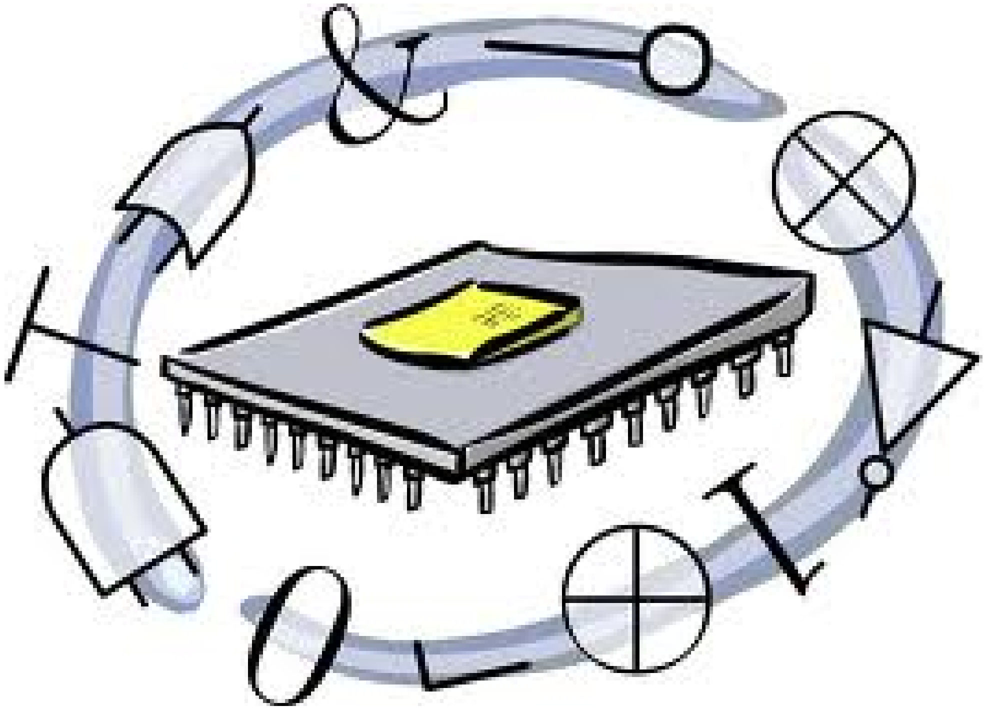
Basic Components in VHDL



23/05/2016

Binary logic

- Logic “variables” on which logic “functions” work
  - Variables: Binary (0 or 1)
  - E.g. A can have the value 0 or 1
    - **A = 0, lamp is out**
    - **A = 1, lamp is on**

Tempus

Binary logic

- Logical levels
  - In digital electronics:
    - signal level: HIGH (1) of LOW (0)
  - (e.g.) TTL- logic:
    - LOW: voltage between 0V en 0.8V
    - HIGH: voltage between 2V en 5 V
  - On Spartan 3E: 3.3V, 2.5V and 1.2V

Binary logic

Logical statements

A logical statement can be true or false

The results usually depends on some independent parameters (inputs)

In the Boolean logic:

AND =	“ . ”
OR =	“ + ”
NOT =	“ ’ ” or “ ¯ ” or “ \ ”
IF =	“ = “

Binary logic

Logic statements & equations

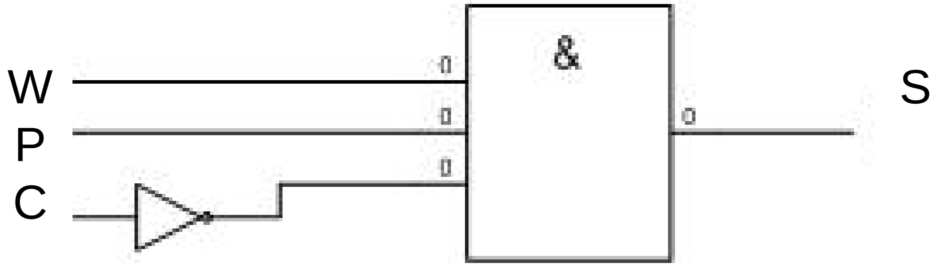
Example: We go for an outdoor <u>swim</u> (S) if the <u>weather</u> is nice (W), the <u>pool</u> is open(P) and it is <u>not</u> <u>crowded</u> (C).

So:	"swim" is true (S = 1)	if
	"nice weather" is true (W = 1)	and
	"pool open" is true (P = 1)	and
	"crowded" is not true (C = 0)	

Logic equation:

$$S = W.P.C'$$

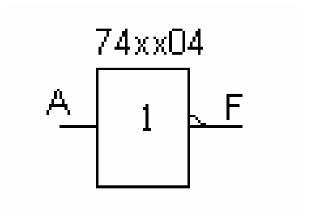
Schematic:



Logic gates

Not gate

IEC-SYMBOOL



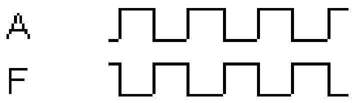
WAARHEIDSTABEL

A	F (NOT)
0	1
1	0

LOGISCHE VERGELIJKING

$F = \overline{A} = /A = A'$

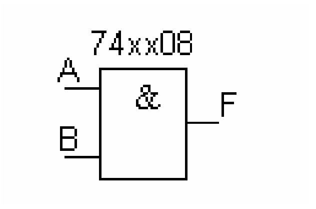
TIJDSDIAGRAMMA



Logic gates

And gate

IEC-SYMBOOL



All possibilities for 2 inputs

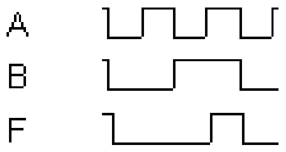
WAARHEIDSTABEL

B	A	F (AND)
0	0	0
0	1	0
1	0	0
1	1	1

LOGISCHE VERGELIJKING

$F = A \bullet B = AB$

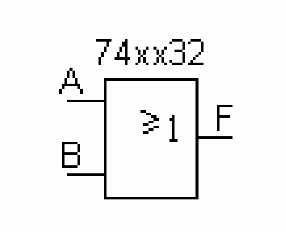
TIJDSDIAGRAMMA



Logic gates

Or gate

IEC-SYMBOOL



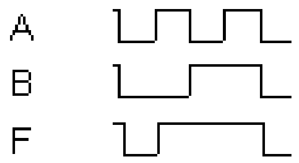
WAARHEIDSTABEL

B	A	F (OR)
0	0	0
0	1	1
1	0	1
1	1	1

LOGISCHE VERGELIJKING

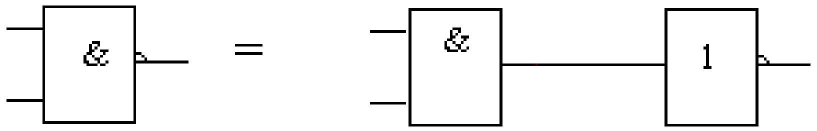
F = A + B

TIJDSDIAGRAMMA

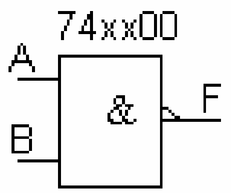


Logic gates

NAND gate



IEC-SYMBOOL



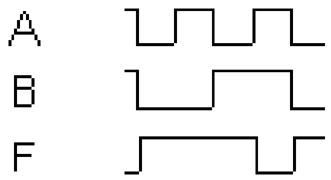
WAARHEIDSTABEL

B	A	AND	F (NAND)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

LOGISCHE VERGELIJKING

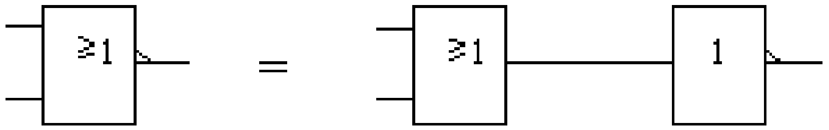
$$F = \overline{A \bullet B} = \overline{AB}$$

TIJDSDIAGRAMMA

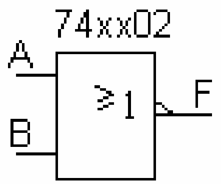


Logic gates

NOR gate



IEC-SYMBOOL



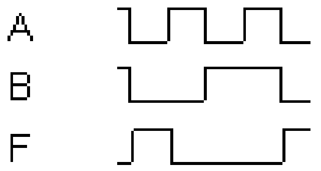
WAARHEIDSTABEL

B	A	OR	F (NOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

LOGISCHE VERGELIJKING

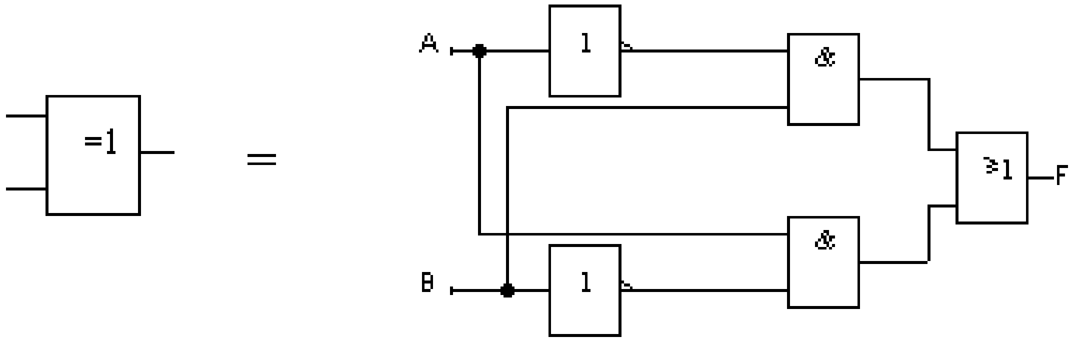
$$F = \overline{A + B}$$

TIJDSDIAGRAMMA

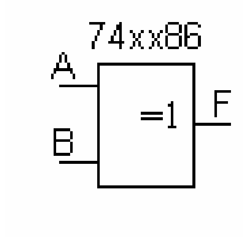


Logic gates

EXOR gate



IEC-SYMBOOL



WAARHEIDSTABEL

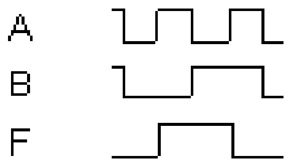
B	A	F (EXOR)
0	0	0
0	1	1
1	0	1
1	1	0

LOGISCHE VERGELIJKING

$F = \overline{A}B + A\overline{B}$

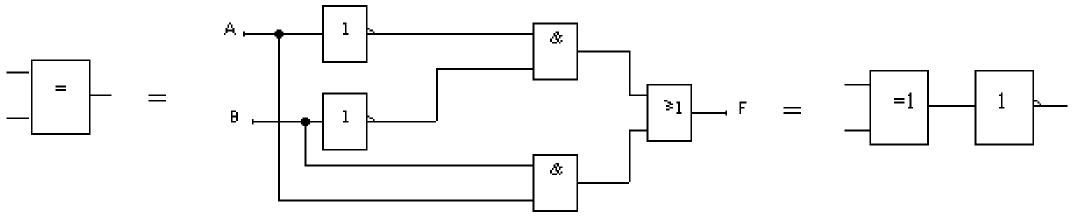
$F = A \oplus B$

TIJDSDIAGRAMMA

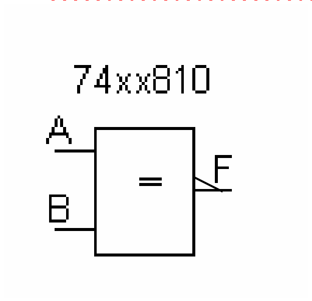


Logic gates

EXNOR gate



IEC-SYMBOOL



WAARHEIDSTABEL

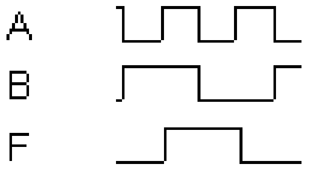
B	A	F (EXNOR)
0	0	1
0	1	0
1	0	0
1	1	1

LOGISCHE VERGELIJKING

$$F = \overline{A}\,\overline{B} + AB$$

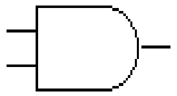
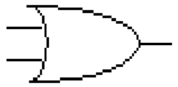
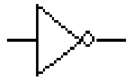
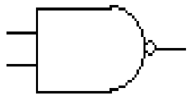
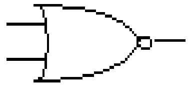
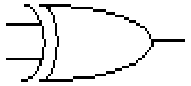
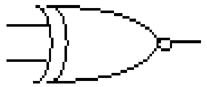
$$F = \overline{A \oplus B}$$

TIJDSDIAGRAMMA



Logic gates

American symbols

AMERIKAANSE SYMBOLIEK		
AND-POORT	OR-POORT	NOT-POORT
		
NAND-POORT	NOR-POORT	
		
EXOR-POORT	EXNOR-POORT	
		

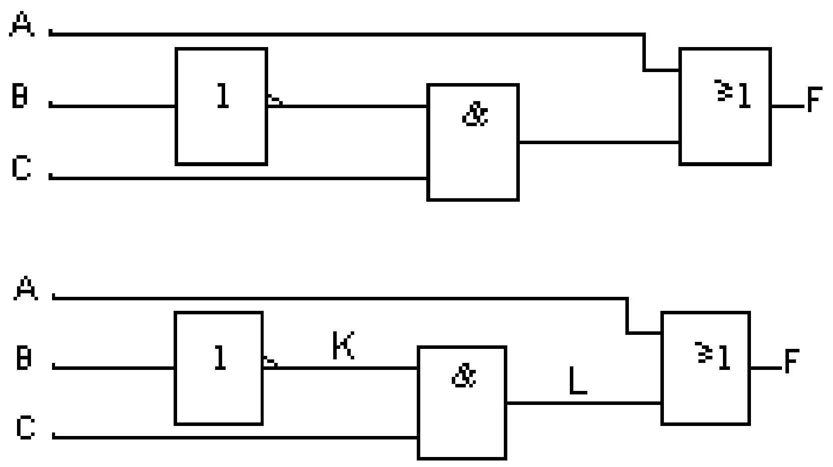
Logic gates

- schematic
- logic equation
- truth table
- timing diagram

$\left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\}$ => they are all connected

Logic gates

- From schematic



- To truth table

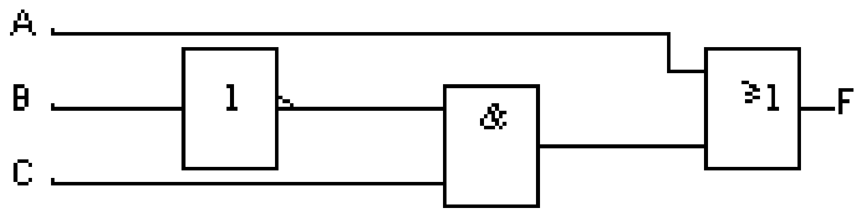
$K = \overline{B}$
$L = K \bullet C$
$F = A + L$

C	B	A	K	L	F
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	1

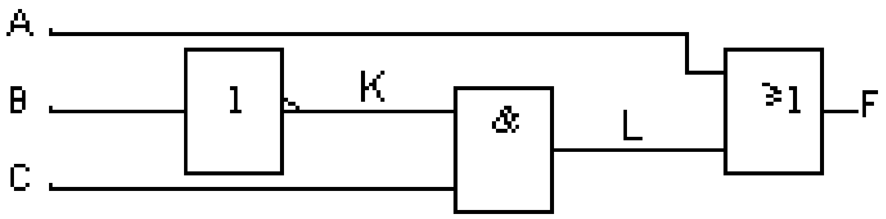
All possiblities for 3 inputs

Logic gates

- From schematic



- To logic equation



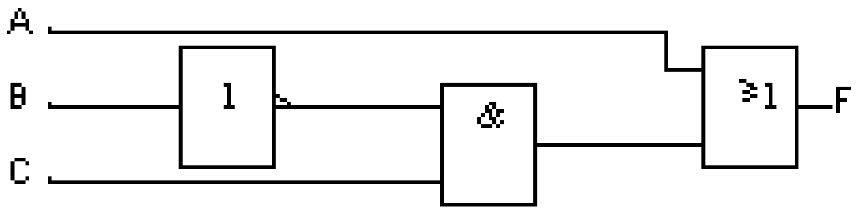
$$F = A + L$$

$$F = A + (K \bullet C)$$

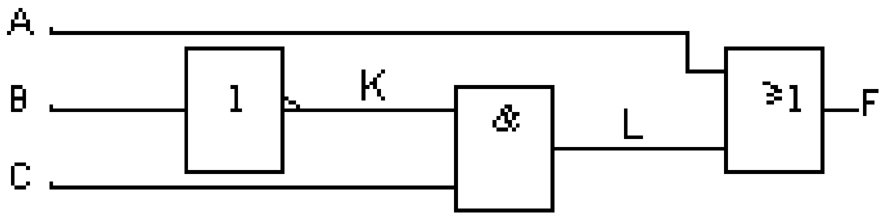
$$F = A + (\overline{B} \bullet C)$$

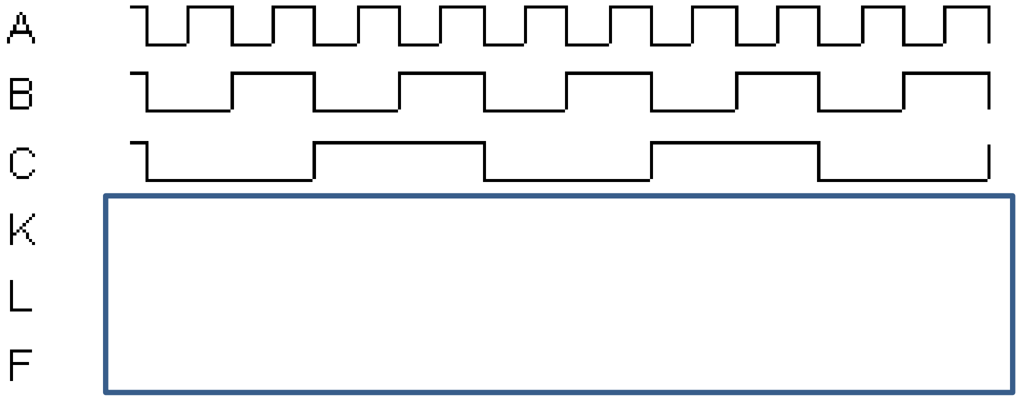
Logic gates

- From schematic



- To timing diagram





Tempus

Logic gates

- From truth table

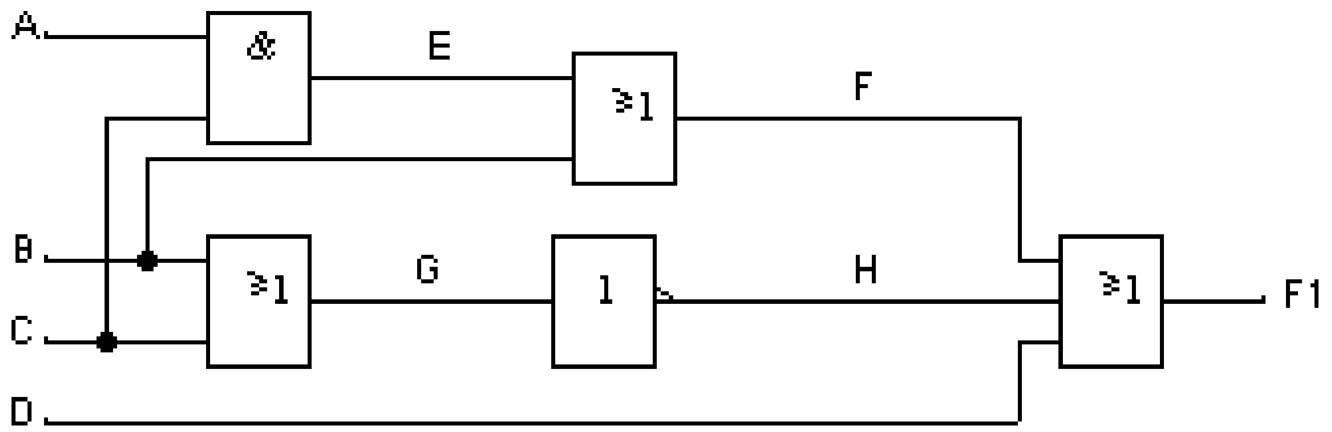
C	B	A	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

