
	0	-0.242	-53.36	0:09:32	-0.174	-93.54	0:17:39	-0.095	-44.97	0:02:46	-0.127	-36.16	0:03:24

TABLE VII

QOR RESULTS AFTER APPLYING FALSE-PATH EXCEPTIONS IN EACH STAGE.

Module	Stage	Area	WNS	TNS
AES	Placement	74,090	-0.208	-41.75
	Place-opt	75,287	-0.208	-40.09
	CTS	75,494	-0.204	-40.84
	CTS-opt	76,199	-0.197	-40.05
	Routing	76,245	-0.200	-40.12
JPEG	Placement	224,418	-0.148	-70.59
	Place-opt	220,105	-0.157	-83.49
	CTS	221,615	-0.166	-83.22
	CTS-opt	221,520	-0.162	-83.73
	Routing	221,537	-0.156	-83.83
LSU	Placement	118,361	-0.183	-75.38
	Place-opt	118,395	-0.153	-69.78
	CTS	118,877	-0.179	-73.23
	CTS-opt	118,784	-0.171	-69.06
	Routing	118,774	-0.160	-69.95
EXU	Placement	76,263	-0.295	-36.06
	Place-opt	76,491	-0.292	-48.21
	CTS	77,041	-0.273	-39.05
	CTS-opt	76,589	-0.323	-41.10
	Routing	76,611	-0.314	-41.93

large runtime and effort. Based on timing correlation results between stages and optimization results in Table VII, we can conclude that timing exceptions need to be extracted after placement or the first post-placement optimization stages, when most timing-critical paths are discovered.

VI. GUIDELINES TO TIMING EXCEPTIONS IN DESIGN OPTIMIZATION

We show the impact of timing exceptions in design optimization and when timing exceptions need to be extracted and applied. Based on the observations from our experiments, we provide the following guidelines for the use of timing exceptions in design optimization.

- Optimization runtime increases rapidly with increasing number of applied timing exceptions. However, such large runtime overhead does not lead to better QOR. Hence, in design optimization, designers need to use only clearly effective timing exceptions.
- First-order timing exceptions that are always beneficial are those without "through" points. With "through" points, the optimization quality is not improved significantly, and can even be degraded.
- When specifying timing exceptions, the declaration form must be as compact as possible.
- Timing exceptions should be extracted after placement or the first post-placement optimization stages, when most of the design's critical paths have been revealed, and most of the required restructuring has already been performed.
- It is better to not use timing exceptions with "through" points in design optimization, unless the timing slack improvement is clear and sufficient.

Our observations imply the following requirements: (1) new design optimization tools or flows, such as exception-aware timing optimization that maintains timing exceptions throughout the flow without preserving the exact points in the netlist specified by timing exceptions, and (2) high-quality constraint compaction tools to compress the number of exception points.

For now, in the context of the traditional timing-driven design methodology, we suggest a design flow to more effec-

tively use timing exceptions in design optimization. Figure 19 illustrates the proposed design flow.

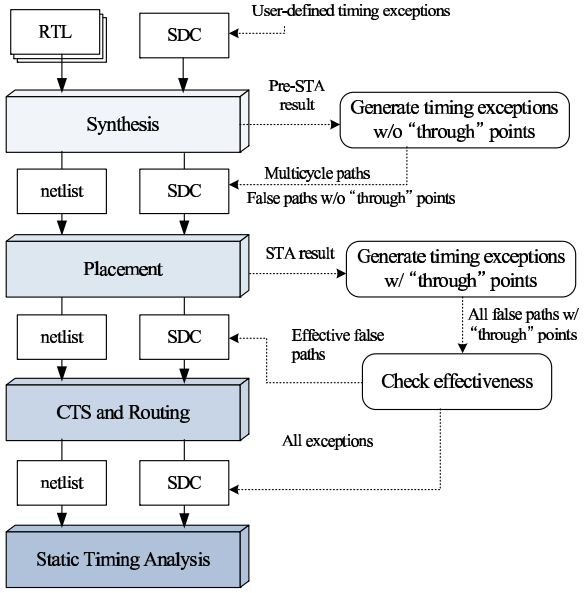

Fig. 19. Our recommended flow for timing exceptions.

According to the proposed flow, when synthesizing RTL codes into gate-level netlists, user-defined timing exceptions which give clear benefits, such as exceptions between different clock domains, can be added to the SDC (*Synopsys* Design Constraints). After synthesis, false paths and multicycle paths that do not have "through" points can be generated and appended into the SDC. We use *Atrenta SpyGlass-TXV* [17] flow for automatic generation of timing exceptions. After placement (or placement optimization), we extract exceptions with "through" points. However, additional 'auditing' of exceptions may be required to find effective ones, since applying too many exceptions with "through" points is harmful to TAT and QOR. Finally, at the static timing analysis (STA) stage, all exceptions can be used, so as to reduce the number of timing violations.

