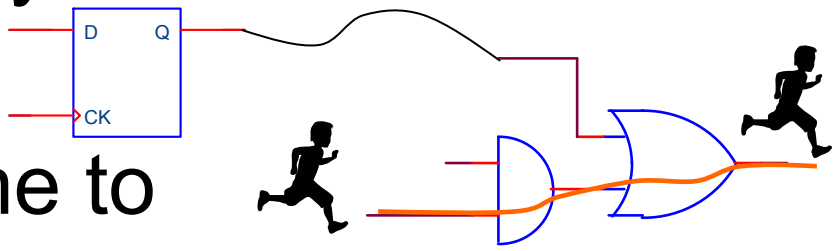
The real hardware

The real hardware

- Propagation delays
  - worst case, typical case, best case…
  - setup/hold, f_{MAX}
- Clock tree and timing analysis
- Power consumption
- Timing of the I/O pins
  - I/O vs. internal registers
  - I/O delays, slew rate settings
- Timing simulations

Back to the real world – timing aspects

- Don't forget, independently of your design entry method, in the real world the design consists of electrically interconnected gates and flip-flops



- The signals need time to propagate through the gates
  - For a pure combinational circuit this can be specified in the delay matrix

Timing aspects – variations (corners)

- The properties of the logic gates and flip-flops are not constant, unlike the electrons, the real gates aren't perfect clonings!
  - Variation inside the same chip
  - Variations chip to chip
  - Variations with the temperature (slower at higher temperatures)
  - Variations with the supply voltage (faster at higher voltages)
  - Dependence on the different capacitive load

Timing aspects – derating factors

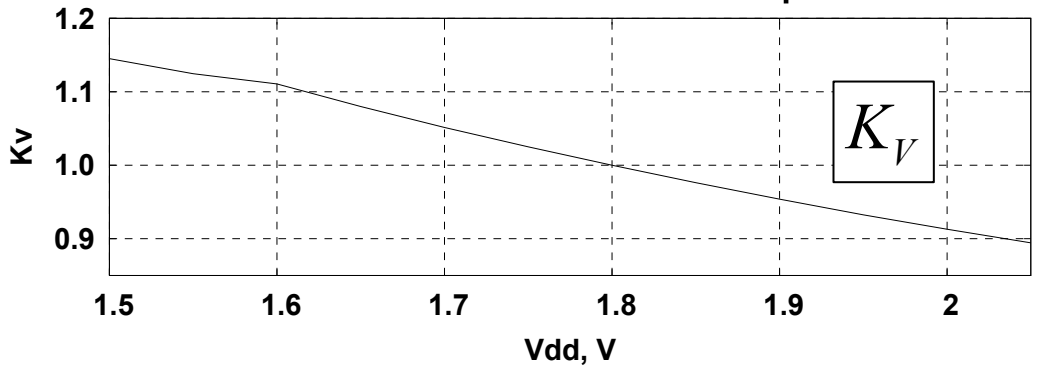
- Derating equation for a single cell

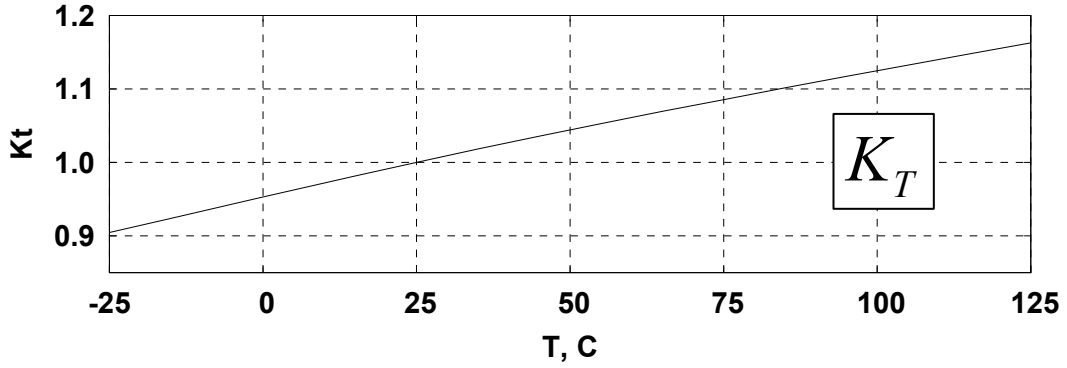
$$Delay(\mathrm{T}, \mathrm{Vdd}, \mathrm{Process}) = Delay_{DATASHEET} \cdot K_T \cdot K_V \cdot K_P$$

Junction Temperature derating factor

Vdd derating factor

Process derating factor

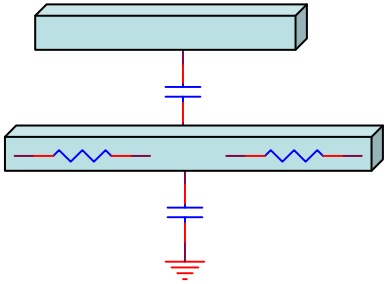


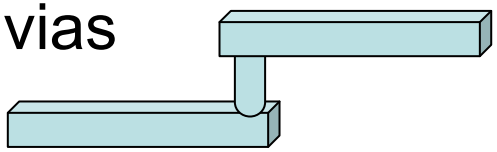


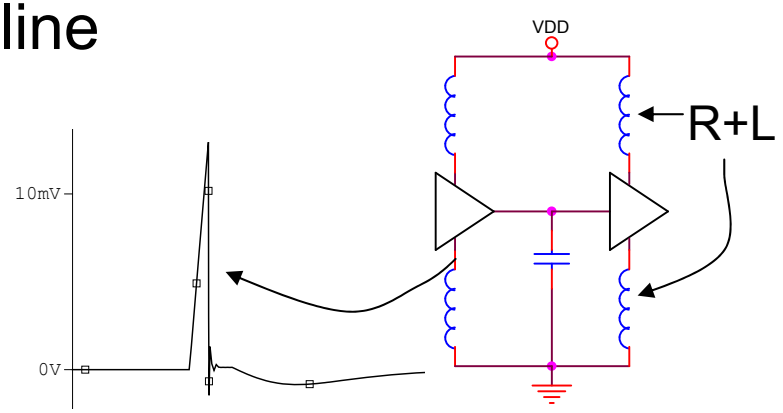
Process	K_P
Slow	1.1989
Typical	1.0000
Fast	0.8648

Timing aspects – routing

- The routing of the signals in the chip is not ideal
  - Capacitive load
  - Capacitive coupling to other signals
  - The Ohmical resistance is not 0
  - Discontinuities at the vias
  - Inductance
  - On the board – transmission line
- The power/gnd is not ideal
  - Inside the chip
  - On the board



