Concept of Logic Synthesis

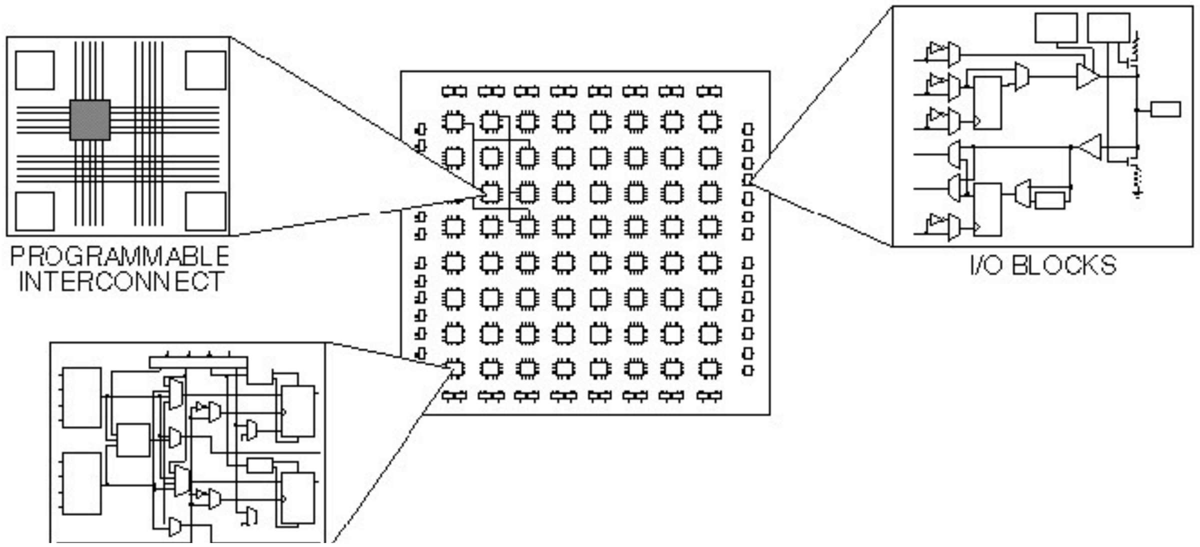
Naehyuck Chang
Computer Systems Design

naehyuck@elpl.snu.ac.kr

Seoul National University

FPGA Design Flow

- Architecture of the Xlinx FPGA
  - Programmable logic block (configureable logic block, CLB), programmable interconnect and programmable I/O block
  - Look-up table (LUT)
    - Arbitrary programmable a Boolean function of k inputs
      - K is 3, 4 or 5 depending on the architecture
  - Dedicated FF
    - Latch or FF



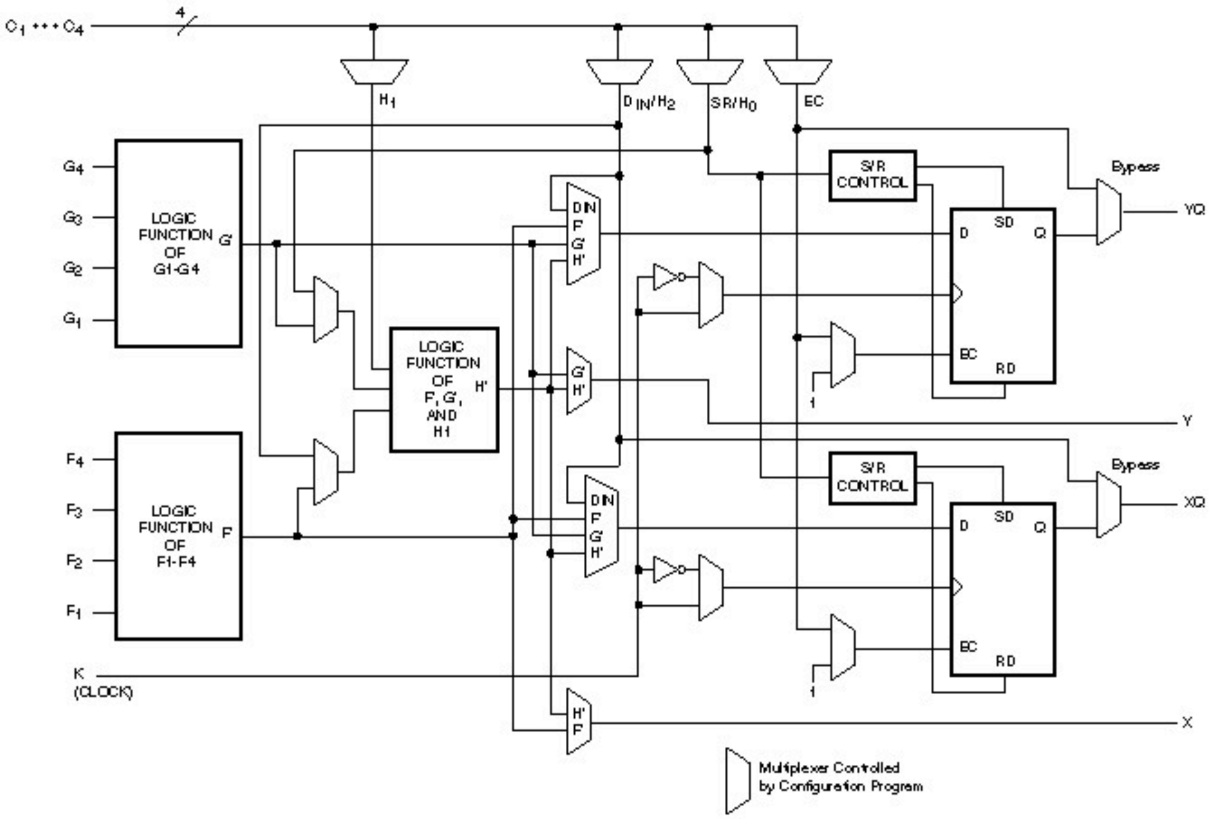
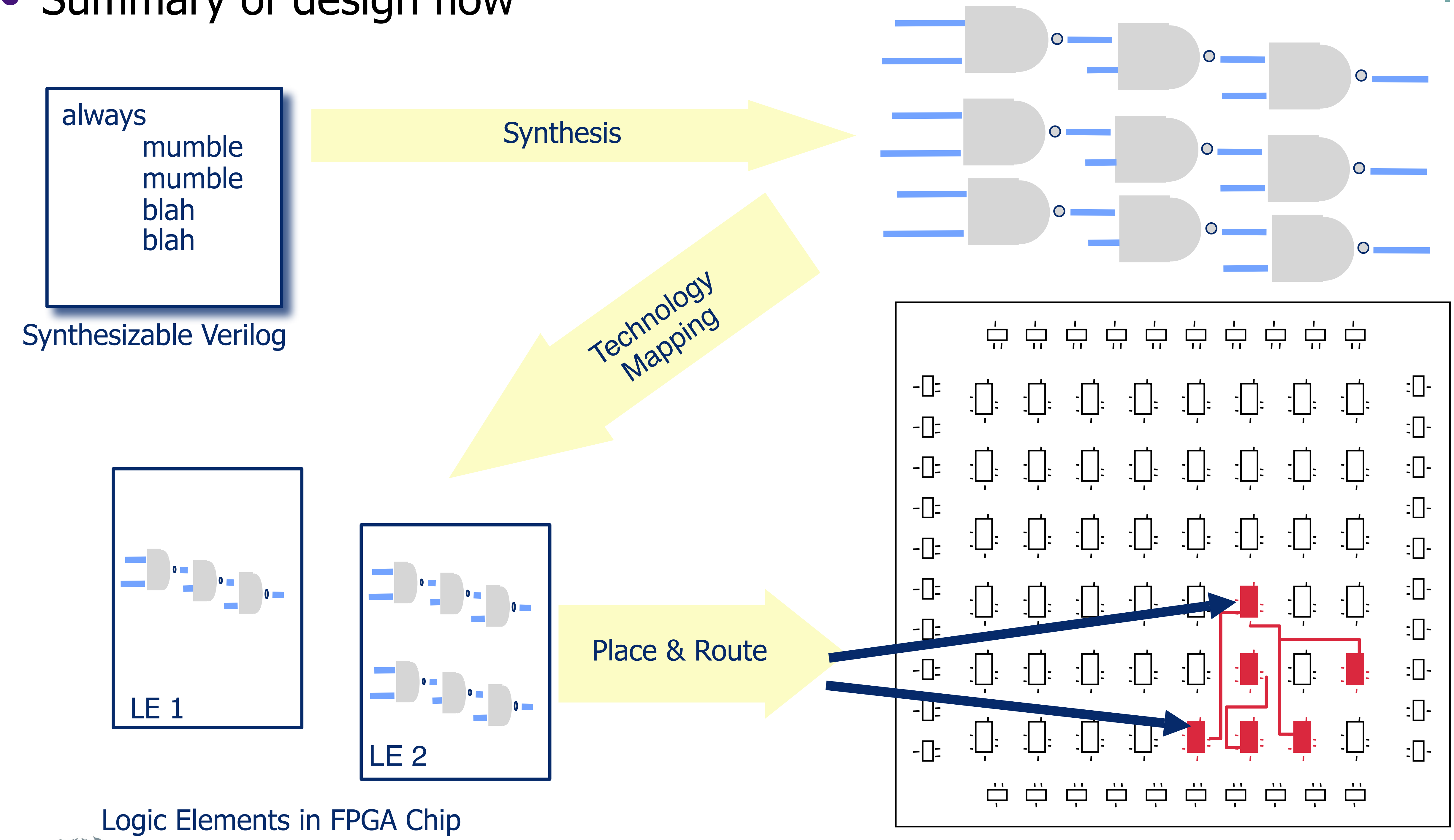


Figure 1: Simplified Block Diagram of XC4000-Series CLB (RAM and Carry Logic functions not shown)

FPGA Design Flow

- Summary of design flow



```
always
        mumble
        mumble
        blah
        blah
```

Synthesizable Verilog

Synthesis

Technology Mapping

Place & Route

LE 1

LE 2

Logic Elements in FPGA Chip

FPGA Design Flow

- Design activities
  - Implemented by complex Computer-Aided Design programs
    - You must know how to parameterize these correctly to get correct results
  - Estimation
    - Estimate likely design parameters
  - Synthesis
    - Translate design into lower level of representation
  - Simulation
    - Mimic design behavior at level of representation to see if it is correct
  - Analysis
    - Analyze design parameters at a level

