Custom Computing & Embedded Systems

FPGA simulation and debug – sequential logic

1. Thematic scope of the exercise:

- description of sequential circuits in VHDL,
- use of subroutines in simulation,
- simulation using the VHDL testbench .

2. Modeling of sequential systems

In the behavioral description of sequential systems, a process is an essential element. It is a construction of the VHDL language, which as a whole is a concurrent instruction, while the instructions within the process are run sequentially - in accordance with the order in which they were written.

During simulation, the process can be triggered based on the sensitivity list or using the 'wait' instruction, but both of these elements cannot appear in the process syntax at the same time.

2.1. Description of the flip-flop

The simplest sequential elements are flip-flops and latches. The difference between them is the trigger method: the latch is triggered by the signal level and the flip-flop is triggered by the signal edge.

Inputs		Outputs	
D	*EN*	*Q*	$\overline{Q}$
0	1	0	1
1	1	1	0
X	0	Q_0	$\overline{Q}_0$

Inputs		Outputs	
D	CLK	*Q*	$\overline{Q}$
0	↑	0	1
1	↑	1	0

↑ = clock transition LOW to HIGH

```
9  entity dlatch is
10     generic (delay: time:=1 ns);
11     Port     (en : in std_logic;
12               rst : in std_logic;
13               d : in std_logic;
14               q : out std_logic);
15 end entity dlatch;
16
17 architecture latch of dlatch is
18
19 begin
20 process(en,rst,d)
21     begin
22     if rst='1' then
23         q<='0' after delay;
24     elsif en='1' then
25         q<=d after delay;
26     end if;
27     end process;
28 end architecture latch;
```

```
9  entity dff is
10     generic (delay: time:=1 ns);
11     Port     (clk : in std_logic;
12               rst : in std_logic;
13               d : in std_logic;
14               q : out std_logic);
15 end entity dff;
16
17 architecture flip_flop of dff is
18
19 begin
20 process(clk,rst)
21     begin
22     if rst='1' then
23         q<='0' after delay;
24     elsif clk='1' and clk'event then
25         q<=d after delay;
26     end if;
27     end process;
28 end architecture flip_flop;
```

The items are described using a process - note the contents of the sensitivity list in both cases.

[!] Create a new Questa simulator project and add .vhd from the course server
[!] Compile from the command line with the command:

```
vcom -2008 -autoorder *.vhd
```

Run the latch and flip-flop simulation with the command:

```
vsim -voptargs=+acc work.f_vs_l_tb
```

Save all master level time waveforms for later verification

[!] Modify the **dlatch**.vhd source by removing signal D from the process sensitivity list
Rerun compilation and simulation:

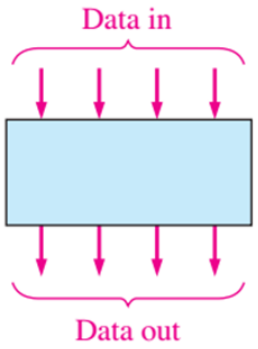
```
vcom -2008 -autoorder *.vhd
vsim -voptargs=+acc work.f_vs_l_tb
```

Compare the received waveforms with previously saved ones

2.2. Description of registers

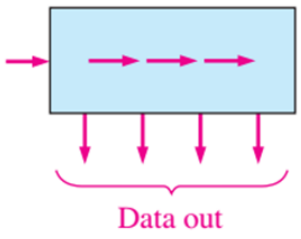
The implementation of a parallel register does not differ much from the description of a single flip-flop - the architecture structure is identical, only the description of the interface differs.

```
9  entity reg4 is
10     generic (delay: time:=1 ns);
11     Port     (clk : in std_logic;
12               d : in std_logic_vector(3 downto 0);
13               q : out std_logic_vector(3 downto 0) );
14 end entity reg4;
15
16 architecture behav of reg4 is
17
18 begin
19 process(clk)
20     begin
21     if rising_edge(clk) then
22         q <= d after delay;
23     end if;
24     end process;
25 end architecture behav;
```



