IMEC TRAINING

DIGITAL DESIGN FLOW

LOGIC SYNTHESIS

OUTLINE

Synthesis flow

Design environment

Constraints
- NanoSoc lab: system analysis and constraint strategy

Clock gate insertion

DFT: scan insertion, RAM wrappers
- NanoSoc lab: CG and DFT insertion

OUTLINE

Synthesis flow

Design environment

Constraints

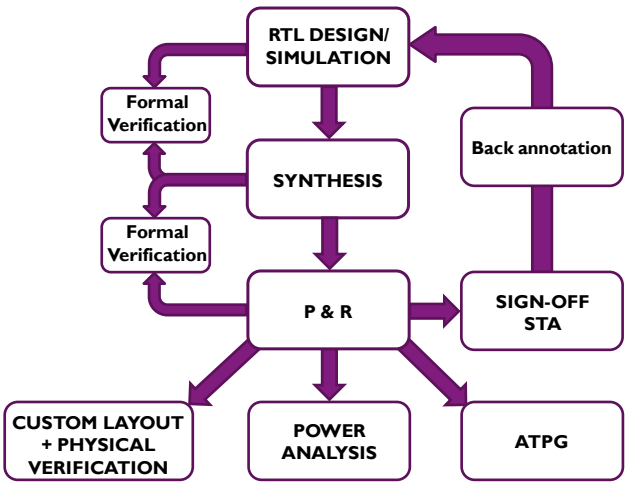
- NanoSoc lab: system analysis and constraint strategy

Clock gate insertion

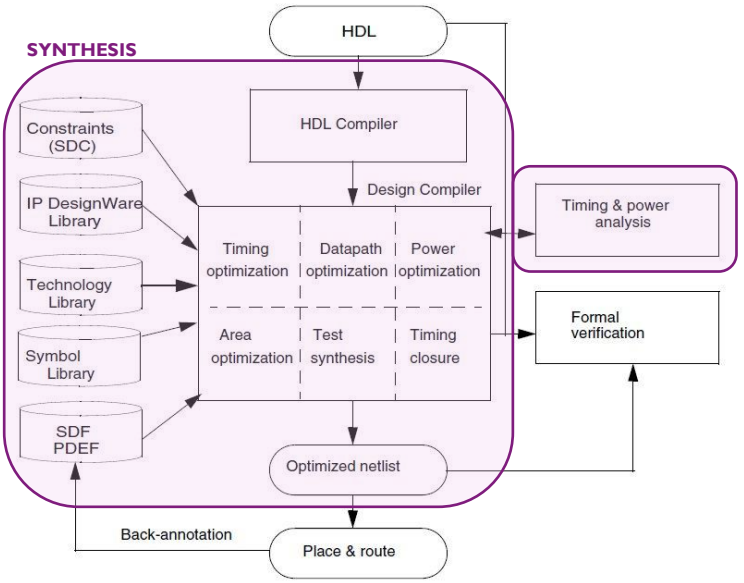
DFT: scan insertion, RAM wrappers

- NanoSoc lab: CG and DFT insertion

SYNTHESIS FLOW - DIGITAL DESIGN FLOW



SYNTHESIS FLOW - DIGITAL DESIGN FLOW



SYNTHESIS FLOW - SYNTHESIS ENGINES

4 basic engines in synthesis tool:

- HDL compilation: Converts the RTL (VHDL, verilog, systemverilog) into an intermediate database format
- Synthesis: Converts the intermediate format into a generic (library independent) logic level netlist
- Mapping: Converts the logic netlist into a gate level netlist of standard cells from a technology library
- Optimization: Optimizes the netlist to meet timing, area and power constraints
  - logic level optimization (library independent): $a+b= \overline{\bar{a}.\bar{b}}$
  - gate level optimization (library dependent): i.e. combined functions in one cell (AOI)

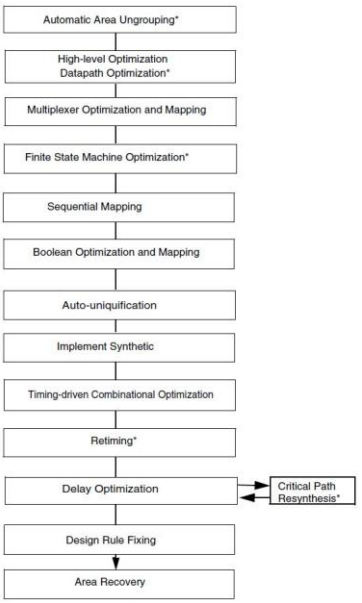
SYNTHESIS FLOW - CONVERSION COMMANDS

analyze = (HDL compiler) RTL syntax rule verification and translation into intermediate library

elaborate = synthesis + logic level optimisation

compile(_ultra) = mapping + gate level optimisation

SYNTHESIS FLOW - OPTIMIZATIONS



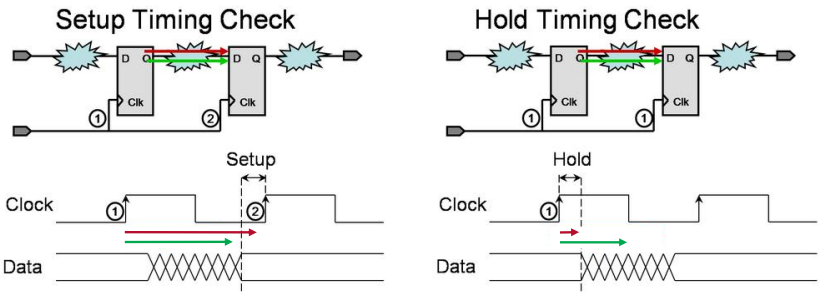
SYNTHESIS FLOW - STA

STA (static timing analysis) performs path delay calculations between:

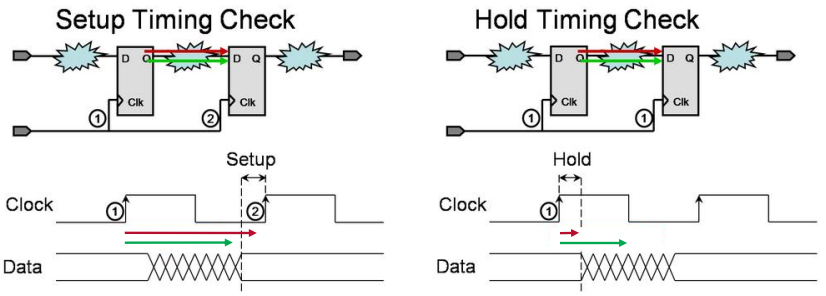
- 2 FF's (reg2reg)
- Inputs to FF's (in2reg)
- FF's to outputs (reg2out)
- Inputs to outputs (in2out)

Path delay calculations are done with STA engine in the process of optimization in order to find a gate netlist solution that meets the timing of the constraints file.

SYNTHESIS FLOW - STA



SYNTHESIS FLOW - STA



Slowest timing (WC)

Fastest timing (BC)

SYNTHESIS FLOW - STA

STA limitations:

- Synchronous design: clocks control when new data is processed
- Can't analyze combinatorial feedback loops
- Clock domain crossings have to be handled by design and described in constraints
- No functional verification: checks timing between any reg2reg, in2reg, reg2out and in2out path in the assumption that data changes every clock cycle (= regardless of real operation) unless constrained otherwise

OUTLINE

Synthesis flow

Design environment

Constraints
- NanoSoc lab: system analysis and constraint strategy

Clock gate insertion

DFT: scan insertion, RAM wrappers
- NanoSoc lab: CG and DFT insertion

DESIGN ENVIRONMENT

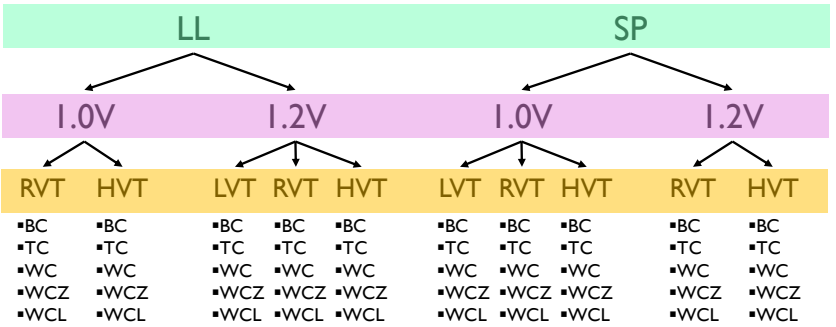
Inputs for synthesis:
- Design RTL (vhdl, verilog, systemverilog)
- Standard cell core/IO libraries
- Memory libraries
- IP libraries
- Analog macro libraries
- Wireload model
- Constraints for timing, area, power

.lib (.db):
•timing
•power
•area

Design environment

DESIGN ENVIRONMENT

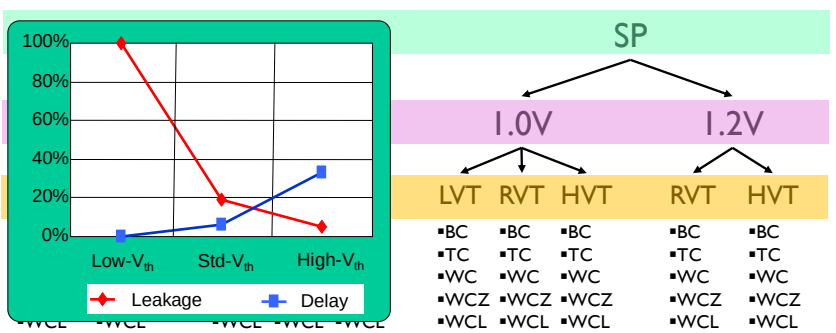
Technology flavours UMC 90nm:



Process flavour: trade-off static power/speed performance → chip level
Supply option: trade-off dynamic power/speed performance → block level
Threshold option: fine-tuning static power/speed performance → path/cell level

DESIGN ENVIRONMENT

Technology flavours UMC 90nm:

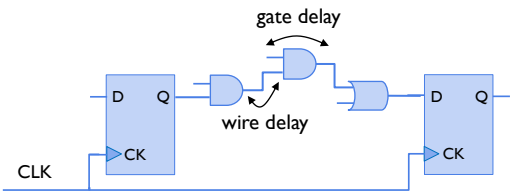


Process flavour: trade-off static power/speed performance → chip level
Supply option: trade-off dynamic power/speed performance → block level
Threshold option: fine-tuning static power/speed performance → path/cell level

DESIGN ENVIRONMENT

Environment settings:

- Gate delay: *set_operating_conditions*,*set_min_library*
  - Calculation based on cell characterisation
- Wire delay: *set_wire_load_model*
  - Estimation based on area of hierarchical module and fanout



DESIGN ENVIRONMENT

Gate delay - operating conditions:

```
set_operating_conditions -analysis_type bc_wc \
    -min BCCOM -min_library coreBC  -max WCCOM -max_library coreWC
set_operating_conditions -analysis_type bc_wc \
    -min BCCOM -min_library ioBC    -max WCCOM -max_library ioWC \
    -object_list [filter [find cell *] {@pad_cell == true}]
```

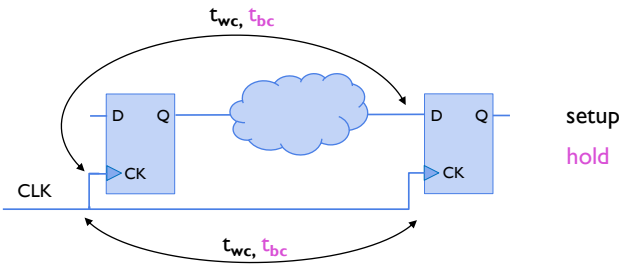
- analysis_type: bc_wc/on_chip_variation
- object_list: overrides previous operating conditions for objects defined → for padcells only 2nd operating conditions are used

Gate delay - library settings:

```
set_min_library coreWC.db  -min_version coreBC.db
set search_path "path/to/libs/and/hdl"
```

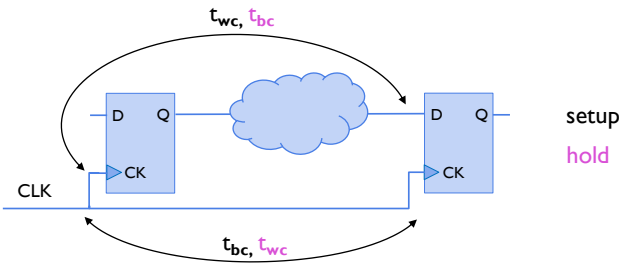
DESIGN ENVIRONMENT

Operating conditions: bc_wc



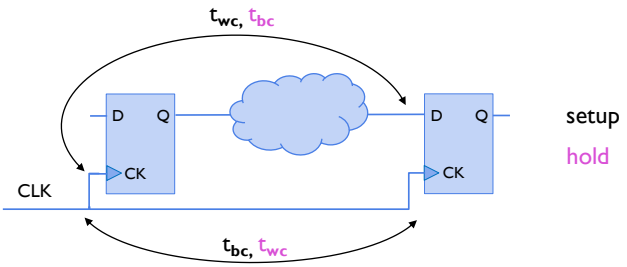
DESIGN ENVIRONMENT

Operating conditions: on_chip_variation



DESIGN ENVIRONMENT

Operating conditions: on_chip_variation



NOT USED: too pessimistic timing results, instead derating factors used on clock path in BE

DESIGN ENVIRONMENT

Wire delay - wireload model:

- Used for estimating load of interconnect based on module area and fanout
- Described in cell library

Library content:

```
                    .
                    .
 wire_load(enG2000K) {
  resistance  : 0.0;
  capacitance : 0.000175 ;
  area : 0.0;
  slope : 0.5;
  fanout_length(1, 584.6) ;
  fanout_length(2, 714.5) ;
  fanout_length(3, 811.1) ;.....
  fanout_length(50, 2097.2) ;
  fanout_length(90, 2576.0) ;
 }
  wire_load(enG5000K) {
  resistance  : 0.0;
  capacitance : 0.000175 ;
  area : 0.0;
  slope : 0.5;
  fanout_length(1, 1443.1) ;
  fanout_length(2, 1487.3) ;
  fanout_length(3, 1561.5) ;.....
  fanout_length(50, 2648.6) ;
  fanout_length(90, 3118.3) ;
 }
```

```
wire_load_selection(DEFAULT) {
        wire_load_from_area(0,43100,enG5K);
        wire_load_from_area(43100,86200,enG10K);
        wire_load_from_area(86200,258600,enG30K);
        wire_load_from_area(258600,431000,enG50K);
        wire_load_from_area(431000,862000,enG100K);
        wire_load_from_area(862000,1724000,enG200K);
        wire_load_from_area(1724000,4310000,enG500K);
        wire_load_from_area(4310000,8620000,enG1000K);
        wire_load_from_area(8620000,17240000,enG2000K);
        wire_load_from_area(17240000,43100000,enG5000K);
```

DESIGN ENVIRONMENT

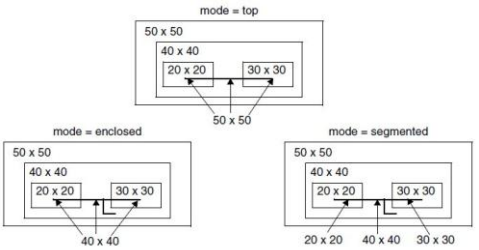
Wire delay - wireload model:

set auto_wire_load_selection true

- Automatically selects appropriate wireload model based on area of hierarchy level the wire is running in

set_wire_load_mode top/enclosed/segmented

- Area estimation model based on hierarchy of wiring



set_wire_load_selection_group -library libname WLgroupname

- Reference to the appropriate WL group in a library

DESIGN ENVIRONMENT

Macro compiler views:

- STA (synthesis, P&R, sign-off): .lib (.db)
- Timing simulation: .vhd, .v
- ATPG: .tmax
- P&R: .lef

OUTLINE

Synthesis flow

Design environment

Constraints
- NanoSoc lab: system analysis and constraint strategy

Clock gate insertion

DFT: scan insertion, RAM wrappers
- NanoSoc lab: CG and DFT insertion

CONSTRAINTS

Design rule constraints:
- all library related constraints defining boundaries of operation for capacitance, transition, fanout
- mostly defined in library but can also be redefined in design: can be specified for the full library or per cell
- highest priority: accuracy is only guaranteed within boundaries of operation of the gates, so when design rule constraints are met

Optimisation constraints:
- design related timing, area and power constraints defined by designer
- only timing constraints are requirements for correct operation
- area and power are merely desirable constraints with no influence on operation

CONSTRAINTS - DESIGN RULE CONSTRAINTS

Fanout:

set_max_fanout 4 [get_designs spi]

- defines allowed fanout for input port/design/libpin
- library: default_max_fanout is defined as general rule

Capacitance:

set_max_capacitance 0.5 [get_designs spi]

set_min_capacitance 0.1 [get_clocks SYSTEM_CLK]

- defines allowed capacitance on net connected to clock/input port/design/libpin
- library: max_capacitance is defined for each output

Transition:

set_max_transition 20 [get_lib_pin -of_object [get_pins I_SDO/O]]

- defines allowed transition on clock/port/design/libpin
- library: default_max_transition is general rule and/or max_transition on each input

CONSTRAINTS - TIMING CONSTRAINTS

Interaction with outside world:

- Clock timing (clock definition)
- Input/output timing (relation to clock)
- Input drive (input transition)
- Output load (output capacitance)

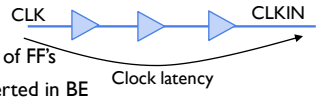
Architecture:

- Different operating modes (case analysis)
- Different clock domains (clock groups)
- Polling protocols between blocks (multi cycle paths, false paths)
- Synchronized inputs/signals (multi cycle paths, false paths)
- Clock muxes (case analysis)
- Clock dividers (clock definition)

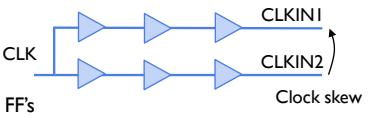
CONSTRAINTS - TIMING CONSTRAINTS

Clock timing:

create_clock -name SYSTEM_CLK -period 10 -waveform {0 5} [get_ports CLK] :
- defines waveform and period of clock



set_clock_latency 2 SYSTEM_CLK :
- defines a delay from the clock port to the clock input of FF's
- represents the delay of the clock tree that will be inserted in BE
- mostly effect on input/output port timing
- options: -source, -min, -max, -early, -late



set_clock_uncertainty 0.5 SYSTEM_CLK :
- defines a delay between different clock inputs of FF's
- represents the skew that may exist on the fanout of different clock branches
- options: -hold, -setup, -from/-to

set_clock_transition 0.1 SYSTEM_CLK:
- defines a transition time on the clock net
- options: -min, -max

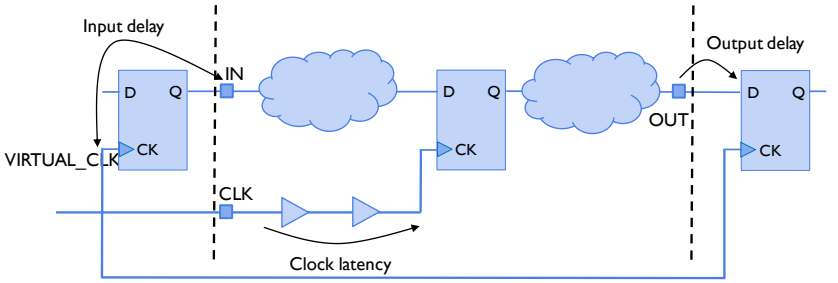
CONSTRAINTS - TIMING CONSTRAINTS

Input/output delays:

create_clock -name VIRTUAL_CLK -period 10 -waveform {0 5}
- defines ideal clock (without latency, skew) that serves as reference to define input/ouput delays

set_input_delay -clock VIRTUAL_CLK 3 [all_inputs]

set_output_delay -clock VIRTUAL_CLK 2 [all_outputs]



CONSTRAINTS - TIMING CONSTRAINTS

Input transitions:

set_input_transition 2.95 [get_ports IN]
- sets transition time on input necessary to calculate delay of input padcell
- options: -min, -max

Output load:

set_load 100 [all_outputs]
- sets output capacitance

set_fanout_load 3 [get_ports PAD_SEROUT]
- sets fanout of output port/design/libpin

CONSTRAINTS - TIMING CONSTRAINTS

Timing exceptions:

set_clock_groups -asynchronous -group SYSTEM_CLK -group SPI_CLK -name async1
- defines two clock domains where no clock crossing data flow needs not be verified

set_multicycle_path 2 -end -from ff1/CK -to ff2/D
- defines a double period margin to go from from-port to to-port
- options: -hold, -setup, -start, -end

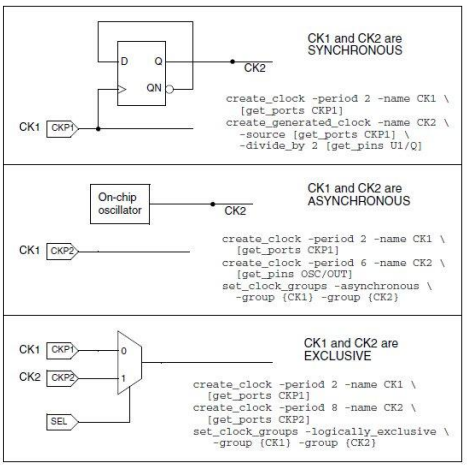
set_false_path -from ff1/CK -to ff2/D
- defines a path that may be ignored during STA
- options: -hold, -setup

set_max_delay 5.0 -from ff1/CK -to ff2/D
- defines a maximum delay between two nodes

set_case_analysis 1 I_SDO/E2
- defines the state of an input during STA
- to be used in multi mode designs or to set cell parameters

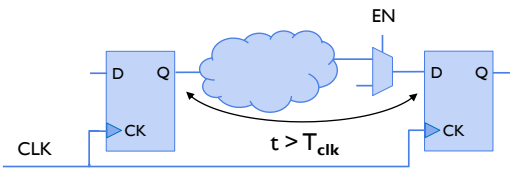
CONSTRAINTS - TIMING CONSTRAINTS

Clock relations:



CONSTRAINTS - TIMING CONSTRAINTS

Multi cycle path/false path:



EN = '1' every 2 CLK cycles

set_multicycle_path 2 -from FF1/CK -to FF2/D → relaxes constraint

set_false_path -from FF1/CK -to FF2/D → removes constraint!!!

