

Synthesis of Digital Systems (COL719)

Hardware Modelling and VHDL

Preeti Ranjan Panda

Department of Computer Science and Engineering
Indian Institute of Technology Delhi

VHDL

VHSIC [Very High Speed Integrated Circuit]
Hardware Description Language

Contents

- Introduction
- Signal assignment
- Modelling delays
- Describing behaviour
- Structure, test benches, libraries, parameterisation
- Standards

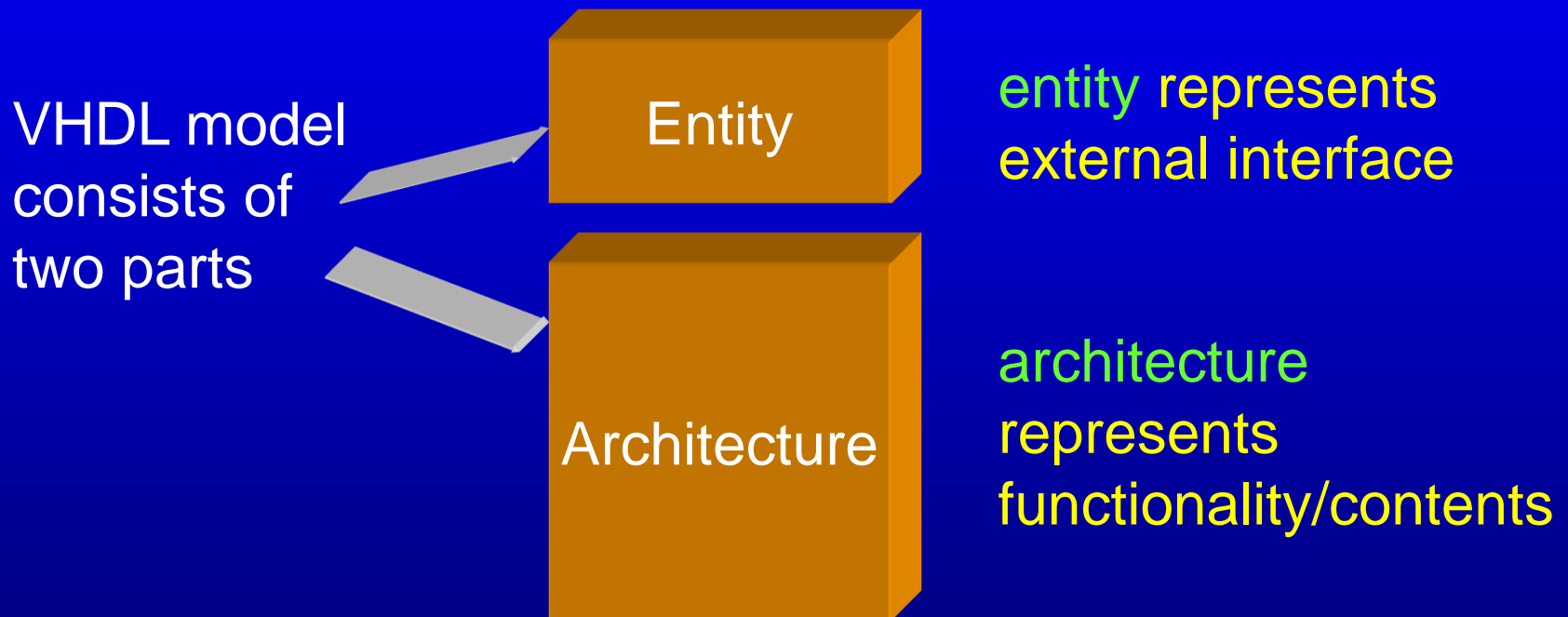
Hardware Description Languages

- VHDL - VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
 - originally meant for simulation/documentation
 - syntax based on Ada/Pascal
- Verilog
 - syntax based on C
- SystemC
 - based on C++
 - system level language

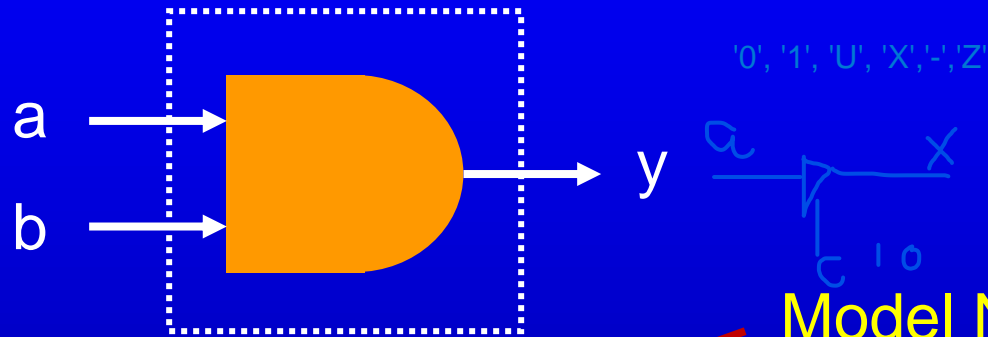
Which HDL to use?

- Coverage of Hardware concepts
 - equally good
- Learning one language eases learning of the other
 - most differences are minor/syntactic
- Status of tool support
 - equally good for both VHDL/Verilog
 - SystemC used more in system level modelling

Fundamental VHDL Objects: entity/architecture pairs



Specifying interfaces: entities and ports



Entity has
interface only.
No functionality.

```
ENTITY and_gate IS
PORT (a: IN BIT;
      b: IN BIT;
      y: OUT BIT);
END and_gate;
```

Port Name

Model Name

Port direction

Port type

Specifying Functionality: architectures

```
ARCHITECTURE data_flow OF and_gate IS  
BEGIN  
  y <= a AND b;  
END data_flow;
```

May have multiple architectures for given entity

- different views
- different levels of detail

Contents

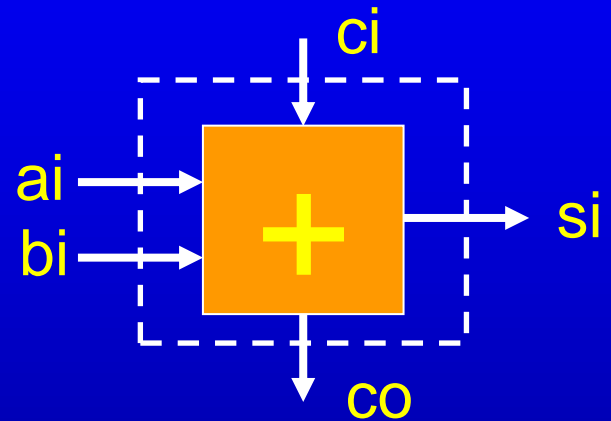
- Introduction
- Signal assignment
- Modelling delays
- Describing behaviour
- Structure, test benches, libraries, parameterisation
- Standards

Specifying Concurrency: Concurrent Signal Assignment

```
ARCHITECTURE data_flow  
OF full_adder IS  
BEGIN
```

```
    si <= ai XOR bi XOR ci;  
    co <= (ai AND bi) OR (bi AND ci)  
          OR (ai AND ci);
```

```
END data_flow;
```



Concurrent Signal Assignments

When is Signal Assignment Executed?

Assignment executed when any signal on RHS changes

```
ARCHITECTURE data_flow  
OF full_adder IS  
BEGIN  
  si <= ai XOR bi XOR ci;  
  co <= (ai AND bi) OR (bi AND ci)  
        OR (ai AND ci);  
END data_flow;
```

Executed when
ai, bi, or ci changes

Executed when
ai, bi, or ci changes

Order of Execution

- Execution independent of specification order

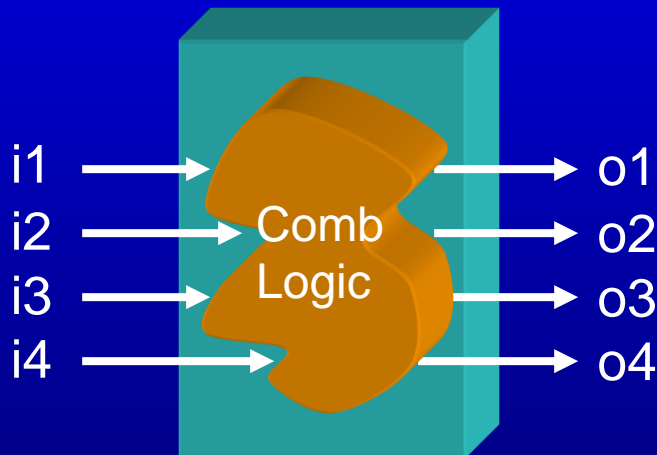
```
ARCHITECTURE data_flow
OF full_adder IS
BEGIN
  si <= ai XOR bi XOR ci;
  co <= (ai AND bi) OR (bi AND ci)
        OR (ai AND ci);
END data_flow;
```

```
ARCHITECTURE data_flow
OF full_adder IS
BEGIN
  co <= (ai AND bi) OR (bi AND ci)
        OR (ai AND ci);
  si <= ai XOR bi XOR ci;
END data_flow;
```

