STA

by

Asif Al Azad
Physical Design Engineer

We strive to be the most trusted partner for the world's hardest engineering problems.

Contents

What is Static Timing Analysis? 3
Why Static Timing Analysis? 3
Design Flow 4
STA at Different Design Phases 5
Limitations of Static Timing Analysis 5
Basic flow for STA 6
Inputs Files 6
Output files 7
Checking design inputs: 7
Check Timing Constraints 8
Check Parasitic Coverage 8
MMMC Design Import Method in Tempus Stylus 9
MCMM File 9
Timing Analysis and Reporting 13
Single Analysis Mode: 13
Best-Case Worst-Case Analysis Mode 14
On-Chip Variation Analysis Mode 15
CPPR/CRPR 15
Reporting Timing 16
What is GBA? 16
What is PBA? 17
Advance Analysis 18
Ultra-Low Vdd Variation Modeling in Tempus 18
What Are On-Chip Variation and Their Sources? 19
What Is Advanced OCV (AOCV)? 20
AOCV: Calculation 21
Basic Clock Constraints 21
Creating clocks 21
Creating clock uncertainties 26
Creating clock transition times/slews 26
Creating clock latencies 26
Flat Level STA 27

We strive to be the most trusted partner for the world's hardest engineering problems.

What is Static Timing Analysis?

Static Timing Analysis (also referred as STA) *is one of the many techniques available to verify the timing of a digital design. The STA is static since the analysis of the design is carried out statically and does not depend upon the data values being applied at the input pins. Given a design along with a set of input clock definitions and the definition of the external environment of the design, the purpose of static timing analysis is to validate if the design can operate at the rated speed. That is, the design can operate safely at the specified frequency of the clocks without any timing violations.*

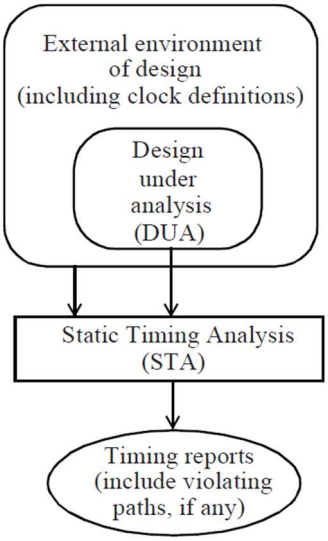


Figure 1-1 *Static timing analysis.*

External Environment: SDC
DUA: Netlist

Why Static Timing Analysis?

Static timing analysis provides a faster and simpler way of checking and analyzing all the timing paths in a design for any timing violations. Given the complexity of present-day ASICs, which may contain 10 to 100 million gates, the static timing analysis has become a necessity to exhaustively verify the timing of a design.

Design Flow

Once a design at the RTL level has been synthesized to the gate level, the STA is used to verify the timing of the design. STA can also be run prior to performing logic optimization - the goal is to identify the worst or critical timing paths.

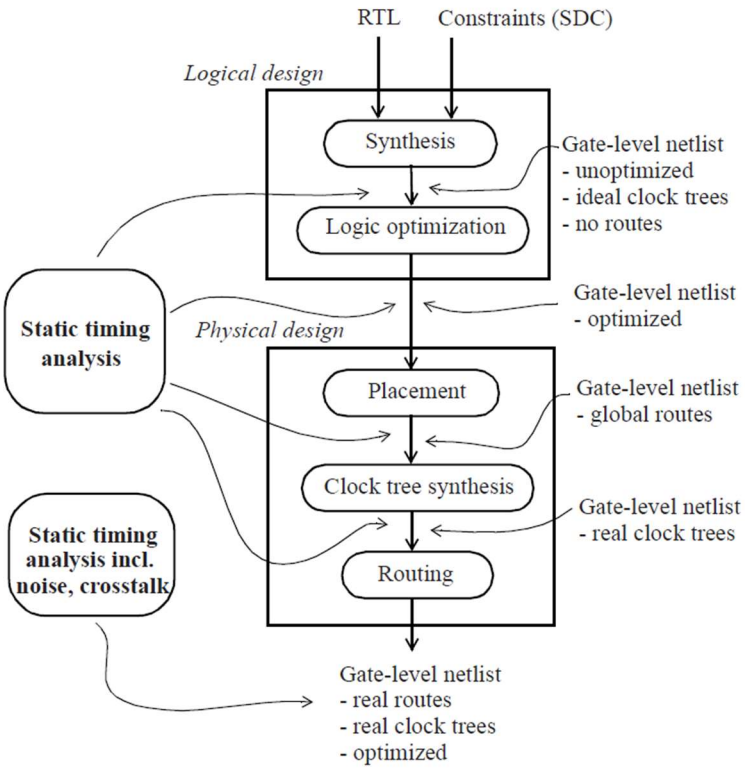


Figure 1-2 *CMOS digital design flow.*

At the start of the physical design, clock trees are considered as ideal, that is, they have zero delay. Once the physical design starts and after clock trees are built, STA can be performed to check the timing again. At the logical design phase, ideal interconnect may be assumed since there is no physical information related to the placement; there may be more interest in viewing the logic that contributes to the worst paths. Another technique used at this stage is to estimate the length of the interconnect using a wireload model. The wireload model provides estimated RC based on the fanouts of a cell.

The static timing analysis can be performed on a gate-level netlist depending on:

i. How interconnect is modeled – i4deal interconnect, wireload model, global routes with approximate RCs, or real routes with accurate RCs.

ii. How clocks are modeled - whether clocks are ideal (zero delay) or propagated (real delays).

iii. Whether the coupling between signals is included – whether any crosstalk noise is analyzed.

STA at Different Design Phases

At the logical level (gate-level, no physical design yet), STA can be carried out using:

i. Ideal interconnect or interconnect based on wireload model.
ii. Ideal clocks with estimates for latencies and jitter.

During the physical design phase, in addition to the above modes, STA can be performed using:

i. Interconnect - which can range from global routing estimates, real routes with approximate extraction, or real routes with signoff accuracy extraction.
ii. Clock trees - real clock trees.
iii. With and without including the effect of crosstalk.

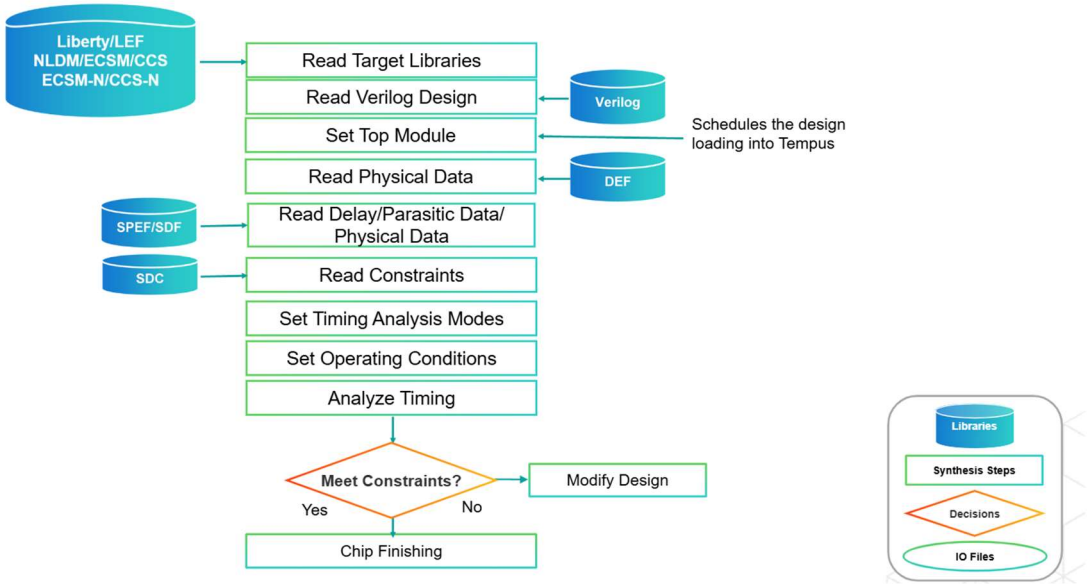
Limitations of Static Timing Analysis

While the timing and noise analysis do an excellent job of analyzing a design for timing issues under all possible situations, the state-of-the-art still does not allow STA to replace simulation completely. Some of the limitations of STA are:

