Elettronica dei Sistemi Digitali
Digital Systems Electronics

Lab1

Multiplexers, Light Emitting Diodes, Switches and Testbench

The purpose of this laboratory session is to learn how to connect simple input-output devices to an FPGA chip and implement a circuit that uses these devices for a specific function. Students have to be familiar with both Modelsim and Quartus prime. Therefore, it is suggested to go through the two available tutorials:

- Use of Modelsim
- Digital design flow with Quartus prime.

Moreover, for every project, the VHDL code must be validated by means of a proper testbench, which generates the inputs necessary to assess the behavior of the designed circuits. To write the testbench, follow the Modelsim tutorial and the general suggestions given at the end of this document.

We will use the switches $SW_{9\text{-}0}$ on the DE1 board as inputs and we light emitting diodes (LEDs) as outputs of our logic circuit.

For each project in this assignment (the same holds for the following laboratory sessions) you have to deliver:

1. A **pdf report** that describes your circuit, what you have done in the lab to complete your design, validate it and run it on the board.
2. Every **source file** necessary to generate from the scratch your project under Modelsim and Quartus Prime (this includes all source files, i.e. files with extension .vhd, and possibly additional files, such as script files, pin assignment files, input-output data files, …). Please, notice that a **testbench** file is requested for every project.
3. **Do not deliver the whole project**, which contains a large amount of files generated by the tool during compilation and synthesis tasks.

Contents:

Abbreviations and acronyms:

IC	– Integrated Circuit
LED	– Light Emitting Diode
MUX	– Multiplexer
VHDL	– Very high speed integrated circuits Hardware Description Language

[VHDL cookbook: http://www.onlinefreeebooks.net/engineering-ebooks/electrical-engineering/the-vhdl-cookbook-pdf.html]

1 – Controlling the LEDs

The DE1 board provides 10 toggle switches, called *SW*9-0 that can be used as circuit inputs, and 10 red lights, called *LEDR*9-0, that can be used to display output values. Figure 1 shows a simple VHDL entity that uses these switches and shows their states on the LEDs. As specified in the code, since there are 10 switches and 10 LEDs, it is convenient to represent them as arrays. Here we have used a single assignment statement for all 10 *LEDR* outputs, which is equivalent to the following individual assignments:

```
LEDR(9)  <= SW(9);
LEDR(8)  <= SW(8);
.  .  .
LEDR(0)  <= SW(0);
```

The DE1 board has hardwired connections between the FPGA chip, the switches and the LEDs. To use *SW*9-0 and *LEDR*9-0 it is necessary to include in your Quartus Prime project the correct pin assignments, which are given in the *DE1_SOC_User_ Manual*. For example, the manual specifies that *SW*0 is connected to the FPGA pin *PIN_AB12* and *LEDR*0 is connected to pin *PIN_V16*. A good way to make the required pin assignments is to import into the Quartus Prime software the file called DE1_SoC.qsf, which is provided on the web site of the course. The procedure for making a pin assignment is described in the first lab session LAB#0.

It is important to realize that the pin assignments in the DE1_SoC.qsf file are useful only if the pin names given in the file are exactly the same as the port names used in your VHDL entity. The file uses the names *SW*[0] . . . *SW*[9] and *LEDR*[0] . . . *LEDR*[9] for the switches and lights. This is the reason why we used these names in Figure 1 (note that the Quartus Prime uses [ ] square brackets for array elements, while the VHDL syntax uses ( ) round brackets).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- Simple module that connects
-- the SW switches to the LEDR lights

ENTITY part1 IS
      PORT ( SW   : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
      LEDR : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)); -- red LEDs
END part1;

