EE1D1: Digital Systems A

Course Lab Manual

2025/2026

Faculty of Electrical Engineering,
Mathematics and Computer Science

TU Delft Delft University of Technology

Challenge the future

During the compilation of every manual, we strive to prevent mistakes and to present the material in an organized and comprehensible manner. Nonetheless, errors and unclarities can occur. If you discover incorrect information, unclarities, or sections that you believe require further elaboration, please inform the responsible teacher. They can incorporate your suggestions and feedback to benefit next year's readers. If necessary they can also clarify parts while the course is running.

Contact Information

Lab Technician
ing. A.M.J. Slats
Email: A.M.J.Slats@tudelft.nl
Tel: +31 15 27 88787
Room: LB 01.260

Lab Technician
ing. M. Schumacher
Email: M.Schumacher@tudelft.nl
Tel: +31 15 27 81850
Room: LB 01.271

Acknowledgements

This lab and manual are the result of the hard work of its many contributors over the past years: ing. B.M. Verdoes, dr. ir. M. Taouil, dr. ir. A.J. van Genderen, ing. X. van Rijnsoever, and ing. A.M.J. Slats.

Contents

CHAPTER 0

Introduction to the EE1D1 Course Labs

Organization and Grading

The course Digital Systems A (EE1D1) has a total of 4 course labs. For each lab 4 hours are scheduled in the Tellegen Hall. Students can work in groups of 2 to complete the labs. The instructions for each lab are all inside this manual. Each lab starts with a set of homework assignments. ***You are strongly recommended to prepare these homework assignments before coming to the lab, to ensure you finish the lab on time.***

For most labs you will need certain files and programs. These can be downloaded from Brightspace. Before continuing, please start by downloading the file `labsDSA.zip`, and unzipping it in a directory on your computer of choice. This file contains all the files you will need for the labs.

Signing off your assignments
The course staff will track which labs you will have completed successfully. The successful completion of each assignment is required to pass the course. You will see the message below at the end of each assignment, telling you to sign off your work with a teaching assistant (TA).

Let the result of the assignment be signed off by the TA now. 

What You Will Learn

During the EE1D1 lectures you've been introduced to SystemVerilog (abbreviated to SV for the remainder of this manual). SV is a Hardware Description Language (HDL). Simply put, you use words and symbols to *describe* a circuit according to the rules set by the language. You can then give that description to other people, and they will know exactly what the circuit does, and how it works.

Not only humans can interpret SV descriptions, computers can too. There exist software programs that are able to read SV files and make models of your circuit. They can then use these models to help you out in your design process. In lab 1, you're going to use one such program, called QuestaSim, to *simulate* the behavior of a circuit. Simulation is an incredibly powerful technique to *verify* whether the circuit you described, functions as intended. Doing verification by hand is tedious and it's easy to make mistakes. For a computer, this is no issue.

In later labs, you're also going to use software tools that are able to build circuits based on your SV descriptions. Ultimately, it's the existence of software tools that *verify* and *synthesize* circuits based on our HDL descriptions, that make HDLs indispensable for the modern digital circuit designer.

Aside from familiarizing yourself with SV and related software tools, you will also do some experiments to learn general lab skills and gain more insight into the behavior of digital circuits.

CHAPTER 1

Lab 1: Combinational Circuits Part I

This lab consists of 3 parts. As mentioned in Ch. 0, there exists a software tool called QuestaSim which can simulate the behavior of circuits described with SystemVerilog. In the first part of this lab, you will use QuestaSim to simulate a simple circuit. In the other parts, you will perform a few experiments to gain more insight in the behavior and design of combinational circuits.

1.1 Homework Assignments (Lab Preparation)

The lab assignments require you to do 3 homework assignments in preparation for the lab session. *You are strongly recommended to prepare these homework assignments before coming to the lab, to ensure you finish the lab on time.*

Homework Assignment 1A

Familiarize yourself with QuestaSim by going through the tutorial in Appendix A.

Homework Assignment 1B.1

Determine the logic expressions for `W`, `X`, `Y` and `Z` in the circuit of Figure 1.1. Which logic function is realized with the circuit?

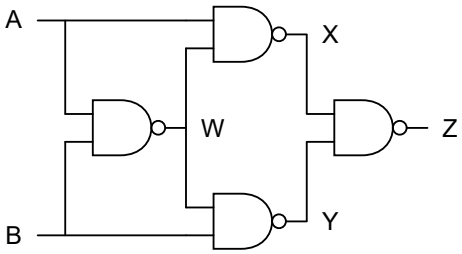


Figure 1.1: A circuit with NANDs

Homework Assignment 1B.2

The expressions of the assignment above give only the behavior in static condition. That is, the state that is finally reached after one or more inputs have changed. However, as we will see in this assignment, when going to the final state, temporarily transient effects (spikes) may occur at some nodes of the circuit.

Suppose that every gate has a gate delay time of t_p = 10 ns (= 10^{-8} seconds). Complete the timing diagram of Figure 1.2 and indicate where spikes will occur.

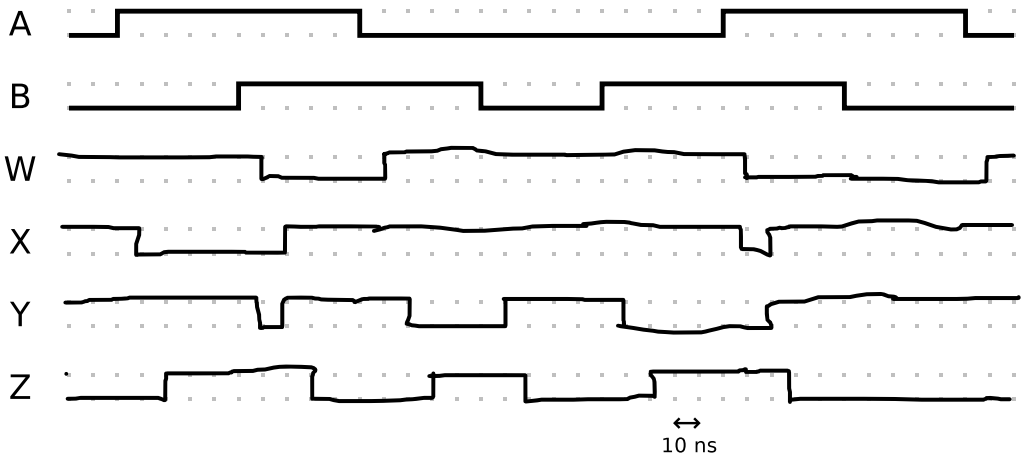


Figure 1.2: Timing diagram

Homework Assignment 1C

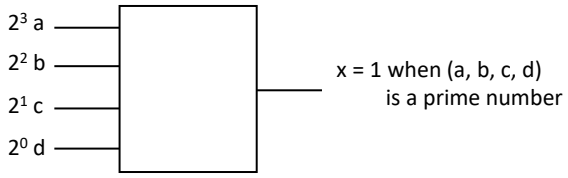
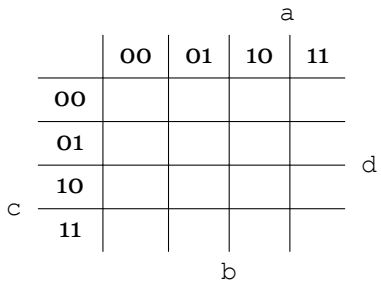


Figure 1.3: Prime number generator

This homework assignment prepares you to design a combinational circuit. The input is a 4-bit binary-coded number / vector that encodes the number sequence 0, 1, ..., 15. (See Figure 1.3) The output `x` should be 1 when the input represents a prime number, and 0 if not. Note that we don't consider the numbers 0, 1, and 2 prime numbers. Complete the following steps:

- Write down the truth table and deduct a minimum sum of products for x with the following K-map.



- Please read the tutorial (Appendix B) on how to use the program 'Espresso'.

Note: if you're unable to install and run Espresso on your own computer, please perform the following steps during the course lab session in the Tellegen Hall.

- Minimize the truth table with Espresso.
- Compare the expression you found yourself with that of Espresso.

Espresso is a two-level minimization program, which means that the minimized circuit has two

levels (AND and OR, apart from any inverters for the input signals). When we allow multiple levels (for example by doing factorisation on the minimum logical expression) we can further reduce the number of components and inputs.

1.2 Lab Assignment 1A: Introduction SV and QuestaSim

In this assignment we will describe a circuit in SV and simulate it with QuestaSim. The circuit realizes the AND function by connecting an inverter to the output of a NAND gate, as shown in Figure 1.4.

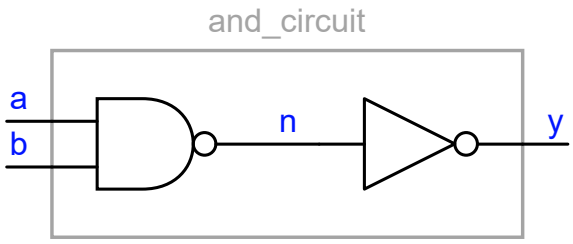


Figure 1.4: The AND circuit

Unzip `labsDSA.zip` (if you did not do so yet) and go to `labsDSA/lab_1/a_introduction`. The files you need in this assignment are `and_circuit.sv` and `and_circuit_tb.sv`. Go through the following steps:

- Open the file `and_circuit.sv`, and go to the module describing the AND circuit. Include the NAND gate and the inverter in the body of the module and map their ports to the ports of the AND circuit.
- Connect the output of the NAND gate to the input of the inverter via a local signal (let's name it `n`). Don't forget to declare this signal!
- Add the AND circuit SV description and the provided testbench to a new QuestaSim project.
- Compile the files, and run a simulation of 250 ns.

Let the result of the assignment be signed off by the TA now. ☑

1.3 Lab Assignment 1B: Spikes

In homework assignment 1B.2, you completed the timing diagram of the circuit 1.1 with gate delays of zero and 10 ns. The presence of a gate delay was able to cause short pulses in the output signal. We call these short pulses "spikes". In this assignment you will simulate the a SV description of the circuit in QuestaSim, and measure a physical copy of the circuit using a function generator and an oscilloscope in the Tellegen Hall.

Spikes in QuestaSim

Open the file `labsDSA/lab_1/b_spikes/spikes.sv`. The delay time chosen here for each NAND is 10 ns. Simulate this file for 400 ns using QuestaSim using the included testbench. Compare the results with your answer for homework assignment 1B.2.

Repeat the simulation, but now make the delay time of each NAND gate 1 ns instead of 10 ns. What happens to the spikes?

Spikes in Real Life

You've been provided with a PCB with the circuit on it. Connect the PCB to the function generator, power supply and oscilloscope as shown in Figure 1.5:

- Use banana-clip cables to connect a power supply of 5V to the PCB: red for VCC and black for GND.
- Apply a block signal (aka square wave) with a frequency of 100 kHz to input A. For the block signal, use the TTL output (0 - 5V) of the function generator and connect this output to the PCB with a BNC-clip cable.
- Make B high by connecting it with a pin to pin wire to VCC.
- Use the oscilloscope to check whether your prediction about spikes in signal X was correct. A probe cable should be used to connect the oscilloscope to the PCB. Check if the probe is set to 1x. Although this is not drawn in Figure 1.5, note that the ground connection of the function generator as well as the ground connection of the probe should both be connected to the ground connection of the circuit.

Let the result of the assignment be signed off by the TA now. 

