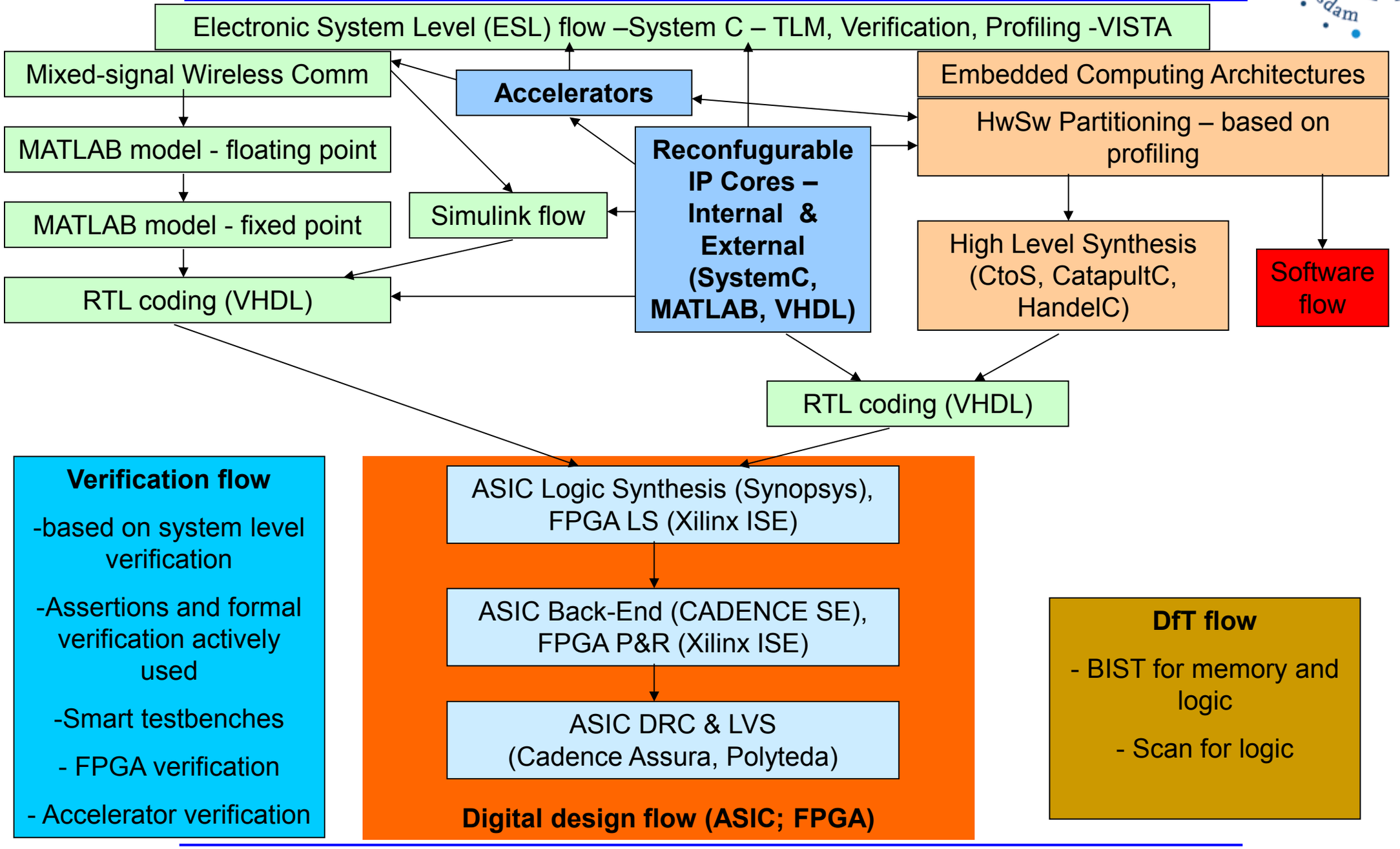
Logic Synthesis

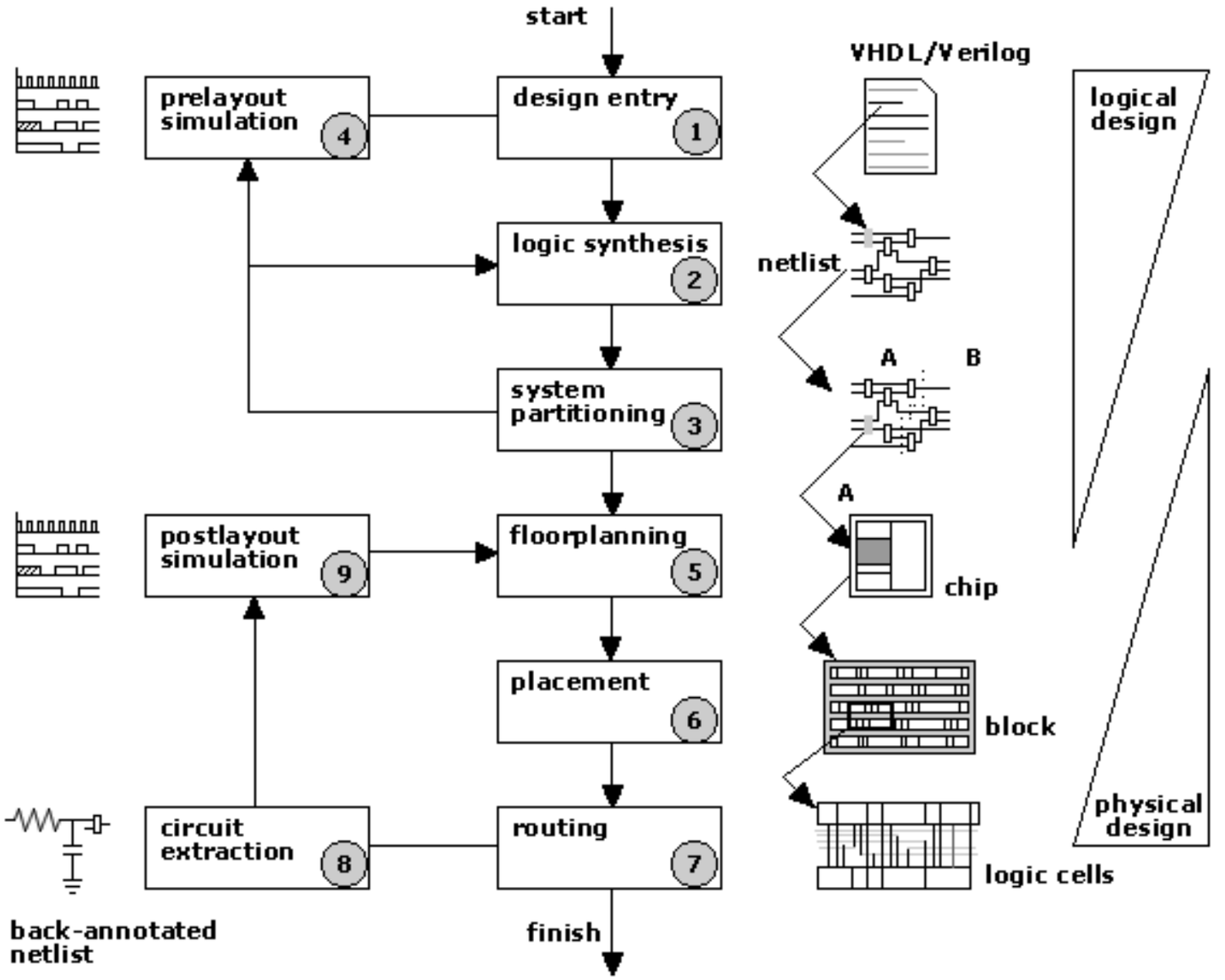
Overview

- Design flow
- Principles of logic synthesis
- Logic Synthesis with the common tools
- Conclusions

System Design Flow

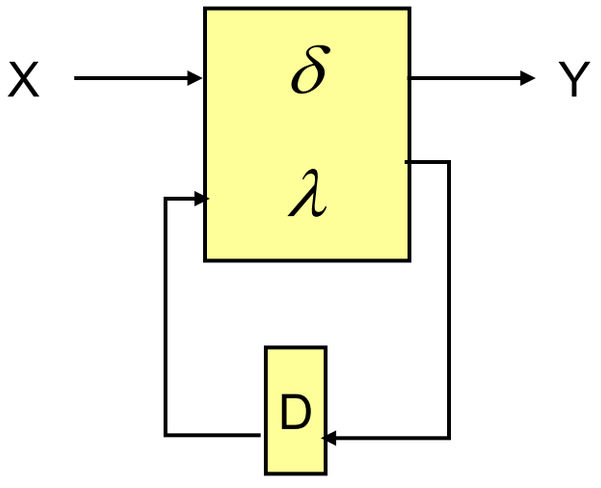


ASIC Design flow



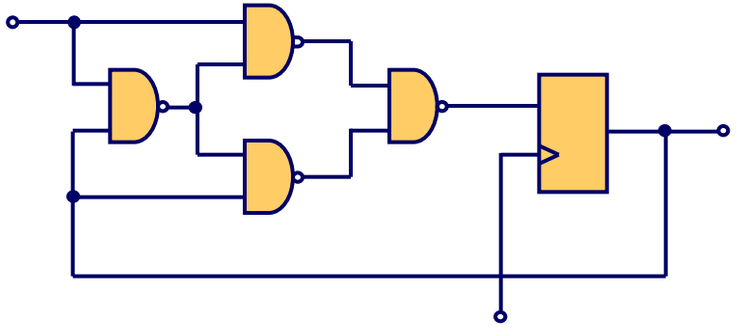
Source: http://www.ami.ac.uk

What is Logic Synthesis?



Given: Finite-State Machine F(X,Y,Z, λ, δ) where:

X: Input alphabet
Y: Output alphabet
Z: Set of internal states
λ : X x Z → Z (next state function)
δ : X x Z → Y (output function)



Target: Circuit C(G, W) where:

G: set of circuit components g ∈{Boolean gates, flip-flops, etc}

W: set of wires connecting G

Source: Logic Synthesis Outline, University Texas

How is Logic Synthesis Performed?

- Based on the input description (HDL code, FSM, boolean description) the goal is to generate the logic gate netlist that performs the defined function

- For this transformation and corresponding optimization different methods can be utilized:
  - MINTERM/MAXTERM optimization
  - Quine McCluskey
  - Espresso algorithm etc.

