OUTLINE

I - INTRODUCTION

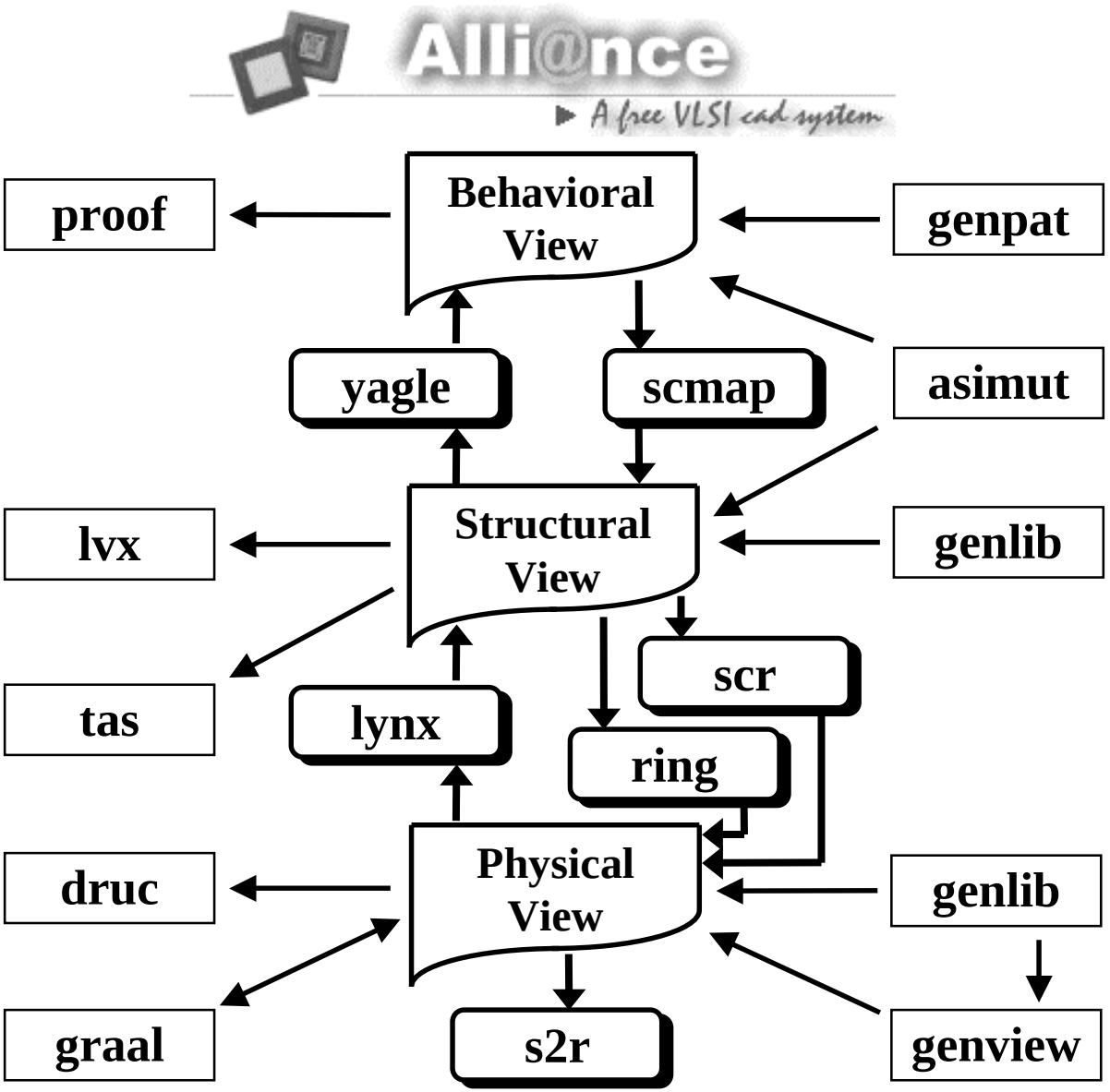
II - DESIGN METHODOLOGY: AN OVERVIEW

III - ABSTRACTION LEVELS IN ALLIANCE

IV - VHDL: A HARDWARE DESCRIPTION LANGUAGE

V - VHDL: THE ALLIANCE SUBSET

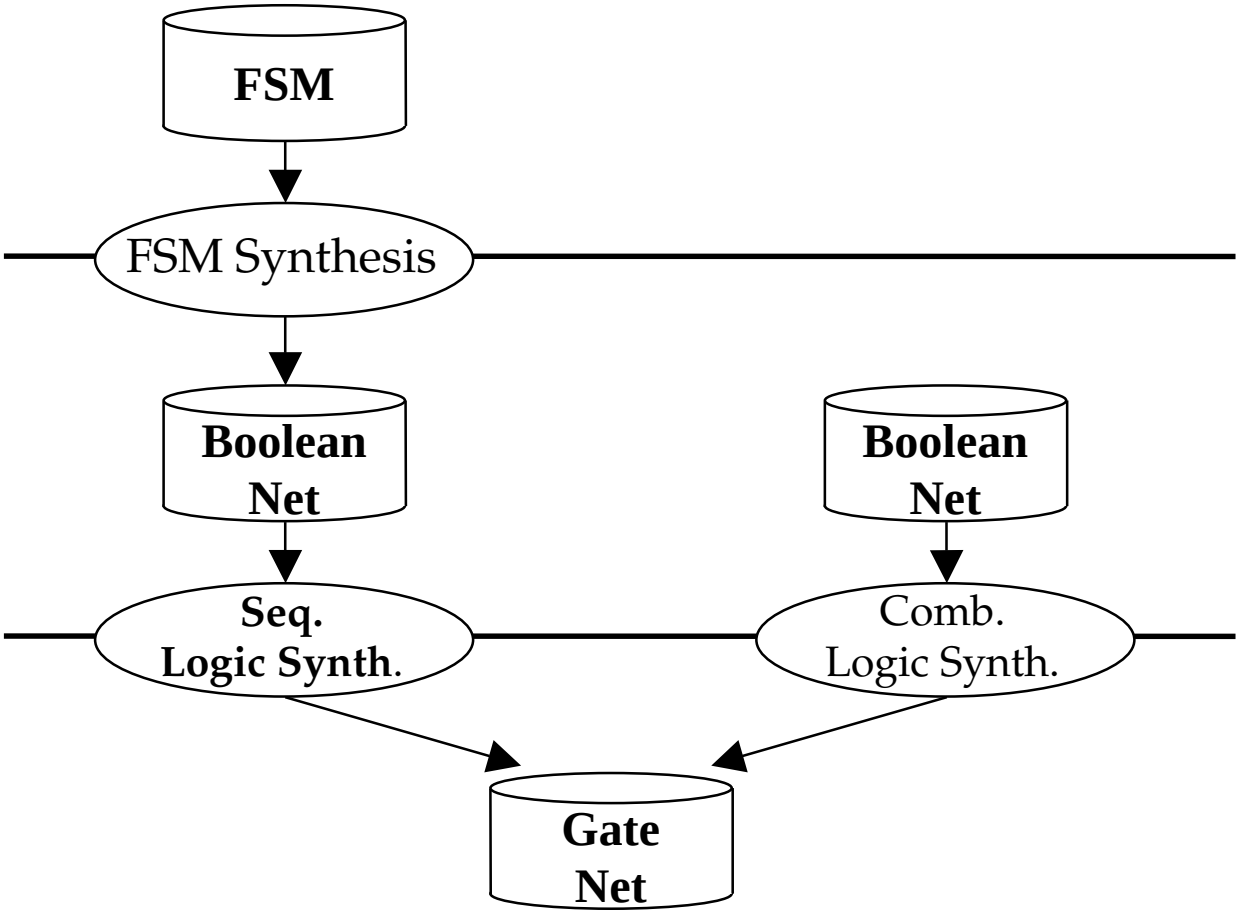
VI - ALLIANCE: A COMPLETE DESIGN SYSTEM


Alli@nce
A free VLSI cad system
Behavioral View
proof
genpat
yagle
scmap
asimut
Structural View
lvx
genlib
tas
lynx
scr
ring
Physical View
druc
genlib
graal
s2r
genview

