

SMS2 Summary

Matthias Jung

21. Juni 2010

1 Hardware Modeling and Synthesis Problem

Synthesis Steps:

- Abstract description with HDL
- Logic Synthesis (Structural description of the circuit using logic gates and registers (Netlist))
- Physical Design

Hardware Description Languages:

- **VHDL**
Very High Speed Integrated Circuit Hardware Description Language.
- **VERILOG**

Original Intentions of HDLs:

- Specification
- Simulation
- Documentation

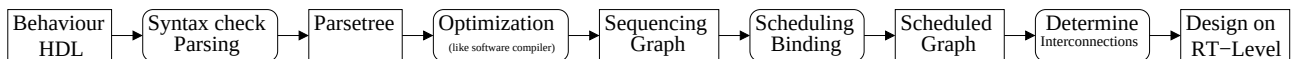
Today Additionally:

- Synthesis
- Testing
- Formal verification

VHDL Characteristics:

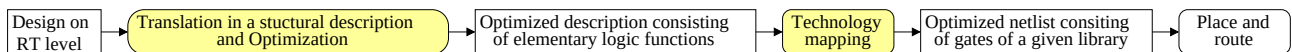
- **Behavioral description:** Description by the sequential flows by *process*.
- **Data flow description:** Structural description of datapath
- **Structural description:** Netlist of simple pre-defined components

High Level Synthesis:



- Optimization problem is very complex, many degrees of freedom exist
- Sophisticated heuristic techniques are available for scheduling and binding
- High-level synthesis tools are available as commercial products
- Not every constraint can be dealt with systematically in a simple way
- An experienced designer usually produces better results

Logic Synthesis:



- **Given:** non optimized netlist consisting of simple logic functions and registers
- **Tools:** library of gate modules supported by the manufacturer
- **Find:** optimized netlist adapted to the library

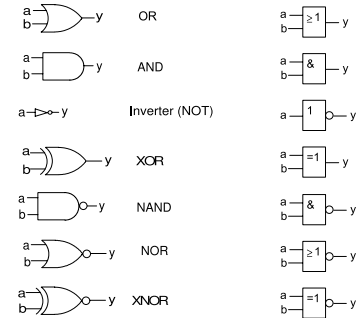
2 Representation of Boolean Functions

The kind of representation of a boolean function has a substantial impact on the cost of the resulting implementation and the algorithmic complexity of the problems that have to be resolved.

Laws of Boolean Algebra:

Idempotency:	$a \cdot a = a$	$a + a = a$
Commutativity:	$a \cdot b = b \cdot a$	$a + b = b + a$
Associativity:	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$	$a + (b + c) = (a + b) + c$
Distributivity:	$a \cdot (b + c) = a \cdot b + a \cdot c$	$a + (b \cdot c) = (a + b) \cdot (a + c)$
Absorption:	$a + ab = a$	$a \cdot (a + b) = a$
Law of 0:	$a + 0 = a$	$a \cdot 0 = 0$
Law of 1:	$a + 1 = 1$	$a \cdot 1 = a$
deMorgans:	$\overline{a \cdot b} = \overline{a} + \overline{b}$	$\overline{a + b} = \overline{a} \cdot \overline{b}$

Gate Symbols:



Boolean Expression:

- a variable is a BE.
- 0,1 are BEs.
- the negation of a BE is a BE
- the conjunction of two BEs is a BE
- the disjunction of two BEs is a BE

$$x = a \cdot b + \overline{a} \cdot b$$

Compact and readable for human but algorithmic treatment is difficult

Truth Table:

Complete list of all points in the boolean space with their corresponding images.

a	b	x
0	0	1
0	1	0
1	0	1
1	1	0

Size grows exponentially with the fanin, barely relevant in practice

Implicant table:

List of sub-spaces (cubes) in the boolean space, so that the cubes cover all the ones of the function but no zeros

a	b	x
-	0	1

Significantly more compact than truth table, good basis for optimization and implementation of two-level circuits

Literal and Monom:

Representation of boolean function in terms of disjunctive and conjunctive forms.

Literal: is a variable in complemented or unimplemented form.

Monom: (product term) consist of (1, literal, conjunction of literals) e.g. $a \cdot \overline{b}$

SOP/DNF (Sum of products): $ab + \underbrace{\overline{a}bc}_{\text{Minterm}(1)} + \overline{b}c$

POS/CNF (Sum of products):

$$(a + b) \cdot \underbrace{(\overline{a} + b + c)}_{\text{Maxterm}(0)} \cdot (\overline{b} + c)$$

CDNF and CCNF (Canonical) :

CDNF: Every boolean function can be expressed as a disjunction of all its one-minterms

CCNF: Every boolean function can be expressed as a conjunction of all its zero-maxterms

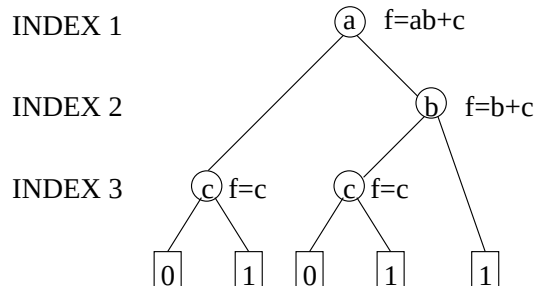
Shannon Decomposition:

Every boolean expression can be converted into its CDNF and CCNF respectively by repeatedly applying the Shannon Decomposition:

$$f = x_i f|_{x_i=1} + \overline{x_i} f|_{x_i=0}$$

$$f = (x_i + f|_{x_i=0}) \cdot (\overline{x_i} + f|_{x_i=1})$$

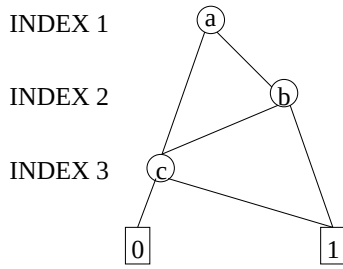
OBDDs: (Ordered Binary Decision Diagram)



Definition:

An OBDD is a directed acyclic graph with one root node. An *index*(ν) is attributed to every non-terminal node. This index refers to the set of input variables. Every non-terminal node has two direct successors $low(\nu)$, $high(\nu)$. A terminal node contains a $value(\nu) \in \{0, 1\}$

- if ν is a 1-terminal node then $f^\nu = 1$
- if ν is a 0-terminal node then $f^\nu = 0$
- if ν is a non-terminal node then $f^\nu = \overline{x_i} \cdot f^{low(\nu)} + x_i \cdot f^{high(\nu)}$

ROBDD: (Reduced OBDD)**Definition:**

A OBDD can be reduced by merging isomorphic subgraphs. A OBDD is called reduced if it contains neither

- a node ν with $low(\nu) = high(\nu)$, nor
- a pair of nodes $\nu, \mu \in V$, so that the subgraphs rooted in ν and μ are isomorphic

Properties of ROBDDs:

An ROBDD is a canonical representation of a boolean function, i.e. if we examine two ROBDDs for the same boolean function, then these ROBDDs are always isomorphic (for a given order of variables). The size of an ROBDD for a given boolean function f varies strongly with the order of variables (optimization problem).