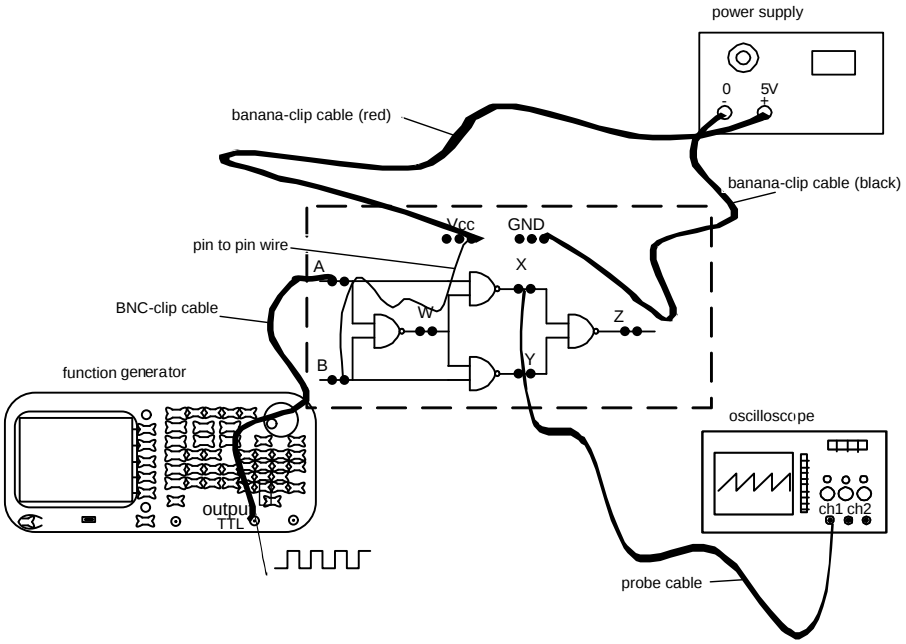


Figure 1.5: Connecting the PCB

1.4 Lab Assignment 1C: Minimization and implementation

Open the file `labsDSA/lab_1/c_minimization`. The file contains module descriptions of the following gates: a 2-input and a 4-input NAND gate, an inverter, and an XOR gate. The file also contains a module called `prime`, which is going to be an SV description of the prime circuit derived in homework assignment 1C.

- If you didn't use Espresso yet to minimize the truth table of the prime circuit (see homework assignment 1C), please do this first.
- Convert the sum of products expression obtained from the K-map or from Espresso, to a NAND-NAND circuit. This requires 4 NANDs with 3 inputs, plus 3 inverters. Try this out.
- Simplify the circuit further by taking an input variable from two product terms of the minimum sum of products out of parentheses, such that an expression results that is the product of that input variable and the XOR operation on the 2 other input variables. This introduces

a third level, but the total number of components required for the circuit is reduced. Do this and check which components are now needed to realize the circuit.

- A good design always starts with drawing the circuit (diagram) on paper. Draw the two circuits (the two-level NAND-NAND implementation and the three-level implementation with the XOR).
- Now choose one of the two circuits to make a structural description in SystemVerilog. Use the file `prime.sv`.
- Test the design with the supplied test bench (included as the module `prime_tb` in `prime.sv`) in QuestaSim for 900 ns.

Let the result of the assignment be signed off by the TA now. 

CHAPTER 2

Lab 2: Combinational Circuits Part II

In this lab, you will create and test an SV description of a circuit that displays the numbers 0 to 7 on a 7-segment display. The inputs will be 8 switches labeled SW7 down to SW0. Assume only one switch is on at a time (one-hot encoding). The display should then show the number of the switch. For example, if you turn on switch SW3, the 7-segment display should show a 3.

A 7-segment display consists of 7 LEDs, which can be used to display numbers and a few letters. Figure 2.1 shows the 7-segment display with each LED segment labeled. Note that the LED controls are inverted! When a LED segment is controlled with a signal value of 0, it is on, and when it is controlled with a signal value of 1, it is off.

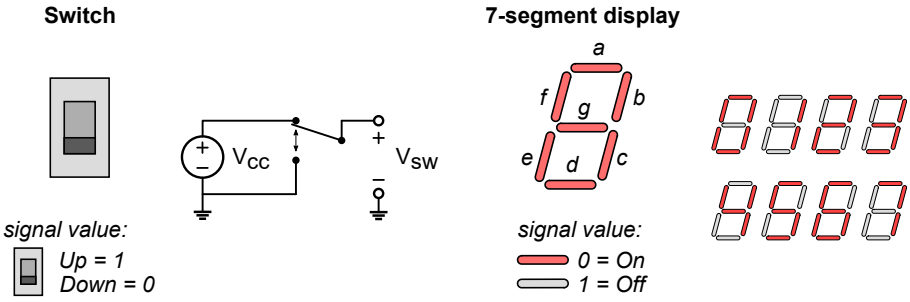


Figure 2.1: Information on the input switches and the output 7-segment display.

The circuit you're going to implement is shown in Figure 2.2. The input switches are the blue signals on the left. The blue signals at the bottom go to the corresponding LEDs in Figure. 2.1. Each LED is on for certain numbers, and off for others.

Here follows an explanation of how the circuit works. Please try to understand it in its entirety.

- An encoder, assuming at most one switch is turned on, outputs the number of the switch that is on in binary. For example, if `sw5` is on, then $y_2y_1y_0$ will be `101`.
- The encoder also outputs a signal `z`. When none of the switches are on, `z` = `0`, turning the output of all NAND gates on, meaning all LED segments are off (remember, their controls are inverted).
- When a switch is on, `z` = `1`, and the NAND gates act as inverters, inverting the output of the multiplexers, so e.g. `a` = a' in that case. The vector $y_2y_1y_0$ is used as a select signal for all 7 multiplexers.
- The output of each multiplexer goes to one of the LED segments of the 7-segment display. For example, if `sw4` is on, each multiplexer will output the signal value that is present at their port labeled `d4`.
- In one of the homework assignments, you will determine what value should be present at every input of every multiplexer. Continuing the example, if `sw4` is on, the signal value at port `d4` of the multiplexer whose output goes to LED segment `a`, should be 0. Do you see why?

As per usual, *you are strongly recommended to prepare the following homework assignments before coming to the lab, to ensure you finish the lab on time.*

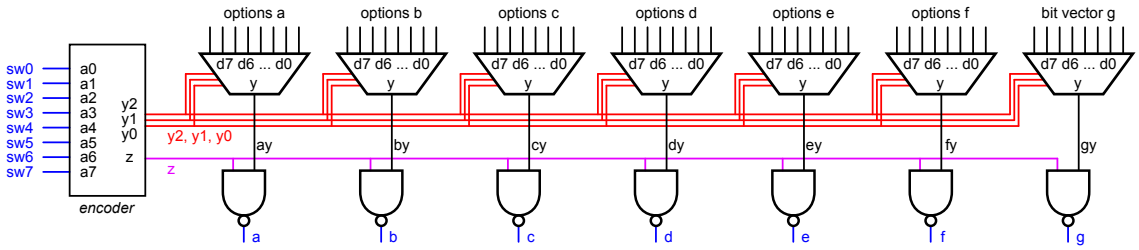
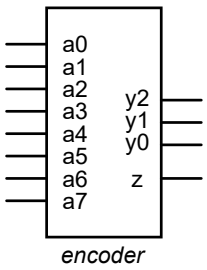


Figure 2.2: Block diagram of the full circuit.

2.1 Homework Assignments (Lab Preparation)

Homework Assignment 2A: 8-to-3 encoder

Given is the following truth table for a 8-to-3 encoder with input ports a_7, a_6, a_5 ... a_0 and output ports y_2, y_1, y_0 and z:



a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	y_2	y_1	y_0	z
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	1	0	0	0	1	0	1
0	0	0	0	1	0	0	0	0	1	1	1
0	0	0	1	0	0	0	0	1	0	0	1
0	0	1	0	0	0	0	0	1	0	1	1
0	1	0	0	0	0	0	0	1	1	0	1
1	0	0	0	0	0	0	0	1	1	1	1

Go through the table and try to understand how the encoder works. Create logic expressions for the output ports y_2, y_1, y_0 and z. Hint: Note that the table doesn't contain all input combinations of $a_7a_6...a_0$, because it's assumed at most one switch is on at a time. You can simplify your expression for the output ports by using OR gates only.

Homework Assignment 2B: 8:1 multiplexer

The logic expression for a 4:1 multiplexer with selection inputs s_1 and s_0, data inputs d_3, d_2, d_1, d_0, and data output y, is $y = s_1 s_0 d_3 + s_1 s_0' d_2 + s_1' s_0 d_1 + s_1' s_0' d_0$

Verify for yourself that this expression is correct. Create the logic expression for a 8:1 multiplexer with selection inputs s_2, s_1 and s_0, data inputs d_7, d_6, d_5 ... d_0, and data output y.

Homework Assignment 2C: 7-segment display

In Figure 2.2, there are 7 multiplexers. Each multiplexer outputs a signal a', b', c', etc. These signals determine the color of each segment of the 7-segment display. The value should depend on which switch is selected, which is encoded by the the value of $y_2y_1y_0$.

The signals $y_2y_1y_0$ are connected to the selection signals $s_2s_1s_0$ of each multiplexer. In the left table below, it is shown how the output y of a multiplexer is detemined by one of its data inputs, depending on the values on its selection signals. Our task is to connect 0's and 1's to the data inputs d0, d1, d2, etc. of each multiplexer, such that each segment has the correct color for the switch that is selected, hence for the value of $y_2y_1y_0$.

In the right table below, you can fill out what should be the values of the different data inputs for each multiplexer. For example, when the switch 3 is selected and number 3 should be displayed, $y_2y_1y_0$ = 011 and segment a (See Figure 2.1) should be on, so a′ should be 1. So, below a′, on the line with 0 1 1, enter a 1. Using this logic, complete the right table below.

y_2	y_1	y_0	y
0	0	0	d0
0	0	1	d1
0	1	0	d2
0	1	1	d3
1	0	0	d4
1	0	1	d5
1	1	0	d6
1	1	1	d7

y_2	y_1	y_0	a′	b′	c′	d′	e′	f′	g′
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0							
1	1	1							

2.2 Lab Assignment 2A: 8-to-3 encoder

Use the logic expression of the 8-to-3 encoder to create a SystemVerilog description of the 8-to-3 encoder from homework assignment 2A. Use the module name `encoder8`. Use the testbench `labsDSA/lab_2/encoder8_tb.sv` to simulate the circuit in QuestaSim for 900 ns.

Let the result of the assignment be signed off by the TA now. 

2.3 Lab Assignment 2B: 8:1 multiplexer

Use the logic expression of the 8:1 multiplexer to create a SystemVerilog of the 8:1 multiplexer from homework assignment 2B. Use the module name `mux8`. Use the testbench `labsDSA/lab_2/mux8_tb.sv` to simulate the circuit in QuestaSim.

Let the result of the assignment be signed off by the TA now. 

2.4 Lab Assignment 2C: Switch to 7-segment display circuit

Create a structural SystemVerilog description of the complete circuit in Figure 2.2. Use the modules you wrote in the previous assignments. Call the top-level module `switch2display`. Use the entries of the truth-table of homework assignment 2C as inputs d_7, d_6, d_5 .... d_0 of each multiplexer that controls a segment. You can just use signal values `1'b0` and `1'b1` at the relevant positions in the port map for the different multiplexer instances. Also create a testbench for the module `switch2display`, such that your QuestaSim simulation ends up looking like Figure 2.3.

Let the result of the assignment be signed off by the TA now. 

2.5 Lab Assignment 2D: Implementation on FPGA

In the previous lab, we explained that there exist software tools that are able to interpret SystemVerilog code and create models of the circuits described in them. One such tool, QuestaSim uses these models to allow you to simulate your SV descriptions. Now, we're going to use a dif-

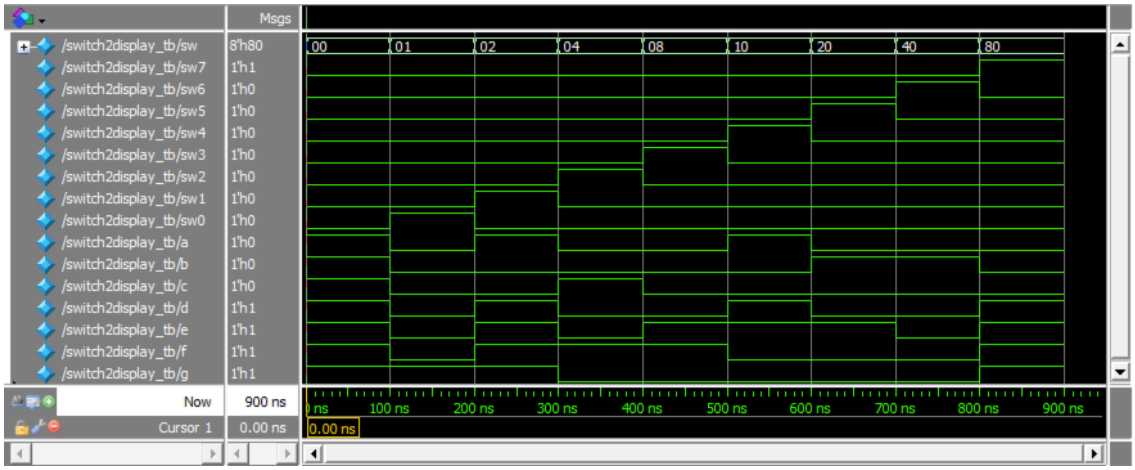


Figure 2.3: Correct simulation result for the switch2display circuit

ferent tool to "build" the circuit described in your SV code! There exist chips, like Field Programmable Gate Arrays (FPGAs), that you can program to behave like the circuit described in an SV description. In this assignment, you're going to implement `switch2display` on an FPGA. By this we mean that you're going to program an FPGA to behave like `switch2display`. The FPGA that we're going to use is part of the Altera DE0 development board (See Appendix D). Aside from the FPGA, this circuit board contains components like switches, leds, and much more to connect to the FPGA in order to test all sorts of circuits.

