Section D

- Design of Combinational Elements and regular array logic
- VHDL Programming

TOPIC COVERED

- **DESIGN OF COMBINATIONAL ELEMENTS & REGULAR ARRAY LOGIC :** NMOS PLA –
- Programmable Logic Devices - Finite State Machine PLA – Introduction to FPGA**.**
- **VHDL PROGRAMMING:** RTL Design – Combinational logic – Types – Operators – Packages –
- Sequential circuit – Sub-programs – Test benches. (Examples: address, counters, flipflops, FSM,
- Multiplexers / De-multiplexers).

VHDL Introduction

VHDL Introduction

- V- VHSIC
  - Very High Speed Integrated Circuit
- H- Hardware
- D- Description
- L- Language

VHDL Benefits

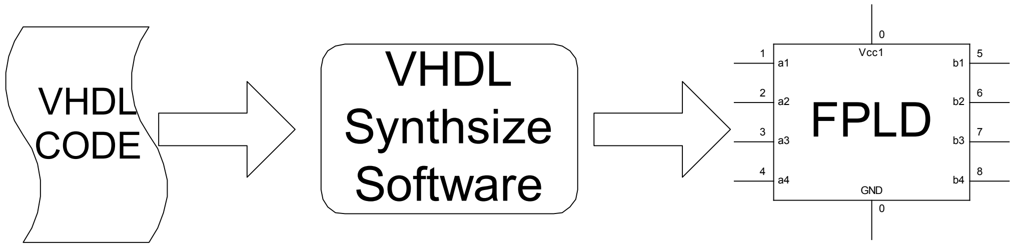
1. Public Standard
2. Technology and Process Independent
   - Include technology via libraries
3. Supports a variety of design methodologies
   1. Behavioral modeling
   2. Dataflow or RTL (Register Transfer Language) Modeling
   3. Structural or gate level modeling

VHDL Benefits (cont)

4. Supports Design Exchange
   - VHDL Code can run on a variety of systems
5. Supports Design Reuse
   - Code “objects” can be used in multiple designs
6. Supports Design Hierarchy
   - Design can be implemented as interconnected submodules

VHDL Benefits (cont)

7. Supports Synchronous and Asynchronous Designs
8. Supports Design Simulation
   - Functional (unit delay)
   - Timing ("actual" delay)
9. Supports Design Synthesis
   - Hardware implementation of the design obtained directly from VHDL code.



10. Supports Design Documentation
    - Original purpose for VHDL – Department of Defense

VHDL Design Units

- Entity Declaration
  - Describes external view of the design (e.g. I/O)
- Architecture Body (AB)
  - Describes internal view of the design
- Configuration Declaration
- Package Declaration
  - Library Declaration
- Package Body

Architecture Body (AB)

- The architecture body contains the internal description of the design entity. The VHDL specification states that a single design entity can contain multiple architecture bodies. Each AB can be used to describe the design using a different level of abstraction.

VHDL Statement Terminator

Each VHDL Statements is terminated using a semicolon

;

VHDL Comment Operator

To include a comment in VHDL, use the comment operator

```
--  This is a comment

 --  This is an example of a comment
y <= 0;  -- can occur at any point
```

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
```

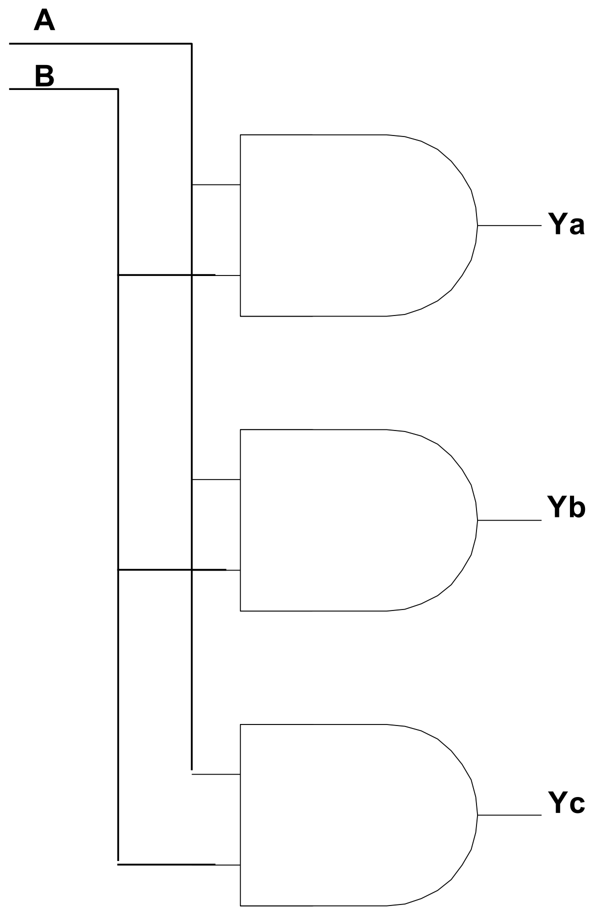
Complete AND GATE Example

