Lecture 3: Logic Synthesis

ECE201A

Some notes adopted from
Andrew B. Kahng
Lei He
Igor Markov
Mani Srivastava
Synopsys
Various Sources

Logistics

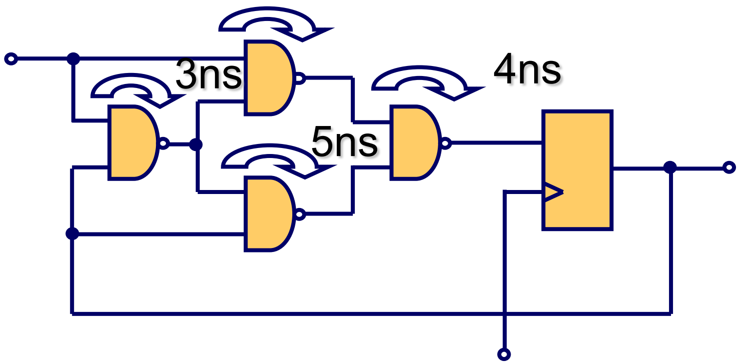
- Lab 1 has been posted on Piazza.
  - OpenAccess (as any design database) can be a struggle for first timers. Give yourself enough time in Lab 1 to get the hang of it
  - Learn to use the OA documentation
    - Classes , derived classes, methods…
  - Second half of today's lecture: OA tutorial
- Today's lecture: logic synthesis

RTL vs. Gate Level

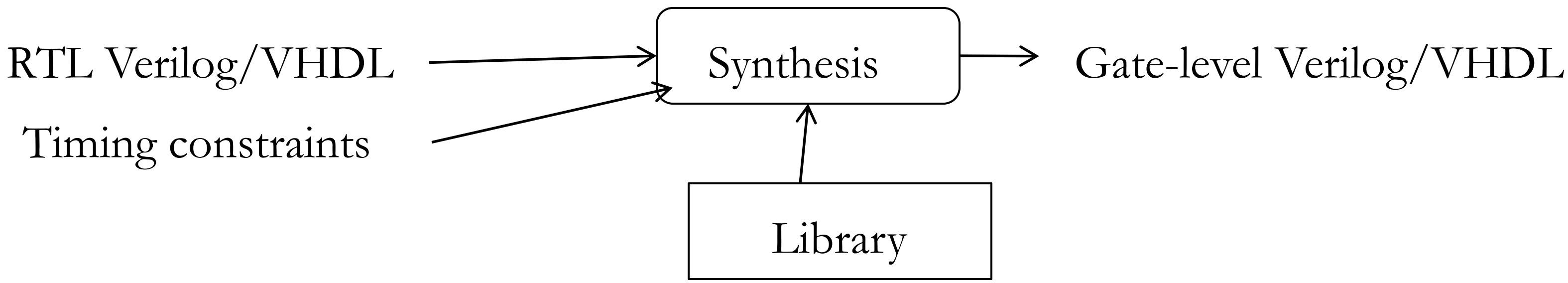
- RTL: Cycle accurate model "close" to the hardware implementation
  - bit-vector data types and operations as abstraction from bit-level implementation
  - sequential constructs (e.g. if - then - else, while loops) to support modeling of complex control flow
- Gate-level: Model as finite-state machine
  - models function in Boolean logic using registers and gates
  - various delay models for gates and wires

```
module mark1;
reg [31:0] m[0:8192];
reg [12:0] pc;
reg [31:0] acc;
reg[15:0] ir;

always
   begin
     ir = m[pc];
     if(ir[15:13] == 3b'000)
        pc = m[ir[12:0]];
     else if (ir[15:13] == 3'b010)
        acc = -m[ir[12:0]];
     ...
   end
endmodule
```



What is Logic Synthesis ?



- How ?
  - Instantiation of primitives (e.g., AND)
  - Macro substitution (e.g., if-else → MUX)
  - Inference (e.g., variable declare → memory)
  - Logic minimization
  - Logic restructuring (e.g., retiming)
  - Miscellaneous: buffer insertion, sizing, pin-swapping, etc
  - Physical input: wireload models, placement-driven synthesis
- Objectives
  - Timing, area, power

Typical Synthesis Scenario

RTL to Network Transformation

- read HDL
- control/data flow analysis
- “elaborate” design (loop unrolling, initialize..)
- “translate/analyze” design (extract FSM, resource sharing..)