- Familiarize yourself with the program Quartus by going through the tutorial in Appendix C. Quartus is a software tool that is able to program the FPGA on the Altera DE0 board to behave like your SV descriptions.
- After you complete the Quartus tutorial, make sure to create a new project for the rest of this assignment.

In this case, we will not create a schematic for the circuit to be put on the FPGA, as was done in the tutorial, but we will use the SystemVerilog descriptions instead.

- Use Project -> Add/Remove Files in Project ... to add the Verilog files for `encoder8`, `mux8` and `switch2display` to the project.
- Click on the file with the module `switch2display` and select "Set as top-level entity".
- Run Processing -> Start compilation
- Assign the ports of the module `switch2display` to the FPGA pins as shown in Table 2.1.
- Rerun Processing -> Start compilation and next program the FPGA.
- Test the working of the switch to display converter on the FPGA board.

Let the result of the assignment be signed off by the TA now. 

Table 2.1: Pin mapping for switch to display converter

Port	In/out	Pin	DE0 Peripheral
a0	input	PIN_J6	SW[0]
a1	input	PIN_H5	SW[1]
a2	input	PIN_H6	SW[2]
a3	input	PIN_G4	SW[3]
a4	input	PIN_G5	SW[4]
a5	input	PIN_J7	SW[5]
a6	input	PIN_H7	SW[6]
a7	input	PIN_E3	SW[7]
a	output	PIN_E11	HEX0_D[0]
b	output	PIN_F11	HEX0_D[1]
c	output	PIN_H12	HEX0_D[2]
d	output	PIN_H13	HEX0_D[3]
e	output	PIN_G12	HEX0_D[4]
f	output	PIN_F12	HEX0_D[5]
g	output	PIN_F13	HEX0_D[6]

CHAPTER 3

Lab 3: Sequential Circuits (Structural)

In this third part of the Digital Systems A course labs we will start designing sequential circuits in SystemVerilog (SV). All the files needed for this lab can be found again in `labsDSA.zip` (downloadable from Brightspace).

During this lab you will:

- Implement a sequential circuit with flip-flops.
- Design, simulate and test an FSM from scratch.

As per usual, *you are strongly recommended to prepare the following homework assignments before coming to the lab, to ensure you finish the lab on time.*

3.1 Homework Assignment (Lab Preparation)

Homework Assignment 3A: 2-Bit Counter

In this assignment you're going to design a 2-bit counter which you will have to simulate in the lab using QuestaSim. The 2-bit counter should count up from 0 to 3 (denoted by count states C0 till C3 respectively) and is controlled by an external "enable" signal (`E`). If `E` is `1`, the counter should increment on the rising clock edge. If `E` is `0`, the counter should retain its value. The finite state diagram of the counter is shown in Figure 3.1.

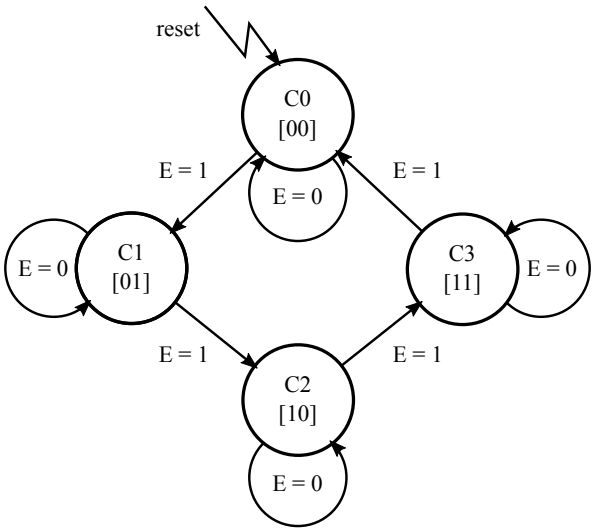


Figure 3.1: Finite State Diagram for 2-Bit Counter

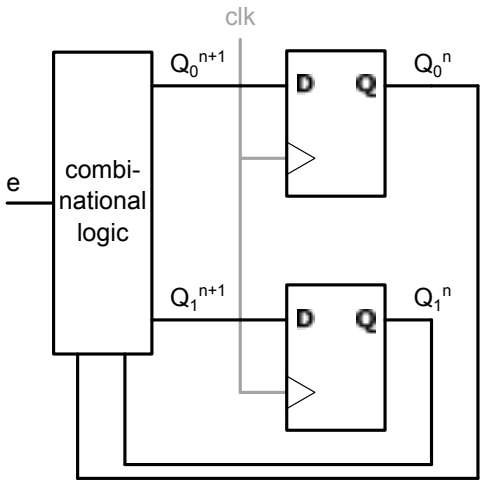


Figure 3.2: Block diagram of a 2-bit counter with enable.

To be able to create the counter in hardware, you need 2 flip-flops to remember the current state, and a combinational network to compute the next state (see Figure 3.2). Verify the state table (see Table 3.1), i.e., check if the values for the next state are correct.

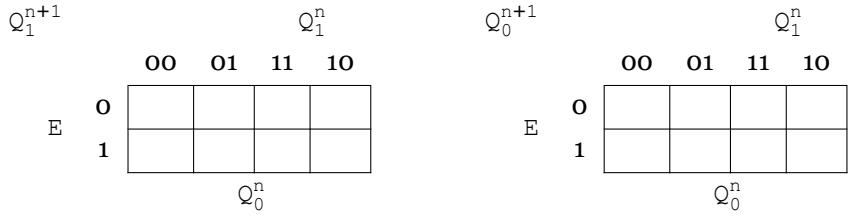
- Derive expressions for Q_1^{n+1}, Q_0^{n+1} from the state table in Figure 3.1. The K-maps of Table 3.2 can be used for that.
- Since $D_i = Q_i^{n+1}$ $(i = 0, 1)$, you should now have all the information to draw the counter circuit. Draw the circuit using gates and flip-flops only. The clock and reset signals don't

Table 3.1: State Table for 2-Bit Counter

	Current state		Input	Next state	
	Q_1^n	Q_0^n	E	Q_1^{n+1}	Q_0^{n+1}
C0	0	0	0	0	0
C0	0	0	1	0	1
C1	0	1	0	0	1
C1	0	1	1	1	0
C2	1	0	0	1	0
C2	1	0	1	1	1
C3	1	1	0	1	1
C3	1	1	1	0	0

need to be included in the drawing because their connections are trivial and would only clutter the drawing.

Table 3.2: K-maps for Q_1^{n+1} en Q_0^{n+1}



Homework Assignment 3B: FSM Sequence Detector

In this assignment, you're going to design a sequence detector. A block diagram that shows the inputs and outputs of this detector are shown in Figure 3.3.

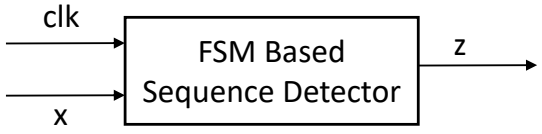


Figure 3.3: Block Diagram of Sequence Detector

The output (`z`) is 1 when the detector detects a certain sequence on the input (`x`) and `z` is 0 otherwise. The input `x` is sampled at each positive clock edge and forms a sequence over time. The sequence that you have to detect in this assignment is `1001`. In Figure 3.4 you can see the state diagram of the detector.

1. Analyze the finite state diagram and try to understand how it works.

Each state is encoded by a state vector $y = y_2y_1y_0$. You can find the state encoding in Table 3.3. *You are required to use these state encodings. Please, don't come up with your own.*

Paired with this description of this assignment is an MS Excel sheet which guides you through the exercise. This Excel sheet can be found in `labsDSA/lab_3/b_detector/fsm_assignment.`

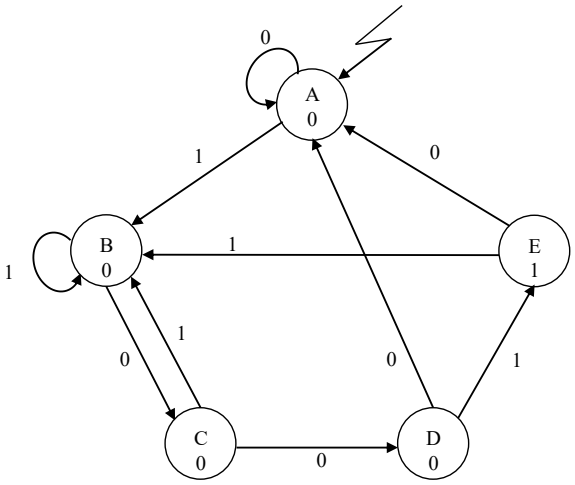


Figure 3.4: State Diagram

Table 3.3: State Encoding Table

	y_2	y_1	y_0
A	0	0	0
B	0	1	1
C	0	1	0
D	1	0	1
E	1	0	0

xlsx. In it you can find the state table, empty Karnaugh maps, and further instructions and requirements. You can start filling it in from top to bottom, from left to right. The exercise is completed once you have obtained correct logical expressions for y_2+, y_1+, y_0+ and z in the form AOI, NAND, AOI and NOR respectively as requested in the Excel sheet. Other than these components, you are only allowed to use 4 inverters to create the inverted signals of for y_2+, y_1+, y_0+ and z, i.e., for y_2+', y_1+', y_0+' and z'.

Note that for current state and next state variables we sometimes use a notation like y_i and y_i^+, and at other times we use y_i^n and y_i^{n+1}.

Below you find some detailed instructions regarding the usage of the Excel sheet.

2. Complete the state table (cells G7:K22) by providing the symbol for the next state, the next state values, and output z. In case of don't cares use capital X. This applies both for the next state symbols as the state variables. (For the current state symbol, also the X has to be used sometimes when the combination of state bits does not correspond to valid state in the state encoding table.)
3. Next, fill in the 8 Karnaugh maps numbered from [1] to [8] on rows 25 to 41 for y2+, y1+, y0+, z, y2+', y1+', y0+' and z', respectively. The values can be derived from the state table you completed in the previous step.
4. Based on the required gate implementation, decide if you are going to use the inverted or non-inverted value for respectively y2+, y1+, y0+, and z, and whether you should derive a minimal product-of-sums or a minimal sum-of-products.
5. Thereafter, derive the required expressions and draw the circuit.

An answer sheet will be provided to you on Brightspace prior to the start of this lab. You can check your answers by copying all cells from your excel sheet (ctrl + a followed by ctrl + c) to the answer

sheet. Mistakes will be indicated in red on the answer sheet.

3.2 Lab Assignment 3A: 2-Bit Counter

In this assignment, you are going to implement the 2-bit counter from homework assignment 3A. Let a TA verify your circuit diagram before you continue with the next steps:

1. Using the given file `labsDSA/lab_3/a_counter/counter.sv`, make a *structural* SystemVerilog description. Note: a structural description contains only instances of other modules, NO logical expressions. First, analyze the inputs and outputs of the module `counter`. The flip-flop outputs of states Q_1^n and Q_0^n (see Table 3.2) have to be connected to `count[1]` and `count[0]` respectively. The SV file already has all the necessary sub-components, so only the signals and the port maps have to be made. Create a QuestaSim project, complete the SystemVerilog description and compile it.
2. Simulate the structural SystemVerilog description with the provided testbench `labsDSA/lab_3/a_counter/counter_tb.sv`. Run the simulation for 4 us.

