TSIU03: Lab 1 - Introduction and Lab Tools

Petter Källström, Mario Garrido

August 25, 2015

Abstract

This lab aims to let you test the equipment (the DE2 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

There are two projects to perform. One simple XOR gate, with two switches as input and a LED as output. The other project, called Div5, will test if a four-input number is divisible by five or not. This is described more in details in section 4, "Your Task 2".

1.1 XOR

The XOR gate is simple. In VHDL it can be described as:

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

or simply

```
y <= a xor b
```

where `a` and `b` are the inputs, and `y` is the output.

The functionality is `y=1` if `a`≠`b`, and `y=0` otherwise.

2 Notations

This tutorial uses the following notations:

- Do an action: Processing→Start Compilation | Ctrl+L | ▶ (any of these options work).
- Set values in dialog boxes etc: Name="my_xor"
- Click on button [OK]
- ⟹*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 board.

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

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

3.1 Create a Project

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

3.1.1 Start Quartus

(Start menu)→All Programs→Altera 12.1→Quartus II...→Quartus II ... (64-bit)

⟹ *This should start Quartus II version 12.1*

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: `H:\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 II".
  - Device="EP2C35F672C6".

  (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`".

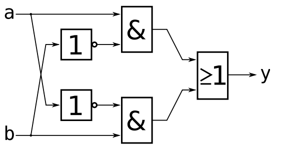
```
library ieee;
use ieee.std_logic_1164.all;

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;
```

"Register transfer level"

Intentional typo. (keep it!)



(xor gate, as it *should* be, without the typo)

Code 1: The complete code for the Tutorial lab.

Save the file as "my_xor.vhd" in `H:\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

(Start menu)→All Programs→Modelsim SE-...→Modelsim | M

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

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="H:/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, like a 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 se 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 t*ns” means the signal is assigned the value x after t ns.
- The “-r *t*ns” means that the given stimuli is repeated every t ns.

Run the Simulation

- Set the “Run Length” field to “400 ns”.
- Click the “Run” button.

```
> run 400ns
```

The waveform result is shown in fig. 1.

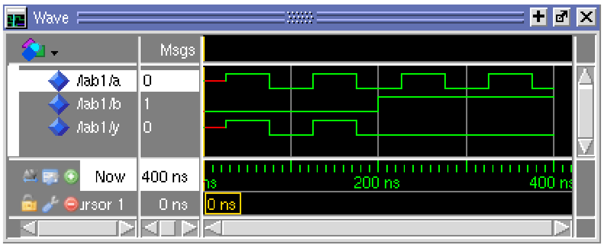


Figure 1: Result of the simulation

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”.

Add→Cursor | A. 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:

- Right click on signal name “a”→Add→New Divider
- Divider Name=Inputs
- (From the transcript: `add wave -divider Input`).

Add another divider:

- Right click on signal name “b”→Add→New Divider
- Divider Name=Outputs

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.

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

- Right click on signal name "`b`"→Properties...
- [Colors...]
- Pick for instance a suitable yellow color.
- [Close]
- [Ok]
- (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.

• Find the Project frame/tab. • Right click on my_xor.vhd →Compile→Compile Out-of-Date.	> `vcom ../my_xor.vhd` (run "`pwd`" to see that you are in a sub directory of the project directory)
Click the "Restart" icon to the left of the Run Length box.	> `restart -force`
Assign `a` and `b` again, just like before.	> `force -freeze a 1 25ns, 0 75ns -r 100ns` > `force -freeze b 0 0ns, 1 200ns`
Run 400 ns again	> `run 400ns`

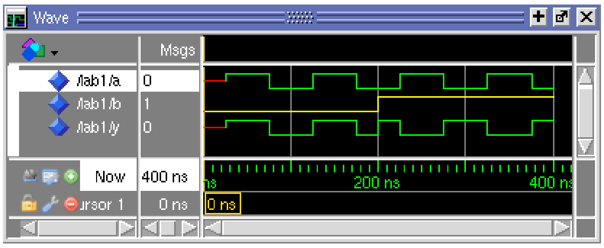


Figure 2: Result of the simulation, when the bug is corrected.

