|  |
| --- |
| University of Calgary |
| ENEL 453 |
| **Laboratory #1 - Introduction: Simulation, Synthesis and Implementation on FPGA** |
| **Lab: B06** |
| **Kyle Derby MacInnis** |
| **Sajna Massey** |
| **1/22/2013** |

|  |
| --- |
| We declare that this laboratory report is entirely our own work and includes no material which has been copied from any other source excepting that material which is clearly identified as the work of others. |

**PART I: First Circuit Implementation in VHDL**

**RTL Schematic for Circuit #1:**

Following is the RTL Schematic for the initial circuit in the laboratory. In this schematic, the circuit is made up from various digital logic gates. This schematic represents the logic performed by the circuit as it was programmed.

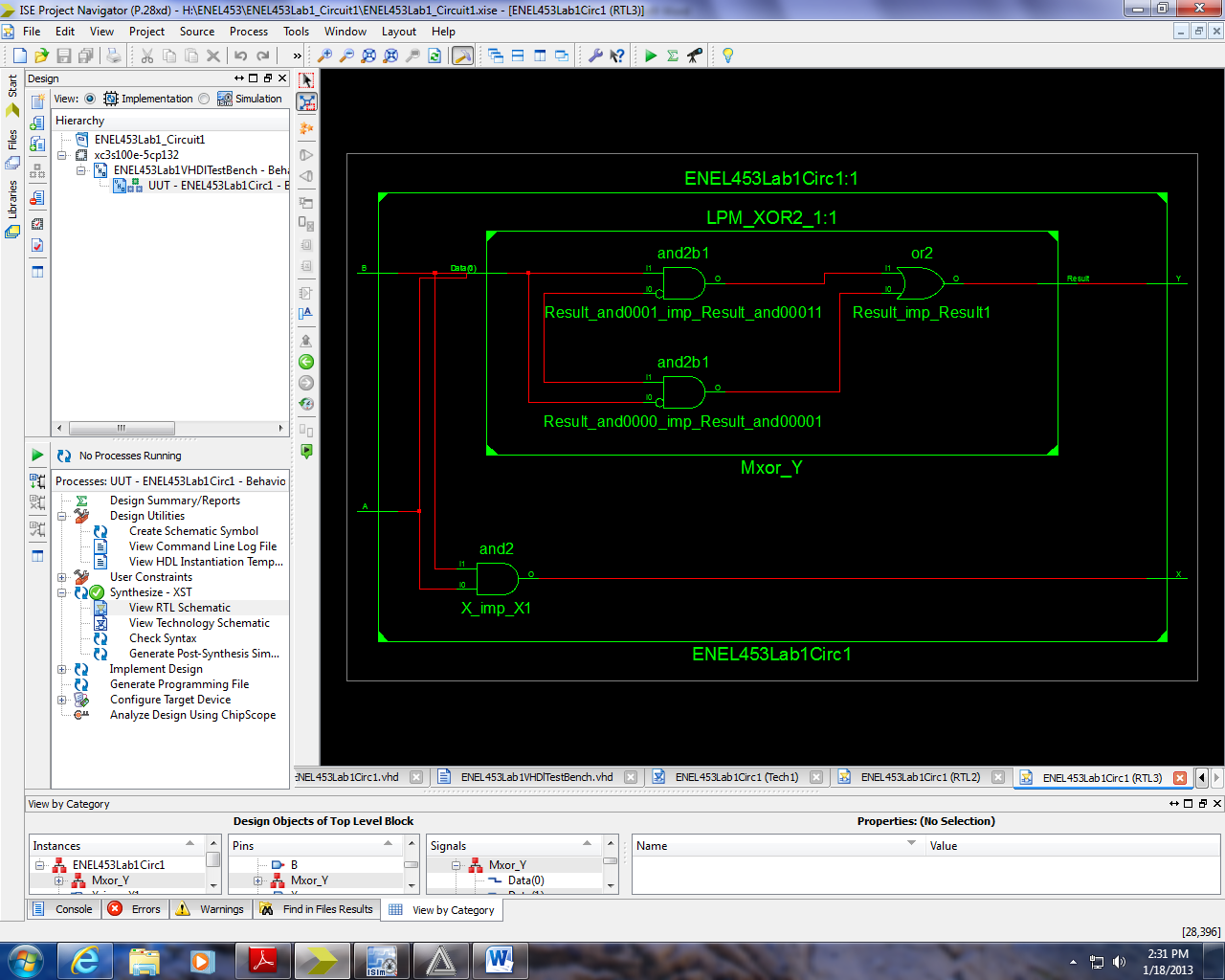


Figure - RTL Schematic

This schematic would be more useful during the debugging stages of the design process as the logic is plainly shown and it is much easier to quickly grasp the intended function from the diagram. This is also a great schematic for implementing components which may be new, as the logic can quickly be seen, and inclusion of the component can be readily designed to accommodate any requirements.

**Technology Schematic for Circuit #1:**

Following is the Technology Schematic for the first Circuit in the Laboratory. As can be seen, there are two look-up tables which in turn are connected to the buffered inputs, and after the values have been found, they are again outputted from the look-up tables through another set of buffers, and to the output ports.