These two are equivalent

Modelling Combinational Logic

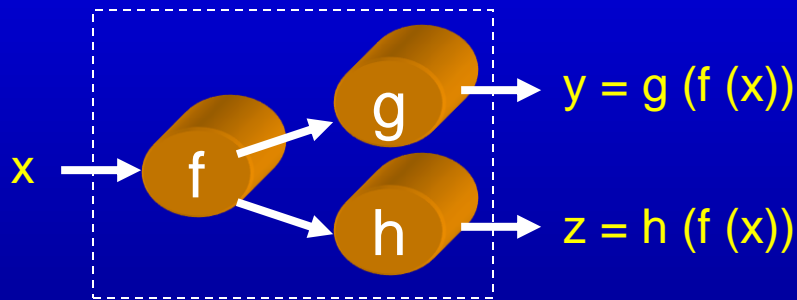
- One concurrent assignment for each output



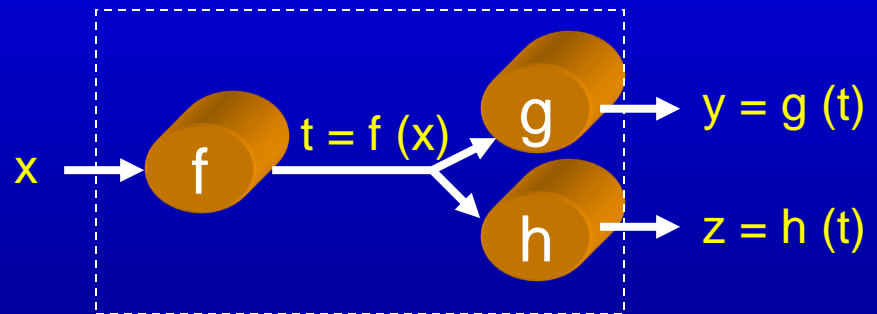
```
ARCHITECTURE data_flow  
OF comb_logic IS  
BEGIN  
  o1 <= i1 and i2;  
  o2 <= (i2 or i3) xor (i1 and i4);  
  o3 <= ...;  
  o4 <= ...;  
END data_flow;
```

When Logic Complexity Increases

- Temporary SIGNALS needed
- Avoid redundant evaluations



Ports: x, y, z



Signal: t

SIGNALS

- Represent intermediate wires/storage
- Internal - not visible outside entity

```
ENTITY comb_logic IS
PORT (i1, i2, i3, i4: IN BIT;
      o1, o2: OUT BIT);
END comb_logic;

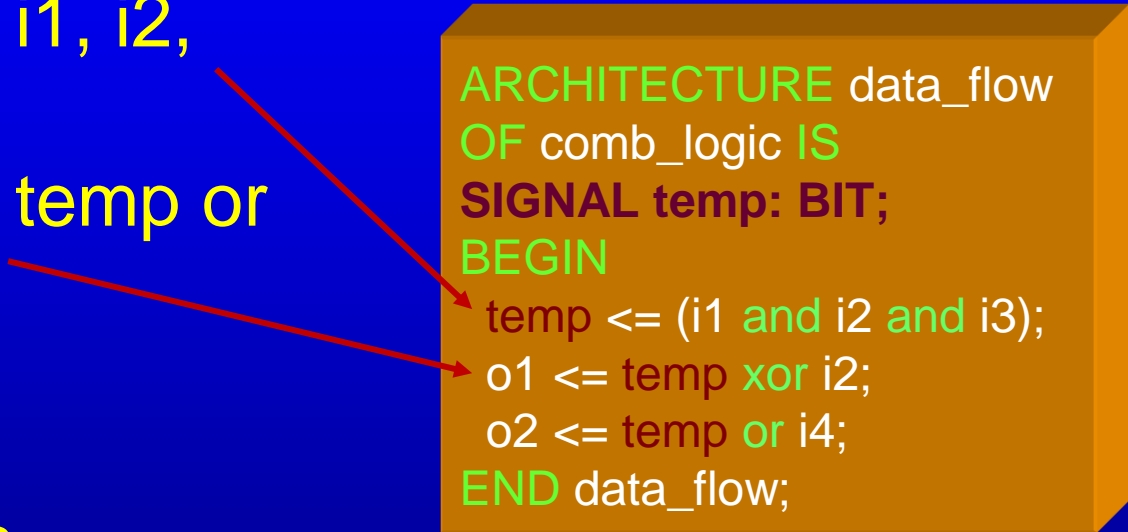
ARCHITECTURE data_flow
OF comb_logic IS
BEGIN
  o1 <= (i1 and i2 and i3) xor i2;
  o2 <= (i1 and i2 and i3) or i4;
END data_flow;
```

```
ENTITY comb_logic IS
PORT (i1, i2, i3, i4: IN BIT;
      o1, o2: OUT BIT);
END comb_logic;

ARCHITECTURE data_flow1
OF comb_logic IS
  SIGNAL temp: BIT;
BEGIN
  temp <= (i1 and i2 and i3);
  o1 <= temp xor i2;
  o2 <= temp or i4;
END data_flow;
```

SIGNALS

- executed when i1, i2, or i3 changes
- executed when temp or i2 changes
- SIGNALS are associated with time/waveforms
- PORT is a special type of SIGNAL



```
ARCHITECTURE data_flow  
OF comb_logic IS  
  SIGNAL temp: BIT;  
BEGIN  
  temp <= (i1 and i2 and i3);  
  o1 <= temp xor i2;  
  o2 <= temp or i4;  
END data_flow;
```

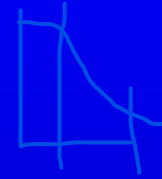
Two red arrows originate from the list on the left. One arrow points from the text 'i2' in the first bullet point to the 'i2' in the expression '(i1 and i2 and i3)'. The other arrow points from the text 'temp' in the second bullet point to the 'temp' in the assignment 'temp <= temp xor i2'.

Contents

- Introduction
- Signal assignment
- Modelling delays
- Describing behaviour
- Structure, test benches, libraries, parameterisation
- Standards

Modelling Delays: inertial delay

- Models gate delays
- Spikes suppressed



```
y <= INERTIAL NOT a AFTER 10 ns;  
y <= NOT a AFTER 10 ns; -- inertial delay is default
```



Modelling Delays: transport delay

- Models wires/transmission lines
 - used in more abstract modelling
- Spikes propagated

```
y <= TRANSPORT NOT a AFTER 10 ns;
```



Events and Transactions

- Event
 - Signal assignment that causes change in value
- Transaction
 - Value scheduled for signal assignment
 - may or may not cause change in value

Events and Transactions: Example

```
ARCHITECTURE demo OF example IS  
  SIGNAL a, b, c: BIT := '0';  
BEGIN  
  a <= '1' AFTER 15 NS;  
  b <= NOT a AFTER 5 NS;  
  c <= a AFTER 10 NS;  
END demo;
```

Source: Z. Navabi, VHDL - analysis and modeling of digital systems

Events and Transactions: Example

ARCHITECTURE demo OF
example IS

SIGNAL a, b, c: BIT := '0';

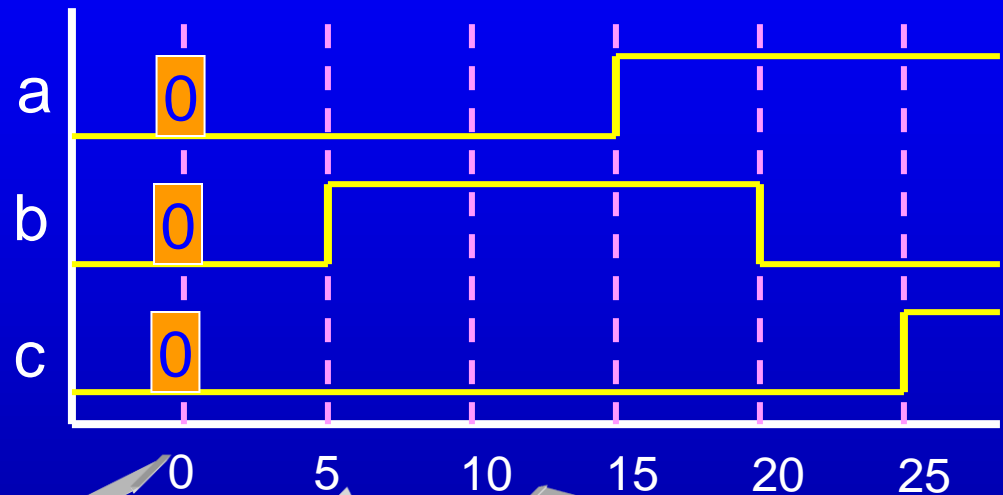
BEGIN

a <= '1' AFTER 15 NS;

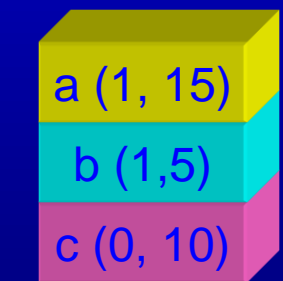
b <= NOT a AFTER 5 NS;

c <= a AFTER 10 NS;

