CS223: Hardware Lab

Tutorial 3
HDL

A. Sahu

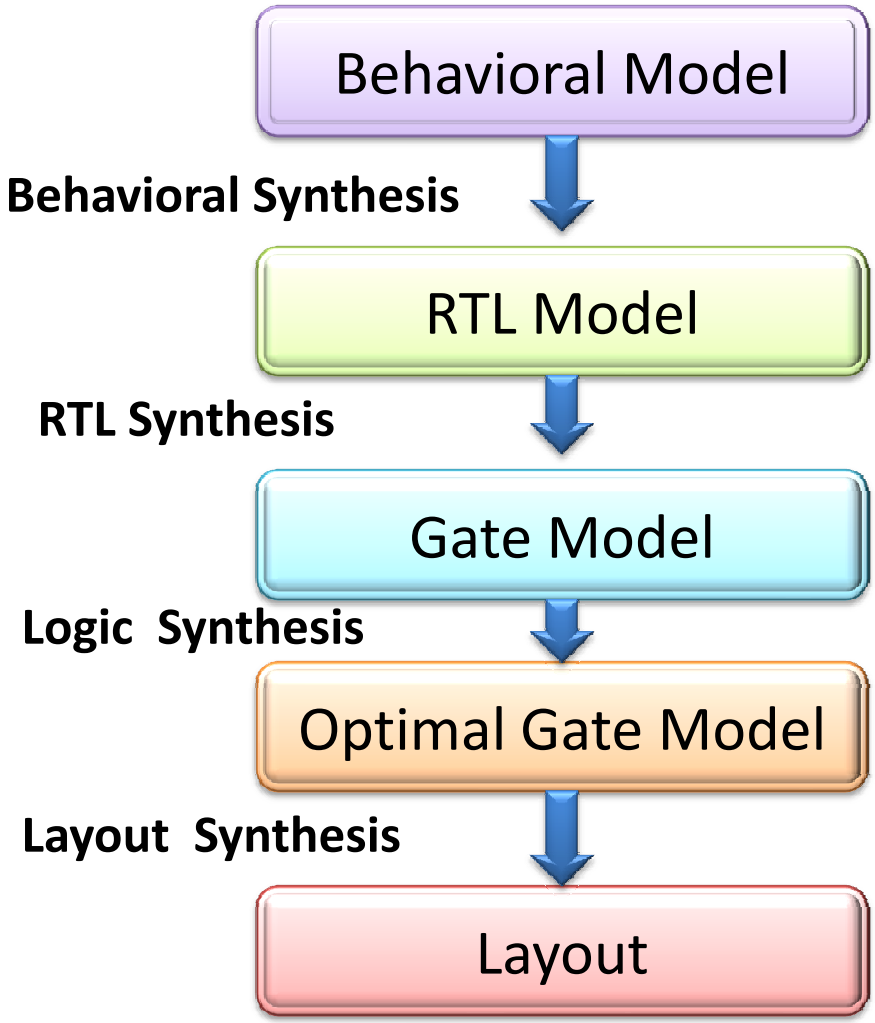
Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

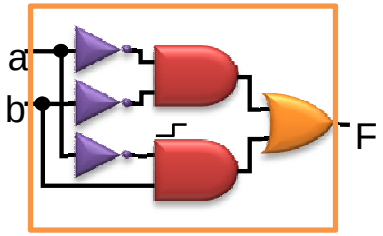
- **VHDL basic language concepts**
- basic design methodology
- Examples

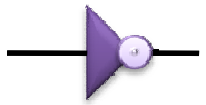
Design Flow

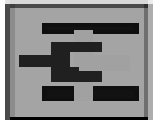



```
for (i=0;i <4;i++)
S = S+ A[i]
```

```
Cycle 1:  T1 = A[0] + A[1]
          T2 = A[2] + A[3]
Cycle 2:  S = T1 + T2
```

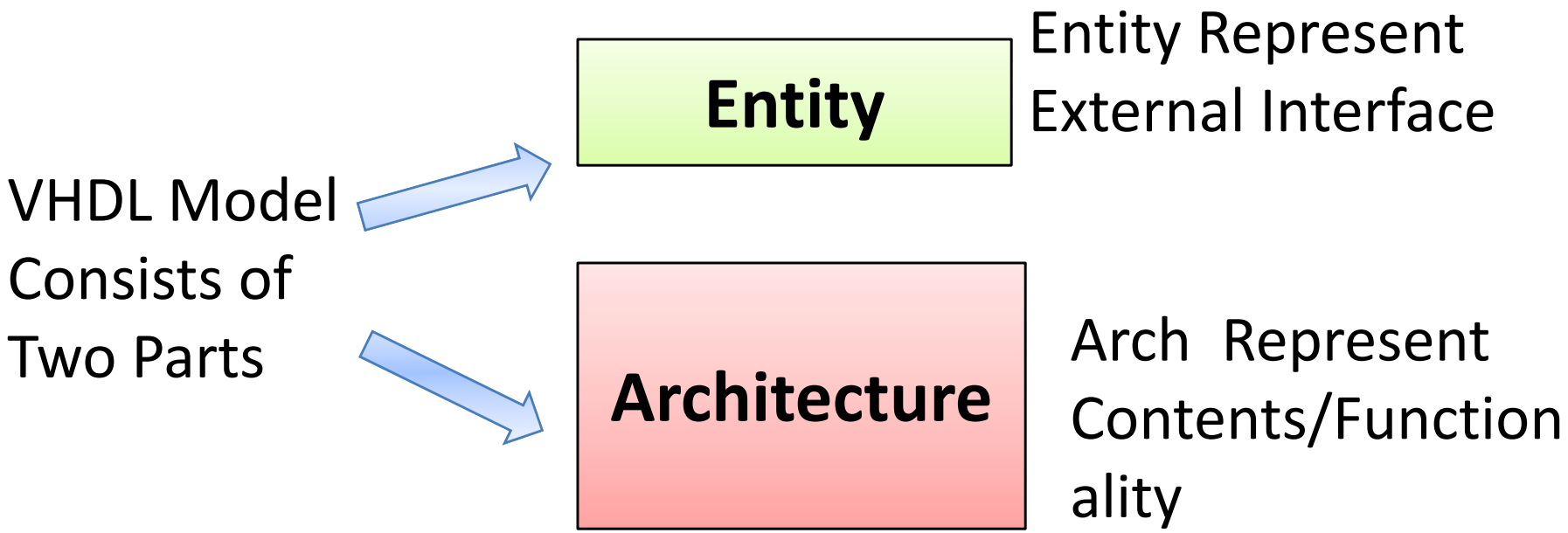






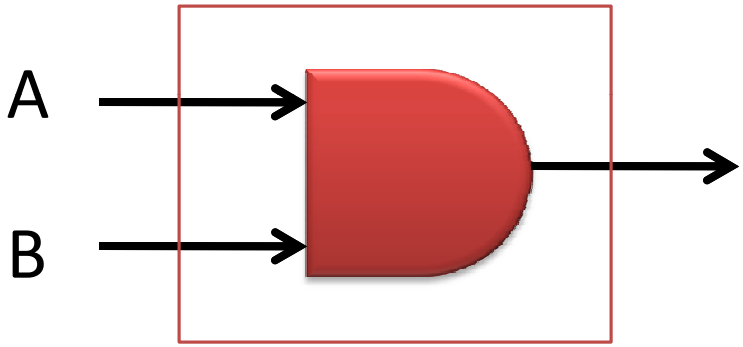
Fundamental VHDL Objects

- Entity and Architecture Pair



VHDL: Entity

- Entity : Represent External Interface




```
ENTITY    and_gate    IS
          PORT (  A:   IN   BIT;
                  B:   IN BIT;
                  Y:   OUT BIT
          );
END and_gate;
```

