Comparator

Task:

- Complete the truth table for a 2-bit comparator (Table 1) and write out the corresponding Boolean equations. Use these equations to describe the comparator in VHDL.
- Use "when .. else" VHDL statement to describe a 2-bit comparator.
- Use "with .. select" VHDL statement to describe a 2-bit comparator.

Table 1: Comparator Truth Table

in1	in2	eq_o	gr_o	ls_o
00	00	?	?	?
00	01	?	?	?
00	10	?	?	?
00	11	?	?	?
01	00	?	?	?
01	01	?	?	?
...	...	...	...	...
11	11	?	?	?

Perform functional simulation to verify correctness of the received VHDL descriptions. Implement and test the designs on FPGA development board. Compare implementation results of all three 2-bit comparators: in RTL Analysis and Synthesis schematics, Synthesis and Implementation reports, etc. How do they differ from each other?

Functional Simulation

In order to test the functionality of the design, e.g. a half adder from lab appendix, it should be simulated with a testbench. Testbench is a VHDL code, which applies stimulus to design entity during simulation. An example of the stimulus for "00" input combination is presented in Listing 1. First of all, inputs are assigned a test value. After a 20ns wait period, the outputs are compared to the expected values using **assert** statement (both *sum* and *carry* outputs should be equal to '0'). An error message is reported if they do not match. The type of message is specified using **severity** statement. In Listing 1 the severity level is set to *error* (the default setting in case **severity** statement is omitted). Other possibilities include *note*, *warning* and *failure* (in case of *failure* the simulation will stop immediately). The exact message to be displayed can be set using **report** statement.

Listing 1: Stimulus for "00" Input Combination

```
a <= '0'; b <= '0';
wait for 20 ns;
assert (sum = '0') and (carry = '0') report "test failed for 00" severity error;
```

Create a new source file and choose *Add or create simulation sources* option. Leave *I/O Port Definitions* in the *Define Module* window empty since testbench does not have any ports. The created testbench source file can be found in the *Sources* window under *Simulation Sources* category in the *Project Manager* flow section layout.

Listing 2: Skeleton Architecture Description for Half Adder Testbench

```
architecture Behavioral of half_adder_tb is

component half_adder is
Port ( a : in STD_LOGIC;
       b : in STD_LOGIC;
       sum : out STD_LOGIC;
       carry : out STD_LOGIC);
end component half_adder;

signal a, b : std_logic := '0';     -- signals for inputs
signal sum, carry : std_logic;   -- signals for outputs

begin

UUT : half_adder
Port map ( a => a,
           b => b,
           sum => sum,
           carry => carry );

stimuli : process
begin
-- insert stimuli here
wait;
end process;

end Behavioral;
```

Double-click the testbench file to open it with the *Text Editor*. The skeleton code generated by Vivado does not contain any necessary declarations and instantiations, so they should be

added manually. Listing 2 provides a skeleton testbench architecture description for the half adder design.

The declarative part of the architecture consists of the unit under test (UUT) component declaration (half adder design) and the declaration of signals that will be mapped to the inputs/outputs of the UUT. Note, that inputs and outputs are declared separately, since inputs are provided with an initial value. Initialization is not needed for the signals that represent outputs since their value depend directly on the value of signals that represent inputs.

The architecture body consists of the UUT component instantiation and the stimuli process. The component instantiation maps ports of the UUT component (on the left) to the signals that are declared in the testbench architecture (to the right). The stimuli process should be altered by adding the code to test all possible input combinations. Note, that the **wait** statement is placed at the end of the process in order to stop the repeating generation of the stimuli (since process loops back to the beginning when **end** statement is reached).

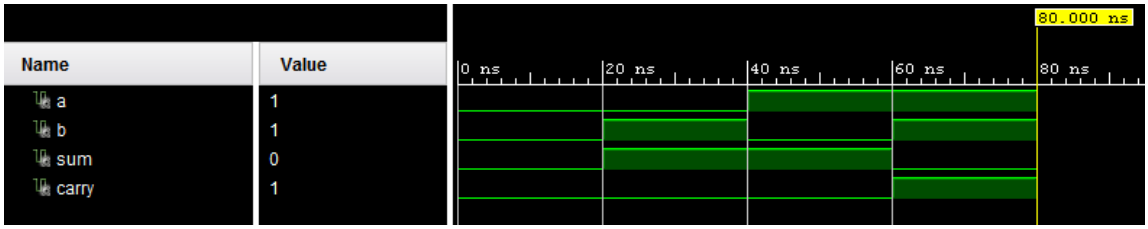

Figure 1: Half Adder Simulation Waveform

When testbench is ready, set it as the top simulation source if this hasn't been done automatically by the tool (or in order to set a different simulation source file as top). Select *Run Simulation* option in the *Flow Navigator* under *Simulation* flow section, click *Run Behavioral Simulation*. The simulation result for half adder design from lab appendix should be similar to the one presented in Figure 1. The simulation result fully corresponds to the truth table of half adder (Table 1 from lab appendix). If **assert** statements have also been used to check the output values, the results are printed in the *Tcl Console* tab. Note, that no message will be printed if the output value matches the specified expected value (an

error message is reported only if they do not match).