- The resulting circuit could be optimized for:

  Performance (setup, hold time delay)

  Power

  Area

Quine-McCluskey Algorithm

Truth table:

	x_3	x_2	x_1	x_0	y
0:	0	0	0	0	1
1:	0	0	0	1	1
2:	0	0	1	0	0
3:	0	0	1	1	1
4:	0	1	0	0	0
5:	0	1	0	1	0
6:	0	1	1	0	0
7:	0	1	1	1	1
8:	1	0	0	0	0
9:	1	0	0	1	1
10:	1	0	1	0	0
11:	1	0	1	1	1
12:	1	1	0	0	0
13:	1	1	0	1	0
14:	1	1	1	0	1
15:	1	1	1	1	0

Implicants (Order 0):

	x_3	x_2	x_1	x_0	
0:	0	0	0	0	→
1:	0	0	0	1	→
3:	0	0	1	1	→
7:	0	1	1	1	→
9:	1	0	0	1	→
11:	1	0	1	1	→
14:	1	1	1	0	✓

Implicants (Order 1):

	x_3	x_2	x_1	x_0	
0, 1:	0	0	0	-	✓
1, 3:	0	0	-	1	→
1, 9:	-	0	0	1	→
3, 7:	0	-	1	1	✓
3, 11:	-	0	1	1	→
9, 11:	1	0	-	1	→

Implicants (Order 2):

	x_3	x_2	x_1	x_0	
1, 3, 9, 11:	-	0	-	1	✓

Prime implicant chart:

	x_3	x_2	x_1	x_0	0	1	3	7	9	11	14	
1, 3, 9, 11:	-	0	-	1		○	○		●	●		$(\bar{x}_2 x_0)$
0, 1:	0	0	0	-	●	○						$(\bar{x}_3 \bar{x}_2 \bar{x}_1)$
3, 7:	0	-	1	1			○	●				$(\bar{x}_3 x_1 x_0)$
14:	1	1	1	0							●	$(x_3 x_2 x_1 \bar{x}_0)$

Extracted essential prime implicants: $(\bar{x}_3\bar{x}_2\bar{x}_1)$, $(\bar{x}_3 x_1 x_0)$, $(\bar{x}_2 x_0)$, $(x_3 x_2 x_1 \bar{x}_0)$

Minimal boolean expression:

$y = (\bar{x}_3\bar{x}_2\bar{x}_1) \vee (\bar{x}_3 x_1 x_0) \vee (\bar{x}_2 x_0) \vee (x_3 x_2 x_1 \bar{x}_0)$

Legend:
Don't-care: ×
Implicant (non prime): →
Prime implicant: ✓
Essential prime implicant: ●
Prime implicant but covers only don't-care: (×)

Quine-McCluskey Algorithm is more effective then Carnaugh maps for larger number of variables

However, the runtime grows exponentially with the number of variables!

It consists of the two steps:

1) Finding prime implicants
   Starting from Order 0-> Order 1 -> Order 2 etc
2) Making prime implicant chart
   Selecting the minimal number of essential prime implicants

   Example is generated at:
   http://www.mathematik.uni-marburg.de

   You can use it to generate some new examples for you

ESPRESSO Algorithm

- Further improvement of the logic minimization results

- Generates only a subset of prime implicants
- Heuristic solver
- Includes the steps of:

  Reduce, to avoid the cubes within the others

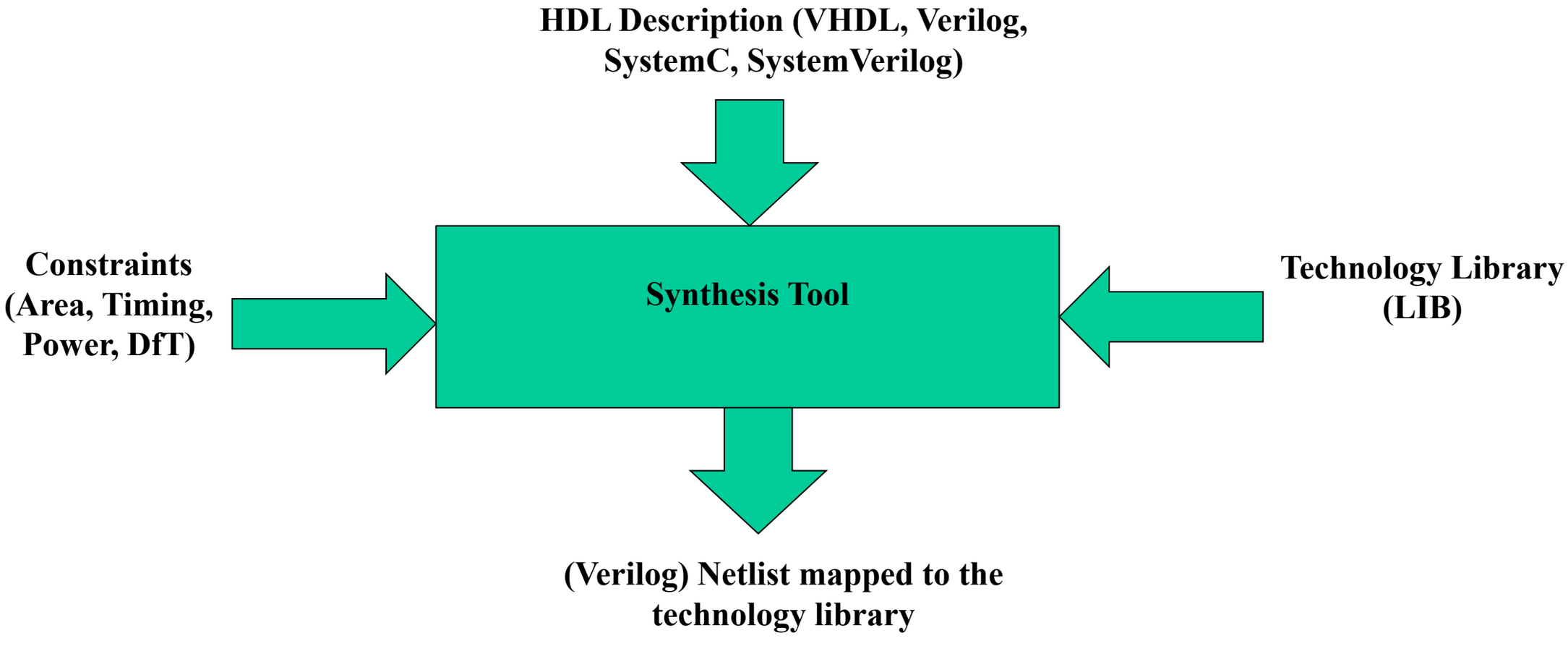
  Extraction of irredundant cover

  similar to QM prime implicant chart

  Expand, to achive maximum size

- Different implementation are possible, achieving different results
- Baseline of mainly all synthesis tools for ASIC/FPGA

Synthesis Process

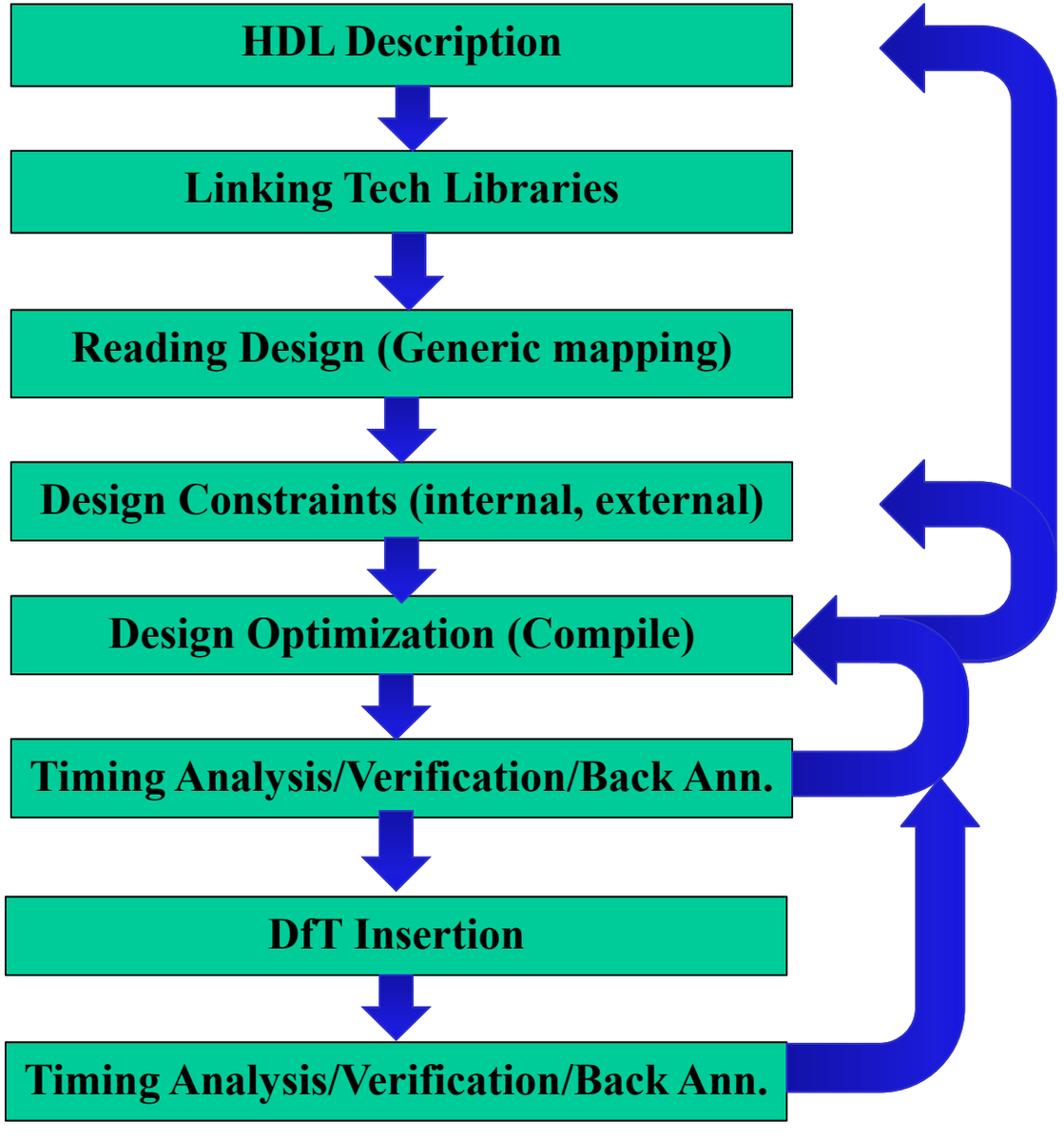


Phases of Synthesis Process

- Logic synthesis is a process by which an abstract form of desired circuit behavior (typically register transfer level (RTL)) is turned into a design implementation in terms of logic gates.
- The logic synthesis at here covered on the example of Synopsys Design Compiler.
- Based on the input (HDL description) and settings (constraints, tech library) the synthesis is performed
- Synthesis includes the phase of gate mapping
- After that comes the phase of optimization which is the process of netlist modifications until the constraints are met

