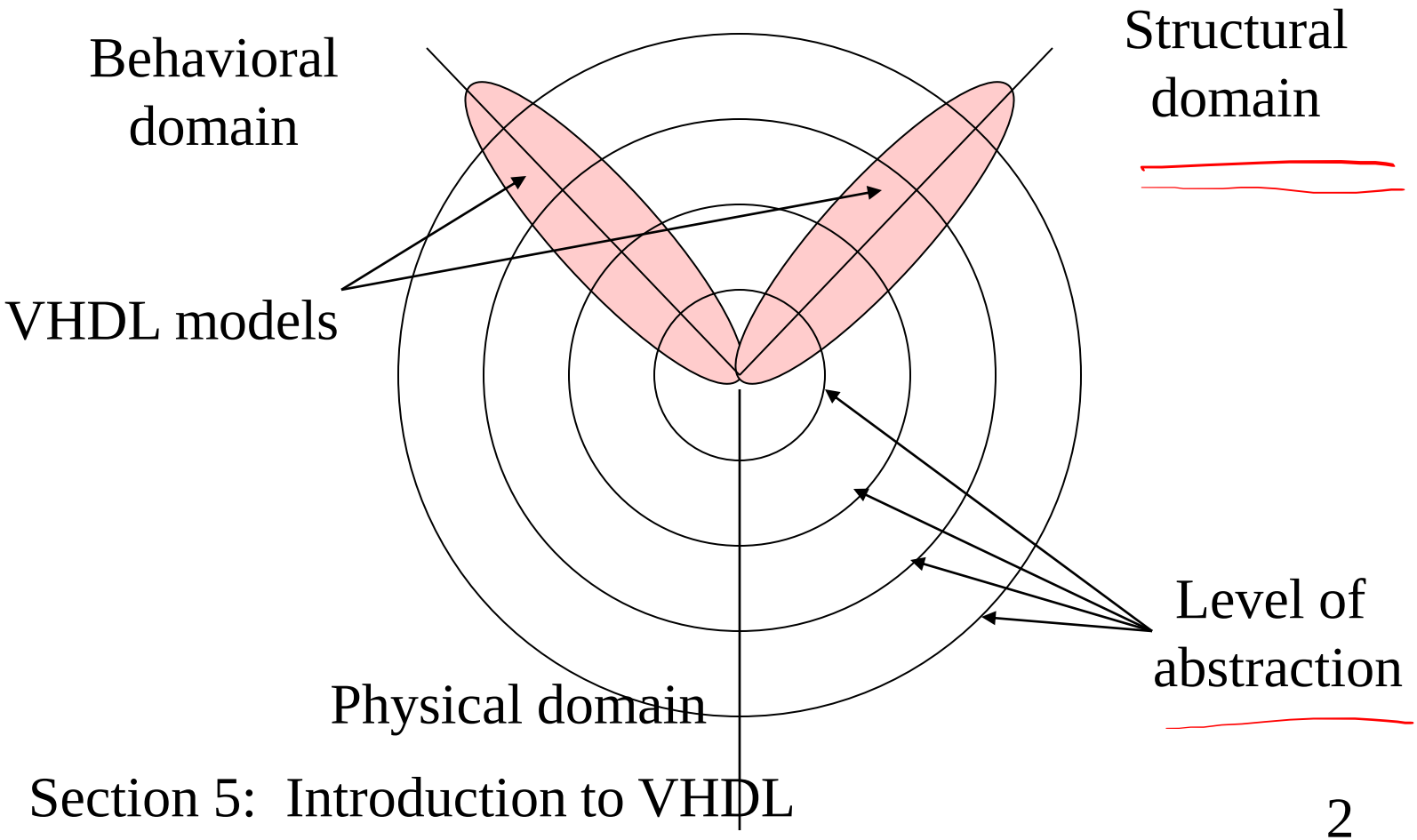
Lecture 27: Introduction to VHDL

M. Balakrishnan
Dept of Computer Science & Engg.
I.I.T. Delhi

Domains of Description : Gajski's Y-Chart



VHDL Development

- US DoD initiated in 80's
- Very High Speed ASIC Description Language
- Initial objective was modeling only and thus only a simulator was envisaged
- Subsequently tools for VHDL synthesis were developed

HDL Requirements

- Abstraction
- Modularity
- Hierarchy
- Concurrency

Concurrency in Hardware

Abstraction

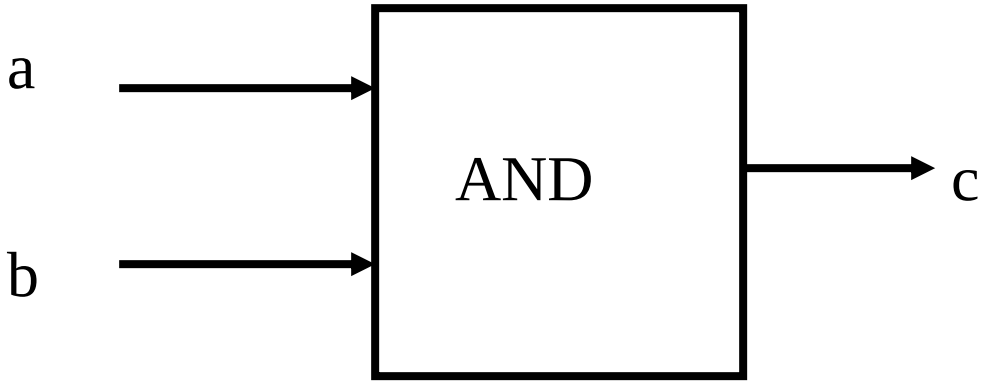
VHDL supports description of components as well as systems at various levels of abstraction

- Gate and component delays
- Clock cycles
- Abstract behavior without any notion of delays

Modularity

- Every component in VHDL is referred to as an entity and has a clear interface
- The interface is called an entity declaration
- The “internals” of the component are referred to as the architecture declaration
- There can be multiple architectures at even different levels of abstraction associated with the same entity

VHDL Example



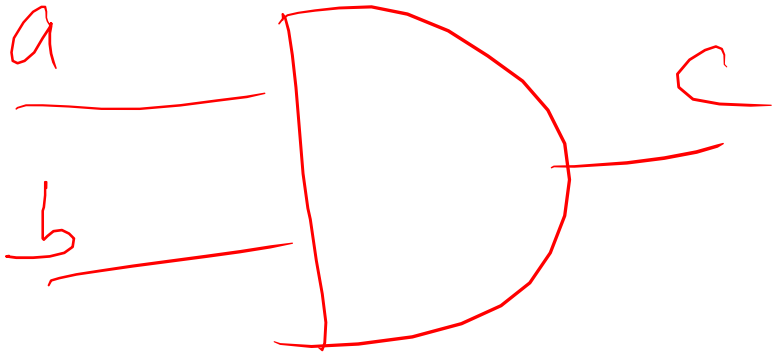
VHDL Description: AND gate

```
entity  AND2 is
  port (a, b: in bit ;
          c : out bit);
  end  AND2;

architecture  beh of AND2 is
begin
  c <= a and b;
end  beh;
```