i. Reset sequence:
ii. X-handling:
iii. PLL settings:
iv. Asynchronous clock domain crossings
v. IO interface timing
vi. Interfaces between analog and digital blocks
vii. False paths
viii. FIFO pointers out of synchronization
ix. Clock synchronization logic
x. Functional behavior across clock cycles:

Despite issues such as these, STA is widely used to verify timing of the design and the simulation (with timing or with unit-delay) is used as a backup to check corner cases and more simply to verify the normal functional modes of the design.

Basic flow for STA



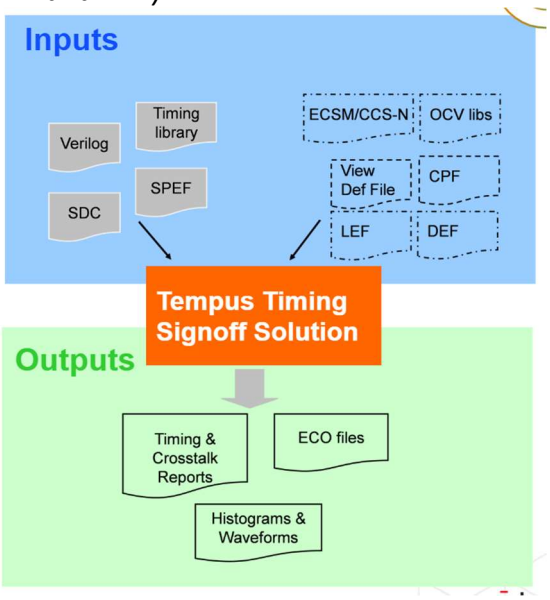
Inputs Files

These files are mandatory files to run STA.

- *SDC*
- *Netlist*
- *SPEF*
- *Timing Libraries*

There are some optional inputs files. These are,

- *Noise Libraries (ECSM-Noise. CCS-Noise or CDB for SI)*
- *CPF (Power intent file)*
- *View Definition file for MMMC analysis*
- *AOCV or POCVM libraries or LVF libraries for on-chip variation effects*
- *Physical Data (LEF and DEF)*



Output files

- *Timing and Crosstalk Reports*
- *Histograms and Waveforms*
- *ECO files*

Checking design inputs:

To check the input files for Tempus to do STA we should do some basic checks,

Check Design and Library Issues

```
check_design
[-out_file fileName ]
[-type {power_intent timing |
opt|cts| signoff|no_check]
```

This command checks for missing or inconsistent library and design data at any stage of the design and writes the result to a text and HTML report.
Tempus software checks the following data:

- *I/Os*
- *Netlists*
- *Timing and physical libraries*
- *Power and ground pins*
- *Tie-High and tie -low pins*
- *SPEF (Parasitic file)*

Reports look like this,

```
Design: dtmf_recvr_core
------ Design Summary:
Total Standard Cell Number   (cells) : 9357
Total Block Cell Number      (cells) : 0
Total I/O Pad Cell Number    (cells) : 0
------ Design Statistics:
Number of Nets                 : 11908
Average number of Pins per Net : 3.26
Maximum number of Pins in Net  : 76
------ I/O Port summary
Number of Primary I/O Ports    : 100
----- Design Rule Checking:
Number of Output Pins connect to Power/Ground *: 0
Number of Insts with Input Pins tied together ?:
1362
Number of TieHi/Lo term nets not connected to
instance's PG terms ?: 0
# Number of cells of input netlist marked dont_use
= 42.
......
Design check done.
 *** Message Summary: 69 warning(s), 0 error(s)
```

Check Timing Constraints

```
check_timing [-type type_list] [-verbose] [-view view_name] \
[port_or_pin_list] [-check_only warning_list]\
[-exclude_warnings warning_list] [-include_warnings warning_list]
```

This command performs variety of consistency and completeness checks on the timing constraints specified for a design.

- *Type can be clocks, clock_cliping, constant_collision, endpoints, inputs, clock_gating_inferred, multiple_clocks, or loops.*

Reports looks like this,

```
                                TIMING CHECK SUMMARY
-----------------------------------------------------------------------------------------
     Warning                          Warning Description                Number of Warnings
-----------------------------------------------------------------------------------------
    clock_expected          Clock not found where clock is expected        4
    no_drive                No drive assertion                             98
    no_input_delay          No input delay assertion with respect to clock 18
    uncons_endpoint         Unconstrained signal arriving at end point     1732
```

Check Parasitic Coverage

```
report_annotated_parasitics [-help] [-include_nets_file string]\
[-view { viewName }] [> {write_fileName}] [{[-list_annotated ] \
[-broken_nets ] [-floating_nets ] [include_nets] \
[-max_missing {max_missing_int}]}]
```

This Command reports backannotated parasitics of the design.

- *The software groups all the nets by net types and annotation status (annotated or not annotated)*

Reports looks like this,

```
# Summary of Annotated Parasitics:
+-------------------------------------------------------------------------+
|    Net Type      |     Count    |   Annotated (%)    | Not Annotated (%) |
+------------------+--------------+--------------------+-------------------+
| total            |       11285  |       10238  90.7% |        1047   9.3% |
+------------------+--------------+--------------------+-------------------+
| 0-term:floating  |          968 |           0   0.0% |         968 100.0% |
| 1-term:no drive  |           24 |           0   0.0% |          24 100.0% |
| 1-term:no load   |           54 |           0   0.0% |          54 100.0% |
| real net         |       10239  |       10238 100.0% |           1   0.0% |
+-------------------------------------------------------------------------+
+-------------------------------------------------------------------+
| Annotated |    Res (MOhm)    |    Cap (pF)      |    XCap (pF)     |
+-----------+------------------+------------------+------------------+
| Count     |        207111    |        217321    |         31888    |
| Value     |        1.7155    |       54.3030    |        5.3840    |
+-------------------------------------------------------------------+
```

Here we mainly focus on real net annotated or not annotated.

MMMC Design Import Method in Tempus Stylus

Tempus supports MMMC (Multi-Mode Multi-Corner) method of importing data.