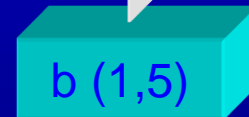
END demo;



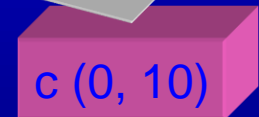
Events
and
Transactions



Transactions
Scheduled



Transaction
Expires
Event Created



Transaction
Expires
No Event

Events and Transactions: Example

ARCHITECTURE demo OF
example IS

SIGNAL a, b, c: BIT := '0';

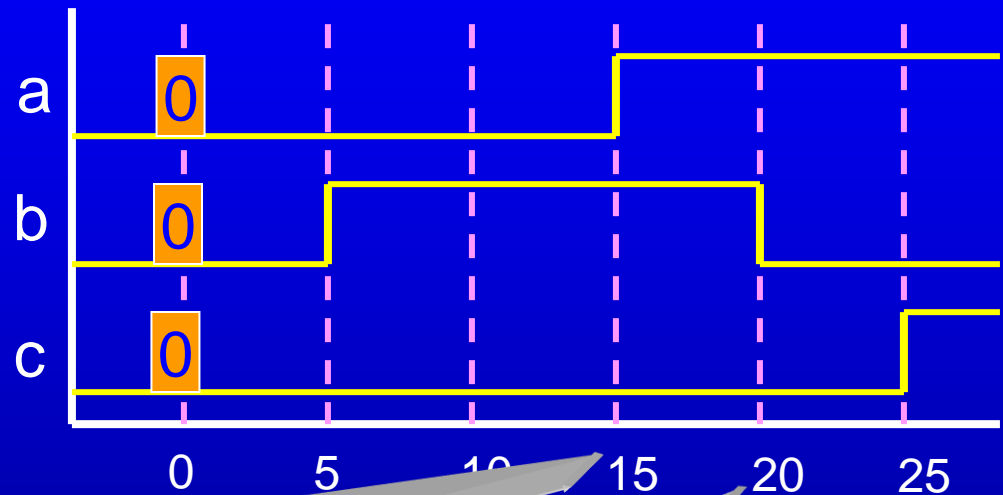
BEGIN

a <= '1' AFTER 15 NS;

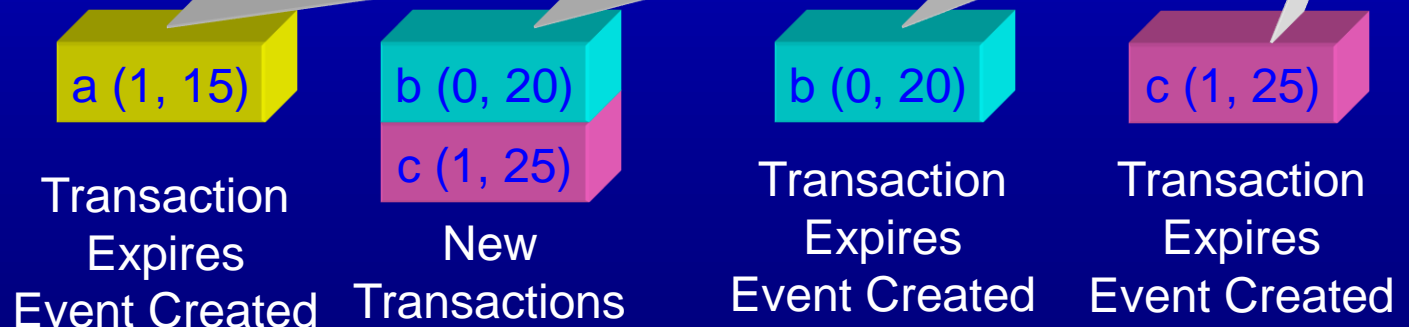
b <= NOT a AFTER 5 NS;

c <= a AFTER 10 NS;

END demo;

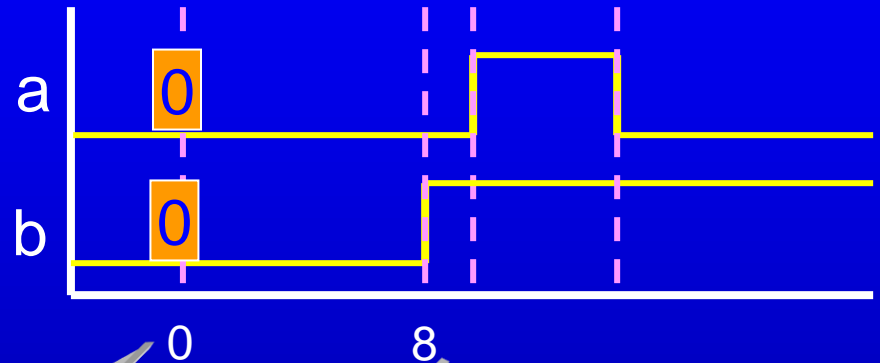


Events
and
Transactions

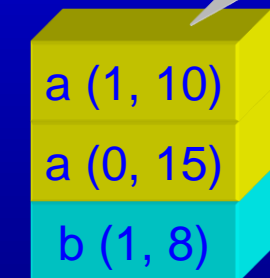


Inertial Delay: Suppressing a pulse

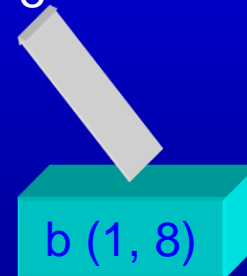
```
SIGNAL a, b: BIT := '0';  
...  
a <= '1' AFTER 10 NS,  
    '0' AFTER 15 NS; -- transport  
b <= NOT a AFTER 8 NS; -- inertial
```



Events
and
Transactions



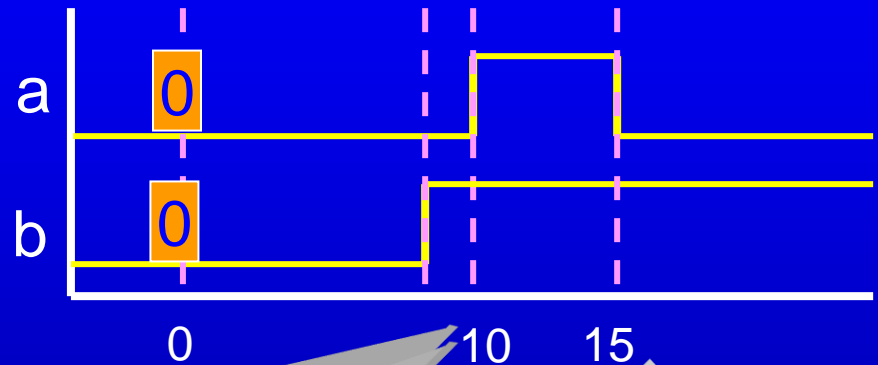
Transactions
Scheduled



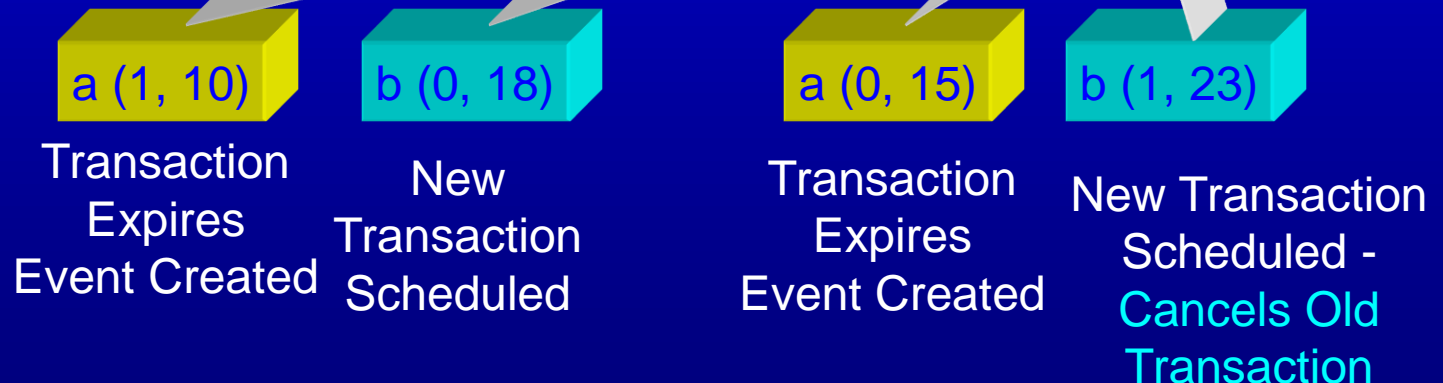
Transaction
Expires
Event Created

Inertial Delay: Suppressing a pulse

```
SIGNAL a, b: BIT := '0';  
...  
a <= '1' AFTER 10 NS,  
    '0' AFTER 15 NS; -- transport  
b <= NOT a AFTER 8 NS; -- inertial
```

