

TSIU03: Lab 1 - Introduction and Lab Tools

Petter Källström and Mario Garrido

August 16, 2021

Abstract

This lab aims to let you test the equipment (the DE2-115 board) and tools (Quartus and ModelSim). Write a very simple VHDL file, compile and simulate it, synthesize it, upload it on the FPGA board and test it.

1 Introduction


In this lab, there are two projects to carry out. Task 1 is a simple XOR gate, with two switches as input and a LED as output. Task 2, called Div5, will test if a four-input number is divisible by five or not.

It is recommended to stick to the names/folders, etc., in the examples/codes given here.

2 Notations

This tutorial uses the following notations:

- Do an action:

Processing→Start Compilation	Ctrl+L	
------------------------------	--------	---

 (any one of these options work).
- Set values in dialog boxes etc:

Name="my_xor"

- Click on button

[OK]

- \implies *Intended result*.
- Select in a tree structure:

Components / Logic Gates / AND2

In ModelSim you can also type in commands to execute, which will be notes as:

(description of how to solve it by click-click-click)	> command to execute
---	----------------------

where you can select any of those methods.

3 Your Task 1: XOR

Your task is to create a project from scratch, write the code for the XOR gate, simulate the circuit and place it on the DE2-115 board.


Remember the XOR gate from the switching theory; $y = a \text{ xor } b$ is true if and only if either a or b , but not both, are true, i.e. if $a \neq b$. This can be written as $(a \text{ and not } b)$ or $(b \text{ and not } a)$.

Keep the following steps in mind, since you will need them in the coming labs (hint: Bring the lab manual to the coming labs).

3.1 Create a Project

First of all, you need a folder, for instance `X:\TSIU03\Lab1`. Create that (or do it later from within Quartus).

3.1.1 Start Quartus

Find and run Quartus II (64 bit)  in the start menu. It is probably located in folder Altera.

\implies *This should start Quartus II version 13.0sp1*

3.1.2 Start the “New Project Wizard”

You should get a welcome screen in Quartus, where you can click [Create a New Project]. If you killed the welcome, then [File→New Project Wizard]. In the “introduction” screen, click [Next].

- **Page 1 of 5 - Directory Name etc.**
 - Project directory: X:\TSIU03\Lab1 (type it, or click [...] and create it).
 - Project name: “my_xor”.
 - Top-level design entity: Same as the project name.
 - Click [Next] – and yes, you want to create the directory, if you are asked.
- **Page 2 of 5 - Add files** Nothing here, [Next].
- **Page 3 of 5 - Family and device.**
 - [Family=“Cyclon IV E”].
 - Select [Device=“EP4CE115F29C7”].

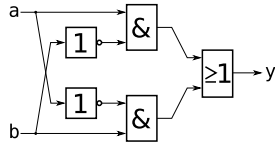
(You can type (parts of) the device name in the filter box.)
- **Page 4 of 5 - EDA Tool Settings.**
 - [Simulation=“ModelSim”, “VHDL”]
 - [Other fields=“None”].
- **Page 5 of 5 - Summary Done,** [Finish].

3.2 Create a VHDL File

[File→New...] [Ctrl+N] [📄]

- In the “New” dialog, select [Design Files / VHDL File].

Fill the file with VHDL code, according to Code 1, including the typo “b and not b”.

<pre> library ieee; use ieee.std_logic_1164.all; entity my_xor is port(a,b : in std_logic; y : out std_logic); end entity; architecture rtl of my_xor is begin y <= (a and not b) or (b and not b); end architecture; </pre>	 <p>(xor gate, as it <i>should</i> be, without the typo)</p>
---	--

Code 1: The complete code for the Tutorial lab.

Save the file as “my_xor.vhd” in X:\TSIU03\Lab1\.

3.3 Synthesize the Project

[Processing→Start Compilation] [Ctrl+L] [▶]

⇒ *This will take a while, you can start Modelsim in the meanwhile.*

Fix eventual syntax and similar errors, and recompile until you get a successful compilation.

Look at all compilation warnings at the bottom of the Quartus window (look, but ignore).

3.4 Simulate the Project

Now you have a complete project, but you have not checked it for logic errors (so it behaves as desired).

3.4.1 Start ModelSim

Find and run Modelsim **M** in the start menu.

⇒ *ModelSim will flash and do things for a while*

Close the “Welcome” screen, if shown.

