EE/CE 6301: Advanced Digital Logic

Bill Swartz

Dept. of EE
Univ. of Texas at Dallas

Session 03

VHDL-for-Synthesis

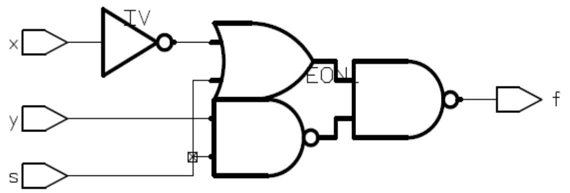
Dataflow Modeling

Dataflow Modeling

- In Dataflow style of modeling, the internal working of an entity can be implemented using concurrent signal assignment.
- VHDL code Example:

```
--Example: 2-1 Mux Dataflow Modeling in VHDL
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY  multiplexer2 IS
PORT ( x, y, s    : IN   BIT ;
              f   : OUT  BIT ) ;
END multiplexer2 ;
ARCHITECTURE multiplexer2_arch OF multiplexer2 IS
BEGIN
f <= (x AND NOT s) OR (y AND s)  ;
END multiplexer2_arch ;
```

- Schematics after Synthesis:



Behavioral Modeling

Behavioral Description

- In Behavioral design, we specify the behavior of the circuit, and do not necessarily know the gate level implementation.
- The internal working of an entity can be implemented using a set of statements
- Synthesis takes care of implementing the circuit in gate level
- May or may not produce the most optimized circuit after synthesis

Example 2: Behavioral 2-1 Mux

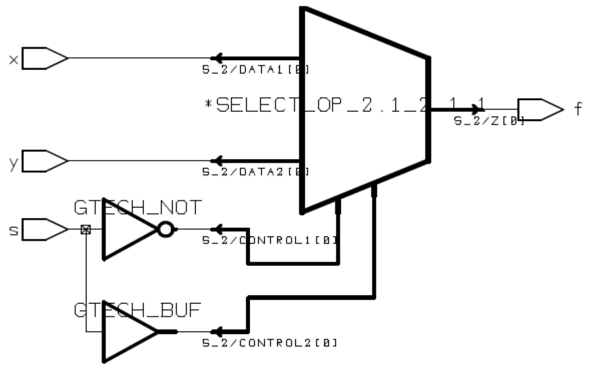
- VHDL Code:

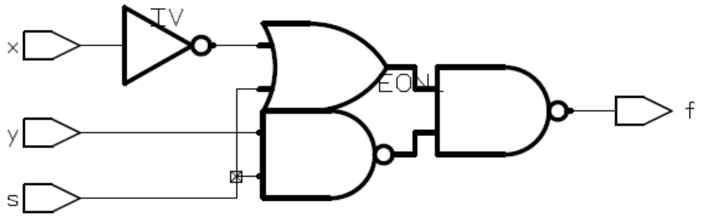
```
--Example 2: Mux2 Behavioral VHDL code
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity behmux2 is

port ( x, y, s    : in   std_logic ;

        f         : out   std_logic ) ;
end behmux2;

architecture behmux2_arch of behmux2 is

begin
   with  s  select
        f <= x when '0',
             y when '1',
            '0' when others;
end behmux2_arch;
```

- Behavior of 2-1 MUX is written using "with-select-when" statement in VHDL.

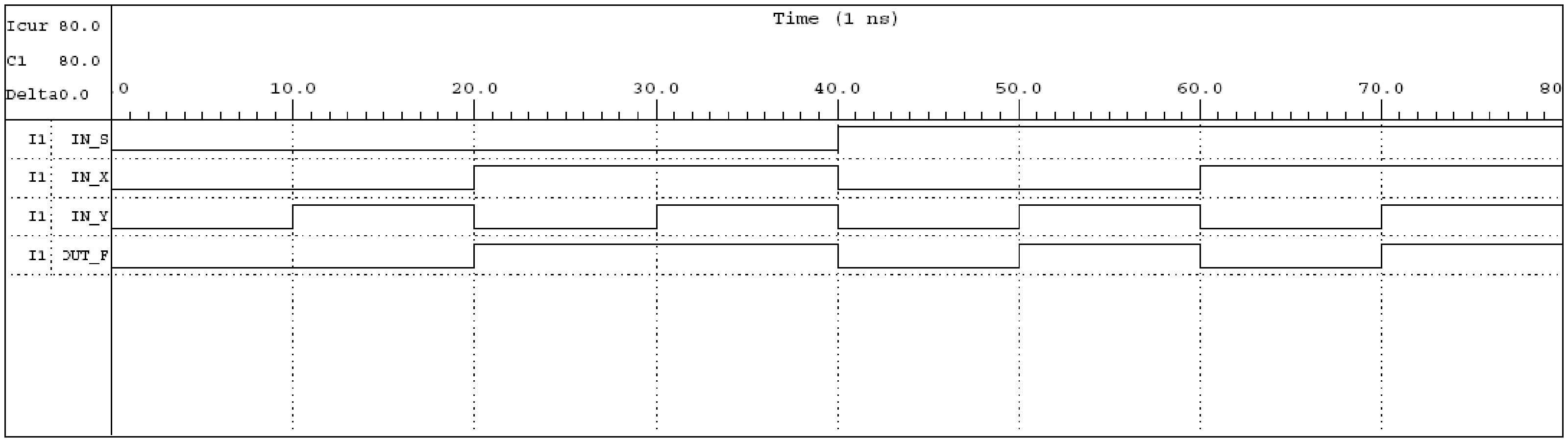
Example 2: Synthesis and Simulation

- Synopsys Synthesis
  - Schematics before and after compilation:





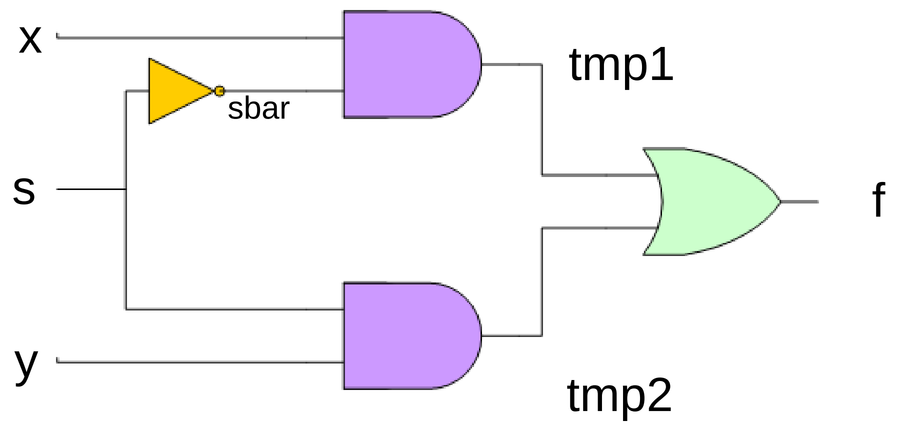
- Wave graph simulation:



Structural Modeling

Structural Description

- Structural design forces the tool to use certain gates with specified connections for synthesis.
- Gate level structure should be known. Example:



- Components and port mapping is used.
- Structure of the circuit would not alter after synthesis.

Structural Modeling

- The implementation of an entity is done through a set of interconnected components
- Contains:
  - Signal declaration
  - Component instances
  - Port maps
  - May contain wait statements (for simulation)
- Before instantiating the component it should be declared using component declaration.
- Component declaration declares the name of the entity and interface of a component.
- By structural modeling, we can force synthesizer to synthesize an exact gate-level structure with no optimization.

Example 3: Structural 2-1 Mux

- VHDL Code:

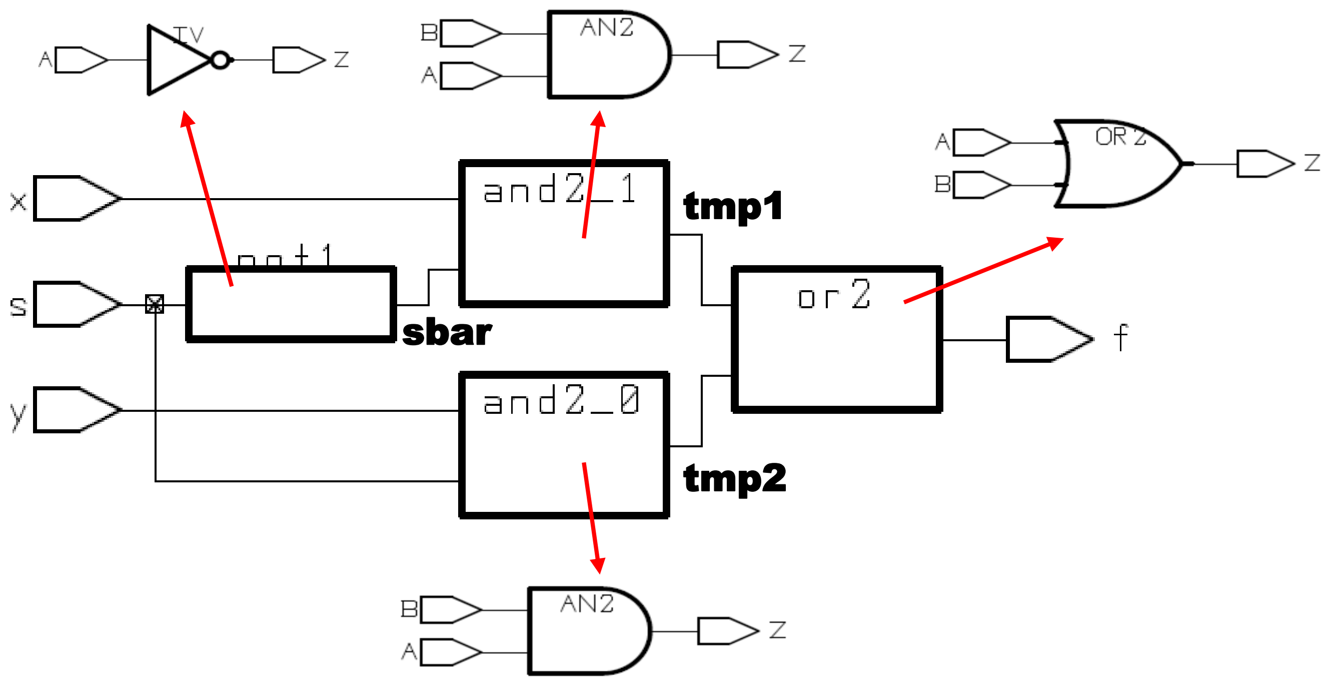
```
--Example 3: Structural 2-1 Mux
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
---------------------------------------
entity and2 is
port ( A, B : in std_logic;
Z : out std_logic);
end and2;
architecture beh_and2 of and2 is
begin -- beh_and2
Z <= A and B;
end beh_and2;
---------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
entity or2 is
port ( A, B : in std_logic;
Z : out std_logic);
end or2;
architecture beh_or2 of or2 is
begin -- beh_or2
Z <= A or B;
end beh_or2;
```

```
--------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
entity not1 is
port ( A : in std_logic;
Z : out std_logic);
end not1;
architecture beh_not1 of not1 is
begin -- beh_not1
Z <= not A;
end beh_not1;
--------------------------------------
--TOP LEVEL MODULE
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
ENTITY strucmux2 IS
PORT ( x, y, s : IN  std_logic ;
           f     : OUT   std_logic ) ;
END strucmux2;
ARCHITECTURE strucmux2_arch OF
    strucmux2 IS
component and2
```

```
port ( A, B : in std_logic;
Z : out std_logic);
end component;
component or2
port ( A, B : in std_logic;
Z : out std_logic);
end component;
component not1
port ( A : in std_logic;
Z : out std_logic);
end component;

SIGNAL tmp1, tmp2, sbar: std_logic;
BEGIN
S_1  : not1 port map( A => s, Z =>
sbar);
S_2  : and2 port map( A => x, B =>
sbar, Z => tmp1);
S_3  : and2 port map( A => y, B => s,
Z => tmp2);
S_4  : or2 port map( A => tmp1, B =>
tmp2, Z => f);
END;
```

Example 3: Synthesis

- Schematics after synthesis and compilation:



- Gate level-structure after synthesis remains just as we specified
  - Useful for test (DFT) purposes for specified circuits.