Let the result of the assignment be signed off by the TA now. 

3.3 Lab Assignment 3B: FSM Sequence Detector

In this assignment, you're going to implement the FSM sequence detector (see homework assignment 3B). The steps are as follows:

1. Implement the FSM in *structural* SystemVerilog by completing the file `labsDSA/lab_3/b_detector/fsm_assignment.sv`. Some components that you have to use such as NAND / NOR / AOI / OAI / Inverter gates can be found in the file `labsDSA/lab_3/b_detector/fsm_assignment_primitives.sv`. Make sure to add both files to your QuestaSim project.
2. Add the testbench `labsDSA/lab_3/b_detector/fsm_assignment_tb.sv` to your QuestaSim project. Complete the testbench such that the simulation runs through every possible state transition at least once. Make sure to change the value of `x` on the falling edge of the clock (any point in time that isn't the rising edge of the clock should be fine too) to avoid potential timing issues in simulation.

Let the result of the assignment be signed off by the TA now. 

CHAPTER 4

Lab 4: Sequential Circuits (Behavioral)

In this fourth session of the Digital Systems A course labs we will first implement the structural description of the counter from lab assignment 3A on an FPGA. Next we will create a *behavioral* description of the counter and also implement this description on the FPGA. Finally, we will create a behavioral description for the FSM sequence detector from lab assignment 3B.

There are no homework assignments for this lab. But you are strongly advised to study the lecture on behavioral SystemVerilog descriptions.

4.1 Lab Assignment 4A.1: Counter on FPGA (Structural)

Go through the following steps to implement the 2-bit counter from lab assignment 3A on an FPGA board:

1. Create a new Quartus project for this assignment.
2. Create a new schematic by clicking on File → New and next selecting *Block Diagram/Schematic File*, and implement the counter circuit according to the description as obtained in lab assignment 3A. Use components from the library `./quartus/libraries/primitives` (pin, logic and storage).
3. Assign the circuit ports to the FPGA pins as shown in Table 4.1. Figure 4.1 shows how they are mapped on the FPGA board.

Table 4.1: Pin mapping for counter

Port	In/out	Pin	DE0 Peripheral
clk	input	PIN_F1	BUTTON[2]
reset	input	PIN_D2	SW[9]
E	input	PIN_J6	SW[0]
Q1	output	PIN_J2	LEDG[1]
Q0	output	PIN_J1	LEDG[0]

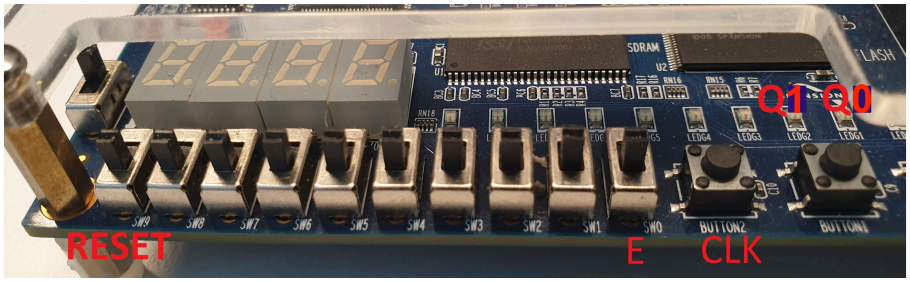


Figure 4.1: Pin mapping on FPGA board

4. Test if the counter works on the FPGA board.

Let the result of the assignment be signed off by the TA now. ☑

4.2 Lab Assignment 4A.2: Counter on FPGA (Behavioral)

Below a behavioral description of a 2-bits counter is given. Replace the logic gates and the flip-flops in the structural SystemVerilog file you created in lab assignment 3A, by this behavioral description (NB copying from the pdf text will not work). Next, extend the code of the always_comb block such that the counter will only count when the signal enable is high, similar as with lab assignment 3A.

```
logic [1:0] next_count;

always_ff @(posedge clk) begin
    if (reset)
        count <= 0;
    else
        count <= next_count;
end

always_comb begin
    next_count = count + 1;
end
```

The flip-flops that were used in the structural description for the counter had an asynchronous clear/reset. What type of reset does the behavioral description of the counter have.

Simulate the circuit using the testbench from lab assignment 3A. Next, use the behavioral description for the counter to implement the counter on the FPGA.

Let the result of the assignment be signed off by the TA now. 

4.3 Lab Assignment 4B.1: FSM Sequence Detector (Behavioral)

Create a behavioral description of the sequence detector FSM from lab assignment 3B, and simulate it with the same testbench of that session from `labsDSA`.

Let the result of the assignment be signed off by the TA now. 

4.4 Lab Assignment 4B.2: Sequence Detector on FPGA

This assignment is optional.

Implement the FSM sequence detector on the FPGA. Use the following pin assignments.

Table 4.2: Pin assignment for assignment 4B.2

Port	In/out	Pin	DEo Peripheral
clk	input	PIN_F1	BUTTON[2]
reset	input	PIN_D2	SW[9]
x	input	PIN_J6	SW[0]
z	output	PIN_F2	LEDG[4]

APPENDIX A

Tutorial QuestaSim

Overview

QuestaSim is a popular software tool that is used to simulate and debug hardware descriptions written in HDLs like SystemVerilog (SV). The goal of this tutorial is to learn how to use the most important basic features of QuestaSim. We will use files from `labsDSA.zip` (found on Brightspace). Download this file and extract it in a directory of your choice.

On Brightspace (under Resources > SystemVerilog Simulation > Install QuestaSim). it is described how you can use QuestaSim on your own PC. If you're working on the PCs in the Tellegen Hall, QuestaSim will come pre-installed. You can find it under Start > QuestaSim-64 10.6g > QuestaSim. By opening the program, you should see a window like the one in Figure A.1.

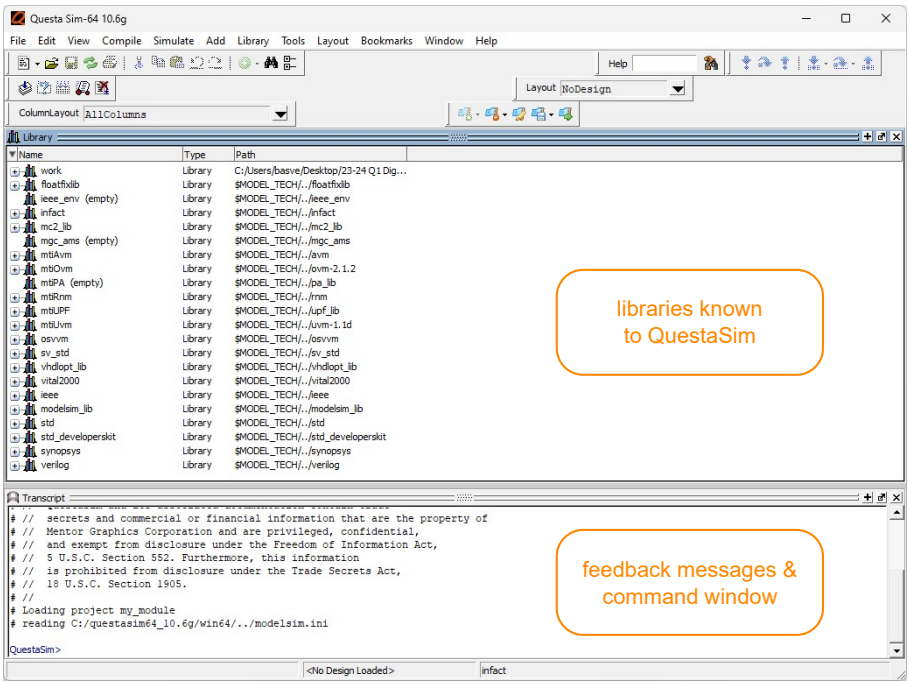


Figure A.1: The main window of QuestaSim

You can see two stacked windows. The Library window contains a list of HDL libraries that are understood by QuestaSim. The Transcript window functions as a command terminal. You can type commands here for the program to execute, and the program will give you feedback messages. Note that you don't *have* to use the Transcript window. Most commands can be given to QuestaSim via the tabs on the top of the window (e.g. File, Edit, View, etc.) instead.

Project Set-up

After extracting the `labsDSA.zip`, go to `labsDSA/tutorial_questasim` to find the SV files for this assignment. To add these files to QuestaSim, we need to make a project and add the SV files to this project. In QuestaSim, go to File > New > Project. Select as Project Location the directory `labsDSA/tutorial_questasim` and give the project a Project Name (e.g. "tutorial_questasim").

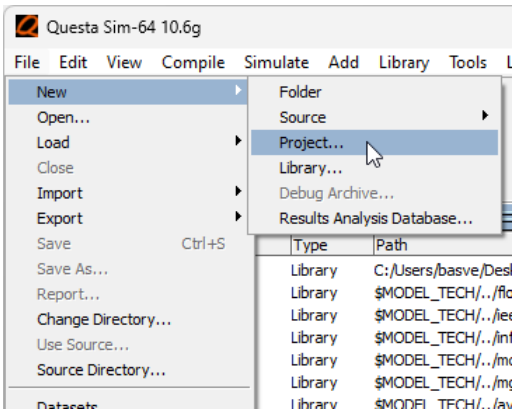


Figure A.2: File > New > Project

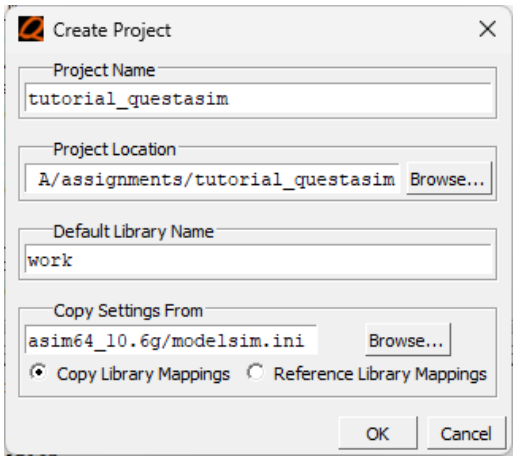


Figure A.3: Specify the project name and directory.

After clicking OK, an "Add Items to the Project" pop-up will appear. Click "Add Existing File" and select the files `my_module.sv` and `my_module_tb.sv`. Click OK, and then close the window.

Editing & Compilation

View the contents of the file by right-clicking on the file name `my_module.sv` and selecting Edit in the pop-up menu, or by clicking twice on the name. QuestaSim will open the file in a new tab.

Editing SystemVerilog Files
While you can edit SV files directly in QuestaSim like this, the in-built text editor can be very annoying to work with. There exist other text editors (e.g. Notepad++, and Visual Studio Code) that are much more user friendly. Note that SV files are plain text files. You can edit them in an external editor and, as long as you save your changes, QuestaSim will be able to use them.

Before we can simulate `my_module`, we need to compile it first. During compilation QuestaSim checks if the syntax of the file is correct, and makes a model for the simulation of the circuit. To compile a file, right-click it and select Compile > Compile Selected.

Since the file we provided you with contains a syntax error, a red cross will appear in the Status tab right of the file name. Double-click the red cross, and a pop-up will appear with a description of the error. The error message says that the compiler was scanning the code, and found the keyword `endmodule` on line 10, while it expected to find a `;` before that. Try to infer this from the error message yourself. After that, add the missing `;` and save the file. The red cross should change into a blue question mark. This means that there have been uncompiled updates to the file. Recompile the file, and a green check mark should appear, indicating that the compilation was successful. Also compile the file `my_module_tb.sv`.

Running a Simulation

Go to the Library tab and click the plus sign to the left of the work library (see Figure A.4). Right-click `my_module_tb` and select "Simulate without Optimization". The QuestaSim window should now look something like Figure A.5.