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

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

| Assignments→Pin Planner | Ctrl+Shift+N |.

Which pins to choose

You should use the two right most switches on the DE2 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.

Device	**Pin**	**Groups**	**Device**	**Pin**	**Groups**	**Device**	**Pin**	**Groups**
SW0 ("`a`")	N25	All	LEDR8	AA14	9, 36, 63	LEDG0	AE22	19, 46, 73
SW1 ("`b`")	N26	All	LEDR9	Y13	10, 37, 64	LEDG1	AF22	20, 47, 74
LEDR0	AE23	1, 28, 55	LEDR10	AA13	11, 38, 65	LEDG2	W19	21, 48, 75
LEDR1	AF23	2, 29, 56	LEDR11	AC14	12, 39, 66	LEDG3	V18	22, 49, 76
LEDR2	AB21	3, 30, 57	LEDR12	AD15	13, 40, 67	LEDG4	U18	23, 50, 77
LEDR3	AC22	4, 31, 58	LEDR13	AE15	14, 41, 68	LEDG5	U17	24, 51, 78
LEDR4	AD22	5, 32, 59	LEDR14	AF13	15, 42, 69	LEDG6	AA20	25, 52, 79
LEDR5	AD23	6, 33, 60	LEDR15	AE13	16, 43, 70	LEDG7	Y18	26, 53, 80
LEDR6	AD21	7, 34, 61	LEDR16	AE12	17, 44, 71	LEDG8	Y12	27, 54, 81
LEDR7	AC21	8, 35, 62	LEDR17	AD12	18, 45, 72			

Table 1: The pinout on the EP2C35F672 FPGA used in this 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).

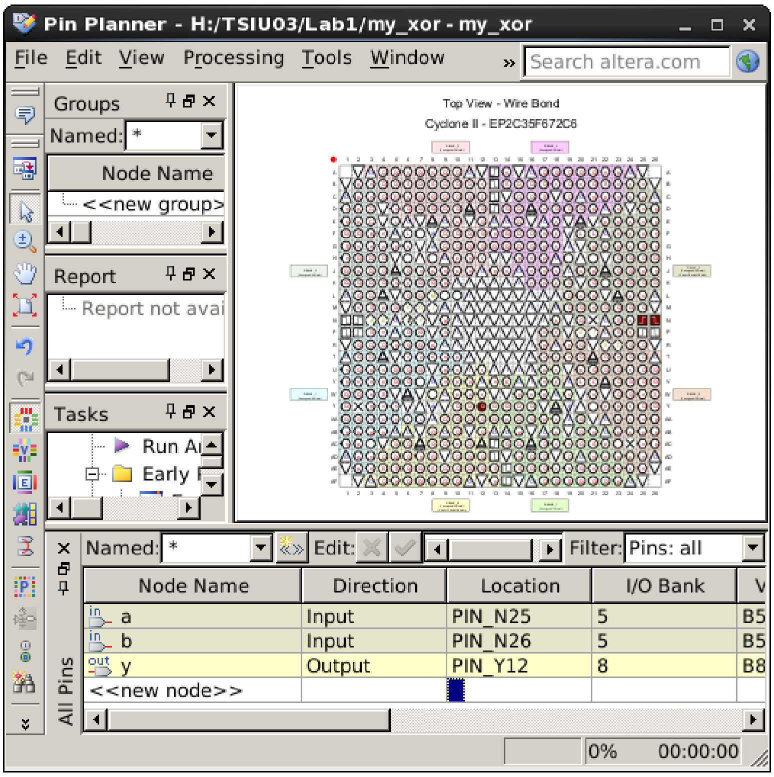
Synthesize the Design Again

| Processing→Start Compilation | Ctrl+L | ▶ |

⟹ *You will now have a "my_xor.sof" file, in folder ouput_files, that contains the configuration data for the FPGA.*

3.6.2 Set up the Network

The DE2 boards are programmed over the network, you have to configure the "Programmer" tool correctly. This must be done every time the DE2 board is moved, or a new board is selected (the programmer remembers your last settings).