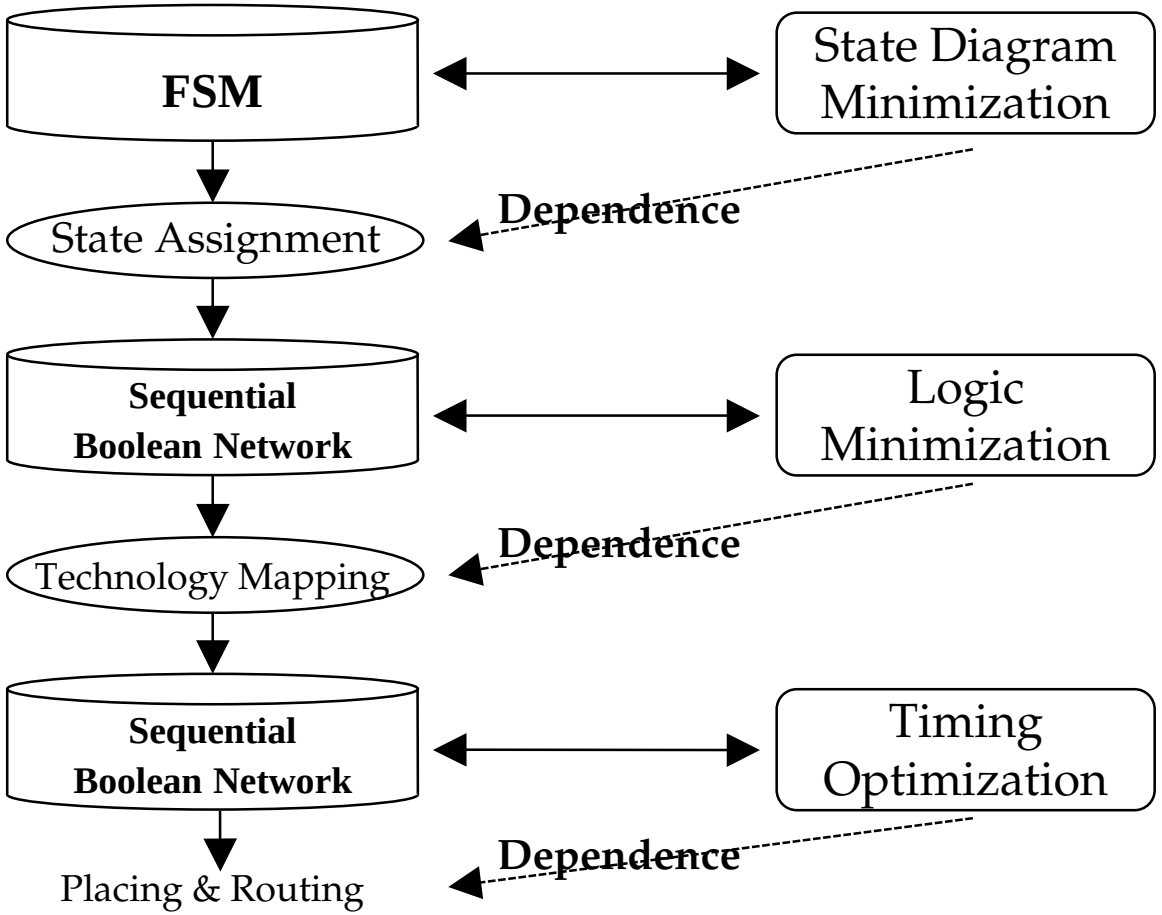
Synthesis Levels

- ✘ Architectural
- ✓ Finite State Machine
- ✓ Logic
- ✓ Layout

Synthesis Architecture



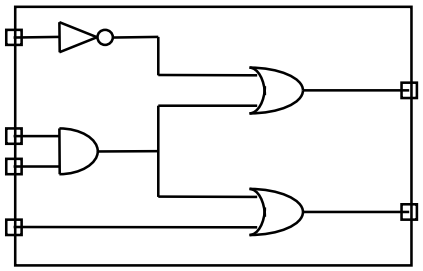
Optimization and Synthesis Flow



Logic Synthesis

```
ENTITY dec2to4 is
PORT(A,B,enable:in BIT;
        vdd,vss,vdde,vsse:in BIT;
        Y:out bit_vector(0 to 3));
end dec2to4;

architecture dflow of dec2to4 is

signal a_bar,b_bar:bit;
signal a1,a2,a3,a4:bit;

begin
   a_bar <= not a;
   b_bar <= not b;
```

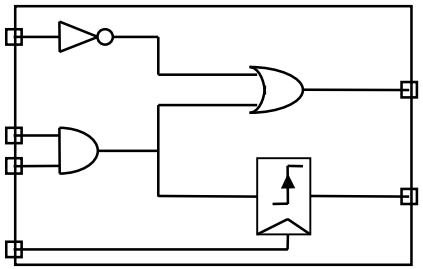
Combinational Logic Synthesis



Gate Netlist = Gate Level Structural Description

```
ENTITY adder is
PORT(A,B,enable:in BIT;
        vdd,vss,vdde,vsse:in BIT;
        ck: in bit;
        Y:out bit_vector(0 to 3));
end dec2to4;

architecture dflow of adder is

signal regstr:reg_vector(0 to 3)
                        register;
signal a1,a2,a3,a4:bit;

begin
```

Sequential Logic Synthesis



Gate Netlist = Gate Level Structural Description

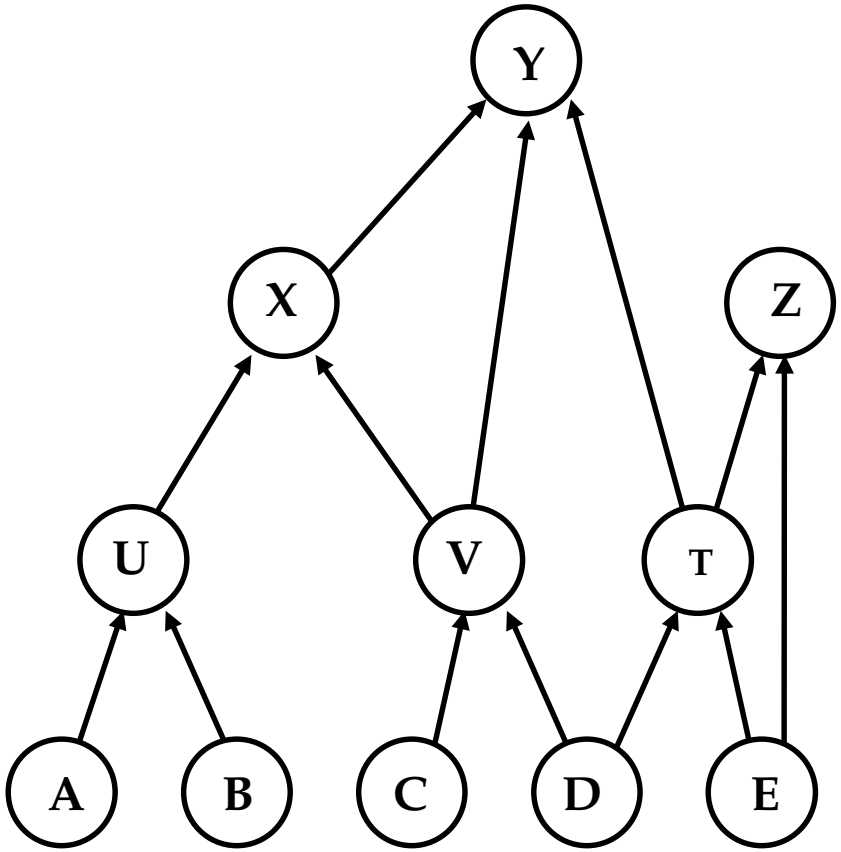
Optimization

Improve Description at Equivalent Level

$$\begin{cases} X = A + \overline{A}.C.D \\ Y = C.D \end{cases} \Rightarrow \begin{cases} X = A + Y \\ Y = C.D \end{cases}$$

Representation

Boolean Network



Binary Decision Diagram (BDD) (1)

Based on the Shannon Theorem

$$F(X_1, X_2, \ldots, X_n) =$$
$$\overline{X_1}.F(0, X_2, \ldots, X_n) + X_1.F(1, X_2, \ldots, X_n)$$

✓ Canonical Representation of a Boolean Equation

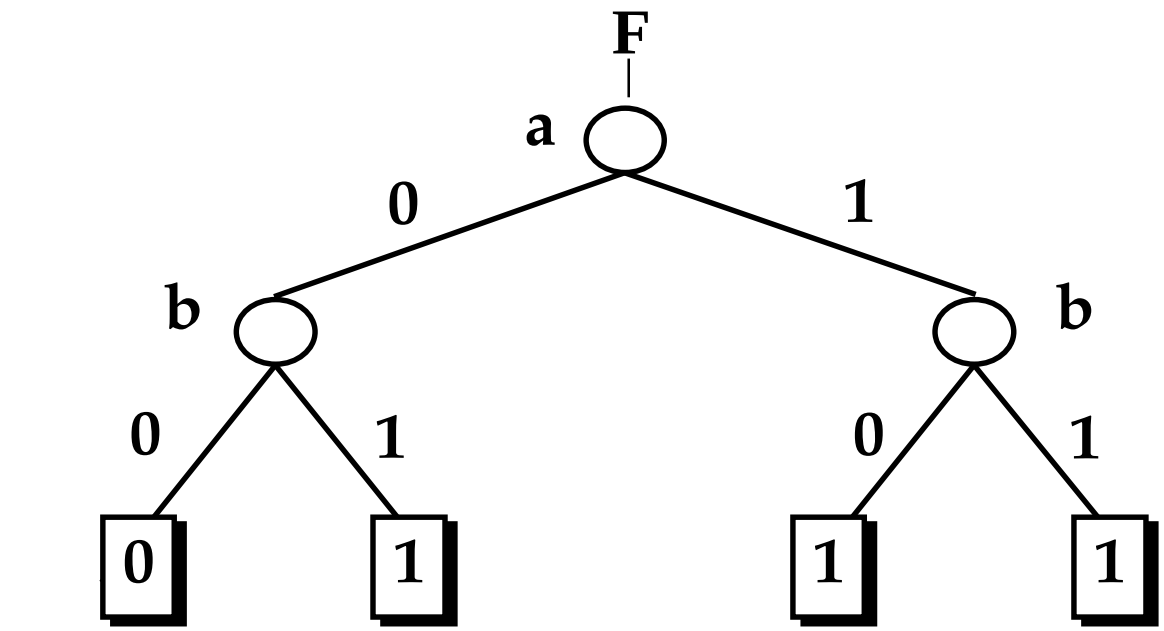
Binary Decision Diagram (BDD) (2)

Example: F(a, b) = a + b

$$
\begin{aligned}
F &= \overline{a}.F(0,b) + a.F(1,b) \\
&= \overline{a}.(0+b) + a.(1+b) \\
&= \overline{a}.(b) + a.(1) \\
&= \overline{a}.\left(\overline{b}.(0) + b.(1)\right) + a.\left(\overline{b}.(1) + b.(1)\right)
\end{aligned}
$$