Don’t-Cares in VHDL

Don’t-Cares in VHDL

- In logic synthesis and logic simulation, a ***don't-care*** is one of the values in a multi-valued logic system that denotes an unknown value, or a value that the designer (for whatever reason) does not care about.
- A don’t-care or X value does not exist in hardware. In simulation, an X value can result from two or more sources driving a signal simultaneously, or the stable output of a flip-flop not having been reached.
- In synthesized hardware, however, the actual value of such a signal will be either 0 or 1, but will not be determinable from the circuit's inputs.
- Synthesis tools can use don't-care values to determine where and how to perform area optimization. For example, a synthesis tool can use don't-care values to reduce the number of states and circuit size of finite state machines.

Example 4: Don't-Cares in VHDL

- Don't-care in digital design denotes a value could be either 0 or 1.
- Don't-cares are specified with '-' in VHDL.
- Suppose f is a function of minterms of a, b, c, and d such that:

$$f(a,b,c,d)=\sum m(1,4,5,7,13)+d(3,6)$$

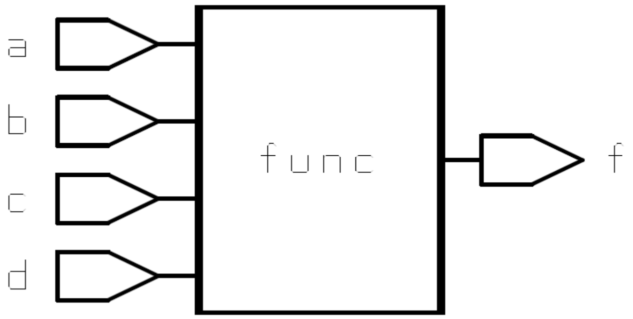
- Example 4 VHDL Code:

```
--Example 4: VHDL code including don't-cares
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity func is
port (
  a,b,c,d: in std_logic;
  f: out std_logic);
end func;
architecture archf of func is
     signal q:std_logic_vector(3 downto 0);
begin
```

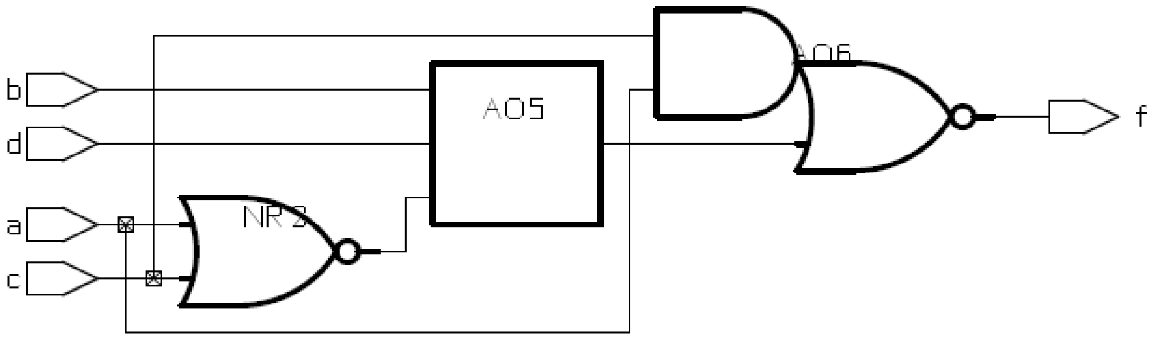
```
    q <= a & b & c & d; --concatenating to produce abcd
    with  q  select
     f<='1' when "0001" | "0100" | "0101" | "0111" | "1101",
        '-' when "0011" | "0110",
         '0' when others;
end archf;
```

Example 4: Synthesis

- f is a function of a, b, c, and d:



- After synthesis and design optimization:



Don’t-Cares in VHDL

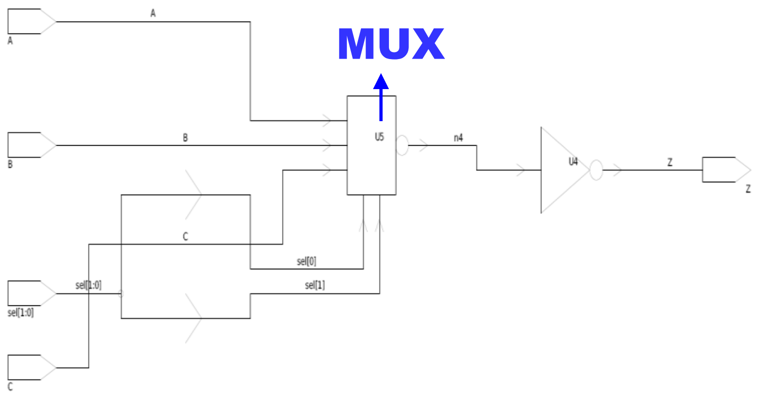
- When **CASE** or **IF** statements do not cover all possible input conditions ***unwanted latches*** may be generated to hold the output.
  - Conditions that are not covered do not mean that they are don’t-cares!

- Including the final **ELSE** clause or **WHEN OTHERS** clause in an **IF** or **CASE** statement can prevent this unwanted latch from being generated.

Don’t-Cares in VHDL (cont’d)

- Example 4.1: 3-1 MUX with all conditions covered in case-when statement using when-others:

```
--Example 4.1: 3-1 MUX with all conditions covered
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
entity mux3r is
port ( A, B, C: in std_logic;
    sel: in std_logic_vector(1 downto 0);
Z : out std_logic);
end mux3r;
ARCHITECTURE mux3r_arch OF mux3r IS
BEGIN
pMux : process (A,B,C,sel)
begin
   case sel is
   when "00" => Z <= A;
   when "01" => Z <= B;
   when others => Z <= C;
   end case;
end process pMux;
END;
```



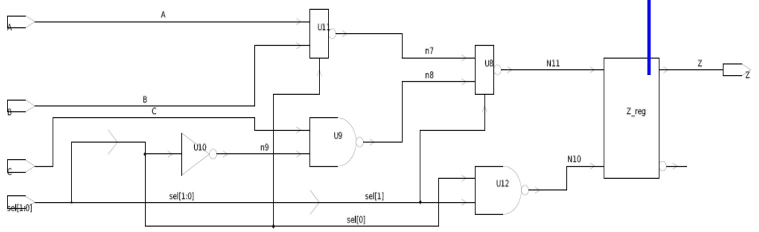
No Latch generated in Synthesis

Don't-Cares in VHDL (cont'd)

- Example 4.2: 3-1 MUX when all conditions are not covered in case-when statement:

```
--Example 4.2: 3-1MUX when all conditions are not covered
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
entity mux3w is
port ( A, B, C: in std_logic;
    sel: in std_logic_vector(1 downto 0);
Z : out std_logic);
end mux3w;
ARCHITECTURE mux3w_arch OF mux3w IS
BEGIN
pMux : process (A,B,C,sel)
begin
   case sel is
   when "00" => Z <= A;
   when "01" => Z <= B;
   when "10" => Z <= C;
   when others => NULL;
   end case;
end process pMux;
END;
```

Latch Inferred!



Synthesizer Analysis Report

Metrics Often Reported

- The three main metrics of the synthesized design (and other metrics) can be reported by the tool.
- Depending on which library Synopsys tool uses (**class**, **gtech**, etc.), design metric values would differ.
- Total combinational area is reported in terms of equivalent 2-input NAND gates
- Non-combinational area would be non-zero for sequential designs containing flip-flops.
- Total dynamic power can be reported by the tool.
- Timing characteristics can be reported, and the values are given with respect to rising (r) and falling (f) times in nanoseconds.

Area Report (2-Input MUX Example)

```
****************************************
Report : area
Design : behmux2
Version: V-2003.12
Date   : Sun Jan 11 03:36:57 2009
****************************************
Library(s) Used:
    class (File: /home/cad/synopsys-2003/syn/libraries/syn/class.db)
Number of ports:                  4
Number of nets:                   5
Number of cells:                  2
Number of references:             2
Combinational area:            4.000000
Noncombinational area:         0.000000
Net Interconnect area:        undefined  (Wire load has zero net area)
Total cell area:               4.000000
Total area:                   undefined
1
design_analyzer>
****************************************
```

Power Report

```
****************************************
Report : power
    -analysis_effort low
Design : behmux2
Version: V-2003.12
Date   : Sun Jan 11 03:36:57 2009
****************************************
Library(s) Used:
    class (File: /home/cad/synopsys-2003/syn/libraries/syn/class.db)
Warning: The library cells used by your design are not characterized for internal power. (PWR-26)
Operating Conditions:
Wire Load Model Mode: top
Design        Wire Load Model            Library
------------------------------------------------
behmux2                05x05             class
Global Operating Voltage = 5
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 0.100000ff
    Time Units = 1ns
    Dynamic Power Units = 100nW    (derived from V,C,T units)
    Leakage Power Units = Unitless
  Cell Internal Power  =   0.0000 nW    (0%)
  Net Switching Power  =   4.5201 uW  (100%)
                         ---------
Total Dynamic Power    =   4.5201 uW  (100%)
Cell Leakage Power     =   0.0000
design_analyzer>
****************************************
```

Timing Report

```
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : behmux2
Version: V-2003.12
Date   : Sun Jan 11 03:36:57 2009
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: x (input port)
  Endpoint: f (output port)
  Path Group: (none)
  Path Type: max
  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  behmux2            05x05                 class
  Point                                    Incr       Path
  -----------------------------------------------------------
  input external delay                     0.00       0.00 f
  x (in)                                   0.00       0.00 f
  U13/Z (IV)                               0.58       0.58 r
  U12/Z (EON1)                             0.90       1.48 f
  f (out)                                  0.00       1.48 f
  data arrival time                                   1.48
  -----------------------------------------------------------
 (Path is unconstrained)
design_analyzer>
```