  Optimization is invoked by the command Compile

Synthesis Flow



HDL Description

- **Input for synthesis process are HDL descriptions of hardware**
  VHDL, Verilog, SystemVerilog, SystemC

- **The quality of the VHDL descriptions affects significantly the synthesis output**
  The guidelines from this course should be followed for better results

- **HDL description is subject to change/optimization in case that the defined constraints cannot be met**
  Example: Processor defined in VHDL cannot meet the target clock frequency
  Solution: Modification of the Processor VHDL IP to introduce the additional pipeline stage and reduce the critical path

Linking Technology Libraries

- **In order to successfully synthesize design the related technology libraries need to be defined and linked**
- **Libraries are usually defined with LIB(erty) files**

  LIB file defines the available cells, their pins, internal timing, power consumption, function etc.
- **Libraries are defined as link or target libraries**

  Link libraries for special hard macros which are used in the code (Memory instances etc.)
- **Symbol library is defining the symbol of the cells if they are used in synthesis tool GUI**

Reading Design (Generic mapping)

- There are two important functions of this phase

  Loading of design in DC starts with analysis of all vhdl and verilog sources followed by elaboration at the top.

- Analysis - Checking of input HDL files

  Syntax check, semantic check

- Elaborate - Translation into generic netlist

  Generic netlist uses the technology independent gates (Boolean gates, functions, registers etc.)

  Report after reading provides the overview of the required resources (number of registers, type of registers)

  The warning messages generated by HDL reader can often give you some important tips about potential problems in your design.

Technology Mapping

- **Process of translating from generic netlist to technology dependent netlist**

  Generic netlist- generic RTL gates and functions: and, or, register, addition, multiplication

  Technology netlist- gates from the library, with specific drive strenght, performance, power, number of inputs

  Process of translation is not always 1-1

- **Two different methods**

  Rule based method

  Algorithmic method

    Representing each function with the subject graph

      for example using 2-input NAND and invertor

    All SC gates are also represented using the same basic gate representation in all possible ways of representation

    The algorithm is developed to look for the representation of the generic function with the lowest cost

    Cost can be defined as number of gates, area, performance etc.

Circuit Optimization

- Optimization of the synthesized netlist is possible

- Circuit restructuring
- Local

  For example different architectures of the adders (RCA, CLA etc)
- Global

  Reducing circuit complexity with Boolean simplification

  Partial collapsing

- Gate resizing

  Using stronger gates in general increases performance but also the area and power consumption

- Adding buffers

  Using buffers could contribute to increasing performance of the high-fanout lines

  Buffers are also used for slowing down the fast (feedthrough) paths

Implementation in the CAD tools - Design Optimization (Compile)

- The previous principles are implemented in the CAD tools
- This steps should based on the generic netlist optimize it and map it into netlist linked to the particular target technology.
  The optimization process tries to fulfil provided constraints
- There are three phases of optimization process:
  Architectural Optimization
  Logic-level Optimization
  Gate-level Optimization
- Architectural optimization – macroscopic abstraction level
  Analysis operations and dependencies
- Logic level optimization
  Subfunction factoring, adding variables to reduce logic complexity
  Flattening – two level logic minimization -> timing optimization
- Gate level optimization – microscopic abstraction level
  Technology mapping
  Timing, DRC, Area optimization

Design Constraints (internal, external)

- **Design constraints include the setting of internal and external parameters of the design**

- **External – operating conditions, wire load models, IO setup**

  Operating conditions: temperature, voltage, process

  Wire load model: represents estimation of interconnect delay

  IO Setup: timing/load of IO signals

- **Internal – design rules, optimization target**

  Design rules – defined by the technology (for example maximal drive strength, max capacitance etc.)

  Optimization target – clock frequency, power, area, special paths/timing exceptions (excluded, multi-cycle, max/min delay), IO delay setup

- **Finally the result of our specification is a synthesis script (tcl based)**

Further optimization techniques (I)

- **Arithmetic optimization can be performed to improve performance**

  Example: Synopsys DC offers Carry Save Adder tree use in the transformation of sum of products