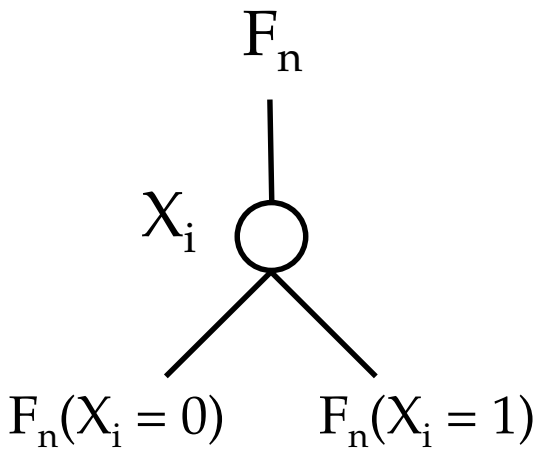
Binary Decision Diagram (BDD) (3)

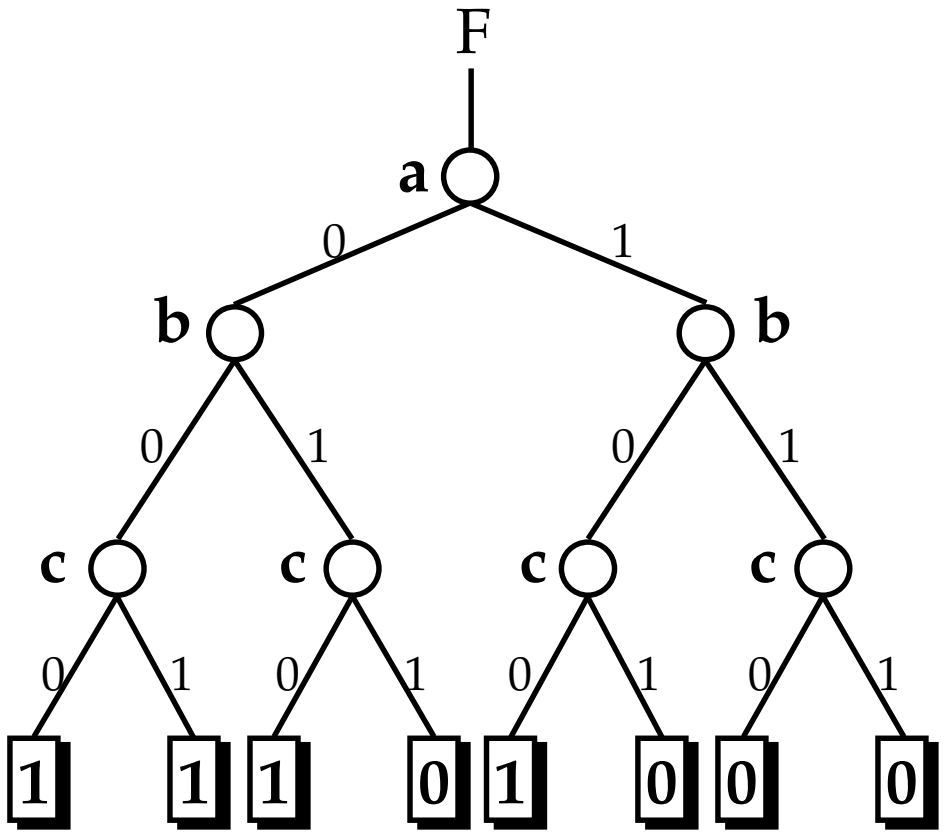
$$F = \overline{a}.\left(\overline{b}.(0) + b.(1)\right) + a.\left(\overline{b}.(1) + b.(1)\right)$$



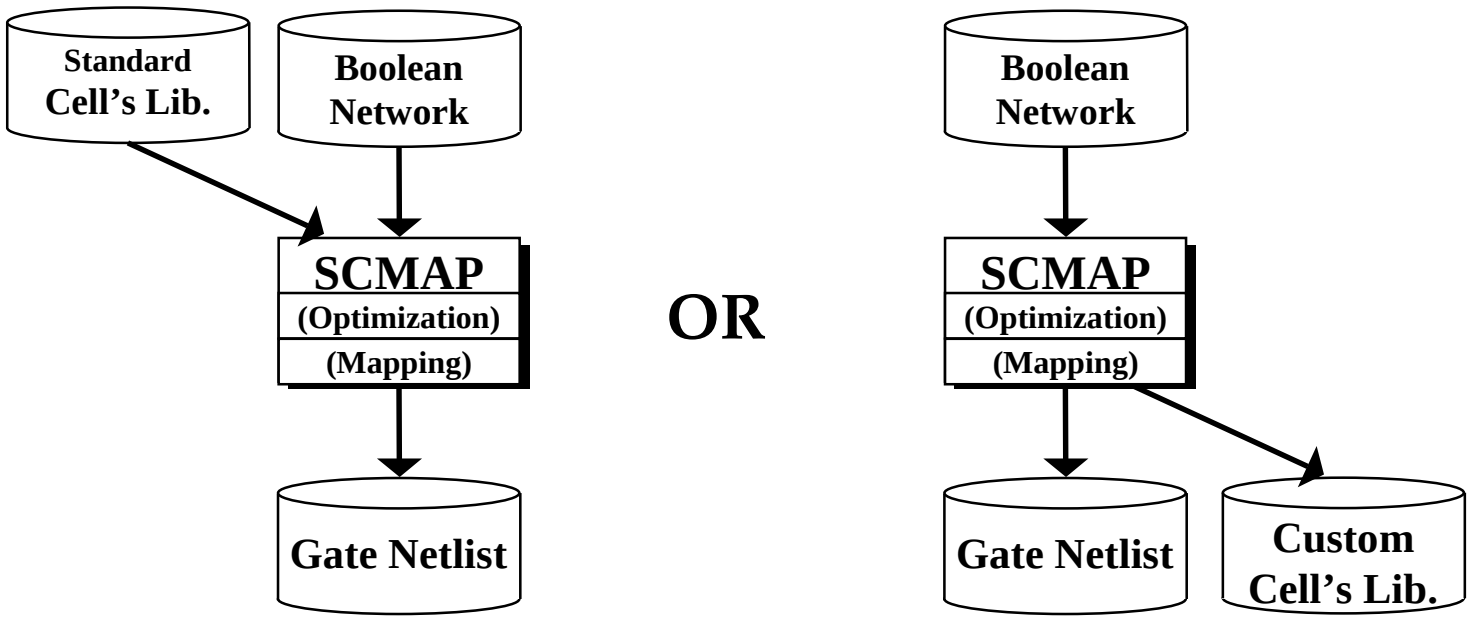
Binary Decision Diagram (BDD) (3)

Ex: $F = \bar{a}.\bar{b} + \bar{a}.b.\bar{c} + a.\bar{b}.\bar{c}$



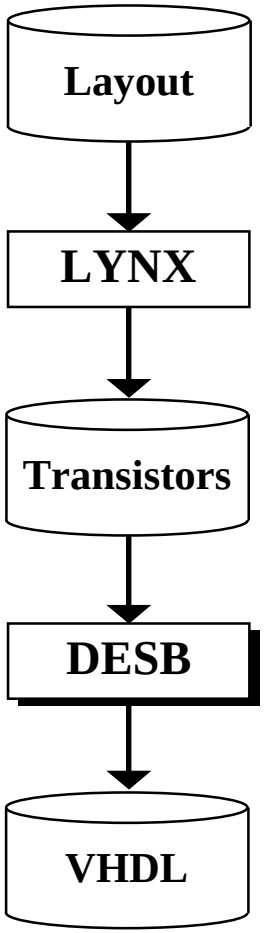


Logic Synthesizer

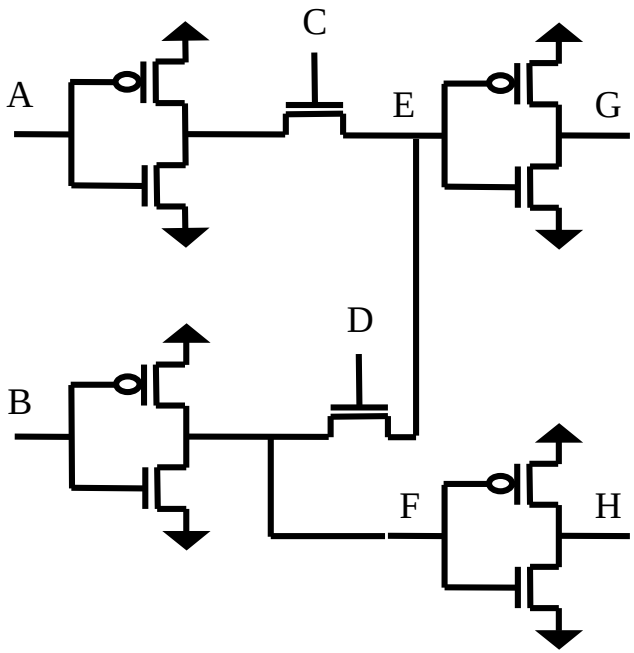


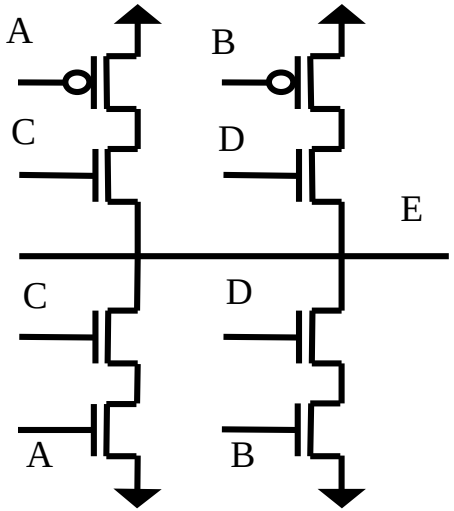
YAGLE: Functional Abstractor (1)