CONSTRAINTS - MACRO

Macro .lib:

- Minimal required: design rule constraints
- If no timing available in .lib ⇒ define optimisation constraints:
  - If synchronous signals: *set_input_delay, set_output_delay* w.r.t. virtual clock
  - If combinatorial path through macro: *set_max_delay*
- If clock input plus output timing available related to clock input

  ⇒ macro is considered a sequential element

OUTLINE

Synthesis flow

Design environment

Constraints

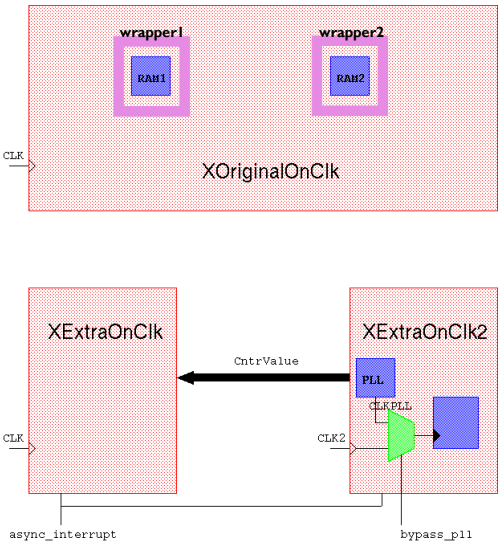
- NanoSoc lab: system analysis and constraint strategy

Clock gate insertion

DFT: scan insertion, RAM wrappers

- NanoSoc lab: CG and DFT insertion

NANOSOC - ANALYSIS



NANOSOC - ANALYSIS

Clocks:
- CLK, CLK2, CLKPLL
- Logically exclusive clocks: CLKPLL/CLK2 (common clockdomain)
- Only clock crossing datatransfer from CLKPLL/CLK2 to CLK (CntrValue)

Macro's:
- 2 RAM's in XOriginalOnClk
- PLL in XExtraOnClk2

DFT in RTL:
- RAM wrappers driven by pad_test
- Synchronous reset bypasses driven by pad_test
- Clock mux driven by pad_bypass_pll selects CLKPLL in functional mode and CLK2 in test mode
- Scan chain driven by pad_scan_enable

NANOSOC - ANALYSIS

Functional mode: *(pad_test = '0', pad_bypass_pll = '0', pad_scan_enable = '0')*

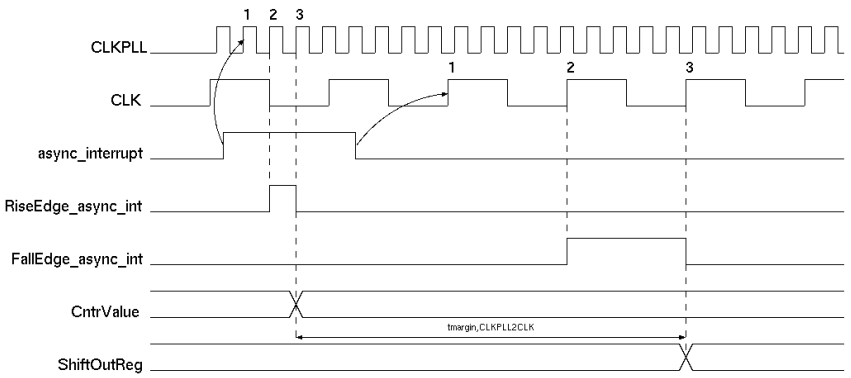
- XExtraOnClk2 runs completely on CLKPLL
- XOriginalOnClk and XExtraOnClk run on CLK
- Synchronisation of pad_reset in CLK domain, pad_reset2 in CLKPLL domain, pad_interrupt in both CLK and CLKPLL domains

Test mode: *(pad_test = '1', pad_bypass_pll = '1')*

- XExtraOnClk2 runs completely on CLK2
- XOriginalOnClk and XExtraOnClk run on CLK
- CLK2 = CLK apart from skew
- RAM wrappers activated
- Synchronised reset circuits bypassed by external resets
- No asynchronous inputs, also CntrValue is now a synchronous signal!

NANOSOC - ANALYSIS

- CntrValue crossing clock domains is controlled by polling on async_interrupt:



Timing of 'CntrValue' crossing clock domains

NANOSOC - CONSTRAINT STRATEGY

Define constraint settings per operation mode: functional mode, test mode

Clock timing constraints:

- Define clock characteristics with *create_clock, set_clock_latency, set_clock_uncertainty*
  - CLK @ 100MHz in functional mode, @ 20MHz in test mode, net latency:0.3ns, source latency:0.4ns, skew: 0.3ns
  - CLK2 @ 20MHz in test mode, irrelevant in functional mode , net latency:0.2ns, source latency:0.2ns, skew: 0.3ns
  - CLKPLL @ 240MHz in functional mode, irrelevant in test mode, net latency of 0, skew: 0.3ns
- Create virtual clock as reference for chip IO's
- Express clock relations with *set_clock_groups*

Clock crossing constraints:

- Set timing exception on signals crossing clock domains with *set_multicycle_path*: CntrValue

NANOSOC - CONSTRAINT STRATEGY

IO timing constraints:

- All synchronous inputs are going to CLK domain, CLK2 domain receives only either asynchronous inputs which get synchronised or operation mode inputs
- Constrain synchronous inputs with *set_input_delay*/*set_input_transition* w.r.t. virtual clock (not necessary for inputs going to black boxes like PLL nor for operation mode inputs like pad_test, pad_bypass_pll)
  - Input delays: 0.5ns
  - Input transitions: min 0.5ns, max 6ns
  - [all_inputs], [get_ports *]
- Constrain synchronous outputs with *set_output_delay*/*set_load* w.r.t. virtual clock (not mandatory for outputs coming directly from black box)
  - Output delays: 0.5ns
  - Output loads: 10pF
  - [all_outputs], [get_ports *]
- Set timing exception on synchronised inputs with *set_false_path*: pad_interrupt, pad_reset, pad_reset2 (in test mode these are also generated synchronously with test clock by test vehicle!)
- Fix state of test mode inputs with *set_case_analysis*: pad_test, pad_bypass_pll, pad_scan_enable

NANOSOC - EXERCISE

Preparation:

- cd ~/private
- cp -r ~group1/public/nanosoc . (subdir. synthesis1, hdl, libraries)
- cd nanosoc

Complete timing constraints file for functional operation according to previously defined specs:

- cd synthesis1/scripts
- nedit timingconstraints.tcl and replace "<<<>>>" with some valid arguments
- Command syntax can be checked inside dc_shell with 'man <cmd>' (see next to invoke dc_shell)

Run synthesis:

- cd ~/private/nanosoc/synthesis1/rundir
- source source_me
- dc_shell
- Copy content of scripts/main.tcl line by line in dc_shell

OUTLINE

Synthesis flow

Design environment

Constraints

- NanoSoc lab: system analysis and constraint strategy

Clock gate insertion

DFT: scan insertion, RAM wrappers

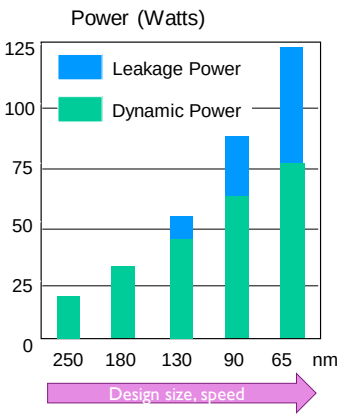
- NanoSoc lab: CG and DFT insertion

CLOCK GATING

Purpose: dynamic power saving on blocks/registers and clock tree by disrupting the access of the clock during inactivity

CLOCK GATING

Purpose: dynamic power saving on inactive blocks/registers and clock tree



CLOCK GATING

Purpose: dynamic power saving on inactive blocks/registers and clock tree

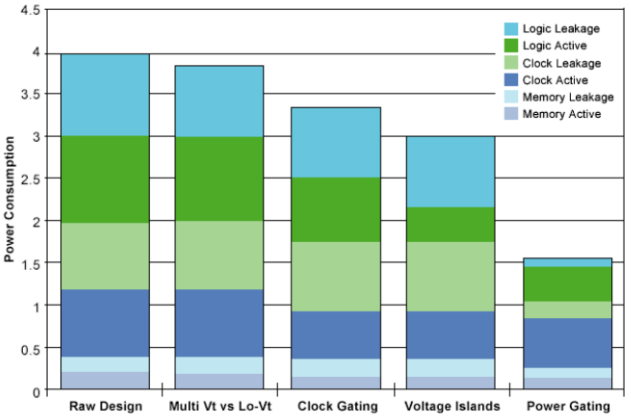
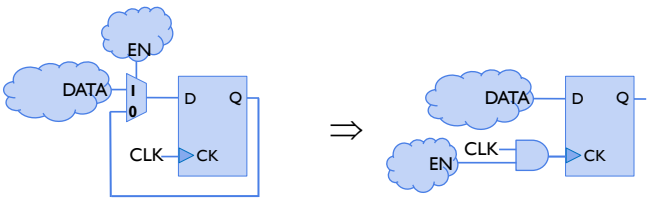


