Achieving Timing Closure

> Adapted from original guide by Mohamed Shalan based on his experience designing the Caravel chip using OpenLane and his digital systems design teaching materials.
>
>

Introduction

Timing closure is the process of satisfying timing constraints for clocked elements through design transformations. Clocked elements such as flip-flops have constraints that must be satisfied. The data fed to a flip-flop must be stable before the clock edge and after the clock edge. As shown in Fig. 16, the setup time constraint is the amount of time required for the input to a flip-flop to be stable before a clock edge. The hold time constraint is the minimum amount of time needed for the input to a flip-flop to be stable after a clock edge. Also, shows the delay time to change the flip-flop output relative to the arrival of the triggering clock edge. This delay is typically referred to as t_{CQ} (time from C to Q). Not satisfying the flip-flop setup and/or hold times constraints will make the flip-flop metastable. This means the data stored in the flip-flop is unknown, which means the state of the whole system becomes unknown.

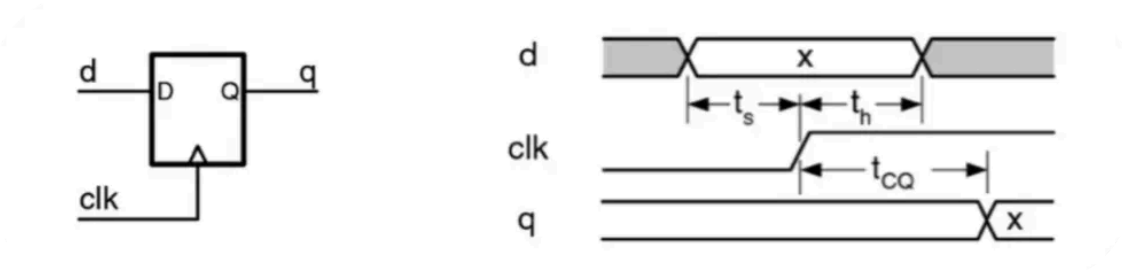


Fig. 16 Flip-flop parameters including Setup (t_s) and hold (t_h) constraints

Timing Paths

To achieve a timing closure, timing violations must be identified and fixed. Determining the timing violations is typically done using static timing analysis (STA) tools. The STA tool (e.g., OpenSTA) identifies the design timing paths and

then calculates the data earliest and latest actual and required arrival times (AAT and RAT) at every timing path endpoint. If the data arrives after (in case of setup checking) or before (hold checking) it is required, then we have a timing violation (negative slack).

There are four types of timing paths for a given design unit based on the type of the starting and ending points of the timing path. A timing path may start at the clock pin of a flip-flop or an input port. It ends at the data input pin (D) of a flip-flop or an output port. Hence there are four types of timing paths (Fig. 17):

- Input to output (in-to-out)
- Input to flip-flop (in-to-reg)
- Flip-flop to output (reg-to-out)
- Flip-flop to Flip-flop (reg-to-reg)
  - The first flip-flop is alternately referred to as the input or data release flip-flop, and the second is referred to as the output or data capture flip-flop.

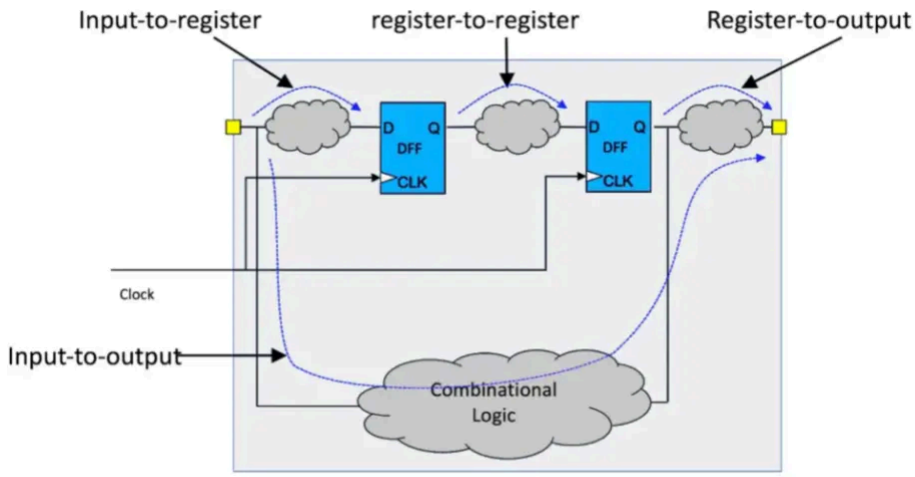


Fig. 17 Examples of Timing Paths

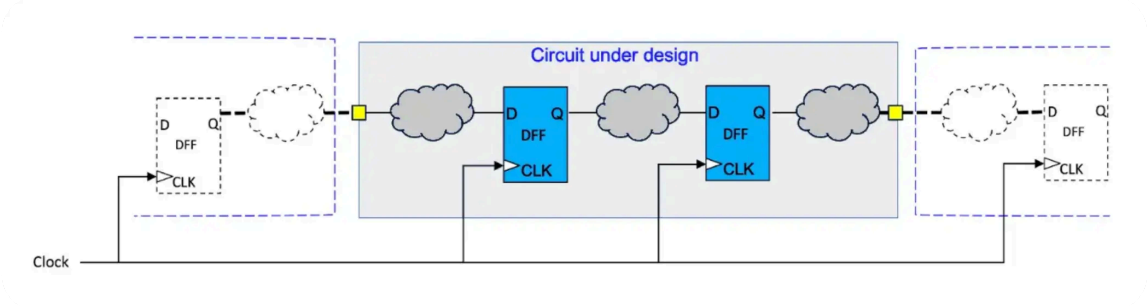


Fig. 18 Register to register timing path

The last three timing path types are actually flip-flop to flip-flop timing paths where the release, capture, or both are outside the design unit, as shown in Fig. 18.