↓

Technology independent Optimizations

- basic logic restructuring and minimization
- crude measures for goals

↓

Technology Mapping

- use logic gates from target cell library

↓

Technology Dependent Optimizations

- timing optimization
- physically driven optimizations

High Level Optimization 1: Resource Sharing

- Two operations can be shared only if no execution path that reaches both operations exists from the start of the block to the end of the block
  - A+B/C+D cant be shared

Z1 <= A+B;

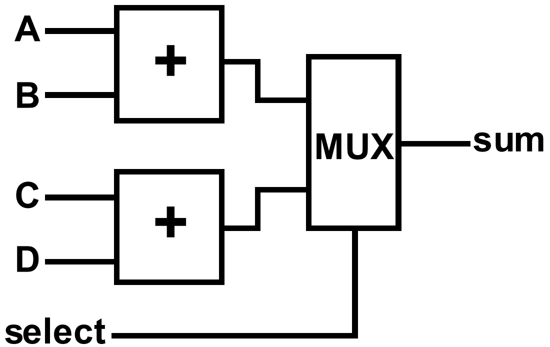
If(COND)

Z2 <=C+D

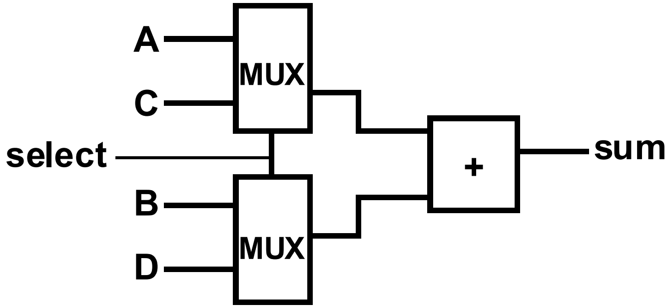
Given the following HDL description:

```
if (select)
    sum <= A + B;
else
    sum <= C + D;
```

One possible implementation:



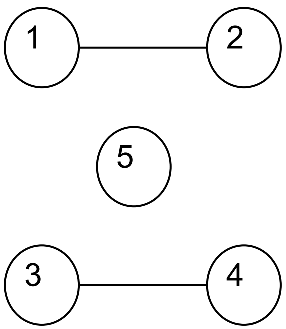
Another, more efficient implementation.