```
Library altera;
Use altera.maxplus2.all;
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Entity and_example is
   port(a,b: in std_logic;
          ya,yb,yc: out std_logic);
End entity and_example;
Architecture test of and_example is
  begin
    --- dataflow model (ya)
        ya <= a and b;
    --  structural model  (yb)
       and2:a_7408 port map(a,b,yb);
```

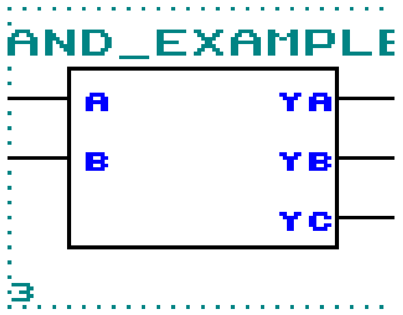
```
--  behavioral model  (yc)
    process(a,b)
      begin
        yc <= ‘0’;
        if((a=‘1’) and (b = ‘1’)) then
         yc <= ‘1’;
         else  yc <= ‘0’;
         end if;
end process;
End architecture test;
```

AND GATE Example (cont)

When synthesized, we obtain the following logic circuit



Synthesis tool creates three AND gates.



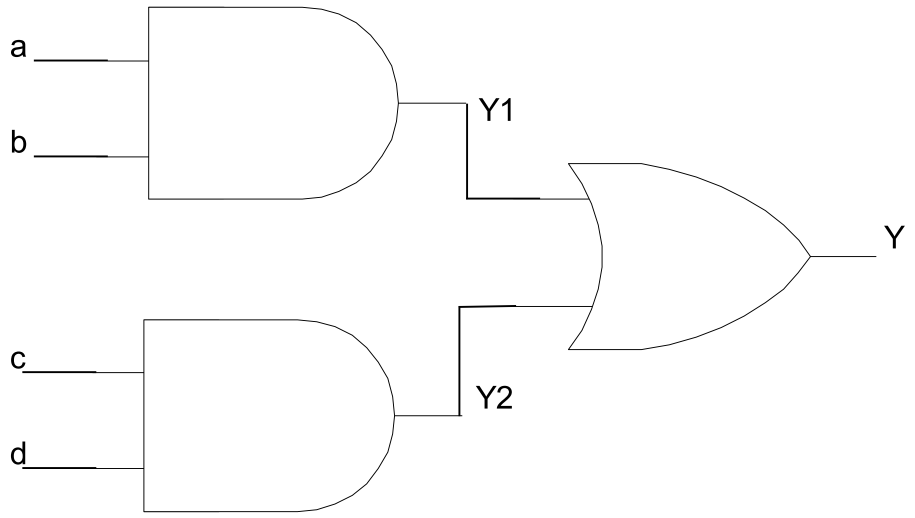
Maxplus II Block Diagram

VHDL Example - Hardware

- It is important to remember that VHDL is a “hardware” language, so you must think and code in “hardware.”
- Statements within the architecture body run “concurrently.” That is, order does not matter!!!
    - We’ll introduce “sequential” statements later when I introduce “process blocks”

VHDL Example – Hardware

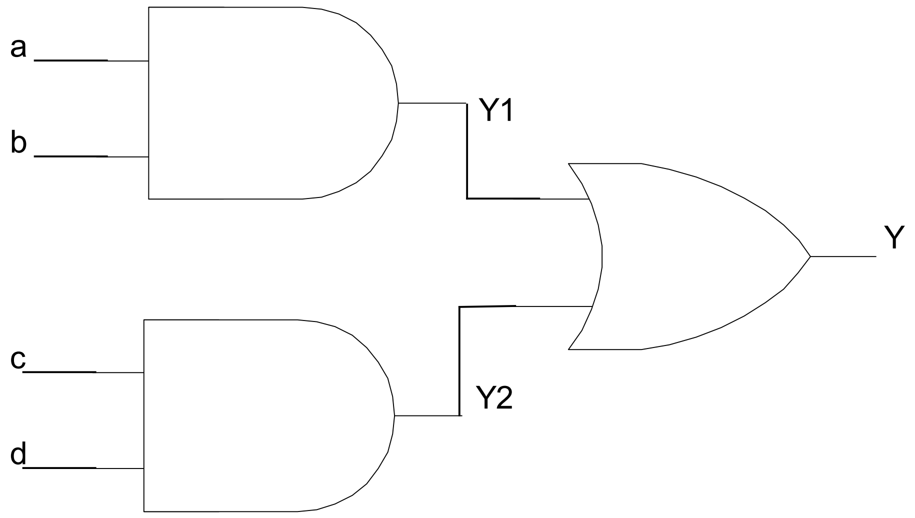
- Example – Logic Circuit



```
--  Code Fragment A
Architecture test of example is
  begin
      y1 <= a and b;
      y2 <= c and d;
      y   <= y1 or y2;
  end architecture test;
```

VHDL Example – Hardware

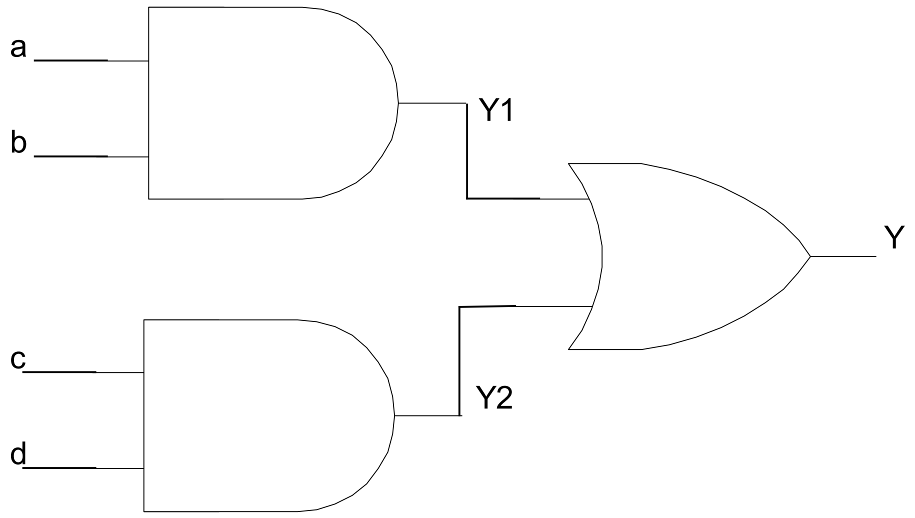
- Example – Logic Circuit