Model Name

Port Type

Port Direction

Port Name

Entity has Interface: No functionality

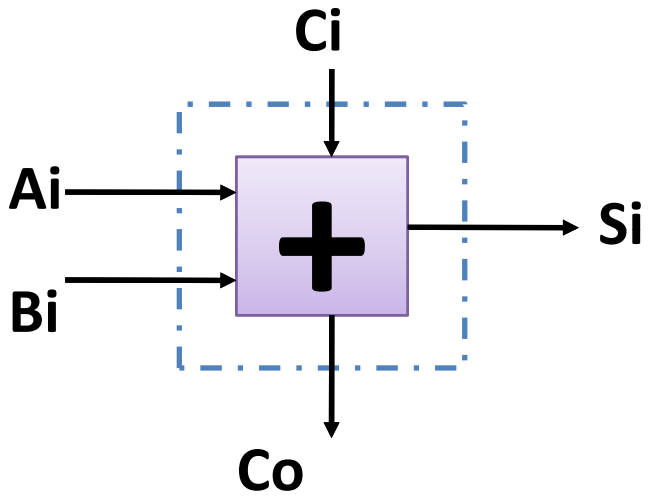
VHDL: Architecture, Specifying functionality

```
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

Specifying Concurrency

- Concurrent Signal Assignments




```
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;
```

Order of Execution

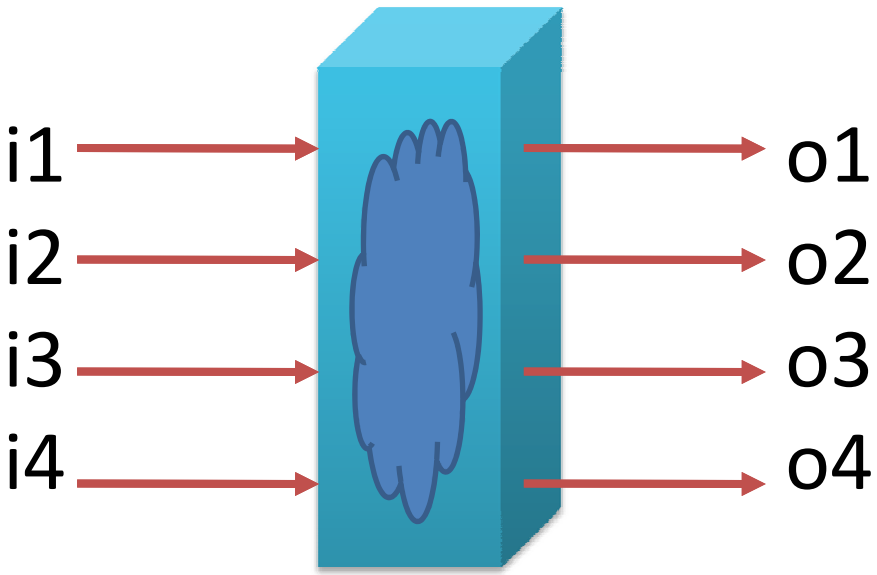
- Execution independent of Specification

```
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;
```

Modeling Combinational Logic

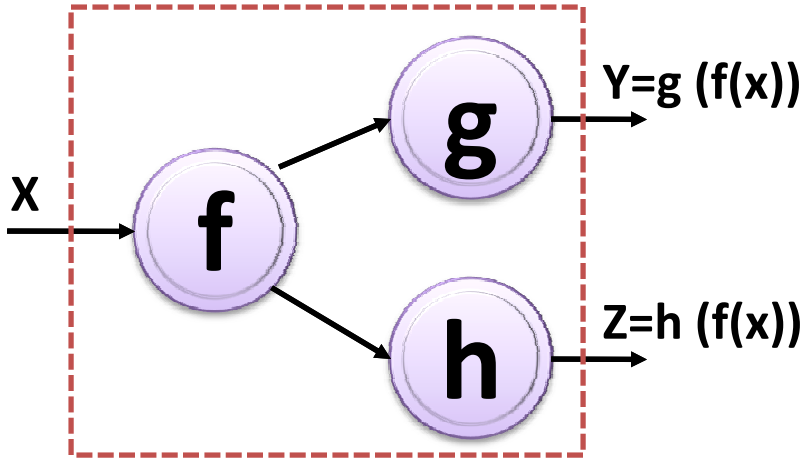
- One concurrent assignment for each output



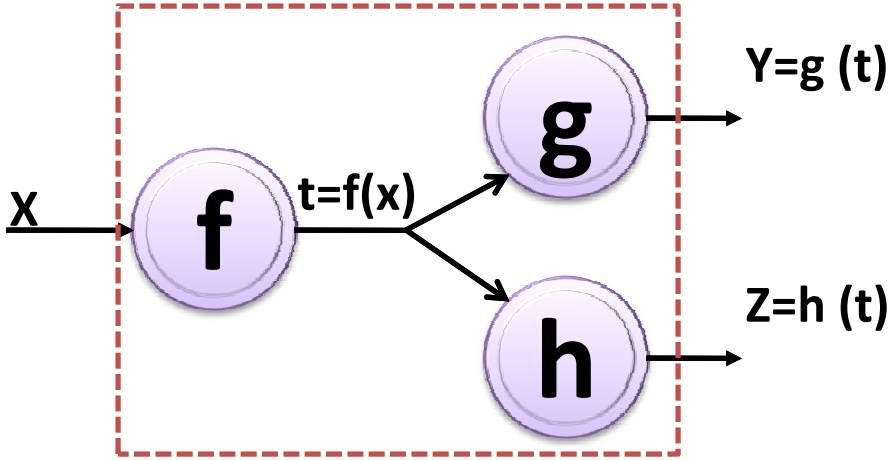

