WPI

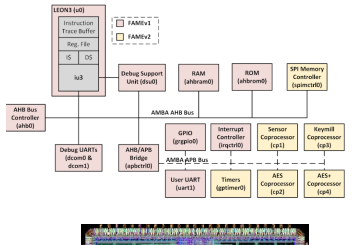
Building Cryptographic ASICs with Open Source Design Tools

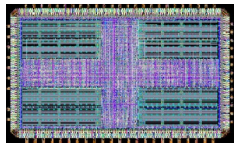
Patrick Schaumont
Electrical and Computer Engineering

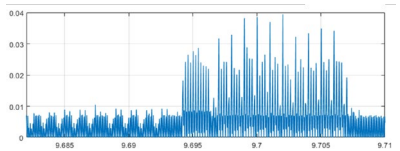
pschaumont@wpi.edu

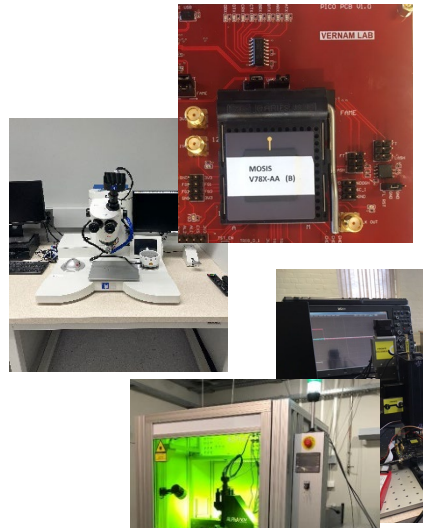
Vernam Lab

Vernam is a Research Center of Excellence in Cybersecurity and Hardware Security in the New England area. A group of eight faculty and their students tackle advanced research challenges across the systems stack.

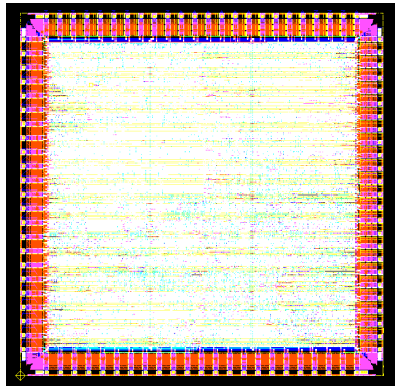




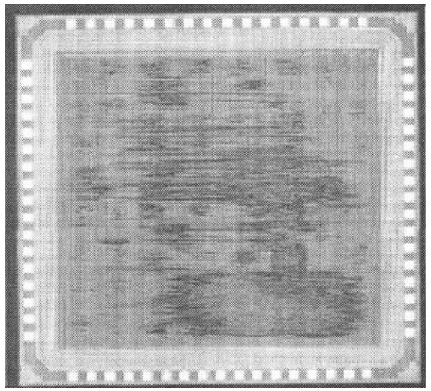




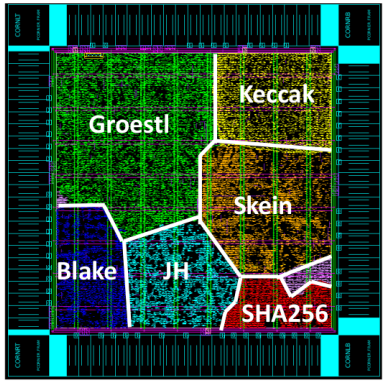
A few cryptographic chips



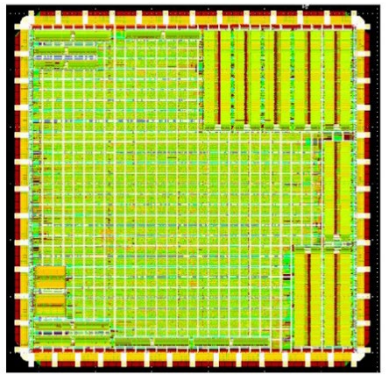
1999 – Modem ☺



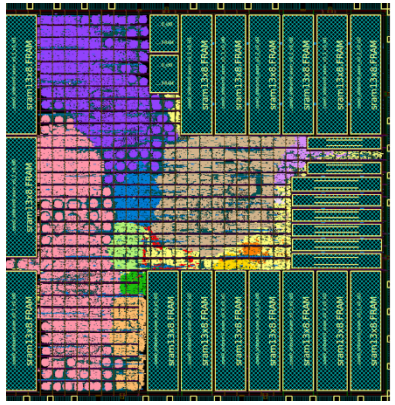
2002 - AES



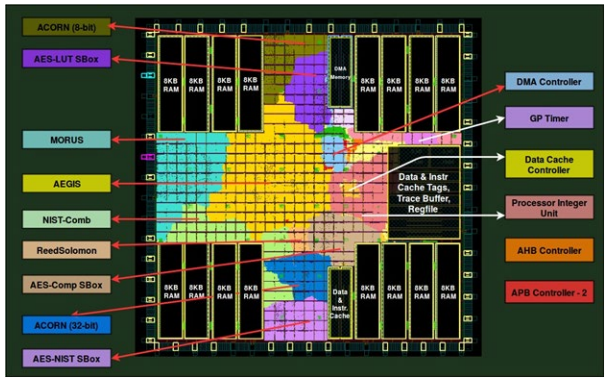
2013 - SHA3



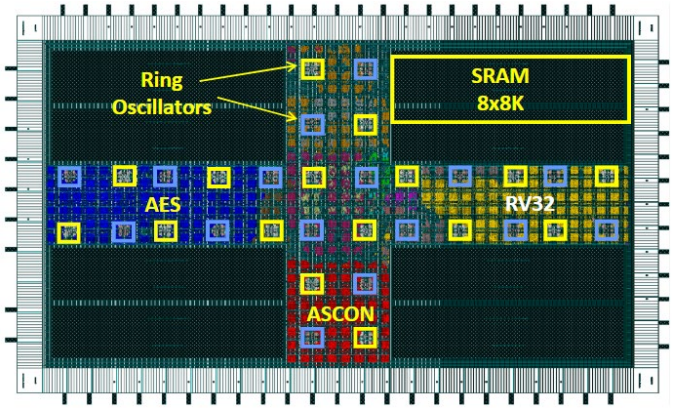
2015 - FAME v1



2017 - FAME v2



2019 - SKIVA + CAESAR



2021 - PICOSOC

The Plan!

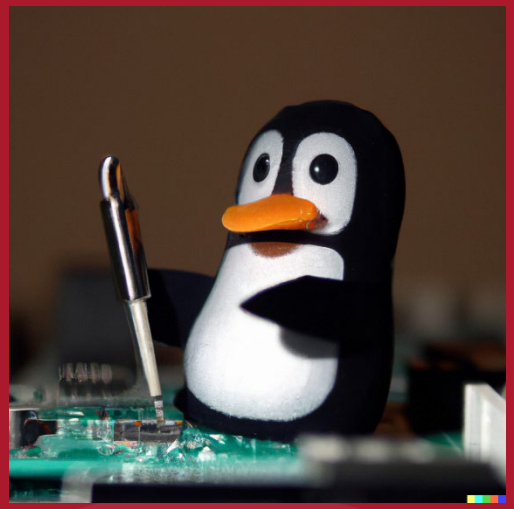
Time	Topic
14:00 – 14:30	Open Source Chips
14:30 – 15:00	Handson I: A Quick Chip
15:00 – 15:30	Timing Analysis
15:30 – 16:00	Break
16:00 – 16:30	Handson II: Analyzing the Critical Path
16:30 – 17:00	Quality Metrics for Cryptographic Hardware
17:00 – 17:30	Handson III: Power to the Presilicon People!

The fine print:
This tutorial makes minimal assumptions on your current hardware design knowledge. I assume that you are cryptographers who happen to be involved with implementation; not the other way around. If you are a super experienced ASIC designer, this tutorial is probably not for you. But there is coffee for everybody at 15:30!

Learning Objectives

1. Learn about the major components in an ASIC layout
2. Learn about the major design steps involved in building an ASIC
3. Use OpenROAD and associated open-source design tools
4. Learn about and use the Skywater SKY130 open-source cell library
5. Build simple cryptographic circuits into ASIC layout
6. Analyse ASIC implementation properties such as timing, area, power

WPI



Part I
Open Source Chips

Why Is All That Hardware Needed?

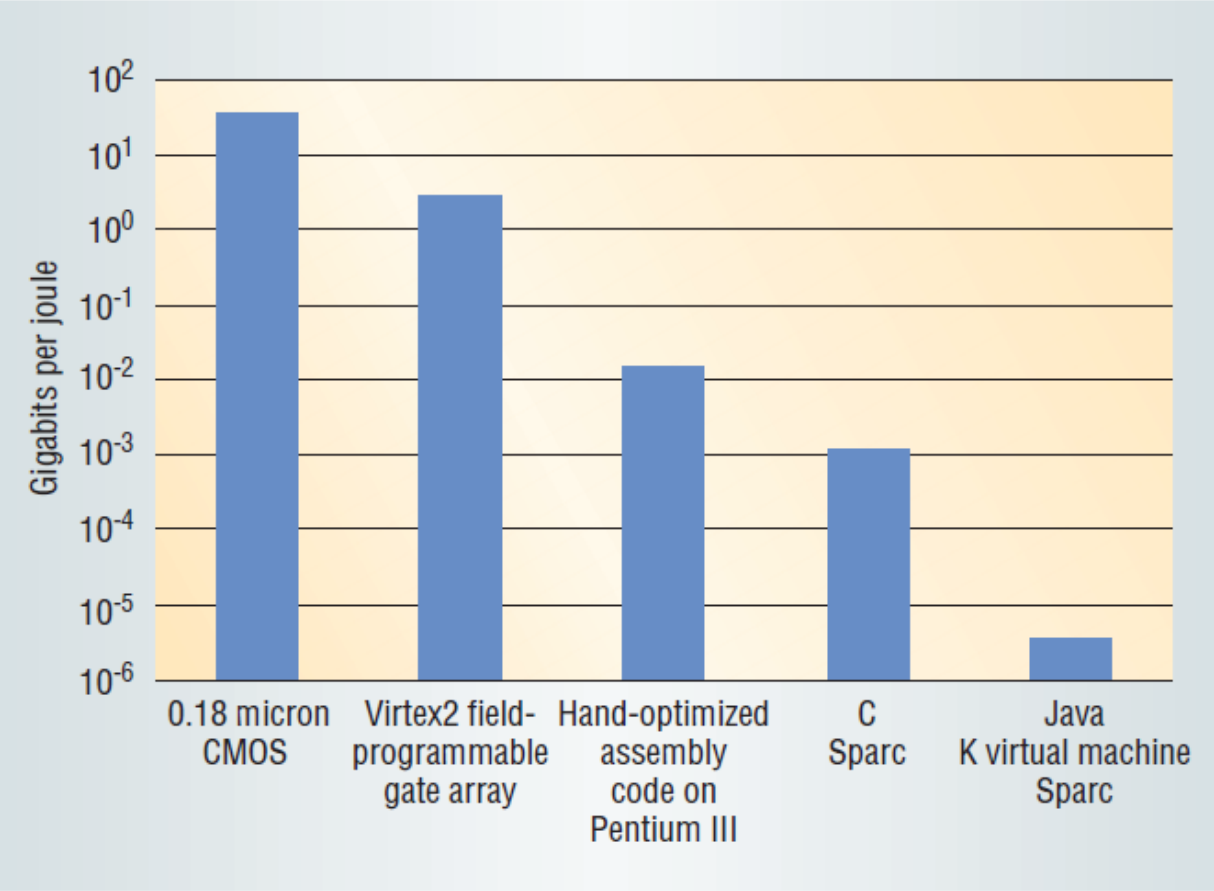


Figure 1. Energy efficiency of the Advanced Encryption Standard on five platforms. The energy efficiency, which spans seven orders of magnitude, is expressed as the number of gigabits the system can encrypt per joule (www.ee.

- For the same functionality, hardware is more energy efficient than software

P. Schaumont and I. Verbauwhede, "Domain-specific codesign for embedded security," in Computer, vol. 36, no. 4, pp. 68-74, ***April 2003****, doi: 10.1109/MC.2003.1193231.*

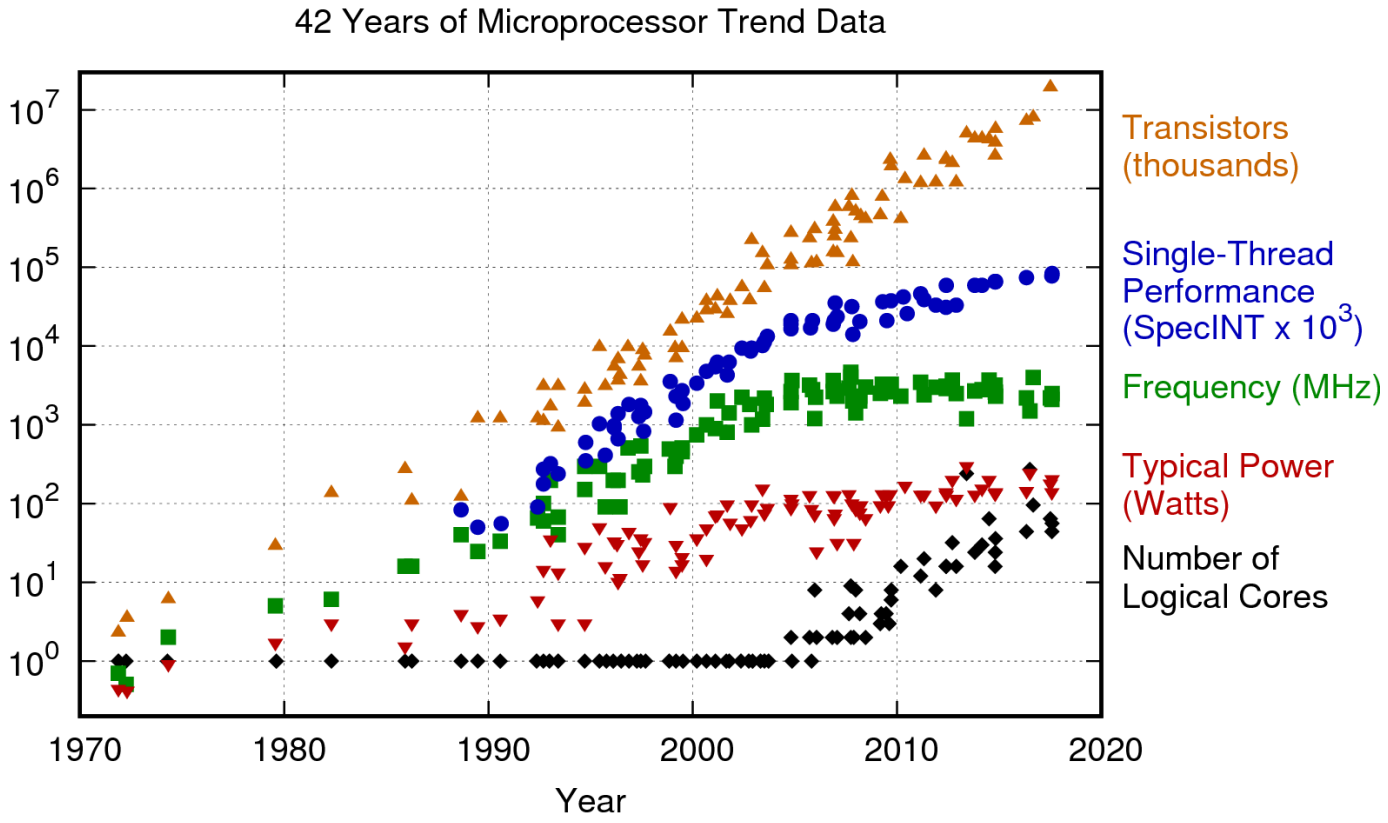
Need for Efficiency Became A Call to Arms!

Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	—	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. LHampson, D. Sanchez, et al., "There's plenty of room at the Top: What will drive computer performance after Moore's law?" in Science, American Association for the Advancement of Science, vol. 368, no. 6495, ***2020****. https://doi.org/10.1126/science.aam9744*

Need for Efficiency Became A Call to Arms!



- In the future, performance gains will come from better utilization of transistors
- "Plenty of room at the top" by
  1. algorithmic innovation
  2. better software (better mapping to hardware)
  3. better hardware (domain specialization)

https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

The Hardware Renaisscance Is Now

- Agile Hardware Development
  - Small teams and quick iterations achieve better results, faster than large teams in waterfall development cycles
- Open Source Hardware Development
  - Based on Reuse and Improvement
  - Tremendous Opportunity from Open Access
- Affects every Aspect of Hardware Design
  - Open Intellectual Property
  - Open Design and Verification Tools
  - Open Technology

IEEE Micro July/August 2020

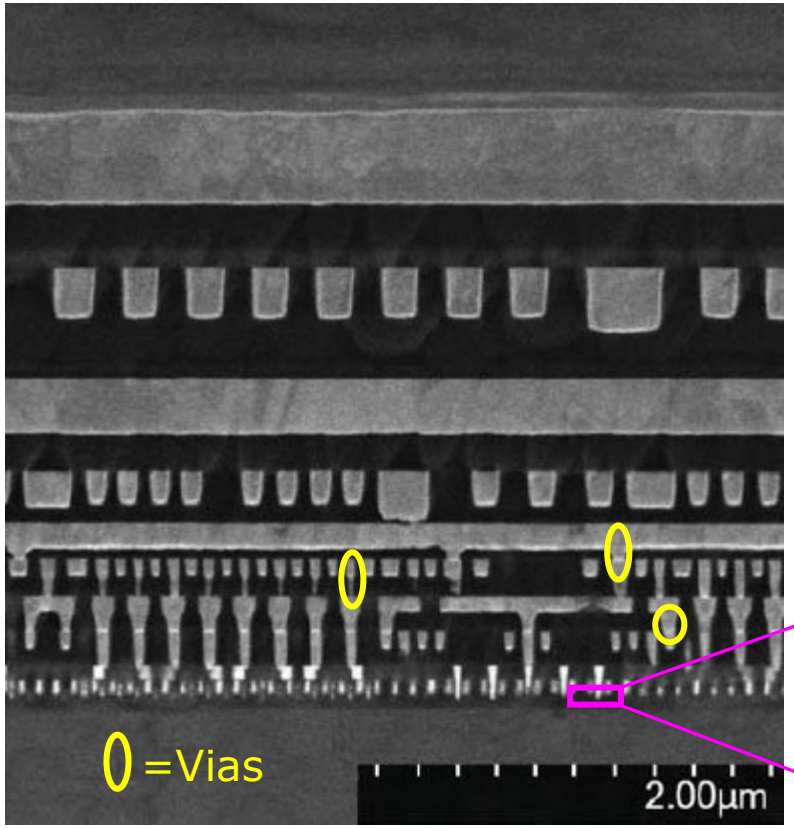
WPI

All right, let's look beyond software optimization and build better hardware!

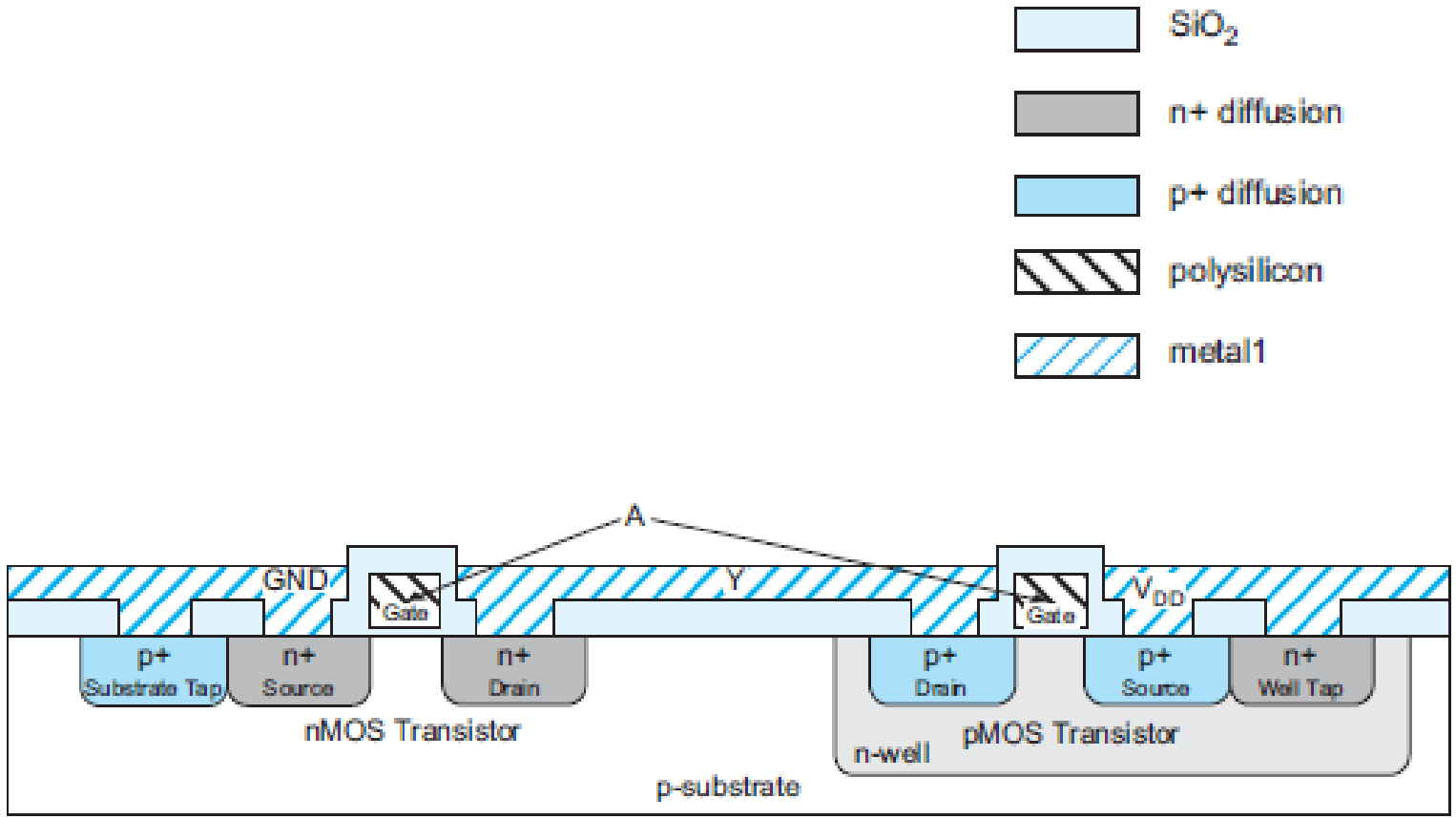
How do you design a chip, really?

Cross Section of a traditional 2D ASIC

IC Cross Section

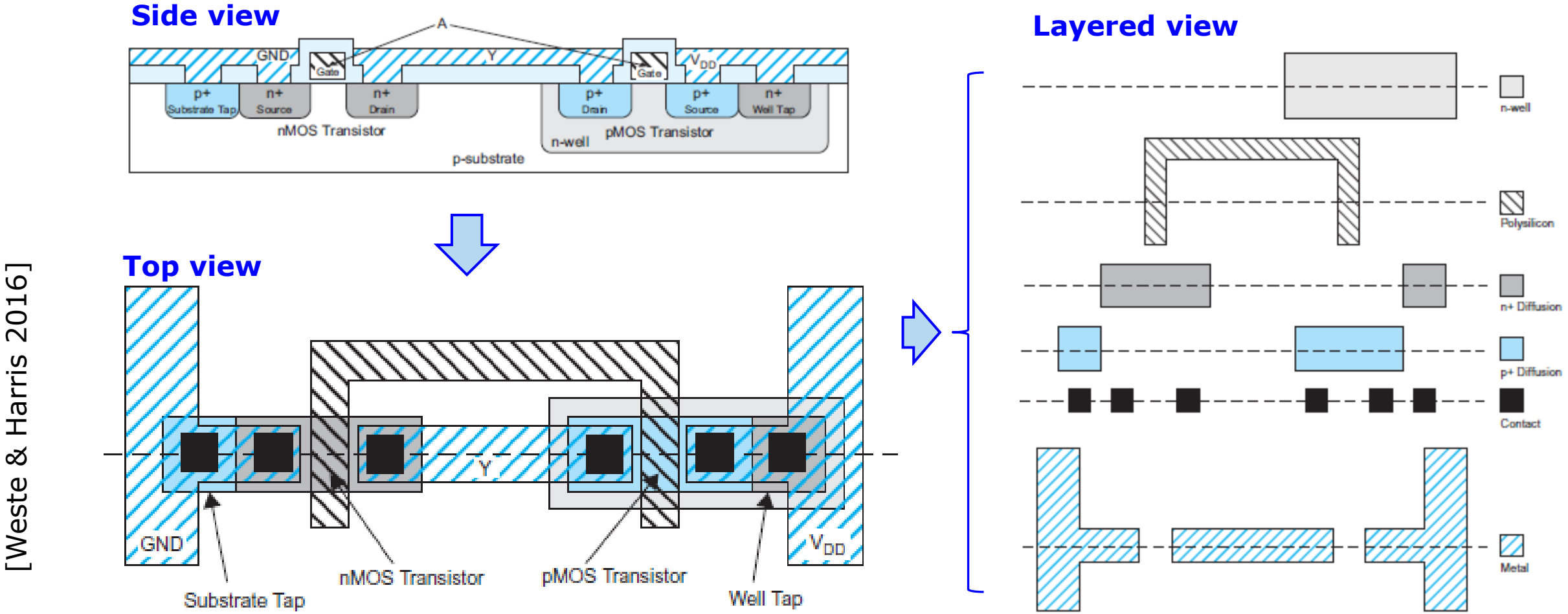


[Teshima 2014]



[Weste & Harris 2016]

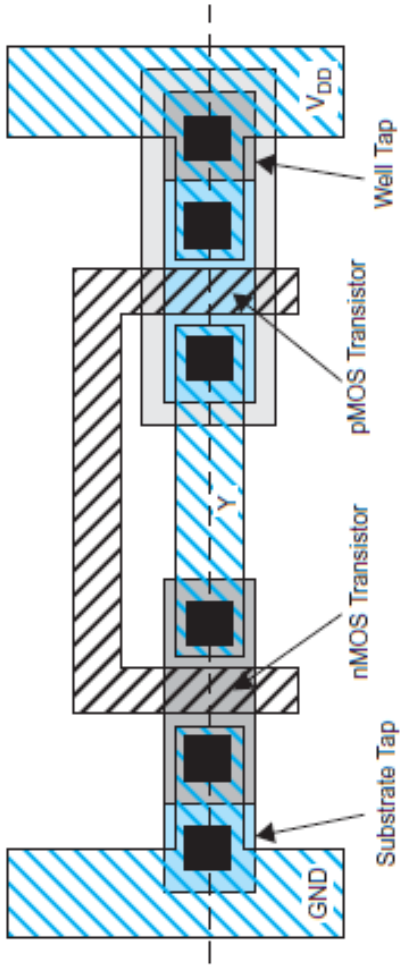
Cross Section of an Inverter

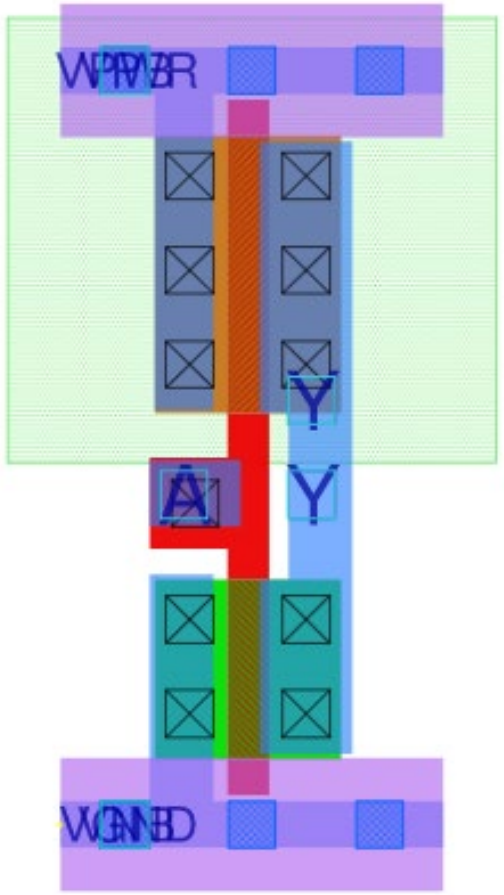


[Weste & Harris 2016]

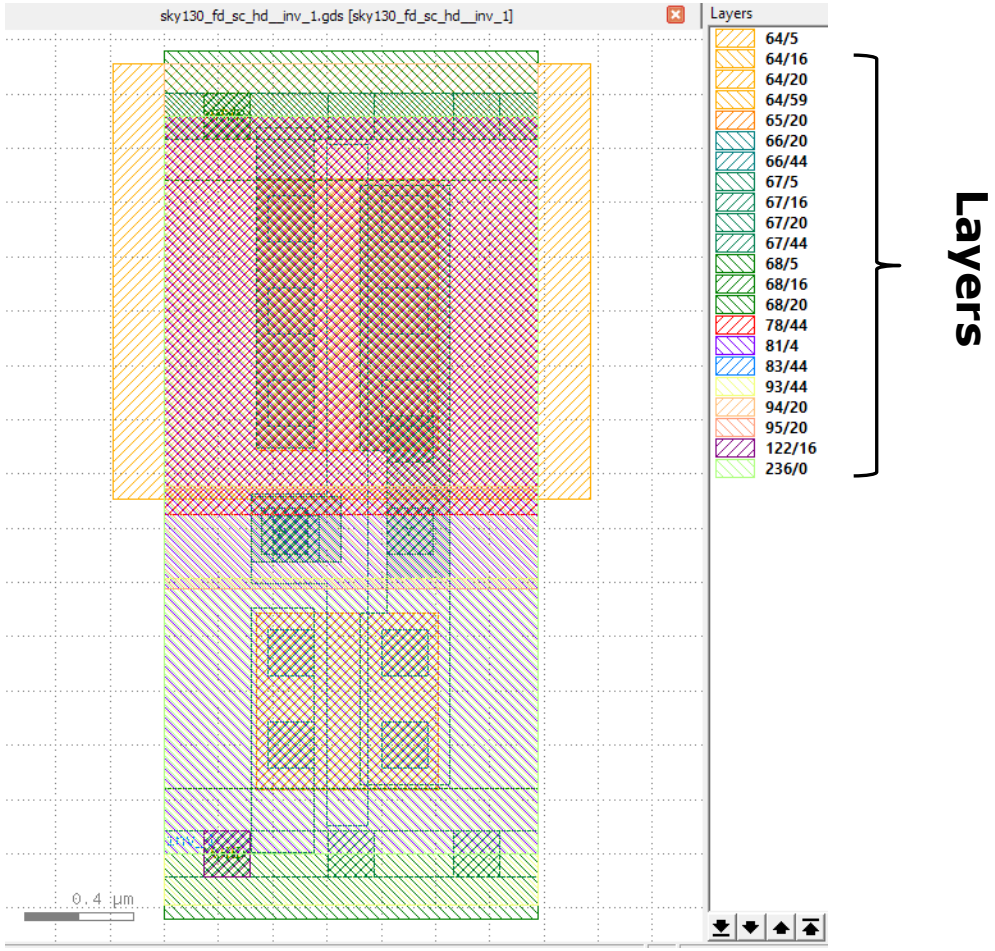
Diagram and GDS View of an Inverter

[Weste & Harris 2016]



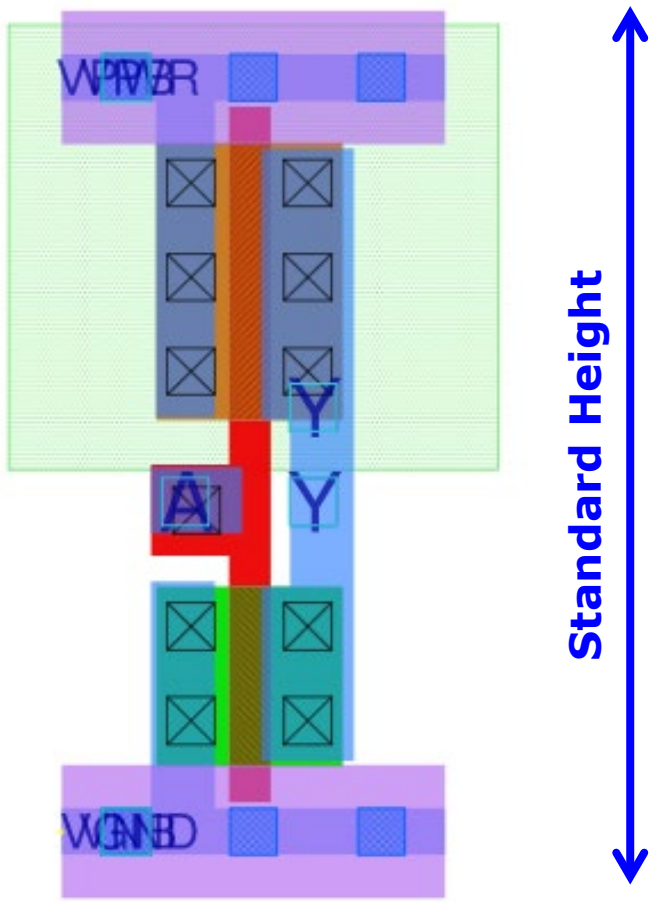


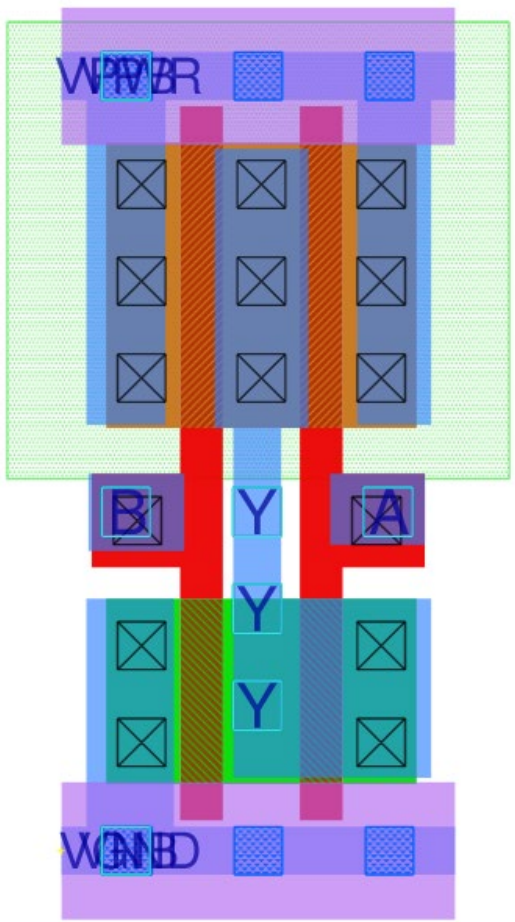
Standard Cell

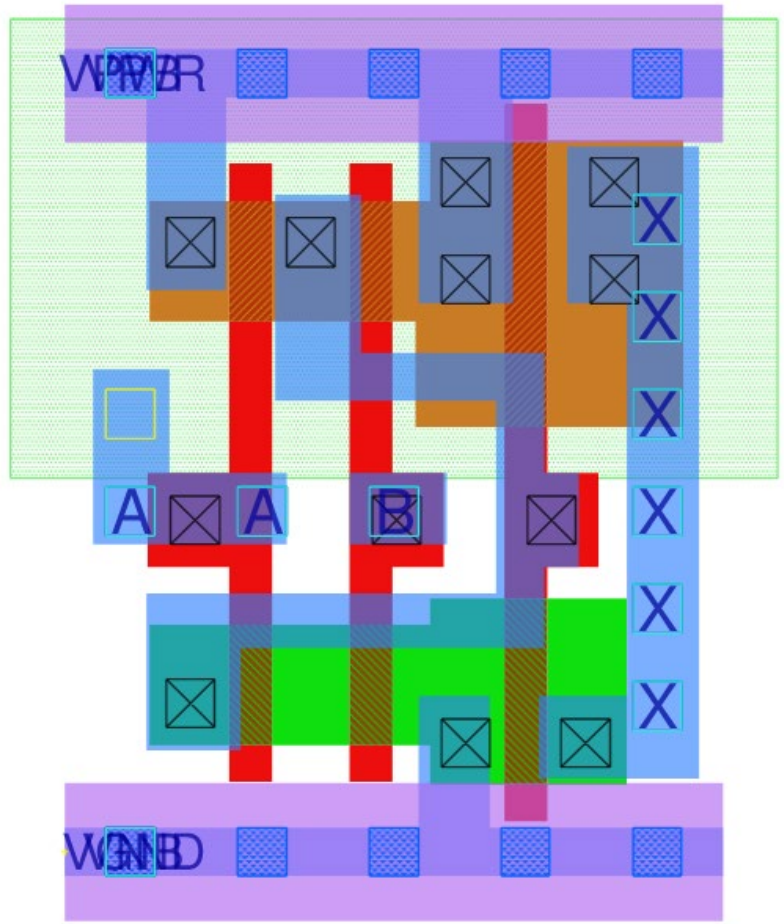


Standard Cell (GDS View)

Standard cells





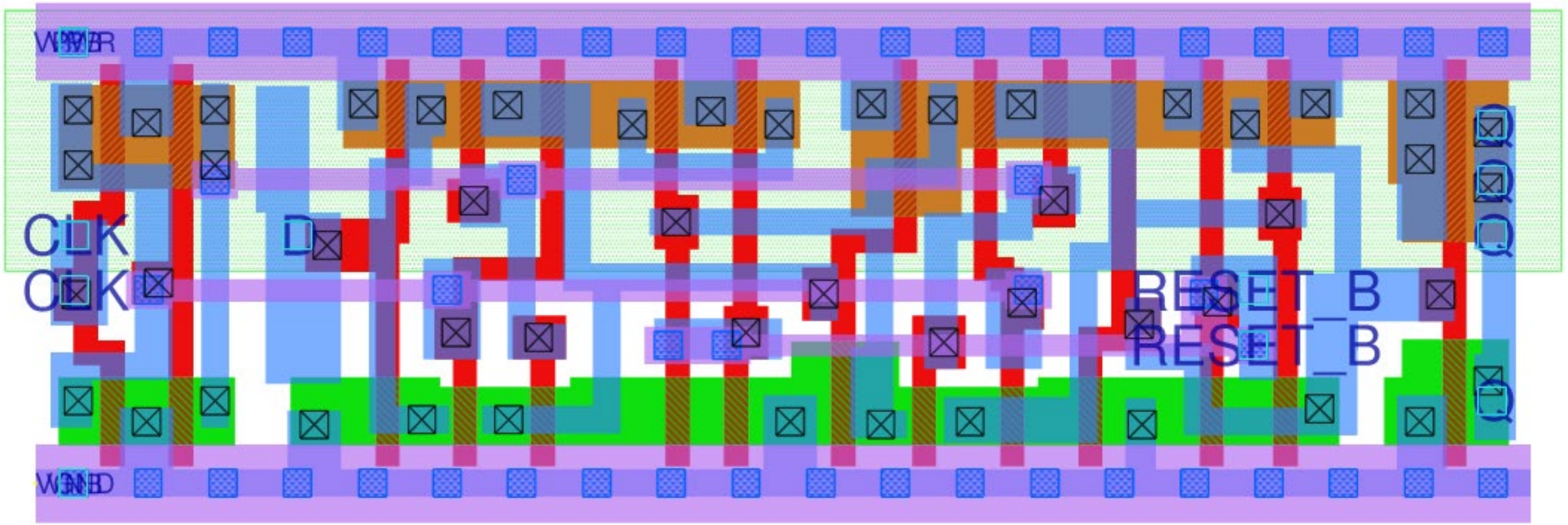


Inverter (2 transistors)

NAND (4 transistors)