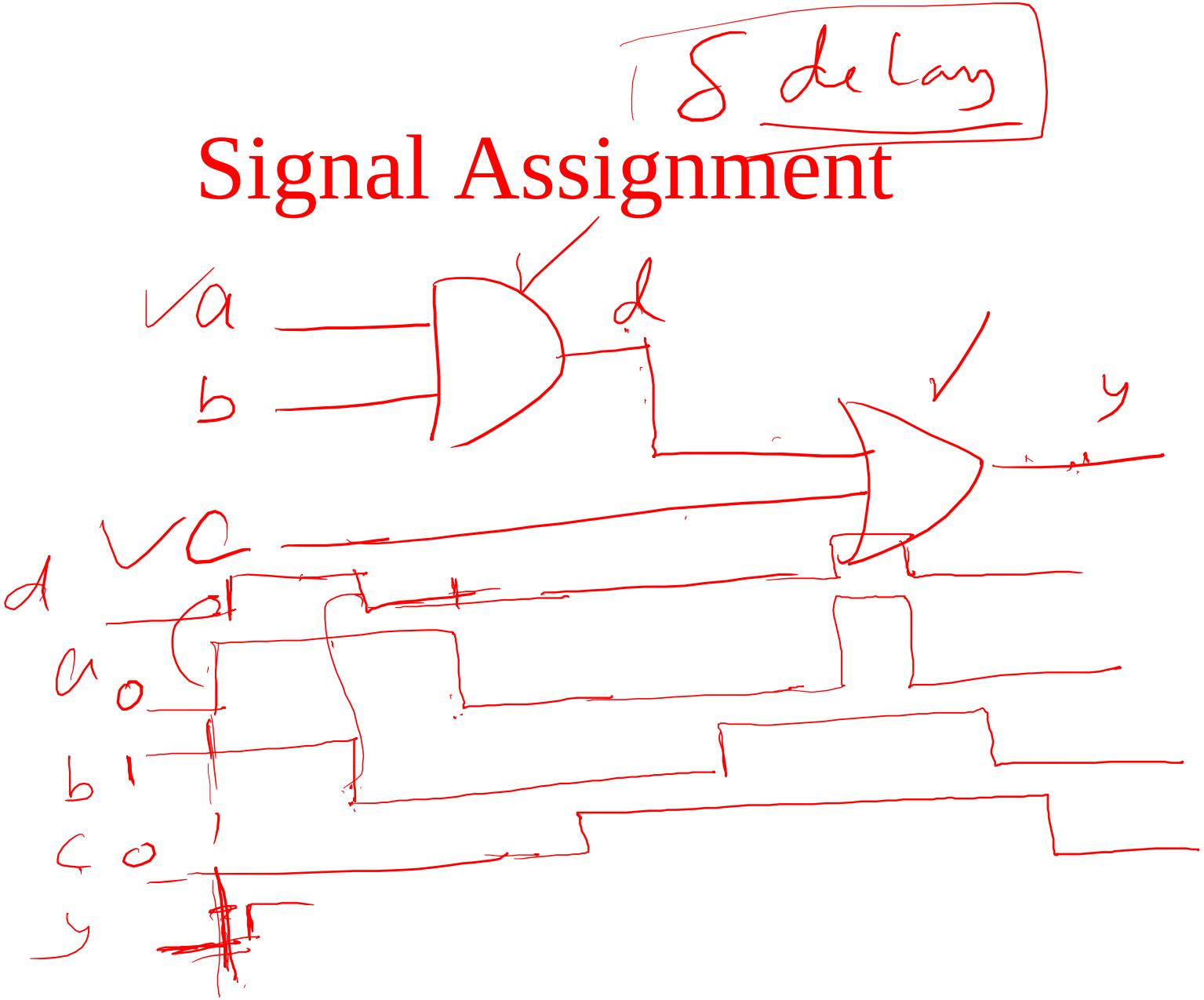
after 1ns;

