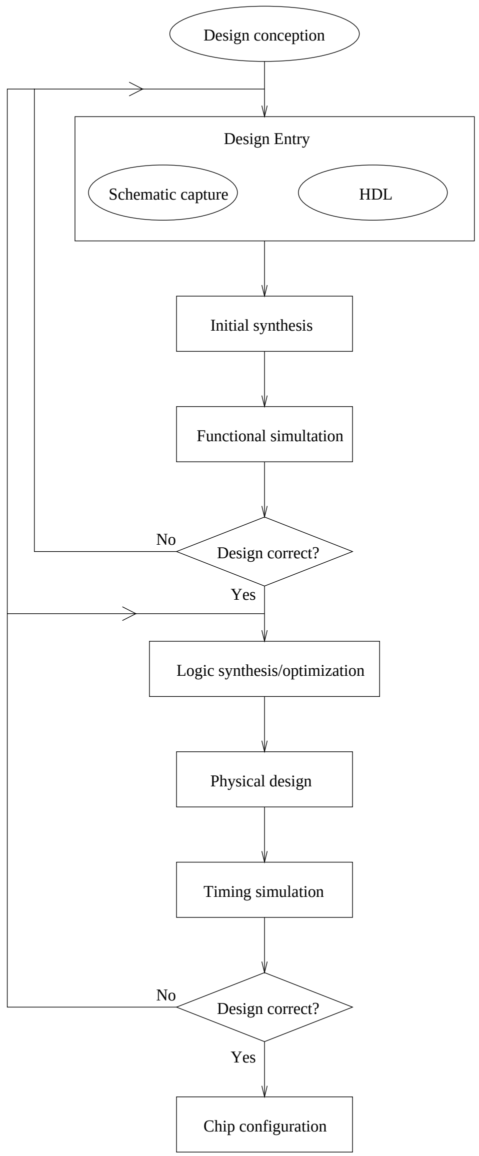
Design Flow and Design Tools

Chip Design Flow



CAD System

A typical CAD system comprises tools for performing the following tasks:

- *Design entry* allows the designer to enter a description of the desired circuit in the form of truth tables, schematic diagrams, or HDL code.
- *Initial synthesis* generates an initial circuit, based on data entered during the design entry stage.
- *Functional simulation* is used to verify the functionality of the circuit, based on inputs provided by the designer.
- *Logic synthesis and optimization* applies optimization techniques to derive an optimized circuit.
- *Physical design* determines how to layout the optimized circuit in a given target technology (full-custom, FPGA, etc.).
- *Timing simulation* determines the expected propagation delays of the implemented circuit.
- *Chip configuration* configures the actual chip to realize the designed circuit.

Design Entry

- Enter into the CAD system a description of the circuit being designed.
- Two approaches: schematic capture and hardware description languages.

1. **Schematic Capture**

- A *schematic capture* tool allows the user to draw a schematic diagram of a circuit in which circuit elements, such as logic gates, are depicted as graphical symbols and connections between circuit elements are drawn as lines.
- The tool provides a *library* which is a collection of graphical symbols that represents gates of various types. The gates in the library can be imported into the user's schematic.
- Previously created subcircuits can also be saved and reused.
- It allows *hierarchical design* to create a circuit that includes within it other smaller circuits.

2. **Hardware Description Languages (HDLs)**

- Resemble programming languages but are used to describe hardware.
- Unlike programming languages, HDLs represent extensive parallel operations.
- Originally intended for documentation and simulation purposes only, HDLs have become popular for use in design entry as input to *synthesis*.
- CAD tools synthesize the HDL code into a hardware implementation of the described circuit.
- Advantages:
  - *Portability:* A circuit specified in a HDL can be implemented in different types of chips and with CAD tools provided by different companies, without having to change the HDL specification.
  - *Hierarchical Design:* HDL code can be written in a modular way to facilitate hierarchical design.
  - *Reuse:* Sharing and reuse of HDL-described circuits is easy.