3.4.2 Create a Modelsim Project

A Modelsim project is structure with settings, pointers to VHDL files etc.

File→New→Project

- Project Name=“my_xor_sim”.
- Project Location=“X:/TSIU03/Lab1/MSim”
- [OK] (Yes, you want to create the folder).

In “Add items to the Project”:

- [Add Existing File]
 - File Name=“../my_xor.vhd”
 - [OK]
- [Close]

Compile→Compile All

In the ModelSim window you will now have a **Transcript** frame in the bottom. It is a kind of command shell for ModelSim. Most actions you do in ModelSim will result in a command that is executed in the transcript window (this is handy if you want to make “.do” files, a TCL script file).

You also see a number of tabs in different other frames:

- The **Library** frame lists all available libraries and their objects. Your module(s) are placed in the “work” library.
- The **Project** frame lists all files included in the current project.
- You might have a **Wave** frame. Here you will see the waveforms of the signals after simulation.

3.4.3 Simulate...

Do the following steps.

Load your design (“Simulate it”)

- Library frame: work / my_xor
- Right click→Simulate

```
> vsim my_xor
```

⇒ After some more flashing you’ll have a new “Object” frame and some new tabs in the existing frames.

Add signals to the wave form

- In the “Sim” tab: Select “my_xor”.
- Object frame: Select all signals.
- Right click→Add To→Wave→Selected signals

```
> add wave sim:/my_xor/*
```


Generate Input Stimuli

- For input “a”: right click on “a” in the Wave window, select “Clock” and enter:
 - offset = “25ns” (when to start)
 - Duty = “50” (percentage of high)
 - Period = “100ns” (period)
- For input “b”: right click, “Force”, Value=“0”. [OK]. “Force” again, Value=“1”, “Delay for”=“200ns”.

```
> force -freeze a 1 25ns, 0 75ns -r 100ns
> force -freeze b 0 0ns, 1 200ns
```

- “ x tns” means the signal is assigned the value x after t ns.
- The “-r tns” means that the given stimuli is repeated every t ns.

Run the Simulation

 <ul style="list-style-type: none">• Set the “Run Length” field to “400 ns”.• Click the “Run” button.	<pre>> run 400ns</pre>
---	---------------------------

The waveform result is shown in fig. 1.

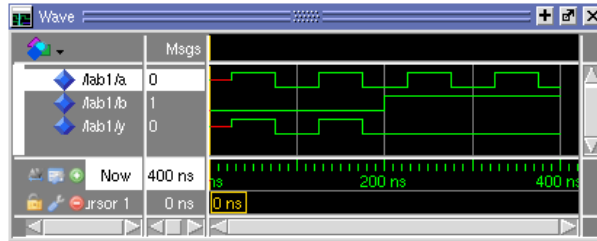


Figure 1: Result of the simulation

3.4.4 View Result

There are a lots of usefull tricks in the waveform window.

Zoom the Wave

Draw a line with the middle mouse button to zoom:

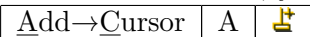
- **down + left/right**: Zoom in (zoom box).
- **up + left**: Zoom fit (show all).
- **up + right**: Zoom out (depends on line length).
- **Ctrl + scroll**: Also works to zoom in/out around the mouse pointer (buggy).

Play around with this until you know how to do.

Measure Time

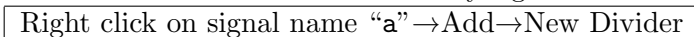

Leftmost in the waves, there is a yellow cursor (look for the yellow box “0 ns” at the bottom). Grab this and move it around in the wave window. The current time point is displayed at the bottom.

With another cursor, you can measure time “distances”.

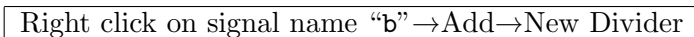
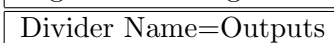
. How many ns wide is a positive pulse on the “y” output signal?

More Tricks

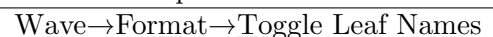
Dividers are usefull when there are many signals. Add a new divider:

- 
- 
- (From the transcript: `add wave -divider Input`).