Concurrent assignment

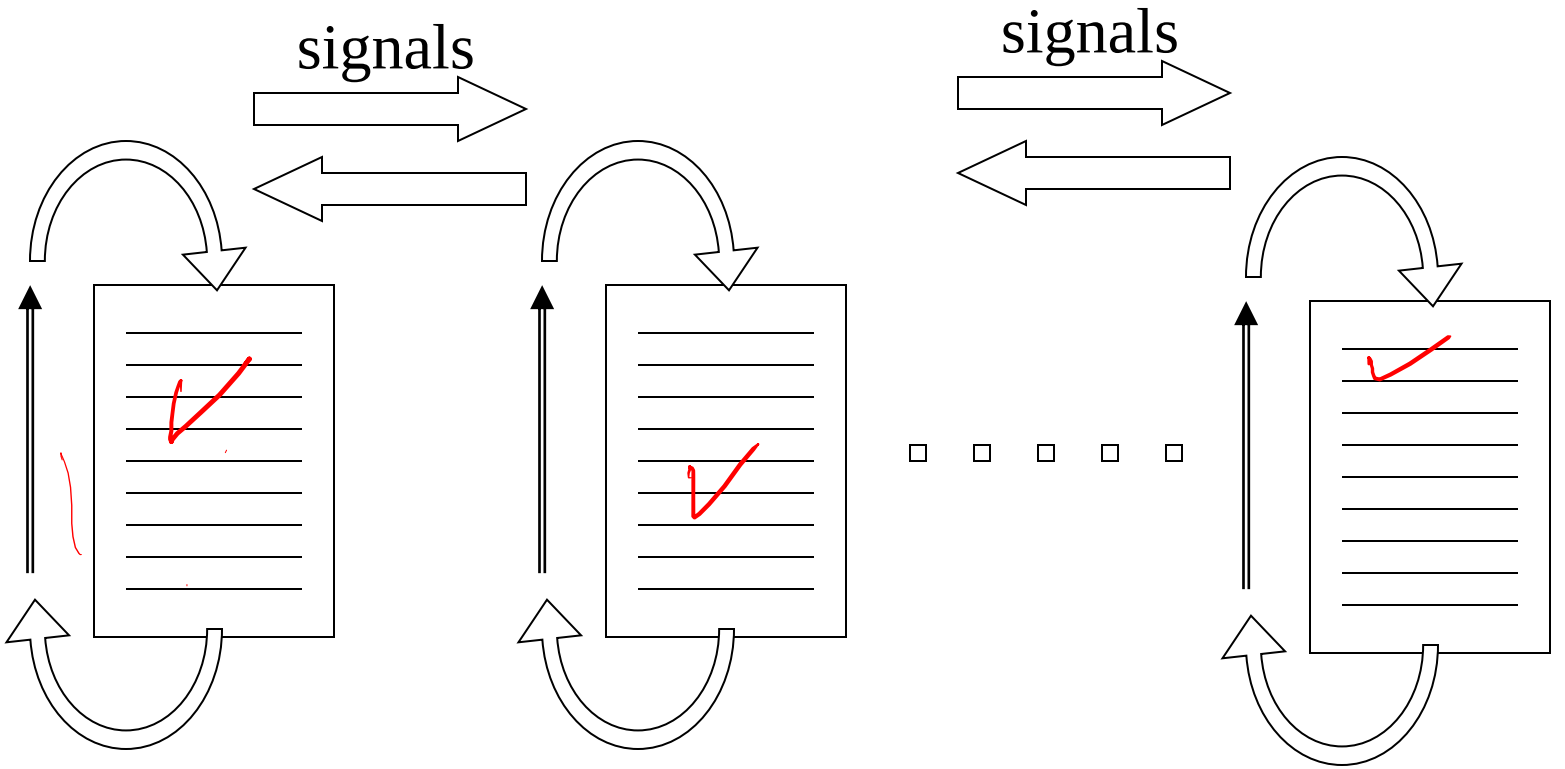


Events
Float Transaction

Signal Assignment



Concurrency in VHDL Descriptions



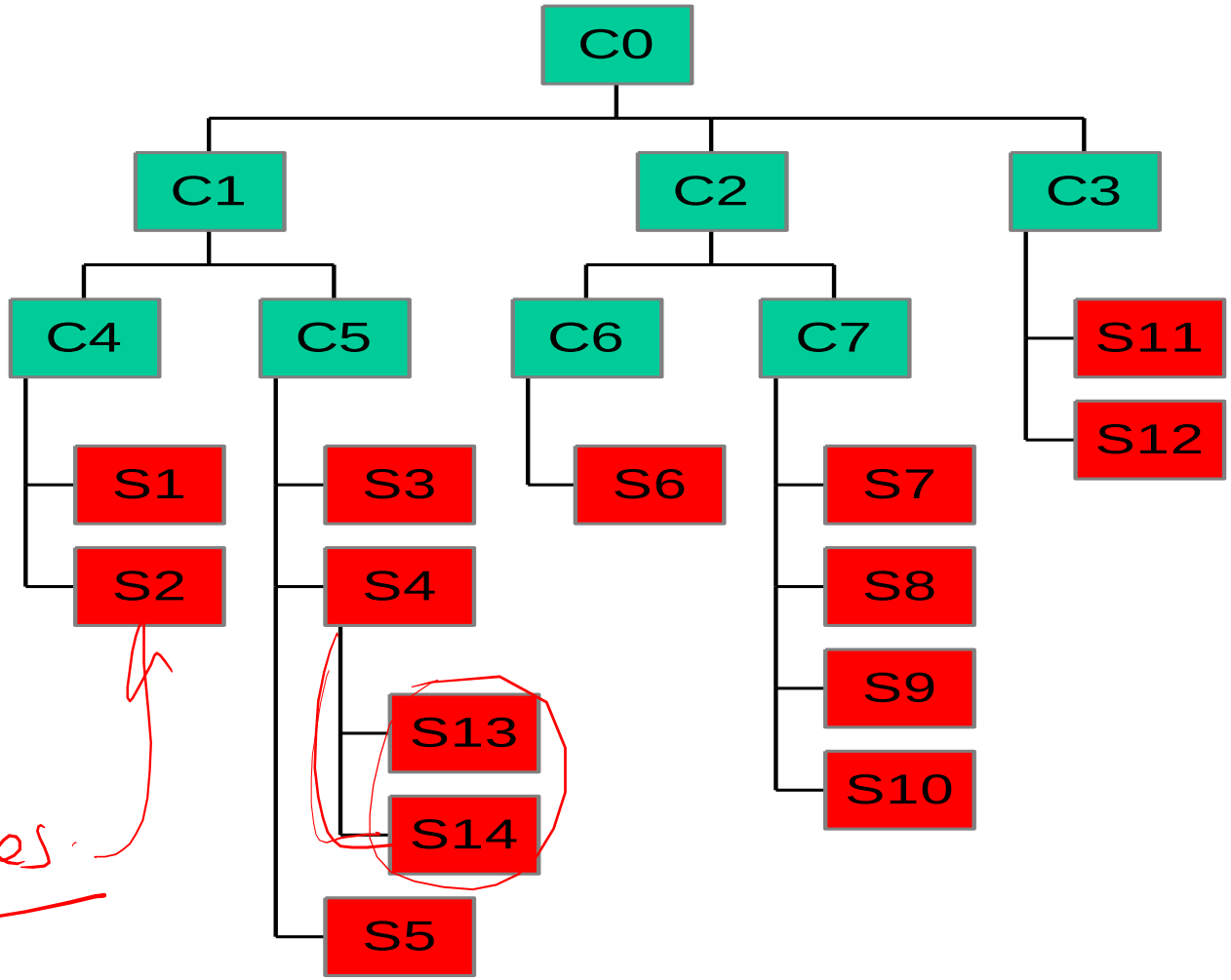
Concurrent and Sequential Computations

- Processes are concurrent
- Sequential activity within each process

Nesting of statements :

- Concurrent statements in a concurrent statement
- Sequential statements in a concurrent statement
- Sequential statements in a sequential statement

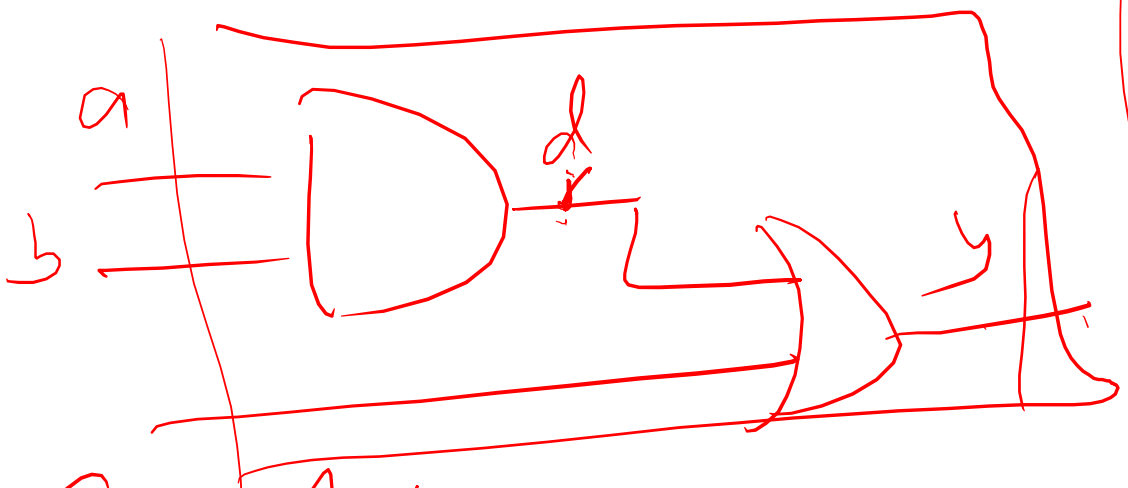
Hierarchy in VHDL



Signals

Variables

Worksheet



~~t+δ~~

t+δ, t+2δ

Arch

~~d <= a AND b;~~

y <= c OR d;

d <= a AND b;