- *read_mmmc reads in the full set of timing libraries, constraints and the associated modes in the design.*
- *If these read_libs, read_mmmc, or read_netlist commands are used then the software schedules these files for reading. The files are actually read while init_design command applied.*
- *write_mmmc command writes out the MMMC file from the current session.*

Task	Logical Data	Physical Data	OA Physical Data
Load Design	*read_mmmc <file>* read_netlist <> -top <> init_design read_def	read_physical -lef *read_mmmc <file>* read_netlist <> -top <> init_design read_def	read_libs -oa_ref_libs <> *read_mmmc <file>* read_netlist <> -top <> init_design
Save Design	write_db out_dir	write_db out_dir	write_db -oa_lib_cell_view <>
Restore Design	read_db {session.dat topcell}	read_db {session.dat topcell} -physical_data	read_db -physical_data \ -oa_lib_cell_view <>

MCMM File

A View Definition file is one of the input data needed for MMMC Signoff Analysis flow. It contains different design views to be analyzed.
An analysis view is a design scenario analyzed using a specific combination of library PVT, constraints and RC extraction.

***Flow**:*

1. *Creating Library Sets*
2. *Creating RC Corners*
3. *Creating Timing Conditions*
4. *Creating Delay Corners*
5. *Creating Constraints Modes*
6. *Creating Analysis Views*
7. *Setting Analysis Views*

We strive to be the most trusted partner for the world's hardest engineering problems.

Library Set:

create_library_set
-name PVT_1P50V_40C
-timing [list lib_1 lib_2 …]
-si {….}
-socv {….}
-aocv {….}
-lvf {….}

```
create_library_set -name PVT_1P35V_60C\
   -timing\
   [list [redacted]ccs_power_slow_1_35V_60C.lib\
         [redacted]_UPF_slow_1_08V_125C.lib\
         [redacted]UPF_slow_1_08V_125C.lib]
```

First, we have to create a library set. Where we can provide Timing libs, Si libs, PVT setting and derating.

RC Corners

create_rc_corner
-name rc_0C
-qrc_tech <qrc tech file>

```
create_rc_corner -name rc_0c\
   -pre_route_res 1\
   -post_route_res 1\
   -pre_route_cap 1\
   -post_route_cap 1\
   -post_route_cross_cap 1\
   -pre_route_clock_res 0\
   -pre_route_clock_cap 0\
   -temperature 0\
   -qrc_tech [redacted]QRC-Min/qrcTechFile
```

Timing Condition

create_timing_condition
-name fast_1p98_0c
-library_sets [list PVT_1P50V_40C]

```
create_timing_condition -name fast_1p98_0C\
   -library_sets [list PVT_1P98V_0C]
```

We strive to be the most trusted partner for the world's hardest engineering problems.

Delay Corner

Delay corner consists of timing condition and RC corner.

create_delay_corner
-name fast_1p98_0C
-timing_condition {fast_1p98_0C}
-rc_corner rc_0c

```
create_delay_corner -name fast_1p98_0C\
    -timing_condition {fast_1p98_0C}\
    -rc_corner rc_0c
```

Constraints Mode

create_constraint_mode
-name func
-sdc_files [list name.sdc]

```
create_constraint_mode -name func\
    -sdc_files\
     [list                                                
```

Reporting Analysis View

```
report_analysis_views
[-type all |setup |hold |
active]
```

Default: all

The report_analysis_views command generates a hierarchical report of all defined views in the design.

- *By default, this command generates for all of the defined views.*
- *-type option to generate a report on specific type of analysis view.*
- *This command should be used after setting active setup and hold views (set_analysis_view) for the design.*

Report should be like this,

```
Multi-mode Analysis View Report
+ ALL Views
    |
    + Analysis View:  view1
      |
      + Delay Calc Corner: default_corner_max
      |    |
      |    + default Power Domain
      |    |    |
      |    |    + library_set libSetSlow
      |    |    |    |
      |    |    |    + timing: ./libs/slow.lib
      |    |    |
      |    |    + rc_corner rcCmax
      |    |
      |    + pd_1Volt Power Domain
      |    |    |
      |    |    + library_set libSetTyp
      |    |    |    |
      |    |    |    + timing: ./libs/typical.lib
      |
      + Constraint Mode: mode1
           |
           + SDC Constraint Files: mission.sdc
```

Timing Analysis and Reporting

Tempus provides various timing analysis mode. This choice of analysis mode is influenced by,

- *How timing library data is chosen and used during calculation.*
- *How delay calculation is performed.*
- *How timing constraint information is interpreted.*

There are three type of analysis modes,

1. *Single:* *Also known as the "max-only"*
2. *BCWC:* *Best-Case/Worst-Case*
3. *OnChipVariation:* *OCV*

Single Analysis Mode:

In this mode, Tempus software uses a single set of delays (using one library group) based on one set of process, temperature, and voltage conditions. Only maximum delay values are used for delay calculation; early and late path delays can be the min and max of this range arising due to variations in sdf_cond, timing derates, and other inputs.

For setup check, the software checks the late launch clock and late data paths against the early capture clock path.

For hold check, the software compares the early arriving data against the late arriving clock at the endpoint.

To specify the single analysis mode, set:

```
read_libs -max_libs maxLib
set_db timing_analysis_type single
set_db timing_analysis_check_type setup
```

Best-Case Worst-Case Analysis Mode

In best-case worst-case (BCWC) analysis mode, the software uses delays from the maximum library group for all the paths during setup checks and the minimum library group for all the paths during hold checks. In this mode, Tempus software considers two operating conditions and checks both operating conditions in one timing analysis run.

For setup/hold check, the software calculates delay values from the maximum/minimum library group, respectively, for data arrival time and network delay of both launch and capture clocks (in propagated mode).

To specify the BCWC analysis mode, set the following:

```
read_libs -max_libs maxLib -min_libs
  minLib

set_db timing_analysis_type
  best_case_worst_case
```

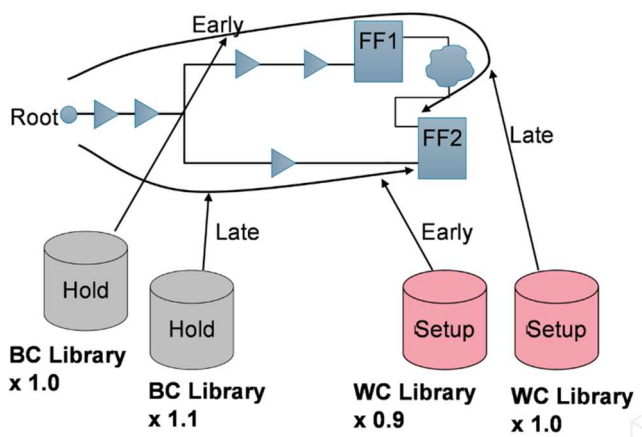
Best-Case Worst-Case Analysis with Time Derating

Using time derating makes the checks more pessimistic by scaling delay for early and late paths.
BCWC for pessimistic setup checks speeds up the clock to FF2. This can effectively make the data from FF1 late to FF2 D-input, causing a setup time violation.

set_timing_derate -max -early 0.9 -late 1.0

BCWC for hold checks makes the latching clock path 10 percent slower. This shifts the clock edge closer to the data edge, revealing a hold failure.