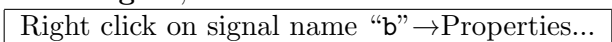
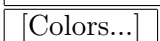
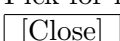

Add another divider:

- 
- 

Move a signal, since the “Output” divider was placed in the wrong place: Grab signal “b” with left mouse button, and drag it to above the “Output” divider.

Toggle Leaf Names, from . This is handy if the complete signal name is too long in the wave window.

Color a signal, this is also usefull when there are many signals:

- 
- 
- Pick for instance a suitable yellow color.
- 
- 
- (When adding a signal from the transcript: `add wave -color yellow sim:/my_xor/b`).

Make sure you remember those things. Each student must be able to do things like this during the demonstration.

Save the Waveform Format

When you have added/formatted the signals you want to the waveform, you can **save the waveform format** as `wave.do` file.

```
File→Save Format... | Ctrl+S
```

Later you can load this to get back the format, e.g. by `> do wave.do`.

You can also add more commands to the `.do` file, e.g. compilation, restart, run, `wave zoom full` etc. A recommendation is however to save those commands in another `.do` file, so you can resave the waveform format after a change, without overwriting all your new commands.

3.4.5 Oh No! Something’s Wrong

The output is assumed to be an XOR function, that means `y` is 1 if exactly one of `a` and `b` are 1.

This clearly does not hold when `b` is 1.

Solve this by correcting the typo in the VHDL file (in Quartus), and save the file:

```
y <= (a and not b) or (b and not a)
```

(You do *not* have to resynthesize in Quartus – just change the typo and save and you’re done)

Resimulate

In ModelSim, recompile the unit, and simulate it again.

<ul style="list-style-type: none"> Find the Project frame/tab. Right click on <code>my_xor.vhd</code> →Compile→Compile Out-of-Date 	<pre>> vcom ../my_xor.vhd (run “pwd” to see that you are in a sub directory of the project directory)</pre>
Click the “Restart” icon to the left of the Run Length box.	<pre>> restart -force</pre>
Assign <code>a</code> and <code>b</code> again, just like before.	<pre>> force -freeze a 1 25ns, 0 75ns -r 100ns > force -freeze b 0 0ns, 1 200ns</pre>
Run 400 ns again	<pre>> run 400ns</pre>

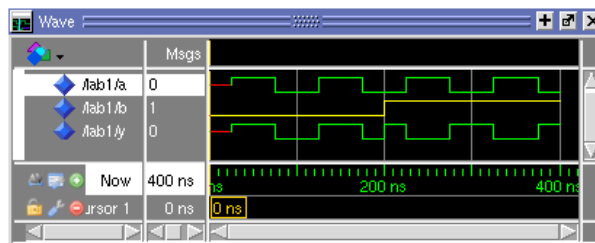


Figure 2: Result of the simulation, when the bug is corrected.

Now it looks okay. Keep the window open, so you can show it during the demonstration.

3.4.6 Extra: Cleaning

When simulating the circuit, Modelsim produces a number of files. You don’t have to do this now, but it can be good to know how to do. Folder `work` is your library where your compiled modules goes (this can be removed, but then you need to recompile the modules again). `*.mti`, `transcript`, `vsim.wlf` can be removed. `*.vhd`, `*.v`, `*.mpf` should be saved (`.mpf` is the Modelsim project).

If you remove the `work` folder, you need to recreate it before you compile anything, using the command: `vlib work`.

3.5 Important Knowledge

In this lab you have only one unit. In lab 3 and 4, and especially in the project, you will have several units that works together. In ModelSim you can select to simulate one unit at a time.

3.6 Configure the FPGA

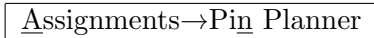
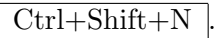
So, you have created a project, simulated it, found a bug and corrected it. Now it's time to test it on the board. Go back to Quartus.

3.6.1 Pin Assignments

You have two input bits and one output bit. You need to tell Quartus which physical pins on the FPGA those should be mapped to, so they will work with the desired keys and LEDs on the board.

Do the following steps.

Start the pin planner

 .

Which pins to choose

You should use the two right most switches on the DE2-115 board (SW0 and SW1), and one of the LEDs. Different lab groups should use different LEDs, so you can see if it is *your* code running on the FPGA board. Table 1 show you which pins you should use (find your lab group number in the “groups” columns). LEDR and LEDG is the red and green LEDs. If you do not have a lab group number, use the number of your table.

