**PROJECT REPORT**

**Static timing analysis of basic circuits using OpenTimer**

**by**

**CHINMAY KHARTADKAR**

**Contents**

[1. Chapter 1 4](#_Toc45957909)

[**Introduction** 4](#_Toc45957910)

[1.1. Purpose 4](#_Toc45957911)

[1.2. Few important definitions 5](#_Toc45957912)

[ The critical path is defined as the path between an input and an output with the maximum delay. 5](#_Toc45957913)

[ The arrival time of a signal is the time elapsed for a signal to arrive at a certain point. 5](#_Toc45957914)

[ Required time is the latest time at which a signal can arrive without making the clock cycle longer than desired 5](#_Toc45957915)

[ The slack associated with each connection is the difference between the required time and the arrival time. 5](#_Toc45957916)

[From the next section we will start to perform STA on reg2reg. 5](#_Toc45957917)

[2. Chapter 2 6](#_Toc45957918)

[**Different Inputs Required for a timing tool** 6](#_Toc45957919)

[2.1. Library files 6](#_Toc45957920)

[2.2. Gate Level Netlist 6](#_Toc45957921)

[2.3. Parasitic information 7](#_Toc45957922)

[2.4. Constrain file 7](#_Toc45957923)

[3. Chapter 3 8](#_Toc45957924)

[**Checking for Hold and Setup Violations** 8](#_Toc45957925)

[3.1. Run Tcl commands on OpenTimer 8](#_Toc45957926)

[3.2. Ran report\_timing to check for any vioaltions 9](#_Toc45957928)

[9](#_Toc45957929)

[4. Chapter 4 10](#_Toc45957930)

[**Ways to fix setup violation** 10](#_Toc45957931)

[4.1. Modifying one of the cells in the data path 10](#_Toc45957932)

[5. Chapter 5 12](#_Toc45957933)

[**Code** 12](#_Toc45957934)

[5.1. my\_netlist.v 12](#_Toc45957935)

[5.2. my\_netlist.timing 13](#_Toc45957936)

[5.3. my\_run.tcl 13](#_Toc45957937)

[5.4. blank.spef 13](#_Toc45957938)

[6. Chapter 6 14](#_Toc45957939)

[**Conclusions** 14](#_Toc45957940)

1. Chapter 1

## **Introduction**

OpenTimer is a new [static timing analysis (STA)](https://en.wikipedia.org/wiki/Static_timing_analysis) tool to help IC designers quickly verify the circuit timing. Key features are:

* Industry standard format (.lib, .v, .spef, .sdc) support
* Graph- and path-based timing analysis
* Parallel incremental timing for fast timing closure
  1. Purpose

Static timing analysis is a method of validating the timing performance of a design by checking all possible paths for timing violations under worst-case conditions. It considers the worst possible delay through each logic element, but not the logical operation of the circuit.

In a [synchronous digital system](https://en.wikipedia.org/wiki/Synchronous_circuit), data is supposed to move in [lockstep](https://en.wikipedia.org/wiki/Lockstep_(computing)), advancing one stage on each tick of the [clock signal](https://en.wikipedia.org/wiki/Clock_signal). This is enforced by synchronizing elements such as [flip-flops](https://en.wikipedia.org/wiki/Flip-flop_(electronics)) or [latches](https://en.wikipedia.org/wiki/Latch_(electronic)), which copy their input to their output when instructed to do so by the clock. Only two kinds of timing errors are possible in such a system:

* A **Max time violation**, when a signal arrives too late, and misses the time when it should advance. These are more commonly known as setup violations/checks which actually are a subset of max time violations involving a cycle shift on synchronous paths.
* A **Min time violation**, when an input signal changes too soon after the clock's active transition. These are more commonly known as hold violations/checks which actually are a subset of min time violations in synchronous path.
  1. Few important definitions
* The critical path is defined as the path between an input and an output with the maximum delay.
* The arrival time of a signal is the time elapsed for a signal to arrive at a certain point.
* Required time is the latest time at which a signal can arrive without making the clock cycle longer than desired
* The slack associated with each connection is the difference between the required time and the arrival time.

From the next section we will start to perform STA on reg2reg.

