

VHDL

by

Prof. Sujata Wakchaure

Introduction to HDL

- With the increasing device densities, the choice of traditional methods has become limited
- It is must for IC designers to go for some Electronic Design Automation (EDA) tool
- Hardware Description Language (HDL) is one of the outcome of EDA tool
- HDL describes hardware structures and behaviors

VHDL

- VHDL (Very High Speed Integrated Circuit HDL)
- It is a programming language that describes a logic circuit by function, data flow, behavior and/ structure
- It was developed in the 1980s under Department of Defense (**DoD**) sponsorship
- A VHDL program is a collection of **modules**
- VHDL is **not case sensitive & free format Language**
- It is mainly used for the development of Application Specific Integrated Circuit (**ASIC**)

VHDL Terms

1. Synthesis:

- Converting code to Circuit

2. Simulation:

- Giving I/P & verifying the functionality of Circuit

3. Test bench:

- Helps in generating I/P signal for simulation

VHDL Comment Operator

- To include a comment in VHDL, use the comment operator

-- This is a comment

VHDL Statement Terminator

- Each VHDL Statements is terminated using a semicolon

;

Signal Assignment Operator

- To assign a value to a signal data object in VHDL, we use the
signal assignment operator

<=

Example:

```
Y <= '1';    -- signal y is assigned the value ONE
```

Fundamental sections of a basic VHDL code

Library
Declaration

Entity

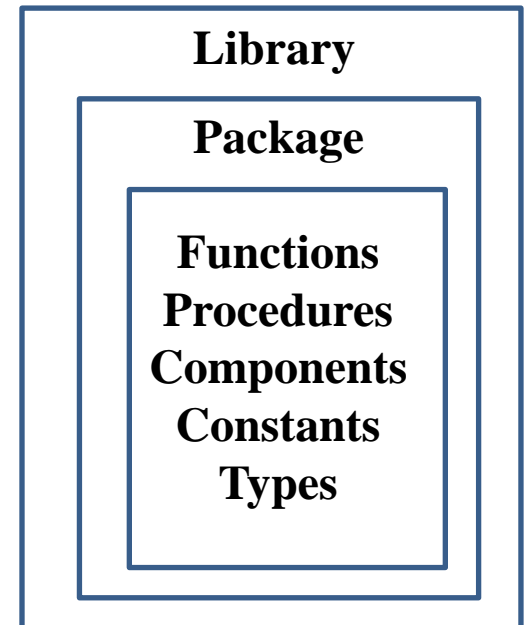
Architecture

Library Declaration

- Contains a list of all libraries to be used in the design
- It is a collection of commonly used piece of code
- Syntax
LIBRARY library_name;
USE
Library_name.package_name.package_parts;

Library Declaration (Contd..)

- At least 3 packages from 3 different libraries are usually needed in a design
 - `ieee.std_logic_1164` (from ieee library)
 - Standard (from standard library)
 - Work (from work library)
- Standardized design libraries are included before entity declaration.



Entity

- **Entity**: shows **inputs** and **outputs**
- It is a list with specifications of all I/P and O/P pins (Ports) of the ckt

- **Syntax:**

```
ENTITY entity_name IS
```

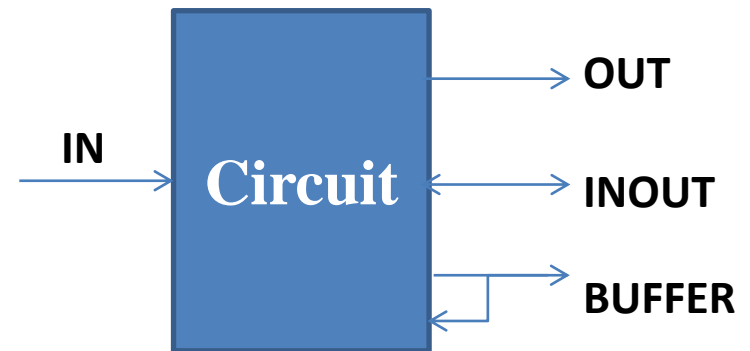
```
    PORT ( Port_name: signal_mode  signal_type;
```

```
           Port_name: signal_mode  signal_type);
```

```
END entity_name;
```

Entity (Contd..)

