Zaporizhzhya National Technical University
Ministry of Education and Science of Ukraine

Software Tools Department

Methodical instructions for laboratory works on discipline «The CAD theory for complex objects and systems»

for student of specialty 122 „ Computer Sciences”,
all forms of study

2017

Methodical instructions for laboratory works on discipline of „The CAD theory for complex objects and systems” for student of specialty 122 «Computer Sciences” all forms of study / Prepared by: Karsten Henke, Anzhelika Parkhomenko, Olga Gladkova. – Zaporizhzhya: ZNTU, 2017. – 39 p.

Prepared by: K.Henke, Dr., TU Ilmenau, Germany
A.Parkhomenko, PhD, Associate Professor
O.Gladkova, Assistant

Reviewer: G.M. Shilo, PhD, Associate Professor

Responsible for issue: S.O. Subbotin, Doctor of Engineering, Professor.

Approved
at the “Software Tools”
department meeting

The protocol № 11
Date: 06.06.2017

Content

Introduction

Rapid Prototyping Board of laboratory GOLDI

View of the Rapid Prototyping (RP) board is shown in Fig. 1. Operation Modes:
1) Web-based rapid prototyping of digital systems:

- design (text based, PCB, automaton graph);
- program the CPLD via the remote lab infrasrtructure;
- test the design via RIA and Web-based RP board.

2) Web-based validation of digital systems:

- identify the function of a given design (black box);
- find malfunctions of a well-known design;
- by manipulating the inputs or upload the truth table or test vectors;
- CPLD not programmable by students.

Main Components of Rapid Prototyping Board are: MAX V – 5M1270Z (CPLD from Altera); Input buttons (8 slide switches, 8 push buttons, 2 rotary hex encoder); LED outputs (4 7-segment displays, 1 LED bar display); Other components (10 MHz crystal oscillator, Frequenzy synthesizer, Piezoelectric oscillator, Incremental encoder, UART (via USB), 25-pin SUB-D connector).

Interaction scheme RP board with the RL infrastructure is shown in Fig.2:

- communication with the remote lab infrastructure via an Interconnection FPGA;
- all inputs of the board are removed from the PCB;
- replaced by a direct connection to the outputs of the „Interconnection“ FPGA:
- sets inputs according user‘s input via the user interface;
- reads outputs directly;
- „Interconnection“ FPGA is connected to the BPU within the remote lab infrastructure.

Web-based User Interface realization as HTML5 RIA (Rich Internet Application) is shown in Fig. 3.

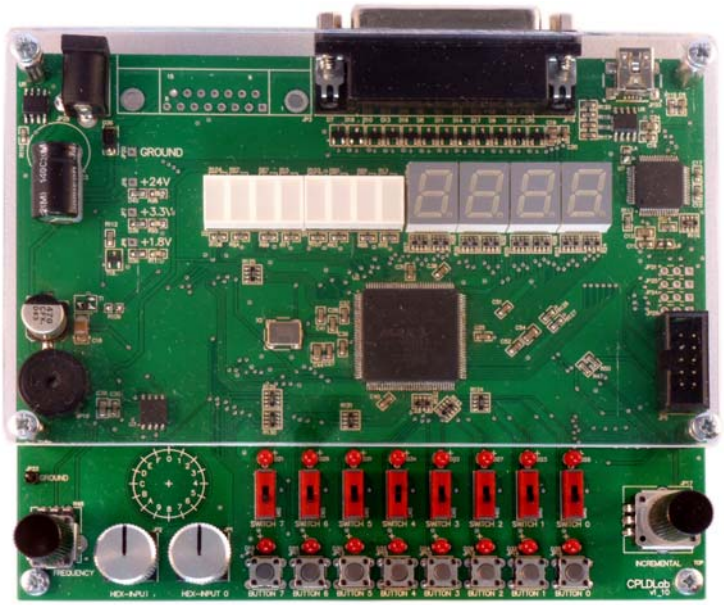


Figure 1 Rapid Prototyping Board

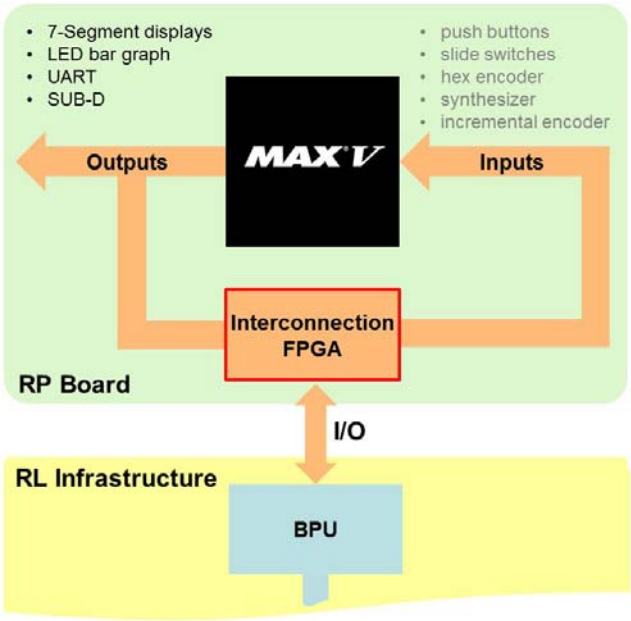


Figure 2 Interaction scheme of RP board with the laboratory infrastructure

Figure 3 Web-based User Interface – RIA

Main interface components are:

- upload the synthesized CPLD code of the design to program the CPLD automatically in the remote lab;
- visual model of the RP Board, to manipulate all the inputs virtually and watch the outputs of the RP board;
- webcam observation in the remote lab to watch the results of the user's action directly.

Design Flow for Digital Systems is shown in Fig. 4. The diagram shows the basic blocks:

- common design tools: implement the control tasks e.g. with Altera's IDE; students can use „self-made" IP core libraries;
- upload the synthesized design to the RP board;
- programming the stand-alone RP board;
- web-based programming of the RP board via the remote lab infrastructure.

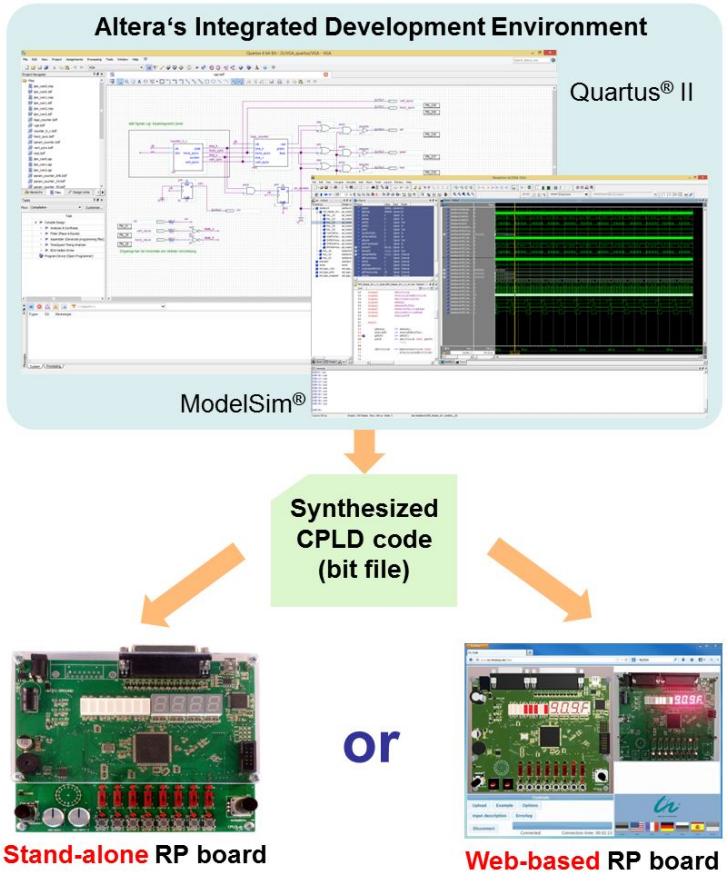


Figure 4 Design Flow for Digital Systems

Text based design methods enter the design by means of logical equations, truth tables or hardware description languages (AHDL, VHDL or Verilog) (Fig. 5).

Graphically based design methods enter the design by using block diagrams and/or schematic diagrams (Fig. 6).

Integrated FSM editor enter the design by using automata graphs (FSM) (Fig. 7).

Simulation of the design by using different simulation tools (e.g. Qsim or Model*Sim*) (Fig. 8) [16].

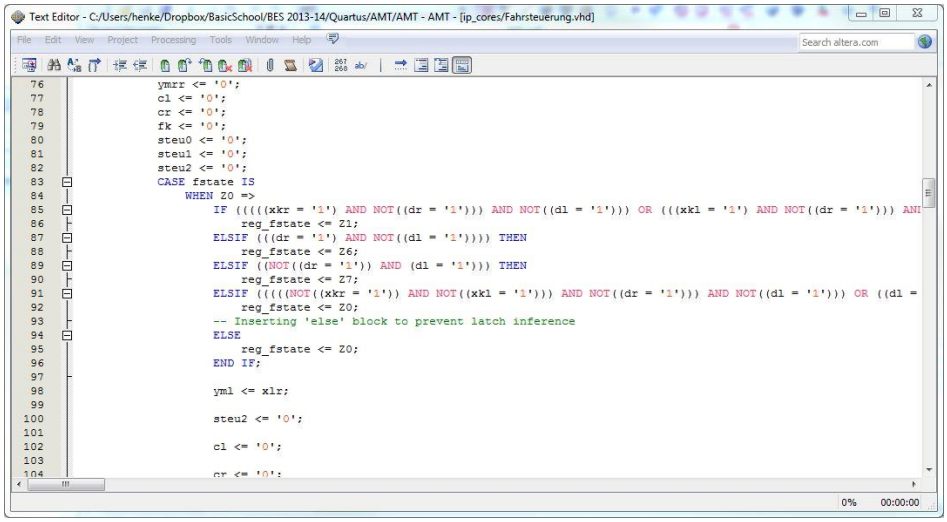


Figure 5 Text based design method

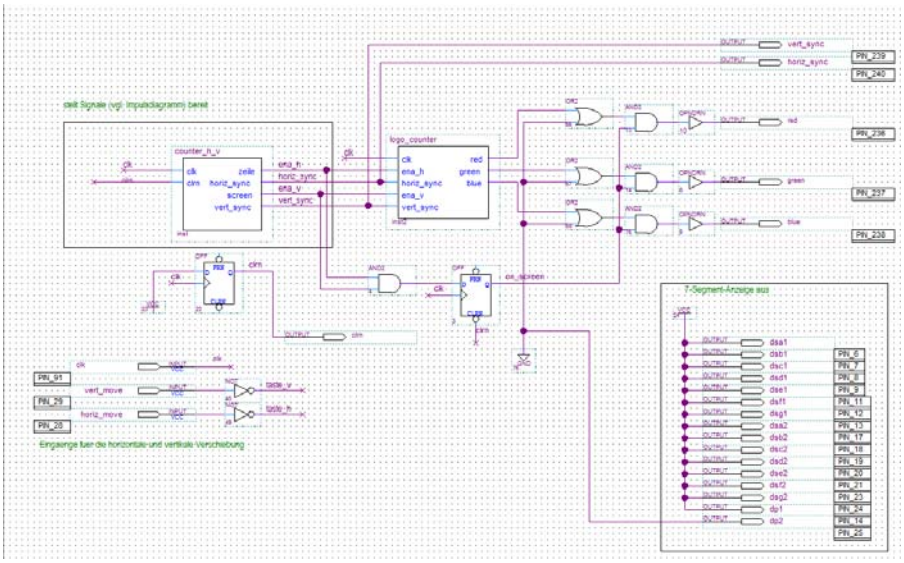


Figure 3.6 Graphically based design method

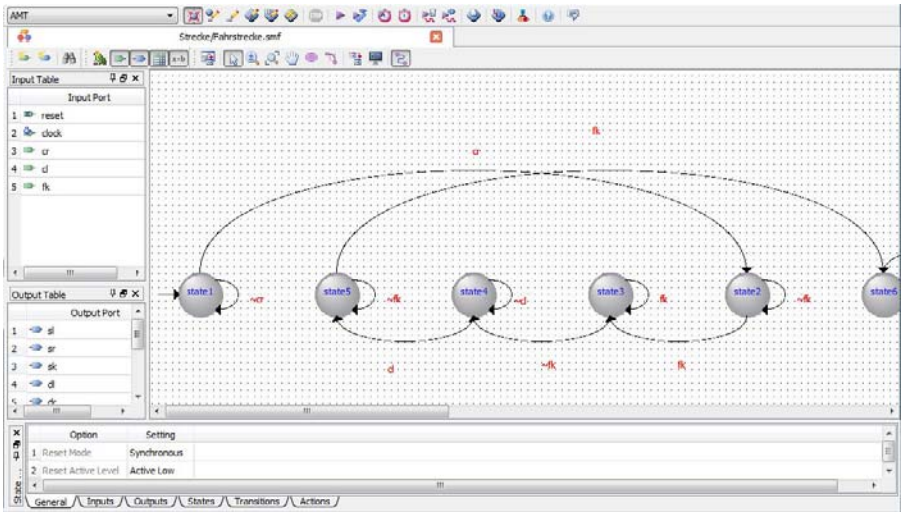


Figure 7 FSM based design method

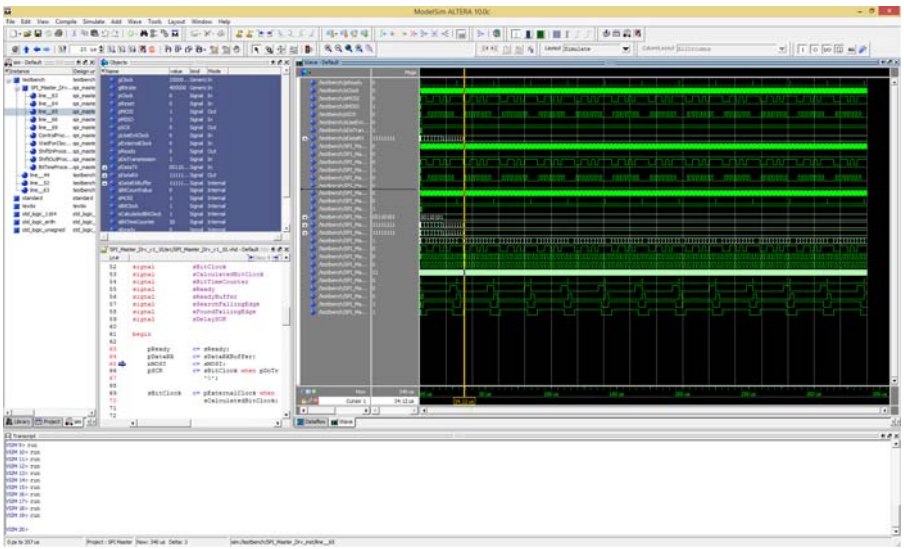


Figure 8 Simulation of the design

Lab 1 - Introduction to Quartus II

1 Introduction

This is a small introduction in some features of Quartus II. We will create a 4 bit full adder step by step as block diagram and an automaton as state machine. Later we will simulate them with ModelSim-Altera.

With a prepared template we will program a complex programmable logic device (CPLD).