Report (cont'd)

- Report of Area, Power and Timing:

```
 (Path is unconstrained)
1
design_analyzer>
****************************************
Report : area
Design : behmux2
Version: V-2003.12
Date   : Sun Jan 11 03:37:01 2009
****************************************
Library(s) Used:
    class (File: /home/cad/synopsys-2003/syn/libraries/syn/class.db)
Number of ports:              4
Number of nets:               5
Number of cells:              2
Number of references:         2
Combinational area:        4.000000
Noncombinational area:     0.000000
Net Interconnect area:    undefined  (Wire load has zero net area)
Total cell area:           4.000000
Total area:               undefined
1
design_analyzer>
****************************************
```

Report (cont'd)

- Report of Area, Power and Timing:

```
Report : power
    -analysis_effort low
Design : behmux2
Version: V-2003.12
Date   : Sun Jan 11 03:37:01 2009
****************************************
Library(s) Used:
    class (File: /home/cad/synopsys-2003/syn/libraries/syn/class.db)
Warning: The library cells used by your design are not characterized for internal power. (PWR-
    26)
Operating Conditions:
Wire Load Model Mode: top
Design        Wire Load Model            Library
------------------------------------------------
behmux2                 05x05            class
Global Operating Voltage = 5
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 0.100000ff
    Time Units = 1ns
    Dynamic Power Units = 100nW    (derived from V,C,T units)
    Leakage Power Units = Unitless
  Cell Internal Power  =   0.0000 nW    (0%)
  Net Switching Power  =   4.5201 uW  (100%)
                         ---------
Total Dynamic Power    =   4.5201 uW  (100%)
```

Report (cont'd)

- Report of Area, Power and Timing:

```
Cell Leakage Power     =   0.0000
1
design_analyzer>
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : behmux2
Version: V-2003.12
Date   : Sun Jan 11 03:37:01 2009
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: x (input port)
  Endpoint: f (output port)
  Path Group: (none)
  Path Type: max
  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  behmux2            05x05                 class
  Point                                    Incr       Path
  -----------------------------------------------------------
input external delay
    0.00        0.00 f
  x (in)
    0.00        0.00 f
  U13/Z (IV)
    0.58        0.58 r
  U12/Z (EON1)
    0.90        1.48 f
  f (out)
    0.00        1.48 f
  data arrival time
    1.48
  ------------------------------------------
    -----------------
   (Path is unconstrained)
1
design_analyzer>
```

Parametric Coding in VHDL

Parametric Coding in VHDL

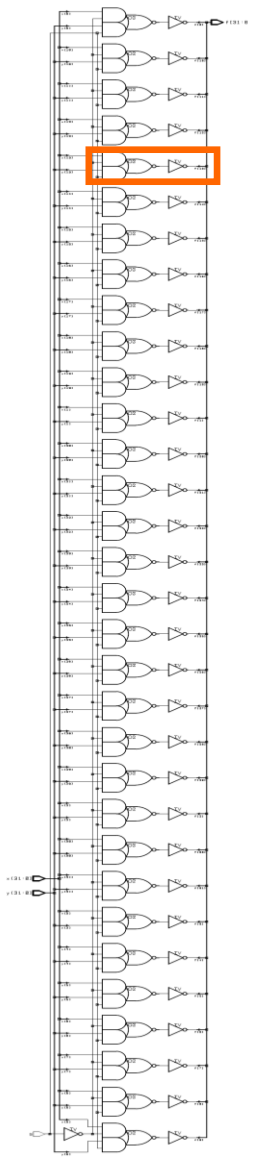
- The concept of *generics* is often used to parameterize components.
- Example 5.1: 32-bit 2-input MUX:

```
--Example 5.1: 32-bit input MUX
library IEEE;
use IEEE.std_logic_1164.all;
entity mux32 is
generic(n : natural := 31);
port(
 x : in std_logic_vector (n downto 0);
 y : in std_logic_vector (n downto 0);
 s : in std_logic;
 f : out std_logic_vector (n downto 0));
 end entity mux32;
 architecture behavior of mux32 is
 begin -- behavior -- no process needed with concurrent statements
   f <= y when s='1' or s='H' else x;
 end architecture behavior;  -- of mux32
```

Example 5.1 (cont'd)

- Schematics after synthesis and design optimization:

- The generic statement allows parametric settings so that the design could be compiled for desired values.



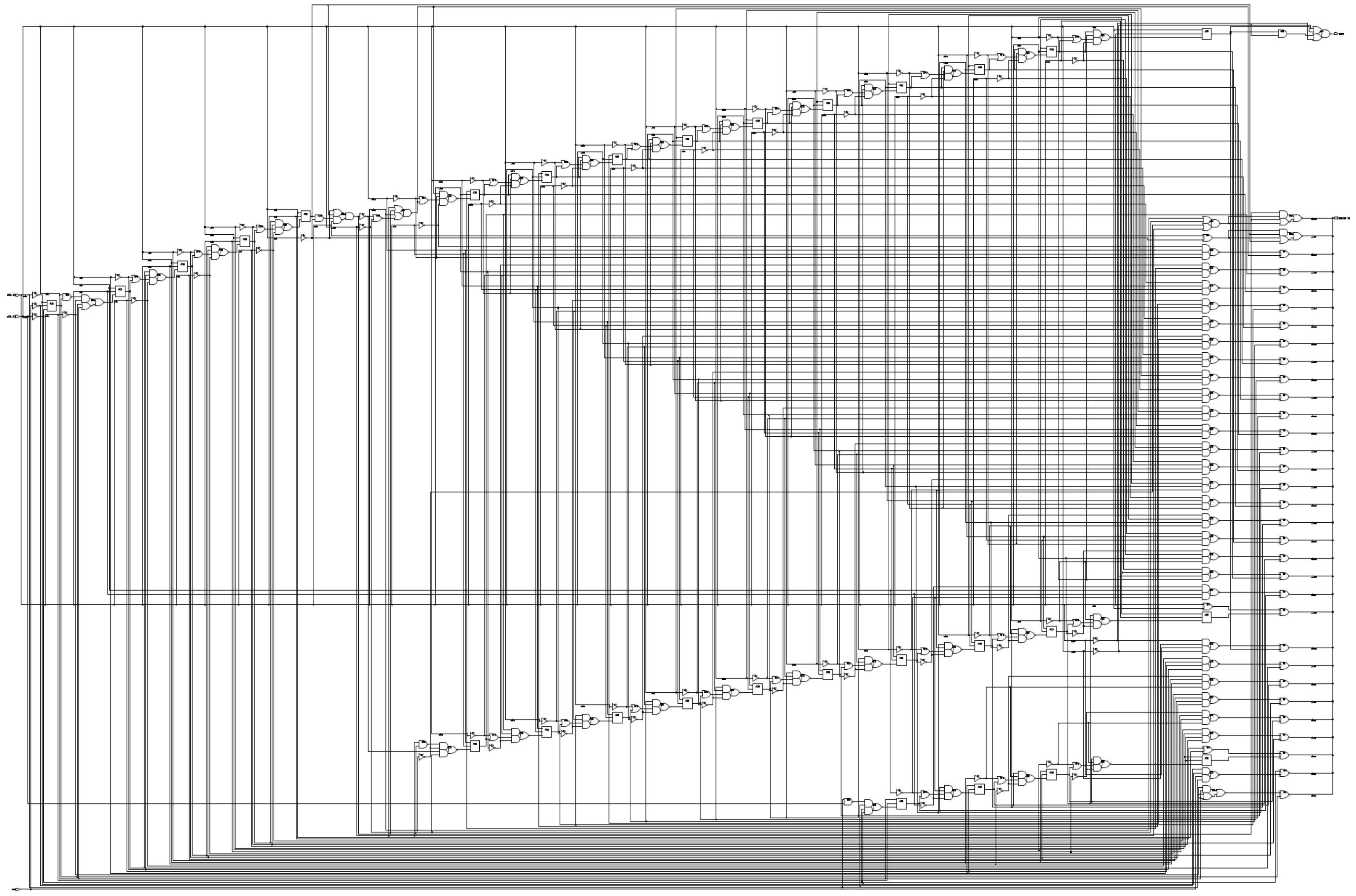
Parametric Loop in VHDL

- Generic statement required in entity.
- Loop is declared using for-loop statement
- Example 5.2: VHDL code for a 32-bit adder:

```
-- Example 5.2: 32-bit input adder
library IEEE;
use IEEE.std_logic_1164.all;
entity add32 is
  generic(n : natural := 31);
    port (a    : in  std_logic_vector (n downto 0);
          b    : in  std_logic_vector (n downto 0);
          cin  : in  std_logic;
          sum  : out std_logic_vector (n downto 0);
          cout : out std_logic);
end entity add32;
architecture behavior of add32 is
begin  -- behavior
  adder: process
           variable carry : std_logic; -- internal
           variable isum : std_logic_vector(n downto 0);  -- internal
         begin
           carry := cin;
           for i in 0 to n loop
             isum(i)  := a(i) xor b(i) xor carry;
             carry   := (a(i) and b(i)) or (a(i) and carry) or (b(i) and carry);
           end loop;
           sum  <= isum;
           cout <= carry;
         end process adder;
end architecture behavior;  -- of add32
```

Example 5.2 (cont'd)

- Schematic after synthesis and design optimization:



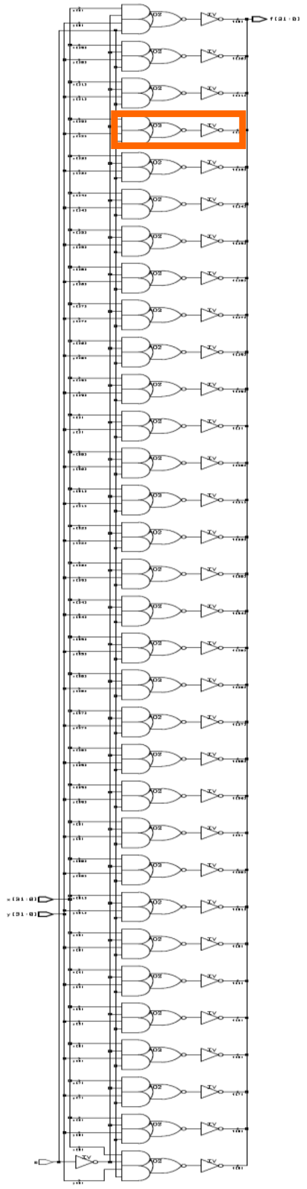
Parametric Coding in VHDL (cont'd)

- Iterative commands declared using for-generate statements
  - Makes copies of concurrent statements
- Example 5.3: 32-bit 2-input MUX with for-generate:

```
--Example 5.3: 32-bit input MUX with for-generate statements
library IEEE;
use IEEE.std_logic_1164.all;
entity mux32_it is
generic(n : natural := 31);
port(
 x : in std_logic_vector (n downto 0);
 y : in std_logic_vector (n downto 0);
 s : in std_logic;
 f : out std_logic_vector (n downto 0));
 end entity mux32_it;
 architecture mux32_it_arch of mux32_it is
 begin
     gen1: for i in 0 to n generate
            f(i) <= (x(i) and not s) or (s and y(i));
     end generate gen1;
 end architecture mux32_it_arch;
```