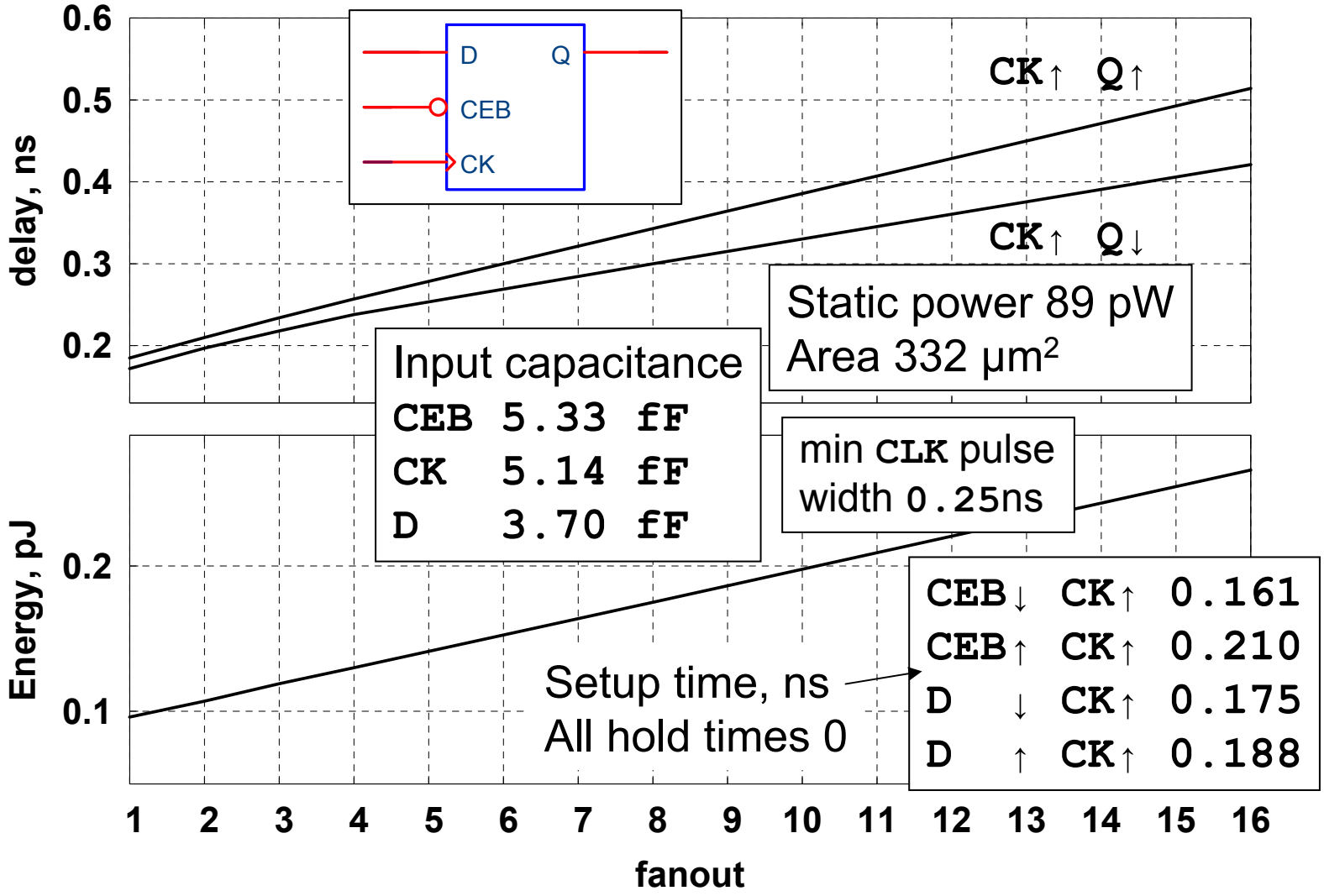




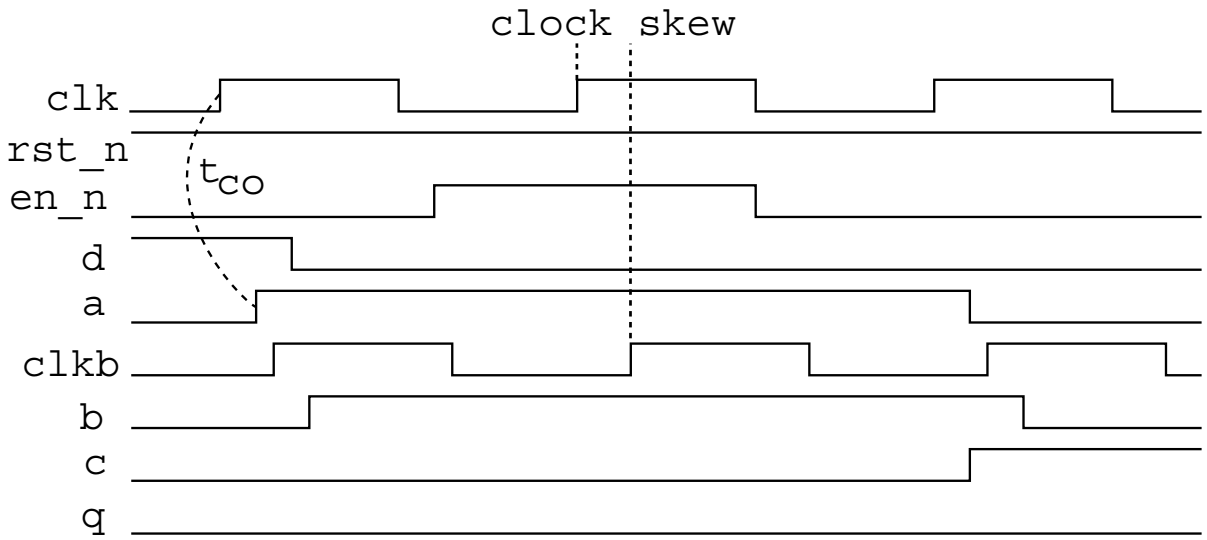
Timing aspects – setup/hold

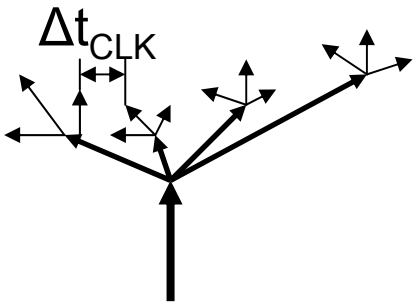
- The data to the flip-flop inputs must be stable some time before (setup time t_{SU}) and after (hold time t_H) the active edge of the clock
  - If the conditions above are not fulfilled, we speak of ***timing violations***
- The flip-flop outputs toggle with some delay after the active edge of the clock
  - *clock to output*

Example DFF ASIC cell



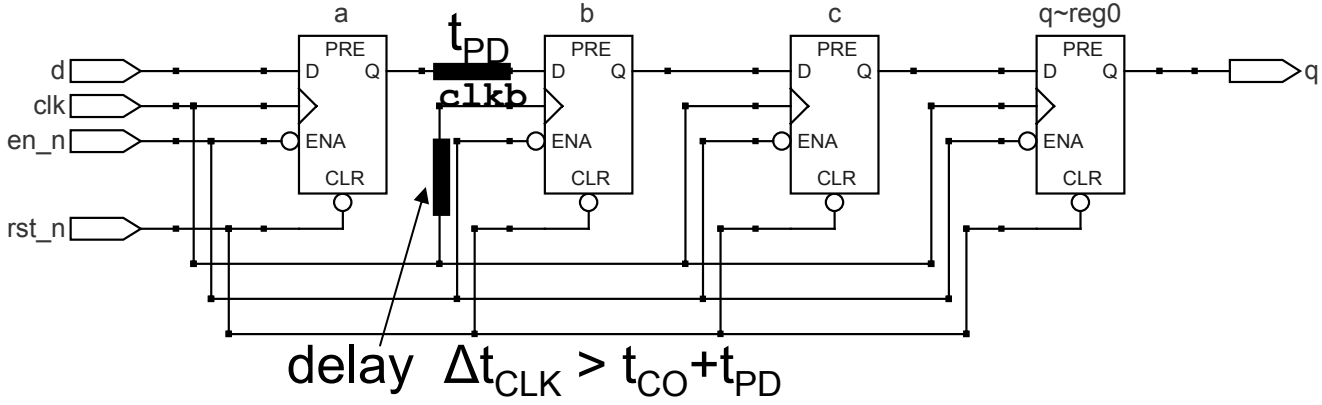
Example with a shift register





$f_{MAX}=1/(t_{CO}+t_{PD}+t_S)_{MAX}$

$\Delta t_{CLK} + t_H < t_{CO}+t_{PD}$



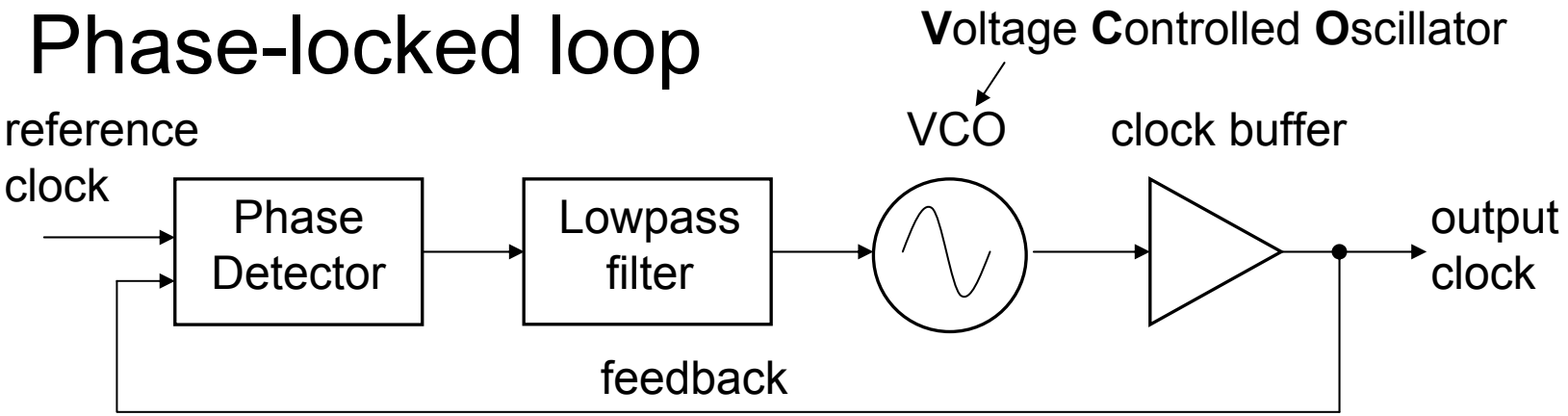
- The first condition gives the max frequency
- If the second is not fulfilled, the circuit will not work correctly even at 1 Hz!!!

Clock distribution – FPGA case

- The FPGAs have special networks for global and high fanout signals like clock, OE, reset
  - using the normal routing for such signals "guarantees" problems!
- The FPGA tools normally insert the proper clock buffer, but it can be instantiated manually
  - Set the clock constraints, even if the frequency is low (e.g. 1MHz or 100Hz for some slow device), otherwise the hold times might be violated!
  - Use PLLs and DCMs to improve the jitter and to multiply the clock frequency or to adjust the phase if necessary

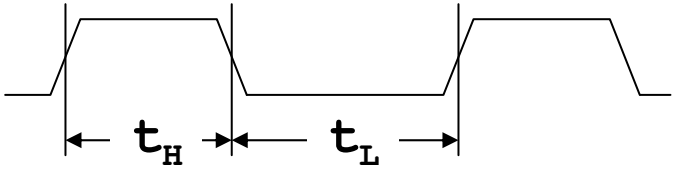
PLL

- Phase-locked loop



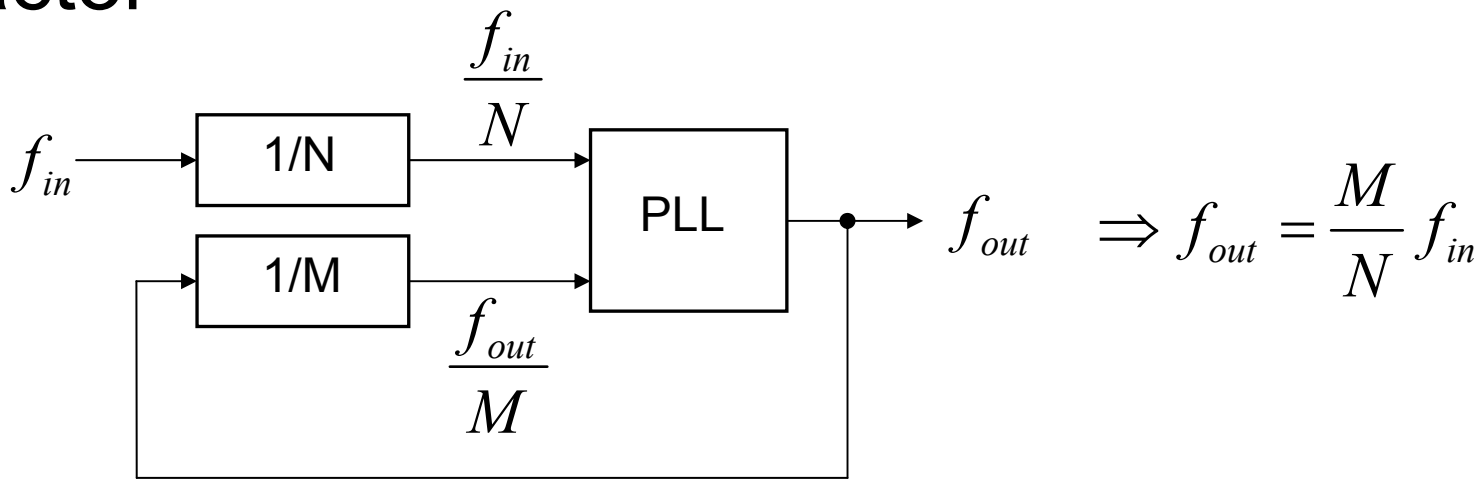
  - Zero delay between input and output
  - Less jitter at the output
  - Restored clock symmetry

