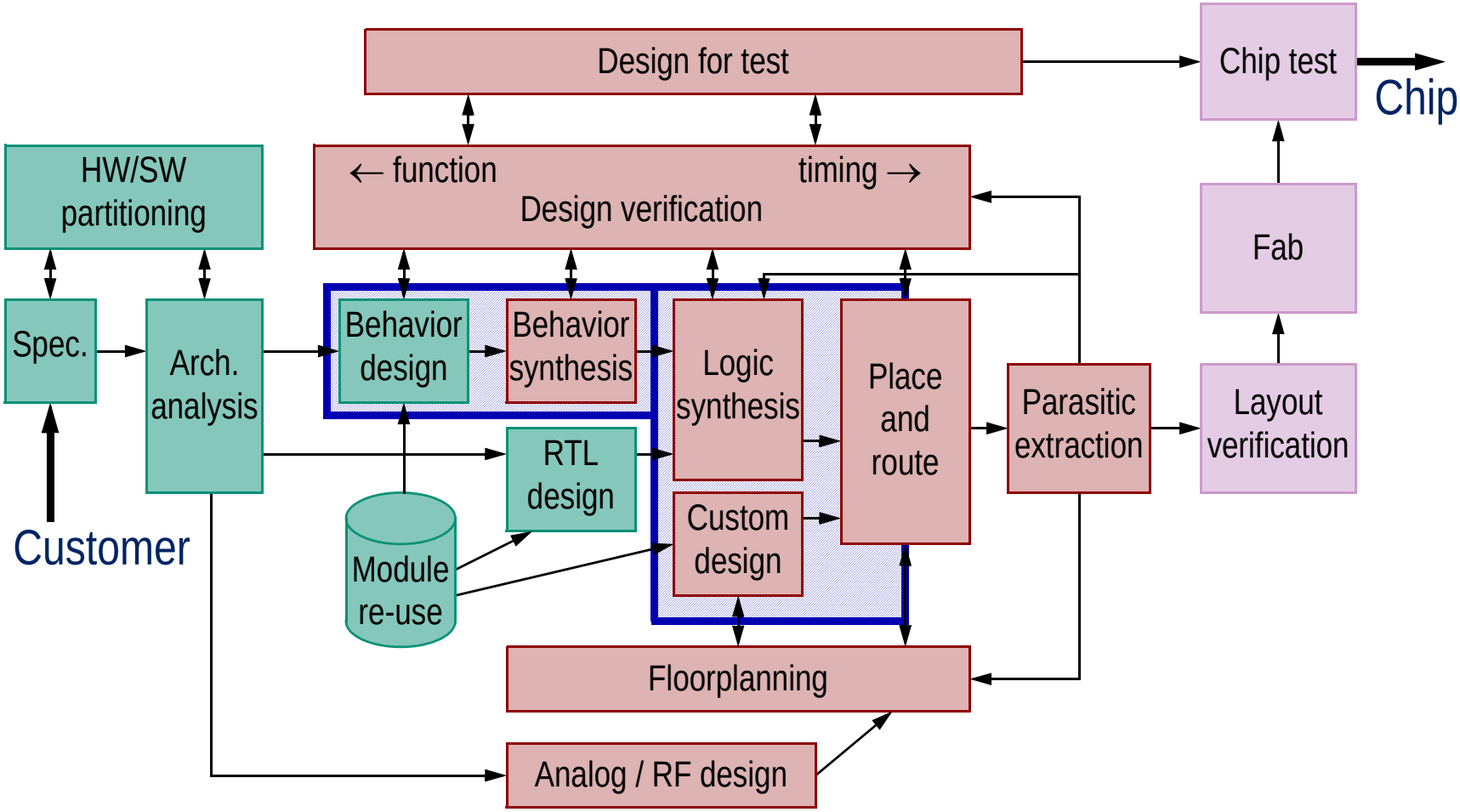
DAT110
Methods for Electronic System Design and Verification

Per Larsson-Edefors
VLSI Research Group

Lecture 4: Synthesis.

Present Scenario: Hardware Synthesis



Synthesis Levels

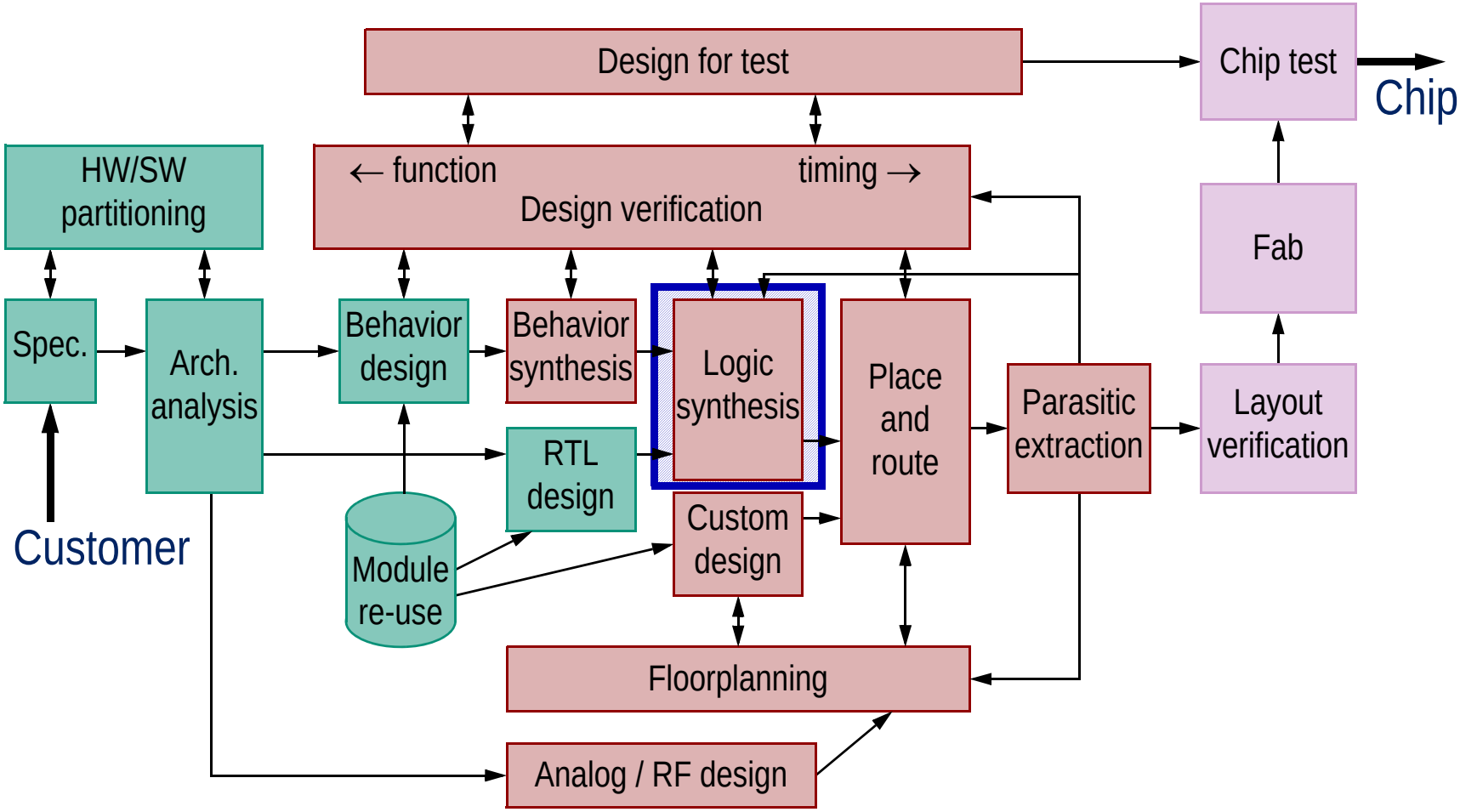
- General design flow.
  1. Behavior (C code or HDL) $\rightarrow$ RTL — High-level synthesis
  2. RTL $\rightarrow$ Physical implementation — Logic synthesis
- Conventional design flow for ASICs.
  1. RTL-specification $\rightarrow$ Generic gate netlist.
  2. Generic gate netlist $\rightarrow$ Cell library.
  3. Cell library $\rightarrow$ Placement & routing.

Topics of Today

- Logic synthesis.
  - Logic minimization.
  - Technology mapping.
- High-level synthesis (HLS).

Logic synthesis

Logic Synthesis



Two Steps inside Logic Synthesis

- Logic minimization starts with RTL descriptions, from which sets of Boolean expressions are extracted.
  - Two-level, multi-level and sequential minimization.
  - Manual methods: Karnaugh maps and Quine-McCluskey tables.
  - Automated methods: ESPRESSO (the first EDA tool for 2-level min.)
- Technology mapping takes a minimized logic expression and maps this to the logic gates of a standard cell library.

The product of logic synthesis is a standard-cell netlist (ASIC) or reconfiguration data for FPGAs etc.

Logic minimization
[*some figures courtesy of Zhou*]

Basic Definitions

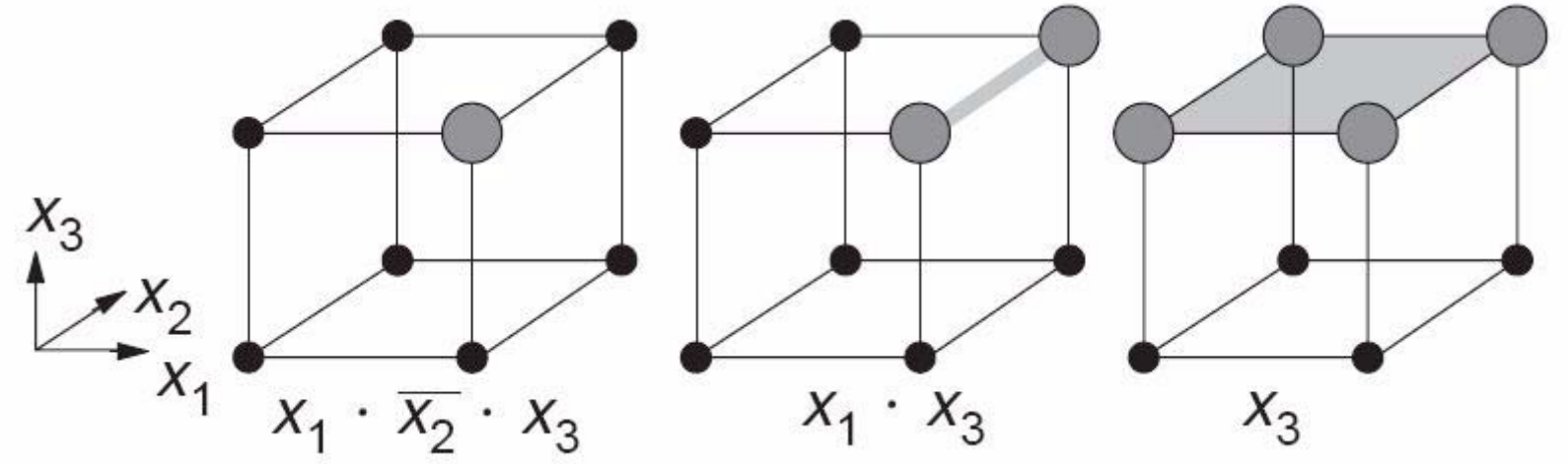
- $B = \{0, 1\}$, $Y = \{0, 1, D\}$, where D is don't care.
- A Boolean function f: $B^m \rightarrow Y^n$.
- m is input count (literals, i.e. variable and inverse variable), n is output count.
- Input variables: $x1$, $x2$, ...
- The value of the output divides B^m into three sets: the ON-set, the OFF-set and the DC-set (= don't care).