1. Chapter 2

## **Different Inputs Required for a timing tool**

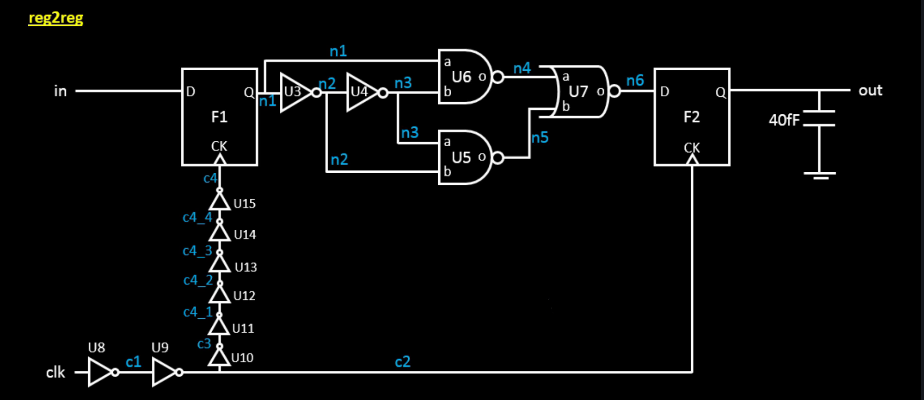
Following are the required inputs: -

* Libraries (timing libraries)
* Gate Level Netlist
* Design Constrains/ timing constrains
* Parasitic information
  1. Library files

These are generally in the form of .libs. These contains the timing information of standard cells/hard macros. Timing details are mentioned in NLDM (Non-Linear Delay Model) Model.   
We have two separate files for setup and hold analysis

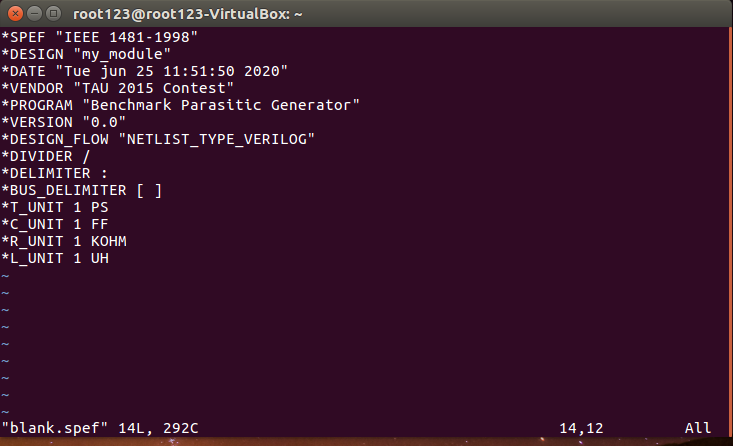
* my\_early.lib (Hold)
* my\_late.lib (Setup)
  1. Gate Level Netlist

Netlist can be of any type(viz. Verilog, vhdl, Spice etc.), it has the dependency only on the timing tool. In OpenTimer it requires a Verilog netlist.



* 1. Parasitic information

Parasitic information has the interconnect capacitance and resistance details. These are used for post-layout. It can be in several formats based upon accuracy, OpenTimer takes SPEF format.



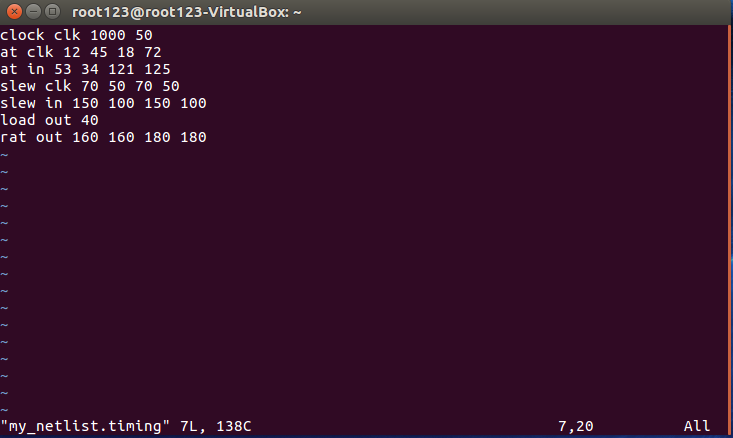
We are performing a pre-layout analysis hence the SPEF file is blank.