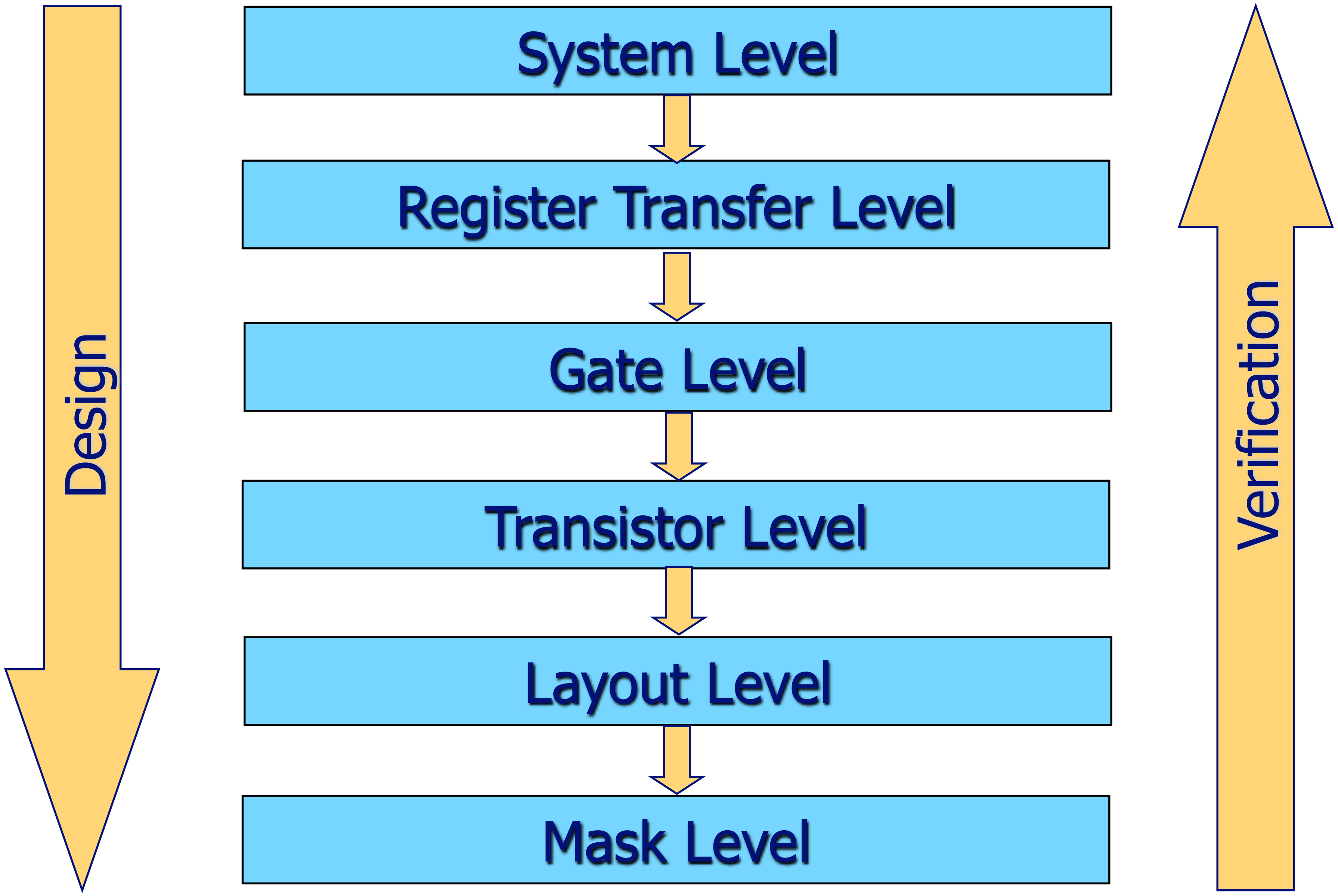
FPGA Design Flow

- Mapping
  - Fit logic produced by synthesis, place it onto a particular programmable logic device, transforming the logic as needed
- Place & Route
  - Place logic in a particular combinational Logic Block on an FPGA, such that the wiring delay between the block and others is acceptable
  - Must place critical circuit portions together to minimize wiring delays
    - Propagation delay of signals depends significantly on routing delay

FPGA Design Flow

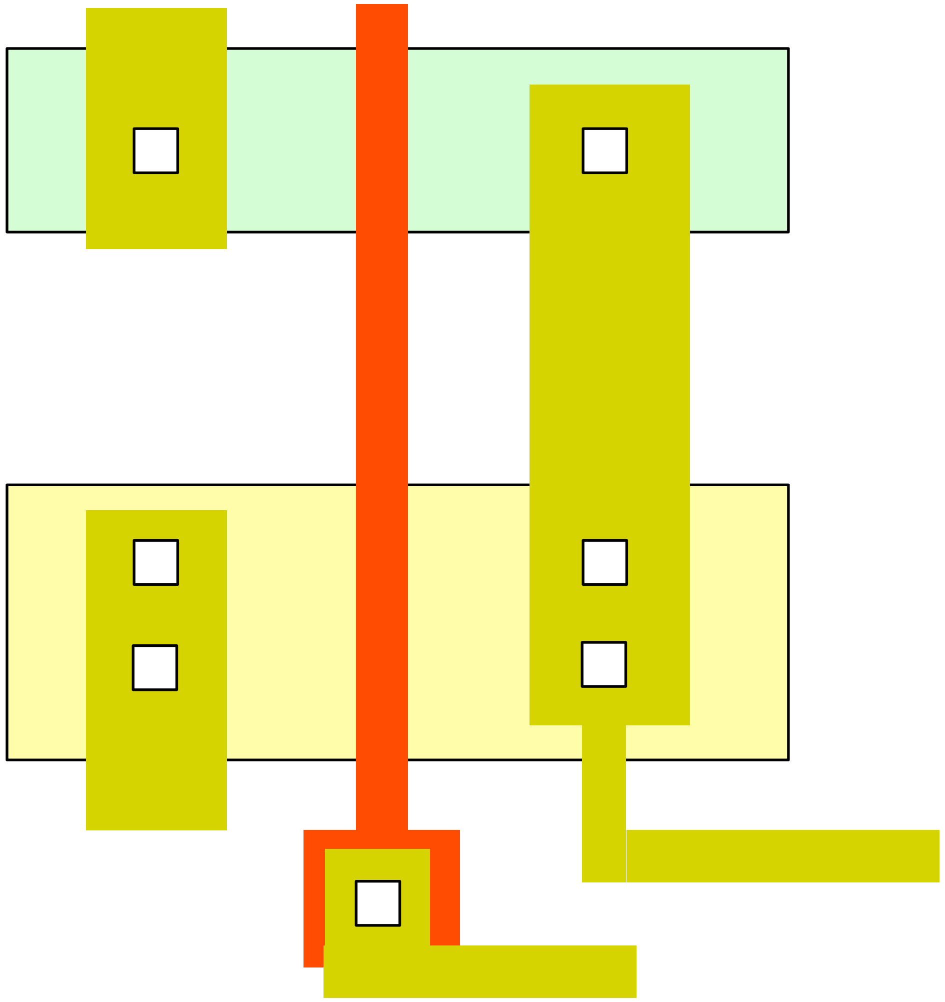
- Design Verification
  - Can simulate the placed & routed device with fairly realistic logic gate delays
  - Simulation always essential to verify correct design behavior
    - Can avoid making application-specific integrated circuits (ASICs), burning field-programmable gate arrays (FPGAs), or making full-custom chips that do not work
  - Must simulate at both behavioral and logic levels
    - Behavioral simulation finds logic errors
    - Logic simulation verifies that Synopsys designed logic correctly

Logic Design Practice



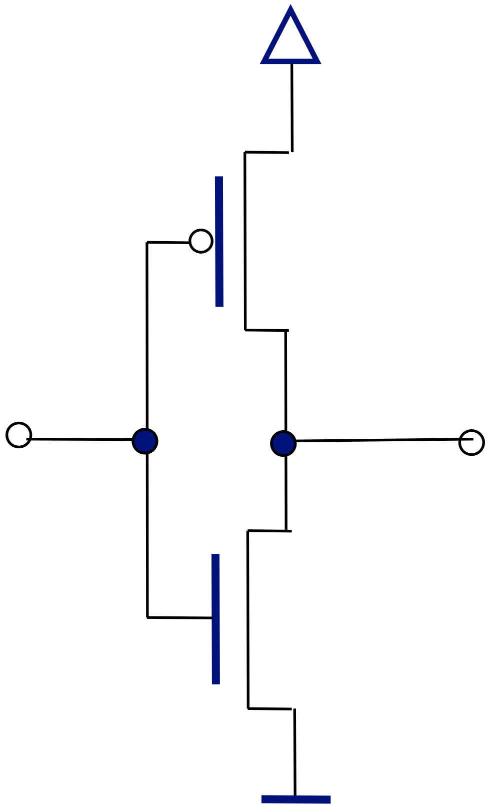
Logic Design Practice

- Layout level
  - Transistors and wires are laid out as polygons in different technology layers such as diffusion, poly-silicon, metal, etc.



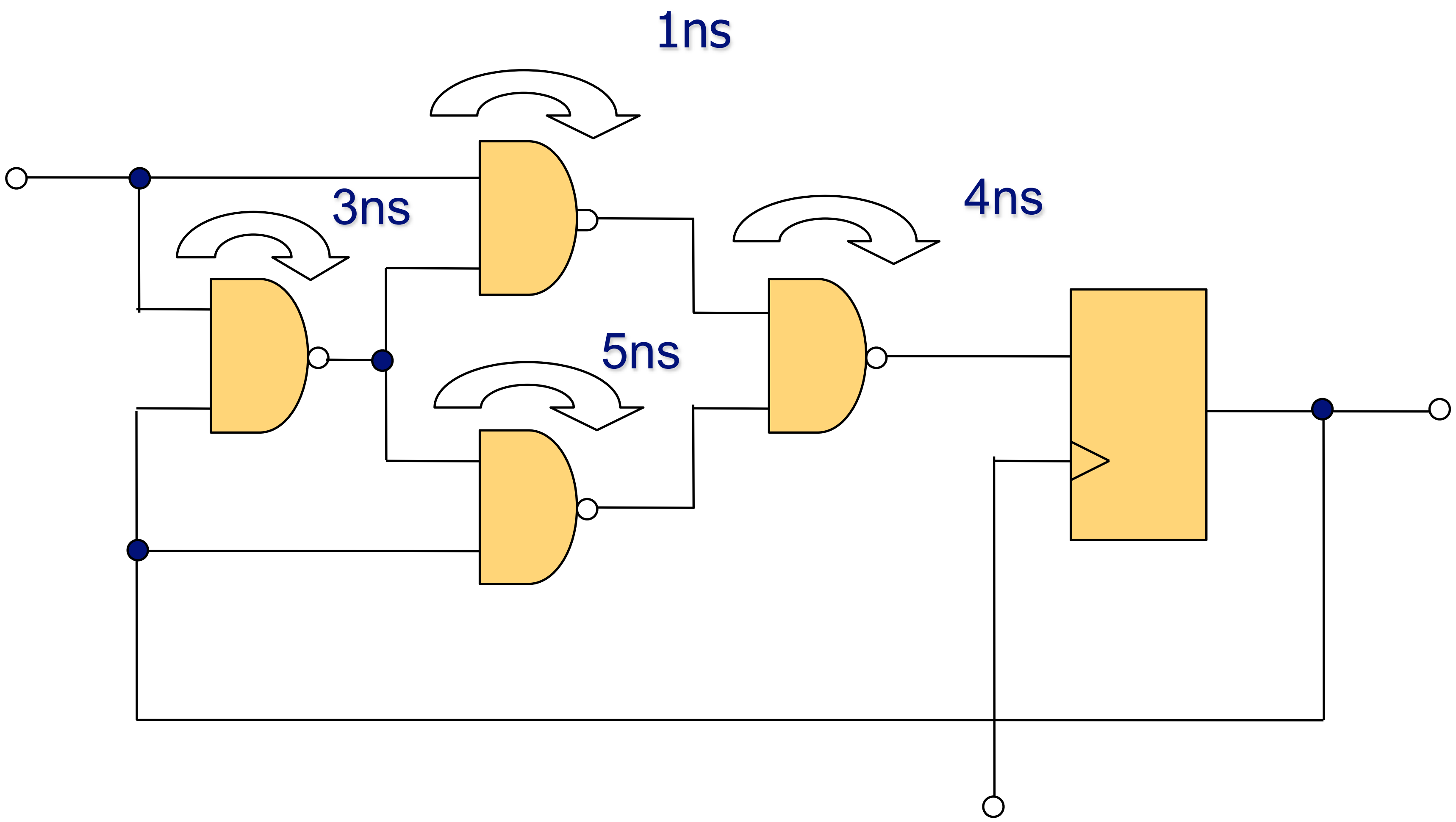
Logic Design Practice

- Transistor level
  - Model on CMOS transistor level
    - depending on application function modeled as resistive switches
      - used in functional equivalence checking
    - or full differential equations for circuit simulation
      - used in detailed timing analysis