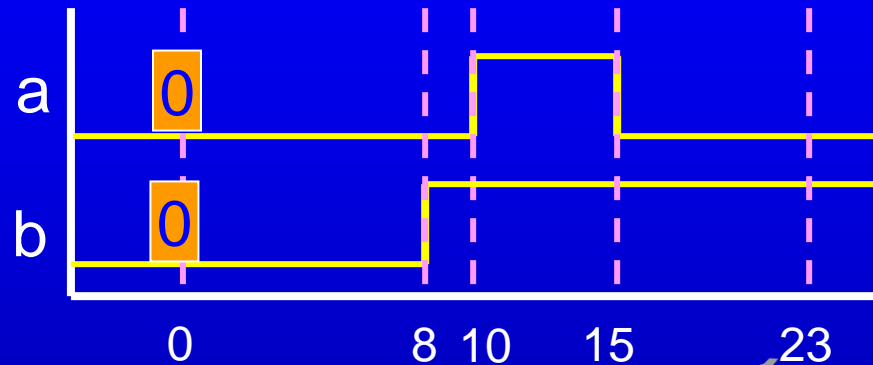


Events
and
Transactions



Inertial Delay: Suppressing a pulse

```
SIGNAL a, b: BIT := '0';  
...  
a <= '1' AFTER 10 NS,  
    '0' AFTER 15 NS; -- transport  
b <= NOT a AFTER 8 NS; -- inertial
```



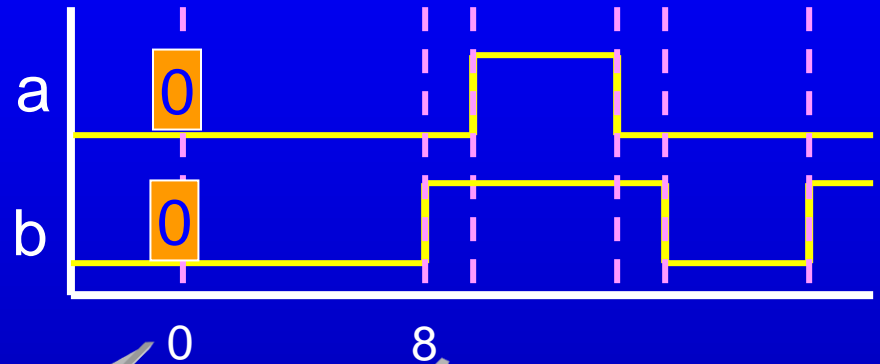
Events
and
Transactions

b (1,23)

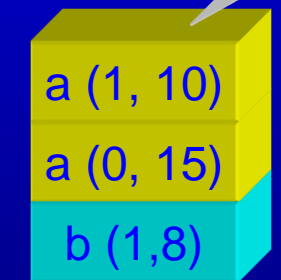
Transaction
Expires
No Event

Transport Delay: Propagating a pulse

```
SIGNAL a, b: BIT := '0';  
...  
a <= '1' AFTER 10 NS,  
    '0' AFTER 15 NS;  
b <= TRANSPORT NOT a  
    AFTER 8 NS;
```



Events
and
Transactions



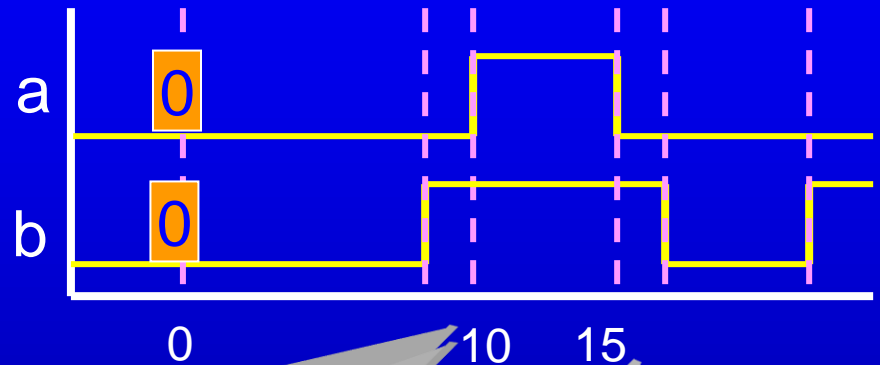
Transactions
Scheduled



Transaction
Expires
Event Created

Transport Delay: Propagating a pulse

```
SIGNAL a, b: BIT := '0';  
...  
a <= '1' AFTER 10 NS,  
    '0' AFTER 15 NS;  
b <= NOT a AFTER 8 NS;
```

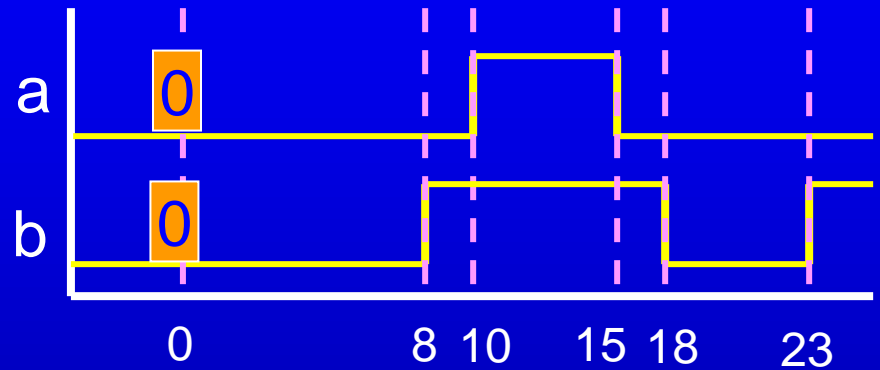


Events
and
Transactions

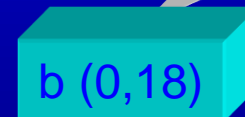


Transport Delay: Propagating a pulse

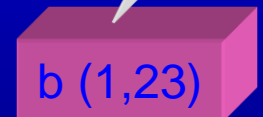
```
SIGNAL a, b: BIT := '0';  
...  
a <= '1' AFTER 10 NS,  
    '0' AFTER 15 NS; -- transport  
b <= TRANSPORT  
    NOT a AFTER 8 NS;
```



Events
and
Transactions



Transaction
Expires
Event Created



Transaction
Expires
Event Created

Generating clocks



```
a <= NOT a AFTER 10 ns;
```

Delta Delays

```
ARCHITECTURE x of y IS  
  SIGNAL b, c: bit;  
BEGIN  
  b <= NOT a;  
  c <= clock NAND b;  
  d <= c AND b;  
END x;
```

zero-delay
signal assignments

a <= 0
(clock = 1)

Delta 1

b <= 1

Delta 2

c <= 0
d <= 1

Delta 3

d <= 0

Delta 4



Simulation time does not advance

Contents

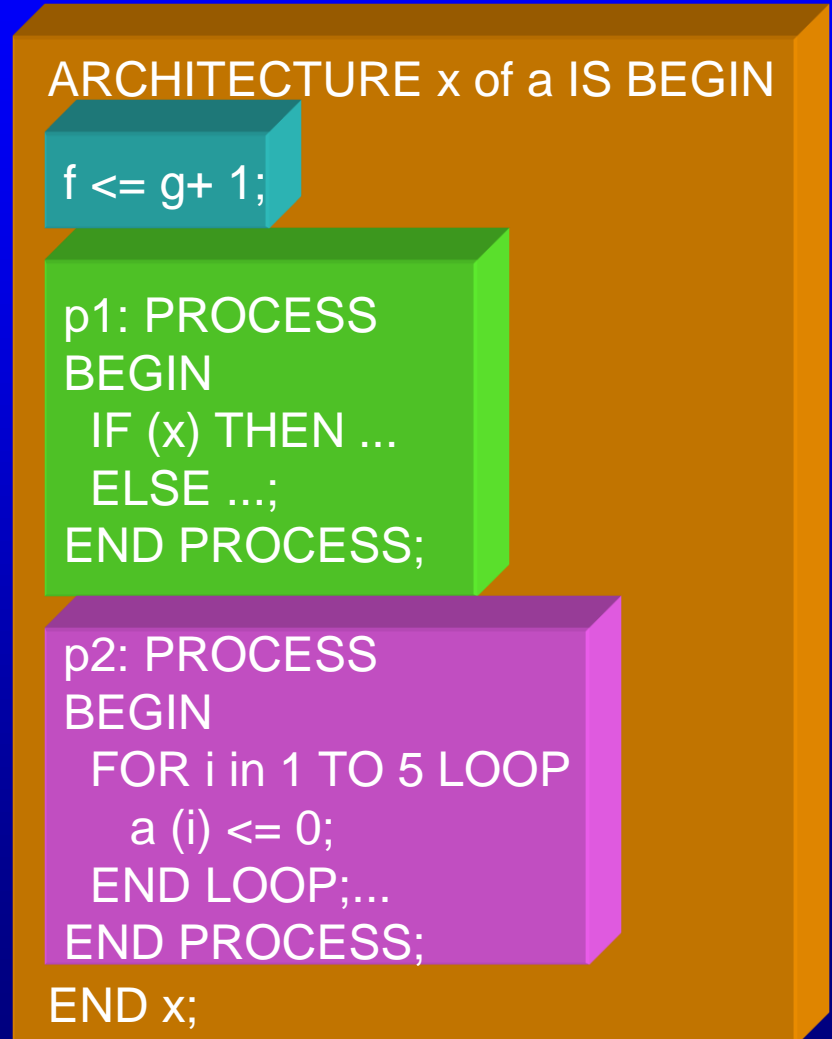
- Introduction
- Signal assignment
- Modelling delays
- Describing behaviour
- Structure, test benches, libraries, parameterisation
- Standards