Minterms and Cubes

- A minterm is a product of all input variables or their negations.

  A minterm corresponds to a single point in B^n (one truth table row).

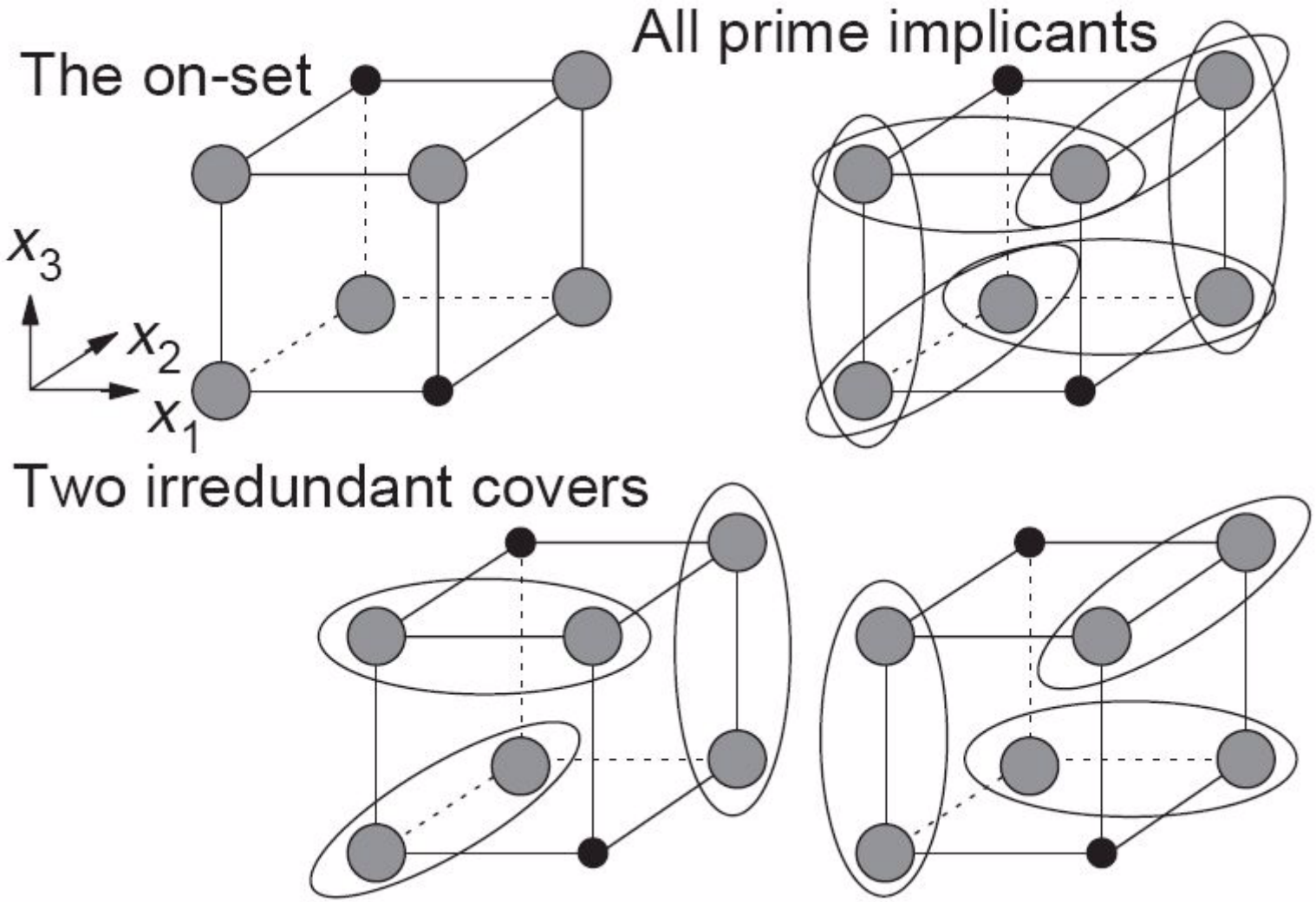
- A cube is a product of the input variables or their negations.
  The fewer the number of variables in the product,
  the bigger the space covered by the cube.



Implicants and Primes

- An implicant is a cube whose points are either in the ON-set or the DC-set.
- A prime implicant is an implicant that is not included in any other implicant.
- A set of prime implicants that together cover all points in the ON-set (and some or all points of the DC-set) is called a prime cover.
- An prime cover is irredundant when none of its prime implicants can be removed from the cover.
- An irredundant prime cover is minimal when the cover has the minimal number of prime implicants.

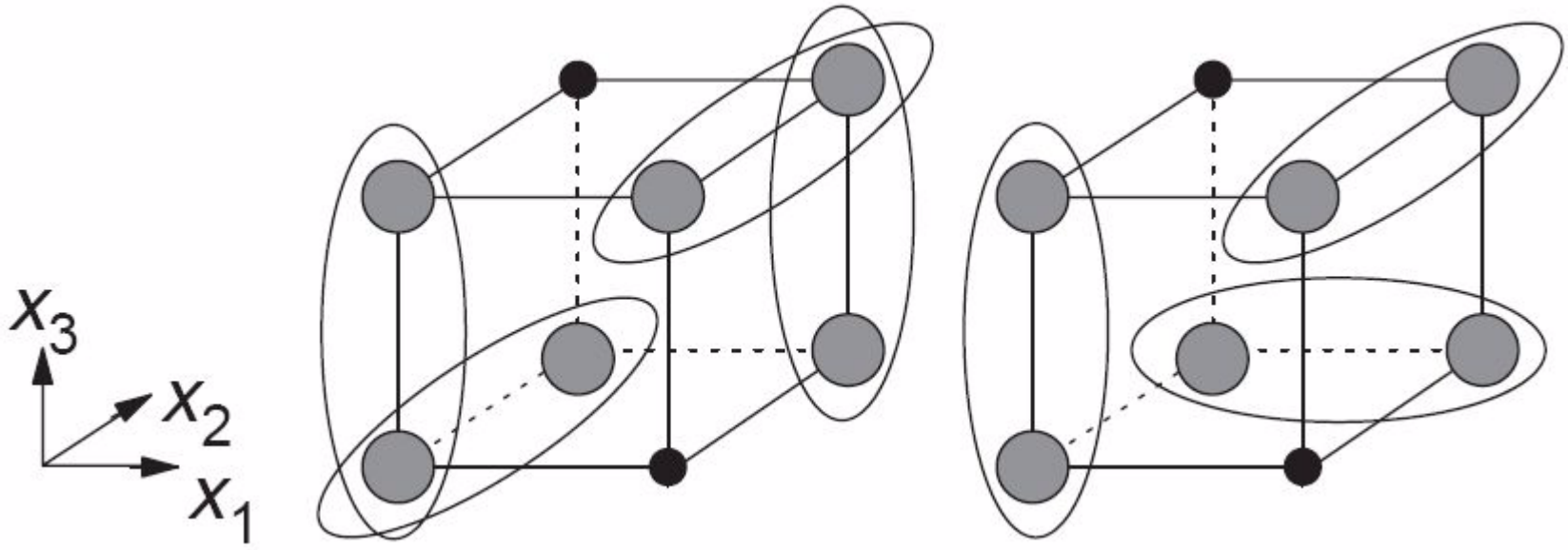
Examples of Covers



Two-Level Logic Minimization

- Popular cost function:
  the number of literals in
  the sum of products expression ("$a\ b + \bar{c}\ d$").
- The goal is to find a minimal irredundant prime cover.

Global or Local Optimum?



$$f1 = \overline{x1}\ \overline{x2} + \overline{x1}\ \overline{x3} + x1\ x2 + x1\ x3$$

$$f2 = \overline{x1}\ \overline{x2} + x1\ x3 + x2\ \overline{x3}$$

Let us Check the Output Result, f1 and f2

$f1 = \overline{x1}\ \overline{x2} + \overline{x1}\ \overline{x3} + x1\ x2 + x1\ x3$

$f2 = \overline{x1}\ \overline{x2} + x1\ x3 + x2\ \overline{x3}$

x1	**x2**	**x3**	**f1**	**f2**
0	0	0	1 1	1 **1**
0	0	1	1	1 **1**
0	1	0	1	1 **1**
0	1	1		
1	0	0		
1	0	1	1	1 **1**
1	1	0	1	1 **1**
1	1	1	1 1	1 **1**

Minimization ...

- Out of

$$f1 = \overline{x1}\ \overline{x2} + \overline{x1}\ \overline{x3} + x1\ x2 + x1\ x3$$
$$f2 = \overline{x1}\ \overline{x2} + x1\ x3 + x2\ \overline{x3}$$

we choose `f2` since it has fewer literals.

Canonical Form

- For the truth table (TT) used, we can create the sum of minterms (a sum-of-products expression):

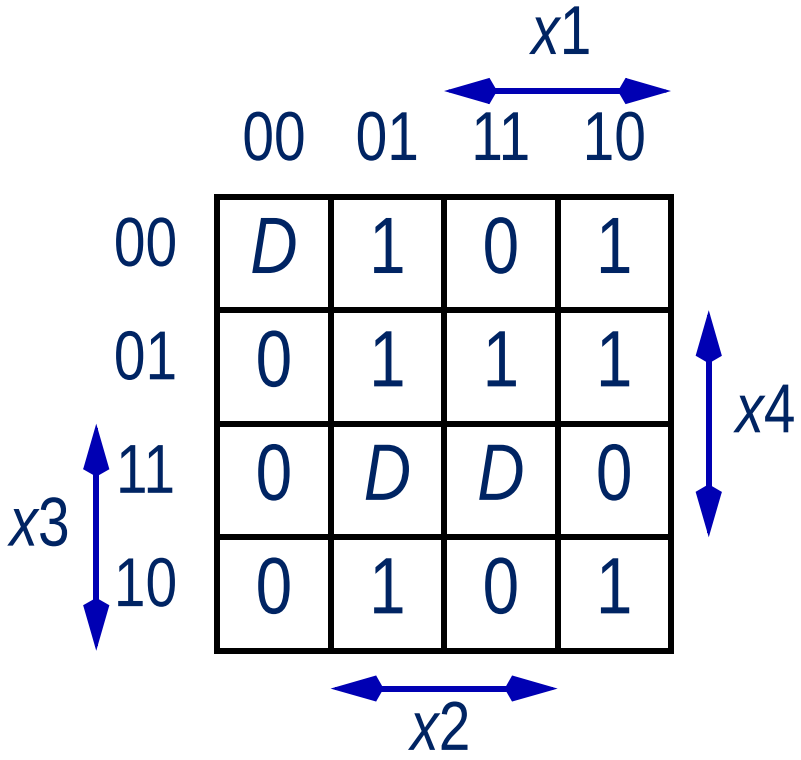
$$\overline{x1}\ \overline{x2}\ \overline{x3} + \overline{x1}\ \overline{x2}\ x3 + \overline{x1}\ x2\ \overline{x3} + x1\ \overline{x2}\ x3 + x1\ x2\ \overline{x3} + x1\ x2\ x3$$

000	**1**
001	**1**
010	**1**
011	**0**
100	**0**
101	**1**
110	**1**
111	**1**

- This expression (and the TT) is on canonical form, since it represents the logic function in a unique way.
- Complex expressions $\Rightarrow$ bad for implementation!
- Uniqueness $\Rightarrow$ useful in e.g. formal verification, since it becomes easy to compare expressions.