- To logic equation

C	B	A	F	productterm
0	0	0	0	
0	0	1	1	$A \bullet \overline{B} \bullet \overline{C}$
0	1	0	0	
0	1	1	1	$A \bullet B \bullet \overline{C}$
1	0	0	1	$\overline{A} \bullet \overline{B} \bullet C$
1	0	1	1	$A \bullet \overline{B} \bullet C$
1	1	0	0	
1	1	1	1	$A \bullet B \bullet C$

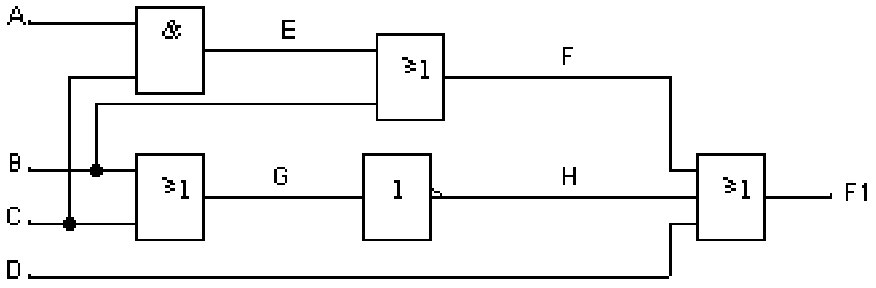
$$F = A\overline{B}\,\overline{C} + AB\overline{C} + \overline{A}\,\overline{B}C + A\overline{B}C + ABC$$

Exercise 1



- Truth table
- Logic equation
- Timing diagram

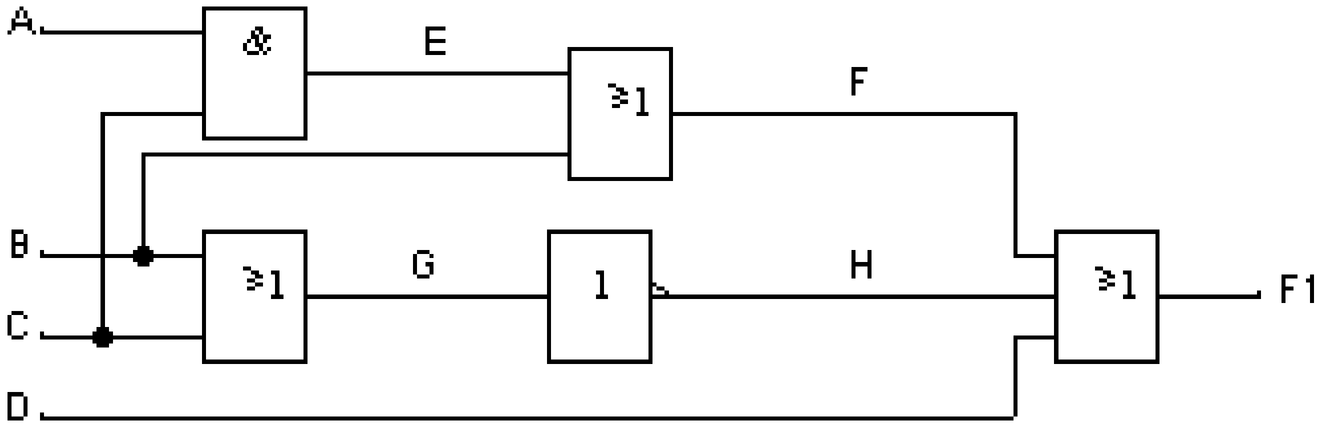
Exercise 1



- Truth table
- F1 is 1 if
  - D is '1' <u>or</u>
  - H is '1' => if G is '0'
  - => if B <u>and</u> C both '0' <u>or</u>
  - F is '1' => B is '1' <u>or</u> E is '1' => if A <u>and</u> C both '1'

D	C	B	A	F1
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Exercise 1



- Logic equation:
- F1 = F + H + D

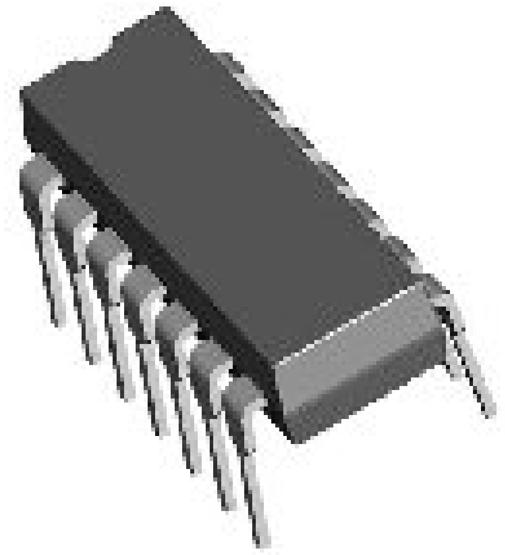
  = E + B + G’ + D

  = (A.C + B) + (B + C)’ + D

Tempus

Logic ICs

Single gates

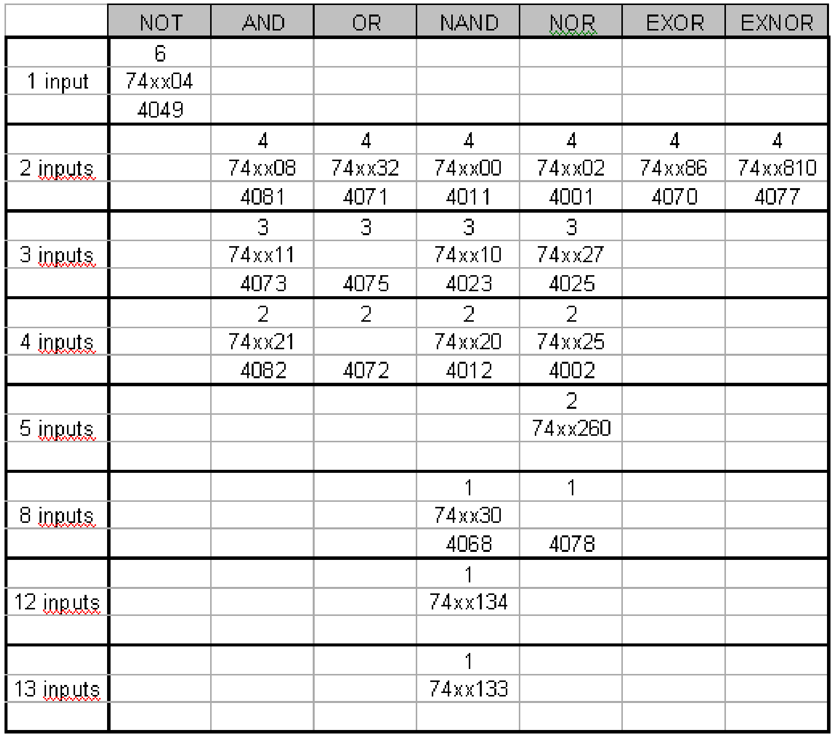


DIL : dual in line

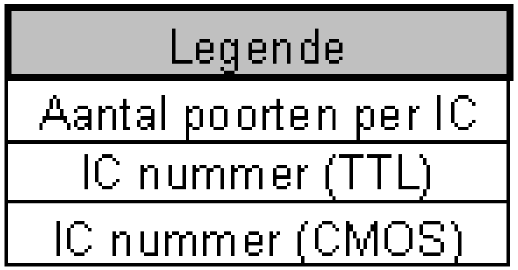
DIP: dual in line package

Logic ICs

Single gates



	NOT	AND	OR	NAND	NOR	EXOR	EXNOR
	6						
1 input	74xx04						
	4049						
		4	4	4	4	4	4
2 inputs		74xx08	74xx32	74xx00	74xx02	74xx86	74xx810
		4081	4071	4011	4001	4070	4077
		3	3	3	3		
3 inputs		74xx11		74xx10	74xx27		
		4073	4075	4023	4025		
		2	2	2	2		
4 inputs		74xx21		74xx20	74xx25		
		4082	4072	4012	4002		
					2		
5 inputs					74xx260		
				1	1		
8 inputs				74xx30			
				4068	4078		
				1			
12 inputs				74xx134			
				1			
13 inputs				74xx133			

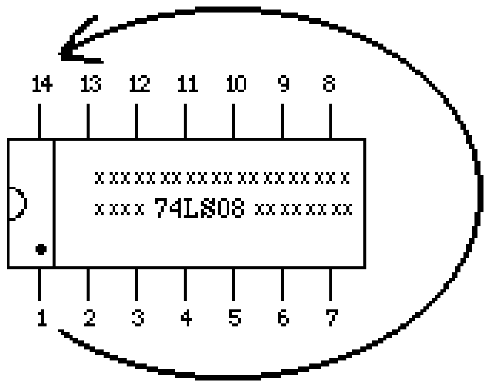


Legende
Aantal poorten per IC
IC nummer (TTL)
IC nummer (CMOS)

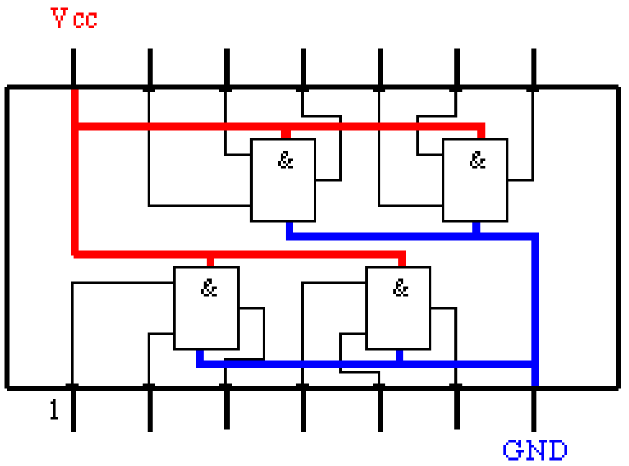
Logic ICs

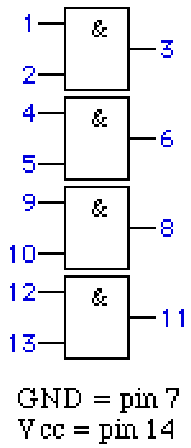
Single gates

- Pin numbering



- Datasheet

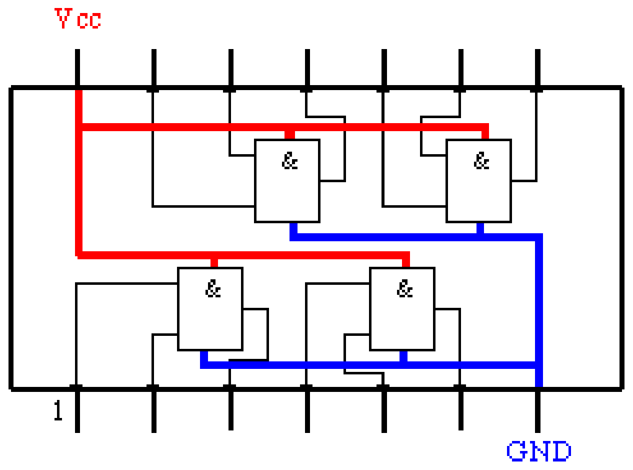


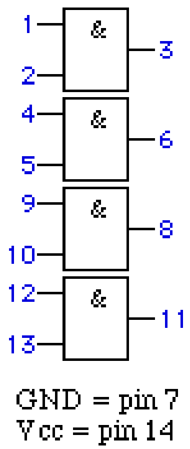


Logic ICs

Single gates

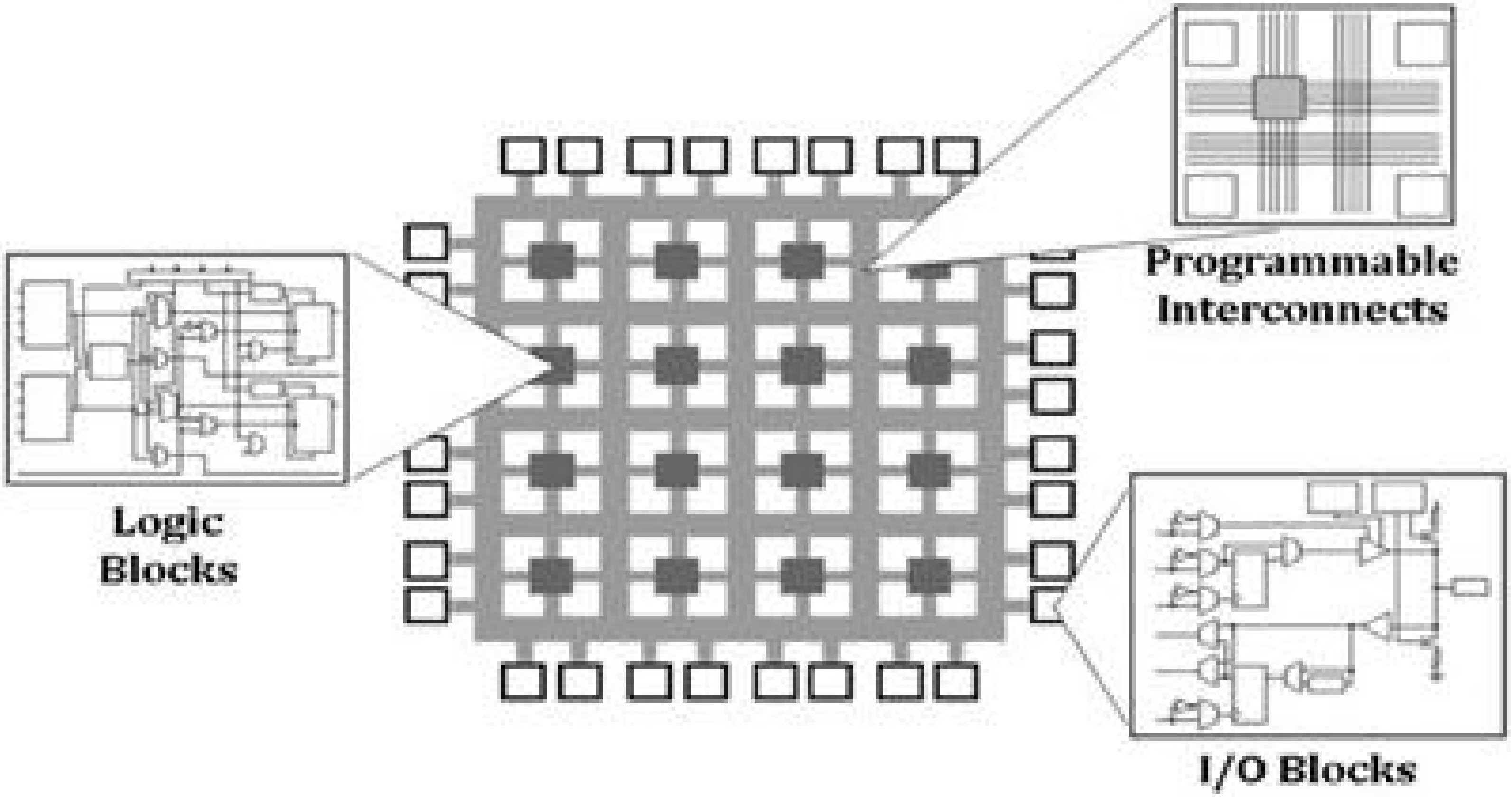
- Impossible to use for complex logic equations!





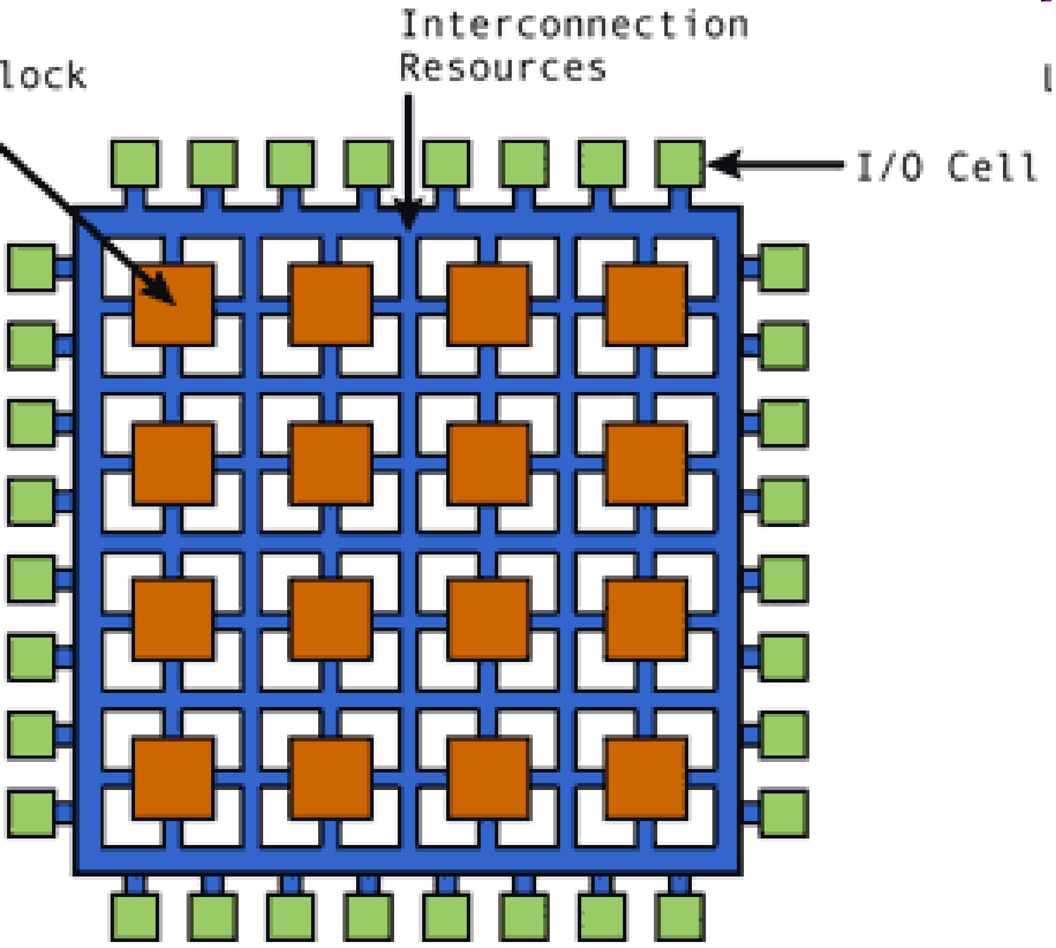
Logic ICs

FPGAs



Logic ICs

FPGAs



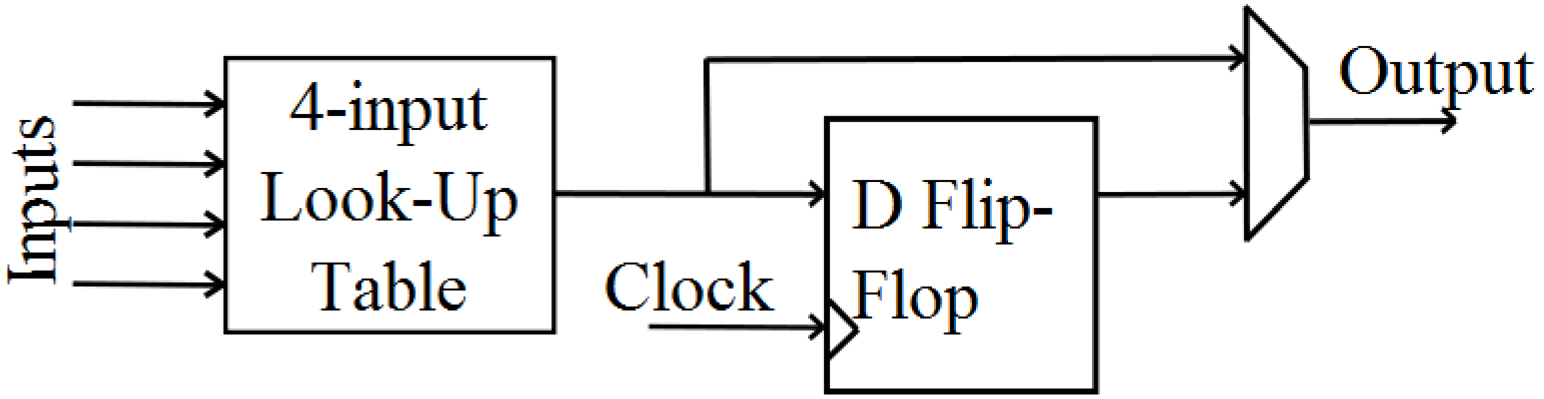
"Field Programmable Logic Array"

- FPGAs can be any digital component
- Reconfigurable

Logic ICs

FPGAs

- Configurable Logic Blocks
  - Combinatorial & sequential part



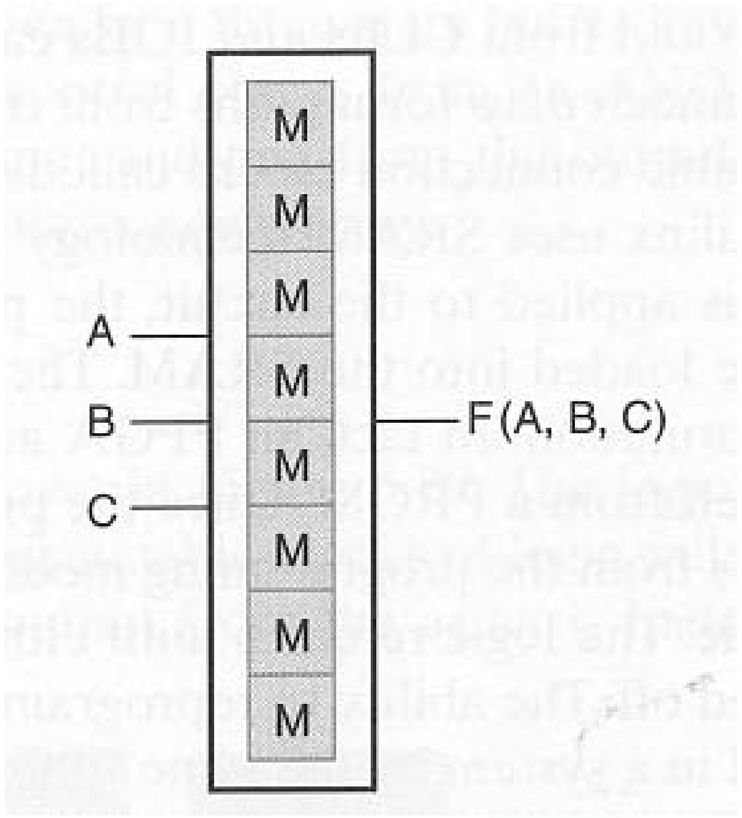
Logic ICs

FPGAs

- Combinatorial Logic = hardware truth table = Look-Up Table

Xilinx cells are based on the use of SRAM as a look-up table. The truth table for a K-input logic function is stored in a 2^K x 1 SRAM.

