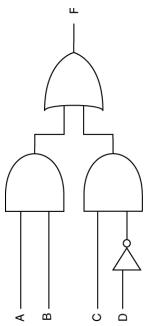
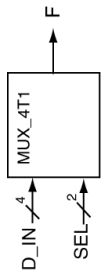
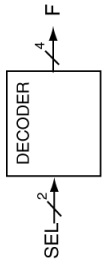
VHDL Cheat-Sheet

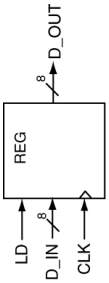
Concurrent Statements		Sequential Statements
Concurrent Signal Assignment (dataflow model)	⇔	**Signal Assignment**
target <= *expression*;		*target* <= *expression*;
A <= B **AND** C; DAT <= (D **AND** E) **OR** (F **AND** G);		A <= B **AND** C; DAT <= (D **AND** E) OR (F **AND** G);
Conditional Signal Assignment (dataflow model)	⇔	***if* statements**

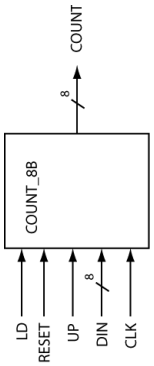
target <= *expressn* **when** *condition* **else**
expressn **when** *condition* **else**
expressn;
		if (*condition*) **then**
		{ sequence of statements }
		elsif (*condition*) **then**
		{ sequence of statements }
		else --(the else is optional)
		{ sequence of statements }
		end if;

F3 <= '1' **when** (L='0' **AND** M='0') **else** '1' **when** (L='1' **AND** M='1') **else** '0';		**if** (SEL = "111") **then** F_CTRL <= D(7); **elsif** (SEL = "110") **then** F_CTRL <= D(6); **elsif** (SEL = "101") **then** F_CTRL <= D(1); **elsif** (SEL = "000") **then** F_CTRL <= D(0); **else** F_CTRL <= '0'; **end if**;
Selective Signal Assignment (dataflow model)	⇔	***case* statements**
with *chooser_expression* **select** target <= *expression* **when** *choices*, *expression* **when** *choices*;		**case** (*expression*) **is** **when** *choices* => {sequential statements} **when** *choices* => {sequential statements} **when others** => -- (optional) {sequential statements} **end case**;
with SEL **select** MX_OUT <= D3 **when** "11", D2 **when** "10", D1 **when** "01", D0 **when** "00", '0' **when others**;		**case** ABC **is** **when** "100" **=>** F_OUT **<=** '1'; **when** "011" **=>** F_OUT **<=** '1'; **when** "111" **=>** F_OUT **<=** '1'; **when others =>** F_OUT **<=** '0'; **end case**;
Process (behavioral model)		
opt_label: **process**(*sensitivity_list*) **begin** {*sequential_statements*} **end process** *opt_label*;		

proc1: **process**(A,B,C)
begin
if (A = '1' **and** B = '0') **then**
F_OUT **<=** '1';
elsif (B = '1' **and** C = '1') **then**
F_OUT **<=** '1';
else
F_OUT **<=** '0';
end if;
end process proc1;		

Description	CKT Diagram	VHDL Model
Typical logic circuit		**entity** my_ckt **is** **Port** (A,B,C,D : **in std_logic**; F : **out std_logic**); **end** my_ckt; **architecture** ckt1 **of** my_ckt **is** **begin** F <= (A **AND** B) **OR** (C AND (**NOT** D)); **end** ckt1; **architecture** ckt2 **of** my_ckt **is** **begin** F <= '1' **when** (A = '1' **AND** B = '1') **else** '1' **when** (C = '1' **AND** D = '0') **else** '0'; **end** ckt2;
4:1 Multiplexor		**entity** MUX_4T1 **is** **Port** (SEL : **in std_logic_vector**(1 **downto** 0); D_IN : **in std_logic_vector**(3 **downto** 0); F : **out std_logic**); **end** MUX_4T1; **architecture** my_mux **of** MUX_4T1 **is** **begin** F <= D_IN(0) **when** (SEL = "00") **else** D_IN(1) **when** (SEL = "01") **else** D_IN(2) **when** (SEL = "10") **else** D_IN(3) **when** (SEL = "11") **else** '0'; **end** my_mux;
2:4 Decoder		**entity** DECODER **is** **Port** (SEL : **in std_logic_vector**(1 **downto** 0); F : **out std_logic_vector**(3 **downto** 0)); **end** DECODER; **architecture** my_dec **of** DECODER **is** **begin** **with** SEL **select** F <= "0001" **when** "00", "0010" **when** "01", "0100" **when** "10", "1000" **when** "11", "0000" **when others**; **end** my_dec;

8-bit register with chip load enable		**entity** REG **is**
		port (LD,CLK : **in std_logic**;
		D_IN : **in std_logic_vector** (7 **downto** 0);
		D_OUT : **out std_logic_vector** (7 **downto** 0));
		end REG;
		
		architecture my_reg **of** REG **is**
		begin
		process (CLK,LD)
		begin
		if (LD = '1' **and rising_edge**(CLK)) **then**
		D_OUT <= D_IN;
		end if;
		end process;
		end my_reg;

8-bit up/down counter with asynchronous reset		**entity** COUNT_8B **is**
		port (RESET,CLK,LD,UP : **in std_logic**;
		DIN : **in std_logic_vector** (7 **downto** 0);
		COUNT : **out std_logic_vector** (7 **downto** 0));
		end COUNT_8B;
		
		architecture my_count **of** COUNT_8B **is**
		signal t_cnt : **std_logic_vector**(7 **downto** 0);
		begin
		process (CLK, RESET)
		begin
		if (RESET = '1') **then**
		t_cnt <= (others => '0'); -- clear
		elsif (rising_edge(CLK)) **then**
		if (LD = '1') **then** t_cnt <= DIN; -- load
		else
		if (UP = '1') **then** t_cnt <= t_cnt + 1; -- incr
		else t_cnt <= t_cnt - 1; -- decr
		end if;
		end if;
		end if;
		end process;
		COUNT <= t_cnt;
		end my_count;