Four new windows appeared: sim, Objects, Processes (Active) and Wave. The sim tab shows the module hierarchy of your design. The hierarchy in this tutorial seems simple enough: the module `my_module` is included in (is a sub-module of) `my_module_tb`. However, the complete hierarchy

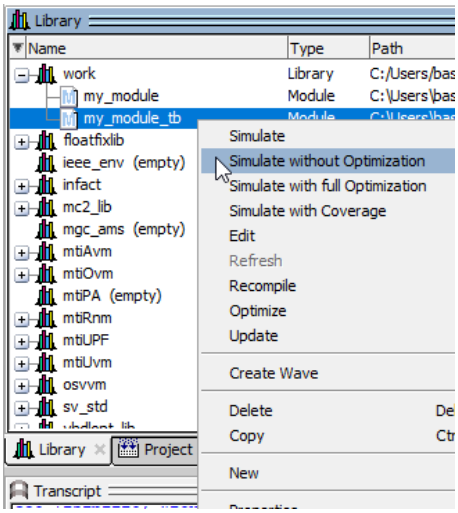


Figure A.4: Start the simulation.

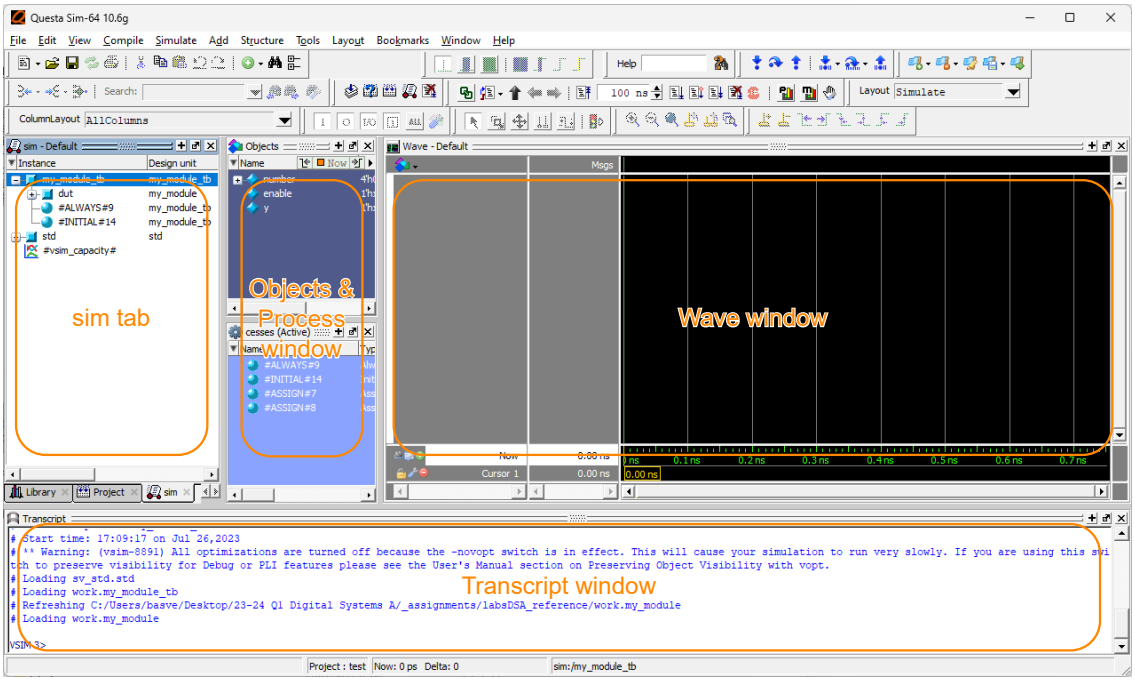


Figure A.5: The simulation window of QuestaSim

looks more like shown in Fig. A.6. The grey blocks are called “processes”. These are the so called “leaves” of the tree diagram of the hierarchy. Processes are blocks of code that add functionality to a design. In contrast, modules merely combine and connect processes (and/or other modules) with each other. The name of each process contains the type (assign, initial, always) and the line number at which it appears in your code. If you haven’t heard of some of these process types yet, you’ll learn about them in one of your next lectures.

The sim tab allows you to navigate the design hierarchy. If you click on a module or process, the internal signals of those units appear in the Objects window, and get highlighted in the Processes window. In the Objects window you can select certain signals, right-click them, and add them to the Wave window. Understanding, and being able to navigate these windows will make your life

much easier once you start simulating or debugging larger systems!

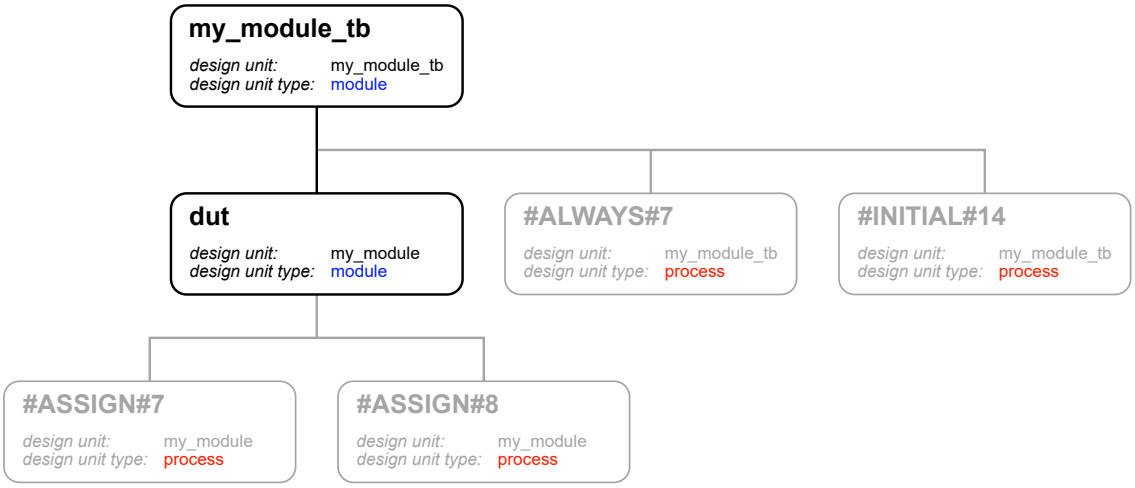


Figure A.6: Hierarchy of the simulation made by QuestaSim.

Click on `my_module_tb` in the sim window, select all signals that appear in the Objects window, right-click them, and select "Add Wave". The signals should now appear in the Wave window as shown in Fig. A.7. Now, to run a simulation, go to the Transcript window at the bottom of the page, and type "run 320 ns". Press enter, and if everything went correctly, you should see waveforms in the Wave window. Click on the Wave window, and press F (zoom fit) to view the entire simulation. It should look like Fig. A.8.

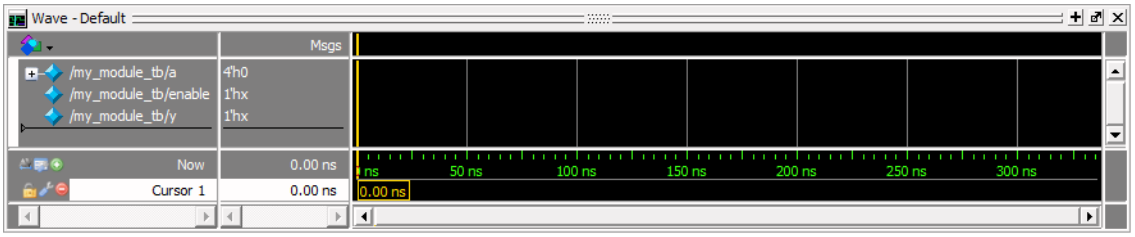


Figure A.7: Wave window with signals defined in `my_module_tb.sv`

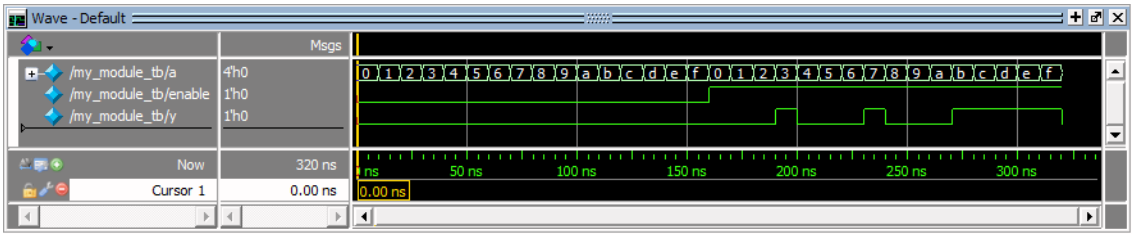


Figure A.8: Simulation result, zoomed out to fit the complete simulation.

And there we go! You now know how to simulate SystemVerilog descriptions of digital circuits in QuestaSim!

APPENDIX B

Tutorial Espresso

Logic synthesis is the mapping of a logical expression (e.g. in SystemVerilog) to a digital circuit. The first part of this is often to simplify the logical expression. The program espresso can be used to calculate a two-level simplification of two-level expressions. Below follows a brief explanation how to work with espresso.

The program works originally in a MS-DOS window. However, a graphical user interface (gui) exists, that makes it easier to work with it. The graphical interface can be opened by starting `espresso_gui`, which is located in the map `C:\Programs`. Then the window shown in Figure B.1 will open.

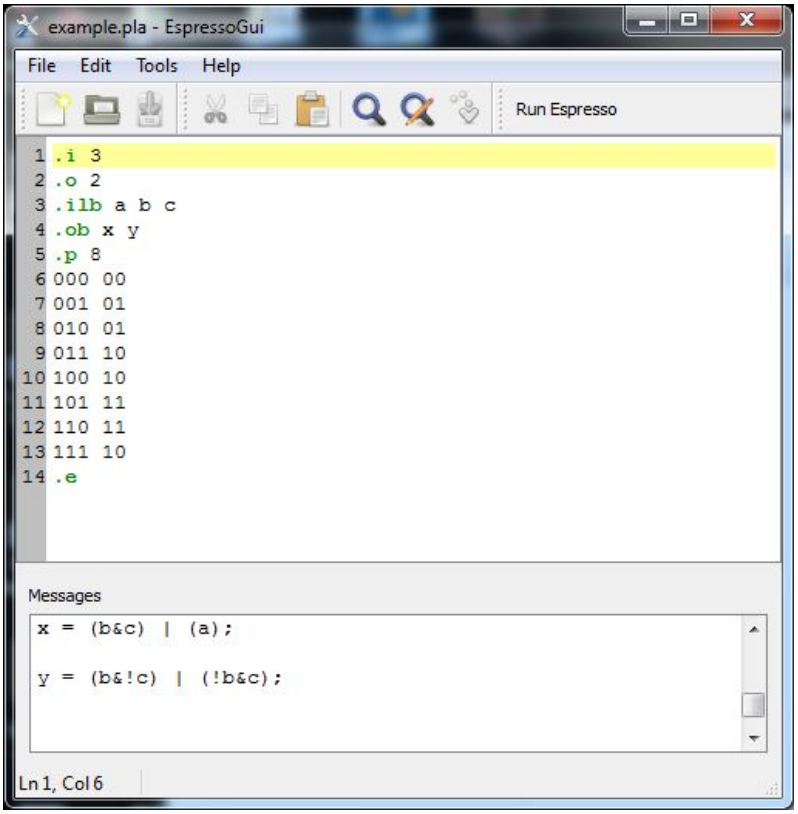


Figure B.1: Graphical User Interface for Espresso

The upper frame is the input frame and the lower frame the output frame. The input for espresso is a text file, see e.g. the file `example.pla`, where the input expressions are written in the form of a truth table (`.pla` is in general conventions the file-extension for truth tables). Create/edit such a file in the input frame or use File -> Open to open an existing file (e.g. file `example.pla`). When using the key F9 or the button Run Espresso, a two-level simplification will be executed and the result will be printed on the output frame.