Modeling CMOS Cells

Complementary metal-oxide-semiconductor (CMOS) technology is the mainstream technology for digital circuits implementation. A CMOS logic gate (also called a cell) is made of complementary and symmetrical pairs of p-type and n-type MOSFETs (Metal-Oxide-Semiconductor Field-Effect Transistors) called the pull-up and pull-down networks (PUN and PDN). The PUN is made of p-type MOSFETS (pMOS transistors), and the PDN is made of n-type MOSFETS (nMOS transistors). The PUN acts as a programmable switch that connects the output of the CMOS gate to VDD when the boolean function implemented by the gate produces logic 1. The PDN connects the output to GND when the boolean function produces logic 0. Fig. 19 shows the PUN and the PDN and the RC model of a 2-input CMOS nand2 cell. The RC model is an approximate model used for delay calculations. In this model, every input pin has a capacitance equivalent to the sum of the gate capacitances of the MOSFETs connected to the pin. The PUN is modeled as a switch and a resistance, the same for the PDN. The diffusion capacitances of the MOSFETs connected to the cell output are modeled as a single internal (also called intrinsic) capacitance (C_{int}) at the cell output.

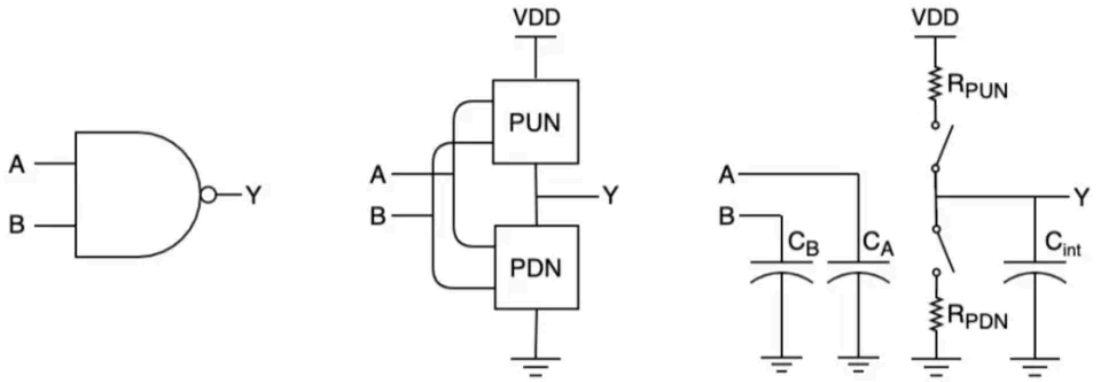


Fig. 19 NAND2 CMOS Cell Electrical RC Model

When the CMOS cell output switches state, the switching speed is governed by how fast the capacitance on the output net can be charged or discharged. The capacitance on the output net is the sum of the parasitic capacitance, the interconnects (wires) capacitances, and the pin capacitances of the cells driven by

this cell, as shown in Figure 5. The cell delay can be approximated as the product of the PUN/PDN resistance and the sum of the intrinsic and load capacitances. As R_{PUN} can be different from R_{PDN} the rising t_{pdr} and falling t_{pdf} delays are different.

$$t_{pdr} = R_{PUN}(C_{in} + C_{load})$$
$$t_{pdf} = R_{PDN}(C_{in} + C_{load})$$

Note

A wire is assumed to be equivalent to a single capacitance. This is only valid for short wires with no vias. In reality, a wire has a lumped resistance and a lumped capacitance. Both rely on the wire dimensions; the longer the wire, the higher the resistance and the capacitance. Table 1 lists the resistance and the capacitance per micro for every routing layer in the sky130 PDK.