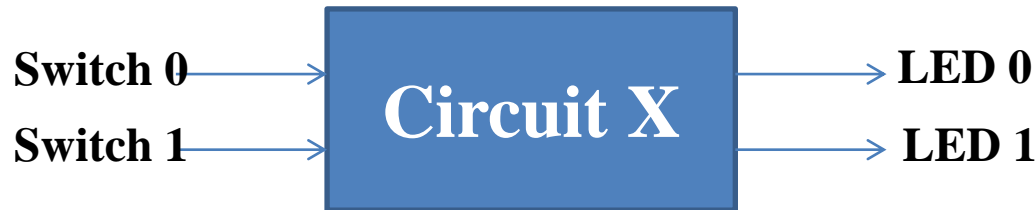
- The mode of signal can be
 - IN
 - OUT
 - INOUT
 - BUFFER



- Example of AND Gate
ENTITY and_gate IS
PORT (a, b : IN BIT;
x: OUT BIT);
END and_gate

Example

- Make a small ckt that has 2 I/Ps Switch 0 & switch 1 and the O/P of ckt is X that is connected to 2 LEDs LED0 & LED 1



Example (Contd..)

Library IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

Entity Switch_LED is

PORT (switch_0: in STD_LOGIC;

switch_1: in STD_LOGIC;

LED_0: out STD_LOGIC;

LED_1: out STD_LOGIC);

End Switch_LED;

Signal Mode



Signal Type i.e. Data type



Architecture

- **Architecture** : internal description
- It specifies what is going on inside the circuit
- It denotes the description of how the ckt should behave or function
- Syntax:
ARCHITECTURE architecture_name OF entity_name IS
[declarations]
BEGIN

END architecture_name;

Architecture (Contd..)

- Architecture has 2 parts:
 - 1. Declarative part:**
 - Where signals & constants are declared
 - 2. Code Part:**
 - From begin to end architecture

Example

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

```
ARCHITECTURE my_arch of and_gate IS
```

```
BEGIN
```

```
  x <= a and b;
```

```
END my_arch;
```

Assignment Statement



Code Part

Example

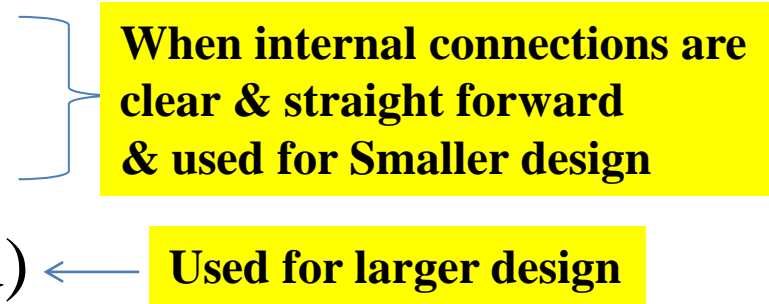
```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

Entity Switch_LED is
    PORT (switch_0: in STD_LOGIC;
          switch_1: in STD_LOGIC;
          LED_0: out STD_LOGIC;
          LED_1: out STD_LOGIC);
End Switch_LED;

ARCHITECTURE Behavioral OF Switch_LED IS
    BEGIN
        LED_0 <= switch_0;
        LED_1 <= switch_1;
    END behavioral;
```

Architecture levels

– Architecture can be written in one of **three** different detail levels

- Structural (**very** detailed)
 - Dataflow (**less** detailed)
 - Behavioral (**least** detailed)
- 
- When internal connections are clear & straight forward & used for Smaller design
- Used for larger design

Few Key Terms of VHDL

1. Assigned:

- E.g. $X \leq Y$
- i.e. X is assigned the value of Y

2. Read/Reading a value:

- $X \leq Y$;
- X is reading Y



Take (Read)

Give (Write)

3. Drivers:

- It is a signal or port that gives a logic value to a signal or port & the person who is giving the value is called as **Driver**

MODES

- **4 types of mode are used in VHDL:**

1. Mode **IN**

- Value can be read but not assigned

2. Mode **OUT**

- Value can be assigned but not read

3. Mode **INOUT**

- Value can be read and assigned

4. Mode **BUFFER**

Data Types

1. BIT Data Type:

- Supports the value '0' & '1'

e.g.

entity driver is

port (A: in BIT;

B: out BIT;

data: inout BIT);

end driver;

Data Types (Contd..)

2. Boolean:

- Supports literals FALSE & TRUE

e.g.

```
variable error_flag: boolean:=true
```

Data Types (Contd..)

3. STD_LOGIC:

- It is most commonly used data type
 - It is defined in the std_logic_1164 package of IEEE library
 - Type std_logic ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
- 'U' : Uninitialized
 - 'X' : Unknown
 - '0' : Logic 0
 - '1' : Logic 1
 - 'Z' : High Impedance
 - 'W' : Unknown
 - 'L' : Low Logic 0
 - 'H' : Low Logic 1
 - '-' : Don't Care

e.g.

```
Entity Driver is  
port (A: in STD_LOGIC;  
       B: in STD_LOGIC);  
End Driver
```


Structural Level

- It describes the digital system as gates or as components blocks interconnected to perform the desired operations
- It is a set of interconnected components (to represent structure)
- Structural level is primarily the graphical representation of the digital system

Structural Level (Contd..)

