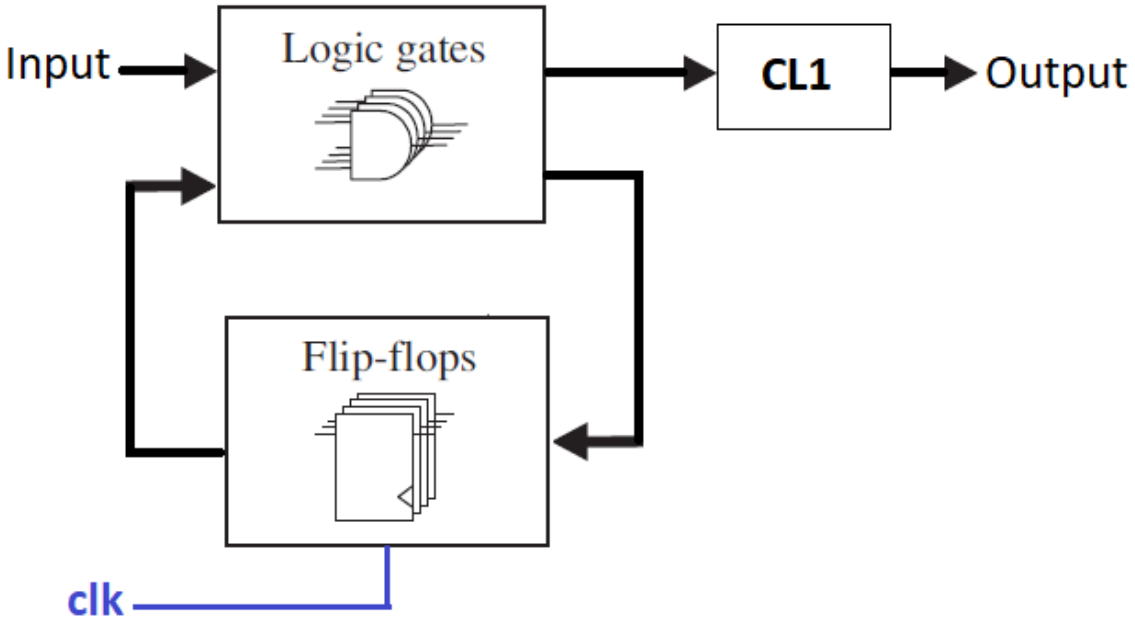
# *FSM based Applicable digital design*

# FSM based Applicable digital design

Applicable digital design are mostly based on FSM forms of ***Moore*** or ***Mealy with stored output*** due to the next reasons:

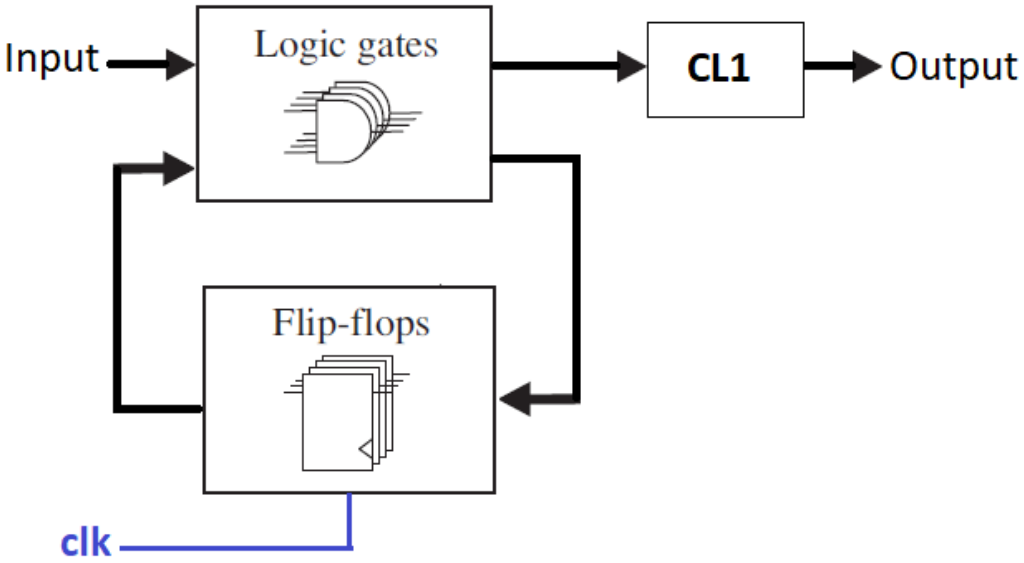
***Reason 1:*** Robustness from the influence of the parasitic logic feedback surrounding the upper section (FSM combinational logic part) .

