Australian National University

Lab One

Getting started with digital design: Introduction to Vivado

Overview

In this course we will be using the FPGA environment as a target to prototype, test and troubleshoot our digital designs. The digital design process has a typical set of steps, from concept to formulation, through to hardware implementation. To assist us with these tasks, we will rely on an environment called Vivado.

In this lab, you will learn to:

- Create projects and design a simple digital system in Vivado.
- Review the RTL structure of a design and simulate its logic behaviour, including assessing the potential impact of transition delays on system performance.

You will also start becoming familiar with:

- Hardware description languages (HDL), specifically Verilog.
- Simple combinatorial logic circuits such as a multiplexer and a 1-bit full adder.
- The concept of testbenches and logic simulation.
- Vivado's simulation environment, ISim.

Australian National University

Table of Contents

Lab Overview

Learning Outcomes

By completing this lab, you will learn to:

1. Create projects and handle files in Vivado.
2. Create design sources and describe simple combinatorial logic circuits in Verilog.
3. Know the essential building blocks of a simulation testbench.
4. Simulate combinatorial designs, including assessing the impact of time delays.

To complete this lab, you need access to the Vivado Design Suite running on Windows 10 or a supported Linux operating system.

Activity 1: Creating your first digital design: Multiplexer [4 marks – approximately 45 minutes]

Background

A multiplexer (a.k.a. MUX) is an electronic switch that connects one of several inputs to the output based on selection signal *S*. Figure 1 describes the simple case of the 1-bit MUX.

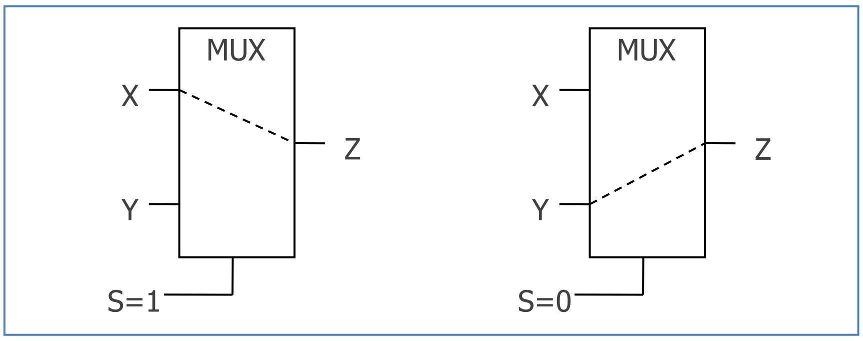


Figure 1: A 1-bit MUX.

The truth table of the 1-bit MUX is shown in Table 1. The 'xx' notation indicates a "don't care" condition. Note that we consider the MUX here as a combinational circuit.

X	Y	S	Z
0	xx	1	0
1	xx	1	1
xx	0	0	0
xx	1	0	1

Table 1 : Truth table of the 1-bit MUX.

The truth table allows us to derive the logic representation of the system. The table can be immediately translated into a Boolean expression, which can in turn be rendered into a schematic or Verilog HDL code.

Next, we derive a minimal Boolean logic representation of the MUX using a Karnaugh Map. Table 2 shows the Karnaugh Map of the 1-bit MUX (notation “X” means X=1).

X Y

XY / S	00	10	11	01
0	0	0	1	1
1	0	1	1	0

Table 2: Karnaugh Map of the 1-bit MUX.

From table 2, we can derive the Boolean logic expression

$$Z = (X \cdot S) + (Y \cdot S') \qquad (1)$$

where $S' = NOT(S)$. The corresponding schematic diagram is shown in Figure 2.

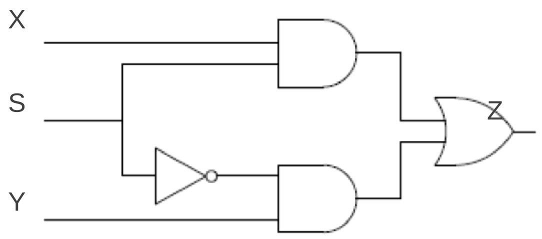


Figure 2: Schematic description of the 1-bit MUX.

This schematic description can be readily implemented in Vivado. **Although the schematic description involves a visually small number of building blocks or logic gates, implementation in the configurable logic of the FPGA may be quite complex and involve thousands or millions of CMOS transistors** (not in the case of this simple MUX, though. This is a very simple circuit).