ARCHITECTURE Behavior OF part1 IS
BEGIN
      LEDR <= SW;
END Behavior;
```

Figure 1 - VHDL code that uses the switches and LEDs on the DE1 board.

You need to do steps and implement a circuit corresponding to the code in Figure 1 on the DE1 board.

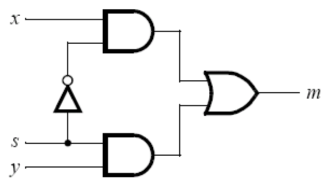
1. First, create two VHDL files for both the **circuit** in Figure 1and the corresponding **testbench**.
2. Then, **create a project under Modelsim**, import the two files and run the behavioural simulation to validate your circuit.
3. **Create a new Quartus Prime project** for your circuit. Select Cyclone V 5CSEMA5F31C6N as the target chip, which is the FPGA chip on the Altera DE1 board.
4. Include in your project the VHDL files and the required **pin assignments** for the DE1 board, as discussed above.
5. **Compile** the project.
6. **Download** the compiled circuit in the FPGA chip.

7. **Test** the functionality of the circuit by toggling the switches and looking at the LEDs.

Use the same I/O pins specified in this document and avoid using if-then-else statements in your VHDL code.
For this project, report the sequence of steps you followed from the initial opening of Quartus Prime up to the final validation of the circuit.

2 - 2-to-1 Multiplexer

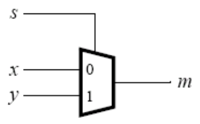
Figure 2*a* shows a sum-of-products circuit that implements a 2-to-1 *multiplexer* with a select input *s*. If $s = 0$ the multiplexer's output *m* is equal to the input *x*, and if $s = 1$ the output is equal to *y*. Part *b* of the figure gives a truth table of this MUX, and part *c* shows its circuit symbol.



a) Circuit

s	*m*
0	*x*
1	*y*

b) Truth table



c) Symbol

Figure 2 - A 2-to-1 multiplexer.

The multiplexer can be described with the following VHDL statement:

```
m <= (NOT (s) AND x) OR (s AND y);
```

You need to write a VHDL entity that includes four assignment statements like the one shown above to describe the circuit given in Figure 3*a*. This circuit has two four-bit inputs, *X* and *Y*, and produces the four-bit output *M*.
If $s = 0$ then $M = X$, while if $s = 1$ then $M = Y$. We refer to this circuit as a four-bit wide 2-to-1 multiplexer. The circuit symbol is shown in Figure 3*b*. The signals *X*, *Y* and *M* are depicted as four-bit wires. You need to do the steps shown below.

1. C**reate a new project under Modelsim**.
2. **Write and include your VHDL file** for the four-bit wide 2-to-1 multiplexer in your project. Use switch SW_8 on the DE1 board as the *s* input, switches SW_{3-0} as the *X* input and SW_{7-4} as the *Y* input. Connect the output *M* to the red lights $LEDR_{3-0}$.
3. Write a **testbench** and simulate the project in Modelsim.
4. **Create a new Quartus Prime project** for your circuit and include VHDL files.
5. Include in your project the required **pin assignments** for the DE1 board. As discussed in Part I, these assignments ensure that the input ports of your VHDL code use the pins on the Cyclone V FPGA that are connected to the *SW* switches. Moreover, ensure that the output ports of your VHDL code use the FPGA pins connected to the *LEDR* lights.

6. **Compile** the project.
7. **Download** the compiled circuit into the FPGA chip.
8. **Test** the functionality of the four-bit wide 2-to-1 multiplexer by toggling the switches and observing the LEDs.

Again, report the sequence of steps you followed from the initial opening of Quartus Prime up to the final validation of the circuit.

3 – 5-to-1 Multiplexer

Figure 2 shows a 2-to-1 multiplexer that handle the two inputs *x* and *y*. Now, consider a circuit in which the output *m* has to be selected from five inputs *u*, *v*, *w*, *x*, and *y*. Part *a* of Figure 4 shows how we can make the required 5-to-1 multiplexer by using four 2-to-1 multiplexers. The circuit uses a 3-bit select input $s_2\, s_1\, s_0$ and implements the truth table shown in Figure 4*b*. A circuit symbol for this multiplexer is given in part *c* of the figure.

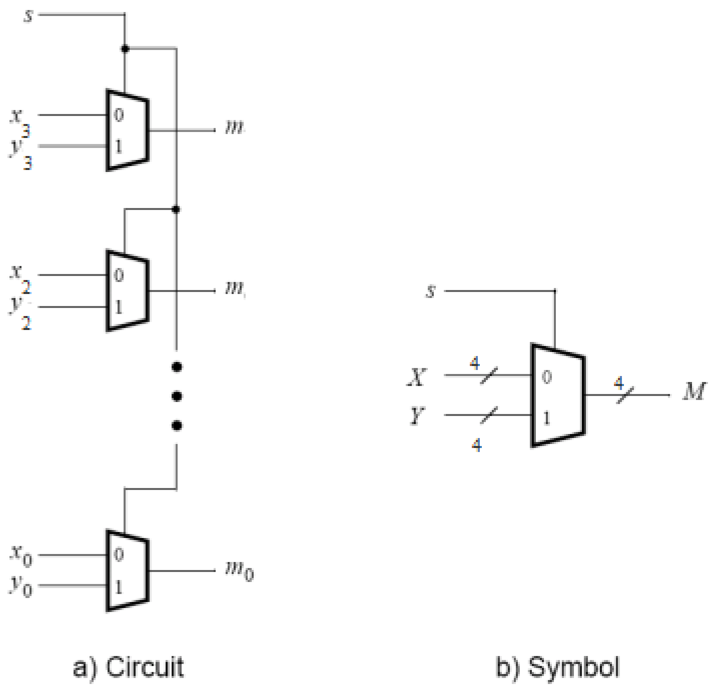


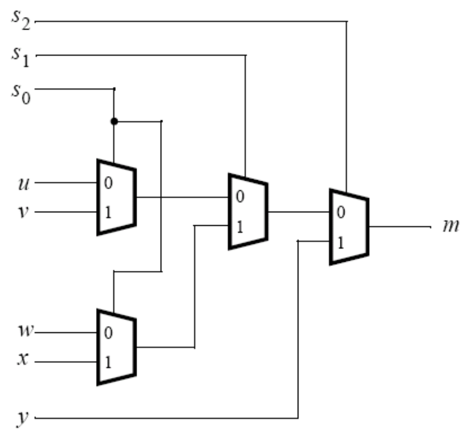
Figure 3 - A four-bit wide 2-to-1 multiplexer.

Figure 3 reminds you that a four-bit wide 2-to-1 multiplexer can be built out of four instances of a 2 to-1 multiplexer. Figure 5 applies this concept to define a three-bit wide 5-to-1 multiplexer. It contains three instances of the circuit in Figure 4*a*.

You need to complete the following steps and implement the 3-bit wide 5-to-1 multiplexer.

1. **Create a new Modelsim project** for your circuit.

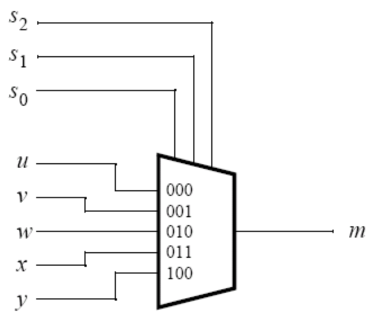
2. **Create a VHDL model** for the three-bit wide 5-to-1 multiplexer. Connect its selection inputs to switches SW_{8-6}, and use the other 6 switches to provide two 3-bit inputs *X* and *Y*. In this case, force the other three inputs to the constant values “101”, “010”,”111”. Connect the output *M* to the red lights $LEDR_{2-0}$.
3. Write a **testbench** and simulate the project in Modelsim.
4. **Create a new Quartus Prime project** for your circuit and include in your project the VHDL files and the required **pin assignments** for the DE1 board. Compile the project.
5. **Download** the compiled circuit in the FPGA IC.
6. **Test** the functionality of the three-bit wide 5-to-1 multiplexer by toggling the switches and observing the LEDs. Ensure that each input from *U* to *Y* can be properly multiplexed on the output *M*.



a) Circuit

s_2	s_1	s_0	m
0	0	0	u
0	0	1	v
0	1	0	w
0	1	1	x
1	0	0	y
1	0	1	y
1	1	0	y
1	1	1	y

b) Truth table



c) Symbol

Figure 4 - A 5-to-1 multiplexer.

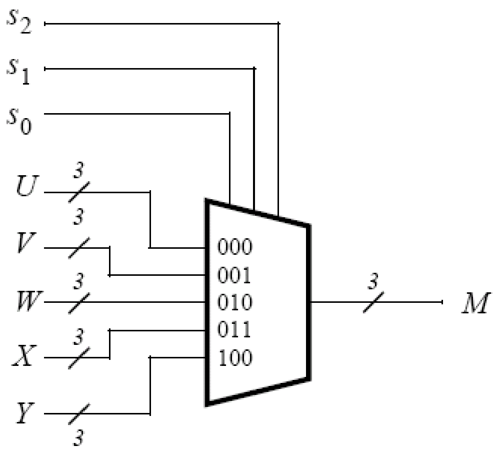


Figure 5 - A three-bit wide 5-to-1 multiplexer.

For this project, report the sequence of steps you followed and describe the circuit you designed. Include in the report the reference to the VHDL source files.

4 – Testing your circuit with a testbench