* 1. Constrain file

For meeting the designs timing requirement, we have to specify what the requirements are and these are set by the constrains. Design constrains can have following details: -

* Boundary conditions such as input and output delay
* Slew rate
* Path exceptions
* Clock frequency

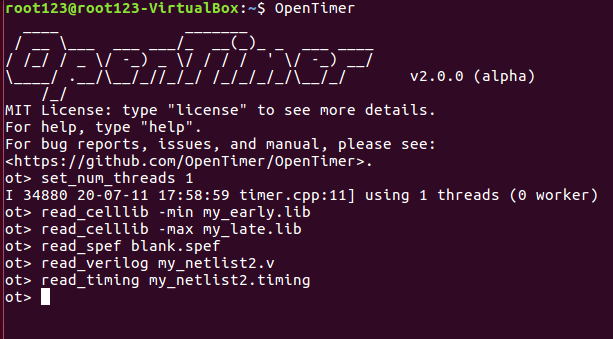
SDC is one of the formats of specifying the details of a constrains file. In OpenTimer we will be using .timing file to specify these constrains. Below snippet show the constrain file which we are using.



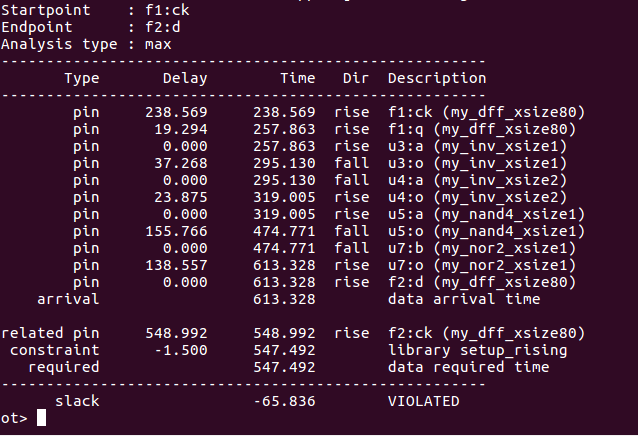
1. Chapter 3

## **Checking for Hold and Setup Violations**

* 1. Run Tcl commands on OpenTimer



* 1. Ran report\_timing to check for any vioaltions



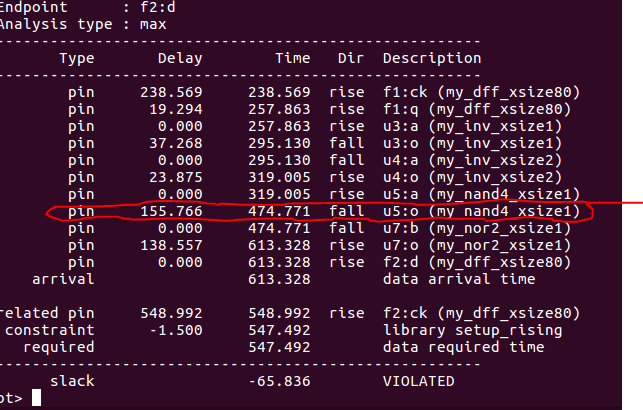
From the report it can be clearly seen that there is a setup time violation.  
The arrival time is 613.328 ps and the required arrival time is 547.492.  
Since AT < RAT we can see there is a setup time violation.

1. Chapter 4

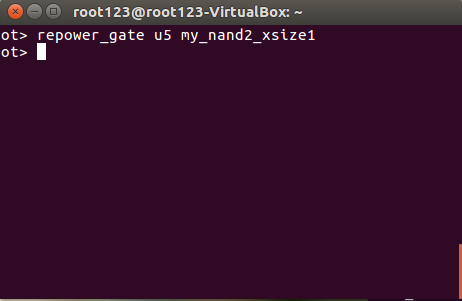
## **Ways to fix setup violation**

There are many ways to improve the setup violations in a circuit.

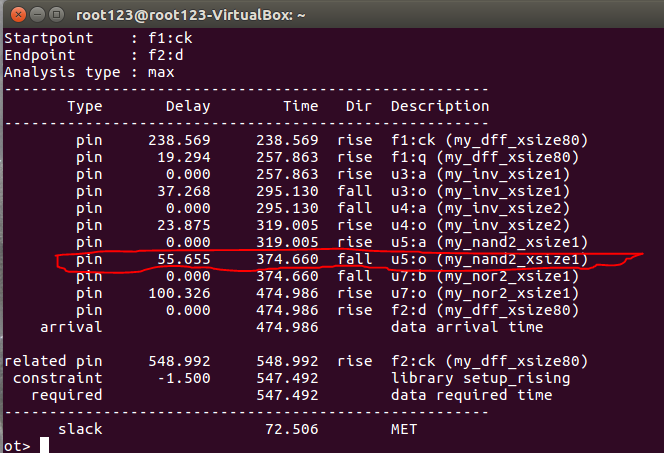
* 1. Modifying one of the cells in the data path



From the highlighted part it can be clearly seen that if we can replace my\_nand4\_xsize1 by a faster cell then we can definitely remove the setup violation.  
After the checking the my\_late.lib I found out that my\_nand4\_xsize1 can be replaced by my\_nand2\_xsize1 which will remove the setup violation.