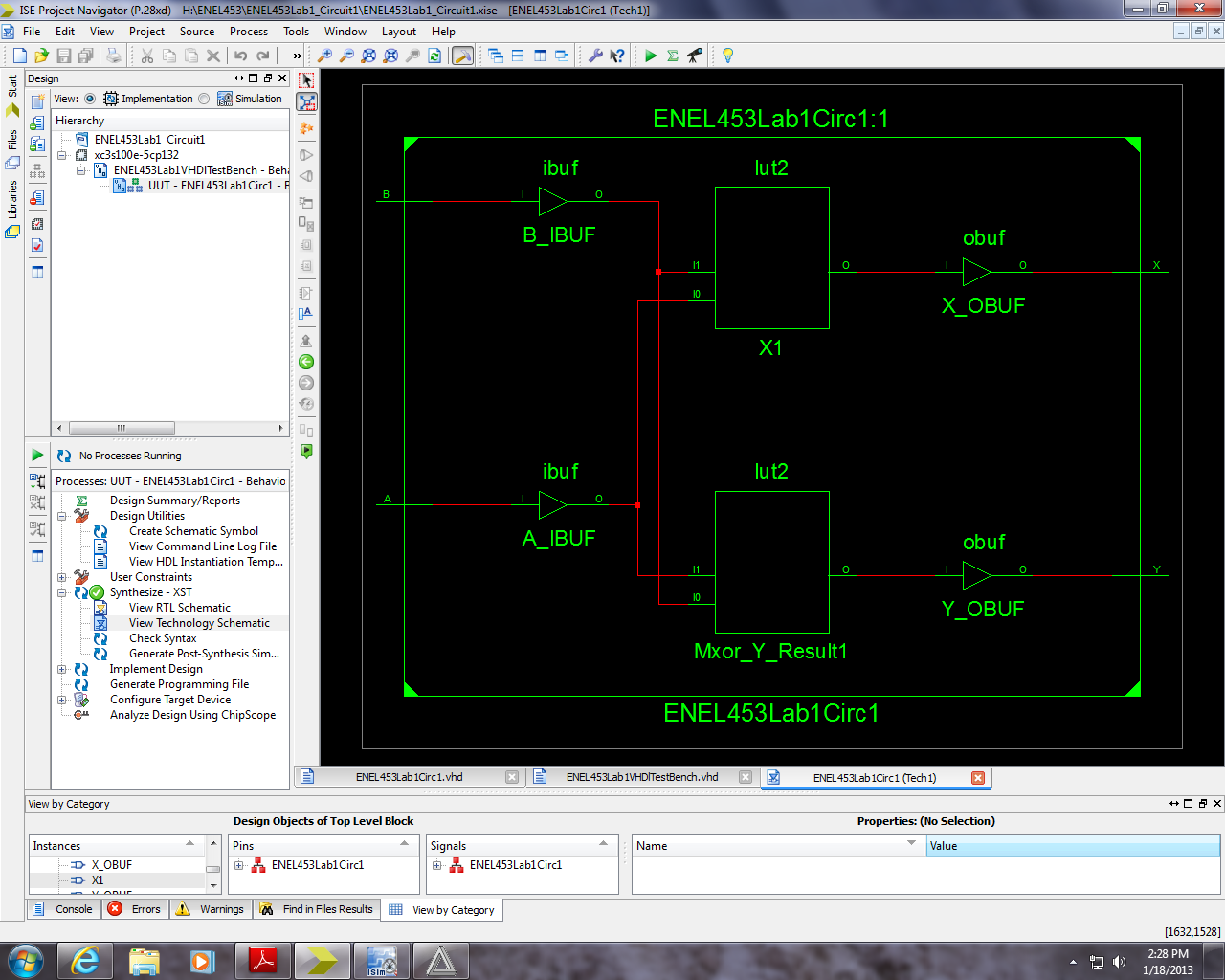
****

Figure 2 - Technology Schematic

**Comparison of RTL and Technology Schematics:**

The difference between the schematics is readily apparent as it goes to show how an FPGA is designed in order to be programmable as intended versus say a permanent circuit. The lookup tables are implemented as they are much easier to modify, and do not require a vast array of logic gates, rather they are large memory banks(RAM) which are then programmed with the various truth tables and some routing between the pins so as to “calculate” the various processes intended.

**Waveform Diagram for Circuit #1:**

The following figure shows the waveform data following the execution of the simulation. It shows the expected results from all the signals. As can be seen, it correctly executed both the AND and the XOR statements for Xout and Yout, respectively.