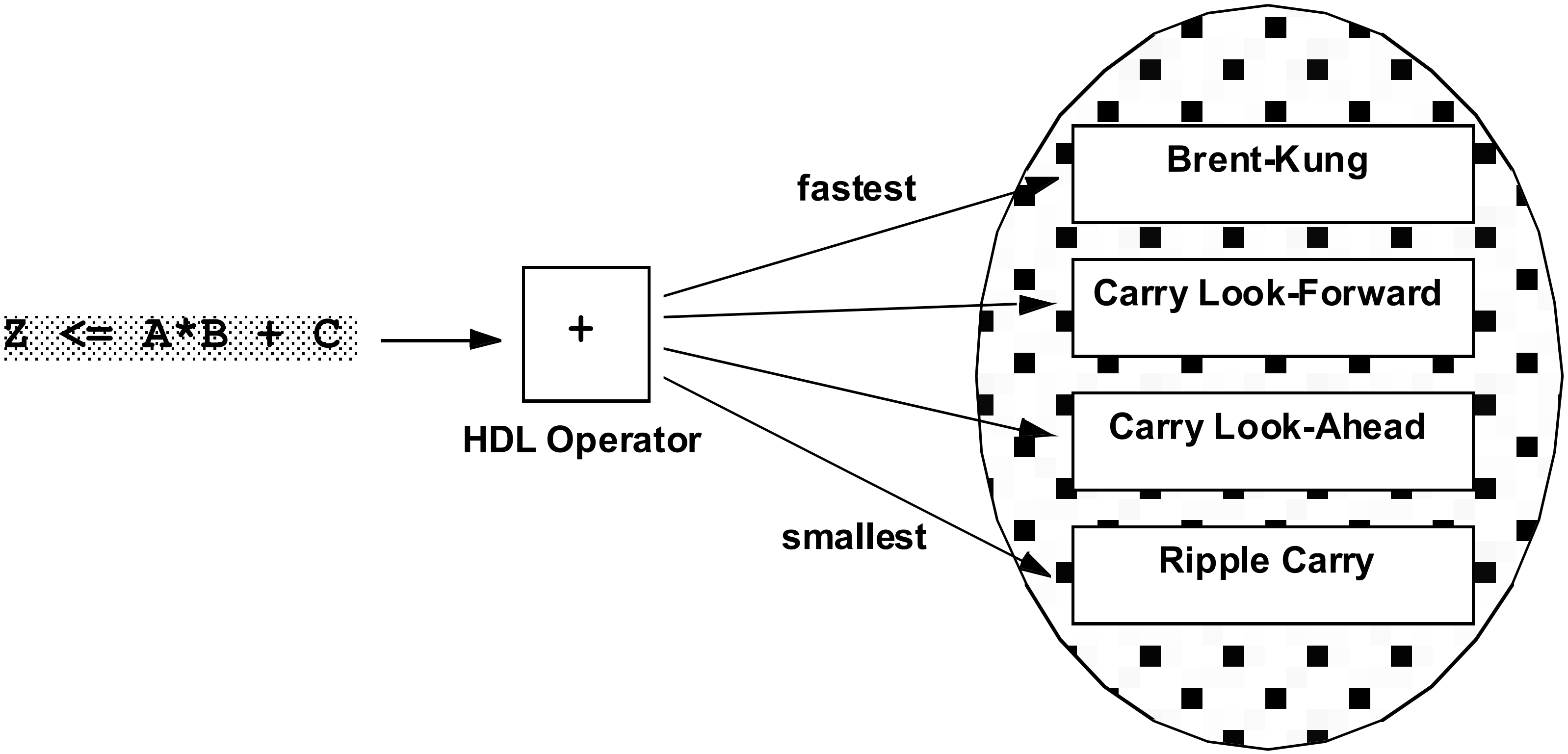
Resource Sharing: Solving It

- Construct a conflict graph
  – Nodes: resources
  – Edge: if two resources cant be shared → conflict
  – Question: if in the right graph all nodes are “adders”. What is the minimum number of adders needed for this Verilog ?
- Find the *chromatic number* of the graph
  – Color the nodes of the graph with minimum number of colors so that no two connected nodes have same color
  – NP Complete problem! → rely on heuristics

Conflict graph



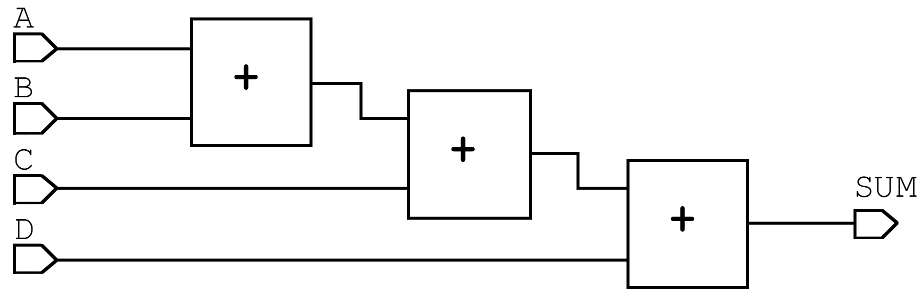
High-Level Optimization 2: Implementation Selection



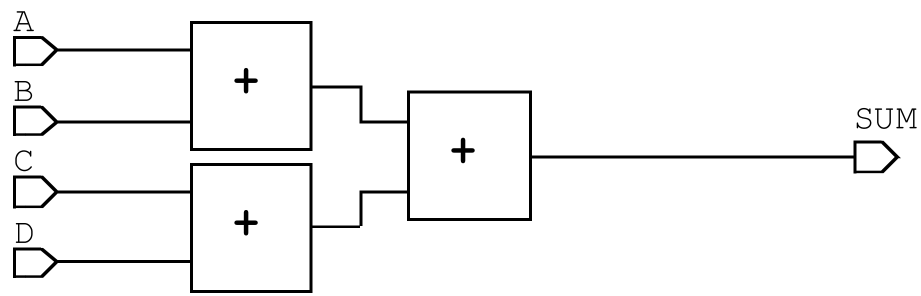
High-Level Optimization 3 – Operator Re-ordering