variation of the clock period, deviation from the ideal clock



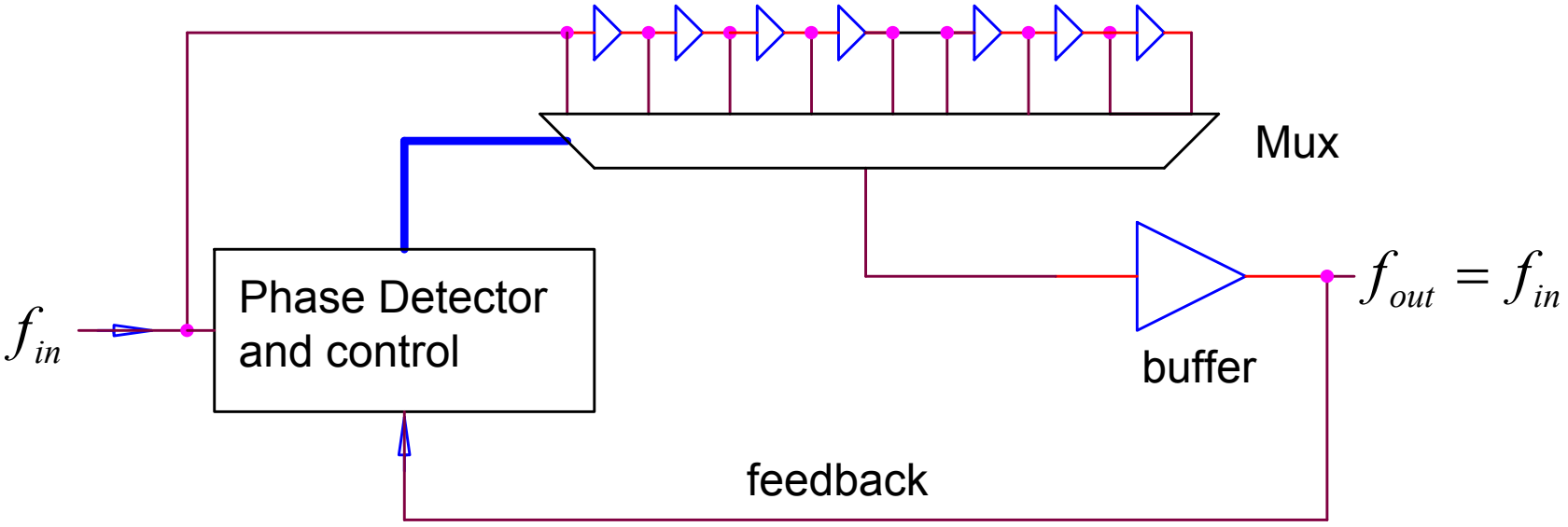
PLL – clock multiplication

- With proper clock dividers at the reference clock and in the feedback, the output clock frequency can be modified by a rational factor



DLL

- Delay-locked loop

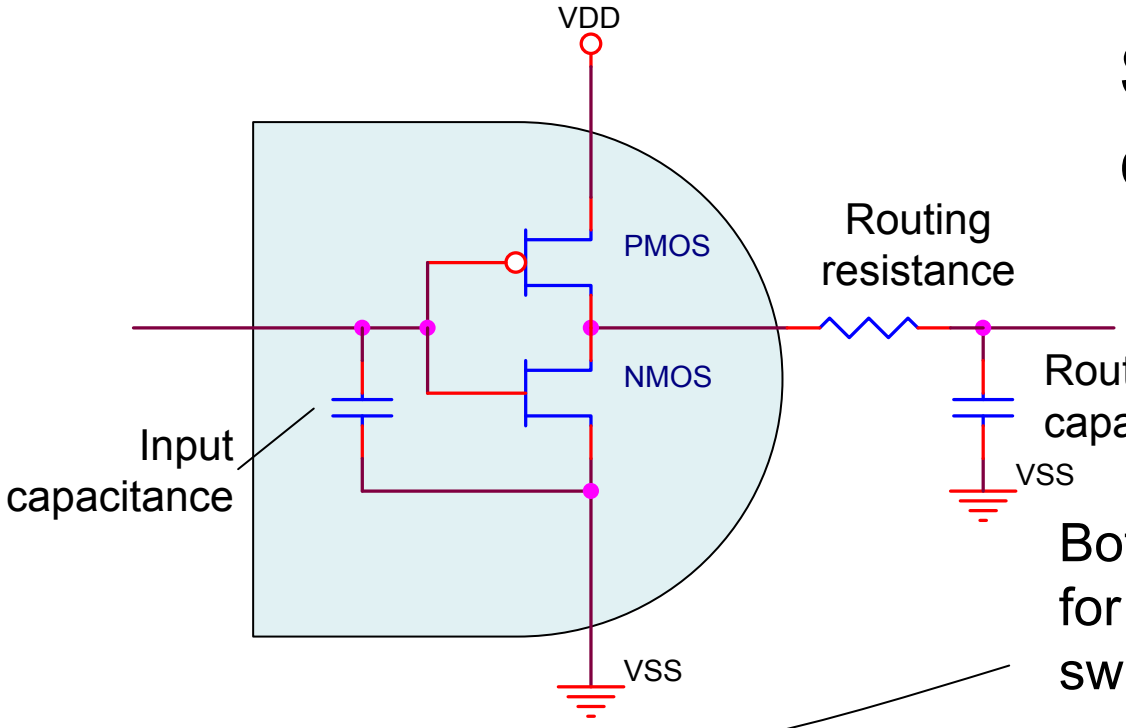


- Compensate the delay in the clock distribution

DLL properties

- DLLs are typically realized digitally, the possible time shift is discrete
- A DLL can not suppress the jitter!
- The DLLs typically have multiple outputs shifted at 0, 90, 180, 270 degree or even double clock output
  - by XORing the 0 and 90 output a double frequency can be produced but at the expense of worse jitter
- Generally the PLLs are better!

Power consumption in CMOS



Static and **dynamic** components

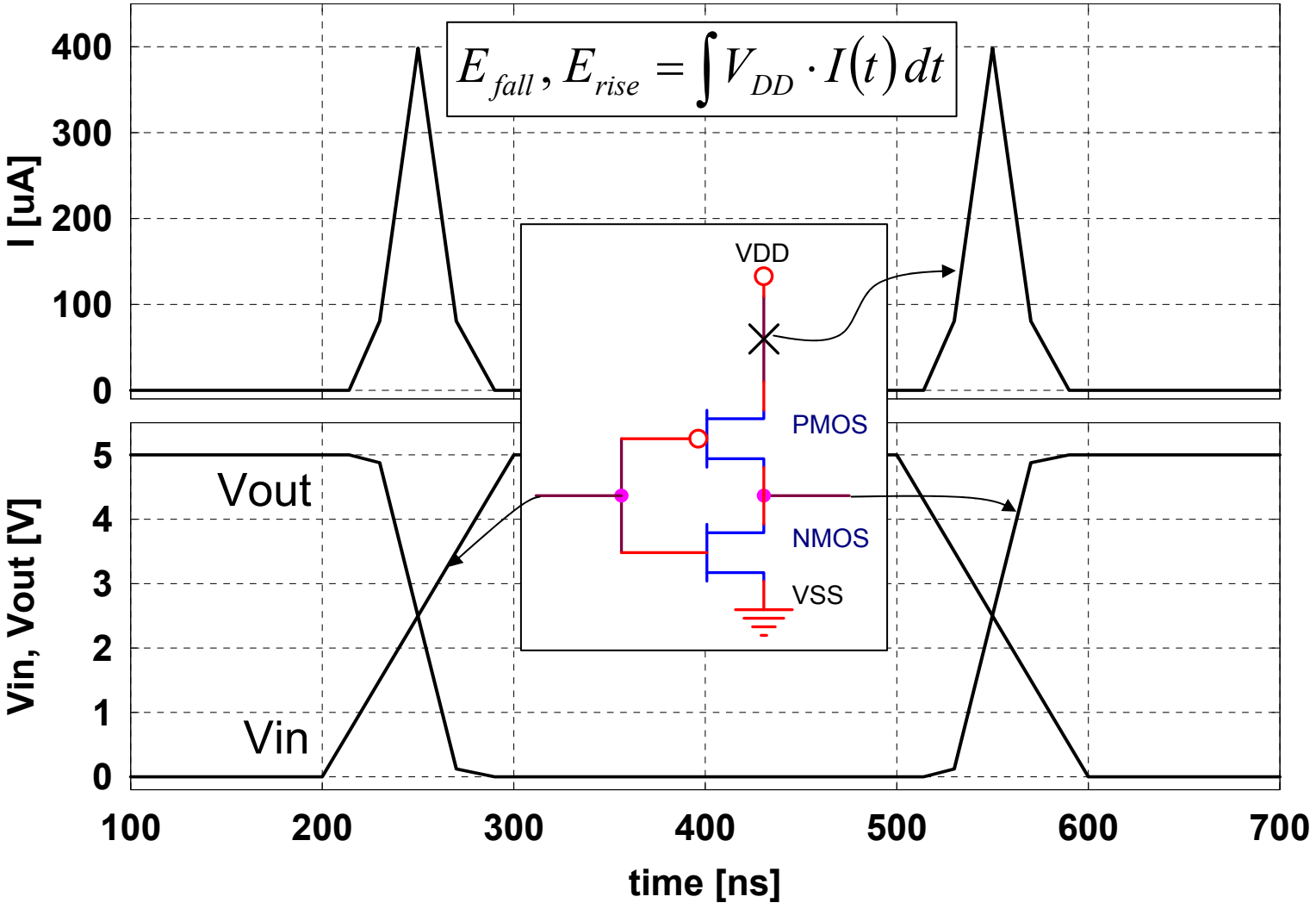
Both transistors conduct for short time while switching (next slide)

$$P(f) = \left(E_{rise} + E_{fall} + C_{fanout} \cdot V_{DD}^2\right) \cdot f_{switching} + P_{static}$$

