Appendix E

VHDL Quick Reference Guide

Category	Definition	Example
Identifer Names	Can contain any letter, digit, or underscore _ Must start with alphabetic letter Can not end with underscore or be a keyword Case insensitive	q0 Prime_number lteflg
Signal Values	'0' = logic value 0 '1' = logic value 1 'Z' = high impedance 'X' = unknown value	
Numbers and Bit Strings	<base>#xxx# B = binary X = hexadecimal O = octal	35 (default decimal) 16#C# = "1100" X"3C" = B"00111100" O"234" = B"010011100"
Generic statement	Associates an identifer name with a value that can be overridden with the **generic map** statement	`generic ( N:integer := 8);`
generic map	Assigns a value to a generic parameter	`generic map (N => 16)`

Signals and Variables Types	**signal** (used to connect one logic element to another)
	variable (variables assigned values in process)
	integer (useful for loop control variables)
		`signal d : std_logic_vector(0 to 3);`
		`signal led: std_logic;`
		`variable q: std_logic_vector(7 downto 0);`
		`variable k: integer;`

Program structure	`library IEEE;`
	`use IEEE.STD_LOGIC_1164.all;`
	
	`entity <identifier> is`
	`  port(`
	`  <port interface list );`
	`end <identifier>;`
	
	`architecture <identifier> of <entity_name> is`
	`begin`
	`  process(clk, clr)`
	`  begin`
	`  {{concurrent_statement}}`
	`  end<identifier>;`
		`library IEEE;`
		`use IEEE.STD_LOGIC_1164.all;`
		
		`entity Dff is`
		`  port(`
		`  clk : in STD_LOGIC;`
		`  clr : in STD_LOGIC;`
		`  D : in STD_LOGIC;`
		`  q : out STD_LOGIC );`
		`end Dff;`
		
		`architecture Dff of Dff is`
		`begin`
		`  process(clk, clr)`
		`  begin`
		`  if(clr = '1') then`
		`  q <= '0';`
		`  elsif(rising_edge(clk))then`
		`  q <= D;`
		`  end if;`
		`  end process;`
		`end Dff;`

Logic operators	`not`
	`and`
	`or`
	`nand`
	`nor`
	`xor`
	`xnor`
		`z <= not y;`
		`c <= a and b;`
		`z <= x or y;`
		`w <= u nand v;`
		`r <= s nor t;`
		`z <= x xor y;`
		`d <= a xnor b;`

VHDL Quick Reference Guide (cont.)

Arithmetic operators	+ (addition) - (subtraction) * (multiplication) / (division) (not synthesizable rem (remainder)	`count <= count + 1;` `q <= q - 1;`
Relational operators	=, /=, >, <, >=, <=	`if a <= b then` `if clr = '1' then`
Shift operators	shl (arg,count) shr (arg,count)	`c = shl(a,3);` `c = shr(a,4);`

process	[<id>] **process**(<sensitivity list>)
	{{process declaration}}
	begin
	{{sequential statement}}
	end process [<id>]
		`process(a)`
		`variable j: integer;`
		`begin`
		`j := conv_integer(a);`
		`for i in 0 to 7 loop`
		`if(i = j) then`
		`y(i) <= '1';`
		`else`
		`y(i) <= '0';`
		`end if;`
		`end loop;`
		`end process;`

if statement	**if**(expression1) **then**
	{{statement;}}
	{{**elsif** (expression2) **then**
	{{statement;}} }}
	[[**else**
	{{statement;}}]]
	end if;
		`if(clr = '1') then`
		`q <= '0';`
		`elsif(clk'event and clk = '1') then`
		`q <= D;`
		`end if;`

case statement	**case** expression **is**
	((**when** choices => {sequential statement;}}))
	{{ … }}
	when others => {sequential statement;}}
	end case;
		`case s is`
		`when "00" => z <= c(0);`
		`when "01" => z <= c(1);`
		`when "10" => z <= c(2);`
		`when "11" => z <= c(3);`
		`when others => z <= c(0);`
		`end case;`

for loop	**for** identifier **in** range **loop**
	{{sequential statement}
	end loop;
		`zv := x(1);`
		`for i in 2 to 4 loop`
		`zv := zv and x(i);`
		`end loop;`
		`z <= zv;`

Assignment operator	**:=** (variable) **<=** (signal)	`z := z + x(i);` `count <= count + 1;`

Port map	instance_name component_name **port map**
	(port_association_list);
		`M1 : mux21a port map(`
		`a => c(0), b => c(1),`
		`s => s(0), y => v);`