The description of registers with serial input is similar. Below is an example of a register with parallel output, shifting right.

```
9  entity reg4sr is
10     generic (delay: time:=1 ns);
11     Port     (clk : in std_logic;
12               d : in std_logic;
13               q : out std_logic_vector(3 downto 0) );
14 end entity reg4sr;
15
16 architecture behav of reg4sr is
17     signal q_tmp : std_logic_vector(q'range);
18 begin
19 process(clk)
20     begin
21     if rising_edge(clk) then
22         q_tmp <= d & q_tmp(3 downto 1);
23     end if;
24     end process;
25 q <= q_tmp after delay;
26 end architecture behav;
```



In fact, all types of registers that perform a shift or rotation operation can be described using the concatenation operator (&). Just replace code line 22 with the appropriate operation:

shift left	`q_tmp <= q_tmp(2 downto 0) & d;`
rotate right	`q_tmp <= q_tmp(0) & q_tmp(3 downto 1);`
rotate left	`q_tmp <= q_tmp(2 downto 0) & q_tmp(3);`

2.3. Description of counters

Digital circuits based on programmable logic mainly use synchronous counters. The principles of their description are consistent with the principles of register description, where, depending on the type of counter, the shift operation is replaced by increment or decrement of a signal of the appropriate data type.

A simple binary synchronous counter, counting forward:

```
7   use IEEE.STD_LOGIC_1164.ALL;
8   use IEEE.numeric_std.all;
9   use IEEE.STD_LOGIC_UNSIGNED.all;
10
11  entity b_cntr4 is
12      Port    (clk : in std_logic;
13               rst : in std_logic;
14               q : out std_logic_vector(3 downto 0) );
15  end entity b_cntr4;
16
17  architecture behav of b_cntr4 is
18      signal q_tmp : std_logic_vector(q'range) := x"0";
19  begin
20  process(clk) begin
21      if rising_edge(clk) then
22          if rst='1' then
23              q_tmp <= x"0";
24          else
25              q_tmp <= q_tmp + 1;
26          end if;
27      end if;
28      end process;
29  q <= q_tmp;
30  end architecture behav;
```

Synchronous decimal counter, with terminal counting output (tc) and combinational clock enable output (ceo):

```
7   use IEEE.STD_LOGIC_1164.ALL;
8   use IEEE.numeric_std.all;
9   use IEEE.STD_LOGIC_UNSIGNED.all;
10
11  entity d_cntr4ceo is
12      Port    (clk : in std_logic;
13               rst : in std_logic;
14               ce : in std_logic;
15               tc : out std_logic;
16               ceo : out std_logic;
17               q : out std_logic_vector(3 downto 0) );
18  end entity d_cntr4ceo;
19
20  architecture behav of d_cntr4ceo is
21      signal q_tmp : std_logic_vector(q'range) := x"0";
22      signal tci : std_logic;
23  begin
24  process(clk) begin
25      if rising_edge(clk) then
26          if rst='1' then
27              q_tmp <= x"0";
28          elsif ce='1' then
29              if tci='1' then
30                  q_tmp <= x"0";
31              else
32                  q_tmp <= q_tmp + 1;
33              end if;
34          end if;
35      end if;
36  end process;
37  -- outputs
38  tci <= '1' when (q_tmp=9) else '0';
39  ceo <= (tci and ce);
40  tc <= tci;
41  q <= q_tmp;
42  end architecture behav;
```

[!] Run the decimal counter simulation with the command :

```
vsim -voptargs=+acc work.d_cntr4ceo_tb
```

Observe all top level waveforms until the end of the simulation
Explain the procedure that ends the simulation

[!] Modify the ***d_cntr4ceo_tb***.*vhd* source so that the simulation ends after two occurrences of the *ceo* pulse. Rerun the compilation and simulation, verify the results achieved

