Real-Time Imaging and Control
Practice

MSCV/ESIREM

Antoine Lavault
antoine.lavault@u-bourgogne.fr

VHDL 1

Most of the exercises will require you to write VHDL on paper. Pay attention to the syntax.

Problem 1

Create a D flip-flop with the following:

1. 2 inputs: D, clk
2. 1 output: q

```
LIBRARY IEEE;
USE IEEE.Std_logic_1164.ALL;

ENTITY dff IS
    PORT (
        Q : OUT STD_LOGIC;
        Clk : IN STD_LOGIC;
        D : IN STD_LOGIC
    );
END dff;
ARCHITECTURE Behavioral OF dff IS
BEGIN
    PROCESS (Clk)
    BEGIN
        IF (rising_edge(Clk)) THEN
            Q <= D;
        END IF;
    END PROCESS;
END Behavioral;
```

Problem 2

Create a synchronous 8-bit counter with asynchronous reset :

1. 2 inputs: reset, clk
2. 1 output: s

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ENTITY COUNT_8B IS
    PORT (
        RESET, CLK : IN STD_LOGIC;
        COUNT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END COUNT_8B;
ARCHITECTURE my_count OF COUNT_8B IS
    SIGNAL t_cnt : unsigned(7 DOWNTO 0); -- internal counter signal
BEGIN
    PROCESS (CLK, RESET)
    BEGIN
        IF (RESET = '1') THEN
            t_cnt <= (OTHERS => '0'); -- clear
        ELSIF (rising_edge(CLK)) THEN
            -- NOTE : addition on unsigned/signed overflows in VHDL.
 ↪
            t_cnt <= t_cnt + 1; -- incr
        END IF;
    END PROCESS;
    COUNT <= STD_LOGIC_VECTOR(t_cnt);
END my_count;
```

Problem 3

Create a NOR gate with a generic number of inputs greater or equal to 4.
You can use the following entity:

```vhdl
entity NORgate is
        generic (
                W : positive := 4 -- number of inputs of the NOR gate
        );
        port (
                input : in std_logic_vector(W - 1 downto 0);
                output : out std_logic
        );
end NORgate;
```

```
ENTITY NOR_gate IS
    GENERIC (
        W : POSITIVE := 4 -- number of inputs
    );
    PORT (
        input : IN STD_LOGIC_VECTOR(W - 1 DOWNTO 0);
        output : OUT STD_LOGIC
    );
END NOR_gate;

ARCHITECTURE behavioural OF NOR_gate IS
    SIGNAL s : STD_LOGIC_VECTOR(W DOWNTO 0);

BEGIN
    s(0) <= '0';
    generator : FOR i IN 0 TO W-1 GENERATE
        s(i + 1) <= input(i) OR s(i);
    END GENERATE;
    output <= NOT(s(W));
END behavioural;
```

Problem 4

Create an 8-bit shift register:

NB: In digital circuits, a shift register is a cascade of D flip flops, sharing the same clock, in which the output of each flip-flop is connected to the 'data' input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the 'bit array' stored in it, 'shifting in' the data present at its input and 'shifting out' the last bit in the array, at each transition of the clock input.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
-- This describes a SIPO (Serail In, Parallel Out) shift-register.
ENTITY SREG_8B IS
    GENERIC (N : INTEGER := 8);
    PORT (
        D_IN, CLK : IN STD_LOGIC;
        --D_OUT : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0)); -- Parallel
 ↪  Out
        D_OUT : OUT STD_LOGIC; -- serial Out
END SREG_8B;
ARCHITECTURE my_count OF SREG_8B IS
    SIGNAL REG_TMP : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
----------------------------------
```

```
BEGIN
    PROCESS (CLK)
    BEGIN
        IF (rising_edge(CLK)) THEN
            REG_TMP <= REG_TMP(N-2 DOWNTO 0) & D_IN;
        END IF;
    END PROCESS;
    D_OUT <= REG_TMP(N-1); -- 1 bit
END SREG_8B;
```

Problem 5

Create a multiplexer (mux) with the following:

1. 2 inputs : data(8), sel(3)
2. 1 output : s(1)
3. 1 CLK for synchronous behavior