Important boolean operation can be easily performed with ROBDDs, e.g.:

Apply: given two ROBDDs for functions f_1 and f_2 , find the ROBDD for the function that is produced if a boolean function is applied to f_1 and f_2 (*OR, XOR, AND...*)

Compose: given two ROBDDs for functions f_1 and f_2 , find the ROBDD for function f_1 when one of its variables x_i is replaced by f_2

Set Covering Problem:

Given:

- A set M of m elements
- A set τ of subsets of M
 $\tau = \{S_i \subseteq M, i = 1, 2, \dots, n\}$
- M is covered by τ $M = \bigcup_i S_i$

Optimization Problem:

Find a subset T of τ that covers M and contains a minimal number of elements of τ

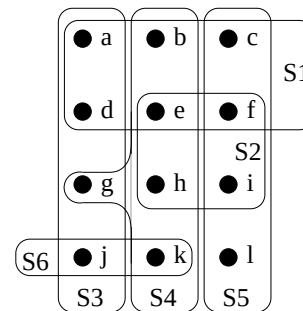
Decision Problem:

Let K be a constant with $0 < K \leq |\tau|$. Is there a subset T of τ that covers M and contains at most K elements of τ ?

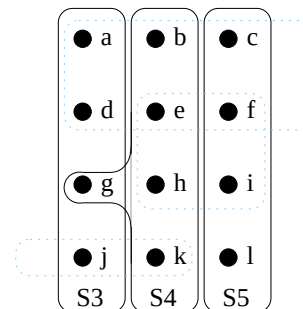
Solution:

- Greedy Heuristic
- ILP Formulation

Set $\tau, M = \{a, b, c, d, e, f, g, h, i, j, k, l\}$:



Subset T : Optimal Cover

**Integer Linear Programming:**

$$\vec{x} \in \{0, 1\}^n$$

$$\vec{x} \cdot \vec{c}^t \quad (\text{Cost function})$$

$$\vec{x} \cdot \vec{A} \geq \vec{b} \quad (\text{Constraint})$$

Application to set covering Problem:

- $\vec{x}$ is an n -dimensional vector indicating if a subset S_i is contained in a set T
- $\vec{c}$ is an n -dimensional vector assigning the costs c_i to the subsets S_i
- $\vec{A}$ is a matrix of dimension $n \times m$ which specifies how the subsets S_i are composed of elements m_i of M
- $\vec{b}$ is an m -dimensional 1-vector then every element is contained in at least one subset and thus $\vec{x}$ is a cover.

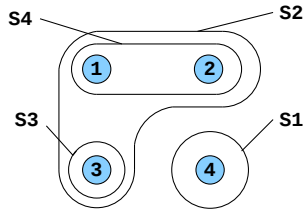
Application of ROBDD to set covering problem:

$$a(\vec{x}) = \prod_{j=1}^m (a_{1j} \cdot x_1 + \dots + a_{nj}x_n) = 1$$

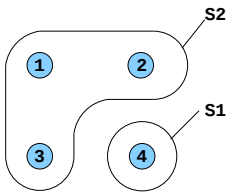
Construct an ROBDD. Every path from root node to terminal node 1 corresponds to a cover. Search for the shortest path to the terminal node to get the optimum cover.

Example:

Problem:



Exact Solution:



ILP Formulation:

$$c = (1, 1, 1, 1)$$

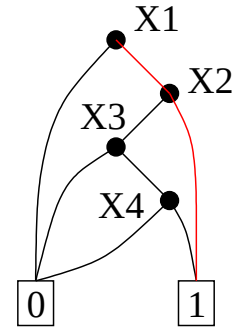
$$(x_1, x_2, x_3, x_4) \cdot \underbrace{\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}}_{\vec{A}} \geq \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{\vec{b}}$$

Solution with IP_Solve:

$$\vec{x} = (1, 1, 0, 0)$$

ROBDD Approach:

$$a(\vec{x}) = (x_1) \cdot (x_2 + x_4) \cdot (x_2 + x_3) \stackrel{!}{=} 1$$

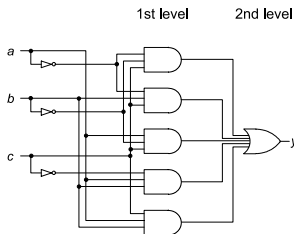


3 Two-Level Logic Synthesis

Two Level Circuit:

Example:

$$y = \overline{a}bc + \overline{a}b\overline{c} + a\overline{b}c + ab\overline{c} + abc$$



Cost Function:

Optimization problem: choose the SOP representation with minimal implementation costs

Definition $cost_{2-level}$:

$$cost_{2-level}(f) > cost_{2-level}(g) \quad \text{if:}$$

- the number of product terms of f is greater than the number of product terms of g ,
- f and g have the same number of product terms, but the number of literals in f is greater than the number of literals in g

Implicant: (Cube)

A product term p is called *implicant* for a function f , if $p = 1 \Rightarrow f = 1$ holds.

Prime Implicant: (Prime Cube)

A implicant is called *prime implicant*, if no literal can be removed from the product term without losing its quality of being an implicant

Essential Prime Implicant: (E. P. Cube)

A prime implicant is called *essential prime implicant* if its part of every potential cover of f .

Basic Procedure of two level minimization:
(Method of Quine and McCluskey)

Given:

a boolean canonical POS function, take all implicants

Step 1:

determine all prime implicants (it.cons., shannon)

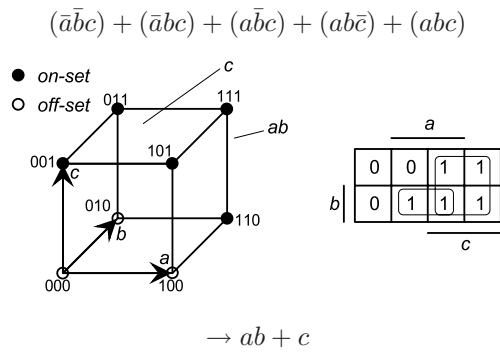
Step 3:

choose a set of prime implicants so that f is covered and $cost_{2-level}$ is minimized.

Theorem: (Quine)

A minimal SOP (according to the cost function $cost_{2-level}$) consists exclusively of prime implicants.

Cube Representation:



Instead of *implicant*, *prime implicant* or *essential prime implicant*, the terms *cube*, *prime cube* or *essential prime cube* are used as well.

1. Calculation of all Prime Implicants Using Iterative Consensus

$$xy + \bar{x}z = xy + \bar{x}z + \underbrace{yz}_{C.T.}$$

Procedure:

- 1: calculate the consensus terms for all pairs of product terms not examined so far and add them to the SOP
- 2: remove every term that is covered by another term.

Repeat 1. and 2. until no new consensus terms are generated. The the product terms off the SOP are all the prime implicants of f . The SOP of all prime implicants of f is called **complete sum**.

