Module 3

# VHDL Syntax

# Basics

| Syntax | What does it do |
|---|---|
| -- | VHDL Comment |
| := | Assigns an initial value |
| => | Assigns object on right value of object on left |
| <= | Assigns object on left value of object on right |
| ; | Indicates end of line |
| : | Used to Specify a data type for an Object |
| '1' | Value of a std_logic |
| "100" | Value of a std_logic_vector (3 bits) |
| Rising_edge | Refers to the rising edge of a signal |

# Example

```
1  -- This is a comment in VHDL
2
3  -- Define signal "A" as type integer with
4  -- initial value of 4
5  signal A    : integer := 4);
6
7  -- Define signal "B" as type unsigned with
8  -- initial value of 0000
9  signal B    : unsigned(3 downto 0) := "0000";
10
11 -- Define signal "B1" as type unsigned with
12 -- initial value of 0000
13 signal B1   : unsigned(3 downto 0) := (others => '0');
14
15 -------------------------------------------------------
16 -- Note
17 -- Signals B and B1 are the same!
18 -------------------------------------------------------
19
20 -- Define signal "C" as type unsigned with
21 -- initial value of 1111
22 signal C    : unsigned(3 downto 0) := "1111";
23
24 -- Define signal "C1" as type unsigned with
25 -- initial value of 1111
26 signal C1   : unsigned(3 downto 0) := (others => '1');
27
28 -------------------------------------------------------
29 -- Note
30 -- Signals C and C1 are the same!
31 -- Do you understand how the others statement works?
32 -------------------------------------------------------
```

# Example

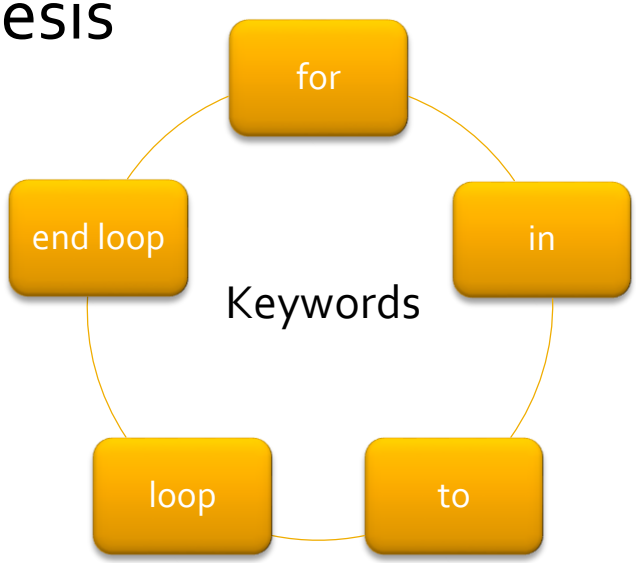
```
1   -- Assigning values to std_logic_vectors
2
3   -- Define signal A as a std_logic_vector data type
4   -- with an initial value of 11111111
5   signal A         : std_logic_vector(7 downto 0) := (others => '1');
6
7   -- Define signal A_reg as a std_logic_vector data type
8   -- with an initial value of 00000000
9   signal A_reg     : std_logic_vector(7 downto 0) := (others => '0');
10
11  ------------------------------------------------------------------
12  -- Note:
13  -- Right now signal A = 11111111 and A_reg = 00000000
14  ------------------------------------------------------------------
15
16  -- Set signal A equal to signal A_reg
17  A <= A_reg;
18  ------------------------------------------------------------------
19  -- Note:
20  -- Right now signal A = 00000000 and A_reg = 00000000
21  ------------------------------------------------------------------
```