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ENTITY mux IS
    PORT (
        clk : IN STD_LOGIC;
        input : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        sel : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        output : OUT STD_LOGIC);
END mux;

ARCHITECTURE bhv OF mux IS
BEGIN
    --std_logic_vector are indexed by type integer !
    PROCESS (clk) BEGIN
        IF rising_edge(clk) THEN
            output <= input(to_integer(unsigned(sel)));
        END IF;
    END PROCESS;
END bhv;
```

Problem 6

Create a VHDL Design that allows the same behavior of the common-cathode BCD[1]-decoder (figure 2), the 74LS48 IC[2].

[1]Binary-Coded Decimal
[2]http://pdf.datasheetcatalog.com/datasheet/motorola/74LS48.pdf

A BCD-decoder displays the value on a 7-segment display (see figure 1).

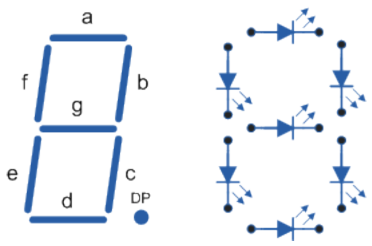

Figure 1: Example of 7 segments display

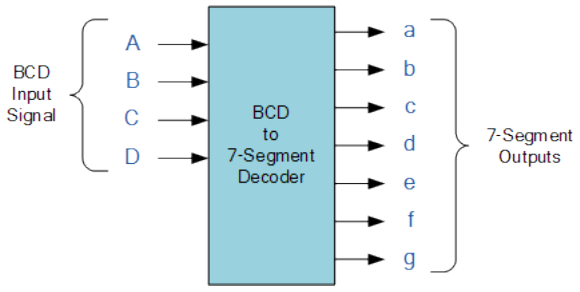

Figure 2: BCD-decoder

NB: 7-segment LED (Light Emitting Diode) or LCD (Liquid Crystal Display) type displays provide a very convenient way of displaying information or digital data in the form of numbers, letters, or even alpha-numerical characters.
Typically, 7-segment displays consist of seven individual colored LEDs (called segments) within one display package. To produce the required numbers or HEX characters from 0 to 9 and A to F, respectively, on the display, the correct combination of LED segments needs to be illuminated and BCD to 7-segment Display Decoders.
A standard 7-segment LED display generally has 8 input connections, one for each LED segment and one as a common terminal or connection for all the internal display segments. Some single displays also have an additional input pin to display a decimal point in their lower right or left-hand corner.

```
---------------------------------------------------------------------
-- BCD_IN: in std_logic_vector(3 downto 0);
-- SSEG:
out std_logic_vector(6 downto 0);
---------------------------------------------------------------------
with BCD_IN select
SSEG <= "0000001" when "0000",
-- 0
"1001111" when "0001",
-- 1
"0010010" when "0010",
-- 2
```

```
"0000110" when "0011",
-- 3
"1001100" when "0100",
-- 4
"0100100" when "0101",
-- 5
"0100000" when "0110",
-- 6
"0001111" when "0111",
-- 7
"0000000" when "1000",
-- 8
"0000100" when "1001",
-- 9
"0001000" when "1010",
-- A
"1100000" when "1011",
-- b
"0110001" when "1100",
-- C
"1000010" when "1101",
-- d
"0110000" when "1110",
-- E
"0111000" when "1111",
-- F
"1111111" when others;
-- turn off all LEDs
```

Problem 7

Propose a VHDL component with a N-bit input x ($N \geq 8$) with output at a high logic level when there are at least 4 '1' and 4 '0' in x.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY bit_counter IS
    GENERIC (
        W : POSITIVE := 8 -- le nombre de bits d'entrée
    );
    PORT (
        I : IN STD_LOGIC_VECTOR(W - 1 DOWNTO 0);
        F : OUT STD_LOGIC
    );
END bit_counter;
```

```
ARCHITECTURE behavioural OF bit_counter IS
BEGIN
    PROCESS (I)
        VARIABLE count_0 : NATURAL RANGE 0 TO W;
        VARIABLE count_1 : NATURAL RANGE 0 TO W;
    BEGIN
        count_0 := 0;
        count_1 := 0;
        FOR k IN W - 1 DOWNTO 0 LOOP
            IF I(k) = '0' THEN
                count_0 := count_0 + 1;
            END IF;
            IF I(k) = '1' THEN
                count_1 := count_1 + 1;
            END IF;
        END LOOP;

        IF (count_0 >= 4 AND count_1 >= 4) THEN
            F <= '1';
        ELSE
            F <= '0';
        END IF;

    END PROCESS;
END behavioural;
```