Logic Design Practice

- Gate leve
  - Model on finite-state machine level
    - Models function in Boolean logic using registers and gates
    - Various delay models for gates and wires



Logic Design Practice

- RTL design
  - 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

```
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
```

Logic Design Practice

- System-level design
  - Abstract algorithmic description of high-level behavior
    - e.g. C-Programming language

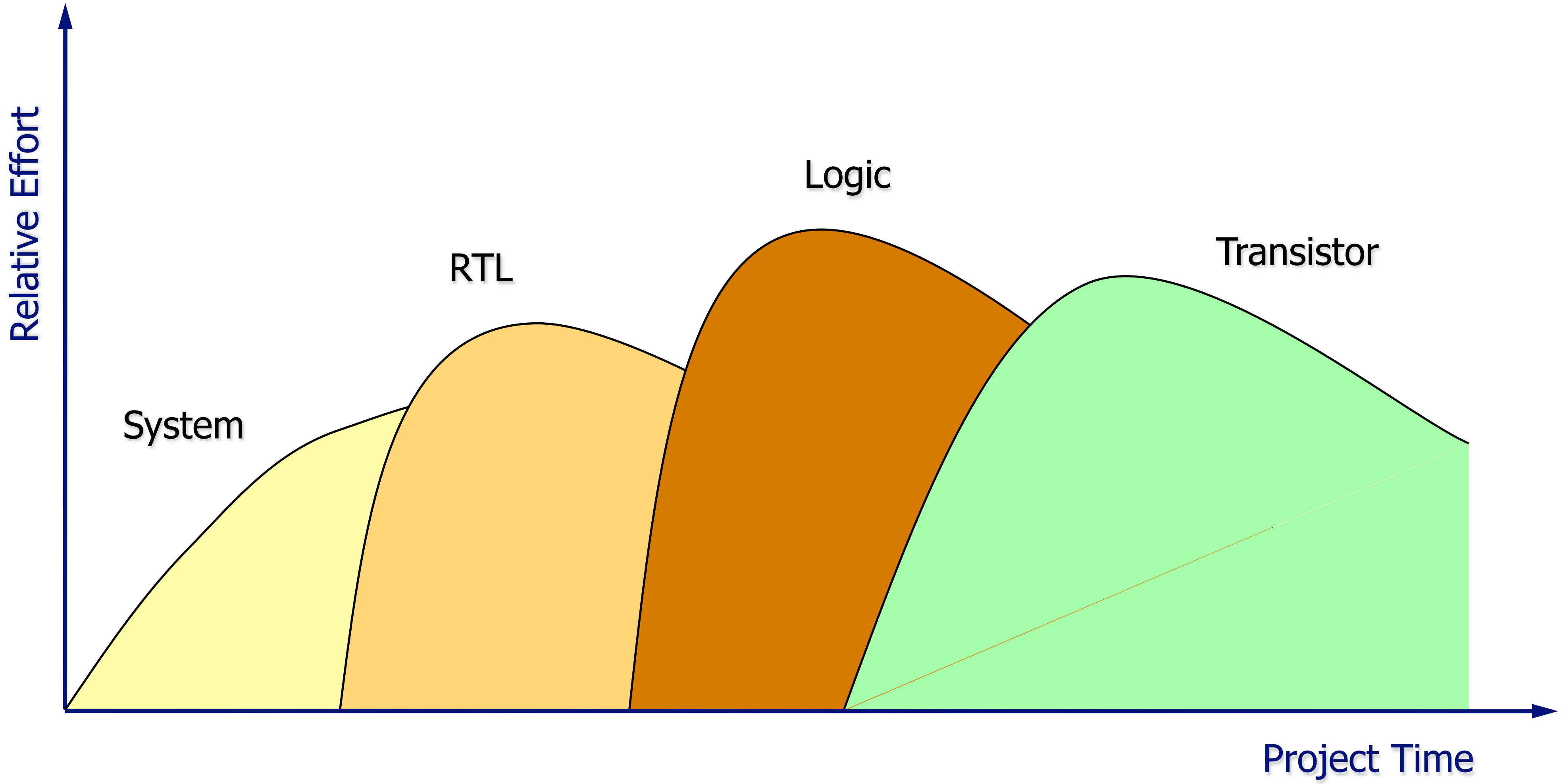
```
Port*
compute_optimal_route_for_packet(Packet_t *packet,
                                   Channel_t *channel)
{
  static Queue_t *packet_queue;

  packet_queue = add_packet(packet_queue, packet);
  ...
}
```

    - Abstract because it does not contain any implementation details for timing or data
    - Efficient to get a compact execution model as first design draft
    - Difficult to maintain throughout project because no link to implementation

Logic Design Practice

- Design phases overlap to large degrees
- Parallel changes on multiple levels, multiple teams
- Tight scheduling constraints for product

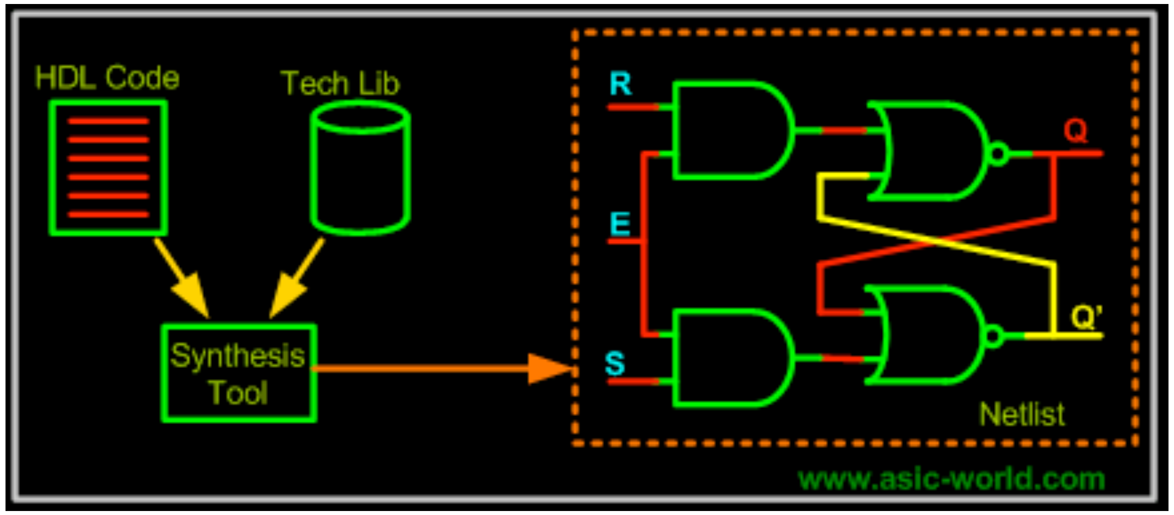


Concept of Logic Synthesis

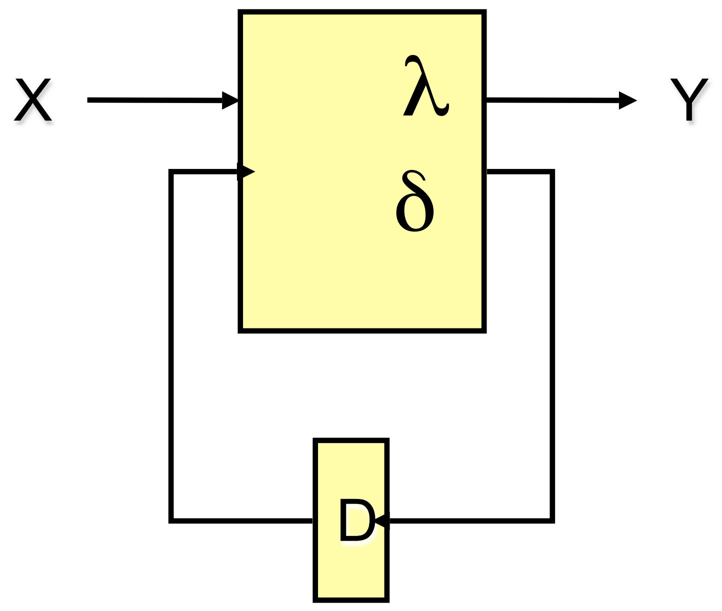
- Now possible to automatically design hardware using a CAD program
  - Automatically translates a high-level hardware description language (Verilog or VHDL) into logic gates
  - Transparent hardware (target technologies or even logic level structures) that save hardware design effort
  - Widely used at all electronics companies
- Resulting hardware design is not always acceptable
  - Designer must check the design to determine its quality
    - If unacceptable, redesign manually using K-maps and lower-level hardware synthesis tools
    - Example: A NJ company went out of business because they used a bad design created with VHDL
      - Too many logic gates, too slow, and too expensive
    - Transparent hardware does not mean the designer does not have to know about the real silicon

Concept of Logic Synthesis

- A process of converting a high-level description of design into an optimized gate-level representation
  - Legacy description
    - Directly describe the circuit structure
    - Schematic diagram and Boolean equations
  - High-level description
    - Hardware description languages
    - Programming language like C
    - Dataflow or Petri Net
    - Matlab Simulink
  - Uses a standard cell library
    - Simple cells, such as basic logic gates like and, or, and nor, or macro cells, such as adder, muxes, memory, and flip-flops.
    - Standard cells put together are called technology library.