Entity half_adder is

```
port (A,B: in BIT;  
      Sum, Carry: out BIT);
```


End half_adder;

architecture Structure_HA of half_adder is

Component xor1

```
port(P, Q: in BIT;  
      R: out BIT);
```


End Component



Defining XOR gate

Structural Level (Contd..)

```
Component and1  
    port(X, Y: in BIT;  
          Z: out BIT);  
End Component
```



Defining AND gate

```
begin  
    X1 : xor1 port_map(A, B, sum);  
    A1 : and1 port_map (A, B, carry);  
  
End Structure_HA;
```

Data Flow Modeling

- It is a set of concurrent assignment statements (to represent dataflow)
- Concurrent signal assignment statements are concurrent statements, and therefore, the ordering of these statements in an architecture body is not important
- E.g. The dataflow model for the HALF_ADDER is described using two concurrent signal assignment statements

architecture Dataflow of HALF_ADDER is

begin

SUM <= A **xor** B **after** 8 ns;

CARRY <= A **and** B **after** 4 ns;

end Dataflow;

For **simulation** purposes only,
you may specify a delay

You **cannot** specify a delay
for **synthesis** purposes

VHDL Data Objects

- We will primarily use the following VHDL data objects:

Signals

Constants

Variables

Signals

- Ports declared in an entity are accessible as **signals** within the architecture(s) and do not need to be re-declared
- Signals are used inside a design
- Signals are data objects in which the value of the object can be changed
- It represents wires within a circuit
- Signals may not be declared in a processor subprogram
- Signals are supported for synthesis, providing they are of a type acceptable to the logic synthesis tool

Signals (Contd..)

- signals use the **<= assignment symbol**

- e.g.

architecture and_gate of anding is

signal a, b, temp: std_logic;

begin

temp <= a and b;

c <= temp and D

end and_gate;

Constants

- Constants are data objects in which the value of the object cannot be changed
- They are defined within an architecture or process declaration block.
- They cannot be implemented in hardware

Constants (Contd..)

Syntax:

constant name: type := value;

Example:

constant s0: std_logic_vector(1 downto 0):= “01”;

Notes:

1. Use a set of single apostrophes to enclose a single bit (e.g. ‘1’).
2. Use a set of quotations to enclose multiple bits (e.g. “01”).

Variables

- Variables are data objects in which the value of the object can be changed
- This change occurs instantaneously
- Variables can only be defined within a *process declaration* block
- They cannot be implemented in hardware

Rules of Variables

- Variables can only be used inside processes
- Any variable that is created in one process cannot be used in another process
- Variables need to be defined after the keyword process but before the keyword begin
- Variables are assigned using the **:= assignment symbol**
- Variables that are assigned immediately take the value of the assignment

Sequential Statements

Process Statements

- In VHDL, sequential statements are executed within a **process** block.

- **Syntax:**

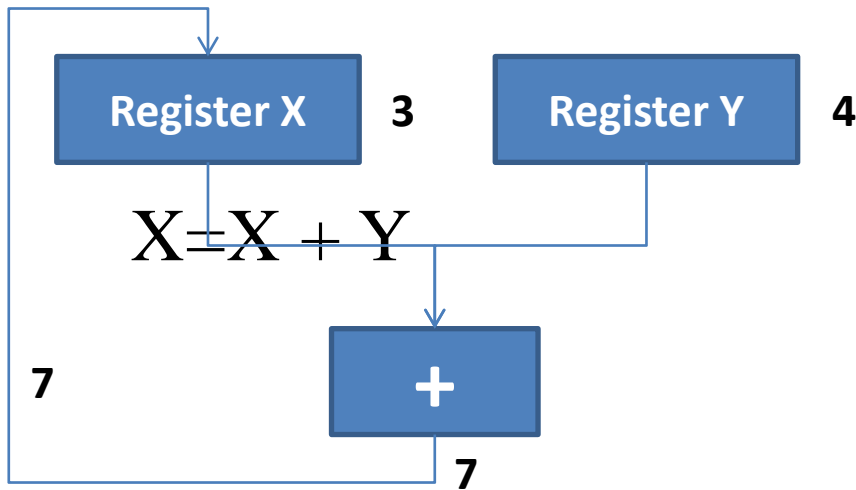
```
[label:] process (sensitivity list)
    constant or variable declarations
begin
    sequential statements;
end process [label];
```