set_timing_derate -min -early 1.0 -late 1.1

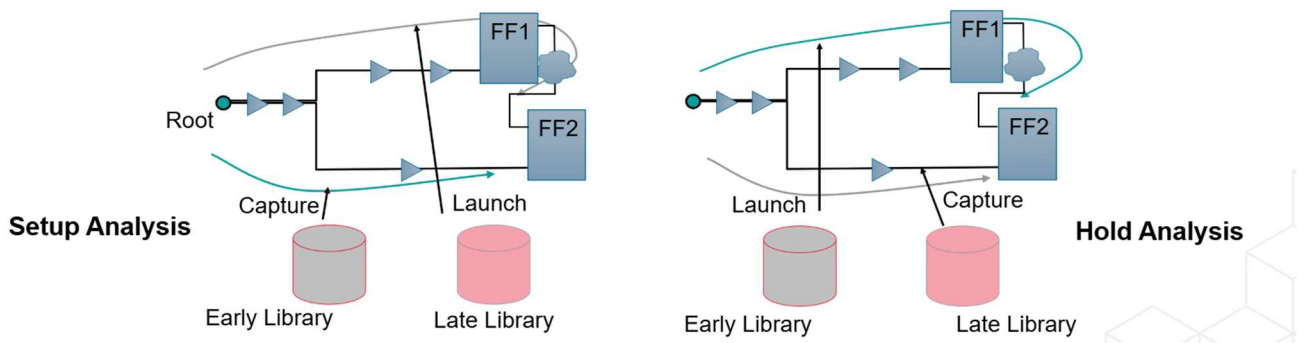


On-Chip Variation Analysis Mode

In the on-chip variation analysis mode, the software calculates the delay for one path based on the maximum operating condition while calculating the delay for another path based on the minimum operating condition for setup and hold checks. These delays are used together in the analysis of each check.

To specify this mode, set the following,

```
read_libs -max_libs Late -min_libs Early ; set_db timing_analysis_type ocv ;
```



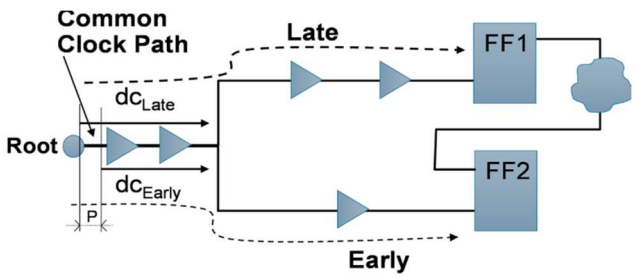
In the figure, for setup analysis, Capture path is taken from the Early library, Launch path and Data path is taken from Late library.
On the other hand, for hold analysis Capture is taken from Late library, , Launch path and Data path is taken from Early library.

CPPR/CRPR

Clock Path Pessimism Removal (CPPR) or Clock Reconvergence Pessimism Removal (CRPR) is the process of identifying and removing the pessimism introduced in the slack reports for clock paths when the launch and capture clock paths have a segment in common

In a CPPR calculation, the difference between the late and early delays for the common clock segment is calculated and then adjusted in the slack calculations as mentioned here:

- *The pessimism: P = dcLate - dcEarly*
- *New slack = slack(w/o CRPR) + P*



To account for CRPR in a common clock path, use:
set_db timing_analysis_check_type setup/hold/both
set_db timing_analysis_cppr setup/hold/both

We strive to be the most trusted partner for the world's hardest engineering problems.

Reporting Timing

```
report_timing [-check_type {setup|hold|clock_gating_setup|..}]  \
[-retime {aocv|ssta|path_slew_propagation|..}]                  \
[-late| -early] [-through pin_list] [-to pin_list][-from pin_list]
```

A timing report is generated that provides information about the various paths in the design. The timing report contains this information:

- *The slack times of the arriving signal at the end node and the associated transitions.*
- *The start/end nodes of each path.*
- *The signal-required times and the actual signal arrival times.*
- *Summary of propagated vs. ideal status of launching and capturing clocks.*
- *Any phase shifts/CPPR values applied to timing check.*

```
#############################################################
#  Generated by:      Cadence Tempus 18.10-b016_1
#  OS:                Linux x86_64(Host ID noi-sakshin)
#  Generated on:      Thu May 10 12:50:10 2018
#  Design:            super_filter
#  Command:           report_timing
#############################################################
AAE DB initialization (MEM=797.918 CPU=0:00:00.1 REAL=0:00:00.0)
#################################################################################
# Design Name: super_filter
# Design Mode: 65nm
# Analysis Mode: MMMC OCV
# Parasitics Mode: SPEF/RCDB
# Signoff Settings: SI Off
#################################################################################
Using master clock 'ref_clk' for generated clock 'pll_clk' in view 'AV_wc_on'
Start delay calculation (fullDC) (1 T). (MEM=806.535)
End delay calculation. (MEM=870.844 CPU=0:00:00.8 REAL=0:00:00.0)
End delay calculation (fullDC). (MEM=862.844 CPU=0:00:01.1 REAL=0:00:02.0)
Path 1: MET Setup Check with Pin AO1/SUM_reg_3_15/CK
Endpoint:   AO1/SUM_reg_3_15/D (^) checked with  leading edge of 'pll_clk'
Beginpoint: bypass             (^) triggered by  leading edge of 'io_clk'
Path Groups: {pll_clk}
Analysis View: AV_wc_on
Other End Arrival Time          0.000
- Setup                         0.241
+ Phase Shift                   8.000
= Required Time                 7.759
- Arrival Time                  5.803
= Slack Time                    1.956
    Clock Rise Edge                      0.000
    + Input Delay                        0.750
    = Beginpoint Arrival Time            0.750
```

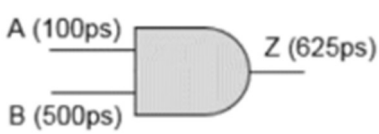
What is GBA?

Static Timing Analysis (STA) software, by default, uses a pessimistic algorithm for timing, which is based on the worst slew propagation (slew merging), referred to as graph-based analysis.

In the GBA mode, the software considers both the worst arrival and the worst slew in a path during timing analysis, even if the worst slew corresponds to an input pin different than the relevant pin for the current path.
This approach is used during the initial timing analysis before the final signoff.

- *This reduces analysis runtime of the whole design.*
- *But it gives a pessimistic result.*

Here, it is assumed that for any input slew, the output slew is 25% more than the input slew. If the slew at B is 500ps, then the slew at Z is 625ps.
In GBA, for calculating delay and slew propagation through this AND gate, the worst input slew (through B) is always considered, irrespective of the fact that the path is through pin A or B.



We strive to be the most trusted partner for the world's hardest engineering problems.

What is PBA?

Path Based Analysis (PBA) involves re-timing the components of a timing path based on the actual slew that is propagated in this path.

- *It considers path-based slews and actual arrival times for both base and SI delay calculations.*
- *It removes the pessimism that is introduced due to slew merging at various nodes in the design when graph-based analysis is run.*
- *It is recommended to use during the final stages of timing closure to ensure accuracy.*

This reduces the area, power dissipation, and ECO cycles of your design but imposes a huge runtime hit.