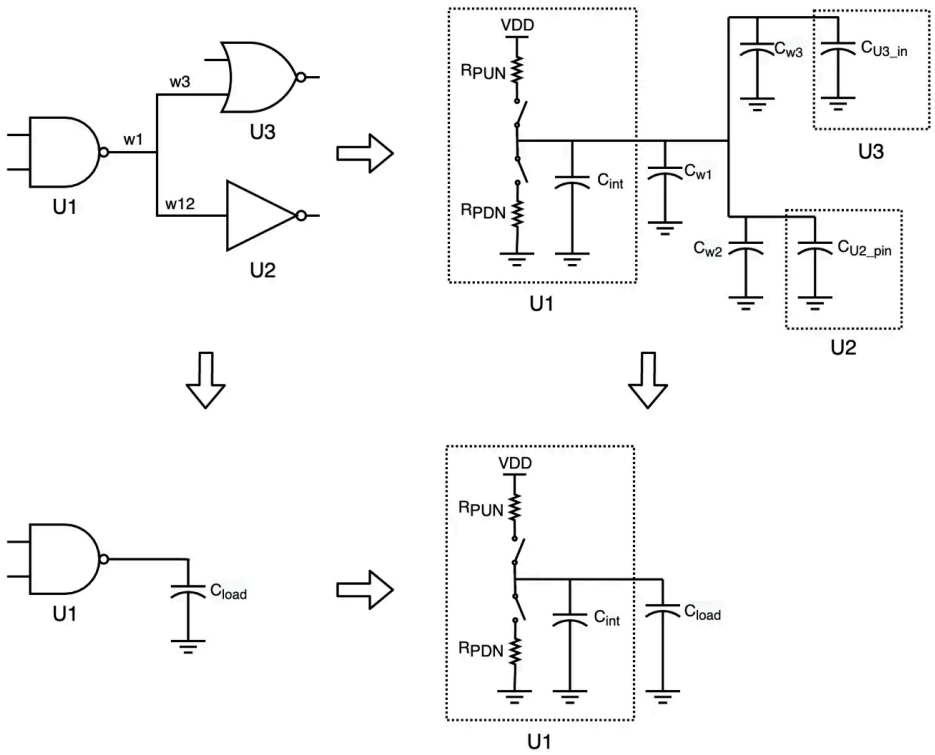


Fig. 20 RC Model to calculate the propagation delay of U1

Table 1 sky130 Routing Layer Resistances and Capacitances

Layer	R/μ (Ω) [1]	C/μ (fF) [2]
li1	71.76	0.15
mcon	9.25	N/A
met1	0.893	0.145
via	4.5	N/A
met2	0.893	0.133
via2	3.369	N/A
met3	0.157	0.146
via3	0.376	N/A
met4	0.157	0.13
via4	0.005	N/A
met5	0.018	0.15

Timing Constraints

As a designer, you need to specify the design constraints (intents) to help different EDA tools carry out their jobs (optimization and analysis) during the implementation process. The constraints are expressed using statements that are based on the TCL language. These commands have an industry-standard which is called SDC (Synopsys Design Constraints) format. The SDC format also covers statements that specify timing exceptions, such as false paths and multi-cycle paths. Please refer to OpenSTA documentation [1] for more information. Here are some examples of basic SDC constraints:

- The clock definition (`create_clock`)
- Delays for input and output ports relative to the clocks (`set_input_delay` and `set_output_delay`)
- The load on output ports (`set_load`)
- The driving cells for input ports (`set_drive`)

A false path is a timing path that exists in the circuit but will never be exercised. For example, in Fig. 21, the path from DIN to DOUT will never be exercised because the selection of MUX2 complements the selection of MUX1; hence, MUX1 and MUX2 can't select A simultaneously. To set the false path use: set_false_path -through w1 -through w2

How about multi-cycle paths? Generally, the combinational path between a launch flip-flop and a capture flip-flop takes only one clock cycle to propagate the data. However, in some cases, it could take more than a single clock cycle to propagate the data through the combinational data path. This is known as a multi-cycle path. In a normal path (non-multi-cycle) the setup check is performed at the destination FF one cycle after launching the data and the hold check is performed at the destination at the same edge that released the data as shown in Fig. 22.