Variables vs. Signals in VHDL

Variables	Signals
use the <code>:=</code> assignment symbol	use the <code><=</code> assignment symbol
Variables can only be used inside processes	signals can be used inside or outside processes
Any variable that is created in one process cannot be used in another process	Signals can be used in multiple processes though they can only be assigned in a single process
Variables need to be defined after the keyword <code>process</code> but before the keyword <code>begin</code>	Signals are defined in the architecture before the <code>begin</code> statement
Variables that are assigned immediately take the value of the assignment	Signals depend on if it's combinational or sequential code to know when the signal takes the value of the assignment

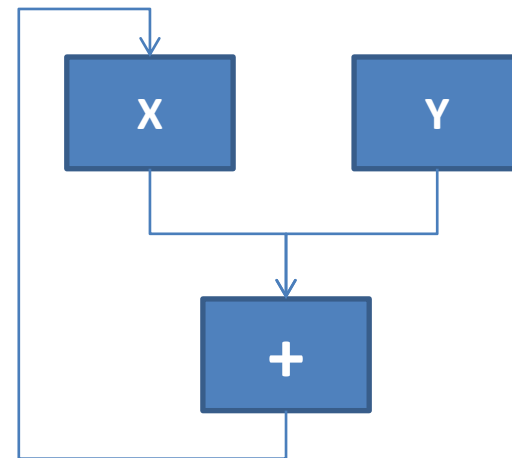
Concurrent Statements

- Example: IF $X=3$ & $Y=4$
- In C Language $X = X + Y$



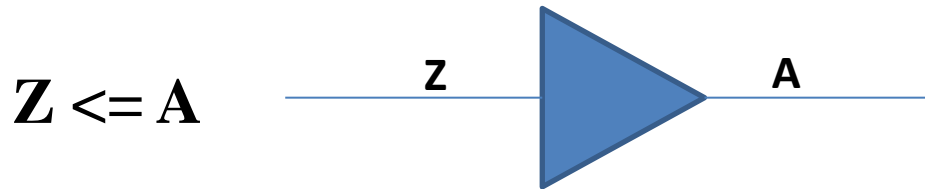
In VHDL

$X \leq X + Y$



Drivers

- It is a signal or port that gives a logic value to a signal or port & the person who is giving the value is called as **Driver**



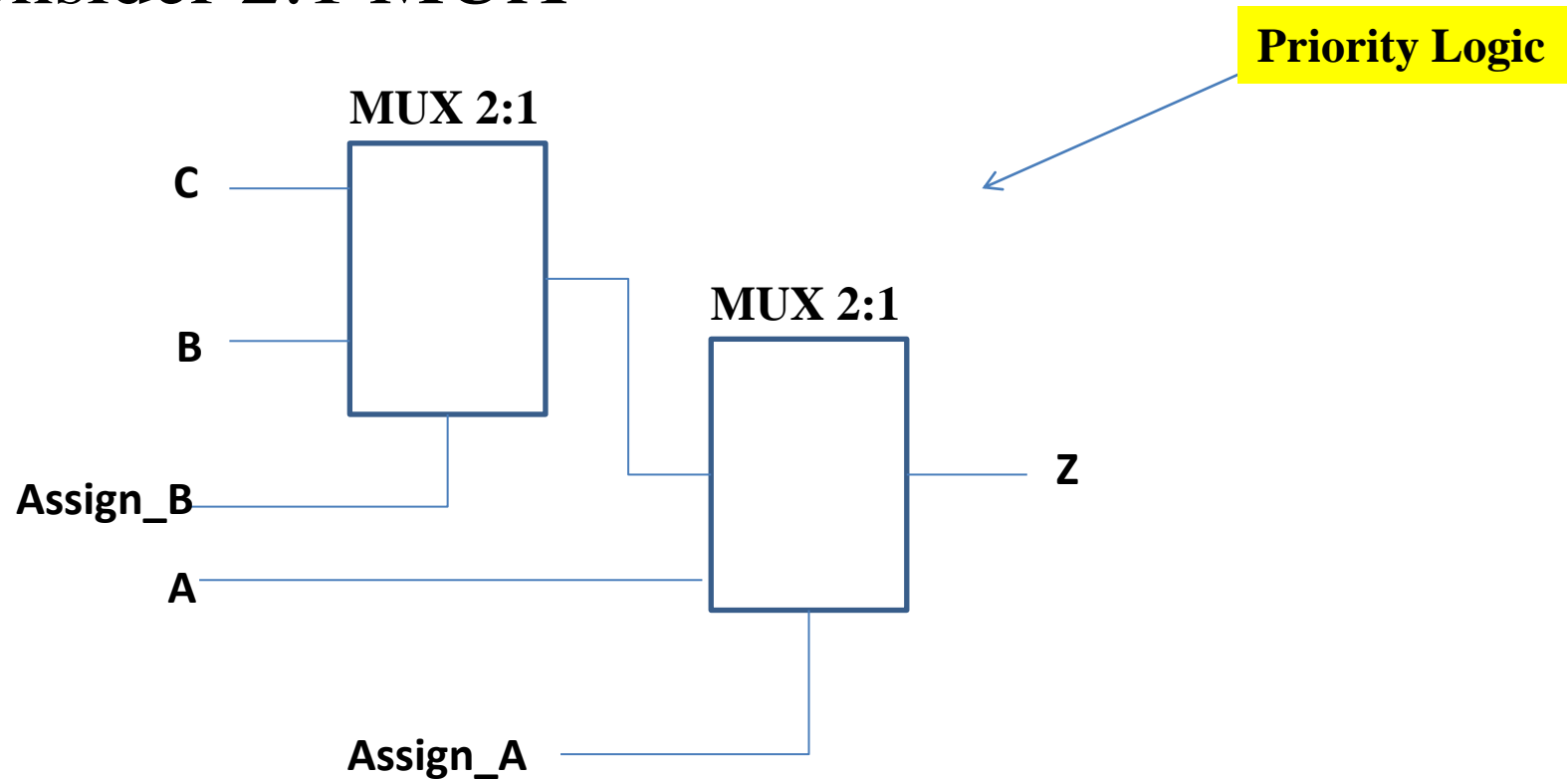
- e.g.**
architecture concurrent of Multiple is
 signal Z, A, B, C, D: std_logic;
 begin
 $Z \leq A$ and B;
 $Z \leq C$ and D;
 end concurrent;