```
ARCHITECTURE data_flow
OF comb_logic IS
BEGIN
        o1 <= i1 and i2;
        o2 <= (i2 or i3) xor (i1 and i4);
        o3 <= ...;
        o4 <= ...;
END data_flow;
```

When Logic Complexity Increase

- Temporary SIGNALS needed
- Avoid redundant evaluations



Ports :
X,Y,Z

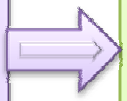


Signal : t

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

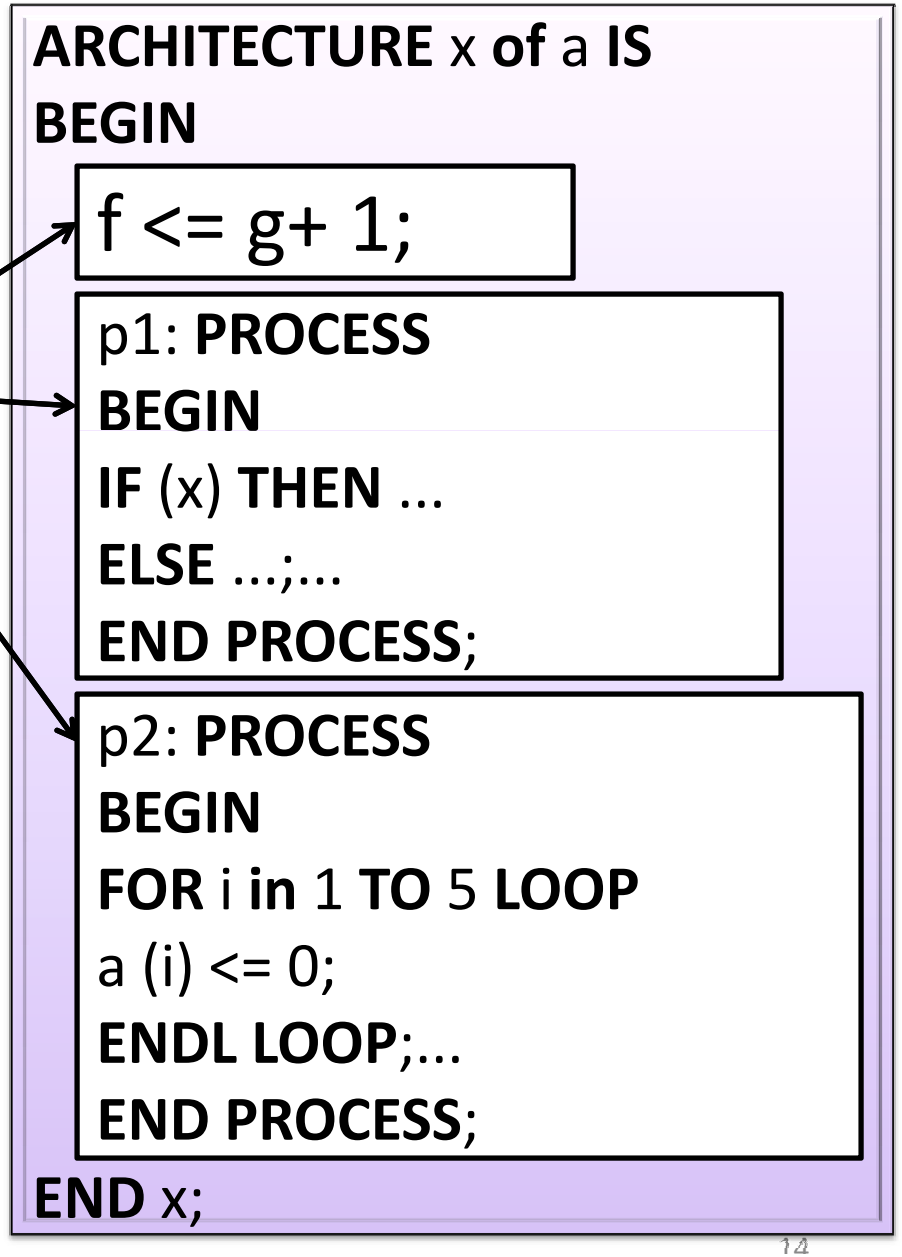
```
ARCHITECTURE data_flow1
OF comb_logic IS
SIGNAL temp: BIT;
BEGIN
  temp <= (i1 and i2 and i3);
  o1 <= temp xor i2;
  o2 <= temp or i4;
END data_flow;
```

Describing Behavior: Processes

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

VHDL PROCESS

- 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

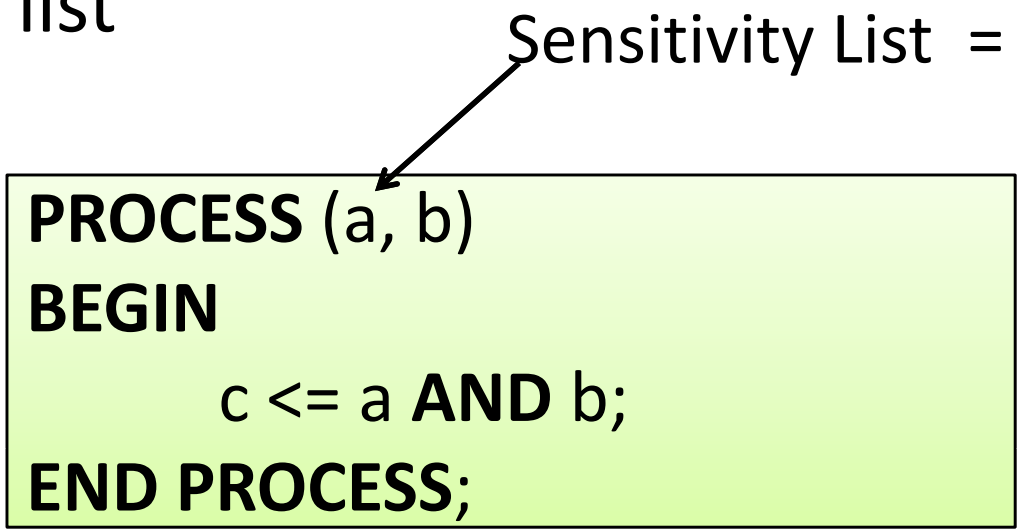


PROCESS Execution Semantics

- 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

```
ARCHITECTURE x of y IS
BEGIN
    PROCESS (a, b)
    BEGIN
        c <= a AND b;
    END PROCESS;

END x;
```

```
ARCHITECTURE x of y IS
BEGIN

        c <= a AND b;

END x;
```

Identical

Need not use PROCESS for modeling simple combinational behavior

Process Synchronization

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