```
--  Code Fragment B
Architecture test of example is
  begin
      y   <= y1 or y2;
      y2 <= c and d;
      y1 <= a and b;

end architecture test;
```

VHDL Example – Hardware

- Example – Logic Circuit



```
-- Code Fragment C
Architecture test of example is
  begin
      y2 <= c and d;
       y   <= y1 or y2;
       y1 <= a and b;

end architecture test;
```

All three code fragments produce the same result

VHDL Syntax

VHDL Syntax – Entity Declaration

Describes I/O of the design. I/O Signals are called ***ports***.

The syntax is:

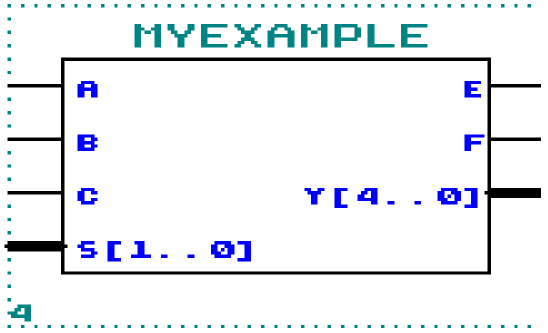
Entity design_name ***is***

  port(signal1,signal2,…..:mode type;
    signal3,signal4,…..:mode type***);***

End entity design_name;

VHDL Syntax – Entity Example

```
Entity my_example is
    port( a,b,c: in std_logic;
                   s: in std_logic_vector(1 downto 0);
                 e,f: out std_logic;
                   y: out std_logic_vector(4 downto 0));
end entity my_example;
```



Maxplus II Block Diagram

Architecture Body Syntax

- ***Architecture*** name ***of*** entity_name ***is***

  internal signal and constant declarations

  Begin

  Concurrent statement 1;

  Concurrent statement 2;

  Concurrent statement 3;

  Concurrent statement 4;

  End architecture name;

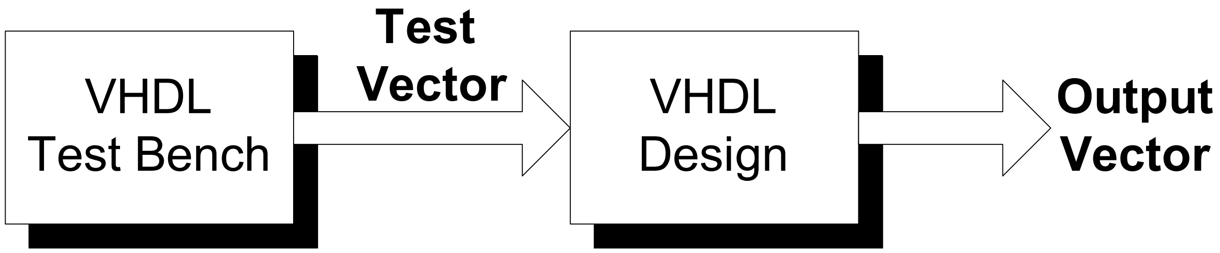
VHDL Program Template

```
Library altera;
Use altera.maxplus2.all;
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;

Entity design_name is
   port(signal1,signal2,…..:mode type;
         signal3,signal4,…..:mode type);
End entity design_name;
```

```
Architecture name of entity_name is
     internal signal and constant
   declarations
   Begin
        Concurrent statement 1;
        Concurrent statement 2;
        Concurrent statement 3;
        Concurrent statement 4;
   End architecture name;
```

Simple Concurrent Statements Assignment Operator

- Assignment operator <=
  - Ex: y <= a and b; -- defines a **AND** gate
  - For **simulation** purposes only, you may specify a delay.
    - Ex: y <= a and b after 10 ns;
  - This is useful if you want to also use VHDL to generate a known test waveform or vector. This is known as a "test bench." However, we will use Maxplus II to generate test vectors. Note, you **cannot** specify a delay for **synthesis** purposes.

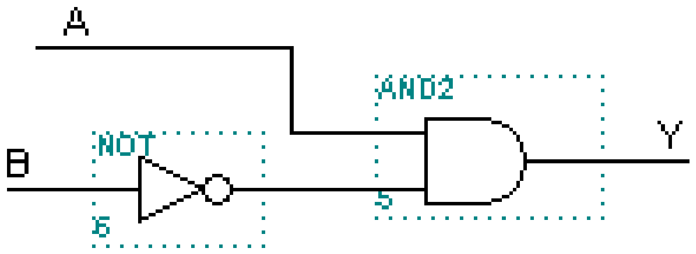


Simple Concurrent Statements Logical Operators

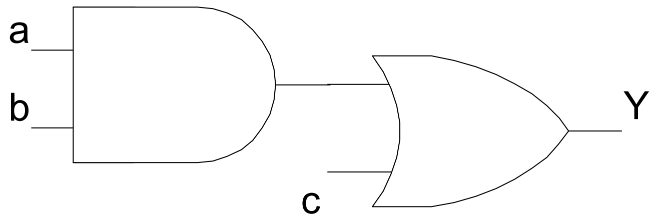
- Logical operators
  - ***And, or, nand, nor, xor, xnor, not***
  - Operates on std_logic or Boolean data objects
  - All operators (except for the not operator) require at least two arguments
  - Ex: y <= a and b; -- AND gate

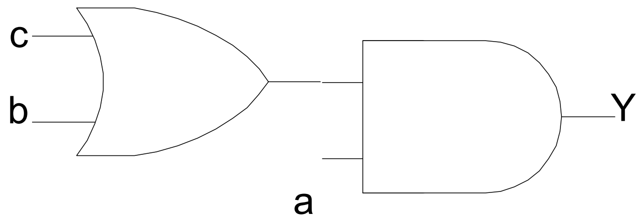
Simple Concurrent Statements Logical Operators

- Logical operators
  - Examples y <= a and not b;



  - Use parenthesis to define order of execution
    - Ex: y<= (a and b) or c; y <= a and (b or c);





Complex Concurrent Statements with-select-when

- with-select-when

  Syntax is