KARNAUGH MAPS

- Function $f(x1, x2, x3, x4) = \Sigma m(4, 5, 6, 8, 9, 10, 13) + \Sigma d(0, 7, 15)$ represented in a Karnaugh map (1953):



  - Veitch maps (1952); these did not use Gray coding.

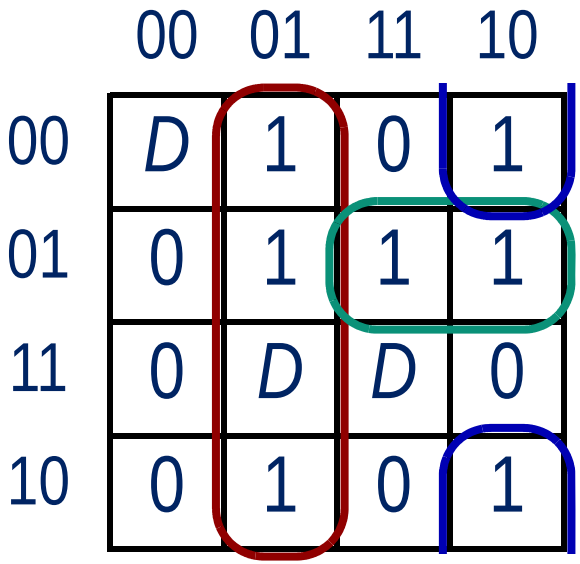
REPRESENTATION

- $\Sigma m(4, 5, 6, 8, 9, 10, 13) + \Sigma d(0, 7, 15)$ means

```
0    0000    D
1    0001    0
2    0010    0
3    0011    0
4    0100    1
5    0101    1
6    0110    1
7    0111    D
8    1000    1
9    1001    1
10   1010    1
...
```

Manual Optimization Using a Karnaugh Map

- Embed all variables 1 or D with as large rings as possible.
  The larger a ring, the fewer the literals in a cube.



- The minimal expression is
  $f = x1\ \overline{x2}\ \overline{x4} + x1\ \overline{x3}\ x4 + \overline{x1}\ x2$.

The Quine-McCluskey Algorithm

- Generation of all prime implicants, followed by extraction of a minimum prime cover.
  1. In phase 1, we select ON-set and DC-set minterm indices, and we group by number of logical ones.
  2. In phase 2, compare the minterms of neighbor groups:
     A 1-bit difference implies adjacency.
     Eliminate the variable representing difference
     (mark it with e.g. '-') and place in next column.
  3. Return to phase 2, and treat the eliminated variables ('-') as variables.

- [Edward McCluskey, "Minimization of Boolean functions," Bell Syst. Tech. J., April 1956]. Based on work by Quine (1952-1955).

Q-M Example - Prime Implicants 1(3)

- We have $f(x1, x2, x3, x4) = \Sigma m(4, 5, 6, 8, 9, 10, 13) + \Sigma d(0, 7, 15)$.

`0000`	(0 - a don't care value)
`0100`	(4 - a logical one)
`1000`	(8 - a logical one)
`0101`	(5 - a logical one)
`0110`	(6 - a logical one)
`1001`	(9 - a logical one)
`1010`	(10 - a logical one)
`0111`	(7 - a don't care value)
`1101`	(13 - a logical one)
`1111`	(15 - a don't care value)

Q-M Example - Prime Implicants 2(3)

0000	0-00 and -000
0100	010- and 01-0
1000	100- and 10-0
0101	01-1 and -101
0110	011-
1001	1-01
1010	(has already been combined with 1000)
0111	-111
1101	11-1
1111	(has already been combined with 0111 and 1101)

Q-M Example - Prime Implicants 3(3)

1.	0-00		no further combinations are possible
2.	-000		no further combinations are possible
3.	010-	01--	combined with row 9
4.	01-0	01--	combined with row 7
5.	100-		no further combinations are possible
6.	10-0		no further combinations are possible
7.	01-1	01--/-1-1	combined with row 4 / 12
8.	-101	-1-1	combined with row 11
9.	011-		combined with row 3
10.	1-01		no further combinations are possible
11.	-111		combined with row 8
12.	11-1	-1-1	combined with row 7

Q-M Example - Minimum_Cover 1(5)

- After all prime implicants have been identified,
  `0-00`, `-000`, `100-`, `10-0`, `1-01`, `01--`, `-1-1`
  which is the minimal cover? (Columns are ON-set rows.)

	4	5	6	8	9	10	13
0-00	X						
-000				X			
100-				X	X		
10-0				X		X	
1-01					X		X
01--	X	X	X				
-1-1		X					X

Q-M Example - Minimum_Cover

- `10-0` and `01--` are special;
  these essential prime implicants are unique for a column and must appear in the minimum cover.

	4	5	6	8	9	10	13
0-00	X						
-000				X			
100-				X	X		
10-0				X		X	
1-01					X		X
01--	X	X	X				
-1-1		X					X

Q-M Example - Minimum_Cover 2(5)

- We will now take away the essential prime implicants ...

	4	5	6	8	9	10	13
0-00	X						
-000				X			
100-				X	X		
10-0				X		X	
1-01					X		X
01--	X	X	X				
-1-1		X					X

Q-M Example - Minimum_Cover 3(5)

- ... and the columns that are covered by these essential prime implicants.

	4	5	6	8	9	10	13
0-00	X						
-000				X			
100-				X	X		
10-0				X		X	
1-01					X		X
01--	X	X	X				
-1-1		X					X

Q-M Example - Minimum_Cover 4(5)

- There isn't all that much left to optimize.

	9	13
0-00		
-000		
100-	X	
1-01	X	X
-1-1		X

- Here we choose `1-01`.

Q-M Example - Minimum_Cover 5(5)

- We have now chosen the following prime implicants: `10-0` and `01--` and `1-01`.
- This corresponds to

$$f = x1\ \overline{x2}\ \overline{x4} + \overline{x1}\ x2 + x1\ \overline{x3}\ x4,$$

which can be compared to the result of the Karnaugh map

$$f = \mathbf{x1}\ \overline{\mathbf{x2}}\ \overline{\mathbf{x4}} + \mathbf{x1}\ \overline{\mathbf{x3}}\ \mathbf{x4} + \overline{\mathbf{x1}}\ \mathbf{x2}.$$

Exact Solutions Are Hard to Compute

- In Quine-McCluskey, the number of prime implicants grows rapidly with the number of inputs: $G = p_1 + p_2 + ... + p_\alpha$,

  where $\alpha = 3^n / n$ [§] and n is the number of inputs:

  1. Generate cover of all primes.
  2. Make G irredundant (in optimum way). Q-M gives an exact minimum.

- ESPRESSO attempts an approximative solution - a heuristic:

  1. Don't generate all prime implicants (as in Q-M phase 1).
  2. Select a subset of primes that still covers the ON-set (similar to Karnaugh maps).

[§] [Hong et al., "MINI: A heuristic approach for logic minimization," IBM J. Res. Dev., 1974]

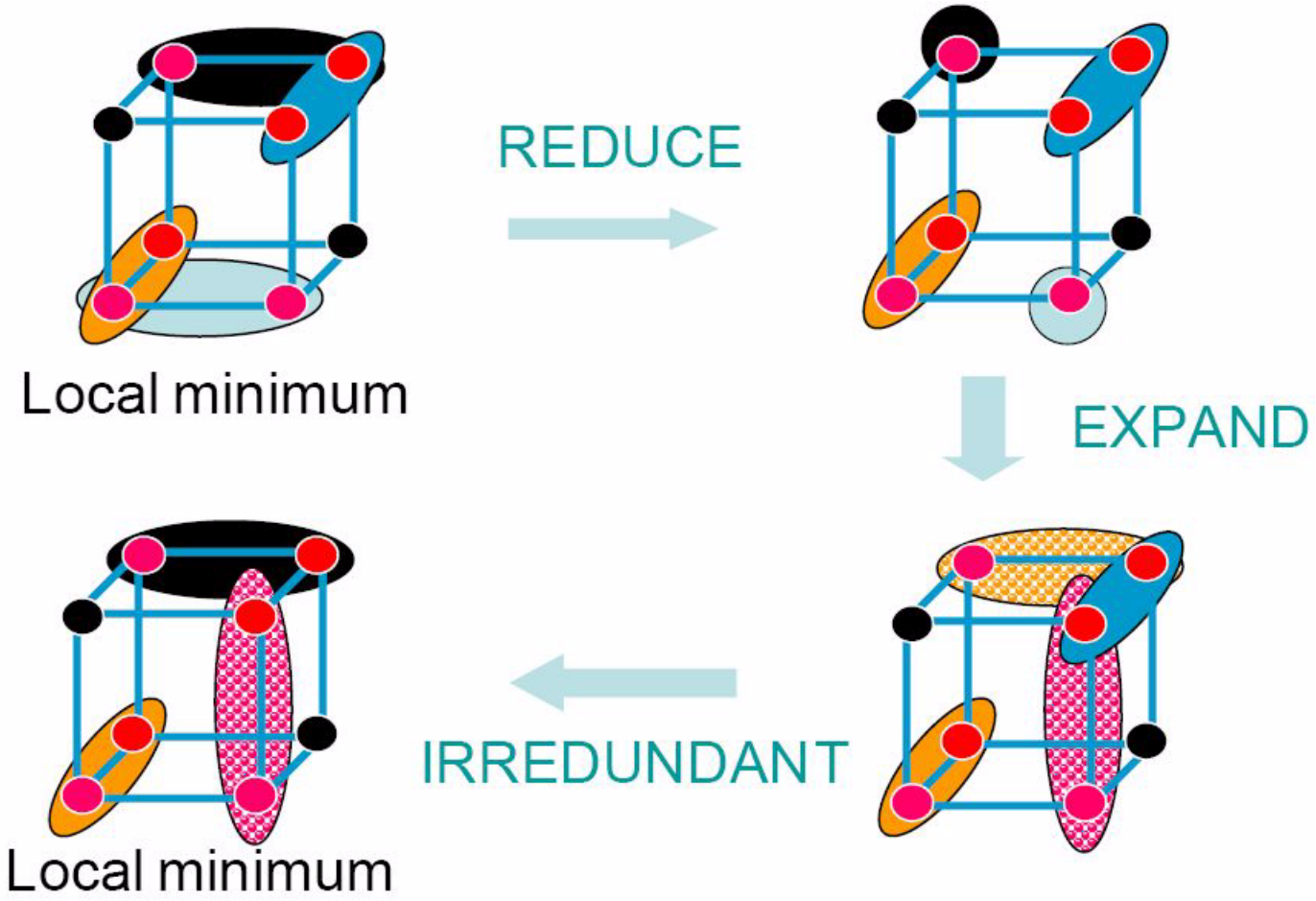
ESPRESSO Core 1(2)

```
Procedure ESPRESSO (F, D, R)  /*F - ON set, D - don't care, R - OFF*/
    R = COMPLEMENT(F+D);      /* Compute complement */
    F = EXPAND(F, R) ;        /* Initial expansion */
    F = IRREDUNDANT(F,D);     /* Initial irredundant cover */
    F = ESSENTIAL(F,D)        /* Detecting essential primes */
    F = F - E;                /* Remove essential primes from F */
    D = D + E;                /* Add  essential primes to D */
```