Procedure

Step One: Create a project in Vivado

1. Launch Vivado and create a new project.

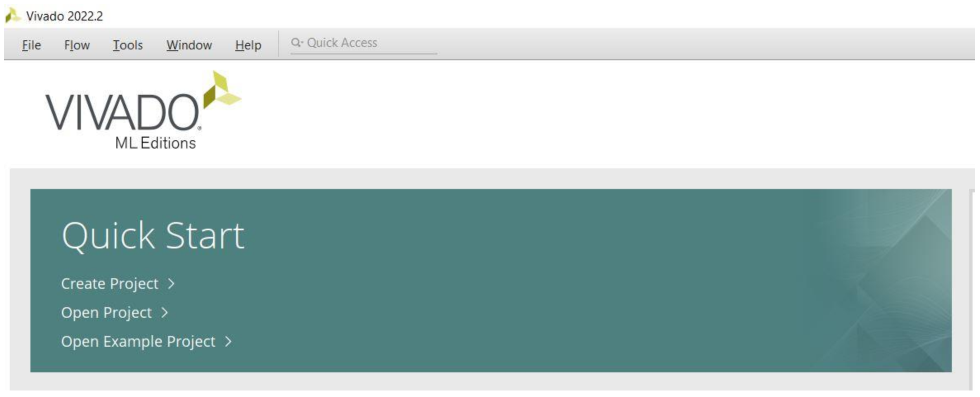


Figure 3: Vivado's Quick Start menu.

Name your project: 'my_mux_project'. Usually, the name should only contain letters, numbers and underscores. **Please avoid using white space** and other characters.

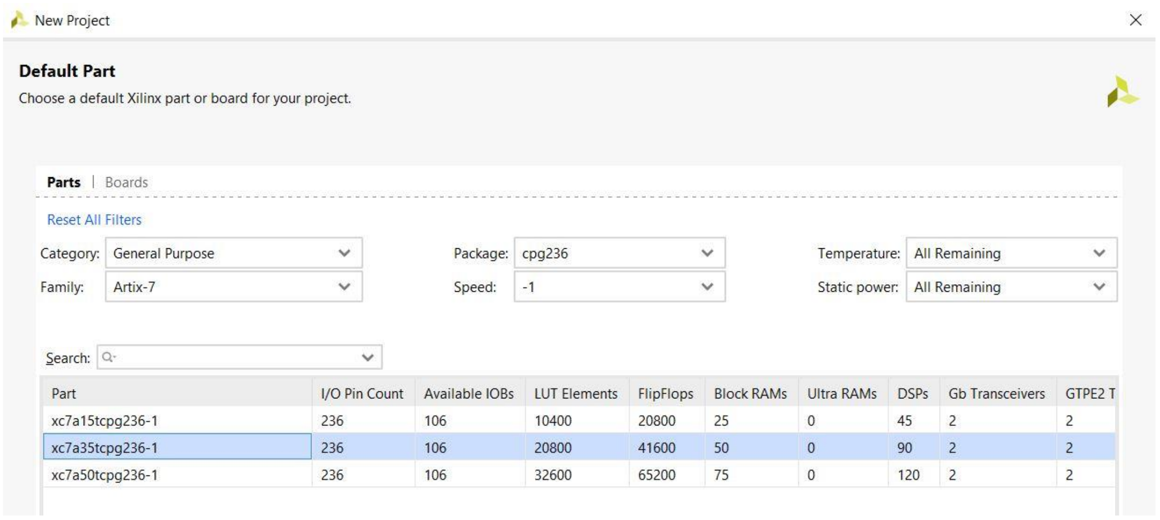
2. Select the directory in which to store the project's files. It is recommended to create a dedicated FPGA development directory on your computer to organise the files. **Please avoid using OneDrive.** If you are using the lab computer, you can save your files locally and then use an USB stick to transfer them to your laptop. **Please avoid using white space in the directory.** It is recommended to use only letters, numbers and underscores.
3. **Make sure the "create subdirectory for your project files" box is checked.** Then click Next.
4. Select your Project Type as RTL Project.

