ES 4 VHDL reference sheet

r.2021.10.28

www.ece.tufts.edu/es/4

GRAY_ITALICS represent user-defined names or operations
Purple constructs are only available in VHDL 2008.

keywords
literals (constants)

```
-- This is a comment
/* Multi-line comment
   (VHDL 2008 only) */
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

You almost always need these libraries; just put this at the top of every file.

```
entity ENTITY_NAME is
  port(
    PORT_NAME : in std_logic; -- Single bit input
    ANOTHER   : out std_logic_vector(3 downto 0) -- 4-bit output
      );
end;
```

No semicolon on the last one!

Don't forget these semicolons!

```
architecture ARCH_NAME of ENTITY_NAME is
  -- Component declarations, if using submodules
  component SUB_ENTITY is
    port(
      -- Port list for the entity you're including
    );
  end component;

  -- Signal declarations, if using intermediate signals
  signal NAME : TYPE;
begin
  -- Architecture definition goes here
end;
```

Just replace `entity` with `component` and put `end component` at the end.

Instantiate a submodule

```
INSTANCE_NAME : MODULE_NAME
    generic map (
        GENERIC => CONSTANT,
    )
    port map(
        PORT => VALUE,
        ANOTHER => LOCAL_SIGNAL
    );
```

Continuous assignments

`RESULT_SIGNAL <= SIGNAL1 and SIGNAL2;`	Also works for or, not, nand, nor, xor
`RESULT_SIGNAL <= '1' when (SIGNAL1 = x"5") else '0';`	Note '=' for comparison (not '==')
`HIGHEST_BIT <= EIGHT_BIT_VEC(7);`	Extract a single bit (7 is MSB, 0 is LSB)
`TWO_BIT_VEC <= EIGHT_BIT_VEC(3 downto 2);`	Extract multiple bits
`SIX_BIT_VEC <= "000" & EIGHT_BIT_VEC(3 downto 2) & SINGLE_BYTE;`	Concatenate bits and vectors

Types

`std_logic`	Basic logic type, can take values 0, 1, X, Z (and others)
`std_logic_vector (n downto m)`	Ordered group of std_logic

`unsigned (n downto m)`
`signed (n downto m)`
	Like std_logic_vector, but have mathematical operations defined

`integer`	Poor for synthesis, but constants are integers by default

Literals

`'0', '1', 'X', 'Z'`

`"00001010", x"0c"` 8-bit binary, hex

`9x"101"`	`3b"101"`	`7d"101"`
9-bit hex	3-bit binary	7-bit decimal

`5, 38, 10000000`

Type conversion

`to_unsigned(INTEGER, WIDTH)`	Use `to_unsigned` for unsigned constants before VHDL 2008.
`unsigned(LOGIC_VECTOR)`	(Same things for signed)
`std_logic_vector(UNSIGNED)`	

Process blocks

```
process (SENSITIVITY) is
begin
  -- if/case/print go here
end process;
```

`wait;` Halt process forever

`wait for TIME ns;` Let simulation time pass
Units include ms, us, ns, ps

If sensitivity includes:

`all`↕ → Combinational logic

`clk`↑ → Flip-flop / register

`clk`↑ + `data`↕ → Latch

Nothing → Testbench (repeated evaluation)

Something else → Bad things you probably didn't want

Specify all signals by name prior to VHDL 2008

Reporting stuff

`assert CONDITION report "MESSAGE" severity error;` Print message if condition is false

`report "MESSAGE" severity error;` Severity can be NOTE, WARNING, ERROR, FATAL
"FATAL" ends the simulation

`report "A is " & to_string(a);` Use `image` function prior to VHDL 2008

`report "A in hex is " & to_hstring(a);`

concatenation

conversion to string

Writing to files (or stdout)

`variable BUF : line;` Declare buffer in process block

`write(BUF, "MESSAGE");` Append message to buffer

`writeline(output, BUF);` Write buffer to stdout (like `report`, but just the text)

`file RESULTS : text;` Declare file handle in process block

```
file_open(RESULTS, "FILENAME", WRITE_MODE);
writeline(RESULTS, BUF);
```

If/else

```
if CONDITION then
  SIGNAL <= VALUE1;
elsif OTHER_CONDITION then
  SIGNAL <= VALUE2;
else
  SIGNAL <= VALUE3;
end if;
```

Note spelling of "elsif"!

Sequential logic

```
process (CLOCK) is
begin
  if rising_edge(CLOCK) then
    -- Clocked assignments go here
  end if;
end process;
```

Case

```
case INPUT_SIGNAL is
  when VALUE1 => OPERATION1;
  when VALUE2 => OPERATION2;
  when others => DEFAULT;
end case;
```

For loop

```
for INDEXVAR in MIN to MAX loop
  -- loop body here
end loop;
```

INDEXVAR will be an integer

To count down:

```
for INDEXVAR in MAX downto MIN loop
```

Enumerated types

```
type TYPENAME is (VAL1, VAL2, VAL3);
```

`signal NAME : TYPENAME;` Just like any other type