Describing Behaviour: Processes

- Signal assignment statements OK for simple behaviour
- Complex behaviour requires more constructs
 - conditionals (IF, CASE)
 - loops (FOR, WHILE)
- Use VHDL PROCESS

VHDL PROCESS

- Execution within a PROCESS is sequential
- Processes are concurrent w.r.t each other
- Signal assignment is a simple special case
- Architecture consists of a set of Processes (and signal assignments) at top level
- Processes communicate using signals

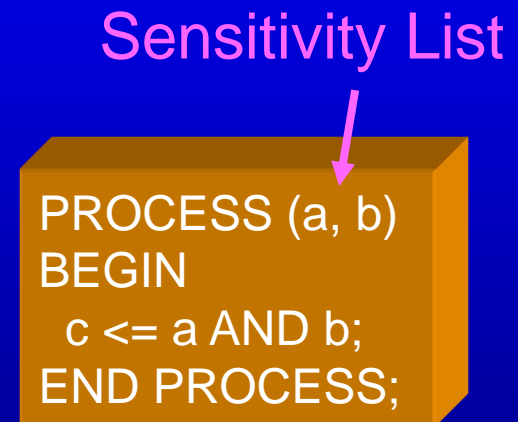


PROCESS Execution Semantics

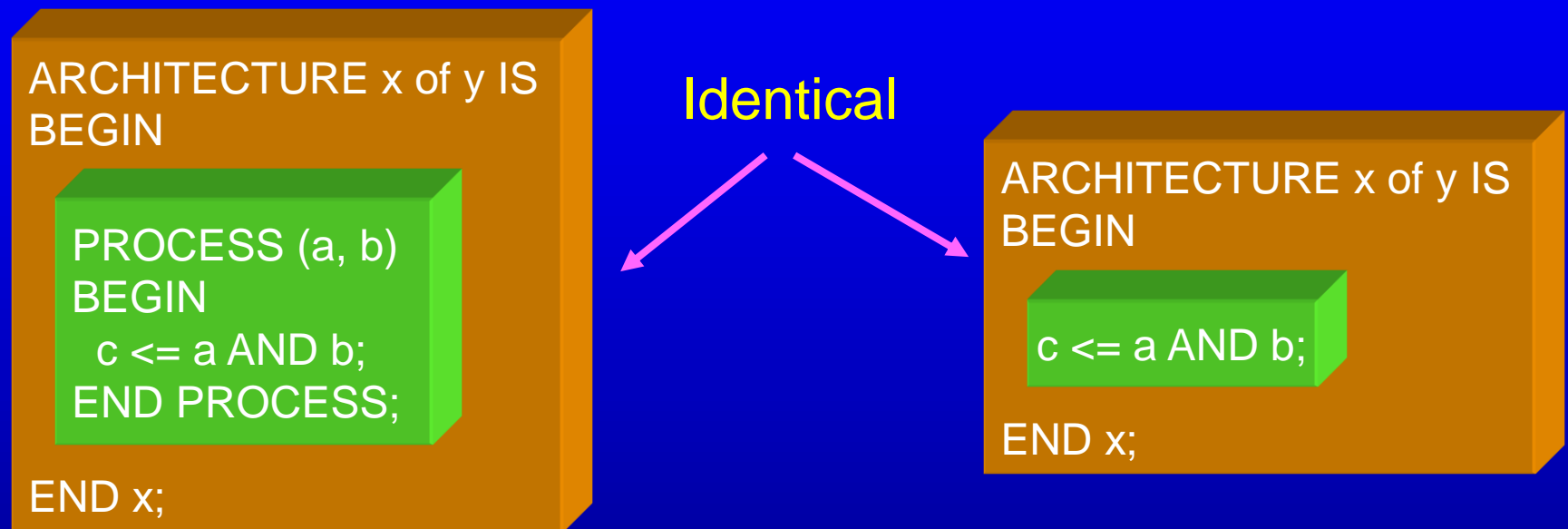
- Need to define when Process is executed
 - suspending/resuming execution
 - more complex than signal assignment (“evaluate when any signal on RHS changes”)
- No notion of “completion” of execution
 - needs to emulate hardware

Process Sensitivity List

- Process is sensitive to signals on Sensitivity List
- All processes executed once at time=0
- Suspended at end of process
- Reactivated when event occurs on any signal in sensitivity list



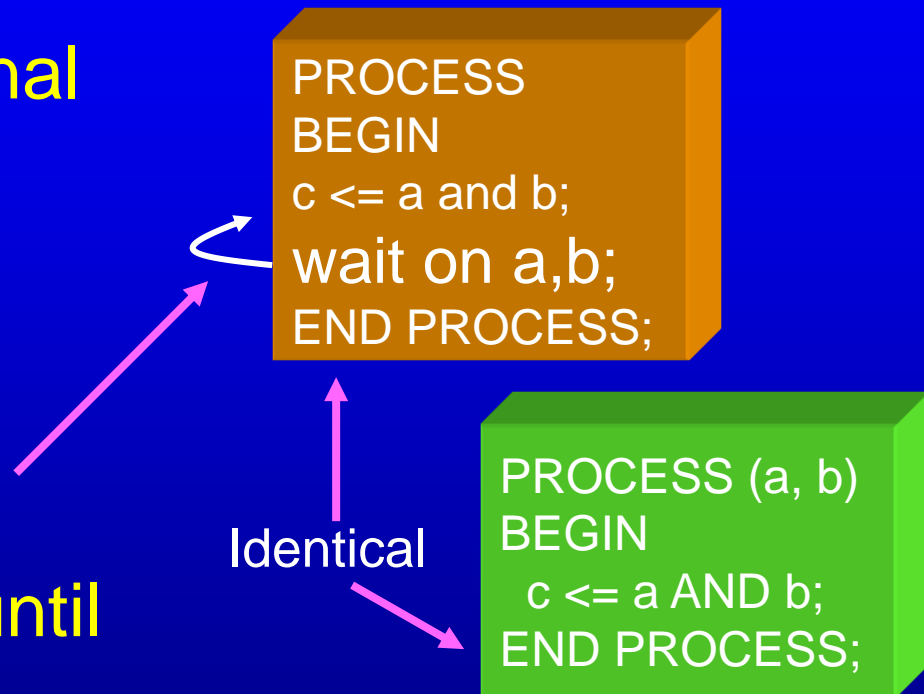
Process and Signal Assignment



Need not use PROCESS for modelling simple combinational behaviour

Process Synchronisation

- Sensitivity list is optional
- **wait** is general synchronisation mechanism
- Implicit infinite loop in process
- Execution continues until suspended by **wait** statement



Synchronisation with WAITs

- Synchronisation with wait more flexible
- Both sensitivity list and wait not allowed in same process
 - process can have any number of waits
- For combinational logic, place ALL input signals in sensitivity list
- For sequential logic, use waits appropriately

WAIT Examples

```
PROCESS  
BEGIN  
  wait for 10 ns;  
  outp <= inp;  
END PROCESS
```

Sample input every 10 ns

```
PROCESS  
BEGIN  
  wait until clk'event and clk='1';  
  q <= d;  
END PROCESS
```

Edge triggered flip flop

```
PROCESS (clk, reset)  
BEGIN  
  IF reset THEN  
    q <= '0';  
  ELSIF clk'event and clk='1'  
    q <= d;  
  END IF;  
END PROCESS
```

Flip flop with Reset

```
PROCESS  
BEGIN  
  outp <= inp;  
END PROCESS
```

Error! (no waits)
(Compare signal
assignment at
architecture level)

Process Variables

- Variables used for local computations
 - within processes
- Not associated with events/transactions
 - unlike signals
- Assignment of value is immediate
 - unlike signals

```
PROCESS
VARIABLE result : BIT;
BEGIN
wait until clk'event and clk='1';
result := '0';
for i in 0 to 6 loop
    result := result XOR inp (i);
end loop;
outp <= result;
END PROCESS;
```

Signal Assignments in Processes

```
PROCESS (x)
BEGIN
  A <= '1';
  A <= '0'; -- overrides
END PROCESS;
```

Multiple
assignments OK.
Sequential
Execution.

```
PROCESS
BEGIN
  -- A='0' here
  A <= '1';
  B <= A; -- B='0'
  wait on x;
END PROCESS;
```

Signal
assignments
take effect
when process
suspends
(next delta)

```
PROCESS
VARIABLE p: BIT;
BEGIN
  -- A='0' here
  A <= '1';
  p := A; -- p='0'
  B <= p; -- B='0'
  wait on x;
END PROCESS;
```

Variable assignments
take effect immediately
(same delta)