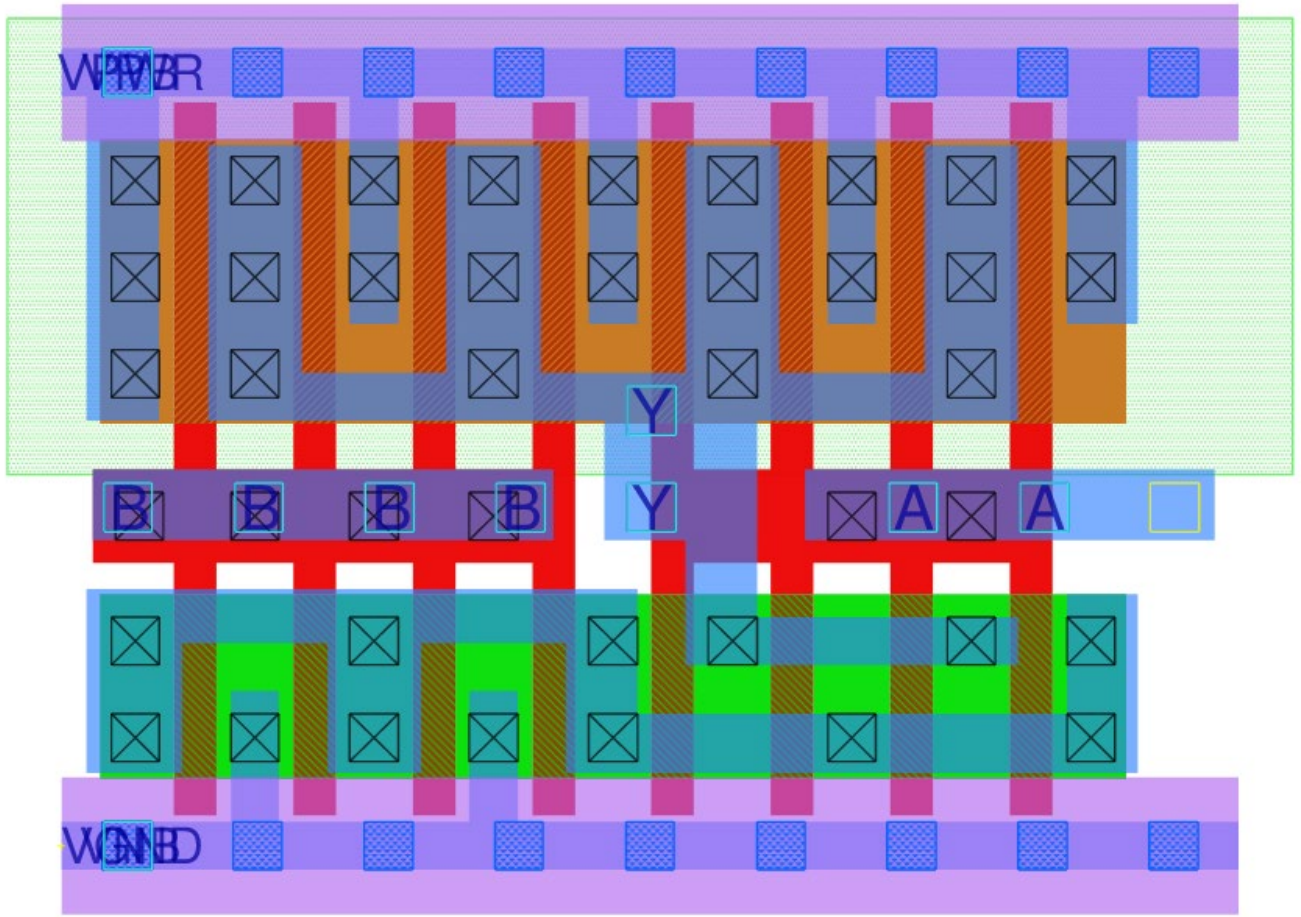
AND (6 transistors)

Standard Cells

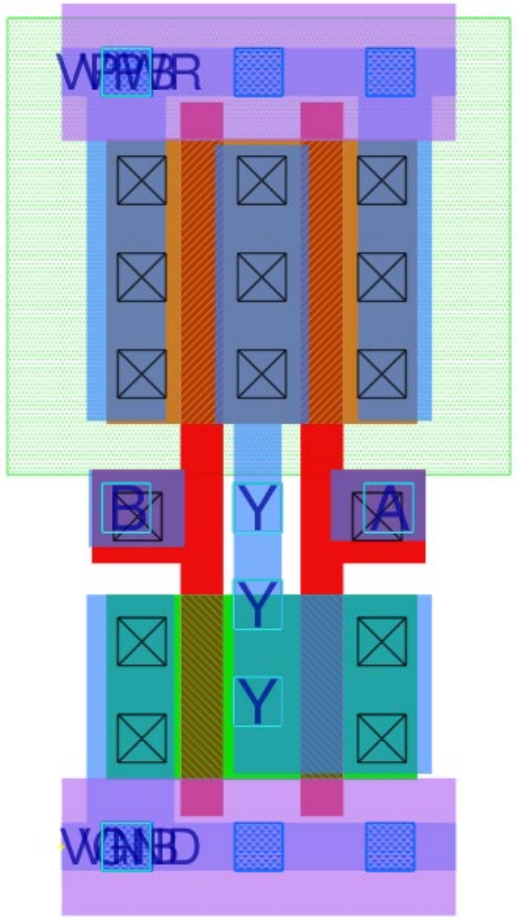


Flip-Flop (28 transistors)

Standard Cells

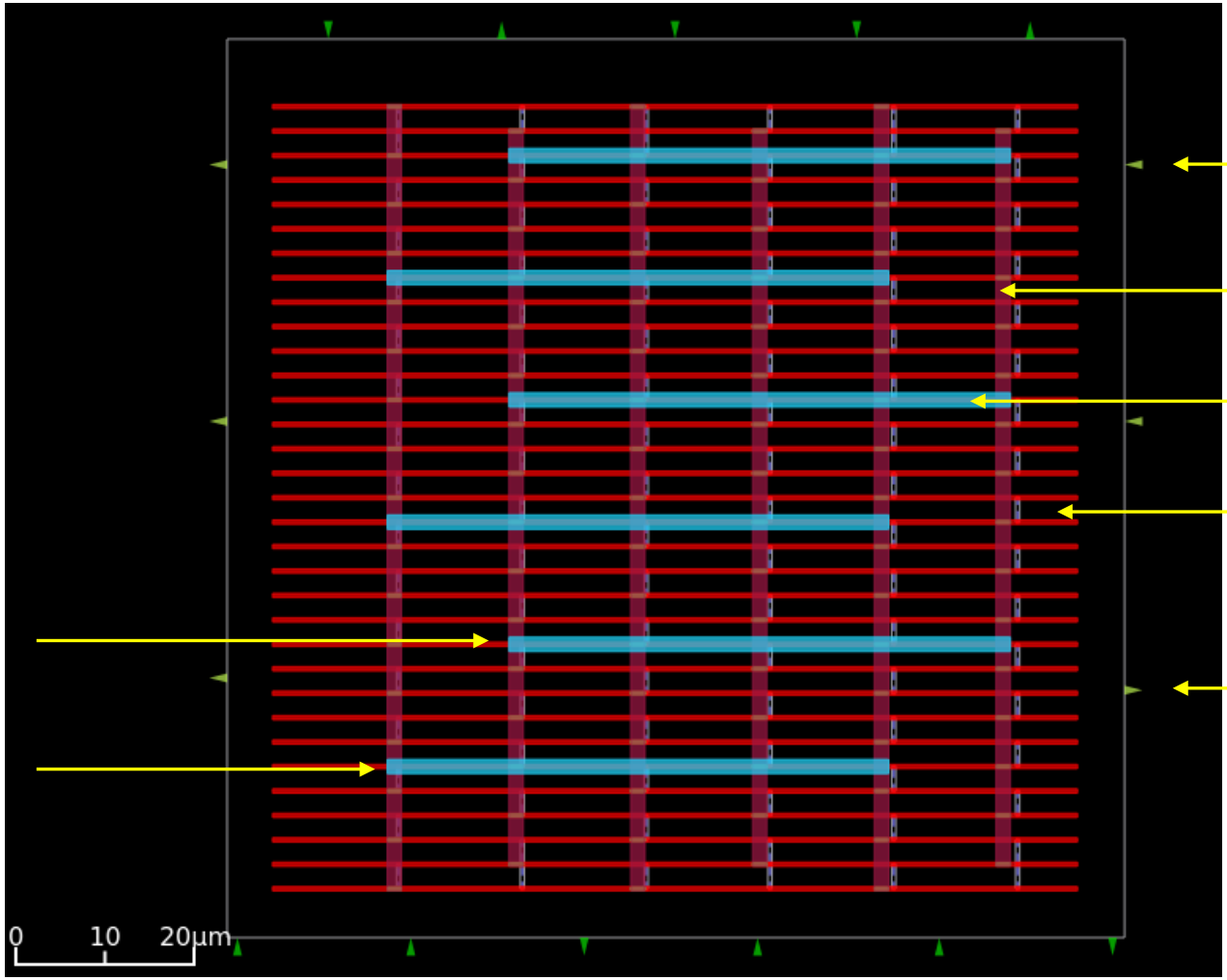


NAND with drive 4 (16 transistors)



NAND with drive 1 (4 transistors)

Standard Cells Laid Out Using A Floorplan



Input Pin

Power Strap on Metal 4

Power Strap on Metal 5

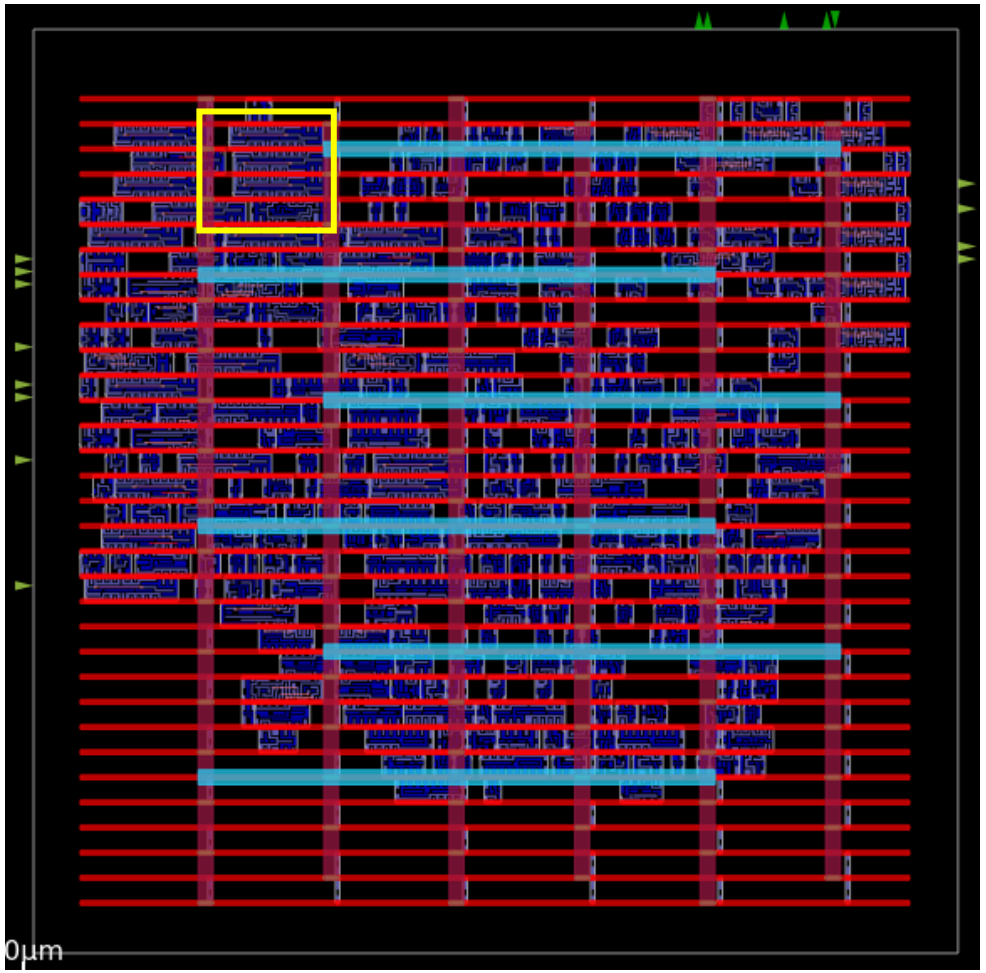
Rows to fit standard cells

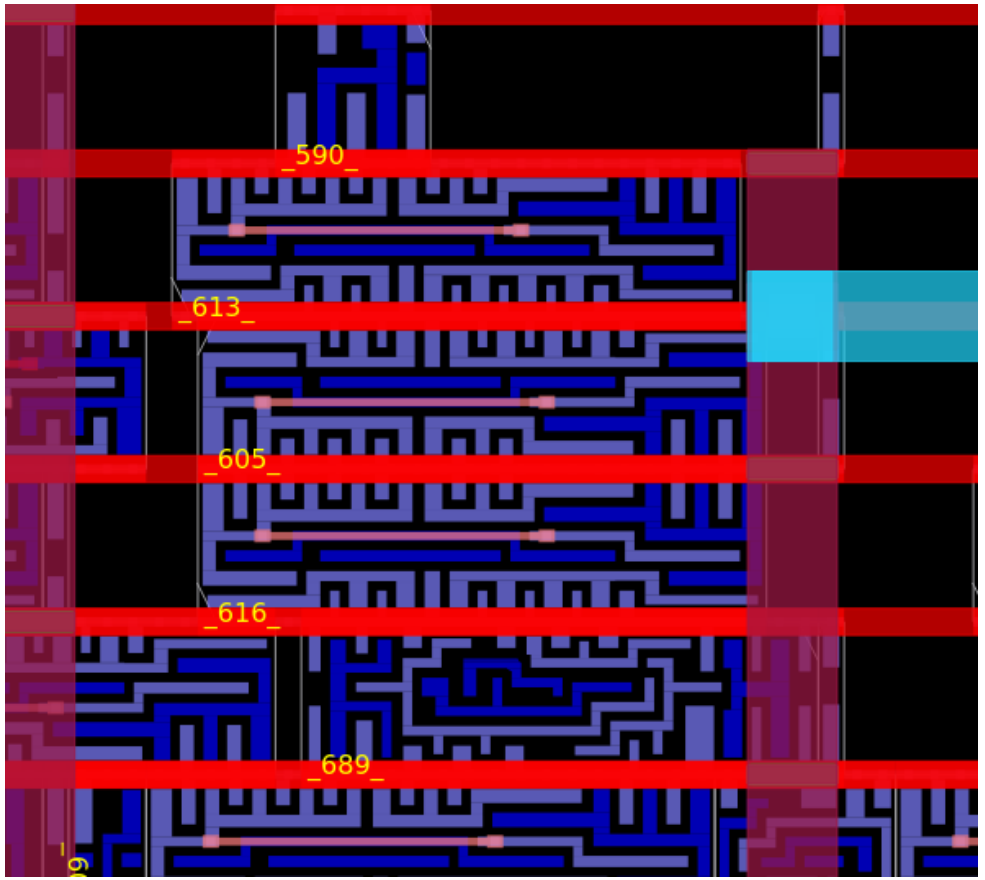
VSS Mesh

Output Pin

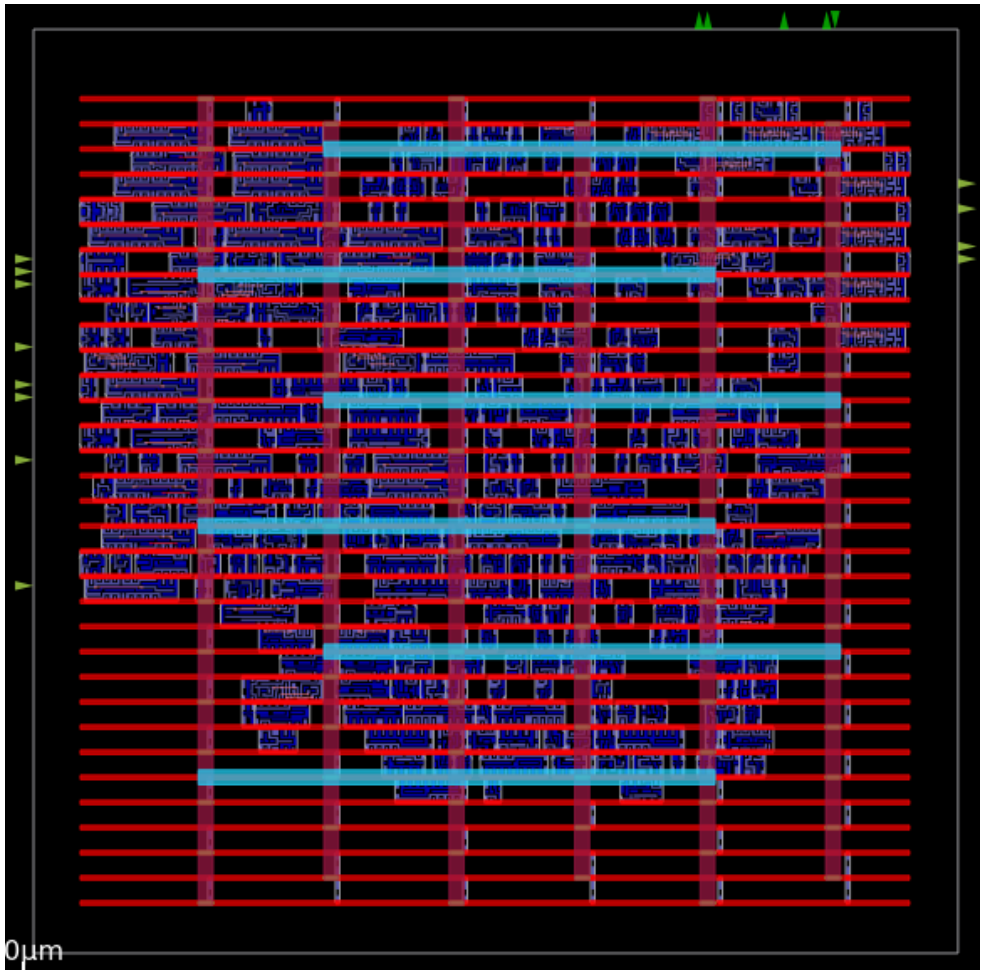
VDD Mesh

Standard Cell Placement





Utilization



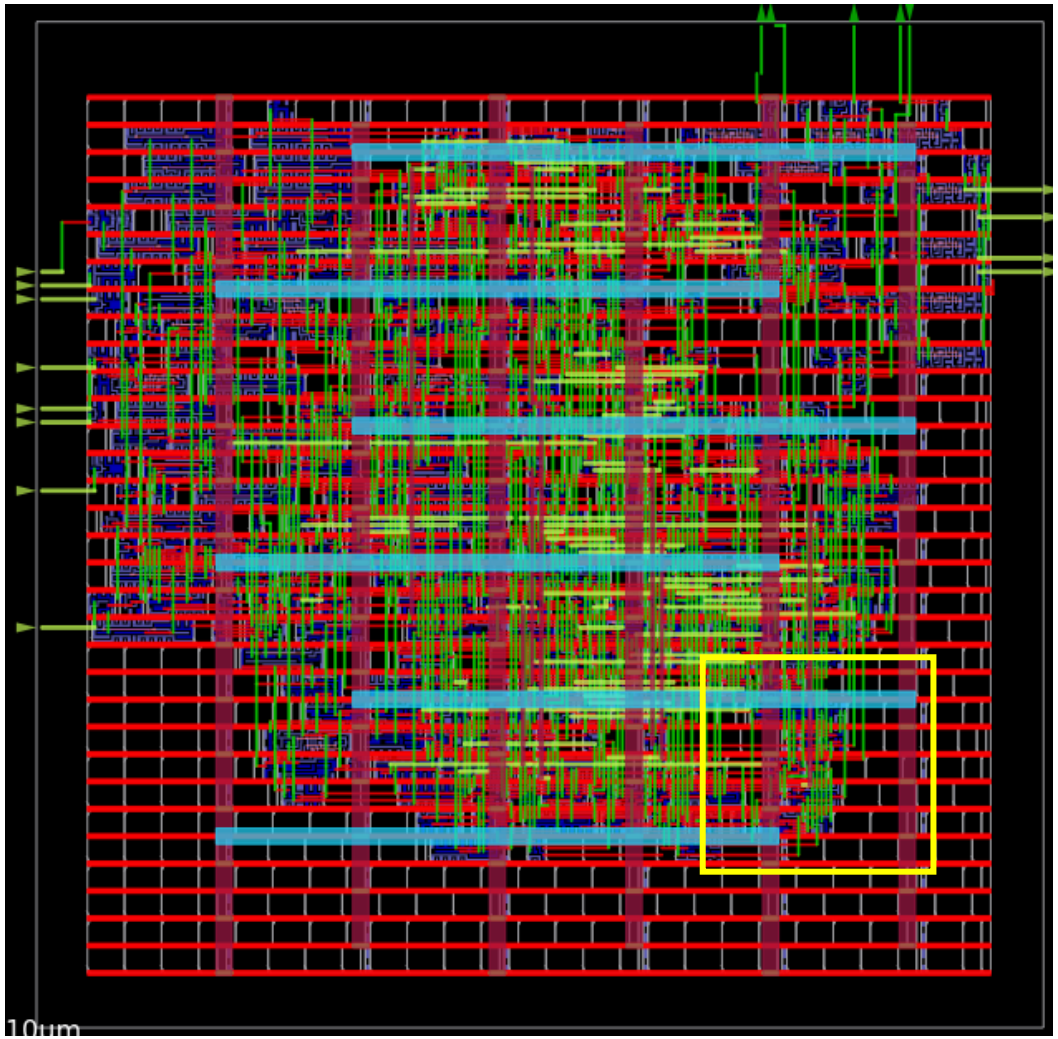
- Utilization (%)

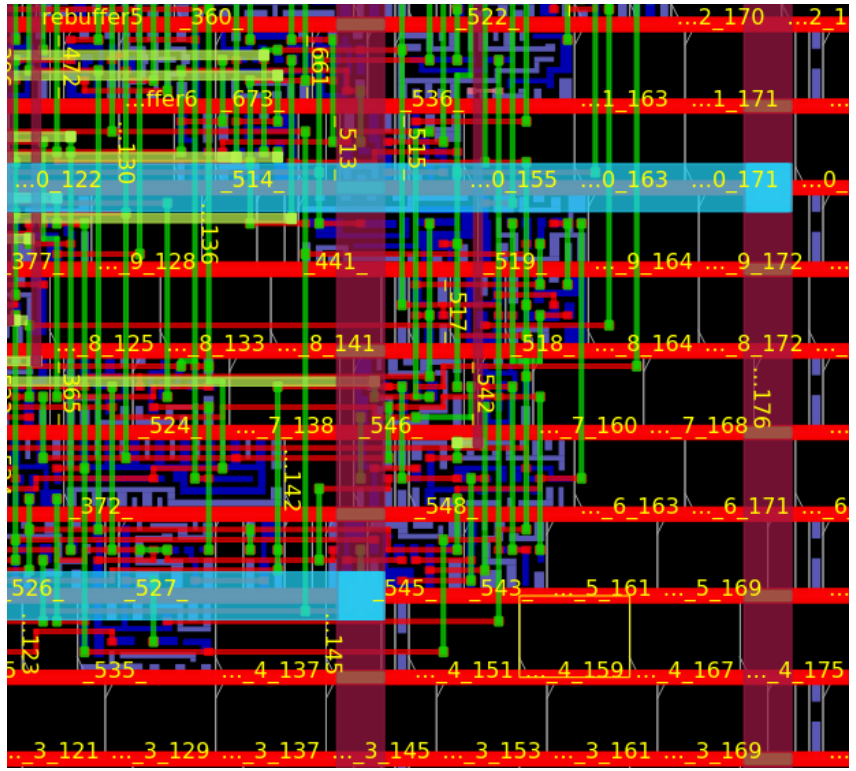
$$\frac{\text{Standard Cell Area}}{\text{Core Area}}$$

Figure has 50% utilization
Typical value is 70-80%

- Excessive utilization makes routing hard (and sometimes impossible)

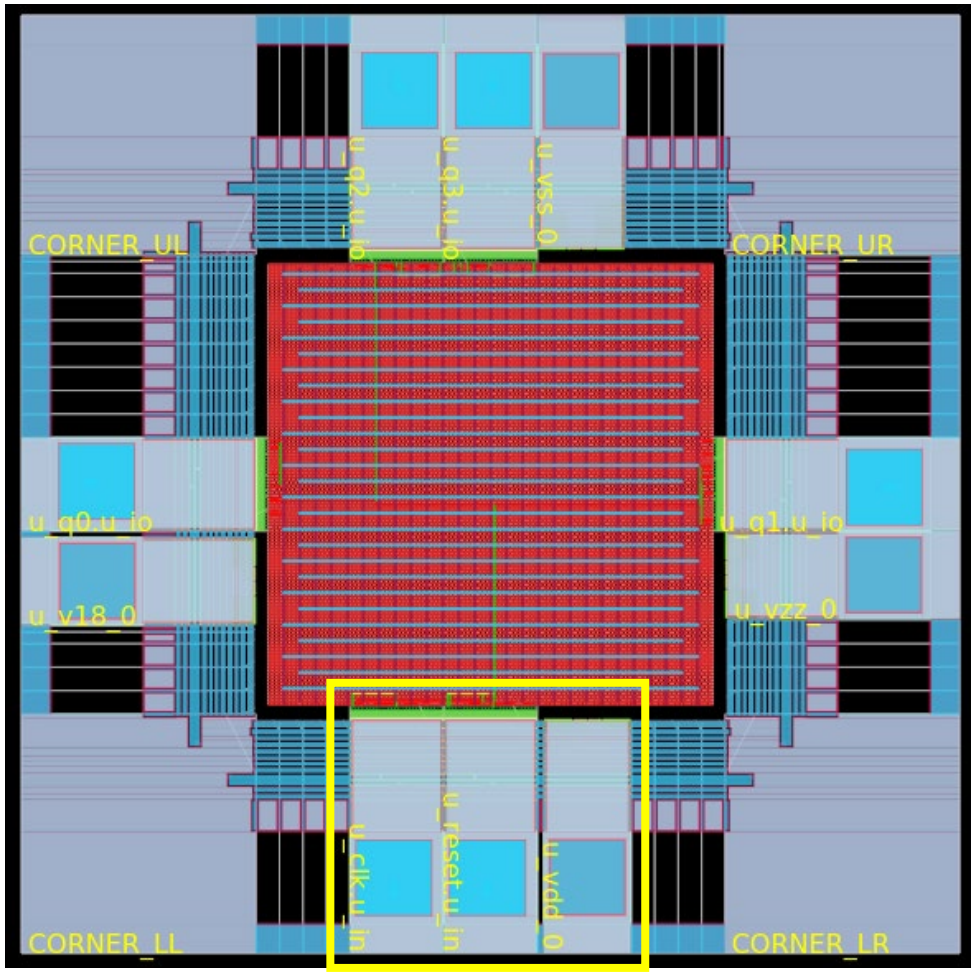
Interconnect



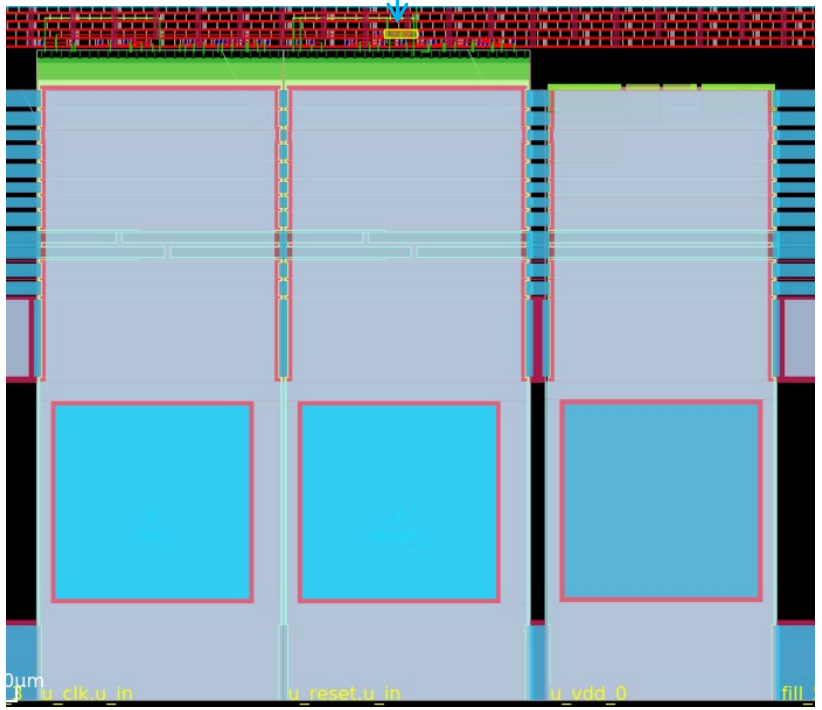


- Filler cells
- Routing congestion

Padcells

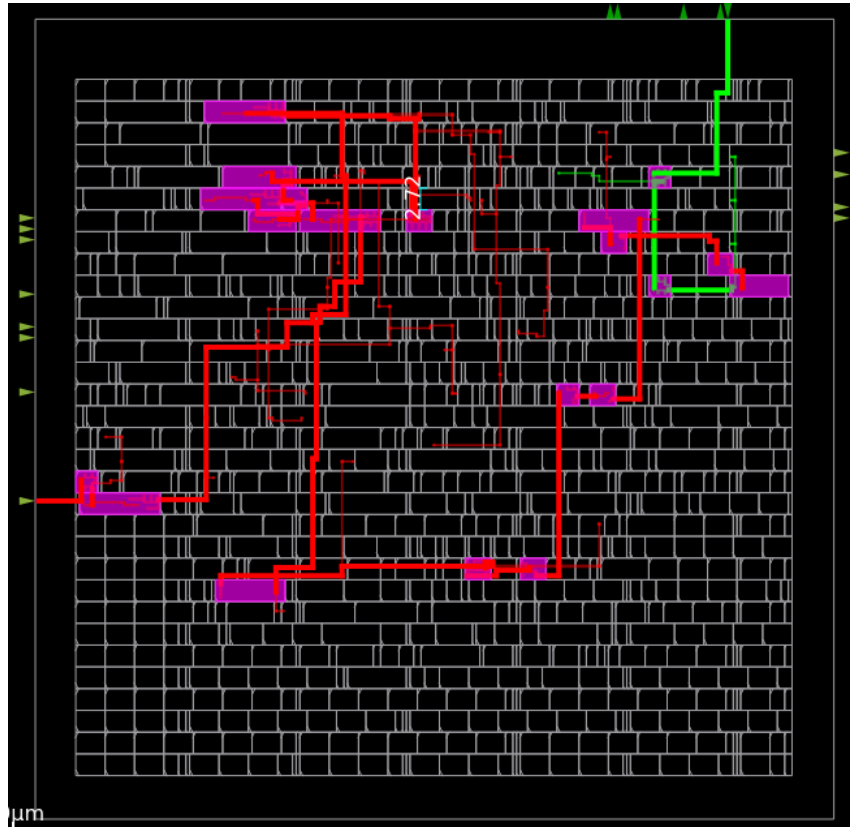


Flip-flop Standard Cell

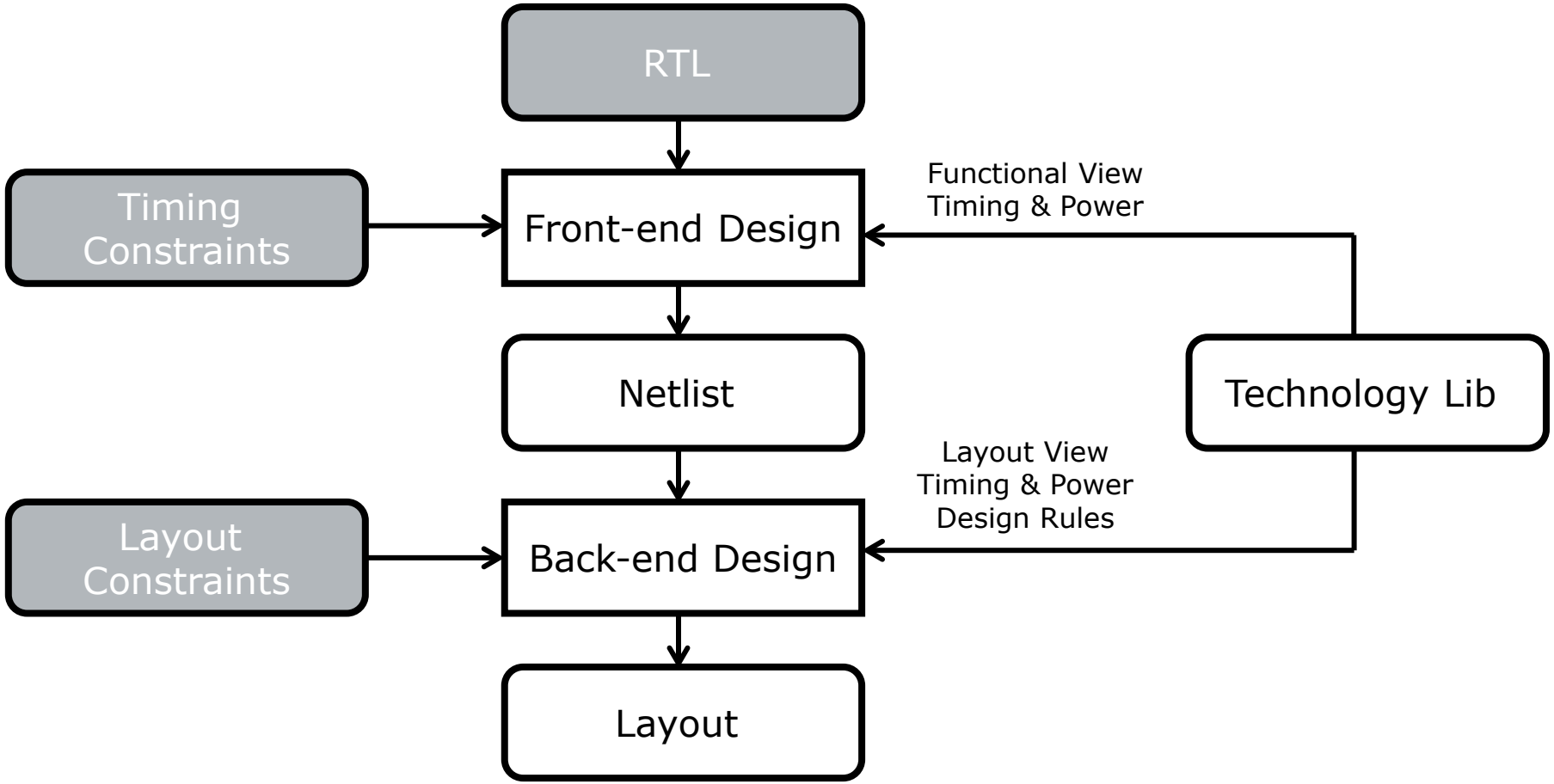


Three quality metrics of hardware

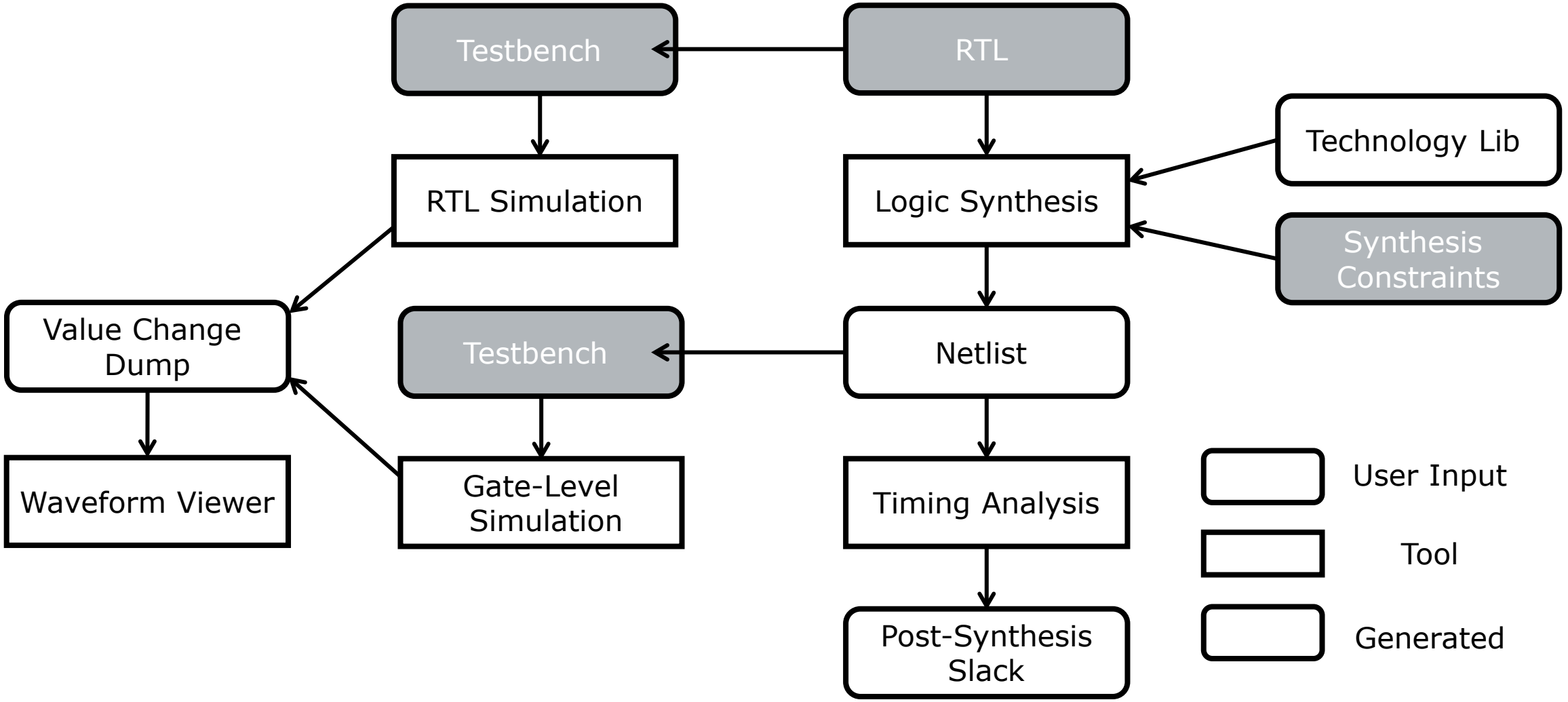
- **Area**
  - Number of logic gates
  - Area of standard cells
  - Silicon area of a placed & routed chip
- **Performance**
  - Throughput/latency of operations
  - The critical path of the chip (cfr Part II)
- **Power**
  - Average Power, Peak Power
  - Static, Dynamic (Internal + Switching) Power
- Cfr Part III