Example 5.3 (cont'd)

- The **generate** statement combines concurrent statements with a looping capability
- Schematics after synthesis and design optimization ->
- The same gate level structure achieved as in Example 5.1

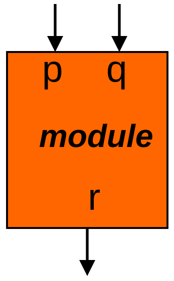


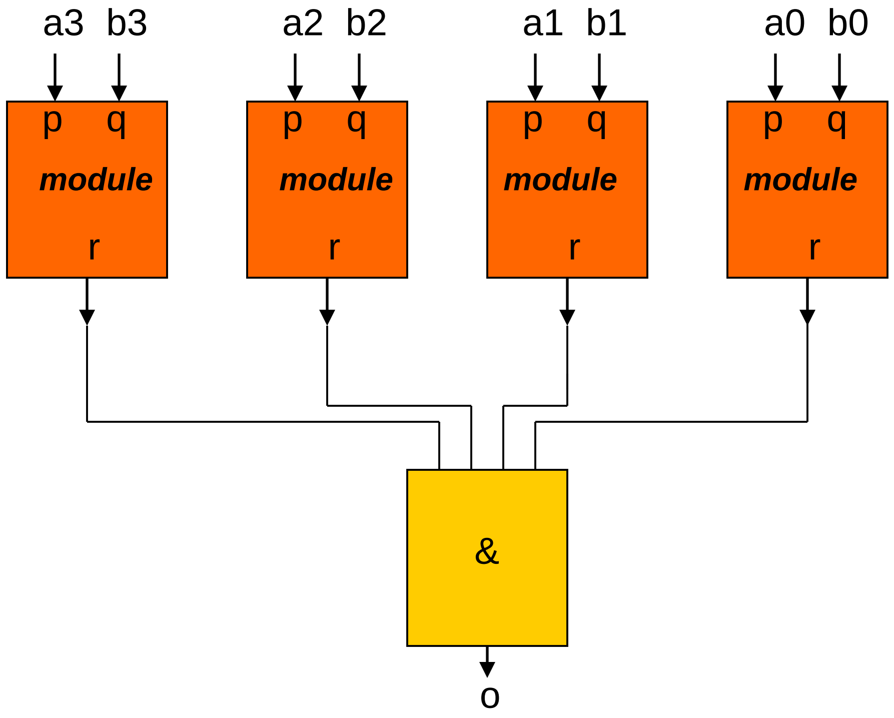
Hierarchical Design in VHDL

Hierarchical Design in VHDL

- VHDL supports design by creating modules of circuit components that can be used in another design

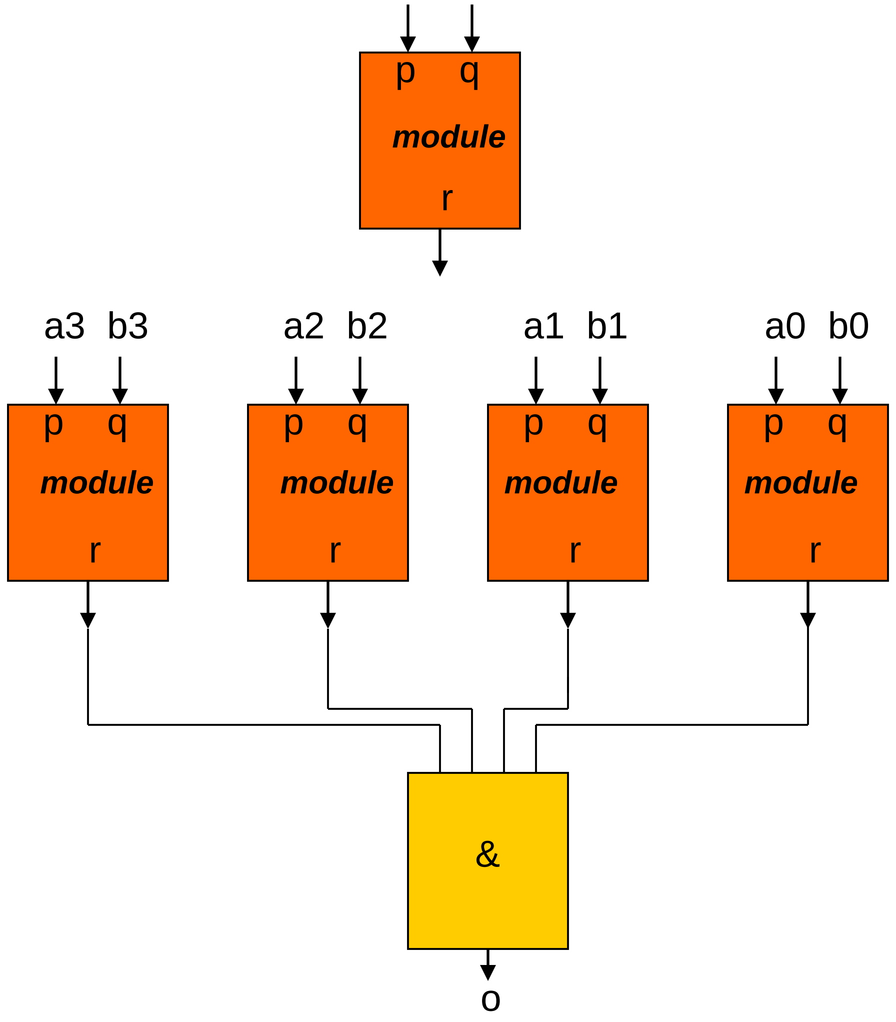
- Components and port mapping used in top level design





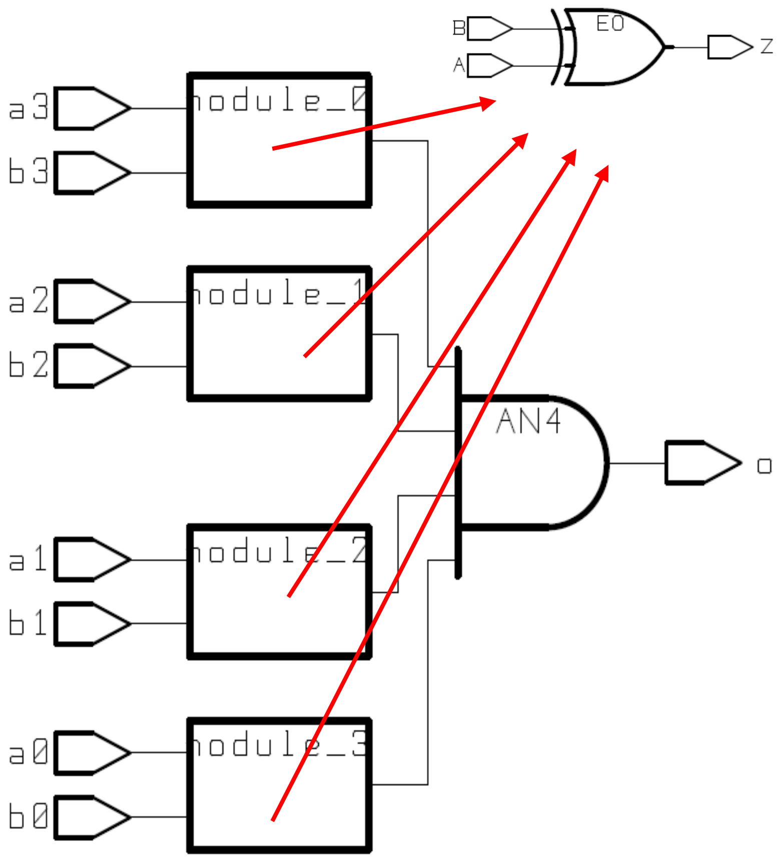
Example 6: A hierarchical Design

```
--Example 6: A hierarchical design
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
entity module is
port ( A, B : in std_logic;
          Z : out std_logic);
end module;
architecture module_arch of module is
begin
   Z <= (A and not B) or (not A and B) ;
end module_arch;
--------------------------------------
--------------------------------------
--TOP LEVEL MODULE
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
ENTITY hier IS
PORT ( a0, b0, a1, b1, a2, b2, a3, b3: IN   std_logic;
         o  : OUT   std_logic ) ;
END hier;
ARCHITECTURE hier_arch OF hier IS
component module
port ( A, B : in std_logic;
          Z : out std_logic);
end component;
SIGNAL p, q, r, s: std_logic;
BEGIN
M_1 : module port map( A => a0, B => b0, Z => p);
M_2 : module port map( A => a1, B => b1, Z => q);
M_3 : module port map( A => a2, B => b2, Z => r);
M_4 : module port map( A => a3, B => b3, Z => s);
o <= p and q and r and s;
END;
```



Example 6: Synthesis

- Schematics design after Synopsys synthesis, design optimization and compilation:
- The module in this design is an XOR gate
- Each module would be optimized alone
- Does not necessarily optimize the whole design altogether

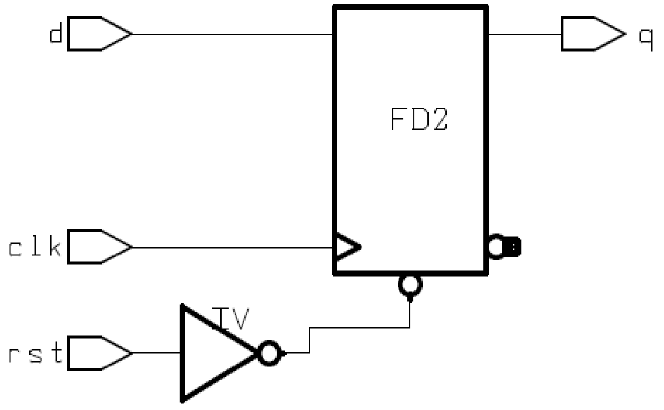


Sequential Modeling

Modeling Flip-Flops

- D-flip-flop behavior

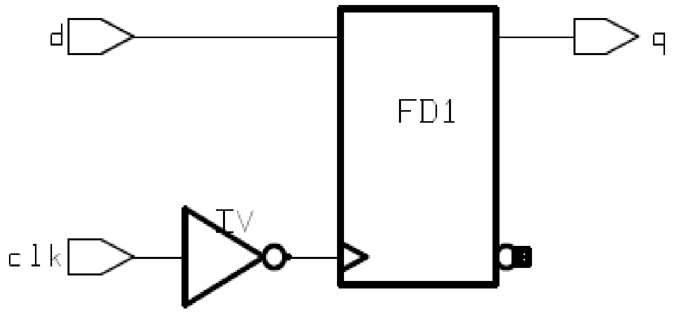
```
-Example 7.1: D-flip-flop with reset
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity dff is
port( d:in std_logic;
      clk,rst:in std_logic;
      q:out std_logic);
end dff;
architecture dff_arch of dff is
begin
 process(rst, clk)
    begin
        if rst = '1' then
            q <= '0';
        elsif clk'event and clk = '1' then
                q <= d; --rising edge triggered
        end if;
    end process;
end dff_arch;
```



Negative Edge Triggered D-Flip-flop

- VHDL Code:

```
-- Example 7.2 Negative Edged Triggered DFF
library ieee;
use ieee.std_logic_1164.all;

   entity dflipn is port (
         d,clk: in std_logic;
         q:  out std_logic );
   end dflipn;

architecture dflipn_arch of dflipn is
   begin
      process(clk)
      begin
      if (clk'event and clk = '0') then
            q <= d; -- set the changes to the negative edge of clock

      end if;
     end process;
end dflipn_arch;
```



Importance of Sensitivity List

Process and Sensitivity List

- A process is a wrapper for sequential statements.
  - Sequential statements model combinational or synchronous logic (or both).
  - Statements within a process are executed sequentially.
  - Beware! Signal assignments are BOTH concurrent and sequential.
- A process is concurrent with other concurrent statements in an architecture.
- An architecture can have multiple processes.
- Signal changes in the sensitivity list cause the process to "run" or be evaluated.
  - All sequential statements in the process are "executed".
  - Variables are updated as statements are sequentially "executed".
  - Signal updates are scheduled as statements are sequentially "executed".
  - Signal updates occur when the process is suspended (finished).
- A common error is to think that signal updates take place when a sequential statement is executed.