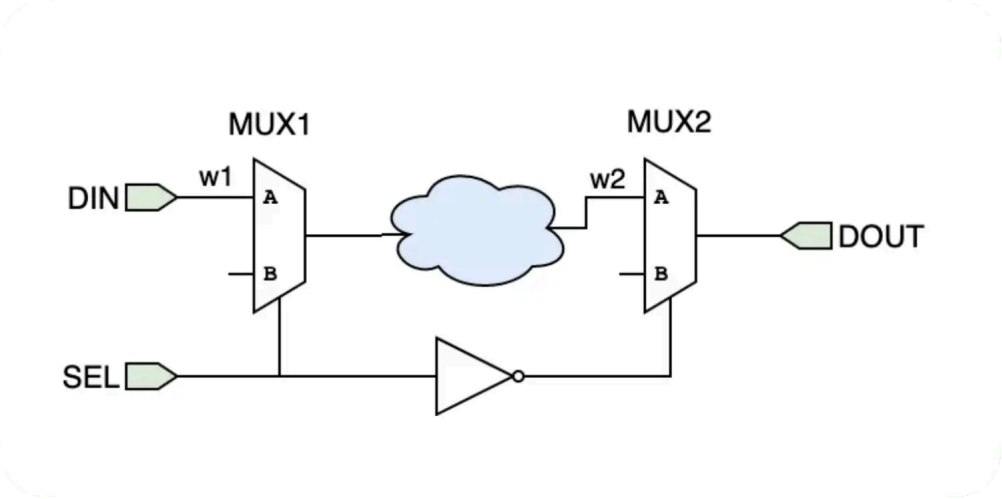


Fig. 21 False Path Example

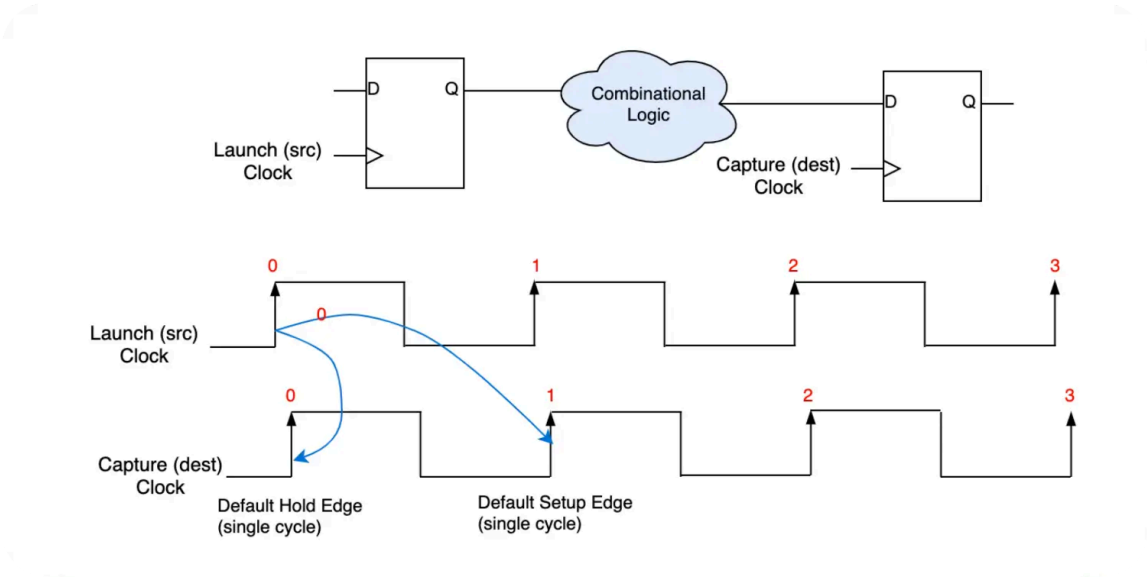


Fig. 22 Constraining a Multi-Cycle Path

For a multi-cycle path that takes N cycles, the setup check is performed after N cycles at the destination and hold check is performed after N-1 cycles. Assuming the combinational data path takes two clock cycles to propagate data to the capture flip-flop, then there should be a multi-cycle path defined for the setup and hold checks between launch and capture flip-flops as follows:

```
// multi-cycle from setup check for launching to capturing clock

set_multicycle_path -setup 2 -from [get_pins launch_ff/Q] -to [get_pins
capture_ff/D]

// multi-cycle from hold check for launching to capturing clock

set_multicycle_path -hold 1 -from [get_pins launch_ff/Q] -to [get_pins
capture_ff/D]
```

To understand how to check whether the setup and hold constraints are met or not, let's consider the circuit and timing diagram in Figure 8. In this circuit the clock is skewed by the value t_{skew}. The clock skew is defined as the difference in clock arrival times to flip-flops. Here the clock edge arrives at the release flip-flop (U11) before the capture flip-flop (U14). In this scenario the skew is referred to as a positive skew. To observe a negative skew, the clock edge has to reach the release flip-flop after it reaches the capture flip-flop. In order for the capture flip-flop (U14) to function correctly the data released by U11 should arrive within the green window. If the data arrives early, hold constraint is violated and if it arrives late setup constraint is violated. The STA tool calculates the AAT of data to U14 for setup constraint check as follows:

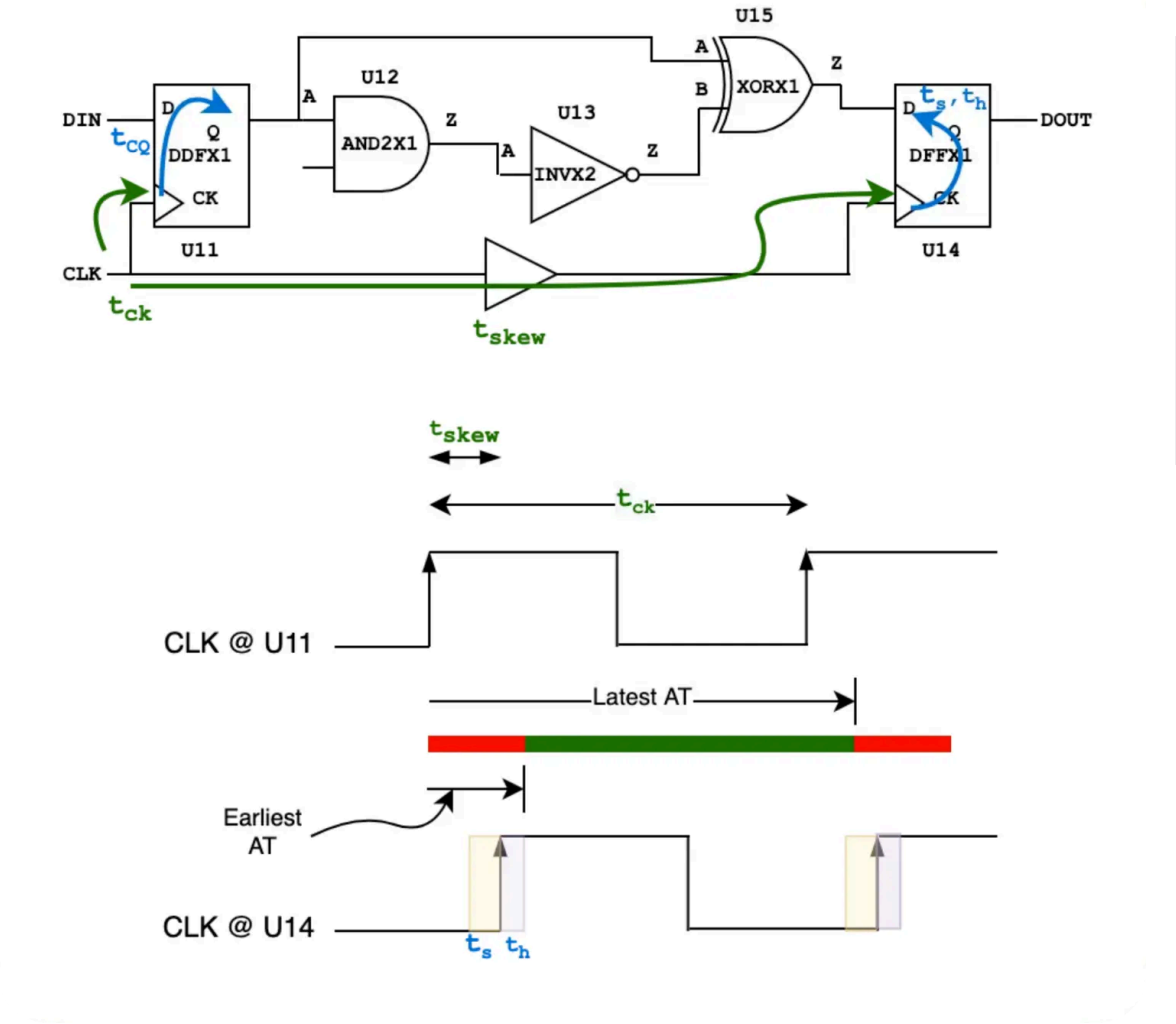


Fig. 23 RAT and AAT Calculation for Setup Constraint Check

To understand how to check whether the setup and hold constraints are met or not, let's consider the circuit and timing diagram in Fig. 23. In this circuit the clock is skewed by the value t_{skew}. The clock skew is defined as the difference in clock arrival times to flip-flops. Here the clock edge arrives at the release FF (U11) before the capture FF (U14). In this scenario the skew is referred to as a positive skew. To observe a negative skew, the clock edge has to reach the release flip-flop after it reaches the capture flip-flop. In order for the capture flip-flop (U14) to function correctly the data released by U14 should arrive within the green window. If the data arrives early, hold constraint is violated and if it arrives late setup constraint is violated. The STA tool calculates the AAT of data to U14 for setup constraint check as follows:

(1) $AAT_{latest} = t_{ck} + t_{CQ} + t_{and} + t_{inv} + t_{xor}$

The required time is calculated using the following:

(2) $RAT_s = t_{ck} + t_{skew} - t_s$

To satisfy the setup constraint, the latest AAT should be less than RAT_{s}. In another words:

(3) $SLACK_s = RAT_s - ATT_{latest} > 0$

For hold constraint check, we use the earliest arrival time

(4) $AAT_{earliest} = t_{ck} + t_{CQ} + t_{xor}$

and the required time can be calculated

(5) $RAT_h = t_{skew} + t_h$

To satisfy the hold constraint, the earliest AAT should be larger than the RAT_h, which means:

(6) $SLACK_s = ATT_{earliest} - RAT_h > 0$

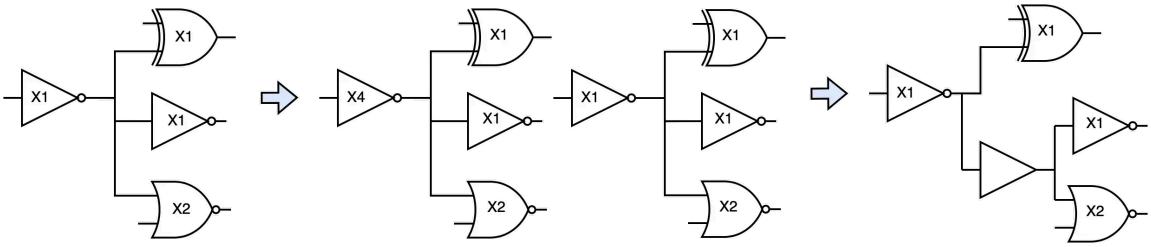
Timing Optimizations

When a violation is identified for a timing path, design transformations such as buffering, sizing, cloning, clock skewing, etc., are performed to ensure that the slack is zero or positive. During physical implementation (also called Place and Route or PnR), these transformations are done after placement and/or global routing. They result in modification of the netlist. Illustrations of some commonly used optimizations are given in Fig. 24.

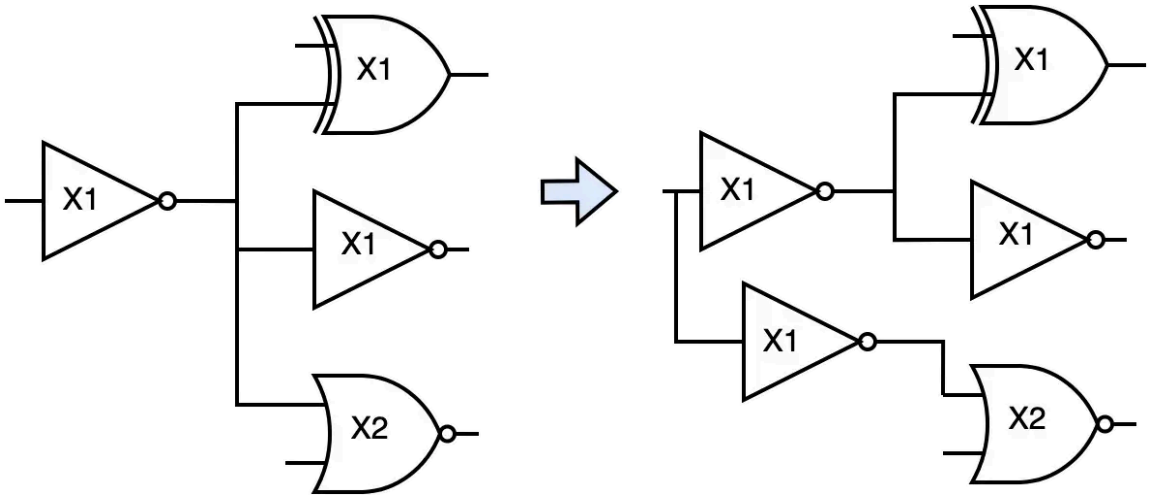
Typically, a standard cell library (SCL) contains multiple sizes of the same cell. For example, the Sky130 HD SCL contains several versions of the nand2 (2-input NAND) cell for the sizes 1, 2, 4, and 8. The cell name reflects its size, for example size 4 nand2 cell is named `sky130_fd_sc_hd__nand2_4` where the `_4` postfix reflects its size. The cell size indicates its driving strength which determines how much current it can provide to its load. This means the higher the driving strength, the lower the cell delay for a given capacitive load. Please note that increasing the size of a cell increases its loading effect on its driver. This means sizing a cell up to reduce its delay is associated with the increase of its driving cell delay.

The buffer is a typical SCL that plays an essential role in circuit timing. A buffer is nothing but two inverters connected back to back, which does not perform any logic function. However, as a CMOS circuit, it produces the input signal. Hence, it generates an excellent sharp digital signal from an input signal with a long transition time. Also, a buffer can reduce the timing effects of long wires by

dividing it into smaller segments. Moreover, it can be used to insert delays to the data path to fix hold violations.