Initial Synthesis

- Translates HDL code or schematic diagram into a network of logic gates in suitable form for use by other programs.
- Output is a lower-level description of the circuit, e.g. a set of logic expressions that describe the logic functions needed to realize the circuit.
- The initial logic expressions produced will be manipulated to produce an equivalent but better circuit after performing functional simulation.

Simulation

Functional Simulation

- Before the circuit is optimized, functional simulation is performed to verify the functionality of the design.
- The user specifies valuations of the circuit's inputs and examines the output of simulation to verify that the circuit operates as expected.
- Functional simulator assumes that the time needed for signals to propagate through the logic gates is negligible.

Timing Simulation

- After the physical design tasks are completed in the chosen technology, timing simulation is performed to verify the circuit realized in the target technology meets the required performance.
- Timing simulation simulates the actual propagation delays in the target technology.

Logic Synthesis

- Synthesis tools automatically map a design composed of simple gates or described with a HDL into an optimized circuit according to the type of logic resources available in the target chip.

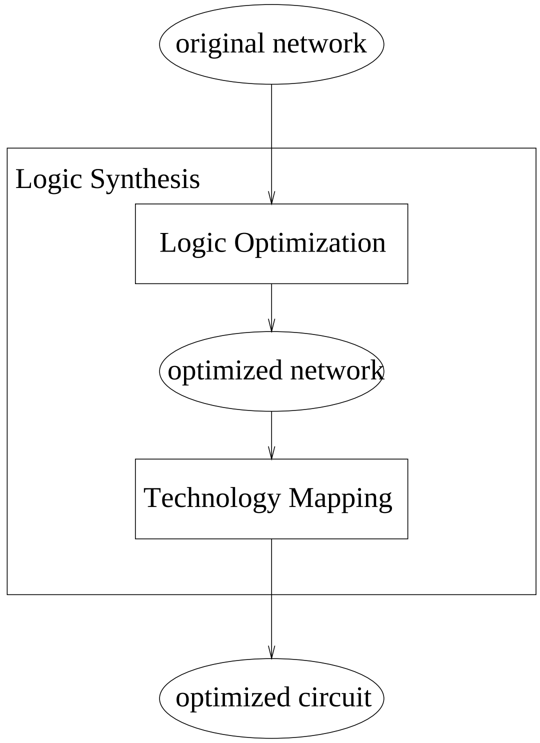
  e.g. If the target is a CPLD, each logic function in the circuit is expressed in terms of the gates available in a macrocell.

  e.g. If the target is a LUT-based FPGA, the number of inputs to each logic function in the circuit needs to match the size of the LUTs.

- The synthesis process is typically divided into two phases:

  i. *Technology independent phase* attempts to generate some optimal representation of the circuit without considering the resources available in the target chip.

  ii. *Technology mapping phase* ensures that the circuit produced by logic synthesis can be realized using the logic resources available in the target chip. e.g. The translation from logic operations to lookup-tables when targeting a design to a LUT-based FPGA.



Logic Synthesis Flow.

Logic Optimization

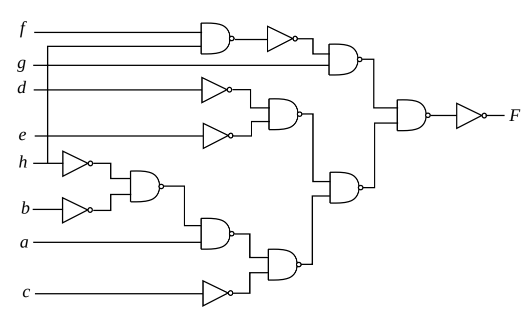
- Technology-independent logic optimization removes redundant logic and simplifies logic wherever possible using techniques like factoring, decomposition, and extraction.
- Some publicly available tools produced by academia
  - Espresso (UC Berkeley) for two-level optimization
  - MIS and SIS (UC Berkeley) for multi-level circuits.
- Commerial products e.g. by Cadence Design Systems, Mentor Graphics, and Synopsys.