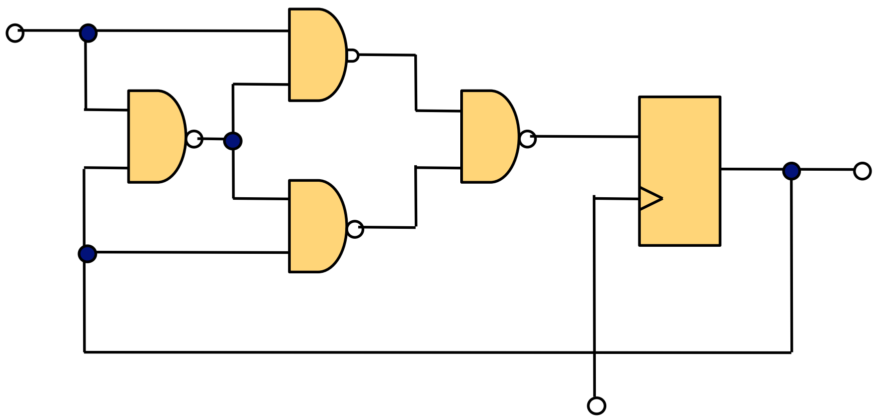


Concept of Logic Synthesis



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

X: Input alphabet
Y: Output alphabet
Z: Set of internal states
λ: $X \times Z \rightarrow Z$ (next state function)
δ: $X \times Z \rightarrow Y$ (output function)



Target: Circuit C(G, W) where:
G: set of circuit components $g \in$ {Boolean gates, flip-flops, etc}
W: set of wires connecting G

Concept of Logic Synthesis

- Objective function for logic synthesis
  - Minimize area
    - in terms of literal count, cell count, register count, etc.
  - Minimize power
    - in terms of switching activity in individual gates, deactivated circuit blocks, etc.
  - Maximize performance
    - in terms of maximal clock frequency of synchronous systems, throughput for asynchronous systems
  - Any combination of the above
    - combined with different weights
    - formulated as a constraint problem
      - "minimize area for a clock speed > 300MHz"
  - More global objectives
    - feedback from layout
      - actual physical sizes, delays, placement and routing

Concept of Logic Synthesis

- Before HDL (Logic synthesis)
  - All the digital circuits were designed manually
    - Draw K-maps, optimize the logic, and draw the schematic
- Impact of HDL and Logic synthesis
  - Less prone to human error because designs are described at a higher level of abstraction
  - Done without significant concern about design constraints
  - Conversion from high-level design to gates is done by synthesis tools, using various algorithms to optimize the design as a whole
    - This removes the problem with varied designer styles for the different blocks in the design and suboptimal designs
    - Logic synthesis tools allow technology independent design
    - Design reuse is possible for technology-independent descriptions

Concept of Logic Synthesis

- Verilog language
  - Concurrent hardware description language
    - Expresses parallelism in the hardware
  - DO NOT code Verilog like a C or FORTRAN program
    - Serializes the hardware operations
    - Leads to a BIG increase in the amount of the hardware

Concept of Logic Synthesis

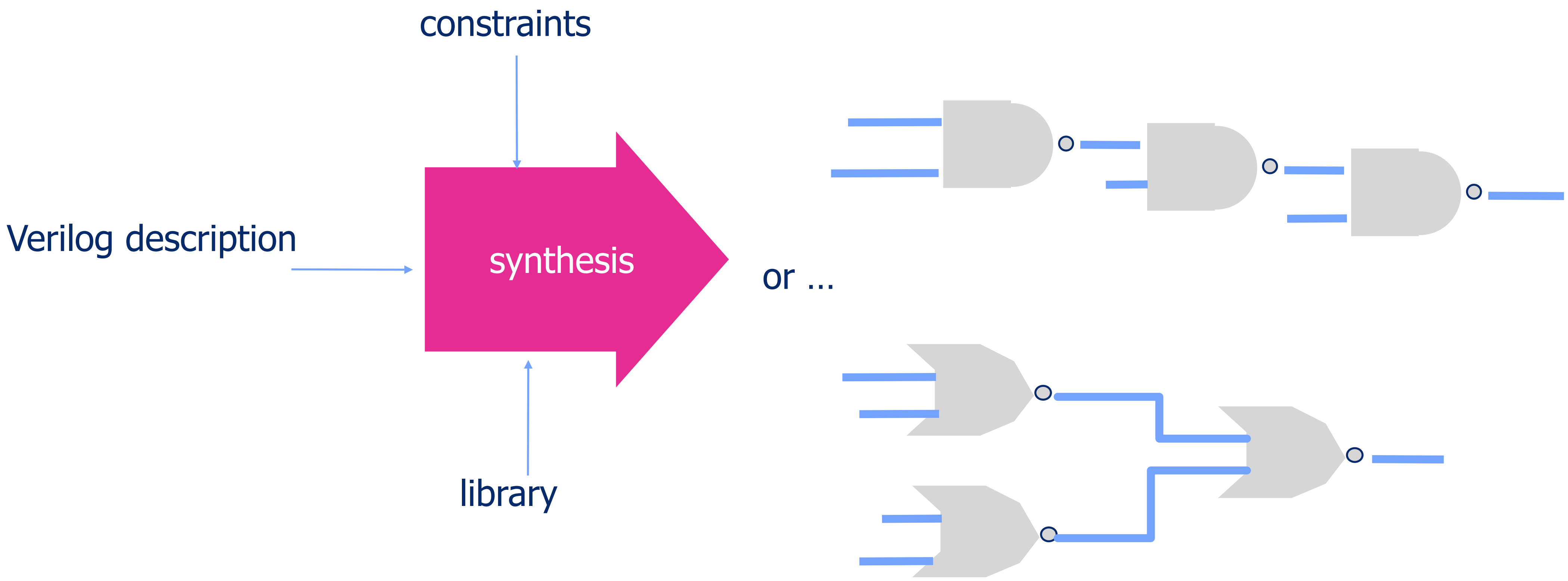
- Verilog versus VHDL
  - VHDL
    - Used in all Dept. of Defense (DoD) military system designs
    - Used throughout Europe, Japan, and IBM
    - Has problems with type conversions between Boolean and arithmetic
    - Originally invented for documentation not logic synthesis
  - Verilog
    - Preferred in the commercial electronics industry
    - Best for converting data types between bit vector and arithmetic notations
    - Best for configuring large designs produced by large design teams
    - Best for describing low-level logic (more concise)
  - Reality: Probably need to know both languages
    - Impossible to say which is better – matter of taste

Concept of Logic Synthesis

- Shortcomings of Verilog or VHDL
  - You lose some control of defining the gate-level circuit implementation
    - You don't have time to do that, anyway
  - Logic synthesized by the Verilog compiler is sometimes inefficient
    - A real problem – Must learn to "explain" the design to the compiler in the "right" way to get maximum hardware parallelism, which leads to the best design
  - Quality of synthesis varies from tool to tool

Logic Synthesis

- Logic synthesis
  - A program that "designs" logic from abstract descriptions of the logic
    - Takes constraints (e.g. size, speed)
    - Uses a library (e.g. 3-input gates)
  - You write an "abstract" Verilog description of the logic
  - The synthesis tool provides alternative implementations



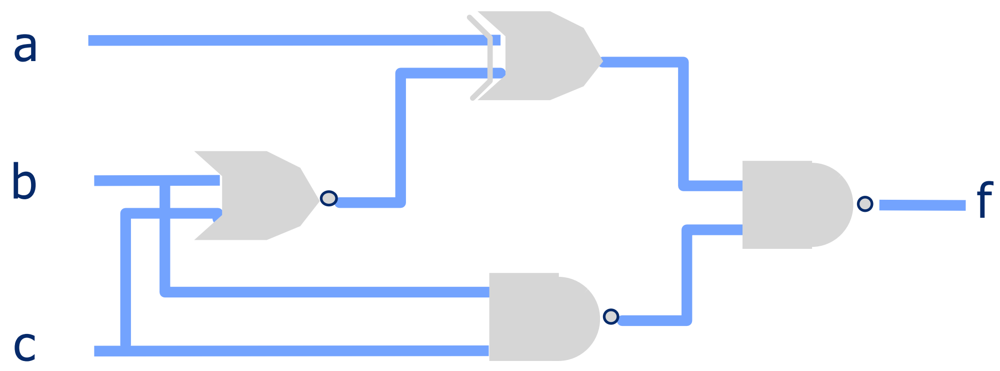
Logic Synthesis

- Example
  - Use of 2-input gate library
  - Use of a gate-level description of Verilog

```
module gate (f, a, b, c);
output          f;
input           a, b, c;

      and       A (a1, a, b, c),
                B (a2, a, ~b, ~c),
                C (a3, ~a, o1);
      or        D (o1, b, c),
                E (f, a1, a2, a3);
endmodule
```

synthesis



Logic Synthesis

- Synthesizable codes
  - The following codes are not synthesizable

Initial statement is not synthesizable

```
module synthesis_initial(
clk,q,d);
input clk,d;
output q;
reg q;

initial begin
 q <= 0;
end

always @ (posedge clk)
begin
 q <= d;
end

endmodule
```

Comparison with x and z is ignored

```
module synthesis_compare_xz (a,b);
output a;
input b;
reg a;

always @ (b)
begin
  if ((b == 1'bz) || (b == 1'bx)) begin
    a = 1;
  end else begin
    a = 0;
  end
end

endmodule
```

Logic Synthesis

- Synthesizable codes
  - Constructs not supported in synthesis

Construct Type	Notes
initial	Used only in test benches.
events	Events make more sense for syncing test bench components.
real	Real data type not supported.
time	Time data type not supported.
force and release	Force and release of data types not supported.
assign and deassign	assign and deassign of reg data types is not supported. But assign on wire data type is supported.
fork join	Use nonblocking assignments to get same effect.
primitives	Only gate level primitives are supported.
table	UDP and tables are not supported.

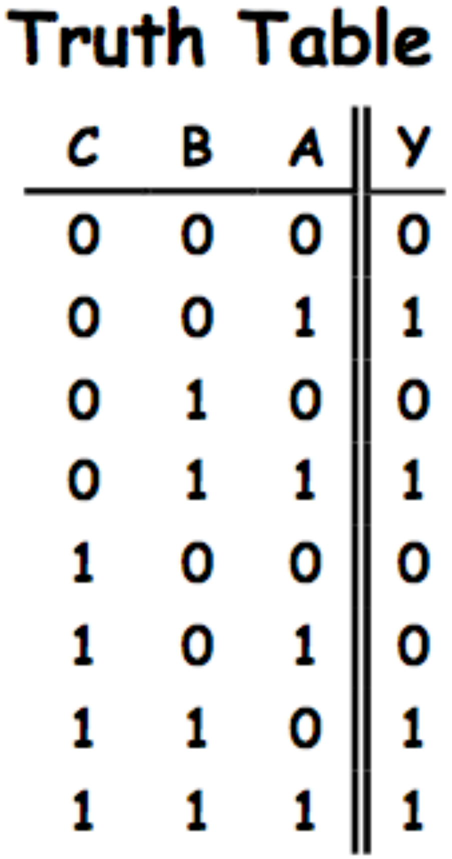
Logic Synthesis

- Synthesizable codes
  - Constructs supported in synthesis

Construct Type	Keyword or Description	Notes
ports	input, inout, output	Use inout only at IO level
parameters	parameter	This makes design more generic
module definition	module	
signals and variables	wire, reg, and tri	Vectors are allowed
instantiation	module instances/primitive gate instances	E.g.- nand (out,a,b), bad idea to code RTL this way
function and tasks	function, task	Timing constructs ignored
procedural	always, if, else, case, casex, and casez	initial is not supported
procedural blocks	begin, end, named blocks, and disable	Disabling of named blocks allowed
data flow	assign	Delay information is ignored
named Blocks	disable	Disabling of named block supported
loops	for, while, and forever	While and forever loops must contain @(posedge clk) or @(negedge clk)

Target Technologies

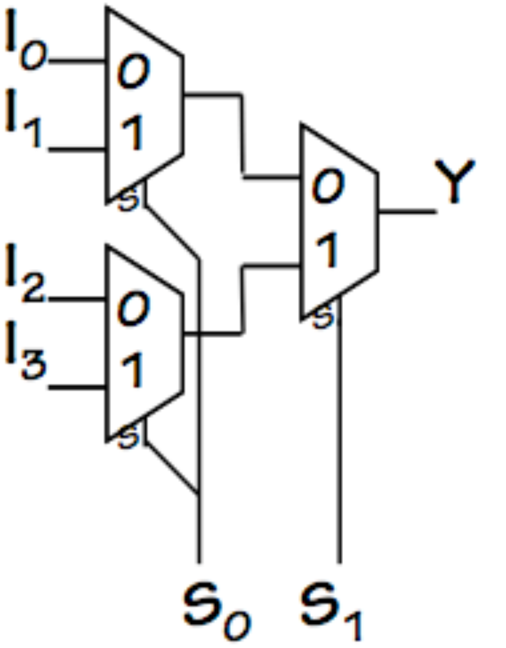
- Using a multiplexer
  - Multiplexer selections are the inputs
  - Multiplexer output is the output
  - Multiplexer inputs are tied either 1 or 0 depending on the logic function