(a) Sizing (b) Buffering



(c) Cloning

Fig. 24 Examples of Timing Optimizations

Tips for Achieving Timing Closure

OpenLane supports automatic timing closure; the `Classic` flow applies design optimizations that favor hold violations fixing after CTS and after global routing. These optimizations are controlled using some flow configurations such as those given by Table 2.

Table 2 OpenLane Timing-related Configuration Variables

Parameter	Use
`RUN_POST_CTS_RESIZER_TIMING`	Specifies whether design timing optimizations should be performed immediately after placement and CTS or not.
`PL_RESIZER_SETUP_SLACK_MARGIN`	Used to guide timing optimization after placement. It instructs the optimizer not to stop at zero slack and try to achieve a positive timing slack.
`PL_RESIZER_HOLD_SLACK_MARGIN`	Used to guide timing optimization after placement. It instructs the optimizer not to stop at zero slack and try to achieve a positive timing slack.
`PL_RESIZER_HOLD_MAX_BUFFER_PCT`	Maximum % of hold buffers to insert to fix hold vios. (PL/CTS)
`PL_RESIZER_SETUP_MAX_BUFFER_PCT`	Maximum % of hold buffers to insert to fix setup vios. (PL/CTS)
`PL_RESIZER_ALLOW_SETUP_VIOS`	Allow setup violations while fixing hold violations after placement and CTS.
`RUN_POST_GRT_RESIZER_TIMING`	Enable/disable timing optimizations after global routing. This is an experimental feature in OpenROAD and crashes may occur.
`GRT_RESIZER_SETUP_SLACK_MARGIN`	Used to guide timing optimization after global routing. It instructs the optimizer not to stop at zero slack and try to achieve a positive slack (the specified margin.)
`GRT_RESIZER_HOLD_SLACK_MARGIN`	Used to guide timing optimization after global routing. It instructs the optimizer not to stop at zero slack and try to achieve a positive slack (the specified margin.)
`GRT_RESIZER_HOLD_MAX_BUFFER_PCT`	Maximum % of hold buffers to insert to fix hold vios after global routing.

Parameter	Use
`GRT_RESIZER_SETUP_MAX_BUFFER_PCT`	Maximum % of hold buffers to insert to fix setup vios after global routing.
`GRT_RESIZER_ALLOW_SETUP_VIOS`	Allow setup violations while fixing hold violations after global routing.

> **Tip**
>
> Click on the variable names in the table above to get the default values.

TheClassic flow performs STA at several points throughout the flow by using the following steps:

- `OpenROAD.STAPrePNR`
- `OpenROAD.STAMidPNR`
- `OpenROAD.STAPostPNR`

As the IDs imply, The first of the steps runs before PnR, i.e., using the synthesized netlist and OpenROAD's bundled OpenSTA. The second runs multiple times at different points in the flow using the in-progress floorplan. The final runs using the post-PnR netlist with parasitics extracted from the final floorplan, giving the most accurate timing analysis of the final manufacturable design.

Each of these steps generates a number of reports, which can be found under `runs/<RUN_NAME>/<STEP_ID>`. The reports contain useful timing information. For example `wns.max.rpt` contains the setup worst negative slack:

```
=======================================================================
Worst Negative Slack (Setup)
=======================================================================
nom_tt_025C_1v80: 0.0
```

The files `max.rpt` and `min.rpt` contain the timing reports for setup and hold constraint analysis (respectively) sorted by the slack (worst first). Here is one such example showing a timing path from the `max.rpt` report:

```
Startpoint: x[9] (input port clocked by clk)
Endpoint: _618_ (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: min
Delay Time Description
---------------------------------------------------------
0.00 0.00 clock clk (rise edge)
0.00 0.00 clock network delay (ideal)
0.20 0.20 v input external delay
0.01 0.21 v x[9] (in)
0.08 0.28 ^ _537_/Y (sky130_fd_sc_hd__nand2_1)
0.06 0.34 v _541_/Y (sky130_fd_sc_hd__o21ai_1)
0.00 0.34 v _618_/D (sky130_fd_sc_hd__dfrtp_1)
0.34 data arrival time
0.00 0.00 clock clk (rise edge)
0.00 0.00 clock network delay (ideal)
0.00 0.00 clock reconvergence pessimism
0.00 ^ _618_/CLK (sky130_fd_sc_hd__dfrtp_1)
-0.05 -0.05 library hold time
-0.05 data required time
---------------------------------------------------------
-0.05 data required time
-0.34 data arrival time
---------------------------------------------------------
0.39 slack (MET)
```

The report snippet in Figure 12 contains a timing report for one of the min timing paths. The report for any timing path is comprised of 4 sections:

- The header: contains basic information about the timing path, such as the start and endpoints
- Data arrival: Shows cells and interconnect delay information from the start point to the endpoint
- Data required: Gives information about when the data is needed at the endpoint.
- The Slack: The difference between data required and data arrival times.