In all laboratory sessions, you are asked to test the VHDL units on the Altera DE1 board. In fact, this board and the FPGA installed on it represent a testing environment for your design. This is actually a physical ("hardware") environment in which you can really test what is going on with your blocks and measure the output of your circuit. It is mainly used in production, where engineers need a fast plug-and-play platform to test a product in series. This custom collection of devices that support the target system is called ***testbed***.

Sometimes, it is not possible to verify the correctness of a design on a physical board for many reasons: time, complexity, price, possible combinations of the inputs values and so on. Should you be asked to develop a single block of a complex system, i.e. an Arithmetic Logic Unit (ALU), a rapid and flexible way to test it is with some kind of virtual environment, in which you can generate stimuli that stay in the "software" domain, just as your VHDL blocks. Sometimes entering the timing and values specifications for each input is tedious and takes time. To avoid this, it is possible to write a *wrapper* to your design unit that, by using the same features of your development software and the same language, can generate all the inputs to your system required for the verification. In addition, it can collect also the output of your circuit and automatically check if some output is wrong, for example by comparing it to a file. This kind of environment is called ***testbench***.

As described in the *Tutorial: Use of ModelSim* document, a testbench is normally written in the same language as the unit you are developing, in our case VHDL, but for other kinds of systems, for example mixed signal systems, it can also be written in VHDL-AMS, Spice, Verilog-A or even TCL. Imagine you have an 8-bit multiplexer 16-to-1: testing the functionality of the circuit by using switches or buttons would take forever. Instead, by using a VHDL process that sequentially generates, nanosecond after nanosecond, all the possible combinations of the inputs, would save time and would take advantage of the speed of the simulation software. It can be used both for functional and timing simulations.

A testbench must not necessarily be synthesizable, because its aim is to virtually generate the input values to your system, while it should take advantage of all the advanced functions of the language. Generally, a testbench is written as a VHDL entity with no input nor outputs, and it instantiates the components under test (also known as DUT, Device Under Test) in the architecture.

ModelSim is distributed by Mentor Graphics, but there exists also a customized Intel/Altera version that we are going to use. The **ModelSim-Intel FPGA 10.5b** (specific for Quartus Prime 16.1) is available at the web page of Altera. You already have it, by installing Quartus Prime Lite.

For this functional simulation, you do not need any modifications to your code when you decide to create a testbench. You just have to instantiate your design from your top-level hierarchy and start writing a wrapper. Figure 6 shows the conceptual block scheme, an example of VHDL files hierarchy. As you can see, your design is instantiated unmodified while the testbench generates inputs to your design and collects outputs, thus it has no port declaration in its entity. Make sure you do not synthesize the testbench, because it has no physical counterpart!

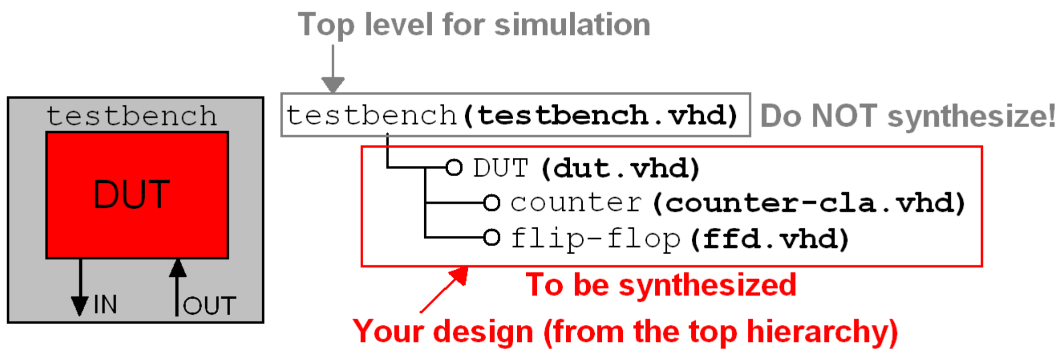


Figure 6 – Testbench hierarchy

Let us focus now on an example, a testbench that verifies the operation of the circuit designed in the previous exercise, the 3bit 5-to-1 multiplexer. For the sake of convenience, create another directory for example *es1tbsim* where you will copy the entity and architecture of your 3bit 5-to-1 multiplexer. If your design has more than one file, copy all the VHDL source of the required sub-unit in that directory. Make sure that for this version you name the multiplexer as b35to1MUX and have a port declaration like:

```
      ENTITY b35to1MUX IS
      PORT (SW: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
            LEDR: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
      END b35to1MUX;
```

This is your DUT. Here all the inputs have been “packed” in the input SW declaration. Since the testbench instantiates the component and is specifically designed for this port map, it is necessary that this is exact. Next times, you will design the testbench and so you will decide the name and the port format of the DUT.

Create a new text file in the same directory with name *testbench.vhd* and cut and paste there the following code. The filename must not necessarily have the same name as the entity it contains, but it is generally recommended to adopt the same name for both entity and file.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- "Empty" entity.
ENTITY testbench_es1 IS
END testbench_es1;

ARCHITECTURE Behavior OF testbench_es1 IS

-- DEVICE UNDER TEST
COMPONENT b35to1MUX
PORT (SW: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
      LEDR: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END COMPONENT;

-- TESTBENCH SIGNALS (WRAPPING UP THE DUT)

SIGNAL U, V, M: STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL S: STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL INPUTS: STD_LOGIC_VECTOR(8 DOWNTO 0);
```

```
BEGIN

-- These signals emulate
-- the switches SW5-0.
-- In this example they are FIXED.
U <= "011";
V <= "100";

-- This emulates selection signals
-- s0, s1 and s2.
-- Control signals are dynamically changed every 20 ns.
-- This PROCESS is kind of a "clock":
-- After the last "WAIT FOR" it loops at the beginning.

-- The instruction WAIT FOR <time> IS
-- ABSOLUTELY NOT SYNTHESIZABLE AND HAS NO
-- PHYSICAL COUNTERPART.
-- IT IS NOT A FLIP-FLOP OR A LOGIC ELEMENT!

PROCESS
BEGIN
  S <= "000";
  WAIT FOR 20 ns;
  S <= "001";
  WAIT FOR 20 ns;
  S <= "010";
  WAIT FOR 20 ns;
  S <= "011";
  WAIT FOR 20 ns;
  S <= "100";
  WAIT;
END PROCESS;

-- This is to align the inputs S...V in the form allowed by the component.
INPUTS <= S & U & V;

-- DEVICE UNDER TEST.
MUX_UT: b35to1MUX PORT MAP (INPUTS, M);
---------------------

-- M is the output of the circuit.

END Behavior;
```