   There is also a check box that says: "Do not specify sources at this time". You don't currently have any source files to specify, so check this box and click Next.
5. You may be asked to select the default file type and language for your project's source files. This is set to Verilog by default, so you don't need to change anything.
6. Specify the Default Part. This refers to the specific FPGA chip you intend to ultimately use with your design, which will be different depending on the specific development board you are using. **This year, all students should be using Basys3.**

   You can search for your part number directly using the search bar or narrow the list of available options by selecting the appropriate options for your development board.

Development Board	**Basys3**
FPGA Family	Artix-7
Package	CPG236
Speed grade	-1
Temperature grade	All Remaining
Part number	xc7a35tcpg236-1

Caution: It is crucial that you select the correct part number for your project. If you select the wrong part, your FPGA design will not successfully deploy to the chip.



New Project

Default Part

Choose a default Xilinx part or board for your project.

Category: General Purpose
Family: Artix-7
Package: cpg236
Speed: -1
Temperature: All Remaining
Static power: All Remaining
Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 T
xc7a15tcpg236-1	236	106	10400	20800	25	0	45	2	2
xc7a35tcpg236-1	236	106	20800	41600	50	0	90	2	2
xc7a50tcpg236-1	236	106	32600	65200	75	0	120	2	2

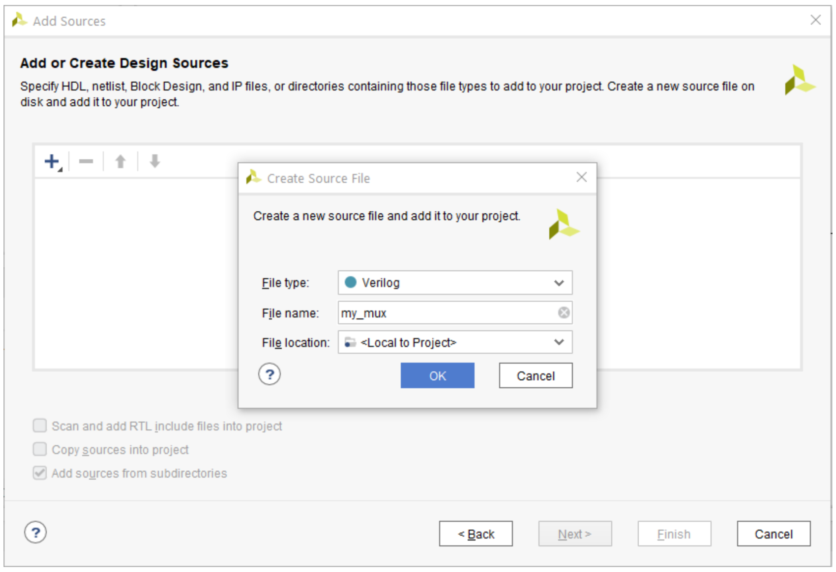
Figure 4: Default Part selection window highlighting the FPGA chip for Basys3.

7. When you've selected your Default Part from the list, click Next.
8. The final step in creating a new project is reviewing the information you've specified in the previous steps. Verify the information and click Finish.

   The Vivado Integrated Design Environment (IDE) will now appear.

Step Two: Create a Verilog design source

1. At the top left of the Vivado window, click on the 'Add Sources' button.
2. You will be presented with three options: to add or create constraints, design sources, or simulation sources. Select design sources and click Next.
3. There are three options for you to choose from: add files, add directory, and create file. Select the create file option.
4. Select the file type as Verilog; name the file as 'my_mux'; set the file location to <Local to Project>, which is the default option. **Please use only letters, numbers and underscores in the file name.**


Add Sources

Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Create Source File

Create a new source file and add it to your project.

