

ECE-238 Logic Design – Lab

- Words written using lower-case letters and possibly one or many hyphens, are used to denote a syntactical category, for example: entity-declaration. Reserved words are written with bold characters, for example: **entity**, **constant**.
- Every replacement rule contains a left hand side expression and a right hand side expression separated by the sign →, which means "looks as" or "may be replaced with".
- |, a vertical line (the *pipe* sign) is used to separate many mutually exclusive alternatives. [], square brackets surround optional things that may occur once or not at all. {}, braces surround optional things that may occur once, many times or not at all. (), parenthesis are used to clarify how and in which order a rule is evaluated. Reserved words and characters surrounded by apostrophes, ' ', are included "as is" in the source code.

Syntax:	Examples:
constant-declaration → constant identifier { ',' identifier } ':' subtype-indication [':'= expression] ':'	CONSTANT CLK_PERIOD : time:= 20 ns; CONSTANT pi : REAL := 3.1415;
signal-declaration → signal identifier { ',' identifier } ':' subtype-indication [register bus] [':'= expression] ':'	SIGNAL a : STD_LOGIC; SIGNAL data: STD_LOGIC_VECTOR(4 downto 0);
entity-declaration → entity identifier is { subprogram-decl. type-decl. constant-decl. signal-decl. } [begin { concurrent-assertion-statement <i>passive-procedure-call</i> <i>passive-process-statement</i> }] end [entity] [entity-name-identifier] ':'	ENTITY black_box IS PORT (a : in std_logic_vector(4 downto 0); b : in std_logic_vector(4 downto 0); add_sub : in std_logic; clk : in std_logic; output : out std_logic_vector(4 downto 0)); BEGIN PeriodCheck(clk); -- Passive procedure call END black_box;
architecture-body → architecture identifier of <i>entity-name</i> is { subprogram-declaration subprogram-body type-declaration subtype-declaration constant-declaration signal-declaration component-declaration } begin { concurrent-statement } end [architecture] [architecture-name-identifier] ':'	ARCHITECTURE Structural OF black_box IS COMPONENT twos_comp -- Comp. declaration PORT (a: in std_logic_vector(4 downto 0); bypass: in std_logic; clk: in std_logic; q: out std_logic_vector(5 downto 0)); END COMPONENT ; SIGNAL sig1: std_logic_vector(5 downto 0); begin VHDL code ... end ARCHITECTURE Structural;
wait-statement → [label ':'] wait [on sensitivity-list] [until <i>boolean-expression</i>] [for <i>time-expression</i>] ':'	WAIT FOR CLK_PERIOD*8; WAIT ON s UNTIL s = '1'; -- Wait for rising edge on s
if-statement → [if-label ':'] if <i>boolean-expression</i> then { sequential-statement } { elsif <i>boolean-expression</i> then { sequential-statement } } [else { sequential-statement }] end if [if-label] ':'	PROCESS (reset,clk) BEGIN IF reset = '1' THEN state<=A; ELSIF clk'EVENT AND clk = '1' THEN state<= next_state; END IF ; END PROCESS ;
case-statement → [case-label ':'] case expression is when choices '='>' { sequential-statement } { when choices '='>' { sequential-statement } } end case [case-label] ':' choices → choice { choice }	PROCESS (s,d0,d1,d2,d3) BEGIN CASE s IS WHEN "00" => Y<=d0; WHEN "01" => Y<=d1; WHEN "10" => Y<=d2; WHEN "11" => Y<=d3; END CASE ; END PROCESS ;
process-statement → [process-label ':'] process ['(' sensitivity-list ') [is] { subprogram-decl. subprogram-body type-decl. subtype-decl. constant-decl. variable-decl. } begin { sequential-statement } end process [process-label] ':'	ARCHITECTURE Behave OF Design IS BEGIN FlipFlop: PROCESS (reset,clk) BEGIN IF reset = '1' THEN q <= '0'; ELSIF clk'EVENT AND clk = '1' THEN q <= d; END IF ; END PROCESS FlipFlop; END ARCHITECTURE Behave;