CSSE4010 A1

David Gaul

s4671313

Lab Session: Wednesday 10-12

Submission Date: 7th August

1 Introduction

There were two main goals to the first practical. The first of these goals was to familiarise the students with the CSSE4010 work environment. Students were required to download Vivado and install the necessary cable drivers. Basic introductory programs were then taught to the student so that they could familiarise themselves with the unique VHDL syntax.

Which is a pretty good syntax. Very Python-y.

The second task, on which this report is based, was to develop a self-checking test bed to automatically and exhaustively check the functionality of the AND2OR design: a simple connection of two AND gates connecting to an OR gate (Diagram schematics will be provided within the design description). Test cases had to be developed in three separate forms: structural, behavioural and flow, and tested individually. Lastly, propagation delay had to be integrated into the simulation to emulate the delays that would be experienced in real life due to wires and computing time.

This report will detail the design process for developing this test bed. It will include a design description, where the methodology is explained, results of the successful simulation, synthesis and implementation before a final discussion of results and conclusion.

Let's get cracking.

2 Design Description

As detailed within the introduction, a test bed had to be created to check the functionality of the AND2OR program. This featured a series combination of two AND gates connecting to an output OR gate. The block diagram representation of this can be found in fig. 1

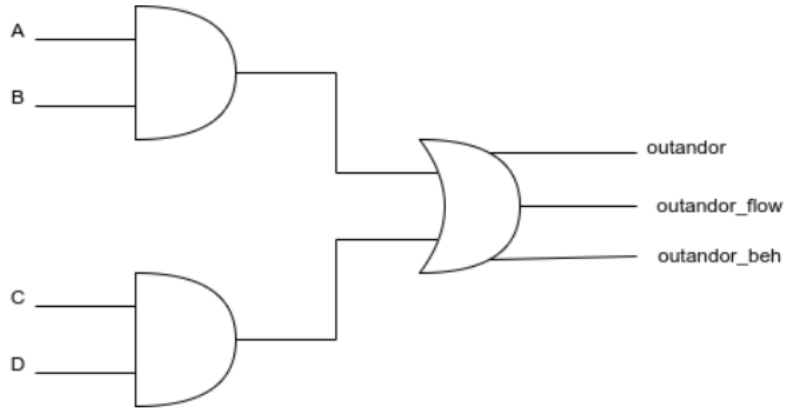

Figure 1: Circuit representation of AND2OR

As can be seen, this model requires the following I/O:

- 4 inputs - a, b, c, d
- 3 outputs - outandor, $outandor\_flow$ and $outandor\_beh$

Where all three outputs are identical, except they are expressed in structural, flow and behavioural syntax respectively. This can be represented in terms of boolean algebra as:

$$F = (a \cap b) \cup (c \cap d)$$

Hopefully I got the $\cap$ and $\cup$ the right way around. I always get them mixed up $-.-$

Anyway

The truth table of this system, which will form the basis of the test bed design, can be found within table 1

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 1: Truth table for AND2OR program

The structural schematic was further verified through the RTL diagram shown in figure 2. As a side note, it can be seen that Vivado's optimisation combined the flow and behavioural models, thus giving them the same output.