Lecture 28: Modeling Styles in VHDL

M. Balakrishnan
Dept. of Comp. Sci. & Engg.
I.I.T. Delhi

Modeling Styles

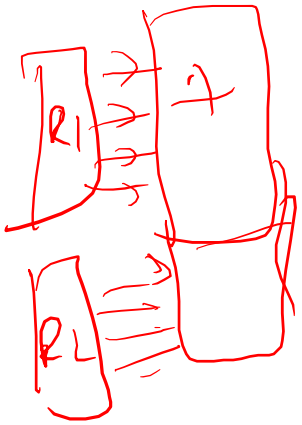
- Semantic model of VHDL
- Structural description
- Data Flow description
- Algorithmic description
- RTL description

Modeling Choices in VHDL

- Behavioral and Structural Domains
  - Several Levels of Abstraction
- Multiple Styles of Behavioral Description:
  - Data Flow Style (concurrent)
  - Procedural Style (sequential)
- Combinations, variations and special cases of these, e.g.,
  - special case of data flow style - FSM described using guarded blocks
  - special case of procedural style - FSM described using case statement in a process

Instantiation

Structural Description



- Carries same information as a NET LIST
- Net List = (Component instances) + (Nets)
- Structural Description in VHDL =
  (Signals) + (Component instances + Port maps)
- Many sophisticated features in VHDL to make it more versatile:
  - * Variety of signal types
  - * Generic components
  - * Generate statements for creating arrays of component instances
  - * Flexibility in binding components to design entities and architectures

Behavioral Description

- **Procedural**
  (textual order => execution order)
- **Sequential statements**
- **Control constructs alter normal sequential flow**

Called Behavioral description in VHDL

- **Non-procedural**
  (textual order NOT => execution order)
- **Concurrent statements**
- **Data flow (or rather data dependency restricts concurrency)**

Called Data flow description in VHDL

Concurrent Statements in VHDL

- process statement -- behavior
- concurrent procedure call -- behavior
- concurrent signal assign. -- data flow
- component instantiation -- structure
- generate statement -- structure
- block statement -- nesting
- concurrent assertion stmt -- error check

Example: 1-bit Full Adder

```
entity FullAdder is
    port (X, Y, Cin: in bit;      -- Inputs
          Cout, Sum: out bit);    -- Outputs
end FullAdder;
```

FA

Architecture

Example: 1-bit Full Adder (contd.)

Entity declaration

Architecture Equations **of** FullAdder **is**

begin -- Concurrent Assignment

Sum <= X **xor** Y **xor** Cin **after** 10 ns;

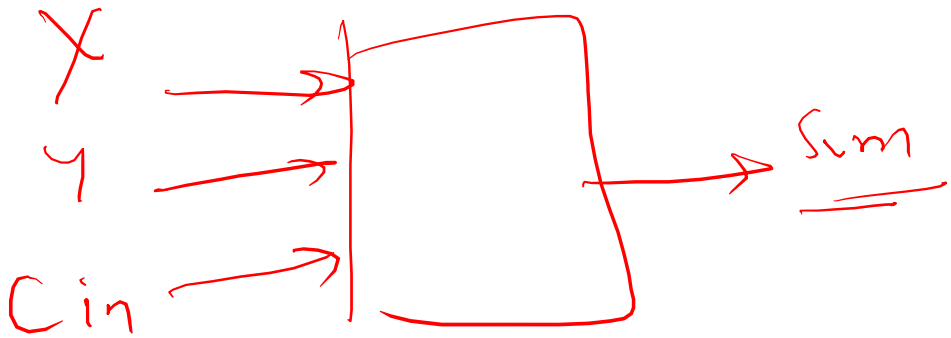
Cout <= (X **and** Y) **or** (X **and** Cin) **or** (Y **and** Cin) **after** 15 ns;

end Equations;

<v, t>

<value, time>

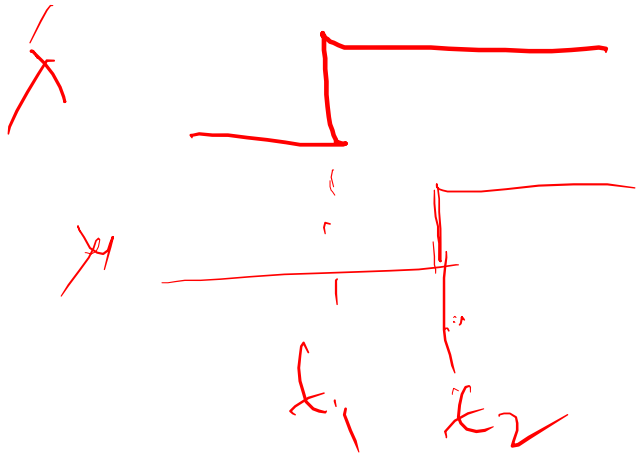
1-bit Full Adder Timing



$t_1 + \delta$

$\langle 0, t_1 + 10 \rangle$

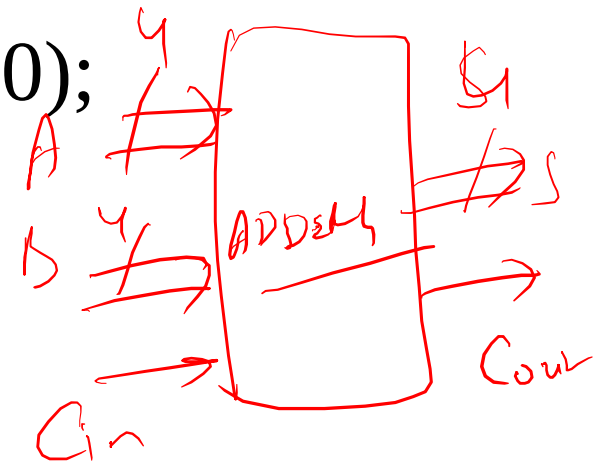
$\langle 0, t_2 + 10 \rangle$

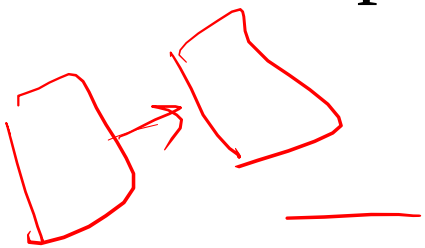


$t_2 - t_1 > 10ns$

Example: 4-bit Adder

```
entity Adder4 is
        port (A, B: in bit_vector(3 downto 0);
        Ci: in bit;              -- Inputs
        S: out bit_vector(3 downto 0);
        Co: out bit);            -- Outputs
end Adder4;
```