Importance of Sensitivity List

- The signal sensitivity list is used to specify which signals should cause the process to be re-evaluated.
- Whenever any event occurs on one of the signals in the sensitivity list, the process is re-evaluated.
- A process is evaluated by performing each statement that it contains. These statements (the body of the process) appear between the **begin** and **end** keywords.
- Synthesis programs do not care about sensitivity list but gives you a warning if they are not complete.

Example 8: Importance of Sensitivity List

- If “d” input is added to the sensitivity list of D-Flip-Flop (Example 7.1), there would just be an overhead for simulation, but no false behavior for simulation or synthesis would be observed if it is not on the list.
  - This behavior is because of the rising-edge detection of D-Flip-Flop.
- Transparent latch should have the “d” input added to the sensitivity list, as change in input should be reflected at output when “en” is high.
- VHDL Code for Latch with input “d” on sensitivity list:

```
--Example 8.1: Transparent Latch
library ieee;
use ieee.std_logic_1164.all;
entity mylatch is
port( d:in std_logic;
     en:in std_logic;
      o:out std_logic);
end mylatch;
```

```
architecture mylatch_arch of mylatch is
begin
 Latch_Data:process(en,d)
 --sensitivity list
  begin
   if (en = '1') then
    o <= d;
 -- If en = 0, then o keeps its old value.
   end if;
 end process Latch_Data;
end mylatch_arch;
```

Example 8 (cont'd)

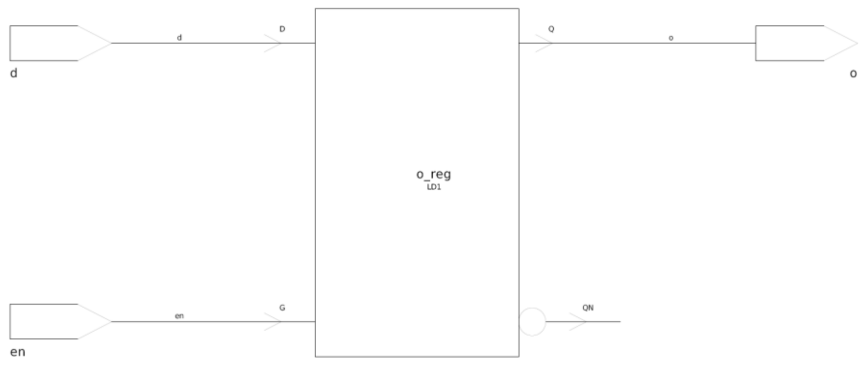
- For combinational logic or latches you have to take more care with the sensitivity list.
- If "d" input is not on sensitivity list of the latch, -> wrong functionality in simulation.
- VHDL code of a latch with incorrect behavior:

```
--Example 8.2: Latch with incorrect behavior
library ieee;
use ieee.std_logic_1164.all;
entity mylatch is
port( d:in std_logic;
     en:in std_logic;
      o:out std_logic);
end mylatch;
```

```
architecture mylatch_arch of mylatch is
begin
 Latch_Data: process(en)
--sensitivity list
  begin
   if (en ='1') then
    o <= d;
 -- If en = 0, then o keeps its old value.
   end if;
 end process Latch_Data;
end mylatch_arch;
```

Example 8: Synthesis & Test bench

- Synthesis of both example 8.1 and 8.2 codes look alike!



- VHDL Code for Test bench:

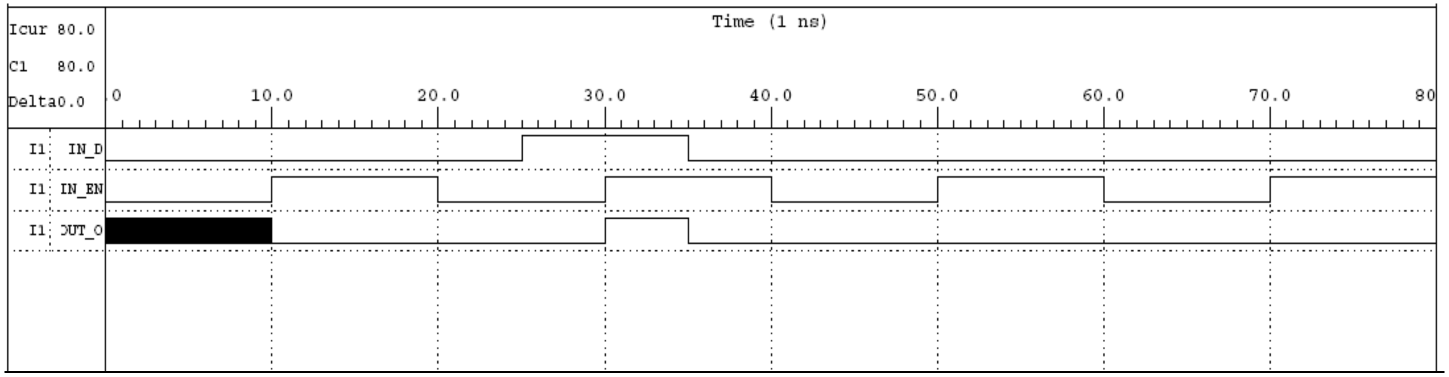
```
--Test bench for Example 8: Latch
library IEEE;
USE IEEE.std_logic_1164.all;
entity tbmylatch is
end tbmylatch;
architecture tbmylatch_arch of tbmylatch is
component mylatch
port( d:in std_logic;
     en:in std_logic;
      o:out std_logic);
end component;
```

```
signal in_d, in_en, out_o: std_logic := '0';
begin
imylatch:mylatch port map(d=>in_d, en=>in_en, o=>out_o);
in_en<= not in_en after 10 ns;
in_d<='0','1' after 25 ns, '0' after 35 ns;
end tbmylatch_arch;
configuration cf_mylatch of tbmylatch is
for tbmylatch_arch
for imylatch:mylatch
use entity WORK.mylatch (mylatch_arch);
end for;
end for;
end cf_mylatch;
```

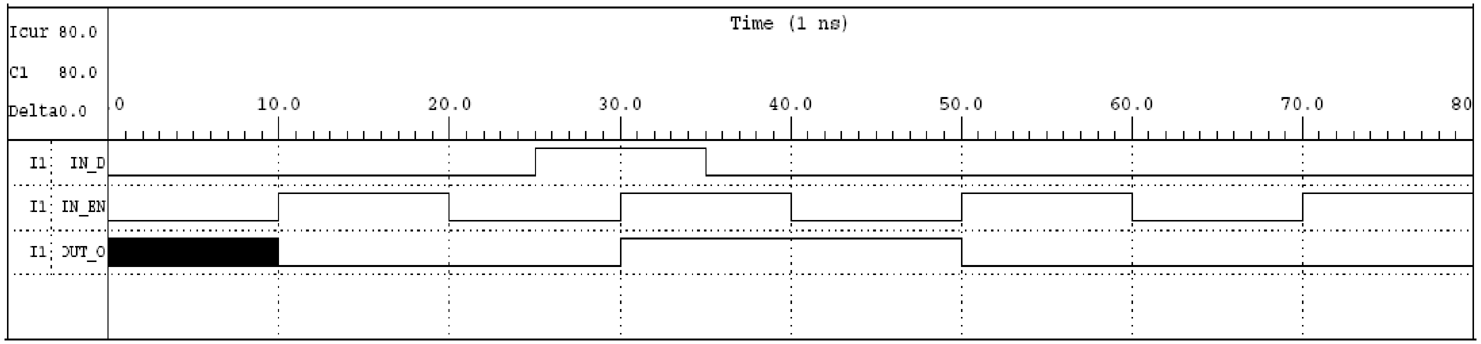
Example 8: Simulation

- Simulation results of Example 8.1:

Correct



- Simulation results of Example 8.2:



Incorrect functionality

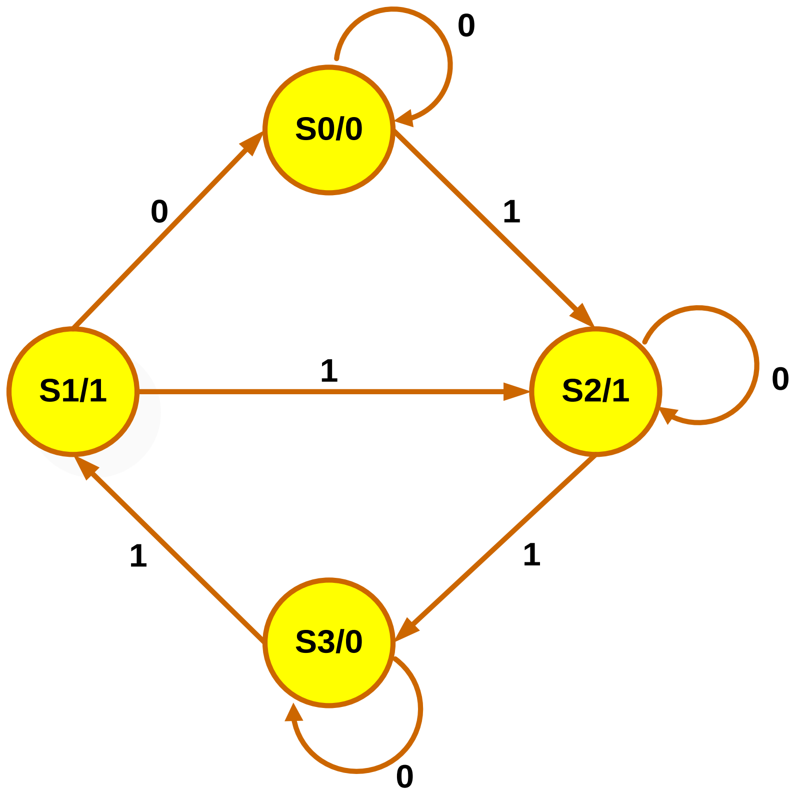
Modeling Sequential FSM

Sequential Logic Using VHDL

- Finite State Machine (**FSM**)
  - —A circuit that has defined states and can switch between them if certain conditions exist
- Moore Machine
  - —A **Moore** machine has output(s) that depend on state only
  - —The FSM has the output(s) written in the state itself
- Mealy Machine
  - —A **Mealy** machine has output(s) that depend on both the state and input(s)
  - — The FSM has the output written on edges