File type: Verilog

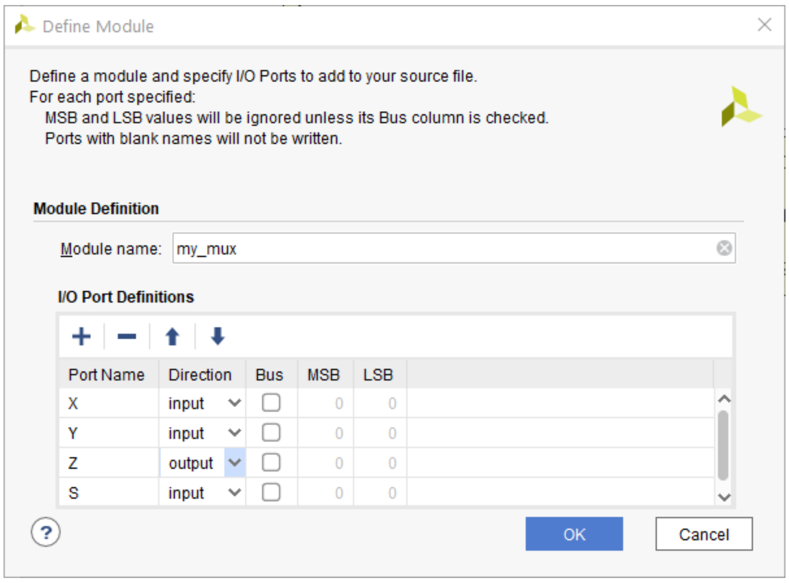
File name: my_mux

File location: <Local to Project>

OK
Cancel
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
< Back
Next >
Finish
Cancel

Figure 5: Creating a new design source file in Vivado.

5. Vivado gives you the option to define your module's I/O properties. You can add and remove input and output ports to your module and specify their bus widths if applicable (see Figure 6). **You can skip this step** because in bigger projects, it is difficult to determine all I/O at the beginning. Simply Click OK to skip through.


Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: my_mux

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
X	input	☐	0	0
Y	input	☐	0	0
Z	output	☐	0	0
S	input	☐	0	0

Figure 6: The option of defining module's I/O when creating a new design source.

6. You will now be returned to the Vivado IDE. The file you just created should appear in the 'Design Sources' directory in the Sources pane at the top left of the screen. Double-click on the file 'my_mux.v' to open it.

7. This module needs to implement the logic function that was derived from the Karnaugh map. Declare the port entities as 'wires' and write the `assign` statement. **The keyword "assign" describes continuous circuit connections.**

```
`timescale 1ns / 1ps
```

```

module my_mux(
    input wire X,
    input wire Y,
    output wire Z,
    input wire S
    );

assign Z = (X & S) | (Y & (~S));

endmodule
```

Compare the logic statement with the Karnaugh map. What we have effectively described here in Verilog is a physical connection between logic gates.

The first line`timescale 1ns / 1ps is only relevant for simulations.

Note that it is a good habit to **make sure the module name is the same as the file name**, e.g., the Verilog file 'my_mux.v' should have the code "module my_mux(…)" in it.

Step Three: Visualising your design

You can review the way in which Vivado interprets your code as a system of interconnected basic constructs (gates, registers, etc). This style of system visualisation is called RTL (register-transfer level) framework.

1. Look for "RTL ANALYSIS" on the left-hand pane of the main Vivado window and click "Open Elaborated Design".
2. After some processing (which may fail if you have outstanding errors in your code), you will be presented with a schematic description of your design, as shown in the next figure. Check if the circuit matches the original design intent. (If you make a change to your code, you need to reload 'Elaborated Design' to see the changes in the schematic).

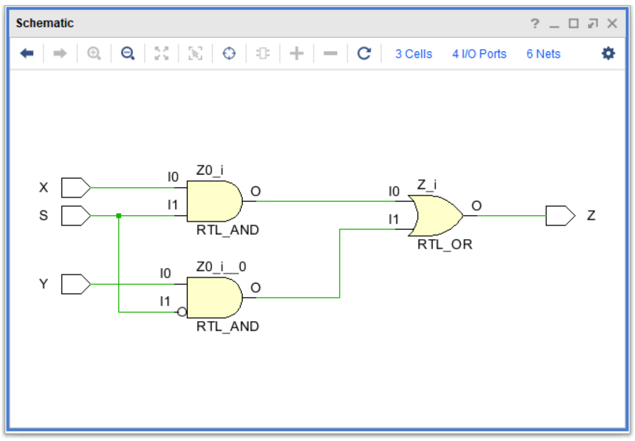


Figure 7: RTL view of the mux design.

Note: You are not required to view the RTL schematic every time, but **you should be aware that this feature exists as it can serve a useful troubleshooting tool.**

Step Four: Simulating your design

Now that you are satisfied that Vivado has interpreted your system in a way that is consistent with your expectations, it is time to simulate operation of the MUX and verify that all is working as it should. To do this we will need a *test bench*. A test bench is a special type of Verilog code file that describes a virtual operating environment for the design under test. The test bench file specifies what test inputs should be fed to your design, and what output signals should be read and analysed. Simulations are run using the internal Vivado Simulator, which is called ISim.