```
PROCESS
BEGIN
wait on a,b;
c <= a and b;
END PROCESS;
```

```
PROCESS (a,b)
BEGIN
c <= a and b;
END PROCESS;
```

Identical

Synchronization with WAITs

- Synchronization 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 10ns

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

Edge Triggered D FlipFlop

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

FlipFlop 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

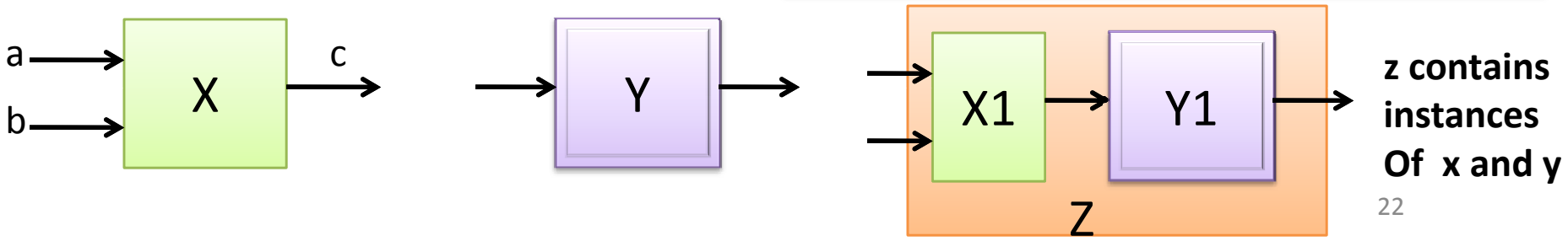
```
PROCESS
VARIABLE result : BIT;
BEGIN
wait until clk'event and clk='1';
 result := '0';
for i in 0 to 6 loop
    result := result XOR inp (i);
end loop;
outp <= result;
END PROCESS;
```

Structural Description

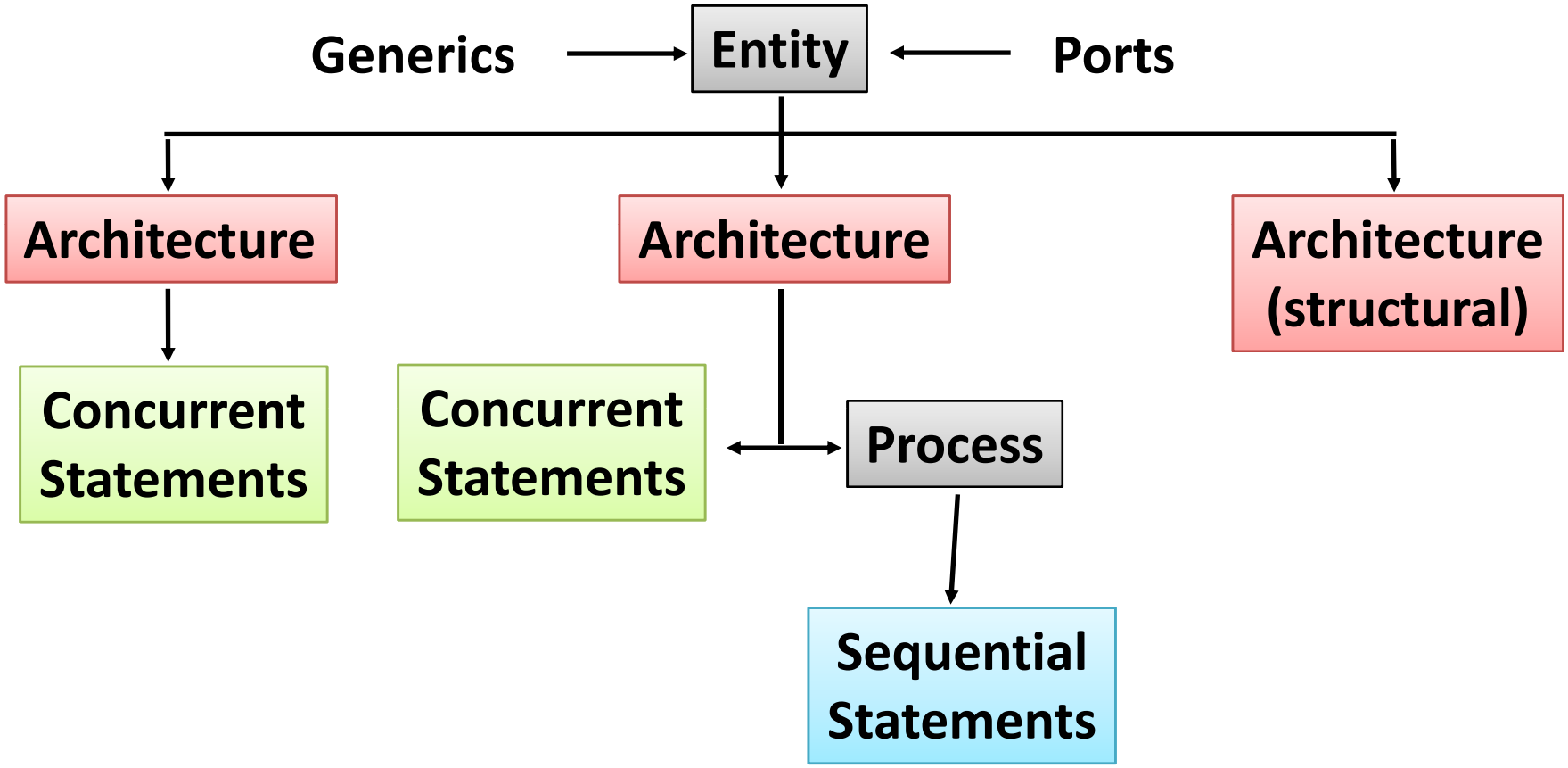
- Instantiation and Interconnection
- **Hierarchy**

```
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 xa;
```



VHDL Models


Generics
Entity
Ports
Architecture
Architecture
Architecture (structural)
Concurrent Statements
Concurrent Statements
Process
Sequential Statements

VHDL Objects : Constants

- Improve the readability of the code
- Allow for easy updating

```
CONSTANT <constant_name>  :  <type_name> := <value>;

CONSTANT PI : REAL := 3.14;
CONSTANT WIDTH : INTEGER := 8;
```

VHDL Objects : Signals

- Signals are used for communication between components
- Signals can be seen as real, physical wires

```
SIGNAL <signal_name> : <type_name> [:= <value>];

SIGNAL enable : BIT;
SIGNAL output : bit_vector(3 downto 0);
SIGNAL output : bit_vector(3 downto 0) := "0111";
```

VHDL Objects : Variables

- Variables are used only in processes and subprograms (functions and procedures)
- Variables are generally not available to multiple components and processes
- All variable assignments take place immediately

```
VARIABLE <variable_name> : <type_name> [:= <value>];

VARIABLE opcode : BIT_VECTOR (3 DOWNTO 0) := "0000";
VARIABLE freq : INTEGER;
```

Signals versus Variables

- A key difference between variables and signals is the assignment delay

```
ARCHITECTURE signals OF test IS
    SIGNAL a, b, c, out_1, out_2 : BIT;
BEGIN
    PROCESS (a, b, c)
    BEGIN
      out_1 <= a NAND b;
      out_2 <= out_1 XOR c;
    END PROCESS;
END signals;
```

d= propagation delay

Time	a b c	out_1	out_2
0	0 1 1	1	0
1	1 1 1	1	0
1+d	1 1 1	0	0

Signals versus Variables

```
ARCHITECTURE variables OF test IS
    SIGNAL a, b, c: BIT;
    VARIABLE out_3, out_4 : BIT;
BEGIN
    PROCESS (a, b, c)
    BEGIN
      out_3 := a NAND b;
      out_4 := out_3 XOR c;
    END PROCESS;
END variables;
```

Time	a b c	out_3	out_4
0	0 1 1	1	0
1	1 1 1	0	1

Attributes

- Language defined attributes return information about certain items in VHDL
  - Types, subtypes
  - Procedures, functions
  - Signals, variables, constants
  - Entities, architectures, configurations, packages
  - Components
- VHDL has several predefined attributes that are useful to the designer
- Attributes can be user-defined to handle custom situations (user-defined records, etc.)

Signal Attributes

- General form of attribute use is

<name> ' <attribute_identifier>

- Some examples of signal attributes

```
X'EVENT -- evaluates TRUE when an event on signal X has just
        -- occured.

X'LAST_VALUE -- returns the last value of signal X

X'STABLE(t) -- evaluates TRUE when no event has occured on
            -- signal X in the past t" time
```

Value Attributes

- 'LEFT -- returns the leftmost value of a type
- 'RIGHT -- returns the rightmost value of a type
- 'HIGH -- returns the greatest value of a type
- 'LOW -- returns the lowest value of a type
- 'LENGTH -- returns the number of elements in a constrained array
- 'RANGE -- returns the range of an array

Value Attributes : Examples

```
TYPE count is RANGE 0 TO 127;
TYPE states IS (idle, decision,read,write);
TYPE word IS ARRAY(15 DOWNTO 0) OF bit;
```

count'left = 0	states'left = idle	word'left = 15
count'right = 127	states'right = write	word'right = 0
count'high = 127	states'high = write	word'high = 15
count'low = 0	states'low = idle	word'low = 0
count'length = 128	states'length = 4	word'length = 16

count'range = 0 TO 127

word'range = 15 DOWNTO 0

Register Example

- This example shows how attributes can be used in the description of an 8-bit register.
- Specifications
  - Triggers on rising clock edge
  - Latches only on enable high
  - Has a data setup time of 5 ns.

```
ENTITY 8_bit_reg IS
        PORT (enable, clk : IN std_logic;
                    a : IN std_logic_vector (7 DOWNTO 0);
                    b : OUT std_logic_vector (7 DOWNTO 0);
        END 8_bit_reg;
```

Register Example Cntd..

- A signal of type std_logic may assume values:
  - 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', or '-‘
- Use of 'STABLE detects for setup violations
- What happens if clk was 'X'?