```
with select_signal select
    signal_name <= value1 when value1_of_select_sig,
                   value2 when value2_of_select_sig,
                   value3 when value3_of_select_sig,
                   value_default when others;
```

Complex Concurrent Statements With-select-when

- **Example**

```
---- library statements  (not shown)
 entity my_test is
    port( a3,a2,a1,a0: in std_logic_vector(3 downto 0);
                    s: in std_logic_vector(1 downto 0);
                    y: out std_logic_vector(3 downto 0));
 end entity my_test;
architecture behavior of my_test is
   begin
        with s select
      y <= a3 when “11”,
           a2 when “10”,
           a1 when “01”,
           a0 when others;  --   default condition
 end architecture behavior;
```

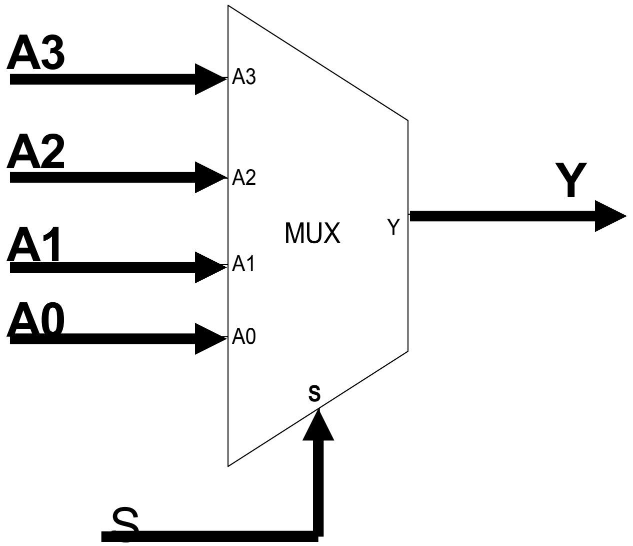
Complex Concurrent Statements With-select-when

- What is the logic expression for y?

$$y_n = a_{0n}\overline{s_1}\,\overline{s_0} + a_{1n}\overline{s_1}s_0 + a_{2n}s_1\overline{s_0} + a_{3n}s_1 s_0$$

$$n = 0, 1, 2, 3$$

- What is this in hardware?
  - A 4-bit 4X1 MUX



VHDL Data Objects

- VHDL is an Object Oriented Programming (OOP) Language. Objects can have values, attributes and methods. We will primarily use the following VHDL data objects:

Signals

Constants

Variables

Data Objects
Signals

- **Signals**
  Signals are data objects in which the value of the object can be changed. There is an implied or explicit delay between the signal assignment and when the signal is updated. We will use signals to represent nets (i.e. wires) in our circuits. They can be implemented in hardware. Signals are defined in port statements and architecture declaration blocks.

Data Objects
Constants

- **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.

Data Objects
Constants

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”).

Data Objects
Variables

- **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.

More about variables later

Sequential Statements
Process Statements

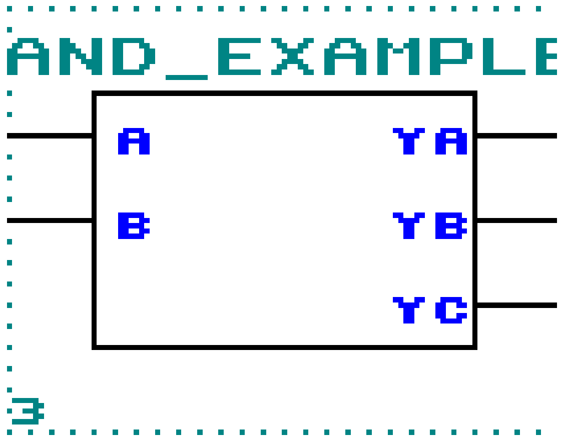
In VHDL, sequential statements are executed within a **process** block. Syntax is:

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

The sensitivity list contains all of the inputs to the process block.

Sequential Statements Process Statements (cont)

A process block is considered a **single** concurrent statement. Let's review our AND example

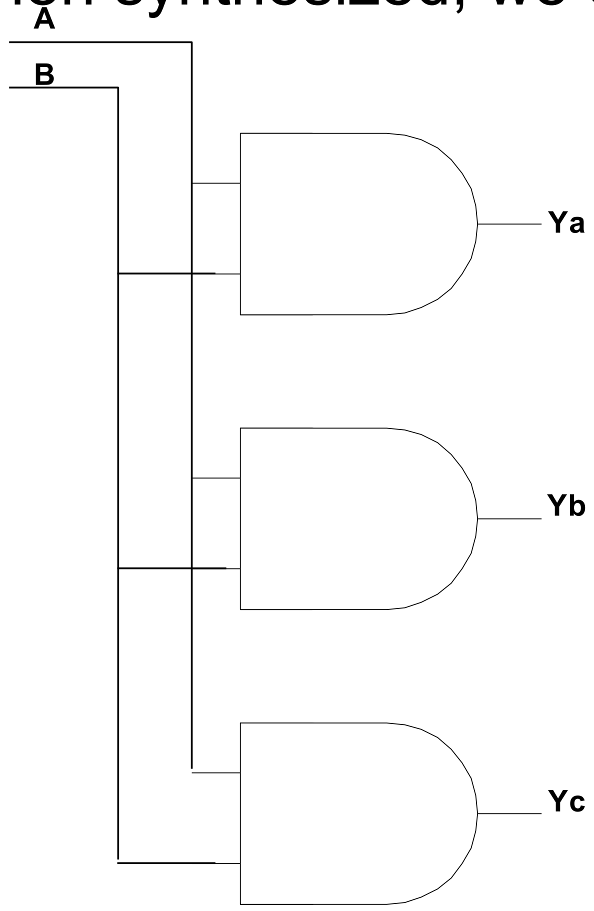


Sequential Statements
Process Statements - Example

```
---- library statements
entity and_example is
   port(a,b: in std_logic;
          ya,yb,yc: out std_logic);
End entity and_example;
Architecture test of and_example is
  begin
    --- dataflow model
         ya <= a and b;