Note: Test bench files are only used for simulation and **have no role in the final hardware implementation.**

1. Right click on *Simulation Sources* and select *Add Sources.*

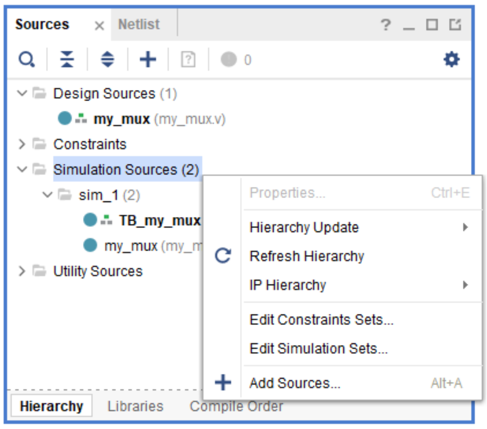
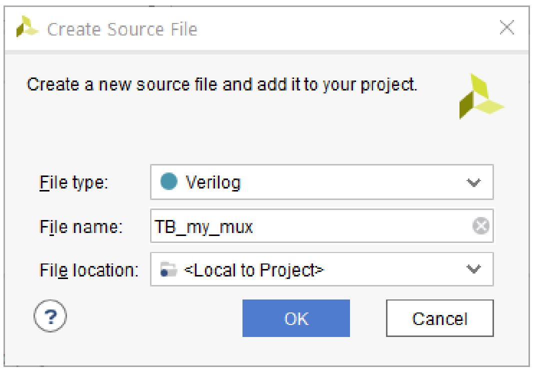


Figure 8: Adding a simulation source.

2. In the window that appears, select *Add or create simulation source*, then click *Next*
3. In the following window, click *Create File* to create a new Verilog source.



Create Source File

Create a new source file and add it to your project.

File type: Verilog

File name: TB_my_mux

File location: <Local to Project>

Figure 9: Naming the simulation source.

4. In the next window, do not assign any inputs and outputs to the new module. Just click through and proceed to the text editor interface
5. Replace the contents of the test bench file with the code on the next page. Replace the 'mux' (highlighted in red) with the correct name of your multiplexer module. This is an example of *instantiation*, where a higher level module (the testbench here), references a lower level module (my_mux.v). **Names must be consistent** to ensure correct linking.

```
 `timescale 1ns / 1ps // timestep duration and time precision

module TB_my_mux;

// Inputs to Unit Under Test (UUT)
   reg X;
   reg S;
   reg Y;

// Output to UUT
   wire Z;

// Instantiate the design to be tested - ensure module name match!
   mux UUT(    //NOTE: the instance name ('UUT' here) can be anything you like
       .X(X),
       .S(S),
       .Y(Y),
       .Z(Z)
   );

// Duration of simulation. Here set at 300 timesteps
   initial begin: stopat
    #300; $finish;
   end

// Signal changes: #xx means enact change after xx timesteps
   initial begin
    X = 1'b0;
    Y = 1'b1;
    S = 1'b1;

    #30X = 1'b1;
    #30S = 1'b0;
    #30Y = 1'b0;

    #30X = 1'b0;
    #30Y = 1'b1;
    #30S = 1'b1;

    #30X = 1'b1;
    #30Y = 1'b0;
    #30S = 1'b0;
   end
endmodule
```

6. Look for the SIMULATION heading in the navigation pane of the main Vivado window, then click on *Run Simulation → Run Behavioural Simulation.*

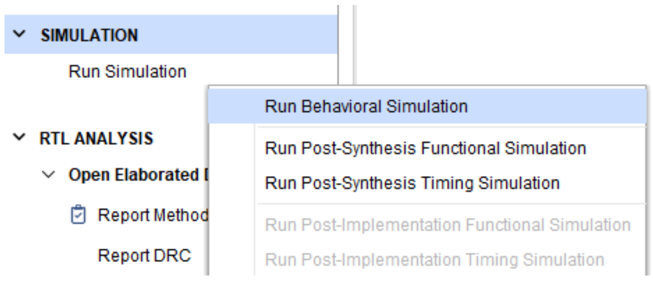

SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated I
Report Method
Report DRC
Run Behavioral Simulation
Run Post-Synthesis Functional Simulation
Run Post-Synthesis Timing Simulation
Run Post-Implementation Functional Simulation
Run Post-Implementation Timing Simulation

Figure 10: Starting the simulation.