- ✓ Generates behavioral data flow VHDL
- ✓ Provides Functional Verifications
- ✓ Does not use any cell library
- ✓ Accepts standard transistor netlist format (VTI, SPICE)

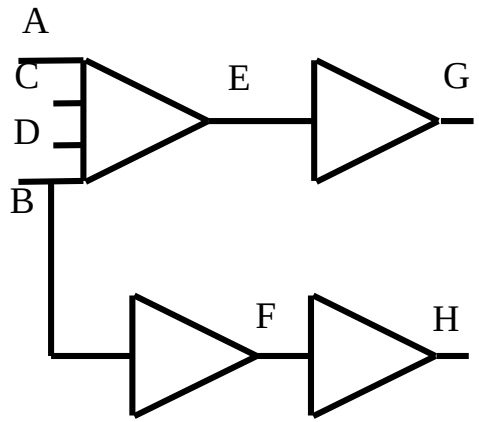


YAGLE: Functional Abstractor (2)





THE GATE 'E'



E <= (NOT A AND C) OR (NOT B AND D);

H <= NOT F;

G <= NOT E;

F <= NOT B;

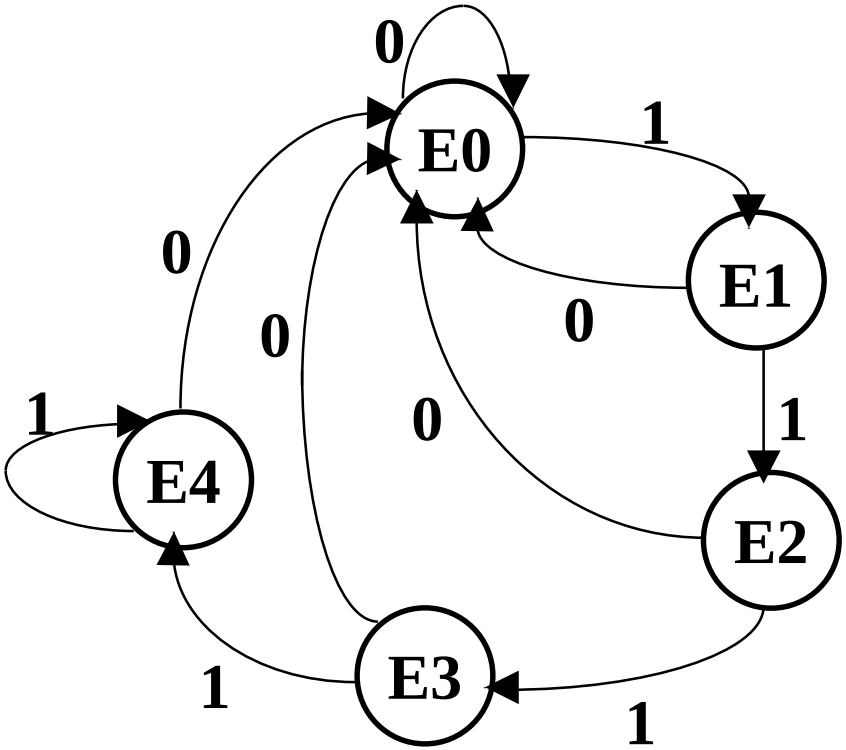
FSM: Finite State Machine (1)

- Models Sequential Circuits
- Two Kinds of FSM
- Graph Representation
- Definition:

$$State(t+1) <= F(i_1,...,i_n,State(t))$$
$$Output_i <= F(i_1,...,i_n,State(t))$$

FSM: Finite State Machine (2)

Example: Four Consecutive Ones Counter



FSM: Finite State Machine (3)

The Description Language

- Standard
- VHDL Subset
- The States are Enumerated Types
- Two Special Signals
- Two Processes

FSM: The Four Consecutive Ones Counter (1)

```
--
Entity counter is port (ck, I, reset: in bit; O: out bit);
End counter;

Architecture automate of counter is
type STATE_TYPE is (E0, E1, E2, E3, E4);
signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
-- pragma CUR_STATE CURRENT_STATE;
-- pragma NEX_STATE NEXT_STATE;
-- pragma CLOCK ck;

begin
    Process(CURRENT_STATE, I, reset)
    begin
        if (reset = '1') then
            NEXT_STATE <= E0;
            O <= '0';
        else
```

FSM: The Four Consecutive Ones Counter (2)

```
case CURRENT_STATE is
        WHEN E0 =>
                if (I='1') then
                        NEXT_STATE <= E1;
                else
                        NEXT_STATE <= E0;
                end if;
                O <= '0';

        WHEN E1 =>
                if (I='1') then
                        NEXT_STATE <= E2;
                else
                        NEXT_STATE <= E0;
                end if;
                O <= '0';
```

FSM: The Four Consecutive Ones Counter (3)

```
WHEN E2 =>
        if (I='1') then
                NEXT_STATE <= E3;
        else
                NEXT_STATE <= E0;
        end if;
        O <= '0';

WHEN E3 =>
        if (I='1') then
                NEXT_STATE <= E4;
        else
                NEXT_STATE <= E0;
        end if;
        O <= '0';
```

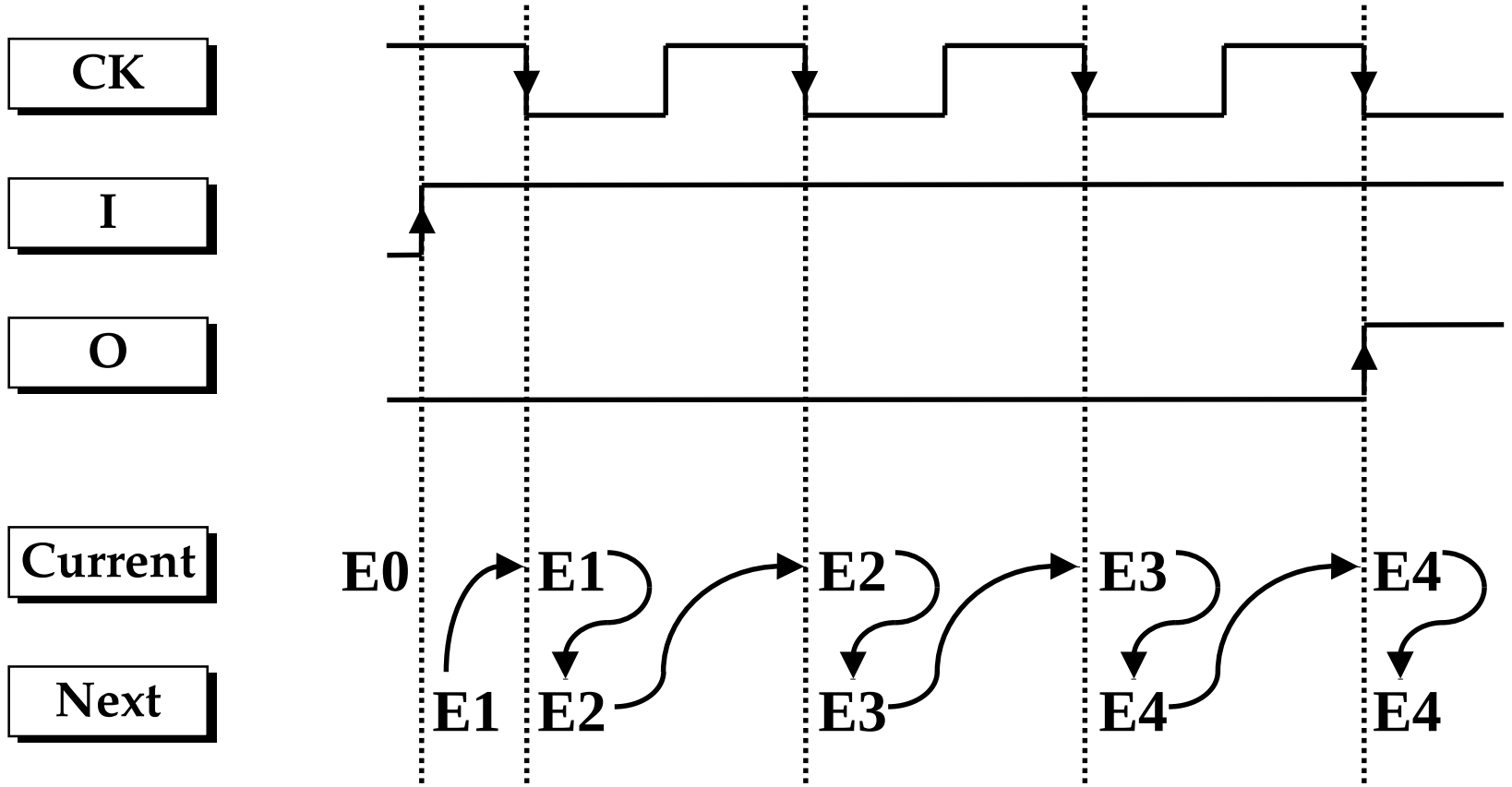
FSM: The Four Consecutive Ones Counter (4)

```
                WHEN E4 =>
                        if (I='1') then
                                NEXT_STATE <= E4;
                        else
                                NEXT_STATE <= E0;
                        end if;
                        O <= '1';

                WHEN others =>
                        assert ('1')
                        report "Illegal State";

        end case;
    end if;
end process;
```

FSM: The Four Consecutive Ones Counter (5)

```
    Process (ck)
    begin
        if (ck = '0' and not ck'stable) then
                CURRENT_STATE <= NEXT_STATE;
        end if;
    end process;
end counter;
```