Truth Table

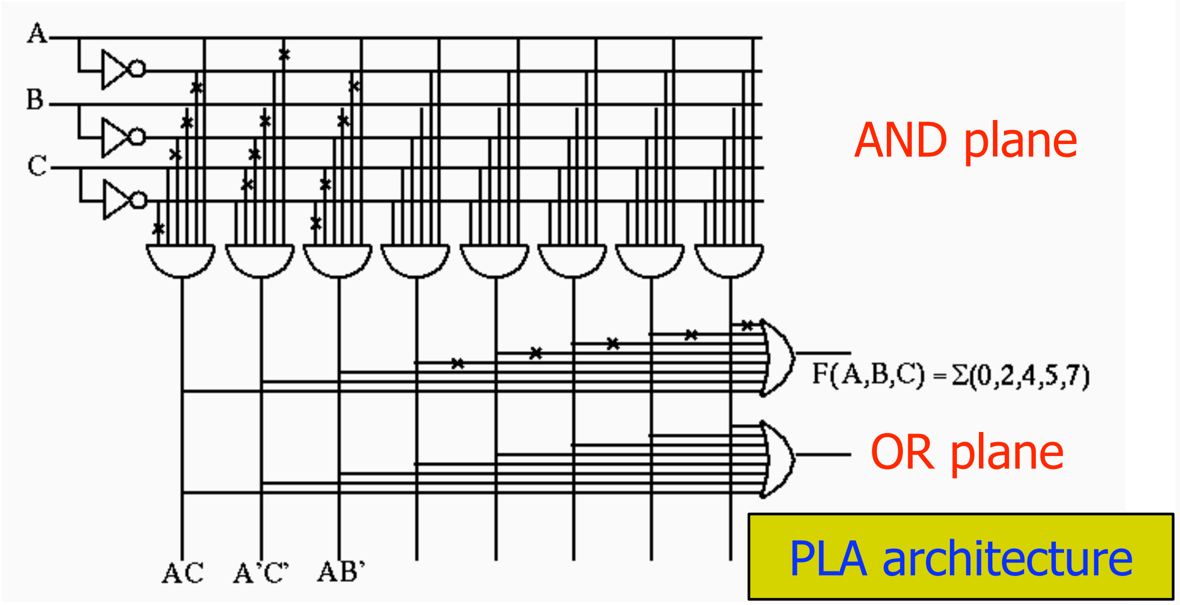
C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