When Statement

- When Statement (When – else Statement):
- It gives Priority logic
- e.g.
A when Assign_A = '1' **else**
B when Assign_B = '1' **else**
C;

When Statement (Contd..)

- Consider 2:1 MUX

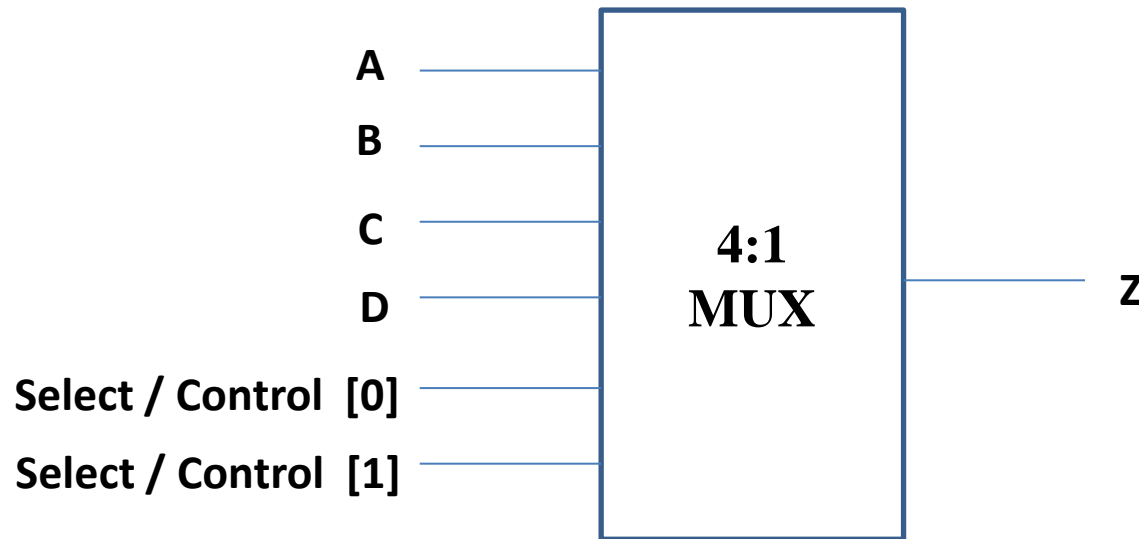


With – Select Statement

- It gives Parallel Logic
- Syntax:
 with choice_expression select
 target <= expression 1 when choice 1
 expression 2 when choice 2
 expression N when choice N;
- With-select stmt evaluates choice expression & compares that value to each choice value

With – Select Statement (Contd..)

- Consider 4:1 MUX



With – Select Statement (Contd..)

Signal A, B, C, D, Z: std_logic;

Signal Control_0, Control_1: std_logic;

with Control select

Z <= A when “00”,

 B when “01”,

 C when “10”,

 D when “11”,

 ‘O ‘ when others;

Processes

- It is a sequential section of VHDL code
- Statements in process are executed one after another

- e.g.

```
Process()  
begin  
-----  
end Process;
```

- The process as a whole will execute concurrently

- e.g.

```
Process A()  
begin  
-----  
end Process A;
```

```
Process B()  
begin  
-----  
end Process B;
```

Processes (Contd..)