PBA Report File Example:

```
#  Design:            dtmf_recvr_core
#  Command:           report_timing -check_type setup -from TDSP_CORE_INST/DECODE_INST/ir_reg* -retime path_slew_propagation -to RESULTS_CONV_INST/r1477_reg[10]/D
##############################################################
Path 1: MET (1.223 ns) Setup Check with Pin RESULTS_CONV_INST/r1477_reg[10]/CK->D
    Retime Analysis:  Data Path-Slew
               View: scan_slow_RCMAX
              Group: m_rcc_clk
        Startpoint: (R) TDSP_CORE_INST/DECODE_INST/ir_reg[9]/CK
             Clock: (R) m_clk
          Endpoint: (R) RESULTS_CONV_INST/r1477_reg[10]/D
             Clock: (F) m_rcc_clk

                     Capture       Launch
       Clock Edge:+   20.000        0.000
      Src Latency:+    0.000        0.000
      Net Latency:+    1.519 (P)    5.306 (P)
          Arrival:=   21.519        5.306

             Setup:-    0.734
       Uncertainty:-    0.400
     Required Time:=   20.386
      Launch Clock:=    5.306
         Data Path:+   13.857
             Slack:=    1.223
    Slack(original):=    1.038

#-------------------------------------------------------------------------------------------------------------------------------
# Timing Point                                          Arc     Edge  Cell       Fanout  Delay   Delay   Trans   Trans  Arrival
#                                                                                          (ns)  Retime   (ns)  Retime     (ns)
#-------------------------------------------------------------------------------------------------------------------------------
  TDSP_CORE_INST/DECODE_INST/ir_reg[9]/CK               CK      R     (arrival)          1       -       -   0.145   0.145    5.306
  TDSP_CORE_INST/DECODE_INST/ir_reg[9]/Q                CK->Q   F     SDFFS_X2           1   0.687   0.687   0.145   0.146    5.993
  TDSP_CORE_INST/DECODE_INST/FE_OFC186_port_address_1_/Z A->Z   F     BUF_X16           44   0.557   0.557   0.146   0.485    6.550
  TDSP_CORE_INST/TDSP_CORE_GLUE_INST/p8211A/ZN          A3->ZN  R     NOR3_X1            2   0.901   0.901   0.485   1.125    7.451
  TDSP_CORE_INST/TDSP_CORE_GLUE_INST/p8090A/ZN          A2->ZN  F     NAND2_X1           1   0.361   0.359   1.138   0.385    7.810
  TDSP_CORE_INST/TDSP_CORE_GLUE_INST/p6871A/ZN          A4->ZN  R     NOR4_X2            2   1.874   1.760   0.660   2.743    9.570
  TDSP_CORE_INST/TDSP_CORE_GLUE_INST/p7851A6133/ZN      A1->ZN  F     NAND2_X4           2   0.528   0.527   2.748   0.941   10.098
  TDSP_CORE_INST/TDSP_CORE_GLUE_INST/p7333D/ZN          A2->ZN  R     NOR2_X4           16   2.574   2.573   0.942   3.733   12.671
  TDSP_CORE_INST/TDSP_CORE_GLUE_INST/p7551A6117/ZN      B1->ZN  F     AOI222_X1          1   1.476   1.476   3.733   1.609   14.147
  TDSP_CORE_INST/TDSP_CORE_GLUE_INST/p8505A6087/ZN      A2->ZN  R     NAND3_X2           1   1.851   1.851   1.609   1.654   15.998
  FE_OFC12_tdsp_data_out_10_/Z                          A->Z    R     BUF_X16            1   0.731   0.731   1.654   0.389   16.729
  FE_OFCC479_FE_OFN12_tdsp_data_out_10_/Z               A->Z    R     BUF_X16            5   0.695   0.695   0.389   0.891   17.424
  RESULTS_CONV_INST/p7148A29049/ZN                      A->ZN   F     INV_X2             1   0.288   0.288   0.891   0.344   17.711
  RESULTS_CONV_INST/FE_OFCC480_n_2024/Z                 A->Z    F     BUF_X32            6   0.615   0.615   0.344   0.148   18.327
  RESULTS_CONV_INST/p7155A28968/ZN                      A2->ZN  R     OAI22_X1           1   0.836   0.836   0.148   1.418   19.163
  RESULTS_CONV_INST/r1477_reg[10]/D                     D       R     SDFF_X2            1   0.000   0.000   1.735   1.418   19.163
#-------------------------------------------------------------------------------------------------------------------------------
INFO: Path Based Analysis (PBA) performed on total '1' paths
```

Advance Analysis

Ultra-Low Vdd Variation Modeling in Tempus

Delay is a very strong function of Vdd and Vt

- *Delay increases >10x from 1V to 0.4V*

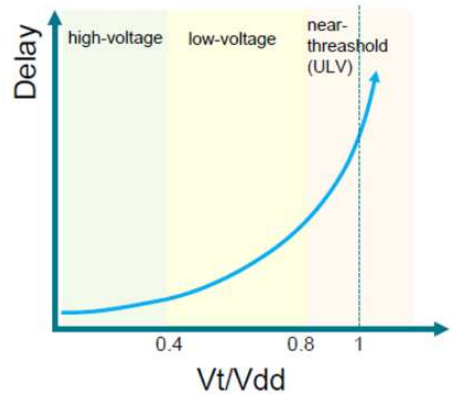
Near-threshold (NT) regimes are when Vt/Vdd> 0.8

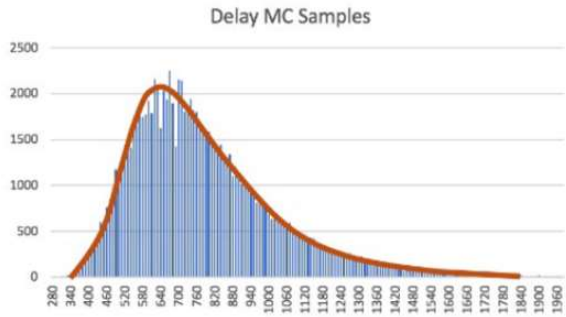
Challenges in STA and characterization

- *Need a wide range of slews in .lib*
- *Wide non-Gaussian distributions need accurate LVF with moments*
- *Tails of transitions play a critical role*

Tempus is fully equipped to accurately handle near-threshold corners

- *Uses current-source technology to model waveform distortions*
- *Propagates detailed waveform along the graph*
- *Advanced SOCV analysis (Ultra mode), correlation and nonlinear effects*





What Are On-Chip Variation and Their Sources?

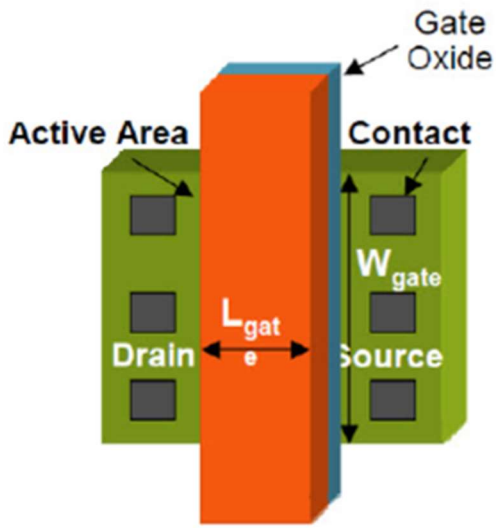
Chip Variations are the intrinsic variability of semiconductors subjected to process variations.

Changes in physical parameters affects electrical parameters, in turn causing delay variations.