FSM: The Four Consecutive Ones Counter (6)



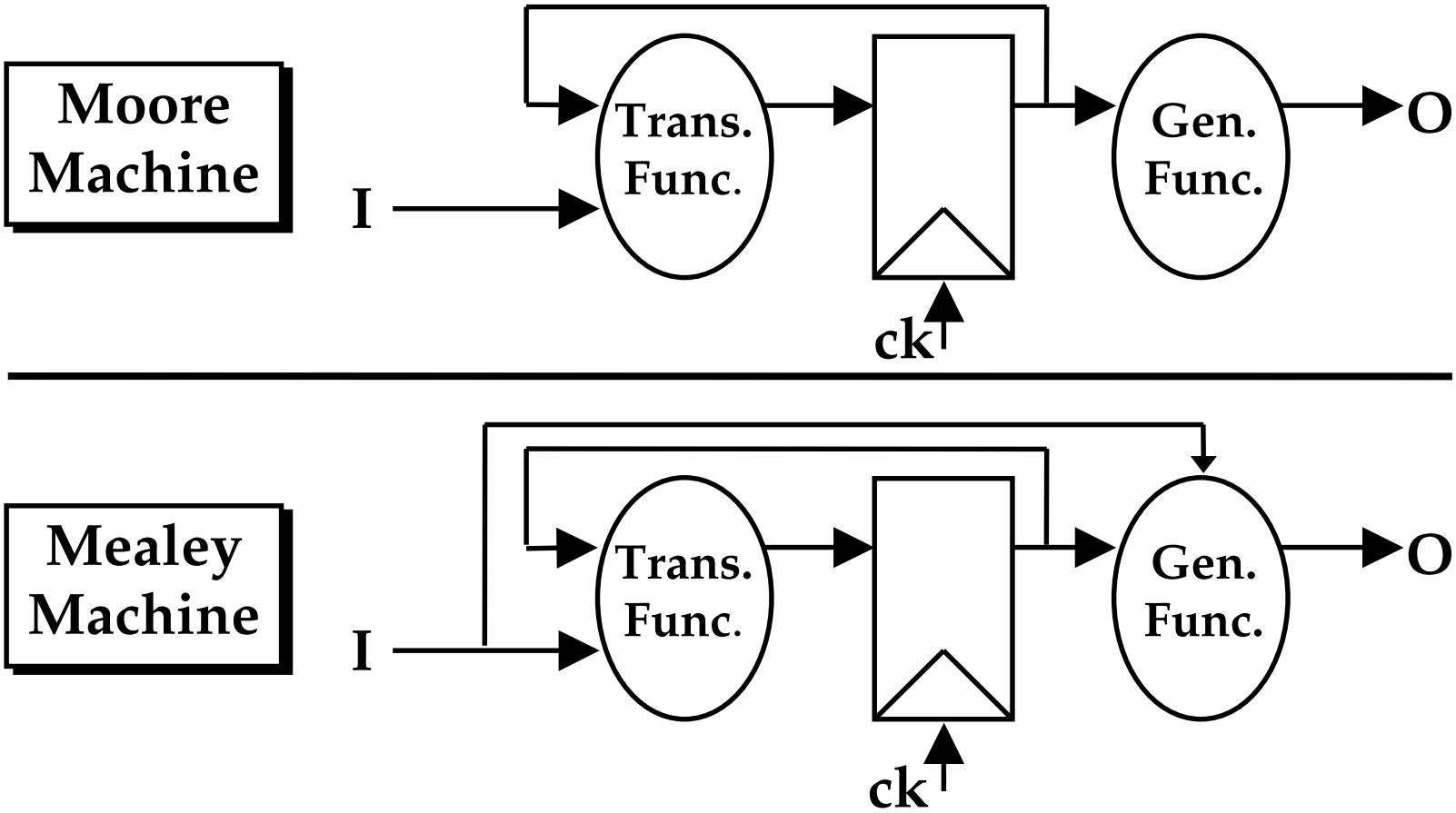
FSM: Warning (1)

All Signals which are Assigned to within a Clocked Process Have Registers on their Outputs

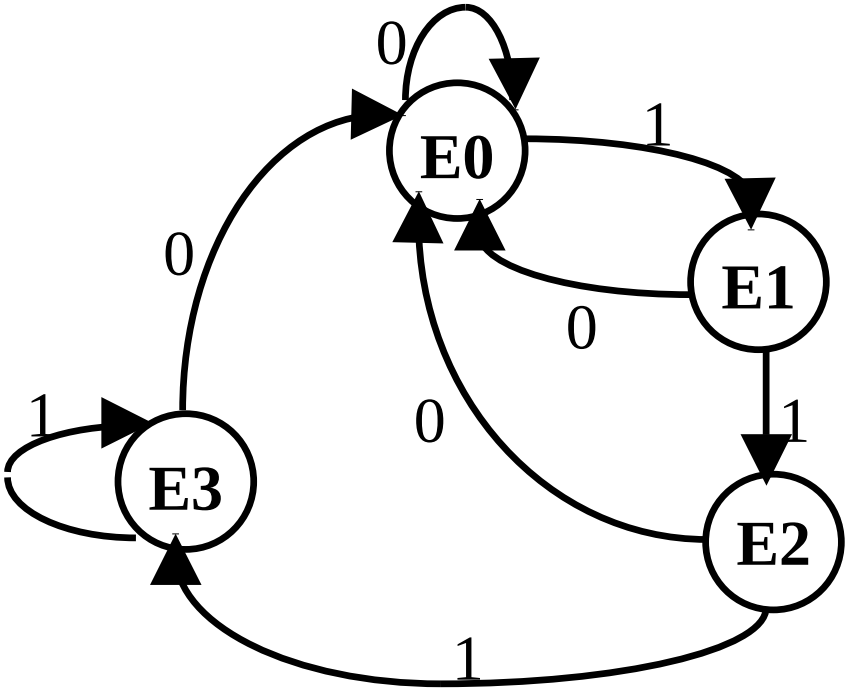
FSM: Warning (2)

Signal Assignments within a Process are Effective only before the wait (implicit or explicit) Statement