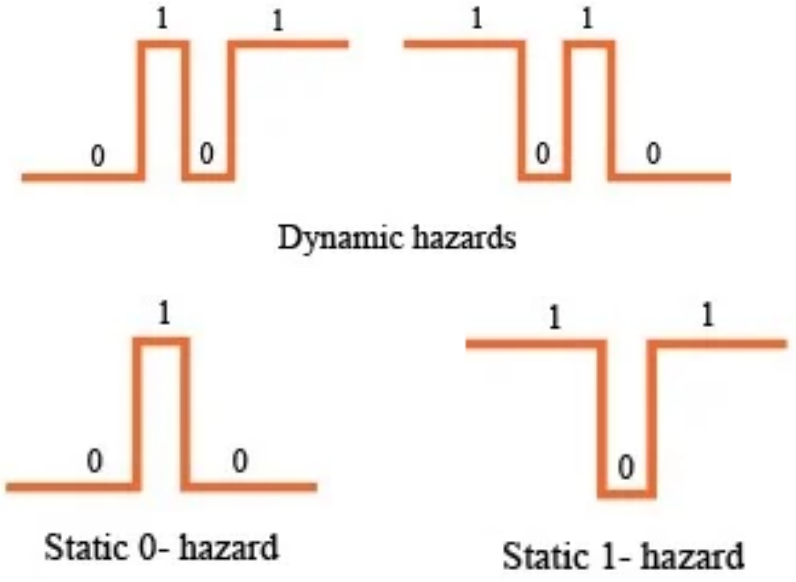


# FSM based Applicable digital design

Applicable digital design are mostly based on FSM forms of ***Moore*** or ***Mealy with stored output*** due to the next reasons:

***Reason 2:*** Robustness from moving *spikes / static 0-, 1-hazards / dynamic hazards* from Input to Output that happened between clk rising edges (excluding spikes at the Output that created due to clk rising edge).

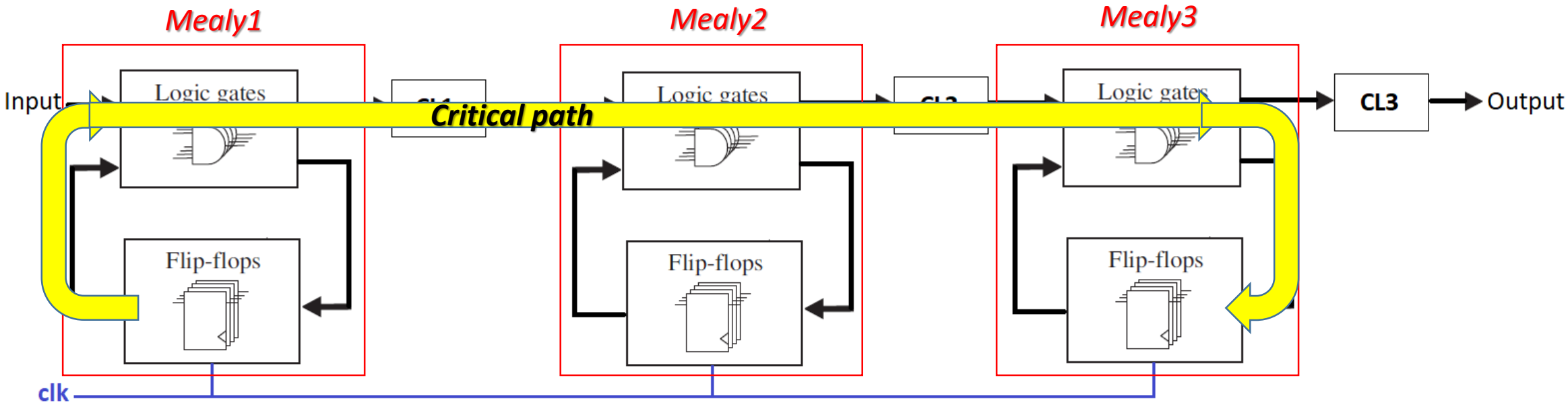


# FSM based Applicable digital design

Applicable digital design are mostly based on FSM forms of ***Moore*** or ***Mealy with stored output*** due to the next reasons:

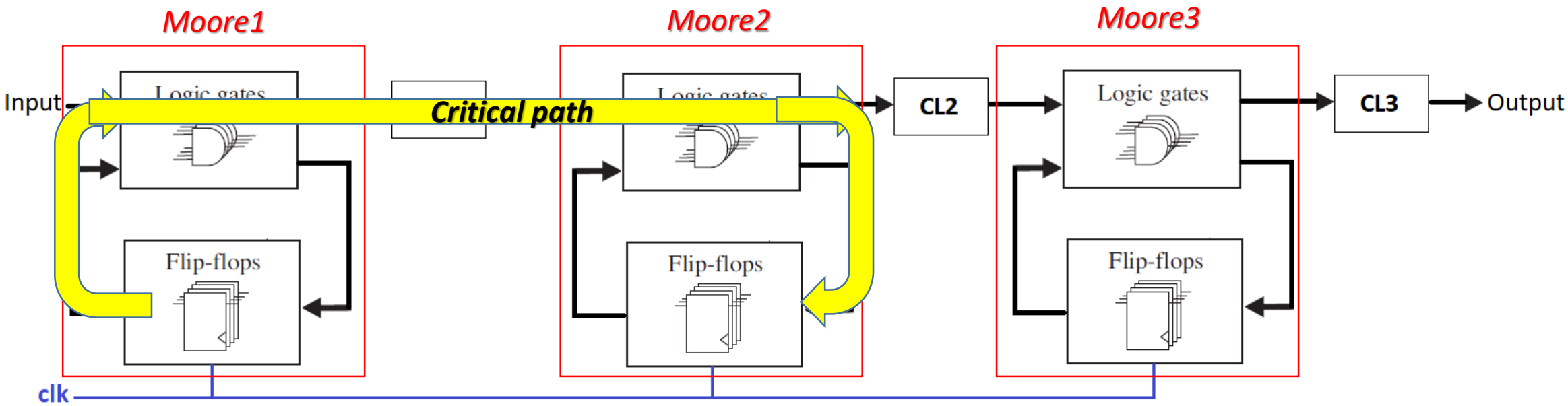
***Reason 3:*** Preventing from increasing the critical path which decreases the system $f_{max}$



# FSM based Applicable digital design

Applicable digital design are mostly based on FSM forms of ***Moore*** or ***Mealy with stored output*** due to the next reasons:

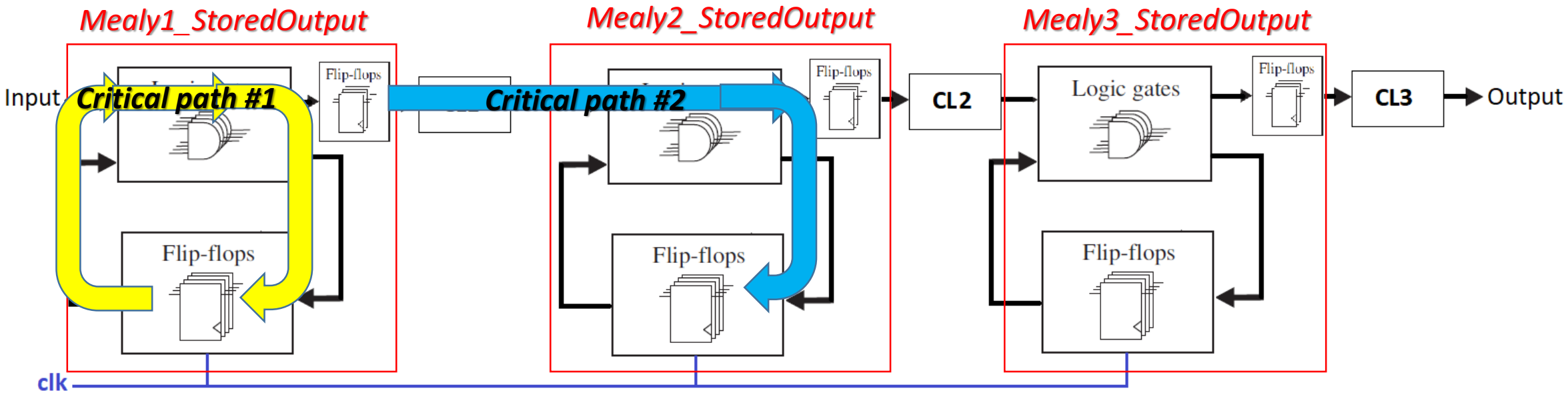
***Reason 3:*** Preventing from increasing the critical path which decreases the system $f_{max}$