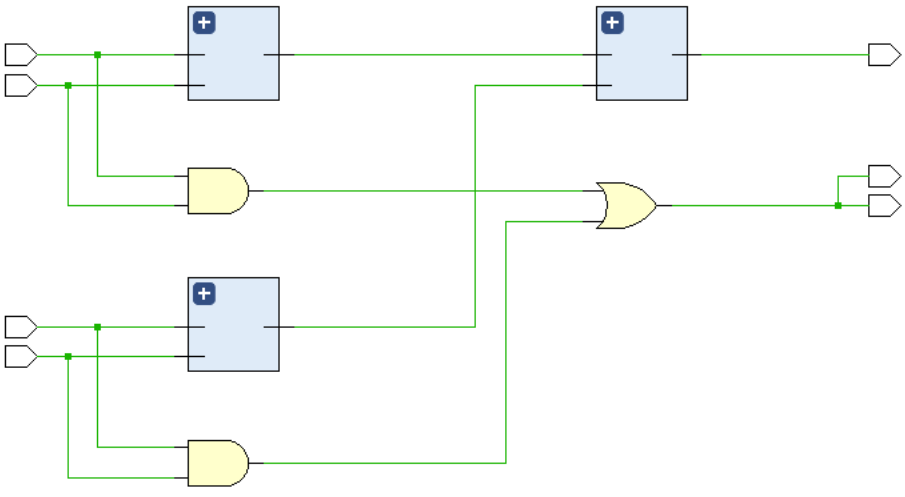

Figure 2: RTL representation of AND2OR

Didn't label inputs and outputs here. We can safely say that we get the idea.

It's worth noting here that this section does not go into great detail into the design methodology of the circuit. That's because we were provided with the file containing two AND gates connecting to an OR gate. No designing required. Even a monkey like me can work with that.

3 Simulation Results

In creating the test bed, it was important to iterate over each of the possible input combinations, and test the outputs for each of the structural, behavioural and flow models. This iteration was achieved using a for loop. Code for this can be found within the appendices. In stress testing the system for every possible input, the following outputs were attained in figure 3

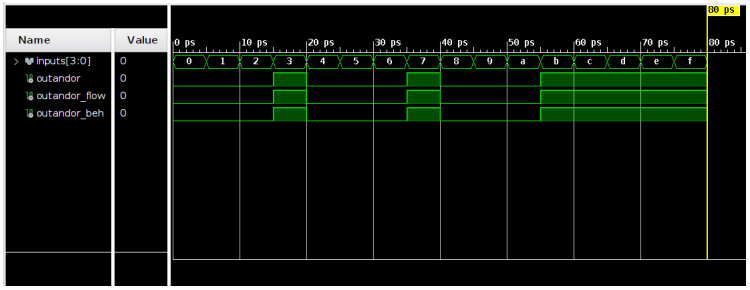

Figure 3: AND2OR simulation without any delays applied

As can be seen in figure 3, the *outandor*, *outandor_flow* and *outandor_beh* each have identical output values which correspond with the truth values specified within table 1. This indicates that the entire system is working as expected. However, it has not fully proven the test bed's ability to detect simulation errors.

For this reason, a secondary test was conducted in which the error checking was intentionally changed. This test scenario was designed such that the test bed expected an output of 0 for *outandor* when an input of $a = 1$ and $b = 1$ was provided. Obviously, this would not happen since $a \cap b = 1$. In this situation, a system error was to be thrown which would display within Vivado's console. The successful outputs of this altered test can be found within figure 4.

```
restart
INFO: [Simtcl 6-17] Simulation restarted
run 80 ps
Error: bad gate - stuck at 1
Time: 20 ps  Iteration: 0  Process: /test_and2gate/input_gen  File: /home/david/Documents/CSSE4010/prac1/prac1.srcs/sim_1/new/test_and2gate.vhd
Error: bad gate - stuck at 1
Time: 40 ps  Iteration: 0  Process: /test_and2gate/input_gen  File: /home/david/Documents/CSSE4010/prac1/prac1.srcs/sim_1/new/test_and2gate.vhd
Error: bad gate - stuck at 1
Time: 60 ps  Iteration: 0  Process: /test_and2gate/input_gen  File: /home/david/Documents/CSSE4010/prac1/prac1.srcs/sim_1/new/test_and2gate.vhd
Error: bad gate - stuck at 1
Time: 80 ps  Iteration: 0  Process: /test_and2gate/input_gen  File: /home/david/Documents/CSSE4010/prac1/prac1.srcs/sim_1/new/test_and2gate.vhd
```