`SUM <= A + B + C + D`

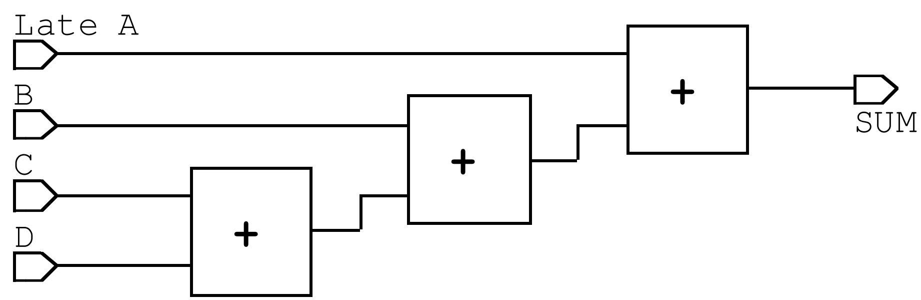
Initial Order
- from left to right



Optimised for Speed
- same delay for all inputs



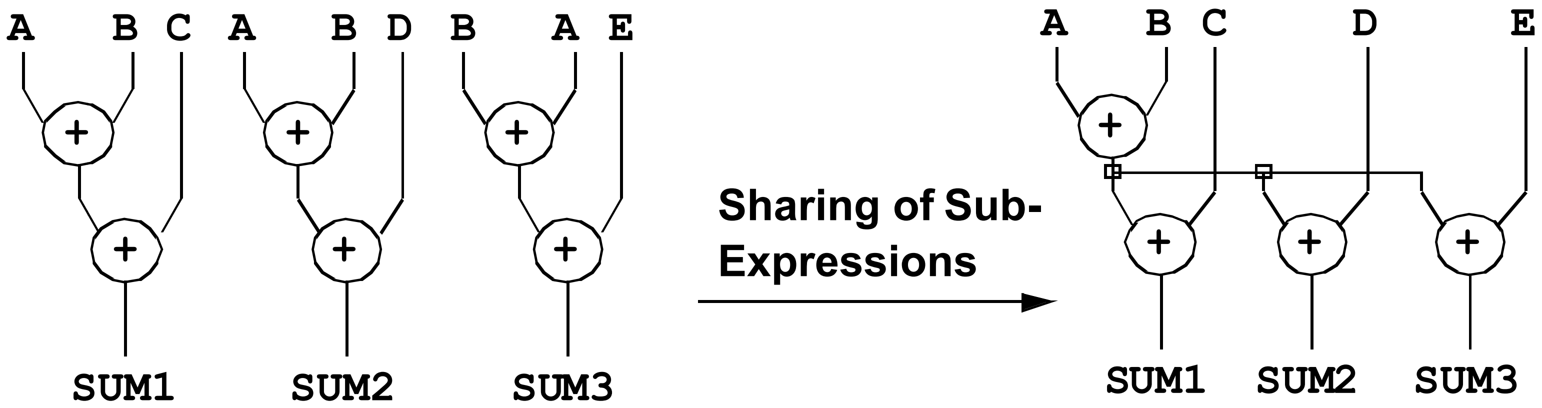
Optimised for Speed
- re-ordering due to large input delay for **A**



Note: **Operators can not be re-arranged if initial order is overridden by use of parenthesis in HDL**