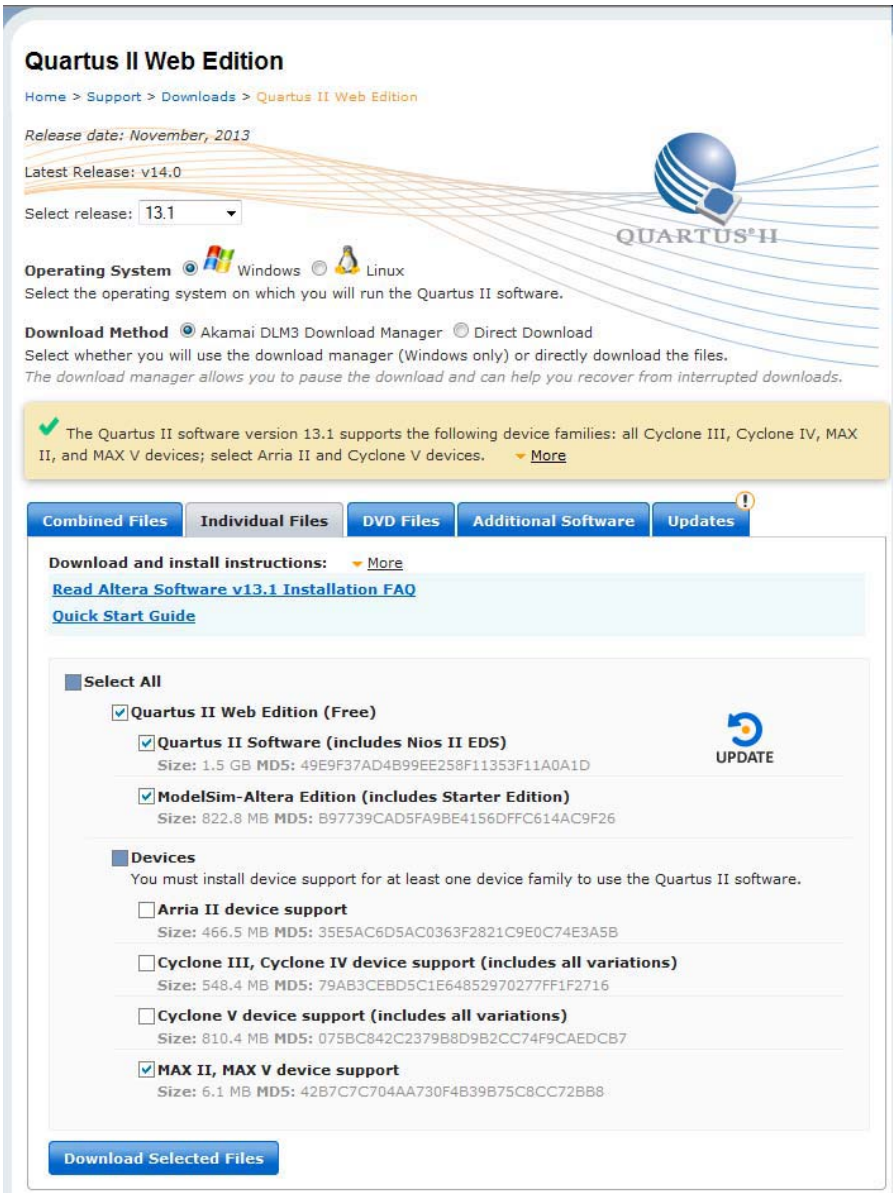
2 Preparation

2.1 Download

You can download Quartus II and ModelSim-Altera here: http://dl.altera.com/?edition=web

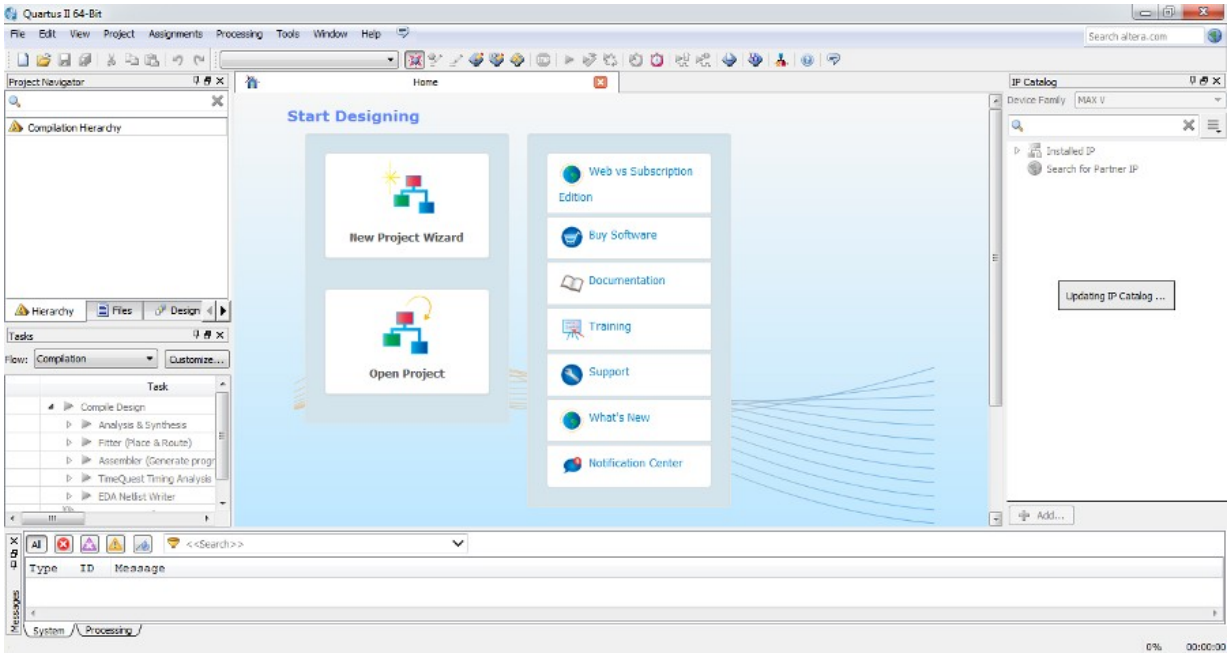
We will use the version 13.1. You can download the *Combined Files* or the *Individual Files*.

With the *Individual Files* you only have to download in *Device* the *MAX II, MAX V device support*, store all in the same directory.



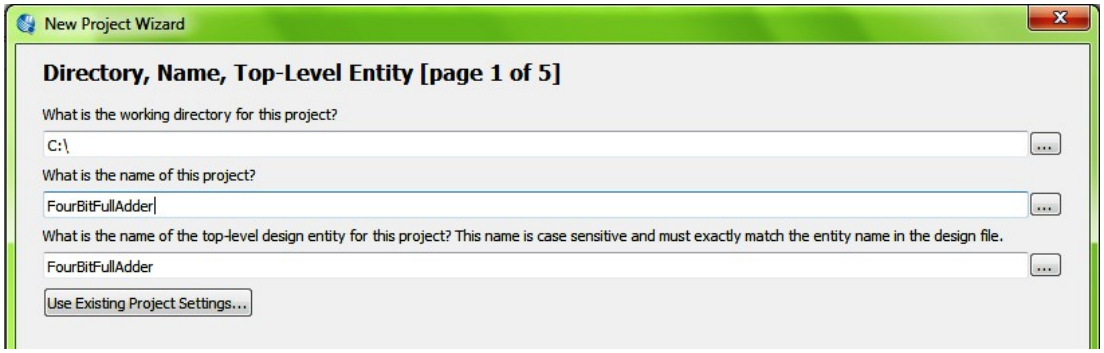
2.2 Create a new project

After starting Quartus II you see the following:

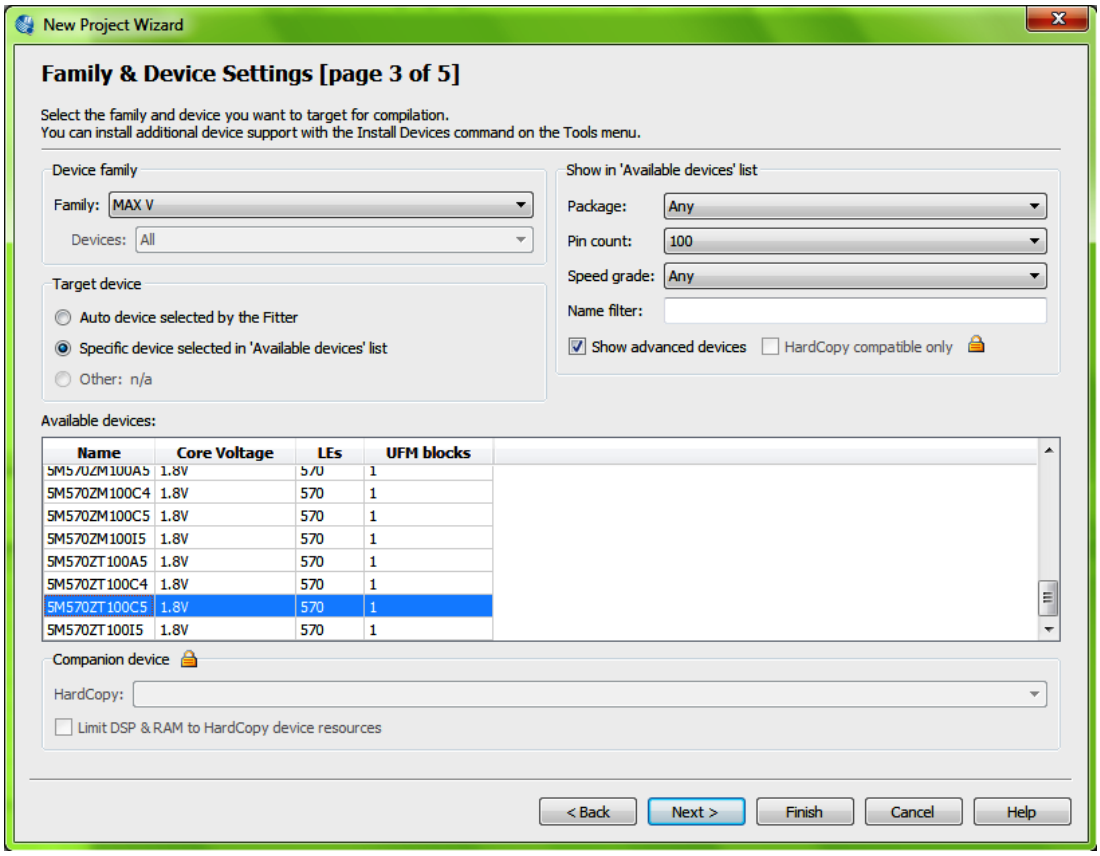


Select ”Create a New Project” and ”Next”. Now you are asked to name your project. Here it is important that the name of the ”top-level design entity” is the same. This name is for the final document, do not name one of the other documents like this.

Here we will name the project ”FourBitFullAdder”. Click twice on ”Next”.

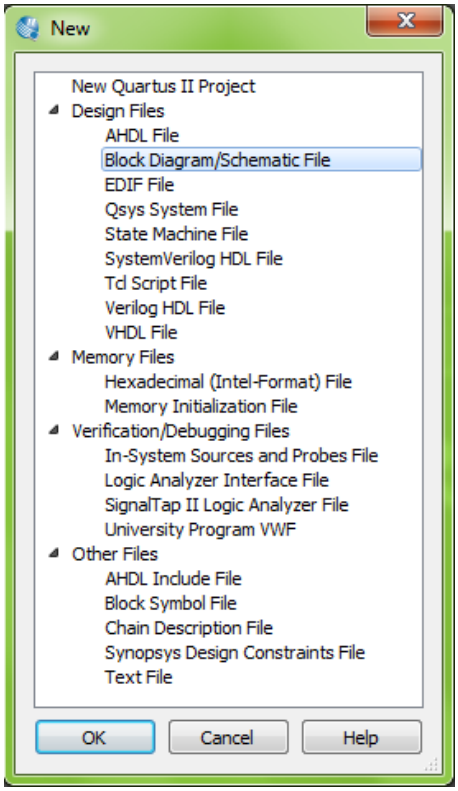


On the following page you select the CPLD you want to use. We need here the MAX V 5M570ZT100C5. But this selection can later be changed. Afterwards you select twice ”Next” and ”Finish”.



3 Design Files

For creating new design files select ”File → New”. In the list you chose the design files.



In the following we will create the 4 bit full adder. So select the Block Diagram File.

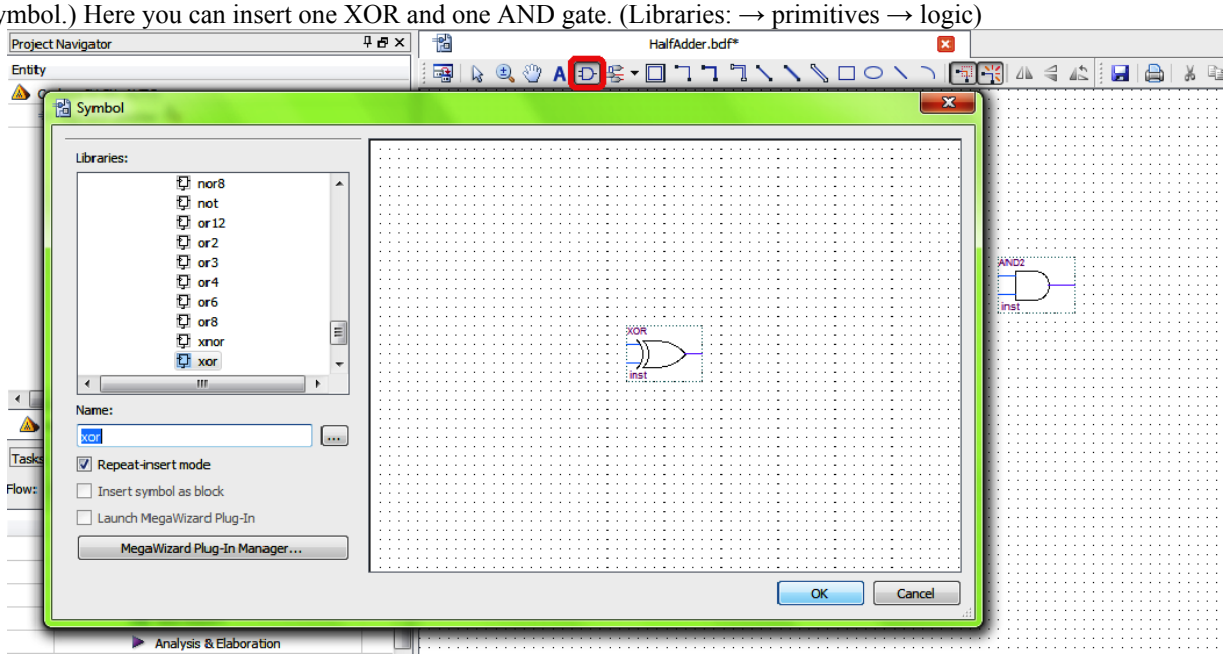
3.1 Block Diagram

At first save your file. If it is not the top-level design entity, name it not like your project.

Because we want to create a 4 bit full adder, we will compose it of 1 bit full adders and these of half adders. So save your file under the name ”HalfAdder”.

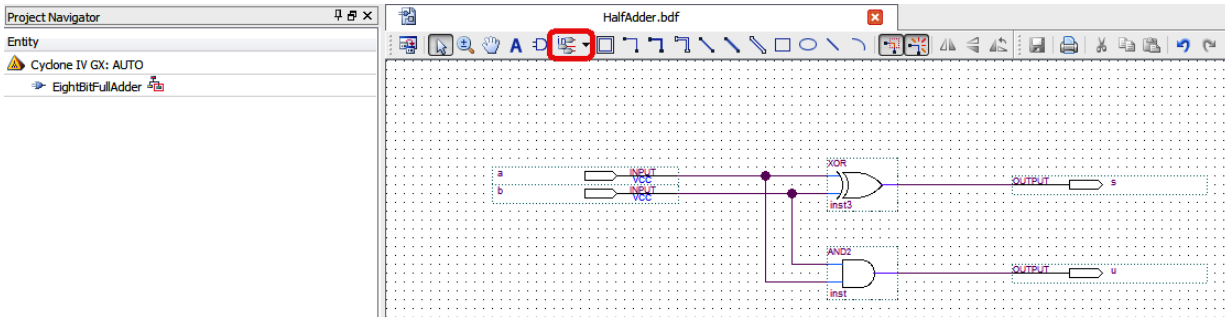
3.1.1 Symbols

On the tool-bar you find the icon for the insertion of symbols, the Symbol Tool. (On the right side of the A - symbol.) Here you can insert one XOR and one AND gate. (Libraries: → primitives → logic)



3.1.2 Pins

Next to the Symbol Tool is the icon for the Pin Tool. Insert two Input-Pins and two Output-Pins. Connect the elements like shown here.