Figure 4: Console outputs for an intentional error for inputs A and B

At this point, it has been proven that both the test bed and the AND2OR file are functioning as expected. The final simulation required the integration of a propagation delay into the system, to emulate the delays experienced due to physical components in real life. To achieve this, a propagation delay was applied to the output *outandor_beh*, which could be seen to have a staggered and delayed output compared to the rest of the system. This could be seen within figure 5

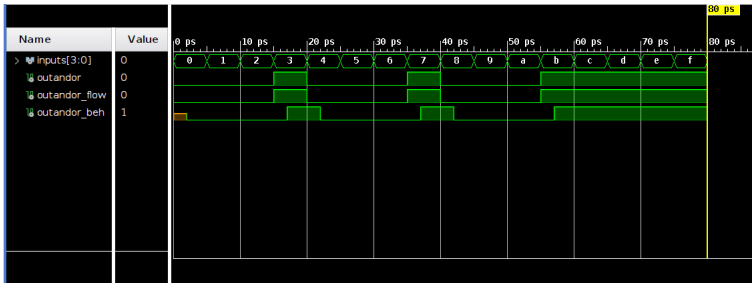

Figure 5: Simulated output for AND2OR with delay

In total, the design was functionally correct. It displayed the correct outputs, had the correct error checking and was able to properly integrate delays.

4 Synthesis and Implementation Results

Upon completing the simulation, it was now necessary to run a synthesis model. This provides a generalised model upload model of the program, which can be further refined by running an implementation. The results of the model's synthesis can be found in figure 6

Report	Type	Options	Modified	Size
Synthesis				
Synth Design (synth_design)				
Utilization - Synth Design	report_utilization		8/3/23, 11:10 AM	6.4 KB
synthesis_report			8/3/23, 11:10 AM	13.6 KB
Implementation				
impl_1				
Design Initialization (init_design)				
Timing Summary - Design Initialization	report_timing_summary	max_paths = 10;		
Opt Design (opt_design)				
DRC - Opt Design	report_drc			
Timing Summary - Opt Design	report_timing_summary	max_paths = 10;		
Power Opt Design (power_opt_design)				
Timing Summary - Power Opt Design	report_timing_summary	max_paths = 10;		
Place Design (place_design)				
IO - Place Design	report_io			
Utilization - Place Design	report_utilization			
Control Sets - Place Design	report_control_sets	verbose = true;		
Incremental Reuse - Place Design	report_incremental_reuse			
Incremental Reuse - Place Design	report_incremental_reuse			
Timing Summary - Place Design	report_timing_summary	max_paths = 10;		

Post-Place Power Opt Design (post_place_power_opt_design)		
Timing Summary - Post-Place Power Opt Design	report_timing_summary	max_paths = 10;
Post-Place Phys Opt Design (phys_opt_design)		
Timing Summary - Post-Place Phys Opt Design	report_timing_summary	max_paths = 10;
Route Design (route_design)		
DRC - Route Design	report_drc	
Methodology - Route Design	report_methodology	
Power - Route Design	report_power	
Route Status - Route Design	report_route_status	
Timing Summary - Route Design	report_timing_summary	max_paths = 10;
Incremental Reuse - Route Design	report_incremental_reuse	
Clock Utilization - Route Design	report_clock_utilization	
Bus Skew - Route Design	report_bus_skew	warn_on_violation = true;
implementation_log		
Post-Route Phys Opt Design (post_route_phys_opt_design)		
Timing Summary - Post-Route Phys Opt Design	report_timing_summary	max_paths = 10; warn_on_violation = true;
Bus Skew - Post-Route Phys Opt Design	report_bus_skew	warn_on_violation = true;
Write Bitstream (write_bitstream)		
report_webtalk		
implementation_log		

Figure 6: Results of synthesis

Furthermore, the utilisation of the system for running this design can be found in figure 7. Note that only a fractional percentage 3% of the board I/O was utilised, as well as one of the 63400 available lookup tables. This could be confirmed upon close analysis into the implemenation's architecture (But that goes far beyond the scope of this assignment. That's a deep, dark world in there).