# FSM based Applicable digital design

Applicable digital design are mostly based on FSM forms of ***Moore*** or ***Mealy with stored output*** due to the next reasons:

***Reason 3:*** Preventing from increasing the critical path which decreases the system $f_{max}$



# *Encoding Style: Binary, OneHot, TwoHot*

# Phase 4 – Implementation

*Tradeoff*:

***#DFFs***

***vs.***

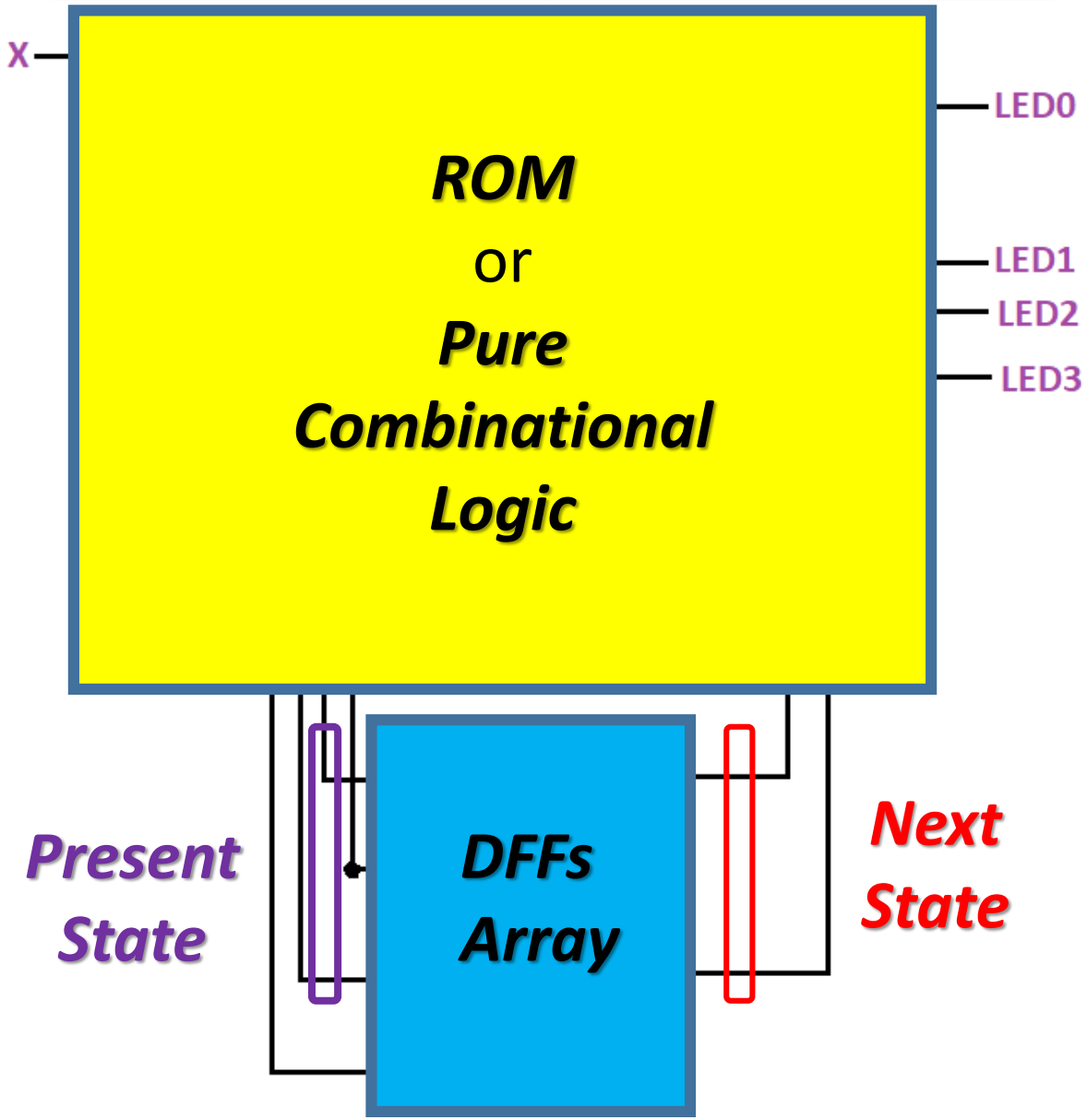
***FANin** and **FANout** size*

*Explanation*:

Depending the States Encoding Allocation (Binary Encoding or Direct Encoding)

***and***

The target HW (LUT based with low FANin but DFFs abundantly as in FPGA or the opposite in ASIC).



# Encoding Style: Binary, OneHot, TwoHot

State encoding example of an 8-state FSM

| STATE | BINARY | TWOHOT | ONEHOT |
|---|---|---|---|
| state0 | 000 | 00011 | 00000001 |
| state1 | 001 | 00101 | 00000010 |
| state2 | 010 | 01001 | 00000100 |
| state3 | 011 | 10001 | 00001000 |
| state4 | 100 | 00110 | 00010000 |
| state5 | 101 | 01010 | 00100000 |
| state6 | 110 | 10010 | 01000000 |
| state7 | 111 | 01100 | 10000000 |

# *Encoding Style: Binary, OneHot, TwoHot*

- In order to encode the states of a FSM, we can select one among several available styles (*Binary, OneHot, TwoHot. Etc.*).
- The default is ***binary***:
  - ✓ *Advantage* - requires the least number of flip-flops (n flip-flops (n bits), can encode up to 2n states).
  - ✓ *Disadvantage* - requires more logic and is slower than the others.
  - ✓ *Recommended in applications where Logic are abundant with high Fan-In, like in ASICs.*
- ***OneHot*** encoding (extreme to binary):

  One bit active per state.
  - ✓ *Advantage* - requires the least amount of extra logic and is the fastest.