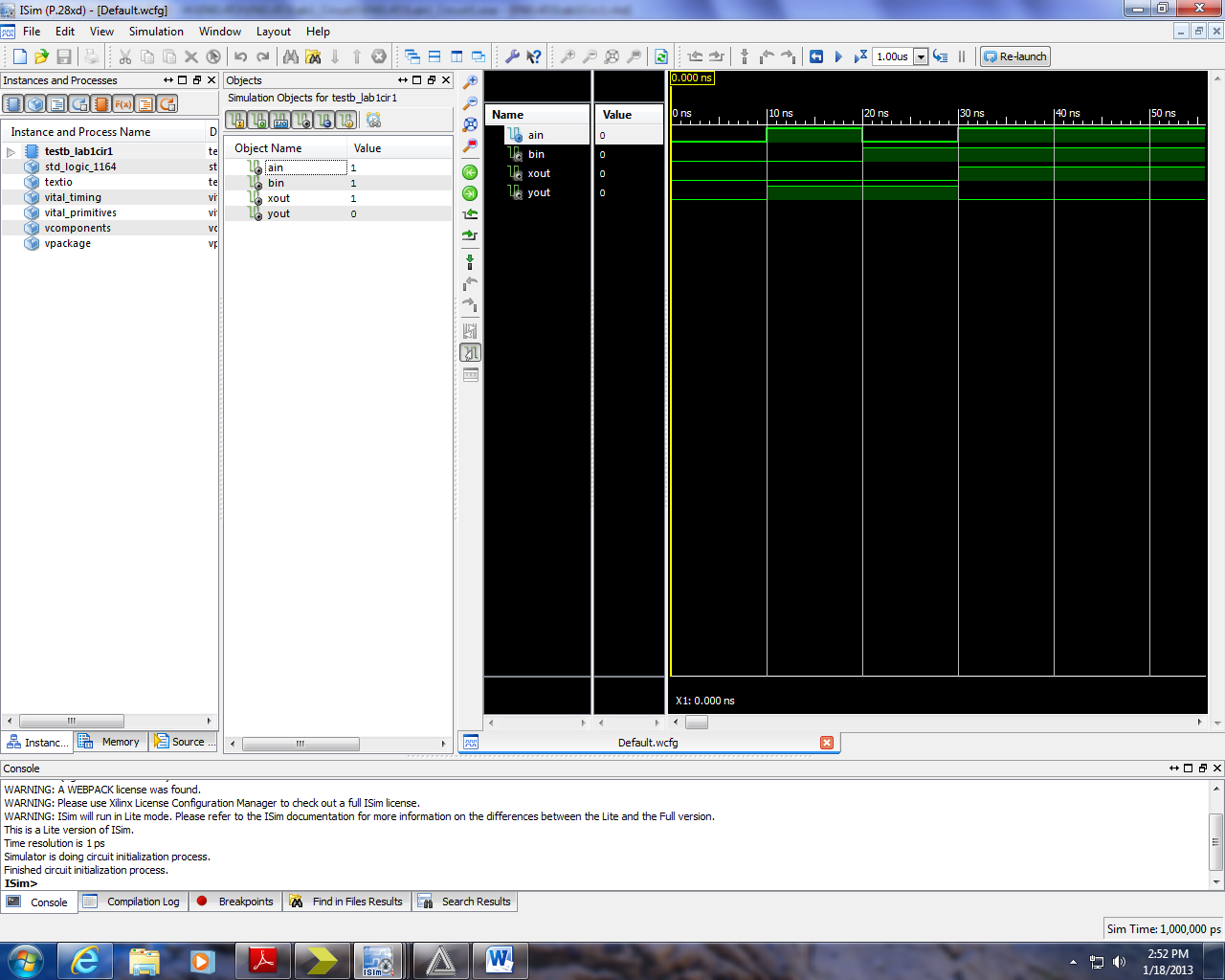


Figure 3 - Waveforms for Structure

**PART II: Sequential Circuit Implentation in VHDL**

**RTL Schematic for Sequential Circuit:**

The following is the RTL Schematic of the Sequential Circuit. As can be seen below, it shows an AND gate connected between the first two pins, and the sequential process on the right, fdce, connected to the output from the gate, the enable, reset, and clock; as well as output the final result to ‘Z’.