2. Calculation of all Prime Implicants Using Iterative Shannon Decomposition

Theorem: Product of Complete Sums

Given are two complete sums f_1 and f_2 . The complete sum for $f = f_1 \cdot f_2$ can be determined by the following steps:

- 1: Form the expression $f_1 \cdot f_2$ and transform it into SOP form using the distributive law, idempotency and complement rule
- 2: Remove all product terms which are covered by other product terms

$ABS(f)$: Simplifies as given by the *theorem*

$CS(f)$: Generate complete sum

$$CS(f) = ABS((x_i + CS(f|_{x_i=0})) \cdot (\bar{x}_i + CS(f|_{x_i=1})))$$

Solution of the Covering Problem in two-level minimization

Problem statement:

Given is a function f and a set τ of all prime implicants of f . Determine a subset T of τ so that the prime implicants in T cover the function f and the costs $costs_{2-level}$ are minimal.

After reducing the covering matrix by utilizing row and column dominance, the covering problem is solved by branch and bound or by BDDs.

Example Iterative Consensus:

$$\begin{aligned}
 & ab + \bar{b}c + bcd \\
 & \quad \swarrow \quad \searrow \\
 & \quad \quad \bar{a}c \quad cd \\
 & \quad \quad \swarrow \quad \searrow \\
 & ab + \bar{b}c + \bar{b}cd + ac + cd \\
 & \quad \quad \swarrow \quad \searrow \\
 & ab + \bar{b}c + ac + cd \\
 & \text{Complete Sum}
 \end{aligned}$$

Example Shannon for Prime Implicants

$$\bar{b}c + bcd$$

Obviously with iterative Consensus: $\bar{b}c + bd$

$$\begin{aligned}
 CS &= ABS((b + c) \cdot (\bar{b} + cd)) = ABS(c\bar{b} + \cancel{bcd} + cd) \\
 &= c\bar{b} + cd
 \end{aligned}$$

Positional Cube Notation:

Positional Cube Notation is a data structure for the representation of two-level circuits. It permits efficient implementations of boolean transformations

Decode Product term as follows:

$$c = (\underbrace{x_1, \dots, x_n}_{InputCube} \mid \underbrace{y_1, \dots, y_m}_{OutputCube})$$

$y_j = 1$ if the input cube is an implicant for $f_j(x)$ and is contained in the selected cover of $f_i(x)$
 $y_i = 0$ if the input cube is not an implicant for $f_j(x)$ or it is an implicant but not in the selected cover of $f_i(x)$

x_i	Value
a	01
$\bar{a}$	10
D.C.	11

Two Level Logic Syn Heuristic minimization of two level circuits:

Problem of the exact minimization of two-level circuits is the complexity of big circuits. A function with n variables can have up to $\frac{3^n}{n}$ prime implicants. The covering problem that has to be solved is NP-complete. For big circuits a heuristic approach is unavoidable.

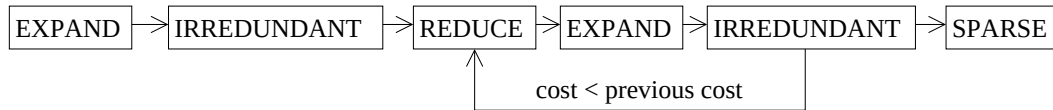
Example for Positional Cube Notation:

$$y_1 = ab + \bar{b}cd + \textcolor{red}{a}\bar{c} \quad \text{and} \quad y_2 = bc + ab\bar{d} + \textcolor{red}{a}\bar{c}$$

$$\left(\begin{array}{cccc|cc}
 1 & 1 & - & - & 1 & 0 \\
 - & 0 & 1 & 1 & 1 & 0 \\
 1 & - & 0 & - & \textcolor{red}{1} & \textcolor{red}{1} \\
 - & 1 & 0 & - & 0 & 1 \\
 1 & 1 & - & 0 & 0 & 1
 \end{array} \right)$$

3.1 ESPRESSO

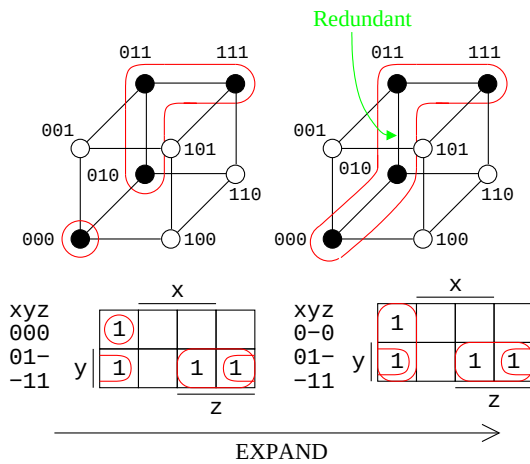
ESPRESSO is a famous heuristic algorithm for minimization of two level circuits. As input ESPRESSO receives a initial set of all implicants. This set is the first solution (probably a really bad solution) which will be improved stepwise.



REDUCE and EXPAND reduce and expand the *input* and *output* cubes. SPARSE reduces the *output* cube again after the last EXPAND and executes additional expansions of *input* cubes that may have become feasible after the final reduction.

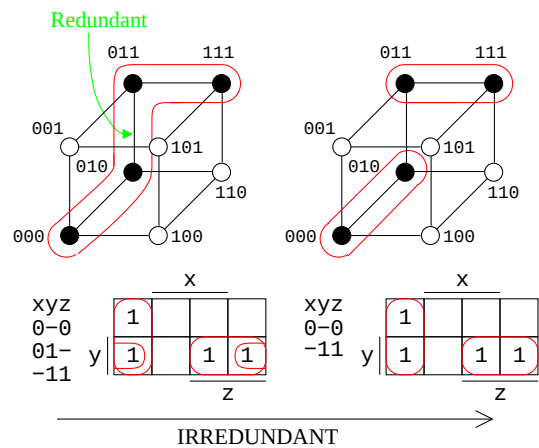
EXPANSION of an Input Cube:

Replace one or more ones or zeros with '-' (remove one or more literals from a product term)



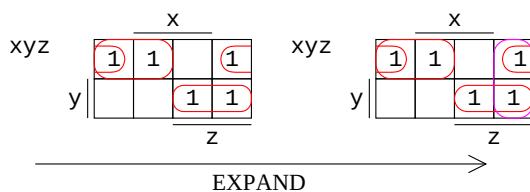
IRREDUNDANT:

Remove a cube from the cover of all functions (remove a row in the cube table)



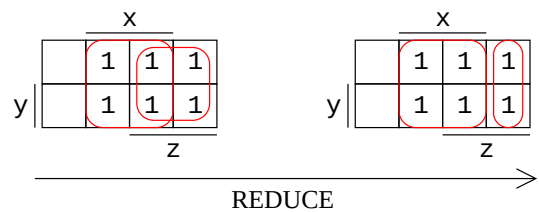
EXPANSION of an Output Cube:

Replace one or more zeros in Output Cube with 1 (add one or more product terms (implicants) to the cover)



REDUCTION of an Input Cube:

Replace '-' in one or more positions of the input cube with 1 or 0 (add one or more literals to a product term)



REDUCTION of an Output Cube:

Replace one or more ones on output cube with zeros (remove implicants from the cover)

Problems to Consider:

- Which expansions and reductions are valid, i.e. lead to a valid cover of the on-set? How do I check covering relations?
- Which direction shall be selected for expansion and reduction?