Example: 4-bit Adder (contd.)

```
Architecture Structure of  Adder4 is
Component FullAdder
    port (X, Y, Cin: in bit; Cout, Sum: out bit);
signal C: bit_vector (3 downto 1);
begin   -- Instantiations
    FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```

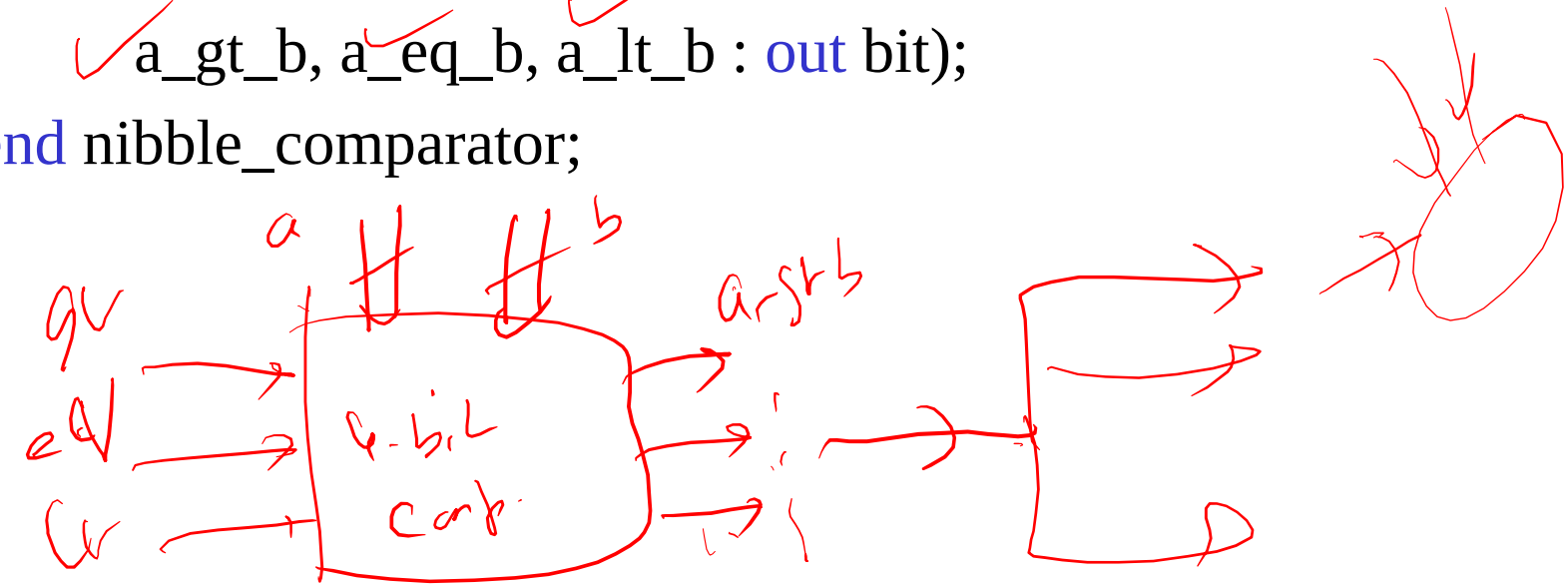
Positional mapping

C1 C2 C3

Structural

Example: 4-bit Comparator

```
entity nibble_comparator is
  port (a, b: in bit_vector (3 downto 0);
        gt,eq,lt : in bit;
        a_gt_b, a_eq_b, a_lt_b : out bit);
  end nibble_comparator;
```



Structural Description (contd.)

```
architecture  iterative of nibble_comparator is
  component comp1
    port (a, b, gt,eq,lt : in bit; a_gt_b, a_eq_b, a_lt_b : out bit);
  end component;
  for all : comp1 use entity work.bit_comparator(gate_level);
  signal im: bit_vector (0 to 8);
begin
  c0:comp1 port map(a(0),b(0), gt, eq, lt, im(0), im(1), im(2));
  c1toc2: for i in 1 to 2 generate
    c:comp1 port map(a(i),b(i),im(i*3-3),im(i*3-2),im(i*3-1),
                     im(i*3+0),im(i*3+1),im(i*3+2));
  end generate;
c3: comp1 port map(a(3),b(3),im(6),im(7),im(8),
                   a_gt_b, a_eq_b, a_lt_b);
end nibble_comparator;
```

Entity

arch

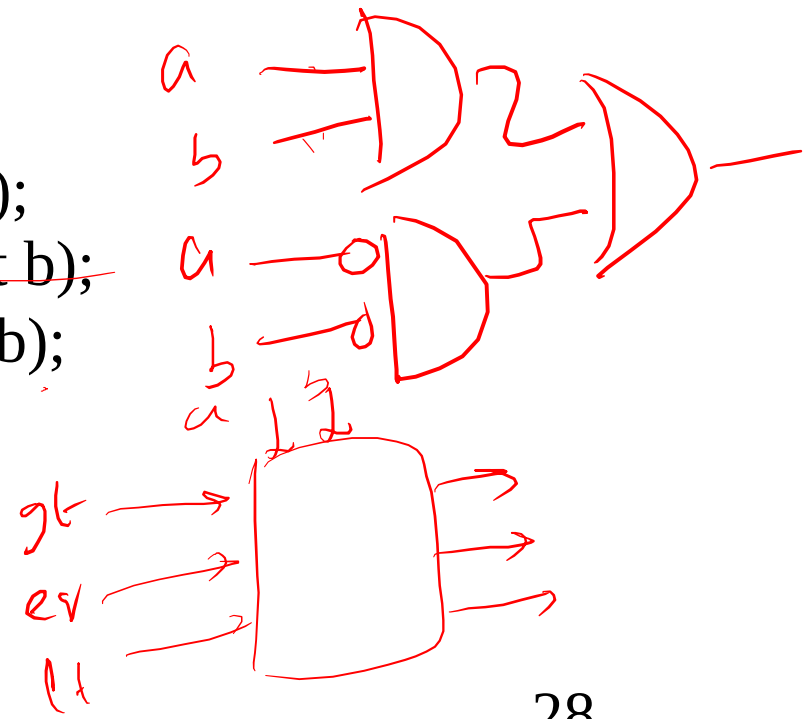
Spatial iteration

~~Temporal iteration~~

Example: 1-bit Comparator (data flow)

lsb → msb