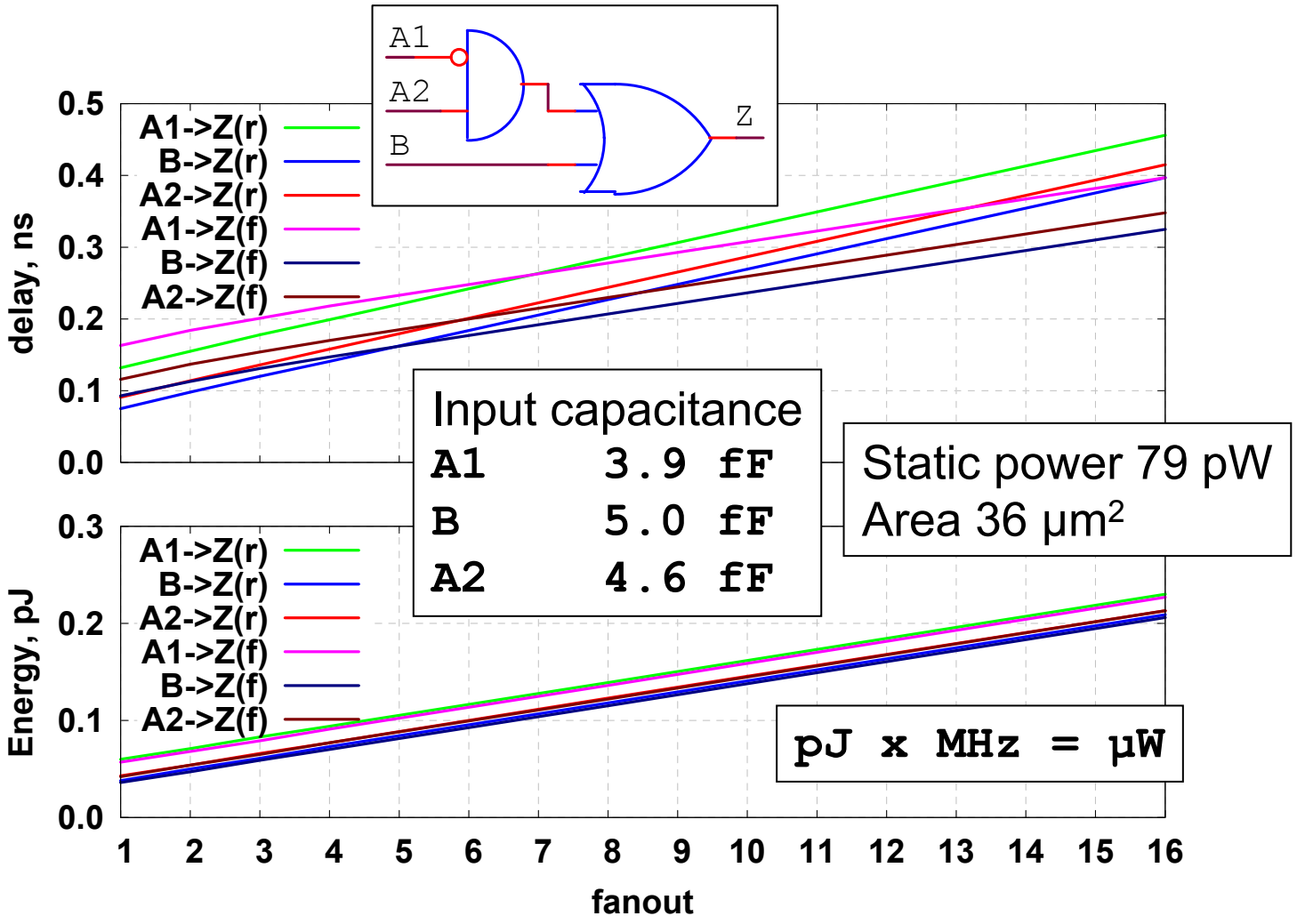
dynamic

Leakage currents, temperature dependent

Power consumption in a CMOS inverter

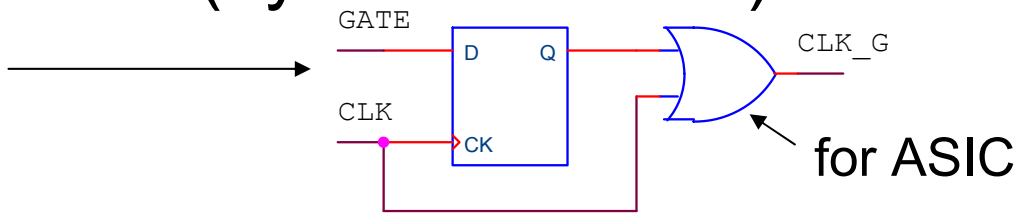


Example – combinational ASIC cell

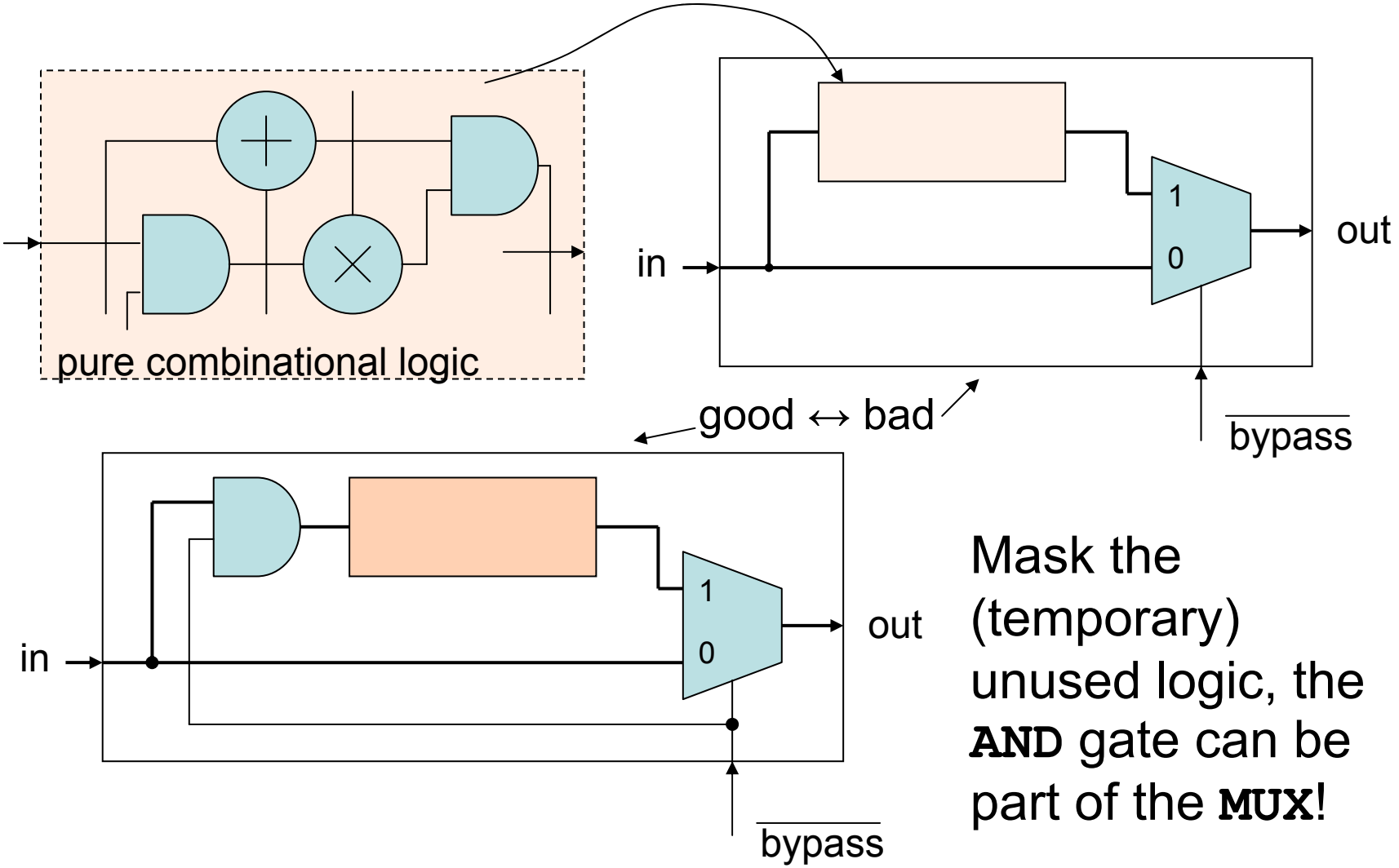


Power minimization strategies

- Technology
  - Smaller voltages and capacitances but leakage currents! 0.5 µm → 0.35 µm → 0.25 µm → 0.18 µm → 0.13 µm → 90 nm → 65 nm → 40 nm → 28nm...
- Clock frequency
  - As low as possible (dynamic control)
  - Clock gating
- Toggle rate
  - Smart masking of unused bits dynamically
  - Pipelining (prevent propagation of glitches)