Node Name	Direction	Location	I/O Bank	V
in a	Input	PIN_N25	5	B5
in b	Input	PIN_N26	5	B5
out y	Output	PIN_Y12	8	B8
<<new node>>				

Figure 3: Pin Planner, when using LEDG8 as output.

Start "Programmer"

Tools→Programmer

Add the .sof File

If there is no list with file "output_files/my_xor.sof", you should add it.

- [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).

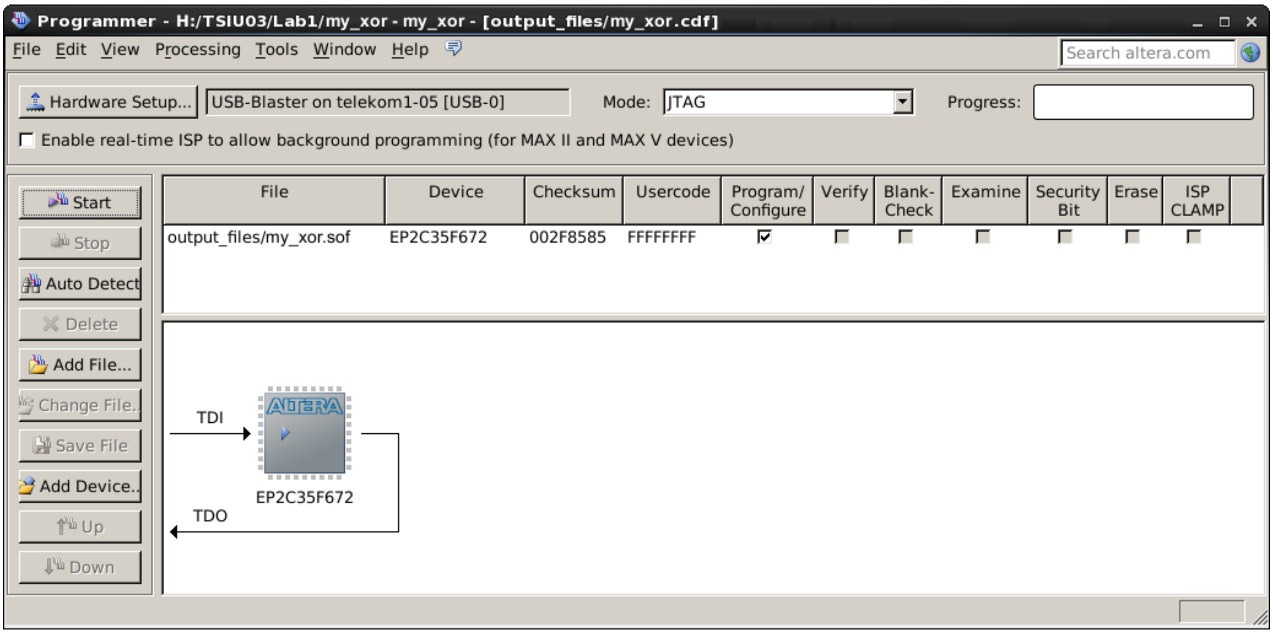
- [Hardware Setup...] In the Hardware Setup dialog:
  - [Add Hardware...] In the Add Hardware dialog.:
    - Hardware type="EthernetBlaster".
    - Server port="1309".
    - Server name and password will be given by the lab assistant.
    - [OK].
  - Currently selected hardware=USB-Blaster on ... .
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

3.6.3 Petters Hand-on Solution: Program the Board

Anyone can program the DE2 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.

Petters hand-on solution to this is as follows (when you want to program the DE2 board):

1. Member x of a lab group walks to the DE2 board.
2. Member y of the same group gives x his/her attention.
3. x waits until the board is free to use.
4. x puts one hand on the board, and gives y a thumb up.
5. y clicks [Start] in the Programmer.