Figure 11. Power reduction techniques. Courtesy Chip Design magazine, 2007 (Ref. 4)

CLOCK GATING

Purpose: power saving on inactive blocks/registers and clock tree

How: move conditional statement of 'if' from datapath to clockpath



CLOCK GATING

Purpose: power saving on inactive blocks/registers and clock tree

Clock gate = gate with different inputs of which at least one input is a clock, the other inputs are considered as enable pins for that/those clock(s) by the synthesis tool

CLOCK GATING

Purpose: power saving on inactive blocks/registers and clock tree

Clock gate = gate with different inputs of which at least one input is a clock, the other inputs are considered as enable pins for that/those clock(s) by the synthesis tool

Can be introduced in design in RTL (mostly on architectural level) or during synthesis (register level)

CLOCK GATING

Purpose: power saving on inactive blocks/registers and clock tree

Clock gate = gate with different inputs of which at least one input is a clock, the other inputs are considered as enable pins for that/those clock(s) by the synthesis tool

Can be introduced in design in RTL (mostly on architectural level) or during synthesis (register level)

Styles:

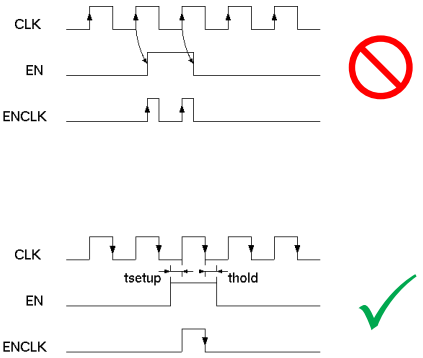
- Latch free clock gates
- Latch based clock gates

CLOCK GATING - CRITERIA

Clock gates are inserted when three conditions are fulfilled:

- Enable condition: mux should exist in register's data path that cannot be removed by optimization
- Setup condition: It checks that the enable signal comes from a register that is clocked by the same clock, and same edge as the register being gated. The enable signal may not be driven by a design input port either. Applies to latch-free clock gating only.
  - Design input constraint can be overridden by setting '*power_cg_derive_related_clock true*'
  - Both clock constraints can be overridden by setting '*power_cg_ignore_setup_condition true*' or by setting '*set_clock_gating_registers -include*'
- Width condition: specified by '*set_clock_gating_style -minimum_bitwidth*'

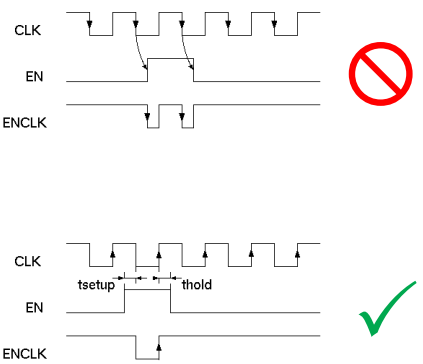
CLOCK GATING - LATCH FREE CG



EN event may not cause any active ENCLK event:

set_clock_gating_check -setup <tsetup> -hold <thold> <instance_name> (-high)
set_clock_gating_style -setup <tsetup> -hold <thold>

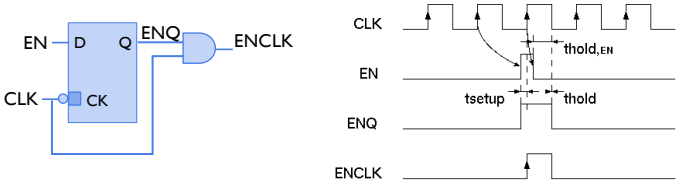
CLOCK GATING - LATCH FREE CG



EN event may not cause any active ENCLK event:

set_clock_gating_check -setup <tsetup> -hold <thold> <instance_name> (-low)
set_clock_gating_style -setup <tsetup> -hold <thold>

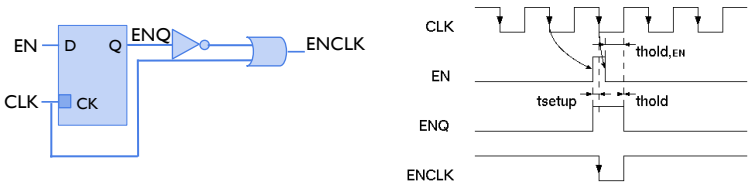
CLOCK GATING - LATCH BASED CG



EN event may not cause any active ENCLK event:

- EN is allowed to change during both states of CLK
- thold correct by construction: ENQ always later than CLK
- Window compared to latch-free CG: full period of CLK available for change of EN

CLOCK GATING - LATCH BASED CG



EN event may not cause any active ENCLK event:

- EN is allowed to change during both states of CLK
- thold correct by construction: ENQ always later than CLK
- Window compared to latch-free CG: full period of CLK available for change of EN

CLOCK GATING - LATCH FREE VS LATCH BASED

Styles:

- Latch free clock gates:
  - clipping or glitching may occur: timing should be verified and controlled
  - commonly used in RTL code to save power on block level
- Latch based clock gates:
  - hold timing is correct by construction
  - mostly added during synthesis to save power on register level
  - also latch-free CG's in RTL code can be converted to latch based style
  - Integrated CG's or composed CG's

CLOCK GATING - INTEGRATED CG

Advantages:

- More compact than composed latch-based CG's
- No skew between latch clock and gate clock
- Setup and hold times defined in cell description
- Powerreduction

CLOCK GATING - INTEGRATED CG

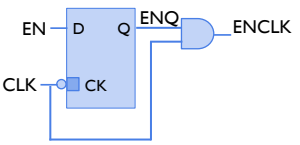
Types of ICG's:

- latch_posedge
- latch_negedge
- latch_posedge_precontrol
- latch_negedge_precontrol
- latch_posedge_postcontrol
- latch_negedge_postcontrol
- latch_posedge_precontrol_obs
- latch_negedge_precontrol_obs
- latch_posedge_postcontrol_obs
- latch_negedge_postcontrol_obs

CLOCK GATING - INTEGRATED CG

Types of ICG's:

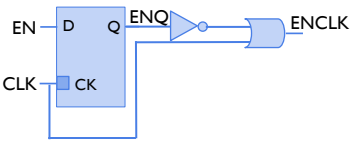
- latch_posedge
- latch_negedge
- latch_posedge_precontrol
- latch_negedge_precontrol
- latch_posedge_postcontrol
- latch_negedge_postcontrol
- latch_posedge_precontrol_obs
- latch_negedge_precontrol_obs
- latch_posedge_postcontrol_obs
- latch_negedge_postcontrol_obs



CLOCK GATING - INTEGRATED CG

Types of ICG's:

- latch_posedge
- latch_negedge
- latch_posedge_precontrol
- latch_negedge_precontrol
- latch_posedge_postcontrol
- latch_negedge_postcontrol
- latch_posedge_precontrol_obs
- latch_negedge_precontrol_obs
- latch_posedge_postcontrol_obs
- latch_negedge_postcontrol_obs



CLOCK GATING - CG OPERATIONS

Check available ICG's in library through attribute:

get_lib_cells fsa0a_c_generic_core_ss0p9v125c/ -filter "@clock_gating_integrated_cell != nil"*

- Finds ICG's in library

get_lib_attribute [get_object [get_lib_cells fsa0a_c_generic_core_ss0p9v125c/ -filter \ "@clock_gating_integrated_cell != nil"]] clock_gating_integrated_cell*

- Finds ICG's types in library

To check timing on coded latch-free CG's during STA:

*set_clock_gating_check -setup 0.05 -hold 0.04 [get_cells -hier *clockgate*]*

Switch coded latch-free CG's to ICG's:

set_clock_gating_style -minimum_bitwidth 1 -positive_edge_logic integrated:GCKESX1 \ -negative_edge_logic integrated:GCBESX1

- Sets ICG type to switch to

replace_clock_gates -global

- To switch to ICG's

CLOCK GATING - CG OPERATIONS

Automatic insertion of ICG's:

set_clock_gating_style -minimum_bitwidth 1 -positive_edge_logic integrated:GCKESX1 \ -negative_edge_logic integrated:GCBESX1

- Sets ICG type to insert

compile(_ultra) -clock_gate

- To insert CG's during mapping

OUTLINE

Synthesis flow

Design environment

Constraints

- NanoSoc lab: system analysis and constraint strategy

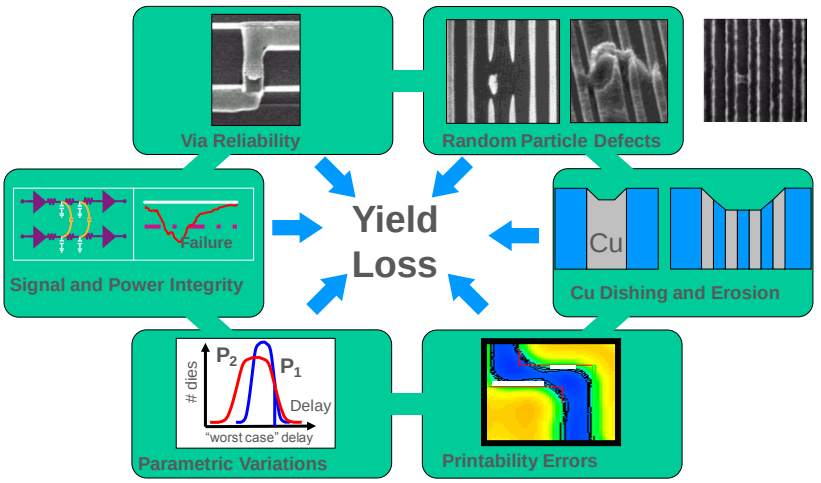
Clock gate insertion

DFT: scan insertion, RAM wrappers

- NanoSoc lab: CG and DFT insertion

DFT - TESTING OF CHIPS

During manufacturing, defects may occur on a chip due to impurities, process effects, ... ⇒ Testing required to detect faulty chips.



DFT - TESTING OF CHIPS

During manufacturing, defects may occur on a chip due to impurities
⇒ Testing required to detect faulty chips.

Test methodologies over the years

- Early days testing by means of functional test vectors

DFT - TESTING OF CHIPS

During manufacturing, defects may occur on a chip due to impurities
⇒ Testing required to detect faulty chips.

Test methodologies over the years

- Early days testing by means of functional test vectors
- Applying dedicated test vectors optimized for failure detection through scan chains added by synthesis (test vectors generated automatically by ATPG tool)

DFT - TESTING OF CHIPS

During manufacturing, defects may occur on a chip due to impurities
⇒ Testing required to detect faulty chips.