7. Once the computations are complete, the simulator window will open showing the simulation results. Under a tab called *Untitled*, you will see the simulated traces for all signals in the

design. Use the *zoom fit* tool to view the whole simulation in one screen. Compare the results with the expected behaviour (i.e., the truth table for the multiplexer). Below is a screenshot of the results

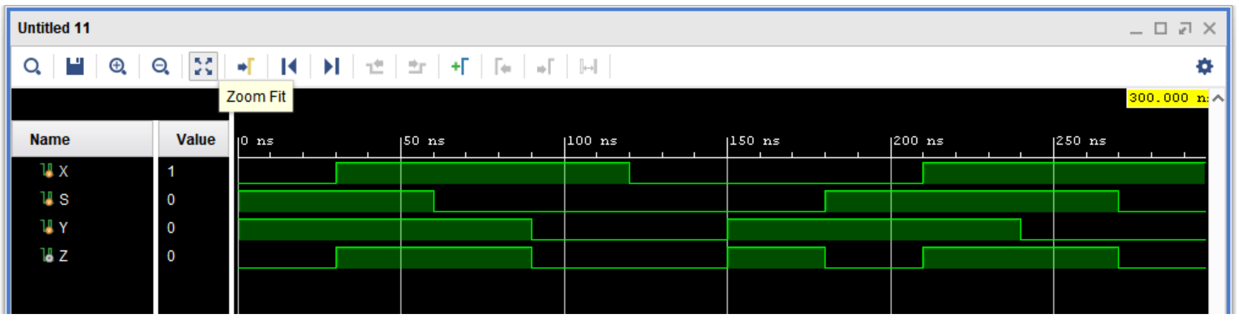


Figure 11: Simulator window showing the simulation results. Note the zoom fit button.

8. Spend a few minutes playing around with the simulator display controls. There are buttons to place markers, zoom in/out and inspect transitions. When you think you are comfortable with the various features, **quickly show the tutor** that you have completed the activity.
9. (Optional: You can change the time steps, and values of X, Y and S. ‘Relaunch Simulation’ (Under ‘Run’ in the drop down menu) to see the difference.

Well done! You have just designed and simulated your first digital design for the course.

Activity 2: Designing a 1-bit Full Adder
[8 marks, approximately 45 minutes]

Background

A full adder is an adder with carry-in and carry-out ports. The truth table of a 1-bit full adder is given in Table 3.

X	Y	C_in	Z	C_out
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Table 3: Truth table of the 1-bit full adder.

Procedure

Your task involves completing the following steps:

1. Derive Boolean expressions for the outputs Z and C_out. Consider XOR gates for Z.
2. Design a module for the 1-bit full adder based on the derived Boolean expressions. Follow the process you have used in the previous section. Use "my_adder" as the name of the design source file.
3. Simulate your module. You may import "TB_Test_Add_1.v" (supplied as a lab resource) as a testbench module. Add a simulation source as done previously (Figure 8), then select *Add or Create Simulation Sources.* Find the option to *Add file* to import the testbench file.

   Make sure the option "Copy sources into project" is checked.

4. When ready, run the behavioural simulation. Your result should be similar to this:

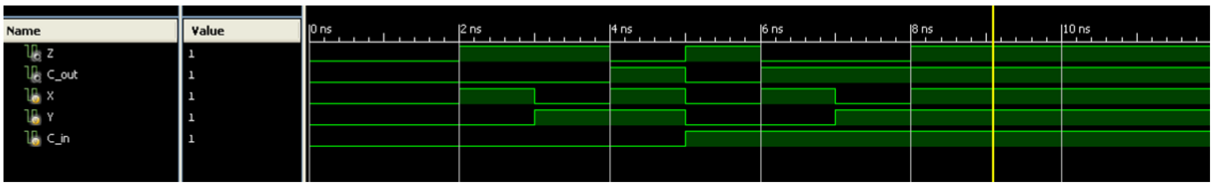


Figure 12: Expected simulation results for the 1-bit adder.