Device	Pin	Groups	Device	Pin	Groups	Device	Pin	Groups
SW0 (“a”)	AB28	All	LEDR7	H19	8, 35, 62	LEDG0	E20	19, 46, 73
SW1 (“b”)	AC28	All	LEDR8	J17	9, 36, 63	LEDG1	E22	20, 47, 74
SW2	AC27	(task 2)	LEDR9	G17	10, 37, 64	LEDG2	E25	21, 48, 75
SW3	AD27	(task 2)	LEDR10	J15	11, 38, 65	LEDG3	E24	22, 49, 76
LEDR0	G19	1, 28, 55	LEDR11	H16	12, 39, 66	LEDG4	H21	23, 50, 77
LEDR1	F19	2, 29, 56	LEDR12	J16	13, 40, 67	LEDG5	G20	24, 51, 78
LEDR2	E19	3, 30, 57	LEDR13	H17	14, 41, 68	LEDG6	G22	25, 52, 79
LEDR3	F21	4, 31, 58	LEDR14	F15	15, 42, 69	LEDG7	G21	26, 53, 80
LEDR4	F18	5, 32, 59	LEDR15	G15	16, 43, 70	LEDG8	F17	27, 54, 81
LEDR5	E18	6, 33, 60	LEDR16	G16	17, 44, 71			
LEDR6	J19	7, 34, 61	LEDR17	H15	18, 45, 72			

Table 1: The pinout on the EP4CE115F29 FPGA used in this lab. SW2 and SW3 is used later in the lab.


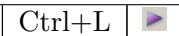
Set the pins

Since you synthesized your code (section 3.3), the Pin Planner knows which pins should be placed, but not where. Hence, you should get a list with the “node names” a, b and y, with empty “Locations”.

Set the **Location** fields to “PIN.*n*”, where *n* is the pin name from table 1. The result should look something like in Fig. 3

Close the Pin Planner (everything should be automatically saved).

Synthesize the Design Again

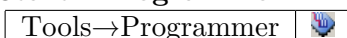
 

⇒ You will now have a “my_xor.sof” file, in folder *output_files*, that contains the configuration data for the FPGA.

3.6.2 Set up the Network

The DE2-115 boards are programmed over the network. You have to configure the “Programmer” tool correctly. This must be done every time the DE2-115 board is moved, or a new board is selected.

Start “Programmer”



Add the .sof File

If there is no list with file “output_files/my_xor.sof”, you must add it.

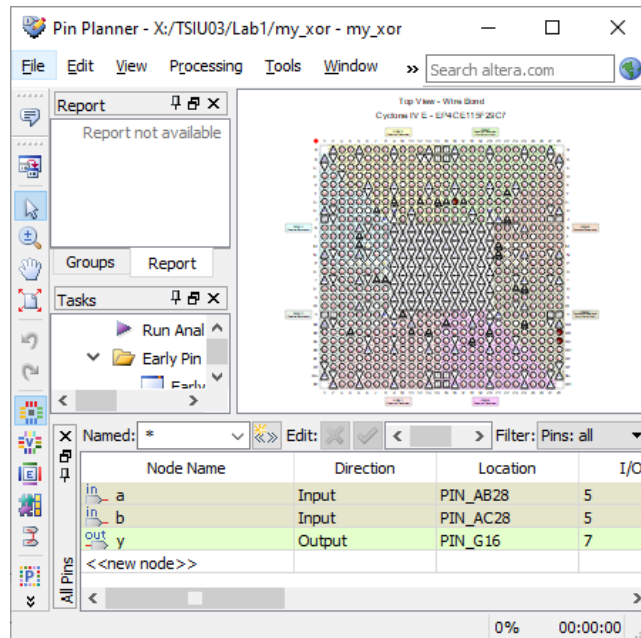


Figure 3: Pin Planner, when using LEDR16 as output.

- [Add File]
- Select “output_files/my_xor.sof“

Set up the Network in the Programmer

This has to be done once per FPGA board (Quartus will remember those settings).

- [Hardware Setup...] In the Hardware Setup dialog:
 - [Add Hardware...] In the Add Hardware dialog:
 - * Hardware type=“EthernetBlaster”.
 - * Server port=“1309”.
 - * Server name and password will be given by the lab assistant.
 - * [OK].
 - Currently selected hardware=USB-Blaster on ...
 - [Close] \implies *The Programmer is shown in Fig. 4.*
- Do *NOT* click on the Start button yet. Continue with the hand-on solution instead.