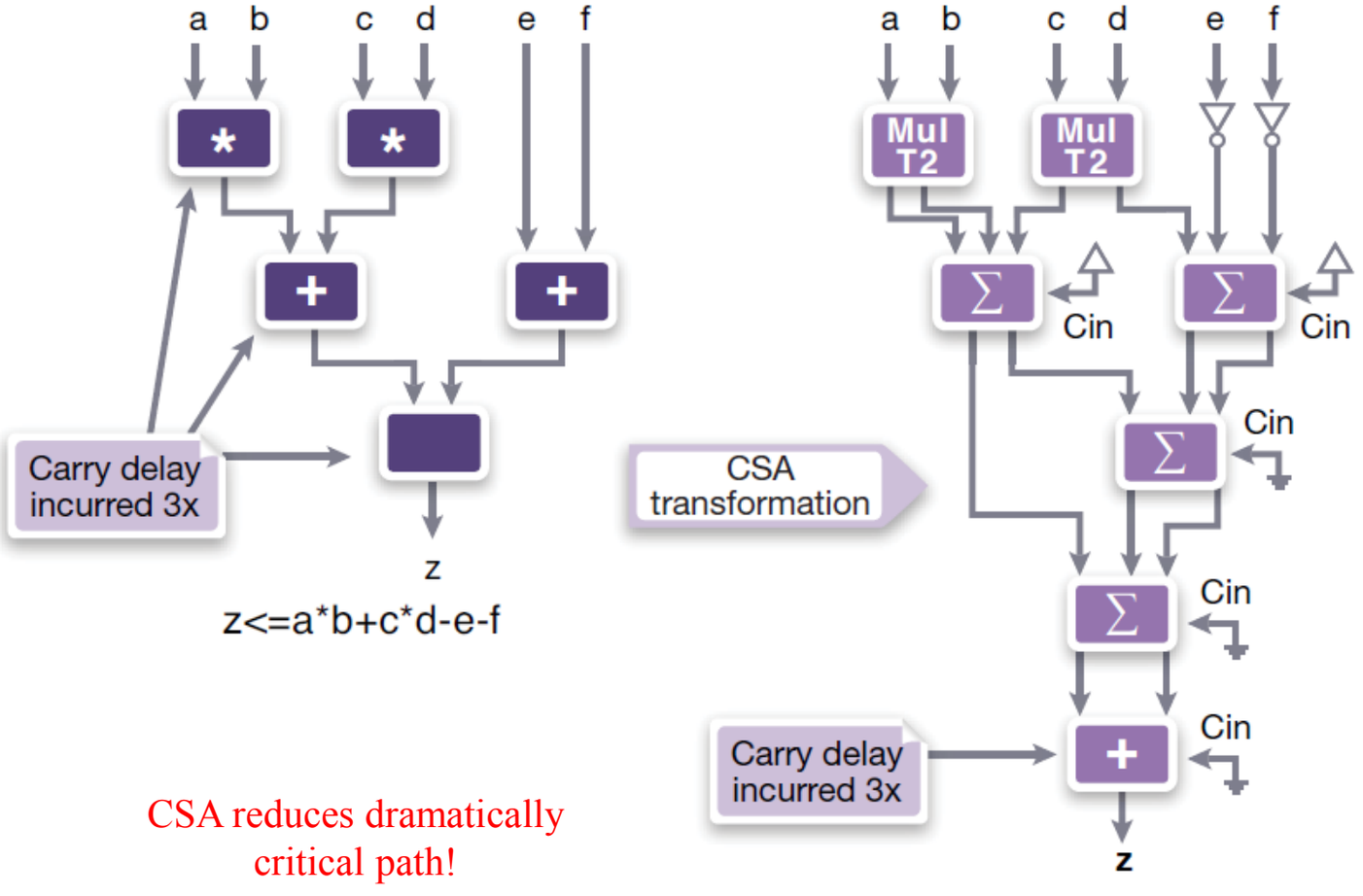
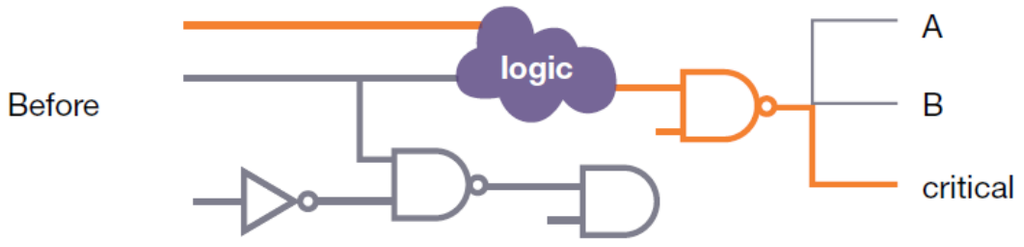


Figure source: Synopsys.com

Further optimization techniques (II)

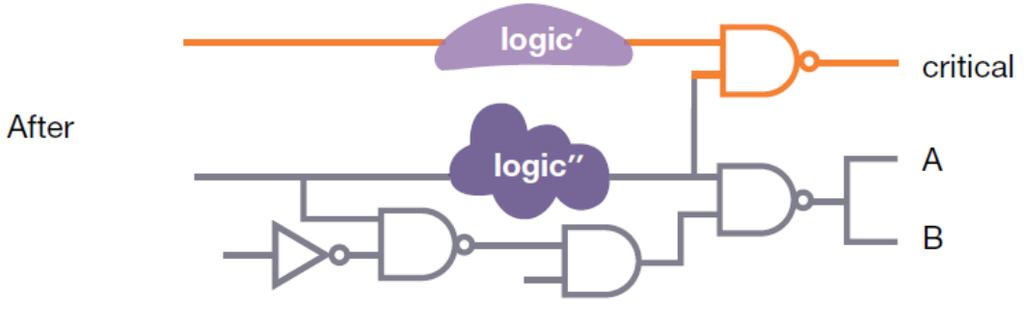
- **Logic duplication**

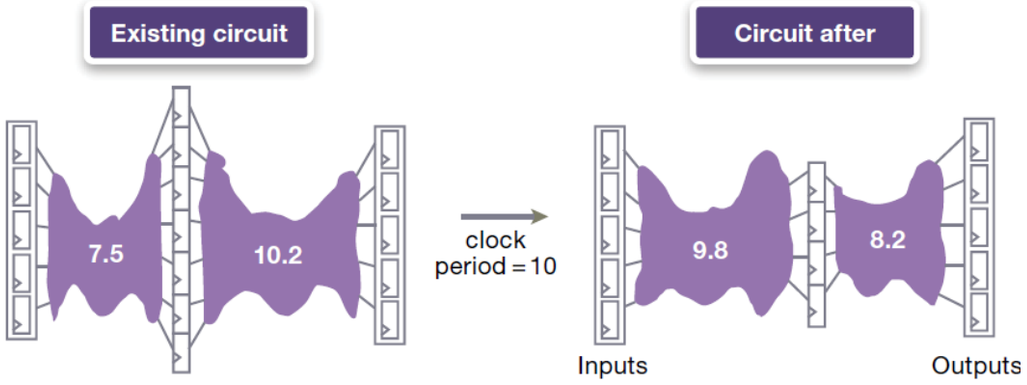
  Reducing the load of the critical path