As an example, the following truth table

a	b	c	x	y
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

can be coded as input for espresso in the following way:

```
.i 3       # declare 3 input variables
.o 2       # declare 2 output variables
.ilb a b c # name the 3 input variables a, b, c
.ob x y    # name the 2 output variables x, y
.p 8       # after this there are 8 input combinations

000 00
001 01
010 01
011 10
100 10
101 11
110 11
111 10
.e         # mark end of input-file)
```

Note that the related (not yet simplified) expressions for x and y are given by:

```
x = a'bc + ab'c' + ab'c + abc' + abc
y = a'b'c + a'bc' + ab'c + abc'
```

With the Run Espresso command, espresso will generate the following screen-output:

```
x = (b&c) | a
y = (b&!c) | (!b&c)
```

which is identical to

```
x = bc + a
y = bc' + b'c
```

This is indeed a proper two-level simplification compared to the original expressions.

APPENDIX C

Tutorial Quartus II

This appendix describes how to program the Altera FPGA using the Quartus II program. We will create a simple circuit diagram that contains a single inverter. The output signal of the inverter can be observed through a LED when we apply an input signal to the inverter through a push button.

Start the Quartus II program so that the window opens as shown in Figure C.1.

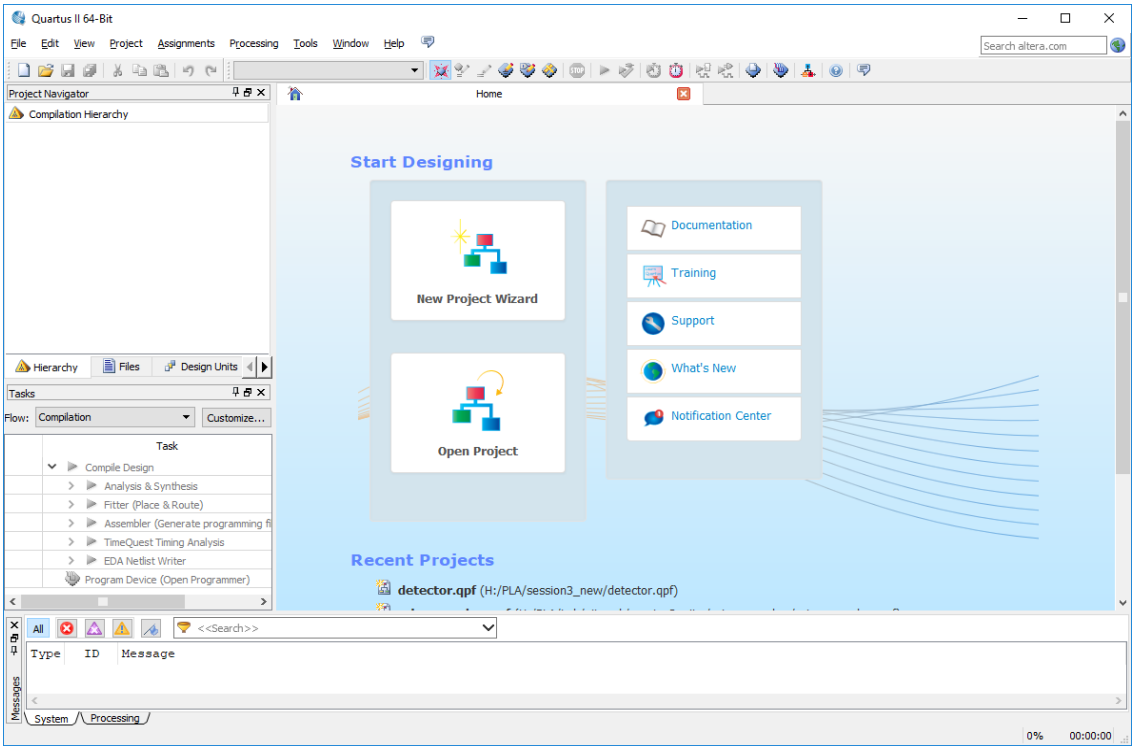


Figure C.1: The Quartus II IDE

Like most IDE programs, also Quartus II is based on working with projects. Start by creating a new project using File → New Project Wizard. Enter the following information:

- Working directory for this project: specify a map where you have write access, i.e. a map under the H: partition.
- Name of this project: e.g. *qtutorial*
- Name of the top-level design entity for this project: e.g. *qtutorial*

Then, click on Next. The window ”Add Files” will open. No files will be added, so click on Next. The window ”Family & Device Settings” will open. Select *Cyclone III* and *EP3C16F484C6*. Then, click on Finish

Now the schematic design can be made. Create a new worksheet by clicking on File → New and next selecting *Block Diagram/Schematic File*. Place an inverter on the work sheet by clicking the right mouse button, selecting Insert → symbol, selecting component ’not’ under ../quartus/library/primitives/logic, clicking the OK button and finally placing the component in the middle of

the worksheet.

Besides the 'not' component, also input and output labels need to be placed that deliver the input and output signals. The labels can be selected by clicking again with the right mouse button, selecting Insert → symbol, and then selecting either an 'input' component or an 'output' component under ../quartus/library/primitives/pin. Place one 'input' pin to the left of the inverter, and place two 'output' pins to the right of the inverter. Change the label names (by selecting the pin label, right clicking the mouse button and then selecting Properties) to 'A' for the input pin, and to 'AINV' and 'ASAME' for the ouput pins.

Use the Orthogonal Node Tool button to connect the input and output labels with wires to the inverter as shown in Figure C.2. Save the file as *qtutorial.bdf*.

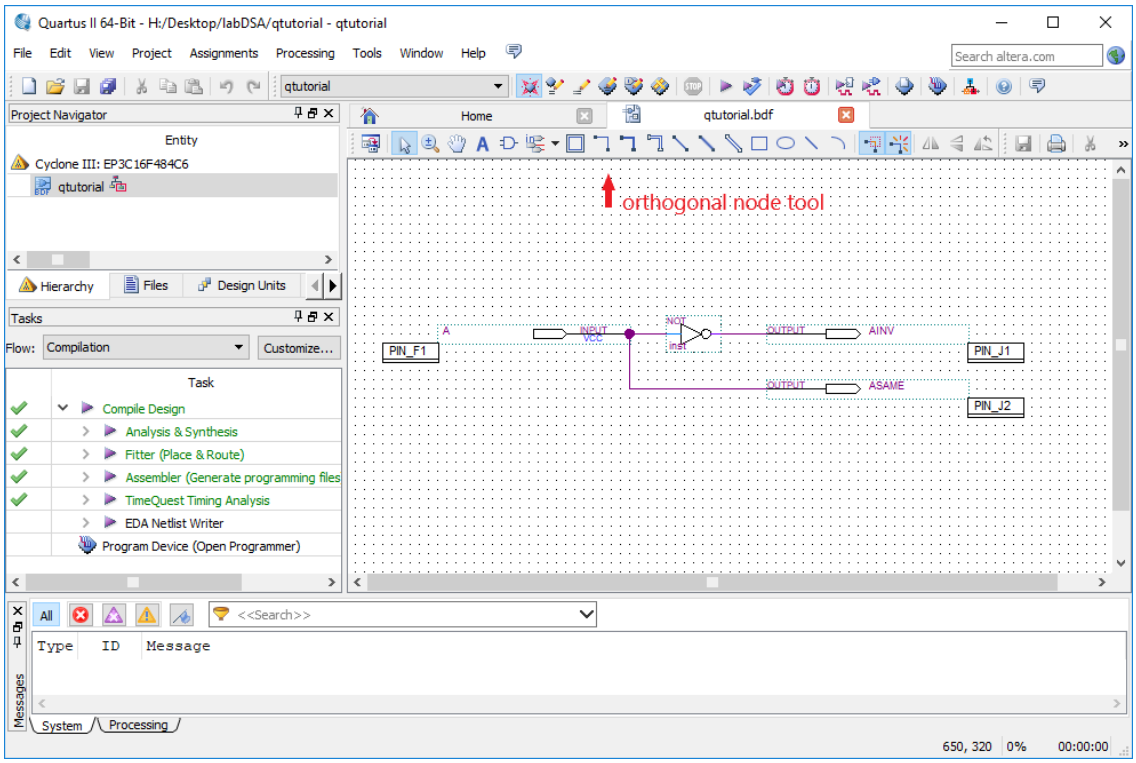


Figure C.2: Inverter in the schematic window of Quartus

Now the pin assignment must be made. The pin assignment connects the input and output labels to the pins of the FPGA. First run a compilation using Processing → Start Compilation. Now the pins will be visible when you start the pin planner using Assignments → Pin Planner. Find the Node Name 'A' and fill in the location PIN_F1. This way A will be connected to the pin PIN_F1 of the FPGA, which is the pin that is connected to push button 2 on the FPGA board. For nodes 'AINV' and 'ASAME', respectively use locations PIN_J1 and PIN_J2 to respectively connect them to LED LEDG0 and LED LEDG1. See also Figure C.3. For more information about pin numbers and pin connections, see Appendix D.

The entire design should now be recompiled. To compile click Processing → Start Compilation. Now the FPGA can be programmed. Click on Tools → Programmer. The programming window opens, see Figure C.4. Make sure the USB-Blaster is selected in the Hardware Setup. By clicking on the Start button the FPGA will be programmed with the file qtutorial.sof, see Figure C.4. Test if your FPGA works as you expected by pushing push button 2 and observing the behaviour of LEDs LEDG0 and LEDG1.