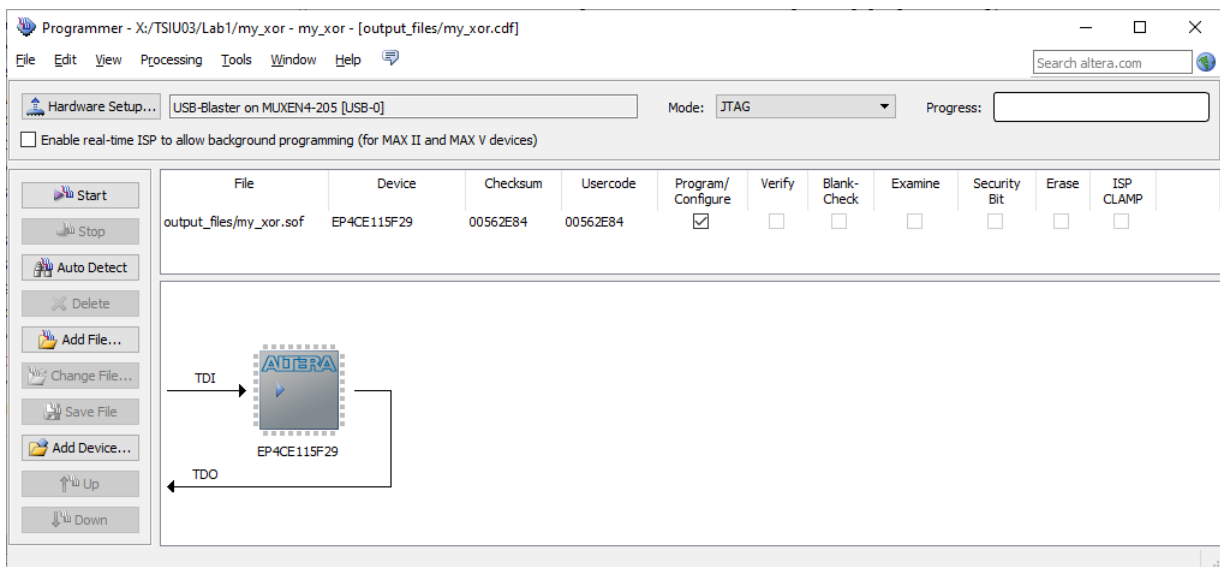


Figure 4: The programmer, after setup of everything.

3.6.3 The Hand-on Solution: Program the Board

Anyone can program the DE2-115 boards at any time. There is a great risk of chaos if you don't have some kind of agreement, due to the risk of collisions. Normally, we use a "hand-on solution", where one group member holds a hand on the board while the other programs it.

Corona edition: Without lab partner, we try another solution:

1. Stand up before you program the board.
2. Look for anyone else standing up (especially at the DE2-115 table).
If so, agree somehow on who can start.
3. Program the board, go there and check the design.

3.6.4 Verify the Design.

Flip SW0 and SW1 on the board and make sure your LED is flashing accordingly.

If the design doesn't work, try to find out why. Did you do the pin planner right? Did you synthesize again? Is it *your* program on the board? Correct what's wrong, and test again.

Save .cdf?

If you close the Programmer, you may be asked to save the file "Chain1.cdf". You can do it if you want. This file contains information about what .sof file and other preferences you have. The hardware device (like server/port/password) is a Quartus setting, and is saved anyway.

3.7 Change to a Schematic File

When working with bigger designs, it is very convenient to use VHDL code for modules, and then connect them together using schematics. A schematic is a module, just like the VHDL file. The schematic is a graphical way to connect components, like AND gates, flip flops, adders, or other modules.

You are now going to create a schematic that uses one instance of your previous module, and define the schematic as the top module.


3.7.1 Create a Symbol for the VHDL File

In order to be able to instantiate a module into a schematic, you need to generate a symbol for the module.

Focus the VHDL file (click on the VHDL code), then

File→Create/Update→Create Symbol Files for Current File

3.7.2 Create a Schematic File

File→New... Ctrl+N 

Design Files / "Block Diagram/Schematic File"


⇒ You will get an empty schematic file.