Condition for the Validity of EXP/RED:

Let F be the cover and D a don't-care set and c_i a cube to be expanded or reduced. Let c'_i be a cube which covers these minterms that are either not covered anymore or newly covered.

$$c'_i \subseteq (F \setminus \{c_i\}) \cup D$$

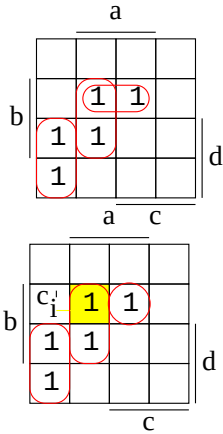
Implementation with positional cube notation and decomposition into cofactors.

Rules for Validity Proof with Positional Cube Notation:

$$c'_i \subseteq f \iff f|_{c'_i} = 1$$

Let F be a cover and c be a cube in positional cube notation. The cofactor $F|_c$ is calculated as follows:

1. Remove all rows of F , whose input cubes conflict with the input cube of c . There is a conflict for an input cube, if there exists a column for which the input cube of F is 1 and c is 0, or vice versa.
2. Remove all rows of F for which a column in the output cube where c and F are 1 doesn't exist.
3. Remove all columns of F in the output cube where c has a 0 and all columns in the input cube where c is not '1'.



Example in Algebraic Notation:

$$f = ab\bar{d} + ab\bar{c} + \bar{a}\bar{c}d$$

$$c_i = ab\bar{d} \quad c'_i = ab\bar{c}\bar{d}$$

$$F \setminus \{c_i\} = ab\bar{c} + \bar{a}\bar{c}d$$

$$c'_i \subseteq F \setminus \{c_i\} \iff ab\bar{c} + \bar{a}\bar{c}d|_{ab\bar{c}\bar{d}=c'_i}$$

$$= b\bar{c}|_{b\bar{c}\bar{d}} = \bar{c}|_{\bar{c}\bar{d}}$$

$$= 1|_{\bar{d}} = 1$$

Example in Positional Cube Notation:

$$F = \begin{pmatrix} 1 & 1 & - & 0 \\ 1 & 1 & 0 & - \\ 0 & - & 0 & 1 \end{pmatrix} \begin{matrix} |1 \\ |1 \\ |1 \end{matrix} \quad c_i = (11-0|1)$$

$$c'_i = (1100|1)$$

$$F \setminus \{c_i\} = \begin{pmatrix} 1 & 1 & 0 & - \\ 0 & - & 0 & 1 \end{pmatrix} \begin{matrix} |1 \\ |1 \end{matrix}$$

$F \setminus \{c_i\}|_{c'_i}$ is now calculated with the rules:

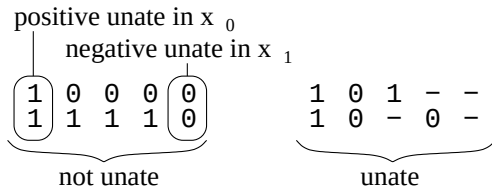
$$1 \rightarrow (110-|1)$$

$$2 \rightarrow (110-|1)$$

$$3 \rightarrow (|1)$$

Definition Unate:

The cube set F is called positive / negative unate in a variable x_i if all cubes have an entry 1/0 or '-' in position i . A Cube set F is called unate if it is unate for all i



Unate Recursive Paradigm:

choose a decomposition of a boolean function into cofactors (shannon) so that unate cofactors are obtained in a few steps as possible. By exploiting properties of unate functions the problems to be solved are substantially simplified

- Every prime cube of a unate cover is an essential prime cube

Theorem (tautology of a unate cover)

A unate cover F is a tautology for an output function if and only if F contains an input cube which only consists of don't-cares. Tautology checking is playing a major role in the operations Irredundant and Reduce.

$$\begin{pmatrix} 1 & 0 & 1 & - & - \\ - & - & - & - & - \\ 1 & 0 & - & 0 & - \end{pmatrix}$$

unate and tautology

Tautology Checking

Perform Shannon-decompositions until the resulting cofactors are unate. In each step, choose a splitting variable with respect to which the function is not unate, and which has many occurrences (a few don't-cares). When a unate cofactor is obtained the above theorem can be applied.

Tautology checking is playing a major role in the operations IRREDUNDANT and REDUCE.

Use Complement Cover for Expansions:

With the cover $\bar{F}$ valid expansions can be derived without the need of tautology checking. A Blocking matrix B is a matrix that indicates for a cube p of the cover F to be expanded if an expansion in the variable x_i conflicts with a cube q_i of the cover $\bar{F}$.

Maximum Expansion of the Cube p :

Find a minimal subset of columns in b so that for every row a column exists which has a 1 in this row. The columns not contained in this subset, belong to variables which can be used for expanding the cube p .

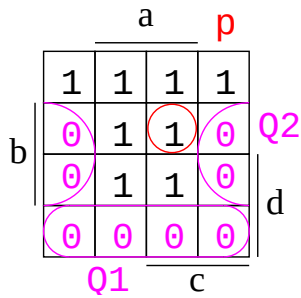
Complement Generation:

$$\bar{f} = x \cdot \overline{f|_{x=1}} + \bar{x} \cdot \overline{f|_{x=0}}$$

Perform Shannon-decomposition in a way that one of these special cases are reached as soon as possible:

- F contains a column for variable x with 1 in every row: $\bar{f} = \overline{f|_{x=1}} + \bar{x}$
- F contains a column for variable x with 0 in every row: $\bar{f} = \overline{f|_{x=0}} + x$
- F is positive unate in x : $\bar{f} = \overline{f|_{x=1}} + \bar{x} \cdot \overline{f|_{x=0}}$
- F is negative unate in x : $\bar{f} = x \cdot \overline{f|_{x=1}} + \overline{f|_{x=0}}$

Example for Blocking Matrix:



Initial Point

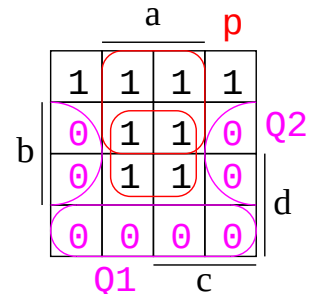
$$\bar{F} = \begin{pmatrix} Q1| & - & 0 & - & 1 \\ Q2| & 0 & 1 & - & - \end{pmatrix}$$

$p = (1 \ 1 \ 1 \ 0)$
Conflicts from p with $\bar{F}$ lead to B :

$$B = \left(\begin{array}{c|cccc} Q1 & 0 & 1 & 0 & 1 \\ Q2 & 1 & 0 & 0 & 0 \end{array} \right)$$

There is a Conflict when omit a or $(b$ and $d)$

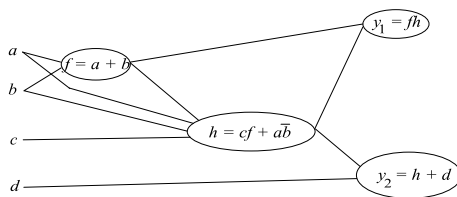
$p_1 = (1 \ 1 \ - \ -) \vee p_2 = (1 \ - \ - \ 0)$
 So omit c and d or omit a and c



Potential covers for that
initial point

4 Multilevel Logic Synthesis

Representation of a Boolean Network:



Functional Decomposition:

Functional decomposition is particularly used in the realization with FPGAs to decompose the circuit according to the size of the CLBs. The actual optimization of the circuit is mostly done by using other methods (e.g. boolean/algebraic methods based on division).