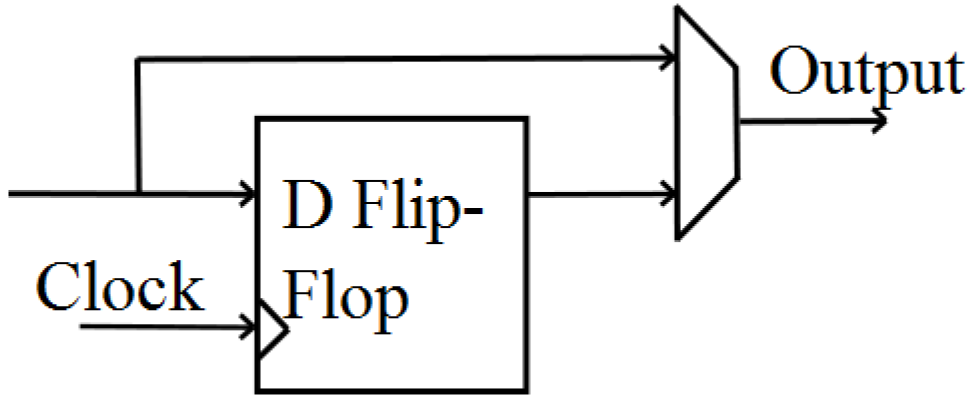
The address lines of the SRAM function as inputs of the logic equation and the output (data) line of the SRAM provides the value of the logic function.



Logic ICs

FPGAs

- D-flipflop is memory element: <u>sequential</u> logic
- It reads input on clock flank
- Is only used when clock is used
- Clock should drive all flipflops at the same time

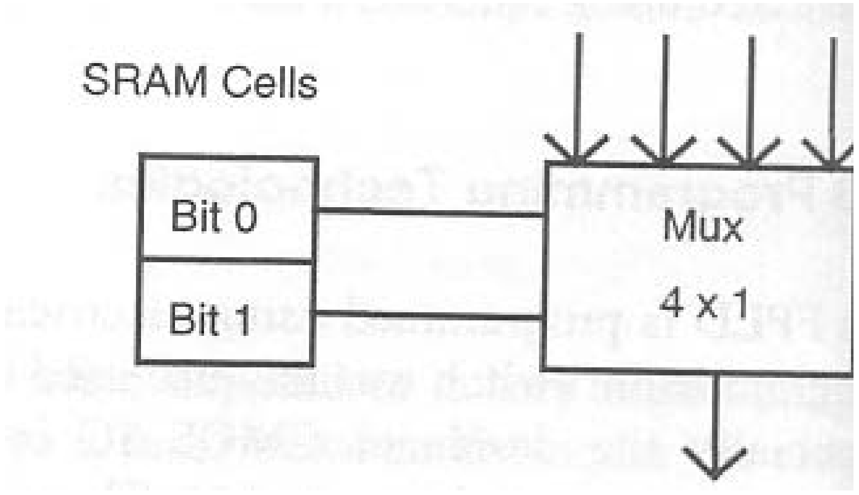


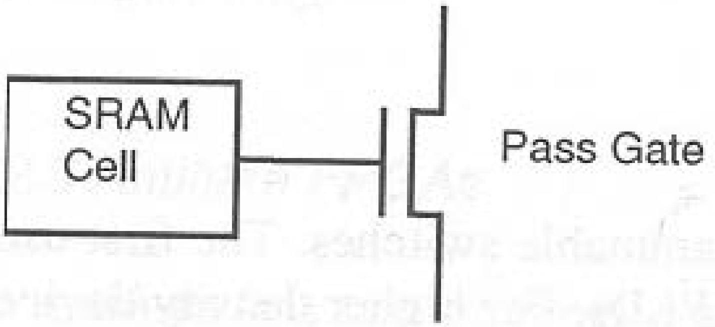
Logic ICs

FPGAs

- Interconnection

  SRAM to make interconnections

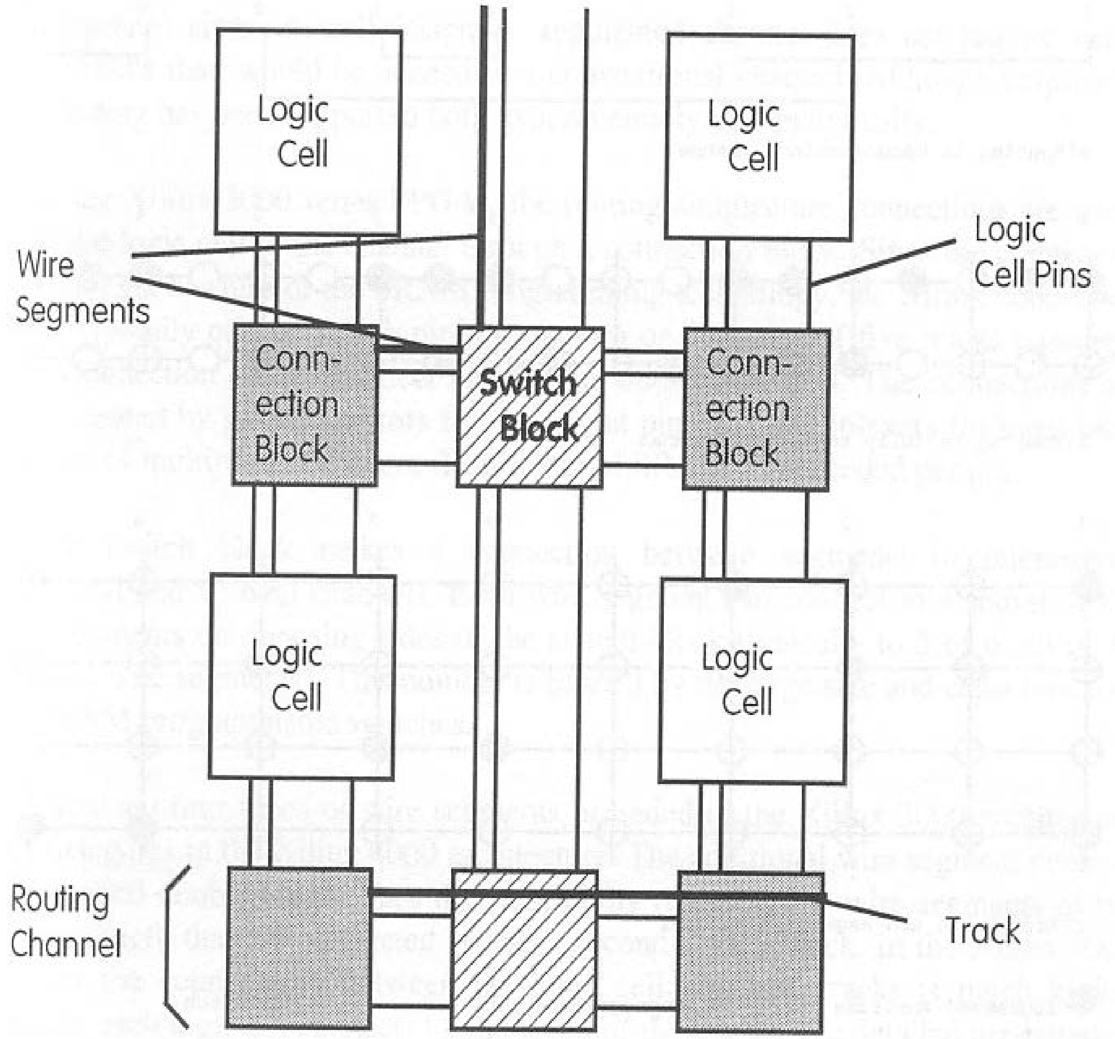




Logic ICs

FPGAs

- Interconnection



Logic ICs

FPGAs

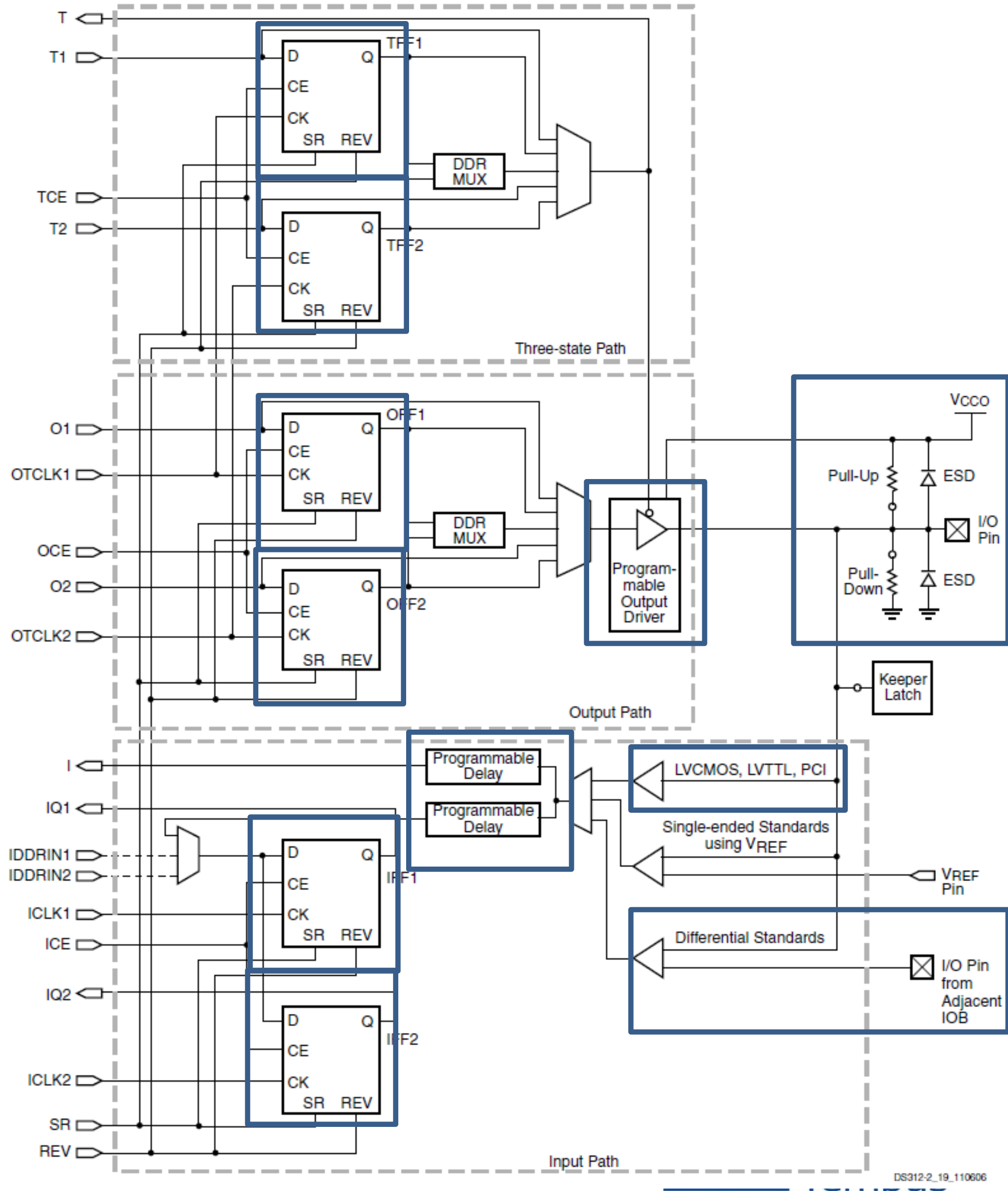
- I/O Block

Input – output conditioning

- IO standards
- Inverting
- Delay
- Pull-up / -down
- Slew rate & drive strength
- Memory elements

  ⇒ Synchronization

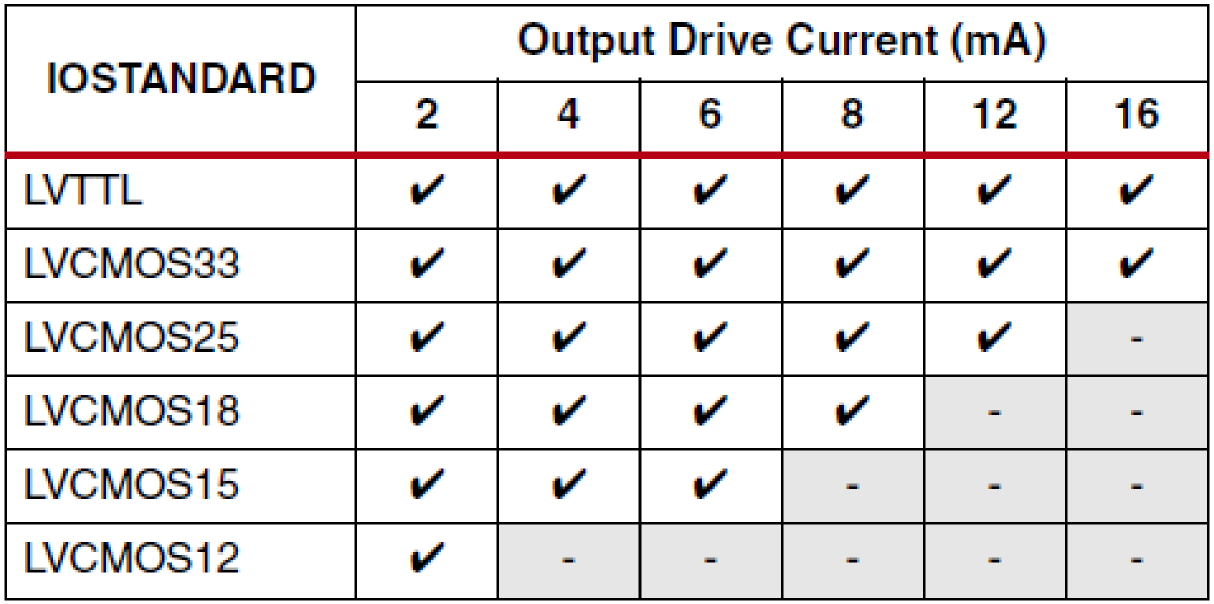
Configuration with SRAM



Logic ICs

FPGAs

- I/O Block

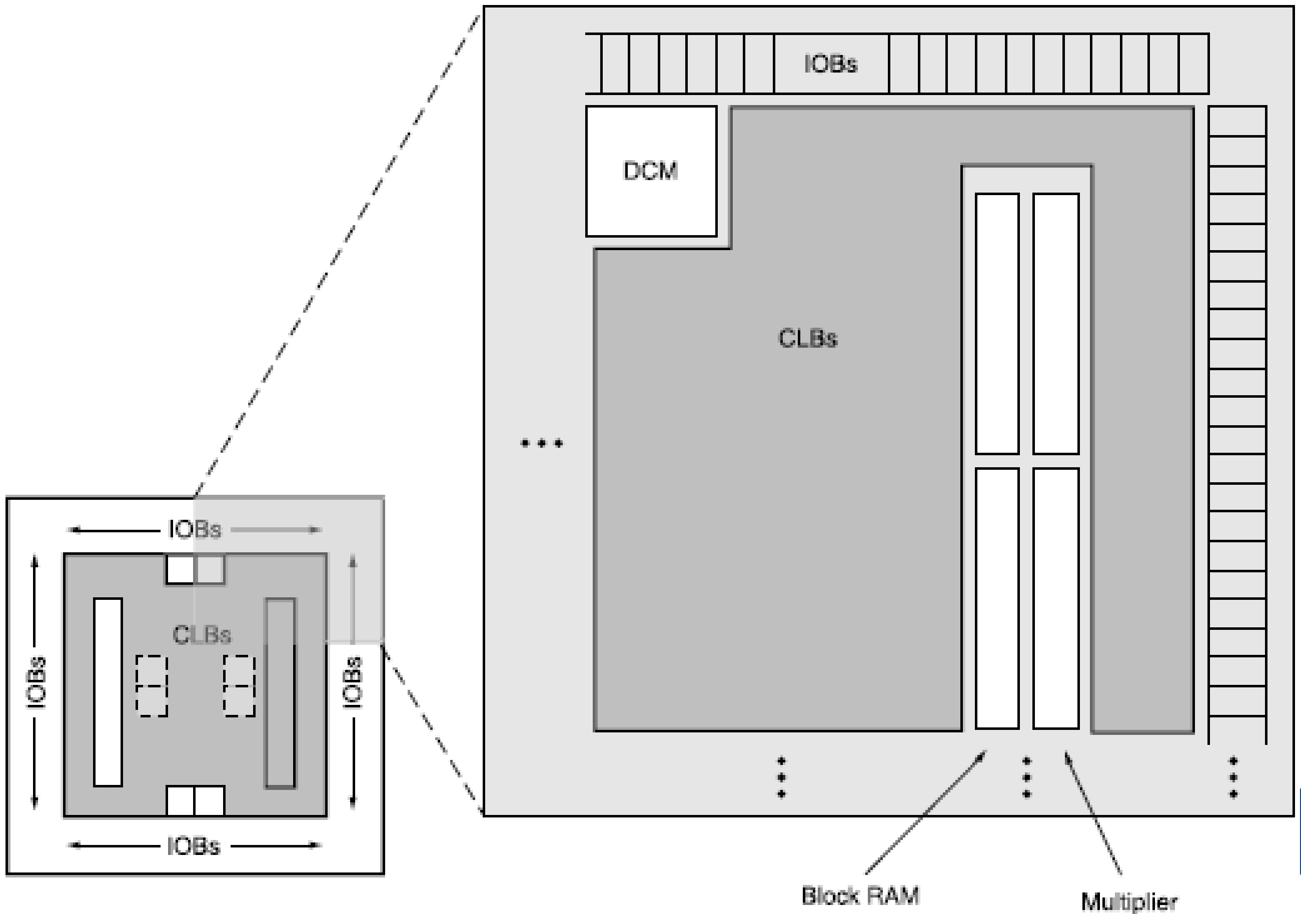


IOSTANDARD	Output Drive Current (mA)					
	2	4	6	8	12	16
LVTTL	✓	✓	✓	✓	✓	✓
LVCMOS33	✓	✓	✓	✓	✓	✓
LVCMOS25	✓	✓	✓	✓	✓	-
LVCMOS18	✓	✓	✓	✓	-	-
LVCMOS15	✓	✓	✓	-	-	-
LVCMOS12	✓	-	-	-	-	-

Logic ICs

FPGAs

- Spartan 3E: architecture => extra hardware



Logic ICs

FPGAs

- Spartan 3E: architecture

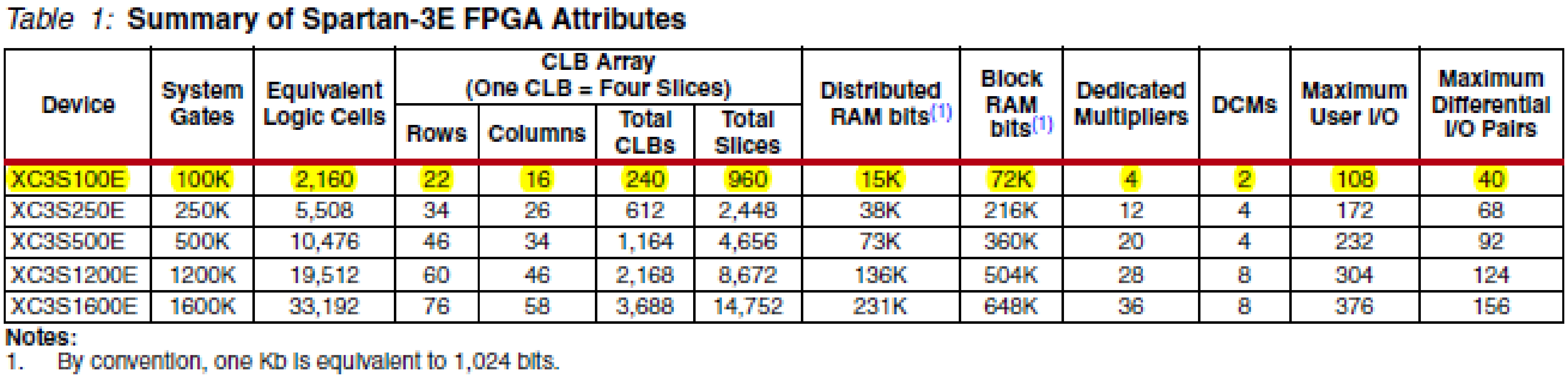


Table 1: **Summary of Spartan-3E FPGA Attributes**

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits[1]	Block RAM bits[1]	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S100E	100K	2,160	22	16	240	960	15K	72K	4	2	108	40
XC3S250E	250K	5,508	34	26	612	2,448	38K	216K	12	4	172	68
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232	92
XC3S1200E	1200K	19,512	60	46	2,168	8,672	136K	504K	28	8	304	124
XC3S1600E	1600K	33,192	76	58	3,688	14,752	231K	648K	36	8	376	156

Notes:
1. By convention, one Kb is equivalent to 1,024 bits.