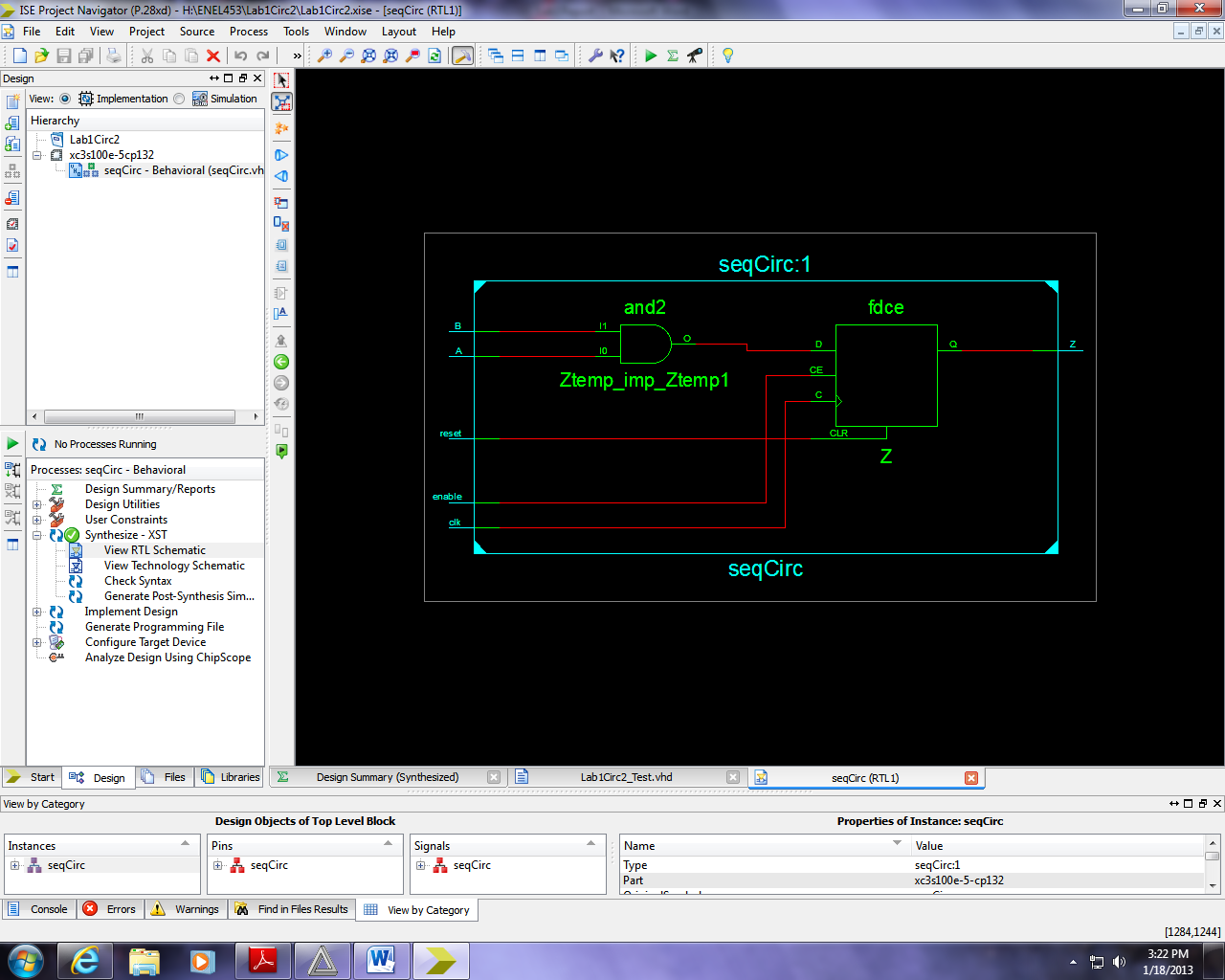


Figure - Sequential Circuit RTL Schematic

**PART III: Broken Circuit Implementation in VHDL**

For the broken circuit, the reason that it shows an error when simulated with a value of:

*A <= ‘1’*

This is due to the fact that the first line in the architecture is a concurrent statement setting the value of Y, whereas the process (sequential) is in contradiction to the concurrent statement resulting in an unknown value for the simulator.

**Concurrent**: *Y <= A and B; -- equals ‘0’ (Because 1 AND 0 = 0)*

**Sequential**: *Y <=A; -- equals ‘1’ (Because A = 1)*

When attempting to synthesize the circuit, an error stating that “multiple drivers” were present for the output signal, Y. This would appear to reinforce the above explanation as a correct diagnosis of the problem. (ie. Concurrent and Sequential Circuits are mixed together)

**PART IV: Additional Questions**

**Question 1:**

Here is the definition for the AND truth table from the file:

CONSTANT and\_table : stdlogic\_table := (

-- ----------------------------------------------------

-- | U X 0 1 Z W L H - | |

-- ----------------------------------------------------

( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ), -- | U |

( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | X |

( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | 0 |

( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | 1 |

( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | Z |

( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | W |

( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | L |

( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | H |

( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )); -- | - |

Based on the file, it looks like the following describes what each of the row/column entries are:

PACKAGE std\_logic\_1164 IS

-------------------------------------------------------------------

-- logic state system (unresolved)