    ---  structural model
       a_7408 port map(a,b,yb);

--  Process Block
    process(a,b)
      begin
        yc <= ‘0’;
        if ((a=‘1’) and (b = ‘1’)) then yc <= ‘1’;
        else yc <= ‘0’;
        end if;
      end process;
End architecture test;
```

Sequential Statements Process Statements

When synthesized, we obtain the following logic circuit



The process statement synthesizes into an AND gate just like the dataflow and structural statements. Note, the process block synthesized AND gate "runs" concurrently with the other synthesized AND gates.

Sequential Statements
Implied Registers

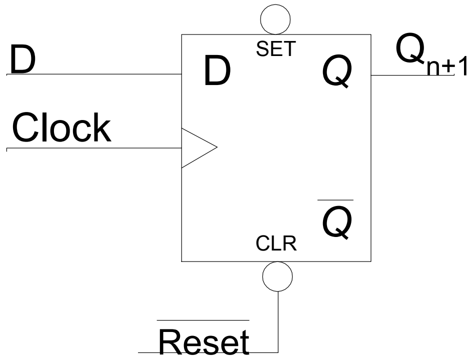
Registers

Sequential Statements Implied Registers

Positive edge triggered D-FF with asynchronous reset

```
Process (d,clock,reset)
   begin
      if (reset = ‘0’) then
            q <= ‘0’;
      elsif( clock’event and clock=‘1’) then
             q <= d;
      end if;
    end process;
```

In hardware, this becomes



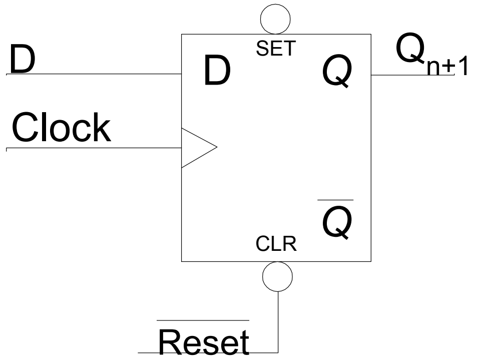
A clock’event is a 0 to 1 or 1 to 0 transition on the clock line.

Sequential Statements
Implied Registers

How does this produce a register?

1. If reset = 0, q is set to 0 (asynchronous reset)
2. If clock line makes a transition from 0 to 1
   - Clock'event and clock = 1
     then q is assigned to d

   But, we have not defined an output for
   1. Reset = 1,
   2. A non Clock'event , or
   3. Clock'Event and Clock = 0



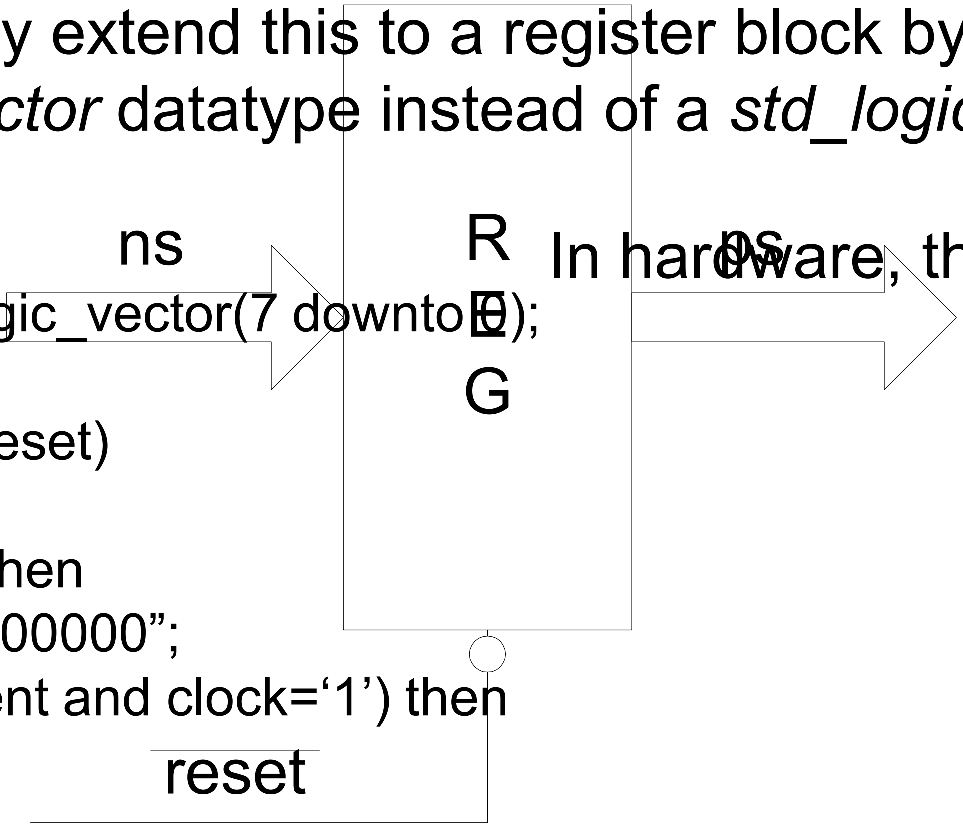
So, VHDL **assumes** we want to retain the current value of q for these conditions and synthesizes a D-FF for us.

Sequential Statements
Implied Registers

We can easily extend this to a register block by using a *std_logic_vector* datatype instead of a *std_logic* datatype.

In hardware, this becomes

```
…….
Signal ns,ps:std_logic_vector(7 downto 0);
……..
Process (ns,clock,reset)
    begin
        if (reset = ‘0’) then
              ps <= “00000000”;
        elsif( clock’event and clock=‘1’) then
               ps <= ns;
        end if;
     end process;
```



Sequential Statements Implied Registers

We can also define a S0 (reset state) and use it to reset the register.