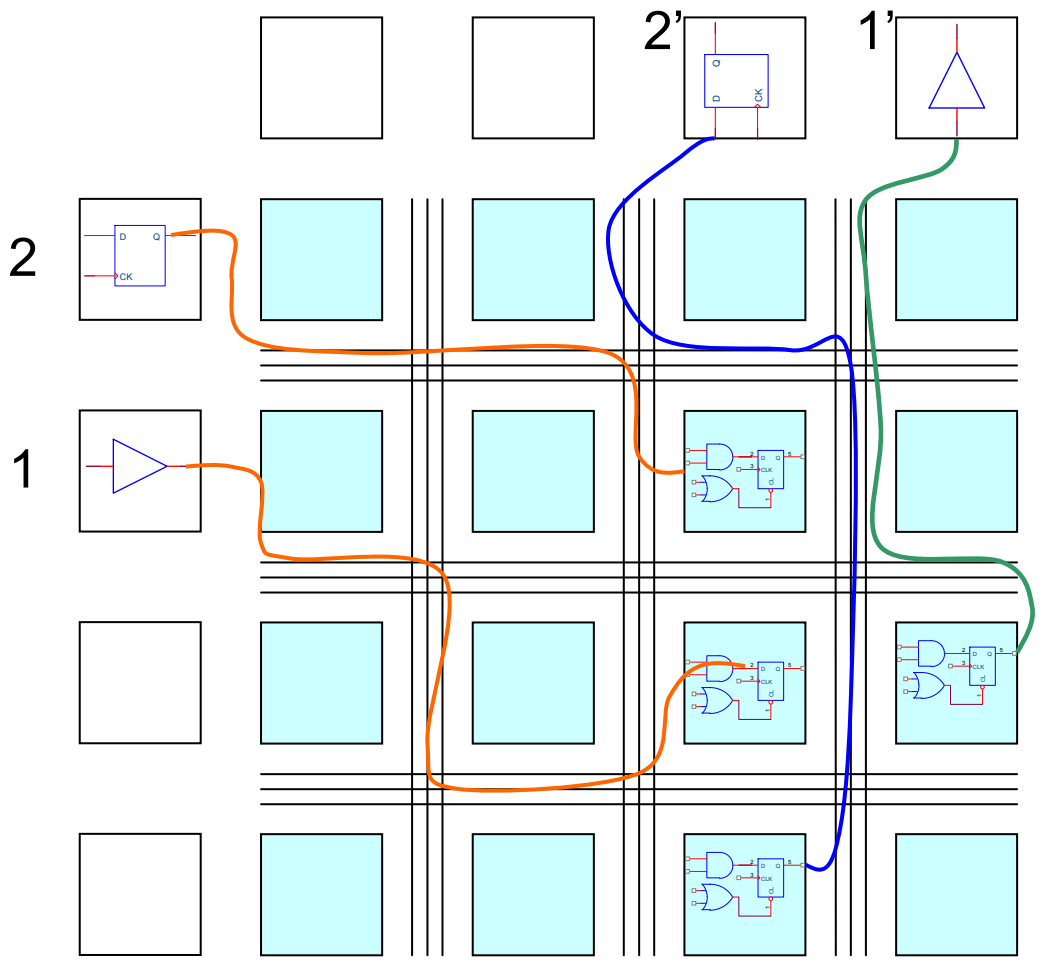


Gating of combinational logic



Mask the (temporary) unused logic, the **AND** gate can be part of the **MUX**!

I/O timing



If the input signal needs to be processed synchronously, then it must be first sampled by a DFF

Two possible solutions:

1) use a DFF inside the chip core

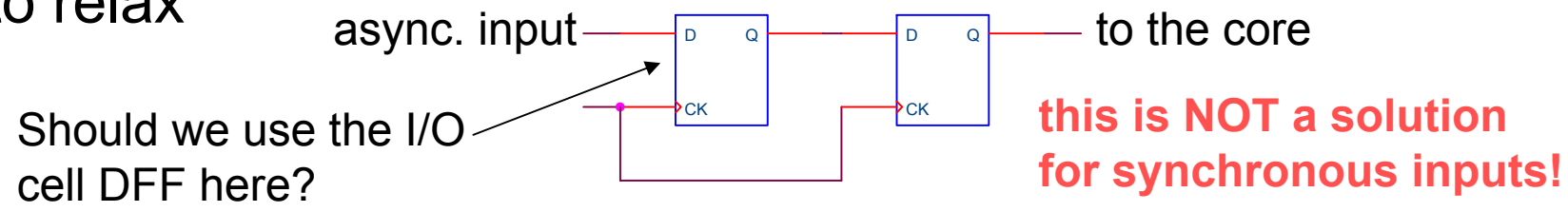
2) use a DFF inside the input cell

In the first case the setup time will be increased by the routing delay in the chip, will vary from pin to pin and will change with each next place & route.

Use the DFFs in the I/O cells !

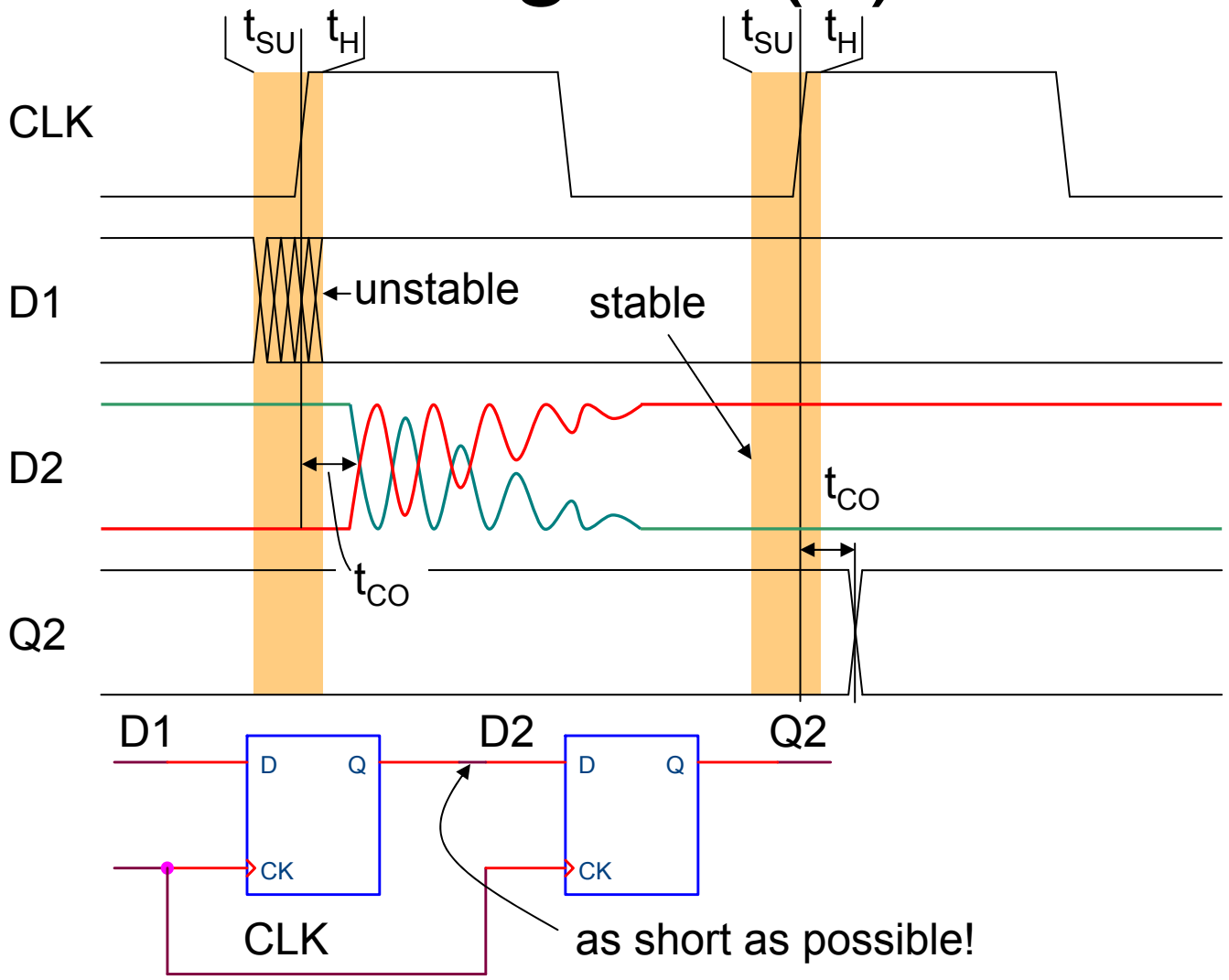
Synchronizing asynchronous signals(1)

- Setup violations are inevitable with **asynchronous** data sources
- The DFF may oscillate if the input changes within the setup window
- The best solution is to use two DFFs coupled directly, without any gates in between, so the first has more time to relax