Programmer - H:/TSIU03/Lab1/my_xor - my_xor - [output_files/my_xor.cdf]

Hardware Setup... USB-Blaster on telekom1-05 [USB-0] Mode: JTAG Progress:

□ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
output_files/my_xor.sof	EP2C35F672	002F8585	FFFFFFFF	☑	□	□	□	□	□	□

Figure 4: The programmer, after setup of everything.

6. x, and possibly the entire group, checks the design.
7. When done, leave the board to another group.

If everyone make sure to keep a hand on the board while his/her group programs the board, it should be fairly safe.

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 Chain.cdf?

If you close the Programmer, you are 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, but is a graphical way to describe it, by connecting 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.

Select the VHDL file (so you see 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_scematic.bdf in H:\TSIU03\Lab1 (not in "output_files"), and make sure the "Add file to current project" check box is checked in the "Save As" dialog.

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

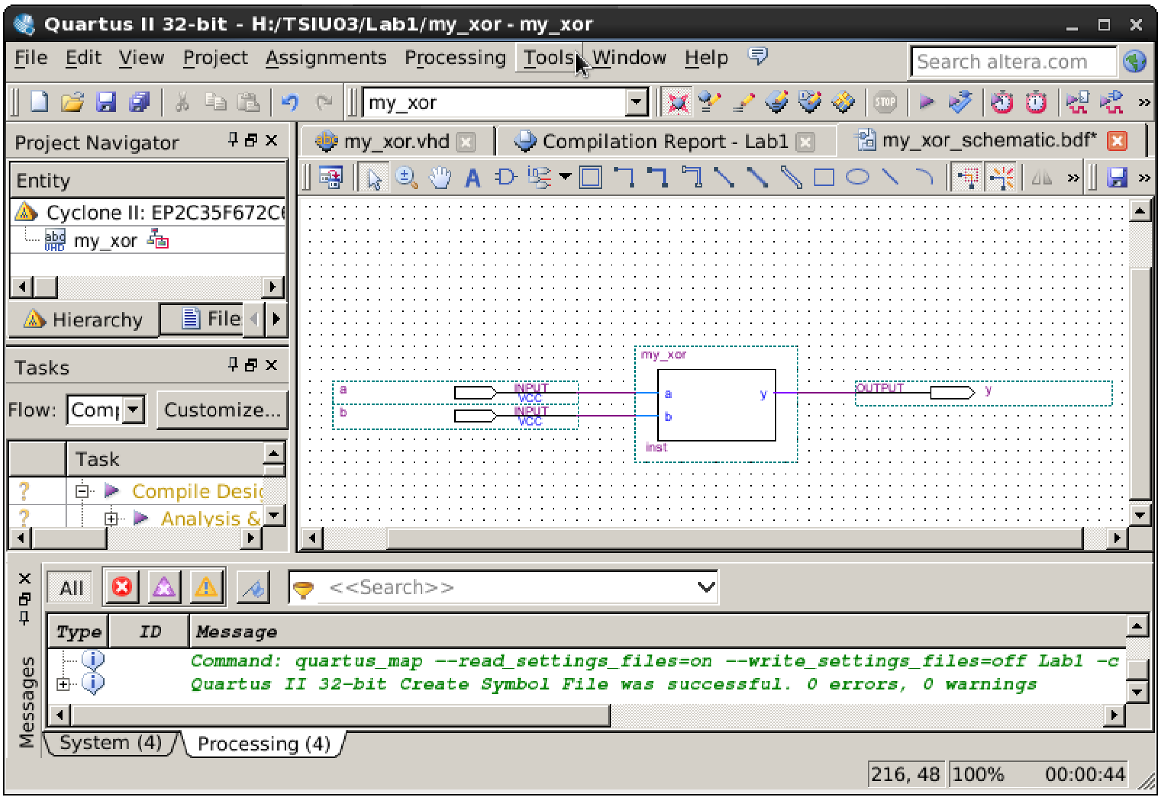
Create schematic pins for all unused ports in "my_xor" in a simpler way:

- Right-click on inst→Generate Pins for Symbol Ports

  ⟹*This will auto format inputs/outputs, vectors, names etc.*
- Drag out the pins from inst, so it looks better.