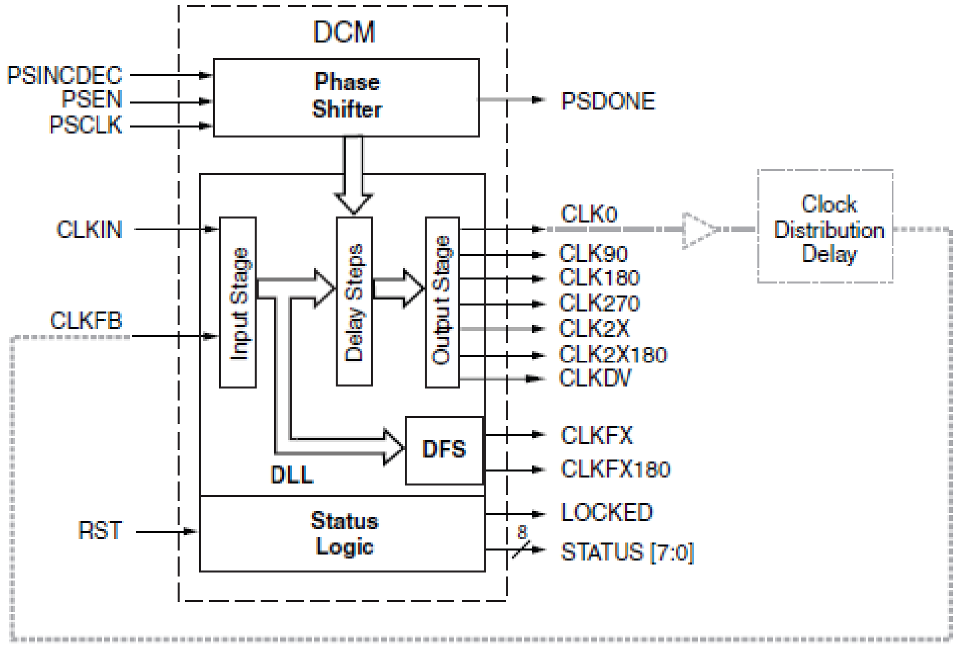
Logic ICs

FPGAs

- Spartan 3E: DCM

  Clock
  - frequency,
  - phase shift
  - skew

  employs a Delay Locked Loop



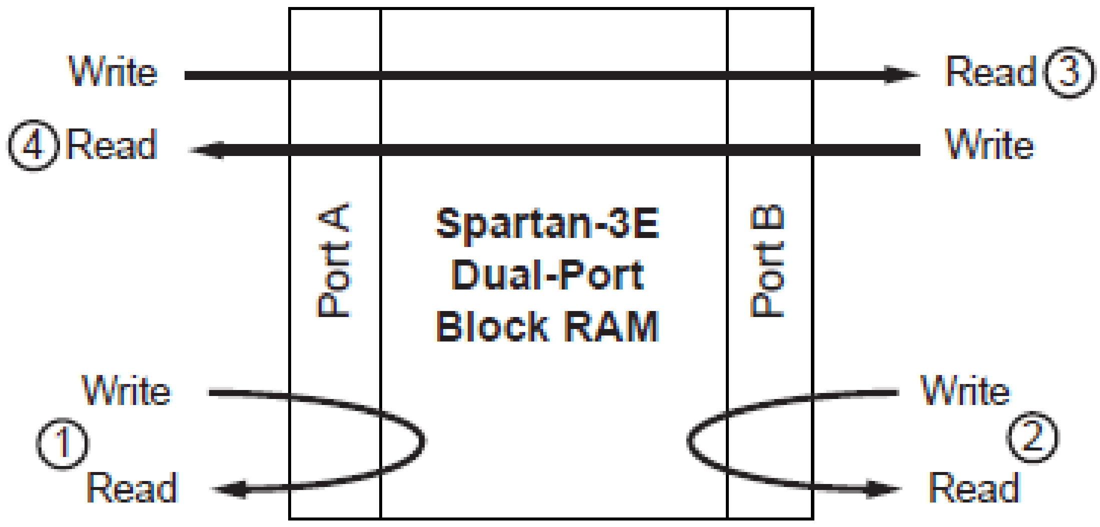
Logic ICs

FPGAs

- Spartan 3E: Block RAM

The Internal Structure of the Block RAM

- dual port structure, called A and B
- independent access to the common block RAM
- maximum capacity of 18,432 bits
- dedicated set of data, control, and clock lines

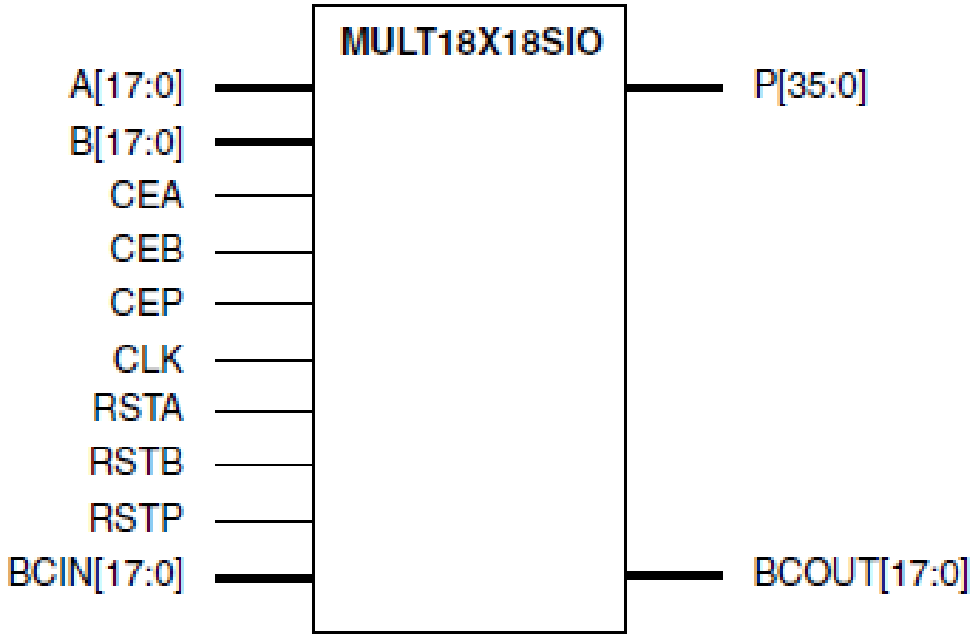


Logic ICs

FPGAs

- Spartan 3E: Dedicated multipliers

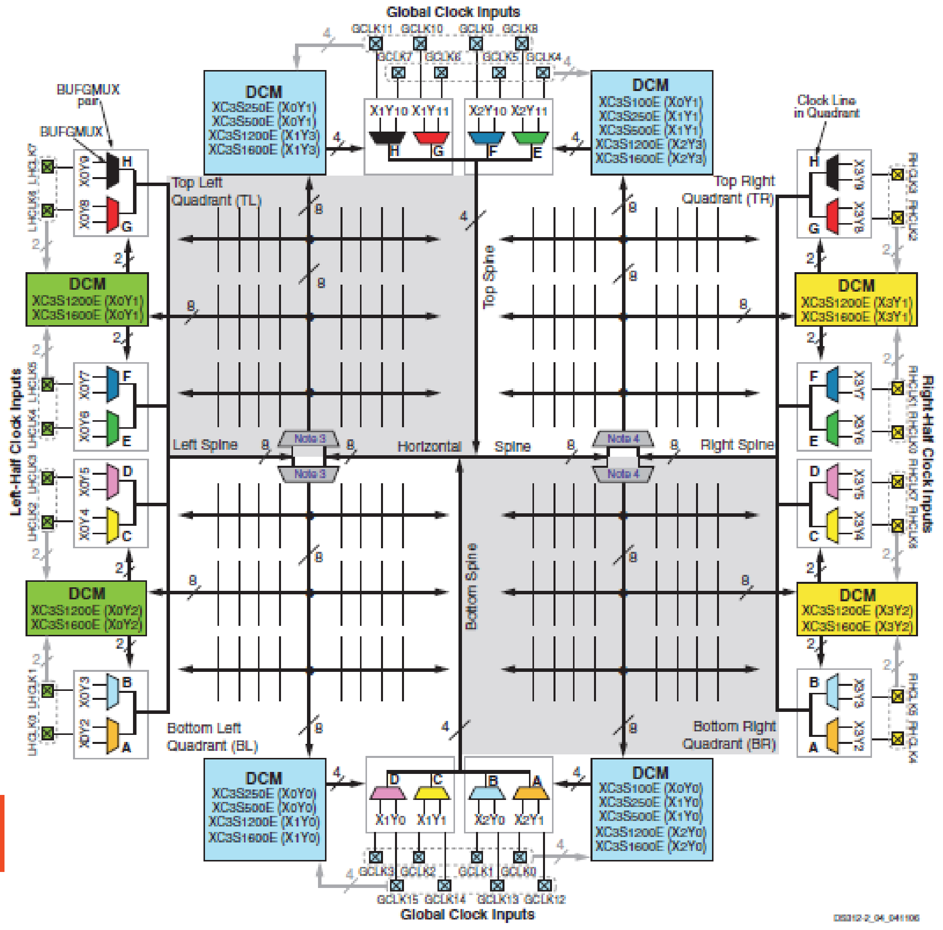
principle operation P = A × B
=> DSP operations



Logic ICs

FPGAs

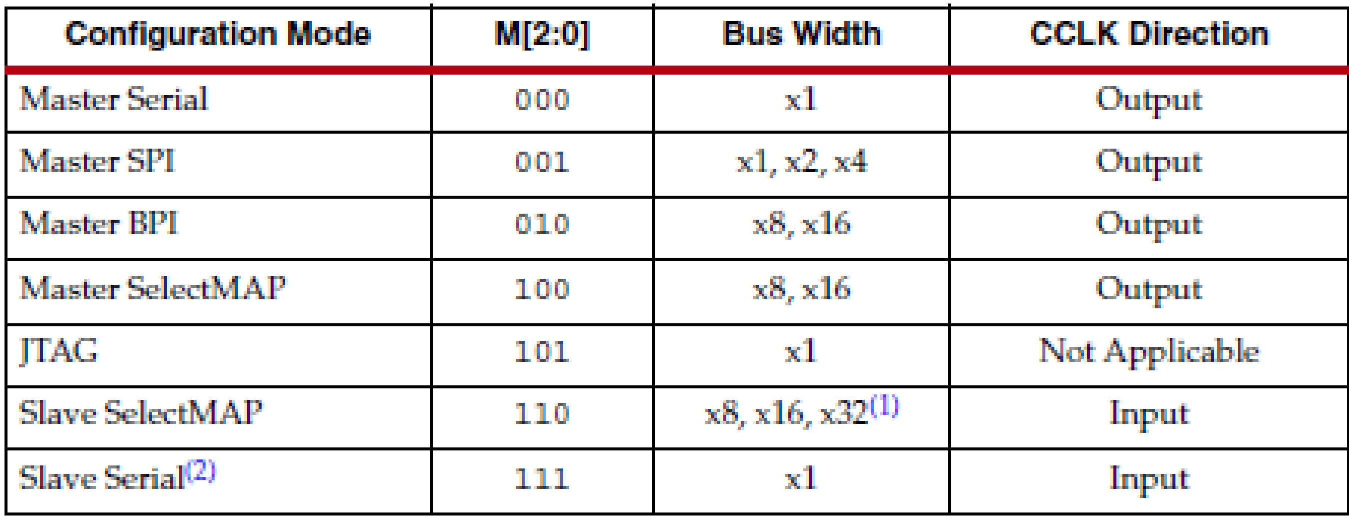
- Spartan 3E: Clock Distribution Network



Logic ICs

FPGAs

- Spartan 3E: Configuration
  - Bits on dedicated places are needed (in SRAM)
  - Where does the bit-file come from?
  - Basys3: 3 options
    - Master SPI: Flash
    - Slave serial on USB stick
    - JTAG: Through USB



Configuration Mode	M[2:0]	Bus Width	CCLK Direction
Master Serial	000	x1	Output
Master SPI	001	x1, x2, x4	Output
Master BPI	010	x8, x16	Output
Master SelectMAP	100	x8, x16	Output
JTAG	101	x1	Not Applicable
Slave SelectMAP	110	x8, x16, x32(1)	Input
Slave Serial(2)	111	x1	Input

VHDL

Spartan 3E: XC3S<u>100</u>E

<u>100K</u> gates!?

How in earth can you address them?

VHDL!!!

VHDL

VHDL

- HDL = Hardware <u>Description</u> Language
- V = VHSIC = Very High Speed Integrated Circuit
  - Language used to describe hardware (not for programming)
  - Result = hardware
    - IC
    - FPGA

VHDL

Structure of VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

begin

led <= sw;

end Behavioral;
```

library and use

library IEEE;

use IEEE.std_logic_1164.all;

Define types and operations

VHDL

Structure of VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

begin

led <= sw;

end Behavioral;
```

Entity declaration

Entity <ent_name> is

Port ();

end <ent_name>;

- Interface description of logic component
- Port signals:
    - Signal name
    - signal direction
    - data types
    - signal width

VHDL

Structure of VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

begin

led <= sw;

end Behavioral;
```

Architecture

Architecture <arch_name> of <entity_name> is

begin

end <arch_name>;

- Implementation of the design

VHDL

Structure of VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

begin

led <= sw;

end Behavioral;
```

Architecture

Architecture <arch_name> of <entity_name> is

begin

end <arch_name>;

- Declarative part:
  - additional signals
  - components
  - …

VHDL

Structure of VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
   Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
          led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

begin

led <= sw;