Test methodologies over the years

- Early days testing by means of functional test vectors
- Applying dedicated test vectors optimised for failure detection through scan chains added by synthesis (test vectors generated automatically by ATPG tool)
- BIST: test circuitry added in RTL to generate test vectors internally

DFT - TESTING OF CHIPS

During manufacturing, defects may occur on a chip due to impurities
⇒ Testing required to detect faulty chips.

Test methodologies over the years

- Early days testing by means of functional test vectors
- Applying dedicated test vectors optimized for failure detection through scan chains added by synthesis (test vectors generated automatically by ATPG tool)
- BIST: test circuitry added in RTL to generate test vectors internally

DFT = Design for Test : adding specific circuitry to the design for test purpose only

DFT - TESTING OF CHIPS

During manufacturing, defects may occur on a chip due to impurities
⇒ Testing required to detect faulty chips.

Test methodologies over the years

- Early days testing by means of functional test vectors
- Applying dedicated test vectors optimized for failure detection through scan chains added by synthesis (test vectors generated automatically by ATPG tool)
- BIST: test circuitry added in RTL to generate test vectors internally

DFT = Design for Test : adding specific circuitry to the design for test purpose only

DFT - SCAN CHAINS

Types of scan chains:

- LSSD scan style
- Clocked-scan scan style
- Multiplexed-FF scan style

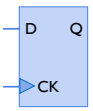
DFT - SCAN CHAINS

Types of scan chains:

- LSSD scan style
- Clocked-scan scan style
- Multiplexed-FF scan style

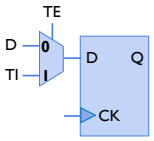
DFT - SCAN CHAINS

Multiplexed-FF scan style: scan cell



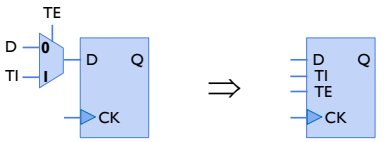
DFT - SCAN CHAINS

Multiplexed-FF scan style: scan cell



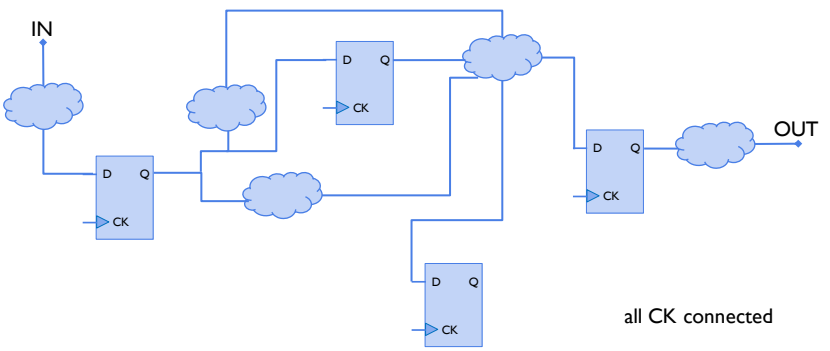
DFT - SCAN CHAINS

Multiplexed-FF scan style: scan cell



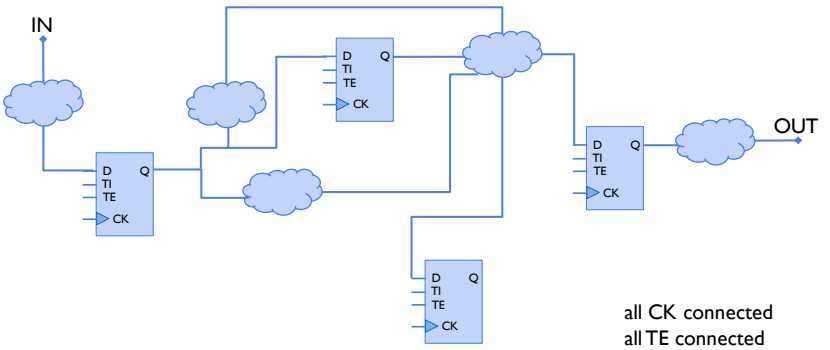
DFT - SCAN CHAINS

Multiplexed-FF scan style: initial circuit



DFT - SCAN CHAINS

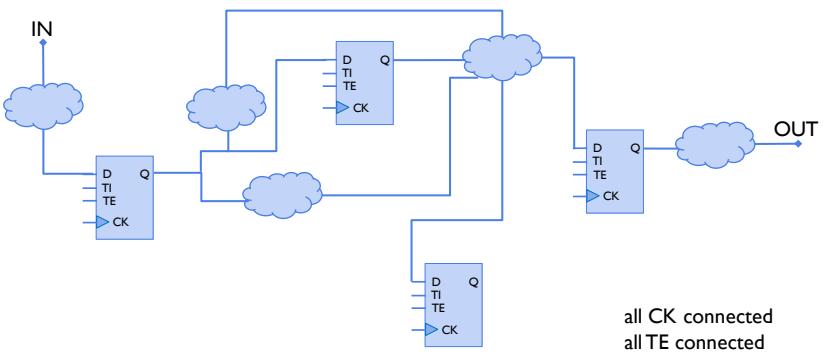
Multiplexed-FF scan style: circuit after scan cell replacement



DFT - SCAN CHAINS

Multiplexed-FF scan style: circuit after scan cell replacement

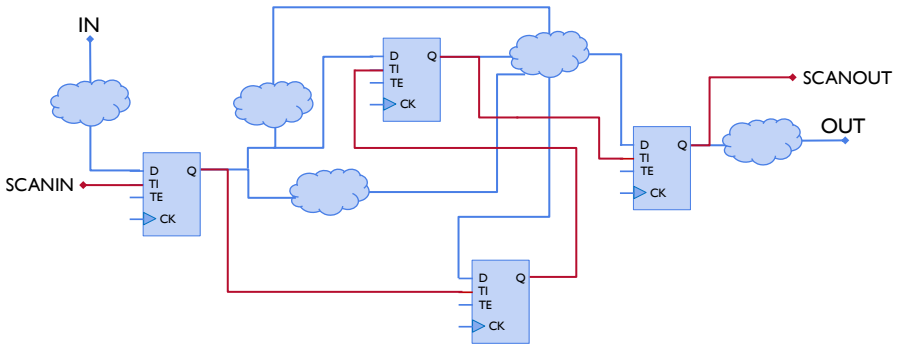
compile .... ***-scan***



DFT - SCAN CHAINS

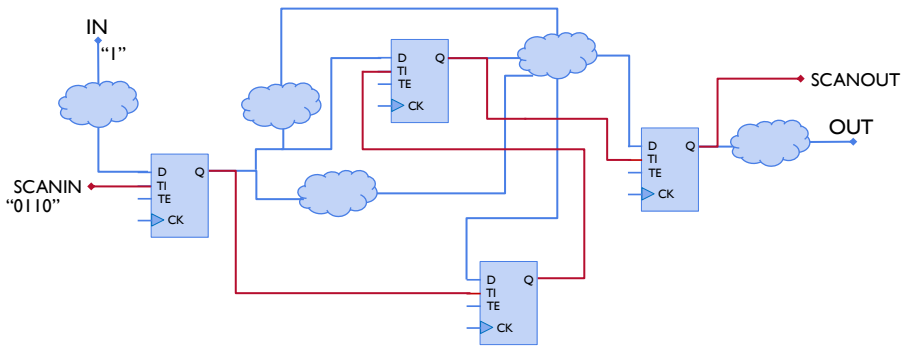
Multiplexed-FF scan style: circuit after scan path insertion

insert_dft



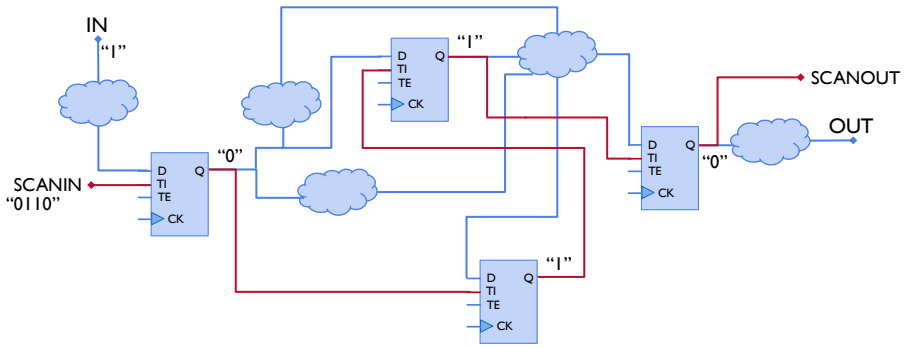
DFT - TEST PROTOCOL

Test protocol: shift-in operation (TE='1')