```
ARCHITECTURE first_attempt OF 8_bit_reg IS
        BEGIN
            PROCESS (clk)
            BEGIN
              IF (enable = '1') AND a'STABLE(5 ns) AND
                  (clk = '1') THEN  b <= a;
              END IF;
            END PROCESS;
        END first_attempt;
```

Register Example Cntd..

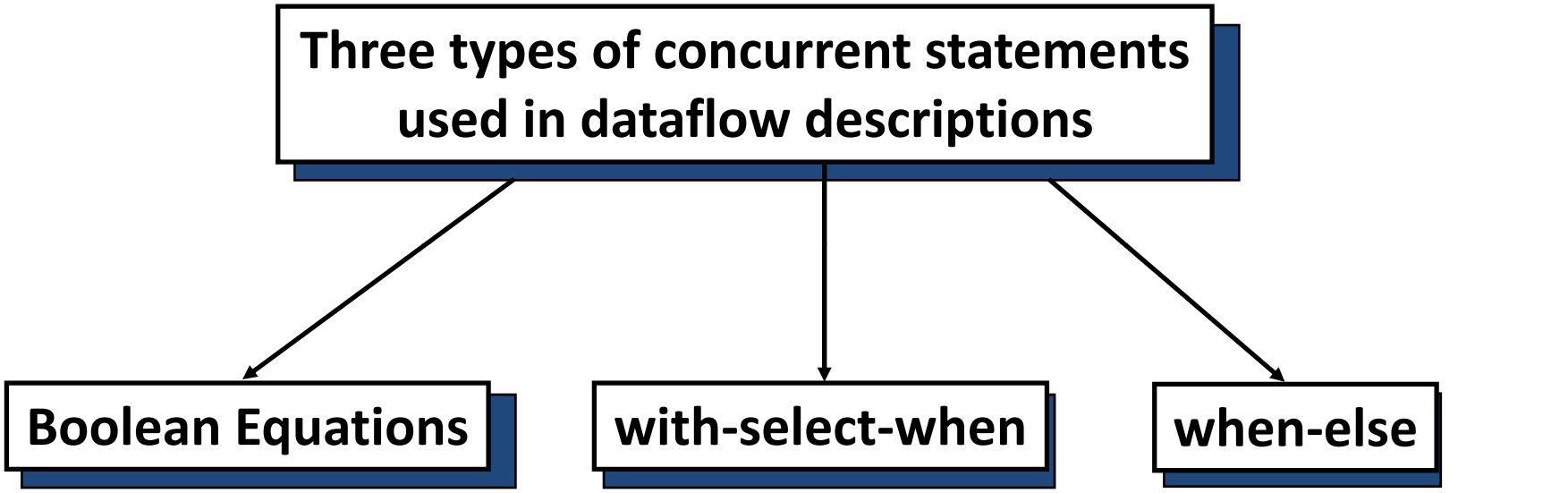
- The use of 'LAST_VALUE ensures the clock is rising from a 0 value

```
ARCHITECTURE behavior OF 8_bit_reg IS
BEGIN
        PROCESS (clk)
        BEGIN
                IF (enable ='1') AND a'STABLE(5 ns) AND
                        (clk = '1') AND (clk'LASTVALUE = '0') THEN
                          b <= a;
                END IF;
        END PROCESS;
END behavior;
```

Concurrent and Sequential Statements

- VHDL provides two different types of execution: ***sequential*** and ***concurrent***
- Different types of execution are useful for modeling of real hardware
- **Sequential statements** view hardware from a *programmer* approach
- **Concurrent statements** are order-independent and asynchronous

Concurrent Statements



For concurrent signal assignments

For selective signal assignments

For conditional signal assignments

Concurrent Statements: Boolean Equations

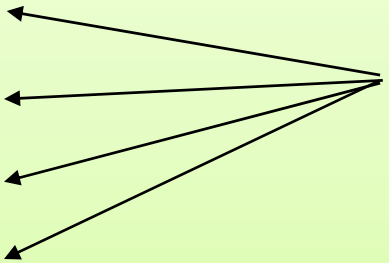
```
entity control is port(mem_op, io_op, read, write: in bit;
                       memr, memw, io_rd, io_wr:out bit);
end control;
```

```
architecture control_arch of control is
begin
        memw  <= mem_op and write;
        memr   <= mem_op and read;
        io_wr   <= io_op and write;
        io_rd    <= io_op and read;
end control_arch;
```

Concurrent Statements: with-select-when

```
entity mux is port(
        a,b,c,d: in std_logic_vector(3 downto 0);
        s: in std_logic_vector(1 downto 0);
        x: out std_logic_vector(3 downto 0));
end mux;
```

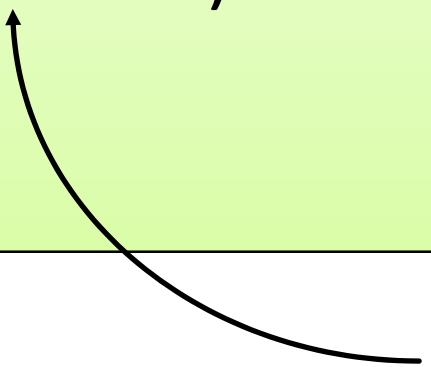
```
architecture mux_arch of mux is
begin
with s select
        x   <=  a when "00",
                b when "01",
                c when "10",
                d when others;
end mux_arch;
```



*possible values of **s***

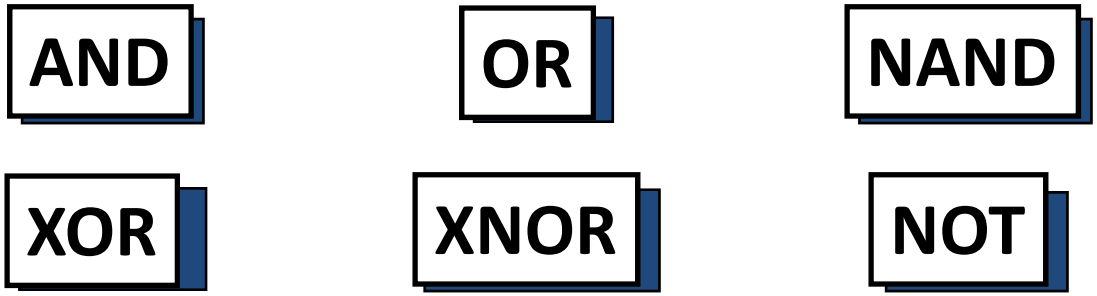
Concurrent Statements: When-else

