Very Large Scale Integration II - VLSI II

Synthesis Flow &

Static Timing Analysis

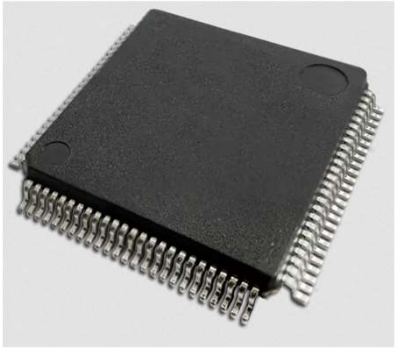
T. A. Fırat Kula

ITU VLSI Laboratories

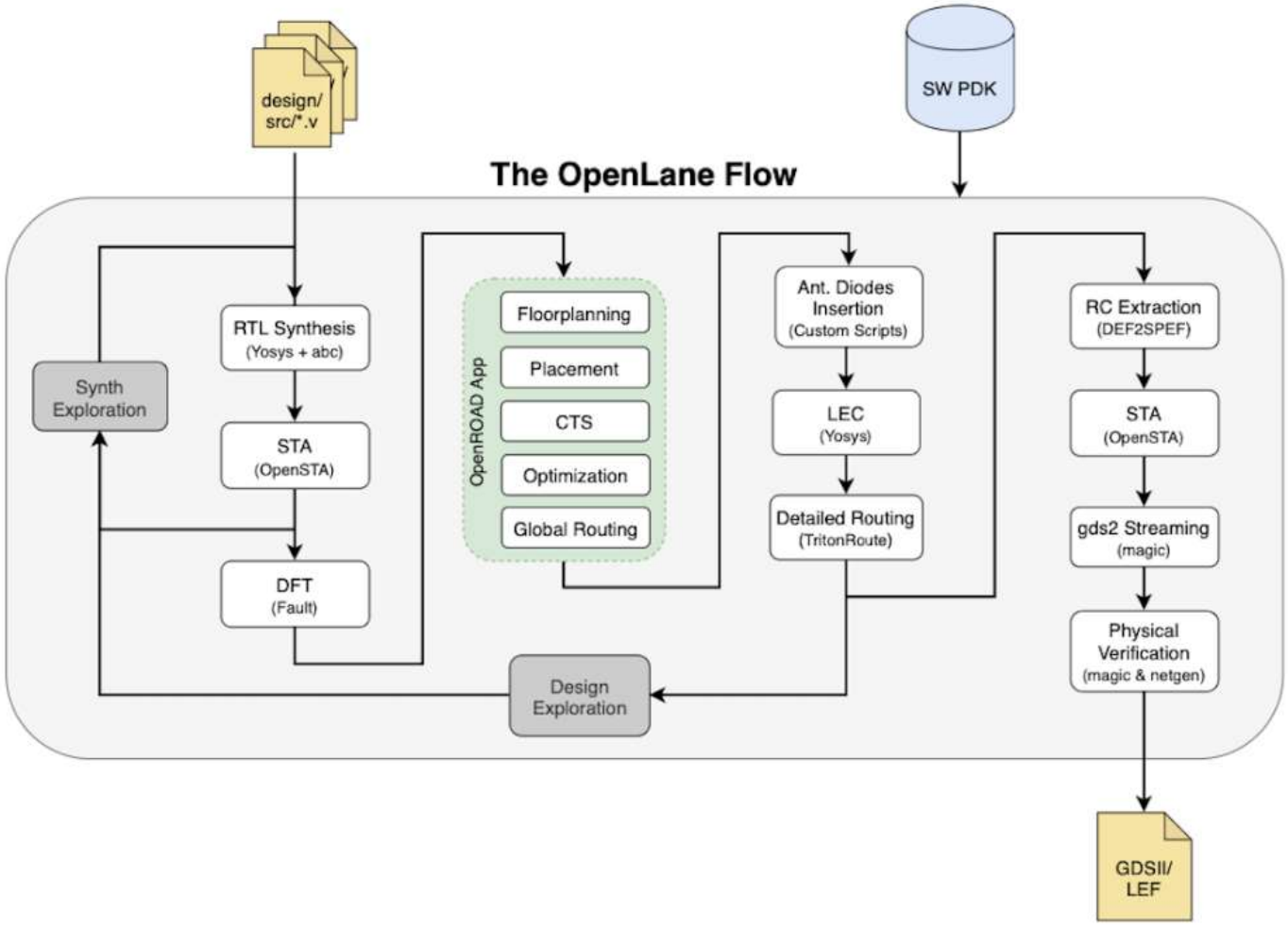
Istanbul Technical University

Digital Chip Design Flow

- A digital chip design flow includes the processes required to create a successfully working chip designed with a specific process library. The RTL (Register-Transfer Level) level design inputs are converted to ready-to-be manufactured GDS (Graphical Design System) files, in the end.
- OpenLane is a set of scripts to provide users an easy to use standard chip flow that integrates a multitude of different open source chip design tools.

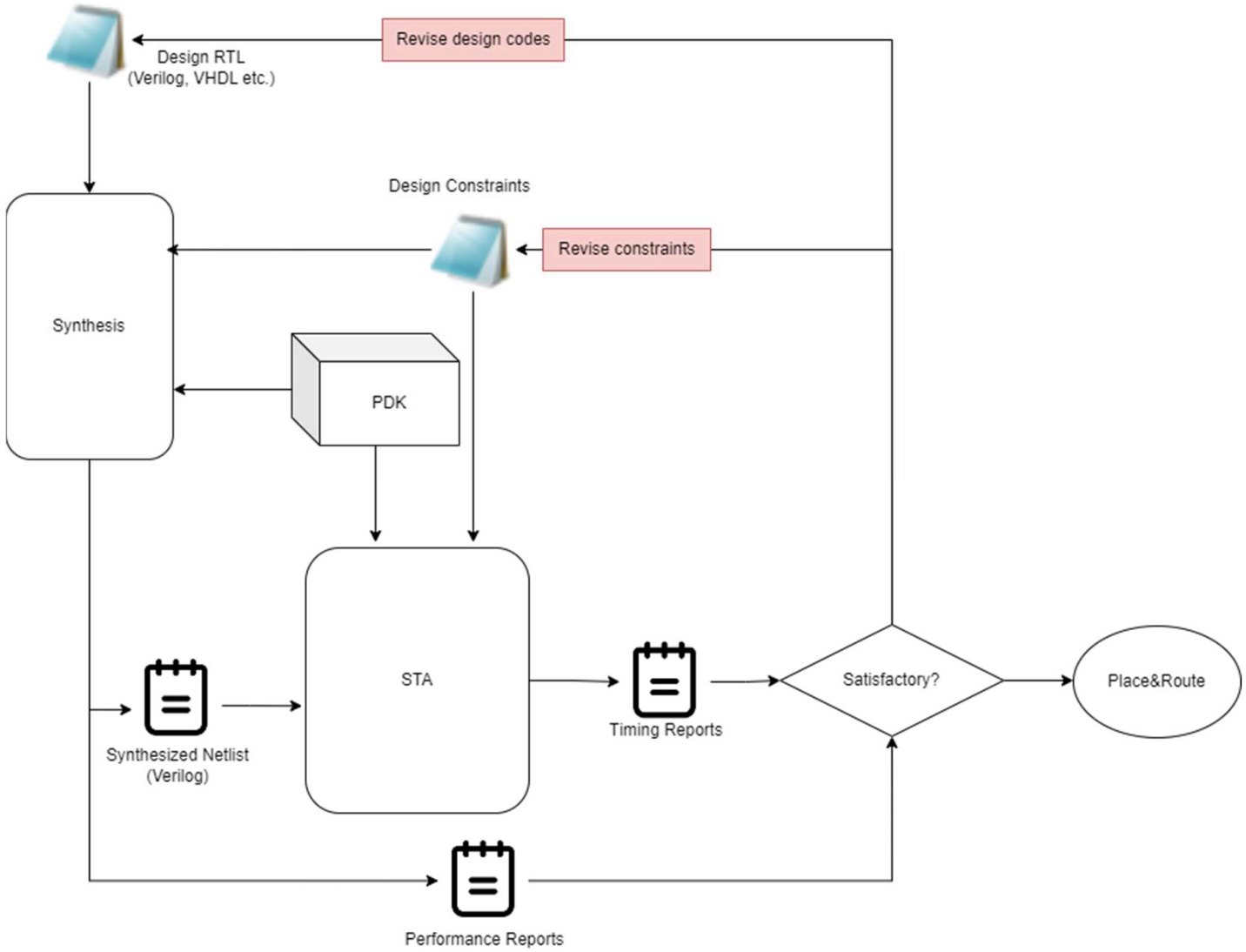


OpenLane Flow



Generic Synthesis Flow

- PDK contains a set of libraries that holds information regarding the cells (building blocks) such as area, power consumption, footprint, geometry, design rules…
- If the results are not satisfactory, we should go back to our design and make changes, or revise our constraints



Step 1: RTL Description

- Describe the circuit we want to design using Hardware Description Languages (HDL)
- Have a set of HDL codes (design RTL library) to be used as input to synthesis tool.
- Very commonly used HDLs: Verilog, SystemVerilog, VHDL