Moore Machine

State Diagram:



Transition Table:

Present State	Next State		Output (z)
	x = 0	x = 1	
S0	S0	S2	0
S1	S0	S2	1
S2	S2	S3	1
S3	S3	S1	0

Example 9: Moore Machine

- VHDL Code:

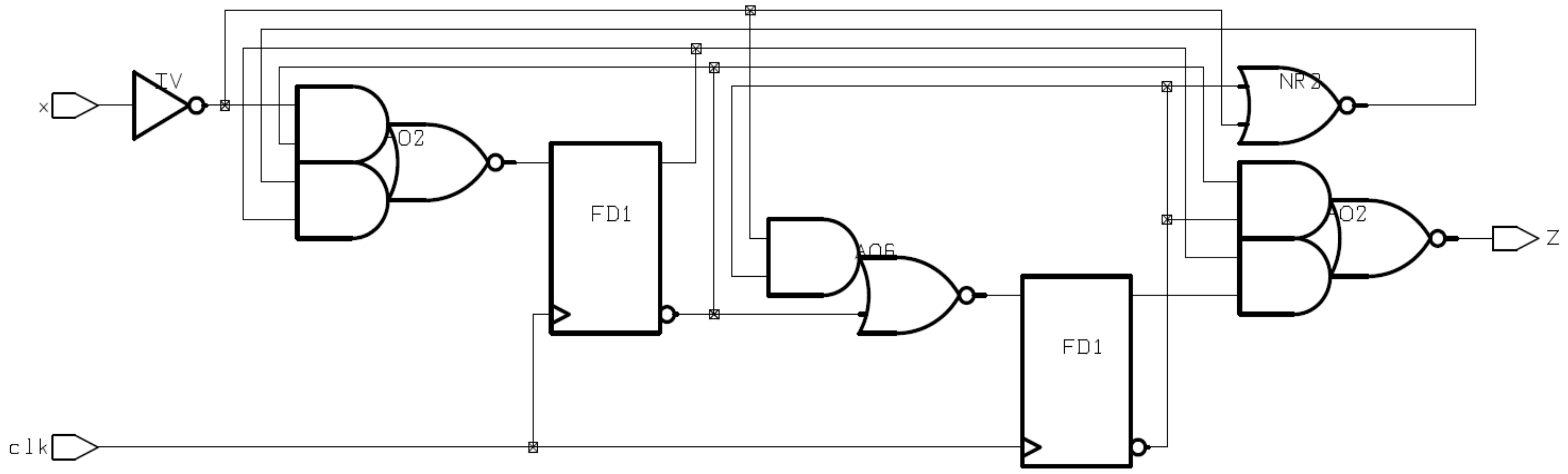
```
--Example 9: Moore machine
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity moore is
port( x, clk:in bit;
      Z:out bit);
end moore;

architecture moore_arch of moore is
type state_type is (S0, S1, S2, S3);
signal current_state, next_state: state_type;

begin
 --process to hold synchronous elements (flip-flops)
 SYNCH:process
 begin
  wait until clk'event and clk='1';
  current_state <= next_state;
 end process;
```

```
 --process to hold combinational logic
 COMBIN:process (current_state, x)
 begin
  case current_state is
   when S0 =>
   Z <= '0';
   if x='0' then
    next_state <= S0;
   else
    next_state <= S2;
   end if;
  when S1 =>
   Z <= '1';
   if x='0' then
    next_state <= S0;
   else
    next_state <= S2;
   end if;
  when S2 =>
   Z <= '1';
   if x='0' then
    next_state <= S2;
   else
    next_state <= S3;
   end if;
  when S3 =>
   Z <= '0';
   if x='0' then
    next_state <= S3;
   else
    next_state <= S1;
   end if;
  end case;
 end process;
end moore_arch;
```

Output is defined outside if-then-else statement

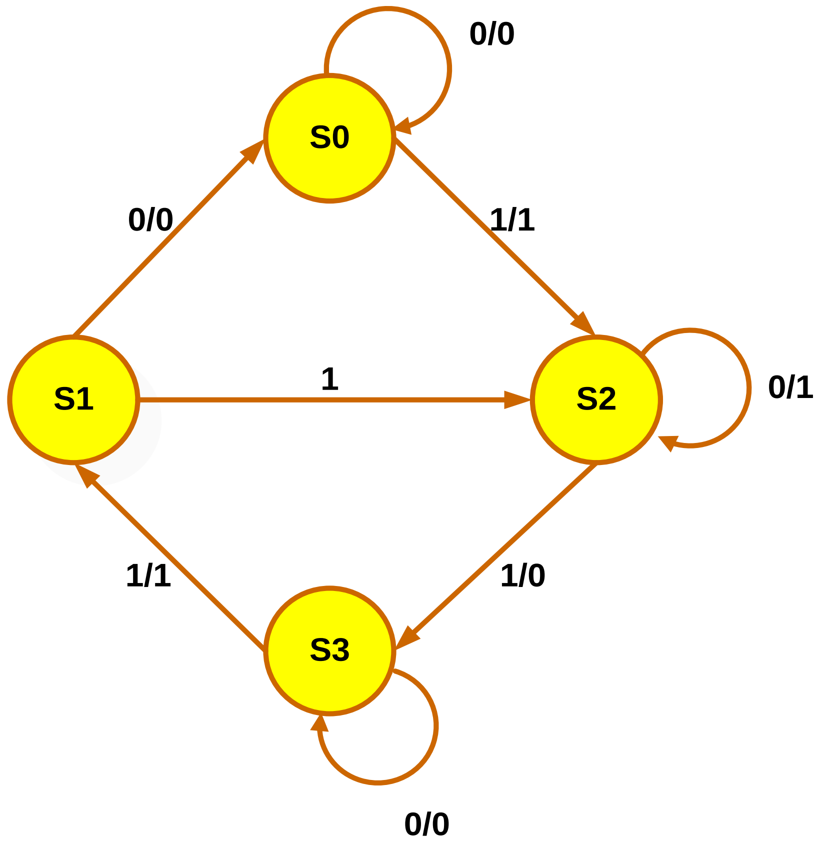
Example 9: Moore Machine (cont'd)

- Schematics of Synthesized logic:



Mealy Machine

State Diagram:



Transition table:

Present State	Next State		Output (z)	
	x = 0	x = 1	x = 0	x = 1
S0	S0	S2	0	1
S1	S0	S2	0	0
S2	S2	S3	1	0
S3	S3	S1	0	1

Example 10: Mealy Machine

- VHDL Code:

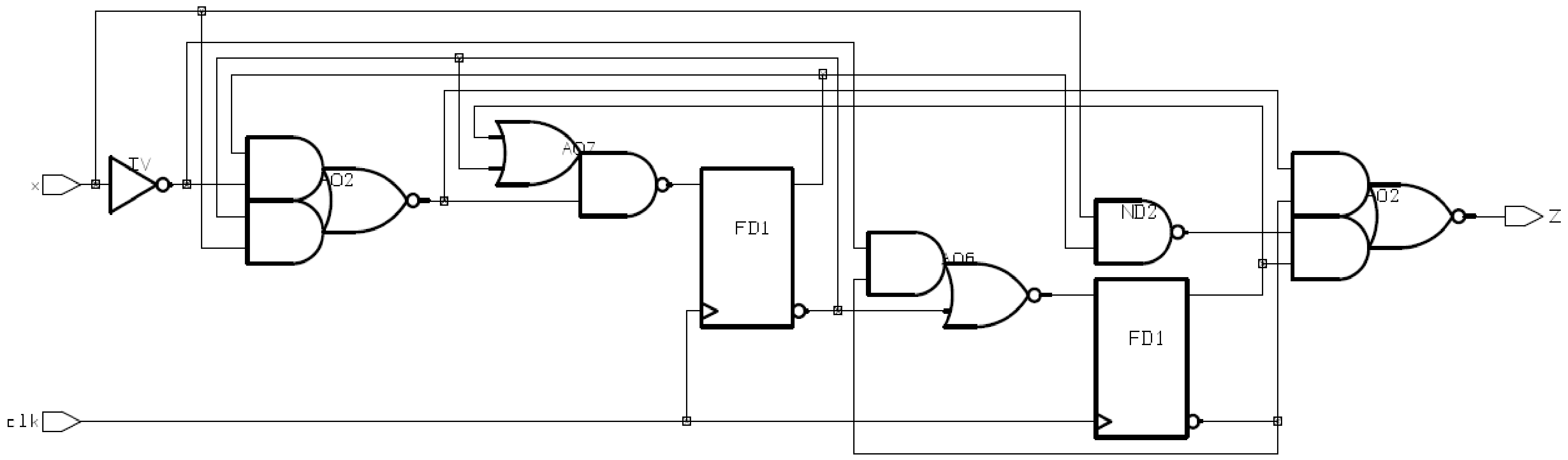
```
--Example 10: Mealy machine
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity mealy is
port( x, clk:in bit;
      Z:out bit);
end mealy;

architecture mealy_arch of mealy is
type state_type is (S0, S1, S2, S3);
signal current_state, next_state: state_type;

begin
 --process to hold synchronous elements (flip-flops)
 SYNCH:process
 begin
  wait until clk'event and clk='1';
  current_state <= next_state;
 end process;
```

```
 --process to hold combinational logic
 COMBIN:process (current_state, x)
 begin
  case current_state is
   when S0 =>
    if x='0' then
    Z <= '0';
    next_state <= S0;
   else
    Z <= '1';
    next_state <= S2;
   end if;
  when S1 =>
    if x='0' then
    Z <= '0';
    next_state <= S0;
   else
    Z <= '0';
    next_state <= S2;
   end if;
  when S2 =>
    if x='0' then
    Z <= '1';
    next_state <= S2;
   else
    Z <= '0';
    next_state <= S3;
   end if;
  when S3 =>
    if x='0' then
    Z <= '0';
    next_state <= S3;
   else
    Z <= '1';
    next_state <= S1;
   end if;
  end case;
 end process;
end mealy_arch;
```

Output is defined inside if-then-else statement

Example 10: Mealy Machine (cont'd)

- Schematics of Synthesized logic:



Asynchronous Modeling

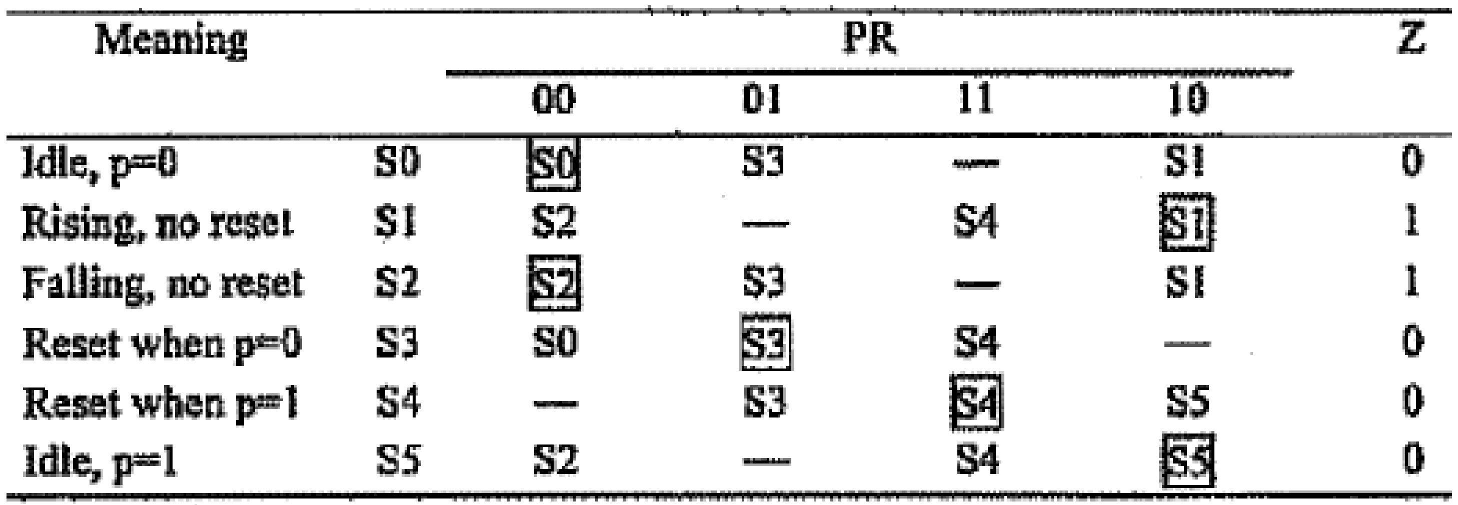
Asynchronous Design

- An asynchronous circuit is one in which synchronization is performed without a global clock

- Advantages:
  - Elimination of clock skew problems.
  - Average-case performance.
  - Adaptivity to processing and environmental variations.
  - Component modularity and reuse.
  - Lower system power requirements.
  - Reduced noise.

Example 11: Asynchronous Design

- Example 11: Design an asynchronous sequential circuit with two inputs P (pulse) and R (reset), and a single output Z that is normally 0. The output should be set to 1 whenever a 0 →1, or 1 → 0 transition occurs on P, and should be reset to 0 whenever R is 1
  - Flow Table (non-optimized)



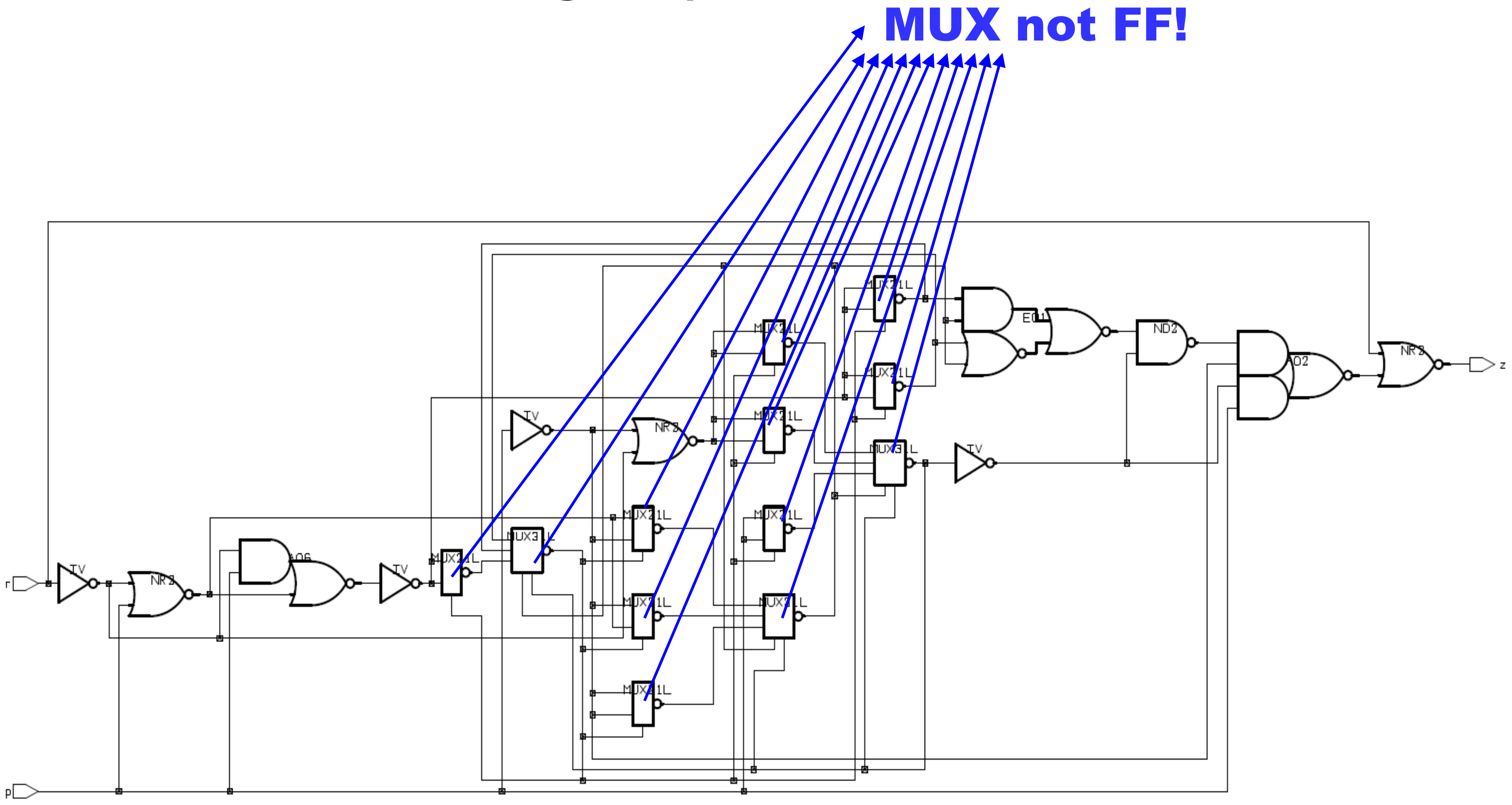
Meaning		PR				Z
		00	01	11	10	
Idle, p=0	S0	$\boxed{S0}$	S3	—	S1	0
Rising, no reset	S1	S2	—	S4	$\boxed{S1}$	1
Falling, no reset	S2	$\boxed{S2}$	S3	—	S1	1
Reset when p=0	S3	S0	$\boxed{S3}$	S4	—	0
Reset when p=1	S4	—	S3	$\boxed{S4}$	S5	0
Idle, p=1	S5	S2	—	S4	$\boxed{S5}$	0

Example 11: VHDL Code

```
--Example 11: Asynchronous modeling in VHDL
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity asyn is
port (
  p,r: in std_logic;
  z: out std_logic);
end asyn;
architecture asyn_arch of asyn is
      type states is (s0, s1, s2, s3, s4, s5);

      signal currentstate : states:= s0;
    begin
        state_trans: process(p,r)
            begin
              case currentstate is
               when s0=>
                        if (p='0') and (r='0') then
                         currentstate<=s0;
                         z<='0';
                        elsif (p='0') and (r='1') then
                            currentstate<=s3;
                            z<='0';
                        elsif (p='1') and (r='1') then
                            currentstate<=s4;
                            z<='0';
                        else
                            currentstate<=s1;
                            z<='1';
                        end if;
               when s1=>
                        if (p='0') and (r='0') then
                         currentstate<=s2;
                         z<='1';
                        elsif (p='0') and (r='1') then
                            currentstate<=s3;
                            z<='0';
                        elsif (p='1') and (r='1') then
                            currentstate<=s4;
                            z<='0';
                        else
                            currentstate<=s1;
                            z<='1';
                        end if;
 when s2=>
         if (p='0') and (r='0') then
          currentstate<=s2;
          z<='1';
         elsif (p='0') and (r='1') then
             currentstate<=s3;
             z<='0';
         elsif (p='1') and (r='1') then
             currentstate<=s4;
             z<='0';
         else
             currentstate<=s1;
             z<='1';
         end if;
when s3=>
         if (p='0') and (r='0') then
          currentstate<=s0;
          z<='0';
         elsif (p='0') and (r='1') then
             currentstate<=s3;
             z<='0';
         elsif (p='1') and (r='1') then
             currentstate<=s4;
             z<='0';
         else
             currentstate<=s1;
             z<='1';
         end if;
when s4=>
         if (p='0') and (r='0') then
          currentstate<=s2;
          z<='1';
         elsif (p='0') and (r='1') then
             currentstate<=s3;
             z<='0';
         elsif (p='1') and (r='1') then
             currentstate<=s4;
             z<='0';
         else
             currentstate<=s5;
             z<='0';
         end if;
                when s5=>
                         if (p='0') and (r='0') then
                          currentstate<=s2;
                          z<='1';
                         elsif (p='0') and (r='1') then
                             currentstate<=s3;
                             z<='0';
                         elsif (p='1') and (r='1') then
                             currentstate<=s4;
                             z<='0';
                         else
                             currentstate<=s5;
                             z<='0';
                         end if;
              end case;
        end process state_trans;
end asyn_arch;
```

Example 11: Synthesis

- Schematics of logic synthesis:



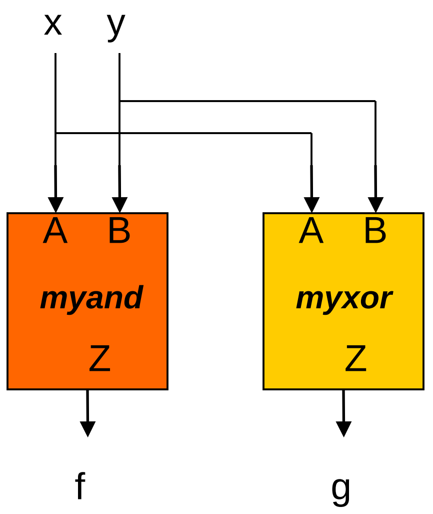
Parallelism and Interaction Among Units

Parallelism and Interaction Among Units

- VHDL concurrent statements allows parallelism among units.