- **Logic retiming**

  Enables logic balancing according to the system requirements





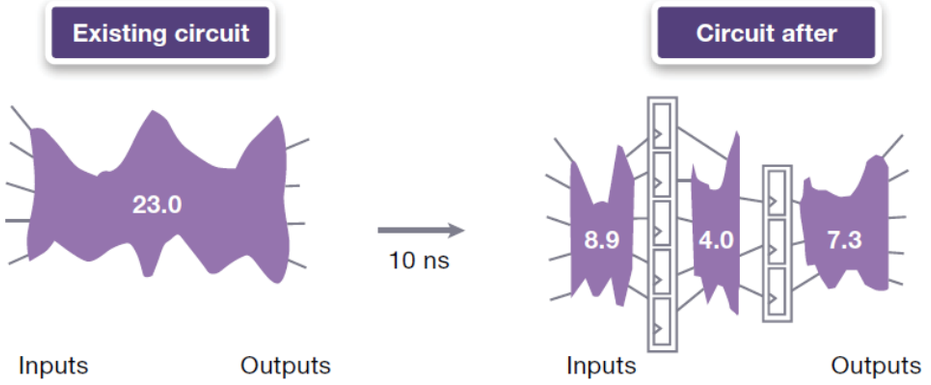


Figure source: Synopsys.com

Typical Commands

- specify the clock properties
  set CLK_NAME clk
  set CLK_PERIOD <period> # in ns
  set CLK_SKEW <skew> # in ns

- Timing exceptions
  set_max_delay
  set_min_delay
  set_false_path
  set_multicycle_path

- IO interface constraints
  set_input_delay
  set_output_delay

- Other constraints
  Set_max_area

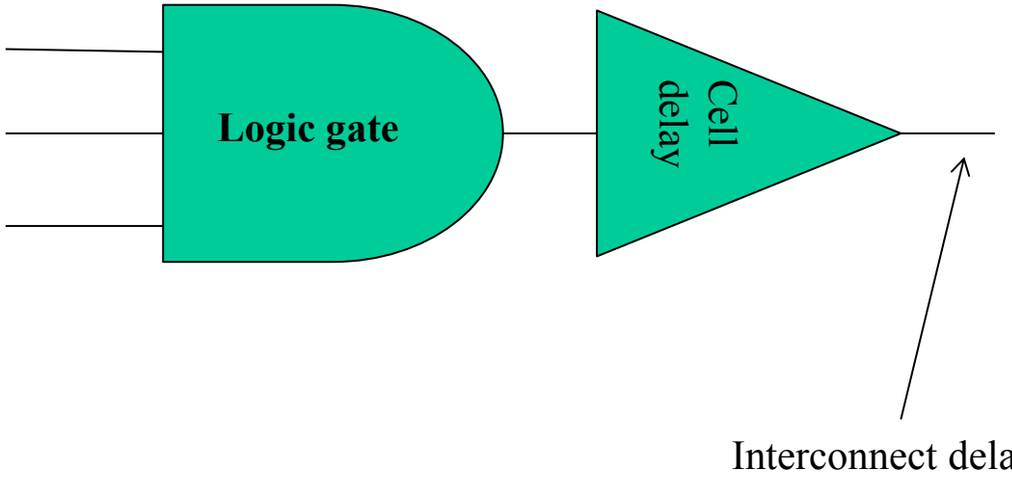
Need for Timing Analysis

- In synchronous designs the performance is defined by the critical path delay in the slowest pipeline stage

- Two elements of delay in the logic circuits

  Cell delay – "processing" delay of the gate

  Interconnect delay – time needed for signal to propagate through the wires



Methods of Timing Analysis

- **Static Timing analysis after Synthesis (Pre-Layout Analysis)**

  Only cell delay

- **Static Timing analysis after Place and Route (also called as Post-Layout Analysis)**

  Cell and interconnect delay

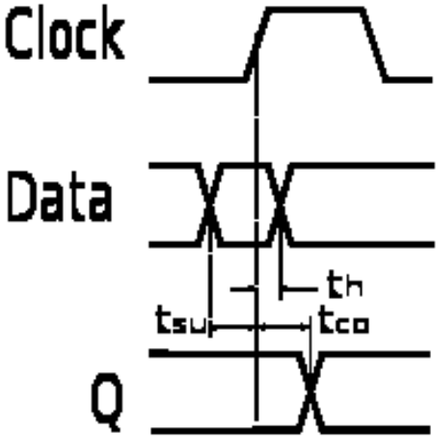
- **Statistical Static Timing Analysis (SSTA)**

  Taking technology variations into consideration

- **Two types of timing closure**

  A hold time, when a signal arrives too early, and propagates one clock cycle too early to the next pipeline stage

  A setup time, when a signal arrives too late, and misses the sampling time of the next pipeline stage



Performing Timing Analysis

- **Determining critical path**
- **Different corners – Process, Voltage, Temperature**

  Worst case corner (slow process, low V, high T) – slowest operation

  Typical case corner (typ process, nominal V, room T) – typical op.