```
entity comp1 is
   port (a, b, gt,eq,lt : in bit; a_gt_b, a_eq_b, a_lt_b : out bit);
  end comp1;
architecture  dataflow of comp1 is
signal s : bit;
begin
   s <= (a and b) or (not a and not b);
   a_gt_b <= (gt and s) or (a and not b);
   a_lt_b <= (lt and s) or (not a and b);
   a_eq_b <= eq and s;
end dataflow;
```

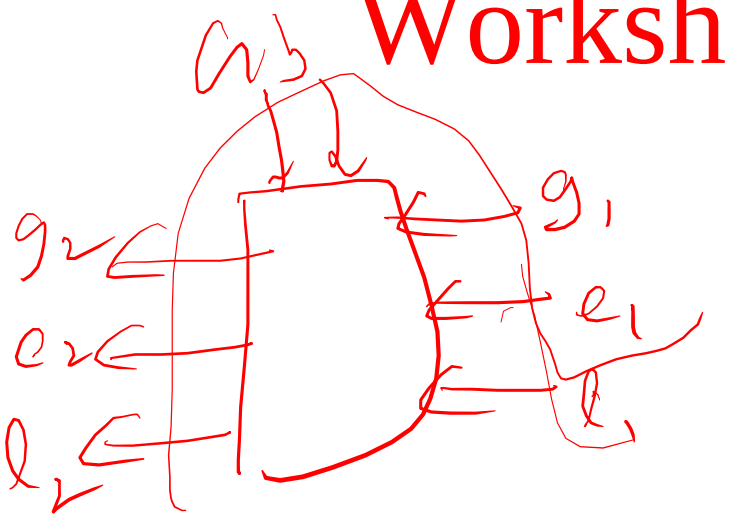


References

- Digital Systems Design Using VHDL
  Charles H. Roth, Jr., PWS Publishing Co.
  Chapter 2 (pp. 44 to 84)

- The Designer's Guide to VHDL
  Peter J. Ashenden, Morgon Kaufmann

Worksheet



$$g_2 = g_1 + e_1 \cdot \overline{a}\,\overline{b}$$

$$e_2 = e_1 \cdot (ab + \overline{a}\,\overline{b})$$

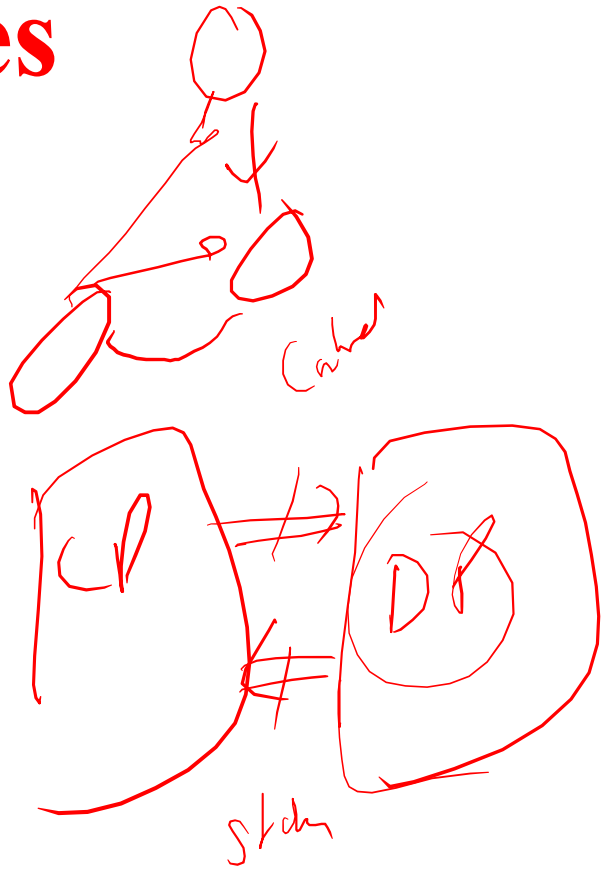
$$l_2 = l_1 + e_1 \cdot \overline{a}b$$

Lecture 29: Behavioral Description in VHDL

M. Balakrishnan
Dept. of Comp. Sci. & Engg.
I.I.T. Delhi

Modeling Styles

- Semantic model of VHDL
- Structural description
- Data Flow description
- Algorithmic description
- RTL description



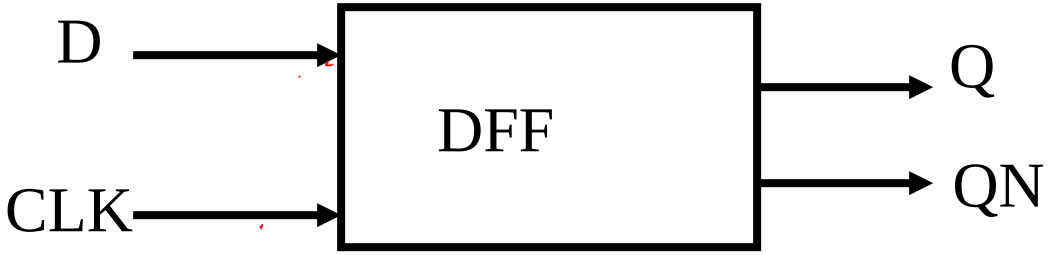
Concurrent Statements in VHDL

- process statement -- behavior
- concurrent procedure call -- behavior
- concurrent signal assign. -- data flow
- component instantiation -- structure
- generate statement -- structure
- block statement -- nesting
- concurrent assertion stmt -- error check

Example: D Flip-Flop

```
entity DFF is
   port (D, CLK: in bit;
         Q: out bit; QN: out bit := '1') ;
end  DFF;
```

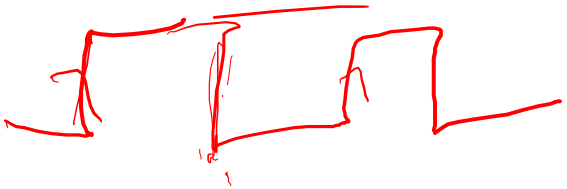
:=0)



Example: DFF (contd.)

sensitivity list