Save this file again.

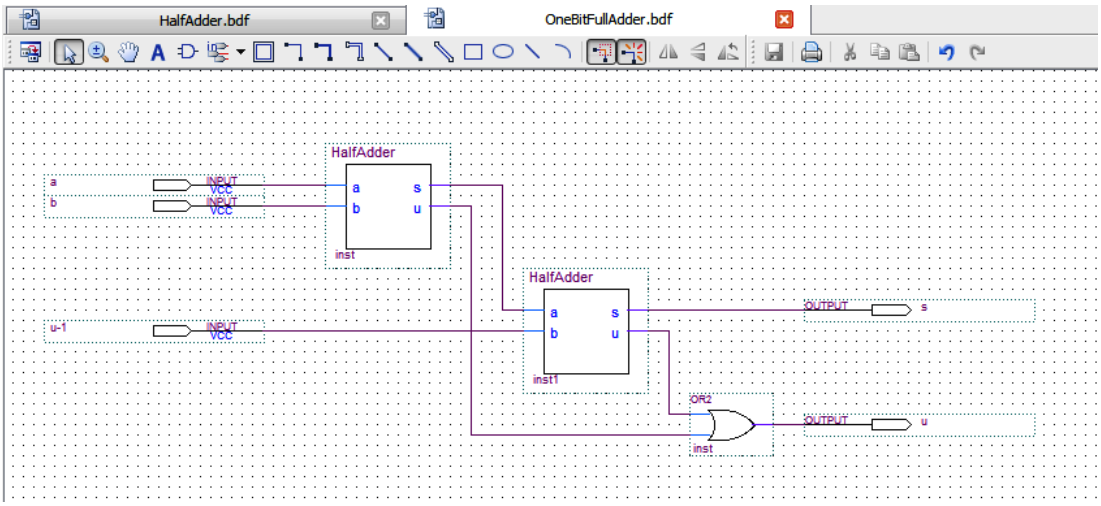
3.1.3 Symbol File

In order to use the half adder for building a 1 bit full adder, we will create a Symbol File for the half adder. With this file we can later insert the full logic of the half adder in an other Block Diagram only with one symbol. To do this, select ”File → Create/Update → Create Symbol Files for Current File”.

3.1.4 4 bit full adder

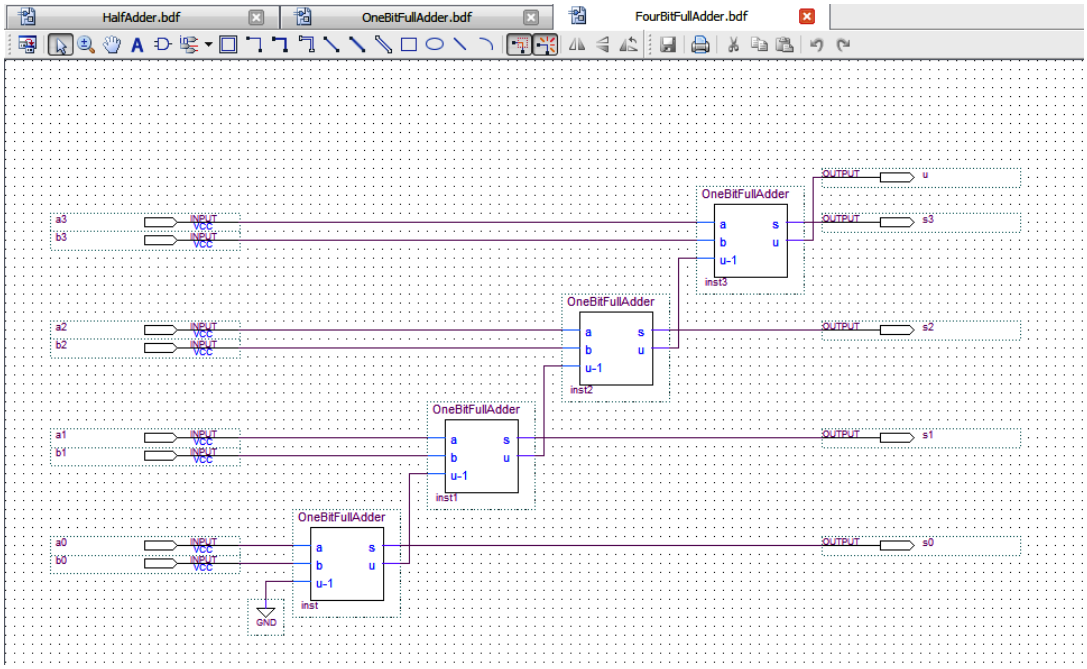
At first we create a 1 bit full adder. Generate a new Block Diagram file with the name ”OneBitFullAdder”.

With the Symbol Tool under ”Libraries: Project → HalfAdder” you can insert the half adder. Now you complete the one bit full adder.



Create again a Symbol File for the 1 bit full adder.

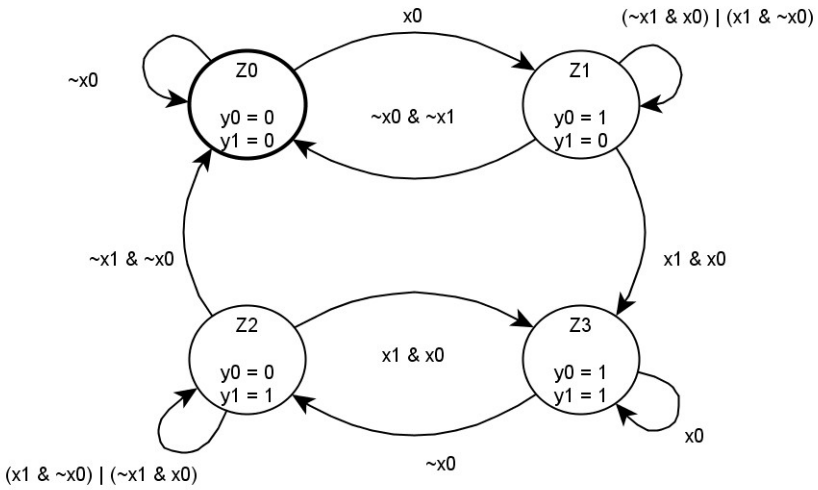
Now complete the 4 bit full adder with four 1 bit full adders in one more Block Diagram file. This is the top-level design entity, so name it like the project ”FourBitFullAdder” and save it.



3.2 State Machine Files

With Quartus II you can although design your project with state machines. Later this files can be used to generate HDL-Files.

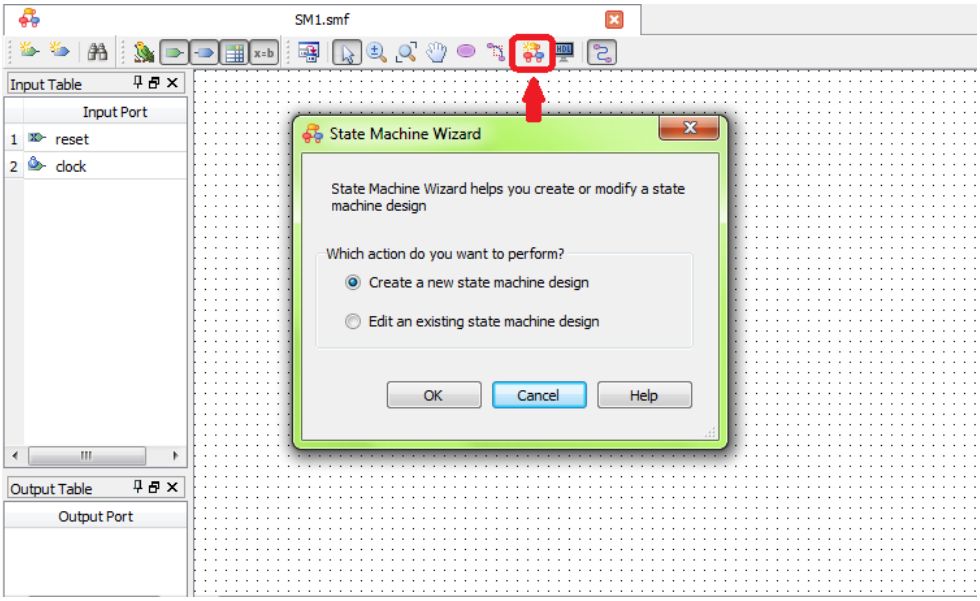
For the state machine select ”State Machine File” in the design file list. As an example we will create this finite state machine:



3.2.1 State Machine Wizard

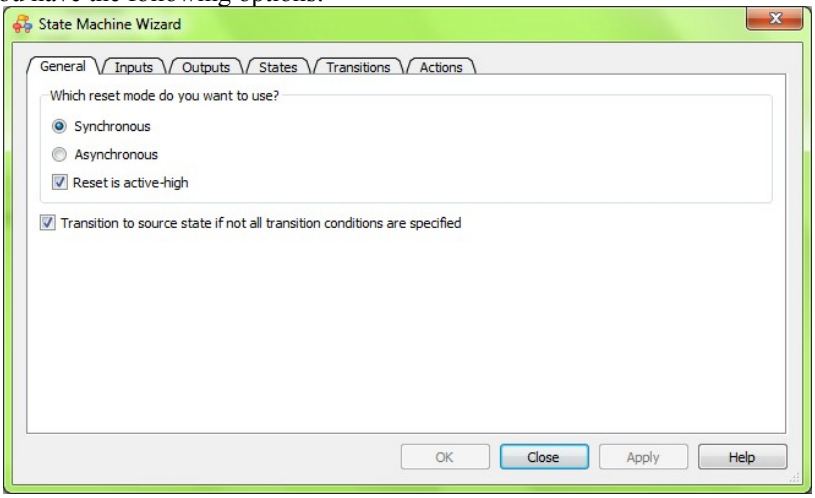
You can assemble your automaton with the tools or use the State Machine Wizard. For a large and complex automaton it is easier to use the State Machine Wizard.

In the tool-bar you find the State Machine Wizard:



Select the action you want to do. Here of course ”Create a new state machine file”. Later you can edit the finite state automaton.

At first you have the following options:



Then under ”Inputs” and ”Outputs” we insert all needed inputs and outputs. Now name all states and one of them with ”Reset = Yes”, at a reset the automaton returns in this state.

General | Inputs | Outputs

Input Port	Controlled Signal
clock	Clock
reset	Reset
x0	No
x1	No
< New >	

General | Inputs | Outputs | States | Transitions | Actions

Output Port	Registered	Output State
y0	No	Current clock cycle
y1	No	Current clock cycle
< New >		

General | Inputs | Outputs | States

State	Reset
Z0	Yes
Z1	No
Z2	No
Z3	No
< New >	

After that define the transitions. Write the bit operations like that:

&AND | ODER ~ NOT

General | Inputs | Outputs | States | Transitions | Actions

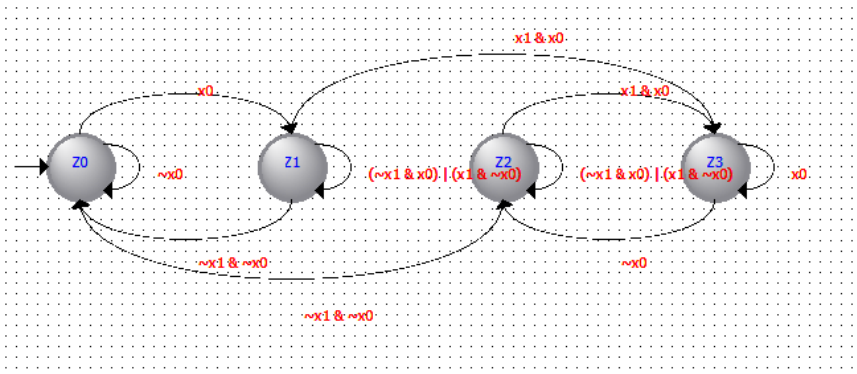
Source State	Destination State	Transition (In Verilog or VHDL 'OTHERS')
Z0	Z0	~x0
Z0	Z1	x0
Z1	Z0	~x1 & ~x0
Z1	Z1	(~x1 & x0) | (x1 & ~x0)
Z1	Z3	x1 & x0
Z2	Z0	~x1 & ~x0
Z2	Z2	(~x1 & x0) | (x1 & ~x0)
Z2	Z3	x1 & x0
Z3	Z2	~x0
Z3	Z3	x0
< New >		

General | Inputs | Outputs | States | Transitions | Actions

Output Port	Output Value	In State	Additional Conditions
y0	0	Z0	
y0	0	Z2	
y0	1	Z1	
y0	1	Z3	
y1	0	Z0	
y1	0	Z1	
y1	1	Z2	
y1	1	Z3	
< New >			

At last we define in ”Actions” the value of each output variable in each state. This definition must be complete, otherwise the value of one output variable in one state is undefined.

Now when you select ”OK”, you can see the ready finite state automaton.

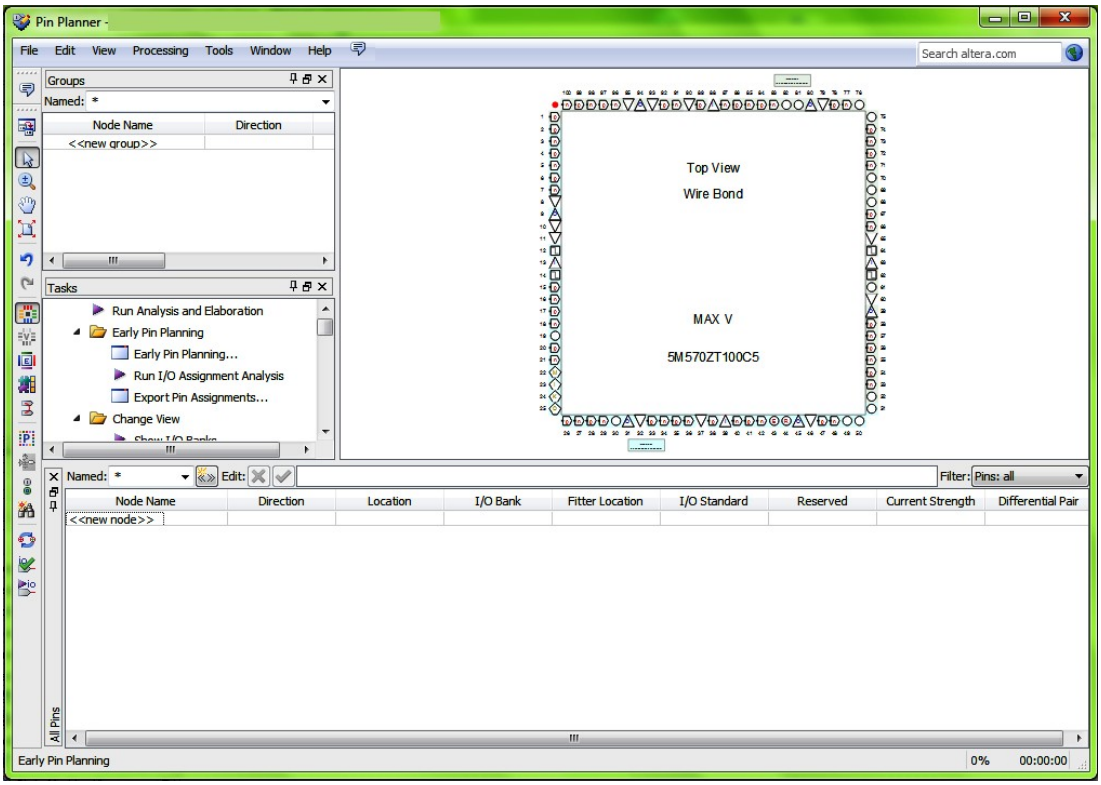


3.2.2 HDL File Generation

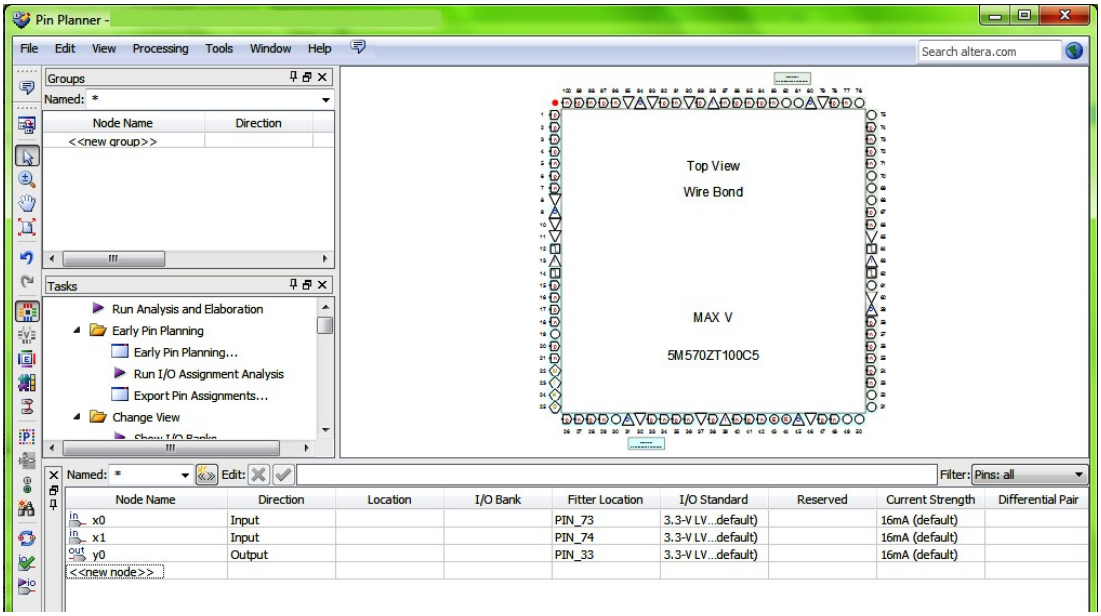
With the symbol on the right side of the State Machine Wizard Tool, you can generate HDL files out of your finite state automaton. Select here ”VHDL”.