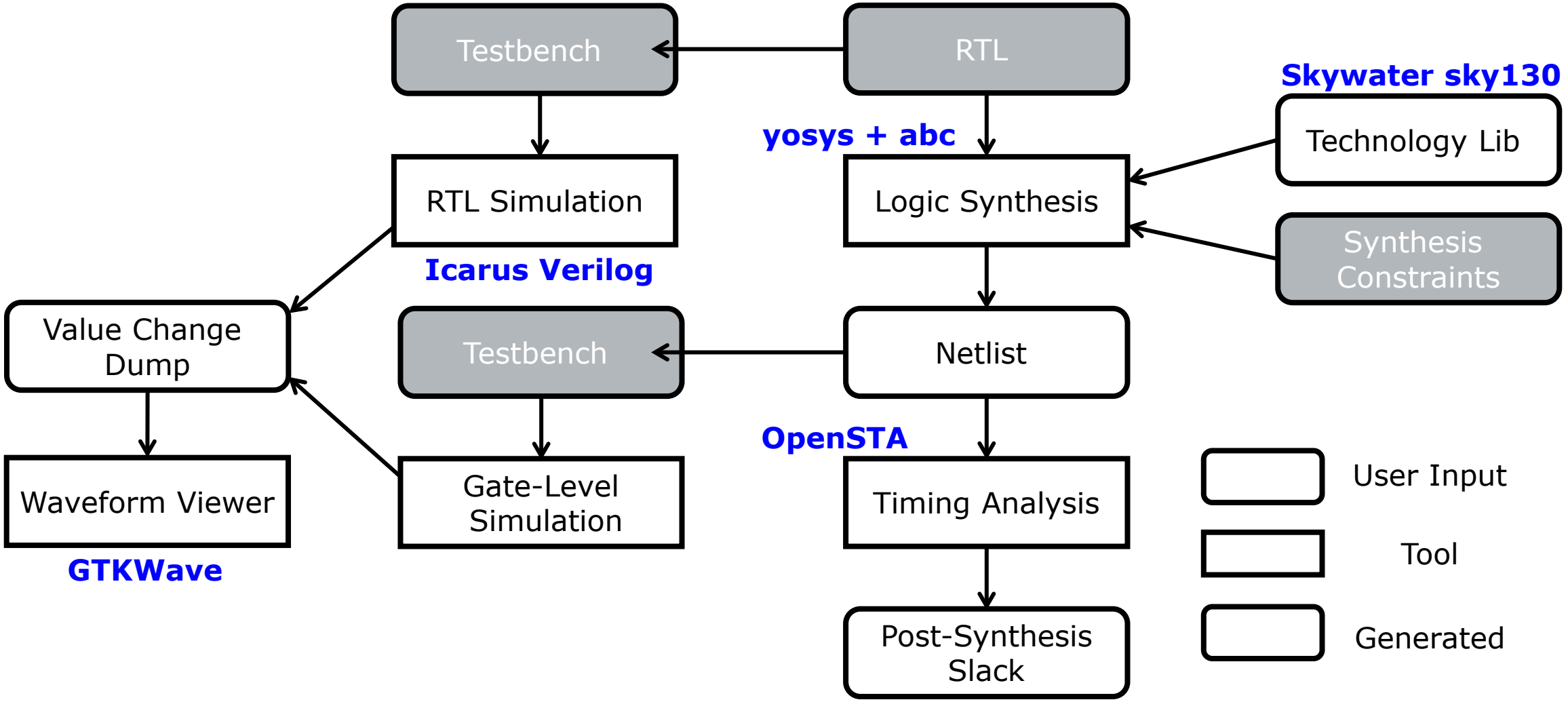
ASIC Design Flow



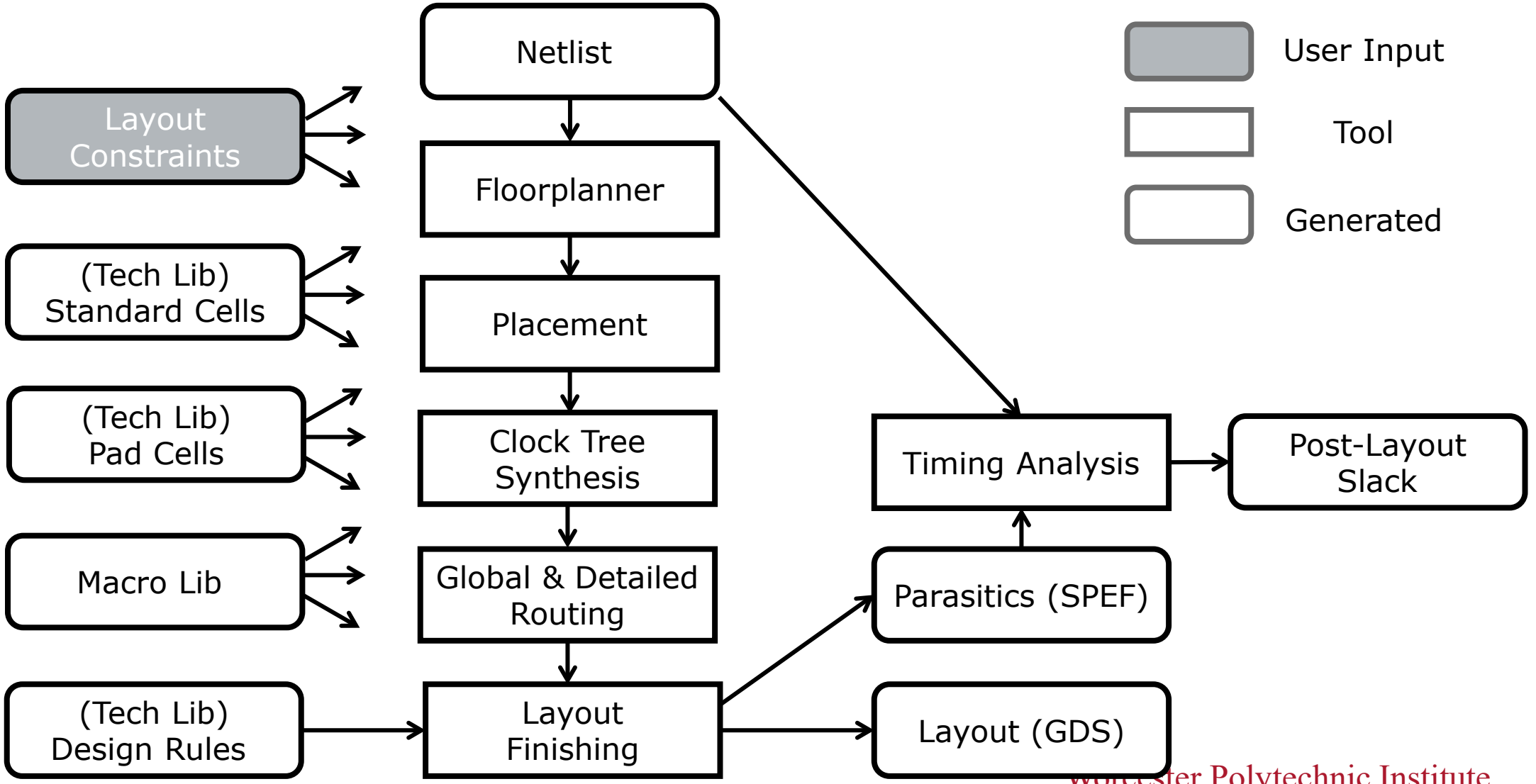
Front-end Design



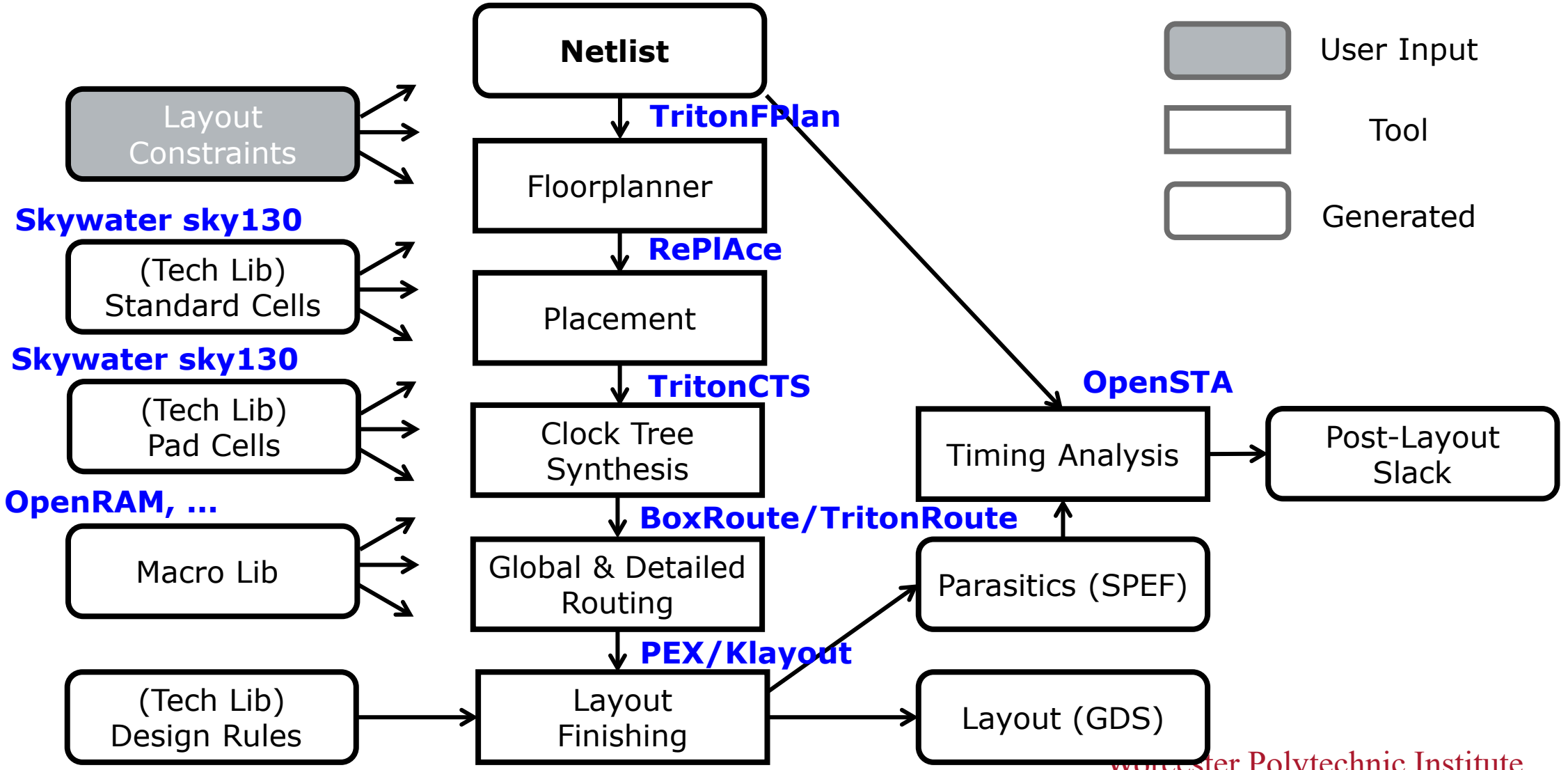
Front-end Design



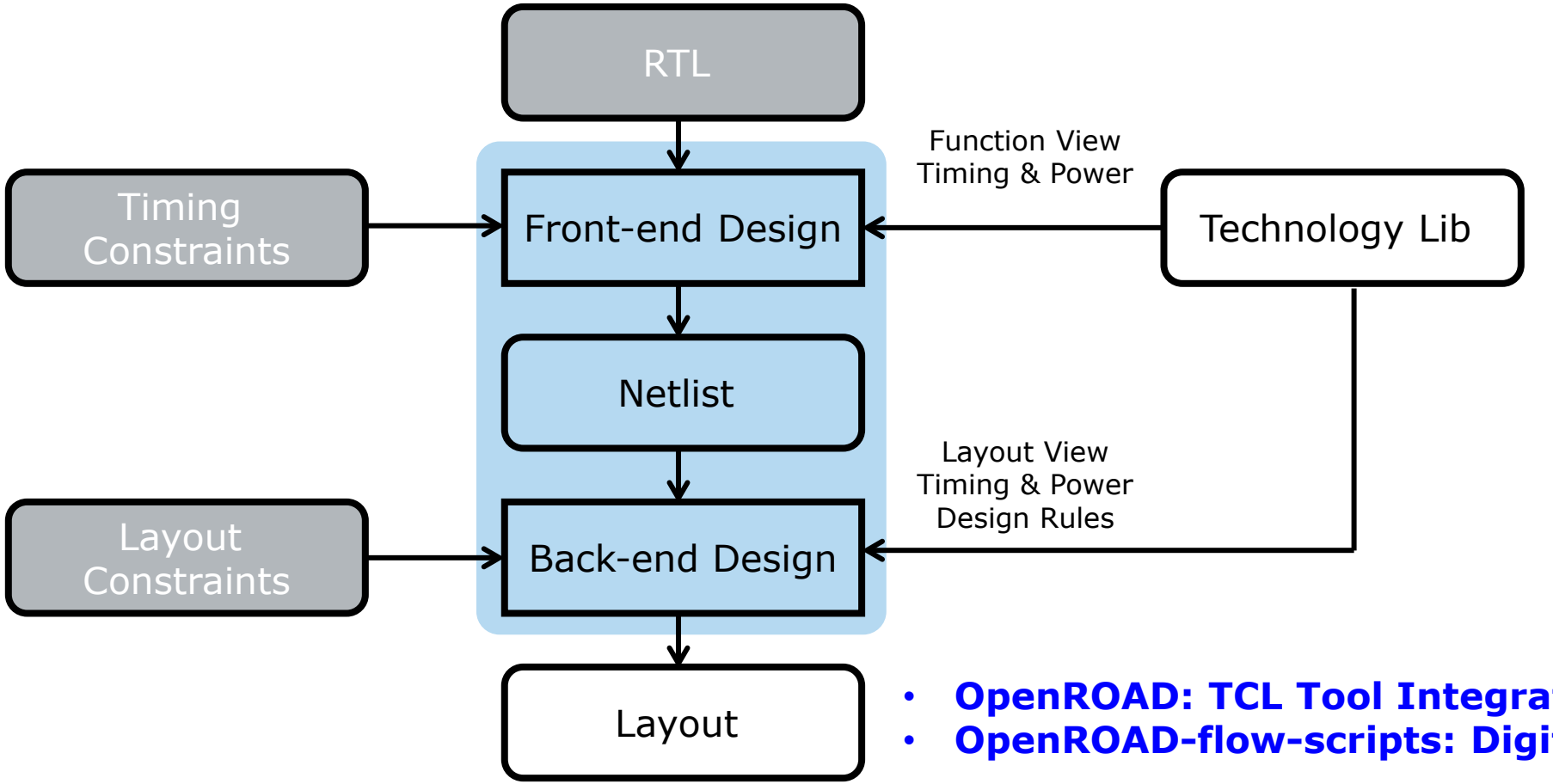
Back-end Design



Back-end Design



ASIC Design Flow



- **OpenROAD: TCL Tool Integration + GUI**
- **OpenROAD-flow-scripts: Digital Flow**

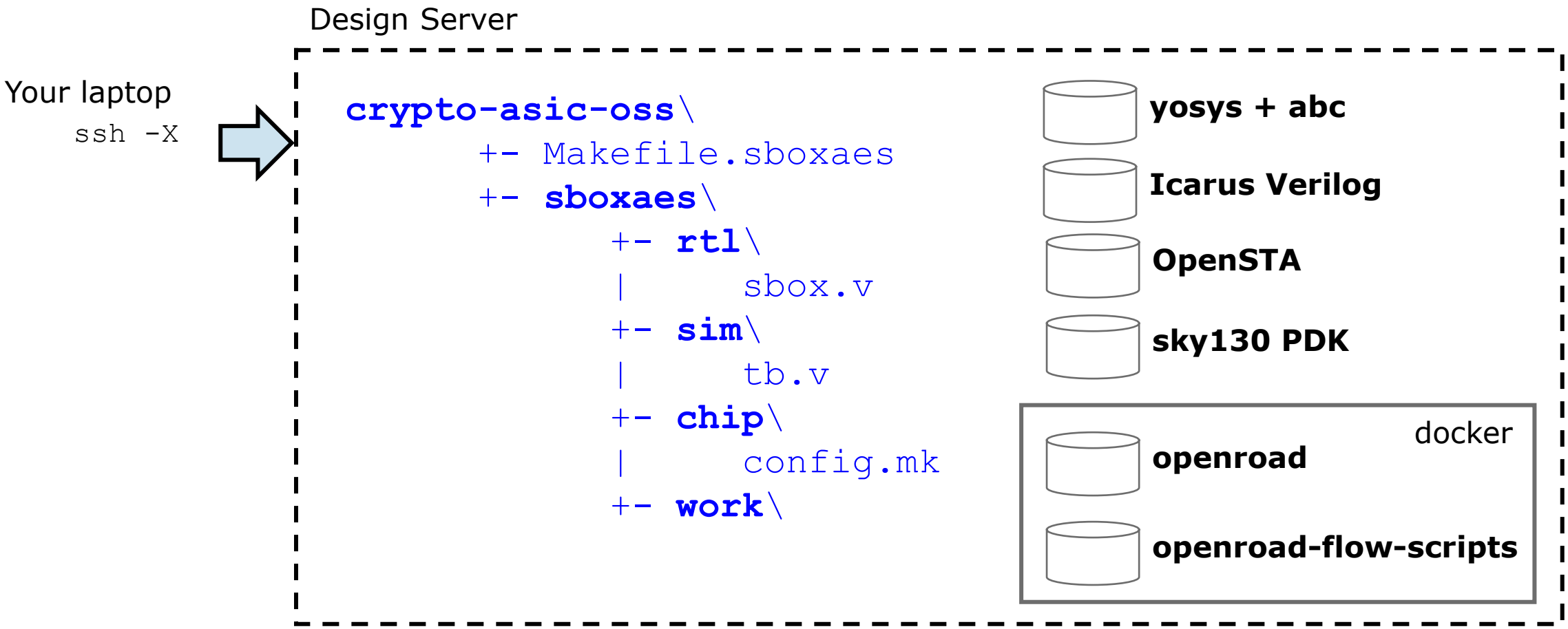
OpenROAD

- https://theopenroadproject.org
  - Started in 2018
  - Led by Andrew Kahng (UCSD), with support of other universities (https://theopenroadproject.org/our-team/)
  - Aims for a free (open-source), no-human-in-the-loop, 24-hour design from RTL to layout ('tape-out')
- Has driven an ecosystem of easily accessible chip fabrication
  - Efabless Caravel (https://caravel-harness.readthedocs.io/en/latest/)
  - Efabless OpenLane (https://github.com/efabless/openlane2)
- Similar ongoing efforts in open-source chip design
  - Silicon Compiler (https://www.siliconcompiler.com/)
  - Chipyard (https://slice.eecs.berkeley.edu/projects/chipyard/)

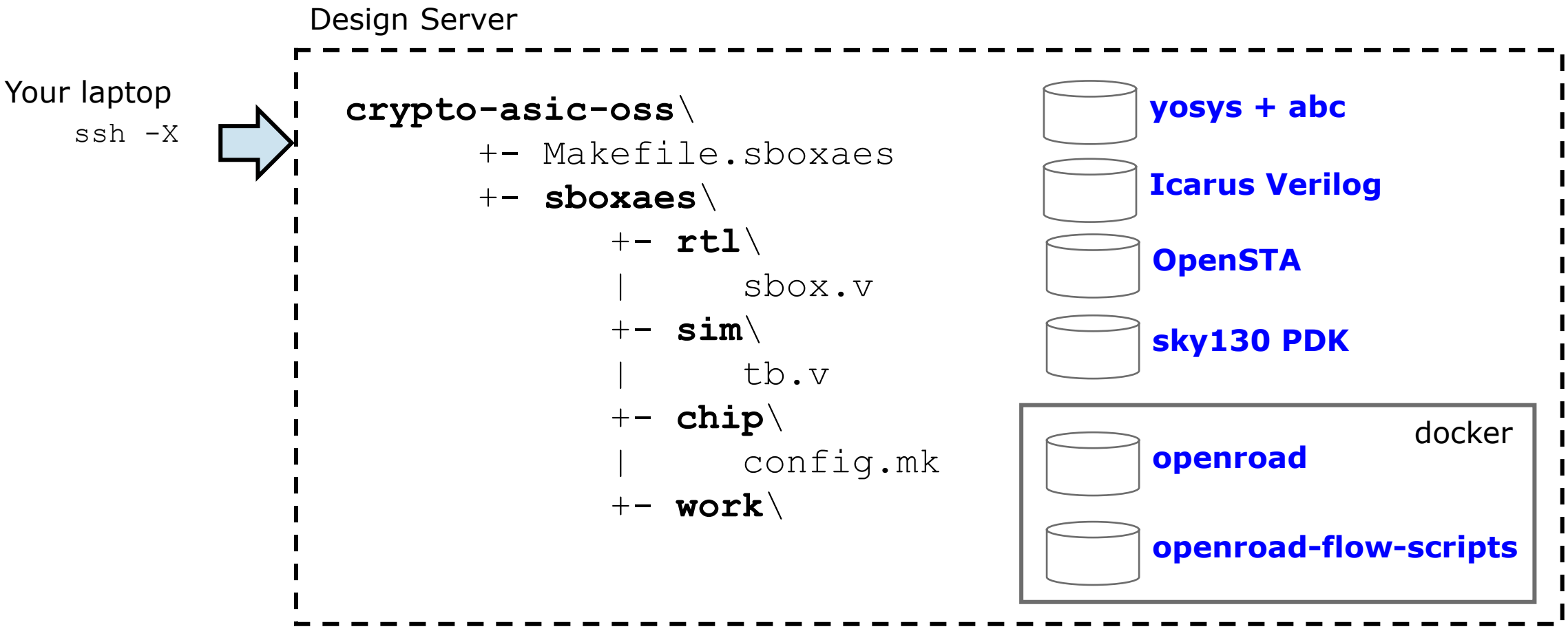
OpenROAD (2)

- The results of OpenROAD are only possible by the contributions of many people over many years in many different aspects of chip design automation
  - Icarus Verilog by Steve Icarus
  - Klayout by Matthias Köfferlein
  - yosys RTL synthesis by Claire Wolf (YosysHQ)
  - abc logic synthesis by Alan Mishchenko (UCBerkeley)
  - OpenSTA by James Cherry (Parallax Software)
  - Skywater Open Source PDK (SKywater and Google)
  - ...

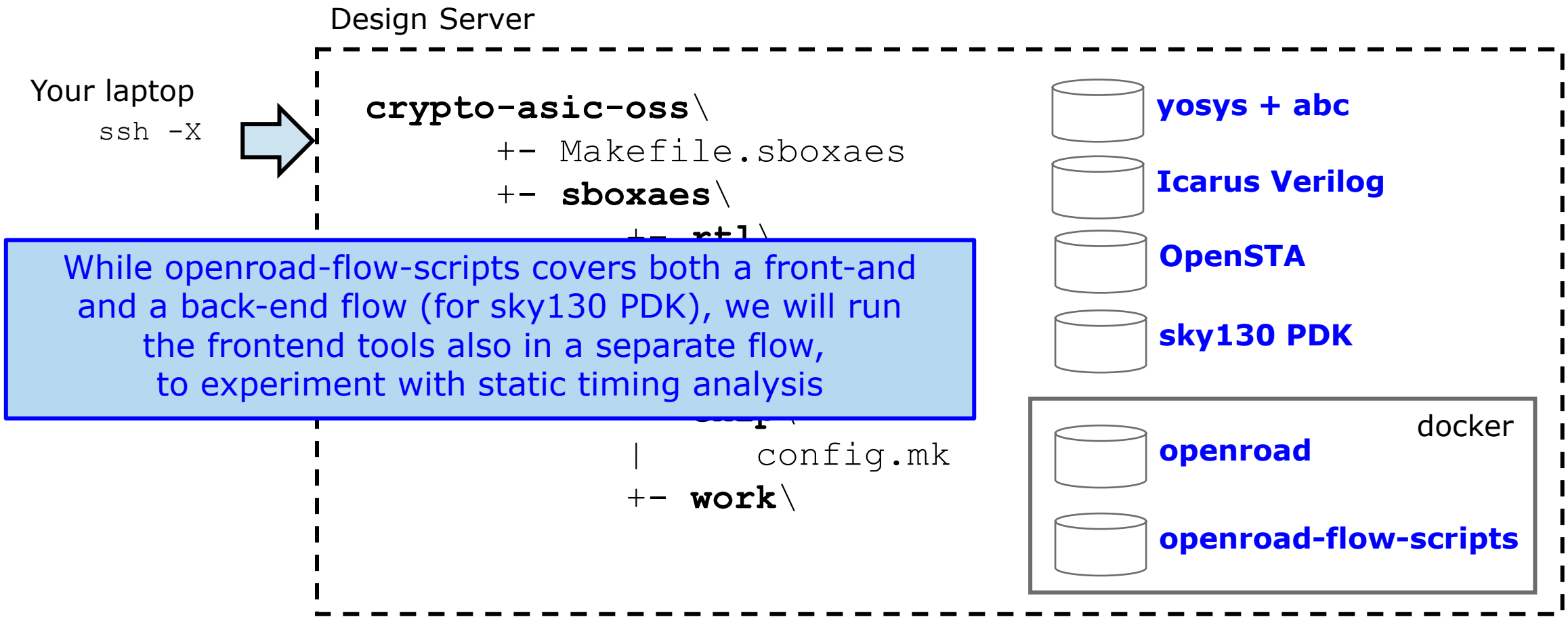
Handson Infrastructure and Tooling



Handson Infrastructure and Tooling



Handson Infrastructure and Tooling



Makefile.sboxaes

```
export FOLDER = sboxaes
export TOP = sbox
export TB = sboxtb.v
export LIB = /root/skywater-
pdk/libraries/sky130_fd_sc_hd/latest/timing/sky130_fd_sc_hd__tt_025C_1v80.lib
export CELLS = /root/skywater-pdk/libraries/sky130_fd_sc_hd/latest/cells
export CLOCK = 5
export CLOCKNAME = clk
export INPUTDELAY = 0
export OUTPUTDELAY = 0
export ABC = abc.speed
export CHIPCONFIG = config.mk

# don't touch
include make.design
```

sbox.v

```
// computational sbox
//     Johannes Wolkerstorfer, Elisabeth Oswald, Mario Lamberger:
//     An ASIC Implementation of the AES SBoxes. CT-RSA 2002: 67-78

module sbox (clk, in, out);
   input         clk;
   input  [7:0] in;
   output reg [7:0] out;

   wire [7:0] data;
   always @(posedge clk)
     out <= data;

   cbox cbox1(.address(in), .data(data));
endmodule

...
```

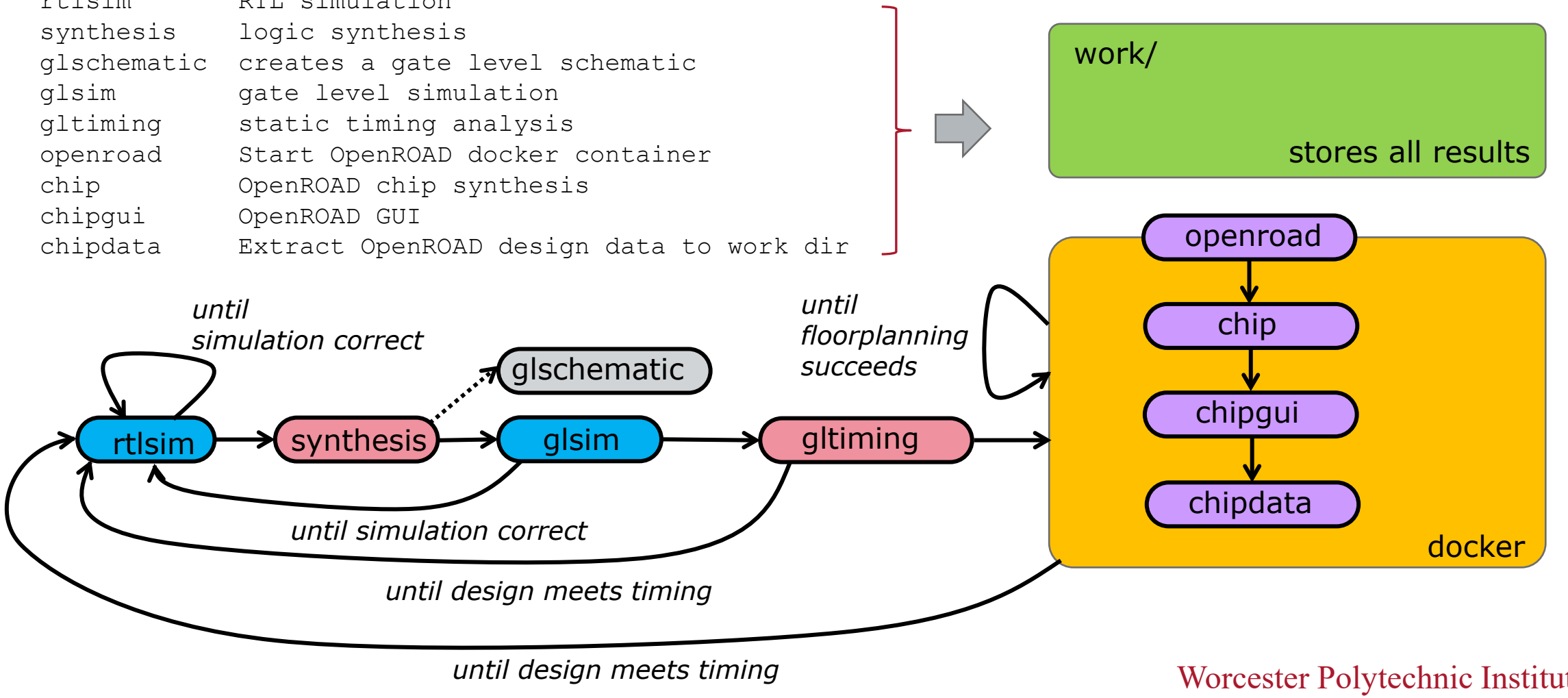
sboxtb.v

```
`timescale 1ns/1ps

module toptb();
   reg  [7:0]  in;
   wire [7:0] out;
   reg        clk;

   sbox dut(.in(in),
            .out(out),
            .clk(clk));

   always begin
      clk = 1'b0;
      #5 clk = 1'b1;
      #5;
   end
```

```
   initial
     begin
        $dumpfile("trace.vcd");
        $dumpvars(0, toptb);

        in = 8'b0;
        @(posedge clk); #1;
        repeat (256)
          begin
             $display("%x -> %x", in, out);
             in = in + 8'b1;
             @(posedge clk); #1;
          end

        $finish;
   end

endmodule
```

Makefile Targets and Use

```
# make -f Makefile.sboxaes
Targets are:
   rtlsim        RTL simulation
   synthesis     logic synthesis
   glschematic   creates a gate level schematic
   glsim         gate level simulation
   gltiming      static timing analysis
   openroad      Start OpenROAD docker container
   chip          OpenROAD chip synthesis
   chipgui       OpenROAD GUI
   chipdata      Extract OpenROAD design data to work dir
```



RTL Simulation

```
# make -f Makefile.sboxaes rtlsim
cd sboxaes/work && make rtlsim
make[1]: Entering directory '/root/crypto-asic-oss/sboxaes/work'
iverilog -y ../rtl ../sim/sboxtb.v
./a.out && mv trace.vcd rtl.vcd && rm -f a.out
VCD info: dumpfile trace.vcd opened for output.
00 -> 63
01 -> 7c
02 -> 77
03 -> 7b
04 -> f2
05 -> 6b
...
```

Yosys Synthesis

```
# make -f Makefile.sboxaes synthesis
test -d sboxaes/work || mkdir -p sboxaes/work
python3 scripts/gen_make_design.py
cd sboxaes/work && make synthesis
make[1]: Entering directory '/root/crypto-asic-oss/sboxaes/work'
yosys -s synth.ys

 /----------------------------------------------------------------------------\
 |                                                                            |
 |  yosys -- Yosys Open SYnthesis Suite                                       |
 |                                                                            |
 |  Copyright (C) 2012 - 2019  Clifford Wolf <clifford@clifford.at>           |
...
```

Yosys Synthesis

```
10. Printing statistics.

=== sbox ===

   Number of wires:                353
   Number of wire bits:            367
   Number of public wires:          11
   Number of public wire bits:      25
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                358
     sky130_fd_sc_hd__a2111oi_0      2
     sky130_fd_sc_hd__a211o_1        2
     sky130_fd_sc_hd__a211oi_1       4
...
```

Yosys Synthesis Script

- The synthesis script influences the quality of result
- Yosys is timing agnostic
  - Timing constraints are handled through logic synthesis tool step in **abc**

synth.ys

```
read_liberty -lib /root/skywater-
pdk/libraries/sky130_fd_sc_hd/latest/timing/sky130_fd_sc_hd__tt_025C_1v80.lib
read_verilog ../rtl/sbox.v
synth -top sbox
flatten
opt -purge
dfflibmap -liberty /root/skywater-
pdk/libraries/sky130_fd_sc_hd/latest/timing/sky130_fd_sc_hd__tt_025C_1v80.lib
abc -D 10000 -script ../../scripts/abc.speed -liberty /root/skywater-
pdk/libraries/sky130_fd_sc_hd/latest/timing/sky130_fd_sc_hd__tt_025C_1v80.lib
setundef -zero
opt_clean -purge
stat -liberty /root/skywater-
pdk/libraries/sky130_fd_sc_hd/latest/timing/sky130_fd_sc_hd__tt_025C_1v80.lib
write_verilog netlist.v
write_json netlist.json
```

work/netlist.v

```
/* Generated by Yosys 0.9 (git sha1
1979e0b) */

(* top =  1  *)
(* src = "../rtl/sbox.v:5" *)
module sbox(clk, in, out);
  wire _000_;
  wire _001_;
  wire _002_;
  wire _003_;
..
  sky130_fd_sc_hd__nand3_1 _378_ (
    .A(_291_),
    .B(_293_),
    .C(_294_),
    .Y(_295_)
  );
..
```

```
...
  sky130_fd_sc_hd__dfxtp_1 _693_ (
    .CLK(clk),
    .D(\cbox1.md.q1 ),
    .Q(out[1])
  );
...
```

Making sense of gates and nets is difficult ..

To find the correspondence between RTL and gate-level, look for flip-flops (dfxtp cells), names of RTL registers, and names of primary input/outputs

Gate Level Simulation

```
# make -f Makefile.sboxaes glsim
cd sboxaes/work && make glsim
make[1]: Entering directory '/root/crypto-asic-oss/sboxaes/work'
..
iverilog -DFUNCTIONAL -c lib.cmd netlist.v ../sim/sboxtb.v
..
./a.out && mv trace.vcd netlist.vcd && rm -f a.out
VCD info: dumpfile trace.vcd opened for output.
00 -> 63
01 -> 7c
02 -> 77
03 -> 7b
04 -> f2
05 -> 6b
06 -> 6f
07 -> c5
```

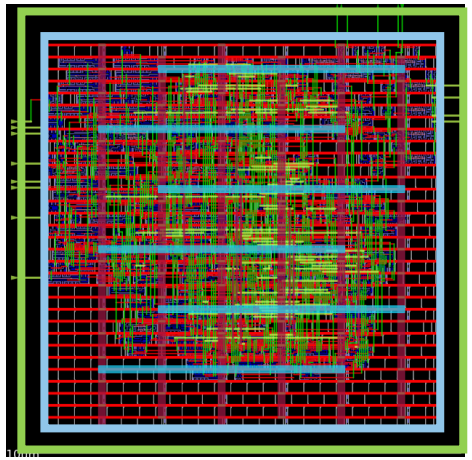
Icarus Verilog Gate Level Simulation of SKY130 is a functional simulation and ignores timing (pure cycle-accurate, no glitches)

- To perform timing-accurate gate-level simulation, use Modelsim, VCS, ...
- To verify the timing of your design, use Static Timing Analysis

Making a Chip with OpenROAD

- chip/config.mk

```
export DESIGN_NAME = sbox
export PLATFORM    = sky130hd

export VERILOG_FILES = ./crypto-asic-oss/sboxaes/rtl/sbox.v
export SDC_FILE      = ./crypto-asic-
oss/sboxaes/work/constraint.sdc

export DIE_AREA    =   0   0  100  100
export CORE_AREA   =   5   5   95   95
```



Starting the docker container

```
# make -f Makefile.sboxaes openroad
cd sboxaes/work && make openroad
make[1]: Entering directory '/root/crypto-asic-oss/sboxaes/work'
xhost +; docker run --rm -it \
-u : \
--network=host --env DISPLAY=localhost:10.0 \
--privileged \
--workdir=/OpenROAD-flow-scripts/flow/crypto-asic-oss \
--volume="/root/.Xauthority:/root/.Xauthority:rw" \
-v /usr/share/X11/xkb:/usr/share/X11/xkb \
-v /root/crypto-asic-oss:/OpenROAD-flow-scripts/flow/crypto-asic-oss \
crypto-asic-oss
access control disabled, clients can connect from any host
```

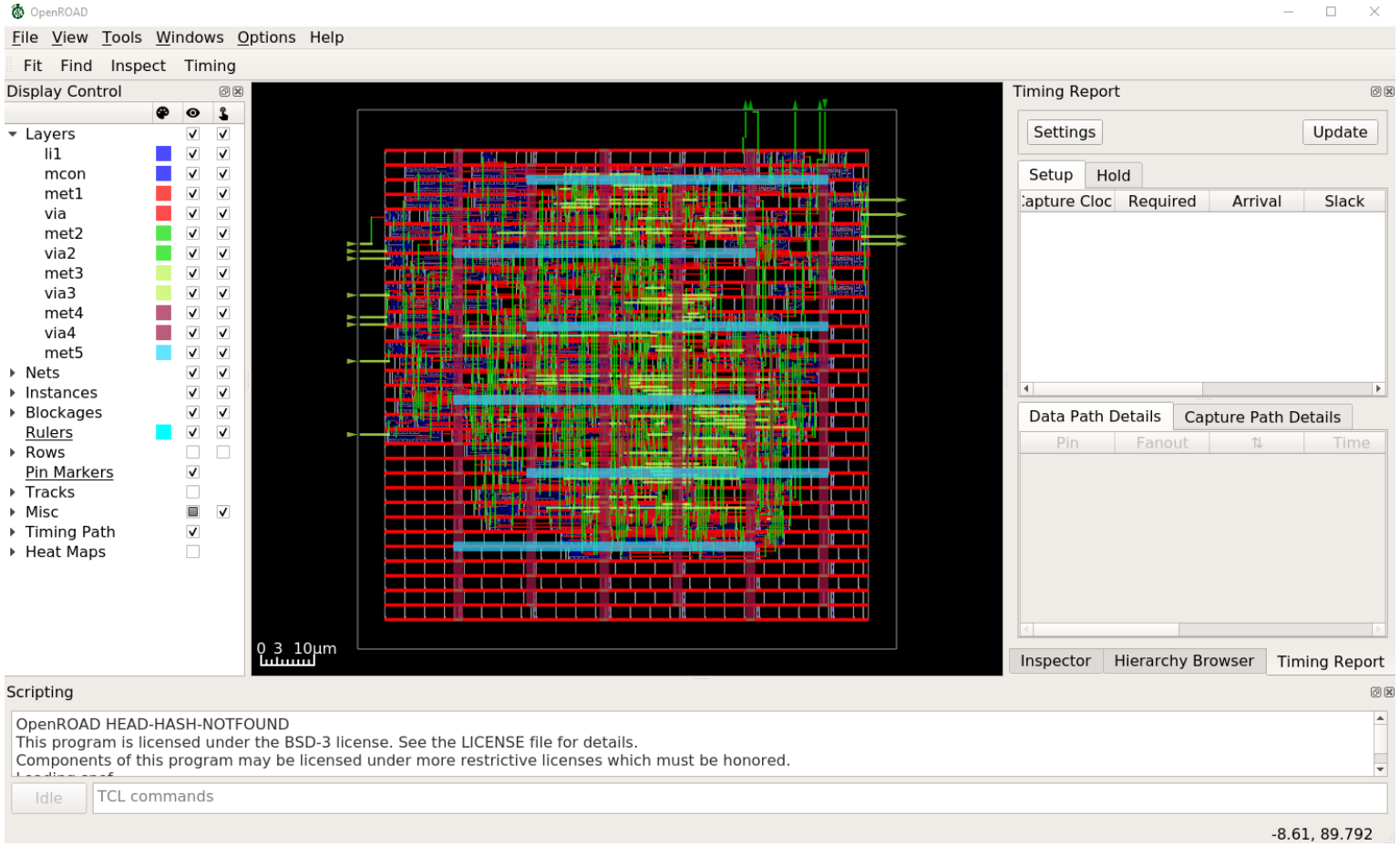
Making the chip

```
# make -f Makefile.sboxaes chip
cd sboxaes/work && make chip
make[1]: Entering directory `/OpenROAD-flow-scripts/flow/crypto-asic-
oss/sboxaes/work'
...
ln -sf 6_1_merged.gds results/sky130hd/sbox/base/6_final.gds
Log                            Elapsed seconds
1_1_yosys                                    4
3_3_place_gp                                 2
3_4_resizer                                  1
4_1_cts                                      6
5_2_TritonRoute                             85
6_1_merge                                    2
6_report                                    19
make[2]: Leaving directory `/OpenROAD-flow-scripts/flow'
```