- Only sequential statements can use variables
- “Process” is primary concurrent VHDL statements used to describe sequential behavior

Processes (Contd..)

- How to write a process statement:
architecture sequential of multiple is
 signal Z, A, B, C, D: std_logic;
 begin
 process (A, B, C, D)
 begin
 Z <= A and B;
 Z <= C and D;
 end process;
 end sequential;



Sensitivity List

Rules of Process

1. If you have one signal, then your process will place only one driver on that signal

e.g. $Z \leq A \text{ and } B;$
 $Z \leq C \text{ and } D;$

2. In process you can have either sensitivity list or wait statement
3. You can have only signal names which you can read

Sequential Statements

1. If Statements: (similar to when-else statement)

Priority Logic

- Syntax:

```
if condition1 then
    {sequential statement}
elseif condition2 then
    {sequential statement}
else
    {sequential statement}
end if;
```

If Statements (Contd..)

- Example:

```
process (A, B, C, X)
begin
    if (X = "0000") then
        Z <= A;
    elseif (X = "0101") then
        Z <= B;
    else
        Z <= C;
    end if;
end process;
```

Case Statement

- Also known as Case-Select statement
- Similar to with-select statement
- Syntax:

Parallel Logic

case expression is

when choice1 => {statement}

when choice2 => {statement}

when others => {statement}

end case;

Case Statement (Contd..)

- Example:

```
process (sel, A, B, C, D)
begin
  case sel is
    when 0 => Y <=A;
    when 1 => Y <=B;
    when 2 => Y <=C;
    when others => Y <=D;
  end case
end process;
```

Rules about a case

Process (A, B, C, X)

begin

case X is

when 0 to 4

Z <=B;

when 5

Z <=C;

when 7 | 9

Z <=A;

when others

Z <=O;

end case;

end process;

Invalid case statements

- e.g. Signal VALUE: INTEGER range 0 to 15;
Signal OUT : BIT;

1. Case VALUE is
end Case;
2. Case VALUE is
when 0 => OUT <= '1';
when 1 => OUT <= '0';
end Case;
3. Case VALUE is
when 0 to 10 => OUT <= '1';
when 5 to 15 => OUT <= '0';
end Case;

NULL Statement

- Does not perform any action
- Used to indicate that when some conditions are met no action is to be performed
- e.g. Case a is
 - when “00” => q1 <= ‘1’;
 - when “01” => q2 <= ‘1’;
 - when “10” => q3 <= ‘1’;
 - when OTHERS <= null;

Types of Processes

1. Combinatorial \Rightarrow combinational logic
2. Clocked \Rightarrow sequential logic

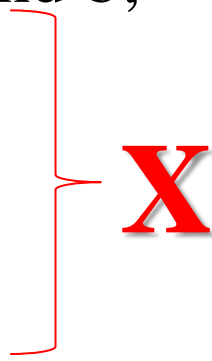
Clocked Process

- e.g. Process (clk)
 begin
 if (clk' event and clk='1') then
 Q <= D;
 end if;
 end process;

Rules of Clock Process

1. Clocked processes having an else clause will generate wrong H/W

e.g. Process (clk)
 begin
 if (clk' event and clk='1') then
 out <= a and b;
 else
 out <= c;
 end if;
 end process;



Rules of Clock Process (Contd..)

2. You can write any other if statement above

if (clk' event and clk='1')

e.g. if (reset = 1)

end if;

if (clk' event and clk='1')

end if



This statement should be last in process

Rules of Clock Process (Contd..)

Example of Process with/without **wait**

```
process (clk,reset)
begin
    if (reset = '1') then
        A <= '0';
    elsif (clk'event and clk = '1') then
        A <= 'B';
    end if;
end process;
```

```
process
begin
    if (reset = '1') then
        A <= '0' ;
    elsif (clk'event and clk = '1') then
        A <= 'B';
    end if;
    wait on reset, clk;
end process;
```

Behavioral Modeling

- Also known as **High Level Descriptions**
- It is a set of sequential assignment statements (to represent behavior)
- The behavioral style of modeling specifies the behavior of an entity as a set of statements that are executed sequentially in the specified order
- This set of sequential statements, that are specified inside a process statement, do not explicitly specify the structure of the entity but merely specifies its functionality.
- A process statement is a concurrent statement that can appear within an architecture

Behavioral Modeling (Contd..)