Save the Schematic

Save the schematic as "my_xor_schematic.bdf" in X:\TSIU03\Lab1 (not in "output_files"), and make sure the "Add file to current project" check box is checked in the "Save As" dialog.

If you save the schematic as my_xor.bdf, Quartus will give an error later on. You need resave it as the new file, and remove the wrong file from the project. Then ask a teacher for help.

Instantiate the module "my_xor"


- Double-click in the schematic 
- Project / my_xor
- [OK]

• Place the symbol somewhere in the schematic.

⇒ Now you have the instance "inst" of module "my_xor".

Add a Pin for the "a" Input

A pin corresponds to a signal in the "port" part of the VHDL file.

- Double-click in the schematic 
- “C:/...” / primitives / pin / input
- [OK]
- Place the input pin to the left of the my_xor module.
Rename the pin to “a” by double-clicking the name.

Connect the Pin with Port “a”

Click , and draw a wire between the pin and the “a” port on inst.

This can be done without consulting the icon. When you grab an unconnected port or end of a wire somewhere, there will be an automatic wire-mode.

Auto-pin the other ports

Create schematic pins for all unused ports in “my_xor” in a simpler way:

- Right-click on inst→Generate Pins for Symbol Ports
⇒ *This will auto format inputs/outputs, vectors, names etc.*
- Drag out the input pin from inst, so it looks better.
⇒ *Wires will be created.*
- Drag out the output pin quite a lot (15-20 dots).

Insert a NOT gate

In order to distinguish the new design from the old, we invert the output. Remove the output wire. Insert a NOT gate between the my_xor output, and the output pin. It is found in “C:/...” / primitives / logic / not. Draw new wires to complete the circuit.

The result is depicted in Fig. 5. Save the schematic.

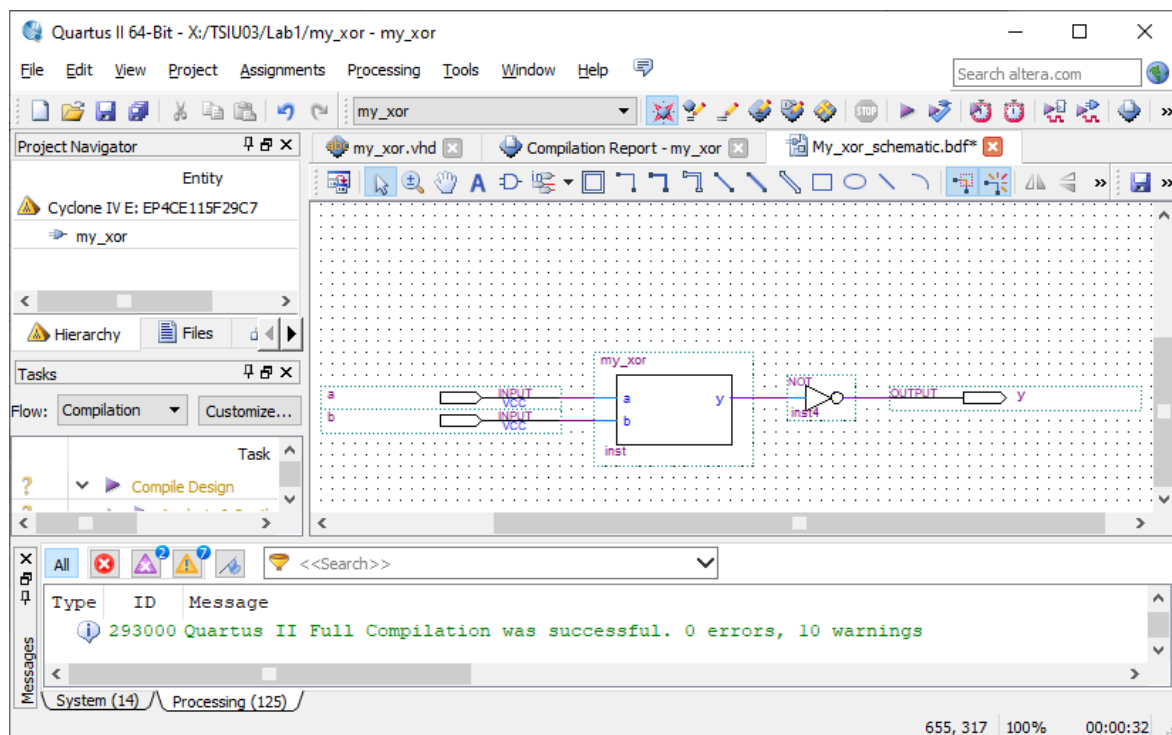


Figure 5: The schematic is done (but not saved).