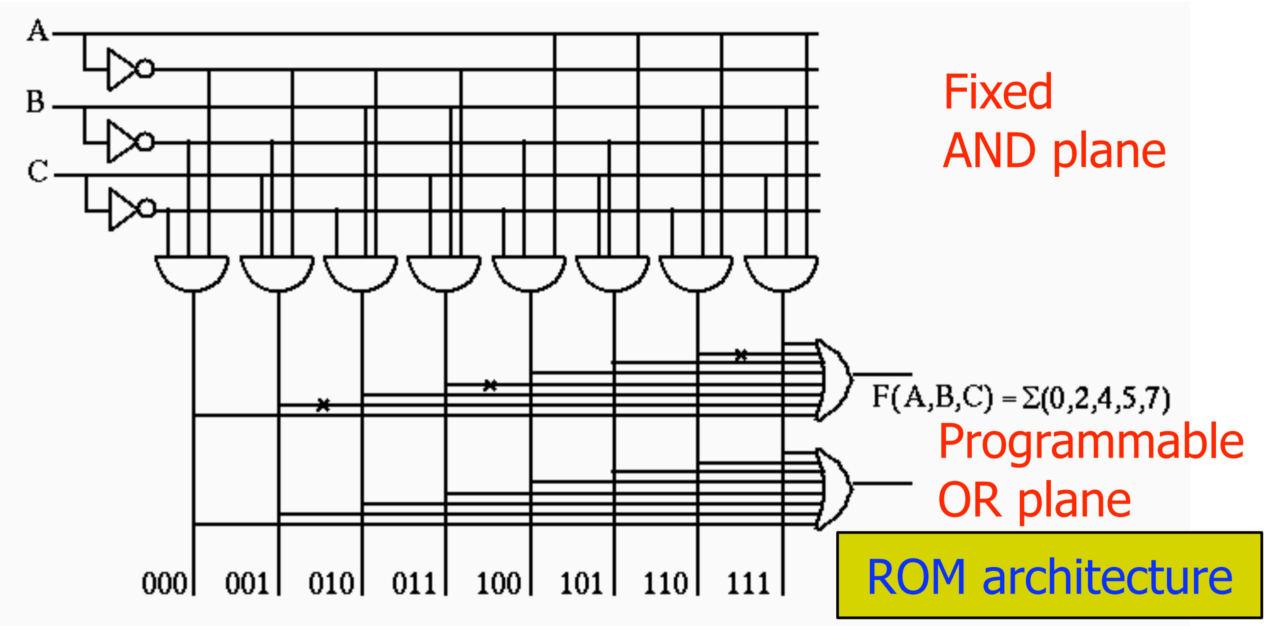
A 4-input Mux implemented as a tree

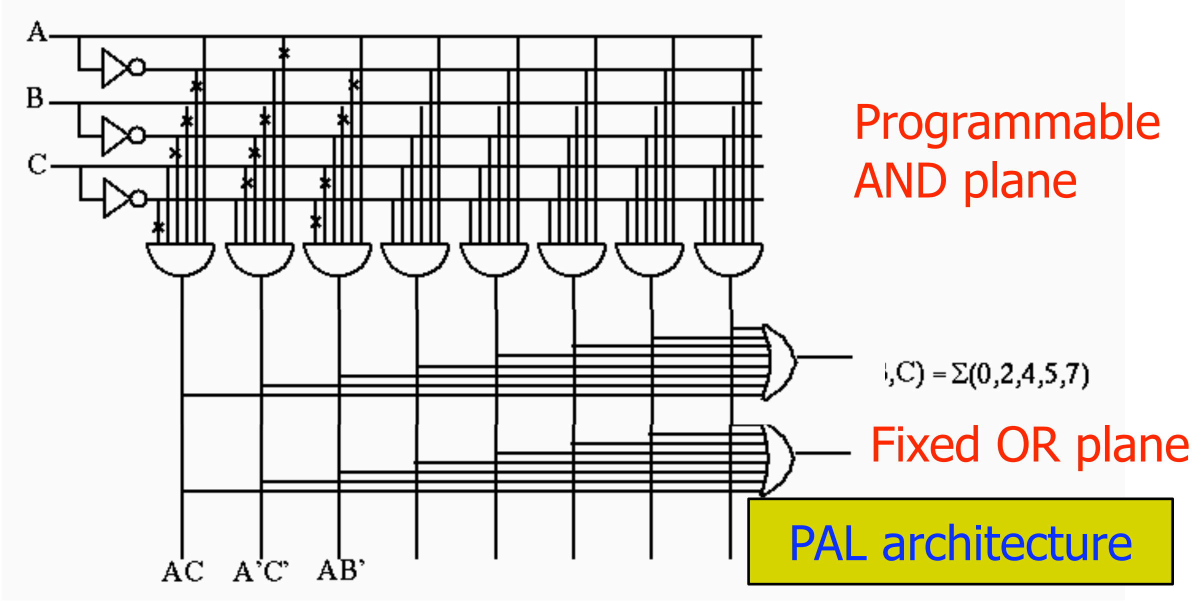


Target Technologies

- Using a ROM and a PLA

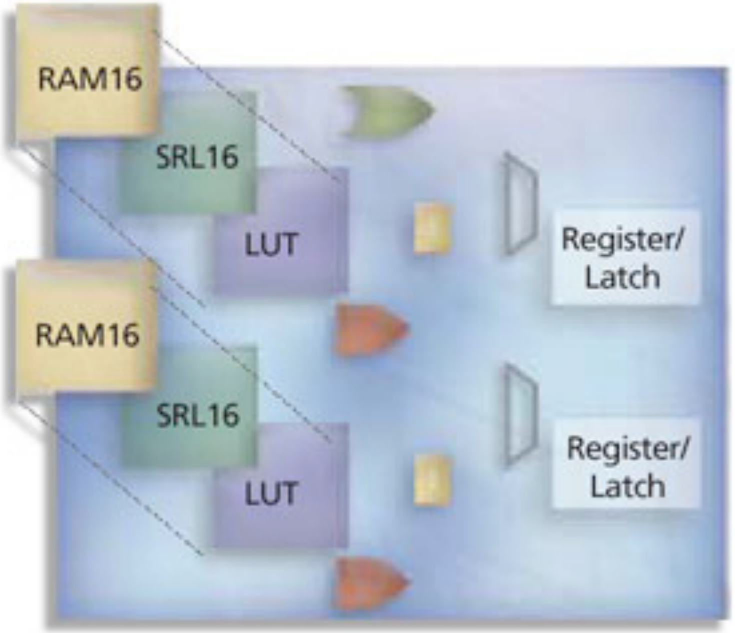


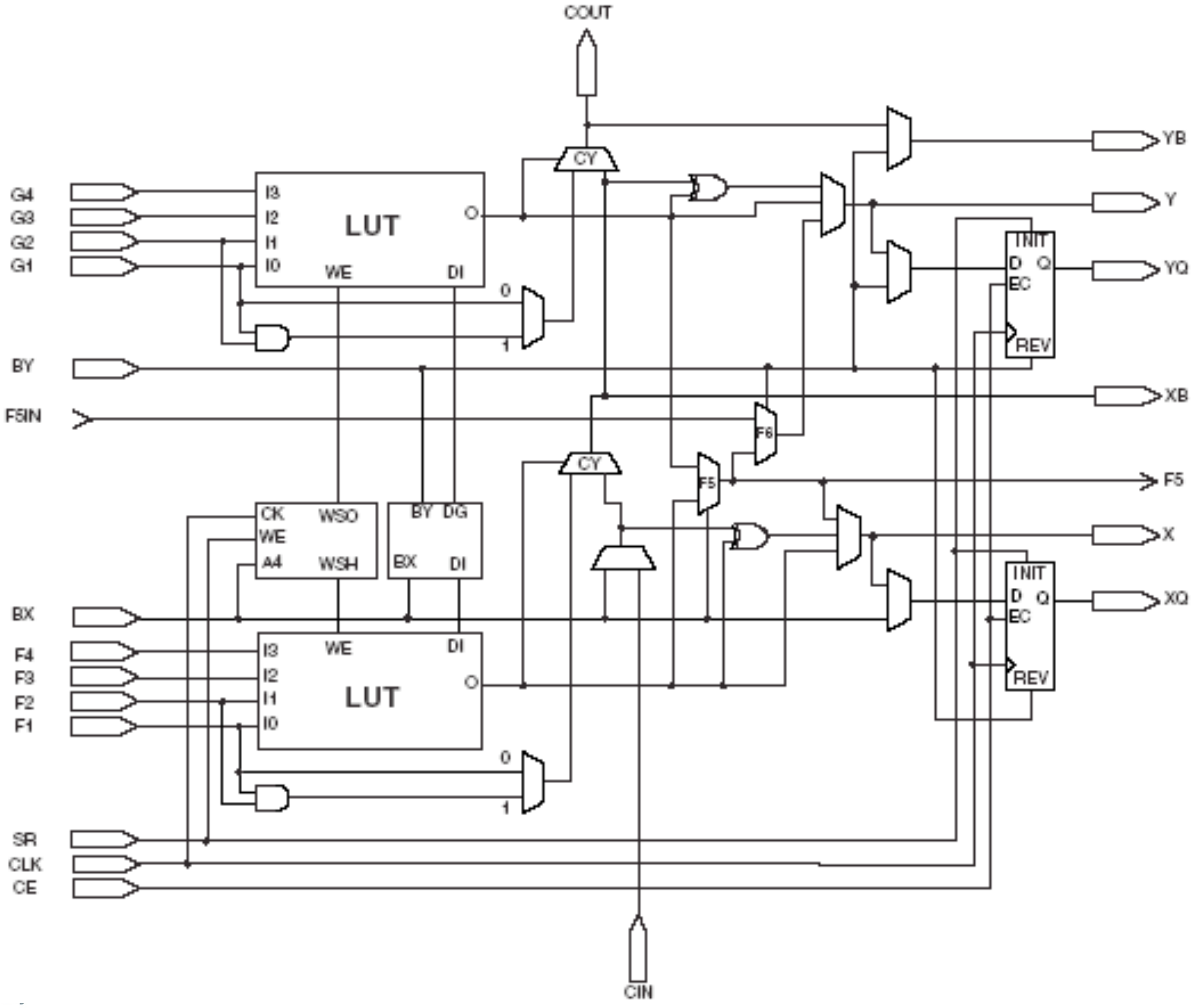




Target Technologies

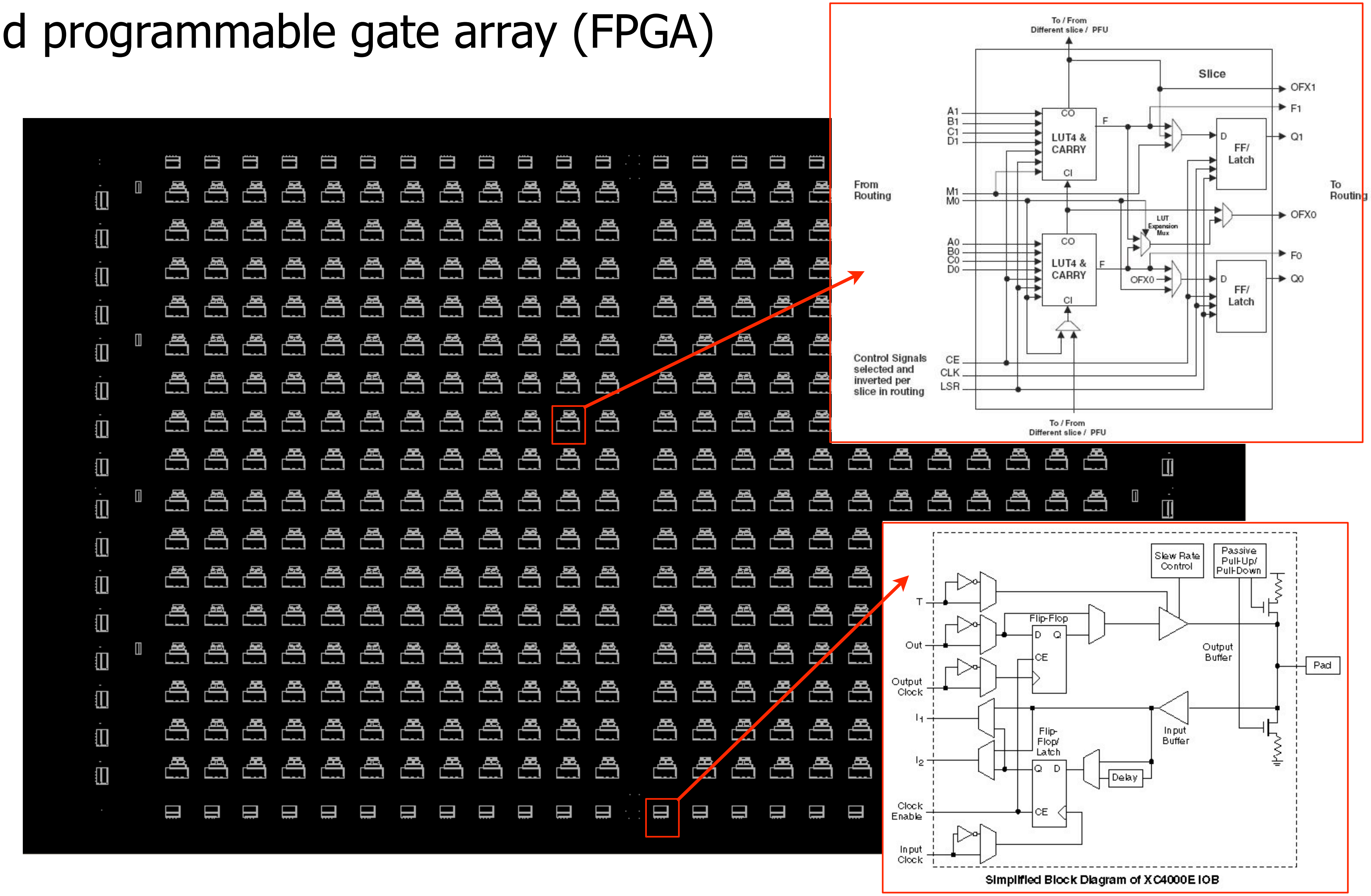
- Using a look-up table
  - Same to the ROM-based logic
    - Small LUT (look up table): 3 to 5 inputs
    - Dedicated flip flops and multiplexers





Target Technologies

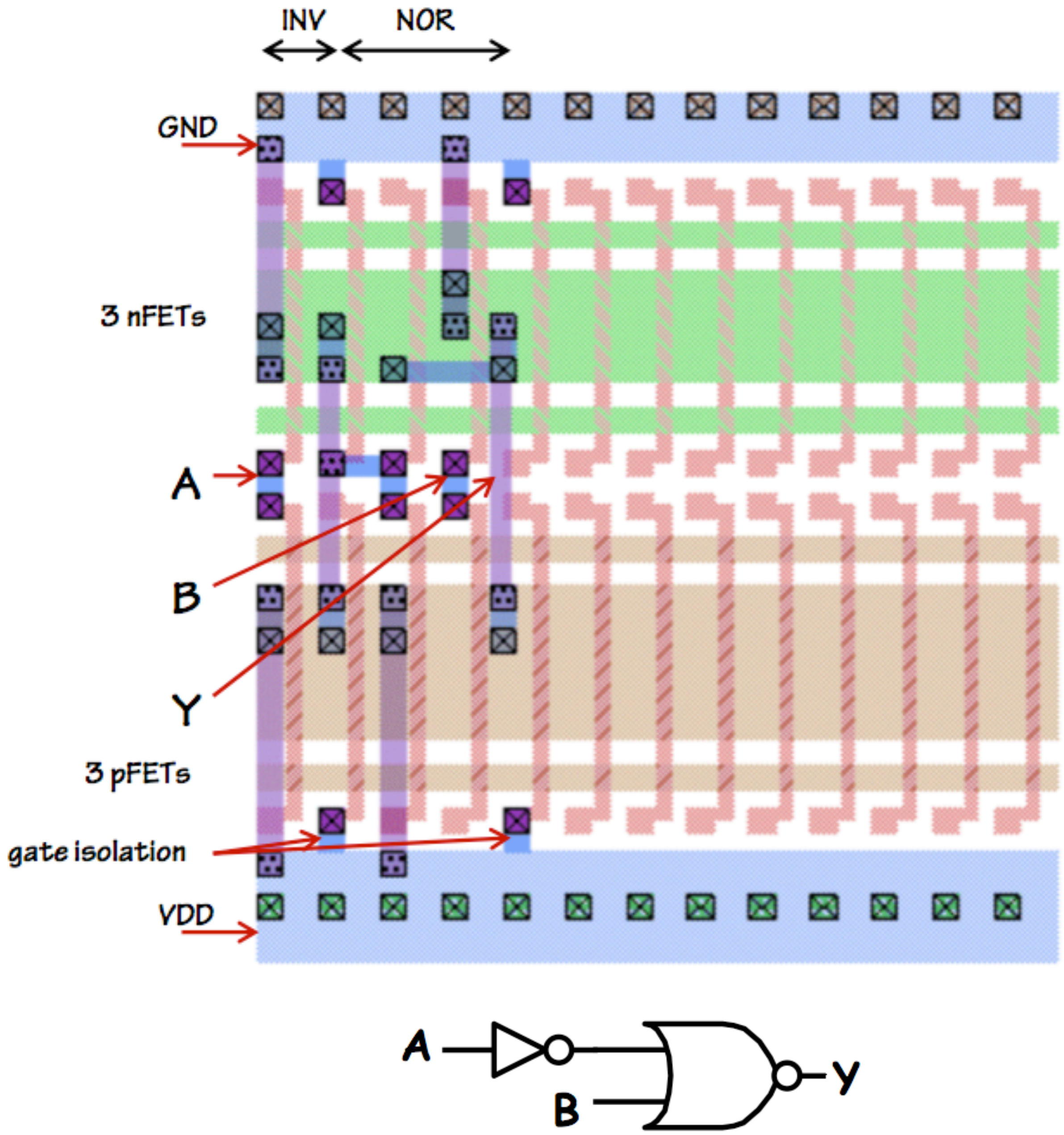
- Field programmable gate array (FPGA)