```
Architecture Beh of DFF is
begin  process (CLK)
        begin if (CLK = '1') then
                Q <= D after 10 ns;
                QN <= not D after 10 ns;
            endif;
        endprocess;
end Beh;
```

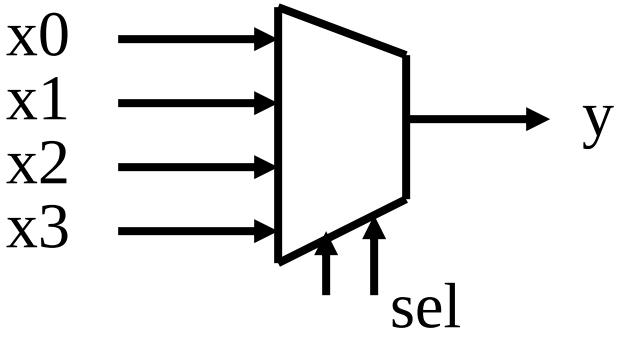


Timing Diagram

Data Flow & Process Statement

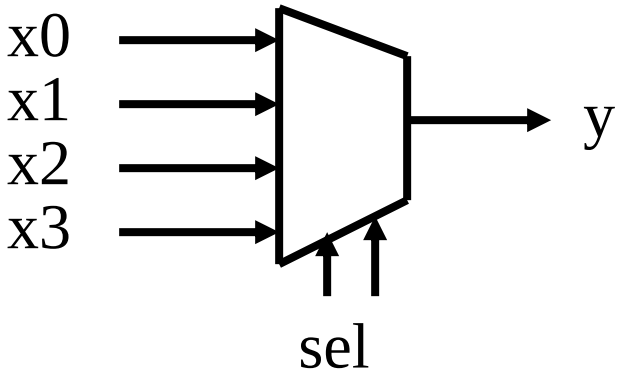
Concurrent Conditional Assignment: 4 to 1 Multiplexer

```
y <=     x0    when    sel = 0
    else x1    when    sel = 1
    else x2    when    sel = 2
    else x3    when    sel = 3
```



CASE Statement: 4 to 1 Multiplexer

```
Case sel is
   when 0 =>   y <= x0
   when 1 =>   y <= x1
   when 2 =>   y <= x2
   when 3 =>   y <= x3
end case
```



Note allowed only within a Process statement

Variables And Signals

multiple process

```
Architecture var of dummy is
    signal        trigger, sum: integer := 0;
begin   process
        variable var1: integer:= 1;
        variable var3, var2: integer:= 2;
        begin   wait on trigger;
                var3 := var1 + var2;
                var1 := var3;
                sum <= var1;
        end process; end var;
```

:bit;

<=

:=

var3 3

var1 3

sum 3

a<=b+c;

Variables and Signals