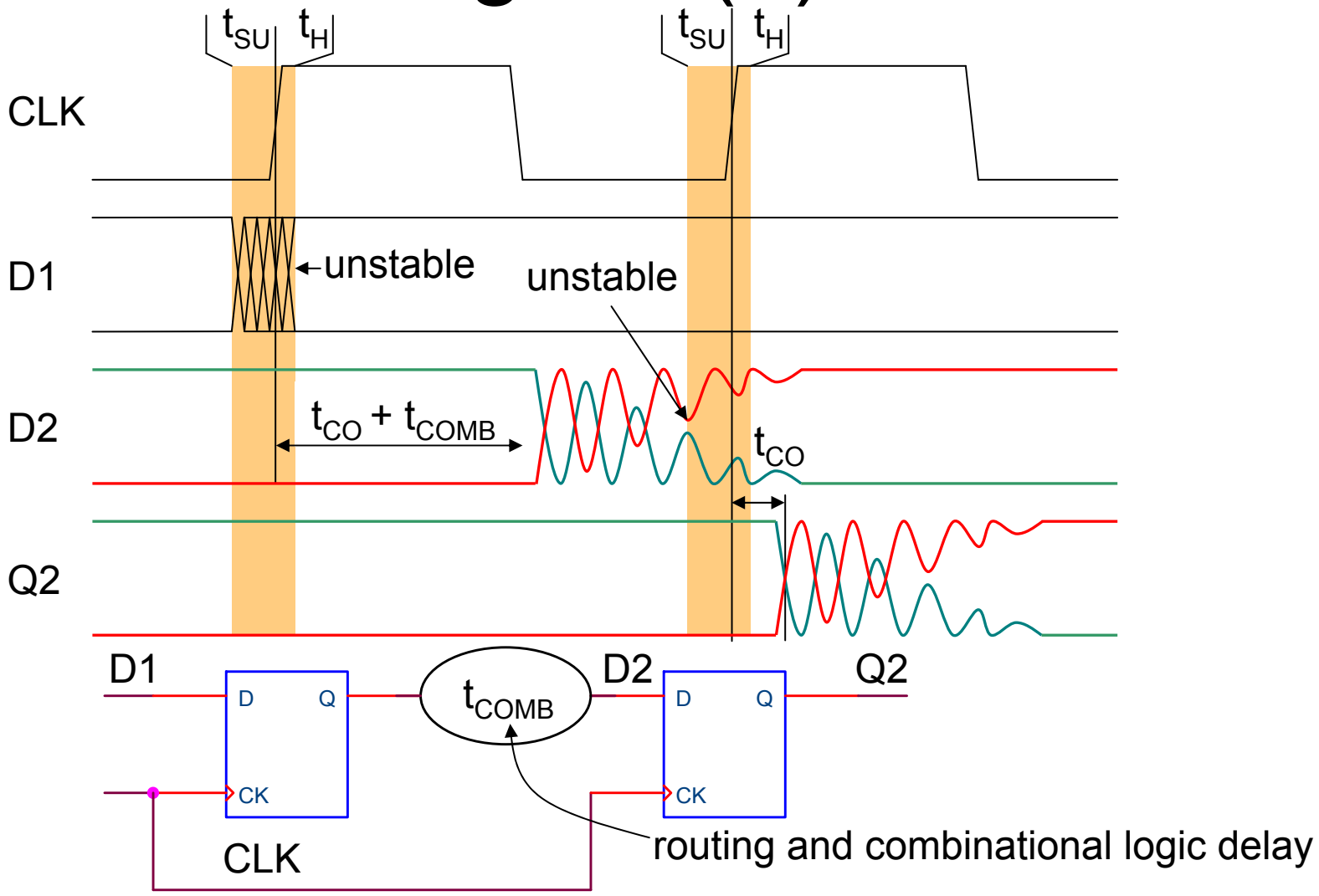


- Use slower clocks (if possible) and faster DFFs with min setup time to minimize the probability for oscillations

Synchronizing asynchronous signals(2)

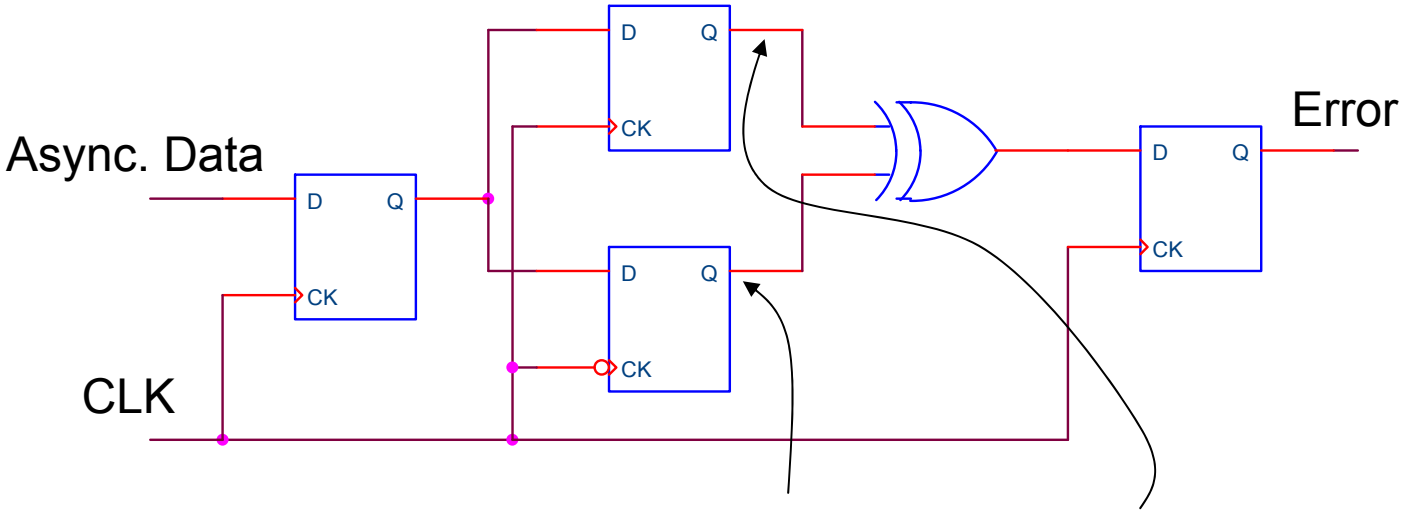


Synchronizing asynchronous signals(3)



Synchronizing asynchronous signals(4)

How to test the synchronizer ?



Error = different values stored in the middle and at the end of the period

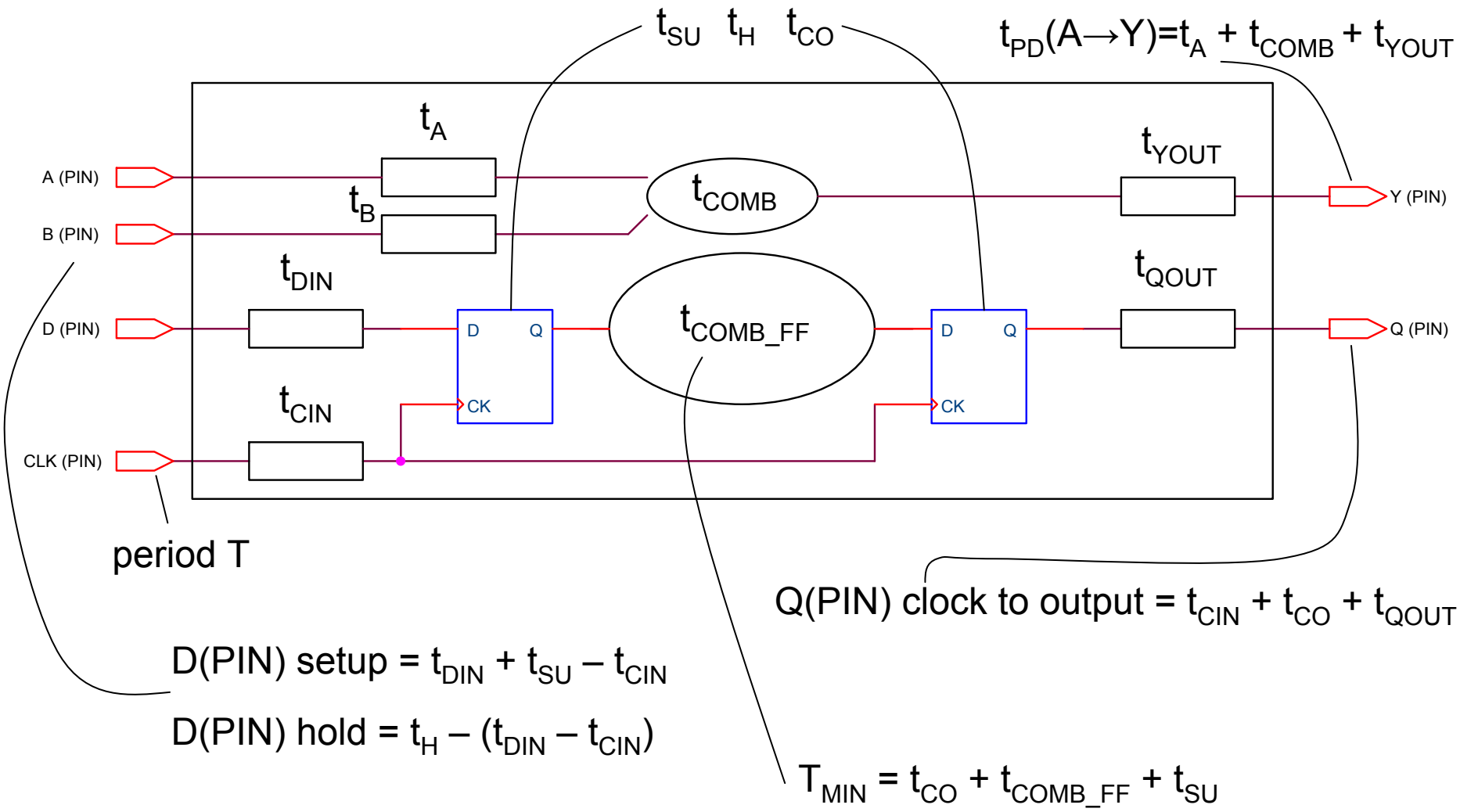
Input pins

- In some FPGAs programmable input delays are available to move the sampling window of the data relative to the clock
  - Set constraints on the input **setup/hold time**!
- The input cells can be programmed for various levels (5V, 3.3V, 2.5V, 1.8V, LVTTL, LVCMOS)
  - read carefully the data sheets and the board documentation!
- Programmable pull-up/down resistors are usually available – use them for inputs which are not always connected to prevent floating
  - floating inputs increase the power consumption of the chip and introduce noise

Output pins

- Similarly when some output signal is synchronous to the clock, use the output DFFs in the I/O cells to have well defined clock to output delay!
- In some FPGAs programmable output delays are available
  - set constraints for **clock to output**!
- The slew rate of the output cells is usually programmable
  - try to use the slow slew rate option on most of the outputs to prevent large current spikes on the V_{IO}
- The output pins are limited to source/sink currents, there are maximum recommended values per pin, per I/O bank (a group of pins)
  - read carefully the datasheets!