-------------------------------------------------------------------

TYPE std\_ulogic IS ( 'U', -- Uninitialized

'X', -- Forcing Unknown

'0', -- Forcing 0

'1', -- Forcing 1

'Z', -- High Impedance

'W', -- Weak Unknown

'L', -- Weak 0

'H', -- Weak 1

'-' -- Don't care

);

So if we then compare this to the truth table, it makes lots of intuitive sense:

* ‘0’ and ‘L’ have the highest priority and force the result to ‘0’ each time.
* ‘U’ has the next highest priority and will force to ‘U’
* ‘1’ is a forcing value and is only forces ‘1’ with other ‘1’ or ‘H’
* ‘X’ are unknown values which are forced by the system whenever something cannot be determined.
* ‘Z’ are high impedance signals, and usually is either an unknown (‘X’), or is forced to ‘0’ or ‘U’.
* ‘W’, ‘H’, and ‘L’ are all lower priority signals. They get overwritten when a forcing signal.
* ‘-‘ are don’t cares and as such they have no priority and have no bearing, they can be whatever signal that is required.

Since we are dealing with real circuits there aren’t always perfect values for logic 1 and 0. Due to this, there are times where the result is not always known and at those times, the result is dependent on the design of the hardware. These are the cases when either an ‘X’ or ‘W’ or ‘Z’ is involved.

**Question 2\*:**

The differences between the std\_logic values of ‘-‘, ‘X’, ‘W’, ‘U’ are roughly as follows:

* Don’t Cares have no bearing, and can be used as either ‘0’ or ‘1’ as needed.

**Question 6\*:**

Based off of the synthesis report, it appears that the circuit requires 4.04 ns to compute the value from the DFF and to output it to the Z port. This is the slowest portion of the circuit (the first part is only 2.5 ns in length). The frequency can be calculated as follows and result in a value of 248 MHz as the maximum frequency for this circuit.

**Question 2:**

The part of the code for the Sequential Circuit which is executed concurrently is the line:

*ZTemp <= A and B;*

As well as the check against the sensitivity list in the process statement to see if any changes have been made.

The part of the code which is executed sequentially is the code contained within the process. This is the portion of the code which executes the **if-elsif-else** statements, and which is only executed should one of the variables in the sensitivity list change.

Debugging a concurrent program is a lot different from a sequential program as you cannot simply place labels (or system outputs) at various stages to ensure proper flow and execution, rather since everything is calculated at the same time, you must cross-reference signals and ensure that no two operations will be conflicting with each other, and if so, you must either reconsider the logic, or you must force it into a sequential model if necessary. An approach which might alleviate some of the difficulties associated with debugging would be to properly layout all the signals involved into a chart or diagram and to separate any signals which are dependent on each other.

**Question 3:**

The .ucf files are used to map signals to physical pins on the hardware. It dictates to the compiler where to program the values into the LUT’s so that they are properly calculated when run. In addition it allows people to state the input locations such as switches, clocks(oscillators), push buttons; and also to connect signals up to outputs such as actuators, LED’s, and the 7-segment display. All signals are digital so they either are running, or not. And then anything else is separately controlled via individual processes (i.e. – Number to be displayed based on pin voltages)

**Question 5:**

Pre-Synthesis and Post-synthesis simulations are different due to the physical circuit which Xilinx synthesizes based on the hardware available.

The Pre-synthesis simulation is purely signal based, and works on just the logic of the architecture. But it does not take into account any physical structures such as LUT’s. It pretty much just runs through the programming.

The Post-synthesis simulation is based on the physical system as chosen at the beginning of the project (Spartan 3E). It then maps various signals to physical pins on the hardware, and loads up a description of the LUT’s. It then runs through the “physical” description when simulating the circuit.n

Question 4\*:

The reason that there is no sensitivity list for the processes included in the testbench is due to the fact that the testbench’s purpose is to specify all changes made, and as such, it should not be looking for any extraneous changes from the circuit.