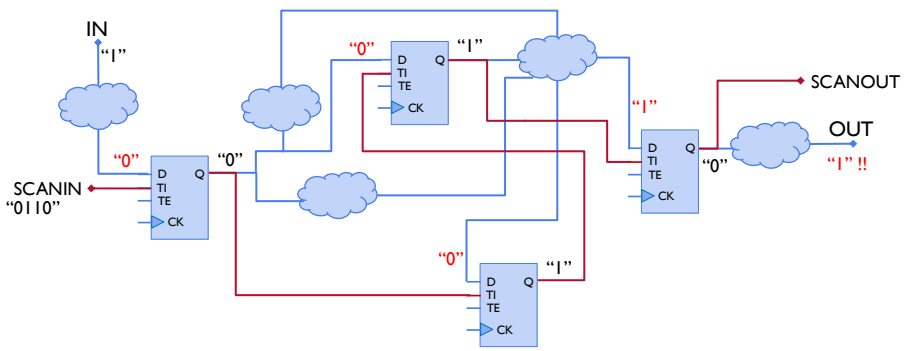
DFT - TEST PROTOCOL

Test protocol : shift-in operation after 4 clock pulses



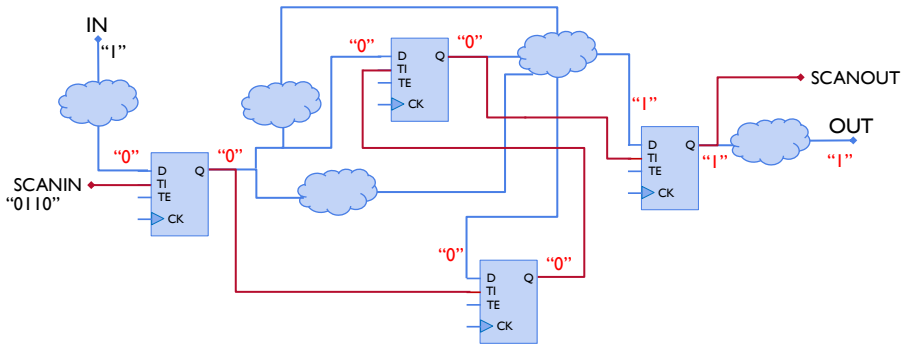
DFT - TEST PROTOCOL

Test protocol: after propagation through combinatorial logic



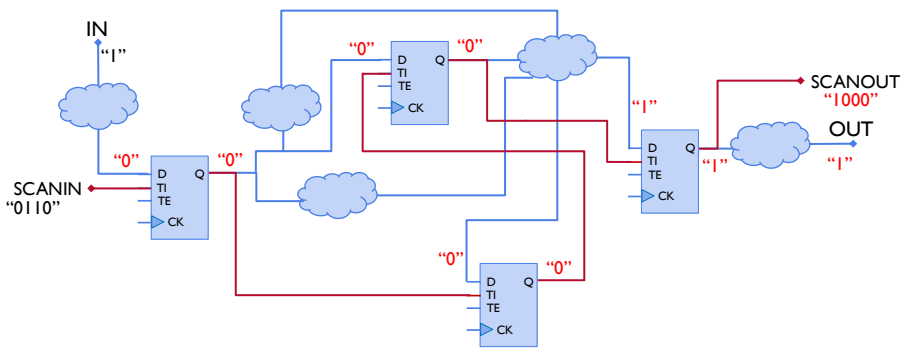
DFT - TEST PROTOCOL

Test protocol: capture operation (TE='0')



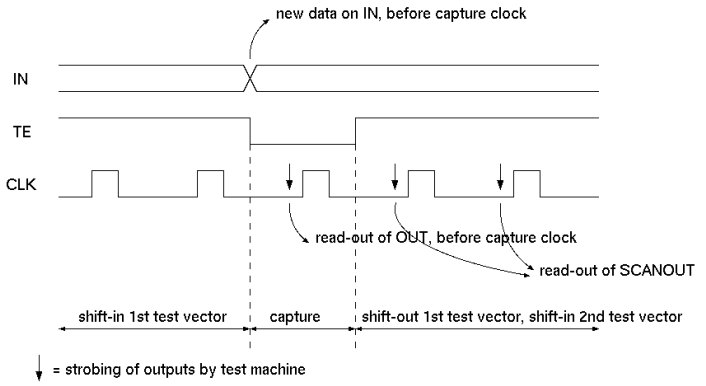
DFT - TEST PROTOCOL

Test protocol: shift-out operation (TE='1') after 4 clock pulses



DFT - TEST PROTOCOL

Test protocol: timing diagram

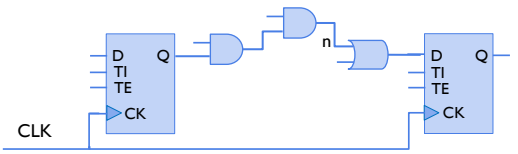


DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.

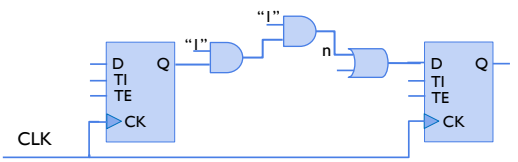
DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.



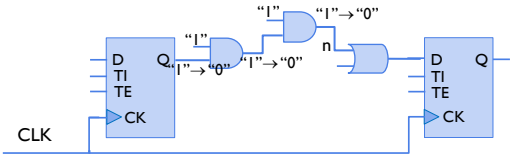
DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.



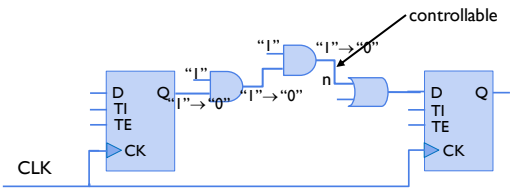
DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.



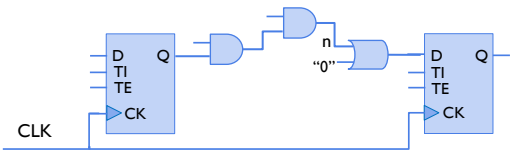
DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.



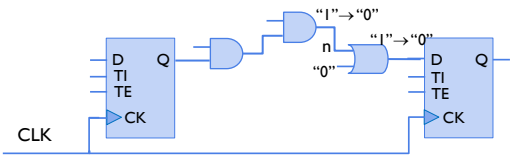
DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.



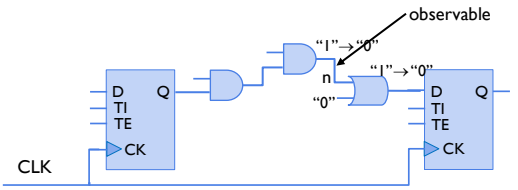
DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.



DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.



DFT - QUALITY OF TEST VECTORS

Scan paths provide means to **control** and **observe** the different logic nodes on the chip. When a node can be controlled and observed with a set of test bits it is called **detectable**. ATPG tools generate these bits for all nodes on the chip resulting in a set of **test vectors**.

Fault coverage = ratio of detectable over total amount of logic nodes. It expresses:

- Quality of test vectors
- Testability of design → can be improved with DFT

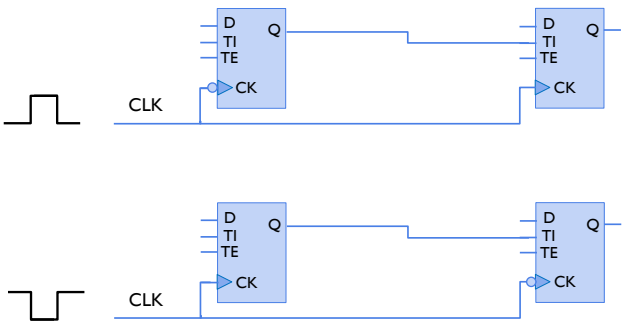
Test coverage: when untestable nodes are omitted from total amount of nodes. It expresses:

- Quality of test vectors

DFT - MIXING CLOCK EDGES

Problem: 2 successive FF's with different clock polarity results in data flowing through 2 FF's in one clock cycle (min timing violation)

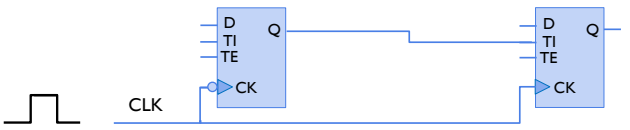
Solution: clock pulse polarity determines the order of falling edge and rising edge FF's chosen by DC



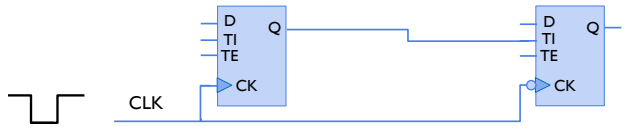
DFT - MIXING CLOCK EDGES

Problem: 2 successive FF's with different clock polarity results in data flowing through 2 FF's in one clock cycle (min timing violation)

Solution: clock pulse polarity determines the order of falling edge and rising edge FF's chosen by DC



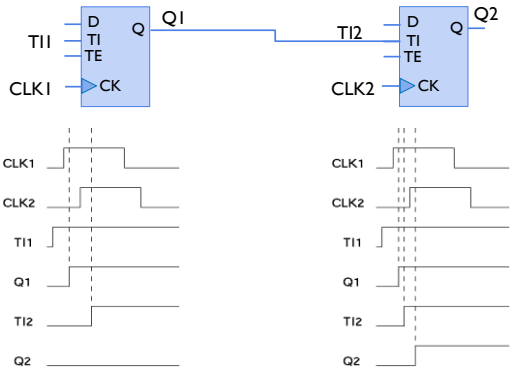
*set_dft_signal -type ScanClock -port CLK -timing [list **45 55**] -view existing_dft*



*set_dft_signal -type ScanClock -port CLK -timing [list **55 45**] -view existing_dft*

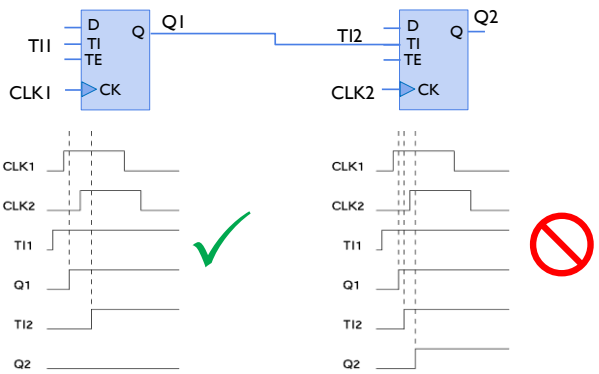
DFT - MIXING CLOCKS

Problem: the slightest unintentional skew on CLK2 could lead to flow-through from TI1 to Q2 in one clock cycle (min timing violation) depending on length of Q1-TI2 delay → unpredictable!



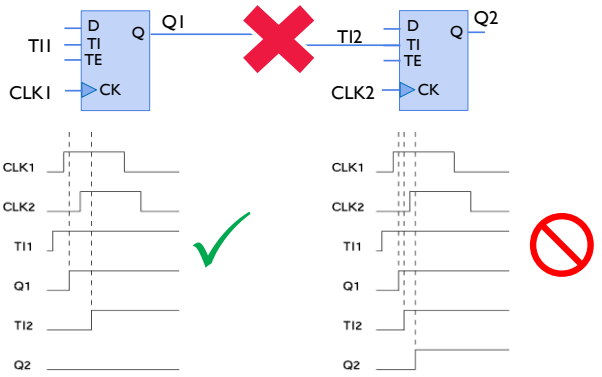
DFT - MIXING CLOCKS

Problem: the slightest unintentional skew on CLK2 could lead to flow-through from TI1 to Q2 in one clock cycle (min timing violation) depending on length of Q1-TI2 delay → unpredictable!



DFT - MIXING CLOCKS

Problem: the slightest unintentional skew on CLK2 could lead to flow-through from TI1 to Q2 in one clock cycle (min timing violation) depending on length of Q1-TI2 delay → unpredictable!