Step 1: RTL Description

- D… using
  H…)
- H… brary)
  b…

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity COMP_bit is
    Port (
    -- Inputs
    a,b: in std_logic;
    -- Outputs
    g: out std_logic; -- a>b: high if 'a' is greater than 'b'
    e: out std_logic; -- a=b: high if 'a' is equal to 'b'
    s: out std_logic  -- a<b: high if 'a' is smailler than 'b'
    );
end COMP_bit;

architecture gate_level_bitwise_comp of COMP_bit is
begin
    g <= a and not b;
    e <= a xnor b;
    s <= not a and b;
end gate_level_bitwise_comp;
```

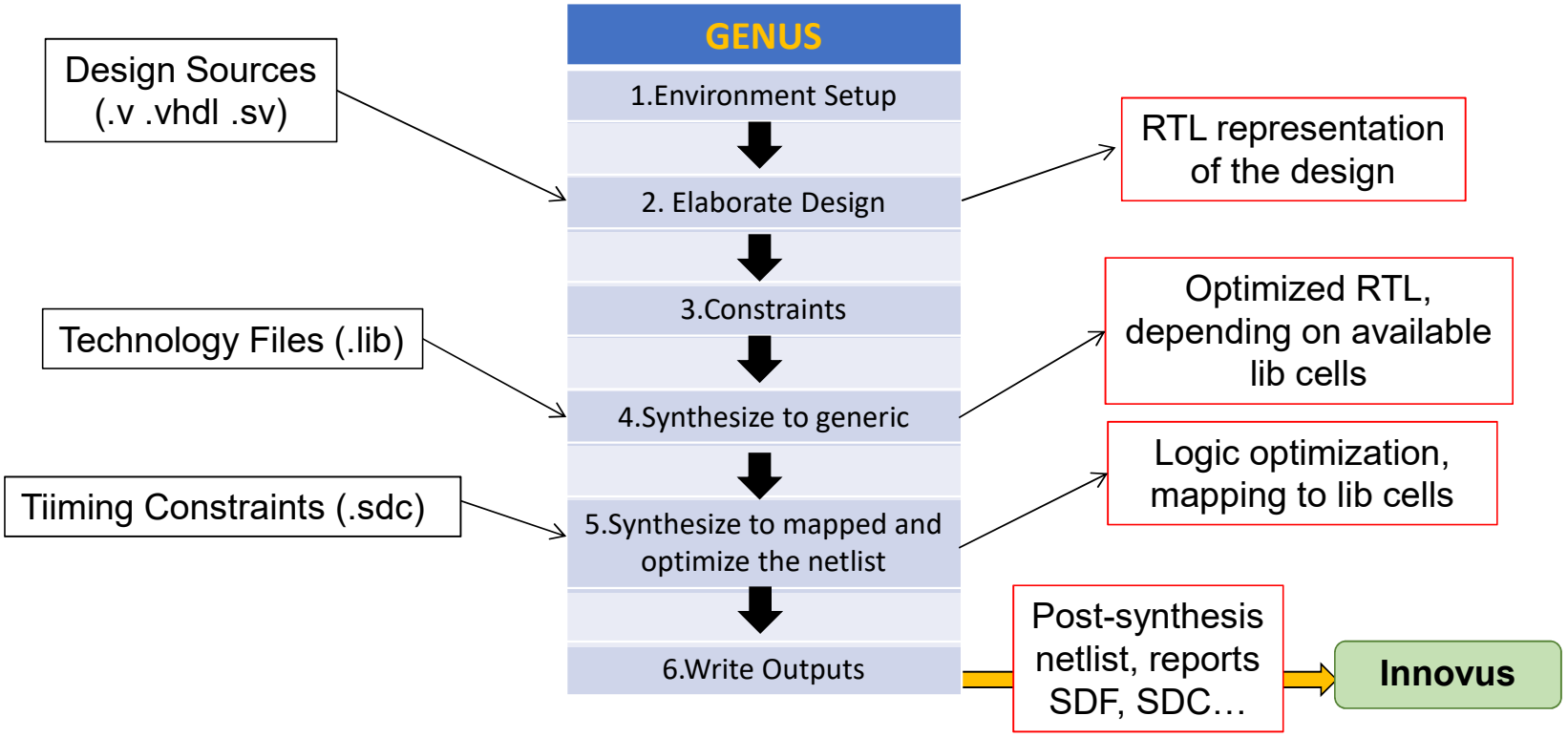
```
module MULT_struct
#(parameter DATA_SIZE = 8)
(
    input [DATA_SIZE-1:0] IN1,
    input [DATA_SIZE-1:0] IN2,
    input MODE,
    output [2*DATA_SIZE-1:0] OUT
);

    wire [DATA_SIZE - 1:0] PP [DATA_SIZE-1:0];  // Partial products
    wire [DATA_SIZE - 1:0] w_sum [DATA_SIZE-1:0];  // Internal sums
    wire [DATA_SIZE -1:0] w_carry;   // Internal carries

    assign w_sum[0] = PP[0];
    assign w_carry[0] = 1'b0;

    genvar i,j;
    generate

        for(j = 0; j < DATA_SIZE; j = j + 1)
        begin
            assign PP[j] =  IN1 & { DATA_SIZE{IN2[j]} };
        end

        for(i = 1; i < DATA_SIZE; i = i + 1)
        begin
            CLA #(DATA_SIZE) adder
```

Step 2: Synthesis

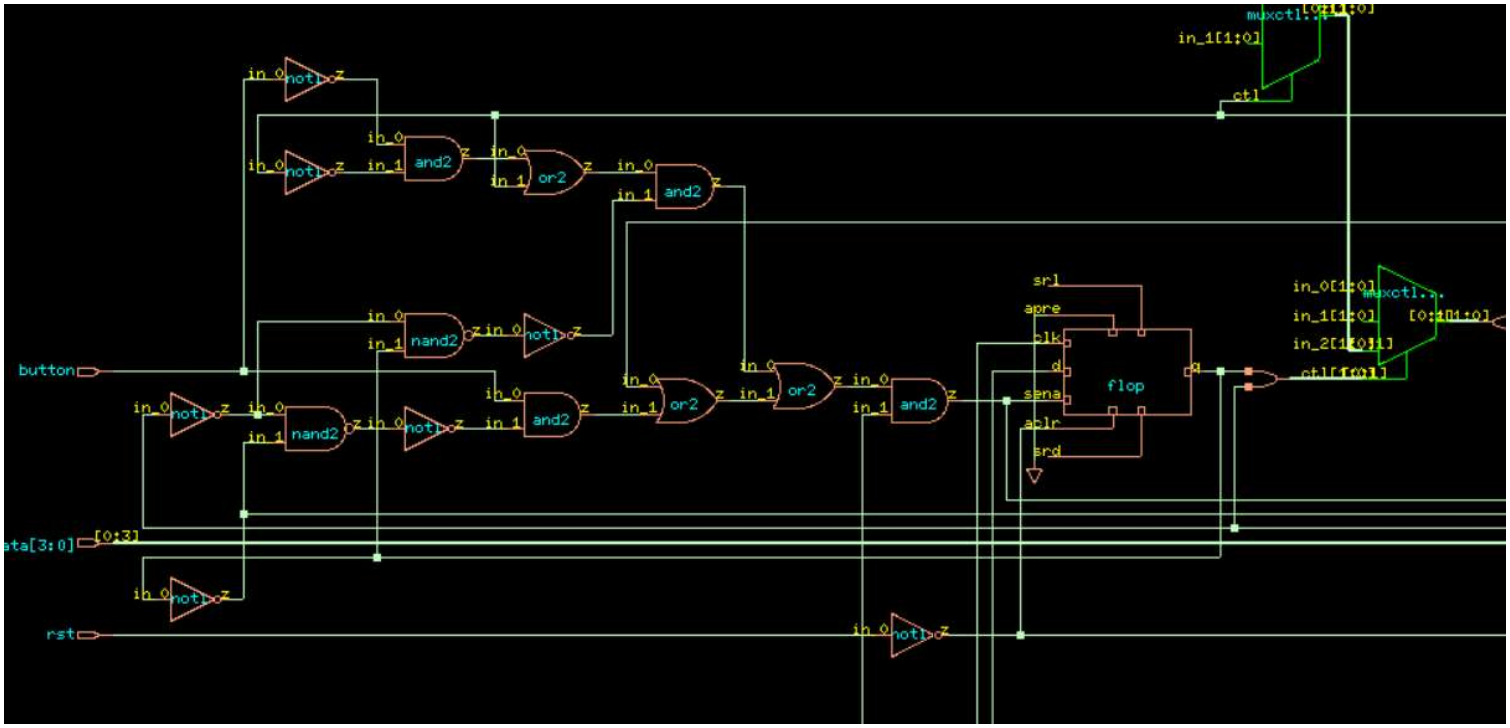
- The job of translating and optimizing the RTL code into the gate level netlist.
- The synthesis tool(yosys) takes HDL codes, liberty file(.lib) and design constraints to generate gate level netlist
- Synthesizing a netlist consists of a bunch of basic steps

SYNTHESIS
1.Environment Setup
2. Elaborate Design
3.Constraints
4.Synthesize to generic
5.Synthesize to mapped and optimize the netlist
6.Write Outputs

Step 2: Synthesis – Basic Flow



Step 2: Synthesis –Elaborate



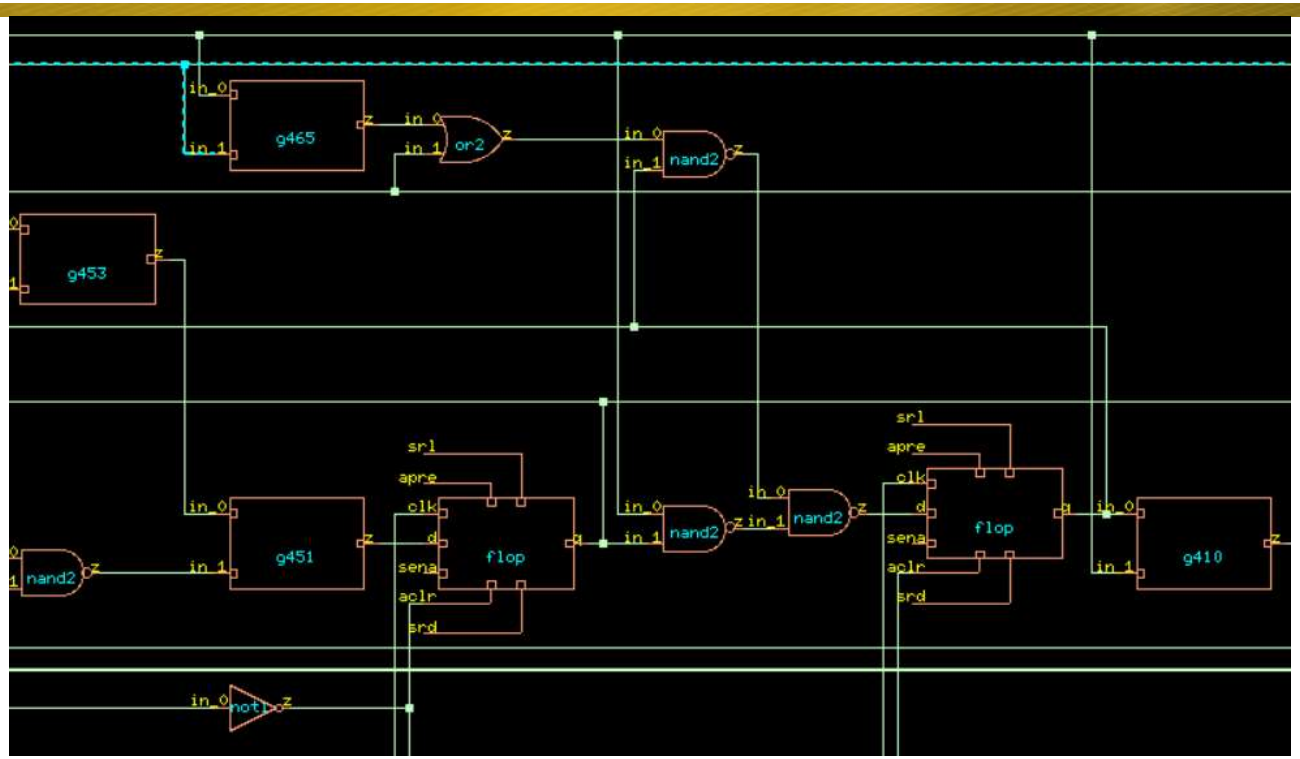
A section of a post-elaboration schematic

Step 2: Synthesis –Generic Synthesis

- Synthesizing to generic optimizes the RTL from elaboration, depending on the cell element types our library have.