However hard OpenLane tries to achieve automatic timing closure; it may not work as expected in rare cases. Here are some tips to aid you in achieving timing

closure:

1. OpenLane sets some timing constraints through configuration variables (clock definition, input and output delays, load on output ports, driving cell for input ports, and maximum allowed transition). However, it is your responsibility to complete the constraints by providing an SDC file custom constraints. The custom SDC file can define timing exceptions and specify different constraints for input and output ports. Use the flow configuration variable `PNR_SDC_FILE` to point to an SDC file for pre/mid PnR steps, and `SIGNOFF_SDC_FILE` to point to an SDC file for `OpenROAD.STAPostPNR`.
    - Having different PnR and signoff SDC files is useful for when you want to over-constrain the PnR steps to possibly achieve better results then perform timing analysis with realistic constraints.
2. OpenLane assumes a single clock domain. Suppose you have multiple clock domains in the same design. In that case, it is your responsibility to adjust the constraints and other OpenLane components to handle them correctly.
3. STA performed immediately after the synthesis stage will not report any hold violations as the clock is considered an ideal net; hence there will be no clock skews to cause hold violations. However, you may have setup violations. You can see hold violations at this stage only if your HDL model has an embedded clock tree.
4. If your design does not have a hard requirement for the clock frequency (for example, USB 1.1 IP core requires a 48MHz system clock to comply with USB specifications), focus on hold violations as your chip can be completely non-functional because of hold violations. In this case, adjust the clock period to eliminate the setup violations. You may use the configuration variables: `PL_RESIZER_ALLOW_SETUP_VIOS` and `GRT_RESIZER_ALLOW_SETUP_VIOS` to instruct the timing optimizer to prioritize hold violations fixing over setup violations fixing.
5. Hold violations are, mainly, caused by clock skew (difference in clock arrival times to flip-flops). A good clock tree will ease things. It is impossible to synthesize a perfectly balanced clock tree to achieve zero clock skew. It is always good to check the clock skew report and address issues causing high skew (for example, wrong placement of a macro). It was found that limiting the clock buffer sizes during CTS through the configuration variable `CTS_CLK_BUFFERS` helps balance the clock tree.
6. Timing optimizations increase the design cell area (hence, the design core utilization) due to cell sizing and buffer insertions. To not explode the design size, timing optimizations stop after inserting buffers after a certain

percentage of buffer insertions. This percentage is controlled by the flow configuration variables mentioned in the table above– if you increase the percentage of buffer insertions, it is recommended to re-run the flow with a lower core utilization (use `FP_CORE_UTIL`) to avoid routing congestions.

7. Spend good time identifying false paths. This is highly dependent on your understanding of the RTL code. EDA tools will fix violations on False paths if they are not instructed not to do so. This will lead to unnecessary optimizations that may explode the design size and/or prevent optimizing True paths.
8. Setup and hold violations may be reported on false paths during STA. Review your violating paths carefully. Update your constraints to include False paths if you find any.
9. It is recommended that you simulate the final gate-level netlist with SDF (Standard Delay Format) annotations. This simulation is the closest to what you see from the fabricated chip. Moreover, timing simulation can help identify false paths. Section 3 gives an overview of how to perform this.
10. Violating maximum capacitance and maximum transition constraints are OK if you don't have setup/hold vios. It is always good to check them as they are indicators of design issues. For example, a high capacitance on a net means high fanout and/or long interconnect. Also, high transition time means a high capacitive load and leads to a higher short circuit power and delays.

Fig. 25 outlines a flowchart for addressing timing closure issues.