Save the file. This is an example of a possible testbench. As you can see, the testbench is a wrapper that contains one or more component instantiations. The entity has no inputs and no outputs and, in this case, the architecture places the component *b35to1MUX* (red) and generates the input signals (blue). In particular, inputs U and V are fixed, while control signal `S` changes with time. The process with no sensitivity list emulates the sequence “000”, “001”, “010”, “011” and “100” with steps of 20 ns. The sequence is periodic since after the last `WAIT FOR`, the process loops at the beginning. The `WAIT FOR` statement can be very useful to generate a clock. For example, by defining a standard logic signal `clk` in the architecture you can emulate a 50% Duty Cycle clock by using the following code:

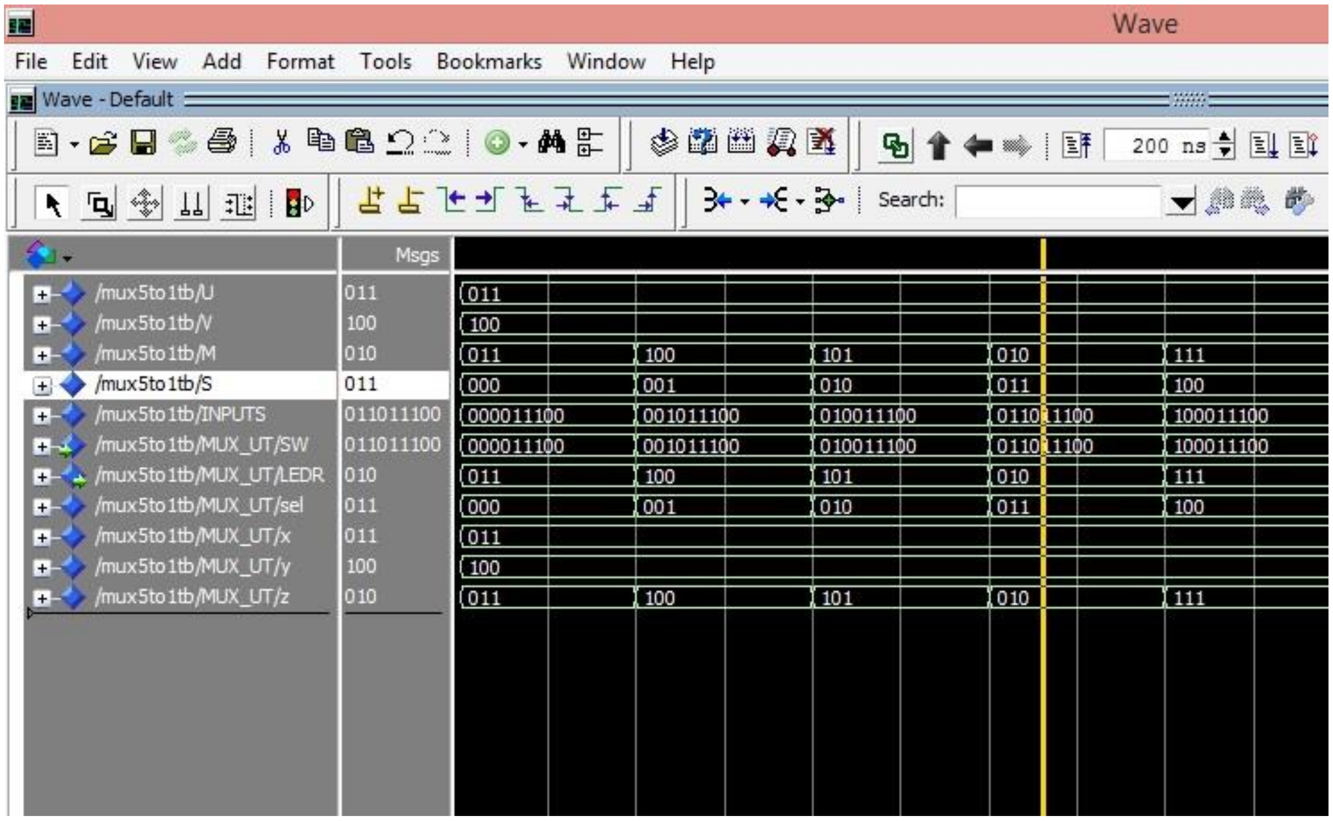
```
      PROCESS
      BEGIN
            CLK <= "0";
            WAIT FOR 1 ns;
            CLK <= "1";
            WAIT FOR 1 ns;
        END PROCESS;
```

Here, the generated clock has 500MHz frequency. Keep it in mind for the next exercises.

By simulating the top-level *testbench_es1*, and selecting all the internal testbench signals U, V and M, it will be possible to run a complete simulation of `MUX_UT` and check if output M corresponds to the correct input selection. It is possible to integrate Quartus Prime and Modelsim-Intel FPGA, but here we will use Modelsim as **a standalone tool**.

Follow the guidelines in the *Tutorial: Use of ModelSim* document to create a new project and simulate the circuit using the given testbench.

By running the simulation for 1µs, you should obtain the waveforms in Figure 7. As you can see, the output of the multiplexer changes every 20ns since its control signal toggles every 20 ns. Inputs U…Y do not change. Output toggles according to the selection bit. Now you can verify if the circuit works correctly or not.
Include in your report the simulation results (e.g. waveform snapshot).



	Msgs	
/mux5to1tb/U	011	011
/mux5to1tb/V	100	100
/mux5to1tb/M	010	011, 100, 101, 010, 111
/mux5to1tb/S	011	000, 001, 010, 011, 100
/mux5to1tb/INPUTS	011011100	000011100, 001011100, 010011100, 011011100, 100011100
/mux5to1tb/MUX_UT/SW	011011100	000011100, 001011100, 010011100, 011011100, 100011100
/mux5to1tb/MUX_UT/LEDR	010	011, 100, 101, 010, 111
/mux5to1tb/MUX_UT/sel	011	000, 001, 010, 011, 100
/mux5to1tb/MUX_UT/x	011	011
/mux5to1tb/MUX_UT/y	100	100
/mux5to1tb/MUX_UT/z	010	011, 100, 101, 010, 111

Figure 7 – Output and inputs after 100 ns simulation.