```
set_db syn_generic_effort medium
# Sets tool's optimization effort during this phase. Can be
low, medium or high

syn_generic <design>  # Run synthesize to generic on design
```

Step 2: Synthesis –Generic Synthesis

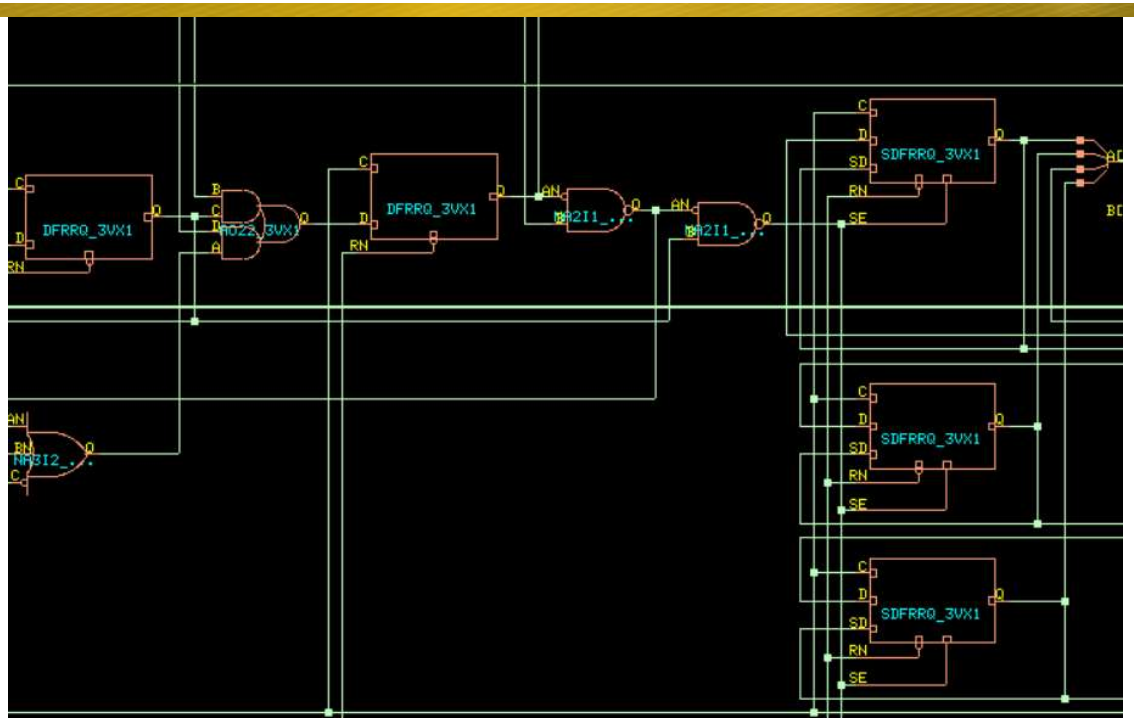


A section of a generic synthesized schematic

Step 2: Synthesis –Synthesize to Mapped & Optimize

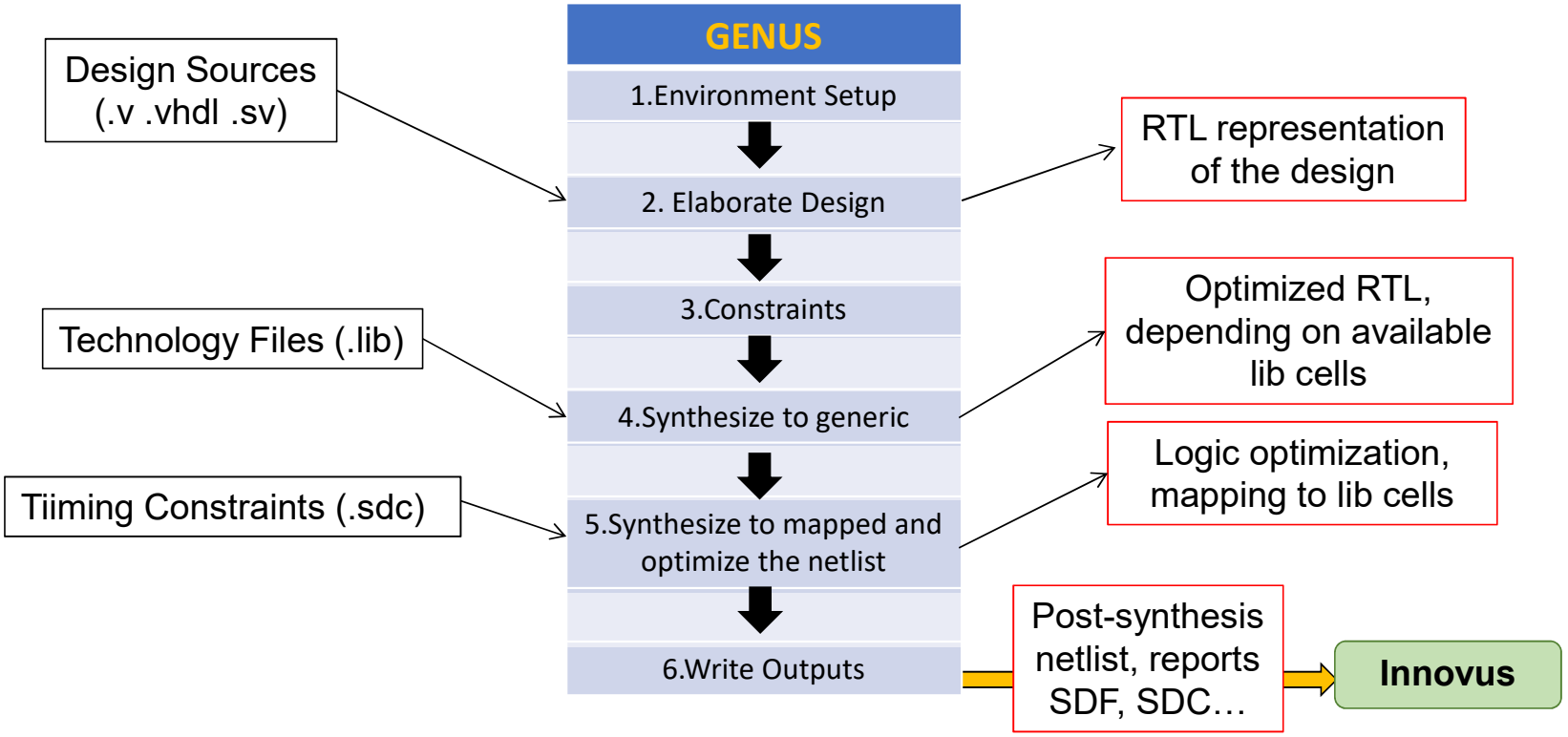
- Synthesizing to map means to apply logic optimization on the generic netlist and replace the cell slots in generic synthesis with actual cell library elements, regarding to target path delays
- Mapped netlist can be further optimized by running “syn_opt” command. This command runs gate level optimizations to improve critical paths, and recover area for non-critical paths

Step 2: Synthesis –Synthesize to Mapped & Optimize

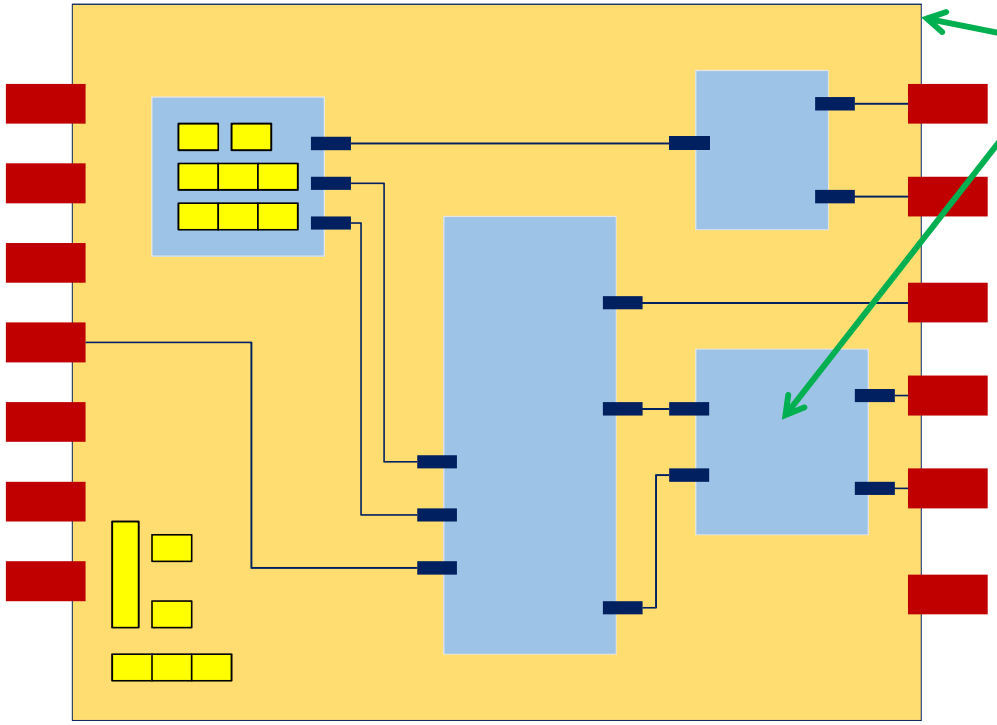


A section of a mapped & optimized schematic

Step 2: Synthesis – Basic Flow



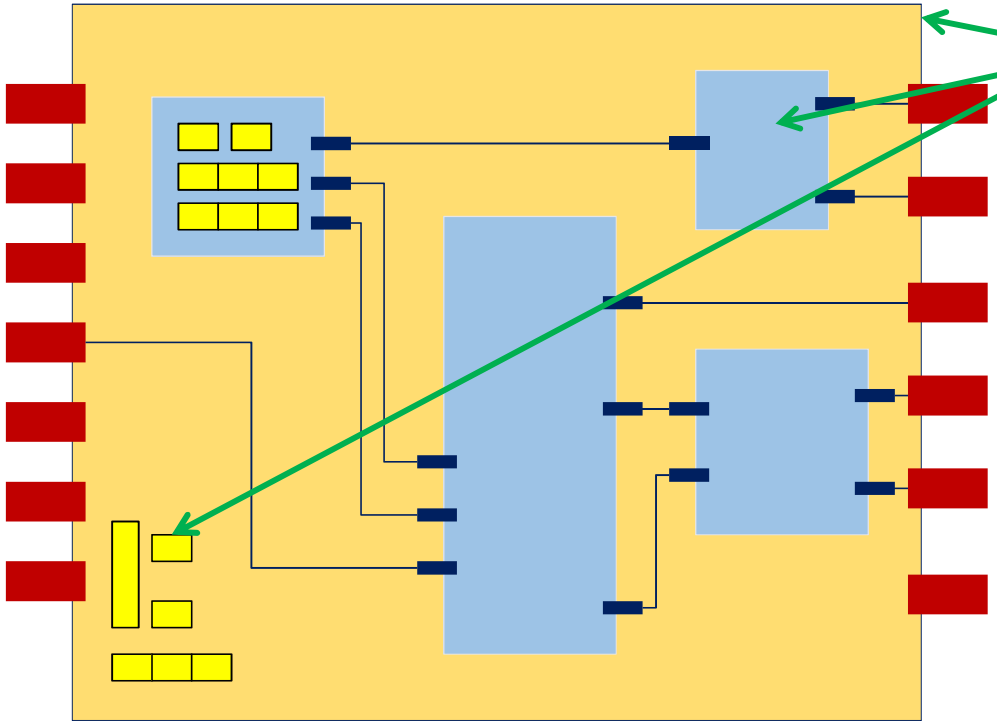
Design Objects



Design: A full circuit or module, container for cells and other sub-objects