```
…….
Signal ns,ps:std_logic_vector(7 downto 0);
Constant S0:std_logic_vector(7 downto 0) := “00000000”;
……..
Process (ns,clock,reset)
   begin
      if (reset = ‘0’) then
           ps <= s0;   --- use ‘reset’ state
      elsif( clock’event and clock=‘1’) then
           ps <= ns;
      end if;
    end process;
```

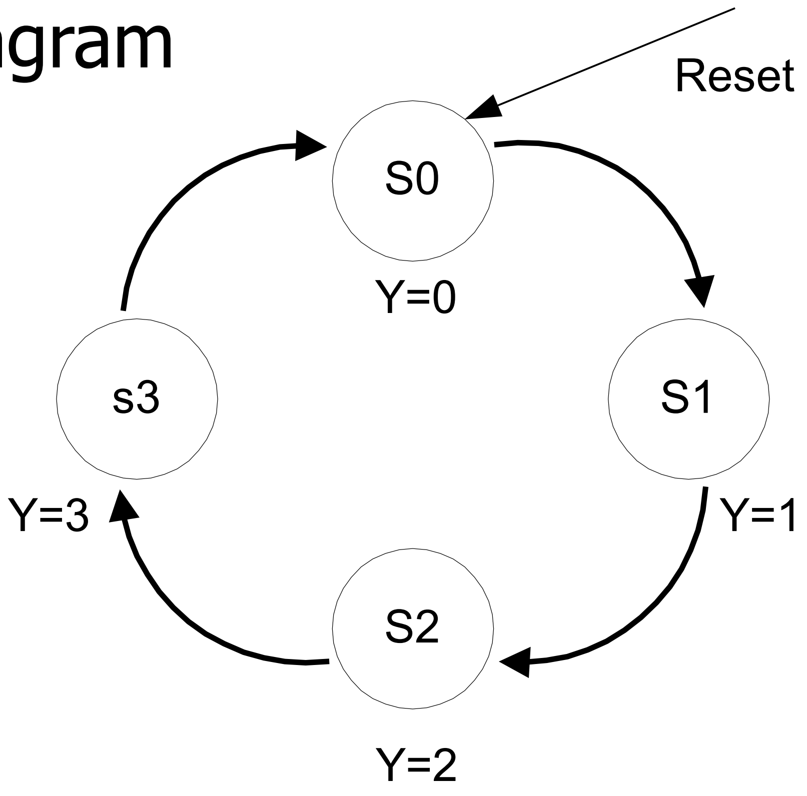
Sequential Statements
Case -When Statement

Use a *CASE-WHEN* statement when priority is not needed.
All FSMs will be implemented using Case-when statements.
Syntax is:

```
Case expression is
    when choice_1 =>
        sequential statements;
    when choice_2 =>
        sequential statements;
            ………….
    when choice_n =>
        sequential statements;
    when others => -- default condition
        sequential statements;
  end case;
```

VHDL FSM Example 1
2-bit Up Counter

- State Diagram



VHDL FSM Example 1

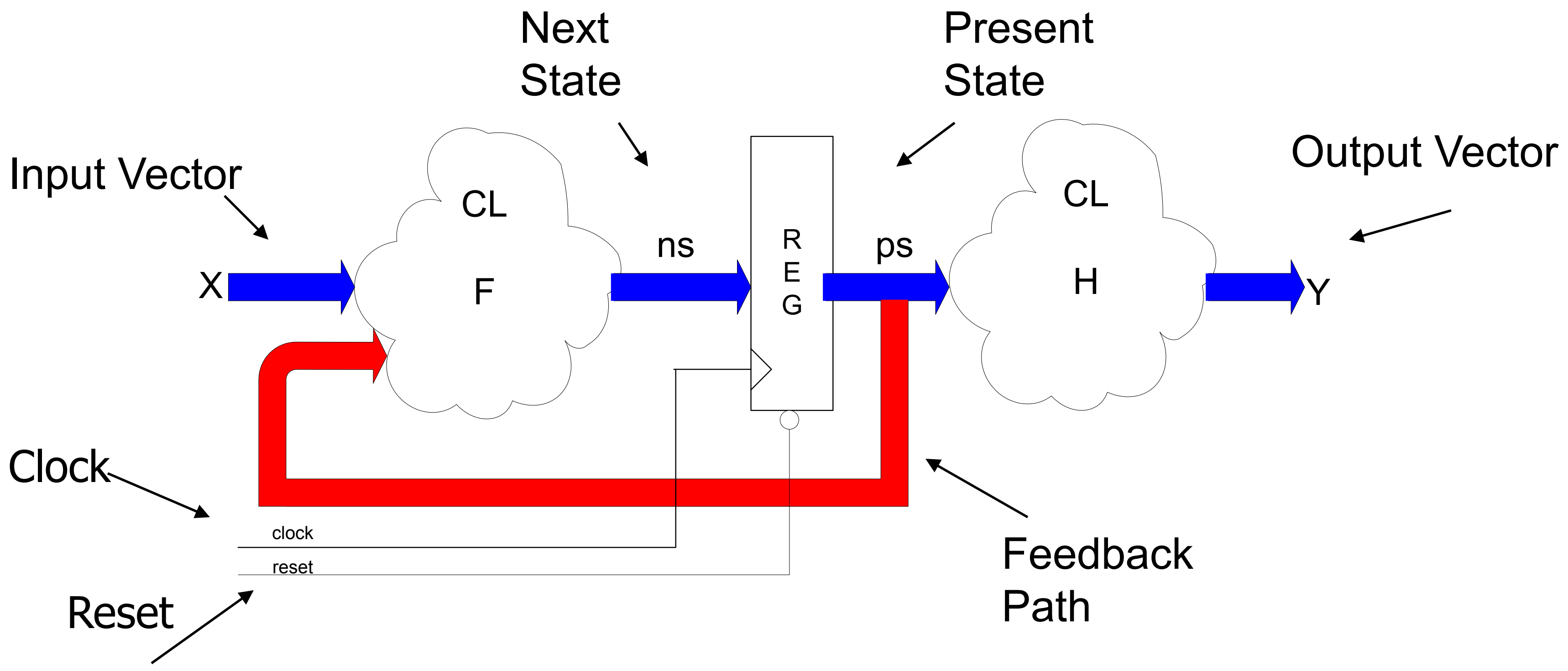
- State Table

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

Let S0 = reset state

Let S0 = 00
S1 = 01
S2 = 10
S3 = 11

Recall Moore FSM



Use a case statement to implement the design since priority is not needed

VHDL Code - Header Info