Boolean and Algebraic Product:

The conjunction of two boolean expressions f q is called a boolean product. In the special case where f and q have no common variables, f q is called an algebraic product.

Boolean Product: $(a + b) \cdot (c + \bar{b})$

Algebraic Product: $(a + b) \cdot (c + d)$

Boolean Division:

$$(q, r) = \text{DIV}(y, f) \quad \text{with:} \quad y = f \cdot q + r$$

If $f \cdot q$ is an algebraic product the division is called **algebraic division**.

Algebraic division is unique, boolean division is not. f is called a algebraic divisor if $q \neq 0$ and $f \cdot q$ is a algebraic product.

Approach in Boolean Division:

- Extend the dividend y by a variable x which represents the divisor f in the expression to be built
- Construct the don't-care set, $DC = x \oplus f = \bar{x}f + \bar{f}x$
- Minimize y using this don't-care set

In Practice: a heuristic suggests a divisor, the boolean network is collapsed into two-level form, this form will be optimized with the don't-care set (ESPRESSO) the optimized function is factorized again.

Example Algebraic Division:

$$(ac + bc + a\bar{b}) : (a + b)$$

Examine cube a : $c, \bar{b}$

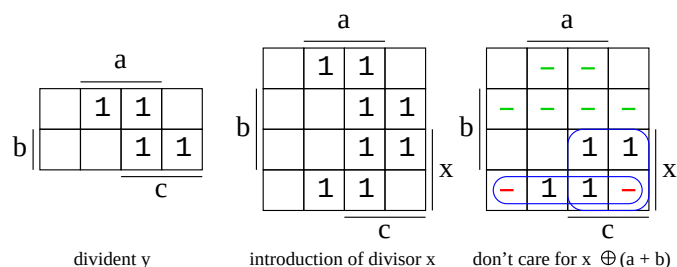
Examine cube b : c

$$y = (\textcolor{red}{c}) \cdot (a + b) + \underbrace{a\bar{b}}_{\text{remainder}}$$

Example Boolean Division:

Introduce a new variable x for the divisor, x is not a independent variable, it has the same values as $a + b$

→ don't care conditions for y : $DC = x \oplus (a + b)$



$$y = xc + \bar{b}x = x(c + \bar{b}) = (a + b)(c + \bar{b})$$

Cube-Free:

A SOP expression y is called cube-free, if no SOP expression q and no cube c with $c \neq 1$ exists, so that $y = cq$ (algebraic indivisible)

Kernels:

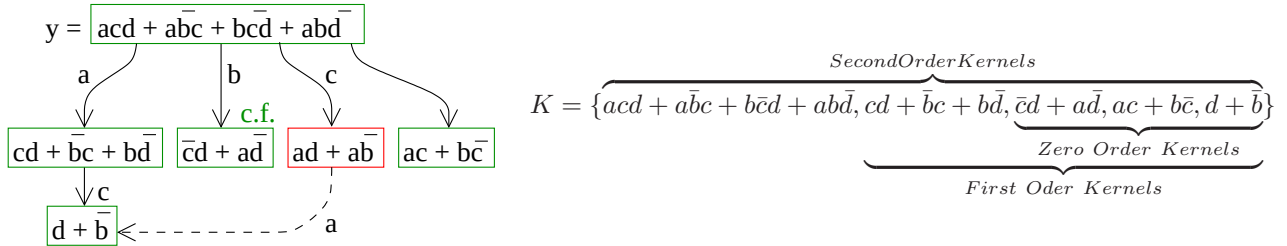
Which expressions are good divisors? The set

$$K(y) = \{y/c | c \text{ is a cube and } y/c \text{ is cube-free}\}$$

is called kernel set of y . A cube c belonging to a kernel $k \in K(y)$ is called co-kernel of k .

Determination of all kernels is often too complex, thus limitation to subsets of kernels. There are approaches, that calculate 0-order kernels first and then stepwise proceed with kernels of higher order.

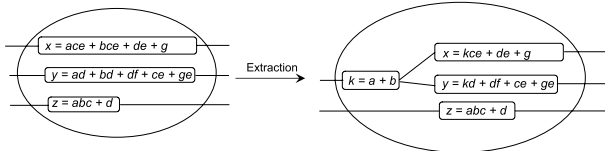
Example for Determination of Kernels:



5 Optimization Strategies

Extraction:

Generates a new node in the boolean network, which corresponds to a common divisor for the functions of the successor nodes. The function of the successor nodes is determined using division.



1. Calculate all kernels
2. Determine the intersection of the kernels and determine a common divisor
3. Introduce a new network node k
4. Divide and connect with the new node

Theorem:

Two SOP expressions f and g have a common multiple-cube divisor d if and only if two kernels $k_f \in K(f)$ and $k_g \in K(g)$ exist, so that the cube representations of k_f and k_g contain common cubes $p_1, p_2, \dots, p_m$ with $m \geq 2$. Then, the cube set $d = p_1 + p_2 + \dots + p_m$ is the common divisor.

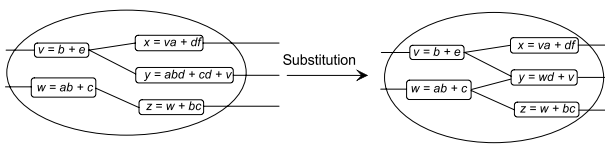
Factorization and Decomposition:

Factorization simplifies the representation of a node in the boolean network but does not change its topology. If a new node in the boolean network is introduced for every factor (thus changing the topology of the network), this is called decomposition.

$$y = ace + bce + de + f \Rightarrow ((a + b)c + d)e + f$$

Substitution:

Generates a new branch in the boolean network. The representation of a node y is simplified by using another network node w as divisor for y , so that y can be simplified by computing y/w (which may also result in other branches being removed).



Simplification

Does not change the topology of the network (unless a variable is completely removed). The representation of (a part of) the Boolean network is simplified, e. g. by the following approach:

1. Generate a single node representation of the considered network function in two level representations (**Collapsing**)
2. Optimization e.g. with ESPRESSO
3. Factorization / decomposition

6 Internal Don't Cares

Controllability Don't Cares: (CDC)

Not all combinations of values are possible for the signals of a boolean network. Let S be a variable associated with a node function $f(x)$ of the boolean network: every combination of value assignments that fulfills $s \oplus f(x) = 1$ can not occur and belongs to the set of CDC.

$$CDC = \sum_i (S_i \oplus f_i(x))$$

In Practice the Calculation is too complex in most cases!