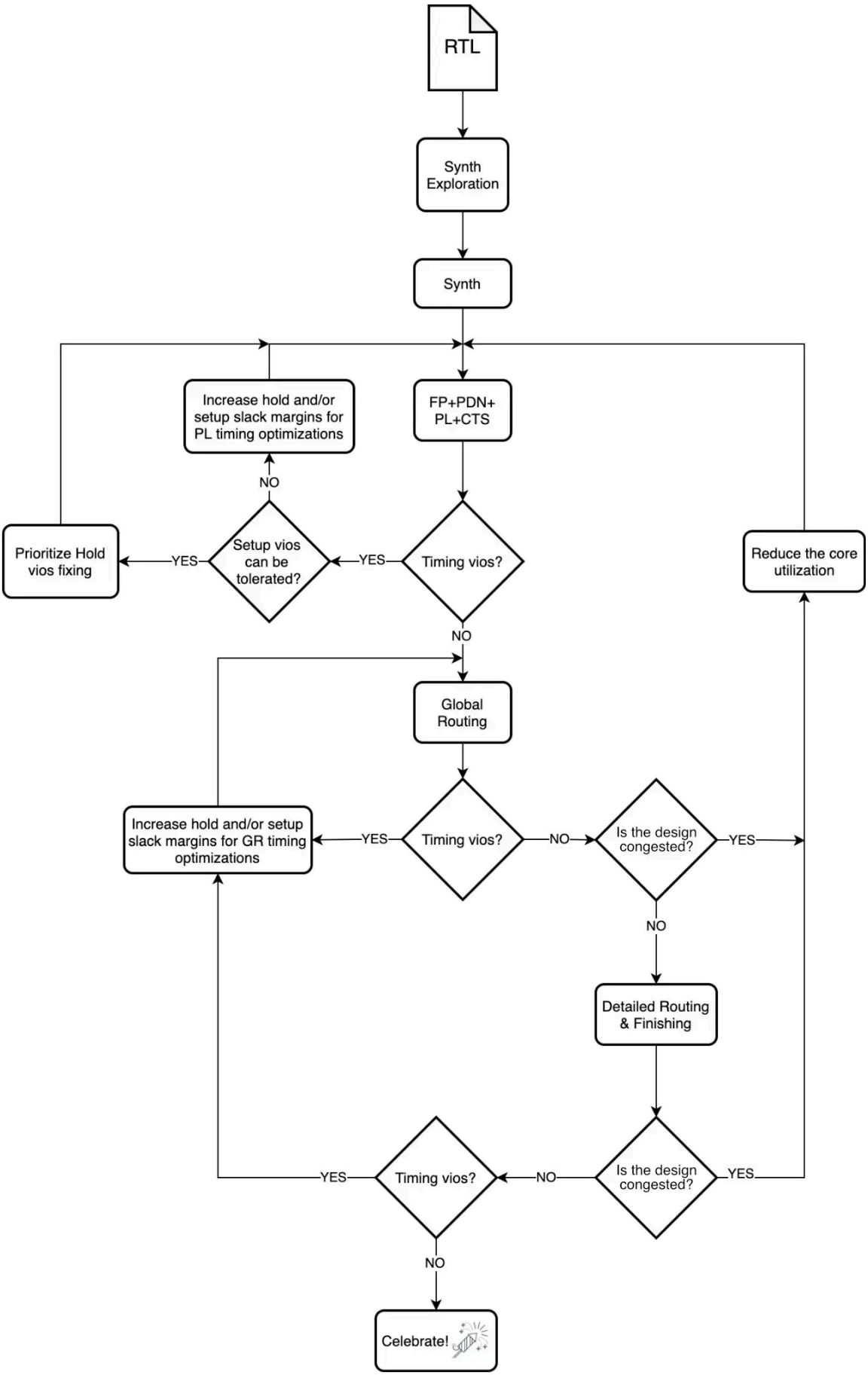

RTL
Synth Exploration
Synth
FP+PDN+ PL+CTS
Timing vios?
YES
Setup vios can be tolerated?
YES
Prioritize Hold vios fixing
NO
Increase hold and/or setup slack margins for PL timing optimizations
NO
Reduce the core utilization
Global Routing
Timing vios?
YES
Increase hold and/or setup slack margins for GR timing optimizations
NO
Is the design congested?
YES
NO
Detailed Routing & Finishing
Is the design congested?
YES
Timing vios?
YES
NO
NO
Celebrate!

Fig. 25 Timing Closure Flowchart

SDF Annotated GL Simulation using CVC

Standard Delay Format (SDF) is an IEEE standard for representing and interpreting timing data for use at any stage of an electronic design process. Typically SDF files are generated by static timing analysis tools and used with simulators for gate-level simulation with timing information. The SDF file contains the timing information of all the cells in the design. It is used to provide timing information for simulating the gate-level netlist. Simulation using timing information is a time-consuming process, and it cannot replace STA. Timing simulation can be helpful to uncover false paths and validate them.

OpenLane has support for SDF generation (found under results/final/sdf/ folder). To use the generated SDF file in simulation, you need to update your testbench to add a line similar to the following line (inside any initial block) for each macro in your design.

```
$sdf_annotate("design.sdf", design_instance_name_in_the_tb);
```

Unfortunately, popular open-source Verilog simulators don't have support for SDF (Verilator) or have limited support that is not entirely useful (iverilog). However, there is a less popular free-of-charge Verilog simulator called CVC by Tachyon DA [2] that has decent support for timing simulation using SDF files. CVC is a proprietary "shared source"[4] simulator, but its license allows using it freely for non-commercial designs.

OpenLane does not include CVC, so you will have to compile it separately. To run timing simulation using CVC, invoke the following:

```
$ cvc +interp +dump2fst <YOUR_NETLIST.v> +incdir+<PATH_TO_SCL_MODEL_FILES>
```

While running the simulation CVC may display messages if timing violations are caught by cell models checkers. An example of such messages is given below:

```
WARN** now 2050218 ps [566] timing violation in
x.y.sky130_fd_sc_hd__dfxtp_1_0_ (diff. 11 ps)

setup(of setuphold)((posedge D):2050207 ps, (posedge CLK):2050218 ps, 76 ps);
```

Reporting violations during simulation that are not identified by static timing analysis, means you did not constraint the design probably; for example, you specified a false path which is not. After identifying and analyzing the violations, you need to re-run your ASIC flow to address the reported violations.

Warning

The Verilog model files for sky130 libraries are known to contain some issues. The corrected files can be found here.

References

[1] James Cherry. OpenSTA Documentation. 2024. URL: https://github.com/The-OpenROAD-Project/OpenSTA/blob/fbfc705282d102cccbdf3472e86fc9da35268ab5/doc/OpenSTA.pdf.

[2] CVC Verilog Simulator. URL: https://github.com/cambridgehackers/open-src-cvc/.

[3] Rakesh Chadha and J. Bhasker. *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Springer US, Boston, MA, 2009. ISBN 978-0-387-93819-6 978-0-387-93820-2. URL: https://link.springer.com/10.1007/978-0-387-93820-2, doi:10.1007/978-0-387-93820-2.

[4] Sridhar Gangadharan and Sanjay Churiwala. *Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints (SDC)*. Springer New York, New York, NY, 2013. ISBN 978-1-4614-3268-5 978-1-4614-3269-2. URL: http://link.springer.com/10.1007/978-1-4614-3269-2, doi:10.1007/978-1-4614-3269-2.

[5] Andrew B. Kahng, Jens Lienig, Igor L. Markov, and Jin Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Netherlands, Dordrecht, 2011. ISBN 978-90-481-9590-9 978-90-481-9591-6. URL: https://link.springer.com/10.1007/978-90-481-9591-6, doi:10.1007/978-90-481-9591-6.

Acronyms on this Page

AAT
: Actual Arrival Time

CLA
: Carry Lookahead Adder

FF
: Flip-flop

GR
: Global Routing

Iverilog
: Icarus Verilog

MUX
: Multiplexer

PL
: Placement

PDN
: Pull-Down Network. Be careful: in other parts of the documentation, PDN refers to power distribution networks.

PUN
: Pull-Up Network

RAT
: Required Arrival Time

RCA
: Ripple Carry Adder

reg
: Register

SEL
: Selection

Vios
: Violations

WLM

Wire Load Model

WNS

Worst negative Slack

[1] The values for via layers (including `mcon`) are the resistance per via, not per micron.

[2] fF = 10^{-15} Farad

[4] The CVC website claims that the product is open source but it uses a **custom** license which is not OSI approved and does not meet either the OSI's "Open Source Definition" nor the Debian Free Software Guidelines, derived from the Artistic License but adding new terms and clauses.