Set this Module as Top Module

Project→Set as Top-Level Entity Ctrl+Shift+J.

Note that you suddenly can see the pin locations in the schematic.

3.7.3 Simulate the Design (Optional, but good experience)

ModelSim does not understand Quartus schematics. So if you want to simulate a schematic, you first need to generate a VHDL file from it. In Quartus (make sure focus is in the schematic):

File→Create/Update→Create HDL Design...

Enter a name of the file (e.g. “my_xor_schematic.vhd”), make sure the VHDL choice is selected. [Ok].

In Modelsim, you have to add the new file to the modelsim project.

Project→Add to Project→Existing File... [Browse] etc.

Compile out-of-date again (right click in the Project tab).

In the Library tab, now simulate “my_xor_schematic”. (If asked, close the current simulation).

Add the new signals, and simulate using the same stimuli as before.

If the schematic contains a wire without name, Quartus invent a name for it in the VHDL file. Typically, this is “SYNTHESIZED_WIRE...”

3.7.4 Test on the FPGA

Back to Quartus. Compile the design again (Ctrl+L), program the DE2-115 board (using the hand-on solution), and verify that it still works.

3.8 Demonstration

When everything works, let the lab assistant verify it, and also check the schematic and waveform, and discuss how you have done. If there is a queue, you can start the design phase of Part 2.

The lab assistant will ask each student to do some of the Modelsim “tricks” (like zoom, measure time etc). Make sure you can do that without looking in the tutorial.

4 Your Task 2: Div5

Now you have to design a circuit and run it on the FPGA. The initial part is just paper work, that can be done while you are waiting to demonstrate Task 1.

Then, you must create a new Quartus project in a similar way as before.

4.1 Functionality and Design Steps

Design a circuit, Div5 that calculates if a 4-bit unsigned binary number is divisible by 5. The input is SW3, SW2, SW1 and SW0 on the FPGA board (SW3 is the MSB). The represented number will range from 0 to 15, i.e., “0000” to “1111” in binary.

The output LED will be ON if the number represented by the four switches is divisible by 5, and OFF otherwise. Remember that also 0 is divisible by 5. The output is the same LED as you used in task 1. Example: if SW3..SW0 = 1100, which is binary for 12, the output should be 0, since 12 is not divisible by 5.

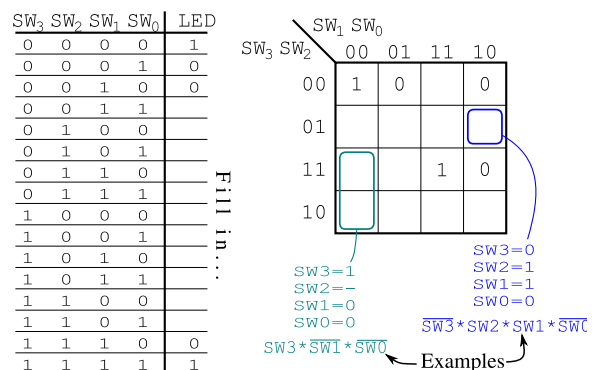
For your help, we supply a printed truth table with Karnaugh map (“k-map”). Do the following

1. Fill in the truth table. 1 for inputs that are divisible by 5 (it should be four of them), and 0 for the others.
2. Translate it into the k-map.
3. Make rings in the k-map.
4. Formulate the rings as an AND-OR formula.
5. Can you simplify the formula using XOR gates?

4.2 Implementation

Create a new Quartus project with the same settings as before (if you still have not demonstrated task 1, you can start a new instance of Quartus).

The project should be named “Div5” and be placed in the same folder as Task 1. Ignore the warnings Quartus gives about having two projects in same folder.



Write the VHDL code for your circuit. It looks just like the XOR gate, but with inputs SW3, SW2, SW1 and SW0, and of course another equation.

Hint: You can split VHDL statements over several lines. I.e. an XOR gate can look like

```
y <= (SW0 and not SW1) or  
      (SW1 and not SW0);
```

Synthesize the code. Do pin placement (see tab. 1). Resynthesize the code.