- *Variation in length, width, thickness.*
- *Doping variation*

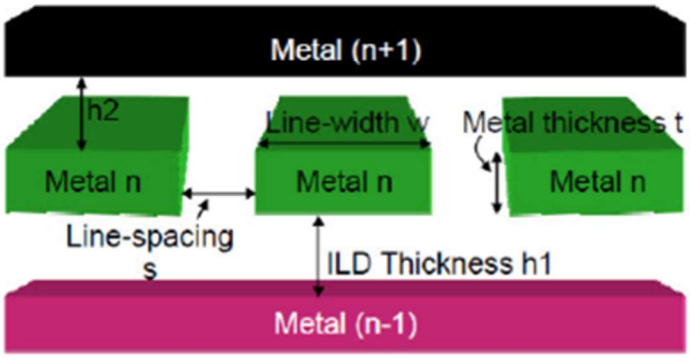
Key Device parameters:

- L_{gate} – channel length
- V_T – threshold voltage
- T_{ox} – oxide thickness
- W_{gate} – Channel Width



Key Interconnect Parameters

- Metal thickness – t
- Dielectric thickness – h_1, h_2
- Metal line width (or line spacing) – w or s



OCV in STA:

- *Affects wire and cell delays.*
- *Clock and data paths affected differently.*
- *Increases pessimism in the design.*
- *Location and depth based variations.*

How to model On-Chip Variations!!!

OCV, AOCV and SOCV !!!

What Is Advanced OCV (AOCV)?

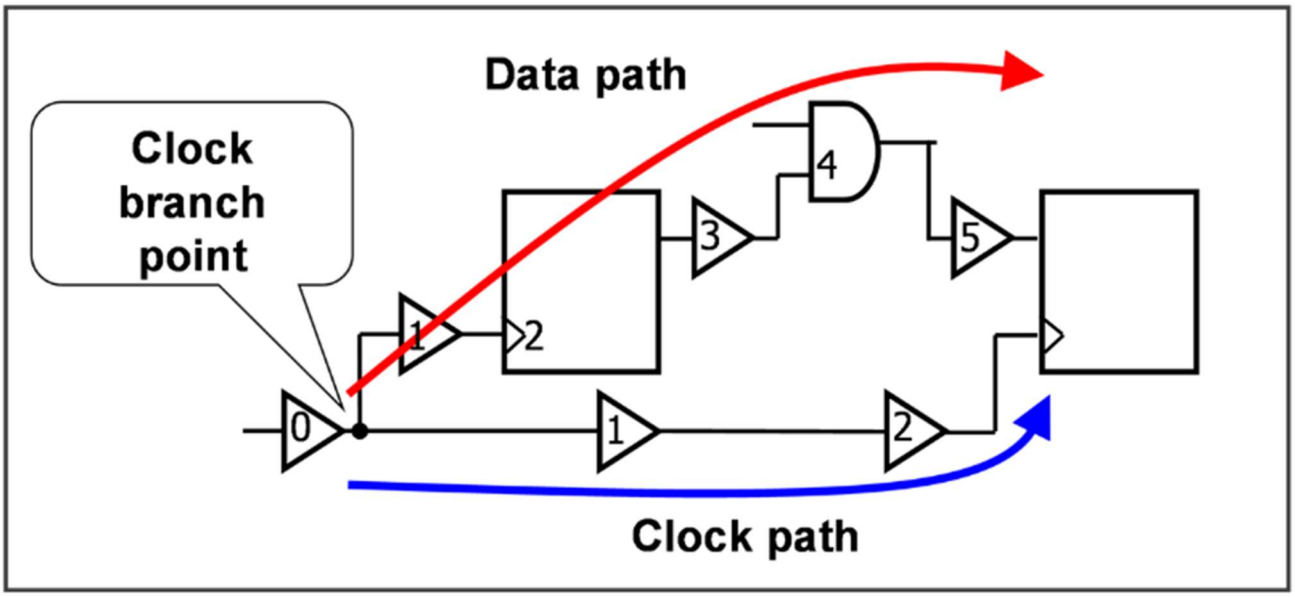
AOCV is a design approach where derates applied to the cells vary with path depth or distance or both.

The shortcoming of OCV is that timing closure gets difficult due to extra pessimism added with fixed derates for the cells.

AOCV reduces the level of derating for each stage on the basis that each successive stage will cancel out the variation.

"Some stages will be faster, others slower. So, the more stages you have, the more it averages out."

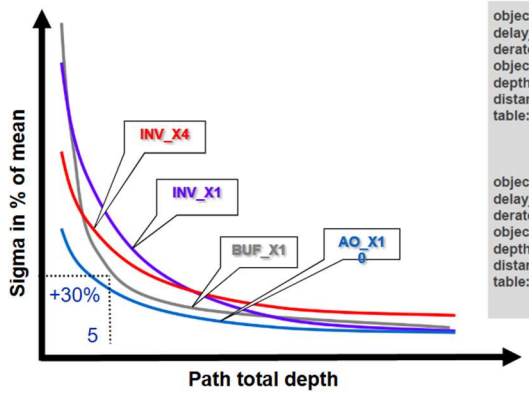
Here, it shows the derate table for a buffer cell based on the depth along the timing graph.



Depth	1	3	5	7	10	15	20	30
Late derate	1.6	1.4	1.3	1.25	1.21	1.17	1.12	1.1
Early derate	0.5	0.6	0.7	0.8	0.88	0.89	0.91	0.95

AOCV: Calculation

In AOCV mode, the software uses derating libraries where an AOCV factor is chosen so that when multiplied by the lib nominal delay value, you get close to the mean ± 3 sigma value for that many stages in the path. As the number of stages increases, the sigma value relative to the mean decreases on the order 1/sqrt(n) and decreases the AOCV derate value.




```
object_type: lib_cell
delay_type: cell
derate_type: early
object_spec: myGate
depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
distance:
table: 0.81 0.84 0.85 0.86 0.87 0.88 0.88 0.89 0.89 0.90 0.91 0.91 0.91 0.91 0.91 0.916 0.92 0.92 0.92 0.92 0.92 0.92 0.93 0.93 0.93
```

Both Early and Late AOCV tables approach 1.0 with increasing path depth

```
object_type: lib_cell
delay_type: cell
derate_type: late
object_spec: myGate
depth: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
distance:
table: 1.28 1.24 1.21 1.19 1.18 1.17 1.16 1.15 1.14 1.13 1.13 1.12 1.12 1.11 1.11 1.10 1.10 1.09 1.09 1.09 1.09 1.08 1.08 1.08 1.08
```

Basic Clock Constraints

Creating clocks

Example 1: A counter runs at a clock frequency = 100 MHz. The clock comes from an input port.

```
# Create clock at the input port clk
# Case 1: The clock has 0° phase shift, 50% duty cycle /``\__/
create_clock -name clk -period 10.00 -waveform {0 5} [get_ports clk]
# Case 2: The clock has a different duty cycle = 60% /```\_/
create_clock -name clk -period 10.00 -waveform {0 6} [get_ports clk]
# Case 3: The clock has 180° phase shift \__/``\
create_clock -name clk -period 10.00 -waveform {5 10} [get_ports clk]
```

Example 2: A counter runs at a clock frequency = 100 MHz. The clock is driven by a divide-by-2 clock divider circuit implemented in the RTL. The master clock comes from an input port.

```
# Create base clock at the input port
create_clock -name clkin -period 10.00 -waveform {0 5} [get_ports i_clk]
# Create divided clock from the base clock
create_generated_clock -name clk -divide_by 2 -source [get_ports i_clk] [get_pins {tff/Q}]
# OR
create_generated_clock -name clk -divide_by 2 -source [get_pins {tff/CLK}] [get_pins {tff/Q}]
```

Notes

- Generated clocks are treated by the STA tool as synchronous to master/base clocks and automatically infers source latency = source latency of base clock + latency between the base and generated clock definition points.
- If you create the divided clock as master clock instead of using *create_generated_clock*, the disadvantage is that it will be treated as asynchronous clock. Hence, latencies should be explicitly defined. So, this is recommended to be avoided.