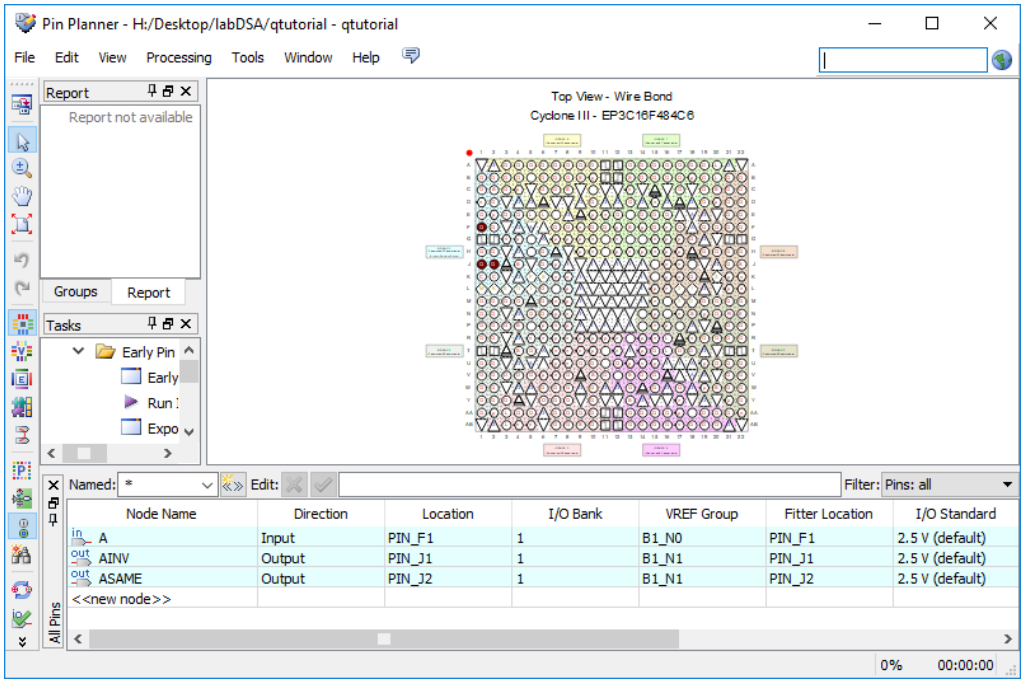


Figure C.3: Editing the pin assignments

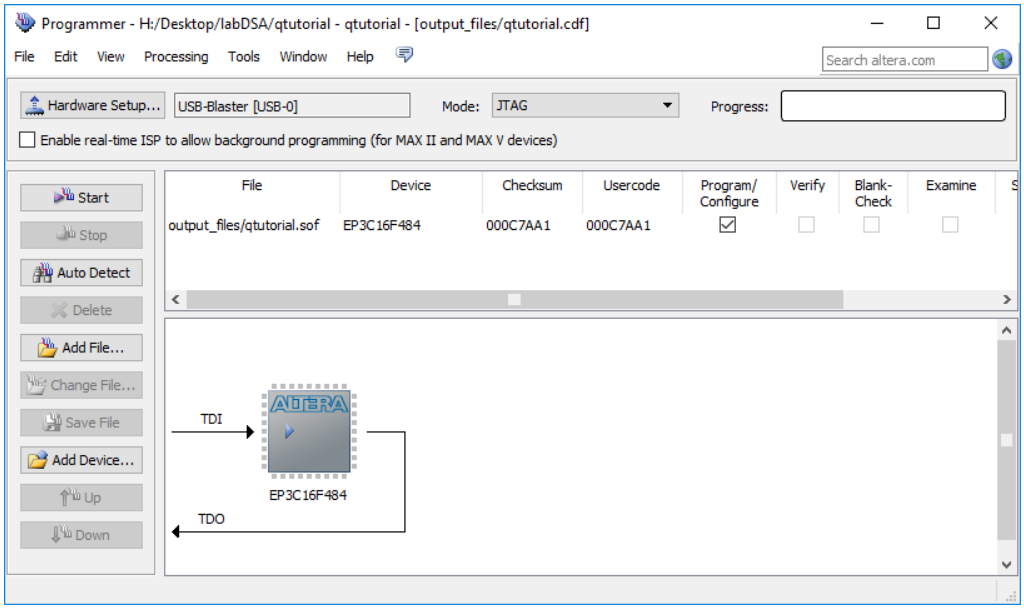


Figure C.4: The Quartus Programmer Window

APPENDIX D

Altera DE0 Education board

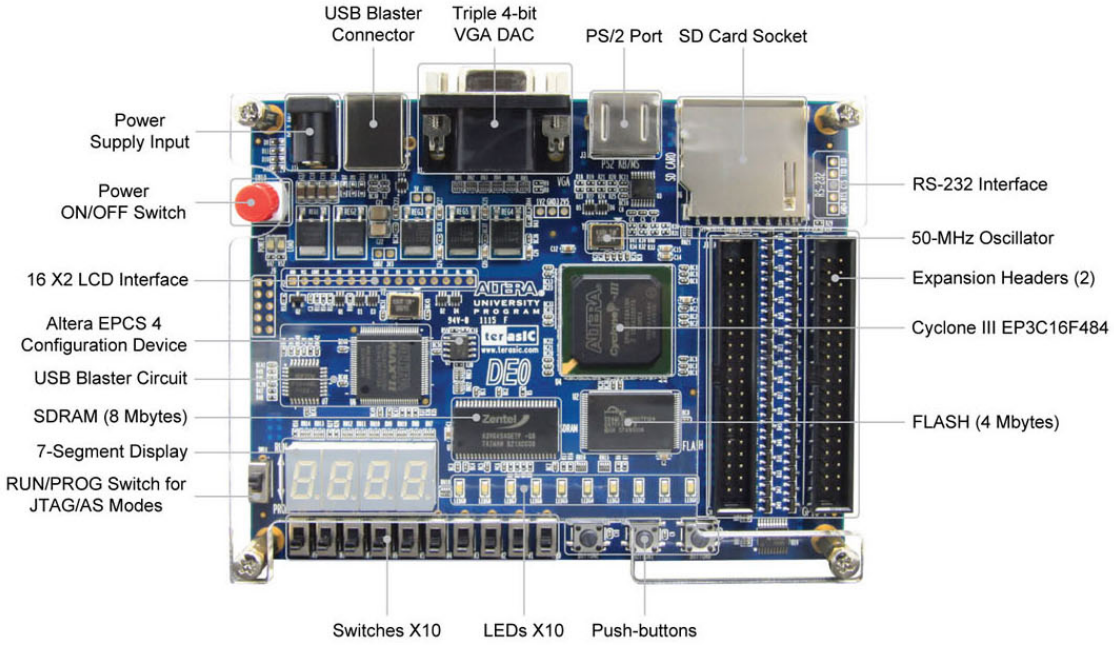


Figure D.1: Altera DE0 education board

The "heart" of the DE0 education board is the Cyclone FPGA EPC3C16F484. The FPGA contains 15408 logic elements, 56 M9K Embedded Memory Blocks, 504Kb Ram, 56 Embedded Multipliers, 4 PLLs and 346 user I/O pins. The DE0 board contains an oscillator which provides the FPGA with a 50 MHz clock signal. The clock signal is connected to pin G21. The DE0 board further contains, among other things, the following components: 10 slide switches, 3 push buttons, 4 7-segment displays and 10 LEDs. All these components are connected directly to the pins of the FPGA, see below.

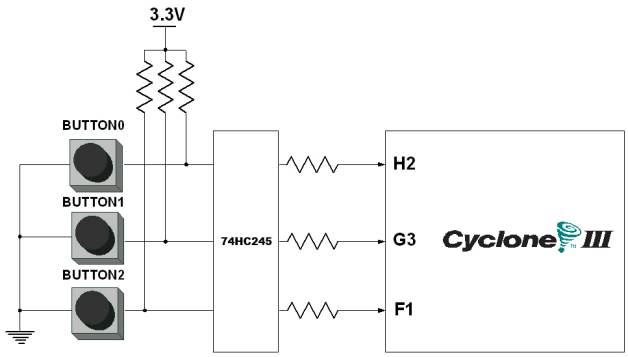


Figure D.2: Connections push buttons

The push buttons give a 0, when pressed, otherwise a 1, see Figure D.2. The slide switches provide a logical 1 when the slide is standing up, see Figure D.3. The LEDs of the 7-segment displays light up when receiving a 0 and are off when receiving a 1, see Figure D.4. Finally there are 10 LEDs connected to the FPGA, that light up when receiving a 1, see Figure D.5.

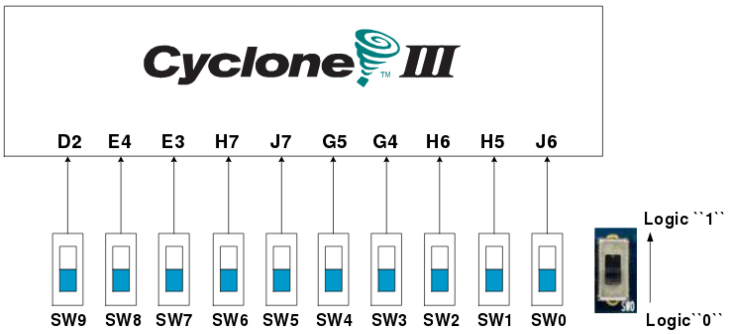


Figure D.3: Connections slide switches

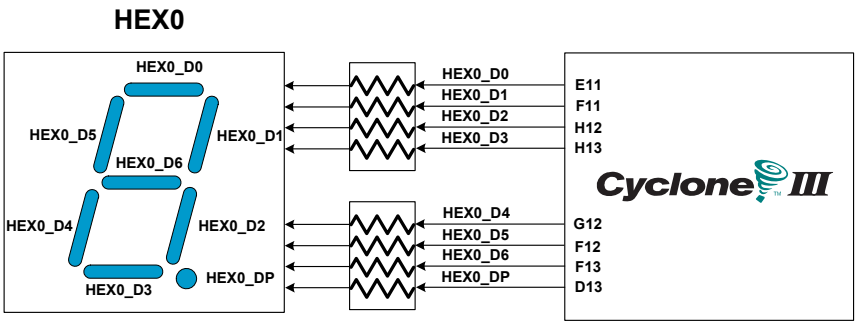


Figure D.4: Connections 7 segment display

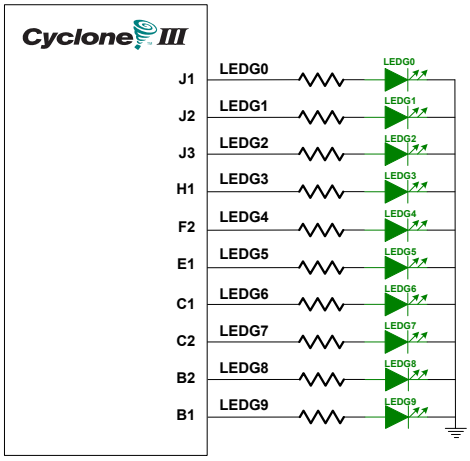


Figure D.5: Connections LEDs

APPENDIX E

Oscilloscope: Tektronix TDS 2022B

E.1 Introduction

An oscilloscope is a commonly-used electrical measuring instrument. It is used to display variations in electrical voltage as a function of time. Whereas in the past analog oscilloscopes were primarily suitable for qualitative measurements (amplitude and shape of a signal), modern digital oscilloscopes can also be used to carry out quantitative measurements (determining the value of a signal). The lab is equipped with the Tektronix TDS 2022B digital storage oscilloscope. This appendix will briefly discuss the operation of this device. For more information, please consult the use manual that is available in the lab room.

E.2 Overview

A diagram of the TDS 2022B is depicted in Figure E.1. The buttons on the devices are grouped, based on functionality.

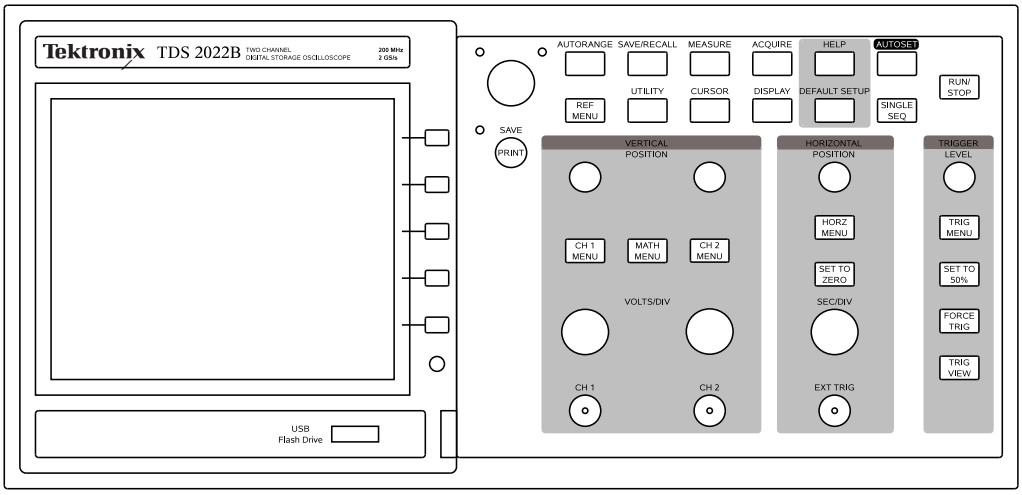


Figure E.1: The Tektronix TDS 2022B Digital Storage Oscilloscope.

E.3 Basic Operation

The basic operating elements of this oscilloscope are more or less the same as those of an analog oscilloscope. The most important part of an oscilloscope is, of course, the display. The waveforms of the input signals are visualized on this display. The screen is divided into squares, in this case: 8 vertical and 10 horizontal squares. These squares are called *divisions*, the abbreviation of which is *divs*. The time is shown on the horizontal axis and the amplitude of the signal on the vertical axis.

E.3.1 Vertical Position

The vertical position group of knobs and buttons are used to modify the vertical scale and position of the waveform. The two large knobs are used to change scale of their respective channels. This is expressed in VOLTS/DIV, the actual setting is displayed in the bottom left corner of the screen,

the first channel being preceded by the text CH1 and the second channel by CH2.

The two smaller knobs are used to move the 0 V reference level. This can be useful when displaying two signals simultaneously. The actual setting is shown at the bottom of the screen while turning the knobs and indicated with a small arrow. The channel number is visible on the left of the screen.

CH1 and CH2 menu

The CH1 and CH2 menus are used to set a number of functions for the two channels. A particular channel can be switched on or off by pressing the channel menu button for that channel twice.

The menu comprises the following options:

- Coupling (GND, DC, AC)
  This is used to read out the input signal as GND (handy for adjusting the oscilloscope), DC (this is used to measure a DC component in the signal along with the rest) or AC (this is used to filter out a DC component).
- BW Limit (OFF(200MHz), ON(20MHz))
  This is used to limit the maximum bandwidth of the oscilloscope.
- Volts/Div (Coarse, Fine)
  This is used to select fine or coarse tuning using the volts/div button.
- Probe (1×, 10×, 100×, 1000×)
  This is used to compensate for the state of attenuation of the probe.
- Invert (OFF, ON)
  This is used to invert the selected channel.