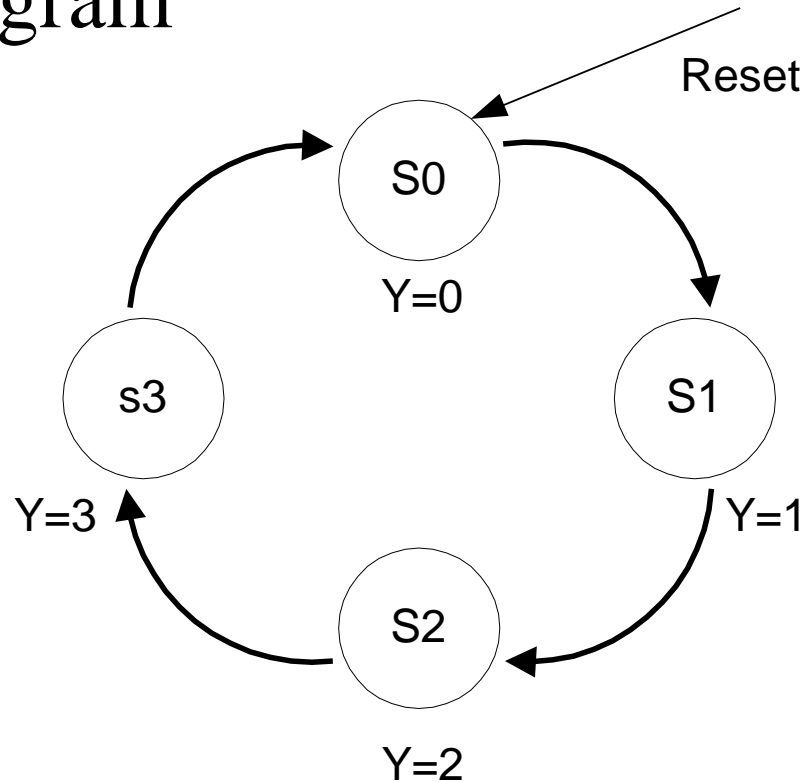
- e.g.
architecture behavioral of and_gate is
begin
 process (a, b)
 begin
 if a='1' and b='1' then
 c <= '1';
 else
 c <= '0';
 end if;
 end process;
end behavioral;

Dataflow Vs. Behavioral Vs. Structural

Dataflow Modeling	Behavioral Modeling	Structural Modeling
Set of Concurrent Statement	Set of Sequential Statements	Closest to Hardware interconnection Set of interconnected components
Logic equations	Addition of 2 binary numbers	Interconnections of gate
Algorithmic Level	Gate level	RTL (Register Transfer Level) Level
Needs Boolean equation to design specification	Needs truth table as design specification	Needs logic diagram as design specification

VHDL Code for 2-bit Up Counter

- State Diagram



VHDL Code for 2-bit Up Counter (Contd..)