Observability Don't Cares:

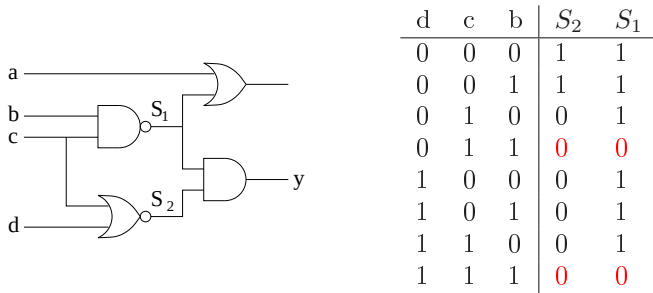
For some value assignments, the function value of an internal network node can not be observed at any primary output of the network. Then, it does not need to be specified.

$\frac{\partial f}{\partial x} = f|_{x=1} \oplus f|_{x=0}$ is called boolean difference.

For the set of observability don't cares at a node v_x in a boolean network G holds: $ODC_x = f|_{x=1} \oplus f|_{x=0}$ where f is the function implemented by G

$$ODC_{global}(x) = \prod_i f|_{x=1} \oplus f|_{x=0}$$

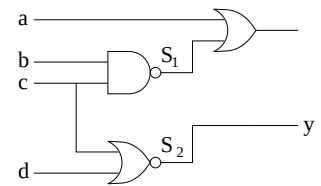
Example Controllability Don't Care:



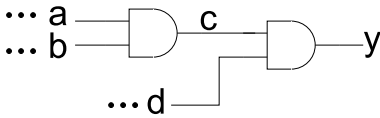
we observe: $S_1 = 0 \Rightarrow S_2 = 0$

$\rightarrow y = S_2$

S_1	S_2	y
0	0	0
0	1	0
1	0	0
1	1	1



Example Observability don't Care:



When line d is assigned a logic 0, the value on the output y remains the same regardless of the logic value on lines a, b, c . As such, variables a, b, c are declared ODCs and marked lazy.

7 Timing Analysis

Linear Delay Model:

... for a node v_j in a boolean network:

$$d(v_{ij}) = \alpha_i + \beta_i \cdot \sum_{k=1}^m \gamma_k$$

d : Total delay of the gate from input pin i to the gate output

α_i : Load independent delay from input pin i to the gate output

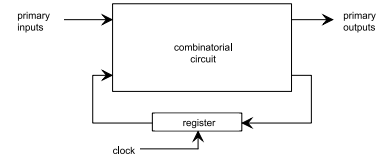
β_i : Load dependent delay from input pin i to the gate output

γ_k : Capacitive load of the **succeeding** gate

m : number of succeeding gates

Clock Speed:

Most digital systems consist of one or more synchronous sequential circuits. The maximum switching delay between inputs and outputs of a combinatorial circuit determines the maximum clock speed.



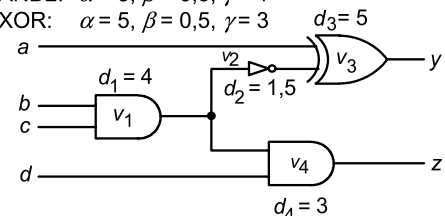
Example for Delay Model:

Delay data for gate:

INV: $\alpha = 0, \beta = 0.5, \gamma = 1$

AND2: $\alpha = 3, \beta = 0.5, \gamma = 1$

XOR: $\alpha = 5, \beta = 0.5, \gamma = 3$



Topological Analysis:

For each node v_i of the circuit, we are interested in the time t_i , when the signal has stabilized. The time t_i is called *data ready time* or *data arrival time*.

Data Ready Time:

$$t_i = d_i + \max_{j: (v_j, v_i) \in E} (t_j)$$

Required Data Ready Time:

$$t'_i = \min_{j: (v_i, v_j) \in E} (t'_j - d_j)$$

Slack:

$$s_i = t'_i - t_i \quad \forall v_i \in V$$

If $s_i = 0$, the node v_i is called topologically critical. All paths that contain only nodes which are topologically critical are called topologically **critical paths**.

False Path Problem:

If we calculate a critical path on which the signal can not propagate.

Example: We calculate the Critical Path $acdyz$ but the signal can only propagate if $e = 1$ and $e = 0$ at the same time.

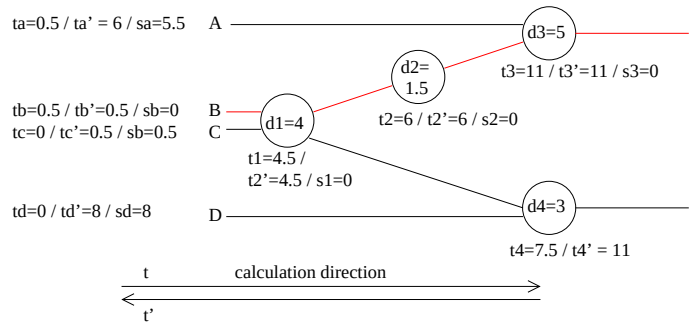
Boolean Difference:

$$\frac{\partial f}{\partial x} = f|_{x=1} \oplus f|_{x=0}$$

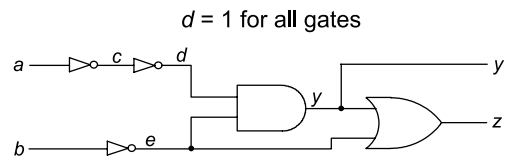
Critical Path:

A critical path in a combinational circuit is a path, where a signal change can be propagated from the start node to the end node and whose end node shows a maximum data arrival time.

Example: (Bellman Ford...)



Example for a False Path:



Dynamic Sensitizability:

Let $p = (v_1, v_2, \dots, v_m)$ be a path in a combinational circuit. The path p is called *dynamically sensitizable*, if:

$$\frac{\partial f_{v_i}}{\partial v_{i-1}} = 1 \quad \forall i = 1, 2, \dots, m \quad (*)$$

hold for every time t_{i-1} when the signal arrives at the gate v_i . A signal change can propagate along a path p if and only if p is dynamically sensitizable.

7.1 Timing Models

Examine the delays of a circuit when applying two sequential input stimuli s_1 and s_2

Model of the Circuit:

Transition Mode:

The signals of the circuit keep the values generated by s_1 until the signals of s_2 arrive. Hence, the delays of the gates and the circuit depend on s_1 and s_2

Floating Mode:

The signals of the circuit do not keep the values generated by s_1 . Hence, the delays of the gates and the circuit depend only on s_2 . For the preceding value, the worst-case is assumed.