```
Architecture sig of dummy is
    signal      trigger, sum: integer := 0;
    signal      sig1: integer:= 1;
    signal      sig3, sig2: integer:= 2;
begin  process
        begin  wait on trigger;
               sig3 <= sig1 + sig2;
               sig1 <= sig3;
               sum <= sig1;
        end process; end sig;
```

Inertial and Transport Delays

y1 <= **transport** x **after** 5 ns; -- transport delay

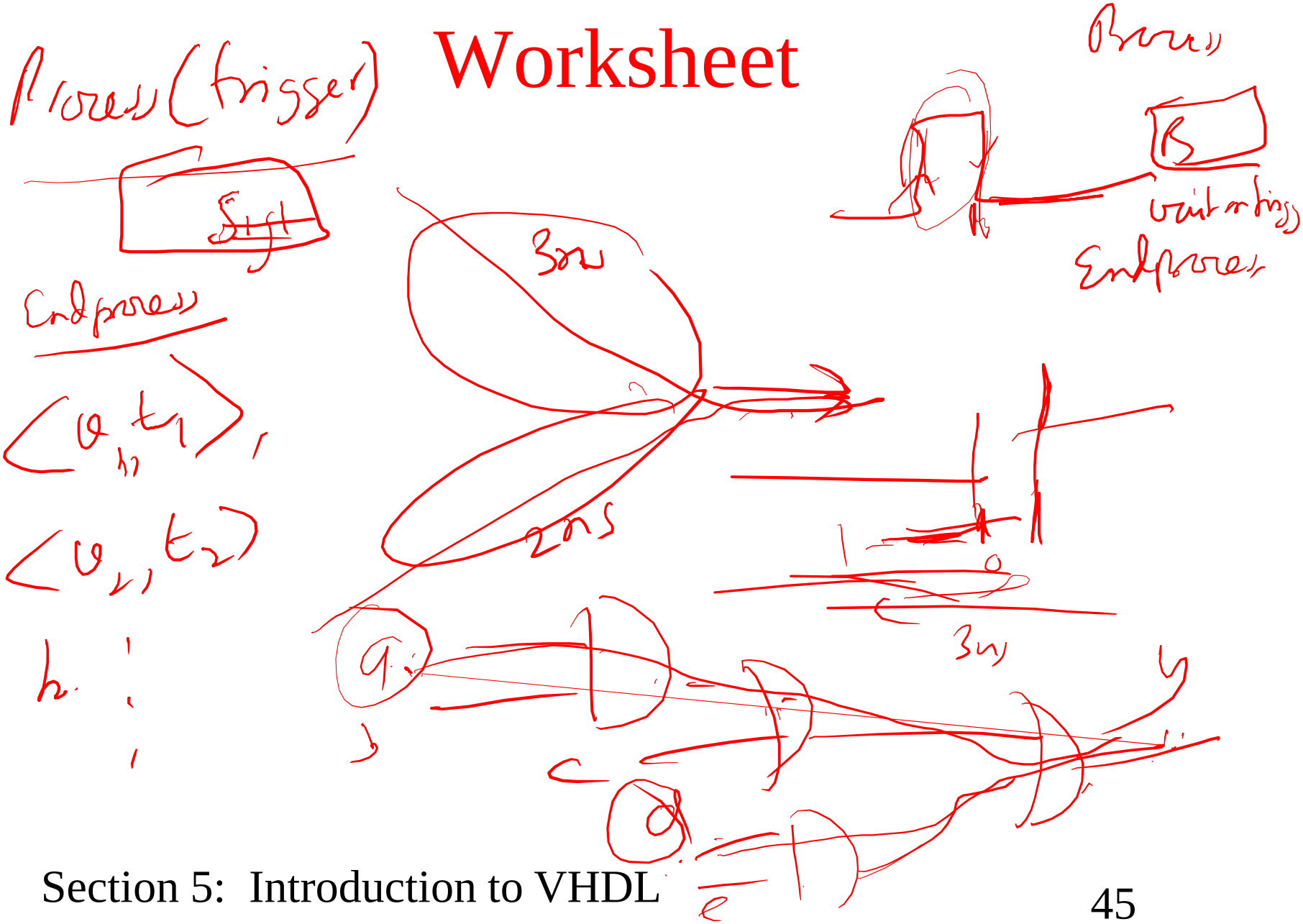
y2 <= x **after** 5 ns; -- inertial delay

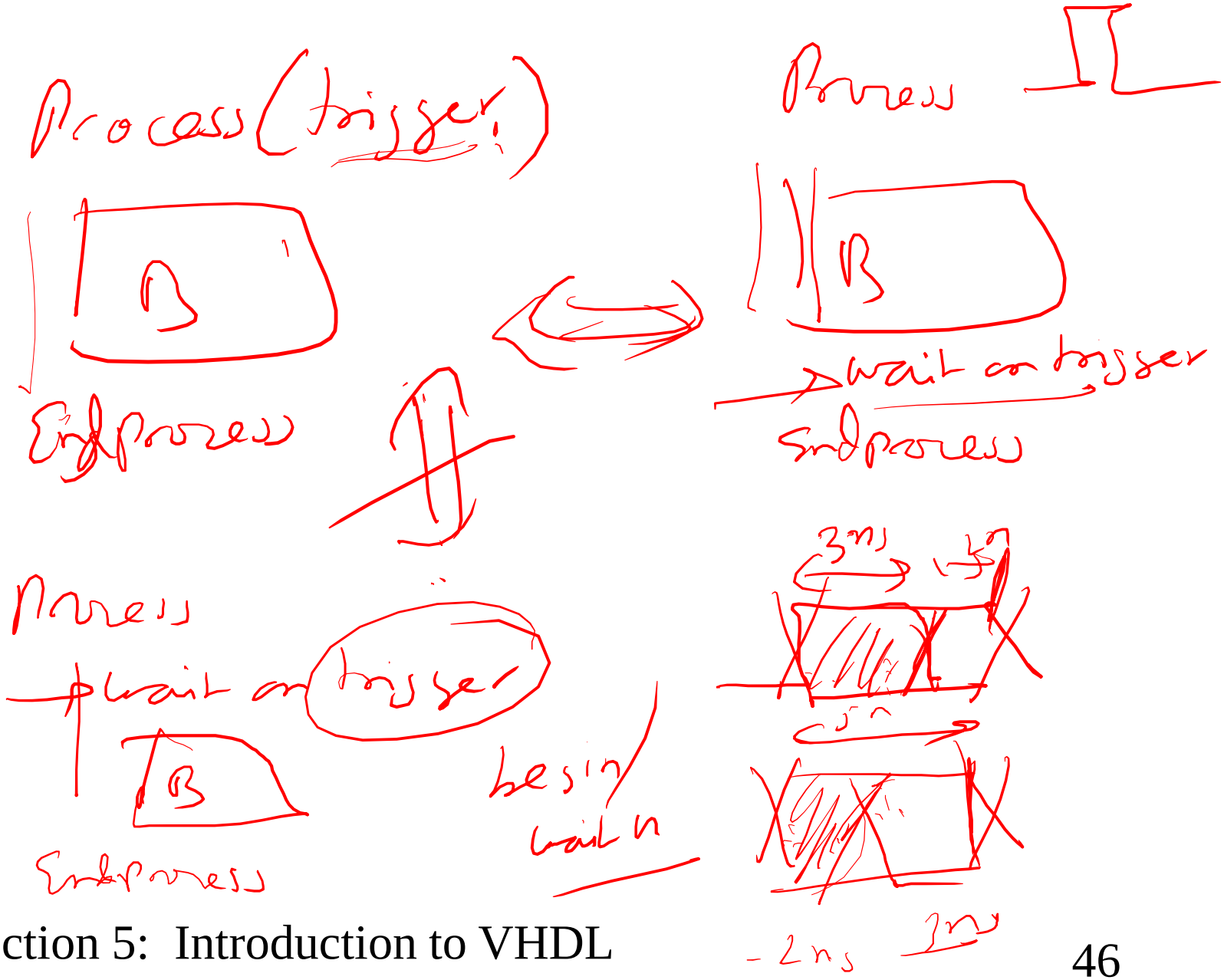
y3 <= **reject 2 ns** x **after** 5 ns; -- mix of inertial and transport delay but rejection of pulse less than 2 ns

Inertial & Transport Delays (contd)

- Inertial delay can reject narrow input pulses/ spikes whereas transport delay would preserve them in the output

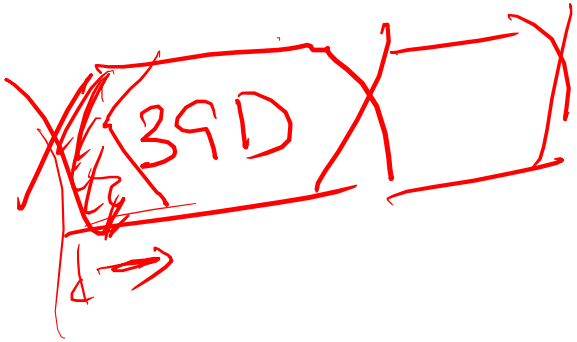
Worksheet


Process (trigger)
Endprocess
Process
Endprocess
3ns
2ns
3ns

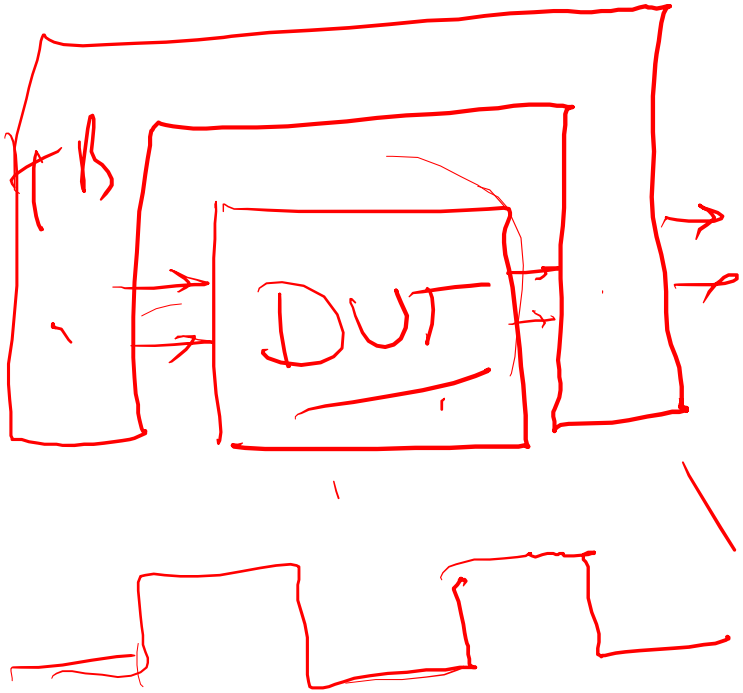

Process (trigger)
B
Endprocess
Process
B
wait on trigger
Endprocess
Process
wait on trigger
B
Endprocess
begin
wait n
3ns
1ns
2ns
-2ns
3ns

Worksheet

Synthesize



Testbench




```
process
a <= 0;
wait for 5ns
a <= 1
wait for 5ns
end process
```