TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Faculty of Automation and Computer Science Department of Computer Science

COMPUTER ARCHITECTURE

Introduction to Creating a VHDL Test Bench in Xilinx Vivado

COMPUTER ARCHITECTURE

Introduction to Creating a VHDL Test Bench in Xilinx Vivado

February 2021

1 Introduction

Simulation: the imitative representation of the functioning of one system or process by means of the functioning of another (as defined by Merriam-Webster)

When it comes to electronic hardware and *Hardware Description Language* (HDL) models, simulators and *Electronic Design Automation* (EDA) simulation tools are used to verify as to whether the *Register-Transfer Level* (RTL) code meets the functional requirements of a given design's specifications.

To achieve this, the necessary signals must be generated alongside the different states and logical input values resulting in a waveform as an output.

It's possible that you may not be able to properly copy-paste the code from the document, therefore a separate GitHub Gist has been created to circumvent any issues.

The link to the UTCN Computer Architecture Test Bench supplementary code is https://gist.github.com/fuzesa/bd6f463f4a8ce7687975eb3e43da064f

2 Basic Terminology

 $\underline{\text{NOTE}}$: When it comes to Verilog, test benches are sometimes referred to as *test fixtures*. Please be aware that two terms are synonymous in concept.

DUT - Device Under Test / UUT - Unit Under Test: The given model or entity undergoing testing.

Input Stimulus: The input signal being fed into the DUT / UUT.

3 Functional vs timing simulation

Functional simulation refers to the concept of testing the underlying operational behavior of the circuit without any consideration for delays associated with placement or routing.

In reality, these aforementioned delays could result in signals not meeting setup time and undesirable output.

Timing simulation on the other hand takes these factors into account based on the speed grade of the component.

Within Vivado, these delays are not configured through any timing analysis tool and they must be incorporated into the device through its design.

4 Common guidelines

- Test benches are basically a part of your project's hierarchy and it is recommended to have them at the top level.

- Within Xilinx, the instantiated components in test benches are often labeled as UUT (Unit Under Test).

- In most cases, the same test bench should be applicable for both functional and timing simulations.

5 Simulation of a combinational logic circuit

First, we are going to implement a test bench for basic circuit with simple logic gates.

The device has three inputs, two binary for different values and a vector select control of a *multiplexer* (MUX). The output of the device will be the output from the MUX

The inputs to the MUX are going to be the result of $00 \rightarrow$ an AND gate $01 \rightarrow$ an OR gate $10 \rightarrow$ and lastly a XOR gate