# Common Keywords

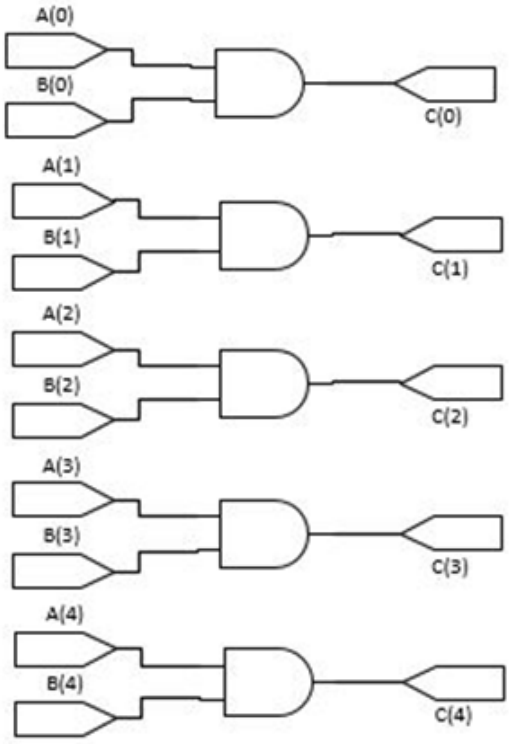
| Word | Definition |
| --- | --- |
| Entity | a primary design unit |
| Architecture | A secondary design unit |
| Port | interface definition, also port map |
| Generic | introduces generic part of a declaration |
| Signal | declaration that an object is a signal |
| Constant | declares an identifier to be read only |
| Begin | start of a begin end pair |
| End | part of many statements, may be followed by word and id |
| Others | Assigns all unpicked bits in a std_logic_vector, unsigned, or signed data type |

# For Loop Statement

- Used to generate multiple instances of same logic
- For loops go away after synthesis

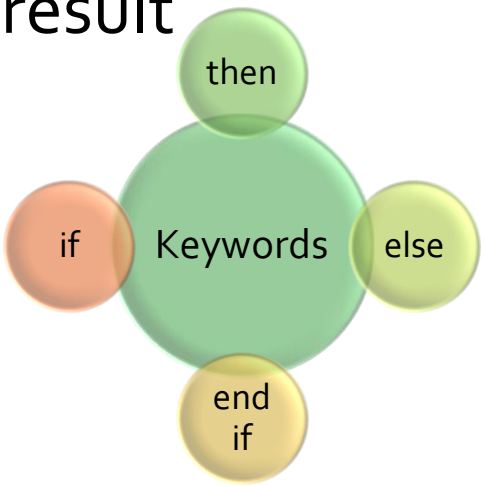



```
-- VHDL for loop example
for a in 0 to 4 loop

    C(i) <= A(i) and B(i);

end loop;
```



# If Statement

- Conditional statement must evaluate to a True or False result




- The else statement is not necessary, if unused replace the else statement with end if

Single "If" "Else" Statement

```
if(enable ='1') then
    result <= A;
else
    result <= (others => '0');
end if;
```

INVALID!

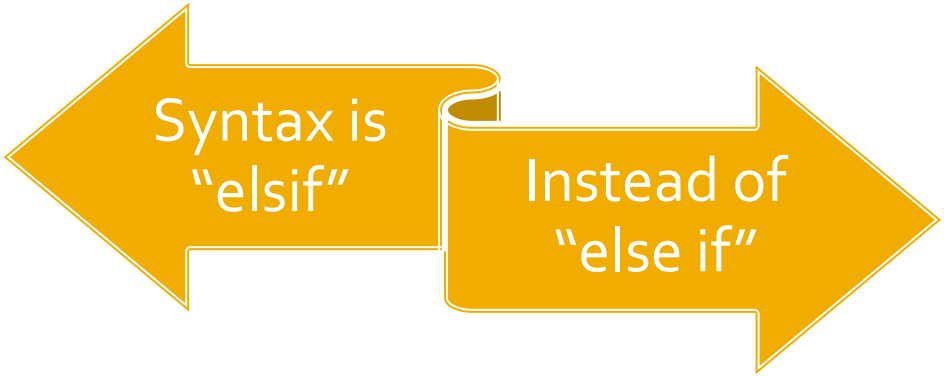
```
if(enable + check) then
    result <= A;
end if;
```