VII. CONCLUSION

Adding timing exceptions is traditionally regarded as helpful for design optimization because overconstraints are removed by the exceptions. However, timing exceptions are not always beneficial and can even degrade quality of results.

In this work, we evaluate the impact of timing exceptions with respect to timing violations, area and optimization runtime. We also explore filtering methods to identify beneficial timing exceptions, and we identify a 'sweet spot' in the design flow when exceptions should be extracted and applied for maximum benefit. We have furthermore proposed a design methodology for beneficial insertion of timing exceptions: (1) critical and effective timing exceptions should be extracted and applied after the placement stage, and (2) ineffective false paths should be pruned for better QOR. We believe that our motivating questions, experimental framework, and initial results can serve as a foundation for the development of novel methodologies that maximally leverage timing exceptions in the IC implementation flow.

Our ongoing work studies several other facets of timing exceptions. First, we seek improved ways of extracting and auditing consistently beneficial exceptions, to improve design QOR and TAT. Second, we seek quantified metrics of both user- and automatically-defined exceptions, to better predict their impact on final design outcomes. Finally, we continue to pursue production-quality timing exception methodologies for general SOC implementation flows.

VIII. ACKNOWLEDGMENTS

The authors thank Neelmani Kumar for useful discussions in coming up with the artificial scalable circuit of Section III.B, and Atrenta, Inc. for support of a timing exception flow.

REFERENCES

[1] D. Brand and V. S. Iyengar, "Timing Analysis Using Functional Relationships", *Proc. ACM/IEEE International Conference on Computer-Aided Design*, 1986, pp. 126-129.
[2] H.-C. Chen and D. H.-C. Du, "Path Sensitization in Critical Path Problem", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 12(2), 1993, pp. 196-207.
[3] P. C. McGeer, A. Saldanha, R. K. Brayton and A. Sangiovanni-Vincentelli, "Delay Models and Exact Timing Analysis", in T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993, pp. 167-189.
[4] S. Devadas, K. Keutzer, S. Malik and A. Wang, "Certified Timing Verification and the Transition Delay of a Logic Circuit", *Proc. ACM/IEEE Design Automation Conference*, 1992, pp. 549-555.
[5] A. P. Gupta and D. P. Siewiorek, "Automated Multi-Cycle Symbolic Timing Verification of Microprocessor-based Designs", *Proc. ACM/IEEE Design Automation Conference*, 1994, pp. 113-119.
[6] K. Nakamura, K. Takagi, S. Kimura and K. Watanabe, "Waiting False Path Analysis of Sequential Logic Circuits for Performance Optimization", *Proc. ACM/IEEE International Conference on Computer-Aided Design*, 1998, pp. 392-395.
[7] W.-C. Lai, A. Krstic and K.-T. Cheng, "Functionally Testable Path Delay Faults on a Microprocessor", *IEEE Design & Test of Computers* 17(4), 2000, pp. 6-14.
[8] J.-J. Liou, A. Krstic, L.-C. Wang and K.-T. Cheng, "False-Path-Aware Statistical Timing Analysis and Efficient Path Selection for Delay Testing and Timing Validation", *Proc. ACM/IEEE Design Automation Conference*, 2002, pp. 566-569.
[9] H. Higuchi, "An Implication-Based Method to Detect Multi-Cycle Paths in Large Sequential Circuits", *Proc. ACM/IEEE Design Automation Conference*, 2002, pp. 164-169.
[10] K. Yang and K.-T. Cheng, "Efficient Identification of Multi-Cycle False Path", *Proc. Asia South Pacific Design Automation Conference*, 2006, pp. 360-365.
[11] A. B. Kahng and S. Mantik, "Measurement of Inherent Noise in EDA Tools", *Proc. International Symposium on Quality in Electronic Design*, 2002, pp. 206-211.
[12] *Using the Synopsys Design Constraints Format Application Note*, http://solvnet.synopsys.com/.
[13] *Sun OpenSPARC Project.*, http://www.sun.com/processors/opensparc/.
[14] *Synopsys DesignCompiler User's Manual.* http://www.synopsys.com/.
[15] *Synopsys PrimeTime User's Manual.* http://www.synopsys.com/.
[16] *Synopsys Astro User's Manual.* http://www.synopsys.com/.
[17] *Atrenta SpyGlass User's Manual.* http://www.atrenta.com/.
[18] *Cadence SOC Encounter User's Manual.* http://www.cadence.com/.
[19] *Magma Blast Fusion User's Manual.* http://www.magma-da.com/.
[20] *Blue Pearl Software, Cobalt.* http://www.bluepearlsoftware.com/.
[21] *FishTail Design Automation.* http://www.fishtail-da.com/.