```
>current_design
#Returns top design

>get_designs
#Returns a list of designs created in current session
```

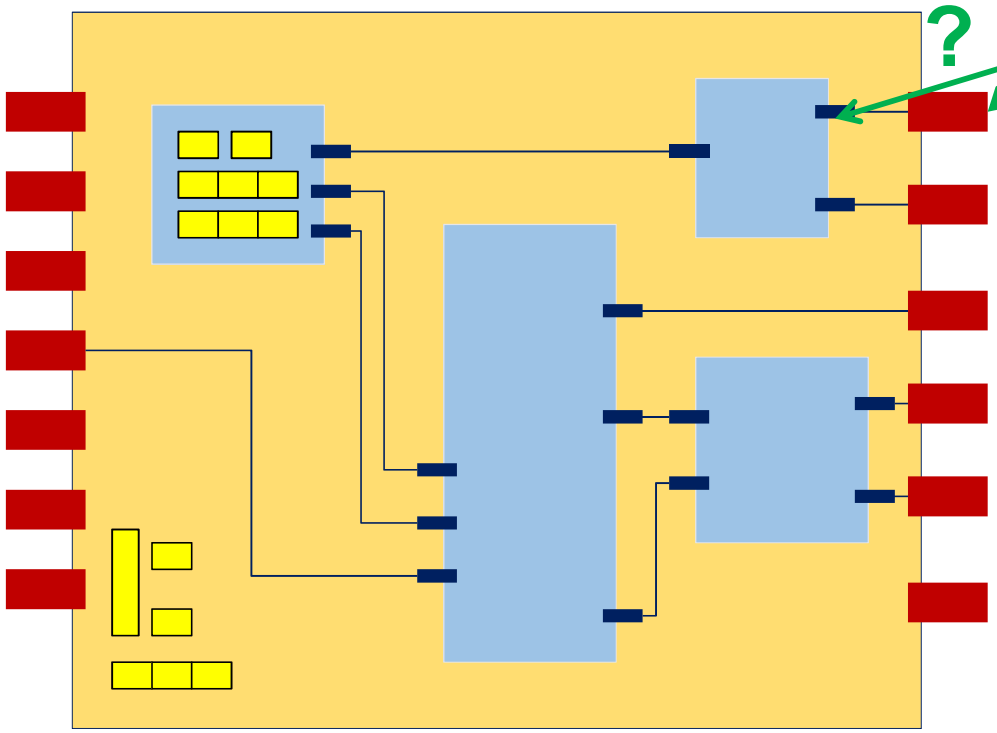
Design Objects



Cells: Instances of a design or a library component

```
>current_design
#Returns top design

>get_cells #Returns
a list of cells in
the specified
design
```

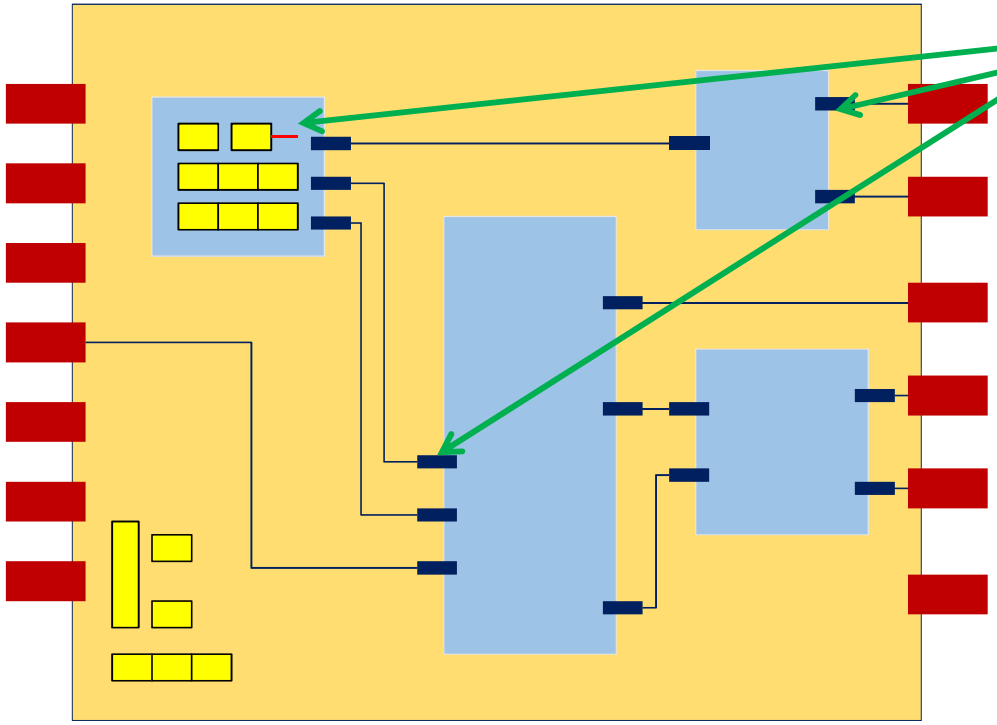
Design Objects



Ports: Signal entry and exit points of a design (top level)

```
>get_ports #Returns the list of ports of the top design
>all_inputs #Returns list of inputs of the top design
>all_outputs #Returns list of outputs of the top design
```

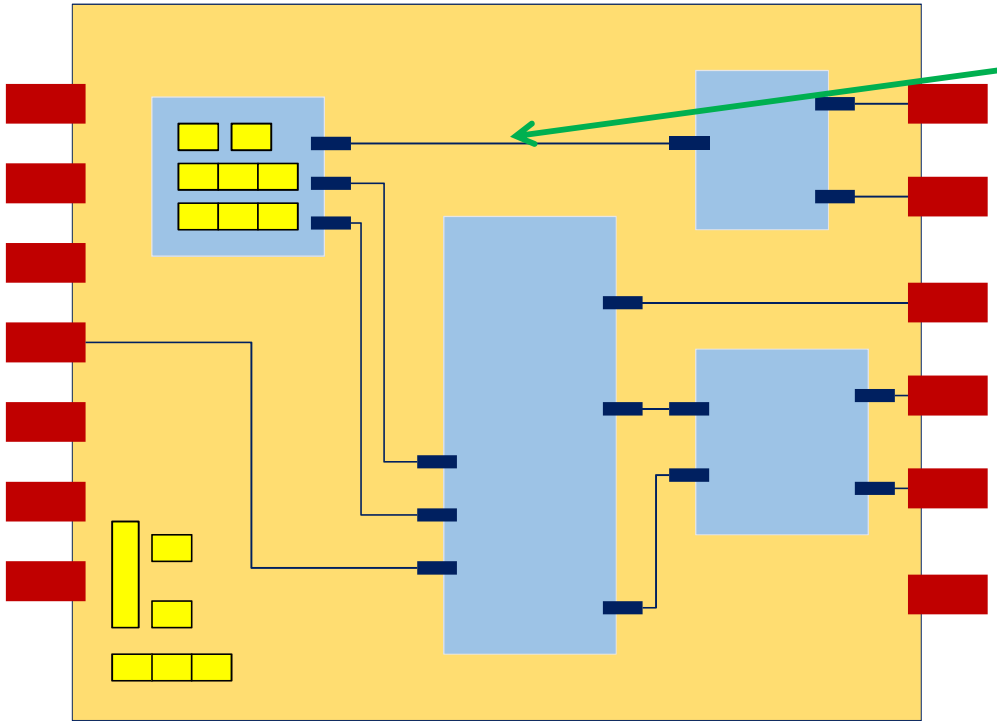
Design Objects



Pins: Hierarchical ports of a desing instance or a library cell