By using above command, we can replace the my\_nand4\_xsize1 can be replaced by my\_nand2\_xsize1.



As we can see from the above snapshot the delay has reduced and the slack has met.

1. Chapter 5

## **Code**

In this section all the Verilog codes which were developed for analysis of the approximate circuits are given below. All the Verilog codes mentioned below are parametrised codes [6].

* 1. my\_netlist.v

module my\_module (  
in,clk,out  
);

// primary inputs  
input in;input clk;

// primary outputs  
output out;

// wires  
wire n1;wire n2;wire n3;wire n4;wire n5;wire n6;wire in;wire clk, wire c4\_4;  
wire out;wire c1;wire c2;wire c3;wire c4;wire c4\_1;wire c4\_2;wire c4\_3;;

// cells  
my\_dff\_xsize80 f1 (.d(in), .ck(c4), .q(n1));  
my\_inv\_xsize1 u3 (.a(n1), .o(n2));  
my\_inv\_xsize2 u4 (.a(n2), .o(n3));  
my\_nand2\_xsize1 u6 ( .a(n1), .b(n3), .o(n4));  
my\_nand4\_xsize1 u5 ( .a(n3), .b(n2), .o(n5));  
my\_nor2\_xsize1 u7 ( .a(n4), .b(n5), .o(n6) );  
my\_dff\_xsize80 f2 ( .d(n6), .ck(c2), .q(out) );

//clock path  
my\_inv\_xsize1 u8 (.a(clk), .o(c1));  
my\_inv\_xsize1 u9 (.a(c1), .o(c2));  
my\_inv\_xsize1 u10 (.a(c2), .o(c3));  
my\_inv\_xsize1 u11 (.a(c3), .o(c4\_1));  
my\_inv\_xsize1 u12 (.a(c4\_1), .o(c4\_2));  
my\_inv\_xsize1 u13 (.a(c4\_2), .o(c4\_3));  
my\_inv\_xsize1 u14 (.a(c4\_3), .o(c4\_4));  
my\_inv\_xsize1 u15 (.a(c4\_4), .o(c4));

endmodule

* 1. my\_netlist.timing

clock clk 500 50  
at clk 0 0 0 0  
at in 244 244 121 125  
slew clk 70 50 70 50  
slew in 150 100 150 100  
load out 40  
rat out 63 63 180 180

* 1. my\_run.tcl

set\_num\_threads 1  
set\_early\_celllib\_fpath my\_early.lib  
set\_late\_celllib\_fpath my\_late.lib  
set\_spef\_fpath blank.spef  
set\_verilog\_fpath my\_netlist.v  
set\_timing\_fpath my\_netlist.timing

* 1. blank.spef

\*SPEF "IEEE 1481-1998"  
\*DESIGN "my\_module"  
\*DATE "Tue jun 25 11:51:50 2020"  
\*VENDOR "TAU 2015 Contest"  
\*PROGRAM "Benchmark Parasitic Generator"  
\*VERSION "0.0"  
\*DESIGN\_FLOW "NETLIST\_TYPE\_VERILOG"  
\*DIVIDER /  
\*DELIMITER :  
\*BUS\_DELIMITER [ ]  
\*T\_UNIT 1 PS  
\*C\_UNIT 1 FF  
\*R\_UNIT 1 KOHM  
\*L\_UNIT 1 UH

1. Chapter 6

## **Conclusions**

Many of the common problems in chip designing are related to interface timing between different components of the design. These can arise because of many factors including incomplete simulation models, lack of test cases to properly verify interface timing, requirements for synchronization, incorrect interface specifications, and lack of designer understanding of a component supplied as a 'black box'. There are specialized CAD tools designed explicitly to analyze interface timing, just as there are specific CAD tools to verify that an implementation of an interface conforms to the functional specification (using techniques such as model checking.