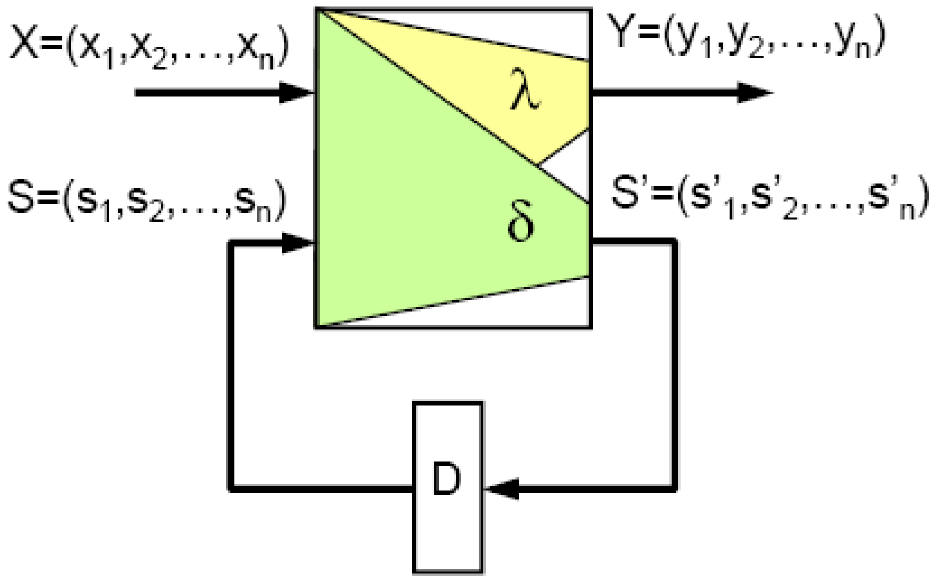
High-Level Optimization 4 – Common Sub Expression Sharing

```
SUM1 <= A + B + C
SUM2 <= A + B + D
SUM3 <= B + A + E
```



Note: Order of within the Sub-Expressions is not important, but the positions must be the same

Basic Logic Model: Finite State Machines



$M(X,Y,S,S_0,\delta,\lambda)$:

X: Inputs

Y: Outputs

S: Current State

S_0: Initial State(s)

δ: $X \times S \rightarrow S$ (next state function)

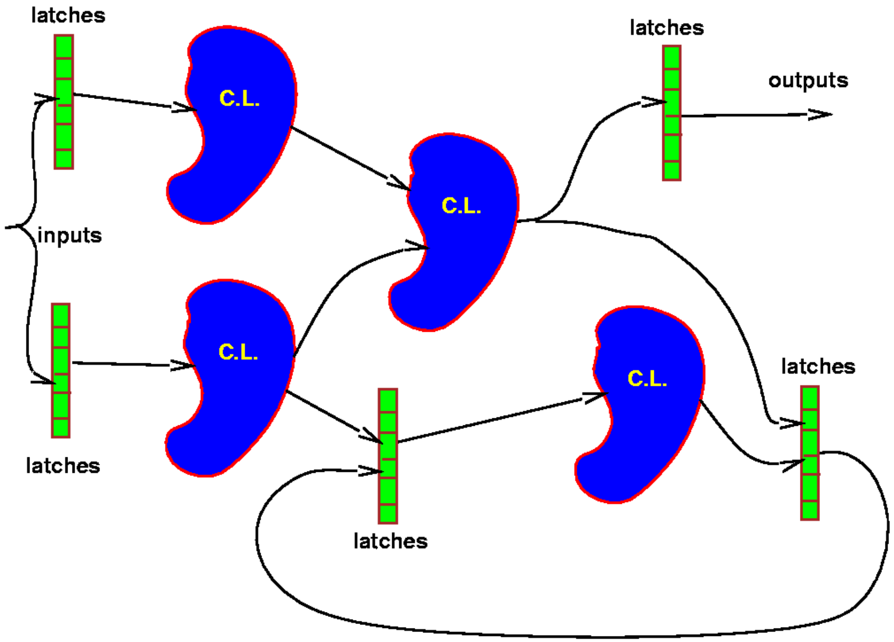
λ: $X \times S \rightarrow Y$ (output function)

Delay element:

- Clocked: synchronous
  - single-phase clock, multiple-phase clocks
- Unclocked: asynchronous

Puneet Gupta (puneet@ee.ucla.edu)

General Logic Structure



- Combinational optimization
  - Keep latches/registers at current position, keep their function
  - Optimize combinational logic in between
- Sequential optimization
  - Change latch position/function