Simplified Block Diagram of XC4000E IOB

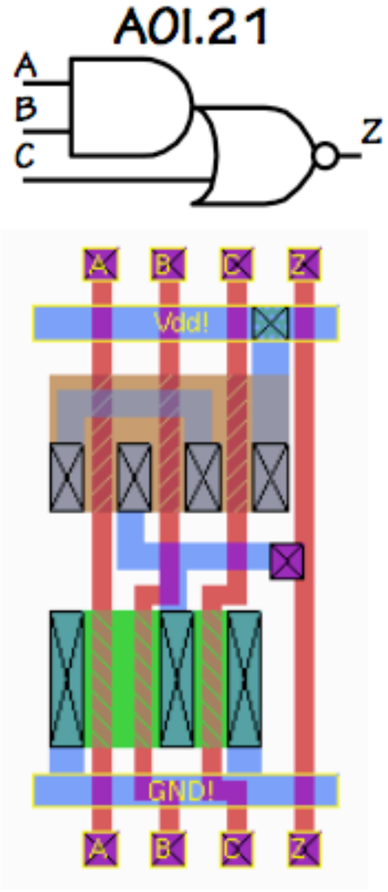
Target Technologies

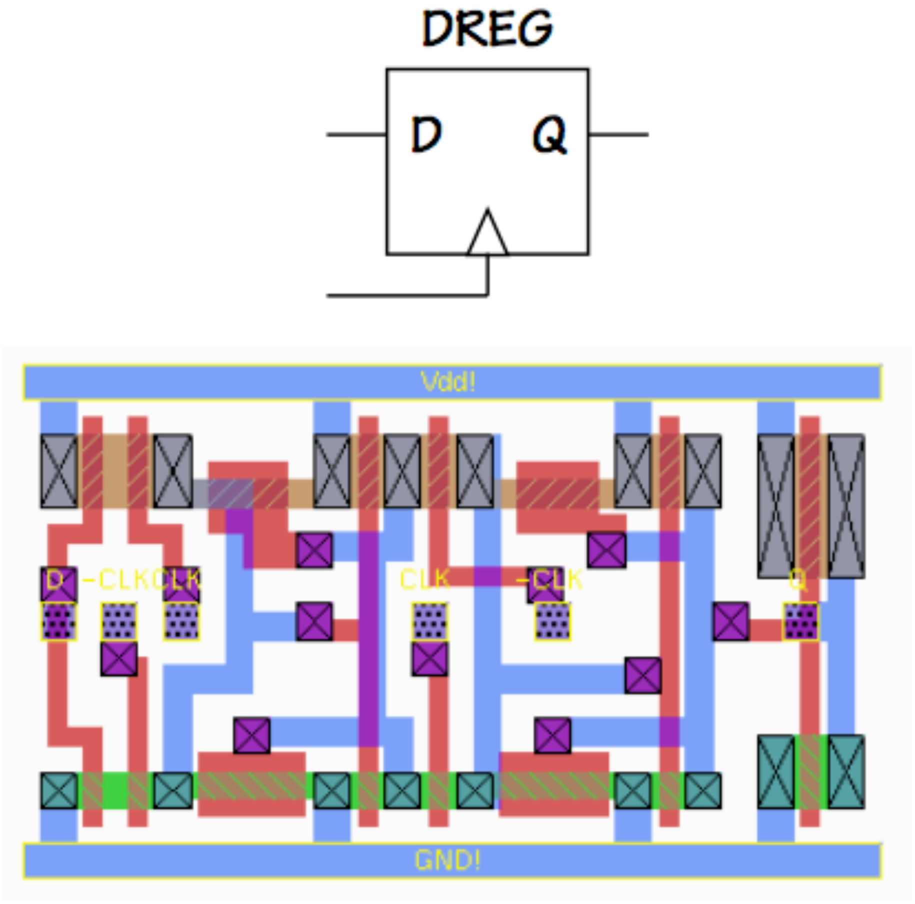
- Gate array
  - The last two layers of metal are used to define the function of the transistors
  - Side-by-side gates are isolated from one another by turning off the gate of a transistor between them



Target Technologies

- Standard cells
  - A library of fixed-pitch logic cells
    - Gates, registers, multiplexes, adders, I/O pads, etc.
    - Verified function, area, power, propagation delay, output rise/fall time as function of load, etc.





Technology-Independent Optimization

- Two-level Boolean minimization
  - Based on the assumption that a smaller and a faster implementation comes from
    - Reducing the number of product terms in an equation
    - Reducing the size of each product term
- Optimizing finite state machines
  - Look for equivalent FSMs that have fewer states
    - FSMs that produce the same outputs given the same sequence of inputs
- FSM state encodings that minimize implementation area
  - Size of state storage + size of logic to implement the next state and output forming logics

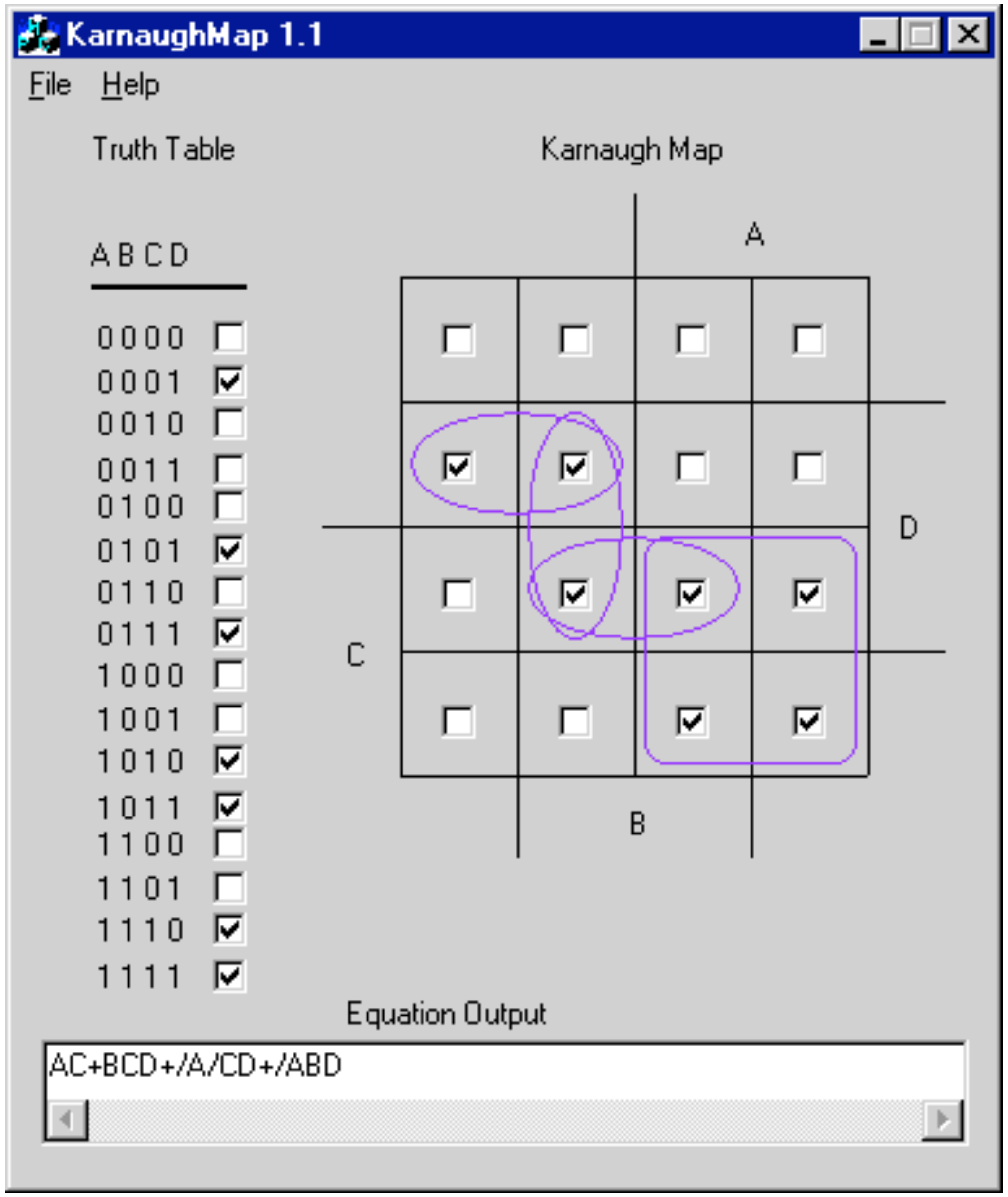
Technology-Independent Optimization

- None of these operations is completely isolated from the target technology
  - But experience has shown that it is advantageous to reduce the size of the problem as much as possible before starting the technology-dependent optimizations

Technology Independent Optimization

- Boolean minimization
  - Algebraic approach

    $\alpha\bar{\beta}+\alpha\beta=\alpha$
  - Karnaugh maps (k-map)
    - Minimization
      - Copy truth table into K-Map
      - Identify subcubes,
      - Select the largest available subcube
      - Write down the minimal SOP realization
    - Drawbacks
      - Only manageable for small circuits (4~5 inputs)
      - More better techniques for computers
      - SOP realizations are not all that relevant
      - Low fan-in gates are better suited to current technologies that SOP (FPGAs, Standard Cells)
      - Sometimes minimal circuits are glitchy
      - Some important circuits are not amenable to minimal SOP realizations