VALID

```
if(enable = '0') then
    result <= A;
end if;
```

# If Statement

- VHDL supports multiple If Else statements.
- Note:




Multiple "If" "Else" Statement

```
if(a_in(0) = '1') then
    encode <= "000";
elsif(a_in(1) = '1') then
    encode <= "001";
elsif(a_in(2) = '1') then
    encode <= "010";
elsif(a_in(3) = '1') then
    encode <= "011";
elsif(a_in(4) = '1') then
    encode <= "100";
elsif(a_in(5) = '1') then
    encode <= "101";
elsif(a_in(6) = '1') then
    encode <= "110";
elsif(a_in(7) = '1') then
    encode <= "111";
else
    encode <= (others => 'X');
end if;
```

# Case Statement

- Checks an inputs against multiple "cases"
- Keywords: case, when, end case
- Note: the direction of the => and <=

```
case inputs is
    when "000" =>
        outputs <= "00";
    when "001" =>
        outputs <= "01";
    when "010" =>
        outputs <= "01";
    when "011" =>
        outputs <= "10";
    when "100" =>
        outputs <= "01";
    when "101" =>
        outputs <= "10";
    when "110" =>
        outputs <= "10";
    when "111" =>
        outputs <= "11";
    when others =>
        outputs <= (others => 'X');
end case;
```

# Signal Assignments

- Signal assignments are used to assign a specific value to a signal
- Signals can be assigned set values such as '1' or 'o' or they can be set to values of other signals
- Signal assignments are always happening. These are **not** sequential operations

```
nxt_state <= state;
shift <= '0';
add <= '0';
load_data <= '0';
data_ready <= '0';
```

# VHDL Process

- A Process is evaluated when a signal in the sensitivity list has changed state
- A VHDL process contains several parts