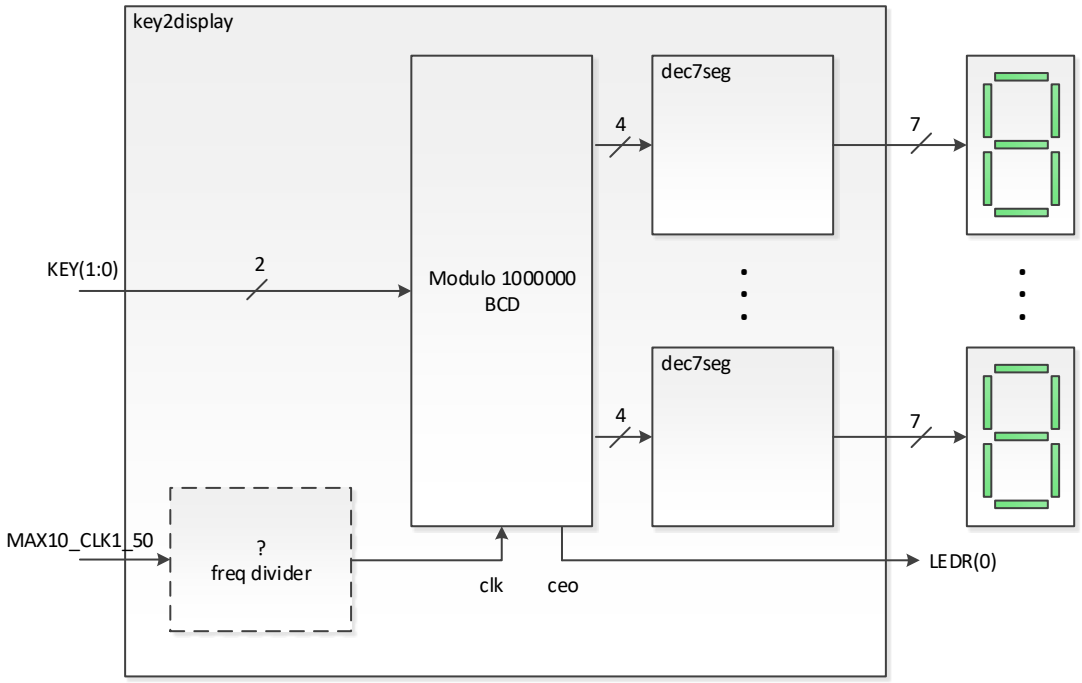
3. The use of subroutines and structure generation

Task A

[!] Add a new file called **key2display**.vhd to the project
Define a design unit with the interface given below

```
entity key2display is
    port (
        max10_clk1_50 : in std_logic;
        key : in std_logic_vector(1 downto 0);
        ledr : out std_logic_vector(0 downto 0);
        hex5,hex4,hex3,hex2,hex1,hex0 : out std_logic_vector(6 downto 0)
    );
end entity key2display;
```

Based on the behavioral description methods you have learned, design a modulo-1_000_000 counter counting in BCD code. In your project, use the *for… generate* structure to automatically generate the layout of circuit.
The designed device should display the result on 7-segment displays. Control using two buttons from the **DE10Lite** board: key (0) – rst , key (1) – ce
Note: KEY buttons are monostable, with a stable high state



[!] Add a new file called **key2display_tb**.vhd to the project
Define a procedural testbench to verify the operation of the device.
Testbench should report the *ceo* pulse duration to the transcription window and the value of the internal bus *q* – the state of the counter output at the moment of *ceo* =1
Prepare a compilation/simulation macro and demonstrate how the system works

Task B

[!] Using the circuits designed in exercises lab1 and lab2, build an illuminated advertisement consisting of at least 10 characters displayed on 7-segment LED indicators.

Requirements:
- ability to change the direction of text scrolling
- ability to change movement speed
- ability to choose two different texts
- preferred control using KEY monostable switches
- demonstration of operation on the **DE10Lite** platform

Note: To display additional characters, you must make your own 7-segment decoder module