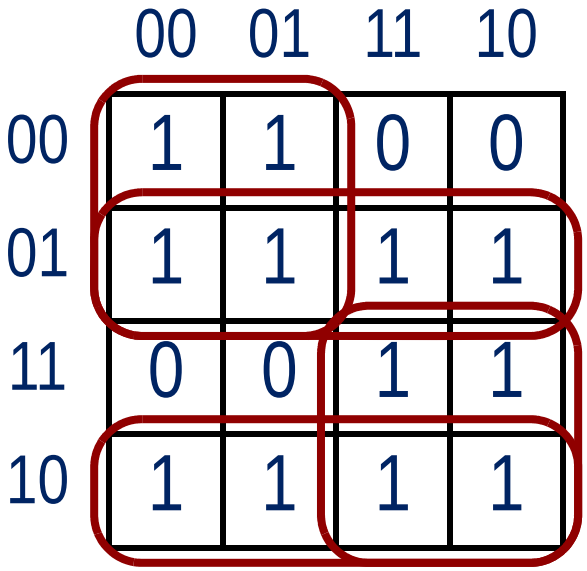
ESPRESSO Core 2(2)

```
WHILE Cost(F) keeps decreasing DO
    F = REDUCE(F,D);            /* Perform reduction */
    F = EXPAND(F,R);            /* Perform expansion */
    F = IRREDUNDANT(F,D);       /* Perform irredundant cover */
ENDWHILE;
F = F + E;
RETURN F;
END Procedure;
```

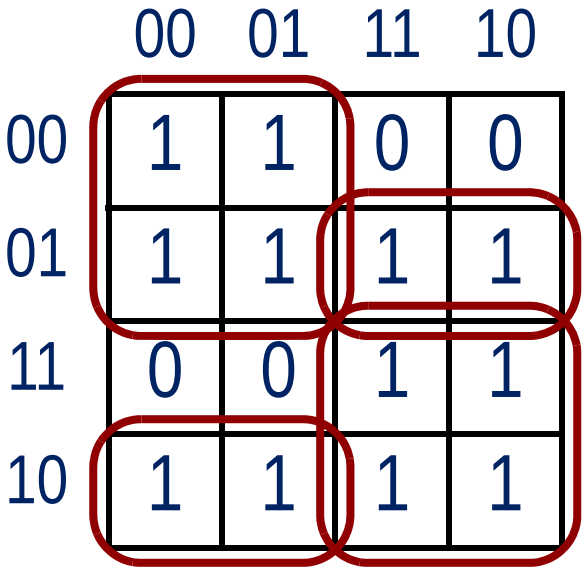
ESPRESSO PHASES 1(3)



ESPRESSO PHASES 2(3)

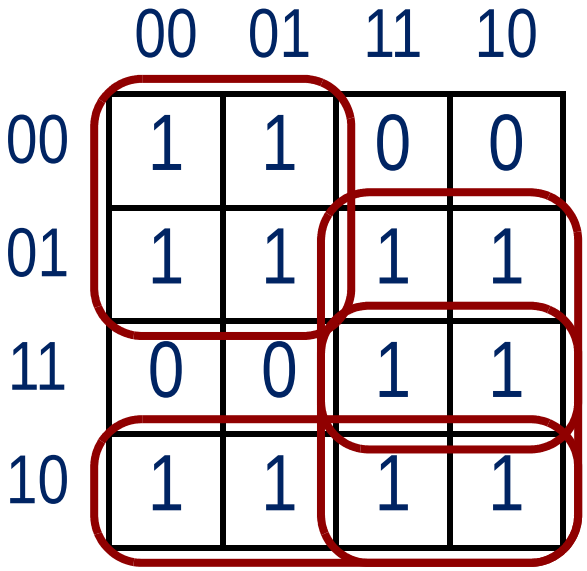


Initial set

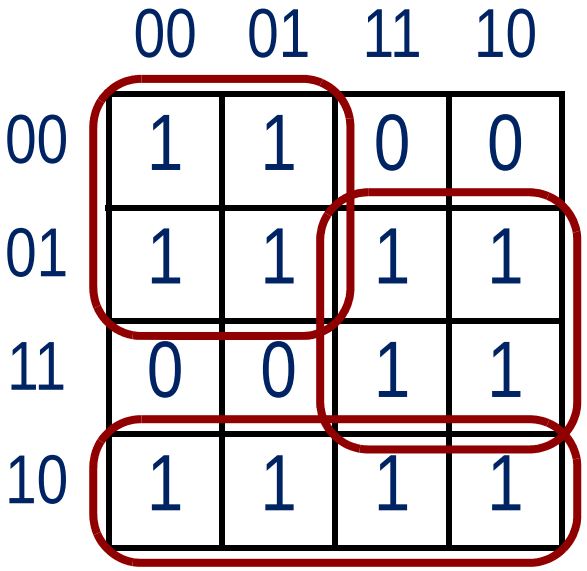


“Reduce” implicants

ESPRESSO PHASES 3(3)



"Expand" implicants



"Irredundant" cover

Early EDA Tools

- MINI from IBM 1974.
- ESPRESSO-I 1981.
  - Robert Brayton et al., "A comparison of logic minimization strategies using ESPRESSO", ISCAS'82.
- ESPRESSO-II developed in 1982 was the first widespread two-level logic minimization EDA tool.
- Read more in Brayton et al.
  [Logic Minimization Algorithms for VLSI Synthesis]
  from Kluwer 1984. (Available from CHANS / SpringerLink.)

Logic Minimization: Conclusion

- All considerations so far are for two-level minimization.
- But multi-level logic minimization is required for efficient solutions.
  - $abe\bar{g} + abfg + a\bar{b}eg + ace\bar{g} + acfg + ac\bar{e}g + de\bar{g} + dfg + d\bar{e}g =$ $(a(b + c) + d)(e\bar{g} + g(f + \bar{e}))$ [see example in coming slides...].
  - Read more [Sec. 6.3.3 of Ch6_LogicSynthesis.pdf]
- Since ESPRESSO was introduced, other, more efficient techniques have been developed: Binary Decision Diagram (BDD) representations.

Technology mapping
[*inset courtesy of Robert Brayton, Berkeley*]

Technology Mapping

Example:

$t_1 = a + bc;$

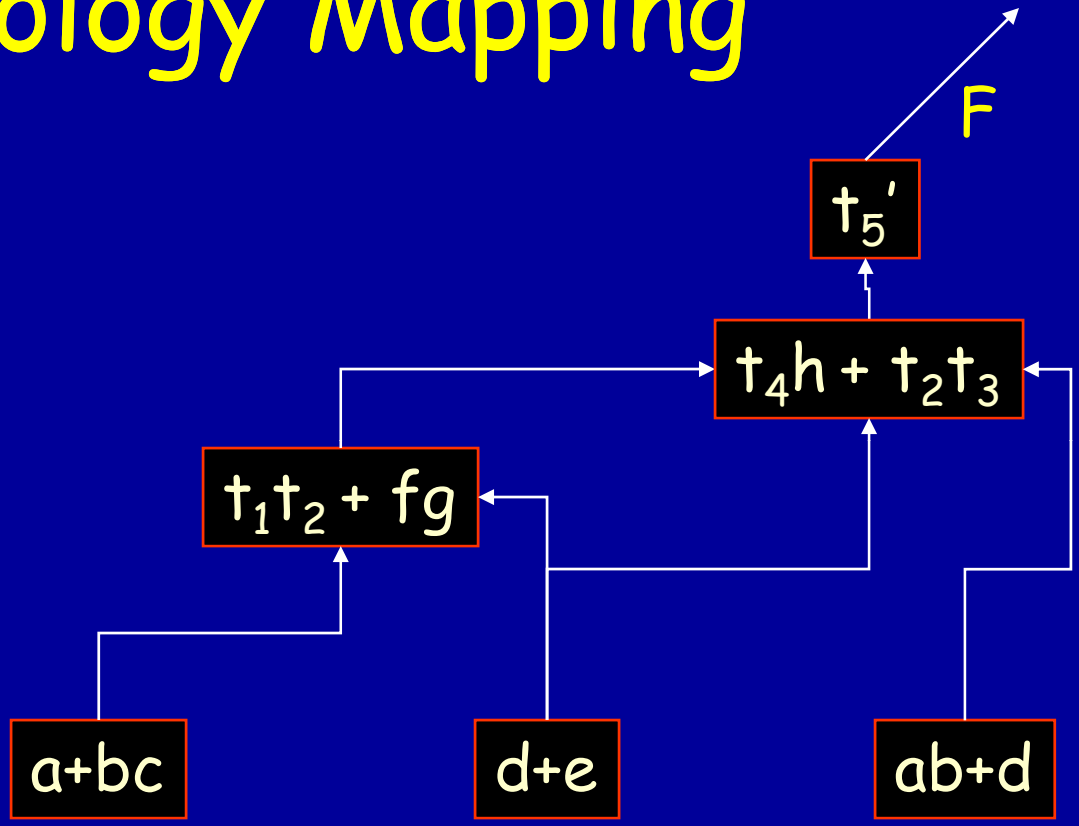
$t_2 = d + e;$

$t_3 = ab + d;$

$t_4 = t_1t_2 + fg;$

$t_5 = t_4h + t_2t_3;$

$F = t_5';$



This shows an unoptimized set of logic equations consisting of 16 literals

Optimized Equations

Using technology independent optimization, these equations are optimized using only 14 literals:

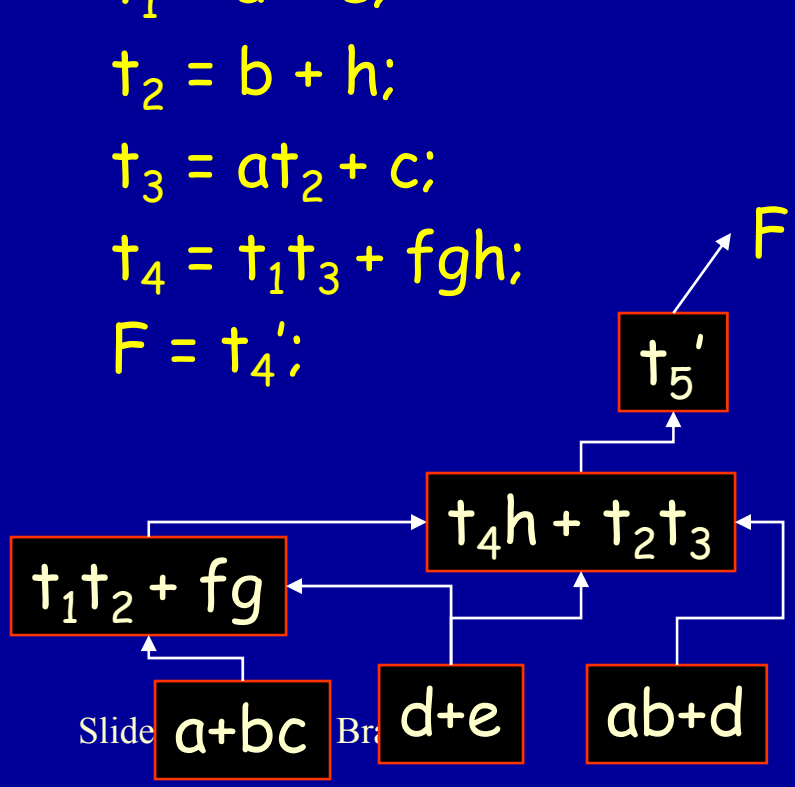
$t_1 = d + e;$

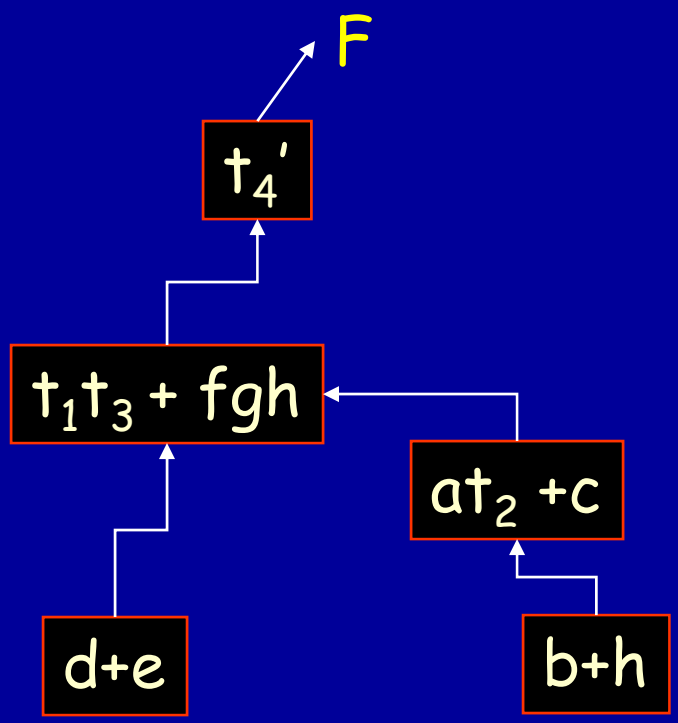
$t_2 = b + h;$

$t_3 = at_2 + c;$

$t_4 = t_1t_3 + fgh;$

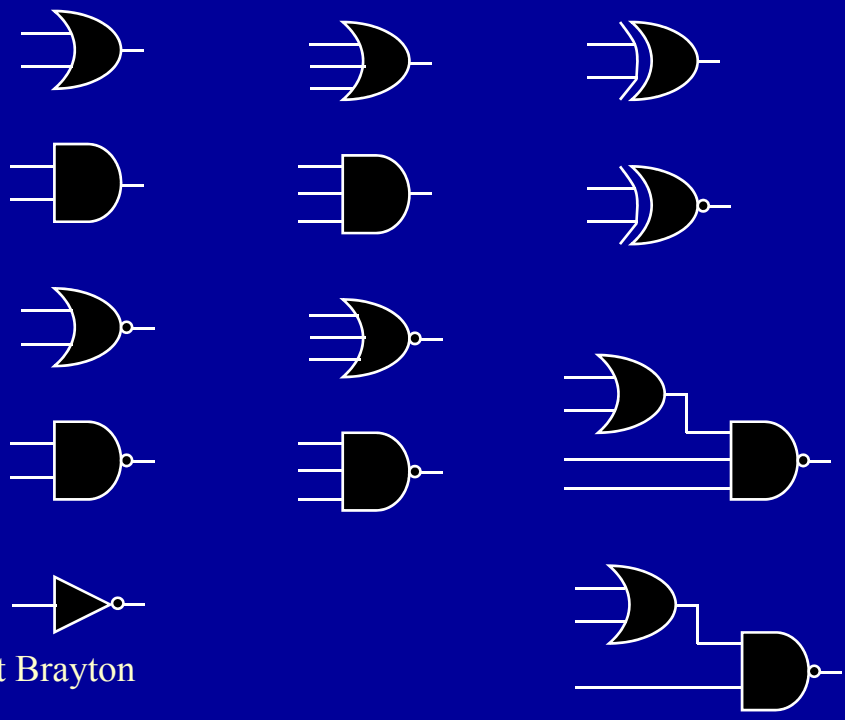
$F = t_4';$



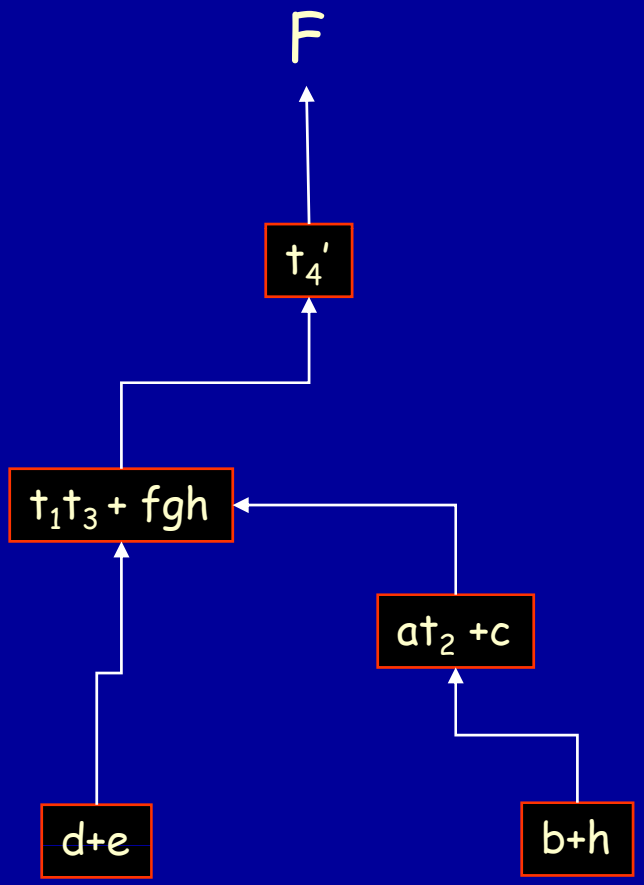


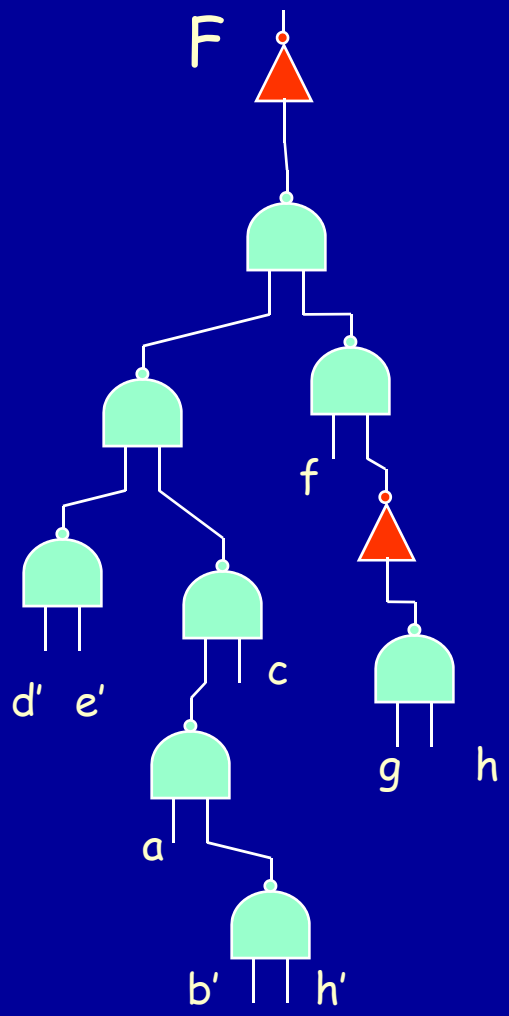
Optimized Equations

Implement this network using a set of gates which form a *library*. Each gate has a cost (i.e. its area, delay, etc.)