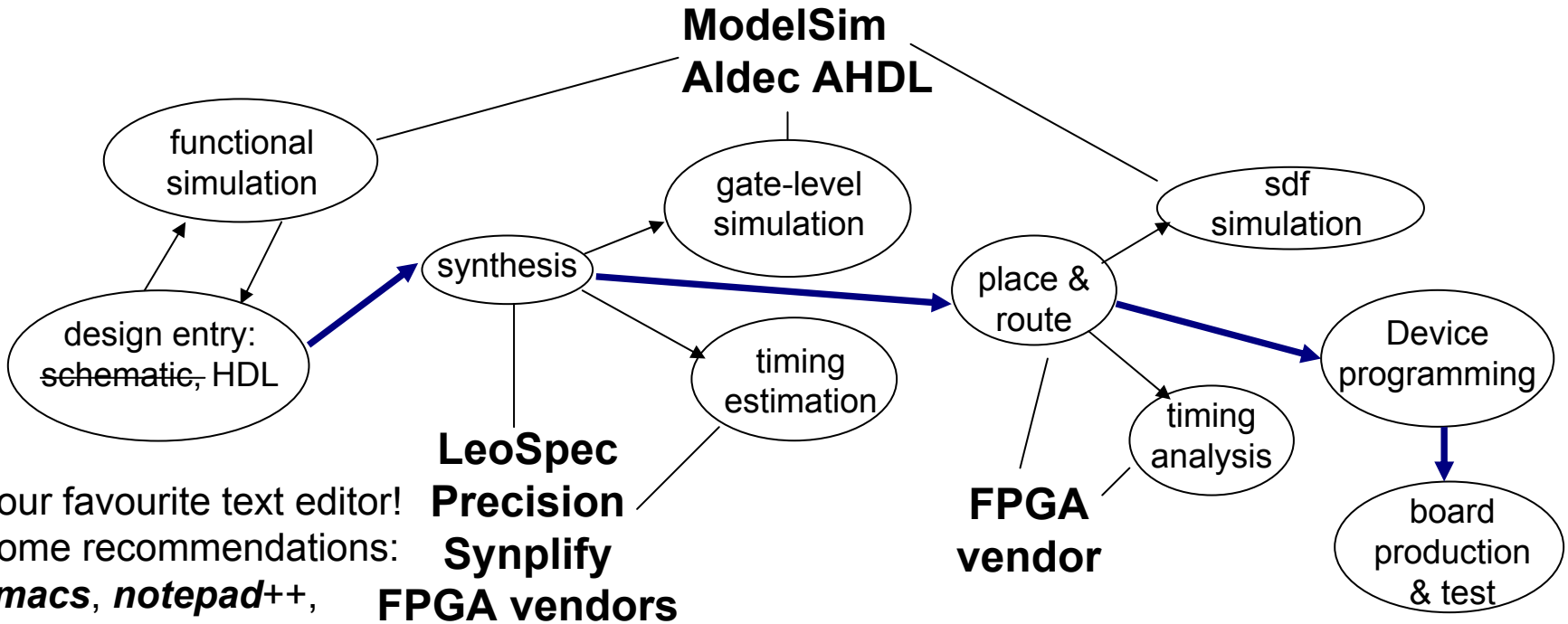
Timing parameters (summary)



Gate level and timing simulations

- The simulations in the FPGA design flow
- Why do we need different types of simulations?
- ModelSim simulations of Xilinx and Altera FPGAs
  - gate level, timing (with backannotation)
  - directory structure, Makefile

Design flow CPLD/FPGA



Your favourite text editor!
Some recommendations:
emacs, ***notepad***++,
nedit, with syntax colouring and more for
`VHDL` and **`Verilog`**

Each step can take seconds, minutes, hours ... (place & route)

Gate level and timing simulations

- The functional simulation is the fastest, but there are cases where it is not good/pessimistic enough, not all errors can be found
  - this is the fastest simulation, needs less computer resources and all source code signals are present with unchanged names
- The netlist after synthesis can be used for simulation
  - this simulation is slower than the functional but faster than the timing, some signals can not be found anymore
- The cell and routing delays are known after p & r – this can be used for more realistic simulation (best/typ/worst)
  - this is the slowest simulation, needs most computer resources, many signals can not be found (especially when the hierarchy is not preserved), new signals are inserted

Gate-level simulation (X)

In case of Xilinx FPGA designs you need to compile (only once) the library **`unisim`** located at:

<Xilinx_install>**`\vhdl\src\unisims\*.vhd`**

Then you need to compile the synthesis output netlist of your top design, for Xilinx the location is:

<Project>**`\netgen\synthesis\<top>.vhd`**

Note that this file will be not automatically generated after synthesis, you need to specify

"Generate Post-Synthesis Simulation Model"

In the *Properties* you can set some options, like VHDL or Verilog output etc.

Timing Simulation (X)

Typically done after placing and routing of the design.

In case of Xilinx FPGA designs you need to compile (only once) the library **`simprim`** located at:

<Xilinx_install>**`\vhdl\src\simprims\*_mti.vhd`**

Then you need to compile the output netlist, for Xilinx the location is:

<Project>**`\netgen\par\<top>.vhd`**

The simulation tool needs the timing information, stored in the **S**tandard **D**elay **F**ile (SDF):

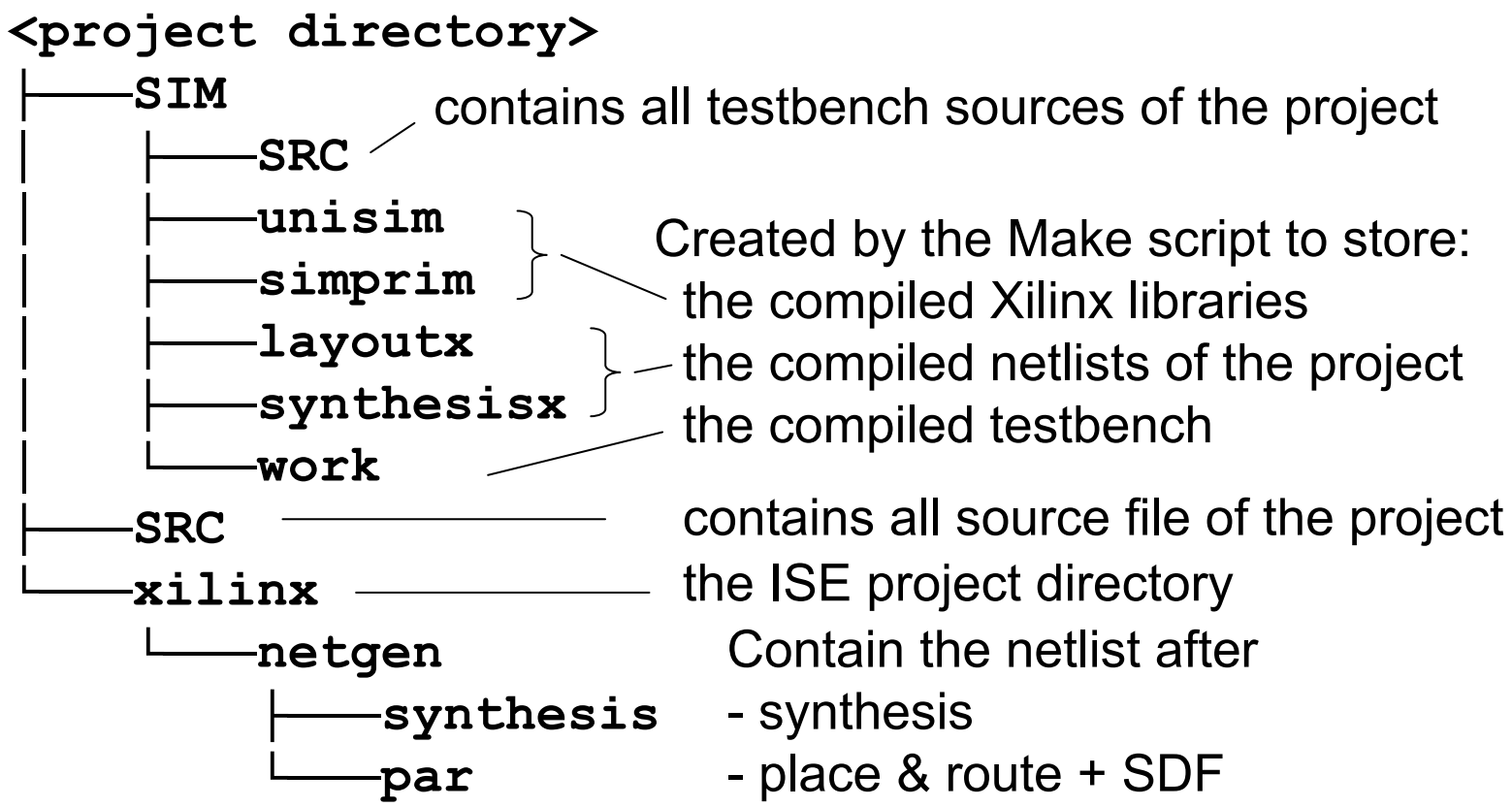
<Project>**`\netgen\par\<top>.sdf`**

Note that these files will be not automatically generated after P & R, you need to specify

"Generate Post-Place & Route Simulation Model"

In the *Properties* you can set some options, like VHDL or Verilog output etc.