Example 3: A counter is clocked by a clock mux which selects between clocks = 50/100/200 MHz, which are driven from input ports. The clock mux is internal to the design and the clocks don't interact outside the mux.

```
# Create all clocks at the input ports
create_clock -name clkA -period 20.00 -waveform {0 10} [get_ports clkA]
create_clock -name clkB -period 10.00 -waveform {0 5} [get_ports clkB]
create_clock -name clkC -period 5.00 -waveform {0 2.5} [get_ports clkC]

# Define logically exclusive clocking
set_clock_groups -logically_exclusive -group {clkA} -group {clkB} -group {clkC}

# OR # Define false paths between clock domains (not recommended, deprecated practice…)
set_false_path -from clkA -to clkB
set_false_path -from clkA -to clkC
set_false_path -from clkB -to clkA
set_false_path -from clkB -to clkC
set_false_path -from clkC -to clkA
set_false_path -from clkC -to clkB
```

Notes

- The clocks are "logically exclusive" because even though all the clocks exist together inside the design, only one of the clocks can drive the counter at a time.
- If the select signal is dynamic, the mux requires clock gating checks. Clock gating checks are automatically inferred by the STA tool for simple cells like clock mux.
- If the select signal is static, it can be constrained by *set_case_analysis* <0/1/2> <select pin>. This performs a constrained timing analysis. If *set_case_analysis* is not used, all the three clocks are taken into account for timing analysis.
- No need to create generated clock at the mux output as the input clocks should be auto-propagated for simple cells like clock mux.

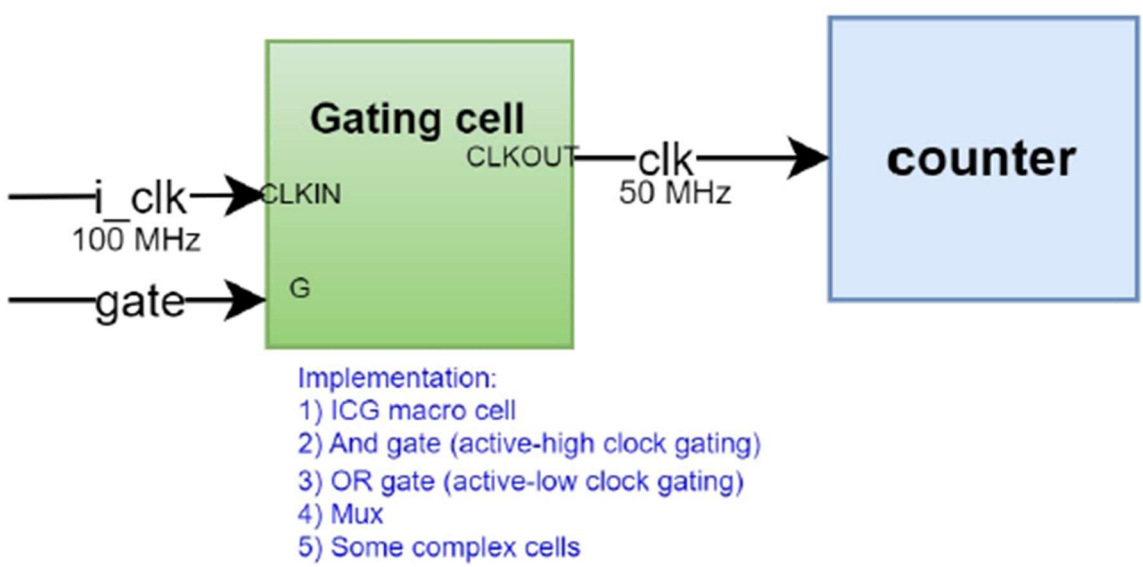
Example 4: A counter is clocked by a divide-by-4 clock divider circuit. The master clock of the clock divider is driven by a clock mux which selects between clocks = 50/100/200 MHz, coming from input ports. The clock mux is internal to the design and they don't interact outside the mux.

```
# Create all base clocks at the input ports
create_clock -name clkA -period 20.00 -waveform {0 10} [get_ports clkA]
create_clock -name clkB -period 10.00 -waveform {0 5} [get_ports clkB]
create_clock -name clkC -period 5.00 -waveform {0 2.5} [get_ports clkC]

# Create generated clock at the divider output from each base clock
create_generated_clock -name clkA_by_4 -divide_by 4 -source [get_ports clkA] [get_pins
{clkdvd/ff1/Q}]
create_generated_clock -name clkB_by_4 -divide_by 4 -source [get_ports clkB] [get_pins
{clkdvd/ff1/Q}] -add
create_generated_clock -name clkC_by_4 -divide_by 4 -source [get_ports clkC] [get_pins
{clkdvd/ff1/Q}] -add

# Define logically exclusive clocking
set_clock_groups -logically_exclusive -group {clkA clkAgen} -group {clkB clkBgen} -group {clkC
clkCgen}
```

Notes

- Pay attention to the grouping of clocks in the exclusion constraint. Synchronous clocks (base and generated) are grouped together.

Example 4: A counter is clocked by a gated clock = 100 MHz from a clock gating cell. The gate signal is dynamically driven in the design. The ungated master clock comes from an input port.

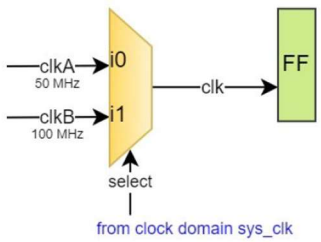


```
# Create base clock at the input port
create_clock -name clkin -period 10.00 -waveform {0 10} [get_ports i_clk]

# Create gated clock at the output of the gating cell
create_generated_clock -name clk -divide_by 1 -source [get_pins igc/CLKIN] [get_pins {igc/CLKOUT}]

# Set clock gating check on the gating cell
set_clock_gating_check -high [get_cells ANDGATE]
set_clock_gating_check -low [get_cells ORGATE]
```

Notes

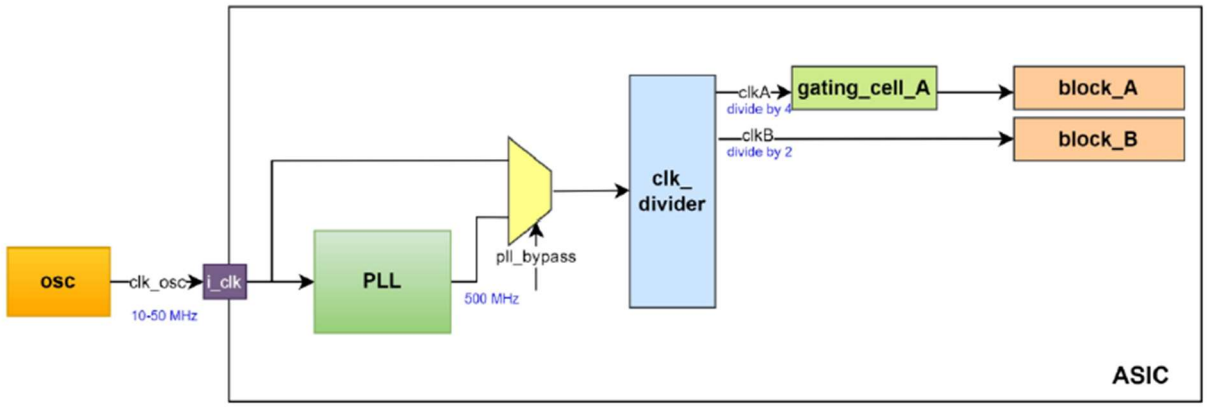
- In case of simple gating cells like AND, OR gate, or ICG macro cells, no need to specify clock gate check as it is automatically inferred.
- In case of simple gating cells like AND, OR gate, or multiplexers, or ICG macro cells, the tool should be able to auto-propagate the clock to the output. In such cases, no need to create generated clock.
- In case of multiplexers, or latch-based gating cells, clock gating check should be explicitly specified.
- In case of complex cells where the setup/hold relation is unknown, the setup and hold margins should be explicitly mentioned for the clock gating check with appropriate flags *(-setup, -hold, -high, -low*).
- In case of multiplexers, the clock gating check happens on select pin. One or more of the input clocks, which you know by design cannot cause glitch on switching from/to, can be disabled from gating check by *set_disable_clock_gating_check <pin>*. And enable *set_enable_clock_gating_check -high/low* on the other pins.