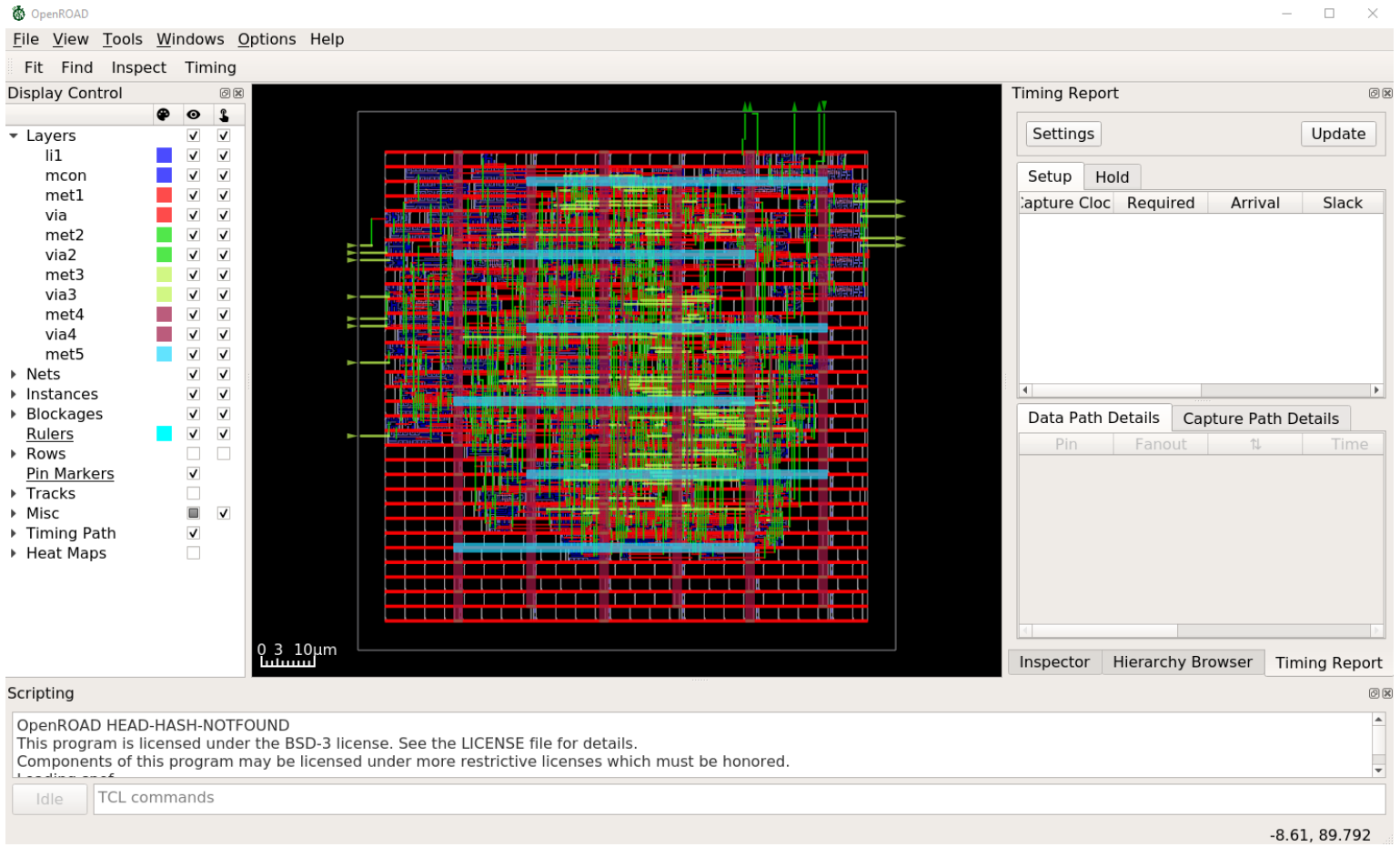
Quick when the design is very small

Looking at the chip

```
# make -f Makefile.sboxaes chipgui
```



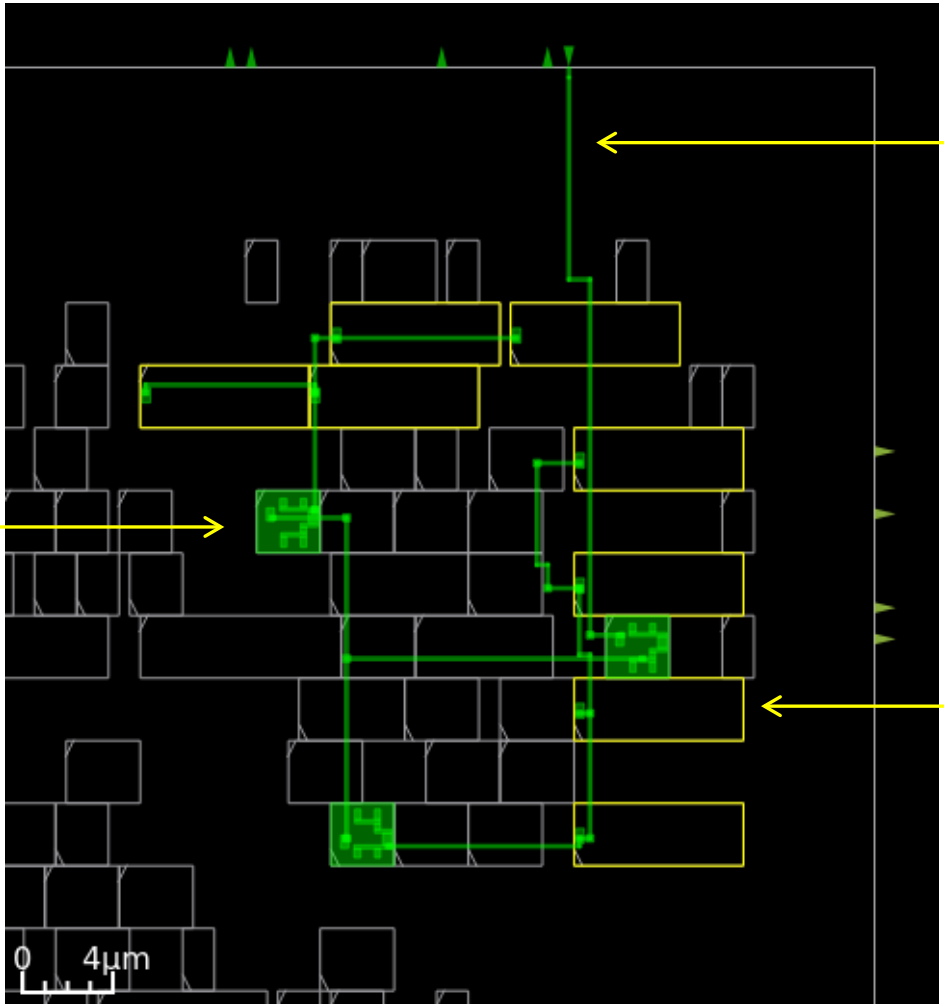
Exploring the GUI



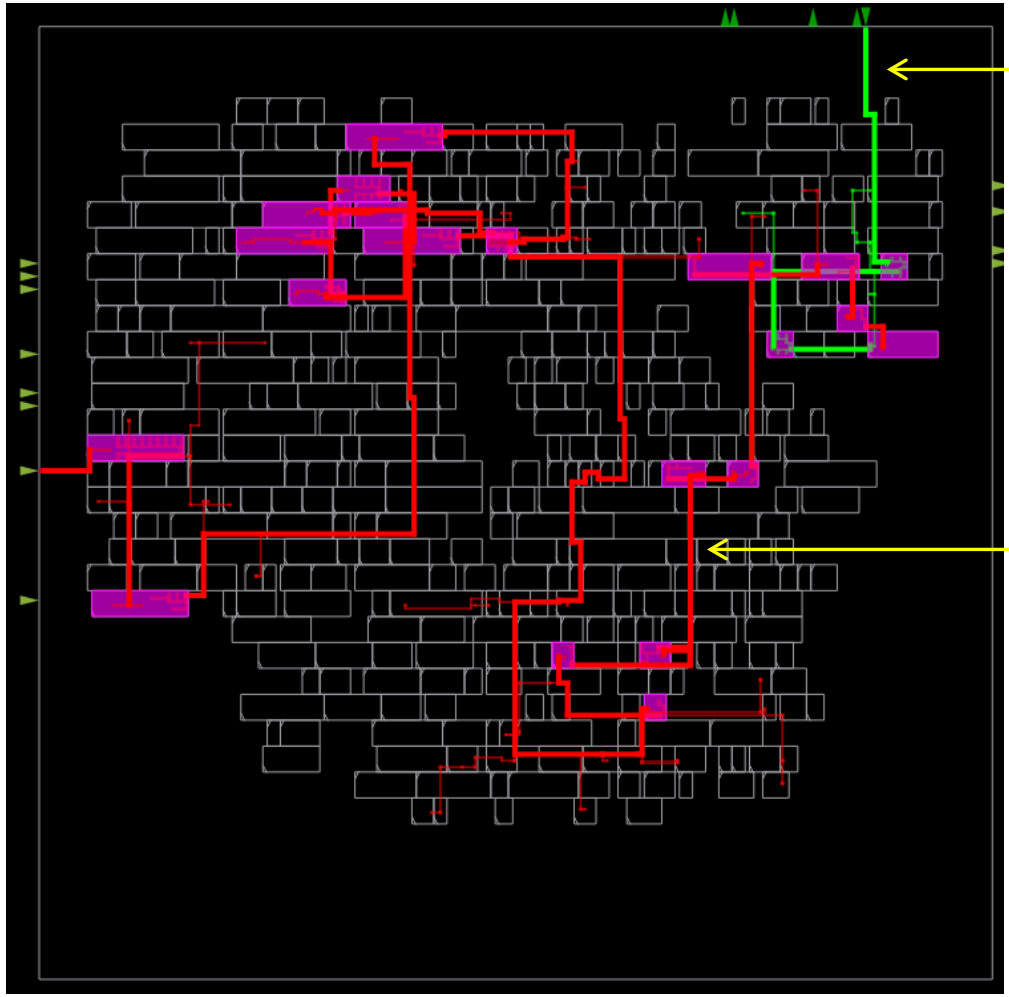
- **Turn on/off individual layers**
- **Highlight specific cells or nets**
- **Highlight the critical path**
- **Produce 'heat maps'**

Cfr Handson I

Exploring the GUI



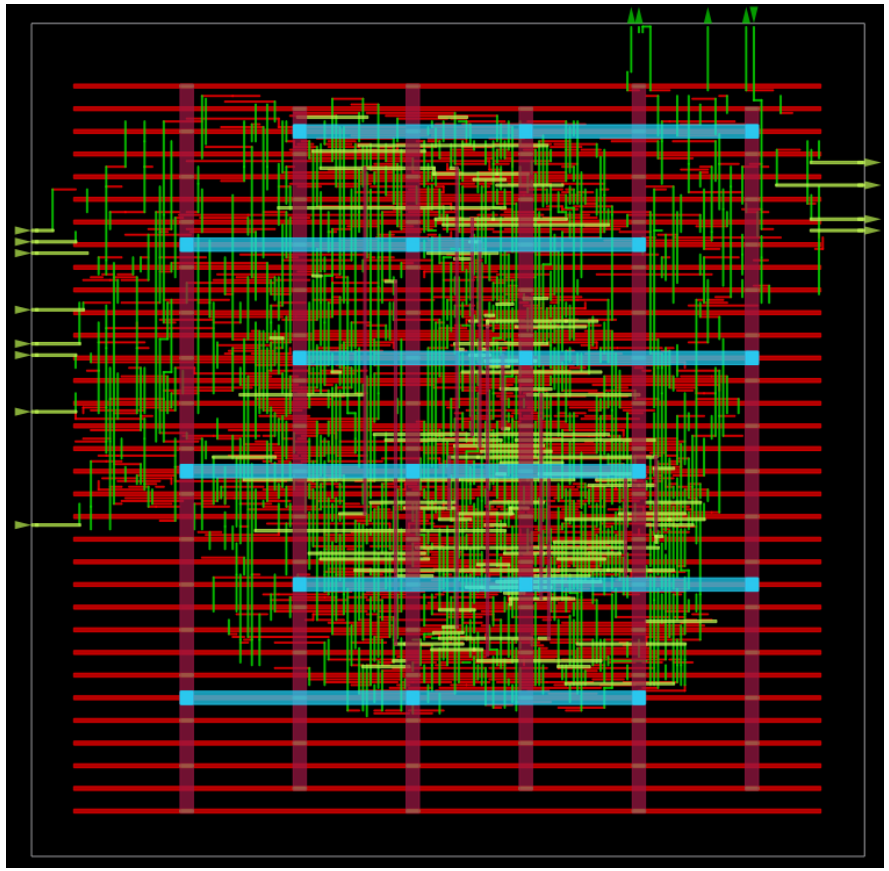
Exploring the GUI



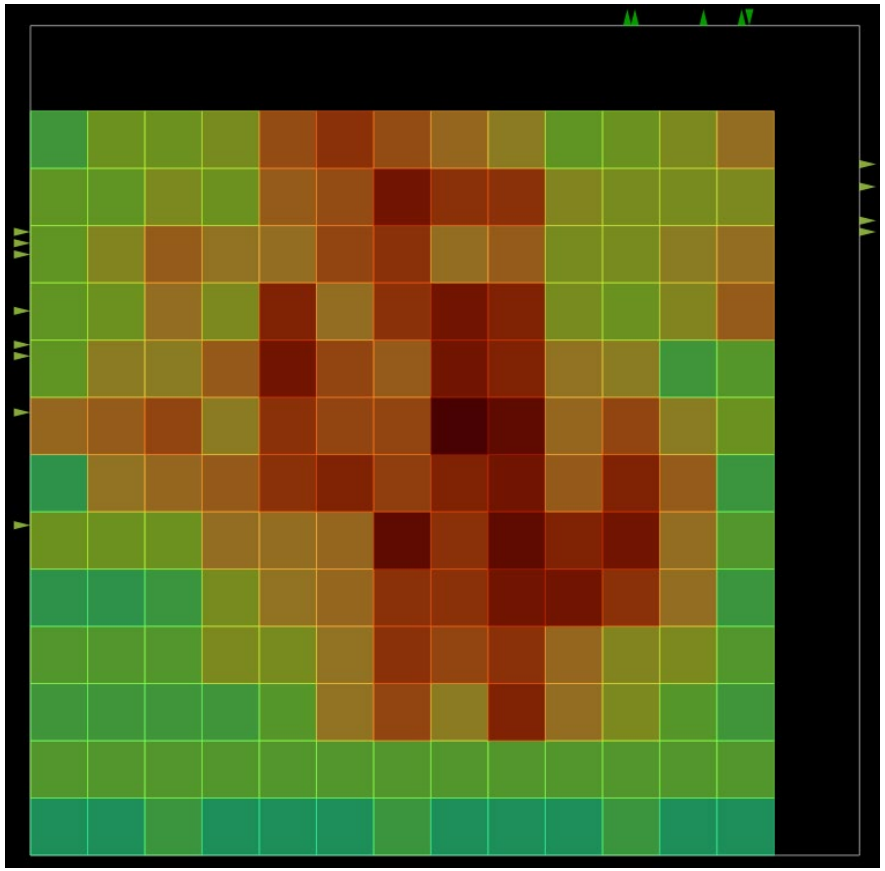
Capture Path Clock
(End-point of Critical path is a flip-flop)

Critical Path
(Slowest Combinational Path)

Exploring the GUI



Routing



Routing Congestion Heatmap

Extracting chip design results

```
# make -f Makefile.sboxaes chipdata
```

Copies chip data from docker container to work/ dir

logs/

```
1_1_yosys.log
2_1_floorplan.json
2_1_floorplan.log
2_2_floorplan_io.json
2_2_floorplan_io.log
2_3_tdms.json
2_3_tdms_place.log
2_4_mplace.json
2_4_mplace.log
2_5_tapcell.json
2_5_tapcell.log
2_6_pdn.json
2_6_pdn.log
3_1_place_gp_skip_io.json
3_1_place_gp_skip_io.log
3_2_place_iop.json
3_2_place_iop.log
3_3_place_gp.json
3_3_place_gp.log
3_4_resizer.json
3_4_resizer.log
3_5_opendp.json
3_5_opendp.log
4_1_cts.json
4_1_cts.log
4_2_cts_fillcell.json
4_2_cts_fillcell.log
5_1_fastroute.json
5_1_fastroute.log
5_2_TritonRoute.json
5_2_TritonRoute.log
6_1_merge.log
6_report.json
6_report.log
```

results/

```
1_1_yosys.v
1_synth.sdc
1_synth.v
2_1_floorplan.odb
2_2_floorplan_io.odb
2_3_floorplan_tdms.odb
2_4_floorplan_macro.odb
2_5_floorplan_tapcell.odb
2_6_floorplan_pdn.odb
2_floorplan.odb ->
2_floorplan.sdc
3_1_place_gp_skip_io.odb
3_2_place_iop.odb
3_3_place_gp.odb
3_4_place_resized.odb
3_5_place_dp.odb
3_place.odb -> 3_5_place_dp.odb
3_place.sdc
4_1_cts.odb
4_2_cts_fillcell.odb
4_cts.odb -> 4_2_cts_fillcell.odb
4_cts.sdc
5_1_grt.odb
5_2_route.odb
5_route.odb -> 5_2_route.odb
5_route.sdc
6_1_fill.odb -> 5_2_route.odb
6_1_fill.sdc
6_1_merged.gds
6_final.def
6_final.gds -> 6_1_merged.gds
6_final.odb
6_final.sdc
6_final.spef
6_final.v
route.guide
updated_clks.sdc
```

reports/

```
congestion.rpt
cts_clk.webp
final_clocks.webp
final_ir_drop.webp
final_placement.webp
final_resizer.webp
final_routing.webp
synth_check.txt
synth_stat.txt
```

WPI

Part I Handson

A Quick Chip

Objectives of Handson I

1. Use the design infrastructure
2. Build a small chip (SBOX) and analyze the intermediate results
3. Compare the chip characteristics of different SBOX variants

Getting Started

- You need an SSH with X11 forwarding into the design server
- Use the IP address provided at the tables
- Use the password

whoneedspublickeyswithapasswordlikethat

After login, move to crypto-asic-oss

```
# cd crypto-asic-oss/
# ls
Makefile.ciphersimon  Makefile.picoaes      ciphersimon  mavg2       sboxpresent
Makefile.counter      Makefile.sboxaes      counter      picoaes     scripts
Makefile.counterchip  Makefile.sboxaeslut   counterchip  sboxaes
Makefile.mavg         Makefile.sboxaespipe  make.design  sboxaeslut
Makefile.mavg2        Makefile.sboxpresent  mavg         sboxaespipe
```

Relevant Examples for Handson I

sboxaes	Computational AES SBOX with output register
sboxaeslut	Lookup Table Based AES SBOX with output register
sboxaespipe	Computational AES SBOX with one pipeline register and output register
sboxpresent	Lookup Table Based PRESENT SBOX with output register

Sample Commands to try out (sboxaes)

- RTL Simulation

```
make -f Makefile.sboxaes rtlsim
```

- Synthesis

```
make -f Makefile.sboxaes synthesis
```

- Gate Level Simulation

```
make -f Makefile.sboxaes glsim
```

- Chip

```
make -f Makefile.sboxaes openroad
make -f Makefile.sboxaes chip
```

- Chip Visualization

```
(while still in openroad docker)
make -f Makefile.sboxaes chipgui
```

Tip:

Look into Makefile.sboxaes to modify parameters related to RTL synthesis and STA

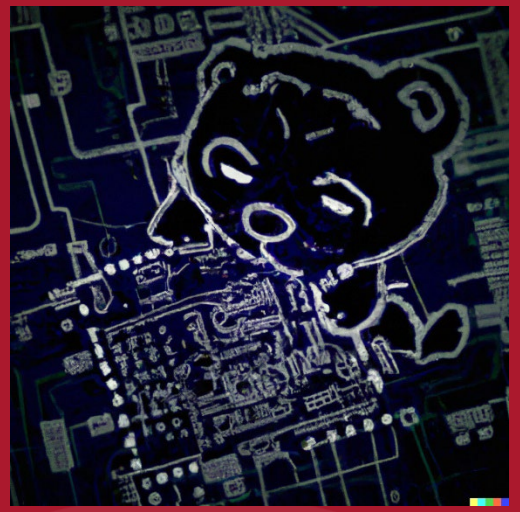
Loop into chip/config.mk to modify parameters related to chip backend

Assignment Handson I

1. Verify that you are able to obtain the same cell count, design area and utilization for sboxaes
2. Pick one of the other SBOXes and compare the cell count for synthesis, the design area, and utilization
3. Look at the chip layout of a chosen design in the chip GUI. Experiment with cell selection, layer selection, design navigation
   hint: consult https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#openroad-gui

	synthesis	design area	utilization (%)
sboxaes	433	3897	50
sboxaeslut			
sboxaespipe			
sboxpresent			

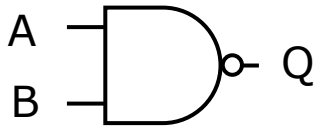
WPI

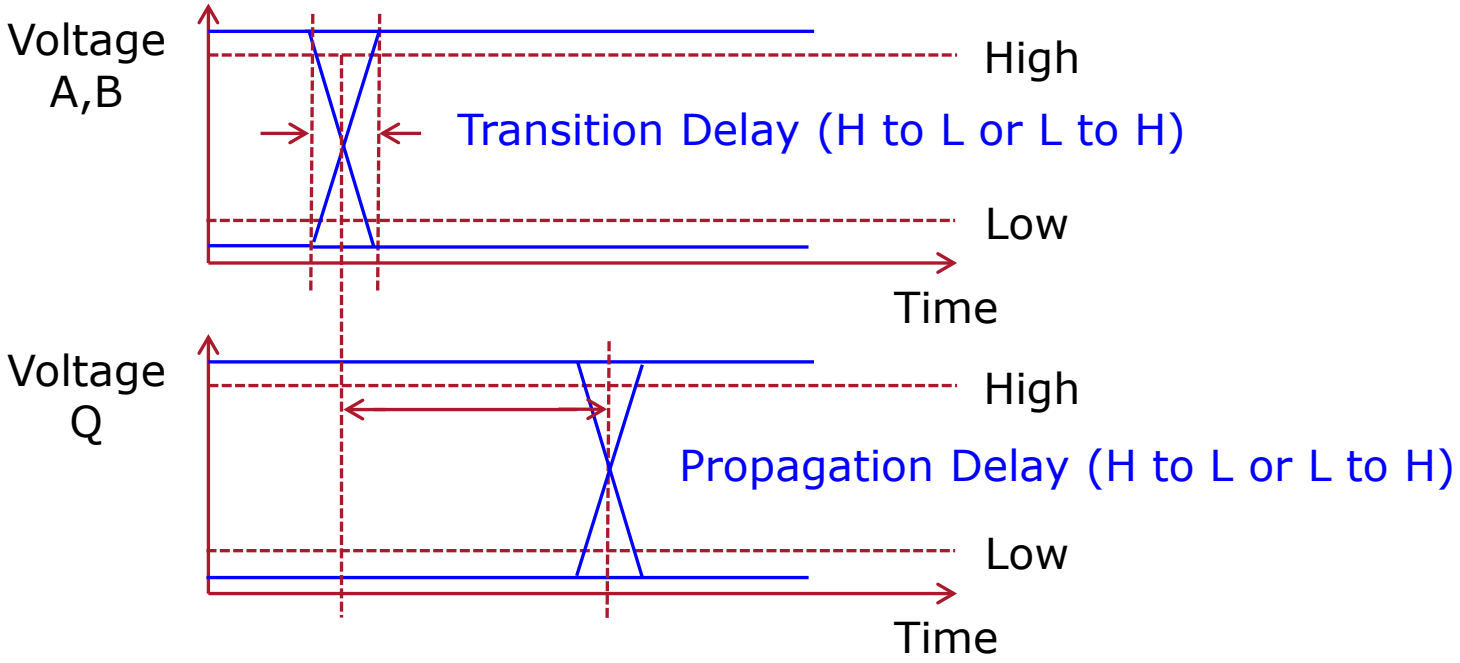


Part II
Timing Analysis

What determines speed in combinational logic?

- Speed means:
  - How fast do we know Q after we apply A, B ?
- To 'apply' a logic level means, electrically, driving the gate input to a high level or a low level

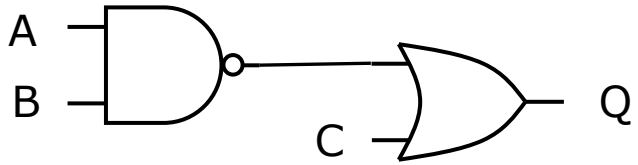




What determines speed in combinational logic?

- Speed means:
  - How fast do we know Q after we apply A, B ?
- To 'apply' a logic level means, electrically, driving the gate input to a high level or a low level
- Transition Delay is the time needed for an input/output to change
- Propagation Delay is the time needed for an output to change as a result of a change to the input
- High-to-low and low-to-high transitions are electrically different, and hence are modeled as independent values

What is the speed of a combinational function?



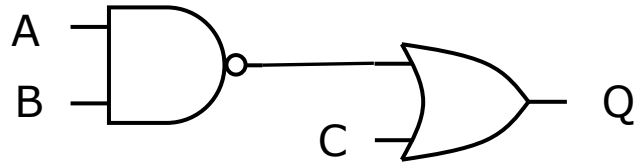
- Speed means:
  - How quickly do we know Q after we apply A, B, C ?
  - Speed of the circuit is determined by the speed of individual gates

- In first order, we can say

$$T_{prop,circuit} = T_{prop,NAND} + T_{prop,OR}$$

Why?

What is the speed of a combinational function?



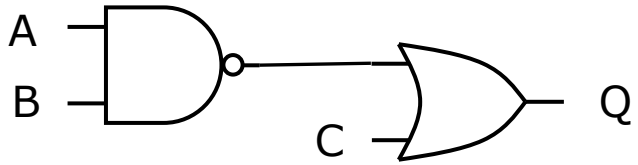
- Speed means:
  - How quickly do we know Q after we apply A, B, C ?
  - Speed of the circuit is determined by the speed of individual gates

- In first order, we can say

$$T_{prop,circuit} = T_{prop,NAND} + T_{prop,OR}$$

The OR input is only known after Tprop,NAND
Q is only stable Tprop,OR after all OR inputs are stable

What is the speed of a combinational function?



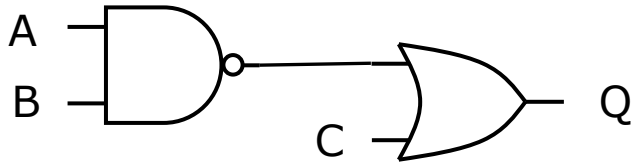
- Speed means:
  - How quickly do we know Q after we apply A, B, C ?
  - Speed of the circuit is determined by the speed of individual gates

- In first order, we can say

$$T_{prop,circuit} = T_{prop,NAND} + T_{prop,OR}$$

However, this is a worst case analysis. Why?

What is the speed of a combinational function?



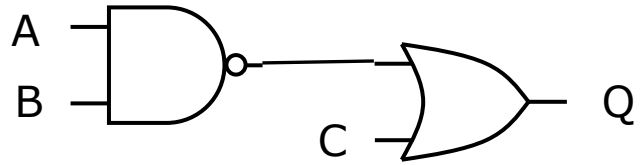
- Speed means:
  - How quickly do we know Q after we apply A, B, C ?
  - Speed of the circuit is determined by the speed of individual gates

- In first order, we can say

$$T_{prop,circuit} = T_{prop,NAND} + T_{prop,OR}$$

Whenever C -> 1, the $T_{prop,circuit} = T_{prop,OR}$
Propagation Delay depends on inputs

What is the speed of a combinational function?

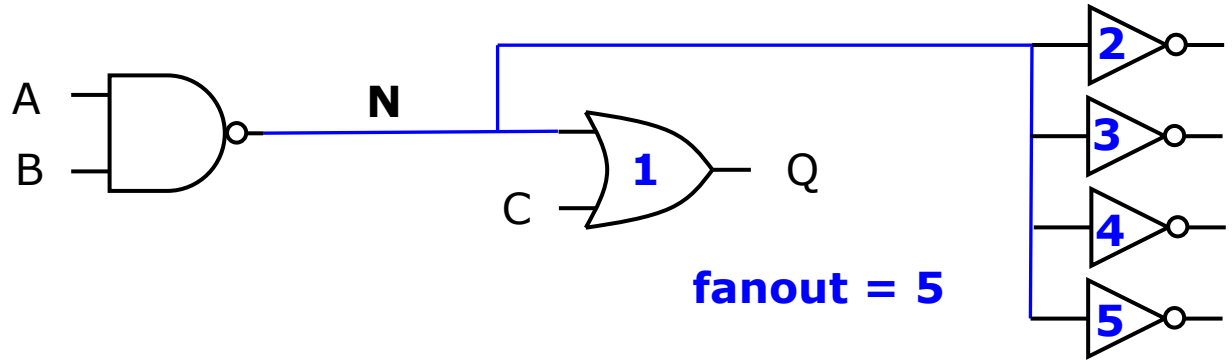


- Speed means:
  - How quickly do we know Q after we apply A, B, C ?
  - Speed of the circuit is determined by the speed of individual gates

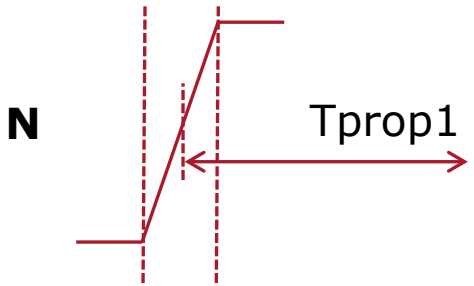
- In first order, we can say that the worst-case delay equals

$$T_{prop,circuit} = T_{prop,NAND} + T_{prop,OR}$$

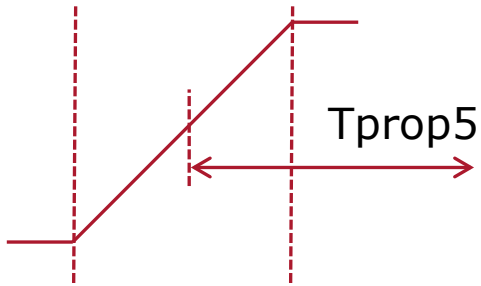
What is the speed of a combinational function?



- Speed means:
  - How quickly do we know Q after we apply A, B, C ?
  - Speed of the circuit is determined by the speed of individual gates and by the fanout of a gate (circuit topology)



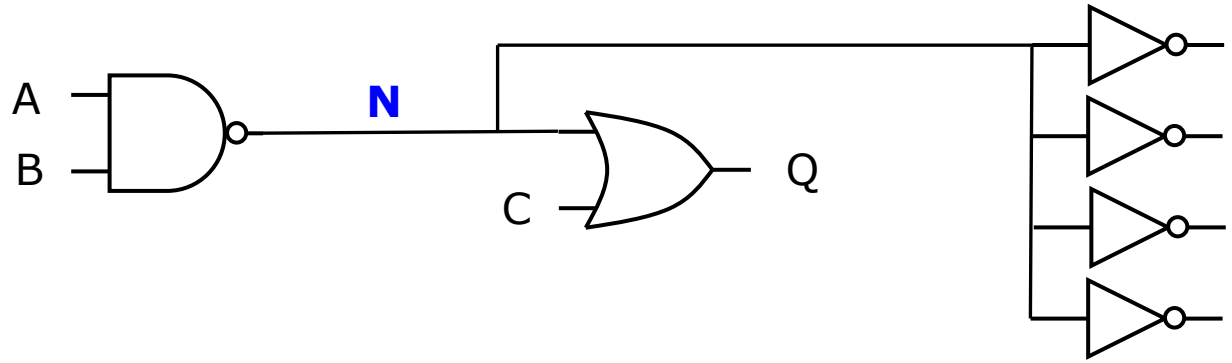
Transition Delay at fanout = 1



Transition Delay at fanout = 5

Tprop5 > Tprop1
due to slower transitions
on node **N**

What is the speed of a combinational function?



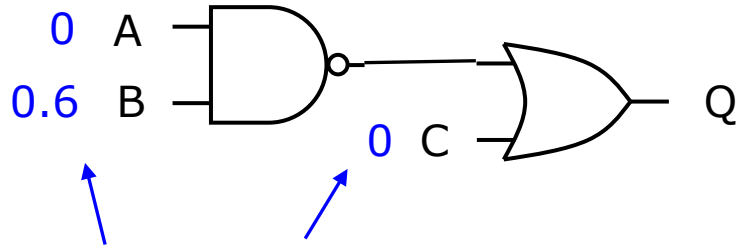
- Speed means:
  - How quickly do we know Q after we apply A, B, C ?
  - Speed of the circuit is determined by the speed of individual gates and by the fanout of a gate (circuit topology)
  - In first order, we can say that the worst-case delay equals

$$T_{prop,circuit} = T_{prop,NAND} + T_{prop,OR} + T_{wire}$$

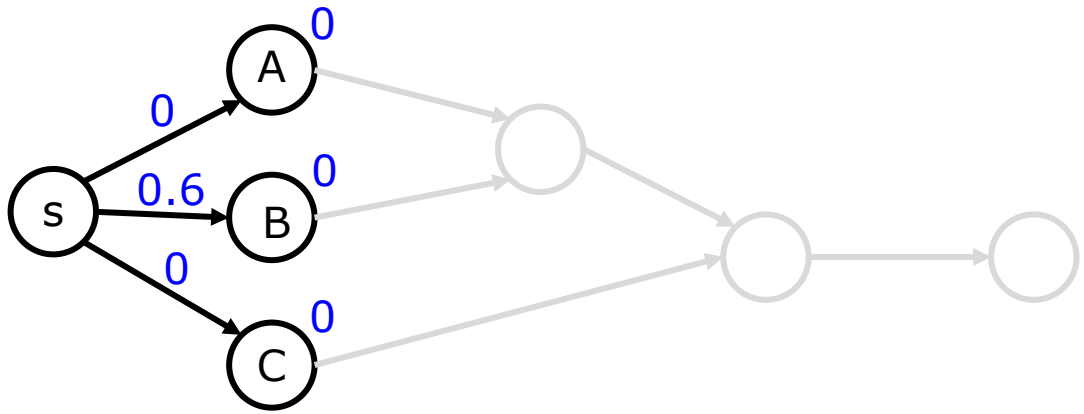
with T_{wire} a delay determined by circuit topology

Tming Model of Combinational Function

- Determine the propagation delay of a circuit output using graph
  - Nodes = gates or primary inputs or primary outputs
  - Edges = timing arcs = delays
    - Input Delays

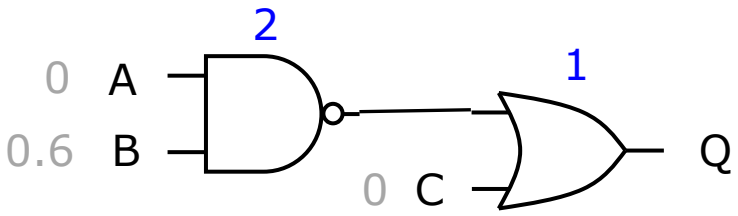


Input Delays
0.6 means: B is available
at T = 0.6 time units

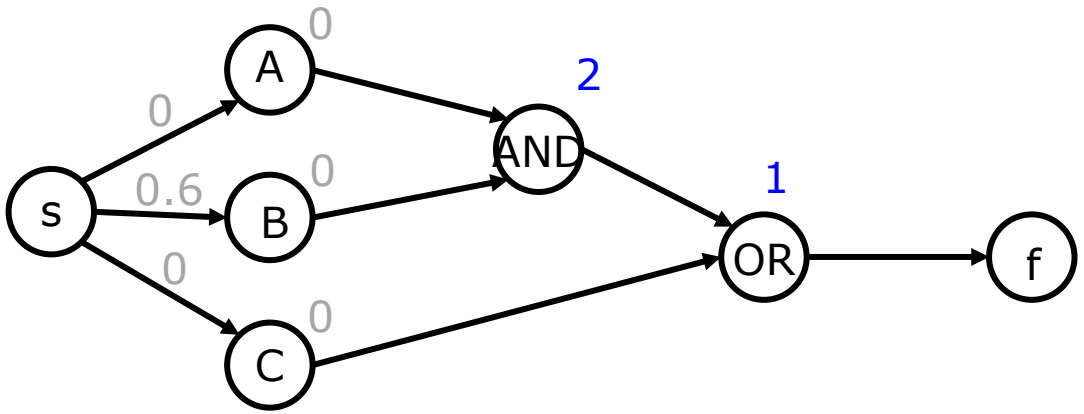


Tming Model of Combinational Function

- Determine the propagation delay of a circuit output using graph
  - Nodes = gates or primary inputs or primary outputs
  - Edges = timing arcs = delays
    - Input Delays
    - Propagation Delays

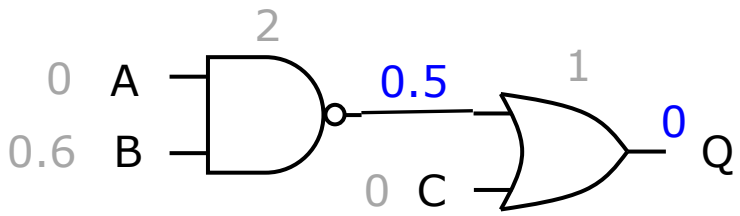


Propagation Delays
2 means: Tprop = 2



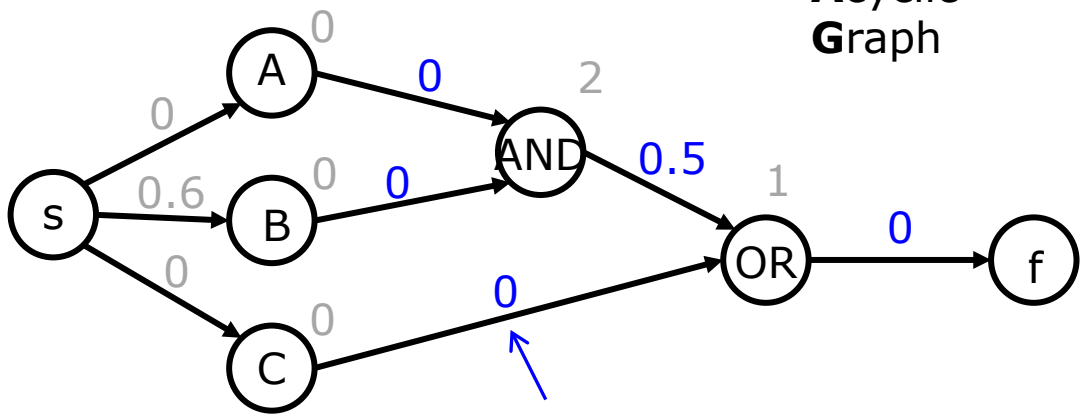
Tming Model of Combinational Function

- Determine the propagation delay of a circuit output using graph
  - Nodes = gates or primary inputs or primary outputs
  - Edges = timing arcs = delays
    - Input Delays
    - Propagation Delays
    - Wire Delays



Wire Delays

DAG
Directed
Acyclic
Graph

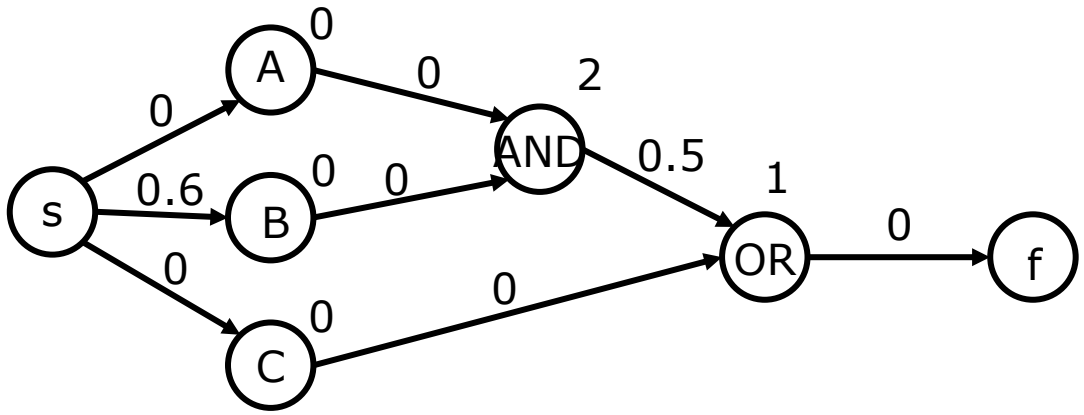


Default wire delay = 0

Tming Model of Combinational Function

- Determine the propagation delay of a circuit output using graph
  - Nodes = gates or primary inputs or primary outputs
  - Edges = timing arcs = delays
- Actual Arrival Time (AAT) = time when a node output is known