Truth Table

ABCD	
0000	□
0001	☑
0010	□
0011	□
0100	□
0101	☑
0110	□
0111	☑
1000	□
1001	□
1010	☑
1011	☑
1100	□
1101	□
1110	☑
1111	☑

Karnaugh Map

□	□	□	□
☑	☑	□	□
□	☑	☑	☑
□	□	☑	☑

Equation Output

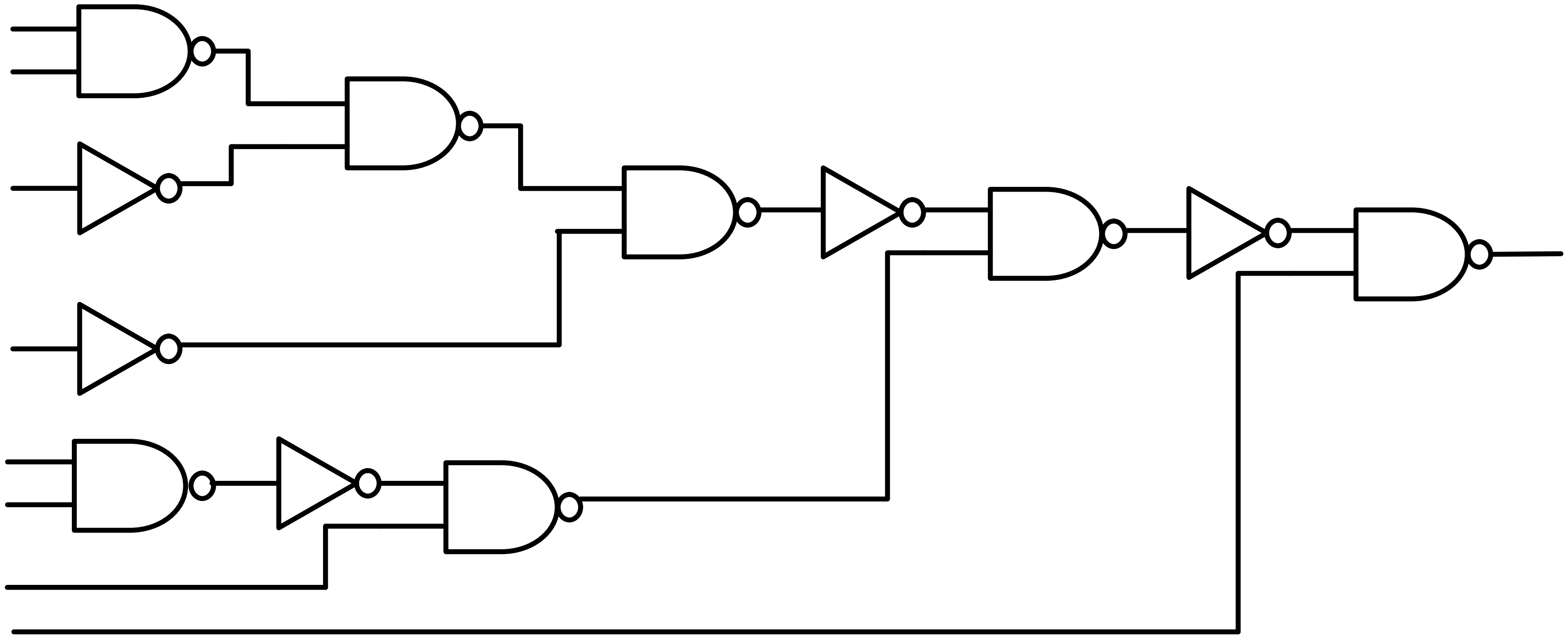
AC+BCD+/A/CD+/ABD

Technology Mapping

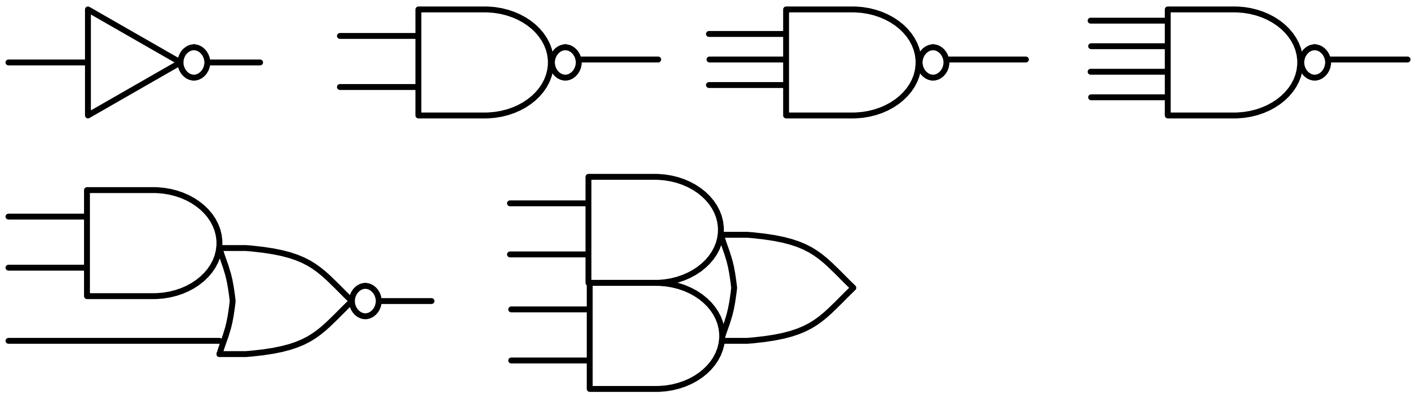
- After minimization of the logic equations, the next step is mapping each equation to the gates in our target gate library
  - DAG covering (K. Keutzer).
  - Represent input net list in normal form
    - Subject DAG: 2-input NAND gates + inverters
  - Represent each library gate in normal form
    - Primitive DAGs
- Goal
  - Find a minimum cost covering of the subject DAG by the primitive DAGs
  - If the subject and primitive DAGs are trees, there is an efficient algorithm (dynamic programming) for finding the optimum cover
  - Partition the subject DAG into a forest of trees
    - Each gate with fanout> 1 becomes root of a new tree
    - Generate the optimal solutions for each tree,
    - Stitch solutions together

Technology Mapping

- Problem statement
  - Subject DAG

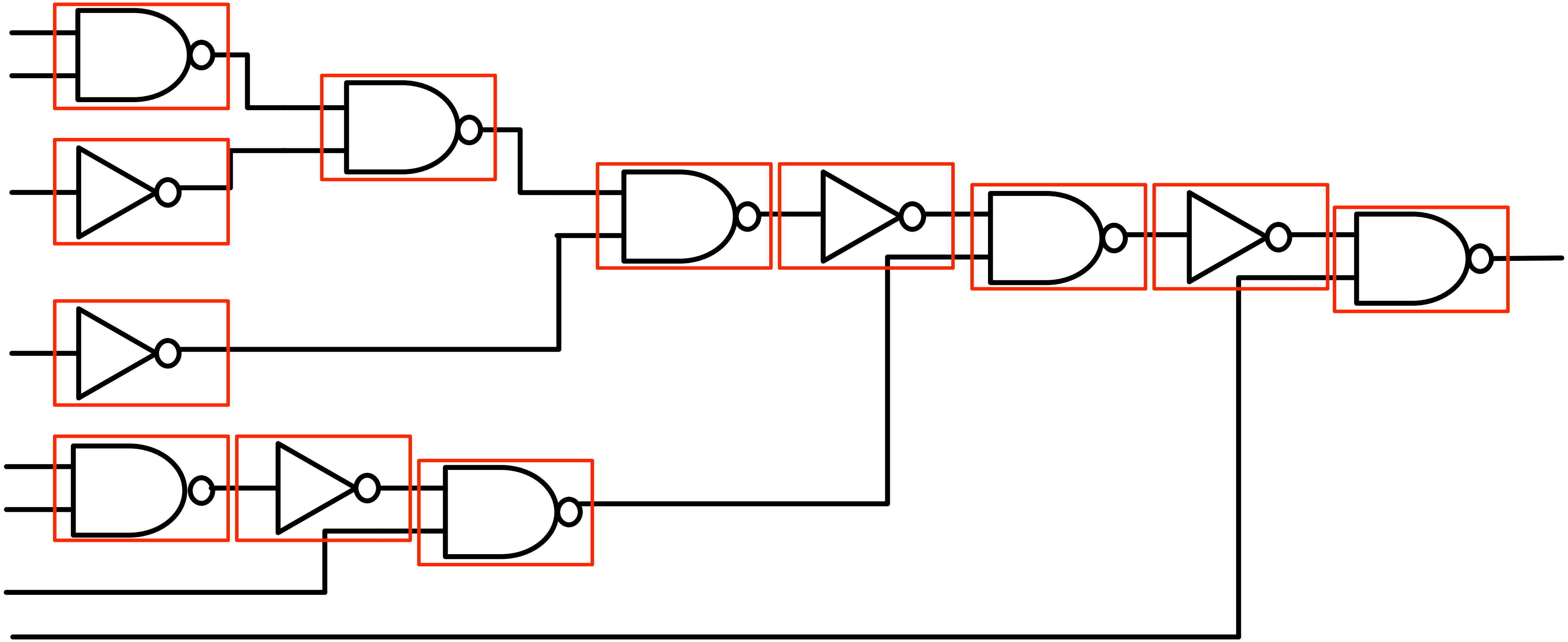


  - Primitive gate library



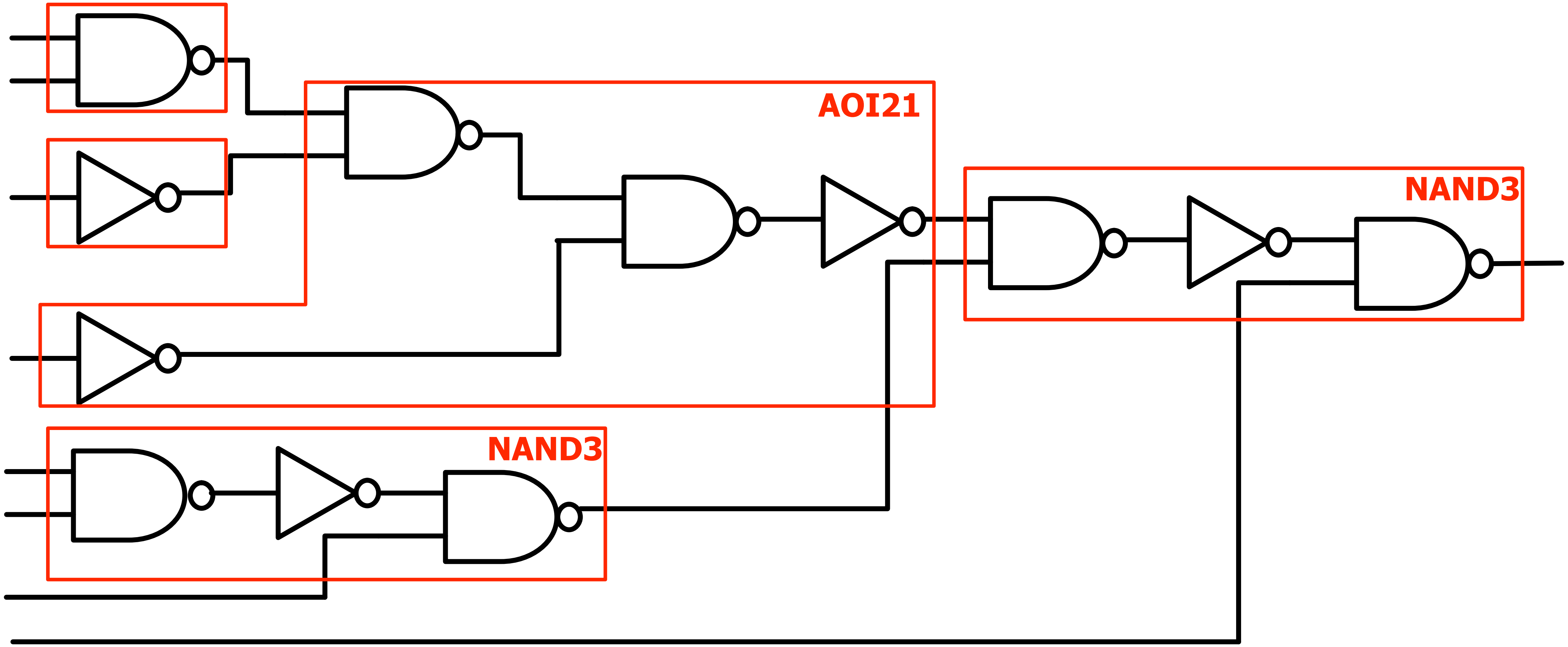
Technology Mapping

- Technology mapping
  - Trivial covering
    - Area: 7 NAND2 (3) + 5 INV (2) = 31



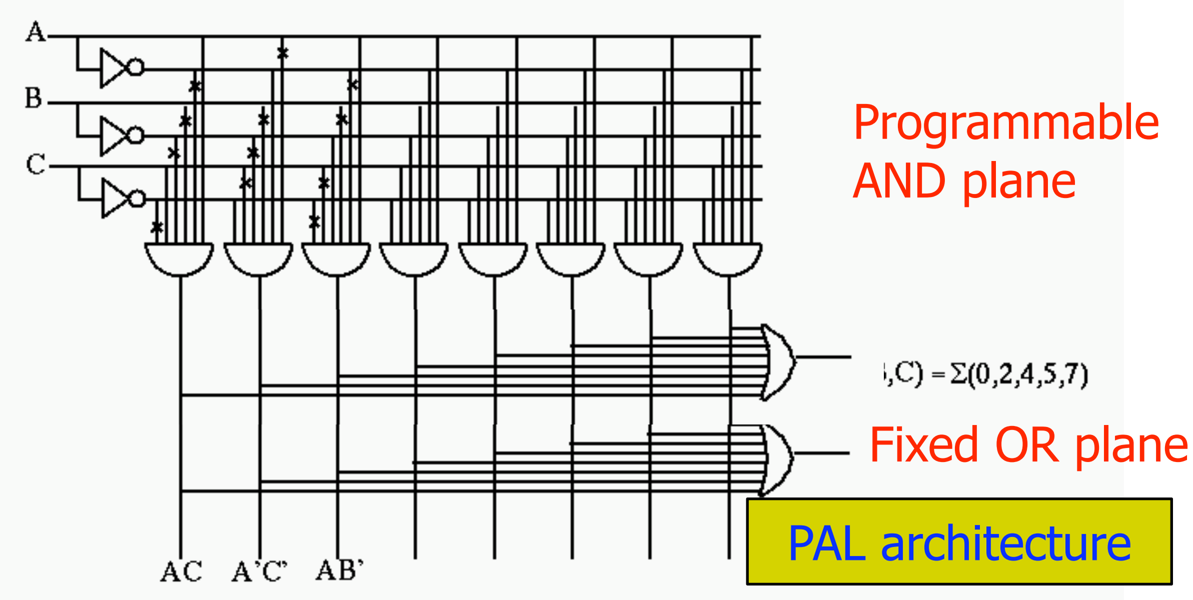
Technology Mapping

- Technology mapping
  - Optimal covering
    - Size: INV (2) + NAND2 (3) 2 + 2 NAND3 (8) + AOI21 (4) = 17



Technology Mapping

- Primitive libraries of technologies
  - 3 input PLA, ROM and PAL?
  - 3 input LUT?
  - 3 to 1 multiplexer?



Programmable AND plane

Fixed OR plane

PAL architecture

Placement and Route

- After placement and route (P&R)