Resource	Utilization	Available	Utilization %
LUT	1	63400	0.00
IO	7	210	3.33

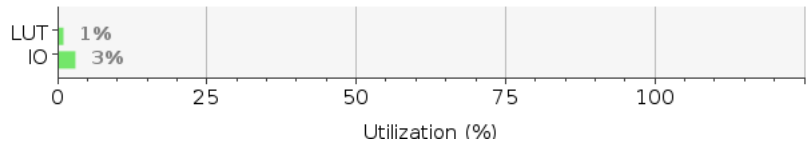

Figure 7: Utilisation report for AND2OR

5 Discussion and Conclusion

In conclusion, it can be seen that the system functions fully as expected. The AND2OR file is able to correctly create the the system shown in figure 1, which maps to the outputs shown in table 1. Error checking has been verified, and the system was able to fully integrate delays. Lastly, the model was able to synthesize and provide outputs to the implementation process as described within its corresponding section.

In total, the entire system works, and there is no need for improvement.

6 Appendices

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity test_and2gate is
--  Port ( );
end test_and2gate;

architecture Behavioral OF test_and2gate IS

COMPONENT and2or
    PORT(
        in1,in2,in3,in4 : IN std_logic;
        outandor, outandor_flow, outandor_beh : OUT std_logic);
END COMPONENT;

signal inputs: std_logic_vector(3 downto 0):= "0000";
signal outandor: STD_LOGIC;
signal outandor_flow: STD_LOGIC;
signal outandor_beh: STD_LOGIC;


BEGIN

uut: and2or PORT MAP (
in1 => inputs(0),
in2 => inputs(1),
in3 => inputs(2),
in4 => inputs(3),
outandor => outandor,
outandor_flow => outandor_flow,
outandor_beh => outandor_beh);

input_gen : process
begin
    inputs <= "0000";
    for I in 0 to 16 loop

        wait for 5ps;

        if (inputs = "0000") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "0001") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "0010") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "0011") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_beh = '1') report "bad gate - stuck at 0" severity error;
        elsif (inputs = "0100") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "0101") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "0110") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "0111") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_beh = '1') report "bad gate - stuck at 0" severity error;
        elsif (inputs = "1000") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "1001") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "1010") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_beh = '0') report "bad gate - stuck at 1" severity error;
        elsif (inputs = "1011") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
```

```
            assert (outandor_flow = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_beh = '1') report "bad gate - stuck at 0" severity error;
        elsif (inputs = "1100") then
            assert (outandor = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_flow = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_beh = '1') report "bad gate - stuck at 0" severity error;
        elsif (inputs = "1101") then
            assert (outandor = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_flow = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_beh = '1') report "bad gate - stuck at 0" severity error;
        elsif (inputs = "1110") then
            assert (outandor = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_flow = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_beh = '1') report "bad gate - stuck at 0" severity error;
        elsif (inputs = "1111") then
            assert (outandor = '0') report "bad gate - stuck at 1" severity error;
            assert (outandor_flow = '1') report "bad gate - stuck at 0" severity error;
            assert (outandor_beh = '1') report "bad gate - stuck at 0" severity error;
        end if;
        inputs <= inputs + '1';
    end loop;
    wait;
end process;
```

Do you remember in 2019 how bushfires ravaged 17 million hectares of Australian soil; close to 50 times the landmass of Germany? In that time, 33 lives, 3'094 houses and one billion animals were consumed consumed in those flames. Ideally, prevention is the best cure, but due to the time and cost associated with assessing potential bushfire threats, National Park rangers and the Royal Fire Brigade lack the propensity to address every hazard before it's too late. If there was a way to assist them with the on-the-ground coverage of these threats, it would save over 1.5 billion dollars annually, as well as ensure the safety of our country.