Model of the Components (Gates):

Fixed Delay Model:

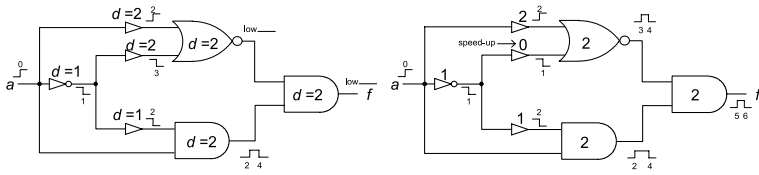
A fixed delay is assigned to every gate.

Bounded Delay Model:

An interval of delays $[0, d]$ is assigned to every gate.

Monotonous Speed-Up:

A delay model fulfills the property of monotonous speed-up, if the delay reduction of a circuit component (gate) either reduces the total delay of the circuit or keeps it unchanged. A delay model with the property of monotonous speed-up is also called robust.



For the transition mode model, the property of monotonous speed-up is not fulfilled, therefore it is not robust.

7.2 Consideration of all Possibilities for Combining the Models for Circuits and Gates:

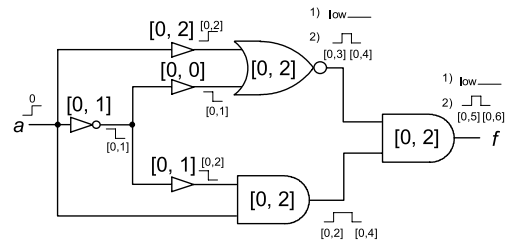
a) Transistion Mode - Fixed Delays:

Determine fixed values for the gate delays by worst-case assumptions, i.e. choose for each gate the maximum of the delay interval as the fixed delay

This Combination is not robust, so this approach leads to an optimistic (!) approximation of the circuit delay, if a gate delay is overestimated (pessimistic).

→ This is not useful for the timing analysis of a circuit in practice.

b) Transition Mode - Bounded Delays:



- Fairly precise modeling of the timing but determination is too complex

c) Floating Mode - Bounded Delays:

In floating model, the most pessimistic assumption is made for the preceding gate, i.e. the possibility that a preceding value speeds up the signal transition at a gate is neglected.

The monotonous speed-up property holds, and the bounded delay model yields the same results as a model that assumes fixed worst-case values for the gate delays. (d)

→ The floating mode model needs to be considered only in combination with fixed gate delays.

d) Floating Mode - Fixed Delays:

controlling value of a gate input:

logic value that controls the value at the gate output, the negation of this value is called non-controlling value.

controlled value of a gate output:

logic value that the gate output assumes, if the controlling value is assigned to one of the gate inputs, the negation of this value is called non-controlled value respectively.

7.3 Dynamic Analysis

Dynamic Analysis takes into account the data arrival times for the sensitization at each gate according to the definition of dynamic sensitizability.

```

Delay Simulate: (nodes are in topological order)
for each ( V as vi )
{
    if( vi is set to controlled value)
    {
        t(vi)=min( t(predecessors(vi) + d);
    }
    else // vi is set to non-controlled
    value
    {
        t(vi)=max( t(predecessors(vi) + d);
    }
}

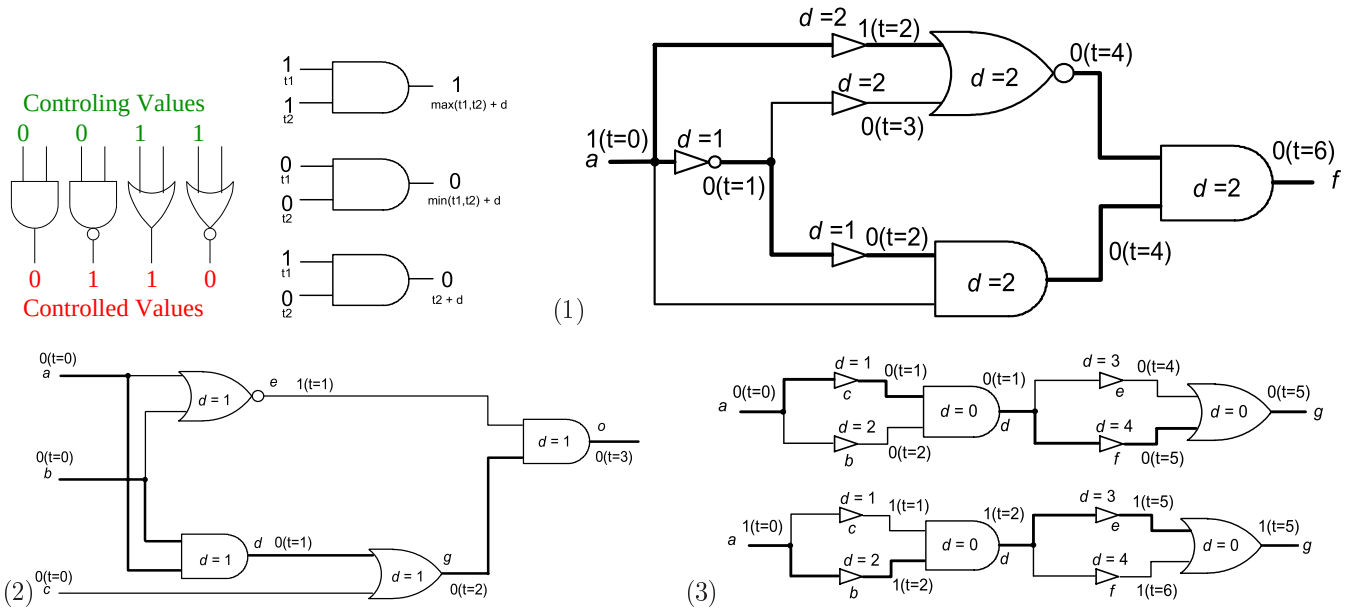
```

Determination of the critical path for dynamic analysis in the floating mode model: (Chen-Du)

Perform `delay_simulate()` and analyze as follows:

1. If the gate output is set to *controlled* then all input nodes are critical which are set to *controlling* and whose data arrival times are minimal.
2. If the gate output is set to non-controlled then all the input nodes are critical, whose data arrival times are maximal.
3. The critical paths of the circuit consist of the nodes that are critical for those input stimuli that cause the maximum data arrival time for the circuit output.

Examples Floating Model and Chen-Du-Criterion:



7.4 Static Analysis

Static Analysis:

Examination of the circuit in the steady state (merely logic analysis, the data arrival times of the individual signals are not of interest) *static sensitizability*

Let $p = (v_1, v_2, \dots, v_m)$ be a path in a combinational circuit. The path is called statically sensitizable, if a signal assignment exists for the inputs of the circuit, such that Eq. (*) holds.

Co-Sensitizeable:

Given a path $p = (x_0, x_1, \dots, x_m)$, an input assignment is said to statically co-sensitize p to 1(0) if and only if $x_m = 1$ ($x_m = 0$) holds and x_{i-1} is set to the controlling value, if x_i is set to the controlled value, $\forall x_i, i = 1, 2, \dots, m$.