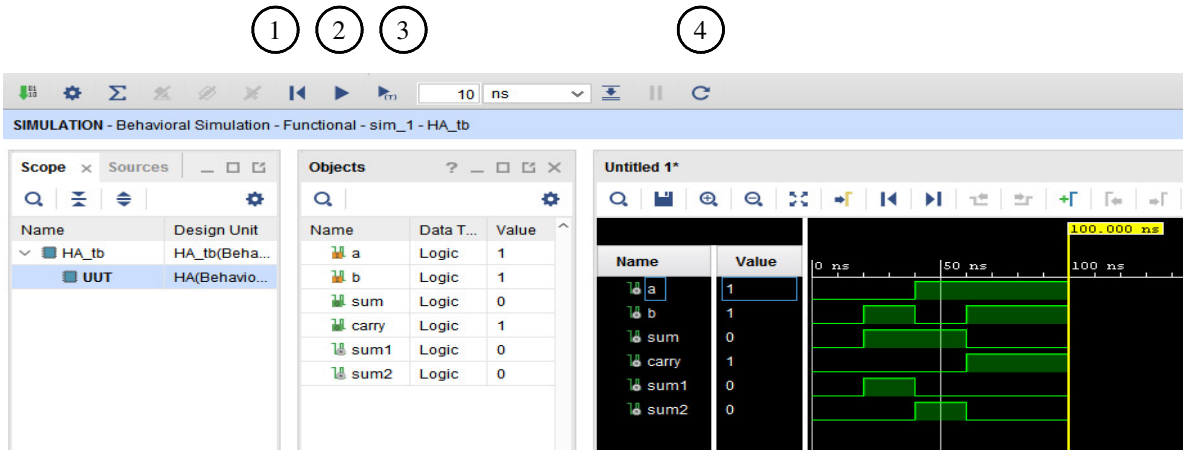

Figure 2: Simulation Layout in Vivado

In case it is required to restart the simulation from the beginning, press the *Restart* button (button labeled as 1 in Figure 2). This will clear the waveform and place cursor at zero time. To run the simulation press either *Run All* button (button labeled as 2 in Figure 2) or *Run for* button (button labeled as 3 in Figure 2). *Run All* command will run the simulation until all signals stop changing. *Run for* command will run the simulation for the amount of time that is specified in the drop down menus to the right. Pushing *Run for* button again will continue simulation from the point when it has been stopped for another specified time period. Finally, in case the source files are changed, the simulator should be relaunched. This can be done by pressing *Relaunch Simulation* button (button labeled as 4 in Figure 2).

Apart from monitoring the values of the signals that are declared in the testbench (and added to the waveform automatically) it is possible to monitor the value on any signal that is declared within the source files as well. For simulation that is shown in Figure 2, the calculation of *sum* signal in Listing 1 from lab appendix is broken down into two parts: signal *sum1* is assigned the value of the first term in the equation and signal *sum2* is assigned the value of the second term. Thus the *sum* signal is calculated by using OR function for signals *sum1* and *sum2*. The *Scope* window in the *Simulation* layout of Vivado (Figure 2) lists all instances in the simulated design hierarchy (with testbench being on top). By expanding the hierarchy it is possible to access any design instance. When design

instance is selected in the *Scope* window, the list of all signals that are declared within that design instance (including input/output ports) is show in the *Object*s window. The modified Half Adder design that is instantiated in the testbench as UUT is selected in Figure 2. Note that *Objects* window lists all inputs/outputs of the Half Added as well as internally declared signals *sum1* and *sum2*. These signals can be drag-and-dropped onto the *Waveform* window. It may be needed to restart the simulation (there is no need to relaunch) to see the values of the newly added signal in the *Waveform* window.

Creating Project Archive

For an easier project sharing or design relocation Vivado can create project ZIP archive. In the *File* dropdown menu select *Project -> Archive* option. In the *Archive Project* window (Figure 3) provide *Archive name* and *Archive location*. Uncheck *Include* options to exclude results of synthesis and implementation runs from the project archive. In this way only design source files will be included, thus archive size should be significantly reduced. Click *OK* to complete creation of the project archive.

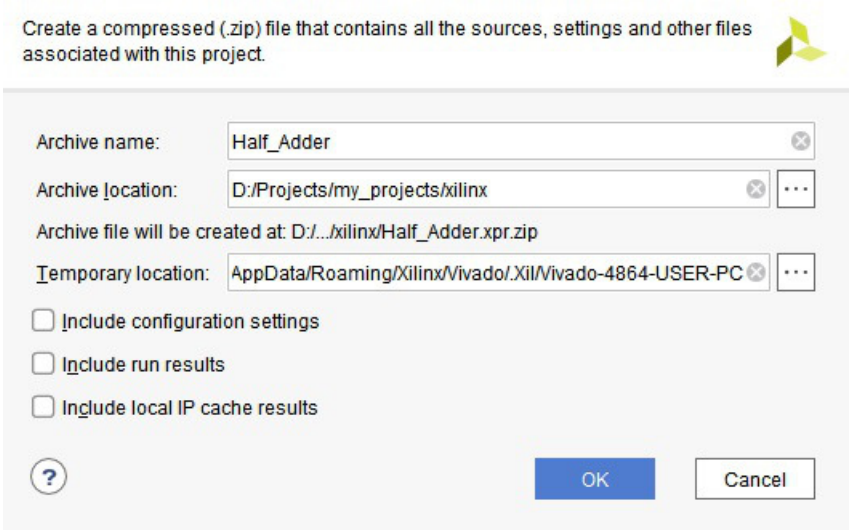

Figure 3: Archive Project Window

Note, that simulation data would still be included in the project archive. It can also take a lot of space depending on the number of simulated signals and its duration. Remove *project_name.sim* folder from the project directory manually before creating the archive.