Figure 1: Sample combinational circuit

The VHDL code for the schematic above is the following:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity sample_comb_circ is
    Port ( A : in STD_LOGIC;
        B : in STD_LOGIC;
        S : in STD_LOGIC_VECTOR (1 downto 0);
        0 : out STD_LOGIC);
end sample_comb_circ;
architecture Behavioral of sample_comb_circ is
begin
with S select
    0 <= A and B when "00",
        A or B when "01",
        A xor B when "10",
```

'0' when others;

end Behavioral;

In the example above, we have named our entity as sample_comb_circ



Figure 2: A new design source has been added to the project

Next we will add a corresponding test bench to the design

First, left click on the sim_1 (1) folder within Simulation Sources and click on the plus sign (+) above Design Sources sources to bring up the dialog.



Figure 3: Add new source file

Then select $\ensuremath{\texttt{Add}}$ or create simulation sources

	Add Sources ×
	Add Sources
HLx Editions	This guides you through the process of adding and creating sources for your project
	○ Add or <u>c</u> reate constraints
	○ Add or create design sources
	Add or create simulation sources
	Cancel
	< back Wext > Finish Cancel

Figure 4: Add Sources dialog options

Then click on Create File

		Add Sources				
dd or Create Simulation So becify simulation specific HDL files and add it to your project.	or directories conta	ining HDL files, to a	ld to your project	Create a new s	source file on disk	
Specify simulation set: 🕒 sim_	1	~				
+ - + +						
	Use Add Files, Add	d Directories or Crea	te File buttons be	low		
	Add Files	Add Directories	<u>C</u> reate File	•		
Scan and add RTL include files	Add Files	A <u>d</u> d Directories	<u>C</u> reate File	2		
 Scan and add RTL include files Copy sources into project 	Add Files	Add Directories	<u>C</u> reate File	3		
 Scan and add RTL include files Copy sources into project Add sources from subdirectorion 	Add Files	Add Directories	<u>C</u> reate File	3		

Figure 5: Create File from Add Sources dialog

Name the file as ${\tt sample_comb_circ_tb}$ and click on the OK button.

Create Source File ×
Create a new source file and add it to
your project.
Us File type: • VHDL • Iow
File name: sample_comb_circ_tb
File location: File location:
Сапсеі з
es into project
ries

Figure 6: Assign a name to the new file

After the file has been added to the table, click the Finish button.

			Ad	dd Sources	×
Add or (Specify sin and add it	Create S mulation s to your p	Simulation Sources specific HDL files, or directo project.	ries containing H	DL files, to add to your project. Create a new source file	on disk 🗼
Specify	y simulatio	on set: 🕞 sim_1	~		
+,	- +	+			
	Index	Name	Library	Location	
•	1	sample_comb_circ_tb.vhc	xil_defaultlib	<local project="" to=""></local>	
		Add	Files Ad	d Directories	
Scar	n and add	RTL include files into project	:t		
Cop	y <u>s</u> ources	into project			
Add	so <u>u</u> rces f	rom subdirectories			
🗹 Inclu	ude all de	sign s <u>o</u> urces for simulation			
?				< <u>B</u> ack <u>N</u> ext > <u>Finish</u>	Cancel

Figure 7: Finish creating a new file

A new dialog window will pop-up asking you to add **ports** to the *entity*. Simply click on the OK button and when asked to verify, click Yes.

fine a module an r each port specif MSB and LSB val Ports with blank	id spe fied: lues w name	cify I, ill be s will	/O Poi ignor not b	ed unle e writte	dd to y ss its B en.	ure ur source file. s column is checked.	4	Def For I	ine a module each port spe 1SB and LSB Ports with blar	and sp ecified: values nk nam	ecify I will be es wil	I/O Po ignor I not b	rts to a red unle be writt	dd to y ess its i en.	your source file. Bus column is checked.	•
odule Definition Entity name: Architecture na I/O Port Defin	n me:	samp Behar s	le_co vioral	mb_ciro	:_tb		8	Mo	dule Definit Entity name: Architecture I/O Port De	ion ?	Th Ar	e moo e you	Defi dule def sure yo	ne Mo inition ou want	dule × has not been changed. to use these values?	8
Port Name	Directi	on	Bus	MSB	LSB				Port Name	Direc	tion	Bus	MSB	LSB		
i	n	~		0	0					in	~		0	0		
)						ОК	Cancel	?)						ОК	Cancel

Figure 8: Verify that no ports will be added to the newly created simulation file

The new simulation source file should now be included within the Simulation Sources

PROJECT MANAGER - MuxWithTB	
Sources ?	_ 🗆 🖒 ×
	•
✓	
sample_comb_circ(Behavioral) (sample_comb_circ.vhc	1)
✓ ☐ Constraints	
🗁 constrs_1	
✓	
✓	
sample_comb_circ(Behavioral) (sample_comb_circ)	.vhd)
sample_comb_circ_tb(Behavioral) (sample_comb_circ_t	:b.vhd)
> 🚍 Utility Sources	

Figure 9: Newly created file has been added under Simulation Sources

Next we will modify the default vhd file generated from the template.

Clear the entire file (delete all the text), so that it would be clutter free and add the following code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- No port declarations necessary
entity sample_comb_circ_tb is
end sample_comb_circ_tb;
architecture Behavioral of sample_comb_circ_tb is
-- Declare component within the scope of the test bench
COMPONENT sample_comb_circ
PORT(
     A : in STD_LOGIC;
     B : in STD_LOGIC;
     S : in STD_LOGIC_VECTOR (1 downto 0);
     0 : out STD_LOGIC
    );
END COMPONENT;
-- Add the necessary input signals
signal A : STD_LOGIC := '0';
signal B : STD_LOGIC := '0';
signal S : STD_LOGIC_VECTOR(1 downto 0) := "00";
-- Add the necessary output signals
signal 0 : STD_LOGIC;
begin
    -- Instantiate the declared component
    uut: sample_comb_circ PORT MAP (
      A => A,
     B => B,
     S => S,
      0 => 0
    );
    -- Begin combinational logic flow
    stim_proc: process
    begin
      -- hold reset state for 100ns
      wait for 100ns;
      -- insert stimulus here
      -- by default the MUX's selection vector is set to "00"
      -- so the it will out put the result from the AND gate
      A <= '1';
      B <= '1';
      wait for 100ns;
      A <= '1';
      B <= '0';
      wait for 100ns;
```

```
-- Set the MUX to output the result from the OR gate
      S <= "01";
      A <= '0';
      B <= '1';
      wait for 100ns;
      B <= '0';
      wait for 100ns;
      -- and lastly the XOR gate
      S <= "10";
      A <= '0';
      B <= '1';
      wait for 100ns;
      A <= '1';
      B <= '0';
      wait for 100ns;
      A <= '1';
      B <= '1';
      wait;
    end process;
end Behavioral;
```