FSM: Different Machine Types

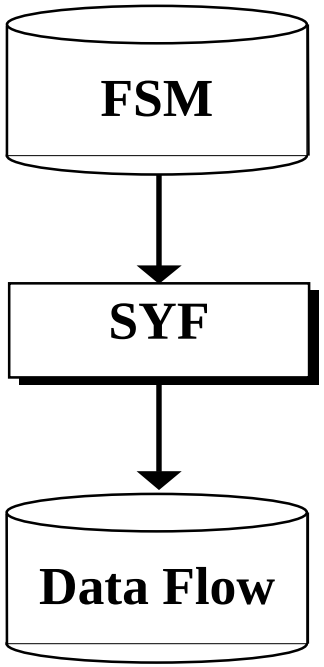


FSM: The Counter with Mealey Machine

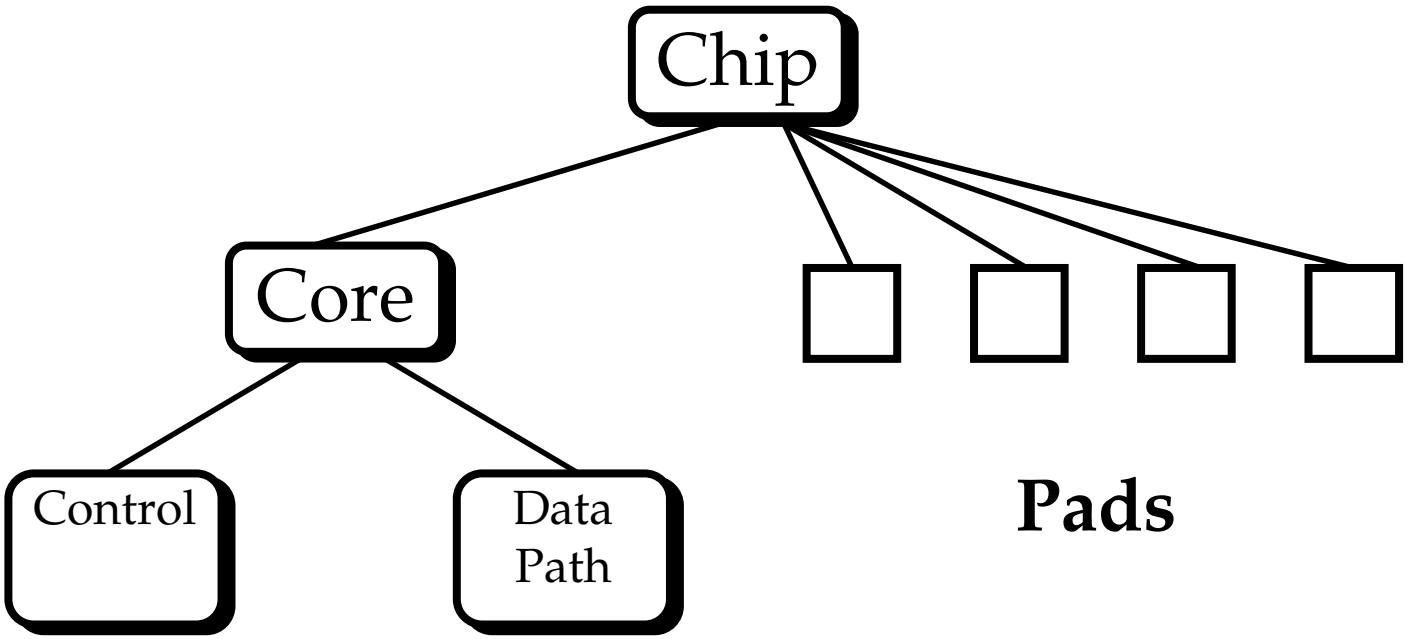


SYF: FSM Synthesizer

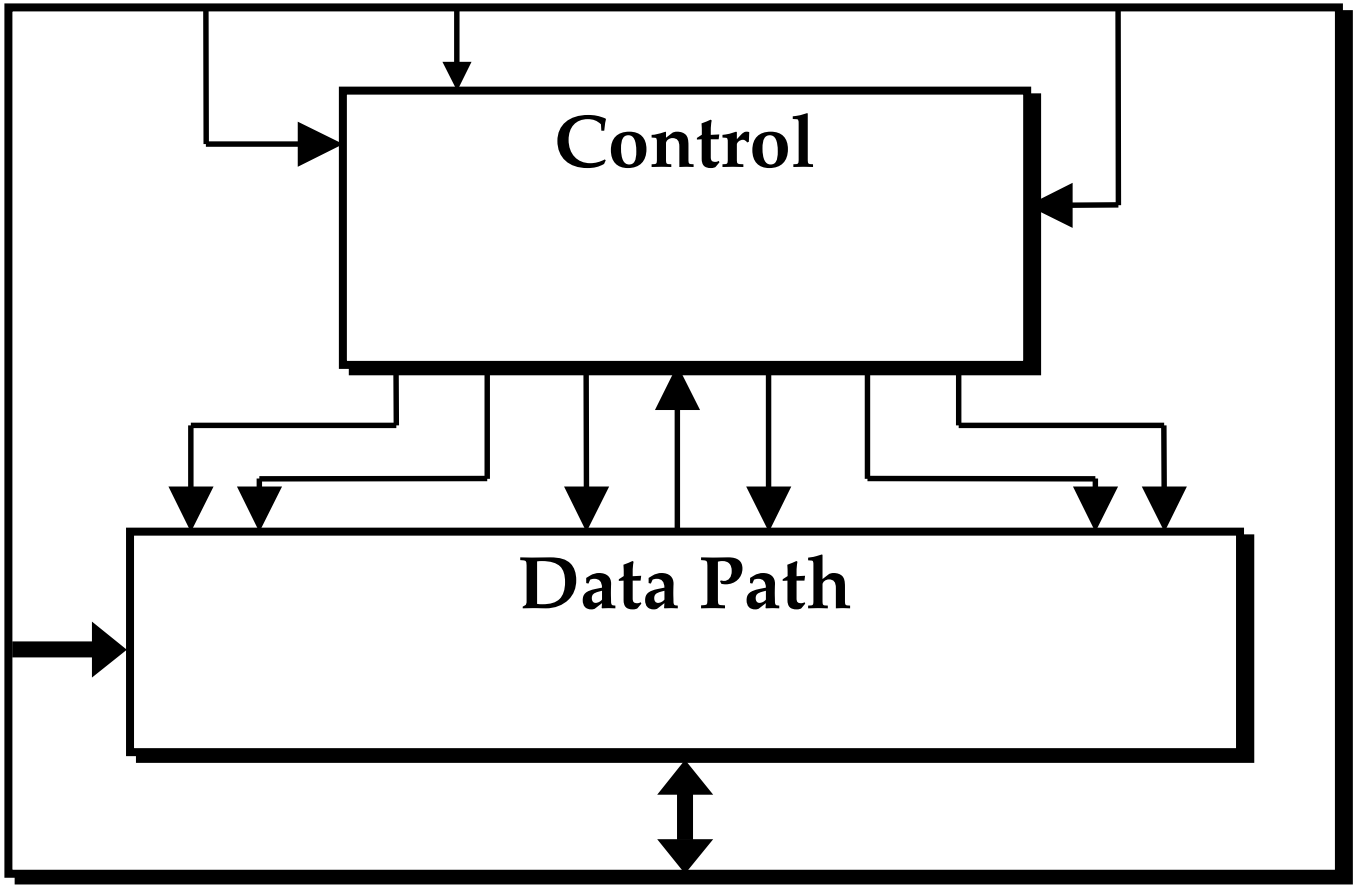
- ✓ Verification
- ✓ Encoding
- ✓ Optimization
- ✓ Driving Data Flow Description



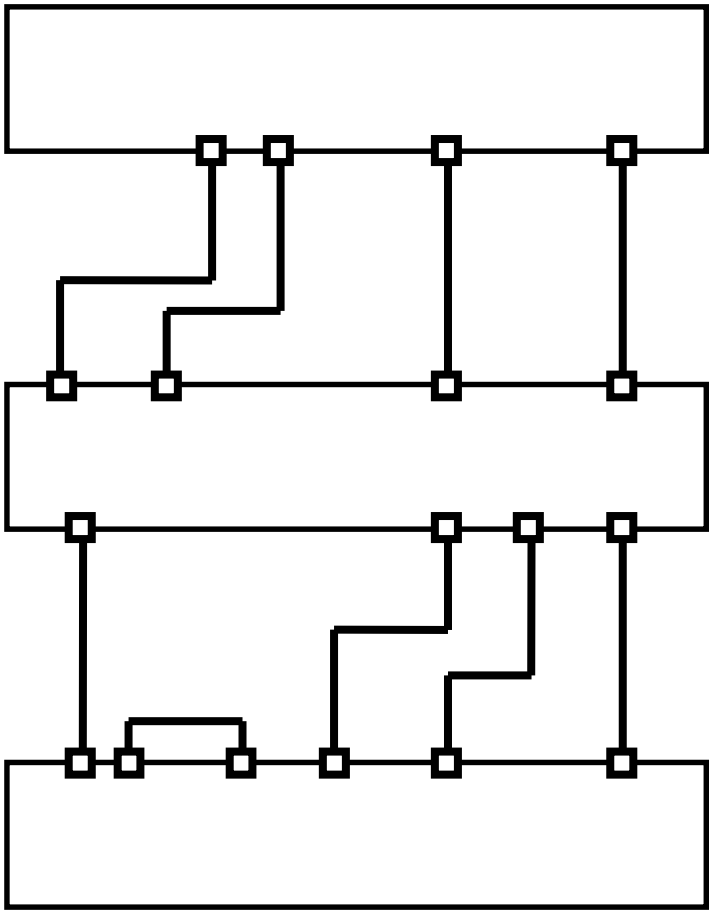
DATA PATH (1)

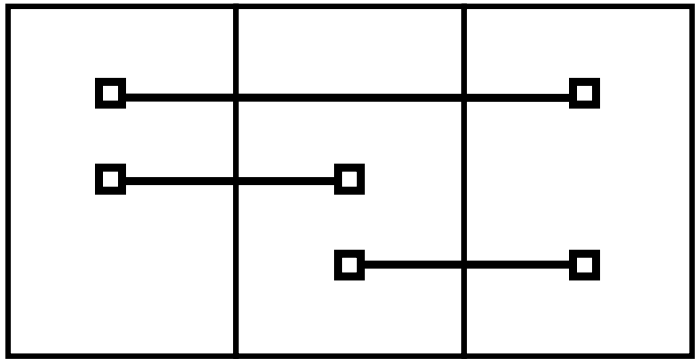


DATA PATH (2)



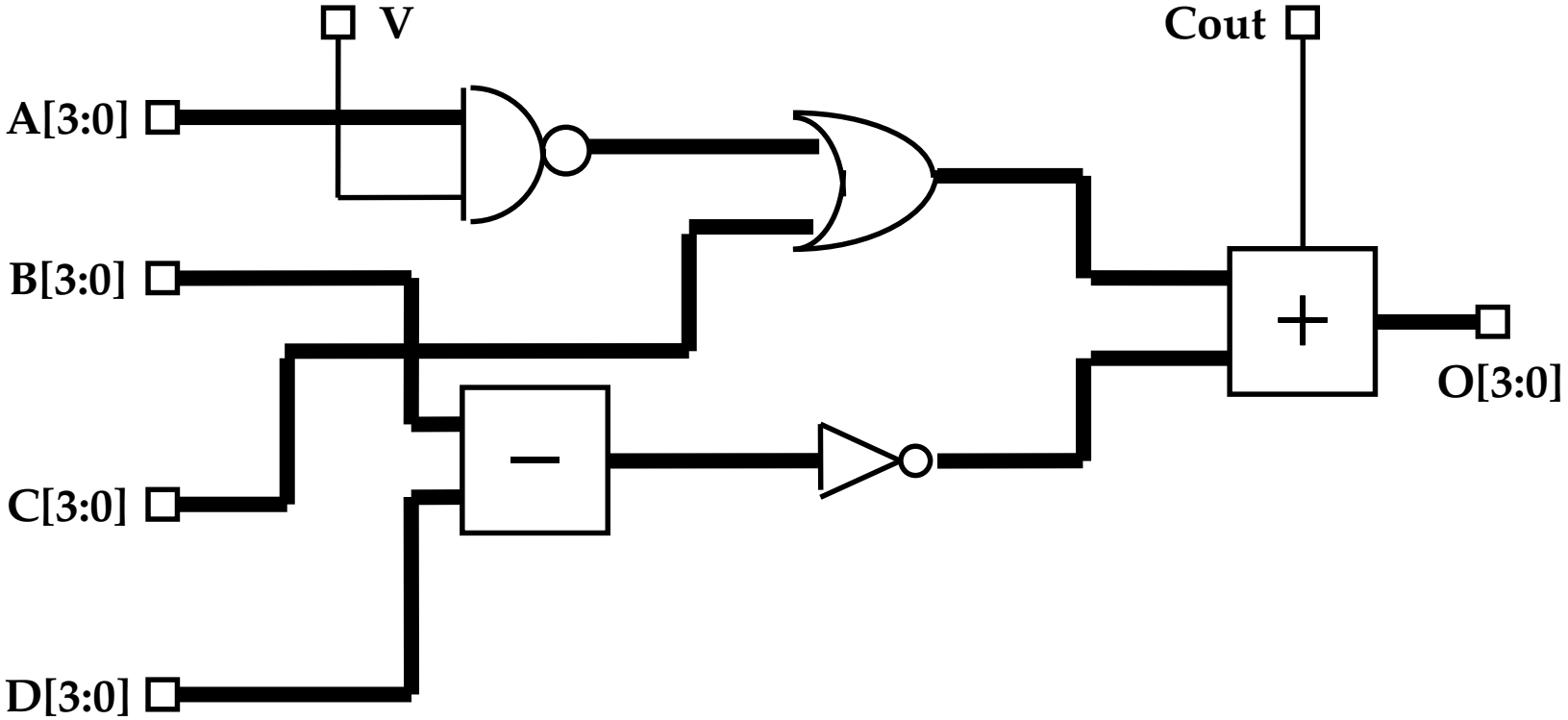
DATA PATH (3)