5. Show your work to the lab demonstrator to have your marks assigned.

Activity 3: Timing hazards in a multiplexer [8 marks, approximately 1 hour]

Background

Simulation as we have done it so far treats the circuits as ideal, i.e., with instantaneous signal changes. However, transition times in digital circuits are nonzero and this can have performance implications, which need to be considered. When designing **asynchronous digital circuits** (i.e., circuits where information transfer is not dictated by a timekeeping signal such as a clock – note: all combinational circuits are asynchronous), attention must be paid to the possible presence of hazards. For a 5-minute extension read on hazards, please go through this site.

Consider again the MUX we obtained in Activity 1. Some extra signal names have been added in the figure below.

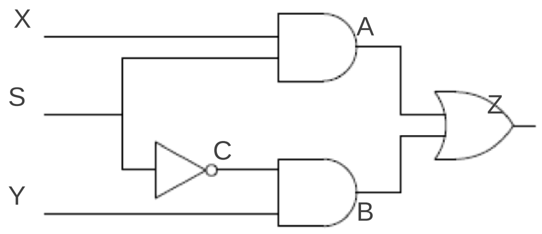


Figure 13: Schematic description of the 1-bit MUX, with intermediate signal names.

By inspection of the circuit, it is visually apparent that not all signal pathways cross the same number of gates, which means they will be affected by varying amounts of delay. Such a condition has the potential to create timing hazards, i.e., output glitches in particular operating conditions.

The Karnaugh map of the circuit (refer back to Table 2) can also confirm the likely presence of hazards. This circuit is implemented as a sum-of-products. *1-hazards can exist where any two adjacent 1s in the table are not covered by the same prime implicant loop*. This is indeed the case for our MUX.

So, is this going to be an issue? You are about to find out.

Procedure

1. Create a new Vivado project called "mux_with_delays".
2. Create a new Verilog source. The module will have inputs X Y and S and output Z. The logic formulation will be the same as you have used in Activity 1. However, the Verilog code will look a little different this time.

```verilog
 `timescale 1ns / 1ps

module my_mux(
    input wire X,
    input wire Y,
    output wire Z,
    input wire S
    );

wire A, B, C;
assign #5 C = (~S);
assign #5 A = X & S;
assign #5 B = Y & C;
assign #5 Z = A | B;
endmodule
```

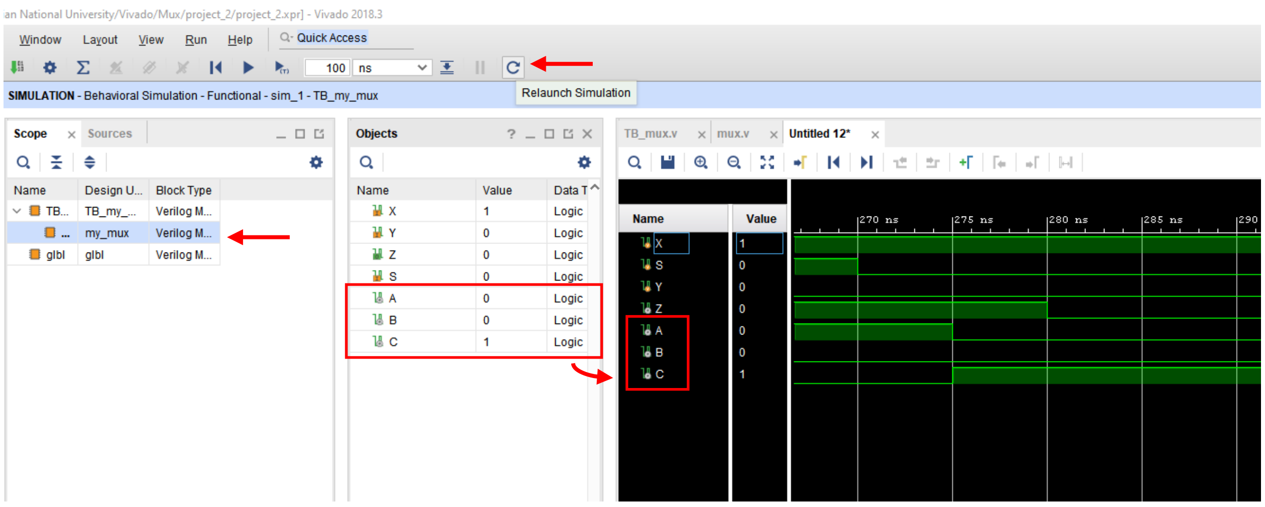
Of note, the above code declares some extra `wire` entities for the purpose of describing intermediate signals. Furthermore, combinatorial `assign` statements have a *#n* parameter specified, which indicates the delay associated with that operation (*n* multiples of the specified timescale). Here, it is assumed that all gates have rise and fall times of 5 ns. Incidentally, it is possible to specify different rise and fall times if desired with the *#(n_rise,n_fall)* syntax.

