VHDL Cheatsheet

A quick reference guide to VHDL (VHSIC Hardware Description Language), covering syntax, data types, operators, and common constructs used in digital circuit design.

Basic Syntax and Structure

Entity Declaration

The `entity` declaration defines the interface of a design.

```
entity entity_name is
  port (
    port_name : mode data_type;
    ...
  );
end entity entity_name;
```

Example:

```
entity AND2 is
  port (
    A : in  std_logic;
    B : in  std_logic;
    Y : out std_logic
  );
end entity AND2;
```

Architecture Body

The `architecture` body implements the behavior of the entity.

```
architecture architecture_name of entity_name
is
  -- Declarations (signals, components, etc.)
begin
  -- Concurrent statements
end architecture architecture_name;
```

Example:

```
architecture Behavioral of AND2 is
begin
  Y <= A and B;
end architecture Behavioral;
```

Libraries and Packages

VHDL uses libraries and packages for predefined types and functions.

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

- `ieee.std_logic_1164`: Standard logic types (std_logic, std_logic_vector).
- `ieee.numeric_std`: Arithmetic operations on std_logic_vector.

Data Types and Operators

Standard Data Types

`std_logic`	Represents a single bit with nine possible values ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-').
`std_logic_vector`	An array of `std_logic` elements. `std_logic_vector(7 downto 0)`.
`integer`	Represents signed integer values. Range is implementation-dependent.
`boolean`	Represents boolean values (`TRUE` or `FALSE`).
`real`	Represents floating-point values.
`time`	Represents time values with a specified unit (e.g., `10 ns`).

Operators

Logical Operators	`and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not`
Relational Operators	`=`, `/=`, `<`, `<=`, `>`, `>=`
Arithmetic Operators	`+`, `-`, `*`, `/`, `mod`, `rem`, `abs`, `**`
Shift Operators	`sll`, `srl`, `sla`, `sra`, `rol`, `ror`
Concatenation Operator	`&` (e.g., `A & B`)

Concurrent and Sequential Statements

Concurrent Statements

Concurrent statements execute in parallel.

- **Signal Assignment:**

```
signal_name <= expression;
```

- **Conditional Signal Assignment:**

```
signal_name <= expression1 when condition else expression2;
```

- **Selected Signal Assignment:**

```
with selector select
  signal_name <= expression1 when choice1,
                 expression2 when choice2,
                 ...
                 expressionN when others;
```

- **Process:**

```
process (sensitivity_list)
begin
  -- Sequential statements
end process;
```

Sequential Statements

Sequential statements execute in order within a process.

- **if-then-else:**

```
if condition then
  -- statements
elsif condition then
  -- statements
else
  -- statements
end if;
```

- **case:**

```
case expression is
  when choice1 =>
    -- statements
  when choice2 =>
    -- statements
  when others =>
    -- statements
end case;
```

- **for loop:**

```
for loop_variable in range loop
  -- statements
end loop;
```

- **while loop:**

```
while condition loop
  -- statements
end loop;
```

- **variable assignment:**

```
variable_name := expression;
```

Advanced Concepts

Components

Components are instances of entities used within an architecture.

- **Component Declaration:**

```
component component_name is
  port (
    port_name : mode data_type;
    ...
  );
end component;
```

- **Component Instantiation:**

```
instance_name : component_name
  port map (
    port_name => signal_name,
    ...
  );
```

Functions and Procedures

Function	A function returns a value and cannot have side effects.
	`function function_name (parameter_list) return return_type is`
	`  -- Declarations`
	`begin`
	`  -- Statements`
	`  return return_value;`
	`end function;`

Procedure	A procedure does not return a value directly and can have side effects.
	`procedure procedure_name (parameter_list) is`
	`  -- Declarations`
	`begin`
	`  -- Statements`
	`end procedure;`

Generics

Generics provide a way to parameterize entities and components.

```
entity entity_name is
  generic (
    generic_name : data_type := default_value;
    ...
  );
  port (
    ...
  );
end entity;
```

Example:

```
entity Adder is
  generic (
    WIDTH : integer := 8
  );
  port (
    A : in  std_logic_vector(WIDTH-1 downto
0);
    B : in  std_logic_vector(WIDTH-1 downto
0);
    Y : out std_logic_vector(WIDTH-1 downto 0)
  );
end entity Adder;
```