end Behavioral;
```

Architecture

Architecture <arch_name> of <entity_name> is

begin

end <arch_name>;

- Definition part:
    - signal assignments
    - processes
    - component instantiations
    - …

Lab: startup

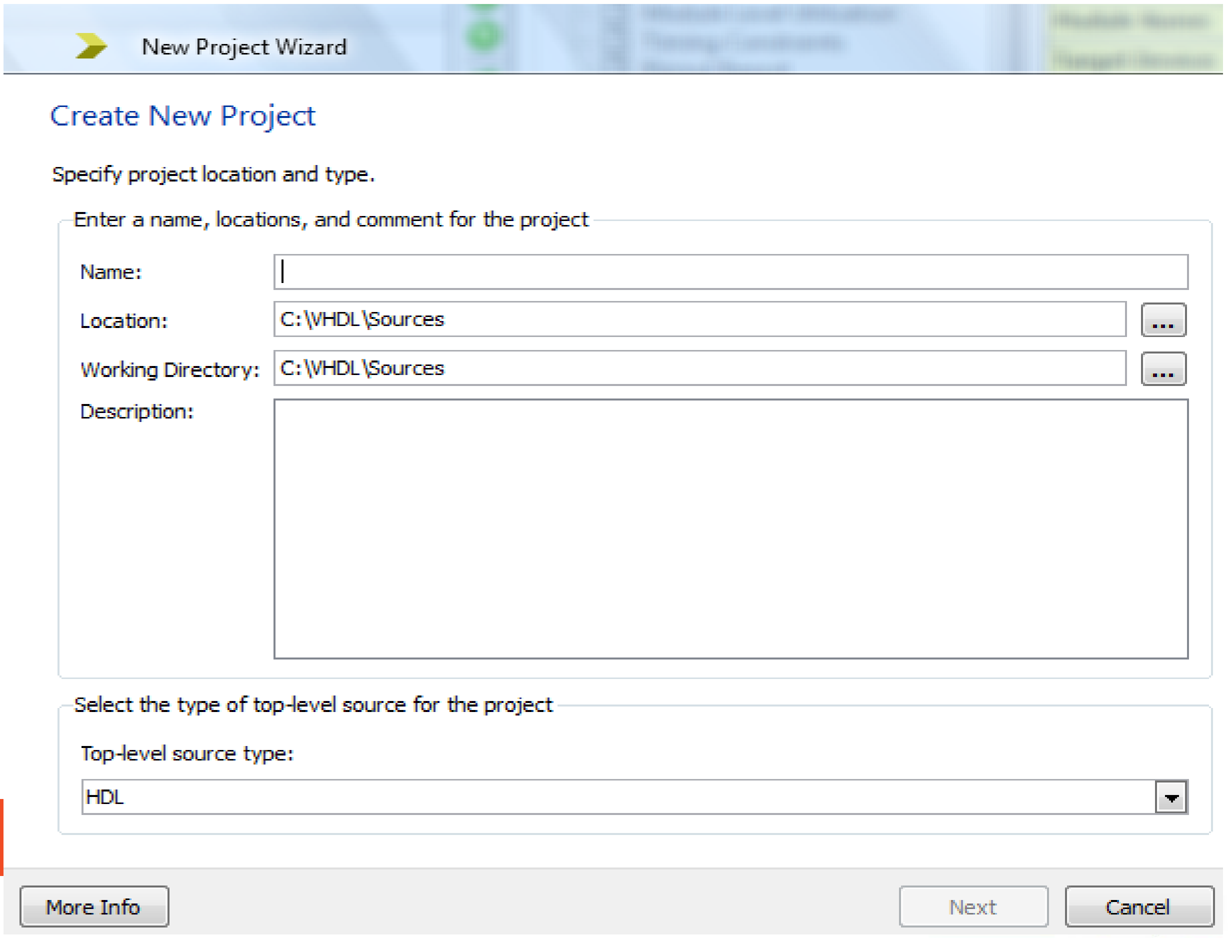
Project set up (Basys2 & ISE)

- Start ISE software: 32 bit Project Navigator

Project set up

- **File => New Project**

Choose meaningful Directory (Name/Project_name)



Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

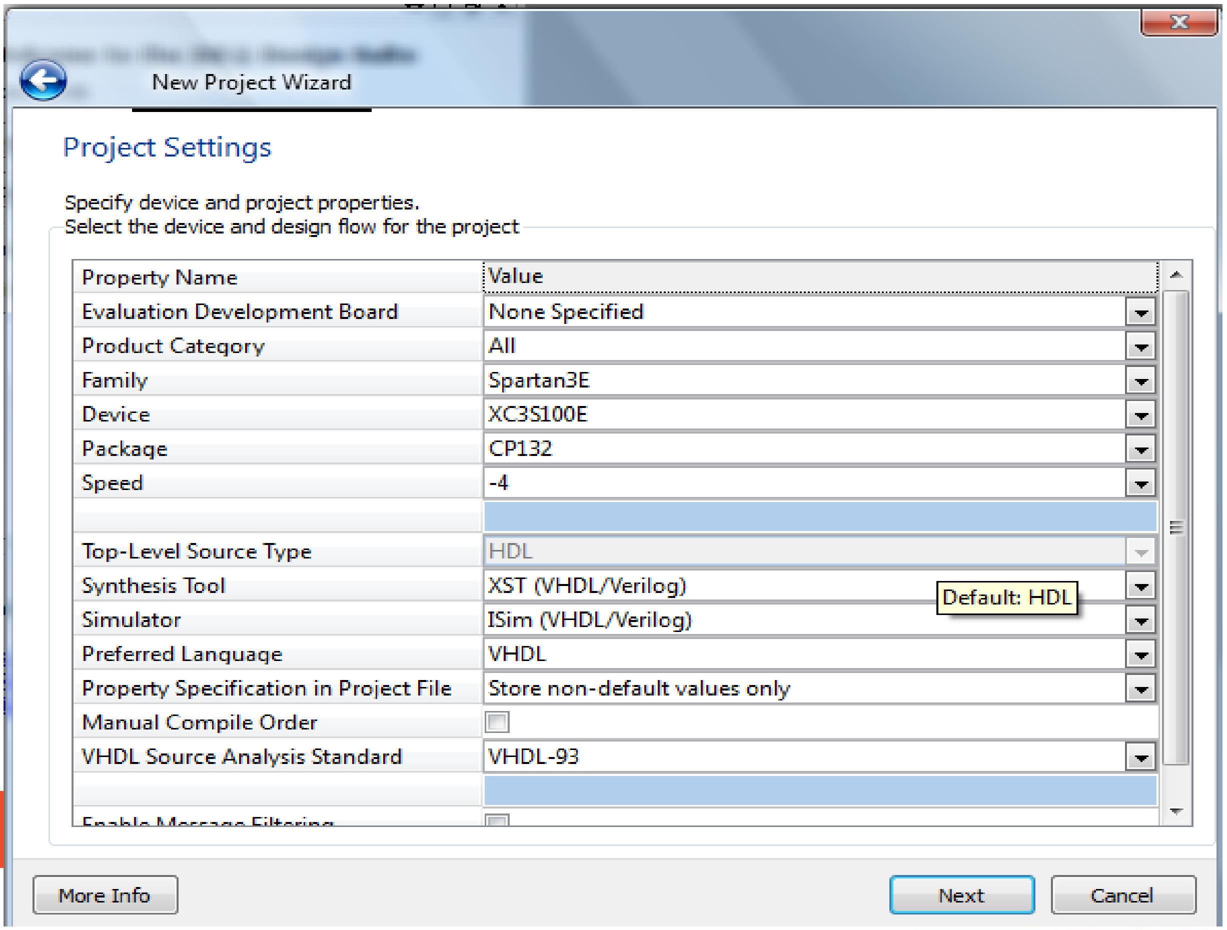
Name:	
Location:	C:\VHDL\Sources
Working Directory:	C:\VHDL\Sources
Description:	

Select the type of top-level source for the project

Top-level source type: HDL

Project set up

- Project Settings (information is available on the device)



Project Settings

Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3E
Device	XC3S100E
Package	CP132
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store non-default values only
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

Default: HDL

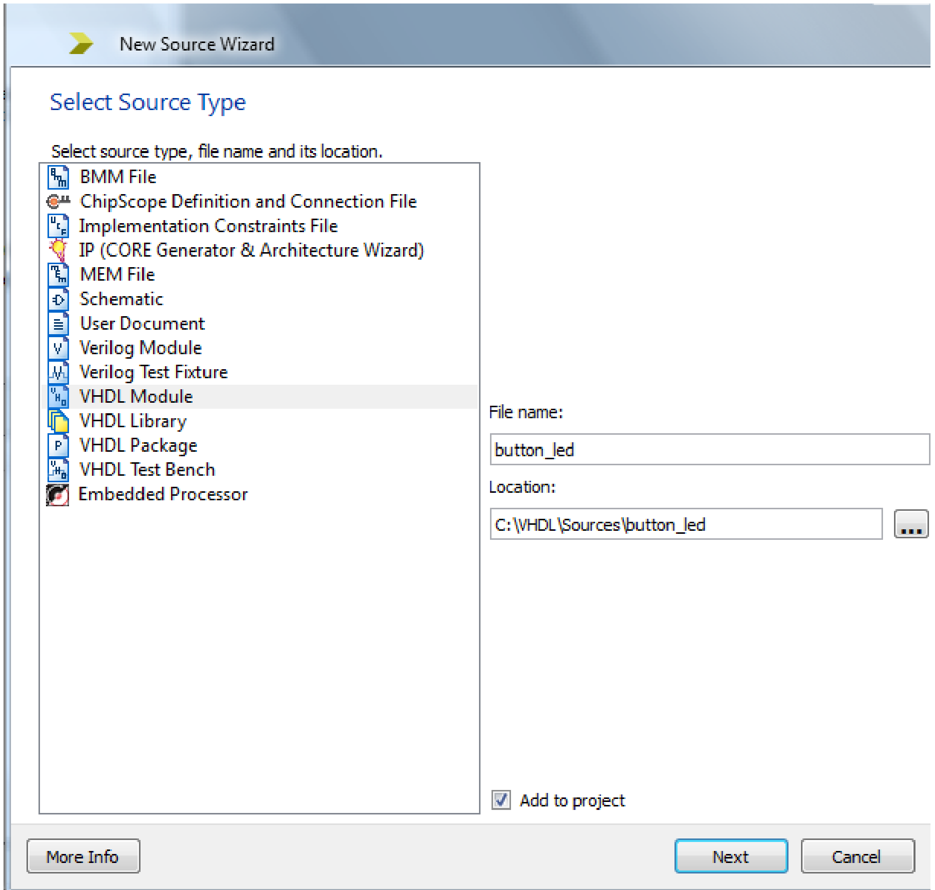
Project set up

- Project Settings (information is available on the device)

Family	Spartan3E
Device	XC3S100E
Package	CP132
Speed	-4
Synthesis Tool	XST
Simulator	ISim

Project set up

- Right click on **project => new source => VHDL Module** & give meaningful name


New Source Wizard

Select Source Type

Select source type, file name and its location.

- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

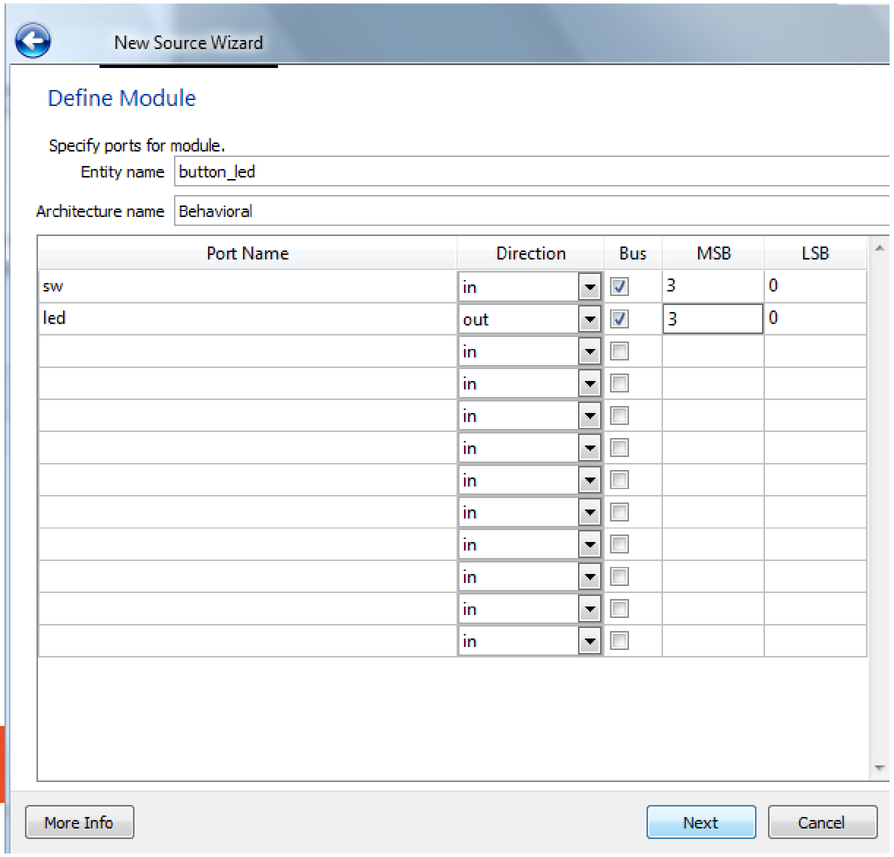
File name: button_led

Location: C:\VHDL\Sources\button_led

☑ Add to project

Project set up

- Define inputs/outputs, individual ports or busses, input, output or both



Define Module

Specify ports for module.

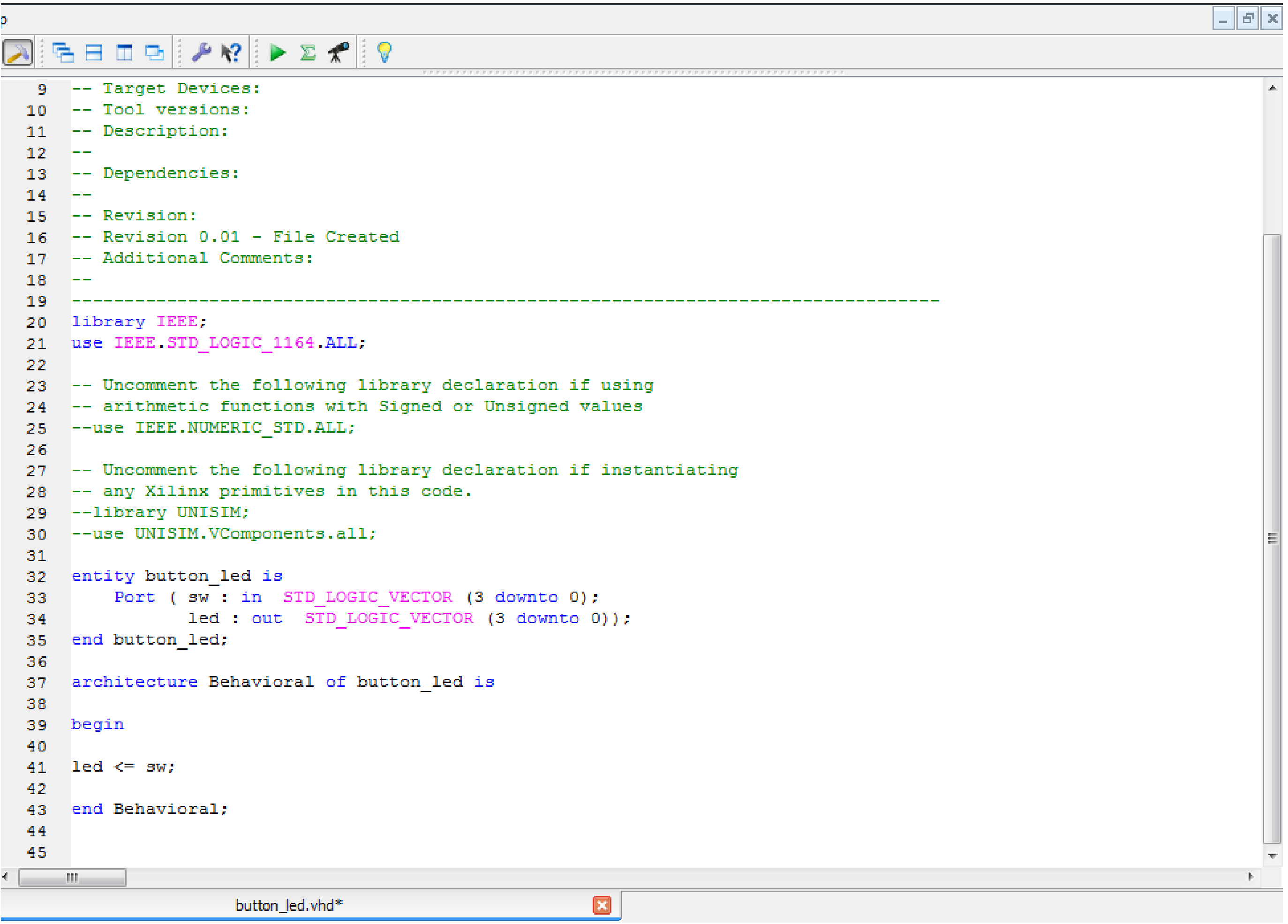
Entity name: button_led

Architecture name: Behavioral

Port Name	Direction	Bus	MSB	LSB
sw	in	✓	3	0
led	out	✓	3	0
	in			
	in			
	in			
	in			
	in			
	in			
	in			
	in			
	in			
	in			

Project set up

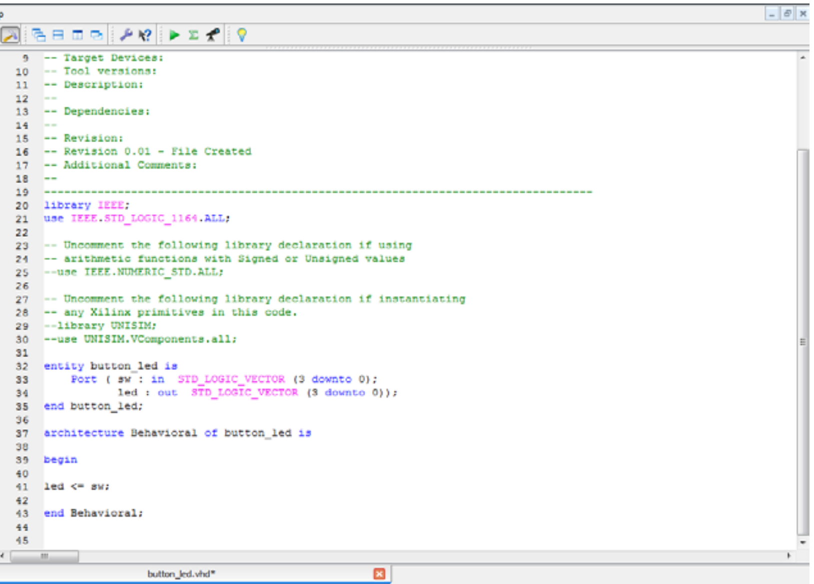
- Write code in editor



```
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

begin

led <= sw;

end Behavioral;
```

Project set up

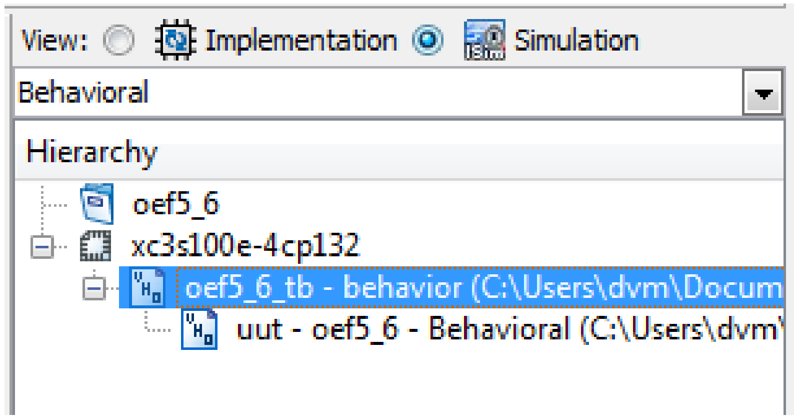
- Write code in editor
  - o This will describe the internal behavior of the logic component:
    - – what does it do?
    - – what is the interface?



```vhdl
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

begin

led <= sw;