Signal Assignments in Processes

```
PROCESS
VARIABLE p: BIT;
BEGIN
  A <= '0';
  if (c) then A <= '1';
  else begin
    p := B;
    A <= p;
  end
  wait on x;
  B <= A;
  A <= B;
  wait on x;
END PROCESS;
```

Signal Assignment
to A equivalent to:
(1) introduction of
temporary variable
tA when A occurs
on LHS and (2)
assignment of tA to
A when process
suspends

```
PROCESS
VARIABLE p, tA, tB: BIT;
BEGIN
  tA := '0';
  if (c) then tA := '1';
  else begin
    p := B;
    tA := p;
  end
  A <= tA;
  wait on x;
  tB := A;
  tA := B;
  A <= tA;
  B <= tB;
  wait on x;
END PROCESS;
```

Equivalent

Contents

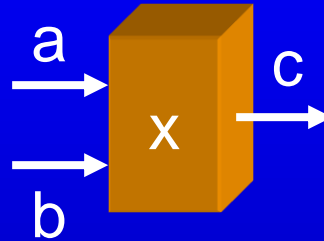
- Introduction
- Signal assignment
- Modelling delays
- Describing behaviour
- Structure, test benches, libraries, parameterisation
- Standards

Structural Description

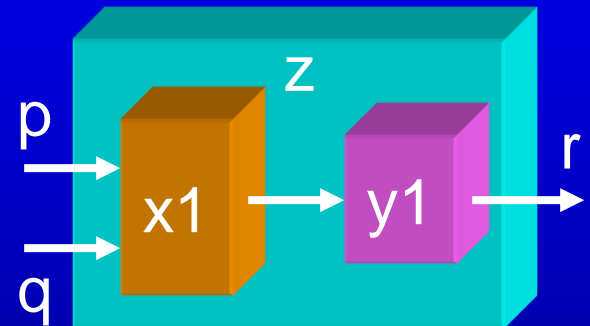
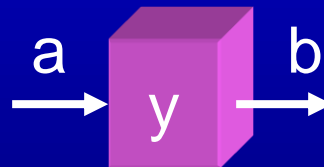
- Instantiation
- Interconnection

Hierarchy

```
ENTITY x IS  
  PORT (a, b: IN BIT,  
        c: OUT BIT);  
END x;  
ARCHITECTURE xa OF x IS  
BEGIN  
  c <= a AND b;  
END xa;
```



```
ENTITY y IS  
  PORT (a : IN BIT,  
        b: OUT BIT);  
END y;  
ARCHITECTURE ya OF y IS  
BEGIN  
  b <= NOT a;  
END ya;
```



z contains
instances of
x and y

Instantiation and Interconnection - 1

```
ENTITY z IS
  PORT (p, q: IN BIT,
        r: OUT BIT);
END z;
ARCHITECTURE structural OF z IS
  COMPONENT xc
```

```
    PORT (a, b: IN BIT; c: OUT BIT);
  END COMPONENT;
  COMPONENT yc
```

```
    PORT (a: IN BIT; c: OUT BIT);
  END COMPONENT;
```

```
  FOR ALL: xc USE WORK.x (xa);
  FOR ALL: yc USE WORK.y (ya);
```

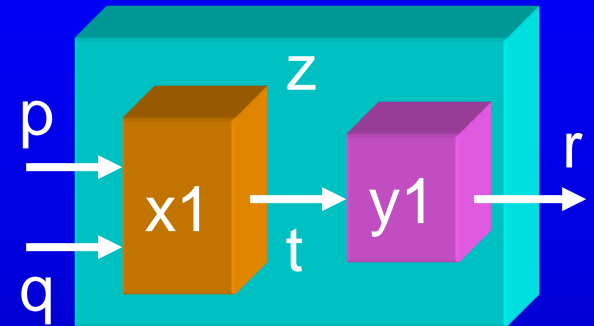
```
  SIGNAL t: BIT;
```

```
BEGIN
```

```
  x1: xc PORT MAP (p, q, t);
```

```
  y1: yc PORT MAP (t, r);
```

```
END structural;
```



Component declaration

Configuration specification
(which architecture?)

Temporary signal

Instantiation

(a=>p,b=>q,c=>t)

Instantiation and Interconnection - 2

Instance name

Component name

```
x1: xc PORT MAP (p, q, t);  
y1: yc PORT MAP (t, r);
```

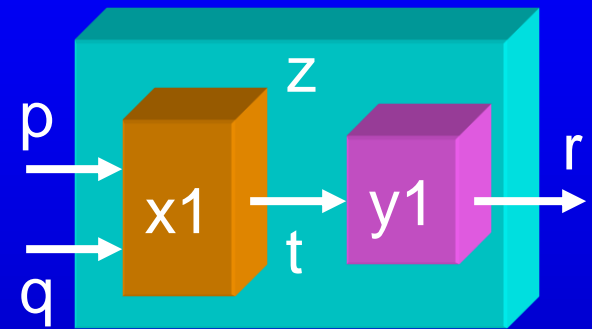
Same name
implies connection

Port association list:
order of names
determines connectivity:

a - p

b - q

c - t



Port Mapping

```
COMPONENT xc  
  PORT (a, b: IN BIT; c: OUT BIT);  
END COMPONENT;
```

Mapping by position: preferred for short port lists

```
x1: xc PORT MAP (p, q, t);
```

Mapping by name: preferred for long port lists

```
x1: xc PORT MAP (b => q, a => p, c => t);
```

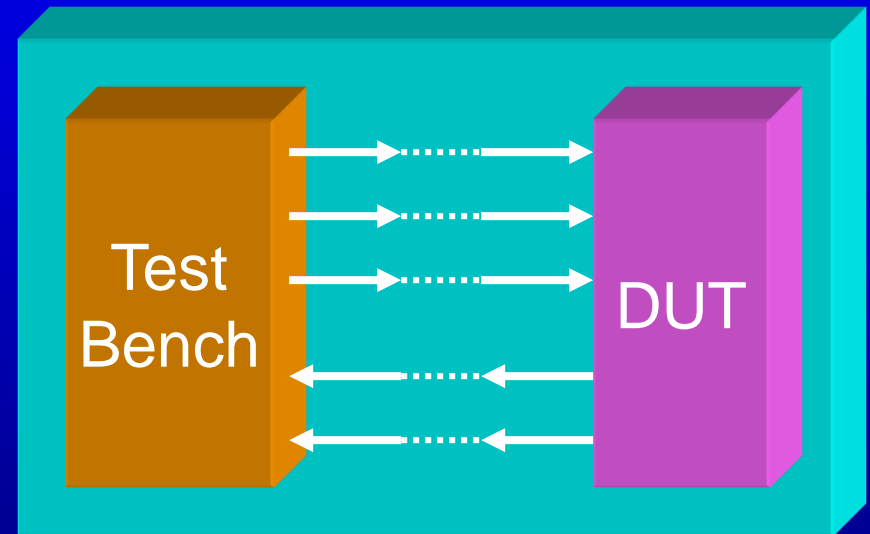
In both cases, complete port mapping should be specified

Test Benches

- Purpose - test correctness of Design Under Test (DUT)
 - provide input stimulus
 - observe outputs
 - compare against expected outputs
- Test Bench is also a VHDL model

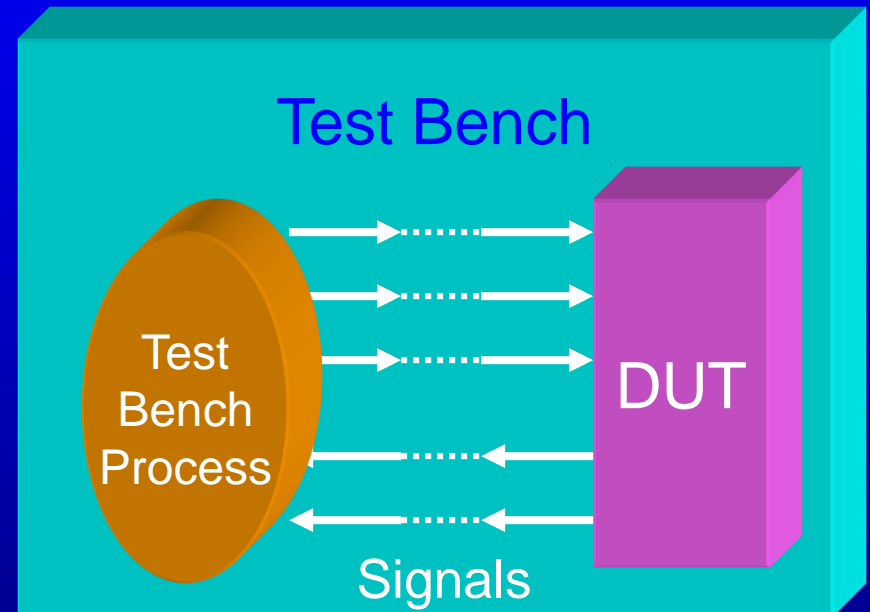
Test Bench Modelling - 1

- Test bench a separate VHDL entity
- Ports are connected to DUT's ports
 - i/p port corresponding to DUT's o/p port
 - o/p port corresponding to DUT's i/p port