Once you have saved the file, observe that the instantiated component now appears within the test bench file.

PROJECT MANAGER - MuxWithTB	
Sources ? _ D 🛙 X	Project Summary × sample_comb_circ.vhd × sample_comb_circ_tb.vhd ×
$\mathbf{Q} \mid \mathbf{X} \mid \mathbf{a} \mid \mathbf{+} \mid \mathbf{B} \mid 0 = 0 \qquad \mathbf{a}$	/home/atis/Projects/Xilinx/Vivado/MuxWithTB/MuxWithTB.srcs/sim_1/new/sample_comb_circ_tb.vhd
✓	
sample_comb_circ(Behavioral) (sample_comb_circ.vhd)	
✓ General Constraints	13; A : in STD_LOGIC; 14: B : in STD_LOGIC:
🕞 constrs 1	15 S : in STD_LOGIC_VECTOR (1 downto 0);
Simulation Sources (1)	16 0 : out STD_LOGIC
	18 C END COMPONENT;
∨ ⊆ sim_1 (1)	19
sample_comb_circ_tb(Behavioral) (sample_comb_circ_tb.vhd) (20 Add the necessary input signals 21 - signal A : STD LOGIC := '0';
uut : sample_comb_circ(Behavioral) (sample_comb_circ.vhd)	22 signal B : STD LOGIC := '0';
> 🗅 Utility Sources	<pre>23 signal S : STD_LOGIC_VECTOR(1 downto 0) := "00";</pre>
	25 Add the necessary output signals
	26 signal 0 : STD_LOGIC;
	27
	28 Degin 29
	30 Instantiate the declared component
	31 uut: sample_comb_circ PORT MAP (
	$32 + A \Rightarrow A$, $32 + B \Rightarrow B$
	34 s $>$ s,
	35 0 => 0
	36⊖); 37
	38 Begin combinational logic flow
	39 🗇 stim_proc: process
	40 begin 41 see hold reset state for 100ps
<u><</u> >>	42 wait for 100ns;
Hierarchy Libraries Compile Order	43
	44 Insert stimulus here 45
Source File Properties ? _ O 🖸 X	46 by default the MUX's selection vector is set to "00"
	47 so the it will out put the result from the AND gate
sample_comb_circ_tb.vhd	46 , A <= '1'; 49 ' B <= '1';
^	50 wait for 100ns;
C Enabled	51 ; A <= '1';
Location: /home/atis/Projects/Xilinx/Vivado/MuxWithTB/MuxWithTB.sr	53 Wait for 100ns:

Figure 10: Component has been instantiated within the simulation file

Now that we have the simulation file ready, let's run it using Vivado's built-in simulator. Within the Flow Navigator on the left side, expand SIMULATION and click on Run Behavioral Simulation.