```
>get_pins #Lists all hierarchical pins of the top design
```

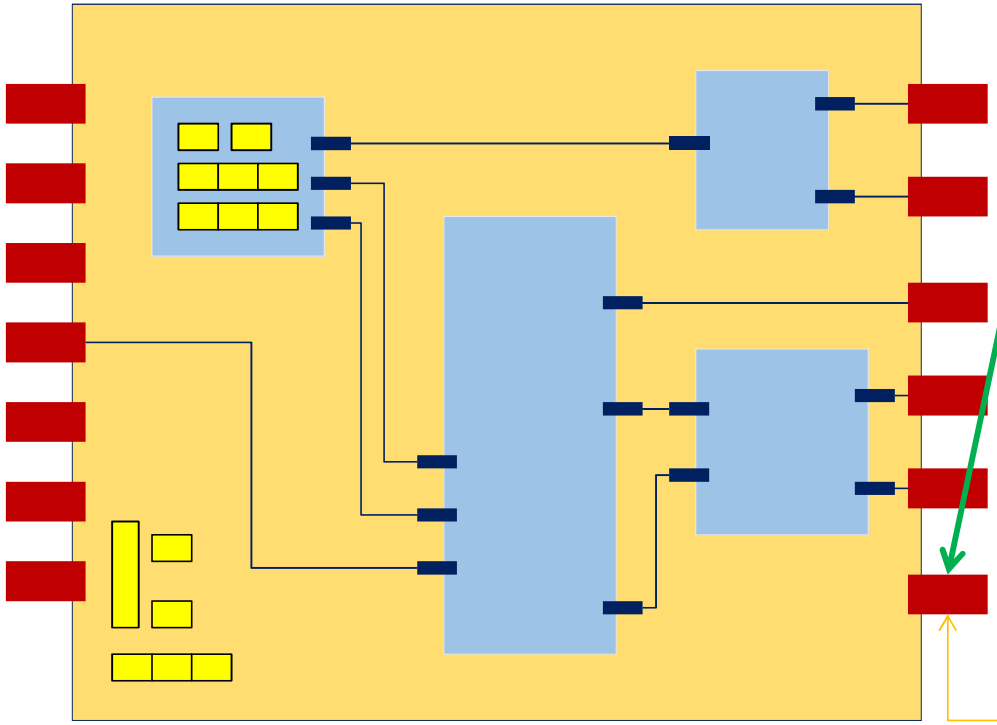
Design Objects



Nets: Interconnections between cell/block pins and ports

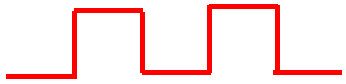
```
>get_nets #Lists all hierarchical pins of the top design
```

Design Objects



Clocks: Ports and pins that drive sequential cells