```
--------------------------------------------------------------
--
--   Program:  fsm1.vhd
--
--   Description:  2-bit up counter.
--
--   Author:  R.J. Perry
--   Date:
--   Revisions:
-------------------------------------------------------------
--   Signal I/O
--------------------------------------------------------------
--   Signal name        Direction       Description
--   clock,reset         in              clock,reset
--   count              out             output count
--------------------------------------------------------------
```

VHDL Code - Entity Declaration

```
--  Call Altera and IEEE packages
library altera;
use altera.maxplus2.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- define entity
entity fsm1 is
  port ( clk,reset: in std_logic;
         count: out std_logic_vector(1 downto 0)
        );
end entity fsm1;
```

VHDL Code - Architecture Dec

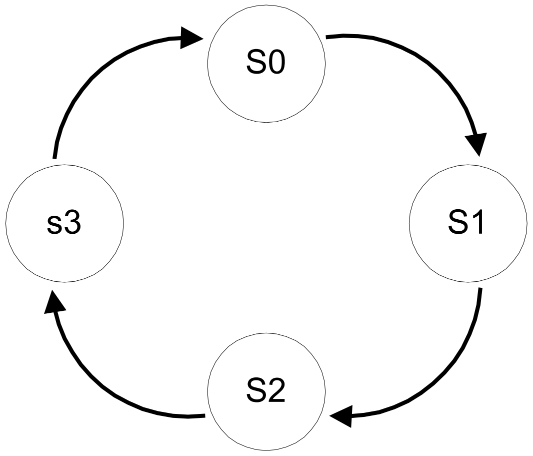
```
-- define architecture

architecture fsm of fsm1 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 -- F Logic

```
--
-- this process executes the F 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;
```

Input into F logic

State Diagram for F Logic

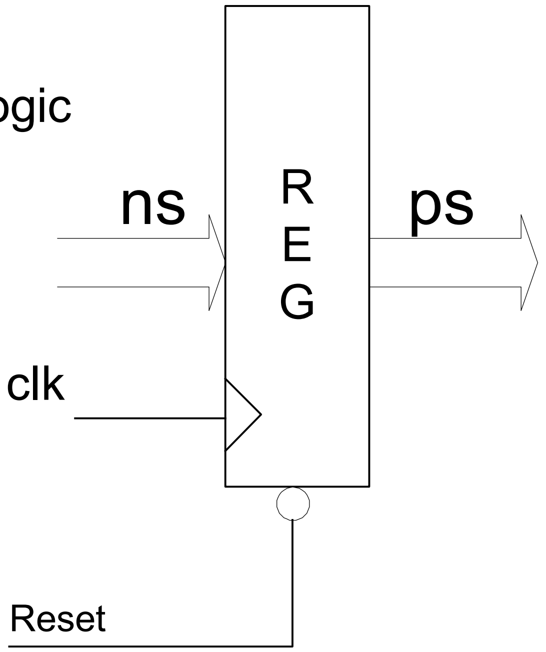


Note: we only need to "describe" the behavior
VHDL will "figure out" the functional relationships

VHDL Code -- Register Logic

```
--
-- This process includes the registers
   implicitly
--
 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;
```