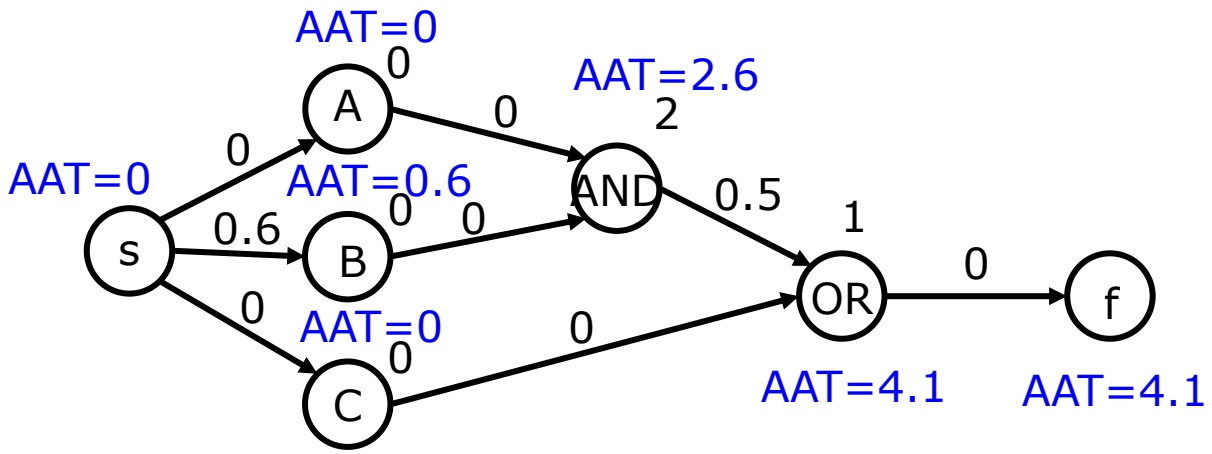
For node v, u = pred(v): AAT(v) = max (AAT(u) + delay(u,v))



Tming Model of Combinational Function

- Determine the propagation delay of a circuit output using a graph model G
  - Nodes = gates or primary inputs or primary outputs
  - Edges = timing arcs = delays
- Actual Arrival Time (AAT) = time when a node output is known

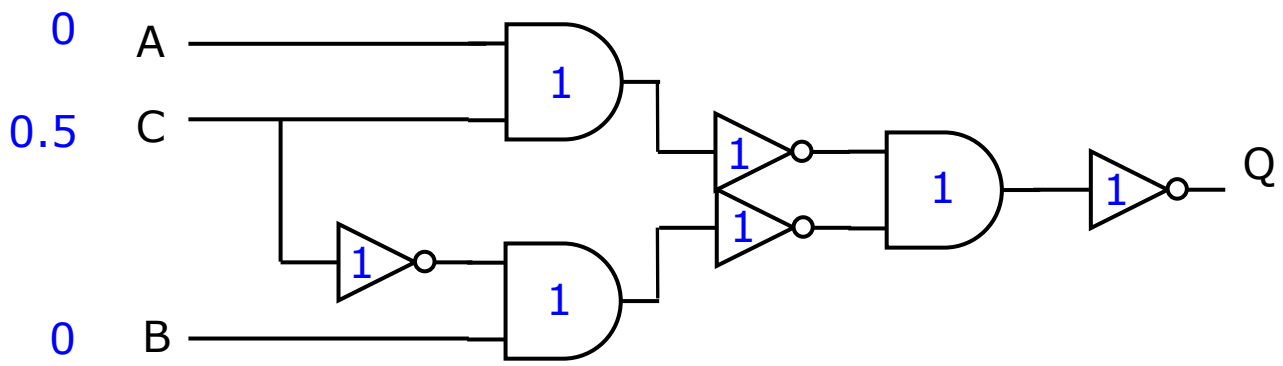
For node v, u = pred(v): AAT(v) = max (AAT(u) + delay(u,v))



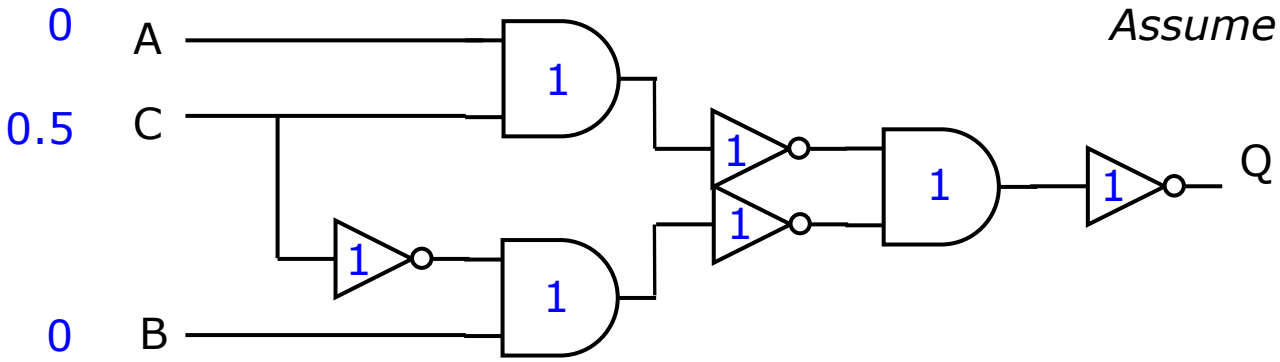
Worst Case Delay = 4.1

Example: Compute the delay of this circuit

Assume all wire delays are 0

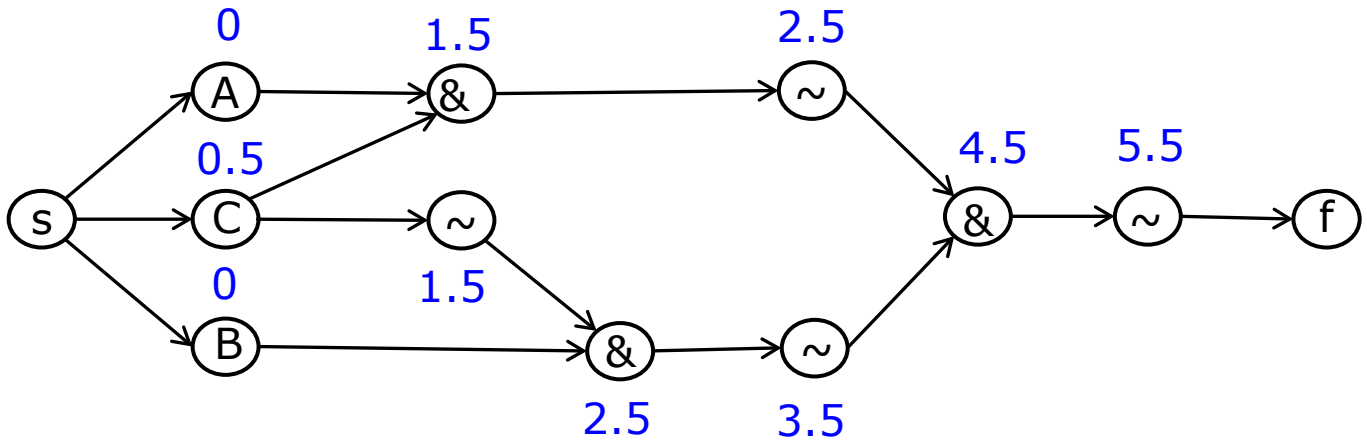


Example: Compute the delay of this circuit



Assume all wire delays are 0

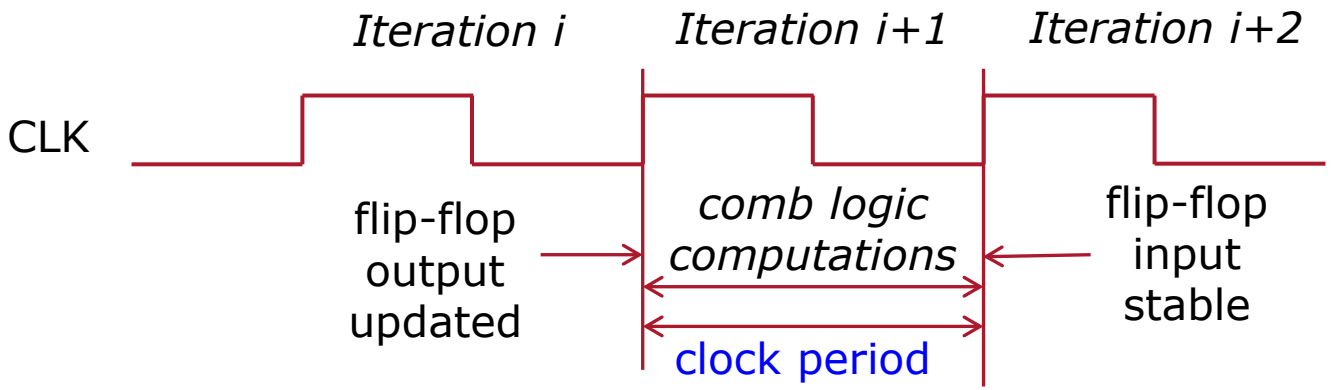
AAT Computation

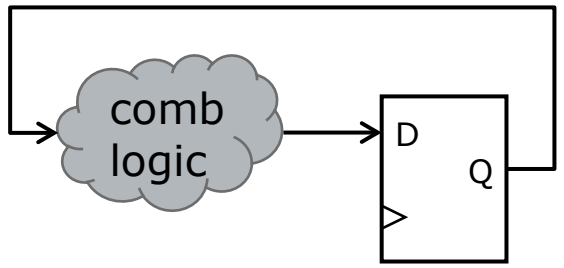


Worst Case Delay = 5.5

Performance in synchronous design

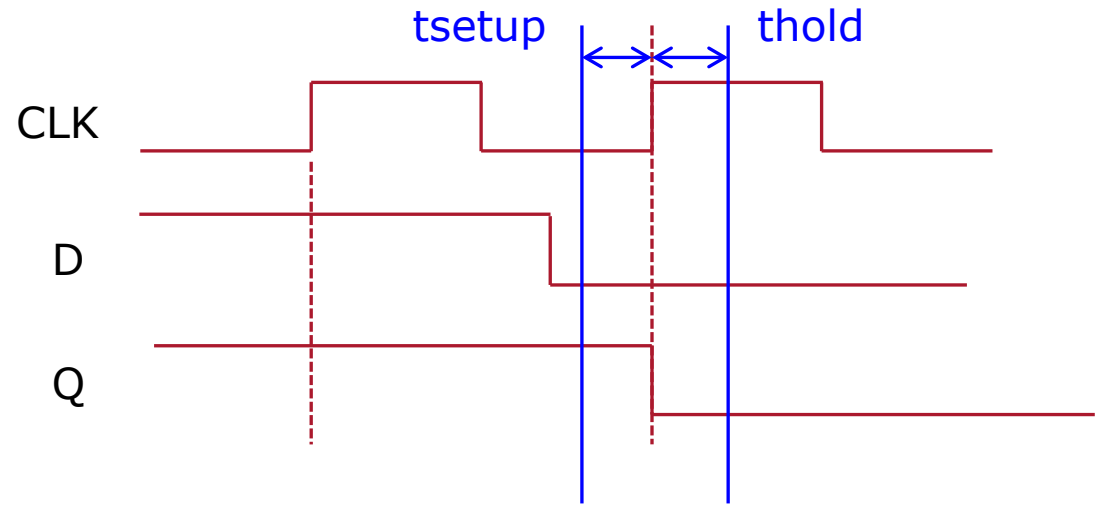
- Most practical circuits implement as iterated computations using flip-flops





The speed of a flip-flop

- Flip-flops introduce additional delays
- These delays affect the time available for comb logic

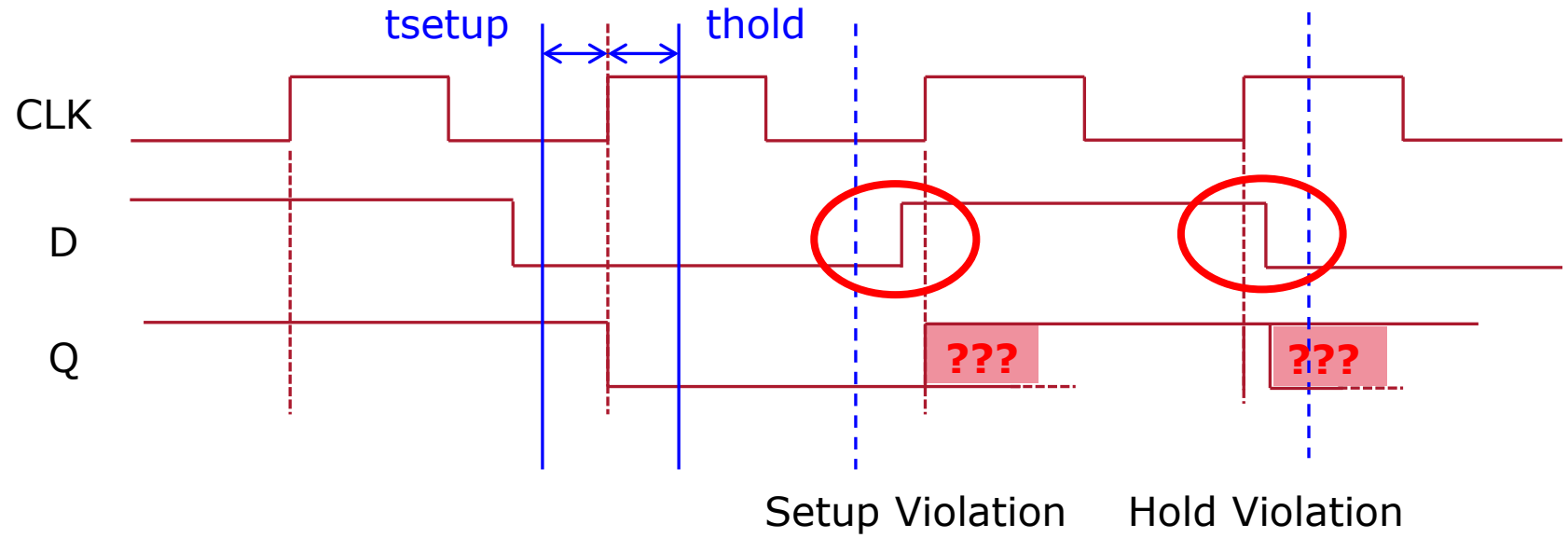


For stable operation, D must not change close to the clock edge

Setup Time = Time D must remain stable before CLK edge
Hold Time = Timd D must remain stable after CLK edge

The speed of a flip-flop

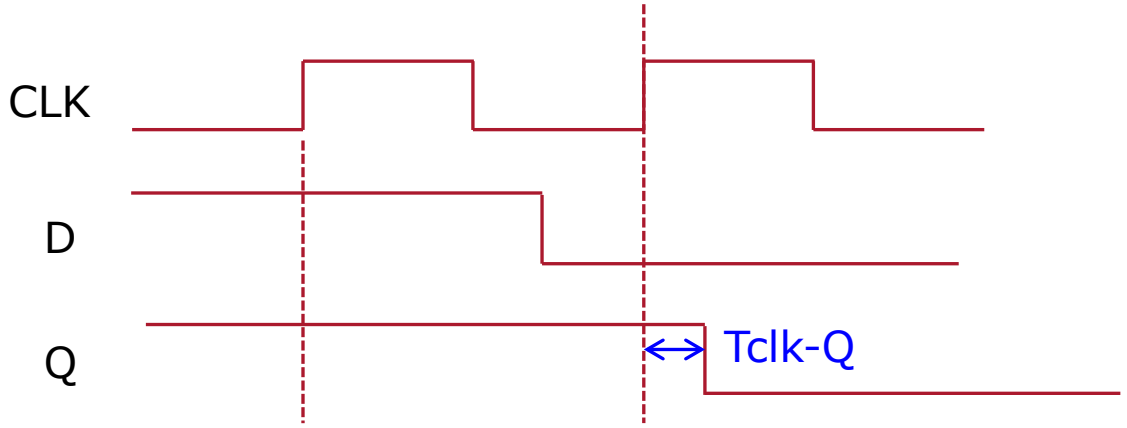
- Flip-flops are built using logic and introduce additional delays
- These delays affect the time available for comb logic



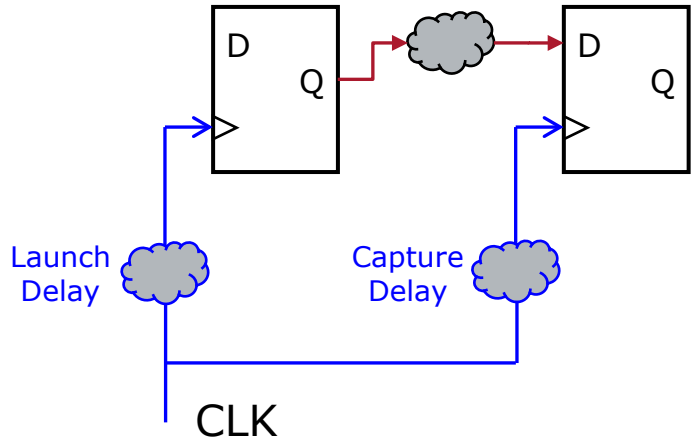
Setup Violation or **Hold Violation** break the correctness of the computation because the flip-flop *may* fail to capture correct data

The speed of a flip-flop

- Flip-flops are built using logic and introduce additional delays
- These delays affect the time available for comb logic
- Two additional timing factors relevant to flip-flop operation: Tclk-Q and Tskew

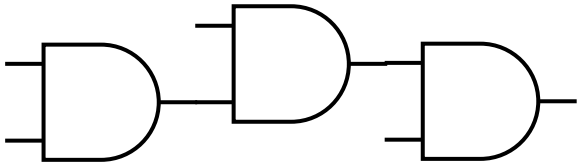


Tclk-Q = Time for Q to become stable after a CLK edge

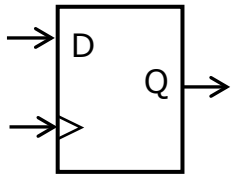


Tskew = Capture Delay – Launch Delay
= Uncertainty in the arrival time of CLK

The speed of iterated computations



Tprop = Worst-case data delay



Tsetup = Data margin before clk edge

Thold = Data margin after clk edge

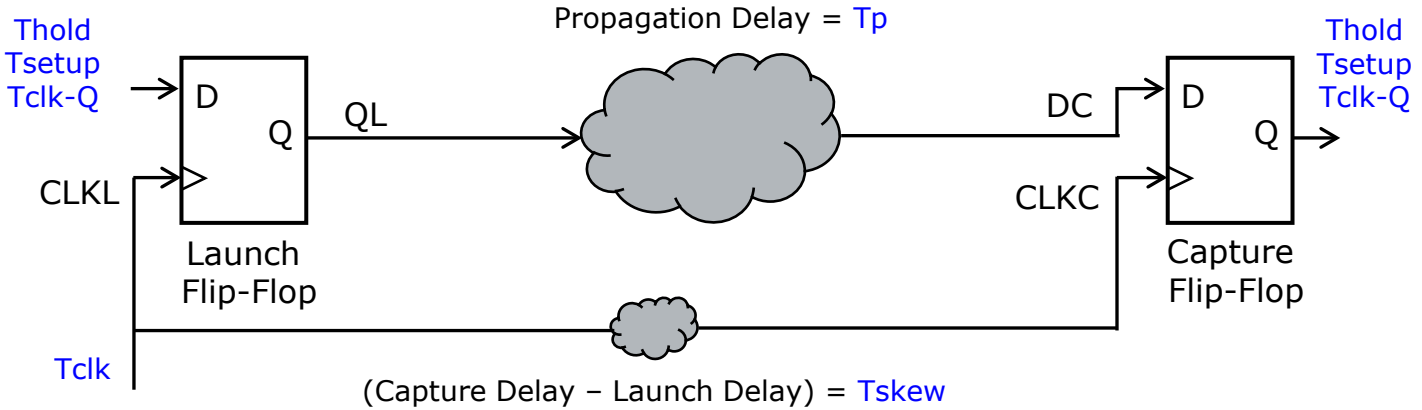
Tskew = clk edge uncertainty

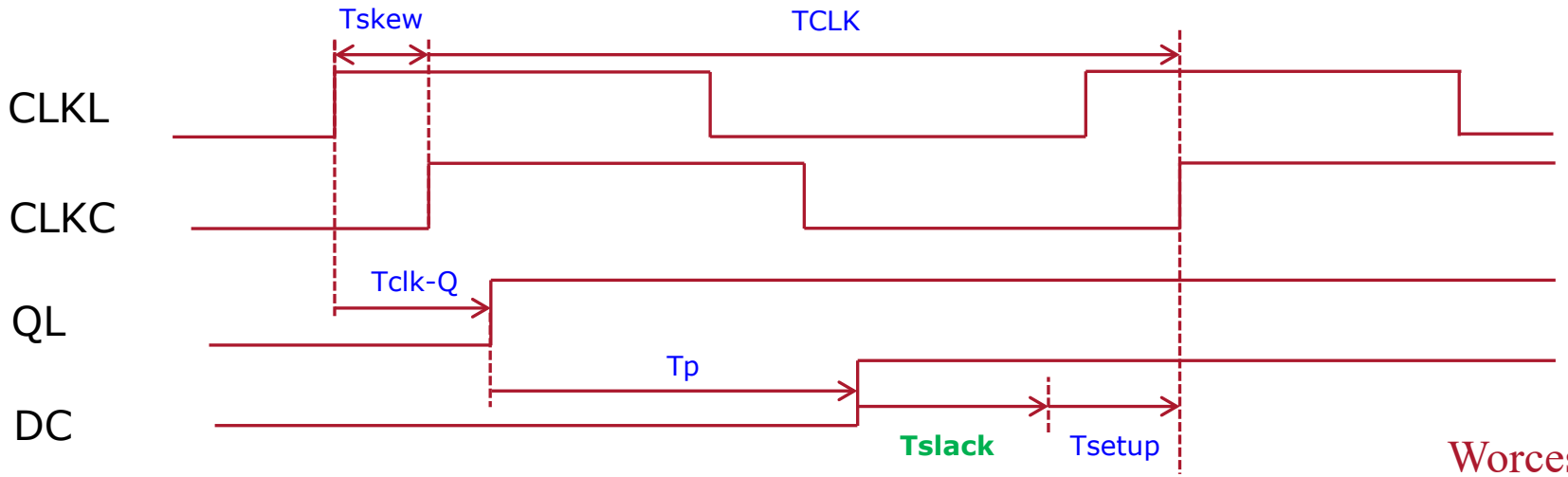
Tclk-Q = output delay after clk edge

To determine the correct operation of a synchronous design, we verify:

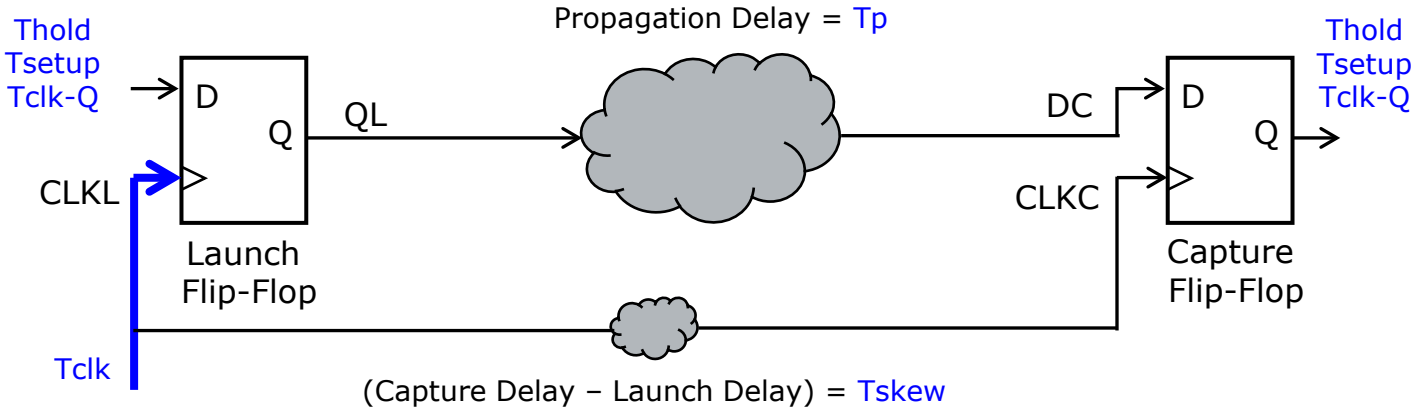
- **No setup timing violations** in any flip flop
- **No hold timing violations** in any flip flop