Figure 11: Click on Run Behavioral Simulation

After the simulation is complete, you will be taken to the resulting view where you can observe the signals that you have described within your test bench. Since the default time scale isn't for the duration of the entire simulation, click on the Zoom Fit button (highlighted in red) to get a better view

s <u>W</u> indow Layout View	<u>Run</u> <u>H</u> elp	Q- Quick Ac	cess	II C								
SIMULATION - Behavioral Sim	ulation - Functiona	al - sim_1 -	sample_comb	_circ_tb								
Scope × Sources	_ 0 6	Objects	× Protoc	ol Ins ? _ 🗆 🖾	sample_	comb_circ	vhd ×	sample_c	omb_circ_tb.	vhd × Untitled	1 ×	
0, ₹ ⊕	•	Q		۰	Q 🖬	ଇ ର	20 • F	$\mathbf{H} \models \mathbf{H}$	12 2r +			
Name De Blo	^	Name	Value Dat	^			Zoom F	it				
V 🗃 s; sample VHDL E		18 A :	1 Logic		Name	Value	999,980 p		999,982 ps	999,984 ps	999,986 pi	s 999,988 ps
a sample VHDLE		IS B	1 Logic		16 A	1						
		14 0	2 Array		1ê B	1						
		80	o Logic		> ⊌S[1:	2						
					18 O	0						

Figure 12: Press Zoom Fit to get a better view of the signals

Observe the signals to verify that the simulation has yielded the desired results.



Figure 13: Verify the behavior of the component based on the results

6 Simulation of a sequential logic circuit

In this example, we are going to detail how to write a test bench for a circuit with a basic clock and without any additional delays included in the clock mechanism.



Figure 14: Sample sequential circuit

This is simple counter with a synchronous reset and the capability to load a pre-defined value into it. It has three inputs, one for the *clock* signal, one for the *reset*, and another two for the *load*ing a value. The code for this schematic is as follows:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.STD_LOGIC_UNSIGNED.all;
entity sample_counter is
    PORT (
          clk
                : in
                      STD_LOGIC;
          reset : in
                      STD_LOGIC;
          load : in
                      STD_LOGIC;
          data : in STD_LOGIC_VECTOR (3 downto 0);
          count : out STD_LOGIC_VECTOR (3 downto 0)
         );
end sample_counter;
architecture Behavioral of sample_counter is
signal s_count: STD_LOGIC_VECTOR (3 downto 0);
begin
    process(clk)
    begin
       if rising_edge(clk) then
          if reset = '1' then
             s_count <= (others => '0');
          elsif load = '1' then
```

```
s_count <= data;
else
s_count <= s_count + '1';
end if;
end if;
end process;
count <= s_count;
end Behavioral;
```

Now, as for the test bench, here we need to pay attention to two certain details.

We would like the behavior of the clock to be independent of the rest of the functionality. One advantage of VHDL, is that anything that's *defined* as a basic *process* gets executed in parallel.

Therefore, we simply need to define a separate process for the clock and another one for the stimulus, just like in the example code.

Another important thing to keep in mind is also that **one control signal** should be changed only in **one process**, in other words avoid assigning values to signals in 2 or more processes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity sample_counter_tb is
end sample_counter_tb;
architecture Behavioral of sample_counter_tb is
-- Add constants
constant T : time := 50 ns:
-- Declare component within the scope of the test bench
COMPONENT sample_counter
   PORT (
          clk : in STD_LOGIC;
         reset : in STD_LOGIC;
         load : in STD_LOGIC;
          data : in STD_LOGIC_VECTOR (3 downto 0);
          count : out STD_LOGIC_VECTOR (3 downto 0)
         );
END COMPONENT;
-- Add the necessary input signals
signal clk : STD_LOGIC := '0';
signal reset : STD_LOGIC := '0';
signal load : STD_LOGIC := '0';
signal data : STD_LOGIC_VECTOR (3 downto 0) := X"D";
-- Add the necessary output signals
signal count : STD_LOGIC_VECTOR (3 downto 0);
begin
    -- Instantiate the declared component
    uut : sample_counter PORT MAP (
                                   clk => clk,
```

