TAU 2018 Contest

Efficient generation of timing reports from an STA graph with updated arrival and required times

**Contest Rules**

v1.0 – Dec 17*th*, 2017

http://www.tauworkshop.com/2018/contest.html

**Contents**

|  |  |  |
| --- | --- | --- |
| [1](#page1) | [Introduction](#page1) | [1](#page1) |
| [2](#page1) | [The TAU 2018 Contest Problem](#page1) | [1](#page1) |
| [3](#page2) | [Provided Code and Documentation](#page2) | [2](#page2) |
| [3.1](#page2) | [Documentation](#page2) | [2](#page2) |
| [3.2](#page3) | [C++ Code](#page3) | [3](#page3) |
| [4](#page4) | [Contest Deliverables](#page4) | [4](#page4) |
| [4.1](#page4) | [Executable](#page4) | [4](#page4) |
| [4.2](#page5) | [Documentation](#page5) | [5](#page5) |
| [5](#page5) | [Evaluation Criteria](#page5) | [5](#page5) |
| [6](#page6) | [Evaluation Framework](#page6) | [6](#page6) |
| [7](#page6) | [Updates](#page6) | [6](#page6) |

**1 Introduction**

This document will outline the TAU 2018 Contest’s deliverables, as well as provide the official contest problem and the evaluation criteria and framework.

**2** **The TAU 2018 Contest Problem**

Given a (flat) circuit design, you are expected to build an accurate, efficient and scalable reporting architecture that provides extensive path reporting capability to the STA users. The specific objective is to build a runtime and memory efficient timing path enumeration and reporting capability which allows user to generate top N critical timing reports on a specific cone of logic.

Figure 1 illustrates the sample netlist and describes the requirement in more detail deliverable on a simple circuit. Given a design (encapsulated in design.v) with input parasitics file and clock constraint, you are expected to generate timing report for a huge number of path specific reporting commands that goes through user specified from/through/to nodes of the design.

Use-model: report\_timing

[–max\_paths ‘M’]

[–nworst ‘N’]

[–from <begin\_pin\_list>] # starting pins of the path (register/CK or input\_port)

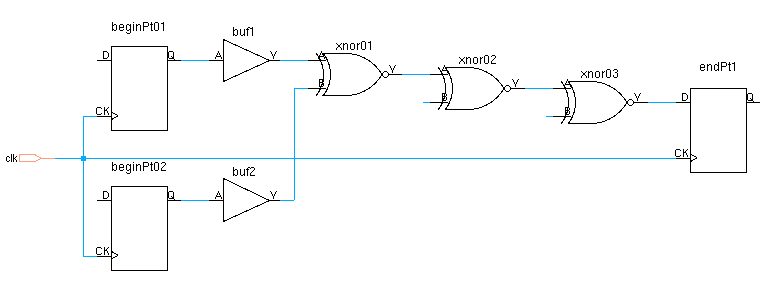
[-through <intermediate\_pin\_list>] # any intermediate pin on the path

[-to < endpoint\_pin\_list>] #endpoints of the path (register/D or output\_port)

The command should report the ‘M’ worst paths of the design, with maximum of nworst (-nworst ) paths to any single endpoint (default 1). The reported paths should be in slack sorted order.

If –from/through/to are specified then the reported paths should be through those pins only.

Following example illustrates the requirement.



Input Command 1: report\_timing –from beginPt01/CK –through xnor01/A

Output:

Path 1: beginPt01/CK -> endPt1/D (Slack:3.445)

Required Time 4.752

Arrival Time 1.307

Slack Time 3.445

----------------------------------------------------

Delay Arrival Edge Pin

----------------------------------------------------

- 1.000 v beginPt01/CK

0.140 1.140 v beginPt01/Q

0.010 1.150 v buf1/A

0.010 1.160 v buf1/Y

0.010 1.170 v xnor01/A

0.010 1.180 ^ xnor01/Y

:

0.000 1.307 ^ endPt1/D

----------------------------------------------------

Input Command 2: report\_timing –from beginPt02/CK –through xnor01/B

Output:

Path 1: beginPt02/CK -> endPt1/D (Slack:3.246)

Required Time 4.752

Arrival Time 1.507

Slack Time 3.245

----------------------------------------------------

Delay Arrival Edge Pin

----------------------------------------------------

- 1.100 v beginPt01/CK

0.140 1.240 v beginPt01/Q

0.010 1.250 v buf2/A

0.110 1.360 v buf2/Y

0.010 1.370 v xnor01/B

0.010 1.380 ^ xnor01/Y

:

0.000 1.507 ^ endPt1/D

----------------------------------------------------

Input Command 3: report\_timing –from beginPt02/CK –through xnor01/A

Output:

No timing path exists for given specification.

The above reported output format is just a suggestion; participants can modify it according to their need. However, the resulting format should report the path startpoint, endpoint, arrival time, required time, and slack in the header and along the nodes of the path.

The evaluation will be done according to a large batch of different from/through/to combinations and the score will be based upon a combination of:

1. Accuracy

2 set of timing reports will be generated and compared for accuracy.

The timing of from/through/to based reports will be compared with the timing w/o –from/through command. For example:

report\_timing –max\_paths 1000 –nworst 1000 –to <end\_point> [Report 1]

report\_timing –nworst 1 –to <end\_point> -from |through [Report 2]

The slack/nodes of the given path of Report 2 should match one of the path of Report 1. Also that should be the worst path going through these nodes in Report 1.

Similar data will also be generated from an industry reference tool and shall be compared against.

1. Performance

After the initial timing update, the runtime and memory requirements of large max\_paths and large number of from/through/to commands reporting runtime will be quantified.

Expectation is that the cost should be quite low as compared to initial timing update and the solution should scale well with increased number of CPUs.

**3 Provided Code and Documentation**

To aid with conceptual understanding and code development, the contest committee have provided the following: (All files are posted on the contest website <https://sites.google.com/view/taucontest2018)>

**3.1** **Documentation**

These files are references, and are intended to provide sufficient standalone education for this contest. However, you should not limit yourselves to only these documents, and are encouraged to research other sources.

1. contest\_rules.pdf: This document explains the basics of the contest, including the problem and evaluation expectations.
2. contest\_education.pdf: This document provides the conceptual understanding necessary for timing analysis, common path pessimism removal (CPPR), delay calculations, and macro modeling.
3. contest\_file\_formats.pdf: This document explains the different file formats used as input as well as the expected output file formats.

**3.2** **C++ Code**

These code bundles are provided by various external sources. They are available *as is* and are intended as aids. You may choose not to use any of these, use a portion, or use all of the existing code.

3

1. NTUTimer.tar.gz: This is the source code from the winners of the PATMOS 2011 Timing Analysis Contest (<http://patmos-tac.inesc-id.pt/news.shtml).>
2. IITimer.zip: This is the source code from the winners of the TAU 2013 Variation Aware Timing Contest ([https://sites.google.com/site/taucontest2013).](https://sites.google.com/site/taucontest2013)
3. UI-Timer.tar.gz: This is the source code from the winners of the TAU 2014 Contest on Removing Pessimism during Timing Analysis (CPPR) [(https://sites.google.com/site/taucontest2014).](https://sites.google.com/site/taucontest2014)
4. iTimerC.gz: This is the binary from the winners of the TAU 2015 Contest on Incremental Timing and CPPR Analysis ([https://sites.google.com/site/taucontest2015).](https://sites.google.com/site/taucontest2015)
5. parser helpers.tar.gz: This is the source code from the ISPD 2013 Discrete Gate Sizing Contest ([http://www.ispd.cc/contests/13/ispd2013 contest.html).](http://www.ispd.cc/contests/13/ispd2013_contest.html) Included are parsers for Liberty and SPEF file formats.
6. OpenTimer.tar.gz: This is the source code for the reference timer (will be used as the golden timer for the TAU 2016 Contest). We recommend comparing your timing reports to that of the original timing reports generated by OpenTimer.