4.3 Simulation

Create a new Modelsim project, placed in X:/TSIU03/Lab1/MSim. Add the Div5.vhd file, compile, load (“simulate”), and add signals to the wave.

Use the following code to generate the stimuli in Modelsim (type or copy-paste):

```
> force -freeze sim:/div5/SW0 0 0ns, 1 10ns -r 20ns  
> force -freeze sim:/div5/SW1 0 0ns, 1 20ns -r 40ns  
> force -freeze sim:/div5/SW2 0 0ns, 1 40ns -r 80ns  
> force -freeze sim:/div5/SW3 0 0ns, 1 80ns -r 160ns  
> run 160ns
```

You do not have to use dividers, colours etc. in this simulation.

4.3.1 Combine Signals

In the Modelsim wave frame. Select the signals SW0 to SW3, right click, and select “Combine signals”. Result name = “SW”.

You will get a new signal in the wave form, that corresponds the value of the SW* switches. Right click on it, “Radix”, “Unsigned”. Now it’s easier to see what happens. If it counts like 0,8,4,12,... it counts SW0 as the most significant bit (MSB). Remove the signal and try again, with different settings.

4.4 Verify in Hardware

Send the design to the FPGA board (using the hand-on solution) and verify the design.

If the simulation works, but not the FPGA implementation, it can for instance have the following reasons:

- Bad input pin placement (the output LED does not react as it should).
- Bad output pin placement (all LEDs are dim).
- Is it *your* design on the board? (the wrong LED are used).

4.5 View the Result

You have created a design and described it to Quartus as VHDL code. Quartus can show how it interpreted it. This is a good way of verify that Quartus can handle what you wrote.

Tools→Netlist Viewers→RTL Viewer.

Note that Quartus sometimes reformulates your expressions using the switching theory laws (e.g. De Morgans law).

You can also see how it is mapped upon the FPGA.

Tools→Chip Planner.

If needed: Maximize the window, and zoom in (Ctrl+scroll) until the blue-ish thing fills the area.

This is now what you see: Lots of light blue boxes. Those are “logic array blocks” (LABs). One of them is darker, this is the one you use in this project, the rest is unused. Zoom in the used LAB (Ctrl+scroll zooms around the mouse pointer).

You now see that the LAB contains 16 smaller modules. Each is a “logic element” (LE). The left part of the LE has a 16 bits truth table, that can implement any boolean expression with up to 4 inputs. The right part of the LE is a D-type flip-flop (DFF). You use one of the LE. Double click it.

You now see a description of the LE, which wires are available/used, the bit pattern of the truth table etc.

4.6 Demonstrate

Make sure the circuit works also on the FPGA board. When it does, it's time to demonstrate. For the demonstration, be ready to

- show the VHDL code
- show the simulation waveform
- discuss how you solved the problem
- send your design to the FPGA board
- show the Chip Planner.

4.7 Extra: Cleaning

When designing a project, Quartus produces a number of files. You don't have to do this now, but it can be good to know how to do.

Folders `db`, `incremental_db`, `simulation` and `output_files` can be removed. Any `.bak`, `.rpt`, `.summary`, `.smsg`, `.pin`, `.jdi`, `.done`, `.qws`, `.sof`, `.pof` can be removed. Any `.vhd`, `.v`, `.bdf` (schematic), `.bsf` (symbol), `.qpf` (Quartus project), `.qsf` (project settings) and `.sdc` (timing requirement) should better be kept.

4.8 Auto Completion Settings (Optional)

If you are annoyed of the auto completion in the Quartus editor, you can change it, or turn it off:

-
-
- Recommendation: Unselect the "Autocomplete text".

Quartus remembers those settings, so you don't have to redo them for every lab.

5 Requirements to Pass

The two tasks are demonstrated with three steps each in this lab:

Physical demo - Demonstrate the design on the DE2-115 board.

Software demo - Show the VHDL, the simulation waveform, etc, to the lab assistant.

Discussion - Discuss what is done, and answer possible simple questions from the assistant.

To pass this lab is just a side effect of doing it. What is important here is to learn how to use the tools.

⇒ **In the following labs, you are supposed to be able to redo those steps** ⇐

When you are done, it can be a good idea to clean the lab folder (See sections "Extra: Cleaning").