- State Table

ps	ns	y
S0	S1	0
S1	S2	1
S2	S3	2
S3	S0	3

Let S0 = 00

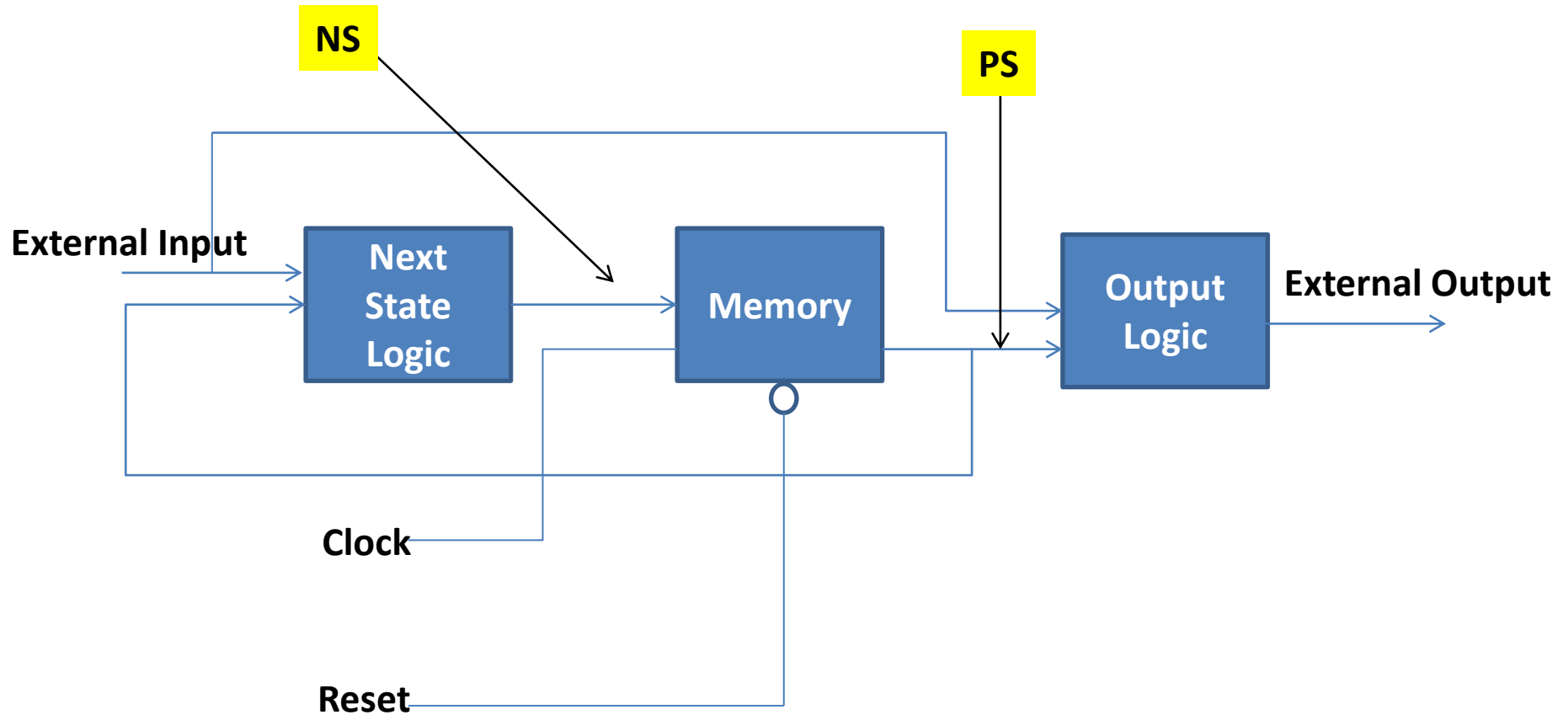
S1 = 01

S2 = 10

S3 = 11

Let S0 = Reset State

Recall Sequential Ckt



VHDL Code for 2-bit Up Counter (Contd..)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- define entity
entity Counter1 is
    port ( clk, reset: in std_logic;
          count: out std_logic_vector(1 downto 0)
        );
end entity Counter1;
```

VHDL Code for 2-bit Up Counter (Contd..)

-- define architecture

architecture Counterarch of Counter1 is

-- define constants

constant s0: std_logic_vector(1 downto 0) := "00";

constant s1: std_logic_vector(1 downto 0) := "01";

constant s2: std_logic_vector(1 downto 0) := "10";

constant s3: std_logic_vector(1 downto 0) := "11";

signal ns, ps: std_logic_vector(1 downto 0);

begin

VHDL Code – Next State Logic

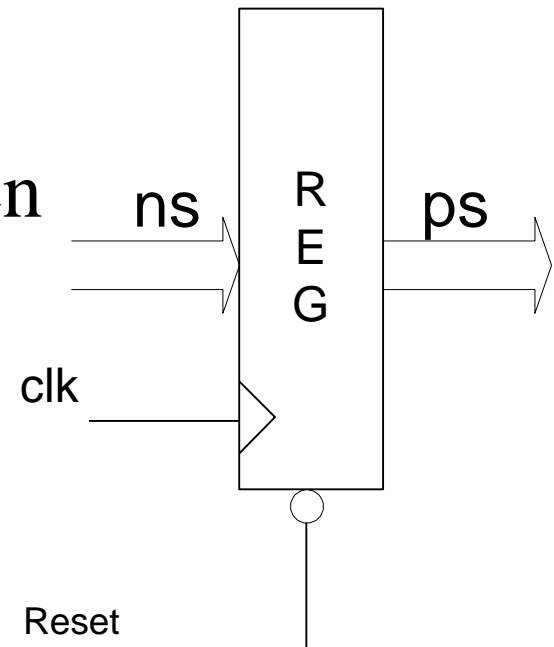
```
process ( ps)
begin
    ns <= s0; -- This is the default output
    case ps is
        when s0 => ns <= s1;
        when s1 => ns <= s2;
        when s2 => ns <= s3;
        when s3 => ns <= s0;
        when others => ns <= s0;    -- default condition
    end case;
end process
```

Note: we only need to “describe” the behavior
VHDL will “figure out” the functional relationships

VHDL Code – Memory Logic

- -- This process includes the registers

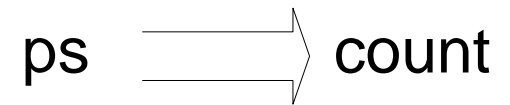
```
reg: process (clk, reset, ns)
begin
    if(reset = '0') then
        ps <= s0;
    elsif (clk'event and clk = '1') then
        ps <= ns;
    end if;
end process reg;
```



VHDL Code – Output Logic

-- Use concurrent statement to implement
Output Logic

```
count <= ps;
```



```
end architecture Counterarch;
```