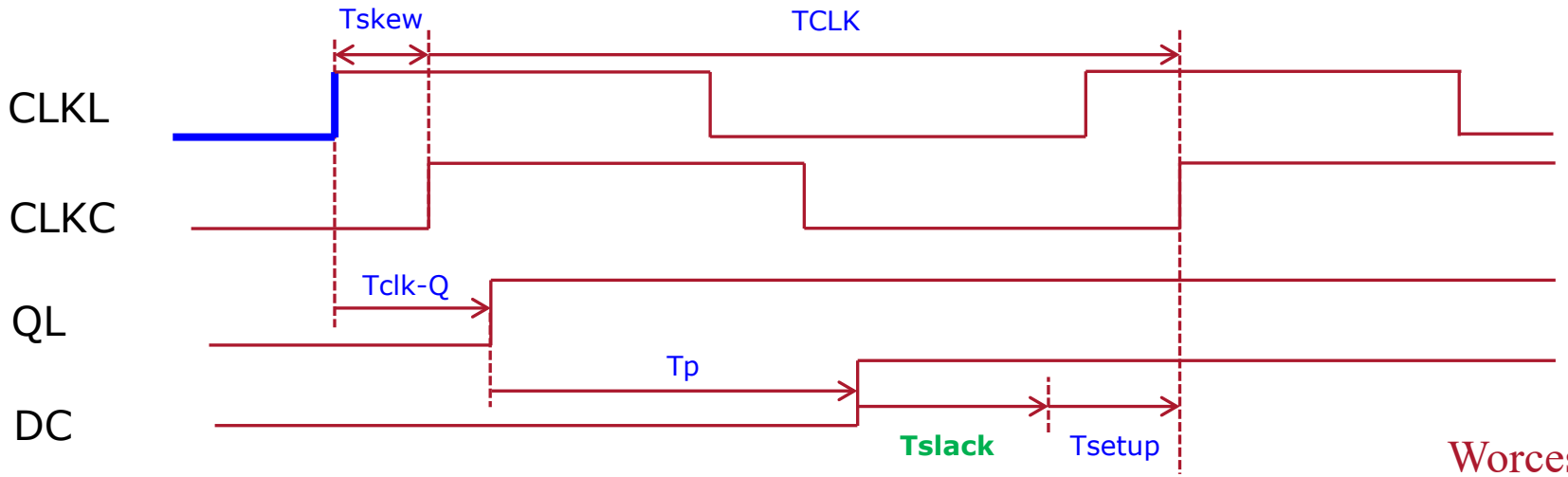
No Setup Timing Violations



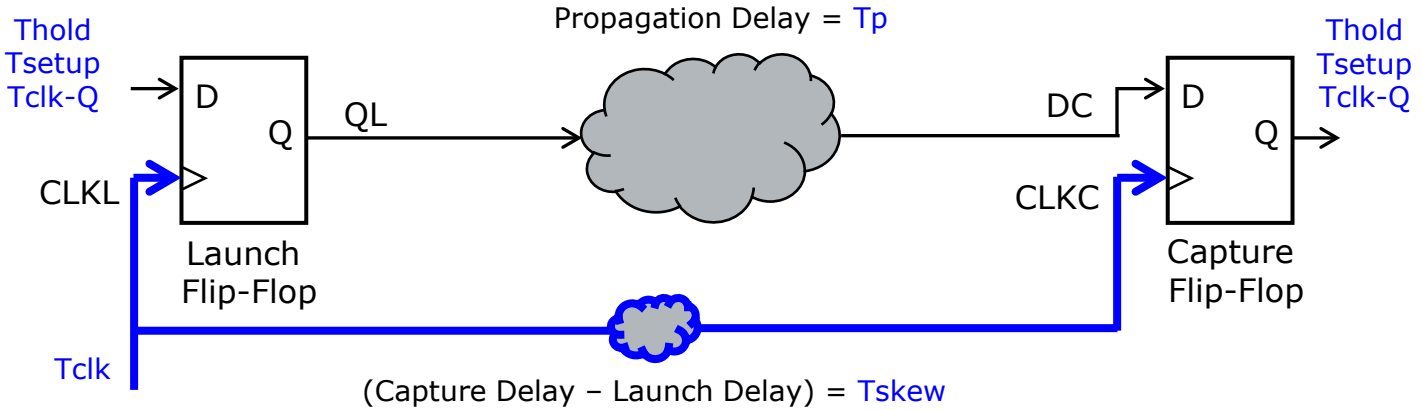


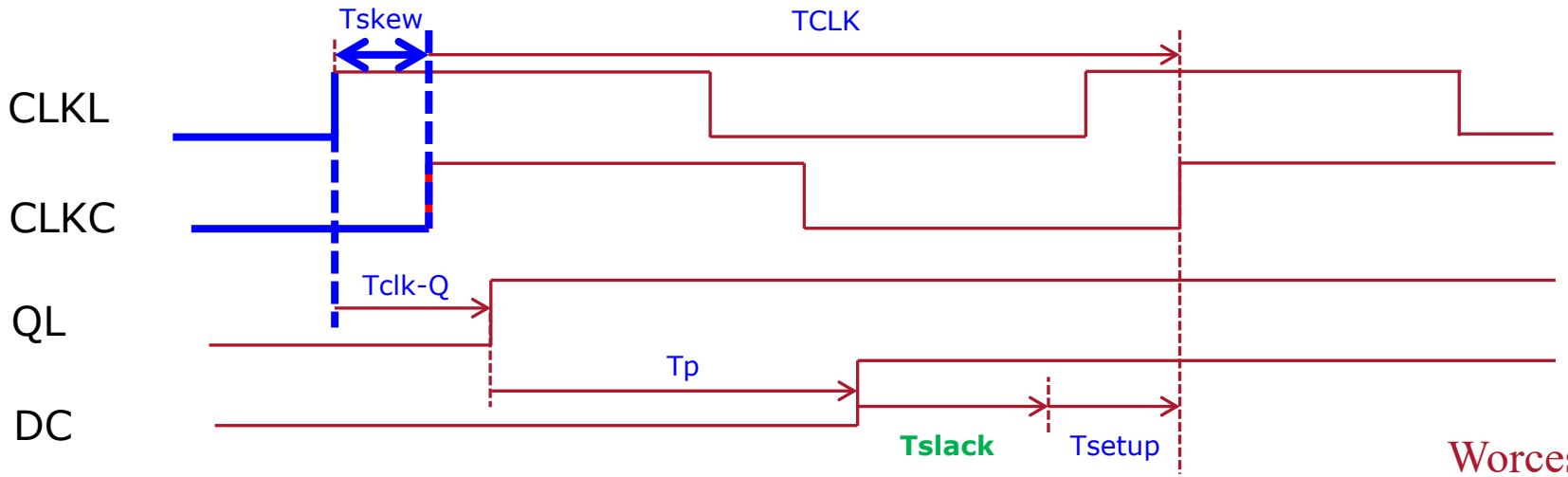
No Setup Timing Violations



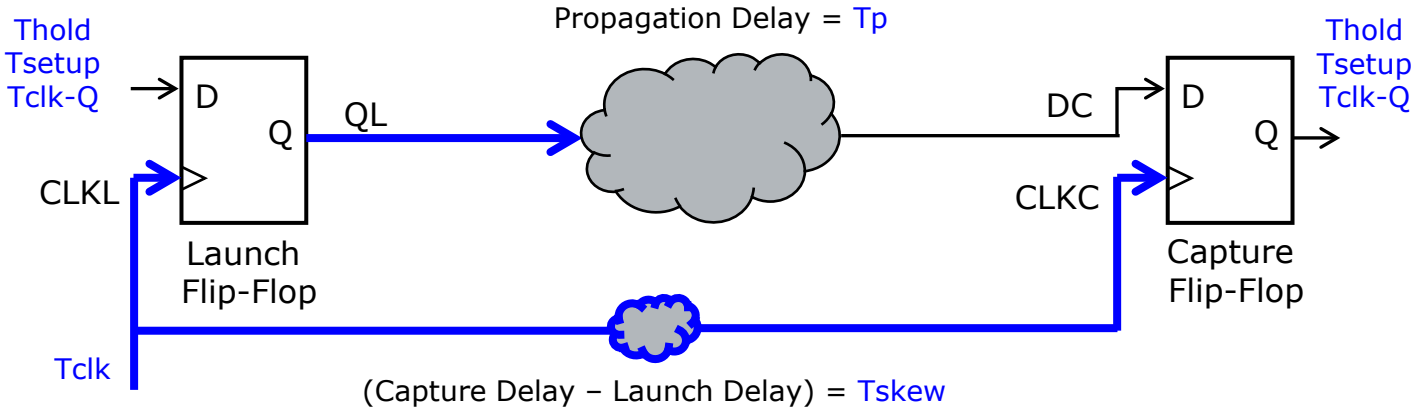


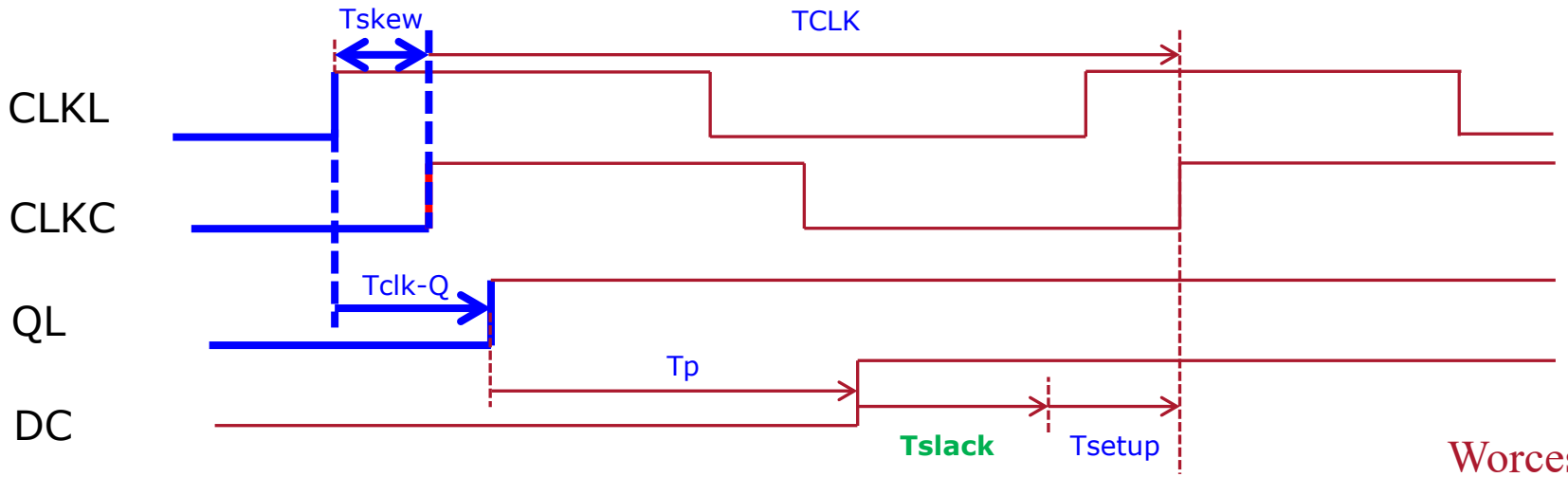
No Setup Timing Violations



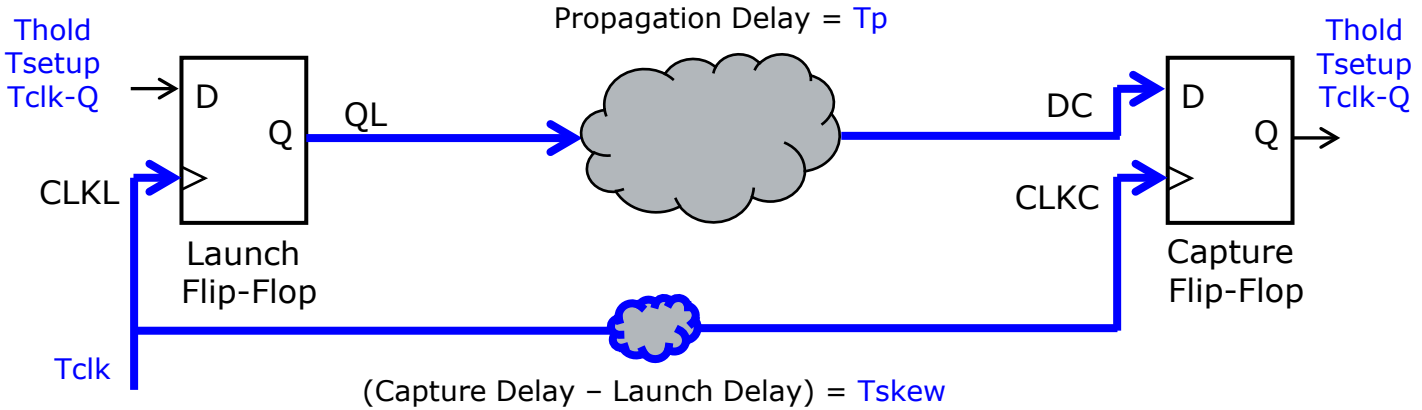


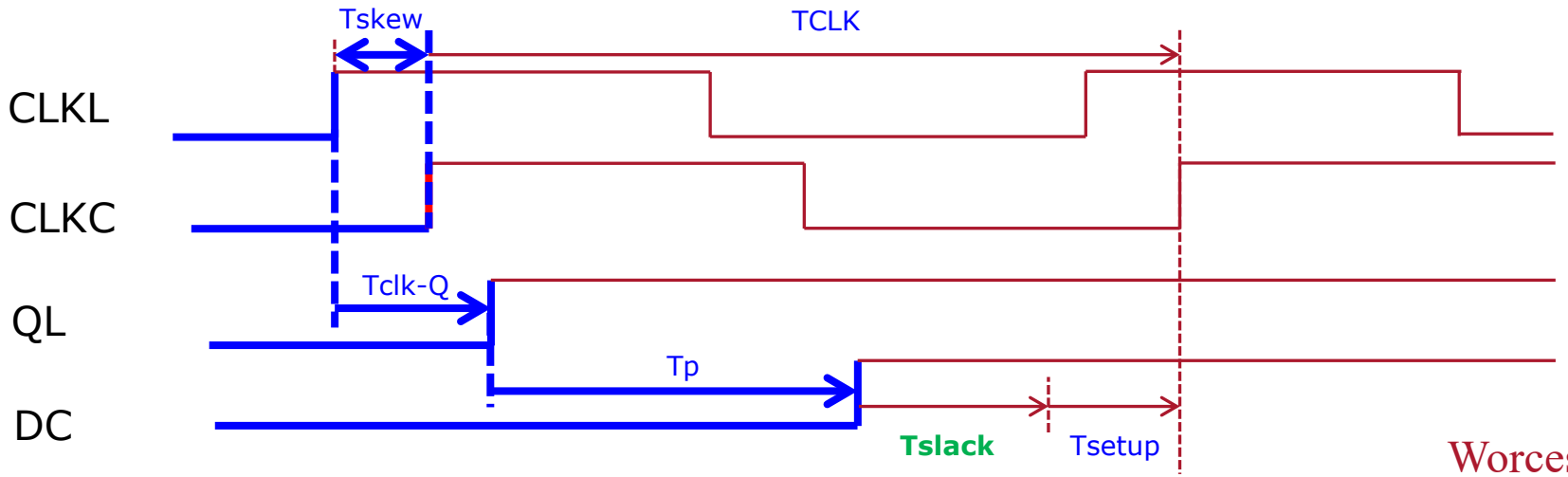
No Setup Timing Violations



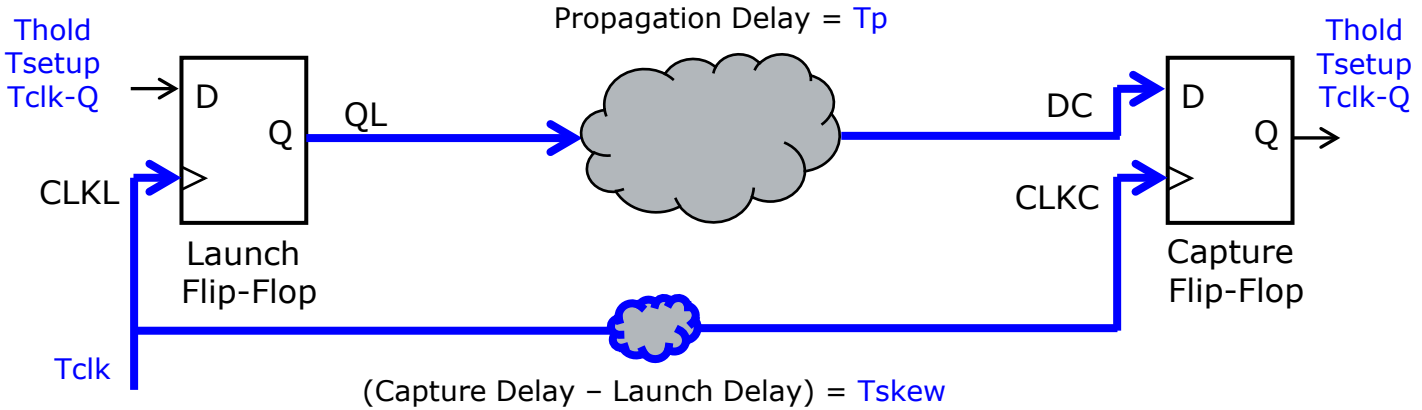


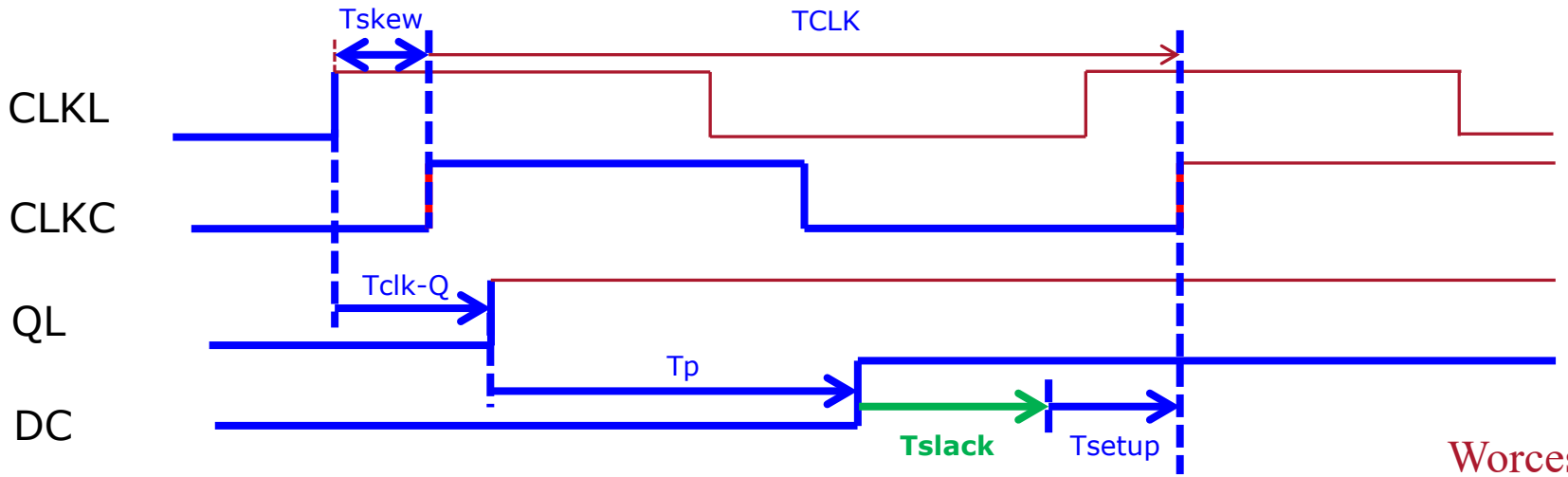
No Setup Timing Violations



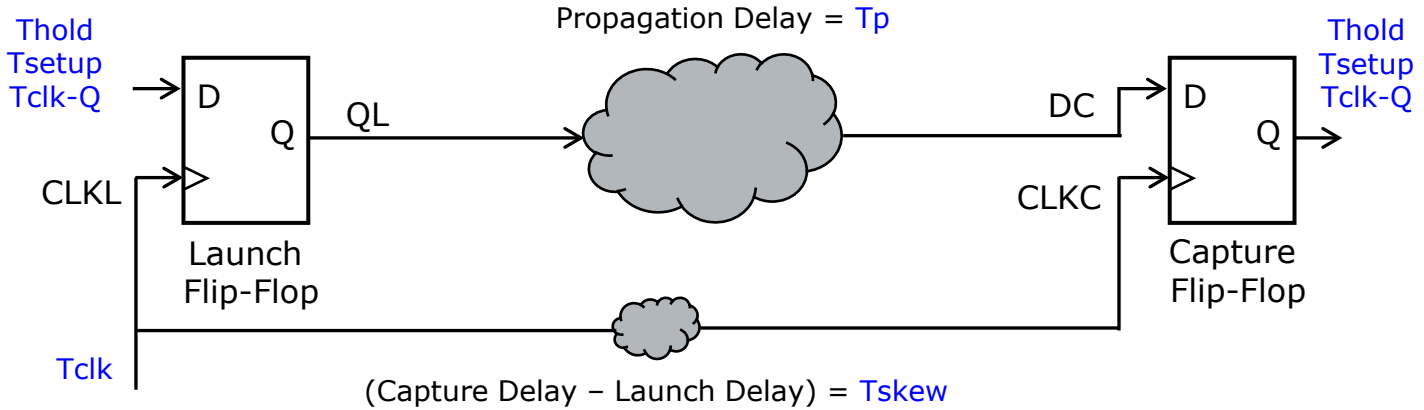


No Setup Timing Violations



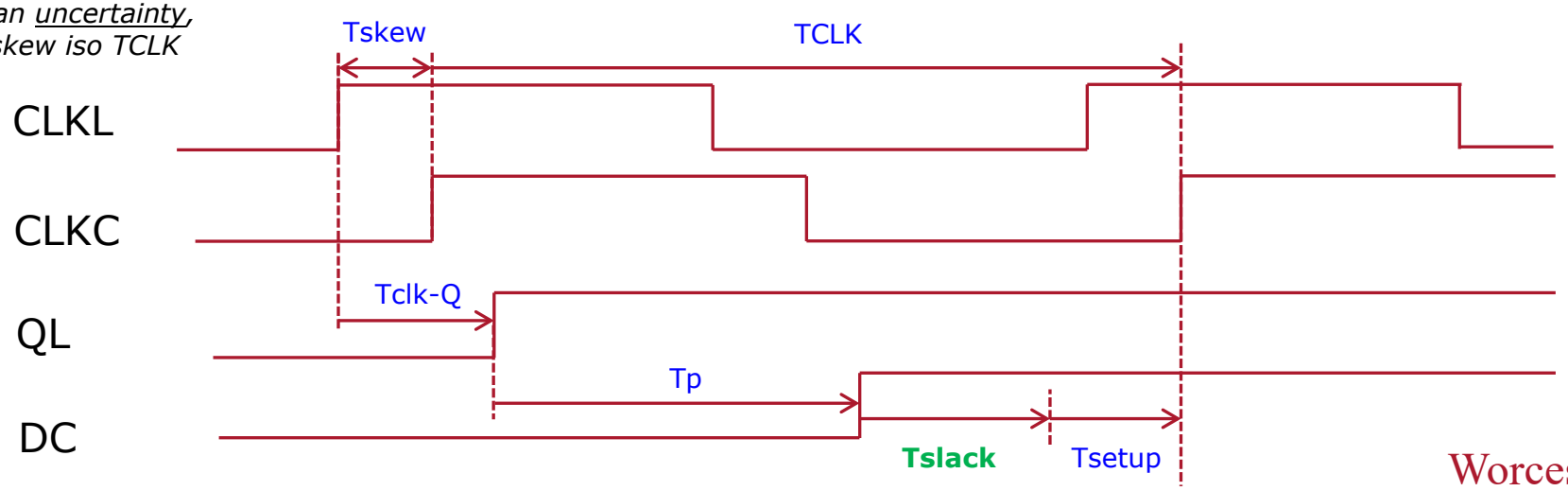


No Setup Timing Violations

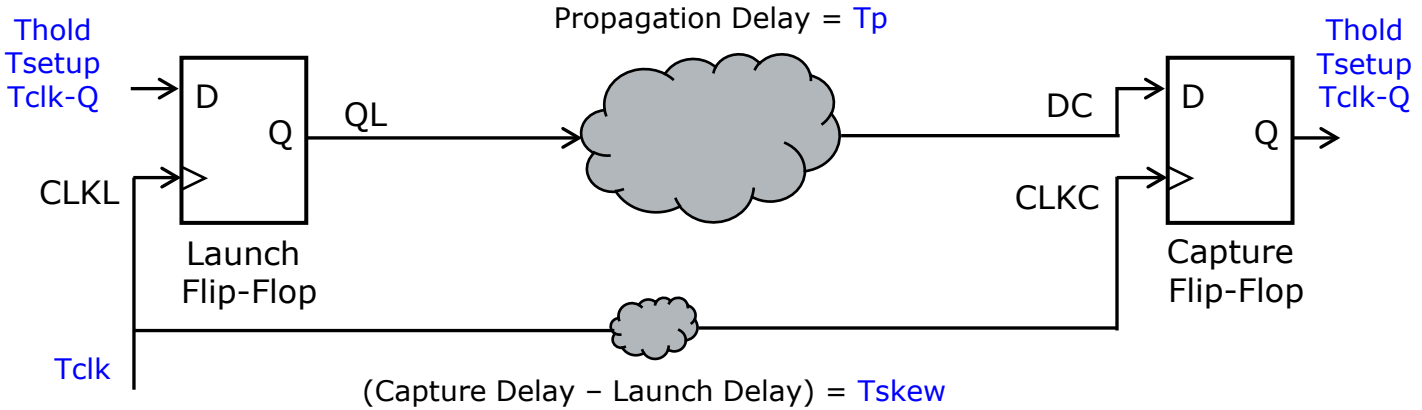


Tslack = TCLK – Tskew – Tclk-Q – Tp - Tsetup

Because Tskew is an uncertainty, we use TCLK – Tskew iso TCLK



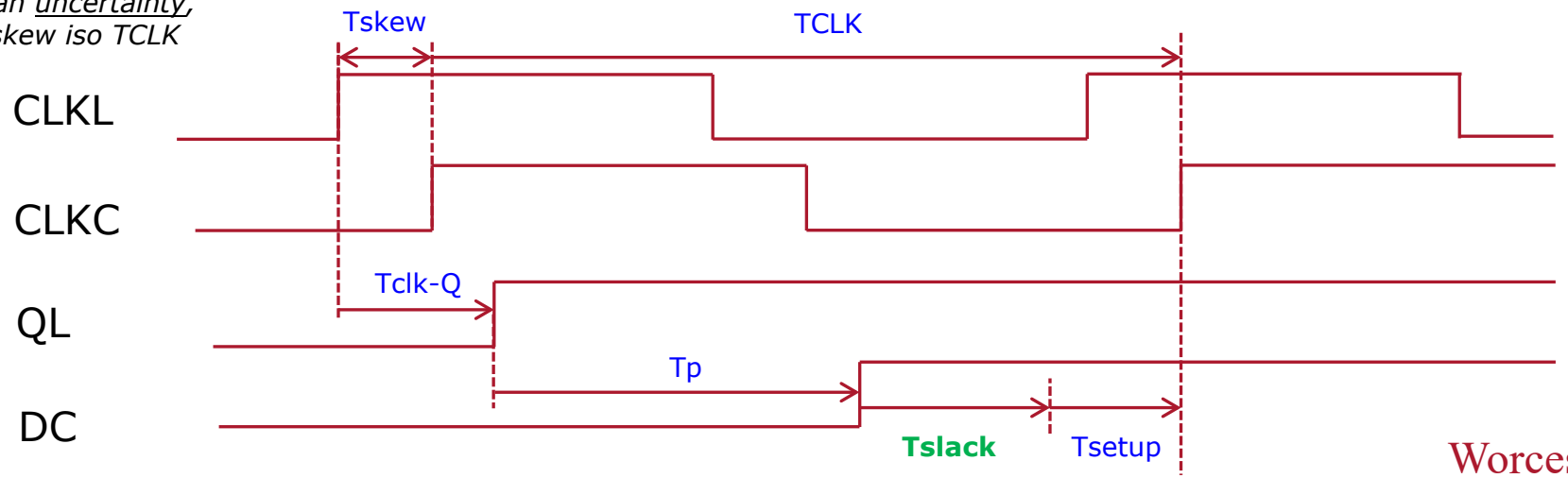
No Setup Timing Violations



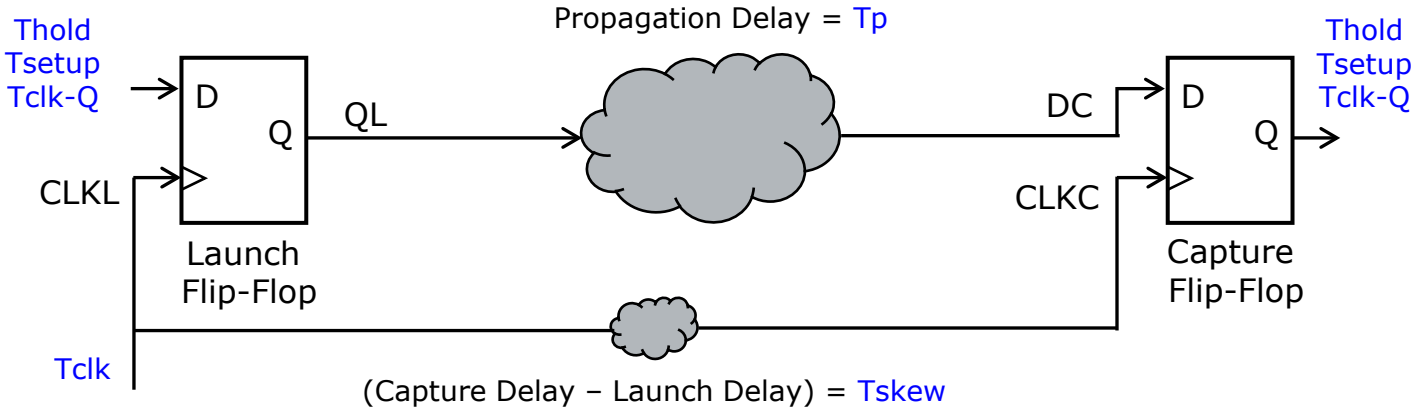
Tslack = TCLK – Tskew – Tclk-Q – Tp - Tsetup

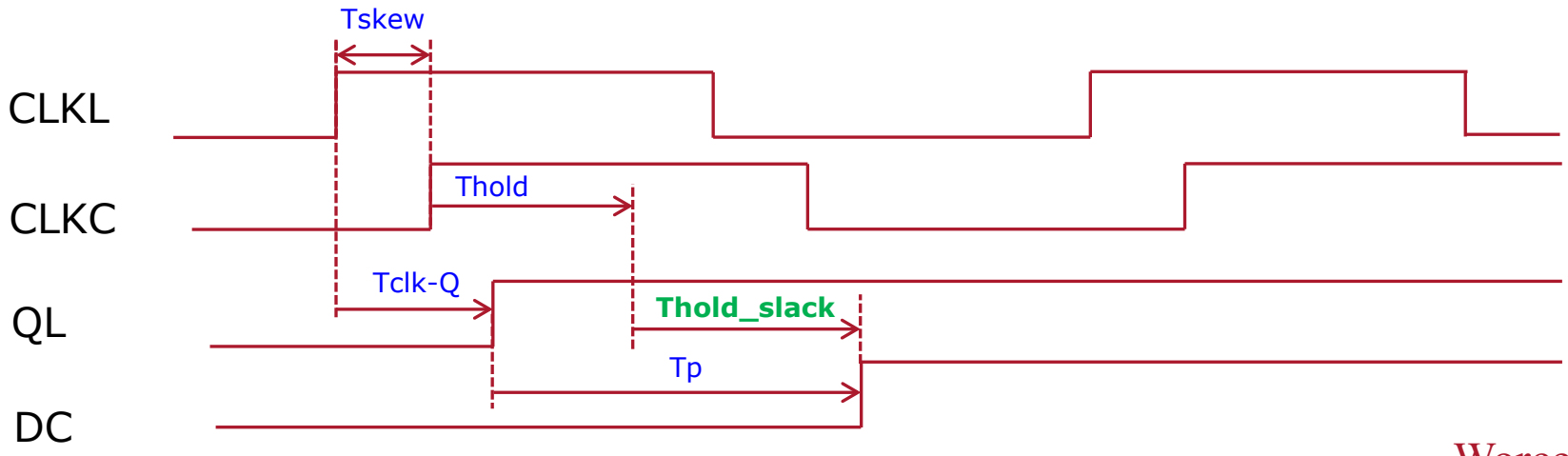
>0	**Positive Slack, OK**
= 0	**Zero Slack, Critical**
< 0	**Negative Slack, Violation**

Because Tskew is an uncertainty, we use TCLK – Tskew iso TCLK

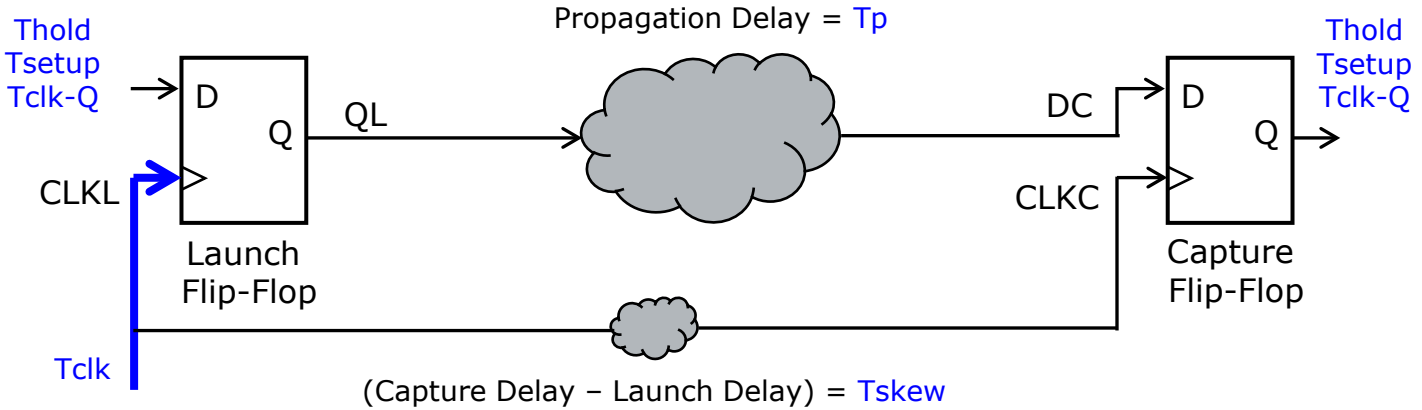


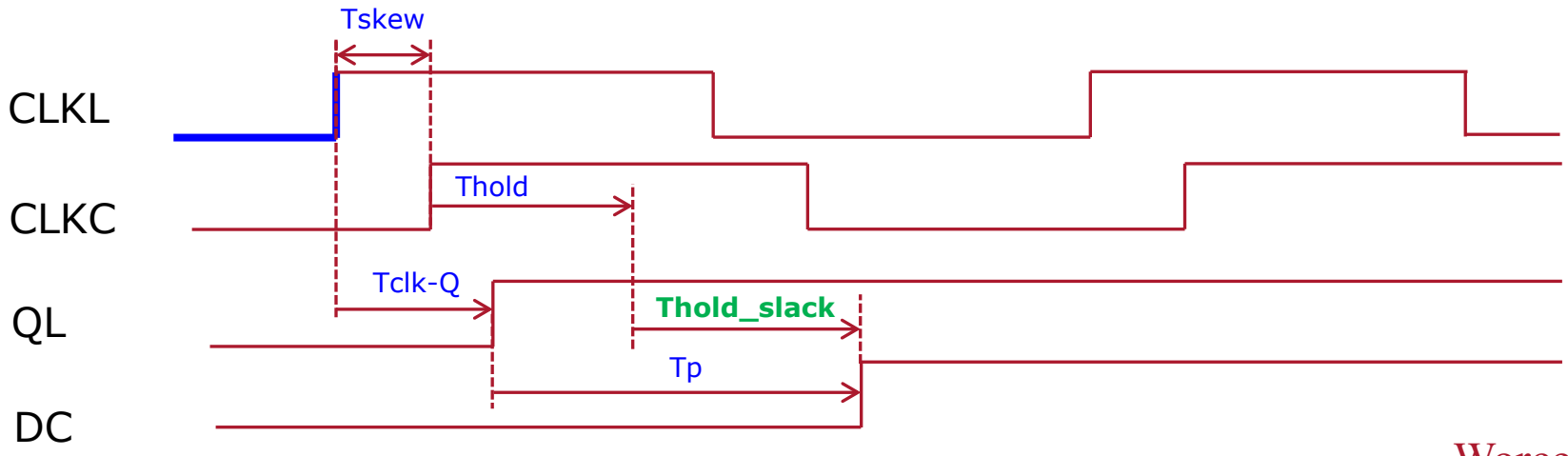
No Hold Timing Violations



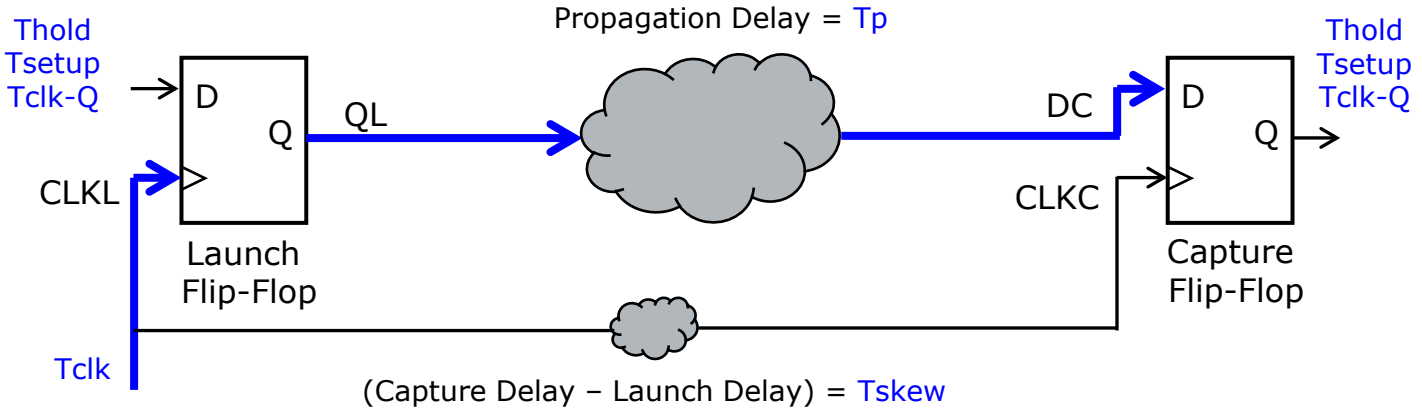


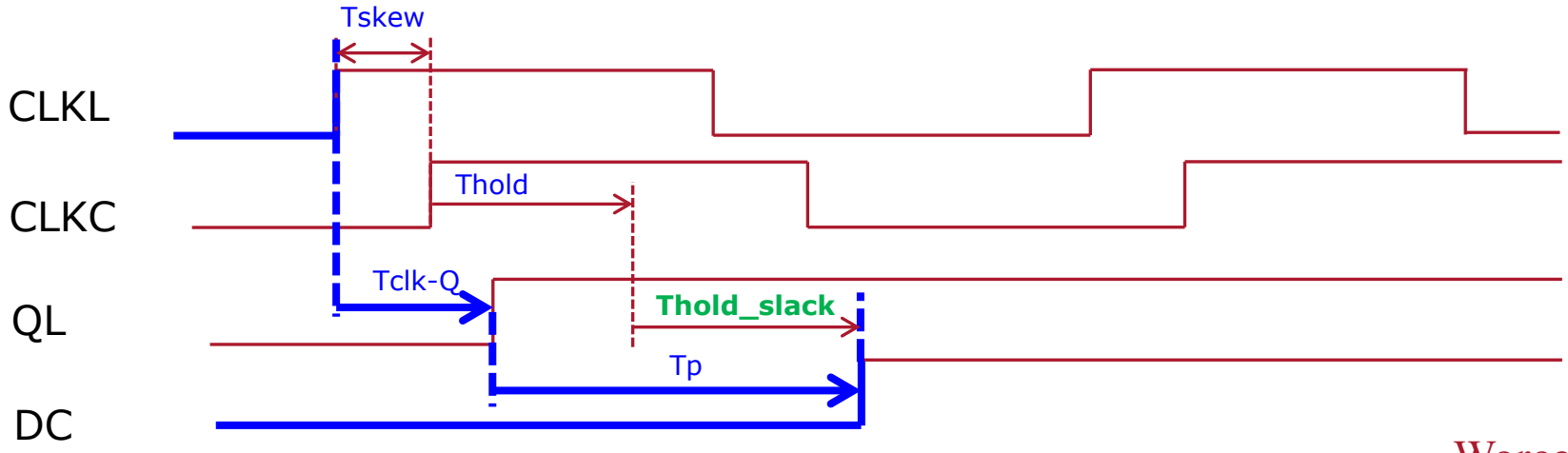
No Hold Timing Violations



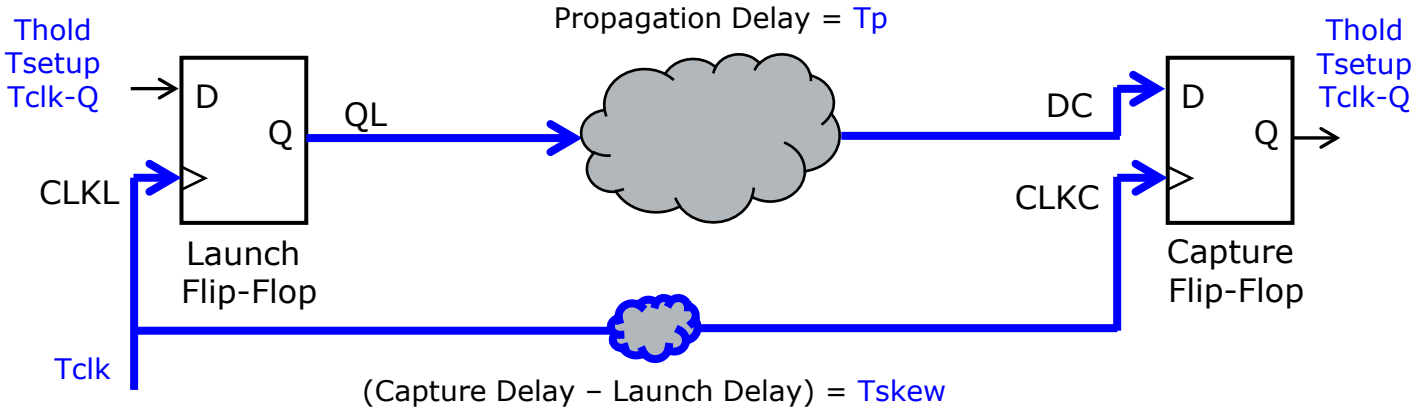


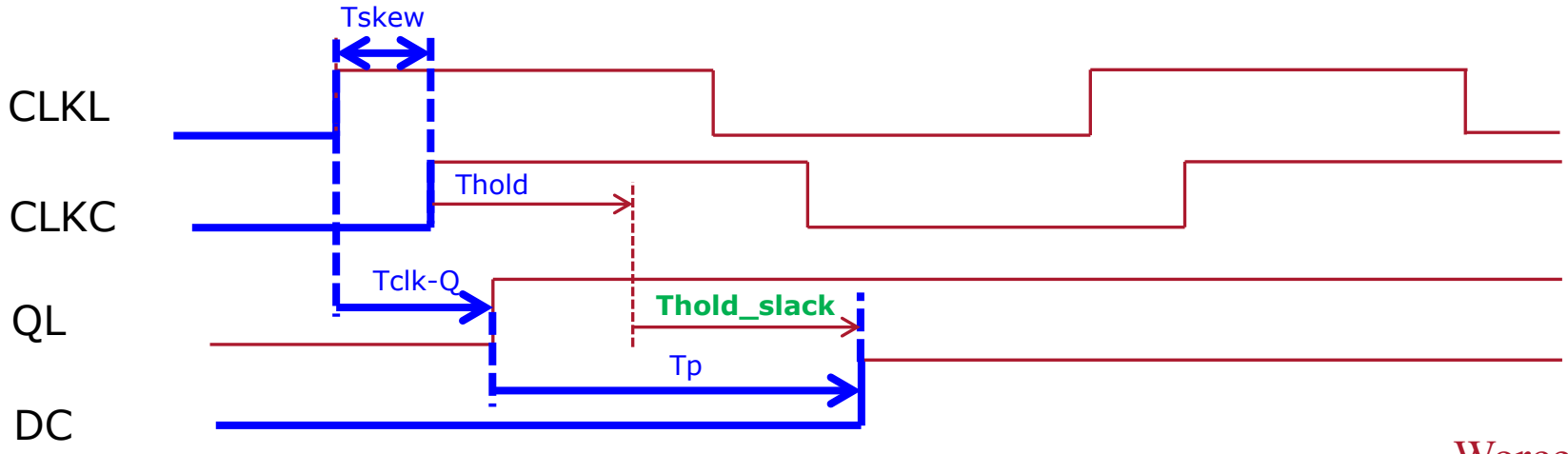
No Hold Timing Violations



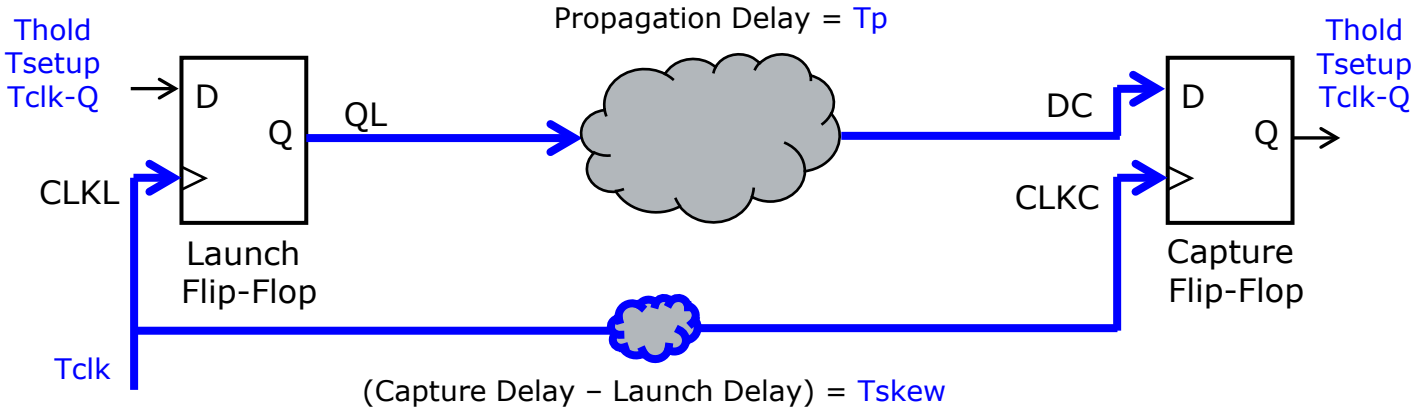


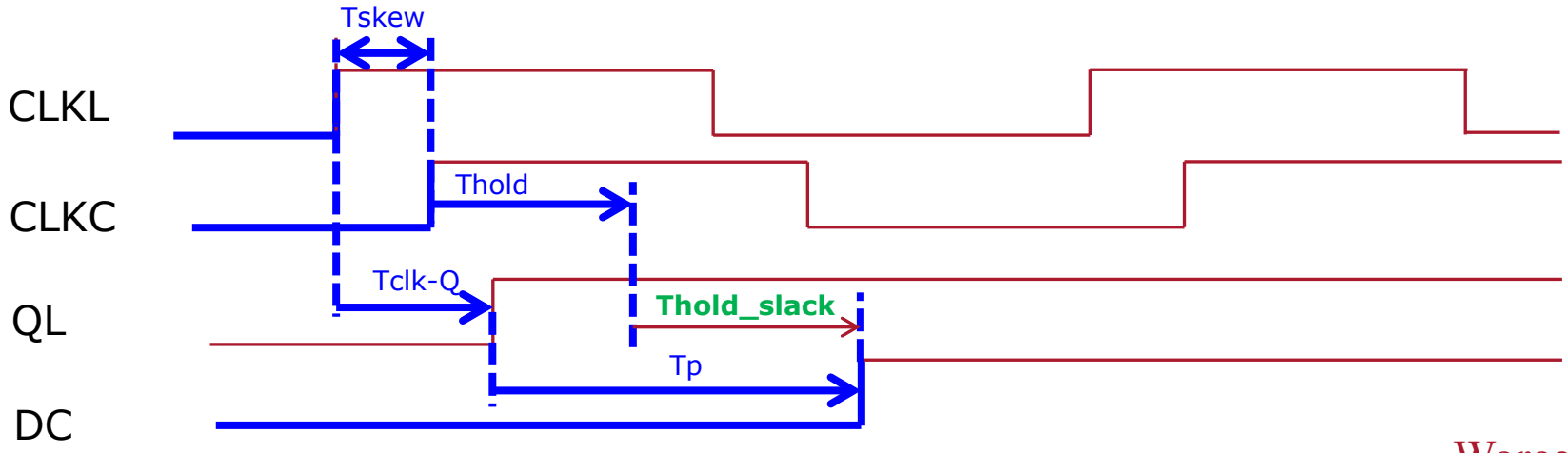
No Hold Timing Violations



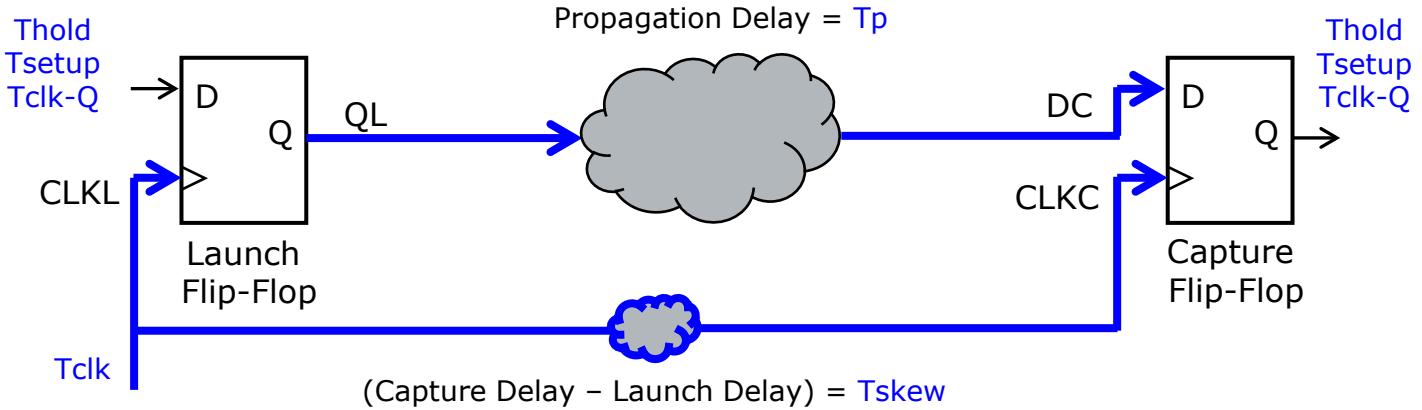


No Hold Timing Violations



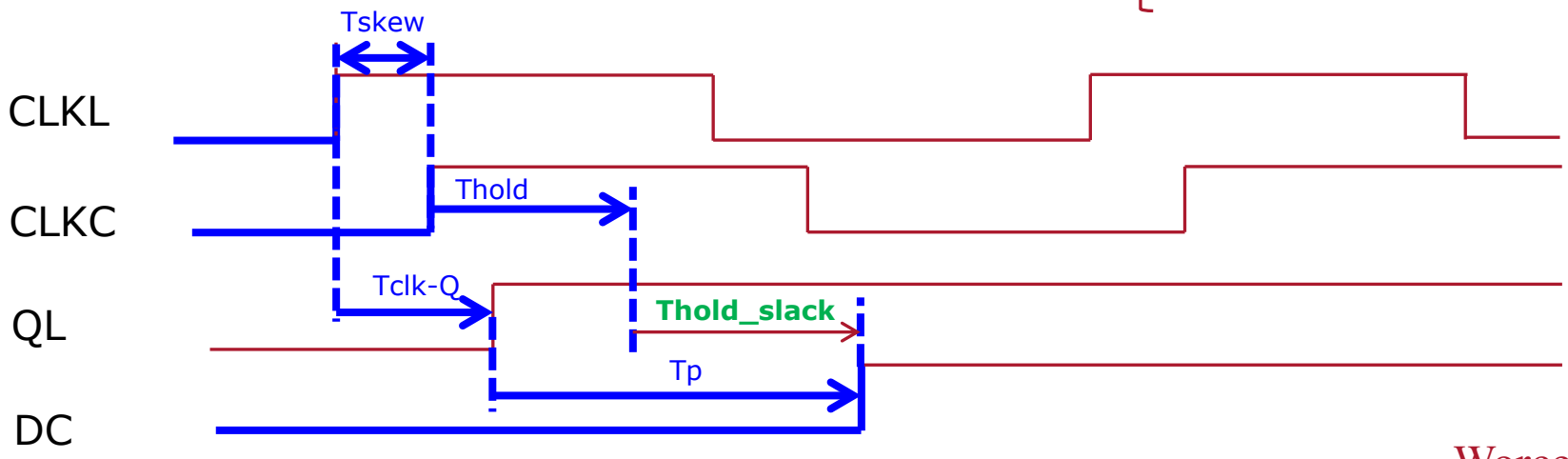


No Hold Timing Violations

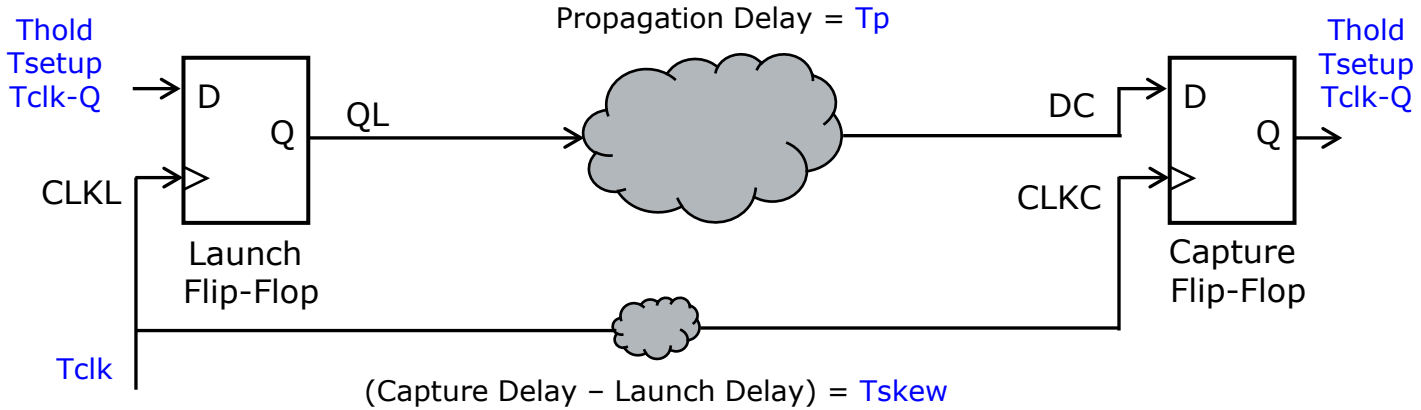


Thold_slack = Tclk-Q + Tp + Tskew – Thold

>0	**Positive Slack, OK**
= 0	**Zero Slack, Critical**
< 0	**Negative Slack, Violation**



Minimum Clock Period, Minimum Tp



"No Setup Timing Violations" requires a minimum clock period

TCLK,min = Tskew + Tp + Tclk-Q + Tsetup

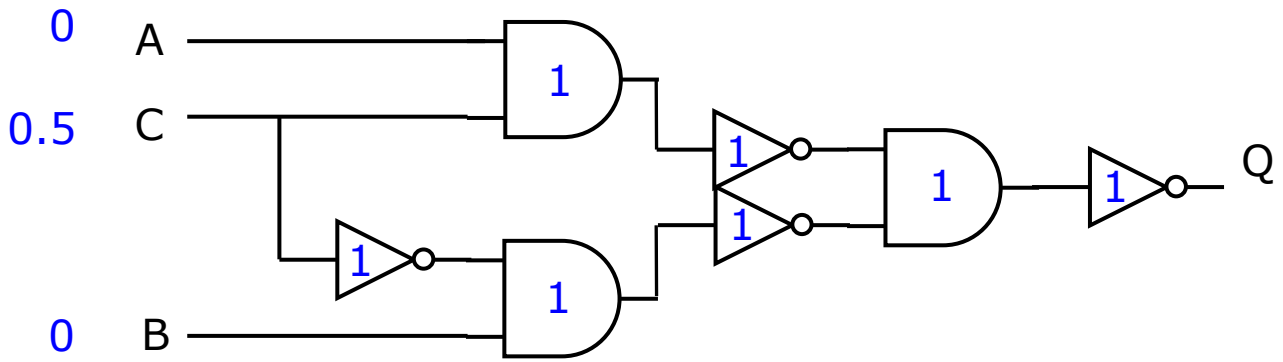
"No Hold Timing Violations" requires a minimum propagation delay