  - ✓ *Disadvantage* - requires the largest number of flip-flops (uses one flip-flop per state - with n flip-flops (n bits), only n states can be encoded).
  - ✓ *Recommended in applications where flip-flops are abundant, like in FPGAs.*
- ***TwoHot*** encoding (inbetween binary and OneHot styles):

  Two bits active per state. uses Two flip-flops per state - with n flip-flops (n bits),

  only n(n -1)/2 states can be encoded).

# States Encoding Allocation

- *Approch1:* Default allocation determined by the synthesis tool IDE (Quartus of ALTEAR / SDK of XILINX) depends the target HW.

```
ARCHITECTURE <arch_name> OF <entity_name> IS
    -- Default allocation as Binary
    TYPE state IS (state0, state1, state2, state3, ...);
    SIGNAL pr_state, nx_state: state;
BEGIN
```

- *Approch2:* Using specific GUI / terminal commands in the synthesis tool IDE.
- *Approch3:* Using specific ***attributes*** in our HDL code which are acquainted by the synthesis tool.
- *Approch4:* Using direct allocation in our HDL code (Generic approach, independent on specific synthesis tool).

# Direct allocation - independent on synthesis tool

- *OneHot* direct encoding:

```
ARCHITECTURE <arch_name> OF <entity_name> IS
    -- Direct 4-BIT OneHot allocation
    SUBTYPE state IS std_logic_vector(3 DOWNTO 0);
    constant state0 : state := "1000";
    constant state1 : state := "0100";
    constant state2 : state := "0010";
    constant state3 : state := "0001";
    SIGNAL pr_state, nx_state: state;
BEGIN
```

- *TwoHot* direct encoding:

```
ARCHITECTURE <arch_name> OF <entity_name> IS
    -- Direct 4-BIT TwoHot allocation
    SUBTYPE state IS std_logic_vector(3 DOWNTO 0);
    constant state0 : state := "0011";
    constant state1 : state := "0101";
    constant state2 : state := "1001";
    constant state3 : state := "0110";
    SIGNAL pr_state, nx_state: state;
BEGIN
```