MATH menu

Calculations relating to the input signal can be carried out with the aid of the MATH menu. This menu can be used to determine the difference between and the sum of signals.

E.3.2 Horizontal Position

The large knob is used to change the horizontal scale (time axis). This is expressed in SEC/DIV, the actual value is shown at the bottom right. It is possible to set two different vertical scales, but only a single horizontal scale, so both channels use the same time base setting.

The small rotary knob for the horizontal position is used to horizontally move the 0 s reference level. The SET TO ZERO button can be used to quickly reset the horizontal position.

E.3.3 Trigger Level

The trigger level is used to determine when to start data acquisition and when to display the waveform. The trigger level is a voltage level indicated by a small arrow on the right of the screen. The waveform is shown starting on the intersection point of the trigger level and the 0 s horizontal position. The trigger level can be changed with the small rotary knob.

TRIG MENU

The trigger menu allows you to change trigger settings. The following options are available:

- Type (Edge, Video, Pulse)
  This is used to select the type of trigger event. Usually Edge is the correct setting.
- Source (CH1, CH2, Ext, Ext/5, AC Line)
  This is used to select the source signal. Normally you should only use CH1 or CH2.

- Slope (Rising, Falling)
  Select the type of edge used to trigger.
- Coupling (DC, Noise Reject, HF Reject, LF Reject, AC)
  Allows filtering of the input signal of the trigger circuitry.

SET TO 50%

This button sets the trigger level to the midpoint of the peaks in the trigger input signal.

FORCE TRIG

Force a trigger to start the data acquisition.

TRIG VIEW

While this button is pressed down, the display shows the trigger input signal. This can be useful when trigger coupling differs from the channel settings.

E.3.4 Halting Acquisition and Single Sequence

Data acquisition can be stopped by pressing the RUN/STOP button. This will also freeze the display. Note that the functions of the MEASURE menu, see E.4.1, still work.

In order to capture a single-shot signal, press the SINGLE SEQ button. After the input has triggered the oscilloscope, it will automatically halt and show the captured waveform on the display.

E.3.5 AUTOSET

This oscilloscope is equipped with functionality to automatically select the correct settings for the time base, voltage scale and the coupling on the basis of the input signals. This takes place by means of the AUTOSET button. Although this may be very useful, the auto set functionality does not work well if the signal to be measured has a very low frequency.

E.3.6 AUTORANGE

This oscilloscope can also automatically set both the horizontal and vertical scales as well as the vertical signal positions. This is done by pressing the AUTORANGE button. If the auto range function is active, the indicator light on the left of the button will be on. Pressing the button again will turn the auto range function off.

E.4 Menus

The various functions of the oscilloscope can be operated by means of menus. If you wish to call up a particular menu, simply press the button with the relevant text. You can then select the settings you want from the menu using the five buttons next to the screen.

E.4.1 MEASURE menu

As already mentioned, a digital oscilloscope can also be used to take measurements. On this digital oscilloscope, the MEASURE menu is used for this purpose. Various measurements can be taken at the same time and the results will be displayed on the menu.

The topmost menu selector button switches between `Source` and `Type`. The option selected will affect how the other menu buttons subsequently work. If `Source` has been selected, the channel to be measured is set using the other buttons. The `Type` button selects the type of measurement. The following options are available:

None no measurement;

Freq displays the frequency of the signal;

Period displays the time period of the signal;

Mean displays the average value of the signal;

Pk-Pk displays the peak-peak value of the signal;

Cyc RMS displays the RMS (root mean square) value of the signal;

Min displays the minimum value of the signal;

Max displays the maximum value of the signal;

Rise Time displays the time between 10% and 90% of the first rising edge;

Fall Time displays the time between 90% and 10% of the first falling edge;

Pos Width displays the time between the 50% level of the first rising edge the next falling edge;

Neg Width displays the time between the 50% level of the first falling edge and the next rising edge.

Note that at least one full period of the signal must be visible in order for these measurements to be accurate.

E.4.2 CURSOR menu

Besides the option in the Measurement menu, you can use the cursors from the CURSOR menu. Cursors can be horizontal (voltage) or vertical (time). Vertical cursors can track the active channel and show the voltage level at a specific time instance. They can also be used as a visual reference level, which can be useful to examine the behavior of a circuit when it is subjected to a sweep input.

The first button in the CURSOR menu can be used to select the type of cursor:

- If you select `Off`, no cursor will be shown;
- If you select `Voltage`, horizontal cursors will be shown;
- If you select `Time`, vertical cursors will be shown.

The second button enables you to change between different sources:

- `Ref A`
- `Ref B`
- `CH1`
- `CH2`
- `Math`

The third menu option shows the absolute value of the difference between the two cursors. This option is not selectable and cannot be changed.

The fourth and fifth menu options show the position of the cursors. The active cursor is highlighted and can be moved by means of the general purpose rotary knob on the top left of the buttons.

E.4.3 ACQUIRE menu

The acquire menu can be used to modify the data that is used to draw the signals on the screen. The following options are possible:

Sample this is the default, it shows directly the sampled data;

Peak Detect this can be useful when measuring a noisy signal, low amplitude noise is slightly dimmed so spikes are easier to see;

Average the displayed signal is the average of an adjustable number of samples. This can be useful to filter out noise

In the upper left corner of the screen, an icon indicates the type acquire option.

E.4.4 DISPLAY menu

The display menu can be used to modify the way the signals are visualized.

E.5 Advanced Options

This digital storage oscilloscope has some advanced options that can be useful.

E.5.1 Storing Screenshots and Data

The oscilloscope can store and recall data from a USB flash drive. Insert a flash drive into the USB port[1] and wait while the oscilloscope examines the drive.

Press the SAVE/RECALL button to setup the action you wish to perform. The following actions are the most useful:

Save Image this option will store a screenshot image in a selectable data format;

Save Waveform this option will store the acquired data points of the selected channel in a CSV spreadsheet file;

Save All this option will store a screenshot, the acquired data points of both channels, and the settings of both channels.

Saving the data will take some time, do not remove the flash drive while the oscilloscope is still writing data!

You can also couple an action to the PRINT button in order to quick access to a save action.

E.5.2 FFT

The oscilloscope also features a low frequency digital spectrum analyzer. This option can be found in the MATH menu by selecting FFT as type of Operation. You can specify the source signal and horizontal and vertical resolution with the VOLTS/DIV and SEC/DIV knobs. The horizontal position knob can be used to select the center frequency on the display.

E.6 Probes

Measurement on an oscilloscope are preferably done with a probe. Probes that attenuate the input signal contain a circuit that has to be tuned before the probe is used. Tuning can be done by connecting the probe to the Probe Comp connector. This output will provide a 5 V, 1 kHz square wave input. If the output on the display is distorted, the probe can be adjusted by turning the small screw in the probe (either near the probe tip, or near the BNC connector). Note that probes with an adjustable attenuation factor bypass this circuitry in the 1× position.

[1] The oscilloscope can only flash drives up to 64 GB capacity, and only when formatted with FAT32. The newer exFat (FAT64) format will not work.

APPENDIX F

Function generator: Tektronix AFG 3021B/C

F.1 Introduction

The lab is equipped with the Tektronix AFG 3021B/C arbitrary function generator. This appendix will briefly discuss the operation of this device. For more information, please consult the user manual that is available in the lab room.

F.2 Overview

A diagram of the AFG 3021B/C is depicted in Figure F.1. The buttons on the devices are grouped, based on functionality.

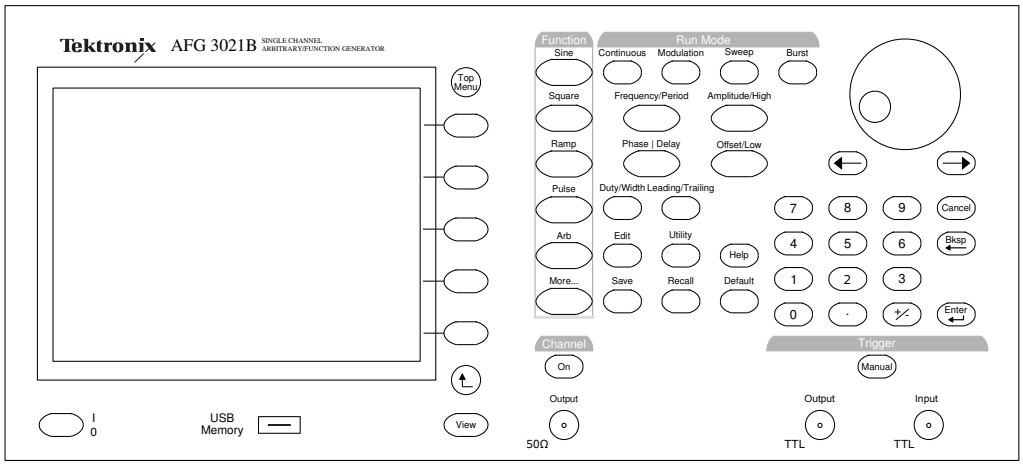


Figure F.1: The Tektronix AFG 3021B/C Arbitrary Function Generator.

F.2.1 Run Mode

On the top, there are four buttons grouped as Run Mode. These perform the following functions:

Continuous continuous operation, selected signal is continuously outputted;

Modulation perform AM, FM, PM, of FSK modulation;

Sweep sweep output over a range of frequencies;

Burst output one or more bursts of a signal.

F.2.2 Function

Directly left from the Run Mode group is the Function selection. These allow quick a choice between the following waveforms:

Sine a sine wave with adjustable frequency, amplitude, and phase;

Square a square wave with adjustable frequency, amplitude, and phase;

Ramp a ramp with adjustable frequency, amplitude, phase, and symmetry;

Pulse a pulse with adjustable frequency, amplitude, delay, and duty cycle;

Arb an arbitrary waveform, fully adjustable;

More… select sinc, Gaussian, Lorentz, Exponential Rise or Decay, or Haversine functions or white noise.

F.2.3 Signal Setup

Under the Run Mode buttons are buttons to set up the selected waveform. Each button has two (related) functions, repeatedly pressing the button will alternate between the functions.

Frequency/Period select the frequency or period of the signal;

Amplitude/High select the amplitude or the high voltage level;

Phase|Delay select the phase shift or the delay;

Offset/Low select the offset or the low voltage level.

When the Pulse waveform is selected, two additional buttons are available:

Duty/Width select the duty cycle or pulse width;

Leading/Trailing select the leading and trailing time.

Pressing one of these buttons highlights the corresponding setting on the screen. The value can now be changed by either typing in the numerical value on the keypad and pressing Enter to confirm, or by using the general purpose rotary knob. The arrow keys directly under the general purpose knob move the cursor and thus the digit changes when turning the general purpose knob.

The screen shows also shows a preview of the signal annotated with all the important voltage levels and frequencies. These levels are only valid when the output impedance is properly set up, as described in Section F.3.

F.2.4 Additional Signal Setup

The buttons directly next to the screen are context-sensitive and are used to modify additional signal parameters. Changing the parameters is identical as described in Section F.2.3.

F.3 Output and Output Impedance

The output of the generator is the leftmost coax connector labeled Output. The output impedance of this output is adjustable. Changing the output impedance will have an influence on the settings of the generator. The amplitude settings on the generator are only valid when the actual output impedance is identical to the set output impedance. By default, the output impedance is set to 50 Ω. The input impedance of, for example the oscilloscope, op-amp inputs and almost any digital IC input is very high. In order to have the actual voltage levels match the settings on the function generator, the output impedance has to be changed to high impedance. You can use the following steps to set the output impedance to high impedance.

- Top Menu (this is the upper button next to the screen),
- Output Menu,
- Load Impedance,
- High Z.

The lower button can be used to return to the main screen.

Note that the output settings are only valid for a fixed load impedance; if the load is highly variable, make sure to verify the actual output on an oscilloscope or use a buffer circuit.

The output signal is enabled or disabled by pushing the On button in the block Channel just above the output connector.