Example for Circuit (2) with Static Analysis:

$$o: \frac{\partial(e \cdot g)}{\partial y} \stackrel{!}{=} 1 = 0 \oplus e = e \rightarrow \mathbf{e} = \mathbf{1}$$

$$g: \frac{\partial(d + c)}{\partial d} \stackrel{!}{=} 1 = c \oplus 1 = \bar{c} \rightarrow \mathbf{c} = \mathbf{0}$$

$$d: \frac{\partial(a \cdot b)}{\partial a} \stackrel{!}{=} 1 = 0 \oplus b = b \rightarrow \mathbf{b} = \mathbf{1}$$

But b is 0 in the dynamic calculation!

Static sensitizability: sufficient for signal propagation, but not necessary

→ A delay model based on static sensitizability can underestimate circuit delays and is therefore incorrect

Statoc Co-Sensitizeable:

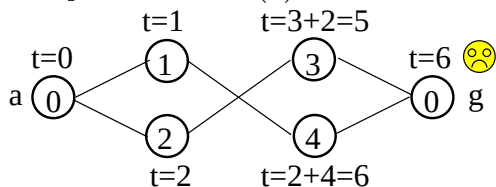
Static co-sensitizability is a necessary condition for a critical path, therefore it yields a correct (non-optimistic) delay model.

Possible Approach:

1. Determination of the data arrival times: as in topological analysis.
2. Determination if the critical path: path with maximum data arrival time at the end node which is statically co-sensitizeable.

Static co-sensitizability can be formulated as a boolean condition and can be evaluated e.g. using OBDDs, SAT.

Example of Circuit (3) with Static Co-Sensitizable Approach:



Co-sensitizability of P yields a maximum circuit delay of 6. However, simulation of both possible signal changes at a (cf. dynamical analysis) yields a maximum circuit delay of 5.

→ Static co-sensitizability is pessimistic

Conclusion:

- Floating mode model with fixed values for the gate delays is less complex than the transition mode model but more pessimistic than the transition mode model.
- Static Analysis is less complex than dynamic analysis but more pessimistic than dynamic analysis.
- Best is the bounded transition model, but this is too complex to handle.
- For complexity reasons, usually only the topologically critical paths are considered in practice!

8 Technology Mapping

8.1 Standard Cells

Problem:

Given: technology-independent optimized description of the circuit (boolean network) and a **cell library**

Find: a network of elements from the library such that the network implements the same boolean function and the costs are minimized

Match:

A match between a cell and a sub-network of the given boolean network occurs, if the sub-network can be replaced by the cell library element without changing the function of the circuit.

Basic Approach:

1. **Matching:** determine all matches, which are possible for the elements of the given cell library in the given boolean network
2. **Covering:** determine a subset of matches, so that the complete boolean network is covered with minimum costs and the network of library elements implements the same function as the technology-independent boolean network.

Formulation as Covering Problem:

Every match m_i causes costs k_i :

$$\min \sum_i m_i \cdot k_i$$

For each gate g_j exists a match (condition c_i):

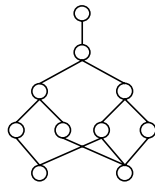
$$\prod_j c_j(m_i) = 1$$

The intermediate variables required for the covers have to be available

$$m_i \Rightarrow v_i(m_k) \stackrel{\Delta}{=} \bar{m}_i + v_i(m_k) = 1$$
$$\Rightarrow \prod_j c_j(m_i) \cdot \prod (\bar{m}_i + v_i(m_k)) = 1$$

Leaf DAG:

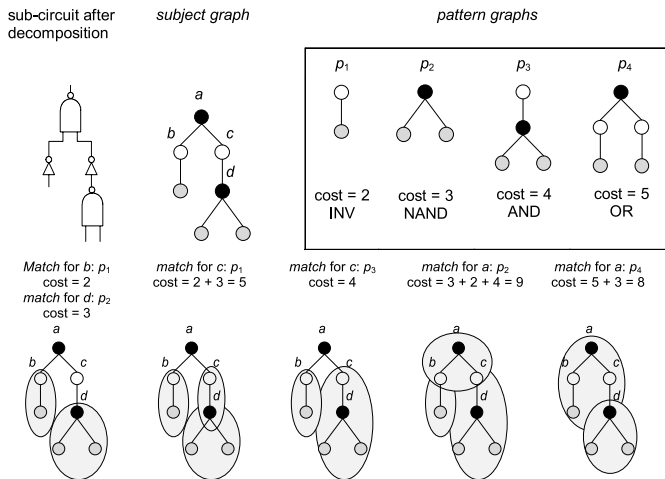
A leaf dag is a directed acyclic graph which has exactly one root node and all nodes except the start node and the end node have exactly one immediate predecessor



Heuristics Approaches: Structural Comparison:

- **Pre-processing** of the boolean network:
 - decomposition:** (e.g. NAND)
 - partitioning:** The circuit is partitioned into sub-circuit with leaf-dag structure.
- After Preprocessing:
 - matching**
 - covering** (dynamic programming)

Example:



Dynamic Programming: Traverse the subject graph starting at the leaves (bottom-up):

- For each node of the subject graph, determine all matches with the pattern graphs of the cell library.
- Determine the costs for each match. The costs of a match are the costs of the cells corresponding to the match plus the costs for the cover of the input node of the cell. The costs for the cover of a node in the subject graph is the minimum of the costs of all matches for this node.
- Save the costs for the cover of a node and the corresponding cell.

The total costs for the cheapest cover of the subject graph are the costs of the cover of its root node. The cheapest cover can be assembled from the saved cells.

Technology mapping taking timing into account:

Problem: in dynamic programming: the values for γ are not known, until the succeeding gate is known!

Solution:

- for every value of γ contained in the library calculate the optimal solution for the given subject graph.
- traverse the circuit backwards starting at the outputs and choose for each node the optimal cover depending on the load values of the topologically succeeding (already determined) gates.

Classic Flow

"Throw it over the wall"

e.g. violation of timing requirements

e.g. "congestion"



Separation of logic synthesis and layout design leads to more and more (expensive!) iterations in the classic flow.

⇒ layout-driven synthesis

Layout Driven Synthesis:

Combination of algorithms for logic and layout synthesis:

Logic optimization before placement, but estimation of timing data and routability by pre-placement using "pseudo cells"

- + relatively small interventions in the standard flow
- + relatively exact layout information
- timing data estimated from pre-placement can be inaccurate
- close coupling causes constraints concerning logic transformations

8.2 FPGAs

LUT:

Application to library-based approaches?

Generate a library cell for each function that can be implemented in the LUT. Apply the approaches developed for standard cells

Problem: the number of functions that can be implemented in the LUT is too big, even if symmetries between variables are taken into account

⇒ no library

Adaption (Dynamic Programming):

given: LUT with k inputs and 1 output...

- **match:** exists for every sub-tree of the subject graph with at most k leaves
- The costs of each *match* are 1
- subject graph has tree structure or leaf-dag structure and may consist of arbitrary gates (nodes) with no more than k inputs

MUX: Library-based approaches for MUX-based architectures are feasible!