4 Pin Planner

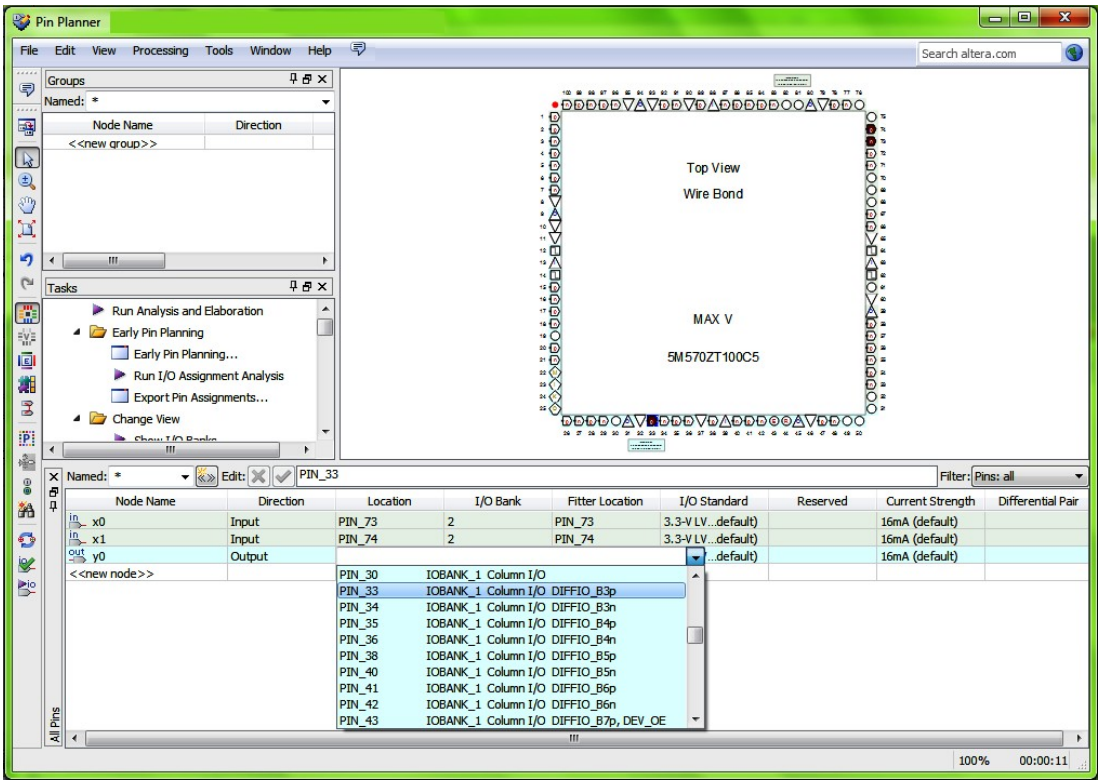
For the Pinout of you design you use the Pin Planner: Assignments → Pin Planner



If you have never done the compilation before, the Pin Planner is empty. Then you have to define all In- and Outputs from your top-level entity yourself (Node Name, Direction). With compilation this is already done:

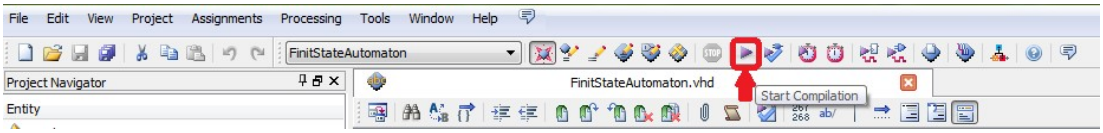


Now you have only to select the Pins (Location):



5 Compilation

To compile your project select the top-level design entity and start the compilation.



If there are some errors, look them up in the Compilation Report. The errors and warnings are listed in the red folders under Messages.

6 Simulation

We will use ModelSim-Altera. If you have installed ModelSim-Altera not with Quartus II, you have to set the path to the executable from ModelSim-Altera:

Tools → Options... and here: General → EDA Tool Options at ModelSim-Altera:

C : \altera\14.1\modelsim ase\win32aloem

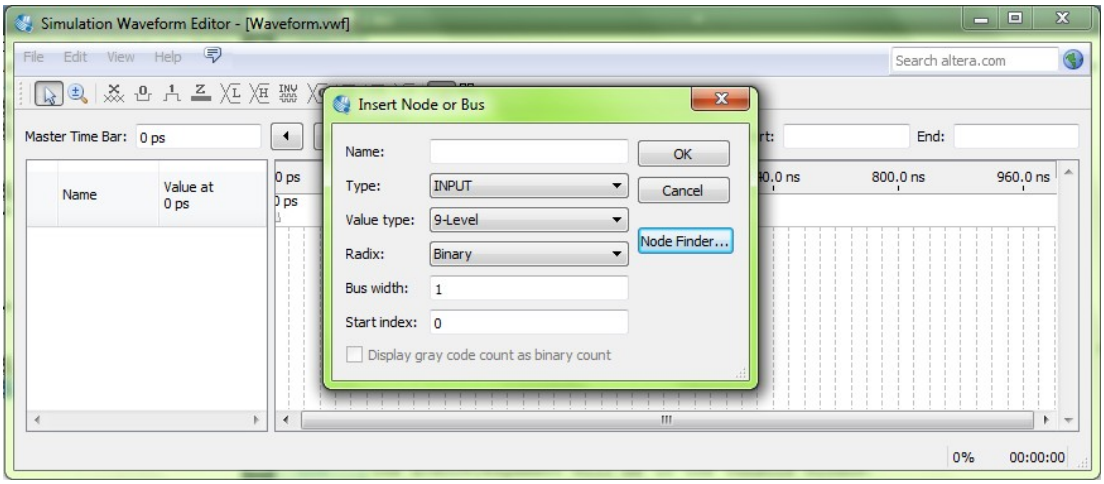
You can only simulate the top-level entity. When you want to simulate an other file, you have to set this file as top-level entity. (In the Project Navigator or ”Project → Set as Top-Level Entity”) After changing the top-level entity you have to compile your project again.

So you can simulate the ”HalfAdder”, the ”OneBitFullAdder” and the state machines too.

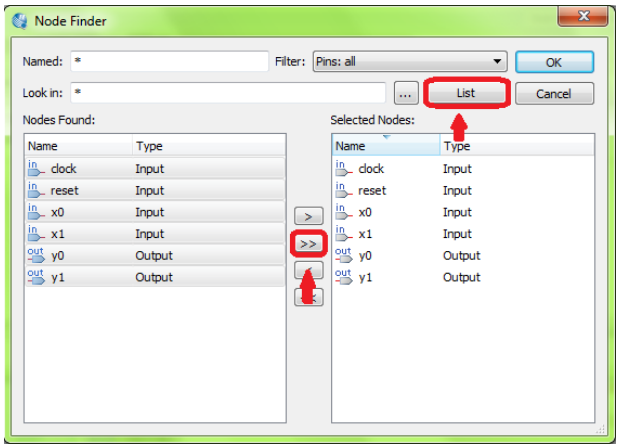
6.1 Simulation Waveform Editor

Now we will create the input file for the simulation. With this file we define the behavior of the inputs over the time. Select ”File → New”. Now we need here the ”University Program VWF” in Verifica-tion/Debugging Files.

In the new window select ”Edit → Insert → Insert Node or Bus...”.



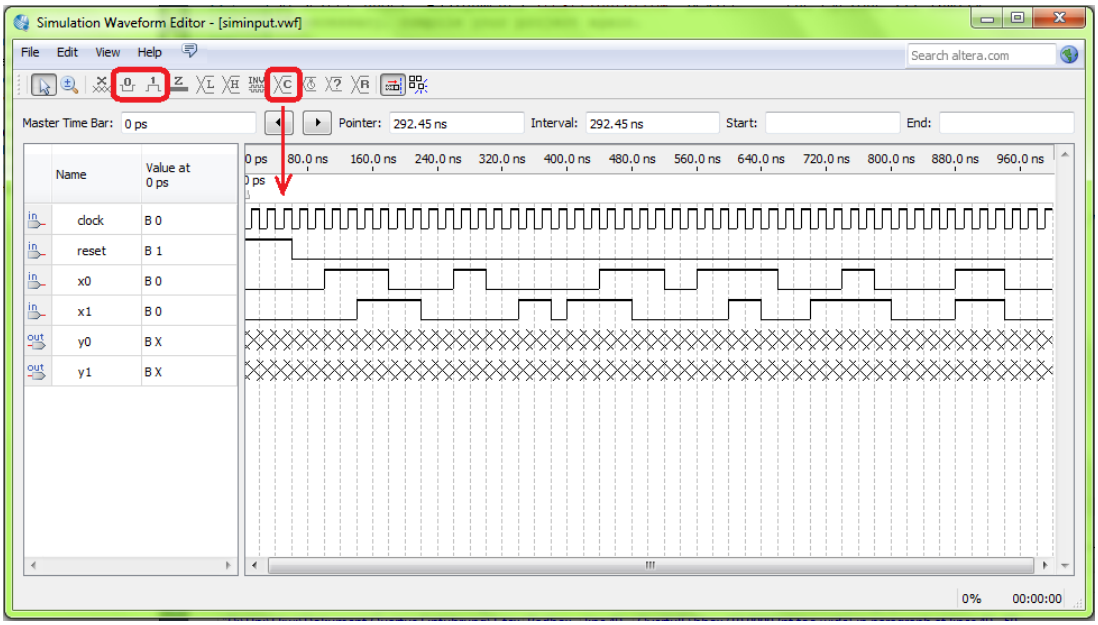
Here click ”Node Finder...”. Now ”List” all Nodes and select them all:



Click twice ”OK”.

In this file you can now define the behavior of the inputs. For the clock-signal it is easier to use the ”Count Value” in the toolbox. Set the values from the other inputs like the behavior you want to test. (You can not change the output values.)

For example like this:

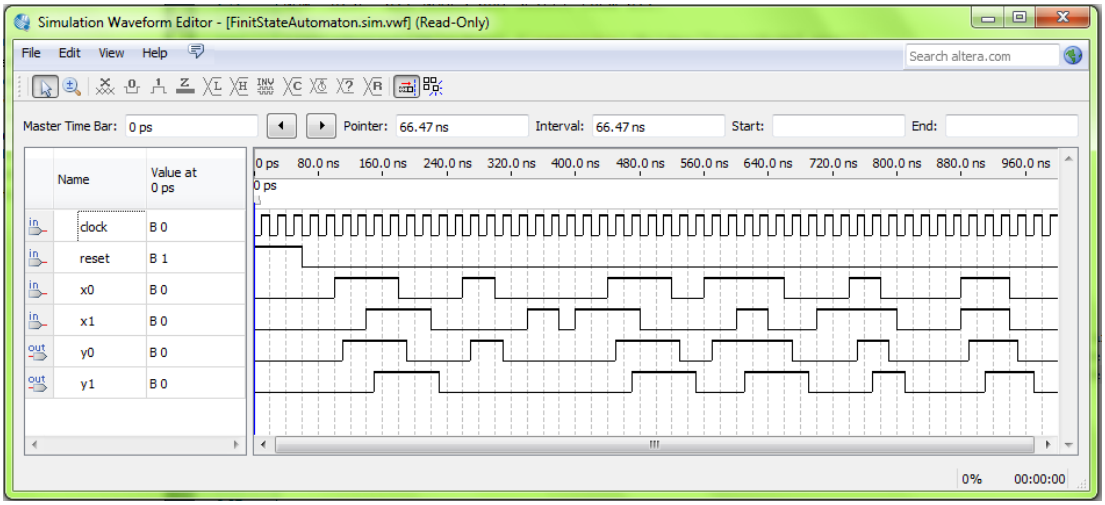


Save the file.

6.2 Start Simulation

You can start the simulation with ”Simulation → Run Functional Simulation”

You get a read-only file with the result of the simulation. It is like the input file, but with the simulated output. You can check here that everything works right.



When you want to simulate this file later again, open the vwf-file and start the simulation again.

7 CLPD Programming

7.1 Template

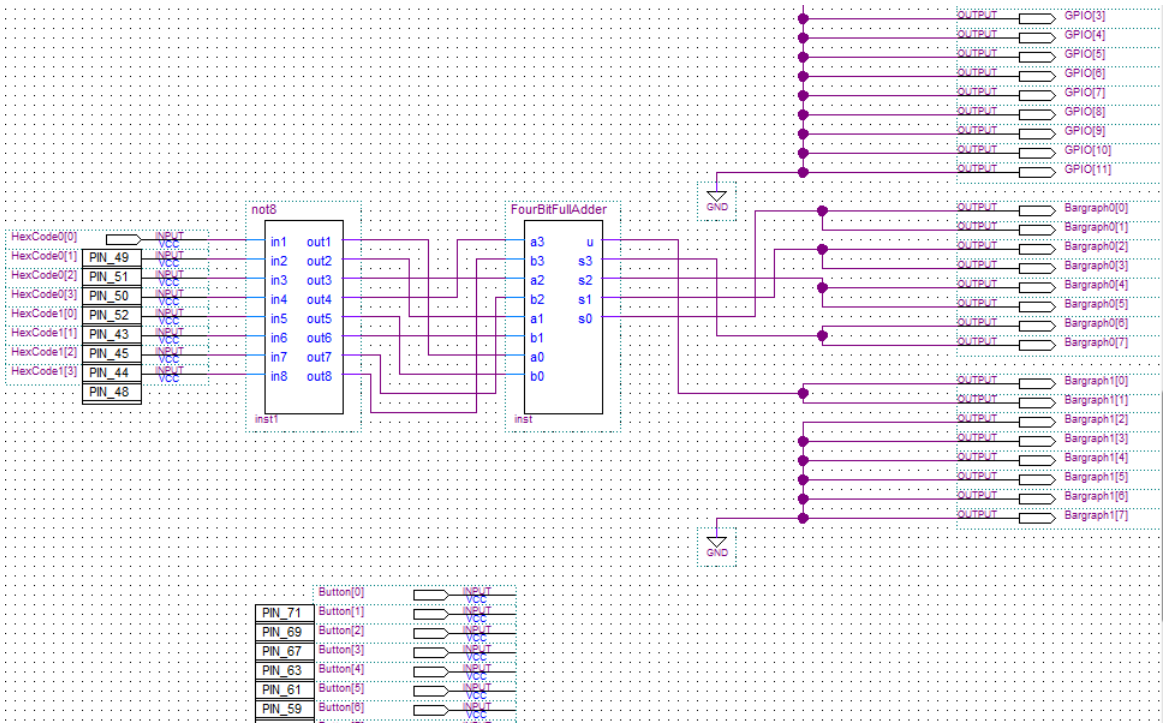
For our CLPD board is a template with all In- and Outputs given. You have only to insert the logic and connect it with the In- and Outputs.