end Behavioral;
```

Project set up

- Simulate the design with a testbench
- Click on simulation in the left upper panel:


View: Implementation [✓ Simulation]

Behavioral

Hierarchy

oef5_6
xc3s100e-4cp132
oef5_6_tb - behavior (C:\Users\dvm\Docum
uut - oef5_6 - Behavioral (C:\Users\dvm\

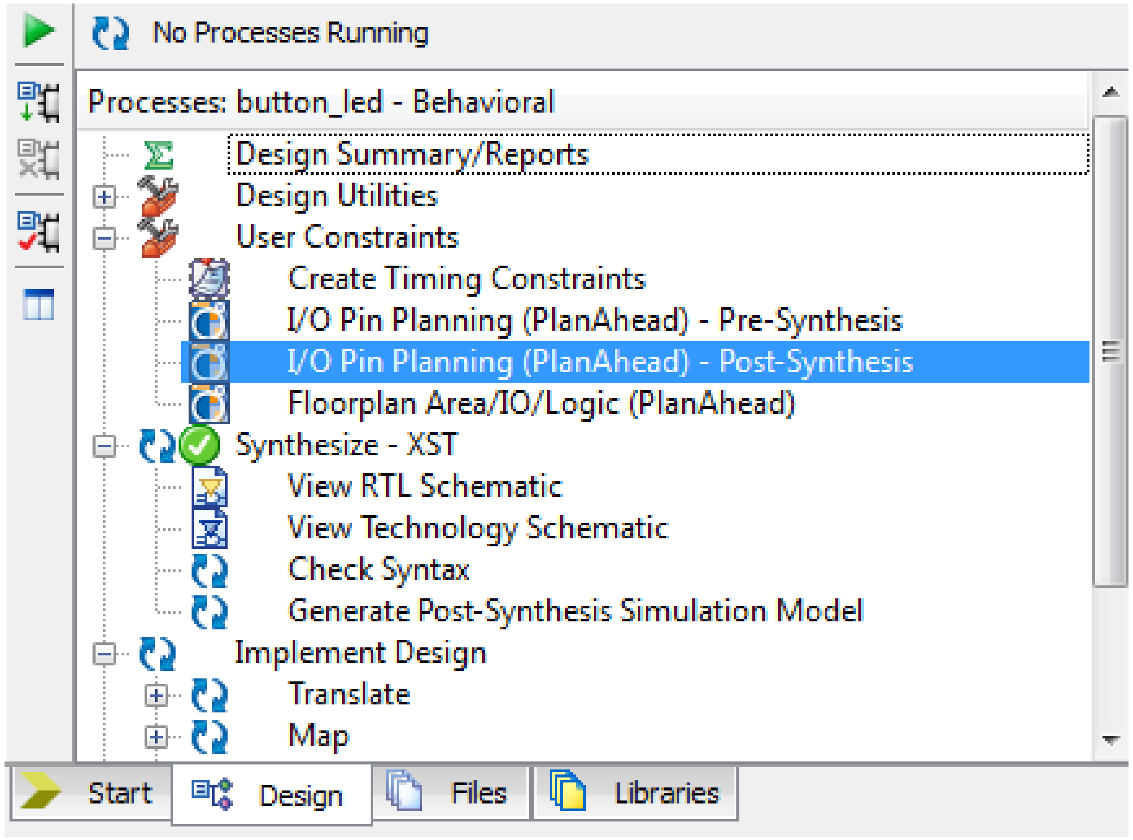
- Write a testbench to
  - simulate the clock
  - simulate all possible input combinations

Project set up

- Add user constraints, a UCF-file.
  - To which FPGA pins are the ports from the entity connected?
  - How are the ports connected to the outside world?
  - What is the I / O standerd?
  - How is the I / O block configured?

Project set up

- Add user constraints, a UCF-file.
  - push “I/O Pin Planning”



No Processes Running

Processes: button_led - Behavioral

- Design Summary/Reports
- Design Utilities
- User Constraints
  - Create Timing Constraints
  - I/O Pin Planning (PlanAhead) - Pre-Synthesis
  - I/O Pin Planning (PlanAhead) - Post-Synthesis
  - Floorplan Area/IO/Logic (PlanAhead)
- Synthesize - XST
  - View RTL Schematic
  - View Technology Schematic
  - Check Syntax
  - Generate Post-Synthesis Simulation Model
- Implement Design
  - Translate
  - Map

Project set up

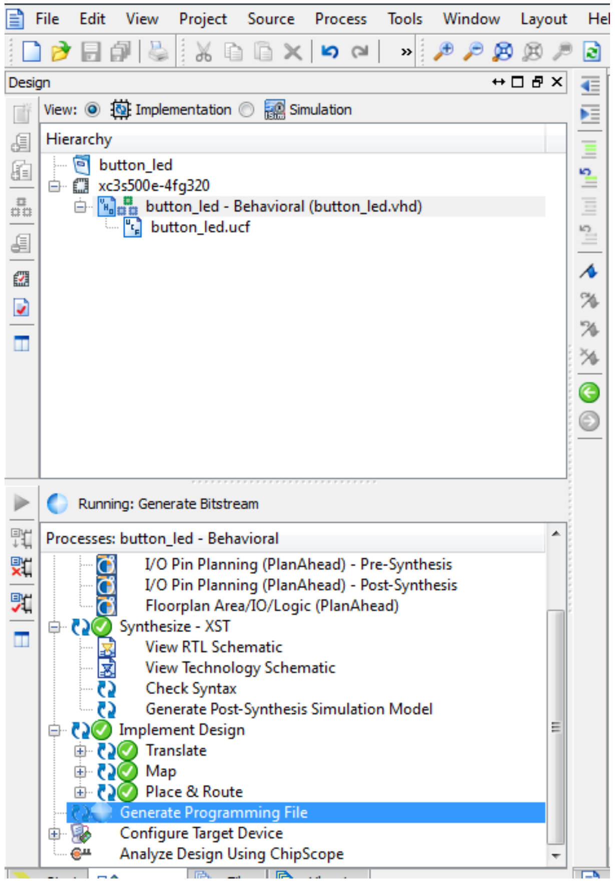
- Push “Edit Constraints” on the processes panel, left.
- Copy all suitable pin locations from the UCF-file

Project set up

- An example is seen below:
  - NET "sw<7>" LOC = "N3"; # Bank = 2, Signal name = SW7
  - NET "sw<6>" LOC = "E2"; # Bank = 3, Signal name = SW6
  - NET "sw<5>" LOC = "F3"; # Bank = 3, Signal name = SW5
  - NET "sw<4>" LOC = "G3"; # Bank = 3, Signal name = SW4
  - NET "sw<3>" LOC = "B4"; # Bank = 3, Signal name = SW3
  - NET "sw<2>" LOC = "K3"; # Bank = 3, Signal name = SW2
  - NET "sw<1>" LOC = "L3"; # Bank = 3, Signal name = SW1
  - NET "sw<0>" LOC = "P11"; # Bank = 2, Signal name = SW0
  - NET "btn" LOC = "G12"; # Bank = 0, Signal name = BTN0
  - NET "Led<3>" LOC = "P6" ; # Bank = 2, Signal name = LD3
  - NET "Led<2>" LOC = "P7" ; # Bank = 3, Signal name = LD2
  - NET "Led<1>" LOC = "M11" ; # Bank = 2, Signal name = LD1
  - NET "Led<0>" LOC = "M5" ; # Bank = 2, Signal name = LD0

Project set up

- **Generate Programming File** runs through: **synthesize => implement design**



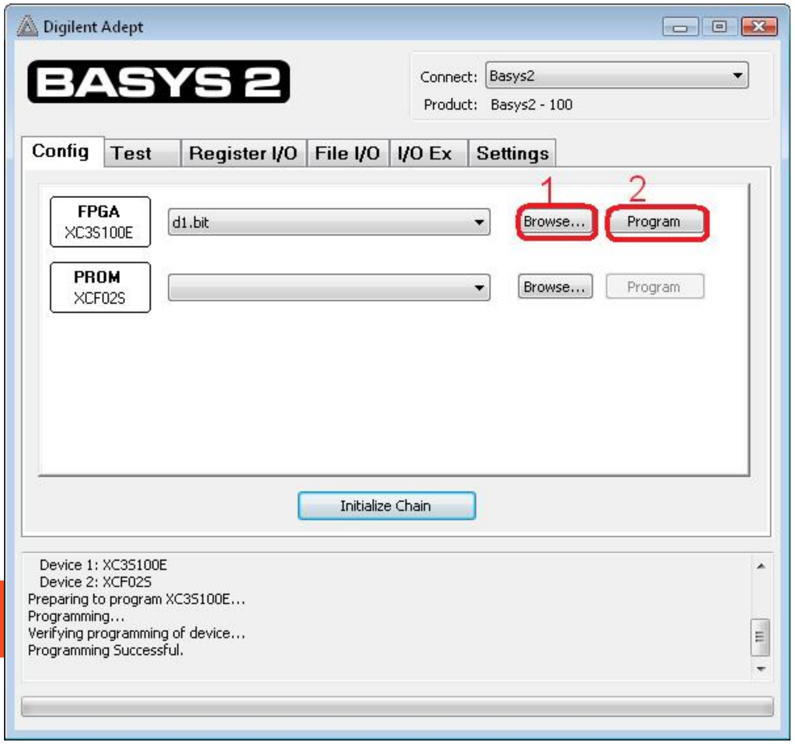
button_led
xc3s500e-4fg320
button_led - Behavioral (button_led.vhd)
button_led.ucf

Running: Generate Bitstream

Processes: button_led - Behavioral
I/O Pin Planning (PlanAhead) - Pre-Synthesis
I/O Pin Planning (PlanAhead) - Post-Synthesis
Floorplan Area/IO/Logic (PlanAhead)
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulation Model
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope

Project set up

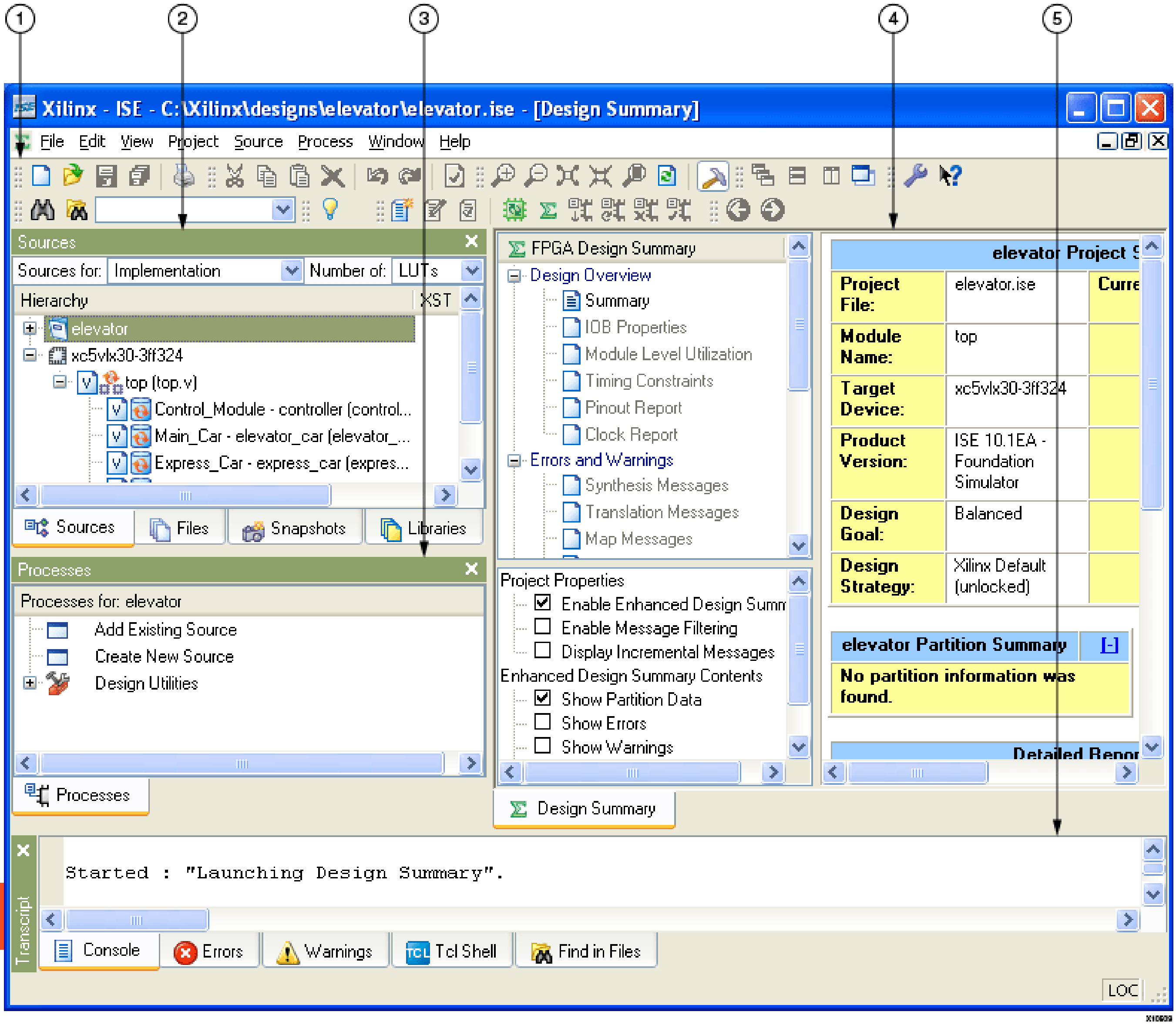
- To implement the design, open Digilent's Adept software
- Browse bit-file and then push program



```
Device 1: XC3S100E
Device 2: XCF02S
Preparing to program XC3S100E...
Programming...
Verifying programming of device...
Programming Successful.
```

ISE software

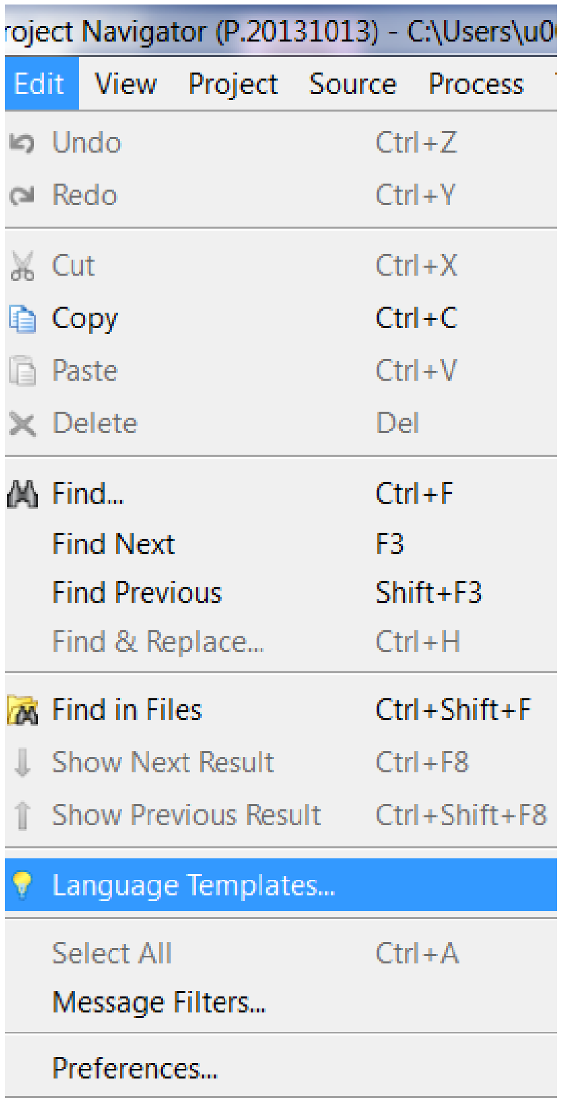
1. Toolbar
2. Sources window
3. Processes window
4. Workspace
5. Transcript window



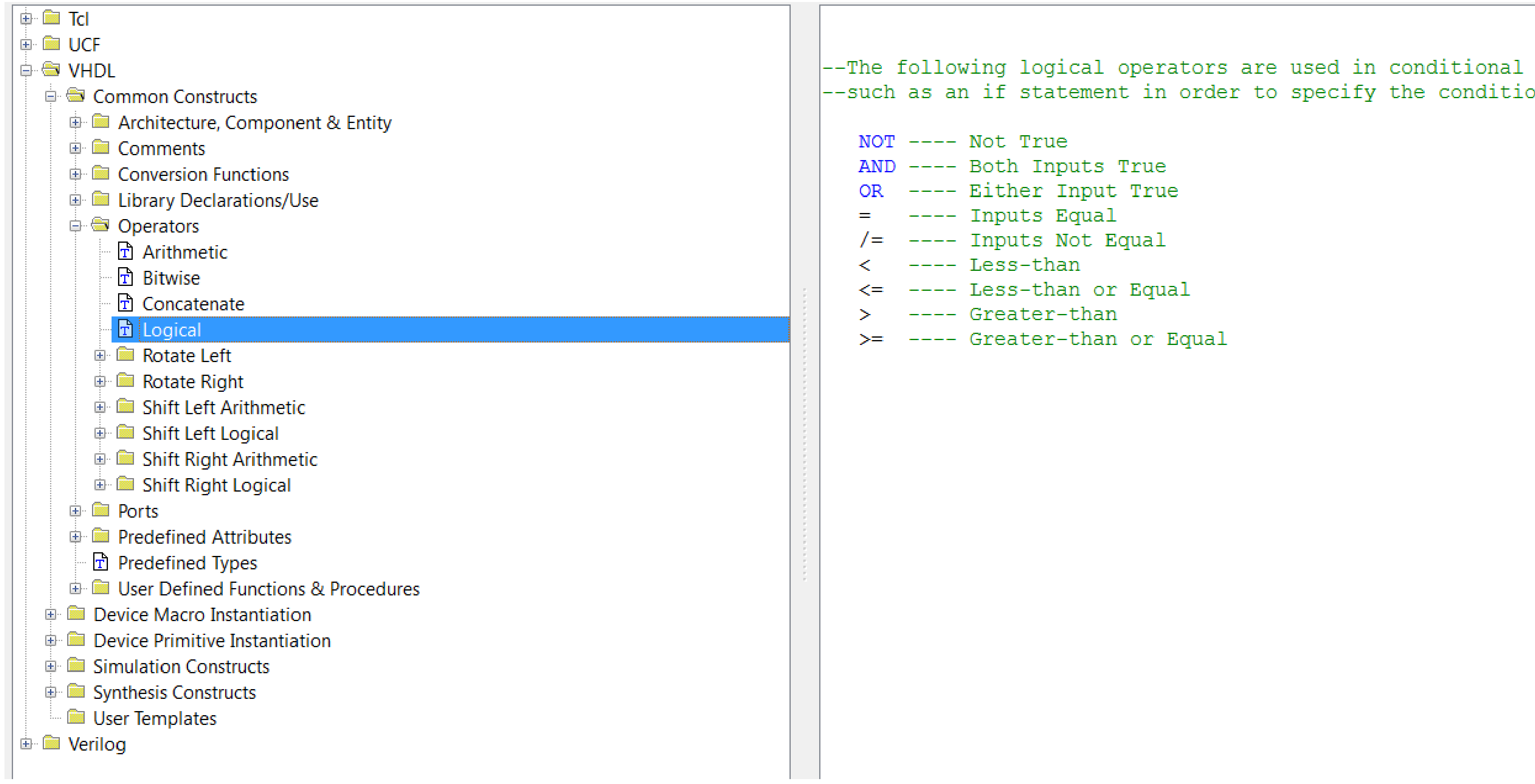
```
Started : "Launching Design Summary".
```

ISE software

- Language templates
  - Useful tool to find coding examples



ISE software



- Tcl
- UCF
- VHDL
  - Common Constructs
    - Architecture, Component & Entity
    - Comments
    - Conversion Functions
    - Library Declarations/Use
    - Operators
      - Arithmetic
      - Bitwise
      - Concatenate
      - Logical
      - Rotate Left
      - Rotate Right
      - Shift Left Arithmetic
      - Shift Left Logical
      - Shift Right Arithmetic
      - Shift Right Logical
    - Ports
    - Predefined Attributes
    - Predefined Types
    - User Defined Functions & Procedures
  - Device Macro Instantiation
  - Device Primitive Instantiation
  - Simulation Constructs
  - Synthesis Constructs
  - User Templates
- Verilog

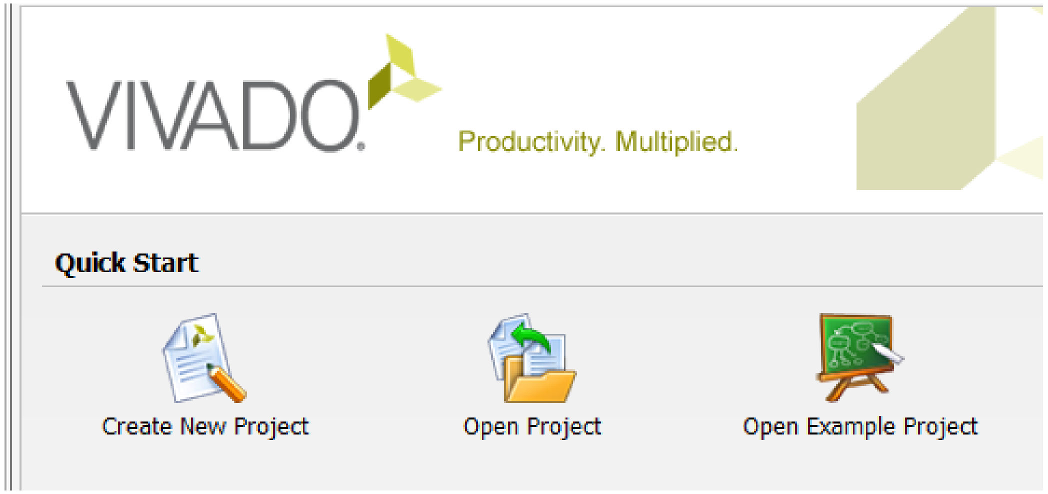
```
--The following logical operators are used in conditional
--such as an if statement in order to specify the conditio