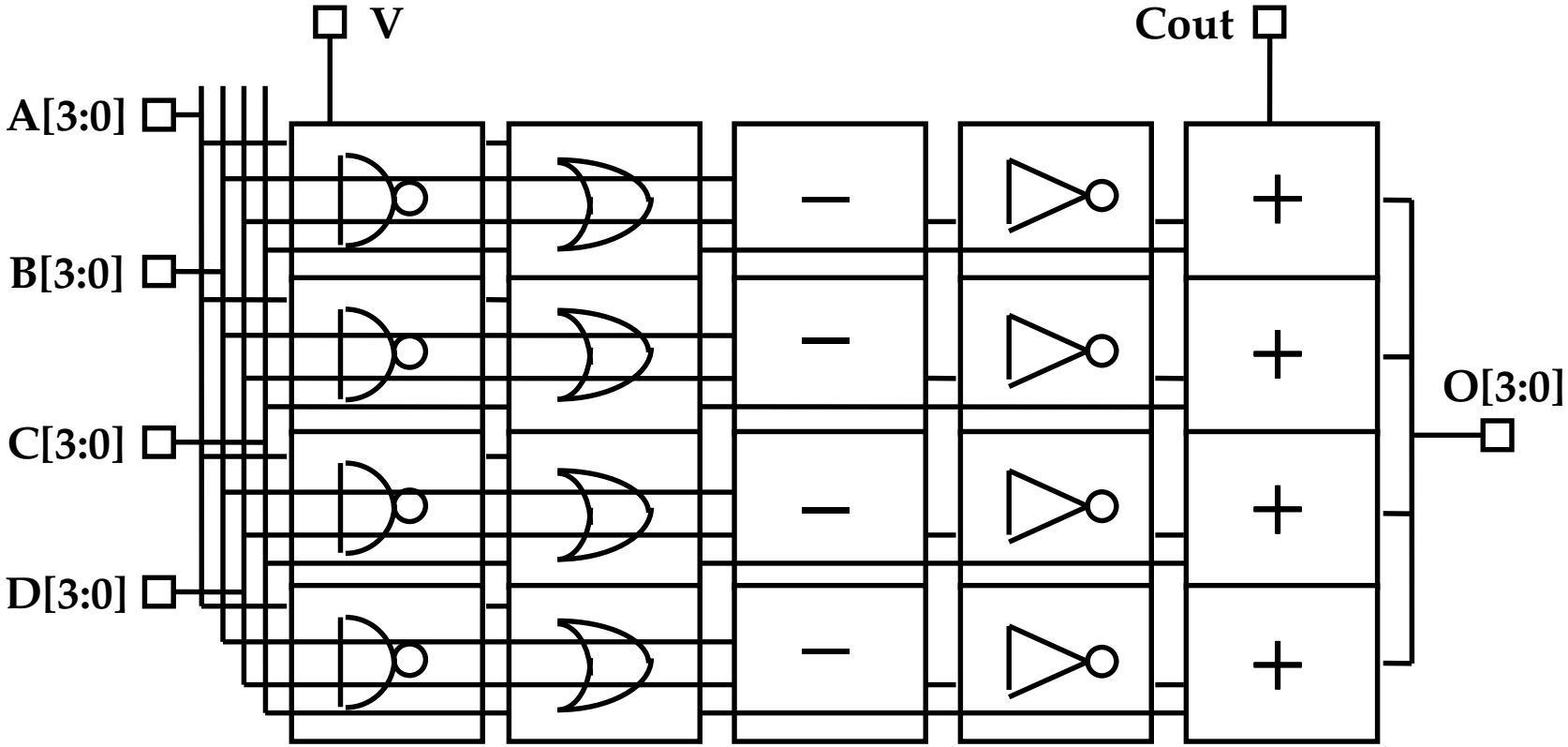


Standard Cells Router
Versus
Data Path Router

DATA PATH (4)



DATA PATH (5)