```
architecture mux_arch of mux is
begin
        x   <= a when (s = "00") else
               b when (s = "01") else
               c when (s = "10") else
               d;
end mux_arch;
```



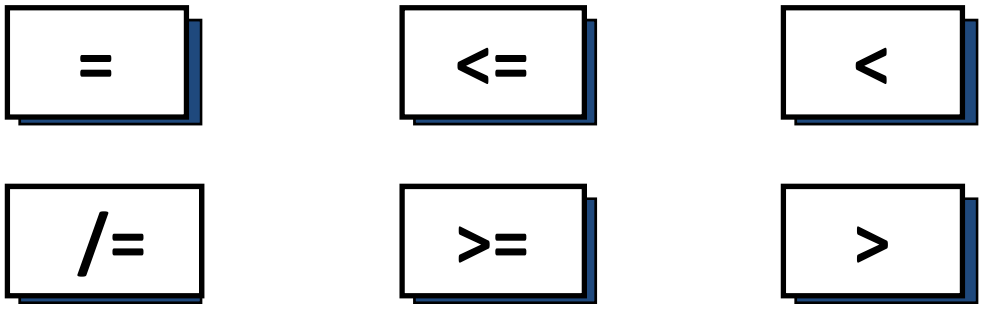
This may be any simple ***condition***

Logical Operators



- Predefined for the types:
  - bit and Boolean.
  - One dimensional arrays of bit and Boolean.
- Logical operators *don't have* an order of precedence X <= A **or** B **and** C
- Will result in a compile-time error.

Relational Operators



- Used for testing equality, inequality, and ordering.
- (= and /=) are defined for all types.
- (<, <=, >, and >=) are defined for scalar types
- The types of operands in a relational operation must match.

Arithmetic Operators

Addition operators

+ - &

Multiplication operators

* / rem mod

Miscellaneous operators

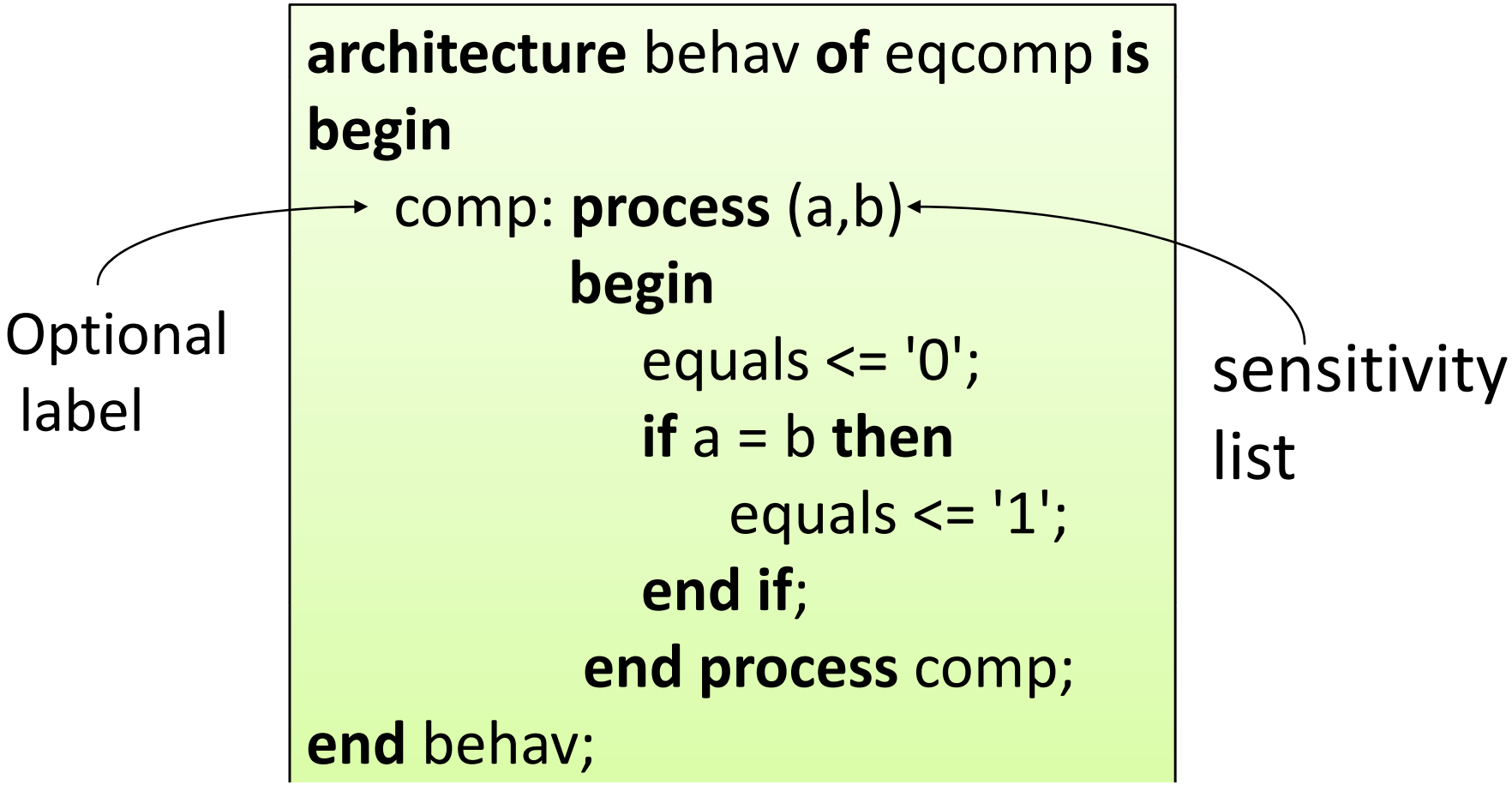
abs **

Sequential Statements

- Sequential statements are contained in a *process*, *function*, or *procedure*.
- Inside a process signal assignment is sequential from a simulation point of view.
- The order in which signal assignments are listed *does affect* the result.

Process Statement

- Process statement that embodies algorithms
- A process has a sensitivity list that identifies which signals will cause the process to execute.



Process Statement

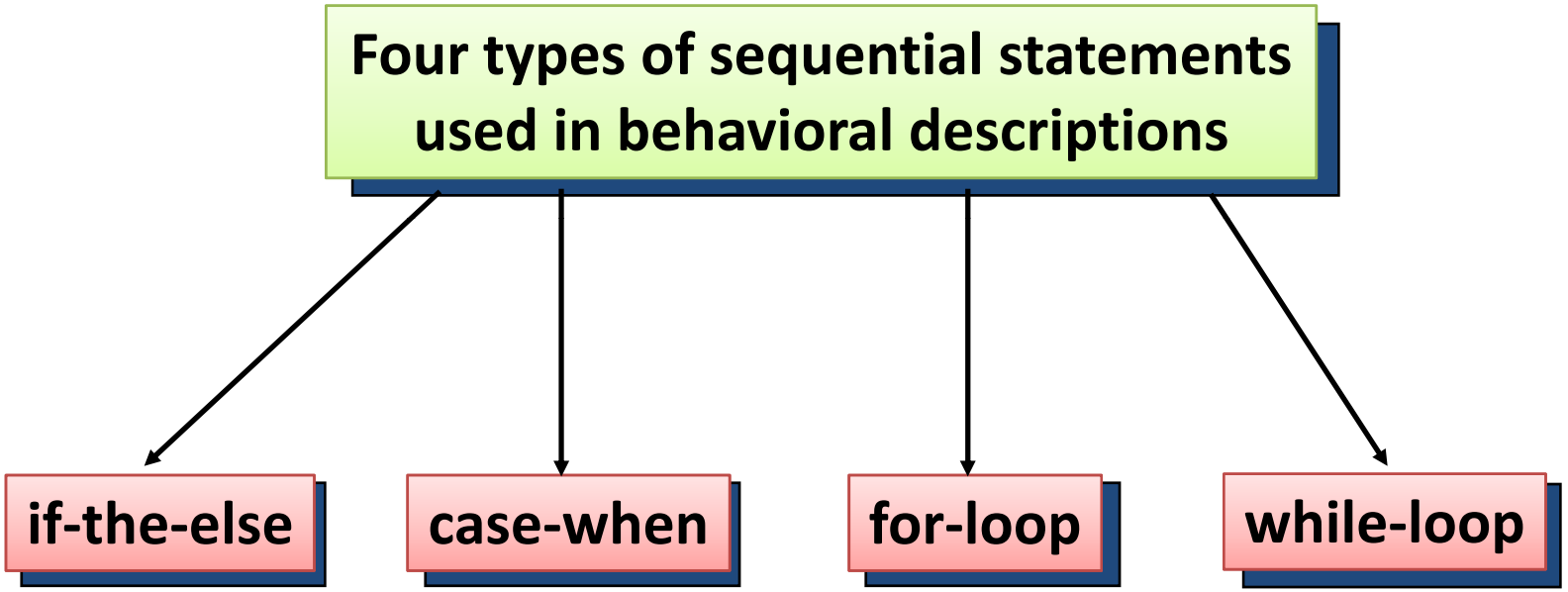
The use of wait statements