reset => reset,

```
load => load,
                                     data => data,
                                     count => count
                                    );
    -- continuous clock
    clk_proc: process
    begin
        clk <= '0';
        wait for T/2;
        clk <= '1';
        wait for T/2;
    end process;
    -- stimuli
    stim_proc: process
    begin
        -- hold reset for one clock cycle
        reset <= '1';</pre>
        wait for T;
        reset <= '0';</pre>
        -- initialize the value for load
        load <= '0';
        -- wait for four cycles and then reset again
        wait for 4 * T;
        reset <= '1';</pre>
        wait for T;
        reset <= '0';
        -- wait for six cycles and then load the value in the data vector
        wait for 6 * T;
        load <= '1';</pre>
        wait for T;
        load <= '0';
        wait;
    end process;
end Behavioral;
```

After running the simulation, make sure to verify the desired output.



Figure 15: Simulation result

In Vivado, the **default simulation time** is set to **1000ns**.

We are able to change that within the simulation settings.

In order to do that, first right-click on Run Simulation which can be found under Simulation within the Flow Navigator.

		₩ <u>∠</u> ℤ % №
Flow Navigator		PROJECT MANAGER - CntrWithL
✓ PROJECT MANAG	ER	C
Settings		Sources
Add Sourcos		$ Q \stackrel{\star}{\underset{\star}{\overset{\star}{\overset{\star}}}} \Leftrightarrow + @ \bullet 0$
Add Sources		✓
Language Temp	lates	sample_counter(Bell
👎 IP Catalog		> 🗁 Constraints
		\sim 🗁 Simulation Sources (1)
✓ IP INTEGRATOR		∨ 🖨 sim_1 (1)
Create Block De	esign	✓ ● ∴ sample_counter
Open Block Des	ian	uut : sample_cou
open block bes	ign	> 🚍 Utility Sources
Generate Block	Design	
✓ SIMULATION		
Run Simulation	Circulation Calific	
	Simulation Setting	JS
✓ RTL ANALYSIS	Reset Behavioral S	Simulation
> Open Elab	Reset Post-Synthe	esis Functional Simulation
	Reset Post-Synthe	esis Timing Simulation
✓ SYNTHESIS	Reset Post-Impler	nentation Functional Simulation
Run Synthe	Reset Post-Impler	nentation Timing Simulation
> Open Synthesiz	ed Desian	

Figure 16: Right-click on Run Simulation

Then within the new dialog window

- (1) First click on Simulation within the **Project Settings** area
- (2) then click on the **Simulation** tab
- (3) lastly, modify the value for the **xsim.simulate.runtime** key

Ÿ	Simulation
roject Settings	Specify various settings associated to Simulation
General	
Simulation	Target simulator: Vivado Simulator 🗸
Elaboration	Simulator language: Mixed 🗸
Synthesis	Simulation set:
Implementation	
Bitstream	Simulation top module name: sample_counter_tb
IP	Generate simulation scripts only
ool Settings	∠
Project	Compilation Elaboration Simulation Netlist Adv. ↔ ≡
IP Defaults	
XHub Store	xsim.simulate.tcl.post
Source File	xsim.simulate.runtime 1000ns 5
Jisplay	xsim.simulate.log_all_sig
VebTalk	xsim.simulate.no_quit
Help	xsim.simulate.custom_tcl
Text Editor	xsim.simulate.wdb
3rd Party Simulators	xsim.simulate.saif_scope
Colors	xsim.simulate.saif
Selection Rules	xsim.simulate.saif_all_sig
Shortcuts	xsim.simulate.add_positi
Strategies	xsim.simulate.xsim.more 🗸 🗸
Remote Hosts Window Behavior	xsim.simulate.runtime
	Specify simulation run time

Figure 17: Steps to modify the simulation time

When you run the behavioral simulation like previously, you can make sure that now the simulation time corresponds to the value that has just been set.



Figure 18: Simulation with the new run time value