  ⟹*Wires will be created.*

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



Command: quartus_map --read_settings_files=on --write_settings_files=off Lab1 -c

Quartus II 32-bit Create Symbol File was successful. 0 errors, 0 warnings

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)

ModelSim does not understand Quartus schematics, and hence, 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".

3.7.4 Compile and Test

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

3.7.5 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 do a design of a circuit using the FPGA. This must be done as a new project.

Design a circuit, `Div5` that calculates if a 4-bit unsigned binary number is divisible by 5.

- The binary number is provided by the switches SW3, SW2, SW1, SW0 on the FPGA (SW3 is the MSB). The represented number will range from 0 to 15, i.e., "0000" to "1111" in binary.
- The output will be a led of the FPGA. The led will be ON if the number represented by the four switches is divisible by 5 and OFF if that number is not divisible by 5. For this purpose, consider that 0 is divisible by 5.
- Use the same LED as for the XOR gate.

If you have not presented the XOR gate when creating new Quartus or Modelsim projects, you should start up new instances of those programs.

4.1 Procedure

1. You have to do the hardware design of your circuit, i.e., determine which are the logic gates that calculate the expected function. For this purpose, remember that you can use a truth table as well as a Karnaugh map.
2. You should try to simplify the function to use few logic gates. Note that the smaller the number of logic gates, the simpler the circuit and the easier it is to describe in VHDL.
3. Describe it in VHDL (in a new Quartus project). The module should be called "`Div5`".
4. Simulate it using a new Modelsim project. Combine signals, as explained in sec. 4.2.3.
5. Implement it in the FPGA. The pinout for SW3..SW0 is AE14, P25, N26, N25.

The name of the entity is preferable to be equal to the name of the .vhd file. This avoids conflicts between circuit names.

4.2 Hints

Some hints for the lab.

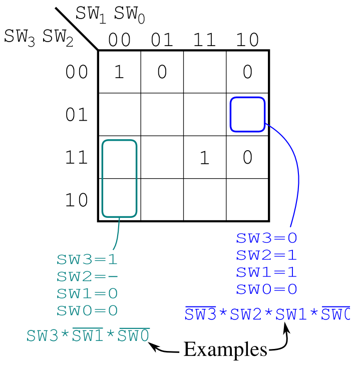
4.2.1 Design Method

A reminder from the switching theory. The typical design stages are performed as

1. Find a truth table.
2. Translate it into a karnaugh map.
3. Make rings in the k-map.
4. Formulate the rings as an AND-OR formula.
5. Can you simplify the formula even more?

SW_3	SW_2	SW_1	SW_0	LED
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	0
1	1	1	1	1

Fill in...



4.2.2 Stimuli

Use the following code to generate the stimuli in Modelsim:

```
> 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
```

Don't forget to add the waves first before running the simulation. You do not have to use dividers, colours etc. in this simulation.

4.2.3 Combine Signals

In the Modelsim wave frame. Select the signals SW0 – 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.

4.3 View 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 looks on the real FPGA.

Tools→Chip Planner.

This is 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. Ctrl+scroll to zoom in the used LAB.

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. Click it.

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

4.4 Auto Completion

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

- Tools→Options...
- Text Editor / Autocomplete Text
- Recommendation: Unselect the "Autocomplete text".

5 Requirements to Pass

The two tasks are demonstrated with three steps each in this lab:
Physical demo - Demonstrate the design on the DE2 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.

For Div5, the discussion includes looking at the design stages, e.g. function table, karnaugh map etc.

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 assumed to be able to redo those steps.