Test Bench Modelling - 2

- Test bench instantiates the DUT
- Stimulus generation and output monitoring in separate VHDL process
- Signals are connected to DUT's ports



Libraries and Packages

- **PACKAGE** - collection of
 - components
 - data types
 - functions/procedures
- **LIBRARY** - collection of **PACKAGES**

Packages

```
PACKAGE util IS
  COMPONENT c IS
    PORT (a: IN BIT, b: OUT BIT);
  END COMPONENT
  TYPE my_int IS INTEGER RANGE -7 TO 7;
  FUNCTION comp (a: BIT_VECTOR)
    RETURN BIT_VECTOR;
END util;
```

Package declaration

```
PACKAGE BODY util IS
  FUNCTION comp (a: BIT_VECTOR)
    RETURN BIT_VECTOR IS
  BEGIN
    RETURN NOT a;
  END comp;
END util;
```

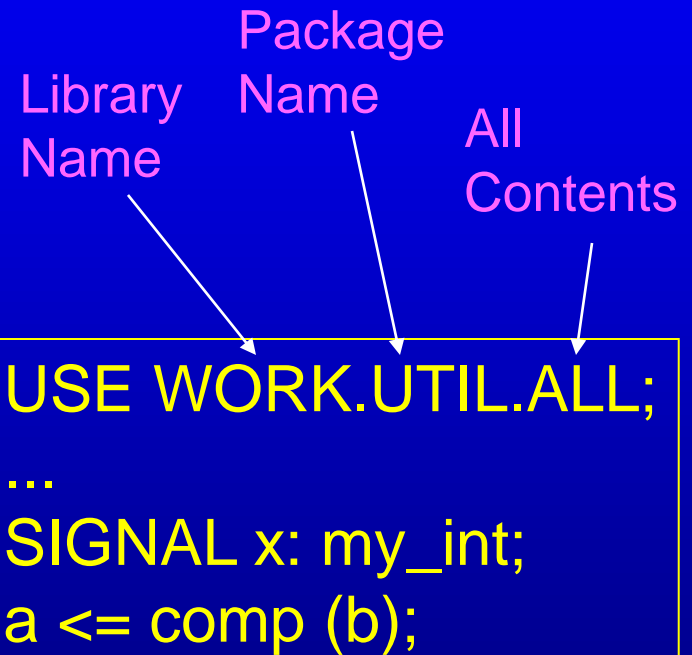
Package body

Using a Package

```
PACKAGE util IS
  COMPONENT c IS
    PORT (a: IN BIT, b: OUT BIT);
  END COMPONENT
  TYPE my_int IS INTEGER RANGE -7 TO 7;
  FUNCTION comp (a: BIT_VECTOR)
    RETURN BIT_VECTOR;
END util;
...
```

Library Name Package Name All Contents

```
USE WORK.UTIL.ALL;
...
SIGNAL x: my_int;
a <= comp (b);
```



Libraries

- **STD**
 - STANDARD
 - types/utilities (BIT, TIME, INTEGER,...)
 - TEXTIO
 - interface to text files
- **WORK**
 - default library for storing user designs
- **STD_LOGIC_1164**
 - multi-valued logic

TEXTIO Package

- Data types and functions for
 - reading from text files
 - writing out text files

```
FILE f: TEXT IS "file_name";  
VARIABLE one_line: line;  
VARIABLE str: STRING;  
...  
READLINE (f, one_line); -- read one line from file  
READ (str, one_line); -- read a word from line  
WRITELINE (g, one_line); -- write one line to file  
WRITE (str, one_line); -- write a word into line
```

Design Parameterisation: GENERICs

```
ENTITY e IS
  GENERIC (delay: TIME := 2 NS; width: INTEGER := 4);
  PORT (a: IN BIT_VECTOR (0 TO width);
        b: OUT BIT_VECTOR (0 TO width));
END e;

ARCHITECTURE a OF e IS
BEGIN
  b <= NOT a AFTER delay;
END a;
```

Default
Value

Generic
Parameters

Passing GENERIC Parameters

```
ENTITY c IS  
  GENERIC (delay: TIME := 4 ns); PORT (a: IN BIT; b: OUT BIT);  
END c;
```

```
ARCHITECTURE a OF e IS  
  COMPONENT c  
    GENERIC (t: TIME:= 4 NS);  
    PORT (a: IN BIT, b: OUT BIT);  
  END COMPONENT;  
  SIGNAL x, y: BIT;  
  FOR ALL: c USE work.c (arc);  
  BEGIN  
    c1: c GENERIC MAP (3 ns)  
      PORT MAP (x, y);  
  END a;
```

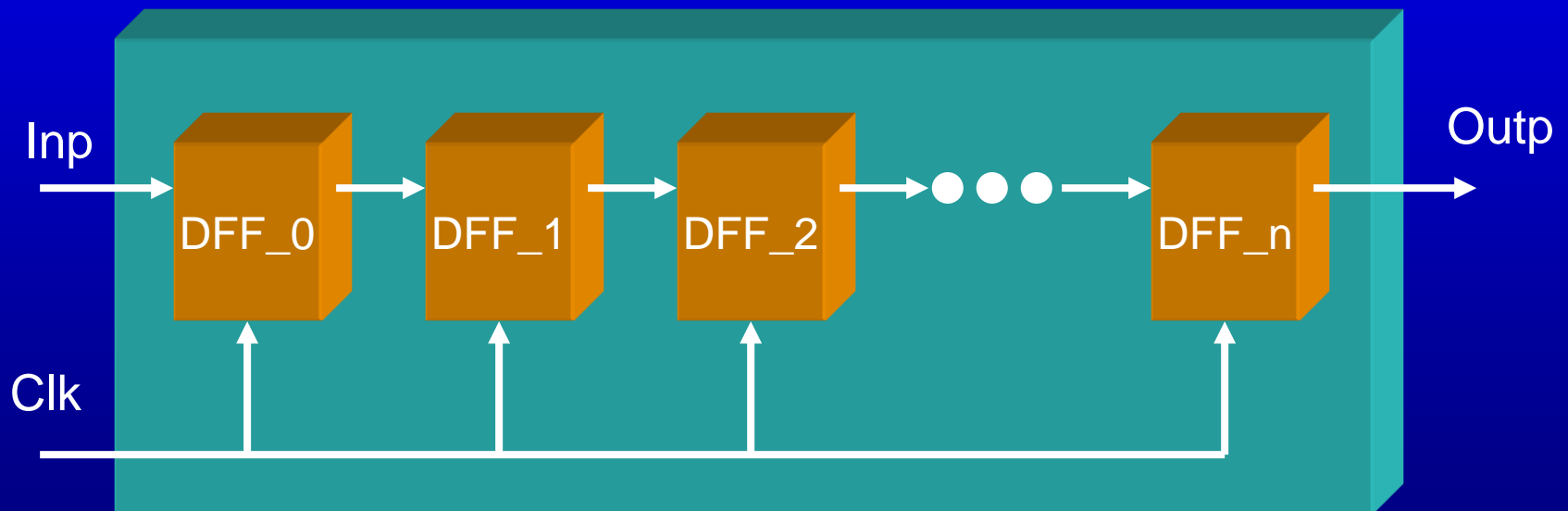
```
ARCHITECTURE def OF e IS  
  COMPONENT c  
    GENERIC (t: TIME:= 4 NS);  
    PORT (a: IN BIT, b: OUT BIT);  
  END COMPONENT;  
  SIGNAL x, y: BIT;  
  FOR ALL: c USE work.c (arc);  
  BEGIN  
    c1: c PORT MAP (x, y);  
  END def;
```