Gate-level and timing simulation directory structure (X)



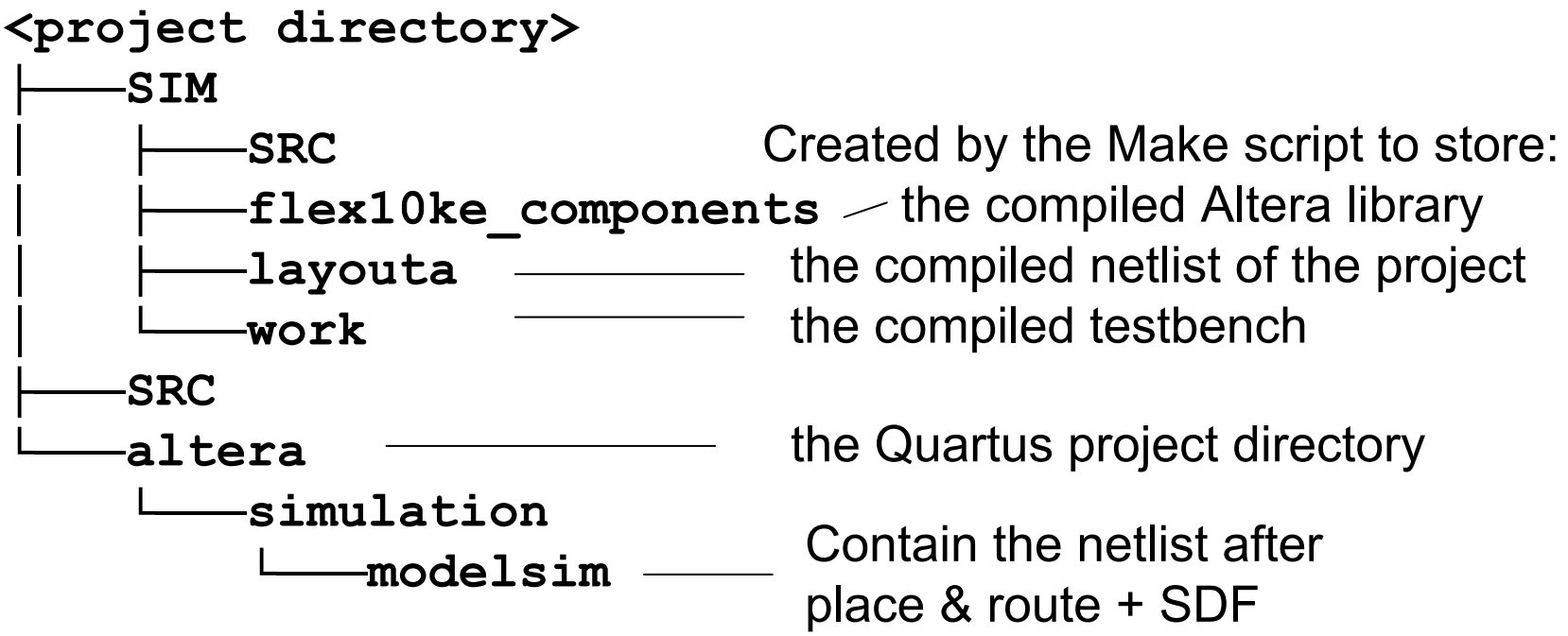
Gate-level and timing simulation Makefile (X 1)


```
xil_sim=c:\Xilinx\9.2\vhdl\src # The location of the Xilinx simulation models
design=cnt3 # The top level design name
testbench=./SRC/clk_gen.vhd ./SRC/$(design)_tb.vhd # The testbench file(s)
net_synx=../xilinx/netgen/synthesis/$(design)_synthesis.vhd # netlist after synthesis
net_parx=../xilinx/netgen/par/$(design)_timesim.vhd # netlist after place & route
sdf_xil=../xilinx/netgen/par/$(design)_timesim.sdf # standard delay file (SDF)
unisim: # The Xilinx unisim library
          vlib $@
          vcom -quiet -93 -work $@ "$(xil_sim)\unisims\unisim_VPKG.vhd"
          vcom -quiet -93 -work $@ "$(xil_sim)\unisims\unisim_VCOMP.vhd"
          vcom -quiet -93 -work $@ "$(xil_sim)\unisims\unisim_SMODEL.vhd"
          vcom -quiet -93 -work $@ "$(xil_sim)\unisims\unisim_VITAL.vhd"
simprim: # The Xilinx simprim library
          vlib $@
          vcom -quiet -93 -work $@ "$(xil_sim)\simprims\simprim_Vpackage_mti.vhd"
          vcom -quiet -93 -work $@ "$(xil_sim)\simprims\simprim_Vcomponents_mti.vhd"
          vcom -quiet -93 -work $@ "$(xil_sim)\simprims\simprim_SMODEL_mti.vhd"
          vcom -quiet -93 -work $@ "$(xil_sim)\simprims\simprim_VITAL_mti.vhd"
synthesisx: unisim $(net_synx) # Compile the synthesis netlist
          vlib $@
          vcom -quiet -93 -work $@ $(net_synx)
layoutx: simprim $(net_parx) # Compile the p & r netlist
          vlib $@
          vcom -quiet -93 -work $@ $(net_parx)
```

Gate-level and timing simulation Makefile (X 2)


```
simsyn: $(testbench) synthesisx # Simulate the synthesis netlist
          vmap libdut synthesisx
          rm -rf work; vlib work
          vcom -quiet -93 -work work $(testbench)
          vsim 'work.$(design)_tb' -t 1ns -do 'wave_synx.do'
simlay: $(testbench) layoutx # Simulate the p & r netlist without timing backannotation
          vmap libdut layoutx
          rm -rf work; vlib work
          vcom -quiet -93 -work work $(testbench)
          vsim 'work.$(design)_tb' -t 1ns -do 'wave_sdfx.do'
simsdf: $(testbench) layoutx # Simulate the p & r netlist with timing backannotation
          vmap libdut layoutx
          rm -rf work; vlib work
          vcom -quiet -93 -work work $(testbench)
          vsim 'work.$(design)_tb' -sdftyp /dut=$(sdf_xil) -t 1ps -do 'wave_sdfx.do'
simsdf_nc: $(testbench) layoutx # Simulate the p & r netlist with timing backannotation
          vmap libdut layoutx   # but no timing checks
          rm -rf work; vlib work
          vcom -quiet -93 -work work $(testbench)
          vsim 'work.$(design)_tb' +notimingchecks -sdftyp /dut=$(sdf_xil) -t 1ps \
          -do 'wave_sdfx.do'
clean: # Clean all library directories
          rm -rf work modelsim.ini transcript vsim.wlf simprim unisim synthesisx layoutx
.PHONY: simsyn simlay simsdf simsdf_nc clean
```

Timing simulation – directory structure (A)



Timing simulation Makefile (A)

```
alt_sim=d:\Programme\q81\quartus\eda\sim_lib # The location of the Altera sim. models
alt_family=flex10ke # Note that the family depends on the chip used in the project!
alt_lib=$(alt_family)_components
design=cnt3 # The top level design name
testbench=./SRC/clk_gen.vhd ./SRC/$(design)_tb.vhd # The testbench file(s)
net_para=../altera/simulation/modelsim/$(design).vho # netlist after place & route
sdf_alt=../altera/simulation/modelsim/$(design)_vhd.sdo # standard delay file (SDF)
$(alt_lib):
          vlib $@
          vcom -quiet -93 -work $@ "$(alt_sim)\$(alt_family)_atoms.vhd"
          vcom -quiet -93 -work $@ "$(alt_sim)\$(alt_family)_components.vhd"
          vmap $(alt_family) $@
layouta: $(alt_lib) $(net_para)
          vlib $@; vcom -quiet -93 -work $@ $(net_para)
simlay: $(testbench) layouta   # Simulate the p & r netlist without
          vmap libdut layouta  # timing backannotation
          rm -rf work; vlib work
          vcom -quiet -93 -work work $(testbench)
          vsim 'work.$(design)_tb' -t 1ns -do 'wave_sdfa.do'
simsdf: $(testbench) layouta  # Simulate the p & r netlist with
          vmap libdut layouta # timing backannotation
          rm -rf work; vlib work
          vcom -quiet -93 -work work $(testbench)
          vsim 'work.$(design)_tb' -sdftyp /dut=$(sdf_alt) -t 1ps -do 'wave_sdfa.do'
clean: # Clean all library directories
          rm -rf functional work modelsim.ini transcript vsim.wlf $(alt_lib) layouta
.PHONY: simlay simsdf clean
```

Recommendations (1)

- Do not use the text editors of the FPGA vendor software tools and of the simulation tools! These programs are not very stable, you can easily lose your work!
- Portability for code reuse:
  - Do not use the FPGA vendor software tools for simulations, and if possible avoid synthesis with them to have more independent design flow
  - Avoid using specific FPGA library components
- Do not work only with GUI
  - try to find out how the software stores all important parameters in the corresponding text files and edit them directly (e.g. constraint files)
  - If possible write scripts for all steps (make), many programs create them automatically – so you can be sure that all steps are done in the right order next time, doing this by mouse clicking is not very reliable

Recommendations (2)

- Avoid becoming a fanatic fan of some FPGA vendor – for each application take the best suitable chip, if your HDL code follows the recommendations above, it will be portable and easy to change the technology
- Maintain your HDL code in repositories (CVS, SVN). Some FPGA software tools offer version control – nobody knows how it works, in most cases not as expected, do not rely on it!
- Put your sources in a well organized directory tree, DO NOT MIX with the hundreds of files generated by the software!