Subject graph





Subject graph of 2-input NANDs and invertors

Slides by Robert Brayton

Algorithmic Approach

A *cover* is a collection of pattern graphs such that

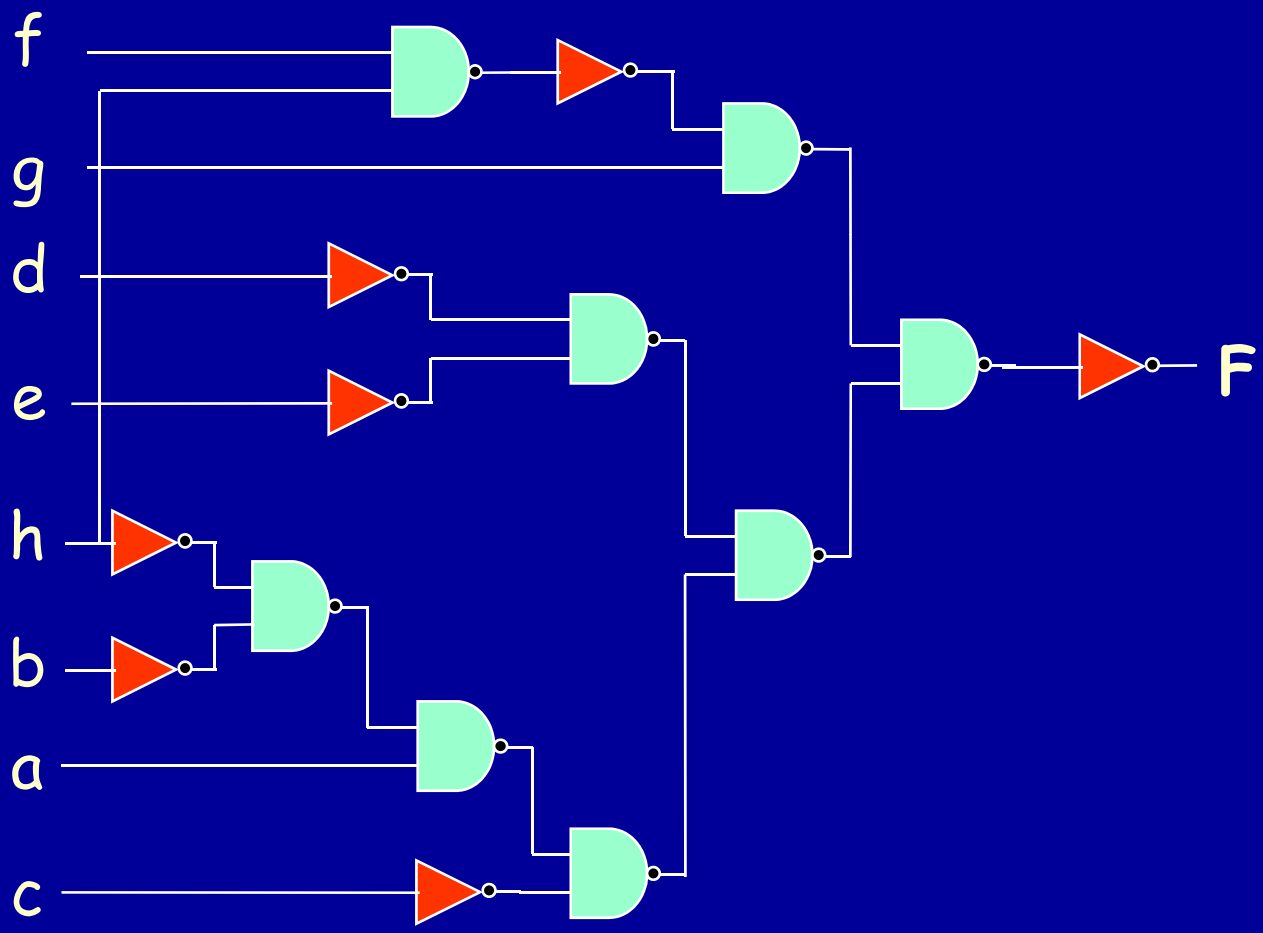
1. every node of the subject graph is contained in one (or more) pattern graphs
2. each input required by a pattern graph is actually an output of some other graph (i.e. the inputs of one gate must exists as outputs of other gates.)

For minimum area, the cost of the cover is the sum of the areas of the gates in the cover.

Technology mapping problem: *Find a minimum cost covering of the subject graph by choosing from the collection of pattern graphs for all the gates in the library.*

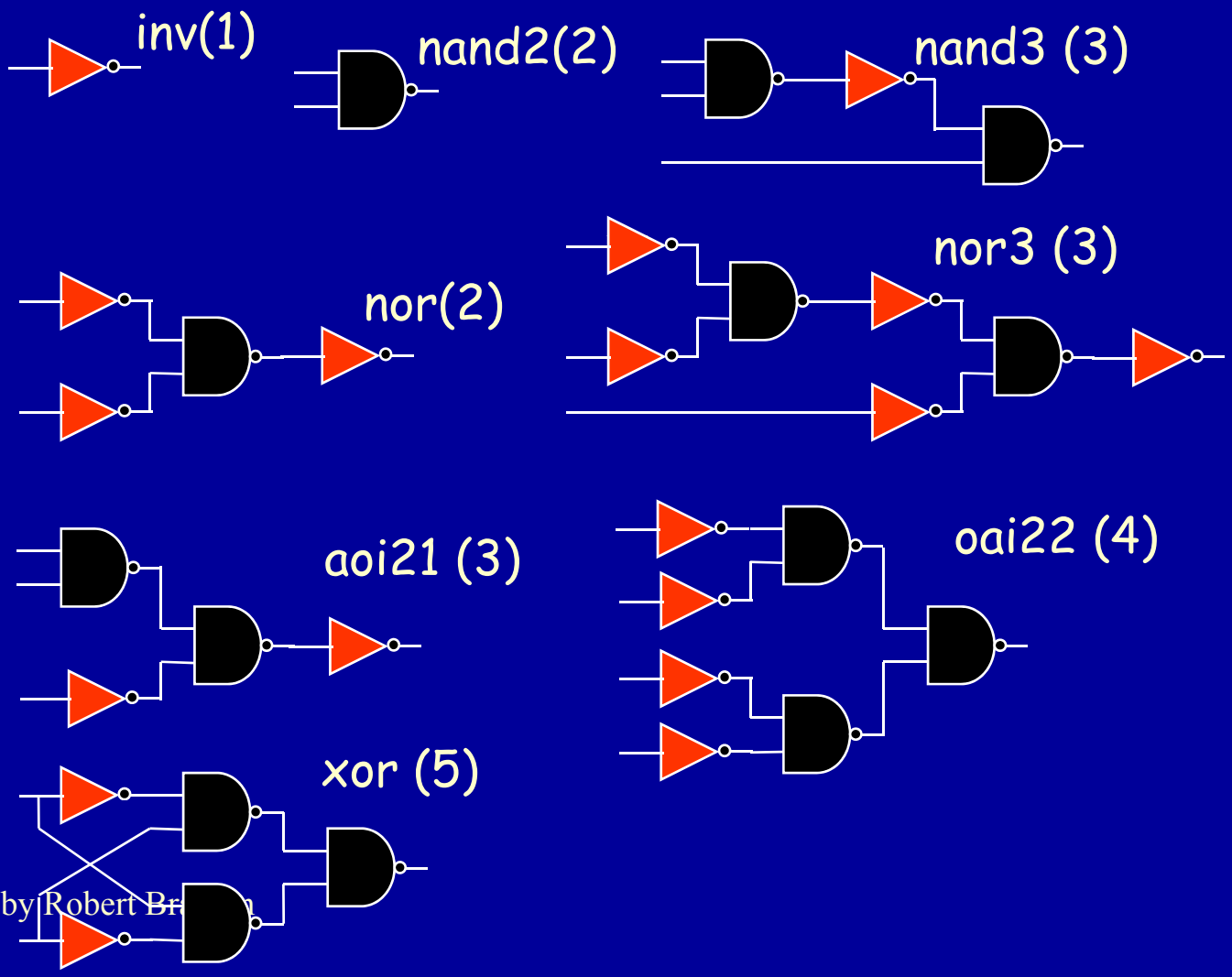
Subject Graph

$t_1 = d + e;$
$t_2 = b + h;$
$t_3 = at_2 + c;$
$t_4 = t_1t_3 + fgh;$
$F = t_4';$



Slides by Robert Brayton

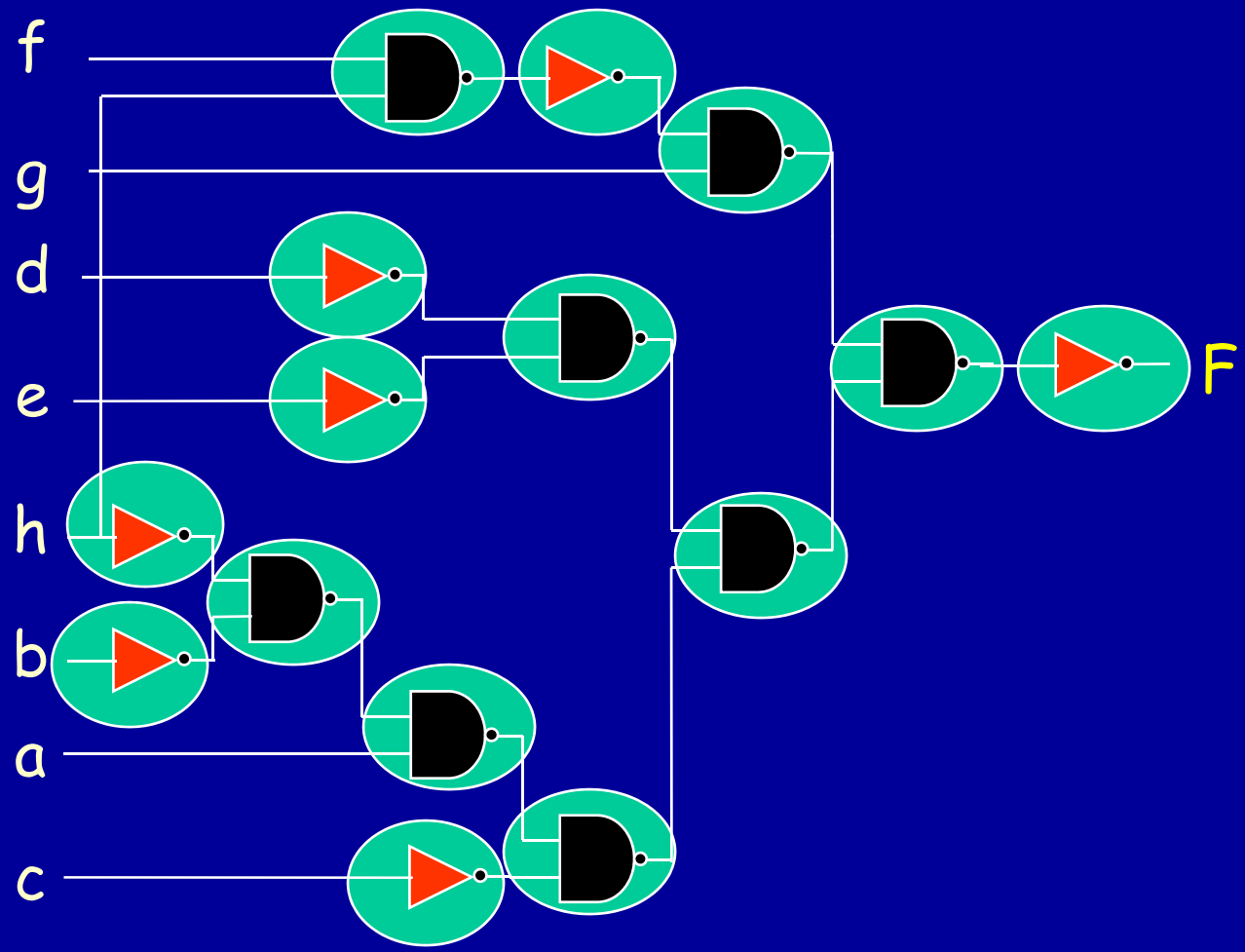
Pattern Graphs for the IWLS Library



Subject graph covering

$t_1 = d + e;$
$t_2 = b + h;$
$t_3 = at_2 + c;$
$t_4 = t_1t_3 + fgh;$
$F = t_4';$