**4** **Contest Deliverables**

Contestants should submit (*i*) a **binary** (ideally static) that is compatible with the evaluation machine (see Section 5) and (*ii*) a brief document detailing their implementation. Contestants are welcome to use any or none of the provided code bases.

**4.1 Executable**

The submitted binary will be expected to take in the following format:

%*>* update\_timing *<*.tau2018*> <*.timing*>*

%> generate\_reports <reporting\_commands file> *<*output file name*>*

Each command file can contain 1 or more report\_timing commands:

report\_timing *<*params>

…

report\_timing *<*params>

* *<*.tau2018*>*:**input**file that contains the benchmark information, including the netlist (**.v**),the parasitics (**.spef**), and the cell libraries (**.lib**).
* *<*.timing*>*:**input**file that describes the timing clock constraint specified to the primaryinputs of the design.
* *<*output file name*>*:**outfile file name**string that will contain the timing report generated from the specified command. In case of multiple commands in the command file, we should be able to append the output to the same file.
* If you choose to call the standalone version of OpenTimer or your choice of timer, you may include an additional binary in your submission. However, it is your responsibility **ahead of time** to ensure that the additional binary is compatible on the evaluation machine.

**4.2** **Documentation**

A 1-2 page PDF document should outline the method of your approach. Figures and images are encouraged when writing up the ideas. A suggested set of topics would be: (*i*) Introduction and short description of the problem, (*ii*) your general approach to solving the problem, (*iii*) key ideas that you would like to highlight, including implementation details, and (*iv*) trade-offs between accuracy and runtime that you consciously decided on. Feel free to expand on this, and add more content. For instance, you may also discuss challenges that you faced, or if given more time, other improvements you would have liked to try.

There is no official document format or template, but you may model it off of any research paper format, e.g., ACM/IEEE conference or journal publication, or report format, e.g., any released contest document.

**5** **Evaluation Criteria**

For each benchmark, contest submissions will receive a *weighted accuracy* score, which will be a combination of **accuracy**, **runtime** and **memory usage**. The accuracy portion will be based on numerical timing values (e.g., slack). As a golden reference, we will also compare the results to standard Industry tool. After accuracy is evaluated, additional runtime factor and memory factor, and size factor modifiers will be applied based on model generation and usage performance, and model size. The final contest ranking will be the average of such weighted accuracies across the set of tested benchmarks. The exact balance of the evaluation criteria is subject to change; further details will be published and updated at a later date.

* **Accuracy**: for timing evaluation, we will compare(*i*) the slacks and (*ii*) the timing path including the edges. Every timing report value will be generated using OpenTimer.

5

* **Runtime**: the contestants will be evaluated both on the amount of time used to generatethe timing reports, as well as the amount of time required for timing analysis when using the macro model by the golden timer.
* **Memory**: the contestants will be evaluated both on the amount of memory used togenerate the timing reports, as well as the amount of memory required for timing analysis when using the macro model by the golden timer.

**Computational Infrastructure.** The submissions will be evaluated tentatively on a machinewith the following characteristics:

* 8× Intel(R) Xeon(R) CPU E7- 8837 @2.67GHz, 24576 KB cache
* *>*24GB of RAM

The following software is installed:

* Red Hat Linux 5
* POSIX Threads;

**The utilization of parallelization techniques (especially multithreaded) is strongly encouraged**. The evaluation machine can execute a maximum of 8 threads concurrently.

**6** **Evaluation Framework**

For each benchmark, contestants will be expected to generate two set of timing reports on evaluationdesigns — one w/o from/through specification which generates large number of paths in slack sorted order and the other report will have specific o/p for given from/through/to parameters.

1. **Updates**
   1. 2017/12/17 Initial version, Amit D .

6