Technology Mapping

- Technology mapping maps the technology-independent Boolean network into the target technology by matching pieces of the network with the logic cells available in the technology-dependent cell library (e.g. standard-cell library for a standard-cell based chip, or LUTs for a LUT-based FPGA).
- An important optimization objective is to minimize the area i.e., total area of the cells for covering the network.
- Another important optimization objective is to maximize circuit performance. A commonly used performance metric during technology mapping is the maximum circuit level.

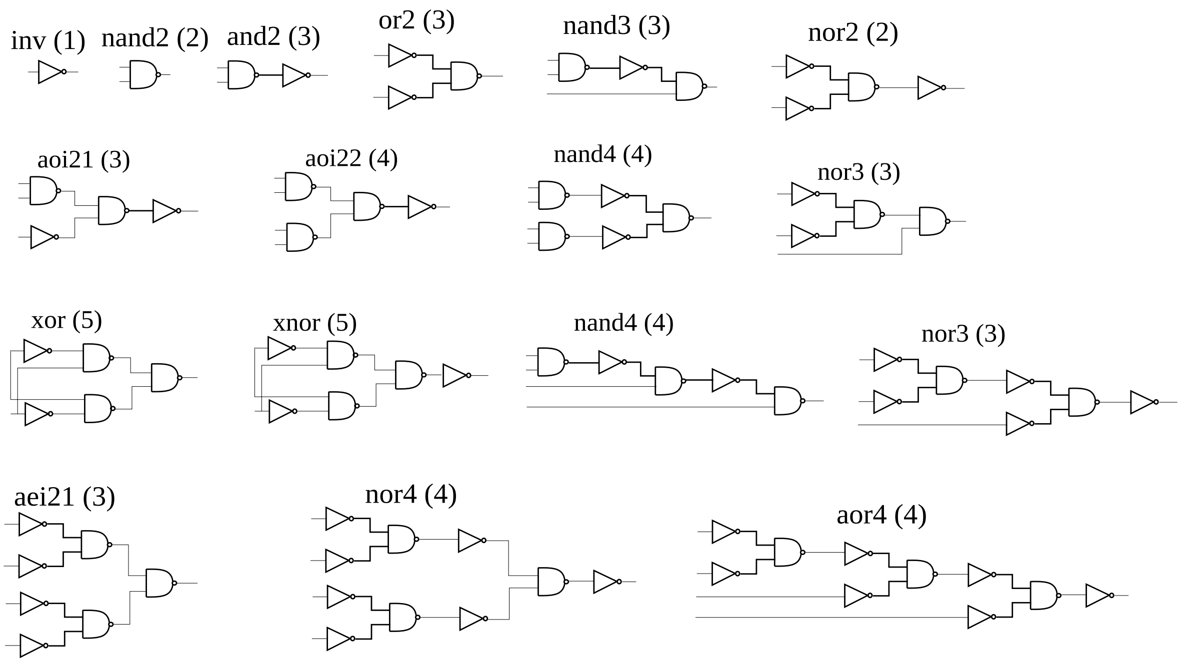
Example

$$
\begin{aligned}
t1 &= a + b\;\, c; \\
t2 &= d + e; \\
t3 &= a\;\, b + d; \\
t4 &= t1\;\, t2 + f\,g; \\
t5 &= t4\;\, h + t2\;\, t3; \\
F &= t5';
\end{aligned}
\quad \xrightarrow{\text{logic optimization}} \quad
\begin{aligned}
t1 &= d + e; \\
t2 &= b + h; \\
t3 &= a\;\, t2 + c; \\
t4 &= t1\;\, t3 + f\,g\,h;
\end{aligned}
$$



An optimized technology-independent Boolean network after logic optimization.