Open the template and add the files you want to use. Therefor select ”Project → Add/Remove Files in Project...”. Open and add all files you want to use, also the lower layers. (For the FourBitFullAdder the OneBitFullAdder and the HalfAdder too.)

If this was successful, the files are listed in the Project Navigator (on the left side) under ”Files”. Now, if not done before, create the Symbol-Files. With the symbols assemble the logic in the template.

7.2 Example: 4 bit full adder

Here the 4 bit full adder is used to add the two hex rotary switches. Because the hex rotary switches are low active you have to negate the signals.



For the output is used the bargraph display.

7.3 Programmer

To program the CLPD we will use the IUT RemoteLab CPLD Programmer. Guarantee that the board is connected with the computer. After starting the Programmer open the POF-File of your project and select ”Program”. This will take a few seconds.

8 Questions

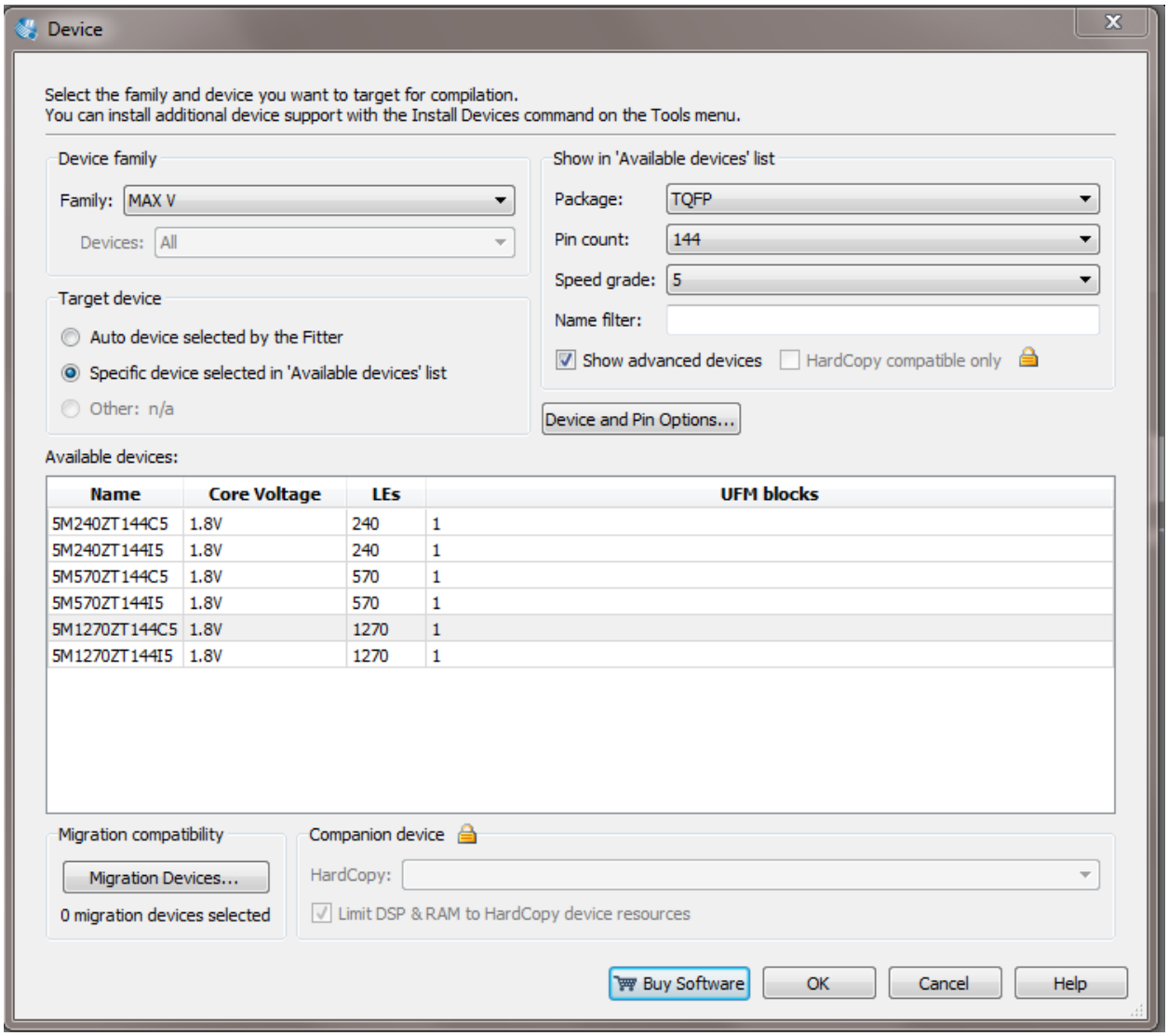
1. What are the main components of rapid prototyping board?
2. Which types of the rapid prototyping board exist?
3. Which ways we can create design?
4. What is Adder?
5. Which type of Adder exist?
6. Where Adder is used?
7. How to make pinout in Quartus and why to do it?

Lab 2
Design of a 7-segment decoder

- In this work we will design 7-segment decoder and program CPLD board.

1. First steps with the system Quartus II

The first steps are to be familiar with the design system Quartus II.
This introduction is for another CPLD - but it doesn't matter for this step.
For all other designs, please use the CPLD type 5M1270ZT144C5 - selectable via “assignment-> device” in Quartus.



I've preconfigured nearly everything in your individual design directories - you can use it.

2. The first project: template

In the folder “template” I prepared a complete design, that you can

- open via the project file CPLDLab_Template_v1_00.qpf,
- compile,

- and also program our CPLD board.

It contains all pinout information, as well as the used CPLD type. It is the basis for all other designs in other directories.

This design only serves to turn off all display elements - to save power.

You can “play” with this design - but please create a separate folder to do this. In this “template” directory please do no modifications.

In the following are the steps for the first really design task: 7-segment decoder …

3. Design of a 7-segment decoder

For almost all the designs we want to display something via 7-segment displays (http://en.wikipedia.org/wiki/7_segment or http://de.wikipedia.org/wiki/Segmentanzeige). To do this your first design task is the design of a 7-segment decoder.

Please note: our segments are low active, i.e. they are switched on by logically 0! Hereby we want to use different specification techniques, e.g.

- the design of a schematic by using different gates (e.g. NOT, AND, OR, ...)
- the design in AHDL description language (see some materials about AHDL in Theory directory),
- the design in VHDL (see some materials about VHDL in Theory directory).

I have prepared a folder "hex2seg" to do this. Via the project file hex2seg_test.qpf you can start your work.

The single decoders please interconnect with the hex encoder switches (I inserted already NOT gates, because they are also low active) as input, and one of the four 7-segment displays as output.

a) Design of the 7-segment decoder as PCB schematic

- Please create a file hex2seg_sch.bdf (as indicated in the introduction).
- Insert the design of the 7-segment decoder by using gates as schematic.
- The inputs please label with x_3, x_2, x_1 and x_0 and the outputs a, b,..., c for the 7 segments.
- Compile the design.
- You can test the design by using the simulation file hex2seg_sch.vwf (analogue of the introduction).
- After you have created a symbol, you can include this into the project file hex2seg_test.bdf and connect to one of the four 7-segment displays and compile.

b) Design of the 7-segment decoder in AHDL-notation

- AHDL: Altera hardware description language
- This description language provides a simple syntax.
- You should enter the design by using equations (with the prepared template file hex2seg_ahdl_equ.tdf) and by using truth table (with the prepared template file hex2seg_ahdl_tab.tdf) for this.
- Please compile the designs and simulate it (by using the same simulation files) and generate a symbol.
- After you have created a symbol, you can include this into the project file hex2seg_test.bdf and connect it to one of the four 7-segment displays, and compile as described above.

c) Design of VHDL-notation, a 7-segment decoder

- VHDL: Very high speed integrated circuit hardware description language (sounds pretty exciting - but it is the world's most used language for hardware designs).
- You should develop for this design also a notation with equations (with the prepared template file hex2seg_vhdl_equ.tdf) and with a kind of truth table (with the prepared template file hex2seg_tvhdl_tab.tdf).
- Please compile the design and simulate it (by using the same simulation files) and generate a symbol.
- After you have created a symbol, you can include this into the project file hex2seg_test.bdf and connect to one of the four 7-segment displays, and compile as described above.

When you finally have connect all 7-segment-displays to the decoders (which are connected to the two hex coding switches on the input side), you can compile the overall design and test it on the stand- alone board or in our GOLDi remote lab in Ilmenau.

Please "zip" now to the results of your first task (the hex2seg folder) and send it back to me - to check it ;-).

In the same manner we want to do it in all other designs.

A note to the folder ip_cores

A widely usable, pre-built function block is known as IP-core (English: intellectual property core).

This includes "intellectual properties" of the developer or manufacturer and is typically licensed or bought, to integrate it into the own design.

We want to use this folder to save often used designs inside (finally to save time), such as

- neg: a device, which generate a signal normal (high-active) and negated (low-active)
- not4: 4 NOT gates as a block
- mux2x1, mux4x1: 2-to-1 or 4-to-1 multiplexer
- prescaler: a frequency divider, which divided the crystal clock (10 MHz) to the desired frequency.

These modules could be used for designs whenever it is necessary.

4. Questions:

1. How much segment has 7-segment display in RP board?
2. To draw the position and index of each segment in a 7-segment display.
3. To write ASCII Table from 0 to F (using 10 and 16 number systems)
4. What is truth table?
5. What kinds of logic gates you know?
6. How the “not4” block works in the design?
7. How the decoder works?
8. What you Know about AHDL and VHDL? What is the difference?

Lab 3 - Modular Combinational Logic

Part: Binary Arithmetic Elements

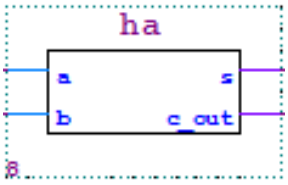
- Read the file "Modular Combinational Logic" on the Theory directory
- Complete the truth tables, Karnaugh maps and derive all the equations from this.

Preliminary

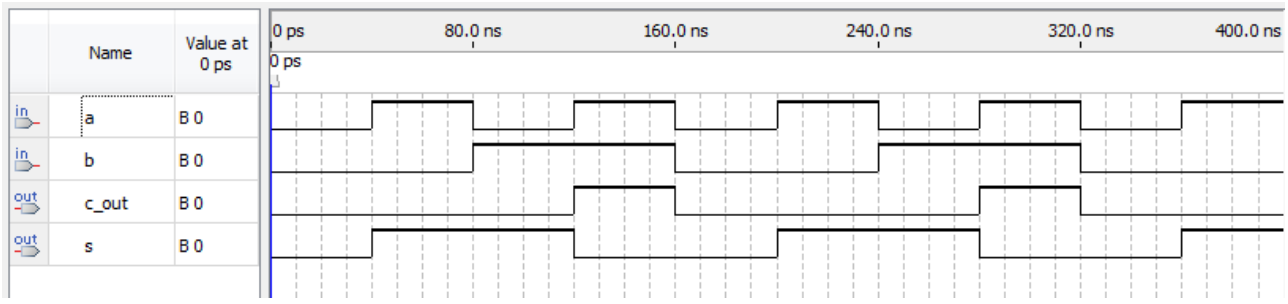
- Select the folder adder inside quartus2work directory for this task.

(1) Basic Binary Adder Circuits

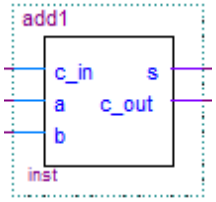
- Open the *adder_test* project (*adder_test.qpf*)
- Design a **Half-Adder**:
  - make a new BDF file with the name *ha.bdf*
  - enter your design as schematic
  - set as top-level-entity
  - compile and create a new symbol with the same name *ha.bsf*



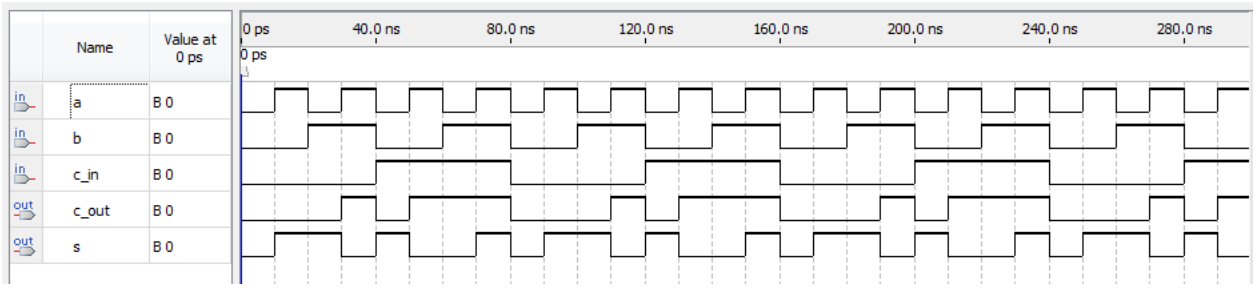
  - simulate your design with the prepared *ha.vwf* file or with your own



- Based on your equations, please design a 1 bit **Full-Adder** (based on two half-adders):
  - make a new BDF file *add1.bdf*
  - enter your design (use only your half-adder and OR symbols)
  - set as top-level-entity
  - compile and create a new symbol with the same name *add1.bsf*