  Best case corner (fast process, high V, low T) – fastest operation

- **Using different corner for calculating min (hold) and max (setup) delay**

  Min delay using Best case, Max delay using Worst case

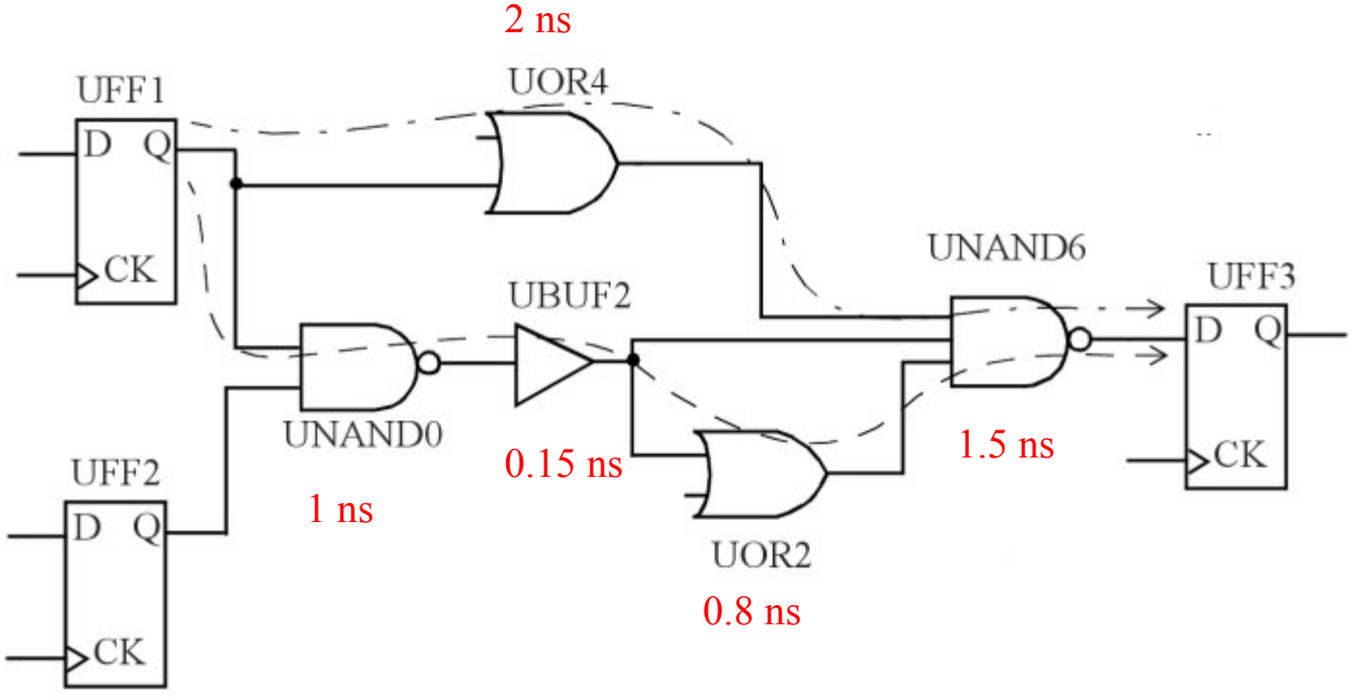


Image source: http://www.vlsi-expert.com

Timing Analysis

- **Synthesized design need to be verified whether the required constraints are achieved**

- **Timing analysis is performed by STA (static timing analysis) which evaluates the critical paths**

```
------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  0.000  |  0.319  |  5.216  |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |
|          All Paths:|  4930   |  3798   |   40    |  2488   |
+--------------------+---------+---------+---------+---------+
|worst               |  0.000  |  0.319  |  5.216  |  0.000  |
|                    |  0.000  |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |    0    |
|                    |  4930   |  3798   |   40    |  2488   |
+--------------------+---------+---------+---------+---------+
|typ                 |  0.000  |  2.774  |  6.565  |  0.000  |
|                    |  0.000  |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |    0    |
|                    |  4930   |  3798   |   40    |  2488   |
+--------------------+---------+---------+---------+---------+
|best                |  0.000  |  3.897  |  7.191  |  0.000  |
|                    |  0.000  |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |    0    |
|                    |  4930   |  3798   |   40    |  2488   |
+--------------------+---------+---------+---------+---------+
```

Verification/Back Annotation

- Further verification need to be performed for DRCs, area, power
- Formal verification could be performed to check the conformance of the generated netlist vs. initial HDL behavioral description
- Additional verification possibility is back-annotation

  In back-annotation the netlist is simulated in HDL simulator together with the annotated cell delay to check whether the functional requirements defined in the testbench are fulfilled

  Post-synthesis simulation require the Verilog netlist and the standard delay format file (SDF) for delay annotation

  Standard synthesis approach takes into account cell delay and for interconnects uses wireload model (rough estimation)

  Physical synthesis takes already into account the layout properties and estimated interconnect delay (good estimation)
- In case of not fulfilled constraints the synthesis flow requires iteration: constraint modification, HDL modification, optimization effort modification

Design for Testability Insertion

- Testability is highly important for industry use of the hardware design

- Different methods for testing of design:
  - Functional
  - Built-in self-test (BIST)
  - Structural – scan tests

- During the process of synthesis the standard flip-flops are replaced with the scan flip-flops and scan-chains are connected
  - This process adds additional delay in the critical path – re-optimization is needed

- Special tools can be used for test vector generation
  - Target 100% fault coverage

More details in the later lectures

Conclusions

- Principles of logic synthesis shown
- The automation of the synthesis provided by CAD tools
- Importance of constraining and timing analysis