DFT - MIXING CLOCKS

Solutions:

- use a common test clock to test circuit
- add lockup-latch in between FF's belonging to different clockdomains
- define appropriate clock schedule
- split scan chain in 2 separate chains

DFT - MIXING CLOCKS

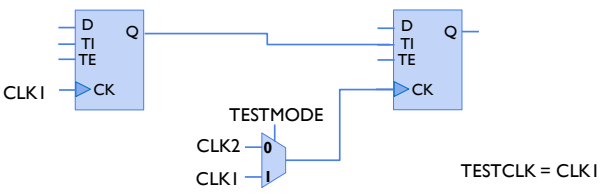
Solutions:

- use a common test clock to test circuit
- add lockup-latch in between FF's belonging to different clockdomains
- define appropriate clock schedule
- split scan chain in 2 separate chains

DFT - MIXING CLOCKS

Solution:

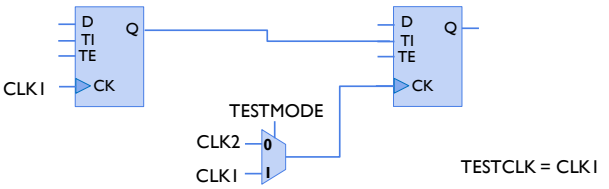
- use a common test clock to test circuit ⇒ define test mode in which you pass same clock to all FF's



DFT - MIXING CLOCKS

Solution:

- use a common test clock to test circuit ⇒ define test mode in which you pass same clock to all FF's



2 modes of operation: normal mode, test mode (single-mode optimization, multi-mode STA):

- *set_case_analysis 0/1 TESTMODE*
- Specify constraints per mode, i.e. test clock's frequency
- Disadvantage: min timing closure may require balancing of all clock domains or local min time fixing of clock crossing signals

DFT - MIXING CLOCKS

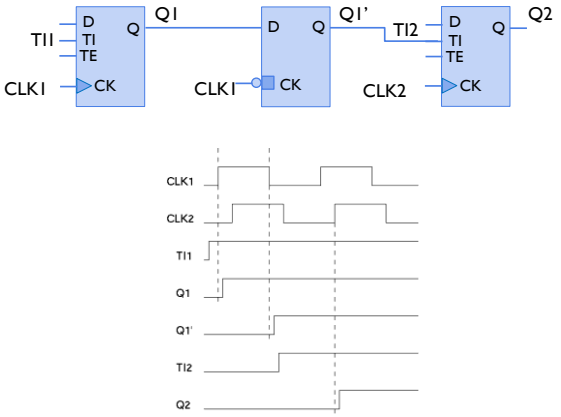
Solutions:

- use a common test clock to test circuit
- add lockup-latch in between FF's belonging to different clockdomains
- define appropriate clock schedule
- split scan chain in 2 separate chains

DFT - MIXING CLOCKS

Solution:

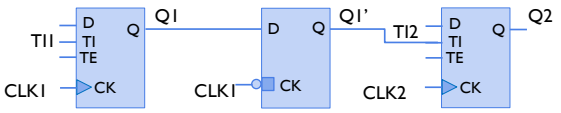
- Automatic insertion of lockup-latch in scan path is resolving shift cycle



DFT - MIXING CLOCKS

Solution:

- Automatic insertion of lockup-latch in scan path is resolving shift cycle



- Disadvantage: problem remains in capture cycle when functional paths are enabled
  ⇒ clocks still need to be scheduled appropriately according to direction of 'clock crossing' data (receiving clock first, sending clock last)

DFT - MIXING CLOCKS

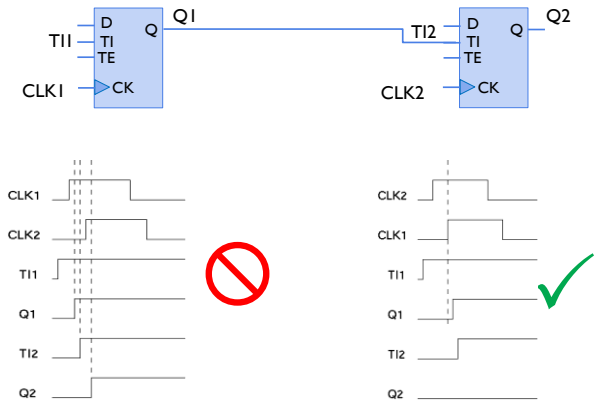
Solutions:

- use a common test clock to test circuit
- add lockup-latch in between FF's belonging to different clockdomains
- define appropriate clock schedule
- split scan chain in 2 separate chains

DFT - MIXING CLOCKS

Solution:

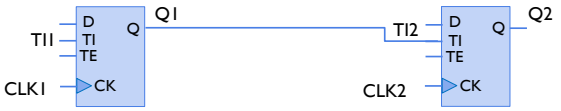
- In case of 2 external clocks, clock order can be controlled: order should reflect direction of 'clock crossing' data



DFT - MIXING CLOCKS

Solution:

- In case of 2 external clocks, clock order can be controlled: order should reflect direction of 'clock crossing' data

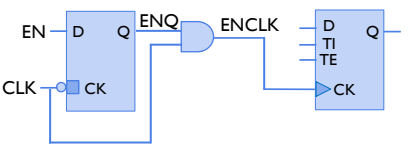


- Choose the appropriate clock order according to capture cycle, scan chain will adapt order for both capture and shift cycle
- Disadvantage: works only if data crosses in one direction in capture cycle

DFT - CLOCK GATES

Problem: When CG's are inserted in the design, clocking of circuitry is dependent on logic driving EN. However, during test mode, all FF's in the scan chains should shift, not to interrupt the chain.

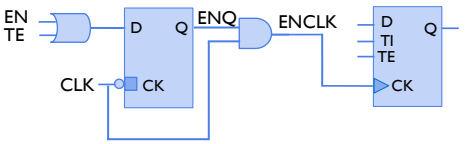
Solution: Add controllability on CG to control clocks during shift operation only, not during capturing, since during capturing, logic path should be enabled in order to test it.



DFT - CLOCK GATES

Problem: When CG's are inserted in the design, clocking of circuitry is dependent on logic driving EN. However, during test mode, all FF's in the scan chains should shift, not to interrupt the chain.

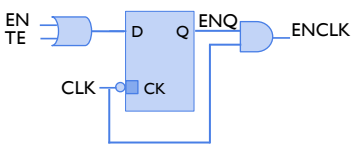
Solution: Add controllability on CG to control clocks during shift operation only, not during capturing, since during capturing, logic path should be enabled in order to test it. ⇒ Can be obtained with TE (scan enable).



DFT - CLOCK GATES

Problem: When CG's are inserted in the design, clocking of circuitry is dependent on logic driving EN. However, during test mode, all FF's in the scan chains should shift, not to interrupt the chain.

Solution: Add controllability on CG to control clocks during shift operation only, not during capturing, since during capturing, logic path should be enabled in order to test it. ⇒ Can be obtained with TE (scan enable).



set_clock_gating_style ..... -control_point before -control_signal scan_enable

DFT - INTEGRATED CG

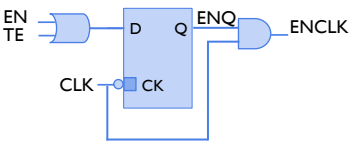
Types of ICG's:

- latch_posedge
- latch_negedge
- latch_posedge_precontrol
- latch_negedge_precontrol
- latch_posedge_postcontrol
- latch_negedge_postcontrol
- latch_posedge_precontrol_obs
- latch_negedge_precontrol_obs
- latch_posedge_postcontrol_obs
- latch_negedge_postcontrol_obs

DFT - INTEGRATED CG

Types of ICG's:

- latch_posedge
- latch_negedge
- latch_posedge_precontrol
- latch_negedge_precontrol
- latch_posedge_postcontrol
- latch_negedge_postcontrol
- latch_posedge_precontrol_obs
- latch_negedge_precontrol_obs
- latch_posedge_postcontrol_obs
- latch_negedge_postcontrol_obs

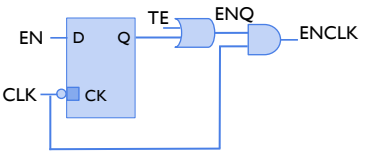


set_clock_gating_style ..... -control_point before -control_signal scan_enable

DFT - INTEGRATED CG

Types of ICG's:

- latch_posedge
- latch_negedge
- latch_posedge_precontrol
- latch_negedge_precontrol
- latch_posedge_postcontrol
- latch_negedge_postcontrol
- latch_posedge_precontrol_obs
- latch_negedge_precontrol_obs
- latch_posedge_postcontrol_obs
- latch_negedge_postcontrol_obs

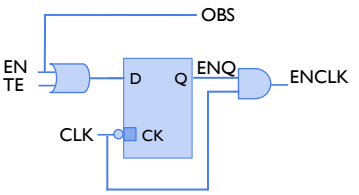


set_clock_gating_style ..... -control_point after -control_signal scan_enable

DFT - INTEGRATED CG

Types of ICG's:

- latch_posedge
- latch_negedge
- latch_posedge_precontrol
- latch_negedge_precontrol
- latch_posedge_postcontrol
- latch_negedge_postcontrol
- latch_posedge_precontrol_obs
- latch_negedge_precontrol_obs
- latch_posedge_postcontrol_obs
- latch_negedge_postcontrol_obs

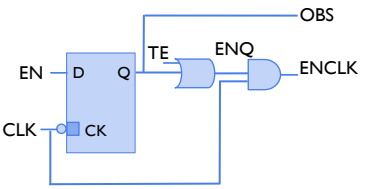


set_clock_gating_style ..... -control_point before -observation_point true \ -control_signal test_mode

DFT - INTEGRATED CG

Types of ICG's:

- latch_posedge
- latch_negedge
- latch_posedge_precontrol
- latch_negedge_precontrol
- latch_posedge_postcontrol
- latch_negedge_postcontrol
- latch_posedge_precontrol_obs
- latch_negedge_precontrol_obs
- latch_posedge_postcontrol_obs
- latch_negedge_postcontrol_obs

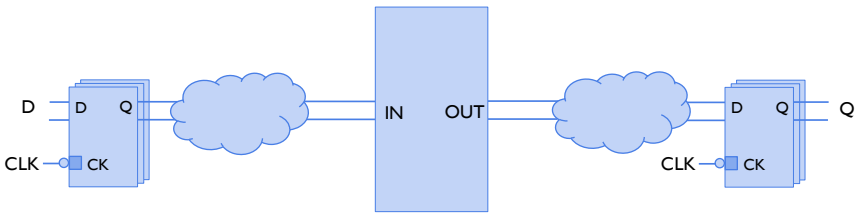


set_clock_gating_style ..... -control_point after -observation_point true \ -control_signal test_mode

DFT - MACRO

Problem: functional description is mostly missing in macro lib (macro is blackbox)