Problem 8

The following code describes a finite state machine.

```
-- A Mealy machine has outputs that depend on both the state and
-- the inputs.        When the inputs change, the outputs are updated
-- immediately, without waiting for a clock edge.  The outputs
-- can be written more than once per state or clock cycle.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mealy_4s IS
        PORT (
                clk : IN STD_LOGIC;
                data_in : IN STD_LOGIC;
                reset : IN STD_LOGIC;
                data_out : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
        );
END ENTITY;
```

```
ARCHITECTURE rtl OF mealy_4s IS
        -- Build an enumerated type for the state machine
        TYPE state_type IS (s0, s1, s2, s3);
        -- Register to hold the current state
        SIGNAL state : state_type;
BEGIN
        PROCESS (clk, reset)
        BEGIN
                IF reset = '1' THEN
                        state <= s0;
                ELSIF (rising_edge(clk)) THEN
                        -- Determine the next state synchronously, based on
                        -- the current state and the input
                        CASE state IS
                                WHEN s0 =>
                                        IF data_in = '1' THEN
                                                state <= s1;
                                        ELSE
                                                state <= s0;
                                        END IF;
                                WHEN s1 =>
                                        IF data_in = '1' THEN
                                                state <= s2;
                                        ELSE
                                                state <= s1;
                                        END IF;
                                WHEN s2 =>
                                        IF data_in = '1' THEN
                                                state <= s3;
                                        ELSE
                                                state <= s2;
                                        END IF;
                                WHEN s3 =>
                                        IF data_in = '1' THEN
                                                state <= s3;
                                        ELSE
                                                state <= s1;
                                        END IF;
                        END CASE;

                END IF;
        END PROCESS;

        -- Determine the output based only on the current state
        -- and the input (do not wait for a clock edge).
        PROCESS (state, data_in)
        BEGIN
                CASE state IS
```

```
                    WHEN s0 =>
                            IF data_in = '1' THEN
                                    data_out <= "00";
                            ELSE
                                    data_out <= "01";
                            END IF;
                    WHEN s1 =>
                            IF data_in = '1' THEN
                                    data_out <= "01";
                            ELSE
                                    data_out <= "11";
                            END IF;
                    WHEN s2 =>
                            IF data_in = '1' THEN
                                    data_out <= "10";
                            ELSE
                                    data_out <= "10";
                            END IF;
                    WHEN s3 =>
                            IF data_in = '1' THEN
                                    data_out <= "11";
                            ELSE
                                    data_out <= "10";
                            END IF;
            END CASE;
    END PROCESS;

END rtl;
```

What is the function described by this finite state machine?
2-bit synchronous up/down counter.

Problem 9

Create the state-machine represented by the following pictures (figure: 3) :

You can assume the output is the number of the state (i.e., $S0 \rightarrow 0$, $S1 \rightarrow 1$, etc...)
Hint: you can use custom sub-types to encode the states like

```
type state_type is (idle, state1, state2,...);
signal present_state, next_state: state_type;
```

Problem 10

The parity function is true if the number of bits equal to '1' at its input is even.
For instance, this function is true for $0000; 1111; 1001$ and false for $0001; 0111$.