Important: **Delay parameters are only relevant in simulation.** Once a design is implemented in hardware, actual signal delays will be determined by the physics of the microchip and cannot be set programmatically.

3. Create the test bench module as a simulation source. To be quick, you can import the TB file you created in Activity 1. Be mindful of matching the module names across the test bench file and the actual module source file to avoid simulation errors.

4. Run a simulation and review the results. Identify the hazard. Under what change of inputs does it occur? How long does it last?

5. Let's dig a little deeper. In the Scope window, click on the UUT *my_mux* object. In the Objects window, you will see a list of all the module signals including internal signals. Drag and drop A, B, and C to the simulation window. Then re-launch the simulation using the loop button (see Figure 14) to compute results for all signals. Review the behaviour of the internal signals and make sure that you have understood what sequence of changes is causing the hazard. This simulation approach (i.e., visualising additional internal signals) is a very good troubleshooting approach for complex designs, so make sure you remember how to do it for the future.


an National University/Vivado/Mux/project_2/project_2.xpr] - Vivado 2018.3
Window
Layout
View
Run
Help
Quick Access
100
ns
Relaunch Simulation
SIMULATION - Behavioral Simulation - Functional - sim_1 - TB_my_mux
Scope
Sources
Objects
TB_mux.v
mux.v
Untitled 12*
270 ns
275 ns
280 ns
285 ns
290

Name	Design U...	Block Type
TB...	TB_my_...	Verilog M...
...	my_mux	Verilog M...
glbl	glbl	Verilog M...

Name	Value	Data T
X	1	Logic
Y	0	Logic
Z	0	Logic
S	0	Logic
A	0	Logic
B	0	Logic
C	1	Logic

Name	Value
X	1
S	0
Y	0
Z	0
A	0
B	0
C	1

Figure 14: Simulation window. The Highlights show the list of additional internal signals (which can be dragged and dropped into the waveform display pane on the right) and the "relaunch simulation" button.

6. Assess the total transition time for the mux (i.e., the time it takes for a change in input to result in a stable output) for different signal transition events. Considering the gate delays and hazards, what is the maximum data rate (signal changes per second) that such a system can process successfully?
7. On the Karnaugh map, name the additional prime implicant needed to resolve the hazard condition:

Y

X

XY / S	00	10	11	01
0	0	0	1	1
1	0	1	1	0

Table 4: Karnaugh Map of the 1-bit MUX with additional prime implicant.

8. Determine the new SOP logic expression for the mux after the addition of the new prime implicant. Sketch the new gate structure and modify the Verilog source accordingly. Ensure that you assign a 5ns delay to the new AND gate you will be adding to the structure.
9. Run a new behavioural simulation and verify that the hazard has now disappeared. Review the maximum operating data rate under these new conditions and discover that… it has NOT changed.

   Comment: The unchanged data rate performance result may seem confusing at first, but does make sense upon greater scrutiny. The concept of "maximum data rate" of a system implies changing an input and then capturing the output after enough time for transients to have exhausted. Glitches from hazards occur as part of transient behaviour and so would also be exhausted within this time, hence the unchanged result.

But wait, there is more!

a. The concept of periodic sampling is more appropriate for synchronous circuits, which only respond to signals at regular time intervals. In a truly asynchronous design, hazards are always bad, since asynchronous systems respond immediately to any signal changes, no matter how short-lived they may be.

b. In practice, the addition of extra logic to remove the hazard would likely slow down a circuit and can introduce new hazards.

10. Show your work to the lab demonstrator to have your marks assigned.

End of Lab One