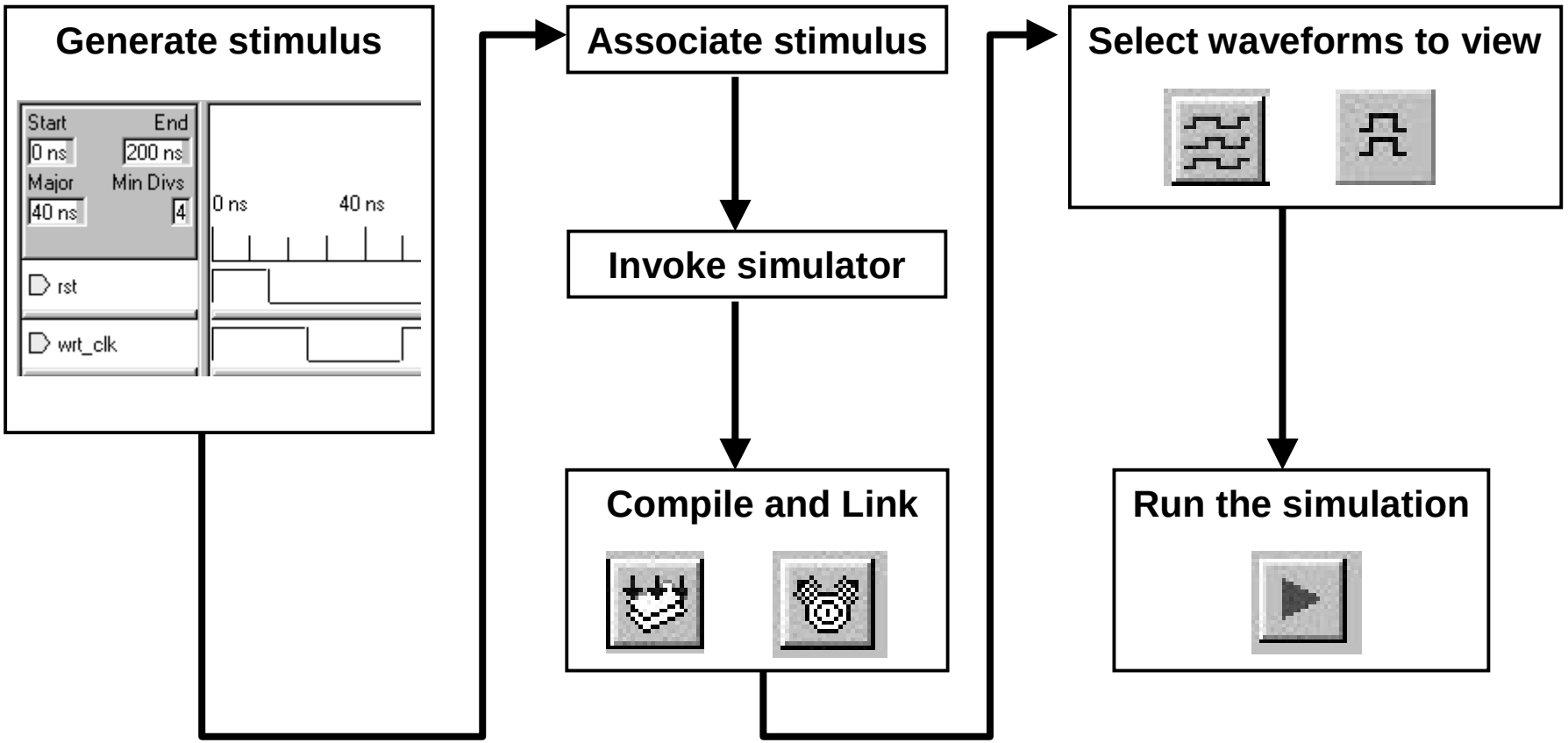


Timing

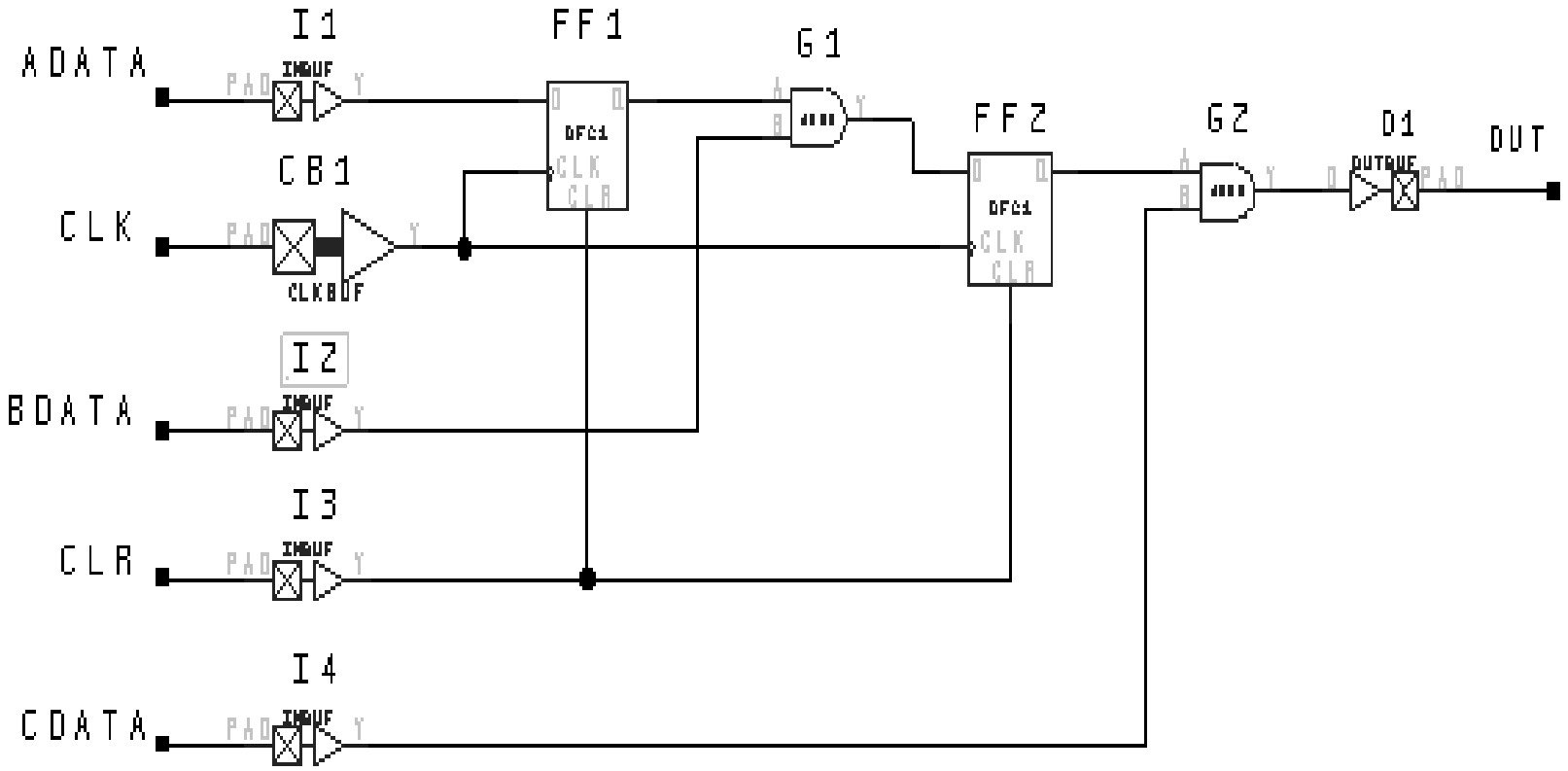
- **Simulators**
  - **Circuit-Level**
  - **Timing**
  - **Switch-Level**
  - **Logic-Level**
- **Verifiers (Pattern Independent)**
  - **Timing**

Simulation Design Flow

For each block you want to simulate . . .



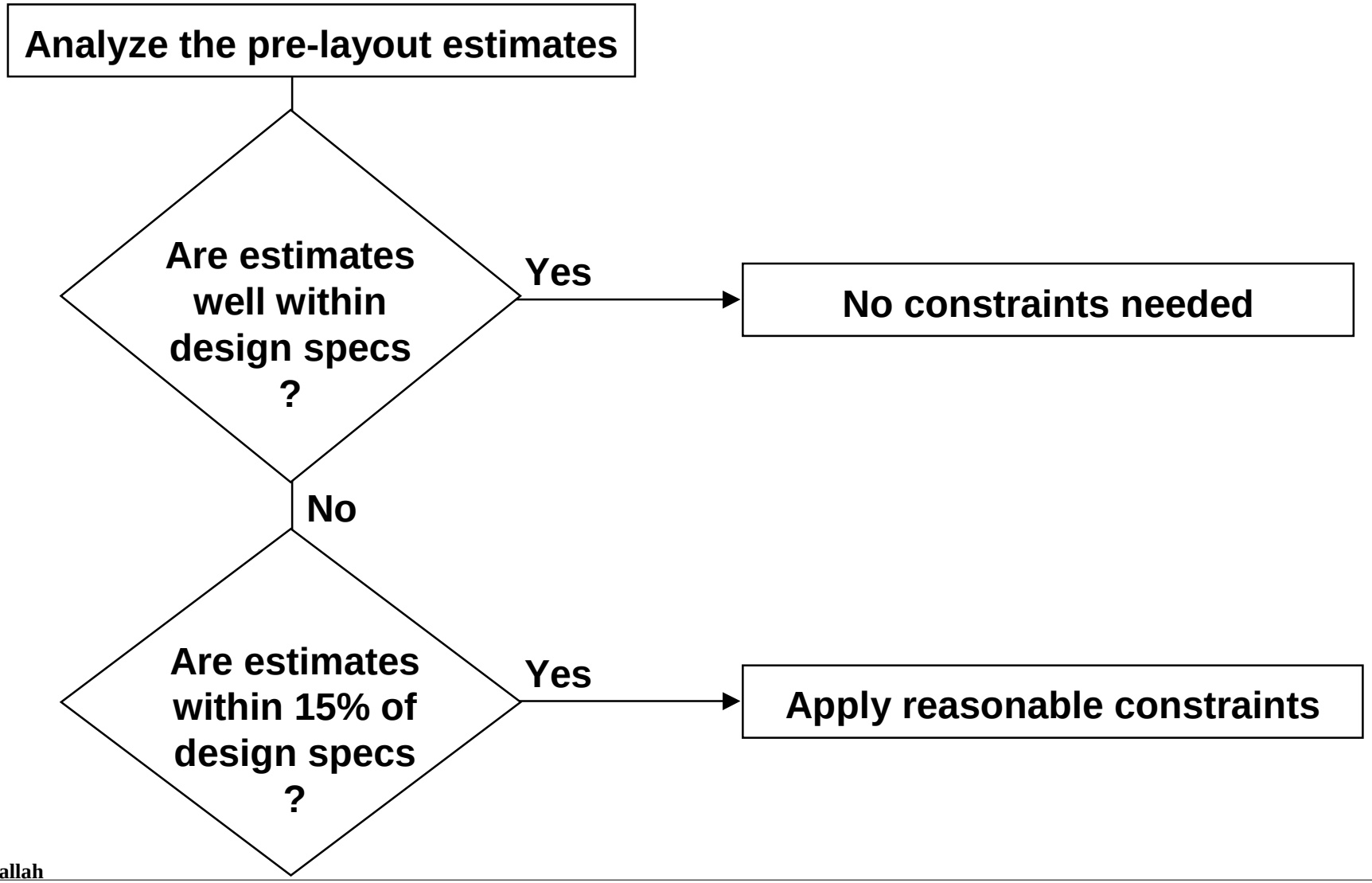
Timing


I1
FF1
G1
ADATA
IMBUF
DFC1
CLK
CLR
CB1
CLK
CLKBUF
FF2
G2
O1
DUT
OUTBUF
I2
BDATA
IMBUF
I3
CLR
IMBUF
I4
CDATA
IMBUF

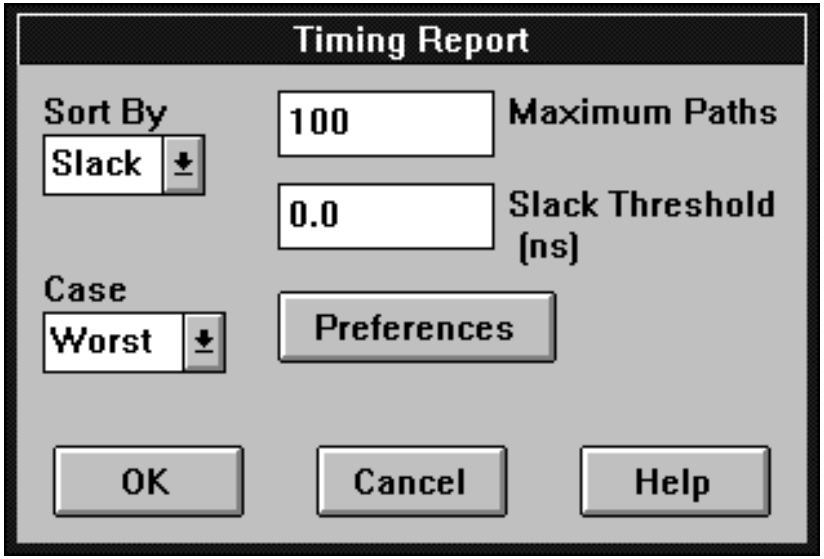
Timing

- **Timing Driven Synthesis**
- **Timing Driven Combiner**
- **Timing Driven Floor-Planner**
- **Timing Driven Place & Route**

Timing


Analyze the pre-layout estimates
Are estimates well within design specs ?
Yes
No constraints needed
No
Are estimates within 15% of design specs ?
Yes
Apply reasonable constraints

Timing




```
--------- Paths from Constraints on <GLOBAL_CLOCKS> ---------

 Actual  Needed   Slack (ID)   Source Pin(Net)  Sink Pin(Net)
=============================================================
  17.1   15.0    -2.1 (WFM0)    FF1:CLK(N35)     FF1:D(N8)
  17.1   15.0    -2.1 (WFM0)    FF1:CLK(N35)     FF2:D(N10)
  17.1   15.0    -2.1 (WFM0)    FF2:CLK(N35)     FF1:D(N8)
```