Tp,min = Thold + Tskew – Tclk-Q

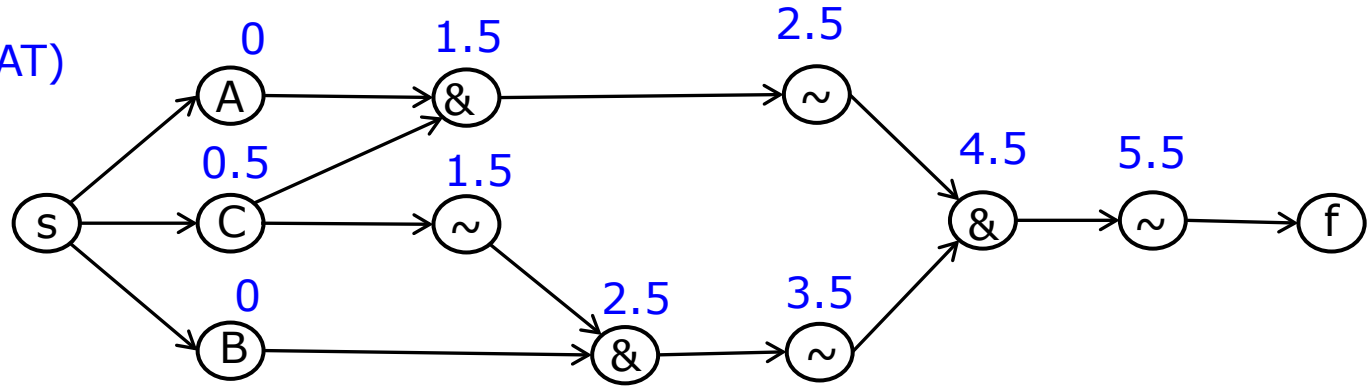
Static Timing Analysis

- For a given synchronous digital logic design and a given clock period Tclk, find the critical path = slowest possible path in the design
  - The Actual Arrival Time (AAT) = the time when a gate will switch
  - The Required Arrival Time (RAT) = the time when a gate should have switched
  - (setup) Slack = RAT – AAT
  - The critical path contains those gates that have minimal slack

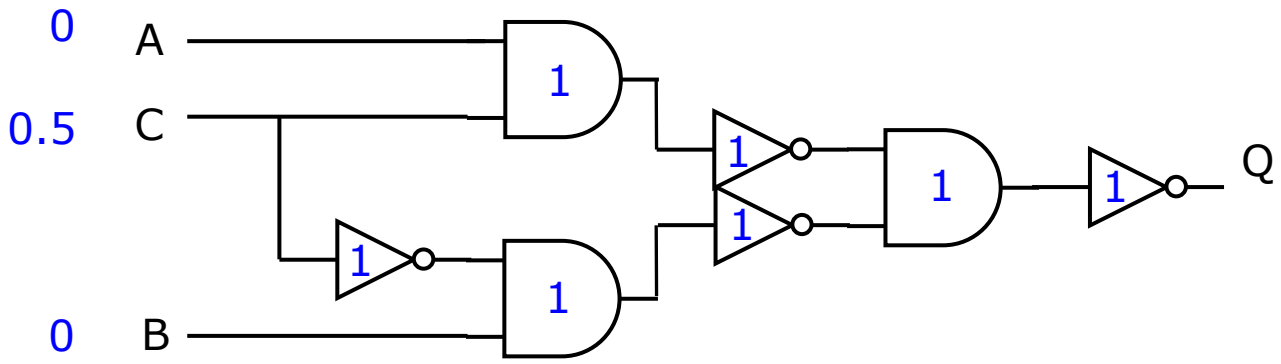
Static Timing Analysis Example



Actual Arrival Time (AAT)



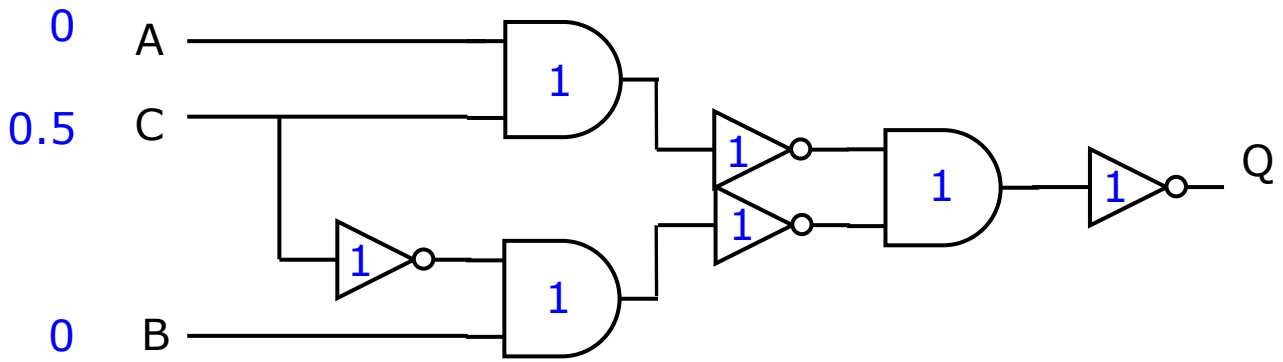
Static Timing Analysis Example



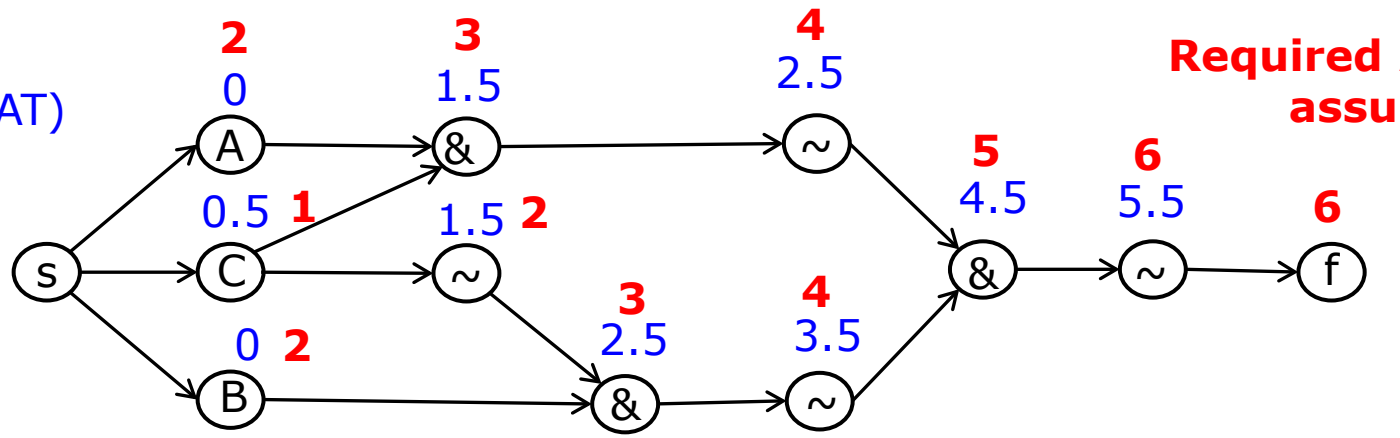
Required Arrival Time (AAT) = time when a node output must be known

For node v, u = succ(v): RAT(v) = min (RAT(u) - delay(u,v))

Static Timing Analysis Example



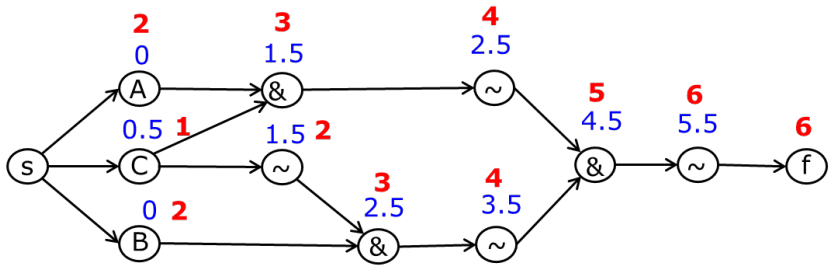
Actual Arrival Time (AAT)

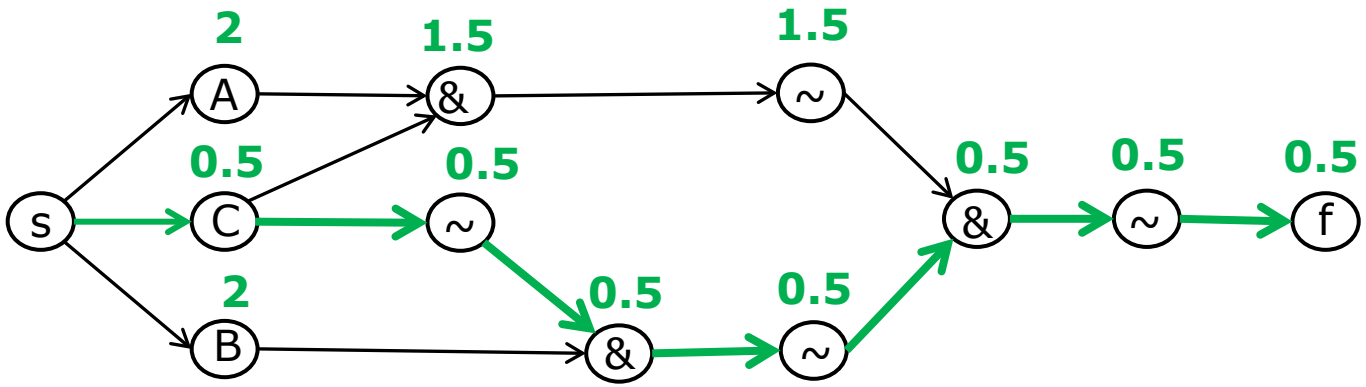


Required Arrival Time (RAT) assuming Tclk = 6

Static Timing Analysis Example

- Slack = RAT - AAT

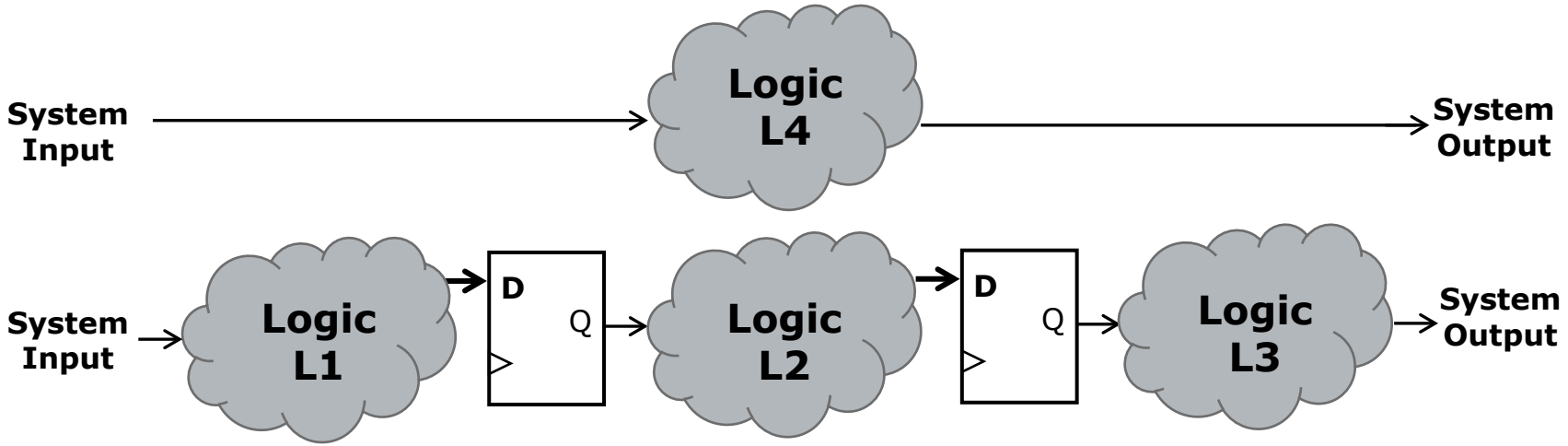




Critical Path = path with nodes with minimum slack

Static Timing Analysis for Synchronous Logic

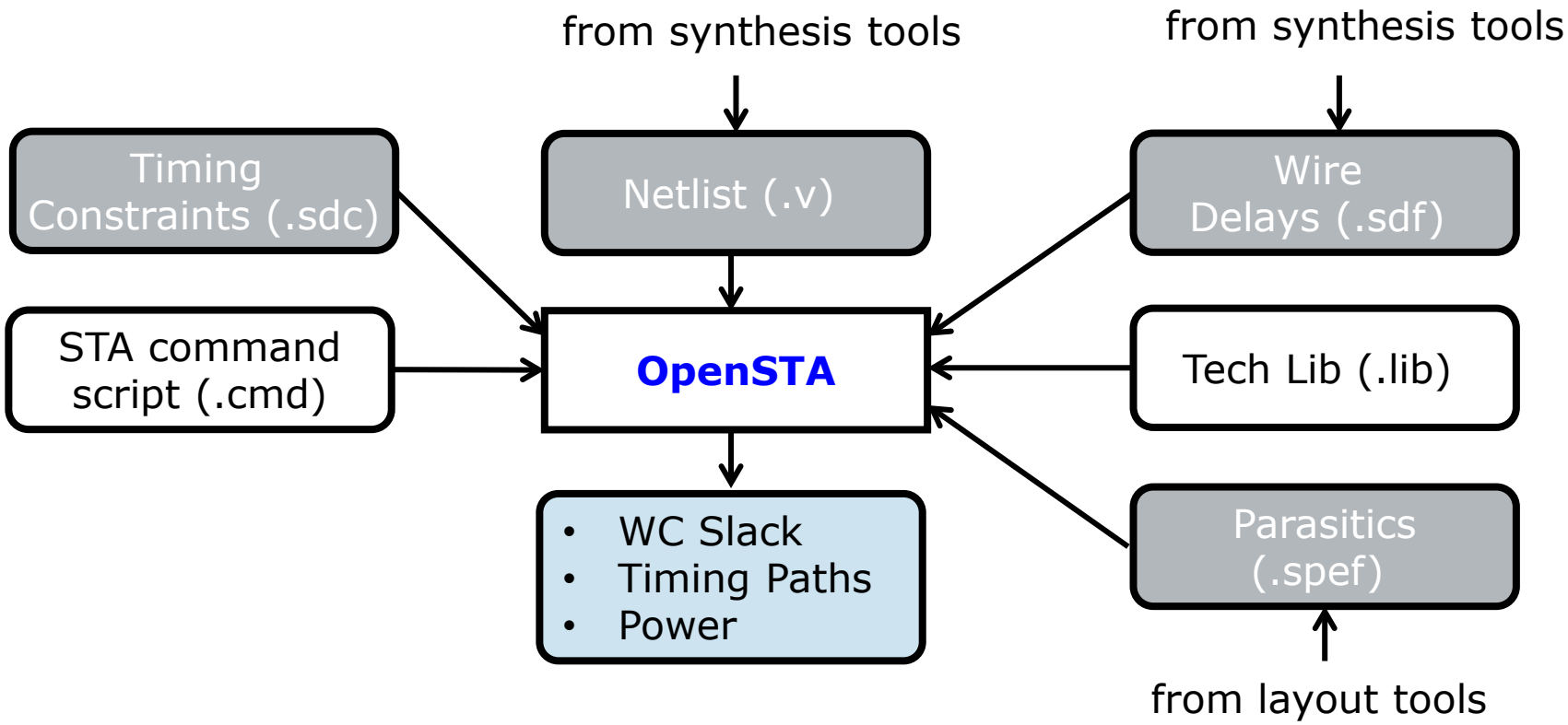
- To verify the overall design, you verify system output slack, and flip-flop slack



- **System-level critical path** is the path of minimum slack from any (system input/flop output) to any (system output/flop input)

OpenSTA

- Static Timing Analysis Tool to compute slack and power



https://github.com/The-OpenROAD-Project/OpenSTA

Sample Timing Constraints File

- At minimum, identifies clock(s) and clock period(s), input delay(s) and output delay(s)

```
current_design sbox
set clk_period 5

create_clock -name clk -period $clk_period {clk}
set non_clock_inputs [lsearch -inline -all -not -exact [all_inputs] clk]
set_input_delay 0 -clock clk $non_clock_inputs
set_output_delay 0 -clock clk [all_outputs]
```

Sample Timing Analysis Output

```
  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock clk (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 v input external delay
   0.00    0.00 v in[7] (in)
   0.43    0.43 ^ _372_/Y (sky130_fd_sc_hd__xnor2_1)
   0.19    0.63 ^ _379_/Y (sky130_fd_sc_hd__xnor2_1)
   0.30    0.93 ^ _380_/X (sky130_fd_sc_hd__lpflow_clkbufkapwr_1)
   0.16    1.08 v _386_/Y (sky130_fd_sc_hd__nand3_1)
   0.44    1.53 ^ _469_/Y (sky130_fd_sc_hd__a21oi_1)
   0.11    1.63 v _475_/Y (sky130_fd_sc_hd__nor3_1)
   0.31    1.94 v _507_/X (sky130_fd_sc_hd__or3_1)
   0.33    2.27 v _514_/X (sky130_fd_sc_hd__a221o_1)
   0.28    2.54 ^ _525_/Y (sky130_fd_sc_hd__a221oi_2)
   0.12    2.66 v _538_/X (sky130_fd_sc_hd__xor2_1)
   0.39    3.05 ^ _573_/Y (sky130_fd_sc_hd__xnor2_1)
   0.46    3.51 v _650_/X (sky130_fd_sc_hd__xnor3_1)
   0.00    3.51 v _693_/D (sky130_fd_sc_hd__dfxtp_1)
           3.51   data arrival time
```

Path Segments

Sample Timing Analysis Output

```
  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock clk (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 v input external delay
   0.00    0.00 v in[7] (in)
   0.43    0.43 ^ _372_/Y (sky130_fd_sc_hd__xnor2_1)
   0.19    0.63 ^ _379_/Y (sky130_fd_sc_hd__xnor2_1)
   0.30    0.93 ^ _380_/X (sky130_fd_sc_hd__lpflow_clkbufkapwr_1)
   0.16    1.08 v _386_/Y (sky130_fd_sc_hd__nand3_1)
   0.44    1.53 ^ _469_/Y (sky130_fd_sc_hd__a21oi_1)
   0.11    1.63 v _475_/Y (sky130_fd_sc_hd__nor3_1)
   0.31    1.94 v _507_/X (sky130_fd_sc_hd__or3_1)
   0.33    2.27 v _514_/X (sky130_fd_sc_hd__a221o_1)
   0.28    2.54 ^ _525_/Y (sky130_fd_sc_hd__a221oi_2)
   0.12    2.66 v _538_/X (sky130_fd_sc_hd__xor2_1)
   0.39    3.05 ^ _573_/Y (sky130_fd_sc_hd__xnor2_1)
   0.46    3.51 v _650_/X (sky130_fd_sc_hd__xnor3_1)
   0.00    3.51 v _693_/D (sky130_fd_sc_hd__dfxtp_1)
           3.51   data arrival time
```

Output Y of cell _386_:

```
sky130_fd_sc_hd__nand3_1 _386_ (
  .A(_297_),
  .B(_301_),
  .C(_302_),
  .Y(_303_)
);
```

Sample Timing Analysis Output

```
  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock clk (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 v input external delay
   0.00    0.00 v in[7] (in)
   0.43    0.43 ^ _372_/Y (sky130_fd_sc_hd__xnor2_1)
   0.19    0.63 ^ _379_/Y (sky130_fd_sc_hd__xnor2_1)
   0.30    0.93 ^ _380_/X (sky130_fd_sc_hd__lpflow_clkbufkapwr_1)
   0.16    1.08 v _386_/Y (sky130_fd_sc_hd__nand3_1)
   0.44    1.53 ^ _469_/Y (sky130_fd_sc_hd__a21oi_1)
   0.11    1.63 v _475_/Y (sky130_fd_sc_hd__nor3_1)
   0.31    1.94 v _507_/X (sky130_fd_sc_hd__or3_1)
   0.33    2.27 v _514_/X (sky130_fd_sc_hd__a221o_1)
   0.28    2.54 ^ _525_/Y (sky130_fd_sc_hd__a221oi_2)
   0.12    2.66 v _538_/X (sky130_fd_sc_hd__xor2_1)
   0.39    3.05 ^ _573_/Y (sky130_fd_sc_hd__xnor2_1)
   0.46    3.51 v _650_/X (sky130_fd_sc_hd__xnor3_1)
   0.00    3.51 v _693_/D (sky130_fd_sc_hd__dfxtp_1)
           3.51   data arrival time
```

Output Y of cell _386_:

```
sky130_fd_sc_hd__nand3_1 _386_ (
  .A(_297_),
  .B(_301_),
  .C(_302_),
  .Y(_303_)
);
```

net _303_

Output Y of cell _469_:

```
sky130_fd_sc_hd__a21oi_1 _469_ (
  .A1(_295_),
  .A2(_303_),
  .B1(_309_),
  .Y(_044_)
);
```

Sample Timing Analysis Output

```
...
   0.28    2.54 ^ _525_/Y (sky130_fd_sc_hd__a221oi_2)
   0.12    2.66 v _538_/X (sky130_fd_sc_hd__xor2_1)
   0.39    3.05 ^ _573_/Y (sky130_fd_sc_hd__xnor2_1)
   0.46    3.51 v _650_/X (sky130_fd_sc_hd__xnor3_1)
   0.00    3.51 v _693_/D (sky130_fd_sc_hd__dfxtp_1)
           3.51   data arrival time


   5.00    5.00   clock clk (rise edge)
   0.00    5.00   clock network delay (ideal)
   0.00    5.00   clock reconvergence pessimism
           5.00 ^ _693_/CLK (sky130_fd_sc_hd__dfxtp_1)
  -0.13    4.87   library setup time
           4.87   data required time
-----------------------------------------------------
           4.87   data required time
          -3.51   data arrival time
-----------------------------------------------------
           1.36   slack (MET)
```

The slack analysis is repeated for every "path group" which includes, e.g.:

- All paths from inputs to register inputs
- All paths from reg inputs to reg outputs
- All paths from reg outputs to outputs
- All paths from inputs to outputs

Analysis can also take care of special cases:

- Asynchronous inputs
- Multiple clocks
- Multi-cycle operations
- Operating conditions (mult-corner analysis)

WPI

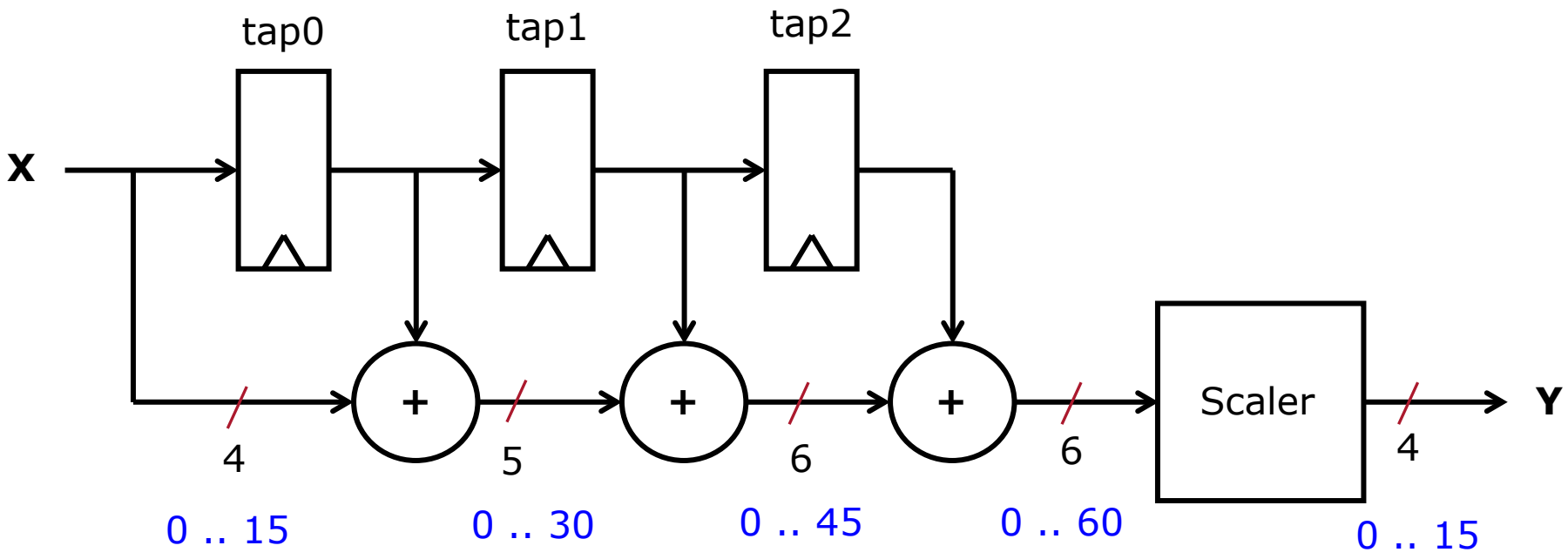
Part II Handson

Analyzing the critical path

Objectives of Handson II

1. Use openSTA
2. Analyze and explain the output of openSTA
3. Examine the impact of INPUTDELAY/OUTPUTDELAY on the critical path
4. Analyze the critical path on an alternate RTL design

Moving Average



scaler: (in + 2) / 4

Run the RTL simulation

```
root@ubuntu-s-2vcpu-4gb-nyc1-01:~/crypto-asic-oss# make -f Makefile.mavg rtlsim
cd mavg/work && make rtlsim
make[1]: Entering directory '/root/crypto-asic-oss/mavg/work'
iverilog -y ../rtl ../sim/tb.v
./a.out && mv trace.vcd rtl.vcd && rm -f a.out
VCD info: dumpfile trace.vcd opened for output.
x  0 y  x
x  0 y  0
x 15 y  0
x 15 y  4
x 15 y  8
x 15 y 11
x 15 y 15
x 15 y 15
x  0 y 15
x  0 y 11
```

Run the gate level synthesis

```
root@ubuntu-s-2vcpu-4gb-nyc1-01:~/crypto-asic-oss# make -f Makefile.mavg synthes
is
test -d mavg/work || mkdir -p mavg/work
python3 scripts/gen_make_design.py
cd mavg/work && make synthesis
make[1]: Entering directory '/root/crypto-asic-oss/mavg/work'
yosys -s synth.ys

 /----------------------------------------------------------------------------\
 |                                                                            |
 |  yosys -- Yosys Open SYnthesis Suite                                       |
 |                                                                            |
 |  Copyright (C) 2012 - 2019  Clifford Wolf <clifford@clifford.at>           |
 |                                                                            |
 |  Permission to use, copy, modify, and/or distribute this software for any  |
```

Inspect intermediate results

```
root@ubuntu-s-2vcpu-4gb-nyc1-01:~/crypto-asic-oss# ls mavg/work/
constraint.sdc  makefile  netlist.json  netlist.v  sta.cmd  synth.ys
root@ubuntu-s-2vcpu-4gb-nyc1-01:~/crypto-asic-oss#
```

netlist.v: gate-level netlist
synth.ys: synthesis script
constraint.sdc: clock constraints for synthesis

Hints:

- Inspect netlist.v in an editor. Study the naming convention for nets and cells.
- Inspect synt.ys in an editor. A list of yosys commands can be found in https://yosyshq.net/yosys/files/yosys_manual.pdf
- You can rerun the synthesis by hand using
  yosys –s synth.ys

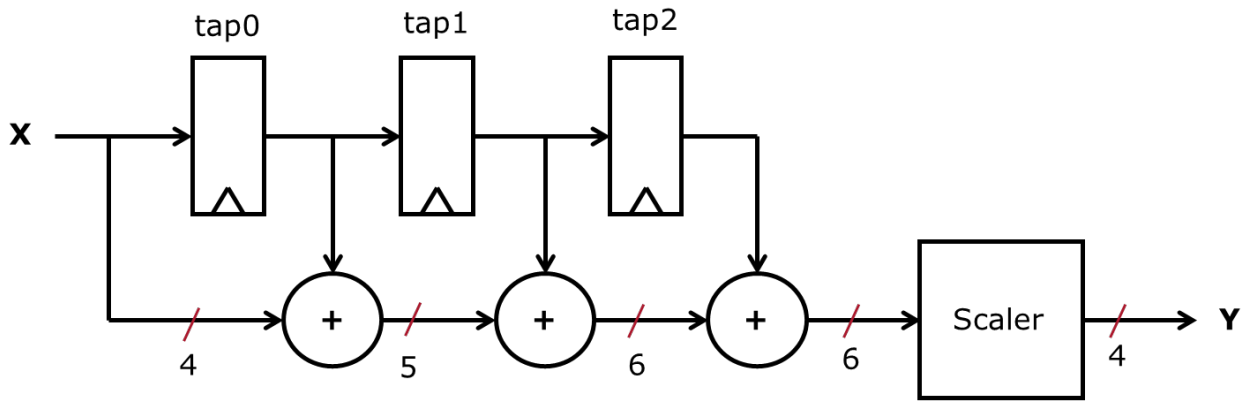
Run Static Timing Analysis

```
root@ubuntu-s-2vcpu-4gb-nyc1-01:~/crypto-asic-oss# make -f Makefile.mavg gltiming
cd mavg/work && make gltiming
make[1]: Entering directory '/root/crypto-asic-oss/mavg/work'
sta <sta.cmd
OpenSTA 2.4.0 555493cba6 Copyright (c) 2021, Parallax Software, Inc.
License GPLv3: GNU GPL version 3 <http://gnu.org/licenses/gpl.html>

This is free software, and you are free to change and redistribute it
under certain conditions; type `show_copying' for details.
This program comes with ABSOLUTELY NO WARRANTY; for details type `show_warranty'.
Warning: /root/skywater-pdk/libraries/sky130_fd_sc_hd/latest/timing/sky130_fd_sc_hd__tt_025C_1
v80.lib line 23, default_fanout_load is 0.0.
Warning: set_input_delay relative to a clock defined on the same port/pin not allowed.
Startpoint: _131_ (rising edge-triggered flip-flop clocked by clk)
Endpoint: y[2] (output port clocked by clk)
Path Group: clk
Path Type: max
```

Question 1

- Study the cells of the critical path and mark on the figure where exactly the critical path starts and ends, and which components are included
  - Hint: use netlist.v and your knowledge of digital design to sketch the path

```
  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock clk (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 ^ _131_/CLK (sky130_fd_sc_hd__dfxtp_1)
   0.36    0.36 ^ _131_/Q (sky130_fd_sc_hd__dfxtp_1)
   0.38    0.74 v _083_/X (sky130_fd_sc_hd__xor3_1)
   0.36    1.09 v _098_/X (sky130_fd_sc_hd__maj3_2)
   0.17    1.26 v _108_/X (sky130_fd_sc_hd__lpflow_inputiso0n_1)
   0.19    1.45 v _115_/X (sky130_fd_sc_hd__o22a_1)
   0.18    1.63 v _119_/X (sky130_fd_sc_hd__o21a_1)
   0.14    1.77 ^ _123_/Y (sky130_fd_sc_hd__o211ai_2)
   0.13    1.90 ^ _126_/X (sky130_fd_sc_hd__a22o_1)
   0.00    1.90 ^ y[2] (out)
           1.90   data arrival time
```



Question 2

- Use static timing analysis to determine the slack for the following four cases
  - Hint: Modify Makefile.mavg to change the INPUTDELAY and OUTPUTDELAY
  - Hint: For each new timing analysis, clear out previous results as follows:
    `make -f Makefile.mavg clean gltiming`

	input_delay = 0	input_delay = 1
output_delay = 0	2.10	
output_delay = 1		

Question 3

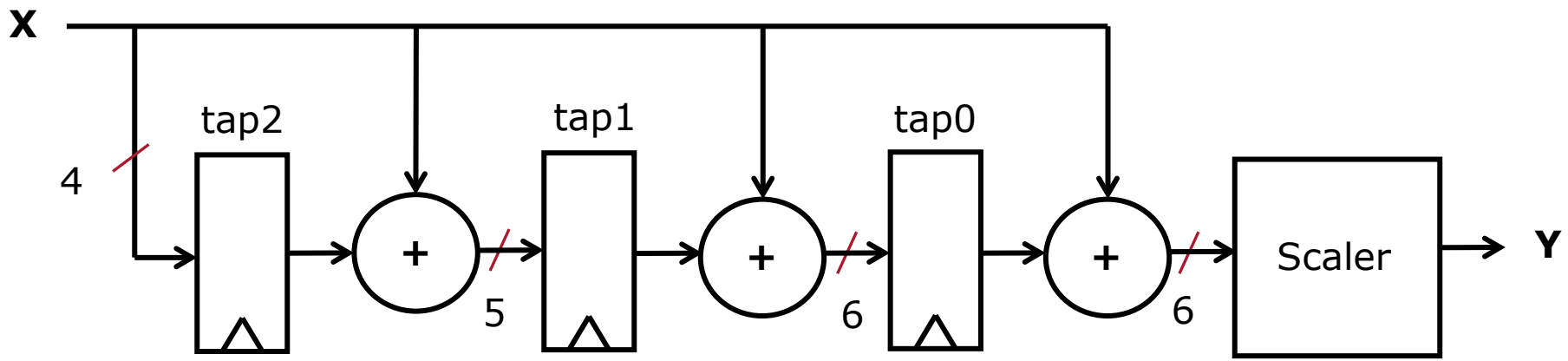
- Does changing the OUTPUTDELAY parameter change the segments included in the critical path? Why (not)?

Question 4

- Does changing the INPUTDELAY parameter change the segments included in the critical path? Why (not)?

Bonus Question

- mavg2 implements the transpose version of mavg



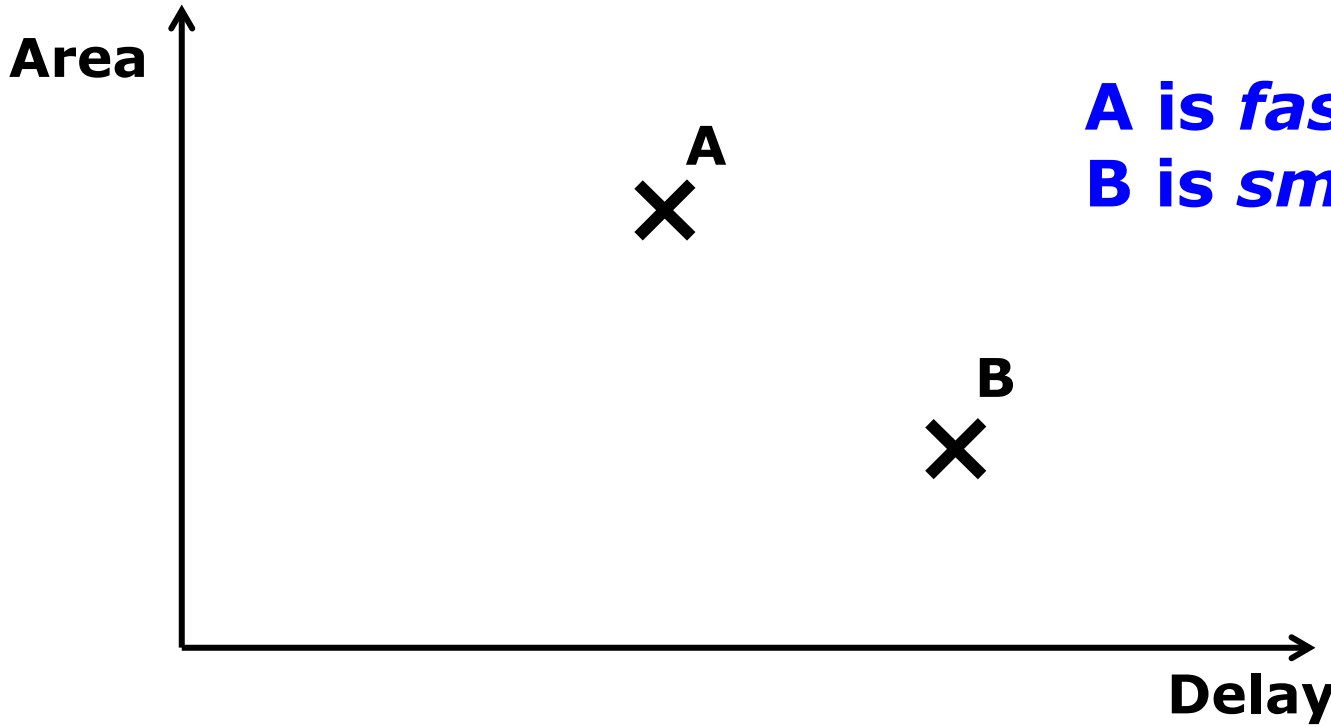
scaler: (in + 2) / 4

- Analyze the critical path of the transpose design
- Compare the area, cell count and critical path of mavg and mavg2
- What can you conclude?