Inputs to reg logic

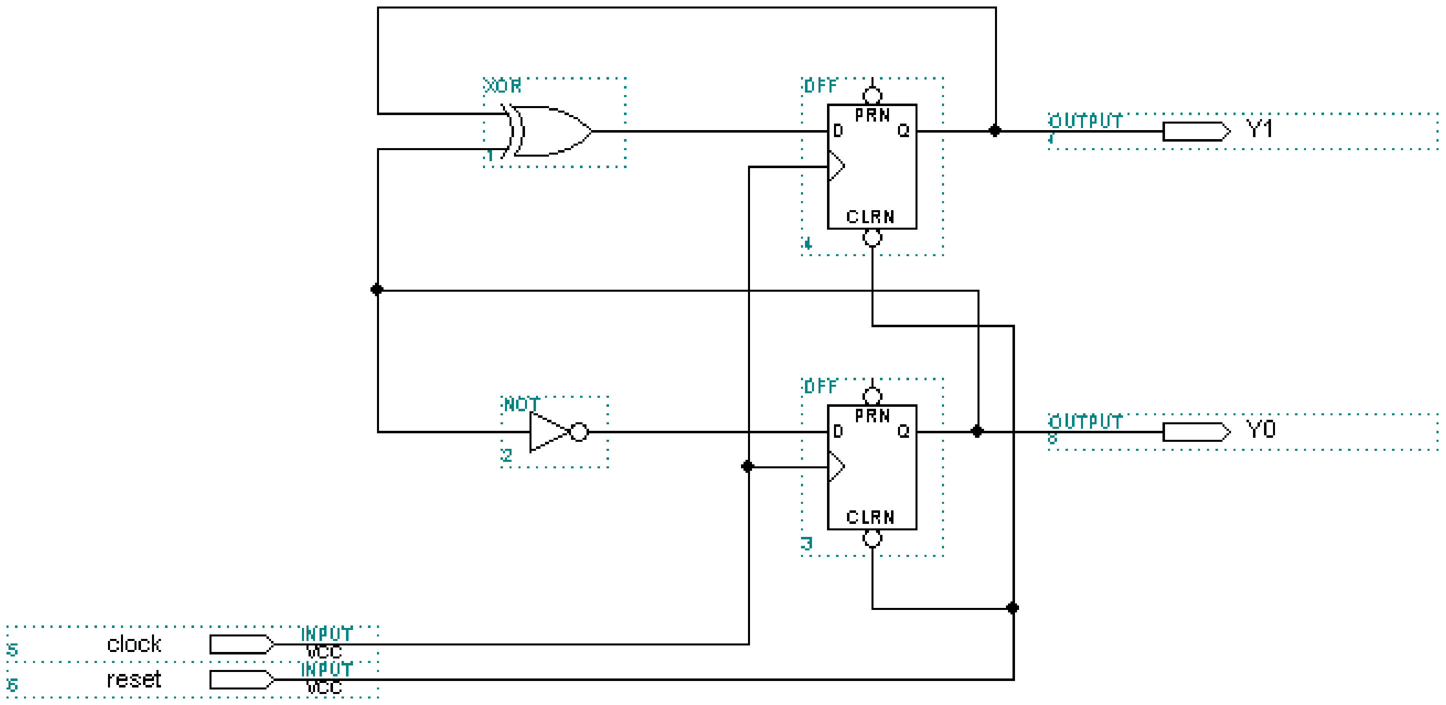


VHDL Code -- H Logic

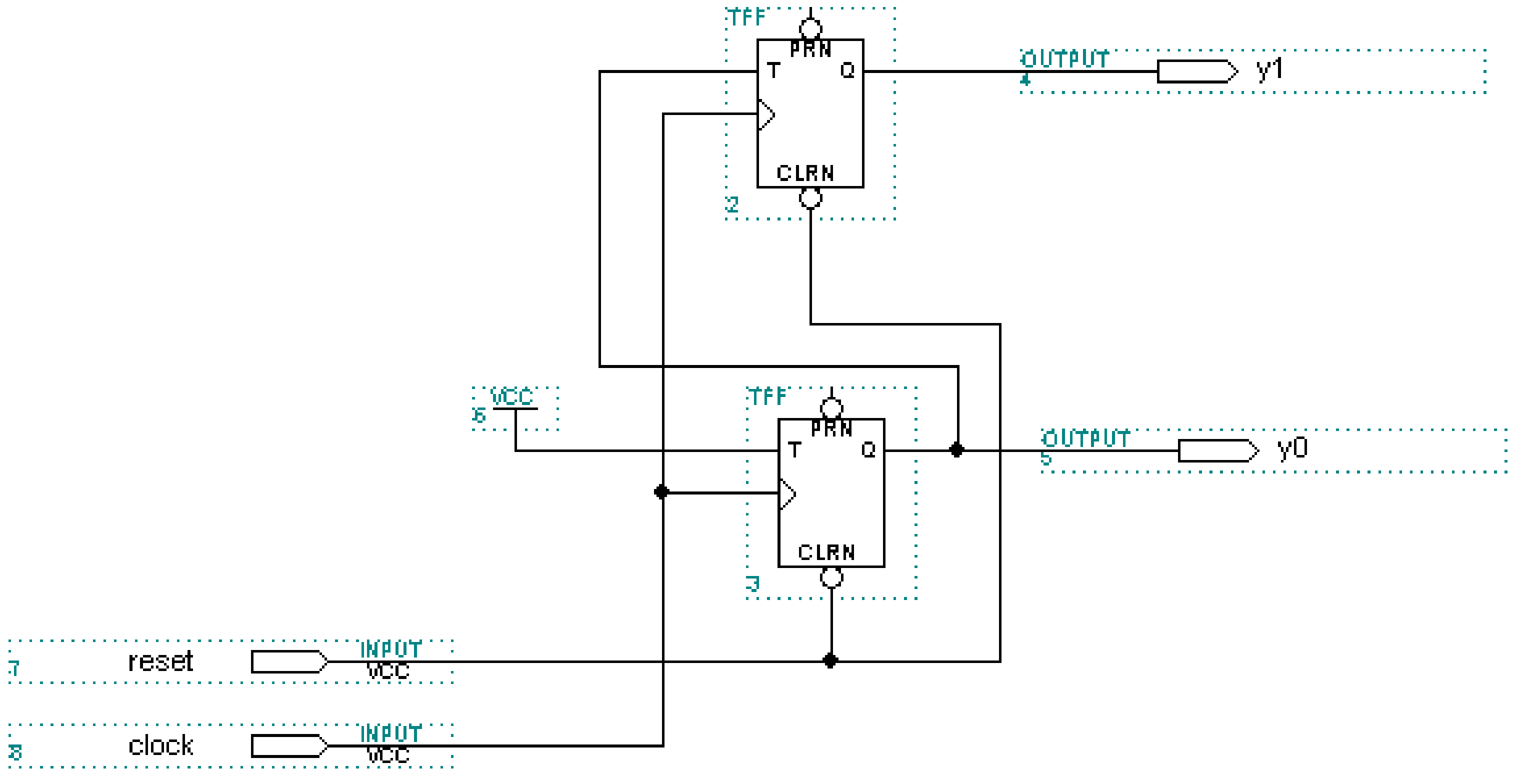
```
--
--  Use concurrent statement to implement H Logic
--

   count <=  ps;

 end architecture fsm;
```

$$\begin{tikzcd}\text{ps} \arrow[r, Rightarrow] & \text{count}\end{tikzcd}$$

Recall – Gate Level Logic Diagram



Maxplus II “Produces”



Is this correct?

T = Toggle FF

We have,

$$n_{s0} <= \overline{p_{s0}}$$

OK, same as before

$$n_{s1} <= \overline{p_{s1}} p_{s0} + \overline{p_{s0}} p_{s1} = p_{s1} \oplus p_{s0}$$

T input

OK, same as before

How does this code
fit into our Moore FSM architecture?