Delay parameter = 3 ns

Default Delay = 4 ns

Conditional and Looped Instantiation

Number of instances of DFF determined by Generic Parameter n



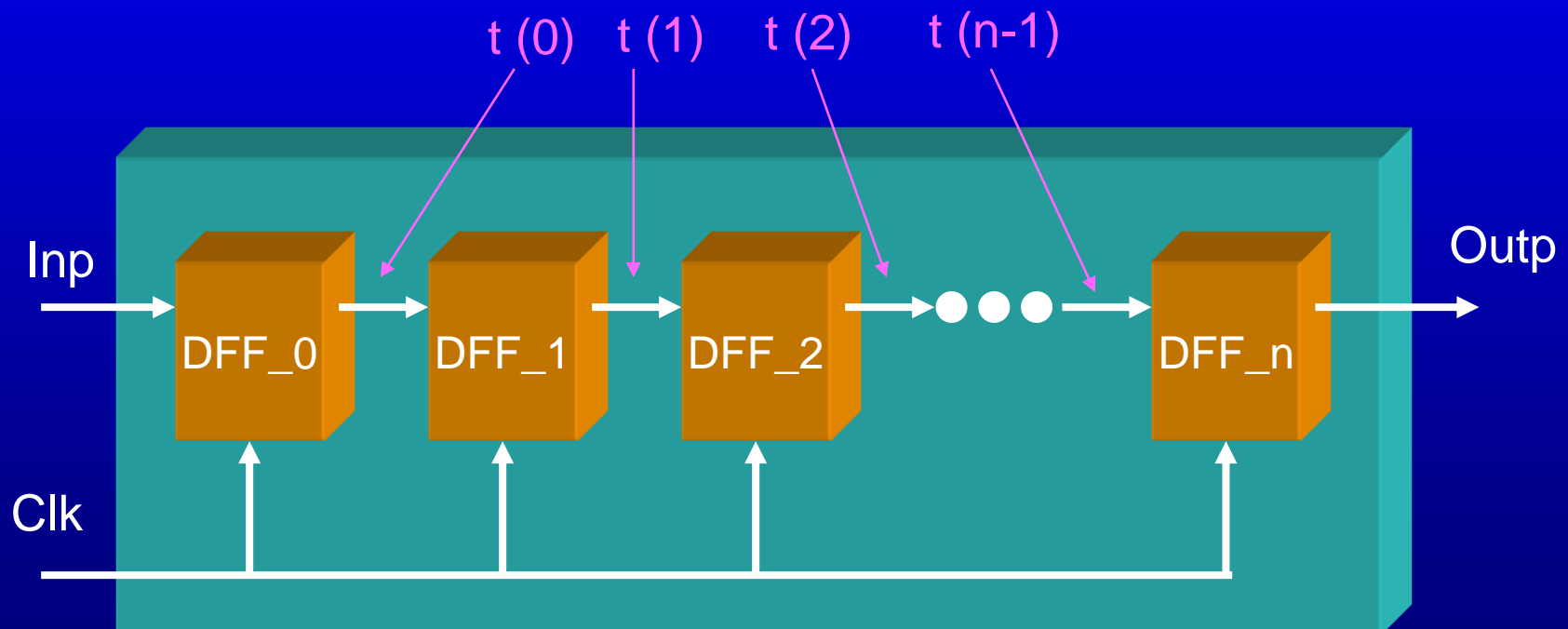
Conditional and Looped Instantiation: GENERATE

GENERIC (n: INTEGER)...

...

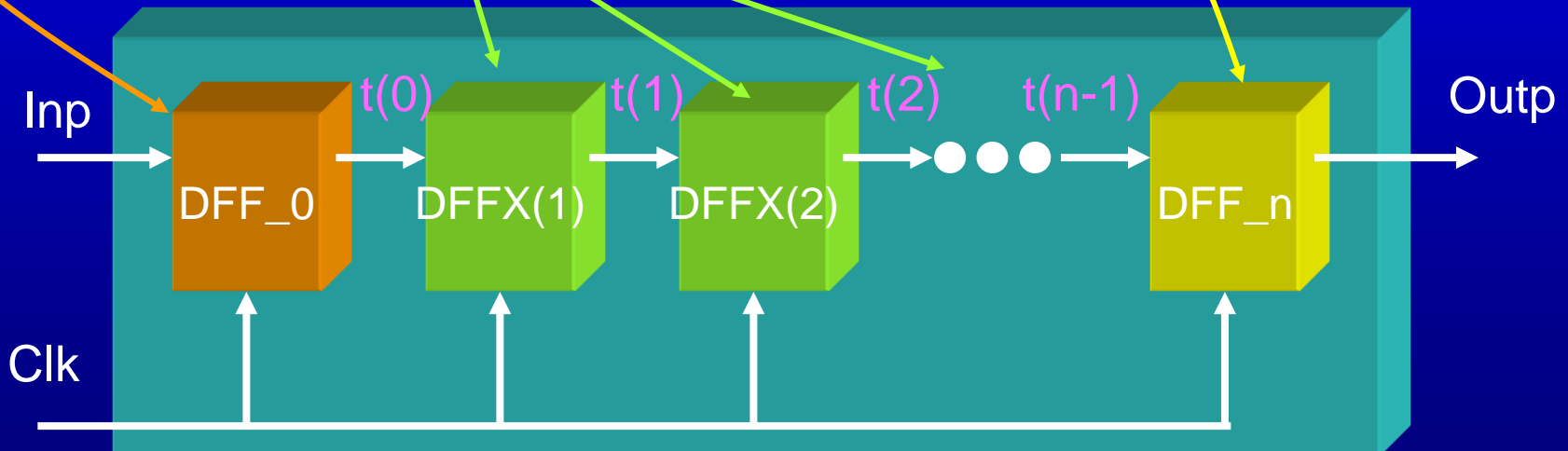
SIGNAL t: BIT_VECTOR (0 TO n-1);

Need intermediate
signal t (0 to n-1)



GENERATE Statement

```
SIGNAL t: BIT_VECTOR (0 TO n-1);  
...  
dff_0: DFF PORT MAP (Inp, Clk, t (0));  
dff_n: DFF PORT MAP (t (n-1), Clk, Outp);  
FOR i IN 1 TO n-1 GENERATE  
  dffx: DFF PORT MAP ( t (i-1), Clk, t (i) );  
END GENERATE;
```



Contents

- Introduction
- Signal assignment
- Modelling delays
- Describing behaviour
- Structure, test benches, libraries, parameterisation
- Standards

VHDL Standards

- Std_LOGIC 1164 Package
 - IEEE Standard
 - Supported by all VHDL simulation/synthesis tools
- VITAL
 - Modelling timing in VHDL

9-valued Logic Type: std_ulogic

- **Modelling CMOS**
 - Current strengths
 - Tristating
- **Modelling Don't Care**
- **Simulation Values**
 - Unknown
 - Uninitialised

```
TYPE std_ulogic IS (  
    'U', -- uninitialised  
    'X', -- unknown  
    '0', -- Forcing 0  
    '1', -- Forcing 1  
    'Z', -- High impedance  
    'W', -- Weak Unknown  
    'L', -- Weak 0  
    'H', -- Weak 1  
    '-', -- Don't care  
);
```

Signal Drivers

```
ARCHITECTURE x...  
SIGNAL a: BIT;  
begin  
  PROCESS  
    begin  
      ...  
      a <= '1';  
      ...a <= '0';...  
    end;  
  PROCESS  
    begin  
      ...  
      a <= '0';  
      ...  
    end;  
  a <= '0';  
end x;
```

Driver for a

Driver for a

Driver for a

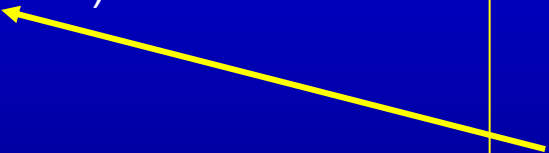
Multiple drivers
not allowed for
same signal!
- leads to conflicts

Resolution Functions

- Multiple drivers allowed only when signal is declared to be **RESOLVED**
 - using RESOLUTION FUNCTION

```
FUNCTION res (values: BIT_VECTOR) RETURN BIT IS  
VARIABLE accum : BIT := '1';  
BEGIN  
  FOR i IN values'RANGE LOOP  
    accum := accum AND values(i);  
  END LOOP;  
  RETURN accum;  
END;
```

Multiple driving
values treated
as vector



Modelling
Wired AND



Resolving std_u logic Signals

- Models the effect of shorting two wires in CMOS

```

TYPE stdlogic_table is array(std_u logic, std_u logic)
    of std_u logic;
CONSTANT resolution_table : stdlogic_table := (
-- U    X    0    1    Z    W    L    H    -
('U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
('U', 'X', '0', 'X', '0', '0', '0', '0', '0' ), -- | 0 |
('U', 'X', 'X', '1', '1', '1', '1', '1', '1' ), -- | 1 |
('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'Z' ), -- | Z |
('U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'W' ), -- | W |
('U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'L' ), -- | L |
('U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'H' ), -- | H |
('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-' )  -- | - | );
    
```

Resolution Function for std_ulogic

```
FUNCTION resolved ( s : std_ulogic_vector )  
  RETURN std_ulogic IS  
  VARIABLE result : std_ulogic := '-';-- weakest state default  
  BEGIN  
    IF (s'LENGTH = 1) THEN RETURN s(s'LOW);  
    ELSE -- Iterate through all inputs  
      FOR i IN s'RANGE LOOP  
        result := resolution_table(result, s(i));  
      END LOOP; -- Return the resultant value  
      RETURN result;  
    END IF;  
  END resolved;
```

Resolved Type: std_logic

- Multiple std_ulogic types resolved into std_logic type

```
SUBTYPE std_logic IS resolved std_ulogic;
```

```
...
```

```
SIGNAL x: std_logic;
```

```
...
```

```
x <= 'Z';
```

```
x <= '1';
```

```
-- No conflict. Value of x resolves to '1'
```

unresolved
type

resolution
function

resolved
type

Overloading

- Standard operators can be overloaded for std_ulogic type

```
FUNCTION "and" (l, r: std_ulogic) RETURN UX01 IS  
BEGIN  
    RETURN (and_table (l, r)); -- 2-d constant array  
END "and";
```

Utilities

- Type conversions
 - to_Bit
 - to_BitVector
 - to_StdUlogic
 - to_StdLogicVector
- Detecting Edges
 - rising_edge
 - falling_edge

Modelling Timing Checks

- Modelling a Flip Flop
 - Propagation delays
 - Setup times
 - Hold times
 - Minimum pulse width
- Many different implementations possible