Example 5: A design is clocked by a clock mux which selects between two clocks = 50/100 MHz, which are driven from input ports. The select signal is dynamically generated from another clock domain running at 25 MHz. By design, clkB is ensured to be low while switching the select signal. Derive the clock gating checks.




```
# Create all base clocks at the input port
create_clock -name clkA -period 20.00 -waveform {0 10} [get_ports clkA]
create_clock -name clkB -period 10.00 -waveform {0 5} [get_ports clkB]

# Set clock gating check only on i0 pin of the mux (active-high)
set_clock_gating_check -high [get_cells mux0]
set_disable_clock_gating_check [get_pins mux0/i1]
```

Example 6: A design has clock distribution as shown in the diagram. Assume that the PLL settings are dynamic, and gating cells are standard ICG macro cells. Constrain all clocks of this design.




```
# Create oscillator (OSC) clock = 50 MHz at the input port
create_clock -name clk_osc -period 200.00 -waveform {0 100} [get_ports i_clk]

# Create PLL output clock
create_clock -name pll_clk -period 2.00 -waveform {0 1.00} [get_pins {pll/CLKOUT}]

# Define logically exclusive clocking between OSC and PLL clocks
set_clock_groups -logically_exclusive -group {clk_osc} -group {pll_clk}

# Created generated clock clkA at the divider output from each base clock
create_generated_clock -name clkA_from_osc -divide_by 4 -source [get_ports i_clk] [get_pins
{clk_divider/ff1/Q}]
create_generated_clock -name clkA_from_pll -divide_by 4 -source [get_pins {pll/CLKOUT}] [get_pins
{clk_divider/ff1/Q}] -add

# Created generated clock clkB at the divider outputs from each base clock
create_generated_clock -name clkB_from_osc -divide_by 2 -source [get_ports i_clk] [get_pins
{clk_divider/ff1/Q}]
create_generated_clock -name clkB_from_pll -divide_by 2 -source [get_pins {pll/CLKOUT}] [get_pins
{clk_divider/ff1/Q}] -add

# Create gated clock for block_A at the gating cell output from each master clock
create_generated_clock -name clkA_from_osc_gated_ -divide_by 1 -source [get_pins
{gating_cell_A/CLKIN}] -master_clock [clkA_from_osc] [get_pins {gating_cell_A/CLKOUT}]
create_generated_clock -name clkA_from_pll_gated -divide_by 1 -source [get_pins
{gating_cell_A/CLKIN}] -master_clock [clkA_from_pll] [get_pins {gating_cell_A/CLKOUT}] -add
```

We strive to be the most trusted partner for the world's hardest engineering problems.

Creating clock uncertainties

Clock jitter and other pessimisms can be modelled as setup and hold uncertainties.

 # Intra-clock uncertainties
set_clock_uncertainty -setup 0.20 [get_clocks clk]
set_clock_uncertainty -hold 0.20 [get_clocks clk]

 # Inter-clock uncertainties (across different clock domains)
set_clock_uncertainty -setup 0.20 -from [get_clocks clk1] -to [get_clocks clk2]
set_clock_uncertainty -hold 0.20 -from [get_clocks clk1] -to [get_clocks clk2]

Creating clock transition times/slews

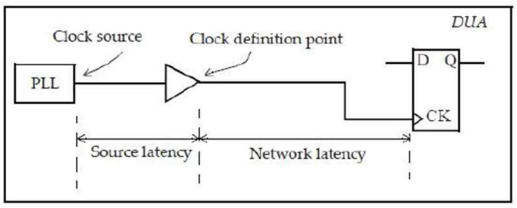
At the output of PLL models or an input port, the STA tool may not be able to compute the transition times automatically, and hence leads to less accurate timing analysis. In such cases, the transition time/slew can be explicitly specified at the source of the clock.

 # Clock transition times (rise/fall)
set_clock_transition -rise 0.10 [get_clocks clk]
set_clock_transition -fall 0.12 [get_clocks clk]

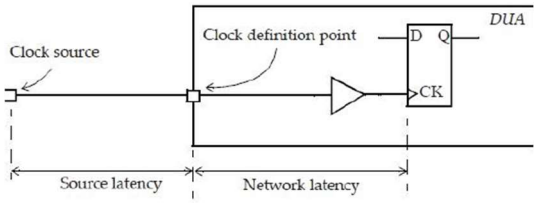
This specification applies only for ideal clocks and is disregarded once the clock trees are built, at which point, actual transition times at the clock pins are used. If a clock is defined at an input port, use the *set_input_transition* specification to specify the slew on the clock [1].

Creating clock latencies

Source and network latencies of clock can be specified. Source latency (insertion delay) is the on-chip or off-chip delay from the actual source of clock to the clock definition point. Network latency is the delay from clock definition point to a clock pin in a flip-flop or some clocked element.



(a) On-chip clock source.



```
# Network latency
set_clock_latency 1.80 -rise [get_clocks clk]
set_clock_latency 1.90 -fall [all_clocks]

# Use set_propagated_clocks in SDC after CTS in ASICs # Source latency
set_clock_latency 1.80 -max -source [get_clocks clk]
set_clock_latency 1.20 -min -source [get_clocks clk]
```

For ASICs, network latency specification is just an estimate used for STA before CTS and is disregarded once the clock trees are built. At this point after CTS, (actual network latency + source latency spec) is used as the net clock latency if set_propagated_clocks is set.

Flat Level STA

For flat level STA the inputs file will be changed slightly.

Flat	Hierarchical
Flat SDC	Top level SDC
Netlist of Top and Sub blocks	Netlist of Top
SPEFs of Top and Sub blocks	SPEFs of Top
lib file only for std cells	lib file for block level and std cells

In Top level STA the blocks act as std cells or macro, where we only need lib (or .db from Synopsys PT) file which contains timing information. The design will not be flattened and we will not be able to see the timing path through the block level. Here we need the timing constraints only for the top-level design.

But in the case of Flat level STA we can see the timing path through the block levels. For this we need the SPEFs and netlist of the block level. We do not need the timing library of the block level here. If there are any constraints going from the top-level design into the block level design then those constraints should be mentioned in the SDC. This we are calling flat SDC.