   NOT ---- Not True
   AND ---- Both Inputs True
   OR  ---- Either Input True
   =   ---- Inputs Equal
   /=  ---- Inputs Not Equal
   <   ---- Less-than
   <=  ---- Less-than or Equal
   >   ---- Greater-than
   >=  ---- Greater-than or Equal
```

Project set up (Basys3 & Vivado)

- Start Vivado software

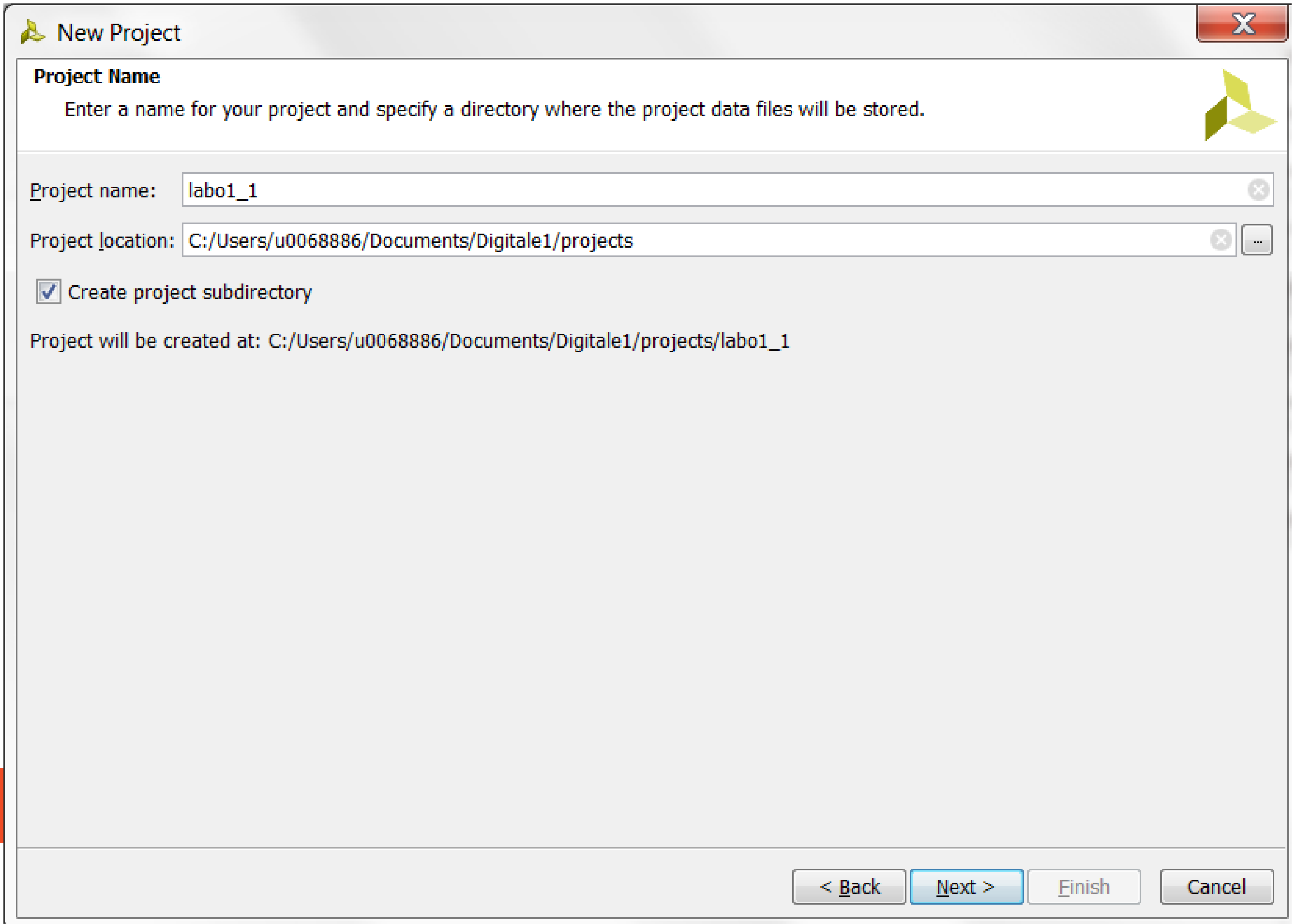
Project set up

- Start a new project



Project set up

- Choose meaningful name & directory



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: labo1_1

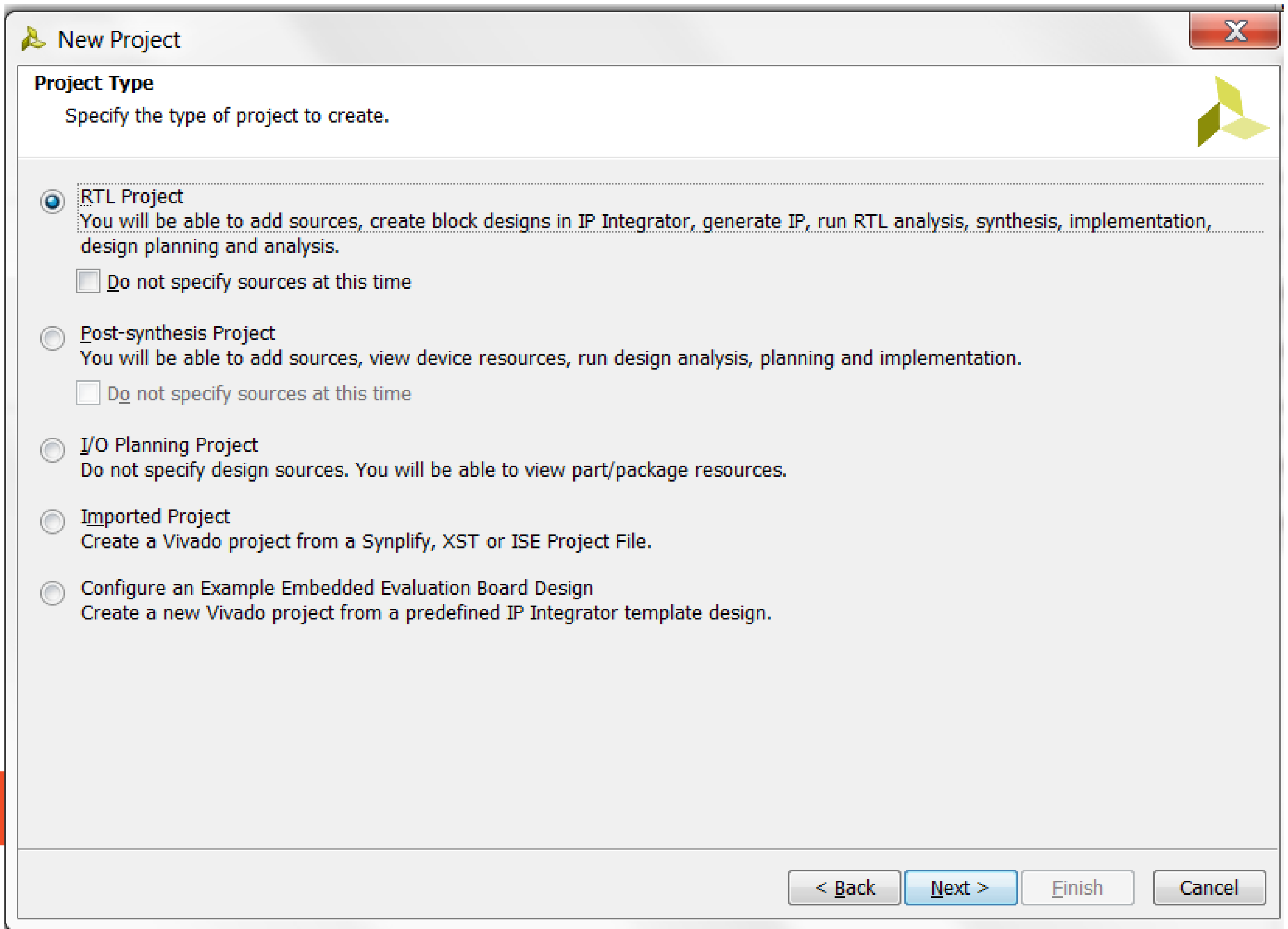
Project location: C:/Users/u0068886/Documents/Digitale1/projects

[✓] Create project subdirectory

Project will be created at: C:/Users/u0068886/Documents/Digitale1/projects/labo1_1

Project set up

- Build a RTL-project



New Project

Project Type

Specify the type of project to create.

(●) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time

( ) Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

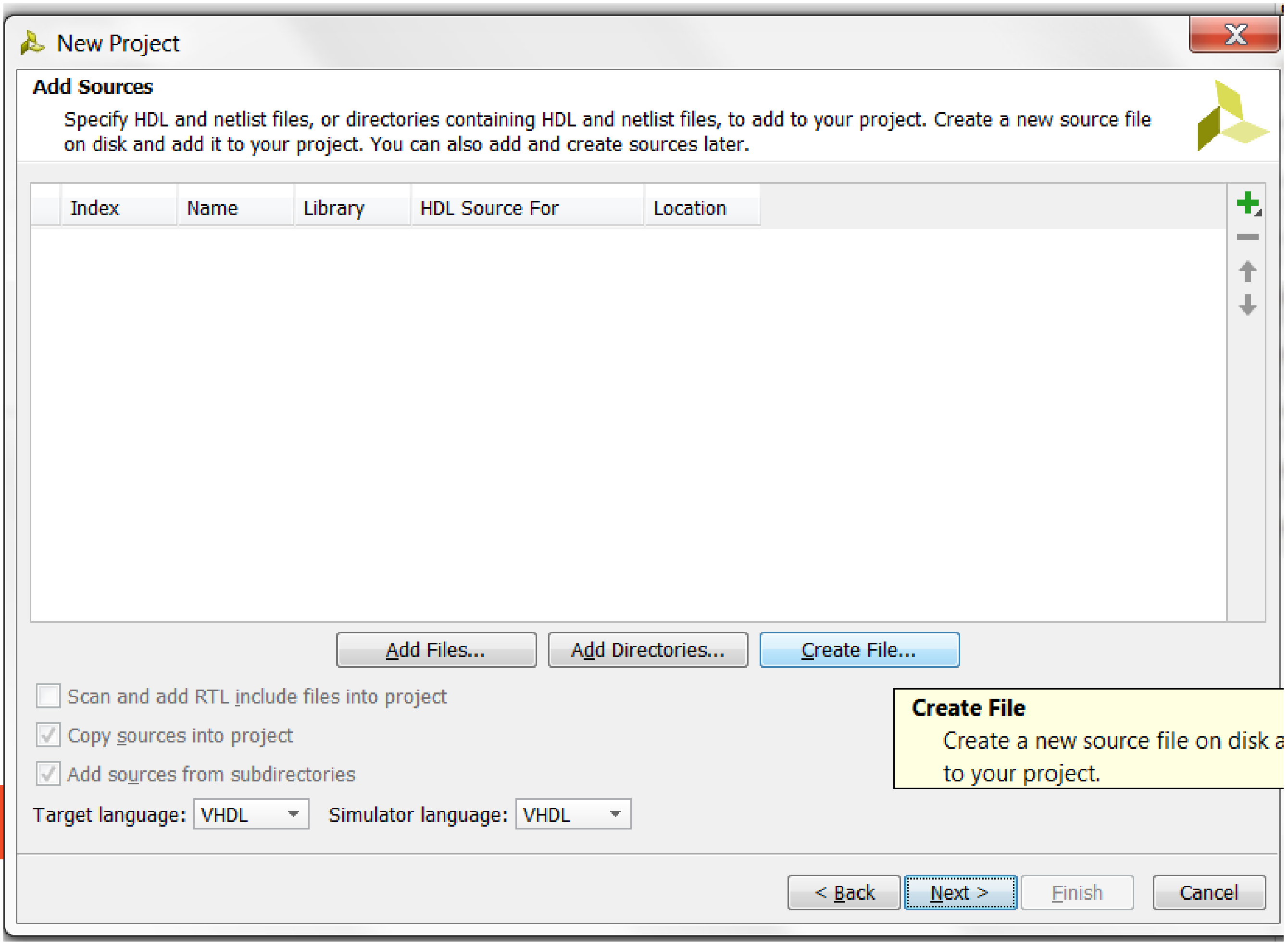
( ) I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

( ) Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

( ) Configure an Example Embedded Evaluation Board Design
Create a new Vivado project from a predefined IP Integrator template design.

Project set up

- Create new sources



New Project

Add Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

	Index	Name	Library	HDL Source For	Location

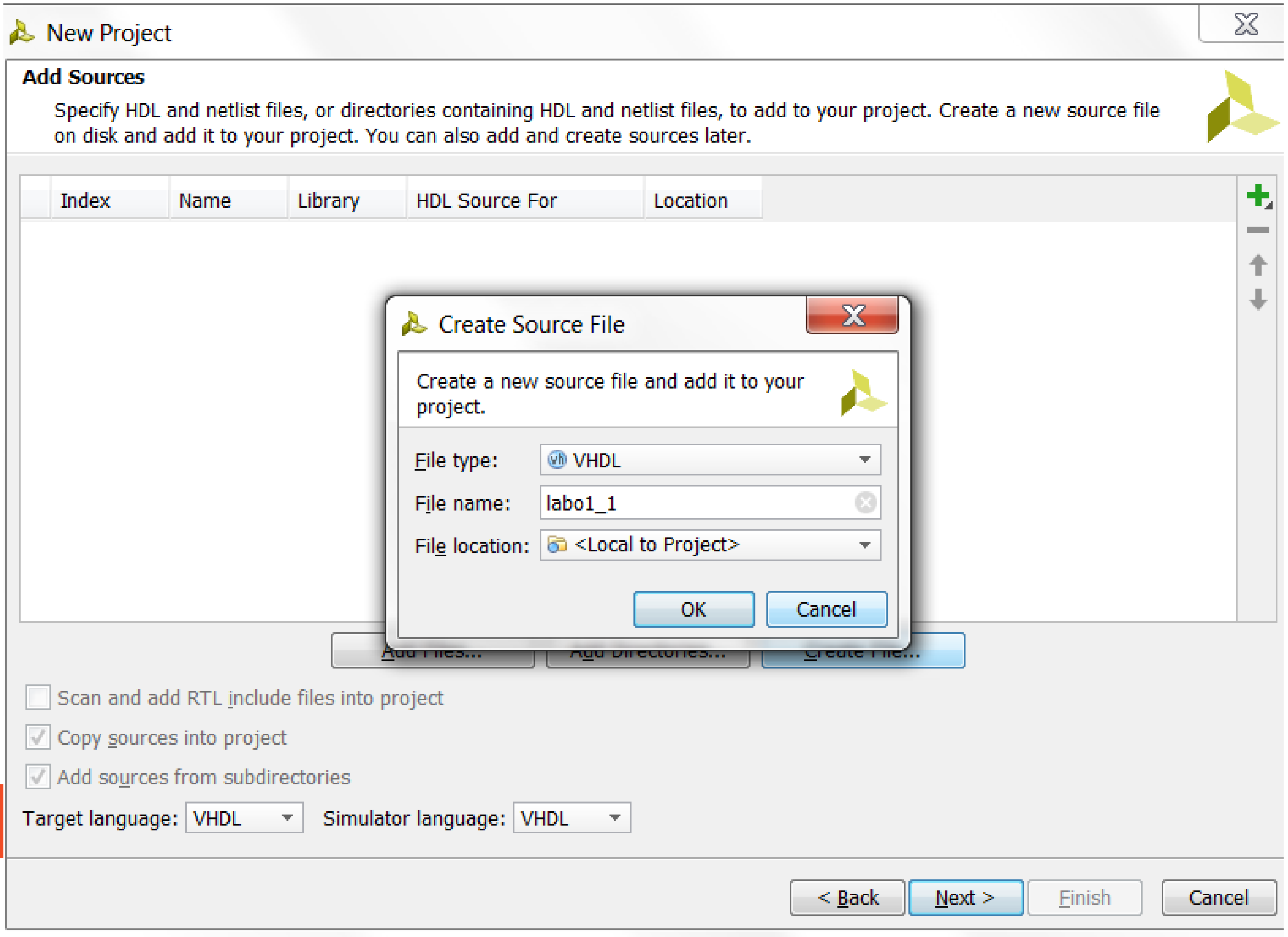
☐ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

Target language: VHDL
Simulator language: VHDL

Create File
Create a new source file on disk a
to your project.

Project set up

- Create new sources



New Project

Add Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Index	Name	Library	HDL Source For	Location

Create Source File

Create a new source file and add it to your project.

File type: VHDL

File name: labo1_1

File location: <Local to Project>

☐ Scan and add RTL include files into project

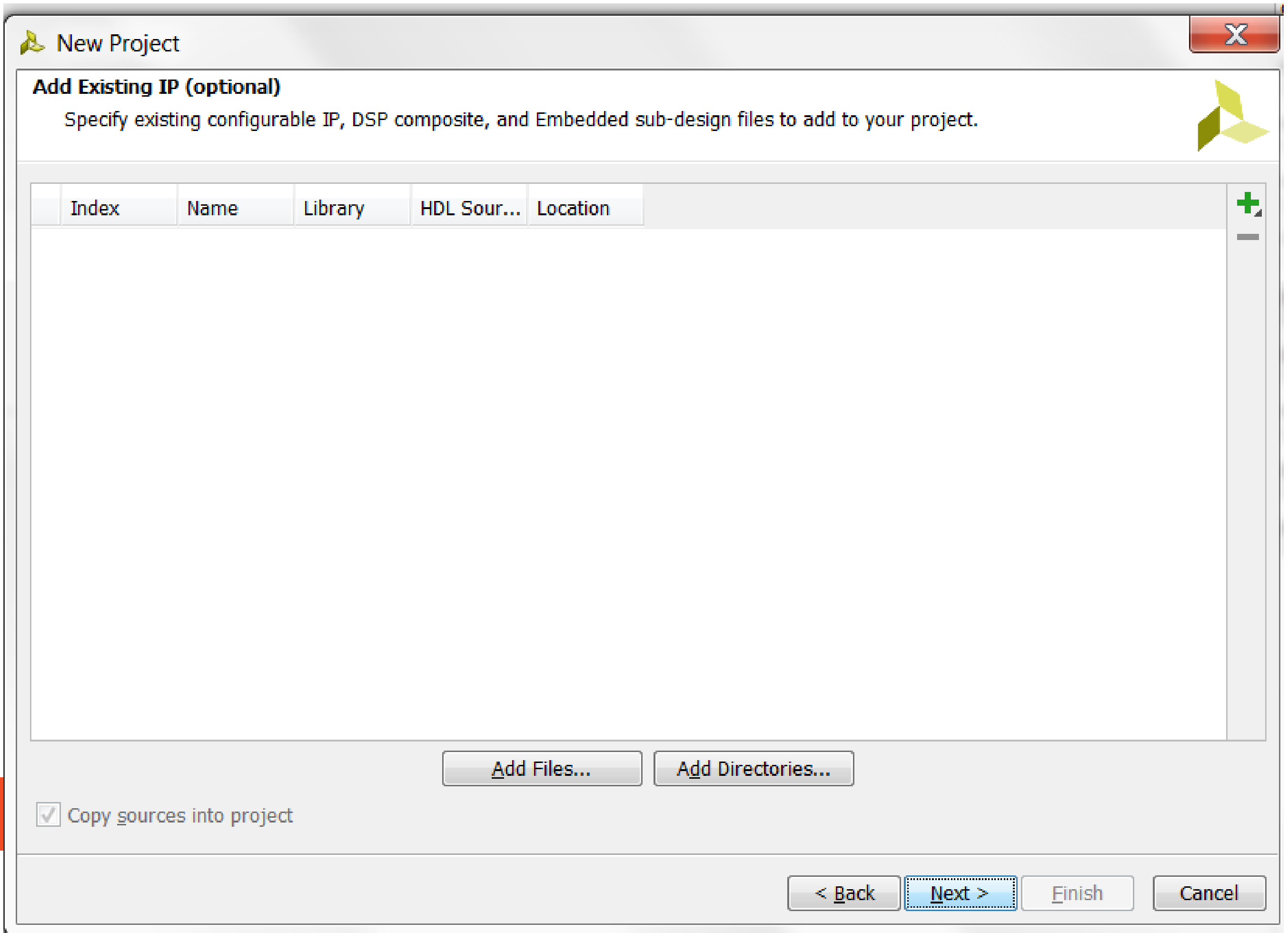
☑ Copy sources into project

☑ Add sources from subdirectories

Target language: VHDL Simulator language: VHDL

Project set up

- Add IP (if needed)



New Project

Add Existing IP (optional)

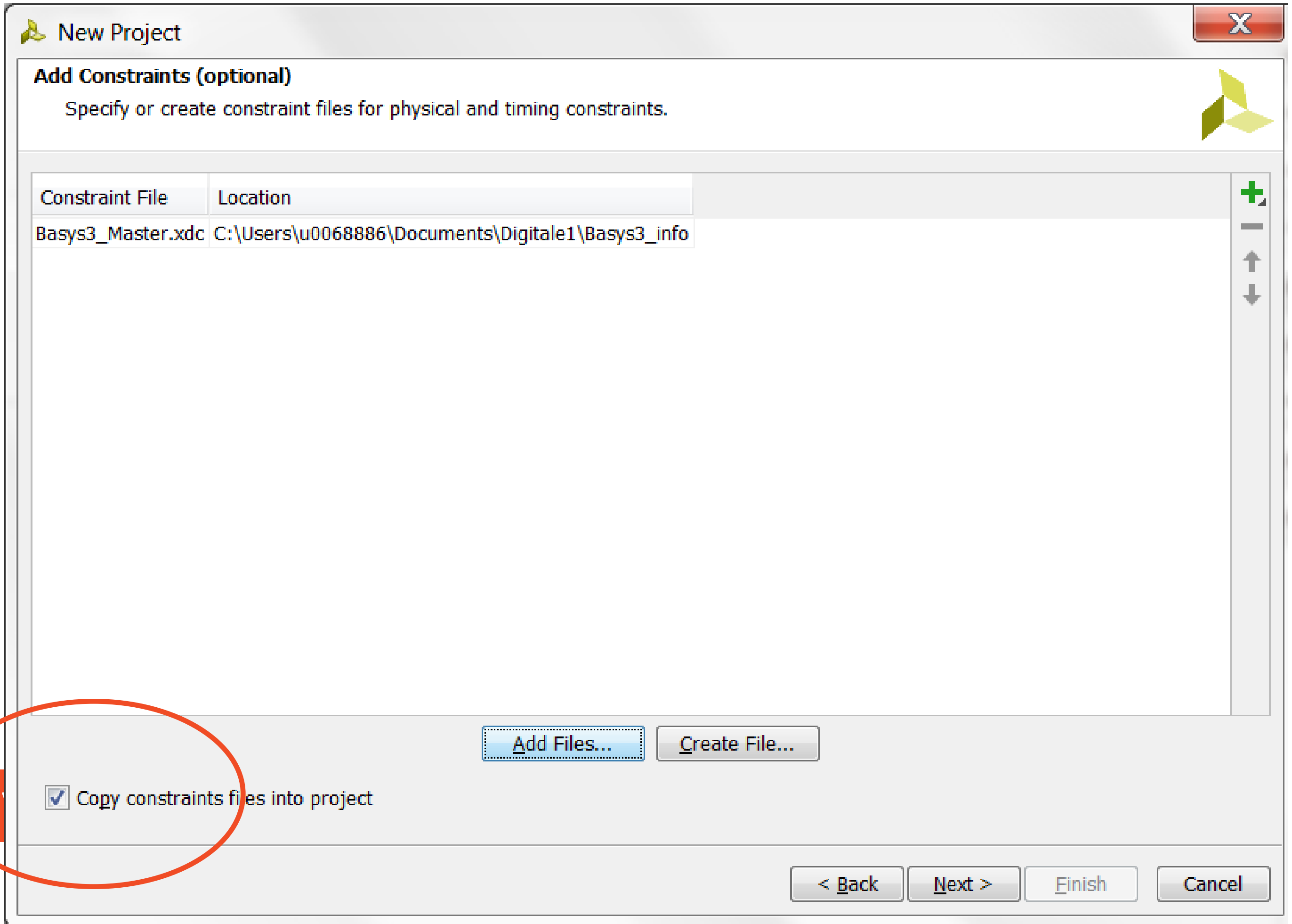
Specify existing configurable IP, DSP composite, and Embedded sub-design files to add to your project.

Index	Name	Library	HDL Sour...	Location

☑ Copy sources into project

Project set up

- Add xdc-file: Basys3_master.xdc (USB or Digilent)


New Project

Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Constraint File	Location
Basys3_Master.xdc	C:\Users\u0068886\Documents\Digitale1\Basys3_info

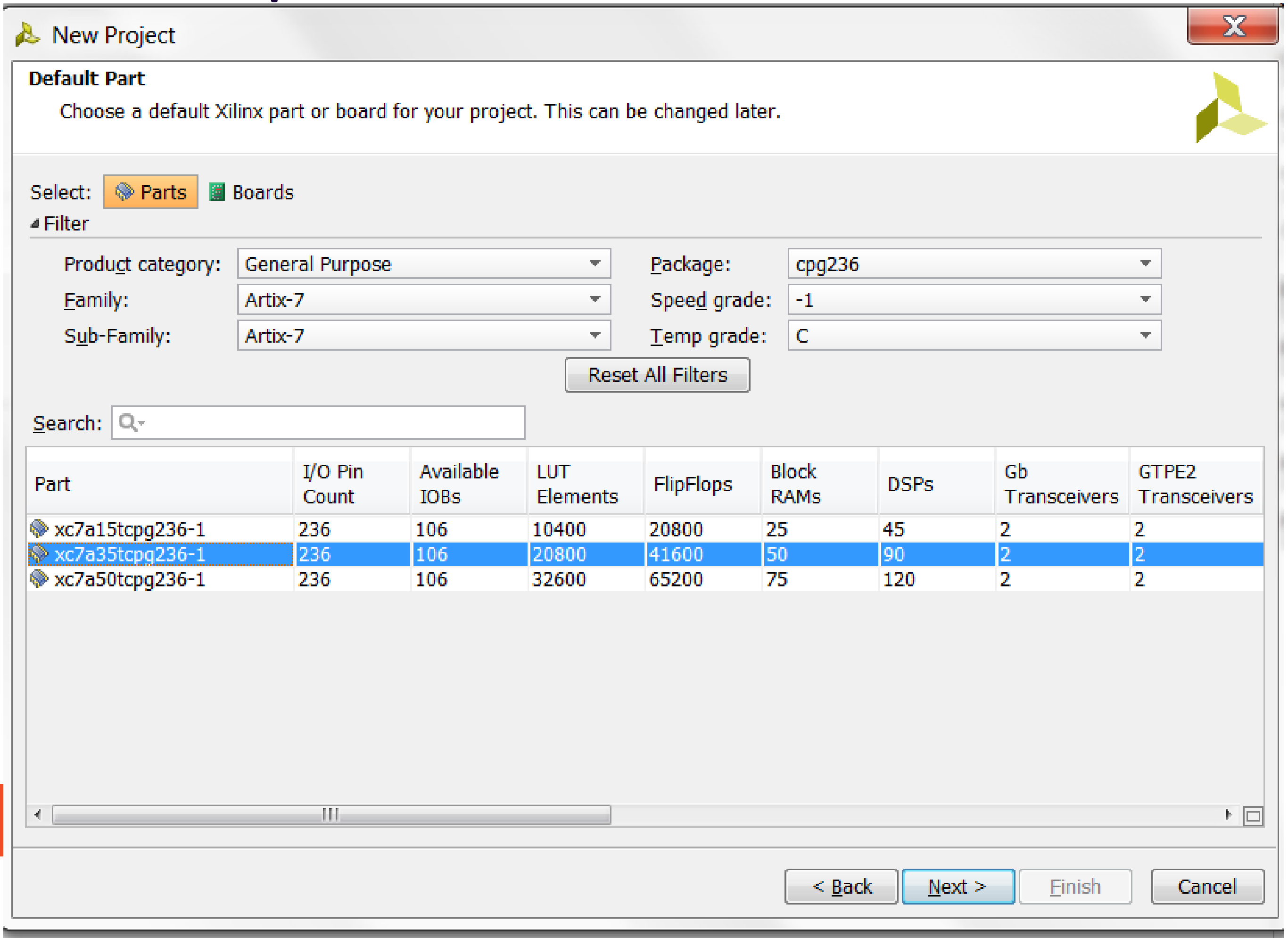
✓ Copy constraints files into project

Project set up

- Define part or board