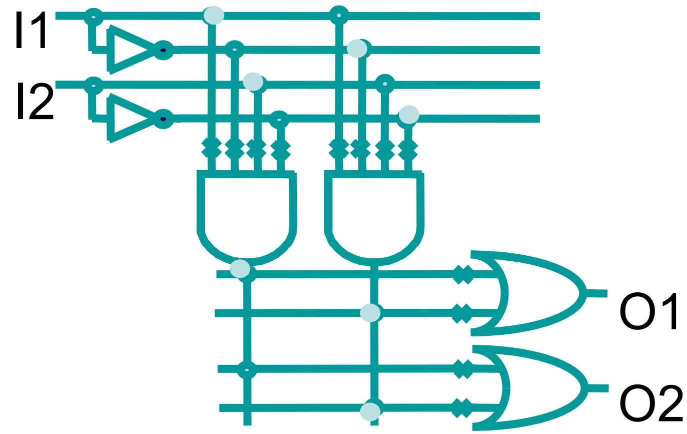
Optimization Cost Criteria

- Area occupied by the logic gates and interconnect (approximated by literals = transistors in technology independent optimization)
- Critical path delay of the longest path through the logic
- Power consumed by the logic gates
- Noise Immunity
- Routability

while simultaneously satisfying upper or lower bound constraints placed on these physical quantities

Logic Optimization Methods

- Two-level logic optimization
  - For sum-of-products (SOP) implementation on PLAs, fewer product terms and fewer inputs to each product term mean smaller area.
  - Karnaugh maps, Quine-McCluskey, Espresso
  - E.g., $F1 = AB + AC + AD$; $F2 = A'B + A'C + A'E$



O1 = I1 I2 + I1' I2'
O2 = I1' I2'

- Multi-level logic
  - E.g., $P = B + C$; $F_1 = AP + AD$; $F_2 = A'P + A'E$ → 3 levels → logic sharing → smaller area
  - Difficult to optimize

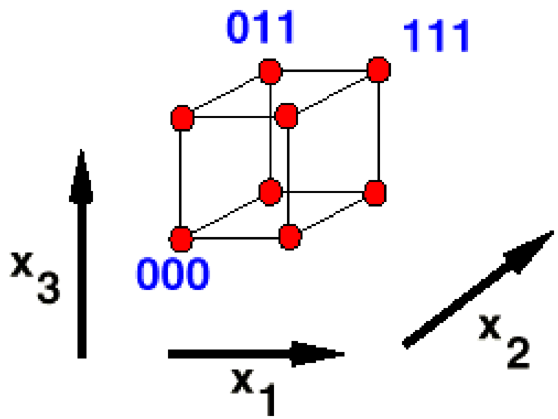
Boolean Functions

$f(x) : B^n \rightarrow B$

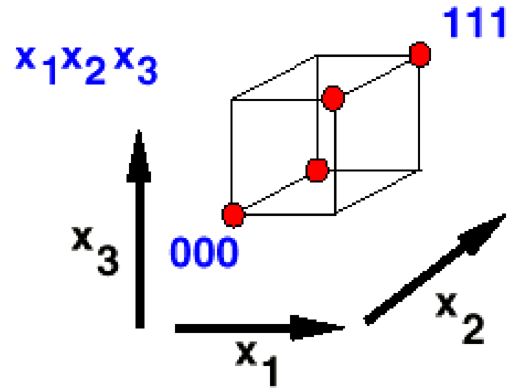
$B = \{0, 1\}, x = (x_1, x_2, \ldots, x_n)$

- $x_1, x_2, \ldots$ are variables
- $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots$ are literals
- each vertex of B^n is mapped to 0 or 1
- the onset of f is a set of input values for which $f(x) = 1$
- the offset of f is a set of input values for which $f(x) = 0$

Logic Functions: $f(x) : B^n \to B$



000
001
010
011
100
101
110
111



000	=>	1
001		0
010		1
011		0
100		0
101		1
110		0
111		1

"truth table"

There are 2^n vertices in input space B^n

There are 2^{2^n} distinct logic **functions**. Each subset of vertices is a distinct logic function: $f^1 \subseteq B^n$