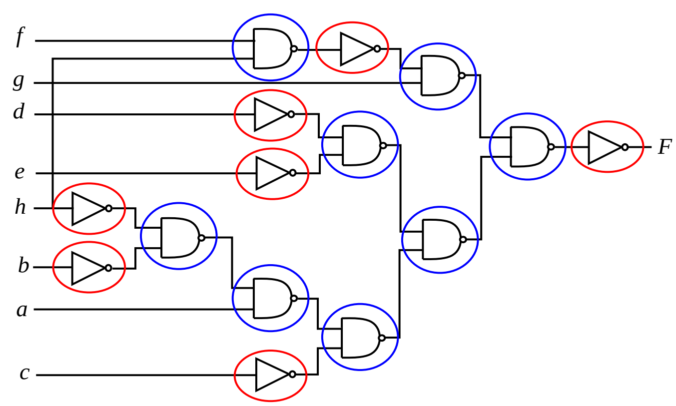
Example (cont'd)



Cells in the target cell library and their area costs .

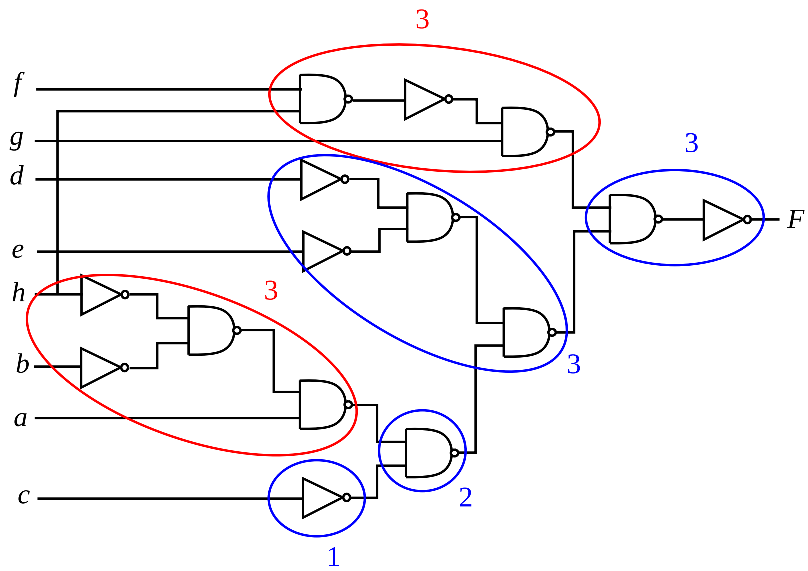
Example (cont'd)

$t1 = d + e;$
$t2 = b + h;$
$t3 = a\ \ t2 + c;$
$t4 = t1\ \ t3 + f\,g\,h;$



A cover using eight nand2 and seven inv for an area cost of 23.

Example (cont'd)



A better covering with an area cost of 15.

A Tree-Covering-by-Trees Approach

- If the *subject graph* and the cells in the library are expressed as trees (typically of 2-input NAND gates and inverters), then an efficient algorithm to find the best cover exists.

- Based on dynamic programming algorithm.

  Given: Subject trees (network to be mapped)
  Forest of pattern graphs (cells in cell library)

  Consider a node v of the subject trees

  - Recursive assumption: For all descendants of v a best match which implements the node is known.
  - Cost of a leaf is 0.
  - Consider each pattern graph which matches at v; compute implementation cost as the cost of the pattern plus the minimum implementation cost of the subtrees rooted at the leaf nodes of the pattern.
  - Choose lowest cost matching pattern to implement v.

- In general, the optimized Boolean network is not a tree and needs to be decomposed into a forest of trees.
- A network can be decomposed into trees by clipping the multiple-fan-out nodes.
- Each tree can be covered optimally using dynamic programming and results are assembled together.
- For LUT technology mapping, we can use the same approach. But in the case of LUTs, the structure of the logic function does not matter in the matching. Given a tree, every subtree that has at most k leaf nodes can be implemented by a single LUT.
- We can also use a similar approach for MUX-based FPGA technology mapping.