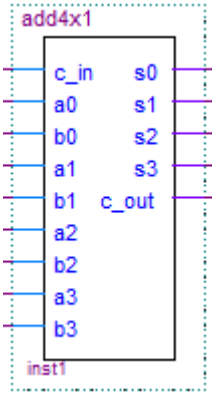


- simulate your design with the prepared *add1.vwf* file or with your own

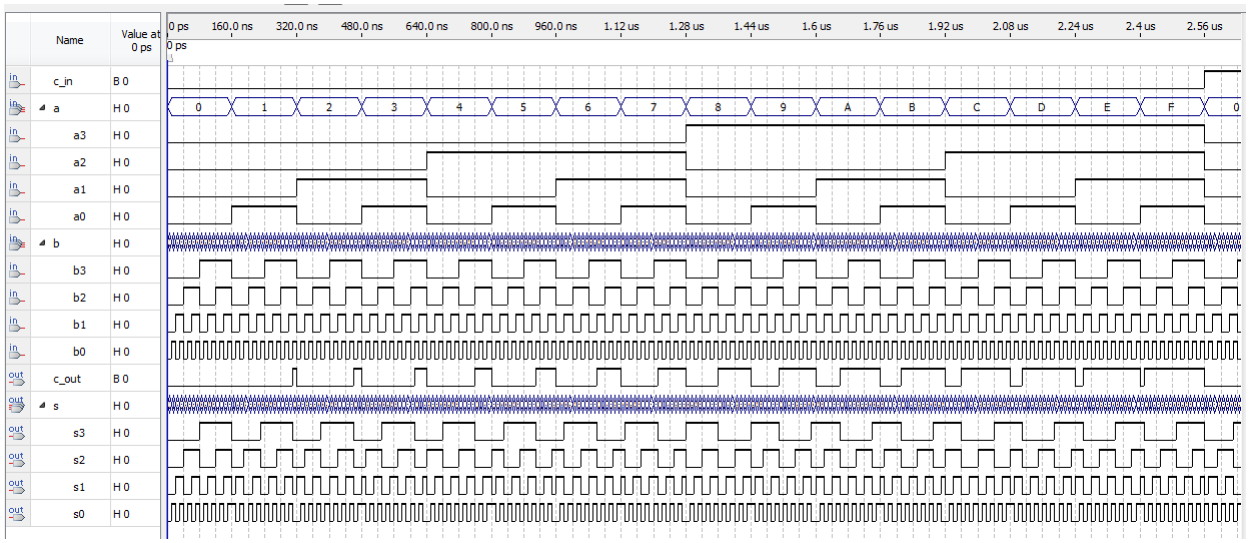


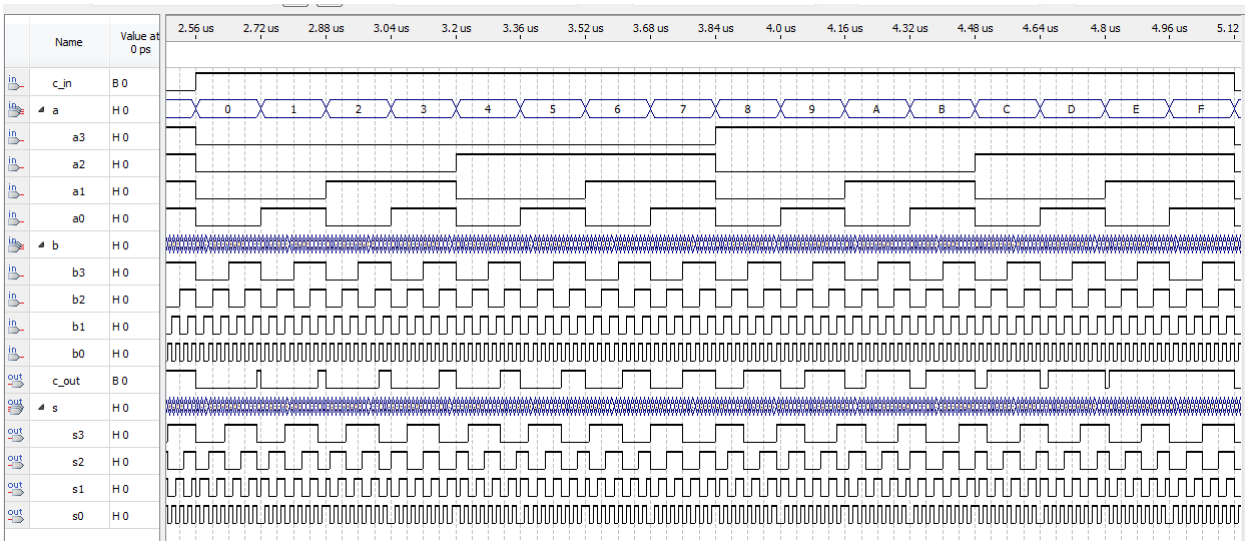
(2) Parallel Binary Adder Circuits

- Based on the full-adder design, please create a 4-bit parallel (ripple carry) adder:
  - make a new BDF file *add4x1.bdf*
  - enter your design (using the *add1* symbols, a full-adder at LSB)
  - set as top-level-entity
  - compile and create a new symbol with the same name *add4x1.bsf*



  - simulate your design with the prepared *add4x1.vwf* file or with your own
  - check the simulation results

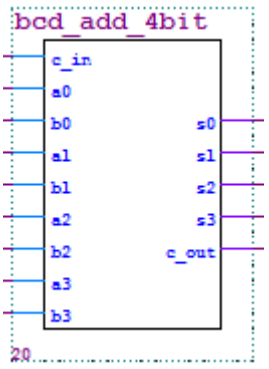




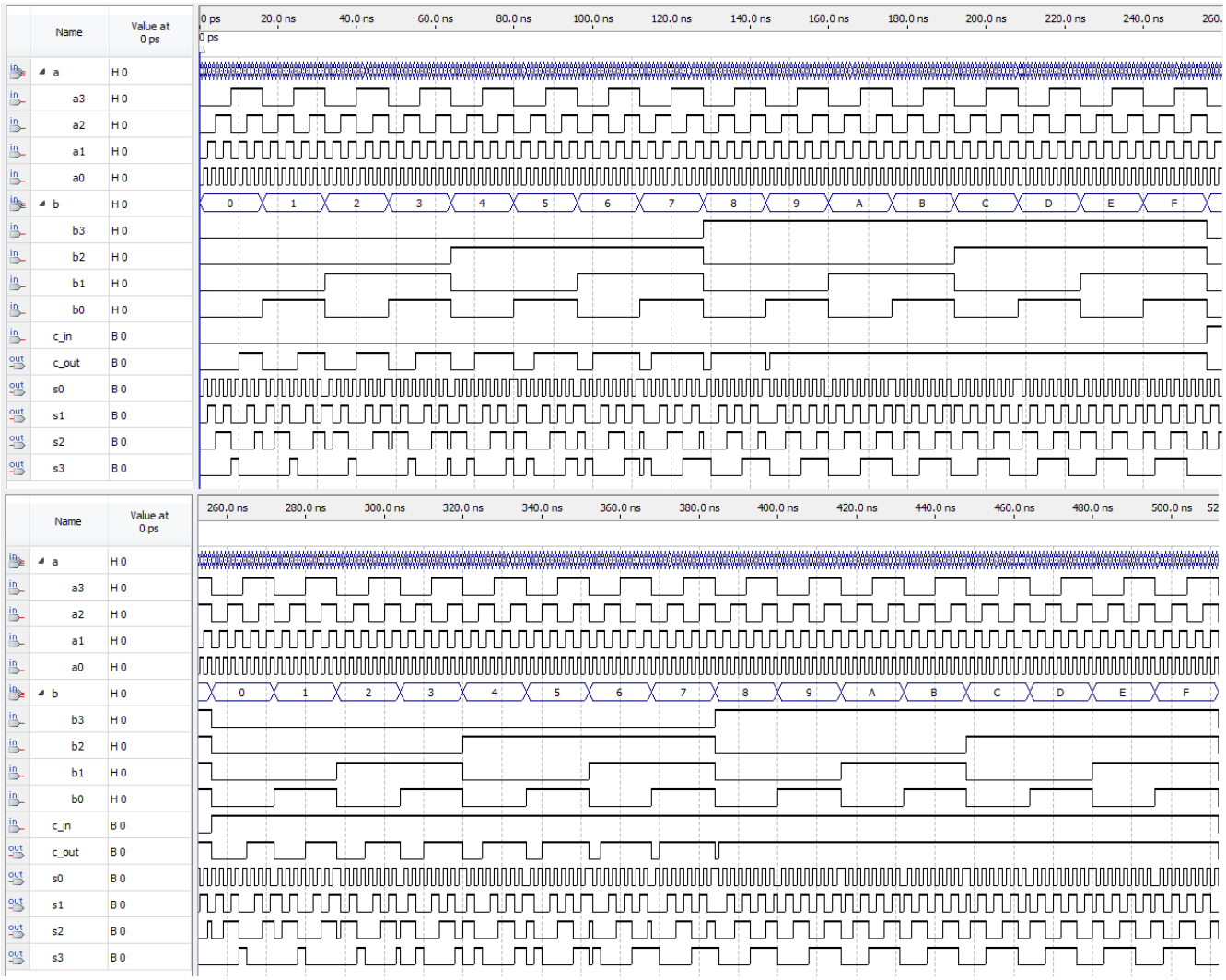
- Now test the whole adder design with the given file adder_test.bdf:
  - place the symbol add4x1 to the adder_test.bdf file
  - connect the symbol to the hex-coding switches and one of the 7 segment displays (using one of your 7-segment-decoder circuits)
  - set as top-level-entity
  - compile and program the result to the board (web-based or stand-alone)
  - test your design

(3) BCD Adder Circuits

- Open the *bcd_adder_test* project (*bcd_adder_test.qpf*)
- Based on the 4-bit adder design, please create a 1-digit BCD adder for the addition of two BCD numbers (one digit):
  - make a new BDF file *bcd_add_4bit.bdf*
  - add the files you need to the project (e.g. the *add4x1.bdf*)
  - enter your design (using two 4-bit adders and a simple additional logic for *invalid BCD correction*)
  - set as top-level-entity
  - compile and create a new symbol with the same name *bcd_add_4bit.bsf*

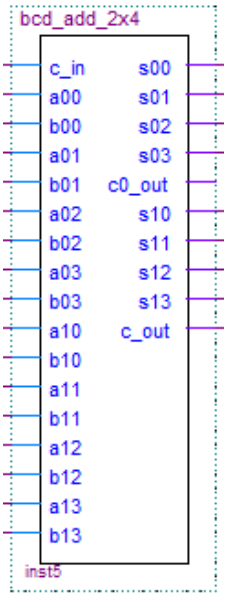


  - simulate your design with the prepared *bcd_add_4bit.vwf* file or with your own
  - check the simulation results



- Now test the whole BCD adder design with the given file *bcd_adder_test.bdf*:
  - place the symbol *bcd_add_4bit* to the *bcd_adder_test.bdf* file
  - connect the symbol to the hex-coding switches and one of the 7 segment displays (using one of your 7-segment-decoder circuits)
  - set as top-level-entity
  - compile and program the result to the board (web-based or stand-alone)
  - test your design

Now we are able to cascade 1-digit BCD adders to Multiple-Digit BCD adders.



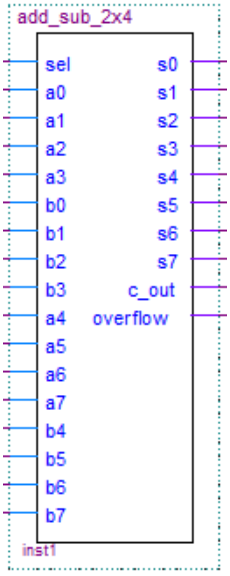
- Based on the 1-digit BCD adder design, please create a 2-digit BCD adder for the addition of two BCD numbers with 2 digits:
  - create a new BDF file *bcd_add_2x4.bdf*
  - enter your design (using two 1-digit BCD adders)
  - note: please label your inputs and outputs as follows:
    - *a[03..00]* for the LSD of a
    - *a[13..10]* for the MSD of a
    - *b[03..00]* for the LSD of b
    - *b[13..10]* for the MSD of b
    - *s[03..00]* for the LSD of s
    - *s[13..10]* for the MSD of s
  - set as top-level-entity

- compile and create a new symbol with the same name (*bcd_add_2x4.bsf*)
- simulate your design

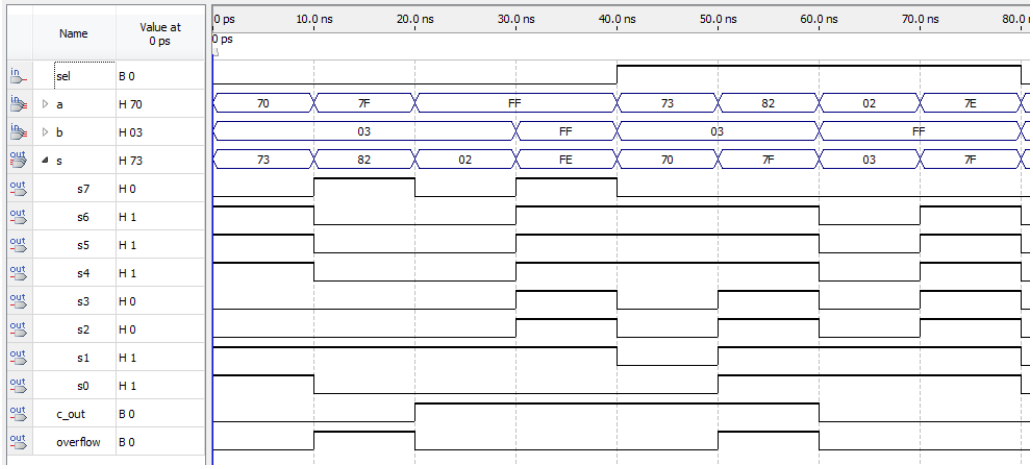
- Now test the whole 2-digit BCD adder design with the given file *bcd_adder_test.bdf*:
  - delete the old *bcd_add_4bit* symbol
  - place the symbols *bcd_add_2x4* to the *bcd_adder_test.bdf* file
  - connect the symbols to the hex-coding switches (for the 2-digit BCD number
  - a) and the 8 slide switches (for the 2-digit BCD number b) as well as to the three 7 segment displays (using one of your 7-segment-decoder circuits). The third 7 segment display is only used to display a "1", if we have a carry out.
  - set as top-level-entity
  - compile and program the result to the board (web-based or stand-alone)
  - test your design

(4) Parallel Binary Adder/Subtractor Circuits

- Open the *add_sub_test* project (*add_sub_test.qpf*)
- Based on the 4-bit adder design and the 2's complement arithmetic, please create an 8- bit adder/subtractor circuit, that will perform the addition (if the selection signal *sel = 0)* or the subtraction (*sel = 1*) and will generate the carry (*c_out*) and overflow (*overflow*) flags.
  - create a new BDF file *add_sub_2x4.bdf*
  - add the files you need to the project (e.g. the *add4x1.bdf*)
  - enter your design – based on the 4-bit adder (*add4x1*)
  - set as top-level-entity
  - compile and create a new symbol with the same name



  - simulate your design with the prepared *add_sub_2x4.vwf* file or with your own
  - check the simulation results

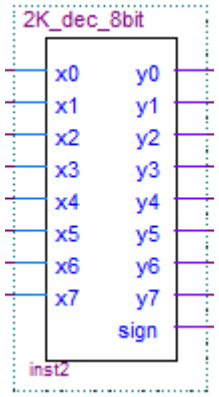


(5) Parallel Binary Adder/Subtractor with 2K-Decoder

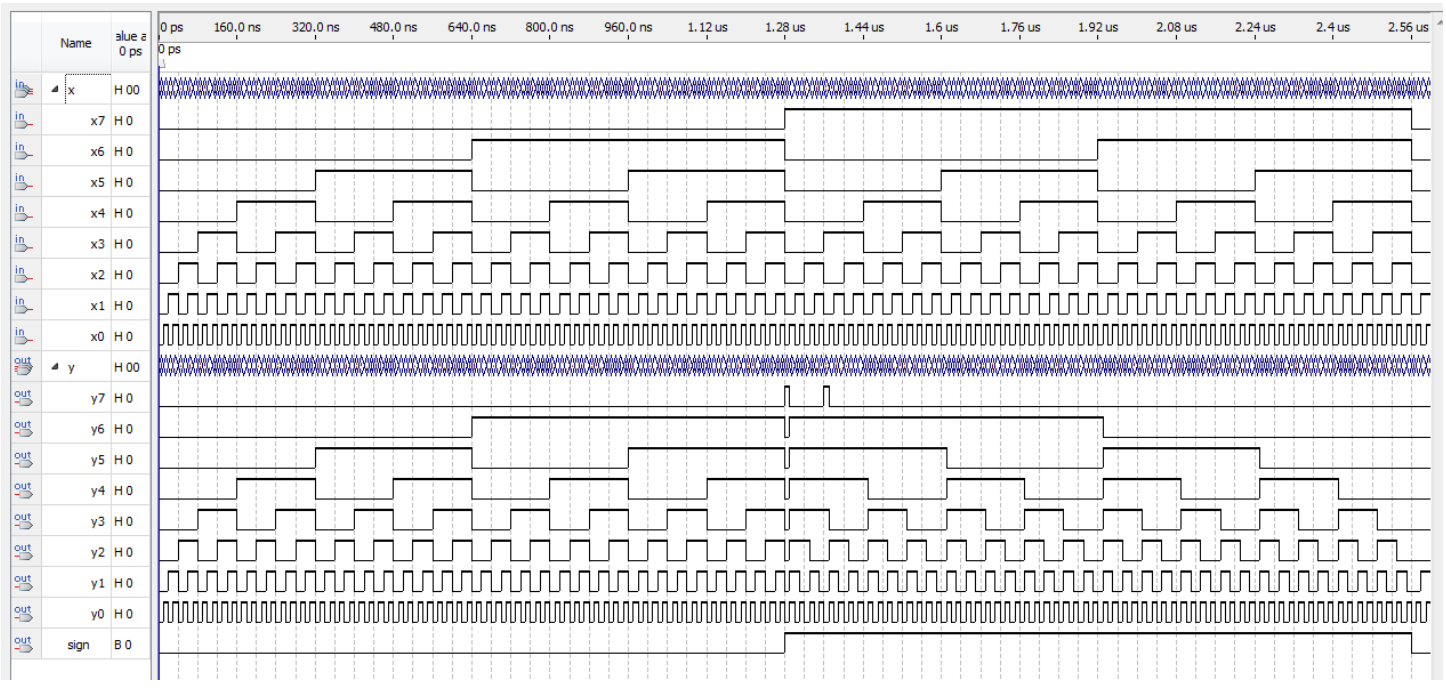
Before we will test the adder/subtractor design and show the result via the 7-segment display, we will design a **decoder for 2'complemet results**. This decoder will produce