```
Proc1:process (a,b,c)
        begin
            x <= a and b and c;
        end process;
```

Equivalent

```
Proc2:process
        begin
            x <= a and b and c;
            wait on a, b, c;
        end process;
```

Sequential Statements



Sequential Statements

if-then-else

```
signal step: bit;
signal addr: bit_vector(0 to 7);
        :
p1: process (addr)
    begin
        if addr > x"0F" then
            step <= '1';
        else
            step <= '0';
        end if;
    end process;
```

```
signal step: bit;
signal addr: bit_vector(0 to 7);
        :
p2: process (addr)
    begin
        if addr > x"0F" then
            step <= '1';
        end if;
    end process;
```

P2 has an implicit memory

Sequential Statements

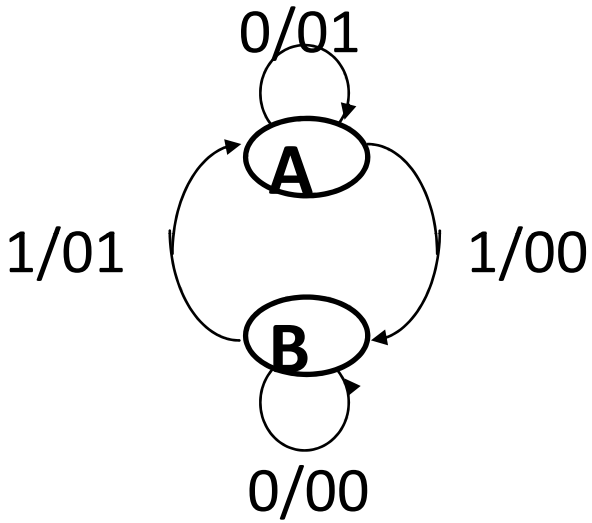
if-then-else (cntd.)

```
architecture mux_arch of mux is
begin
mux4_1: process (a,b,c,d,s)
            begin
                if  s = "00" then          x <= a;
                elsif  s = "01" then       x <= b;
                elsif  s = "10" then       x <= c;
                else                       x <= d;
                end if;
             end process;
end mux_arch;
```

Sequential Statements

case-when

```
case p_state is
    when  A  =>  y <= '0'; z <= '1';
        if  x = '1'  then  n_state <= B;
        else               n_state <= A;
        end if;
    when B  =>  y <= '0'; z <= '0';
        if  x = '1'  then  n_state  <= A;
        else               n_state <= B;
        end if;
end case;
```



inputs: x

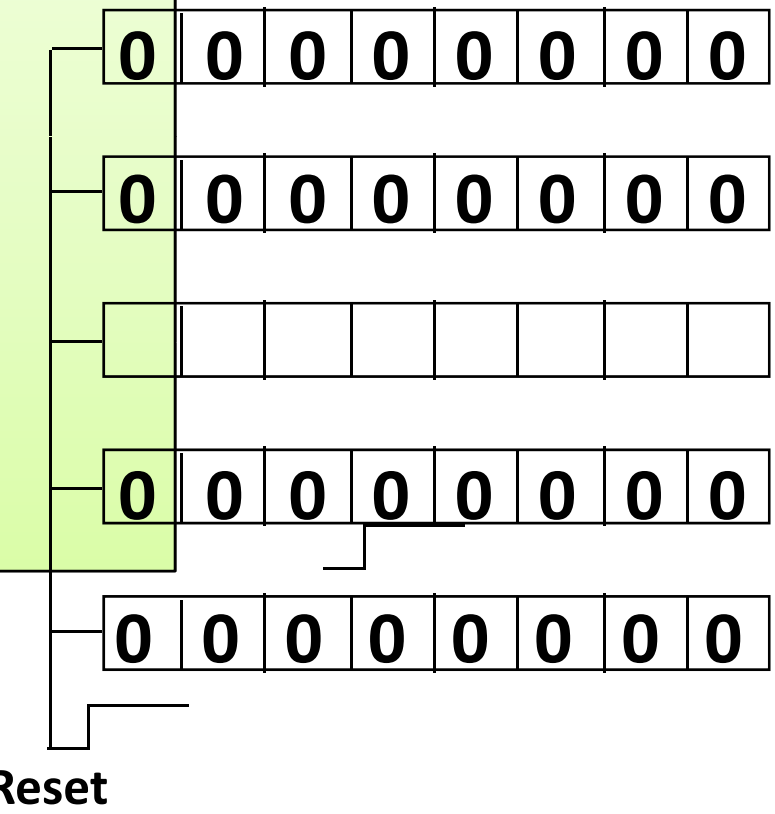
outputs: y,z

Sequential Statements : for loop

```
type register is bit_vector(7 downto 0);
type reg_array is array(4 downto 0) of register;
signal fifo: reg_array;
process (reset)
begin
    if reset = '1' then
        for i in 4 downto 0 loop
            if i = 2 then next;
            else    fifo(i) <= (others => '0');
            end if;
        end loop;
    end if;
end process;
```

Sequential Statements : for loop