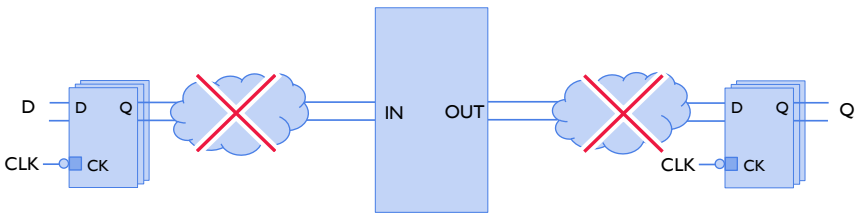
- Logic in front of macro inputs is not observable
- Logic behind macro outputs is not controllable



DFT - MACRO

Problem: functional description is mostly missing in macro lib (macro is blackbox)

- Logic in front of macro inputs is not observable
- Logic behind macro outputs is not controllable

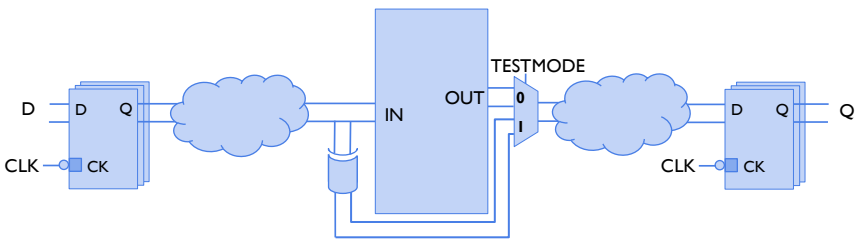


DFT - MACRO

Problem: functional description is mostly missing in macro lib (macro is blackbox)

- Logic in front of macro inputs is not observable
- Logic behind macro outputs is not controllable

Solution: increase testability of design with test wrapper



DFT - SCAN PATH SPECS

Test protocol:

- Strobe before clock
- Strobe after clock

Test frequency: generally lower than operating frequency to prevent damage due to high power consumption (much more nodes toggle thanks to high efficient test vectors)

New ports or existing ports can be defined as scan input/output.

PO used as scan output: output mux is placed in front of padcell

Bidirectional ports used as scan input/output: control logic is placed around padcell.

DFT - SCAN PATH SPECS

Test protocol:

- *set test_default_delay 0* : defines when inputs (PI and scan inputs) are applied in capture cycle (before strobe and clock edge)
- *set test_default_bidir_delay 0* : same but for bidirectional paths + specifies when output path is released
- *set test_default_strobe 20* : defines when output (PO and scan outputs) is read by tester
- *set test_default_period 50* : defines cycle time for both shifting and capturing

Scan style:

- *set test_default_scan_style multiplexed_flip_flop* : defines scan style (default)

Scan ports:

- *set_dft_signal -type* ***Reset*** *-active_state 0 -port [get_port ARST] -view existing_dft*
- *set_dft_signal -type* ***ScanEnable*** *-port [get_port CS] -active_state 1 -hookup_pin [get_pin I_CS/O] \ -view spec -usage {scan clock_gating}*
- *set_dft_signal -type* ***ScanDataIn*** *-port [get_port SDI] -view spec*
- *set_dft_signal -type* ***ScanDataOut*** *-port [get_port SDO] -view spec*
- *set_dft_signal -type* ***ScanClock*** *-port [get_port CLK] -timing [list 25 45] -view existing_dft*
- *set_dft_signal -type* ***ScanClock*** *-port [get_port SCK] -timing [list 45 25] -view existing_dft*
- *set_dft_signal -type* ***TestMode*** *-port [get_port TEST_ENABLE] -active_state 1 -view existing_dft*

DFT - SCAN PATH SPECS

Scan path definitions:

- *set_scan_configuration -replace false -style multiplexed_flip_flop -clock_mixing mix_edges -chain_count 1*
- *set_scan_configuration -style multiplexed_flip_flop -clock_mixing mix_clocks -chain_count 1*
- *set_scan_path ScanChain1 -view spec -scan_data_in [get_port SDI] -scan_data_out [get_port SDO] -scan_master_clock [get_port CLK]*

DFT - SCAN PATH SPECS

Creation of test protocol

- *create_test_protocol*: creation of test protocol based on scan specs (style, ports, timing) and verification of the specs' consistency

Verification of design against test protocol:

- *dft_drc [-pre_dft] [-coverage_estimate]*: verifies consistency of design against created test protocol

Scan chain preview:

- *preview_dft* - reports the effects of scan insertion based on the scan specs without implementing scan chain

Scan chain insertion:

- *insert_dft*

OUTLINE

Synthesis flow

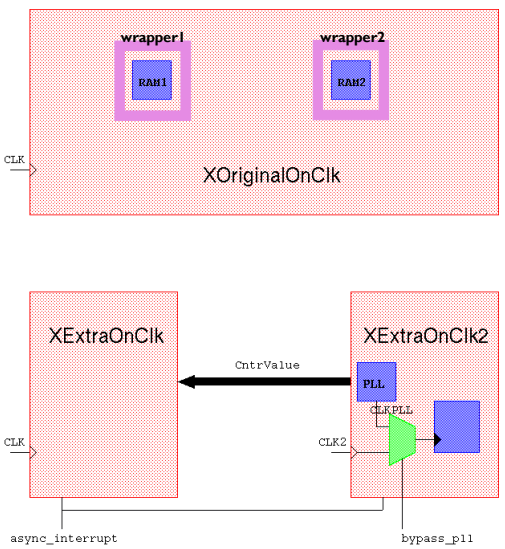
Design environment

Constraints
- NanoSoc lab: system analysis and constraint strategy

Clock gate insertion

DFT: scan insertion, RAM wrappers
- NanoSoc lab: CG and DFT insertion

NANOSOC - ANALYSIS



NANOSOC - ANALYSIS

Clocks:
- CLK, CLK2, CLKPLL
- Logically exclusive clocks: CLKPLL/CLK2 (common clockdomain)
- Only clock crossing datatransfer from CLKPLL/CLK2 to CLK (CntrValue)

Macro's
- 2 RAM's in XOriginalOnClk
- No functional description of RAM in .lib: RAM's are black boxes
- PLL in XExtraOnClk2

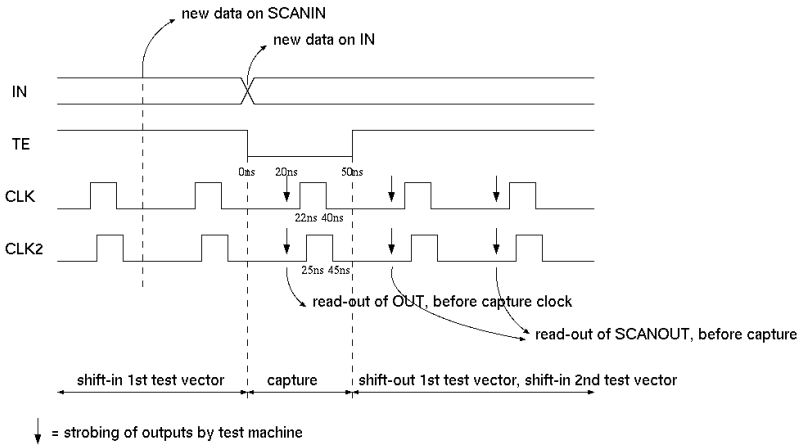
DFT in RTL:
- Test wrappers: driven by pad_test added to improve logic's testability
- Synchronous reset bypasses: driven by pad_test
- Clock mux: driven by pad_bypass_pll selects CLKPLL in functional mode and CLK2 in test mode
- Scan chain: driven by pad_scan_enable

NANOSOC - ANALYSIS

Functional mode: *(pad_test = '0', pad_bypass_pll = '0', pad_scan_enable = '0')*
- XExtraOnClk2 runs completely on CLKPLL
- XOriginalOnClk and XExtraOnClk run on CLK
- Synchronisation of pad_reset in CLK domain, pad_reset2 in CLKPLL domain, pad_interrupt in both CLK and CLKPLL domains

Test mode: *(pad_test = '1', pad_bypass_pll = '1')*
- XExtraOnClk2 runs completely on CLK2
- XOriginalOnClk and XExtraOnClk run on CLK
- CLK2 = CLK apart from skew: *set_clock_latency -source*
- RAM wrappers activated
- Synchronised reset circuits bypassed by external resets
- No asynchronous inputs, also CntrValue is now a synchronous signal that should propagate in 1 cycle!

NANOSOC - TEST STRATEGY



NANOSOC - TEST STRATEGY

1 scan chain with mixed clocks (CLK and CLK2)

no lock-up latch required since timing relation of clocks chosen such that no min timing violations are possible for data crossing (CntrValue), both in shift and capture cycle

Scan input: existing port pad_prom_din

Scan output: existing port pad_prom_oe

NANOSOC - EXERCISE

Preparation:

- cd ~/private/nanosoc
- cp -r ~group1/public/nanosoc/synthesis2 .

Complete scan insertion file according to previously defined strategy:

- cd synthesis2/scripts
- nedit insertscan.tcl and replace "<<<>>>" with some valid arguments
- Command syntax can be checked inside dc_shell with 'man <cmd>' (see next to invoke dc_shell)

Run synthesis:

- cd ~/private/nanosoc/synthesis2/rundir
- source source_me
- dc_shell
- Copy content of scripts/main.tcl line by line in dc_shell

NANOSOC - CONSTRAINTS TEST MODE

Clock timing constraints:

- Clock characteristics:
  - CLK @ 20MHz in test mode, rising @ 20ns, falling @ 38ns
  - CLK2 @ 20MHz in test mode, rising @ 22ns, falling @ 40ns
  - CLKPLL irrelevant in test mode

Clock crossing constraints:

- CntrValue can now also be considered as a synchronous data path since clock relation of 2 concerning clocks is tailored to allow synchronous operation

NANOSOC - CONSTRAINTS TEST MODE

IO timing constraints:

- All inputs and outputs are now synchronous signals (test machine)
  - VIRTUAL_CLK is reference for inputs, 20MHz rising @ 0ns, falling @ 22.5ns
  - Input delays: 0.5ns w.r.t. rising edge of VIRTUAL_CLK
  - VIRTUAL_CLK_STROBE is reference for outputs, 20MHz rising @ 18ns, falling @ 38ns
  - Output delays: 0.5ns w.r.t. rising edge of VIRTUAL_CLK_STROBE
- Fix state of test mode inputs with *set_case_analysis*: pad_test, pad_bypass_pll, pad_scan_enable