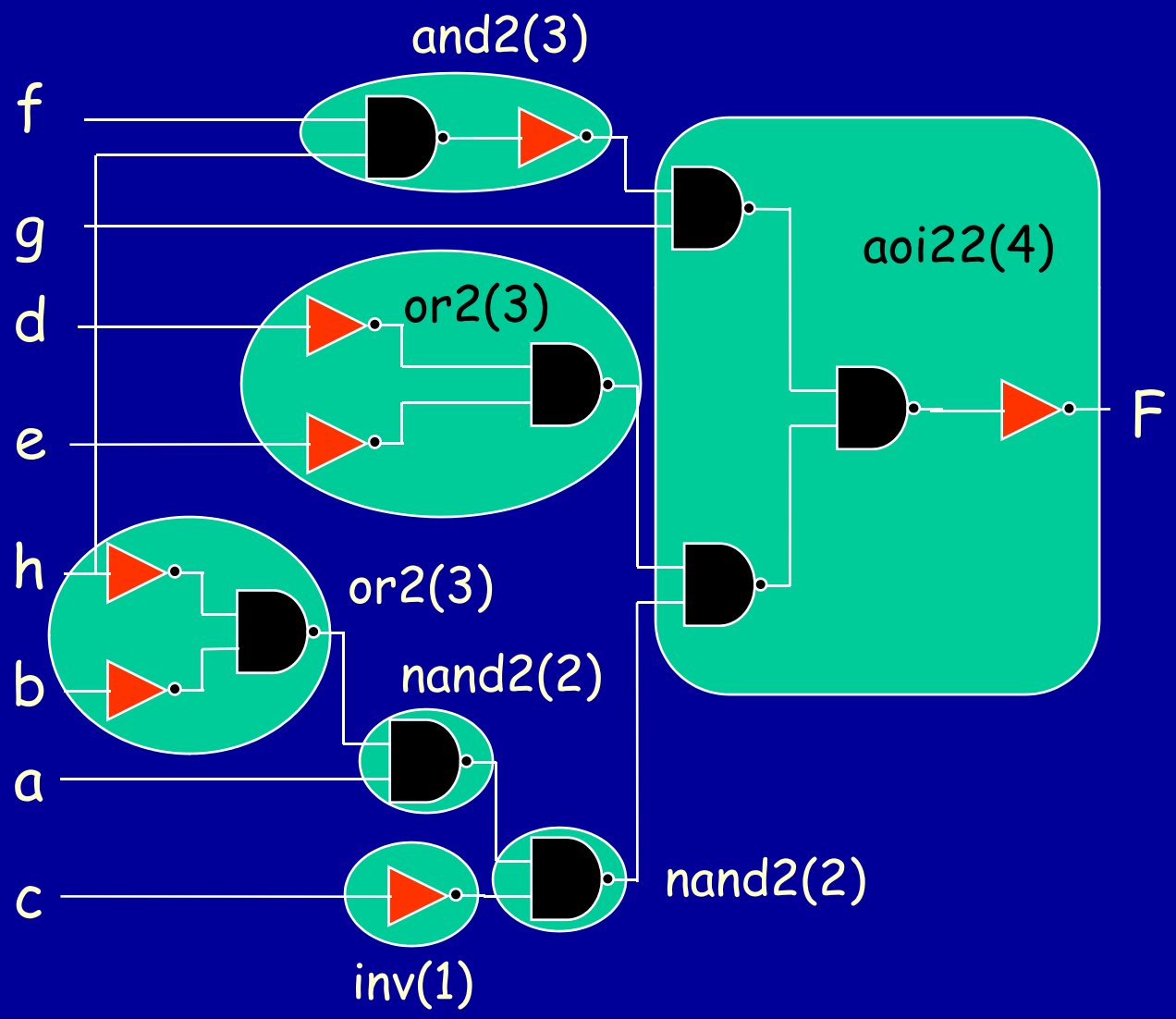
Total cost = 23



Slides by Robert Brayton

Better Covering

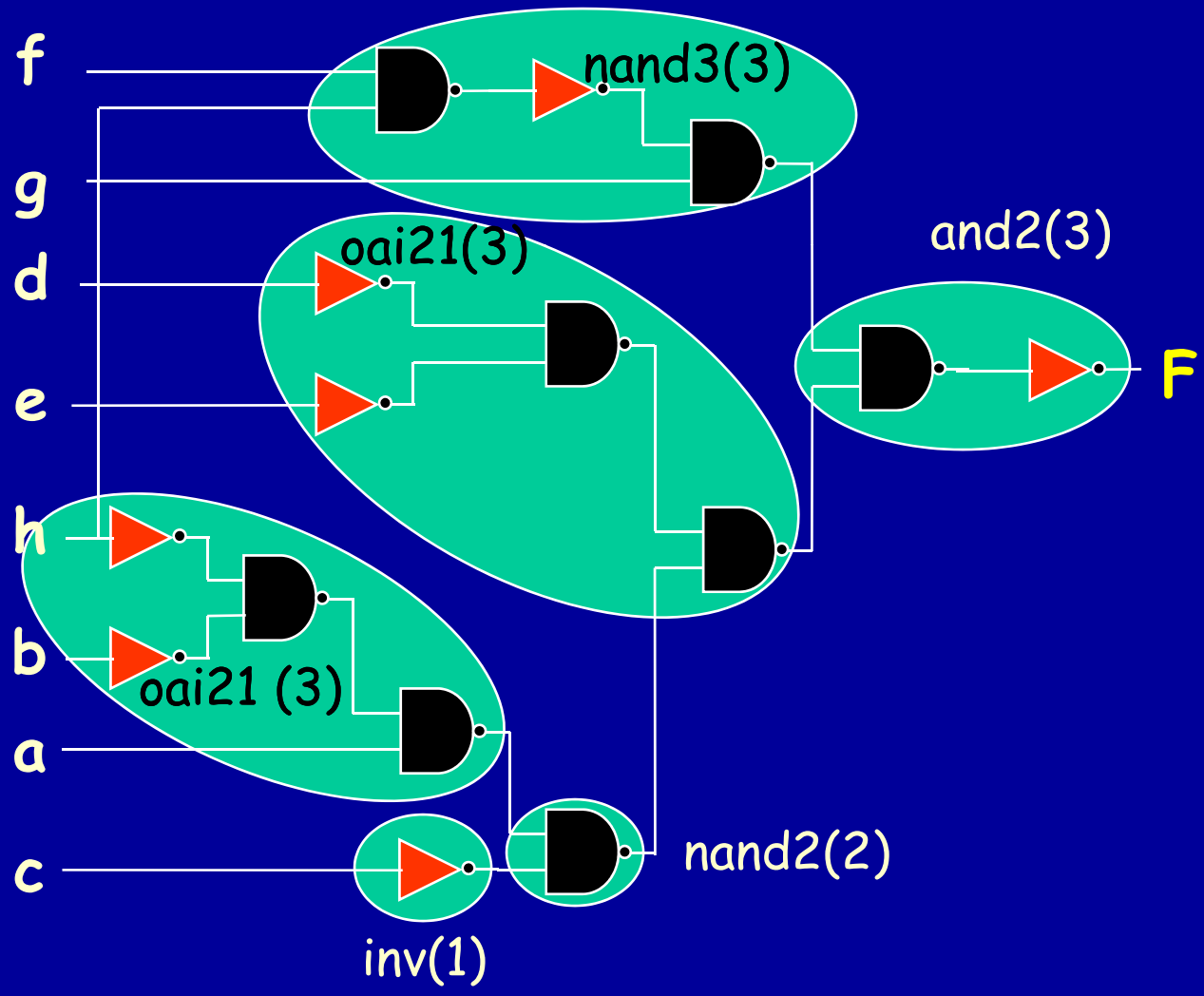
$t_1 = d + e;$
$t_2 = b + h;$
$t_3 = at_2 + c;$
$t_4 = t_1t_3 + fgh;$
$F = t_4';$



Total area = 19

Alternate Covering

$t_1 = d + e;$
$t_2 = b + h;$
$t_3 = at_2 + c;$
$t_4 = t_1t_3 + fgh;$
$F = t_4';$



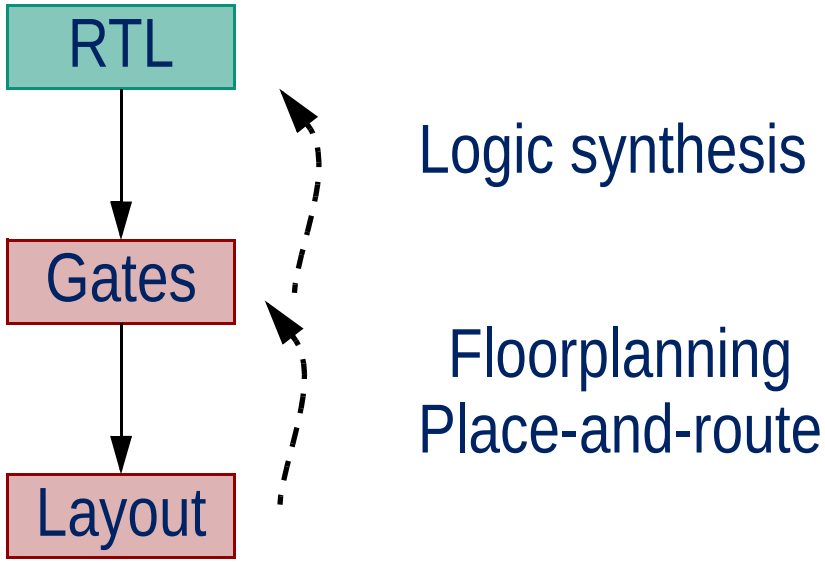
Total area = 15

Technology Mapping

- Graph covering (algorithmic) approaches.
  - Inspired by code generation in compilers (K. Keutzer, DAGON,1987).
  - Dynamic programming provides optimization with linear complexity, however, partitioning into forest of trees makes optimization local.
  - Choice of base functions in subject graph is critical.
- Rule-based approaches iteratively perform local transformations.
  - Can merge technology-independent and -dependent phases.
- Technology mapping for remapping one design to a different process technology: design migration.
- Read more [Sec. 6.3.4 of Ch6_LogicSynthesis.pdf].