```
type register is bit_vector(7 downto 0);
type reg_array is array(4 downto 0) of register;
signal fifo: reg_array;
process (reset)
begin
    if reset = '1' then
        for i in 4 downto 0 loop
            if i = 2 then next;
            else    fifo(i) <= (others => '0');
            end if;
        end loop;
    end if;
end process;
```



Sequential Statements : while loop

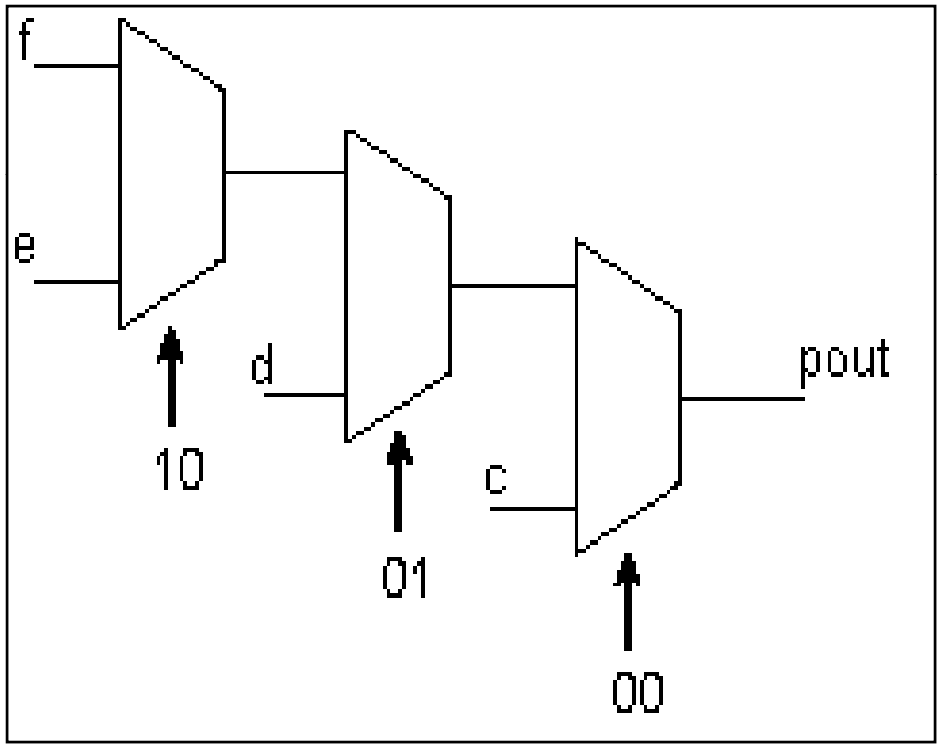
```
type register is bit_vector(7 downto 0);
type reg_array is array(4 downto 0) of register;
signal fifo: reg_array;
process (reset)
    variable i: integer := 0;
begin
    if reset = '1' then
        while i <= 4 loop
            if  i /= 2 then  fifo(i) <= (others => '0');
            end if;
            i := i + 1;
        end loop;
    end if;
end process;
```

Functions and Procedures

* High level design constructs that are most commonly used for:
    - Type conversions
    - Operator overloading
    - Alternative to component instantiation
    - Any other user defined purpose
* The subprograms of most use are predefined in:
    - IEEE 1076, 1164, 1076.3 standards

Mutually exclusive conditions

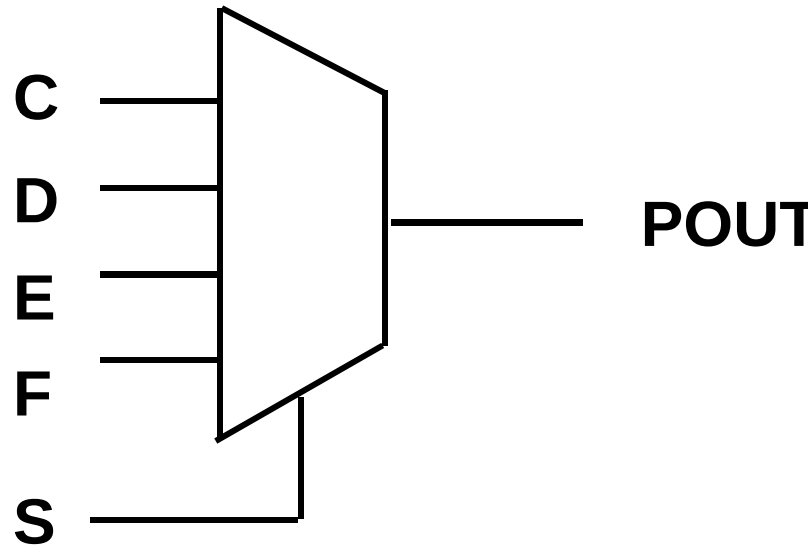
```
myif_pro: process (s, c, d, e, f)
begin
   if s = "00" then
      pout <= c;
   elsif s = "01" then
      pout <= d;
   elsif s = "10" then
      pout <= e;
   else
      pout <= f;
   end if;
end process myif_pro;
```



This priority is useful for timings.

Use a case for mutually exclusive things

```
mycase_pro: process (s, c, d, e, f)
  begin
    case s is
    when "00" =>
      pout <= c;
    when "01" =>
      pout <= d;
    when "10" =>
      pout <= e;
    when others =>
      pout <= f;
    end if;
  end process mycase_pro;
```



There is no priority with case.

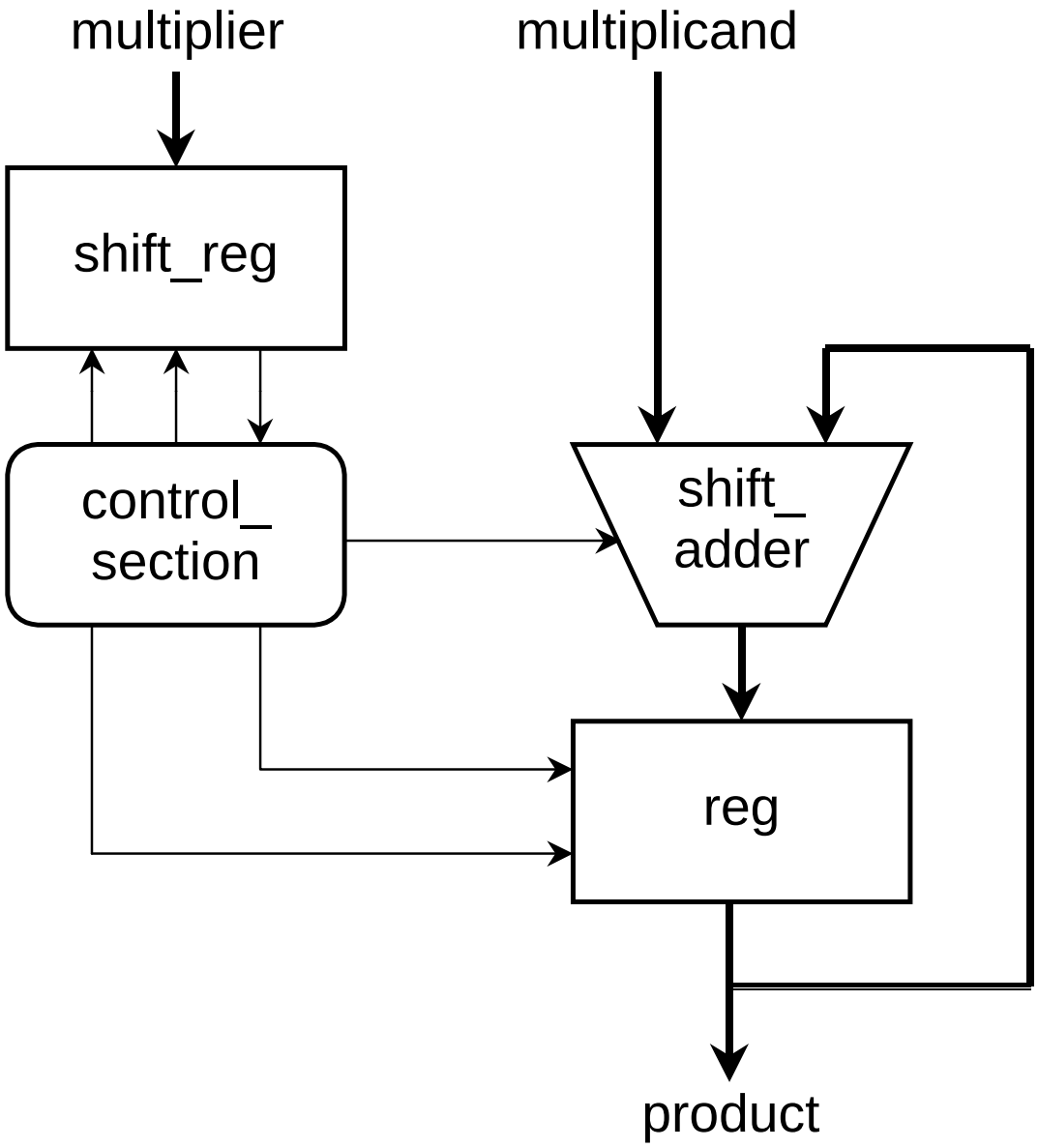
4 ways to DO IT -- the VHDL way

- Dataflow when-else, when
- RTL Clocked, Transfers
- Behavioral Process, For loop, While loop, if then else
- Structural Using Components
- Mixed :

Mixed Behavior and Structure

- An architecture can contain both behavioral and structural parts
  - process statements and component instances
    - collectively called *concurrent statements*
  - processes can read and assign to signals
- Example: register-transfer-level (RTL) Model
  - data path described structurally
  - control section described behaviorally

Mixed Example



Thanks