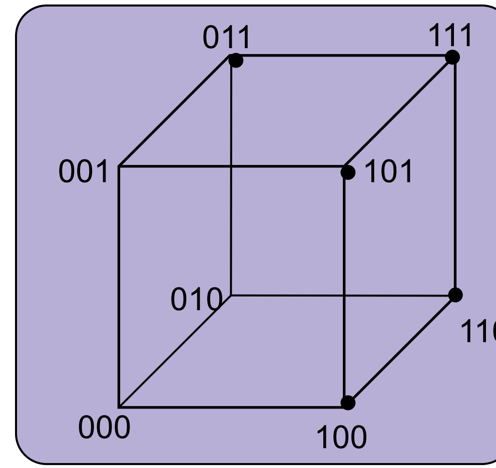
There are ∞ number of logic **formulas**

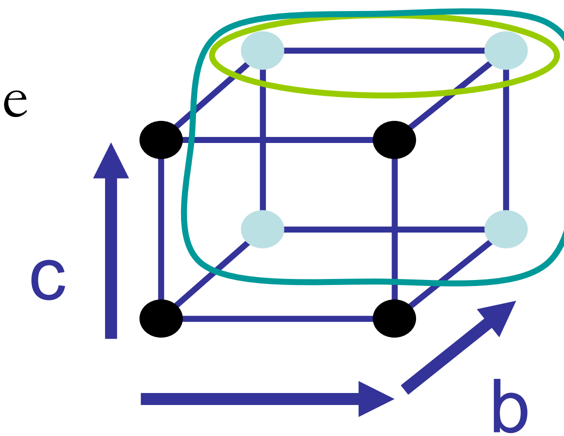
$$\begin{aligned} f &= x + y \\ &= x\overline{y} + xy + \overline{x}y \\ &= x\overline{x} + x\overline{y} + y \\ &= (x+y)(x+\overline{y}) + \overline{x}y \end{aligned}$$

SYNTHESIS = Find the "best" formula (or "representation")

Cube Representation

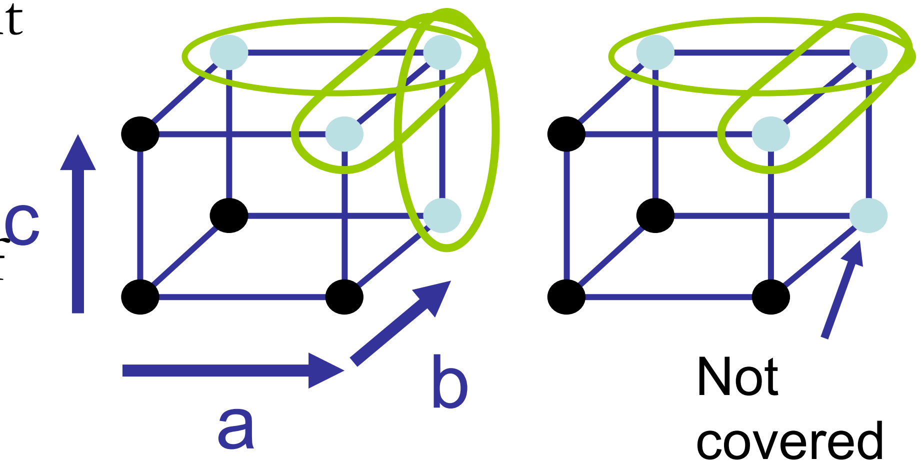
- N-dimensional cube
  - Marking top points in which function has individual values, receive its geometrical representation, for example function Y=V(3,4,5,6,7) is shown
- A SOP can be thought of as a set of cubes. A set of cubes that represents f is called a cover of f.
  - F={ab, ac, bc} is a cover of f = ab + ac + bc.
- *Prime Cube (Prime Implicant):* if there is no other cube that contains it → removing a literal will make it invalid
  - *E.g., b* is prime but *bc* is not
- *Prime Cover:* if all contained cubes are prime





Irredundant Covers

- A cube c of a cover C is irredundant if C fails to be a cover if c is dropped from C
- A cover is irredundant (minimal) iff all its cubes are irredundant (for example, F = a b + a c + b c)
- Logic minimization:
  - find a minimum prime and irredundant cover for a given function.
    - Prime cover leads to min number of inputs to each product term.
    - Min irredundant cover leads to min number of product terms.



5 min break