Name

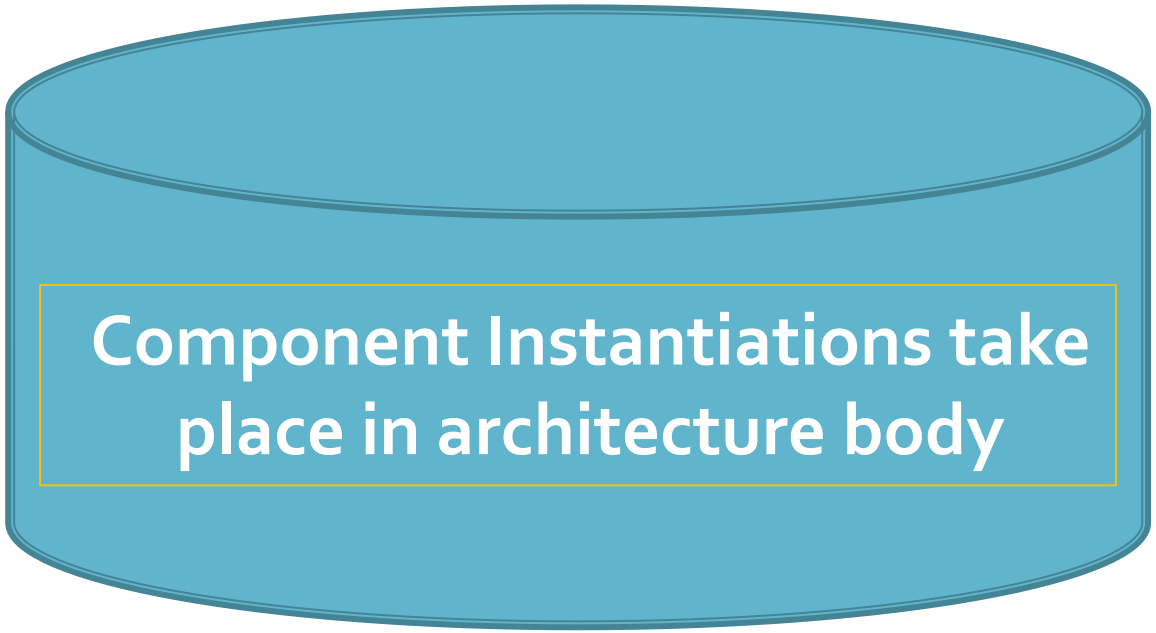
Sensitivity List

Begin

```
state_proc: process(clk)
    begin
    if rising_edge(clk) then
        if(reset = '0') then
            state <= init;
        else
            state <= nxt_state;
        end if;
    end if;
end process state_proc;
```

End Process

# Component Instantiation



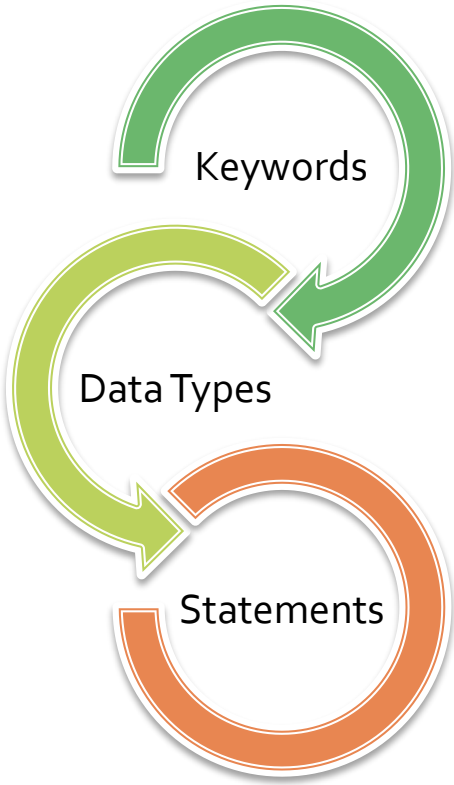

```
component Hex_to_7_Seg is
port (
     seven_seg    : out std_logic_vector(6 downto 0);
     hex          : in  std_logic_vector(3 downto 0));
end component;
```

# Example


```
1   -- USR
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.numeric_std.all;
5
6   entity USR is
7   generic (
8       data_width  : integer := 8);
9   port (
10      A           : out std_logic_vector(data_width - 1 downto 0);
11      I           : in std_logic_vector(data_width - 1 downto 0);
12      S           : in std_logic_vector(2 downto 0);
13      reset       : in std_logic;
14      clk         : in std_logic);
15  end USR;
16
17  architecture behavior of USR is
18
19  signal A_reg    : std_logic_vector(data_width - 1 downto 0);
20
21  begin
22
23      A <= A_reg;
24
```

# Example

```
25      USR_proc: process(clk)
26      begin
27          if(rising_edge(clk)) then
28              if(reset = '0') then
29                  A_reg <= (others => '0');
30              else
31                  case S is
32                      when "00" => -- Hold
33                          A_reg <= A_reg;
34
35                      when "01" => -- Right shift
36                          A_reg(data_width - 1) <= '0';
37                          A_reg(data_width - 2 downto 0) <= A_reg(data_width - 1 downto 1);
38
39                      when "10" => -- Left shift
40                          A_reg(data_width - 1 downto 1) <= A_reg(data_width - 2 downto 0);
41                          A_reg(0) <= '0';
42
43                      when "11" => -- Parallel Load
44                          A_reg <= I;
45
46                      when others => -- Error code
47                          A_reg <= (others => 'X');
48
49                  end case;
50              end if;
51          end if;
52      end process USR_proc;
53  end behavior;
```



# Summary

As with any language VHDL has specific ways of assigning values, creating comments, etc…

VHDL supports If, and case statements as well as for loops

Everything that is not inside a process is always happening (**VHDL is NOT sequential**)

Component Instantiations are references to other VHDL designs