DHD T2 Cheat Sheet

VHDL: Very High Speed Integrated Circuit Hardware Descriptive Language

✅ VHDL Code Structures for All Three Modeling Styles

🛠️ 1. Behavioral Modeling (Sequential)

- **Used for describing the behavior of the circuit.**
- Uses **process blocks** and sequential statements.

💡 Basic Structure:

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

entity Behavioral_Model is
    port (
        clk     : in std_logic;           -- Clock input
        reset   : in std_logic;           -- Reset input
        d       : in std_logic;           -- Input
        q       : out std_logic           -- Output
    );
end entity;

architecture Behavioral of Behavioral_Model is
begin
    process (clk, reset)   -- Sequential block
    begin
        if reset = '1' then
            q <= '0';       -- Reset the output
        elsif rising_edge(clk) then
            q <= d;         -- Output follows input on clock edge
        end if;
    end process;
end Behavioral;
```

🔥 Key Points:

- `process` **block** → Sequential execution.
- **Signal assignments (`<=`)** take effect after the process suspends.
- Used for **registers, counters, state machines**, etc.

🛠️ 2. Dataflow Modeling (Concurrent)

- **Describes how data flows** between inputs and outputs.
- Uses **concurrent assignment statements** (no process block).

💡 Basic Structure:

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

entity Dataflow_Model is
    port (
        a, b   : in std_logic;         -- Inputs
        c      : out std_logic          -- Output
    );
end entity;

architecture Dataflow of Dataflow_Model is
begin
    -- Concurrent signal assignment
    c <= a AND b;  -- Output c is always equal to a AND b
end Dataflow;
```

🔥 Key Points:

- **Concurrent execution** → Statements are executed **in parallel**.

- Used for **combinational logic** like AND, OR gates, multiplexers, etc.
- **No process block** → directly assign expressions.

🛠️ 3. Structural Modeling (Component-based)

- Describes the **circuit's structure** by interconnecting different components.
- Uses **components and port mapping**.

💡 **Basic Structure:**

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

entity Structural_Model is
    port (
        a, b   : in std_logic;           -- Inputs
        c      : out std_logic           -- Output
    );
end entity;

architecture Structural of Structural_Model is

    -- Declare components
    component AND_Gate
        port (
            x, y  : in std_logic;
            z     : out std_logic
        );
    end component;

begin
    -- Instantiate the AND_Gate component
    U1: AND_Gate
        port map (
            x => a,
            y => b,
            z => c
        );
end Structural;
```

🔥 Key Points:

- **Component declaration** → Define the external entity.
- **Instantiation** → Connect inputs and outputs using `port map`.
- Used for **larger designs**, combining smaller components.

🚦 Key Differences Between the Modeling Styles

Style	Execution Type	Use Case	Characteristics
Behavioral	Sequential (inside process)	Describes **behavior**	Uses `process` block
Dataflow	Concurrent	Describes **data flow**	No process block, concurrent
Structural	Component-based	Describes **structure**	Uses components and mapping

---

Operators and Basic Syntax of Important Things:

1. A: in std_logic_vector (3 downto 0): MSB at 3 and LSB at 0 means 4 bit **(done)**
2. Concatenation: “&”
3. Case-when (below)
4. When-else (below)
5. If-then-else (below)
6. Signal and Variable declaration **(done)** (signal <=, variable :)
7. Clock declaration and uses (in the doc, have to learn tomorrow)
8. Component declaration (is similar to that of an entity) **(done)**
9. Component instantiation **(done)** (U0 : <component name> port map (<compoent variable> => top level entity variable)

✅ Types of Commands in VHDL

🟢 Sequential Commands

1. **If-Then-Else Command**
    - Executes statements sequentially based on a condition.
    - Written inside the `process` block.

Syntax:

```
if (condition) then

    -- statements

elsif (another_condition) then

    -- statements

else

    -- statements

end if;
```

Example (2x1 Mux):

```
if (Sel = '0') then

    F <= I0;

else

    F <= I1;

end if;
```

---

2. **Case-When Command**
    - Executes different statements based on multiple conditions.
    - Used when there are multiple possible values for a variable.
    - Written inside the `process` block.

Syntax:

```
case expression is

    when condition1 =>

        -- statements

    when condition2 =>

        -- statements

    when others =>

        -- statements

end case;
```

Example (4x1 Mux):

```
case Sel is

    when "00" => F <= I0;

    when "01" => F <= I1;

    when "10" => F <= I2;

    when "11" => F <= I3;

    when others => F <= '0';

end case;
```

🔵 Concurrent Commands

1. **When-Else Command**
   - Used to execute operations concurrently based on conditions.
   - Written **outside** the `process` block.

Syntax:

```
target_signal <= expression when condition else
                expression when condition else
                expression;
```

Example:

```
F <= '1' when A = "11" else
     '0' when A = "00" else
     '0' when A = "10" else
     '0';
```

---

2. **With-Select Command**
   - Selects one of several values based on the value of an expression.
   - Written **outside** the `process` block.

Syntax:

```
with expression select
    target_signal <=
    value1 when condition1,

    value2 when condition2,

    value3 when condition3,

    value4 when condition4;
```

Example:

```
signal selector : std_logic_vector(1 downto 0);

signal output   : std_logic_vector(3 downto 0);


with selector select

    output <=

    "0001" when "00",

    "0010" when "01",

    "0100" when "10",

    "1000" when "11";
```

6. Clock Descriptions

- **Clock transitions trigger the process.**
- **Four methods for describing clocks:**
    1. `if (clock'event and clock='0')` **→ Trigger on falling edge.**
    2. `if (clock='1')` **→ Trigger on high clock.**
    3. `wait until clock='1';` **→ Wait for rising edge.**
    4. `if (clock'event and clock='0' and clock'last_value='1')` **→ Trigger on falling edge after rising.**

Sequential vs. Concurrent Statements:

1. **Sequential Statements:**
    - **Executed line by line.**
    - **Written inside a process block.**
    - **Use** `if-then-else` **and** `case` **commands.**
2. **Concurrent Statements:**
    - **Executed simultaneously.**
    - **Written outside the process block.**
    - **Use** `when-else` **and** `with-select` **commands.**

Examples of Coding (w/ Testbench although not in syllabus) (4-1 Mux):

1. **Behavioral:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_4_1 is
   port (
      S1, S0 : in std_logic;     -- Select lines
      D0, D1, D2, D3 : in std_logic; -- Inputs
      Y : out std_logic          -- Output
   );
end MUX_4_1;

architecture Behavioral of MUX_4_1 is
begin
   process(S1, S0, D0, D1, D2, D3)
   begin
      case (S1 & S0) is
         when "00" => Y <= D0;
         when "01" => Y <= D1;
         when "10" => Y <= D2;
         when "11" => Y <= D3;
         when others => Y <= '0';
      end case;
   end process;
end Behavioral;
```

2. **Dataflow:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_4_1 is
   port (
      S1, S0 : in std_logic;
      D0, D1, D2, D3 : in std_logic;
      Y : out std_logic
   );
end MUX_4_1;

architecture Dataflow of MUX_4_1 is
begin
   Y <= D0 when (S1 = '0' and S0 = '0') else
```

```
        D1 when (S1 = '0' and S0 = '1') else
        D2 when (S1 = '1' and S0 = '0') else
        D3 when (S1 = '1' and S0 = '1') else
        '0';
end Dataflow;
```

3. Structural:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_4_1 is
   port (
      S1, S0 : in std_logic;
      D0, D1, D2, D3 : in std_logic;
      Y : out std_logic
   );
end MUX_4_1;

architecture Structural of MUX_4_1 is

   component AND2
      port (A, B : in std_logic; Y : out std_logic);
   end component;

   component OR4
      port (A, B, C, D : in std_logic; Y : out std_logic);
   end component;

   signal A, B, C, D : std_logic;

begin
   -- AND gates for selection
   U1: AND2 port map(S1, S0, A);
   U2: AND2 port map(S1, not S0, B);
   U3: AND2 port map(not S1, S0, C);
   U4: AND2 port map(not S1, not S0, D);

   -- OR gate to combine outputs
   U5: OR4 port map(A, B, C, D, Y);
end Structural;
```

4. OR Gate with Testbench:

Design:

```
Library ieee;
Use library ieee.std_logic_1164.all;

Entity or_gate is
Port ( A,B : in STD_logic; Y : out STD_LOGIC)
End entity;

Architecture behavioral or_gate is
Begin
Y <= A or B;
End behavioral;
```

Testbench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_or_gate is
end tb_or_gate;

architecture Behavioral of tb_or_gate is
    -- Component Declaration
    component or_gate
        Port (
            A : in STD_LOGIC;
            B : in STD_LOGIC;
            Y : out STD_LOGIC
        );
    end component;

    -- Signals for the testbench
    signal tb_A : STD_LOGIC := '0';    -- Use tb_A instead of A
    signal tb_B : STD_LOGIC := '0';    -- Use tb_B instead of B
    signal tb_Y : STD_LOGIC;           -- Output signal

begin
    -- Port Mapping
    uut : or_gate
        port map (
            A => tb_A,        -- Map tb_A to A of OR gate
            B => tb_B,        -- Map tb_B to B of OR gate
            Y => tb_Y         -- Map tb_Y to Y of OR gate
        );
    -- Process to apply test cases
    process
    begin
        -- Test Case 1: 0 OR 0 => 0
        tb_A <= '0'; tb_B <= '0';
        wait for 10 ns;

        -- Test Case 2: 0 OR 1 => 1
        tb_A <= '0'; tb_B <= '1';
        wait for 10 ns;

        -- Test Case 3: 1 OR 0 => 1
        tb_A <= '1'; tb_B <= '0';
        wait for 10 ns;

        -- Test Case 4: 1 OR 1 => 1
        tb_A <= '1'; tb_B <= '1';
        wait for 10 ns;

        wait;  -- Keeps the simulation running
    end process;
end Behavioral;
```