WPI

Part III
Quality Metrics for Cryptographic Hardware

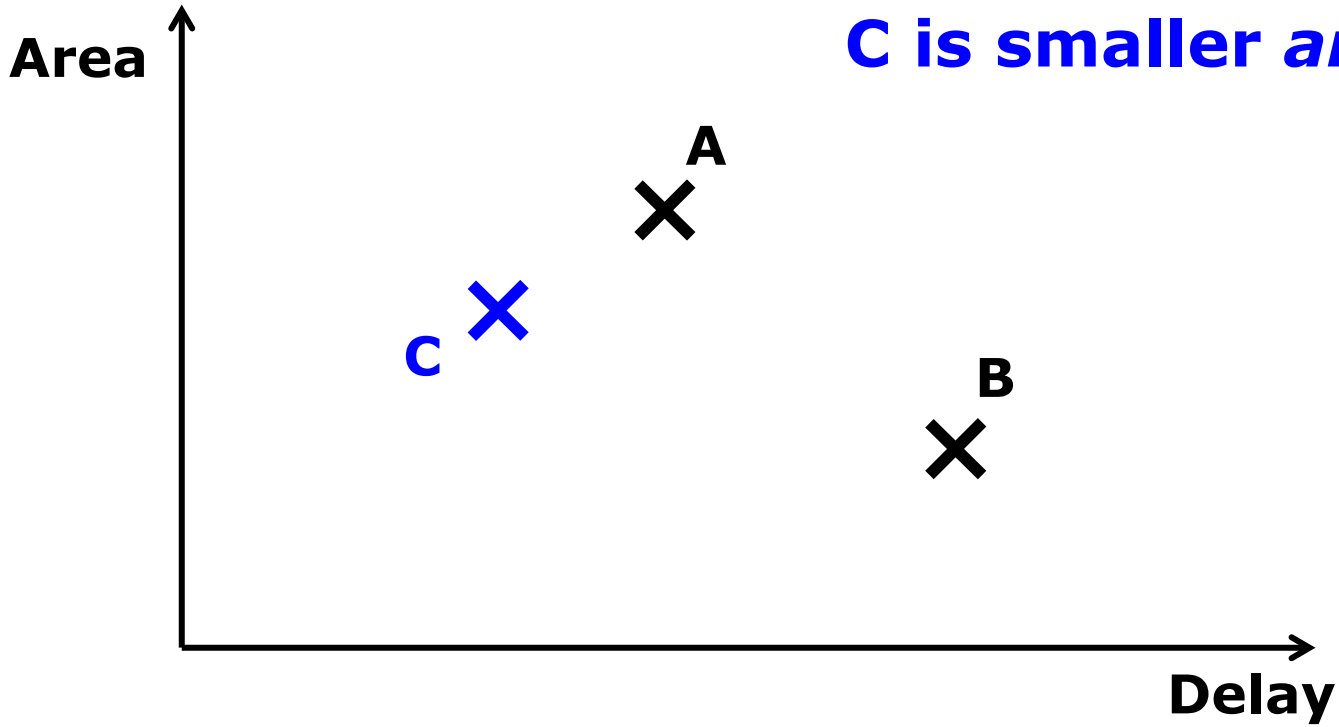
Comparing Hardware Designs



A is *faster* than B
B is *smaller* than A

It's impossible to say which one is better: A or B.
Depending on your design target, you may prefer design A or design B

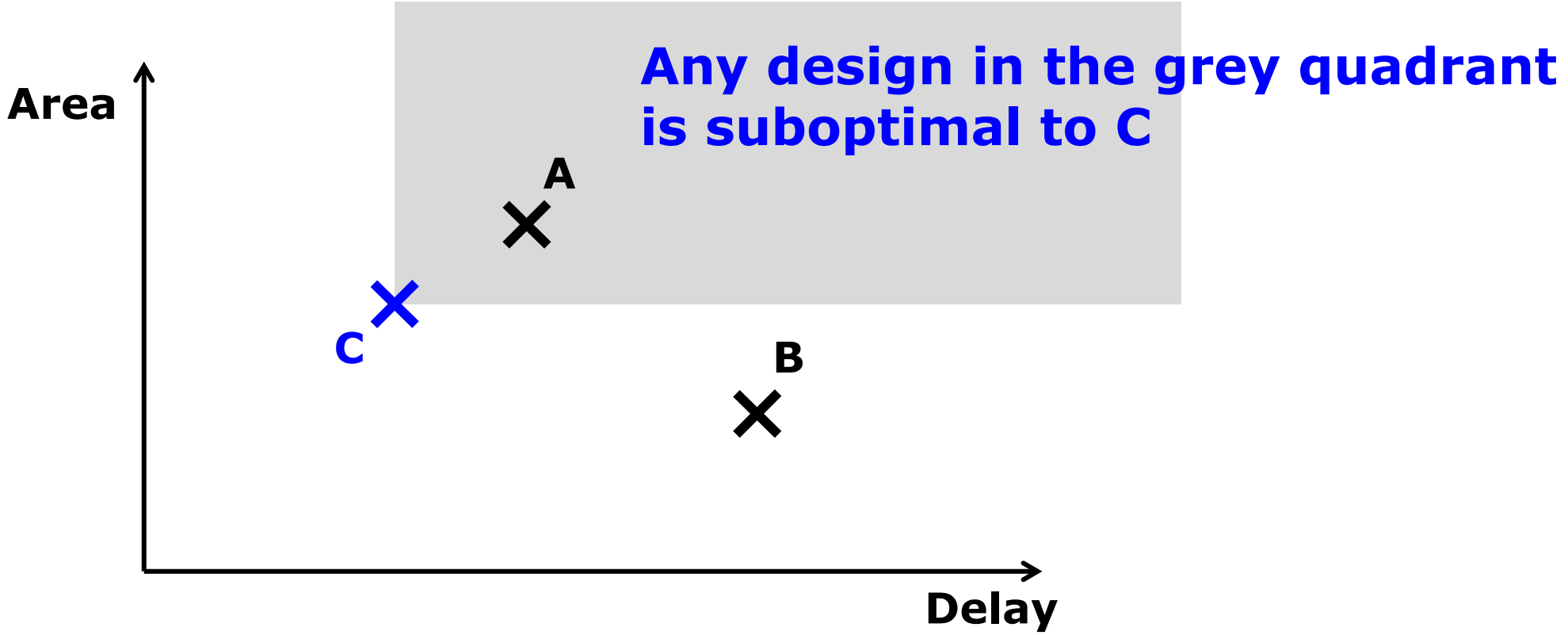
Comparing Hardware Designs



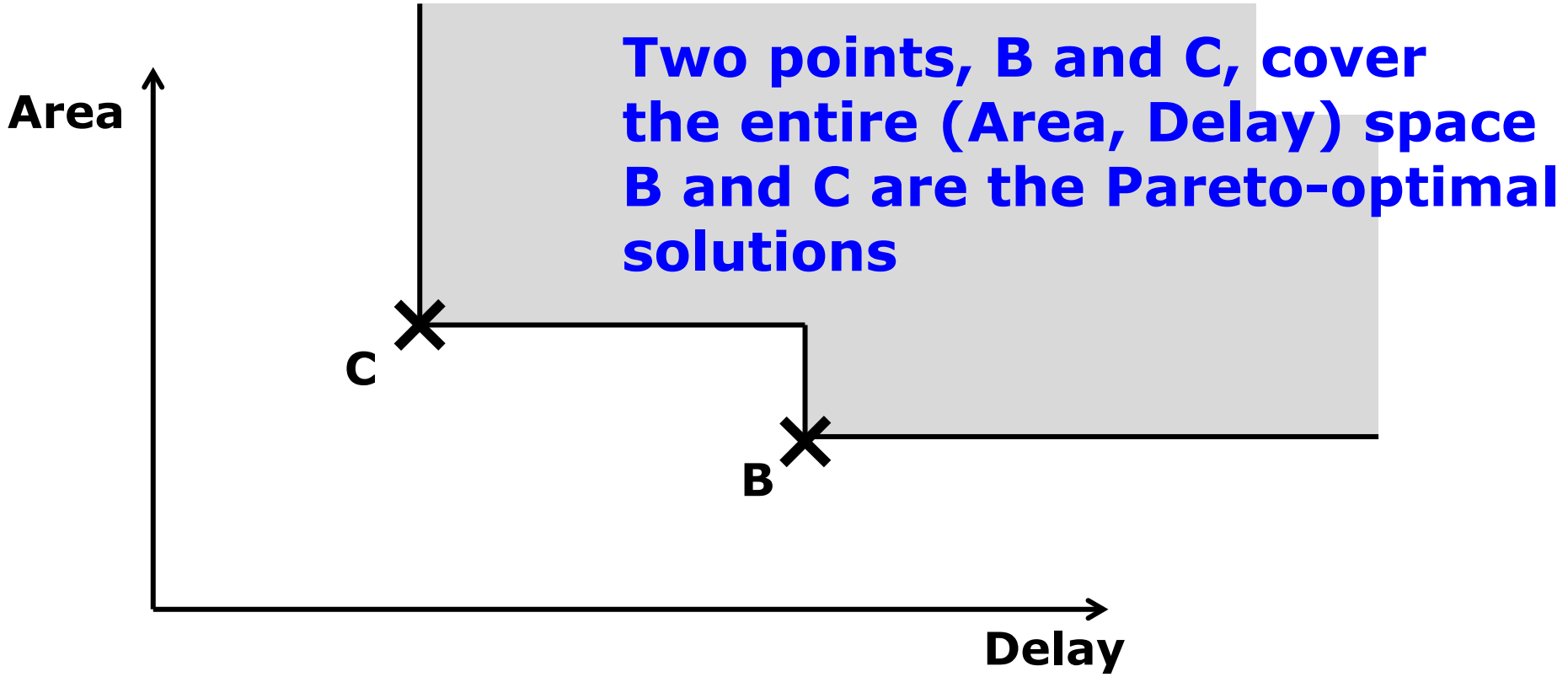
C is smaller *and* faster than A

C is always a better choice than A
Hence, design A is irrelevant for further consideration

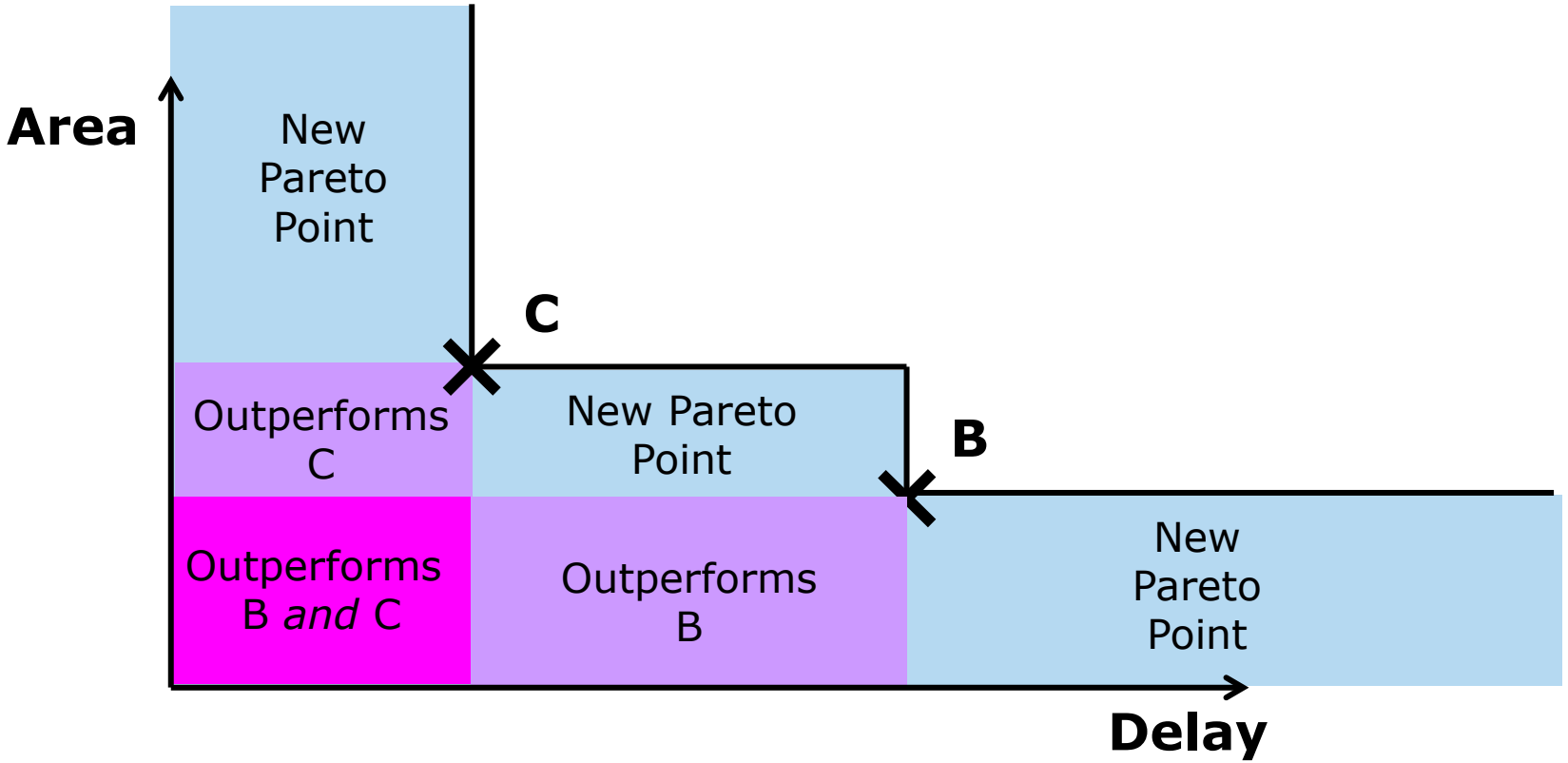
Comparing Hardware Designs



Comparing Hardware Designs



Comparing Hardware Designs



Any design that claims to outperform B or C must lie in the colored area

Pareto-optimal design

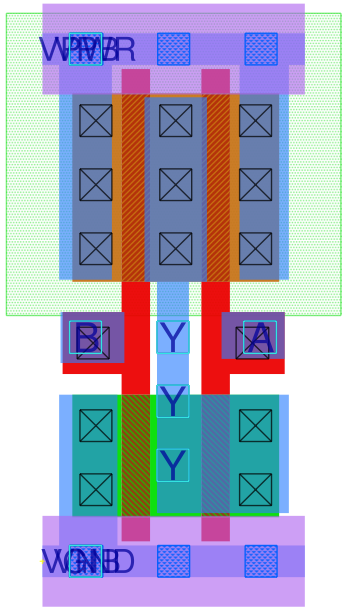
- The design space of a hardware implementation is characterized by a limited number of Pareto-optimal points

- There may be additional criteria besides area and delay
  - In particular, clock frequency, and hardware interfaces are constrained by system integration characteristics.
  - Example: Clock for passive RFID crypto derived from RF carrier
  - Example: A RISCV coprocessor uses a 32-bit bus

- Conversely, one could consider *only* low delay, or *only* low-area.
  - Such designs show what *can* be achieved, but they have limited practical use

Measuring Area: Gate Equivalent (GE)

- A trick to express silicon area in a "technology-independent" manner is to define a standard gate (e.g. NAND2 with drive 1)
- Then the equivalent gate count of the complete design is given by



$$GE = \frac{\text{Area of Design}}{\text{Area of NAND2}_1}$$

Gate Equivalent (GE) Pitfalls

- Post-synthesis and Post-layout GE counts are different
- Several versions of a standard cell library may exist
  - E.g. Skywater 130 comes in 7 different versions, with a NAND2-1

Version	GE unit (sq micron)
high density	3.7536
high speed	4.7952
low power	4.7952
low speed	4.7952
medium speed	4.7952
high density low leakage	5.0048
high voltage	9.7680

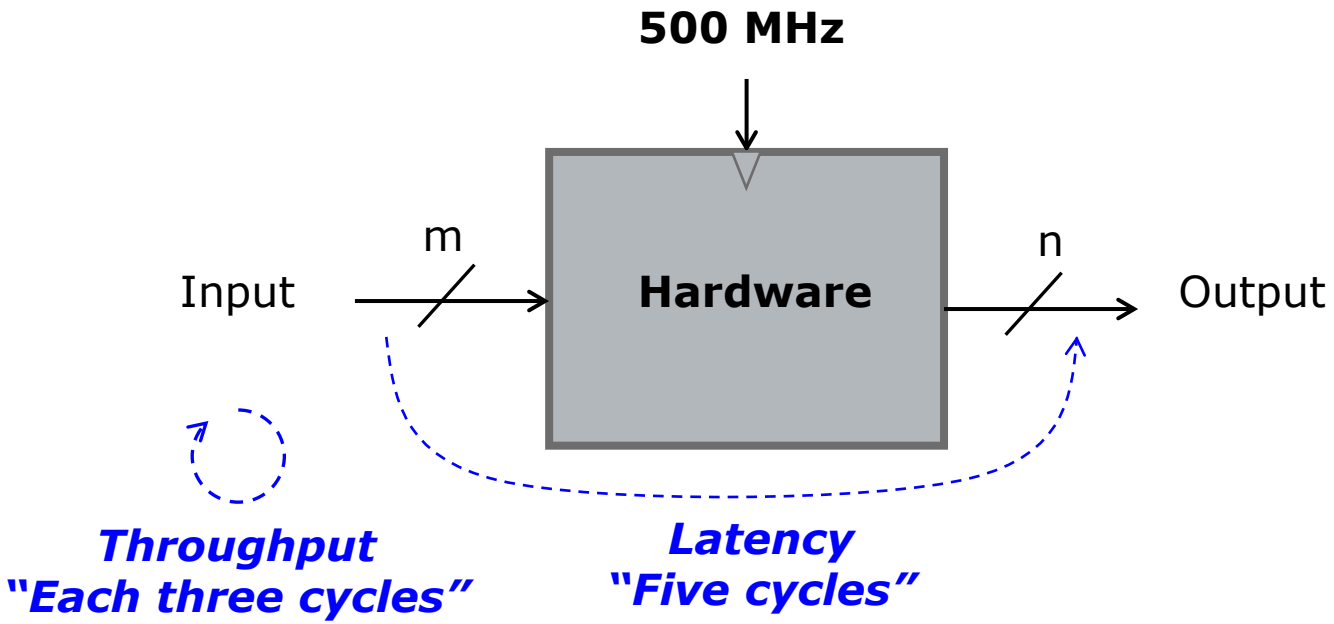
Gate Equivalent (GE) Pitfalls (2)

- Over different libraries, GE variation is even more likely
  - Different libraries may have a different set of cells types
  - Different technologies may use different interconnect (affecting utilization)
  - Different technologies use different design rules and cell topologies
  - The set of drive strengths available per cell varies with the library

Recommendation:
- When you list GE, always list the exact technology, library, version
- Don't compare GE across technologies, across design flows, across libraries
- When in doubt, list area in sq micron, not in GE

Performance

- The critical path (or maximum clock frequency) is a popular measure in hardware performance
- Critical path does not reflect throughput or latency



Performance

- Performance depends on the environment
- Timing Analysis will evaluate multiple *corners*

```
define_corners typ wc bc
read_liberty -corner typ .../sky130_fd_sc_hd__tt_025C_1v80.lib
read_liberty -corner wc .../sky130_fd_sc_hd__ss_100C_1v40.lib
read_liberty -corner bc .../sky130_fd_sc_hd__ff_n40C_1v76.lib

...

report_checks -corner bc
report_checks -corner typ
report_checks -corner wc

report_power -corner bc
report_power -corner typ
report_power -corner wc
```

Performance

- Performance depends on the environment
- Timing Analysis will evaluate multiple *corners*

```
define_corners typ wc bc
read_liberty -corner typ .../sky130_fd_sc_hd__tt_025C_1v80.lib
read_liberty -corner wc .../sky130_fd_sc_hd__ss_100C_1v40.lib
read_liberty -corner bc .../sky130_fd_sc_hd__ff_n40C_1v76.lib

...

report_checks -corner bc
report_checks -corner typ
report_checks -corner wc

report_power -corner bc
report_power -corner typ
report_power -corner wc
```

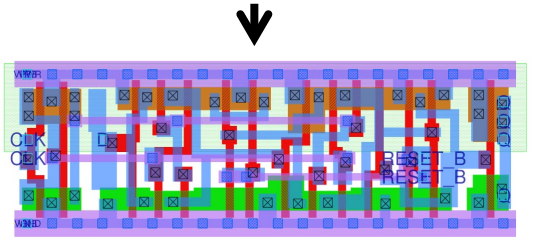
SBOXAES corner	Slack (ns)	Total Power (mW)
WC	-4.56	1.47
TYP	0.30	2.47
BC	0.93	2.28

And what about Power?

- See last year's talk on "Tools and Methods for Pre-silicon Analysis of Secure Hardware"

  https://summerschool-croatia.cs.ru.nl/2022/slides/Patrick.pdf

- Key points:
  - Power Consumption in CMOS is tied in first order to transitions
  - Delay modeling tools can also handle gate-level power modeling
  - Reported Cell Power has three components: switching, internal, leakage power



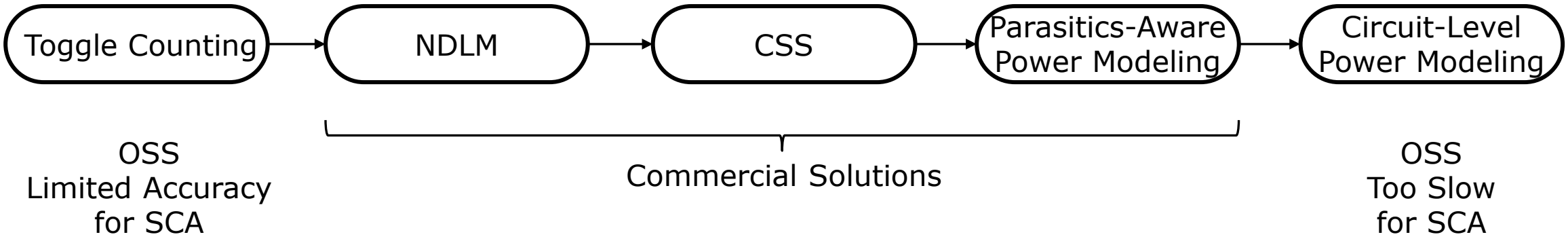
Static Timing Analysis tools report Power

Group	Internal Power	Switching Power	Leakage Power	Total Power (Watts)	
Sequential	4.18e-03	4.39e-04	2.65e-08	4.62e-03	9.0%
Combinational	1.78e-02	2.88e-02	4.43e-08	4.66e-02	91.0%
Macro	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Pad	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Total	2.19e-02	2.92e-02	7.08e-08	5.12e-02	100.0%
	42.9%	57.1%	0.0%		

- Power is computed using a default activity (0.1 for inputs at 50% duty cycle), and activities are propagated probabilistically through the network

Tools for Side Channel Power Analysis

- There is an extensive amount of work on pre-silicon tooling for side-channel analysis (https://ileanabuhan.github.io/Tools/)
- However, very few OSS frameworks for cycle-level power estimation
  - CASCADE - https://github.com/dsijacic/CASCADE
  - TOFU - https://gitlab.lrz.de/tueisec/tofu



Toggle Counting

- Use the activity of gates (recorded during a simulation) as a metric for their energy/power consumption
  - Every transition represents a unit energy
  - Transitions per time unit represent a relative power metric
- Unit-toggle metric misses to capture
  - Standard cell drive capability
  - Standard cell fanout
  - Standard cell internal activity (internal power)
- But, still useful as a first evaluation when cell-level modeling detail is available

WPI

Part III Handson

Power to the Presilicon People!

Objectives of Handson II

1. Take an AES core through synthesis, simulation, chip design
2. Perform power estimation using toggle counting
3. Perform simple power analysis to identify main chip activities

picoaes design

- Encryption/Decryption Core with 32-bit datapath integrated on a 32-bit microprocessor bus
- Core design by Joachim Strombergsson (https://github.com/secworks/aes)
- Five cycles per round (4x4 sbox lookup + rest of round)
- Offline keyschedule

Design Tasks: RTL simulation

```
# make -f Makefile.picoaes rtlsim
cd picoaes/work && make rtlsim
make[1]: Entering directory '/root/crypto-asic-oss/picoaes/work'
iverilog -y ../rtl ../sim/test_picoaes.v
./a.out && mv trace.vcd rtl.vcd && rm -f a.out
VCD info: dumpfile trace.vcd opened for output.
Plaintext:  16b576b600a49804d81267644b80e292
Key:        fb0b38bcad60b76c73377dfd9ce5692f
Ciphertext: 33b661a74d164dc7b811f54fe5a5832c
Ciphertext CORRECT
Plaintext:  fb8587bdac1c369369173bceb2ed4785
Key:        fb0b38bcad60b76c73377dfd9ce5692f
Ciphertext: 2287d7fc410a4e2059c15b4a2a2b3375
Ciphertext CORRECT
```

Design Tasks: Logic Synthesis

```
# make -f Makefile.picoaes synthesis
...
29. Printing statistics.

=== picoaes ===

   Number of wires:                 17898
   Number of wire bits:             21066
   Number of public wires:             62
   Number of public wire bits:       2591
   Number of memories:                  0
   Number of memory bits:               0
   Number of processes:                 0
   Number of cells:                 20992
...
```

Design Tasks: Gate level simulation

```
# make -f Makefile.picoaes glsim
...
./a.out && mv trace.vcd netlist.vcd && rm -f a.out
VCD info: dumpfile trace.vcd opened for output.
Plaintext:  16b576b600a49804d81267644b80e292
Key:        fb0b38bcad60b76c73377dfd9ce5692f
Ciphertext: 33b661a74d164dc7b811f54fe5a5832c
Ciphertext CORRECT
Plaintext:  fb8587bdac1c369369173bceb2ed4785
Key:        fb0b38bcad60b76c73377dfd9ce5692f
Ciphertext: 2287d7fc410a4e2059c15b4a2a2b3375
Ciphertext CORRECT
```

-> This simulation generates work/netlist.vcd

Design Tasks: Toggle count analysis using Tofu

First, copy the gate-level VCD to a working directory in tofu

```
# cd ~/tofu
# cp -r example picoaes
# cd picoaes
# cp ../crypto-asic-oss/picoaes/work/netlist.vcd .
```

-> This simulation generates work/netlist.vcd

Design Tasks: Toggle count analysis using Tofu

Next, modify settings_example.json

```
{
    "vcdGlob": "netlist.vcd",                          <- Change this line as shown
    "pickleGlob": "netlist.pickle",                    <- Change this line as shown
    "signalsFileNameLiterals": "signals_name.json",
    "signalsFileName": "signals.json",
    "signalPropertiesFile": "signal_properties.pickle",
    "leakageModel": "HammingDistance",
    "window": false,
    "windowFrom": null,
    "windowTo": null,
    "valueExtractFunction": "valueExtractIndex",
    "writeTraces": true,
    "writeTracesBatchSize": 10,
    "traceFileName": "traces.h5",
    "align": false,
    "downsample": 1,                                   <- Change this line as shown
    "format": "lascar"
}
```

Design Tasks: Toggle count analysis using Tofu

Next, modify signals_name.json

```
{
"include" : [
    "module toptb->module dut"
],
"exclude" : [
]
}
```

<- Change this line as shown

<- Make this line empty as shown

Design Tasks: Toggle count analysis using Tofu

Next, perform toggle count power analysis

```
# make
...
2023-06-02 12:35:13,571 : synthesize.py : INFO : processing: 1 traces
2023-06-02 12:35:13,895 : synthesize.py : INFO : traces consist from toggles of 144801
bits
2023-06-02 12:35:13,922 : synthesize.py : INFO : traces consist from 141167 signals
2023-06-02 12:35:14,238 : synthesize.py : INFO : traces consist from 379 sample points in
time
2023-06-02 12:35:14,275 : synthesize.py : INFO : ----------
2023-06-02 12:35:14,275 : synthesize.py : INFO : capturing trace: 0
2023-06-02 12:35:14,275 : synthesize.py : INFO : reloading numeric ids
2023-06-02 12:35:14,294 : synthesize.py : INFO : extracting state updates from picklefile
2023-06-02 12:35:14,474 : synthesize.py : INFO : iterating through all state updates
2023-06-02 12:35:14,474 : synthesize.py : INFO : performing 1034875 updates
2023-06-02 12:35:15,608 : synthesize.py : INFO : capturing trace took 1.332801 seconds
2023-06-02 12:35:15,610 : synthesize.py : INFO : ----------
2023-06-02 12:35:15,610 : synthesize.py : INFO : synthesize traces from pickles finished
```

Design Tasks: Toggle count analysis using Tofu

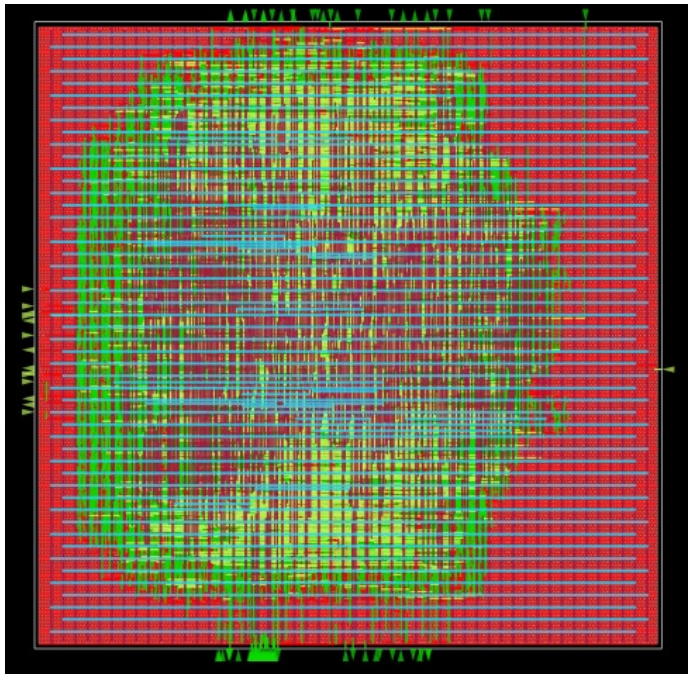
Finally, you can visualize the toggle trace output using Python/Matplotlib

```
# python3
import h5py
import matplotlib.pyplot as plt
f = h5py.File('traces.h5','r')
dset = f['leakages']
plt.plot(dset[0])
plt.show()
```

Design Tasks: Create chip (optional)

This takes a long time (~40 minutes), so you may want to start this immediately in a separate terminal and let it run while you work on toggle counting

```
# make -f Makefile.picoaes openroad
# make -f Makefile.picoaes chip
# make -f Makefile.picoaes chipgui
```



Question 1

- The power plot shows the power from two encryptions that use the same key
  - Plaintext: 16b576b600a49804d81267644b80e292
    Key: fb0b38bcad60b76c73377dfd9ce5692f
  - Plaintext: fb8587bdac1c369369173bceb2ed4785
    Key: fb0b38bcad60b76c73377dfd9ce5692f
- Identify all relevant AES processing stages from the power consumption
- Next, observe that one of these two encryptions shows an anomaly in the power trace. Can you explain what is happening?

Question 2

- Observe that one of these two encryptions shows an anomaly in the power trace. Can you explain what is happening?

WPI

Logging in to a cloud design server

Login command

- Log in to the server (assuming IP 123.123.123.123) as follows:

  ssh –X root@123.123.123.123

  - Use the IP address of your assigned design server
  - The password is whoneedspublickeyswithapasswordlikethat
  - If you are on Windows, I recommend MobaXterm
  - If you are on Apple, install an X server before logging in

- The design servers will be taken offline at the end of the day
  - To set up your own environment, consult 'Configuring a Design Workstation'
  - The tutorial materials are on https://github.com/Secure-Embedded-Systems/crypto-asic-oss

WPI

Configuring the Design Workstation

Workstation for Open-Source ASIC Design

- Basic Hardware
  - Dual-CPU
  - 4 GB Main Memory
  - 80 GB Disk
- Basic Software
  - Ubuntu 22.04.2 LTS

- Components
  - Icarus Verilog
  - Yosys
  - Parallax Open STA
  - Skywater 130 Std cell library
  - OpenROAD (under docker)

- Installing latest available stable version of every tool

Workstation for Open-Source ASIC Design

```
# Basic upgrade
apt-get update
apt-get upgrade
apt install unzip

# Docker
apt-get install ca-certificates curl gnupg
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
      sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
chmod a+r /etc/apt/keyrings/docker.gpg
echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] \
   https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  tee /etc/apt/sources.list.d/docker.list > /dev/null
apt-get update
apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin \
        docker-compose-plugin
```

Workstation for Open-Source ASIC Design

```
# OpenROAD
git clone --recursive https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts
cd OpenROAD-flow-scripts
./build_openroad.sh
docker image tag openroad/flow-centos7-builder crypto-asic-oss

# iverilog
apt install iverilog gtkwave

# yosys
apt install yosys

# Skywater PDK
apt install make
git clone https://github.com/google/skywater-pdk.git
cd skywater-pdk
#--> !! manually remove yosys and netlistsvg dependencies from skywater-pdk/environment.yml
SUBMODULE_VERSION=latest make submodules -j3 || make submodules -j1
make timing
```

Workstation for Open-Source ASIC Design

```
# OpenSTA
apt install cmake
apt install g++ bison flex swig tcl tcl-dev clang zlib1g-dev
wget https://www.davidkebo.com/source/cudd_versions/cudd-3.0.0.tar.gz
tar zxfv cudd-3.0.0.tar.gz
cd cudd-3.0.0
./configure
make
make check
make install
make clean
cd ..
rm cudd-3.0.0.tar.gz
git clone https://github.com/The-OpenROAD-Project/OpenSTA.git
cd OpenSTA
mkdir build
cd build
cmake ..
make
make install
make clean
cd ..
```

Workstation for Open-Source ASIC Design

```
# netlistsvg
apt install npm
npm install -g netlistsvg

# crypto-asic-oss repo
git clone https://github.com/Secure-Embedded-Systems/crypto-asic-oss.git

# tofu
git clone https://gitlab.lrz.de/tueisec/tofu

# handson assignment
git clone https://github.com/Secure-Embedded-Systems/crypto-asic-oss.git
```

WPI

Spoiler Alert

Assignment Handson I Solution

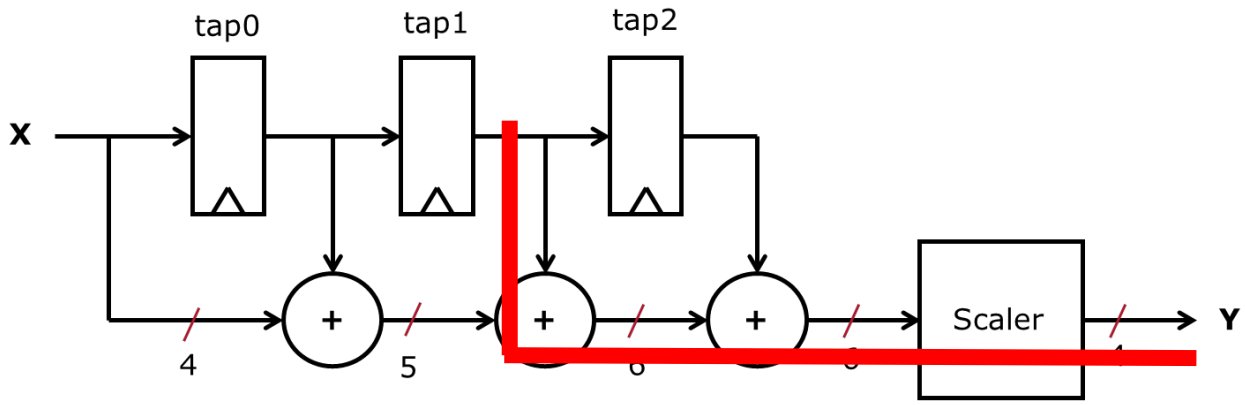
Assuming you do not change the design parameters from the repository, you should find:

	synthesis	design area	utilization (%)
sboxaes	433	3897	50
sboxaeslut	779	5425	41
sboxaespipe	367	2280	29
sboxpresent	40	269	18

Answer Handson II Question 1

- Study the cells of the critical path and mark on the figure where exactly the critical path starts and ends, and which components are included
  - Hint: use netlist.v and your knowledge of digital design to sketch the path

```
  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock clk (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 ^ _131_/CLK (sky130_fd_sc_hd__dfxtp_1)
   0.36    0.36 ^ _131_/Q (sky130_fd_sc_hd__dfxtp_1)
   0.38    0.74 v _083_/X (sky130_fd_sc_hd__xor3_1)
   0.36    1.09 v _098_/X (sky130_fd_sc_hd__maj3_2)
   0.17    1.26 v _108_/X (sky130_fd_sc_hd__lpflow_inputiso0n_1)
   0.19    1.45 v _115_/X (sky130_fd_sc_hd__o22a_1)
   0.18    1.63 v _119_/X (sky130_fd_sc_hd__o21a_1)
   0.14    1.77 ^ _123_/Y (sky130_fd_sc_hd__o211ai_2)
   0.13    1.90 ^ _126_/X (sky130_fd_sc_hd__a22o_1)
   0.00    1.90 ^ y[2] (out)
           1.90   data arrival time
```



Answer Handson II Question 2

- Use static timing analysis to determine the slack for the following four cases
  - Hint: Modify Makefile.mavg to change the INPUTDELAY and OUTPUTDELAY
  - Hint: For each new timing analysis, clear out previous results as follows:
    `make -f Makefile.mavg clean gltiming`

	input_delay = 0	input_delay = 1
output_delay = 0	2.10	1.4
output_delay = 1	1.1	0.4

Answer Handson II Question 3

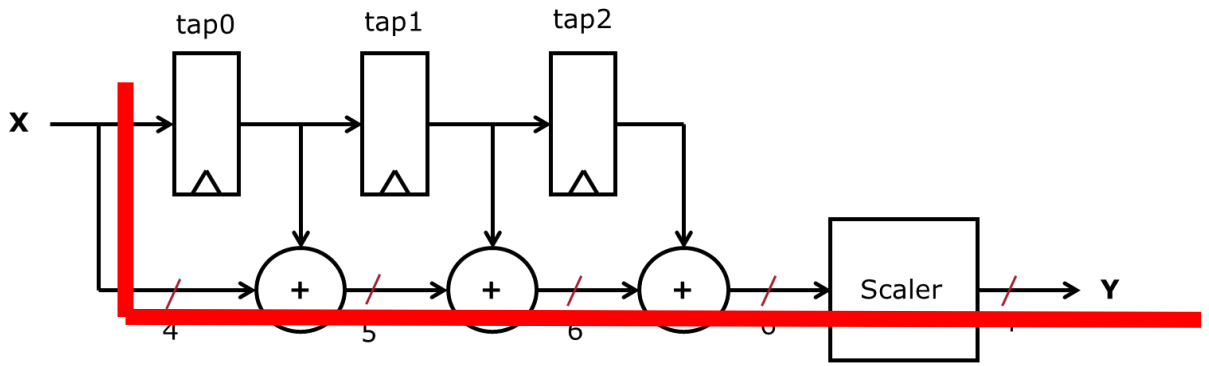
- Does changing the OUTPUTDELAY parameter change the segments included in the critical path? Why (not)?

  The segments of the critical path remains identical. The OUTPUTDELAY merely extends the critical path, since the output Y is already included in the critical path

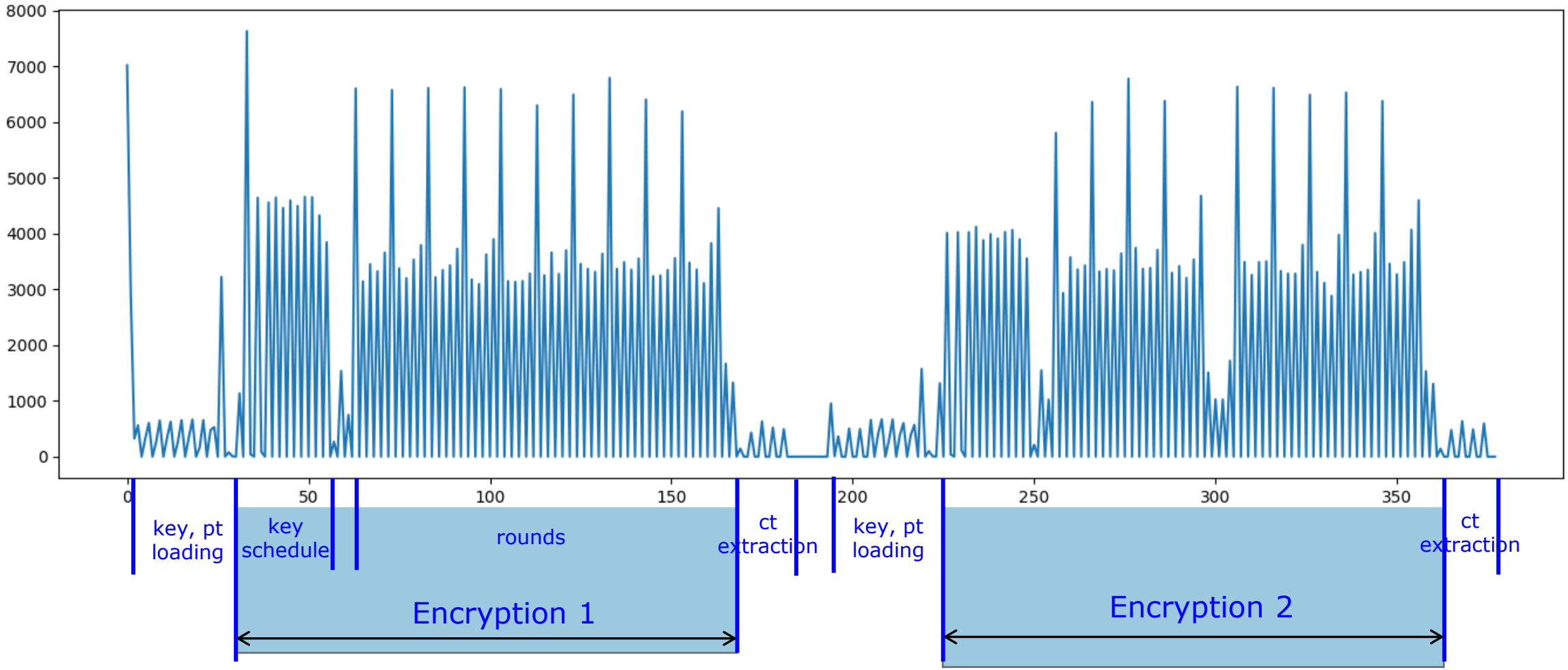
Answer Handson II Question 4

- Does changing the INPUTDELAY parameter change the segments included in the critical path? Why (not)?

  Changing input delay changes the shape of the critical path. Without INPUTDELAY, the primary input X is not included in the critical path. With INPUTDELAY, the primary input X becomes included in the critical path. Note that INPUTDELAY is larger than CLK->Q delay of a flip-flop, making X the slowest input in the circuit.



Answer Handson III Question 1



Answer Handson III Question 2

- Round 5 of the second encryption shows a dip in power consumption.
- The toggle was generated using a Hamming Distance model
- Because the sboxes are computed in groups of 4, a low hamming distance would occur if the four words of the state happen to be identical
- Furthermore, all five clock cycles of round 5 show a lower power consumption. The activity over the entire round causes fewer toggles.
- We conclude that round 5 has a highly biased state, for example all-zeros or all-ones.
- You can verify this by inserting the key/plaintext into an AES calculator such as https://www.nayuki.io/page/aes-cipher-internals-in-excel