Logic Synthesis: Conclusion

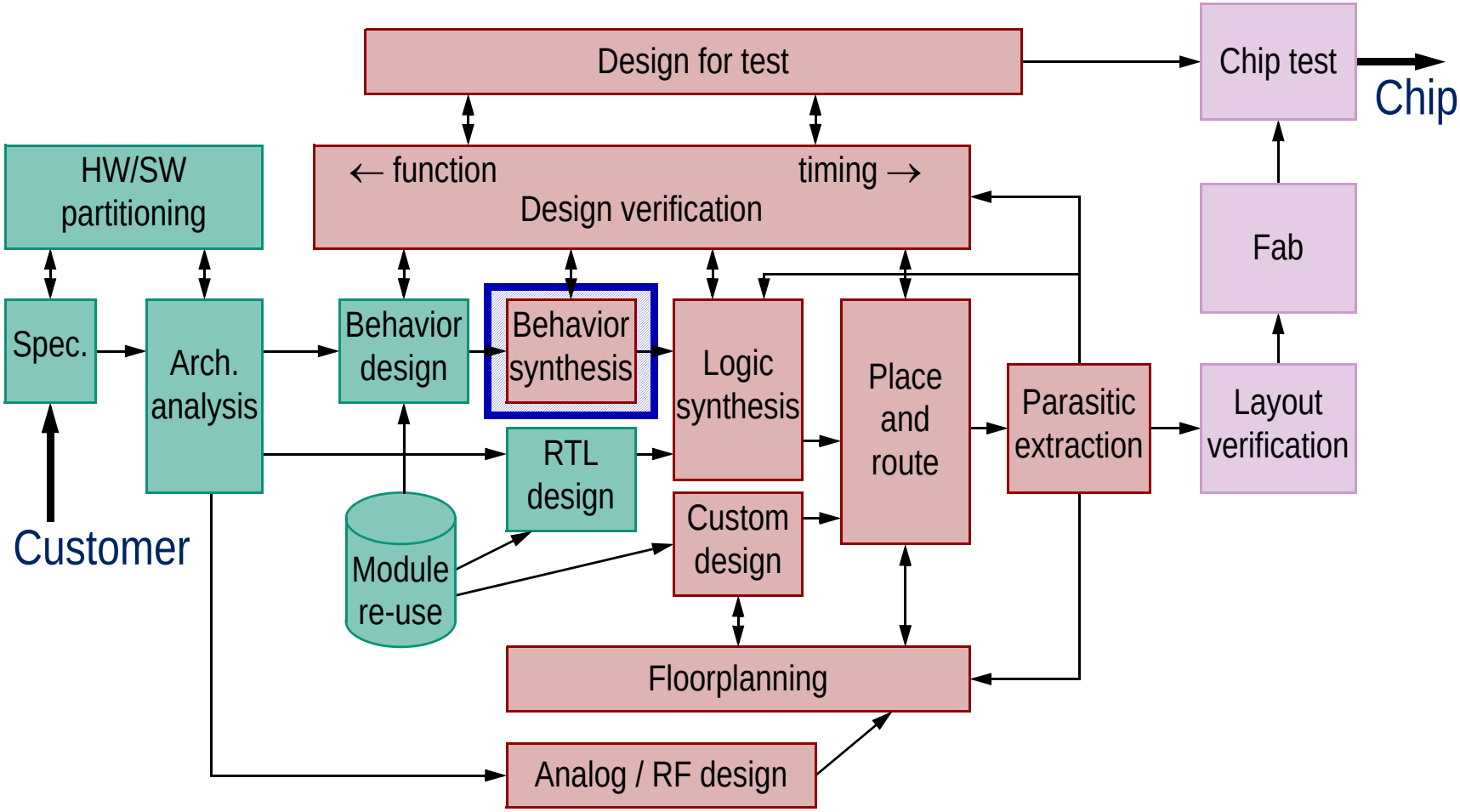
- Multi-level logic minimization followed by technology mapping are used in present-day systems.
- Technology scaling $\Rightarrow$ wires become more important for each tech generation: Gate-based synthesis is not sufficient.



- Physically knowledgeble synthesis is needed to avoid problems with timing closure; this would entail either better models of physical hardware earlier or use of rapid prototyping (quick and dirty “optimization” that resembles true optimization).

High-level synthesis
[*some text courtesy of Gerez*]

HIGH-LEVEL SYNTHESIS



High-Level Synthesis (HLS)

- Logic synthesis ...
  - starts from a register-transfer level (RTL) description; where circuit behavior in each clock cycle is established.
  - uses logic minimization/tech mapping techniques to optimize the design.
  - generates a standard-cell netlist.
- High-level (architectural/behavioral) synthesis on the other hand ...
  - starts from an abstract behavioral description (for example C code).
  - generates an RTL description.
  - HLS is still far less widespread than logic synthesis.

Strict Mapping Targets

- Efficient design exploration would require flexible hardware targets, to enable optimal implementations. This is a challenge!
- The mapping of behavioral code to RTL is done, assuming a certain number of RTL building blocks (the architectural template), for example:
  - functional units.
  - registers.
  - multiplexers.
  - buses.
  - tri-state bus drivers.

Design Parameters for Hardware Model

- Clocking strategy: single or multiple phase clocks.
- Interconnect: allowing or disallowing buses.
- Clocking of functional units: allowing or disallowing ...
  - multicycle operations.
  - chaining.
  - pipelined units.

Control and Datapath Synthesis

- Hardware is normally partitioned ...
  - datapath (arithmetic):
    a network of functional units, registers, multiplexers and buses.
    The actual “computation” takes place in the datapath.
  - control (FSM and memory):
    ensures data are present at the right place at
    a specific time (memory handling),
    presents the right instructions to a programmable unit, etc.
- Traditionally high-level synthesis has concentrated on datapath synthesis.

HLS Tools

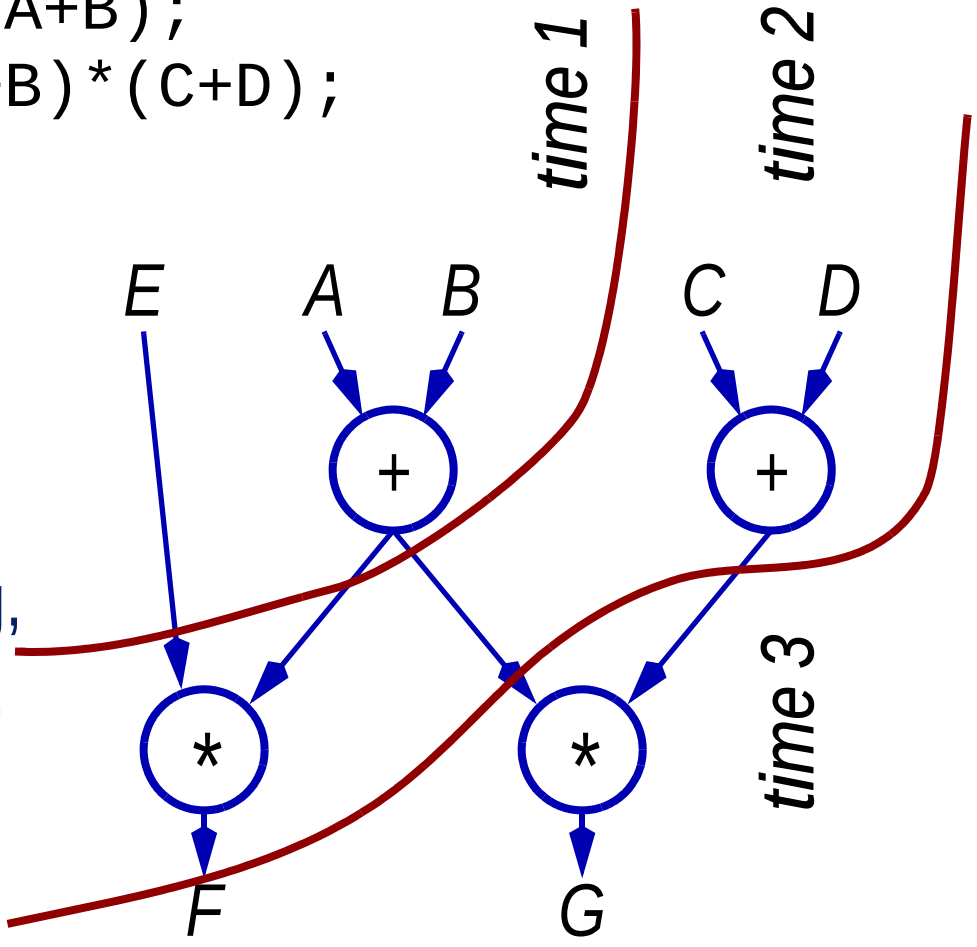
- Mentor/Siemens:
  - Catapult C represents general-purpose HLS.
- Cadence:
  - Stratus includes Forte Cynthesizer data-centric HLS and C-to-Silicon Compiler general-purpose HLS.
- Synopsys:
  - Synphony C Compiler (previously PICO from Synfora) represents general-purpose HLS.
  - SPW represents data-centric HLS.

HLS Flow

- Input: Either a conventional programming language, such as C, or an HDL (better at expressing HW parallelism).
- The description has to be parsed and transformed into an internal representation; here conventional compiler techniques can be used.
- As internal representation, a data-flow graph (DFG) may be suitable (hence no explicit information on control flow). In a DFG there are ...
  - vertices (nodes) that represent computations/operations.
  - edges that represent precedence relations.

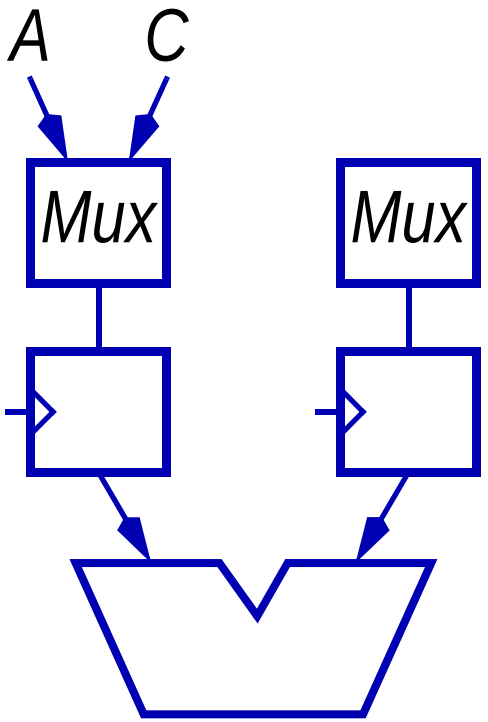
Computation, DFG and RTL

```
F = E*(A+B);
G = (A+B)*(C+D);
```



Limited resources ⇒ scheduling, allocation, binding.

Only datapath here, muxes need control signals of course.



- Read more [Ch5_ESL_and_HLS.pdf].

Synthesis: Conclusion

- High-level synthesis:
  Behavior (C code or HDL) $\rightarrow$ RTL blocks visible.
- Logic synthesis:
  RTL code of blocks $\rightarrow$ Physical implementation.
  - Logic minimization.
  - Technology mapping.