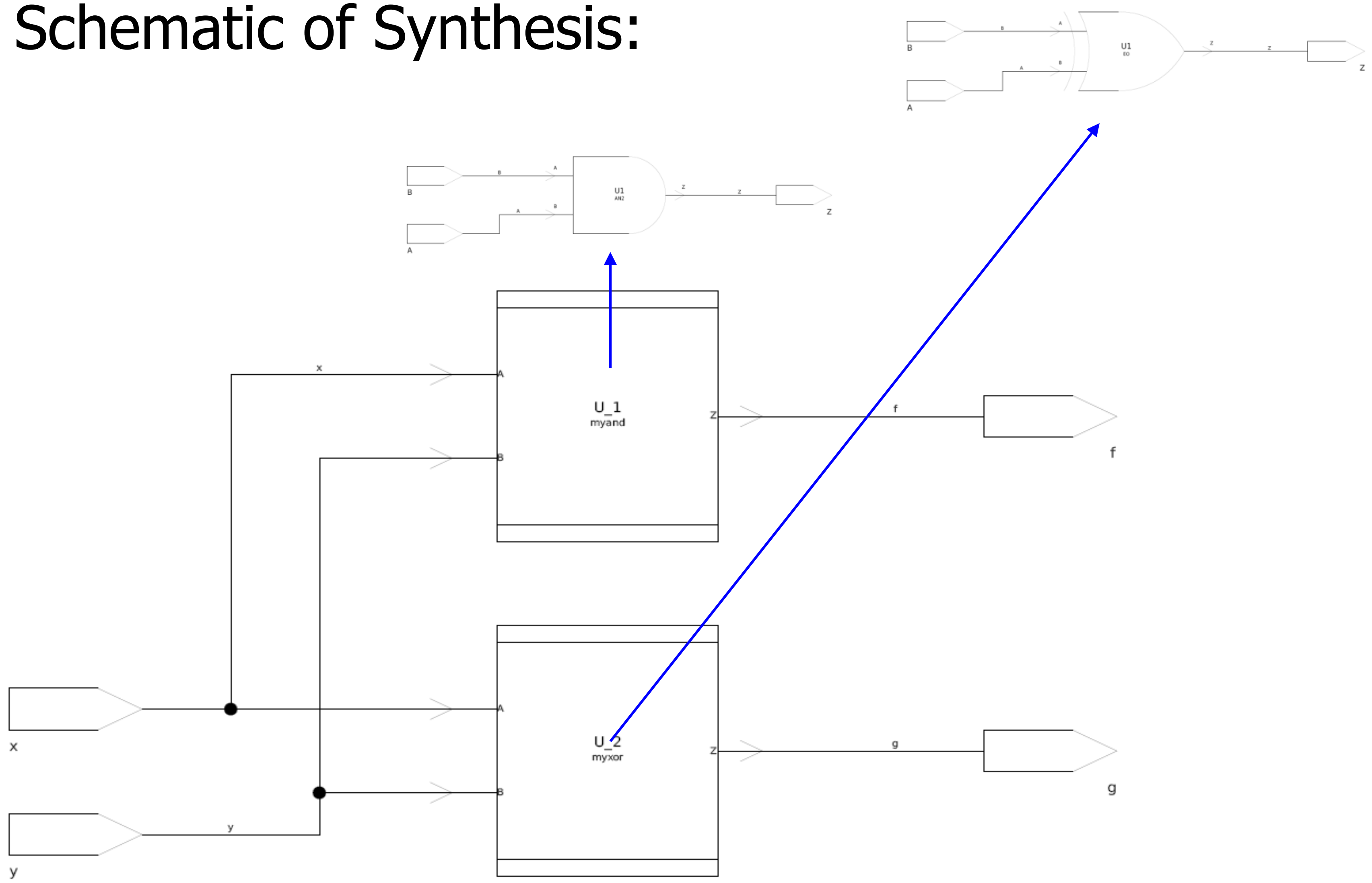
```
--Example 12: Parallelism & Interaction
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
--------------------------------------
entity myand is
port ( A, B : in std_logic;
Z : out std_logic);
end myand;
architecture myand_arch of myand is
begin
Z <= A and B after 10 ns;
end myand_arch;
--------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
entity myxor is
port ( A, B : in std_logic;
Z : out std_logic);
end myxor;
architecture myxor_arch of myxor is
begin
Z <= A xor B after 20 ns;
end myxor_arch;
```

```
--TOP LEVEL MODULE
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
ENTITY mydesign IS
PORT ( x, y : IN  std_logic ;
       f, g : OUT   std_logic ) ;
END mydesign;
ARCHITECTURE mydesign_arch OF mydesign IS
component myand
port ( A, B : in std_logic;
Z : out std_logic);
end component;
component myxor
port ( A, B : in std_logic;
Z : out std_logic);
end component;
BEGIN
U_1 : myand port map( A => x, B => y, Z => f);
U_2 : myxor port map( A => x, B => y, Z => g);
END;
```



Example 12: Synthesis

- Schematic of Synthesis:



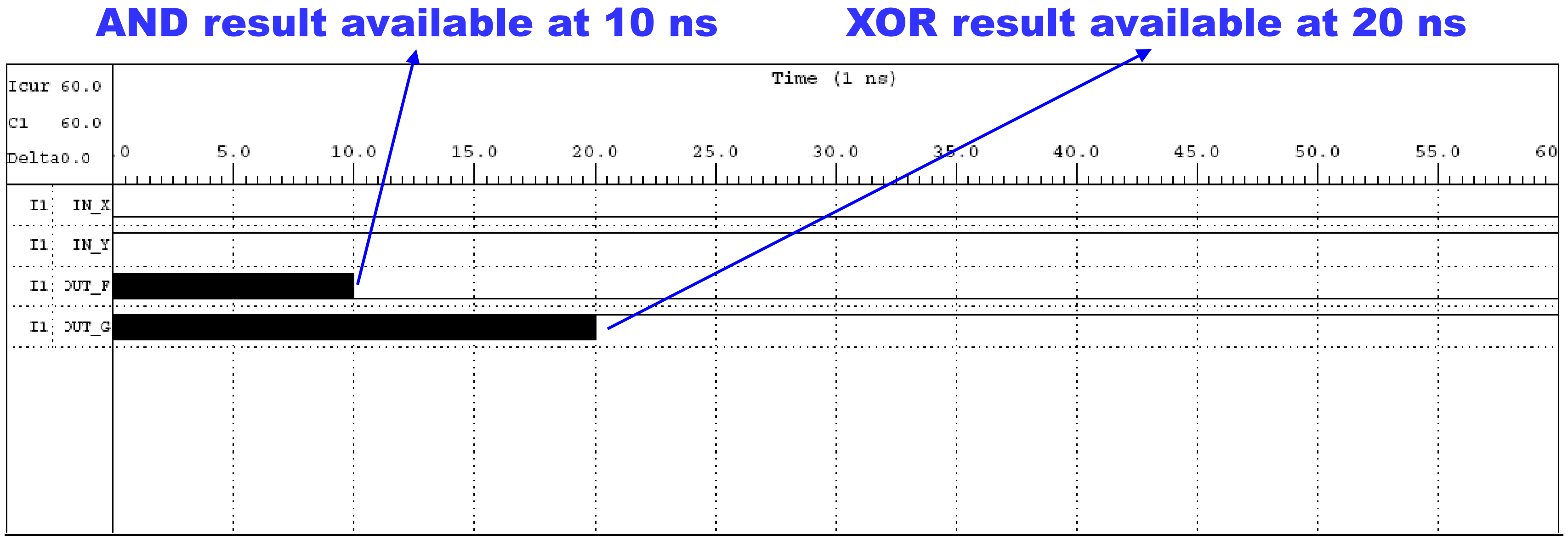
Example 12: Test Bench

- Test bench:

```
--Test bench for Example 12
library IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_components.all;
entity tbmydesign is
end tbmydesign;
architecture tbmydesign_arch of tbmydesign is
component myand
port ( A, B : in std_logic;
Z : out std_logic);
end component;
component myxor
port ( A, B : in std_logic;
Z : out std_logic);
end component;
component mydesign
PORT ( x, y :in std_logic;
       f, g  :out std_logic);
end component;
signal in_x, in_y, out_f, out_g: std_logic;
begin
imydesign:mydesign port map(x=>in_x, y=>in_y, f=>out_f, g=>out_g);
in_x<='0';
in_y<='1';
end tbmydesign_arch;
configuration cf_mydesign of tbmydesign is
for tbmydesign_arch
for imydesign:mydesign
use entity WORK.mydesign(mydesign_arch);
end for;
end for;
end cf_mydesign;
```

Example 12: Simulation

- Wave graph Simulation:



- Outputs observed at respective times

Implementing Memory in VHDL

Example 13: Memory Module in VHDL

```
--Example 12: A 4*4 RAM module
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity SRAM is
generic(        width:          integer:=4;
                depth:          integer:=4;
                addr:           integer:=2);
port( Clock:                    in std_logic;
      Enable:                   in std_logic;
      Read:                     in std_logic;
      Write:                    in std_logic;
      Read_Addr:                in std_logic_vector(addr-1 downto 0);
      Write_Addr:               in std_logic_vector(addr-1 downto 0);
      Data_in:  in std_logic_vector(width-1 downto 0);
      Data_out:                 out std_logic_vector(width-1 downto 0)
);
end SRAM;

architecture behav of SRAM is

type ram_type is array (0 to depth-1) of
      std_logic_vector(width-1 downto 0);
signal tmp_ram: ram_type;

begin
    -- Read Functional Section
    process(Clock, Read)
    begin
      if (Clock'event and Clock='1') then
          if Enable='1' then
                if Read='1' then
                    -- builtin function conv_integer change the type
                    -- from std_logic_vector to integer
                    Data_out <= tmp_ram(conv_integer(Read_Addr));
                else
                    Data_out <= (Data_out'range => 'Z');
                end if;
          end if;
      end if;
    end process;
```

```
    -- Write Functional Section
    process(Clock, Write)
    begin
      if (Clock'event and Clock='1') then
          if Enable='1' then
                if Write='1' then
                    tmp_ram(conv_integer(Write_Addr)) <= Data_in;
                end if;
          end if;
      end if;
    end process;

end behav;
```

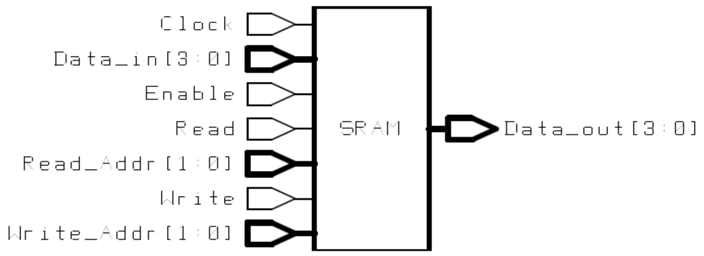
Write

Use of arrays

Read

Example 13: Synthesis

- Schematics of logic synthesis:


Clock
Data_in[3:0]
Enable
Read
Read_Addr[1:0]
Write
Write_Addr[1:0]
SRAM
Data_out[3:0]