- a sign-bit *sign* that signalises positive and negative numbers
- 0 .. 7 if the hex input value is 0 .. 7 (positive numbers, *sign = 0*) and
- -1 .. -8 if the hex input value is F .. 8 (negative numbers, *sign = 1*)

- Open the *add_sub_2K_test.qpf* project
- Based on the 1-bit full adder design and the 2's complement arithmetic, please create a 8-bit 2's complement decoder:
  - create a new BDF file *dec_2K_8bit.bdf*
  - add the files you need to the project (e.g. the *add1.bdf*)
  - enter your design – based on the 1-bit full adders (*add1*)
  - set as top-level-entity
  - compile and create a new symbol with the same name



  - simulate your design with the prepared *dec_2K_8bit.vwf* file or with your own
  - check the simulation results



- Finally we can test the whole 8-bit adder/subtractor design with the file *add_sub_2K_test.bdf*:
  - add the files you need to the project (e.g. the *add_sub_8x1.bdf*)
  - place the symbols *add_sub_8x1* and *2K_dec_8bit* to the *add_sub_2K_test.bdf* file
  - connect the 1-byte input signals a and b to the two hex-coding switches as well as the 8 slide switches
  - connect the output signals of the adder/subtractor with the 2's complement decoder

- for the select signal *sel*, please use one of the push buttons (to choose either addition or subtraction), because all of the slide switches are in use
- for the 1-byte result use two 7-segment displays
- connect the sign-bit *sign* to the third 7-segment display to display a minus if the result is negative
- connect the carry and overflow flags to two LEDs of the LED bar display
- set as top-level-entity
- compile and program the result to the board
- test your design

Questions:

1. How the Half-adder works?
2. How look like the half-adder truth table and Boolean function of the half-adder two outputs?
3. What is Karnaugh Map Methods and how it works?
4. What is Parallel Binary Subtraction, what logic components include?
5. What you know about Overflow and Carry bits?

Lab4 - Modular Combinational Logic

Part: Comparator

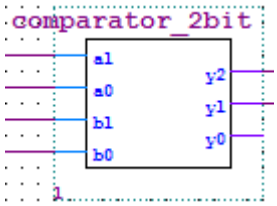
- Read the file "Modular Combinational Logic"
- Complete the truth tables, Karnaugh maps and derive all the equations from this.

Preliminary

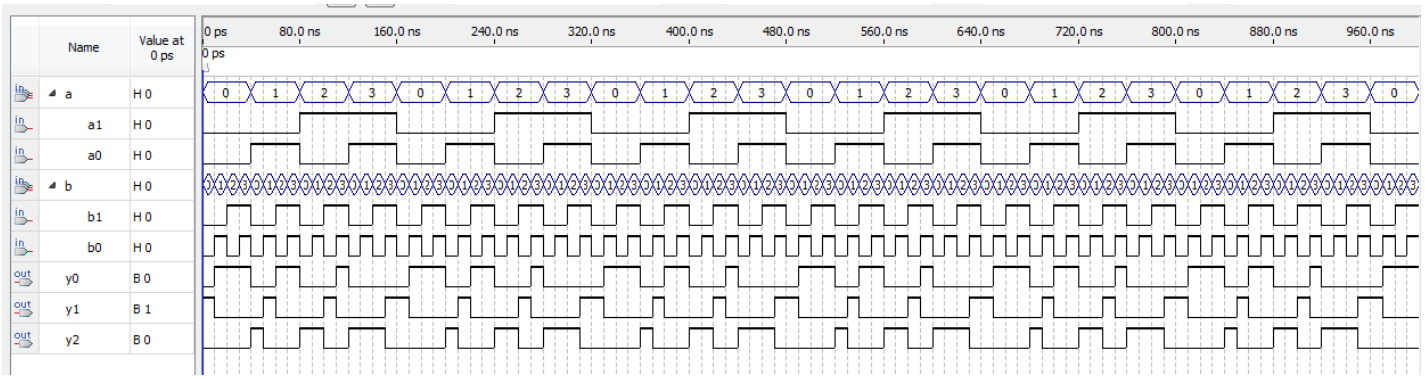
- Select the folder comparator inside the *max2work* root directory for this task.

(1) 2-bit Comparator

- Open the *comparator_test* project (*comparator_test.qpf*)
- Design a **2-bit Comparator**:
  - create a new BDF file *comparator_2bit.bdf*
  - enter your design (it is so simple – so a schematic would be the best solution)
  - set as top-level-entity
  - compile and create a new symbol with the same name



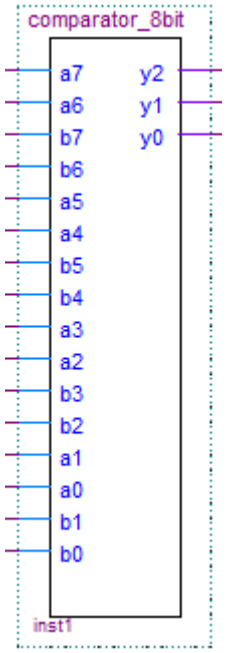
- simulate your design with the prepared *comparator_2bit.vwf* file or with your own
- check the simulation results



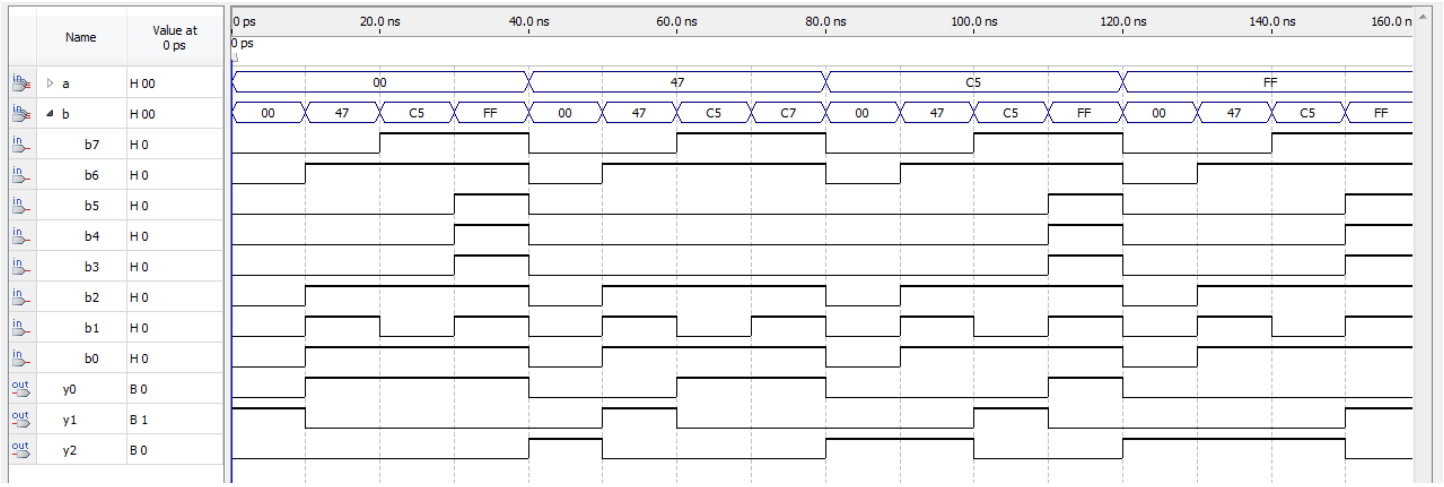
(2) 8-bit Comparator

- Based on the 2-bit comparator and following the comparison technique (shown in section VI.3), please create an 8-bit comparator:
  - create a new BDF file *comparator_8bit.bdf*

- enter your design (using the *comparator_2bit* symbols)
- set as top-level-entity
- compile and create a new symbol with the same name *comparator_8bit.bsf*



- simulate your design with the prepared *comparator_8bit.vwf* file or with your own
- check the simulation results

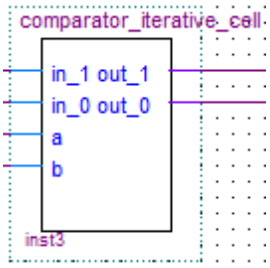


- Now test the whole adder design with the file *comparator_test.bdf*:
  - place the symbol *comparator_8bit* to the *comparator_test.bdf* file
  - connect the input *a* to the hex-coding switches and the input *b* to the 8 slide switches
  - connect the three output signals y to three LEDs of the LED bar display
  - set as top-level-entity
  - compile and program the result to the board
  - test your design

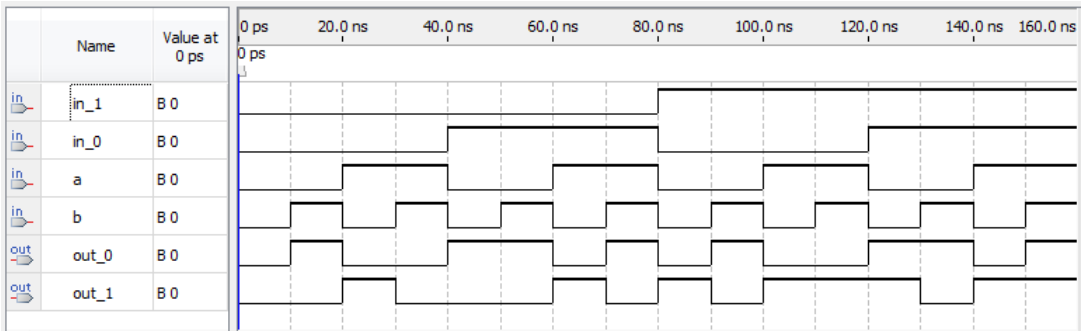
- Close the project

(3) 4-bit iterative comparator

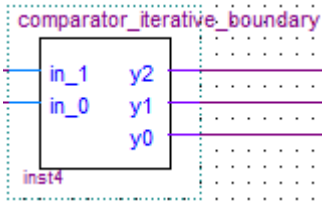
- Open the comparator_iterative_test project (comparator_iterative_test.qpf)
- Design a **1-bit comparator cell**:
  - create a new BDF file *comparator_iterative_cell.bdf*
  - enter your design
  - set as top-level-entity
  - compile and create a new symbol with the same name



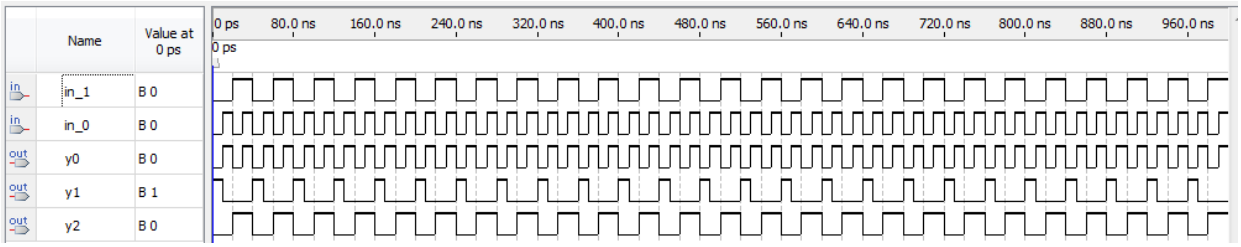
  - simulate your design with the prepared *comparator_iterative_cell.vwf* file or with your own
  - check the simulation results



- Now design the **boundary cell** to determine the result of the MSB comparator cell.
  - create a new BDF file *comparator_iterative_boundary.bdf*
  - enter your design
  - set as top-level-entity
  - compile and create a new symbol with the same name

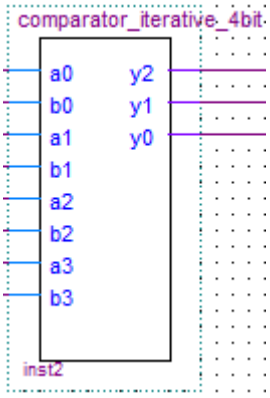


  - simulate your design with the prepared *comparator_iterative_boundary.vwf*
  - file or with your own
  - check the simulation results

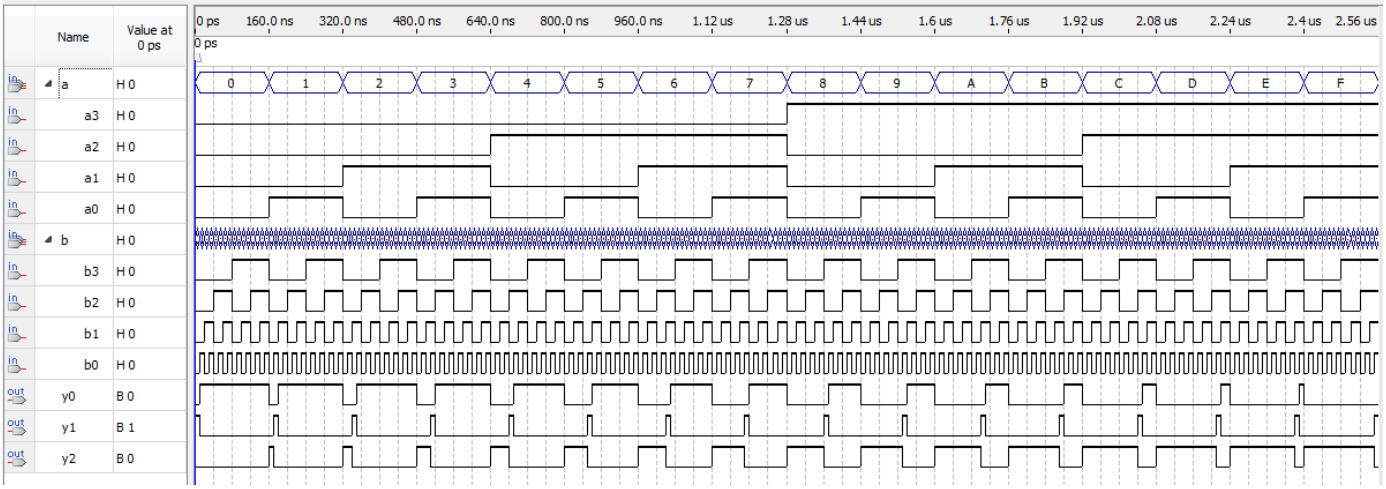


- Based on these components create an **4-bit iterative comparator**:
  - create a new BDF file *comparator_iterative_4bit.bdf*

- enter your design (using the 1-bit comparator and boundary cells)
- set as top-level-entity
- compile and create a new symbol with the same name



- simulate your design with the prepared *comparator_iterative_4bit.vwf* file or with your own
- check the simulation results



- Now test the whole adder design with the file *comparator_iterative_test.bdf*:
  - place the symbol *comparator_iterative_4bit* to the *comparator_iterative_test.bdf* file
  - connect the input *a* to the hex-coding switch 0 and the input *b* to the hex- coding switch 1
  - connect the three output signals y to three LEDs of the LED bar display
  - set as top-level-entity
  - compile and program the result to the board
  - test your design

Questions:

1. What is comparator and how it is works?
2. What kinds of comparator were created?
3. How is work 1-bit Comparator? To draw truth table, and write Boolean function?
4. What components the 8-bit comparator includes?