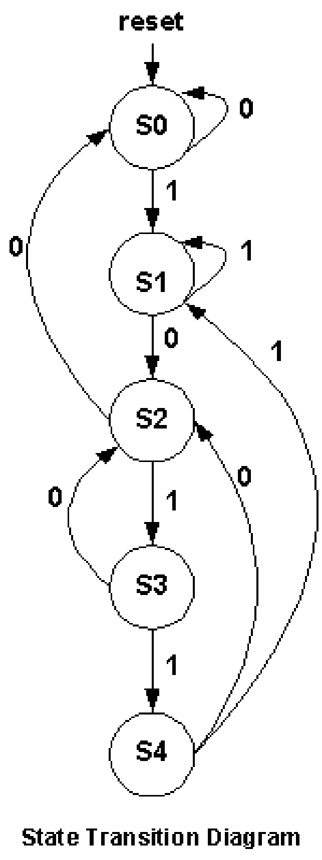

Figure 3: State machine

```
entity parity is
        generic (        W : positive := 4); -- number of input bits
        port (
                I : in std_logic_vector(W - 1 downto 0);
                F : out std_logic   -- 1 if parity, 0 if not.
        );
end parity;
```

Write a testbench completely checking the `parity` module with an arbitrary number of bits.

Remark 1. *You can instantiate a UUT with the following snippet:*

```
UUT: entity f(arch) generic map (W) port map (testVector, response);
```

VHDL functions or external files can get the expected test responses.
You can also use the `assert...report` *statement.*
Finally, it is easy to iterate over N bits with a for loop and get all possible values.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY parity_tb IS
    GENERIC (W : POSITIVE := 5); -- le nombre de bits d'entrée de
 ↪  l'UUT
END parity_tb;

ARCHITECTURE behavioural OF parity_tb IS

    SIGNAL testVector : STD_LOGIC_VECTOR(W - 1 DOWNTO 0);
    SIGNAL response : STD_LOGIC;

    -- parity check function
    FUNCTION parityEven(input : STD_LOGIC_VECTOR(W - 1 DOWNTO 0))
 ↪  RETURN BOOLEAN IS
        VARIABLE sum : NATURAL := 0;
    BEGIN
        FOR k IN 0 TO W - 1 LOOP
            IF input(k) = '1' THEN
                sum := sum + 1;
            END IF;
        END LOOP;
        -- even parity iif sum is 0 mod 2
        RETURN (sum MOD 2 = 0);
    END parityEven;

BEGIN

    -- module instantiation
    UUT : ENTITY work.parity(behavioural) GENERIC MAP (W) PORT MAP
 ↪  (testVector, response);

    -- exhaustive test
    PROCESS
    BEGIN
        FOR k IN 0 TO 2 ** W - 1 LOOP
            -- this will work in simulation
            testVector <= STD_LOGIC_VECTOR(to_unsigned(k, W));
            --          report integer'image(k);
            WAIT FOR 10 ns;
            ASSERT (parityEven(testVector) = (response = '0'))
            REPORT "input error " & INTEGER'image(k) SEVERITY error;
        END LOOP;
        REPORT "simulation, all done ! " SEVERITY failure;
    END PROCESS;

END behavioural;
```

Problem 11

A billionaire, Bruce W., fears the privacy of his "special" affairs and wants to add a CCTV system to his manor.
The cameras used in the system are numbered $0, 1, 2, 3 \cdots N$ and can detect intruders, whatever the time of day is. When an intruder is detected, the camera sends a signal on a dedicated data path.
To connect all these cameras, you are tasked to design a system that activates an alarm when at least one camera detects an intruder and gives the ID number of the activated camera (as a binary number). Since Mr. W. is feared by his enemies, and for good reasons, he is not expected to see more than one intruder at a time. But the system should raise an error flag if, by any chance, more than one camera detected an intruder.

1. Give a VHDL code describing the case with 4 cameras
2. Same question with a generic number of cameras.

```
ARCHITECTURE archflexible OF security IS
BEGIN
    PROCESS (camera_alert)
        VARIABLE count : NATURAL RANGE 0 TO N;
    BEGIN
        count := 0; -- persistent variables
        --  should be reset on every iteration
        intruder_alarm <= '0'; -- default value
        FOR k IN N - 1 DOWNTO 0 LOOP
            IF camera_alert(k) = '1' THEN
                camera_code <= to_unsigned(k, camera_code'length);
                intruder_alarm <= '1';
                count := count + 1;
            END IF;
        END LOOP;
        IF count >= 2 THEN
            error <= '1';
        ELSE
            error <= '0';
        END IF;
    END PROCESS;
END archflexible;
```