```
>create_clock #Create a clock waveform and connect it to a port
>get_clocks #Returns the clock object matching the pattern
>all_clocks #Lists all created clocks in the design
```

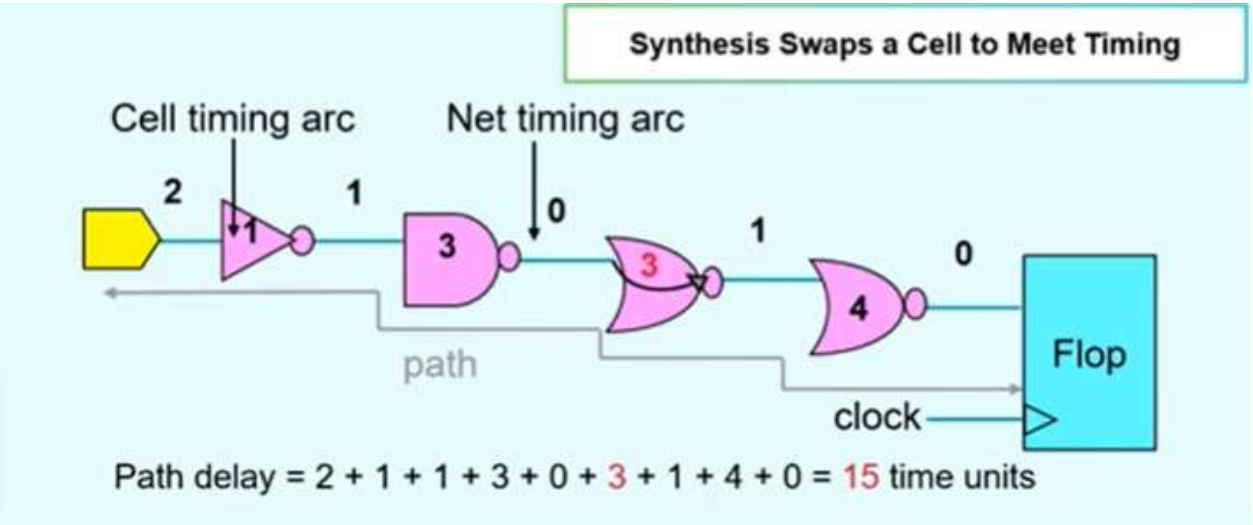


Step 3: Static Timing Analysis (STA)

- STA is a process of adding cell delays and net delays to obtain path delays, and comparing the path delays to the timing specification
  - Path delays and calculated, then suitable cells are chosen from the timing library to meet design requirements
  - Optimizations are done so the circuit can work at the specified clock frequency

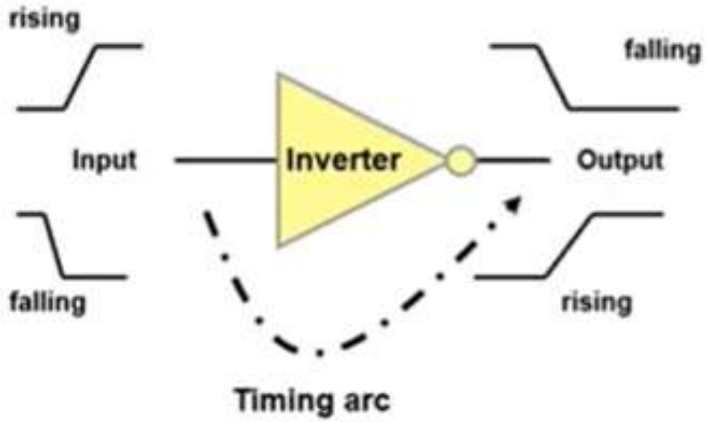
Step 3: Static Timing Analysis (STA)

- STA is a process of adding cell delays and net delays to obtain path delays, and comparing the path delays to the timing specification



Step 3: Static Timing Analysis (STA)

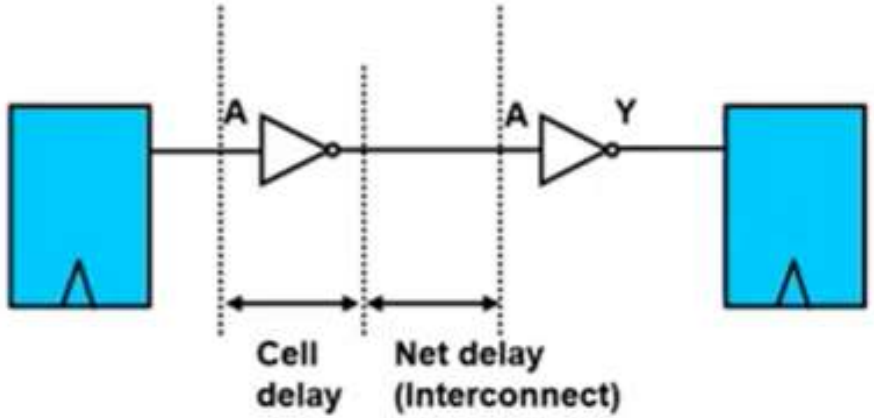
- Timing Arcs: Imaginary arcs that represent the relation of an input causing a change on the output. They provide an illustration for functionality of a gate



- **Rise and fall sections of timing arcs are often not symmetrical**
- **This info is provided by the timing library (.lib)**

Step 3: Static Timing Analysis (STA)

- Cell Delays: Signal delay caused by the switching of transistors within a cell
- Net Delays: Delay between the time a signal is first applied to a net and time it takes to reach other devices connected to that net.



Step 3: Static Timing Analysis (STA)

- Signal Transition: Time taken for signal to change from low->high or high->low
- Slew: Rate of transition (Volts per nanoseconds)
- Signal transition and slew are interchangeable in design tools
- Usually, signal transitions are calculated from slew rate and threshold information located in .lib files

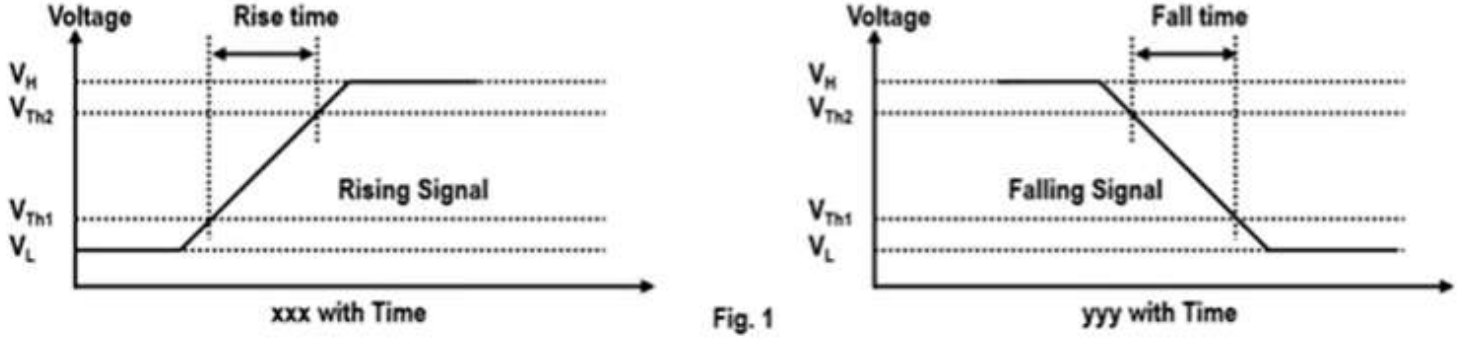
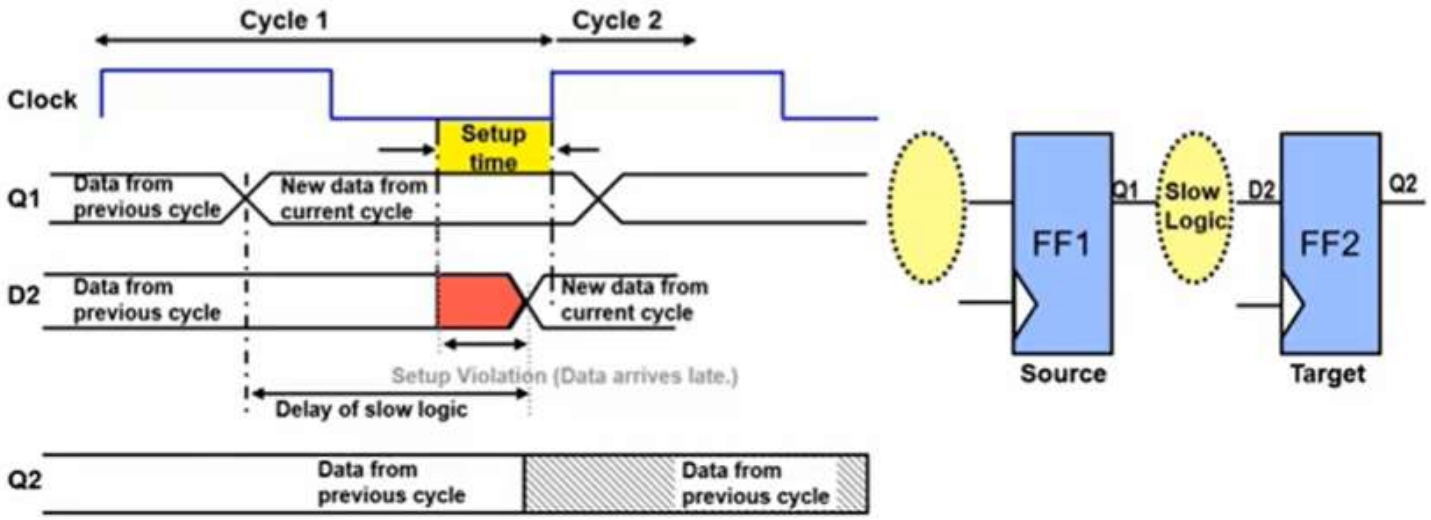


Fig. 1

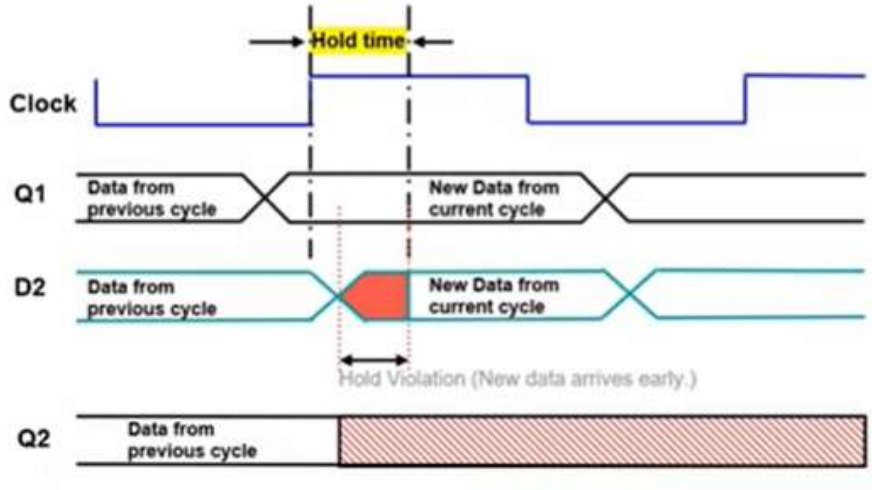
Step 3: Static Timing Analysis (STA) – Setup Time

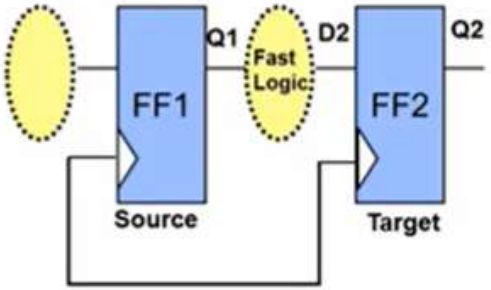
- The duration of time that the flip-flop input must be stable before the triggering edge of the clock.
- If this is not met, the data at Q will be unknown.



Step 3: Static Timing Analysis (STA) – Hold Time

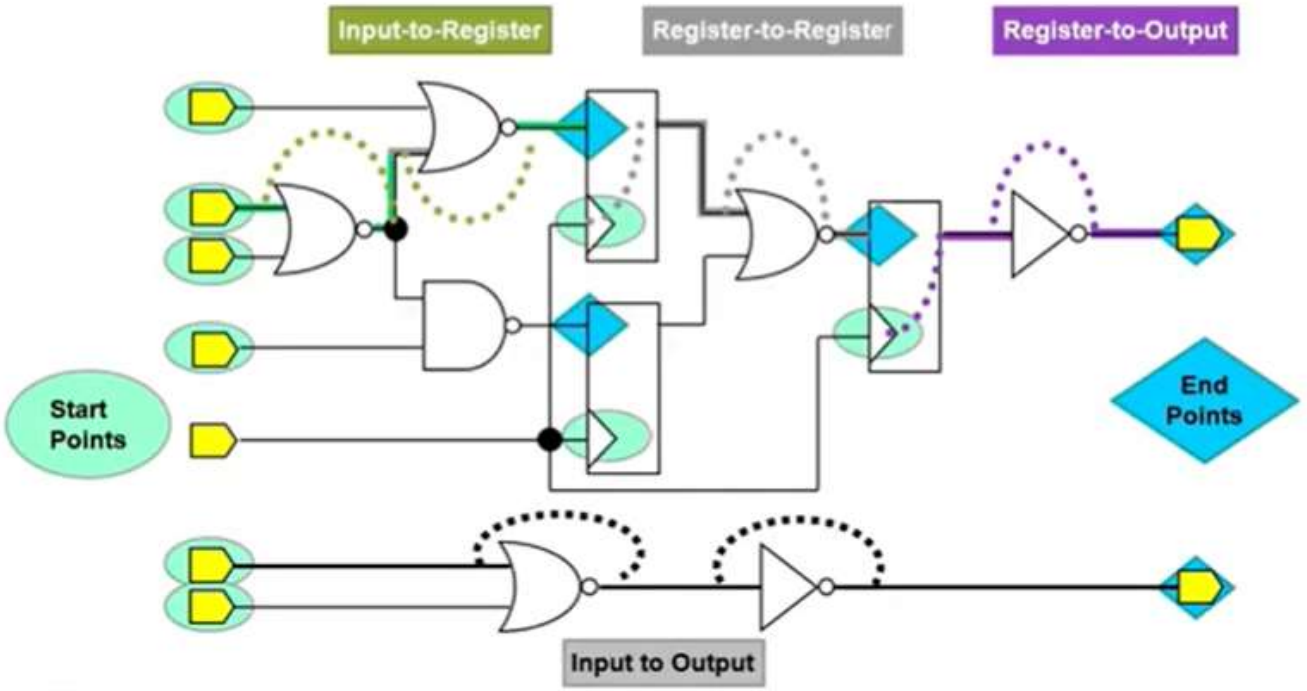
- The duration that the flio-flop input D must be stable after triggering edge of the clock
- If this is not met, the data at Q will be unknown.





Step 3: Static Timing Analysis (STA) – Timing Paths

- Combination of all timing arcs from a start point to an end point
- There are 4 basic timing path groups



Step 3: Static Timing Analysis (STA) – Timing Paths

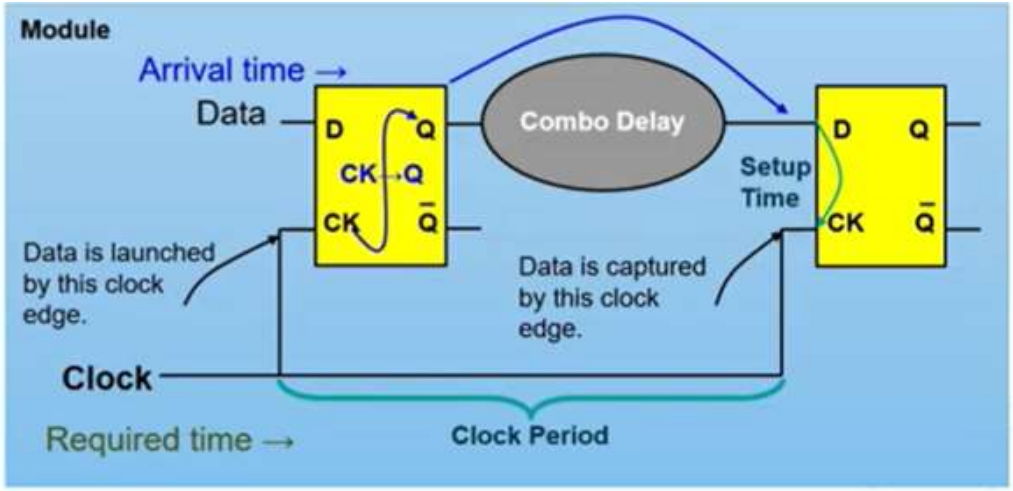
- Required Time: The time required for a signal to arrive. Determined by timing constraints provided (like clock period).
- Arrival Time: Time that signal actually arrives.
- Slack: Required Time – Arrival Time

- Positive slack means a path met timing, whereas negative slack means the path violated the timing.

- All paths must meet the timing for your circuit to work as expected in all cases

Step 3: Static Timing Analysis (STA) – Timing Paths

- **Register to Register**

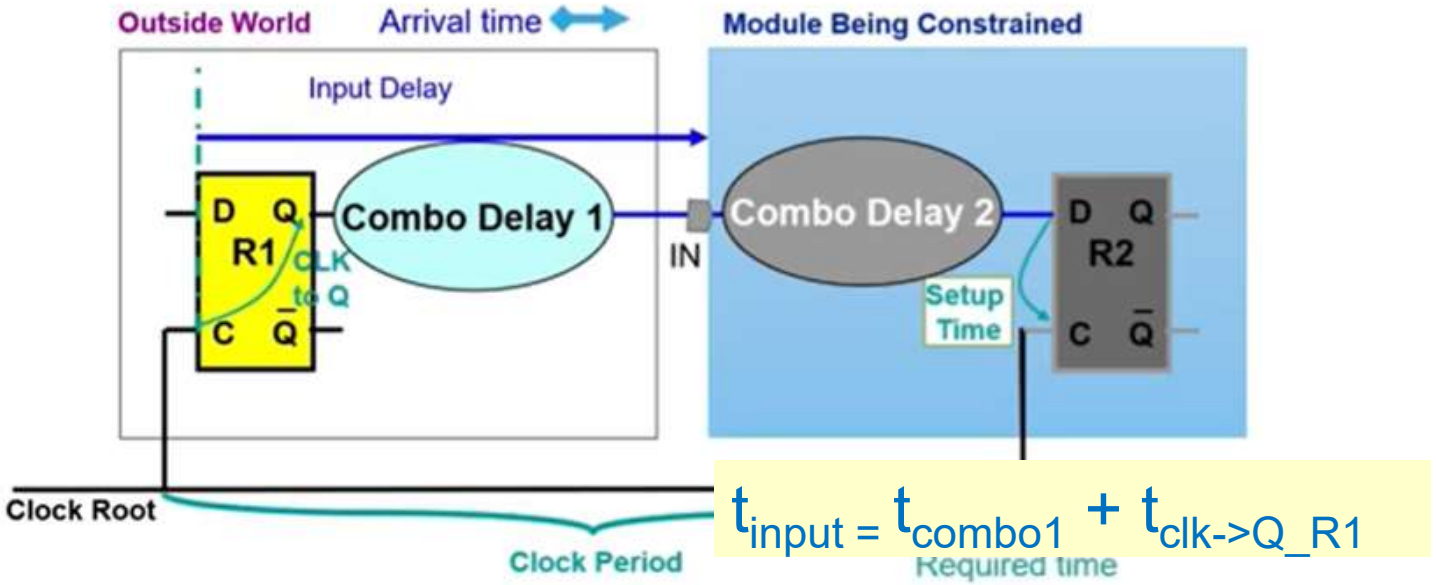


Arrival Time: $t_{CLK->Q} + t_{combo}$ < Required Time: $T_{CLK} - t_{setup}$

!! Clock non-idealities also included in these equations, if defined

Step 3: Static Timing Analysis (STA) – Timing Paths

- **Input to Register**
  - e

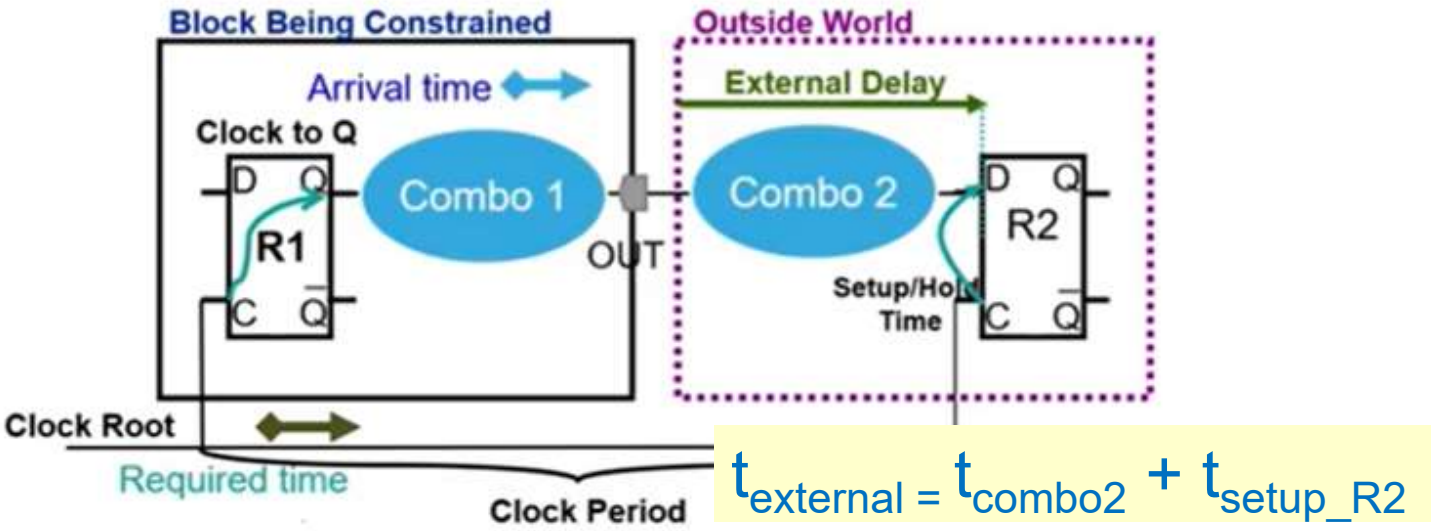


$t_{input} = t_{combo1} + t_{clk->Q\_R1}$

Arrival Time: $t_{input} + t_{combo2}$ < Required Time: $T_{CLK} - t_{setup\_R2}$

Step 3: Static Timing Analysis (STA) – Timing Paths

- **Register to Output**

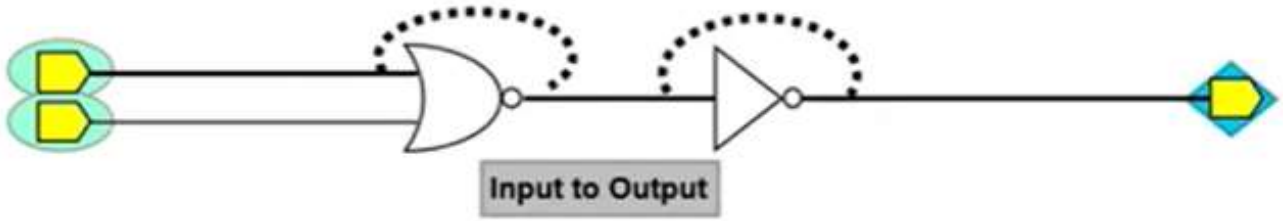


$t_{external} = t_{combo2} + t_{setup\_R2}$

Arrival Time: $t_{clk->Q} + t_{combo1} + t_{external}$ < Required Time: T_{CLK}

Step 3: Static Timing Analysis (STA) – Timing Paths

- **Input to Output (Pure Combinational Path)**
  - Not clocked
  - Can be constrained with virtual clocks or path delay constraints



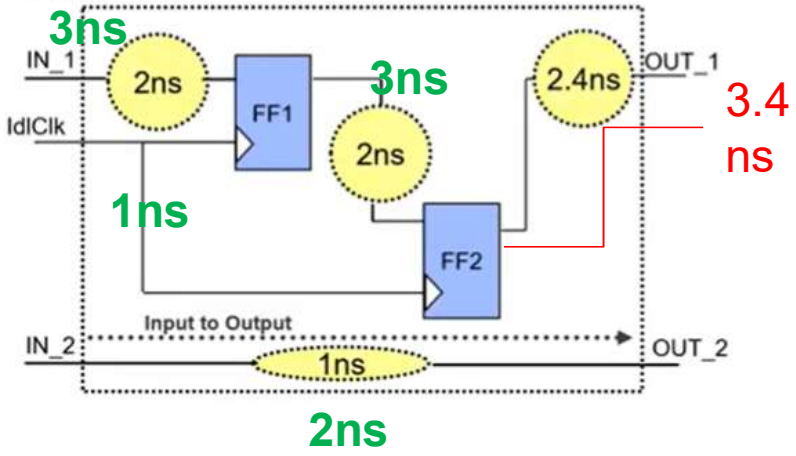
Arrival Time: input_delay+output_delay+combo_delay < Required Time: T_{CLK}

Step 3: Static Timing Analysis (STA) – Timing Paths

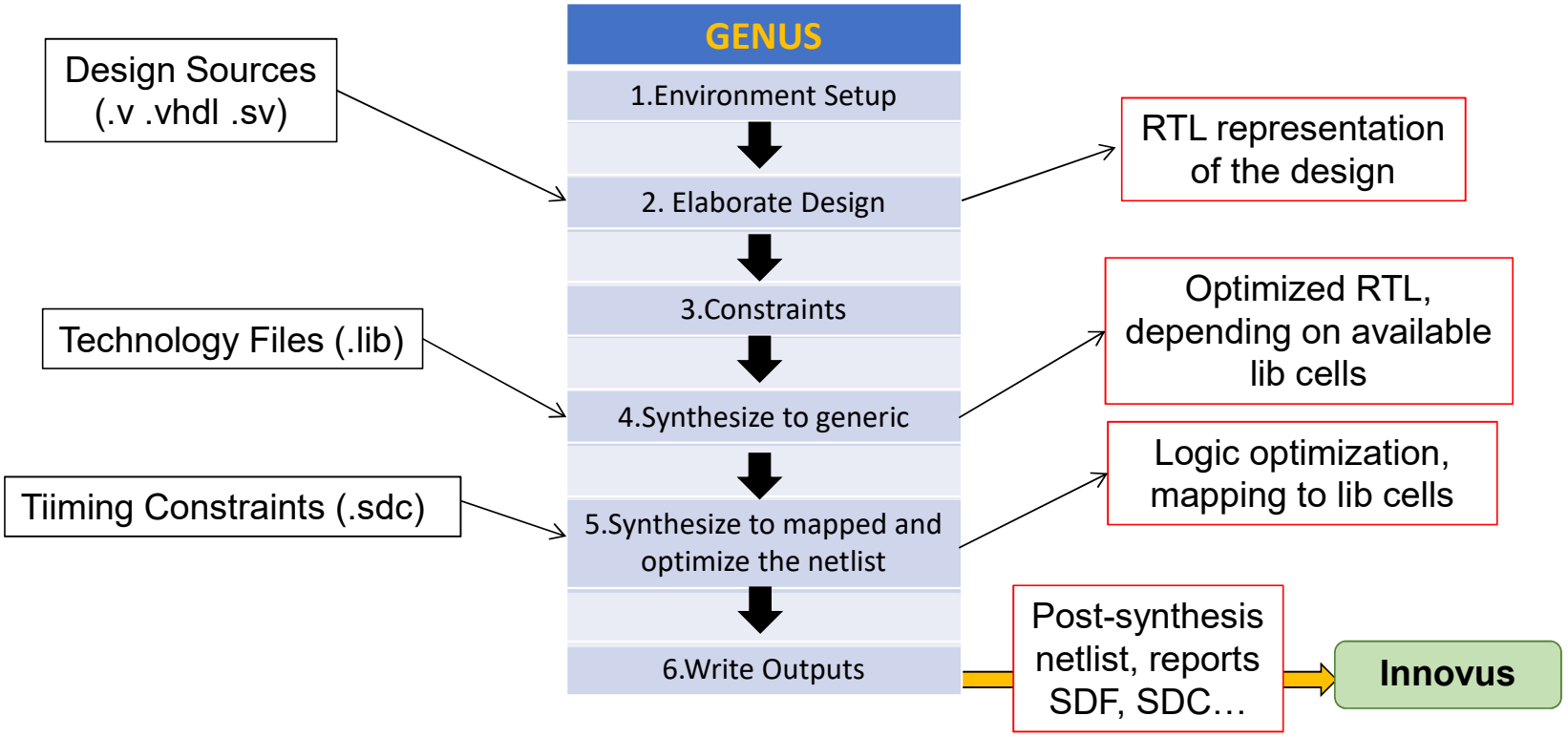
Here are the parameters for this activity:

- The clock-to-Q delay of each flop is 0.5 ns.
- The setup of each flop is 0.5 ns.
- All input and output delays are 0.5 ns.
- The clock period of Clk is 4 ns.

What is the worst timing slack based on this illustration?



Step 2: Synthesis – Basic Flow



Constraints

- Set of restrictions/specifications that design tools must abide (design must meet through optimization)
- Mostly based of SDC (Synopsys Design Constraints) along with some other custom commands and equivalent writing styles, changing from tool to tool.
- Equivalent writing styles, changing from tool to tool.
- Can be given to the tool from an SDC file, or as Genus commands

Frequently used constraint types are:

Timing constraints, power constraints, net delay constraints, environmental constraints, design rules for manifacturing

❑ ***Ref: Genus Command Reference/SDC Constraints (genus_comref)***

Constraints

- **Design Rule Constraints:**
  - `set_max_capacitance`
  - `set_max_fanout`
  - `Set_max_transition`
  - …

❑These type of constraints are provided by fabrication process, for the resulting circuit to be manufacturable. Thus, these constraints cannot be relaxed

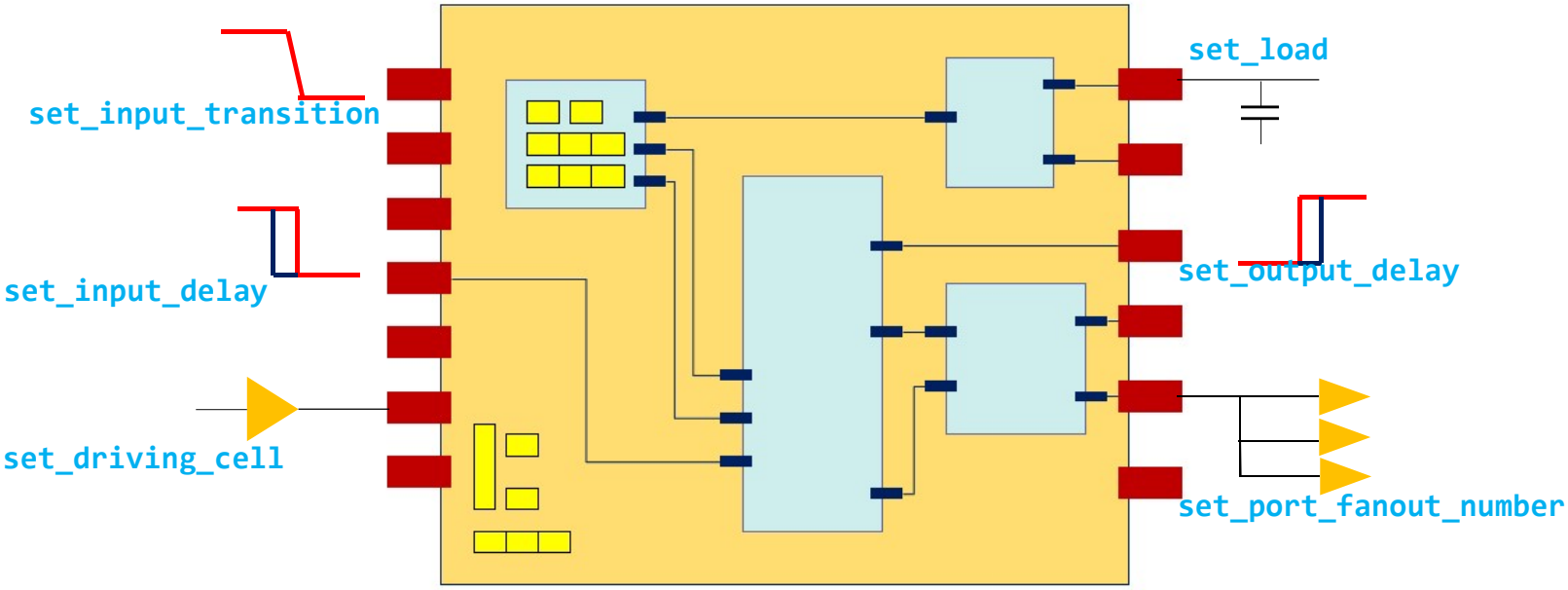
Constraints

- **Environmental Constraints:**
  - `set_drive`
  - `set_driving_cell`
  - `set_load`
  - `set_port_fanout_number`

❑In real life, ports are driven/drive other components. These can be modeled using environmental constraints

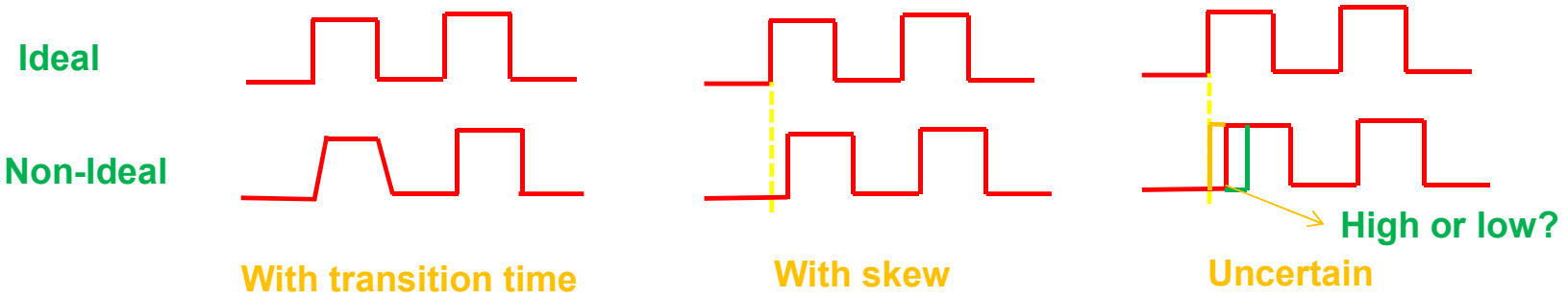
Constraints

- **Environmental Constraints:**



Constraints

- **Clock Constraints:** !! Clocks are essential for timing analysis
  - `create_clock`
  - `Set_clock_skew  #latency`
  - `set_clock_uncertainty`
  - `set_clock_transition`

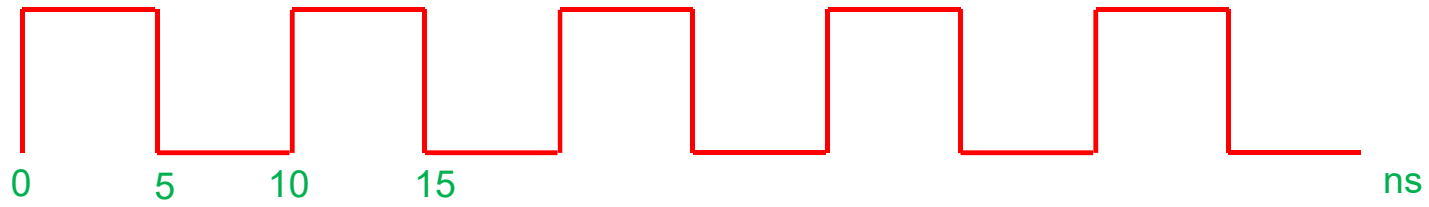


Constraints

- **Clock Constraints:**

Example:

```
create_clock -name "clock1" -period 10 -waveform {0 5} clk
#Creates a clock object named "clock1", with 10 ns period,
leading edge at 0 ns and trailing edge at 5ns. It will be
applied to the port named "clk" in the design.
```



Constraints

- **Clock Constraints:**
  - There are more advanced concepts on clocks, such as:
    - Creating generated clocks
    - Designs with multiple clocks
    - Defining Asynchronous clocks
    - Clock Gating
    - Clock-Domain Crossing

Constraints

- **Power Constraints:**
  - `set_max_leakage_power`
  - `set_max_dynamic_power`

❑Used to set goals for dynamic and static power consumption of the circuit

❑There exists many attributes on low power synthesis, check attribute reference for attributes starting with “lp_”

❑***Ref: Genus Low Power for Legacy UI (genus_lowpower_legacy)***

Constraints

- **Exceptions:**
  - `set_max_delay`
  - `set_min_delay`

  ❑Max and min delay constraints are used to set path delay goals to purely combinational paths

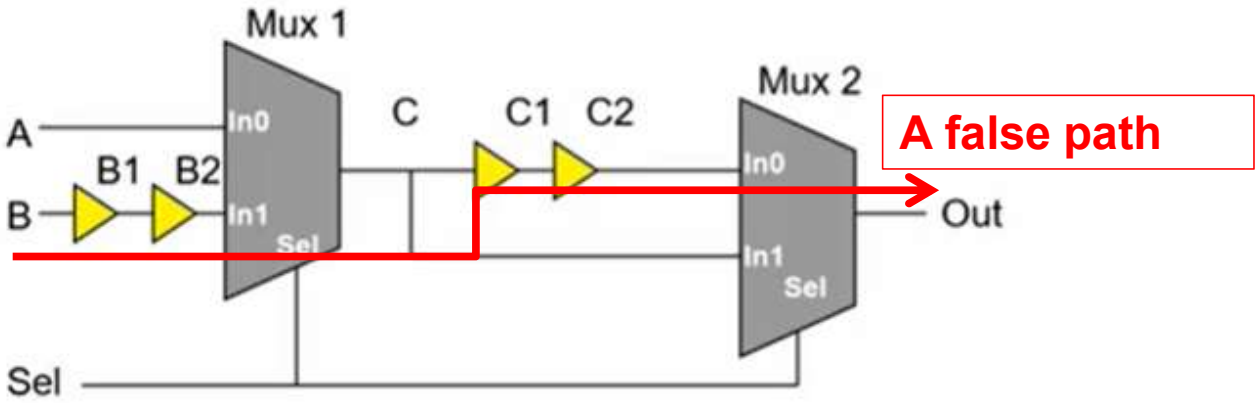
Example:

```
set_max_delay –from [get_pins adder/in*] –to [get_pins
adder/out*] 2
#Constraints the delay between all combinations of “in” and
“out” pins of module “adder” to 2 ns.
```

Constraints

- **Exceptions:**
  - `set_false_path`

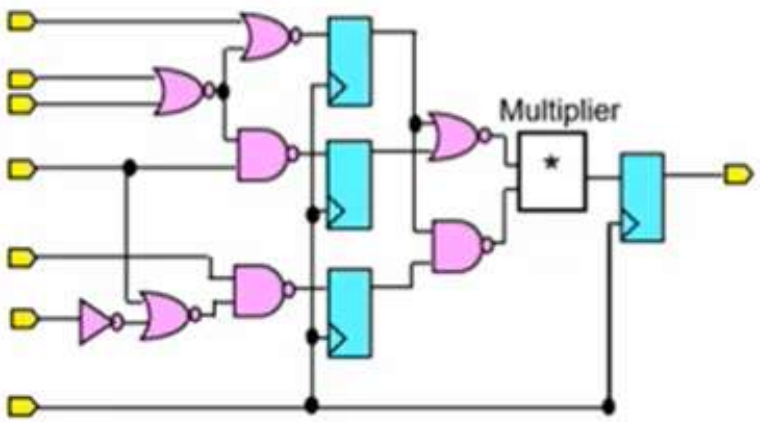
  ❑Can be used to exclude some paths from being analyzed.



Constraints

- **Exceptions:**
  - `set_multicycle_path`

  ❑Used to add extra clock cycles to some combinational logic if required



References

- *https://openlane.readthedocs.io/en/latest/getting_started/index.html*
- https://github.com/The-OpenROAD-Project/OpenLane
- https://github.com/google/skywater-pdk
- *https://www.youtube.com/channel/UC5qqAsDzbA0zAQNBBQVsS0Q - Cadence YouTube Channel*