Lab5 - Modular Combinational Logic

Part: ALU

- Read the file "Modular Combinational Logic"
- Complete the truth tables, Karnaugh maps and derive all the equations from this.

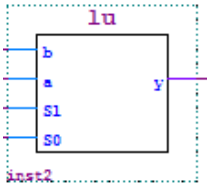
Preliminary

- Select the folder *alu* inside the *max2work* root directory for this task.
- Open the project *alu_test.qpf*

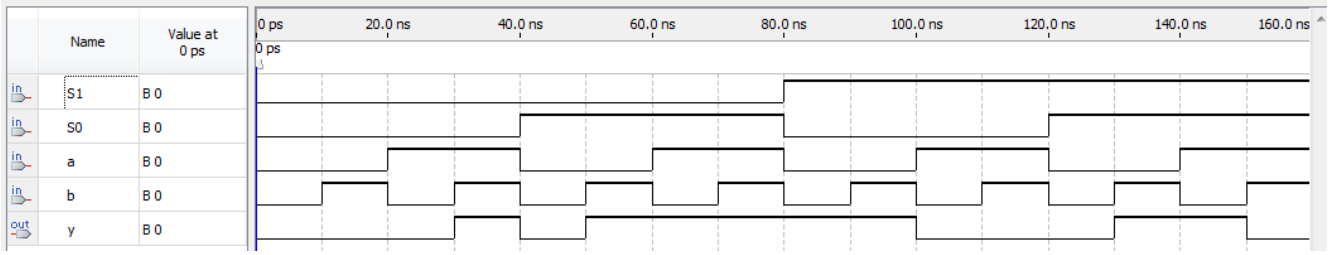
We will develop the whole 4-bit ALU design in a *hierarchical, top-down* fashion. For the 1- bit ALU slice we have to develop the **logic unit** (LU), the **arithmetic unit** (AU) and the **flag generator** as separate modules. Finally we have to cascade 4 of the 1-bit ALU slices to create our 4-bit ALU.

(1) LU – Logic Unit

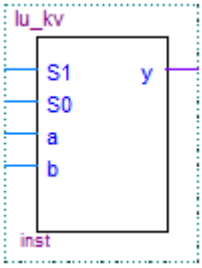
- Design a **1-bit logic unit based on single primitive logic and a multiplexer**:
  - create a new BDF file *lu.bdf*
  - enter your design (you can use the *mux4x1* from *ip_cores*)
  - set as top-level-entity
  - compile and create a new symbol with the same name *lu.bsf*



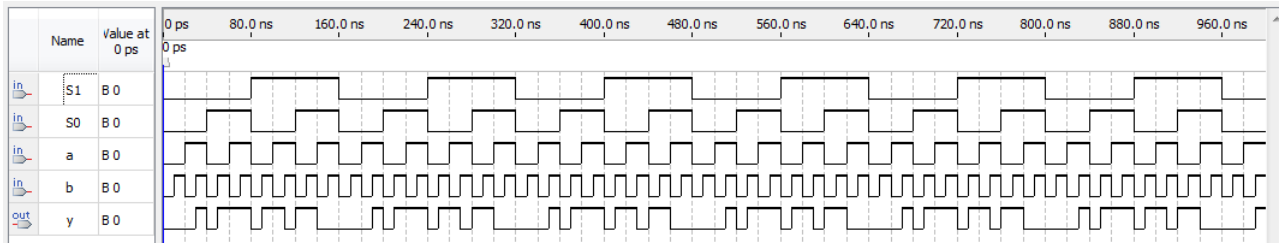
  - simulate your design with the prepared *lu.vwf* file or with your own
  - check the simulation results



- Design a **1-bit logic unit based on a two-level gate logic** as an alternative design:
  - create a new BDF file *lu_kv.bdf* (KV for a Karnaugh-Veitch map based design)
  - enter your design
  - set as top-level-entity
  - compile and create a new symbol with the same name



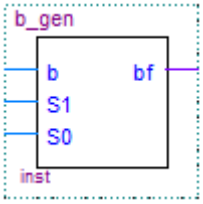
- simulate your design with the prepared *lu_kv.vwf* file or with your own
- check the simulation results



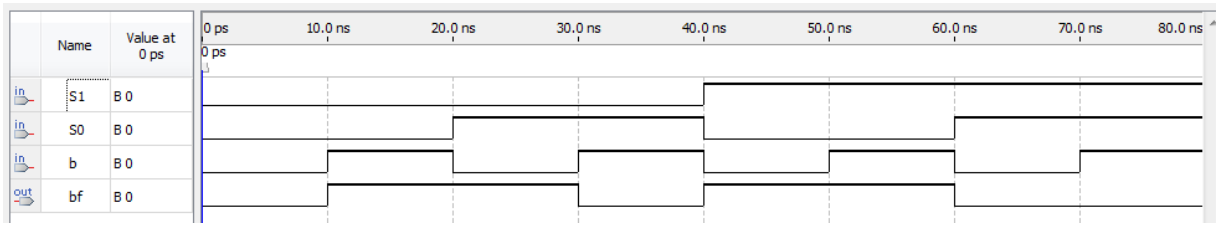
For further designs you can choose one of these two designs.

(2) ALU – Arithmetic Unit

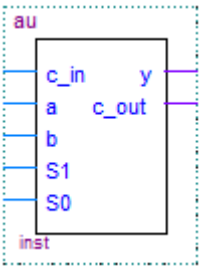
- Design the circuit **B-GEN** that will generate the *bf* input for the full-adder according the selection bits *s1* and *s0*:
  - create a new BDF file *b_gen.bdf*
  - enter your design
  - set as top-level-entity
  - compile and create a new symbol with the same name



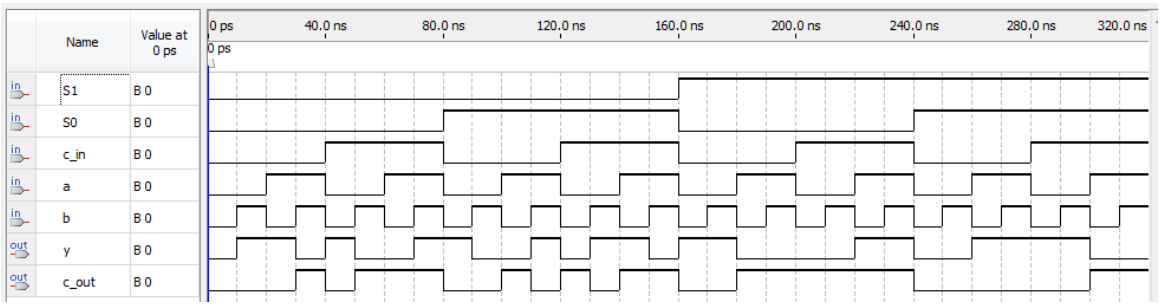
  - simulate your design with the prepared *b_gen.vwf* file or with your own
  - check the simulation results



- Design a **1-bit arithmetic unit**:
  - based on the *b_gen* and the *add1* circuits create a new BDF file *au.bdf*
  - enter your design (add the files you need to the project: *add1*, *ha*)
  - set as top-level-entity
  - compile and create a new symbol with the same name

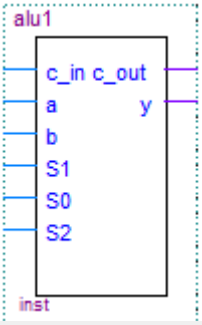


  - simulate your design with the prepared *au.vwf* file or with your own
  - check the simulation results

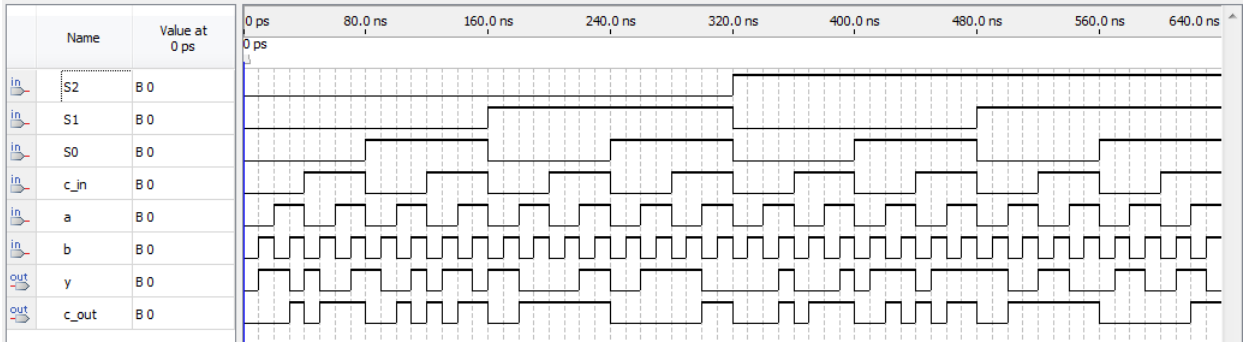


(3) 1-bit ALU slice

- Design an **1-bit ALU slice:**
  - create a new BDF file *alu1.bdf*, based on the lu and the au circuits as well as a *2-to-1 multiplexer* circuit
  - enter your design
  - set as top-level-entity
  - compile and create a new symbol with the same name



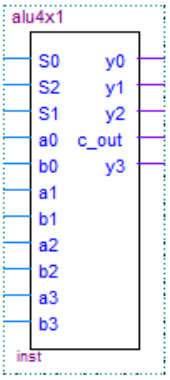
  - simulate your design with the prepared *alu1.vwf* file or with your own
  - check the simulation results



(4) 4-bit ALU

Now we are ready to design the whole 4-bit ALU.

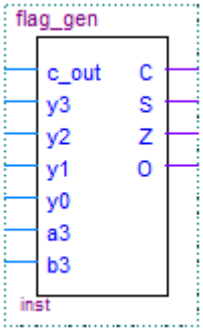
- Design a **4-bit ALU**:
  - create a new BDF file *alu4x1.bdf*, based on the *alu1* circuits
  - don't forget to connect the LSB carry-input *c-1* with the selection-code signal *s0* to generate the initial carry-input
  - enter your design
  - set as top-level-entity
  - compile and create a new symbol with the same name



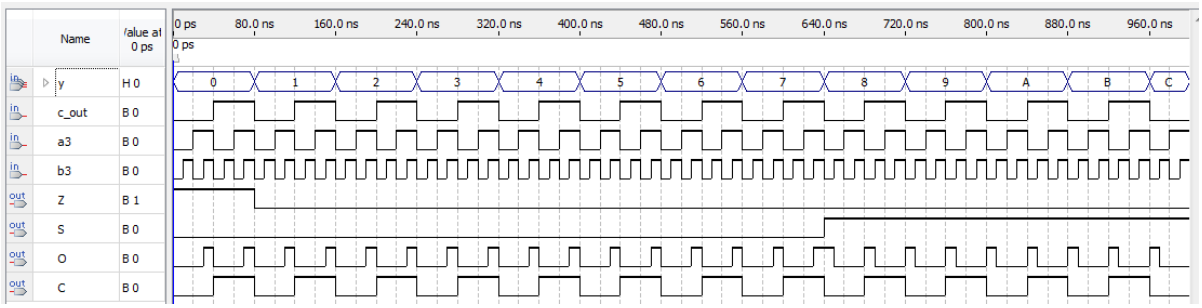
  - simulate your design with the prepared *alu4x1.vwf* file or with your own
  - check the simulation results

(5) Flag Generator

- Design a **flag generator** circuit that determines C, O, Z and S conditions:
  - create a new BDF file *flag_gen.bdf*
  - enter your design
  - set as top-level-entity
  - compile and create a new symbol with the same name



  - simulate your design with the prepared *flag_gen.vwf* file or with your own
  - check the simulation results



- Now test the whole alu and flag design with the file *alu_test.bdf*:
  - place the symbols *alu4x1* and *flag_gen* to the *alu_test.bdf* file
  - connect the flag generator to the 4-bit ALU module
  - connect the inputs *a* and *b* to the two hex-coding switches and the selection code signals *s* to three of the 8 slide switches to select the different operations
  - connect the output signals y to one of the 7-segment displays
  - connect the four flag outputs to four LEDs of the LED bar display
  - set as top-level-entity
  - compile and program the result to the board
  - test your design

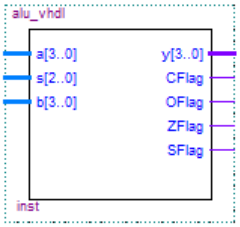
- Close the project

(6) 4-bit ALU design in VHDL

- Open the alu_vhdl_test project (*alu_vhdl_test.qpf*)
- Design the same 4-bit ALU functionality using VHDL specification:
  - open the VHDL file *alu_vhdl.vhd*
  - enter your design

```
-- 4-bit Arithmetic Logic Unit
-- Capable of implementing 4 arithmetic and 4 logic functions
-- Input : operands a and b, selection code s
-- Output: result y

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY alu_vhdl IS
PORT(
    a       :   IN  STD_LOGIC_VECTOR(3 downto 0);
    s       :   IN  STD_LOGIC_VECTOR(2 downto 0);
    b       :   IN  STD_LOGIC_VECTOR(3 downto 0);
    y       :   OUT STD_LOGIC_VECTOR(3 downto 0);
    CFlag,OFlag,ZFlag,SFlag :   OUT STD_LOGIC);
END alu_vhdl;

ARCHITECTURE arch OF alu_vhdl IS

    INSERT YOUR CODE HERE

END arch;
```

- set as top-level-entity
- compile and create a new symbol with the same name



- simulate your design with the prepared *alu_vhdl.vwf* file or with your own
- check the simulation results

- Now test the whole ALU design with the file *alu_vhdl_test.bdf*:
  - place the symbols *alu_vhdl* to the *alu_vhdl_test.bdf* file
  - connect the inputs *a* and *b* to the two hex-coding switches and the selection code signals *s* to three of the 8 slide switches to select the different operations
  - connect the output signals y to one of the 7-segment displays
  - connect the four flag outputs to four LEDs of the LED bar display
  - set as top-level-entity
  - compile and program the result to the board
  - test your design

Questions:

1. What is ALU, where it is used?
2. To draw ALU inputs and outputs.
3. What is flag register in ALU and what the each kind of the flags means?
4. What modules include the ALU? To tell about each of them.

Literature

1. K. Henke, G. Tabunshchyk, H.-D. Wuttke, T. Vietzke, St. Ostendorff “Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems”, iJOE – Volume 10, Issue 5, 2014 - pp.57-62
2. A. Parkhomenko and other. Remote and virtual tools in engineering: student textbook/ general editorship Dr.Ing.Karsten Henke. – Zaporizhzhya: Dike pole, 2016. –pp. 250
3. Sebven-segment display [Electronic resource] Wikipedia – http://en.wikipedia.org/wiki/7_segment
4. MAX+PLUS® II Programmable Logic Development System AHDL, Copyright © 1995 Altera Corporation – p. 243
5. VHDL Tutorial [Electronic resource] Nand-Land – https://www.nandland.com/vhdl/tutorials/tutorial-introduction-to-vhdl-for-beginners.html
6. Xin Dong, Minghai Peng. Project Report for ASIC Design “Design of a 4-bit comparator” Department of Electrical and Computer Engineering Concordia University – p.35