  - General purpose
  - Artix 7
  - Artix 7
  - CPG236
  - Speed grade -1
  - C
  - 35T

Project set up

- Define part or board



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts Boards

Filter

Product category:	General Purpose	Package:	cpg236
Family:	Artix-7	Speed grade:	-1
Sub-Family:	Artix-7	Temp grade:	C

Reset All Filters

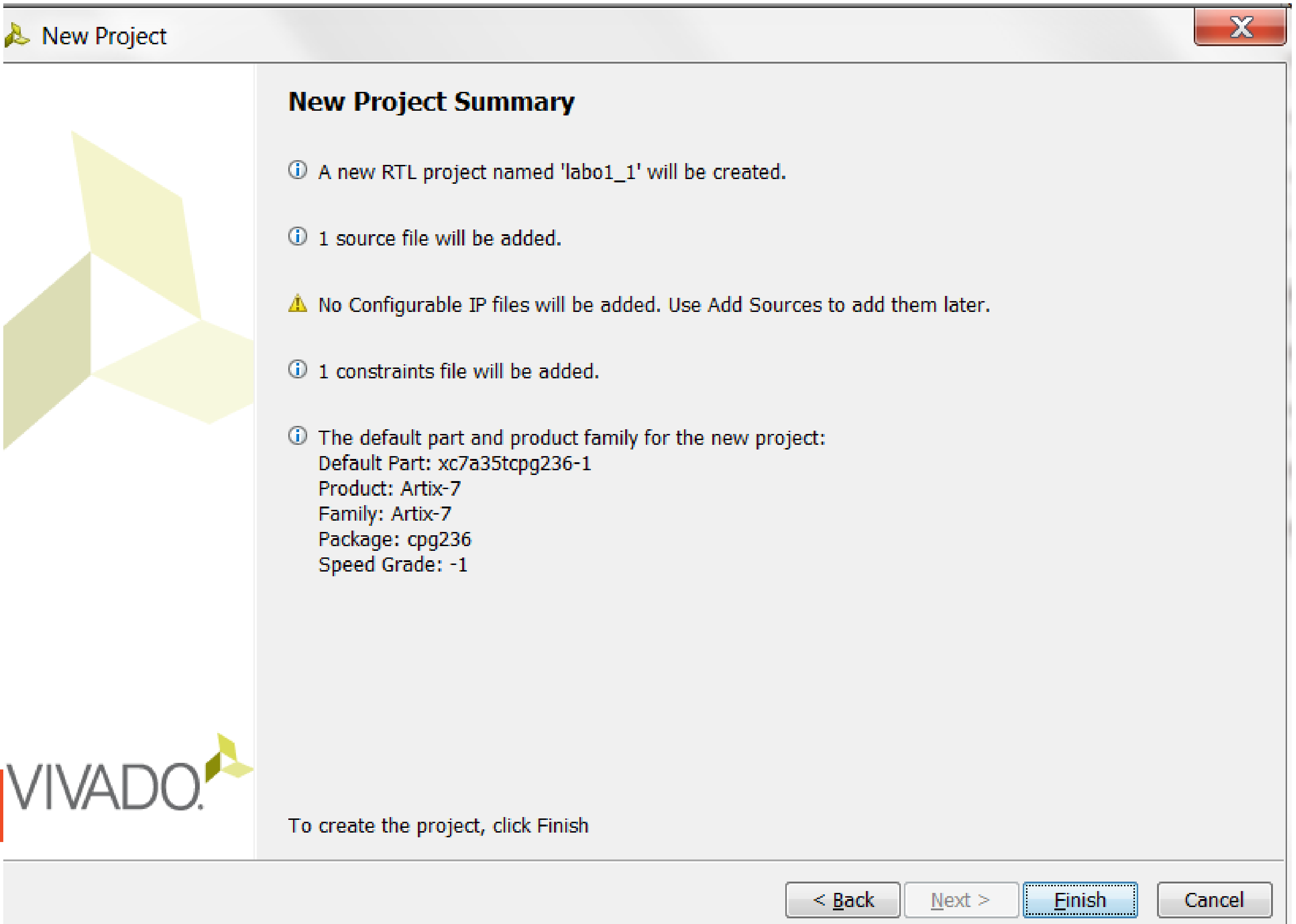
Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE2 Transceivers
xc7a15tcpg236-1	236	106	10400	20800	25	45	2	2
xc7a35tcpg236-1	236	106	20800	41600	50	90	2	2
xc7a50tcpg236-1	236	106	32600	65200	75	120	2	2

< Back Next > Finish Cancel

Project set up

- Finish


New Project

New Project Summary

ⓘ A new RTL project named 'labo1_1' will be created.

ⓘ 1 source file will be added.

⚠ No Configurable IP files will be added. Use Add Sources to add them later.

ⓘ 1 constraints file will be added.

ⓘ The default part and product family for the new project:
Default Part: xc7a35tcpg236-1
Product: Artix-7
Family: Artix-7
Package: cpg236
Speed Grade: -1

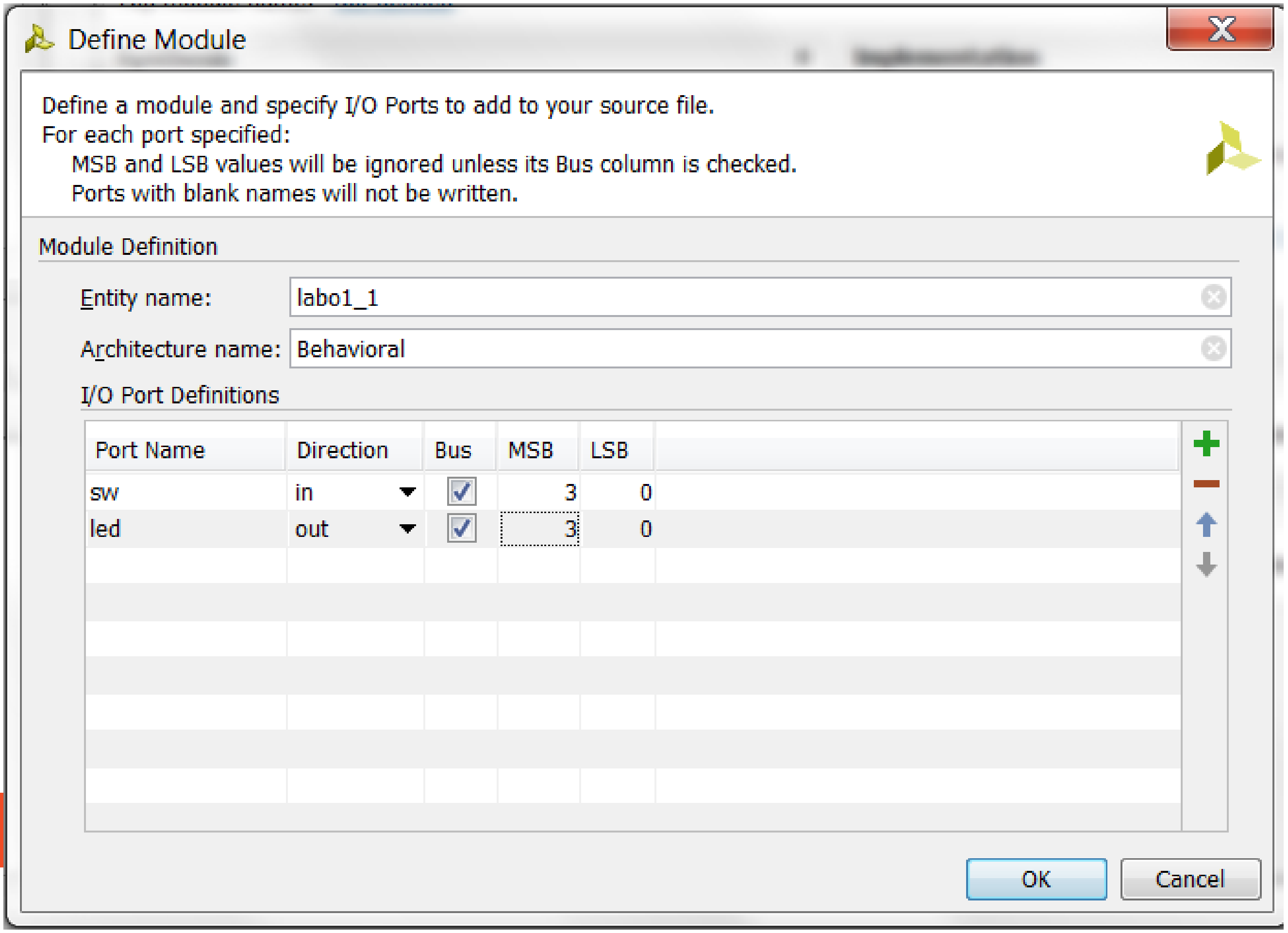
To create the project, click Finish

Project set up

- Define ports
  - Name,
  - Direction,
  - Width (vector or not)
  - Type is always std_logic(_vector)

Project set up

- Define ports



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
 MSB and LSB values will be ignored unless its Bus column is checked.
 Ports with blank names will not be written.

Module Definition

Entity name: labo1_1

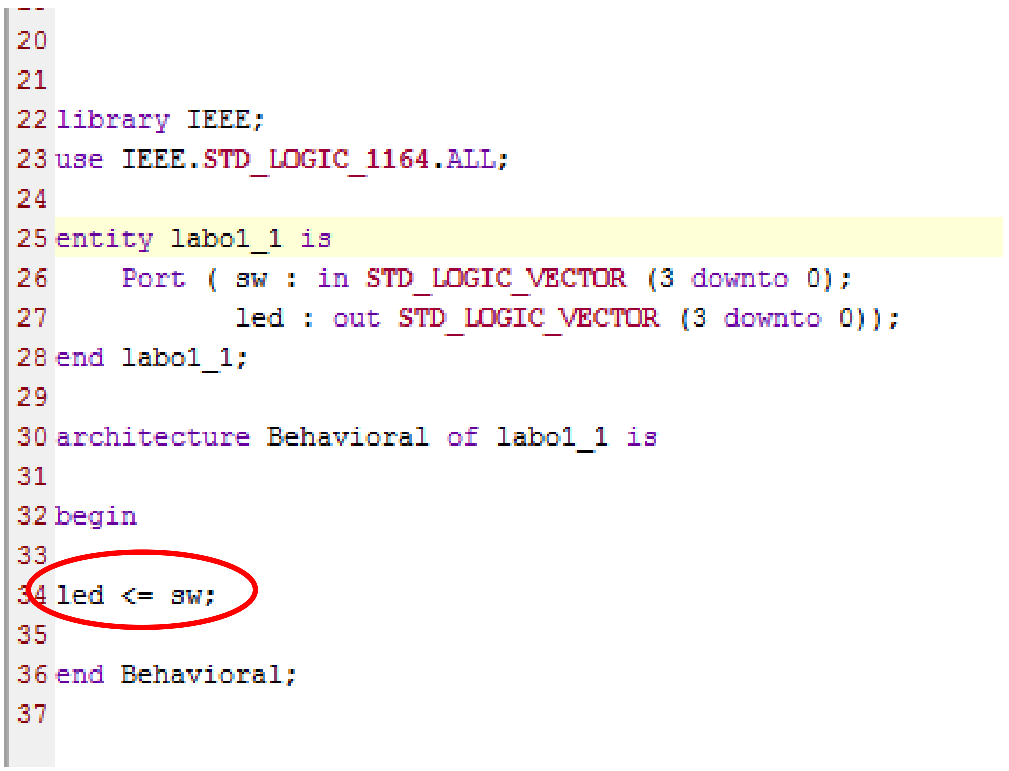
Architecture name: Behavioral

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
sw	in	✓	3	0
led	out	✓	3	0

Project set up

- Write code

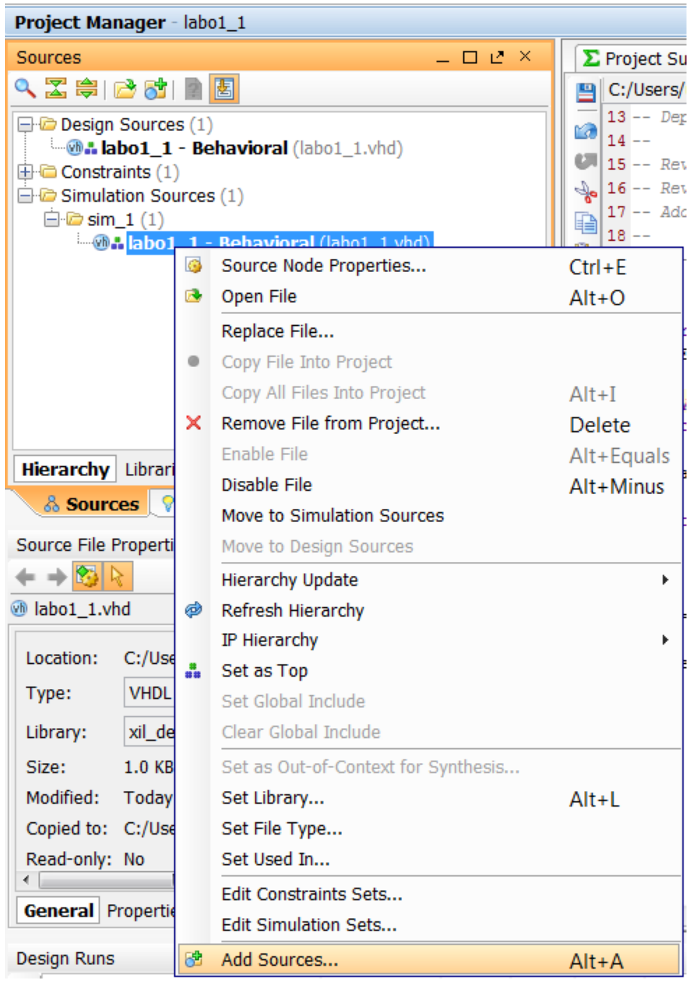


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity labo1_1 is
    Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
           led : out STD_LOGIC_VECTOR (3 downto 0));
end labo1_1;

architecture Behavioral of labo1_1 is

begin

led <= sw;

end Behavioral;
```

Tempus

Project set up

- Simulate your code
  - Add testbench



Project Manager - labo1_1

Sources

- Design Sources (1)
  - labo1_1 - Behavioral (labo1_1.vhd)
- Constraints (1)
- Simulation Sources (1)
  - sim_1 (1)
    - labo1_1 - Behavioral (labo1_1.vhd)

Source Node Properties...	Ctrl+E
Open File	Alt+O
Replace File...	
Copy File Into Project	
Copy All Files Into Project	Alt+I
Remove File from Project...	Delete
Enable File	Alt+Equals
Disable File	Alt+Minus
Move to Simulation Sources	
Move to Design Sources	
Hierarchy Update	
Refresh Hierarchy	
IP Hierarchy	
Set as Top	
Set Global Include	
Clear Global Include	
Set as Out-of-Context for Synthesis...	
Set Library...	Alt+L
Set File Type...	
Set Used In...	
Edit Constraints Sets...	
Edit Simulation Sets...	
Add Sources...	Alt+A

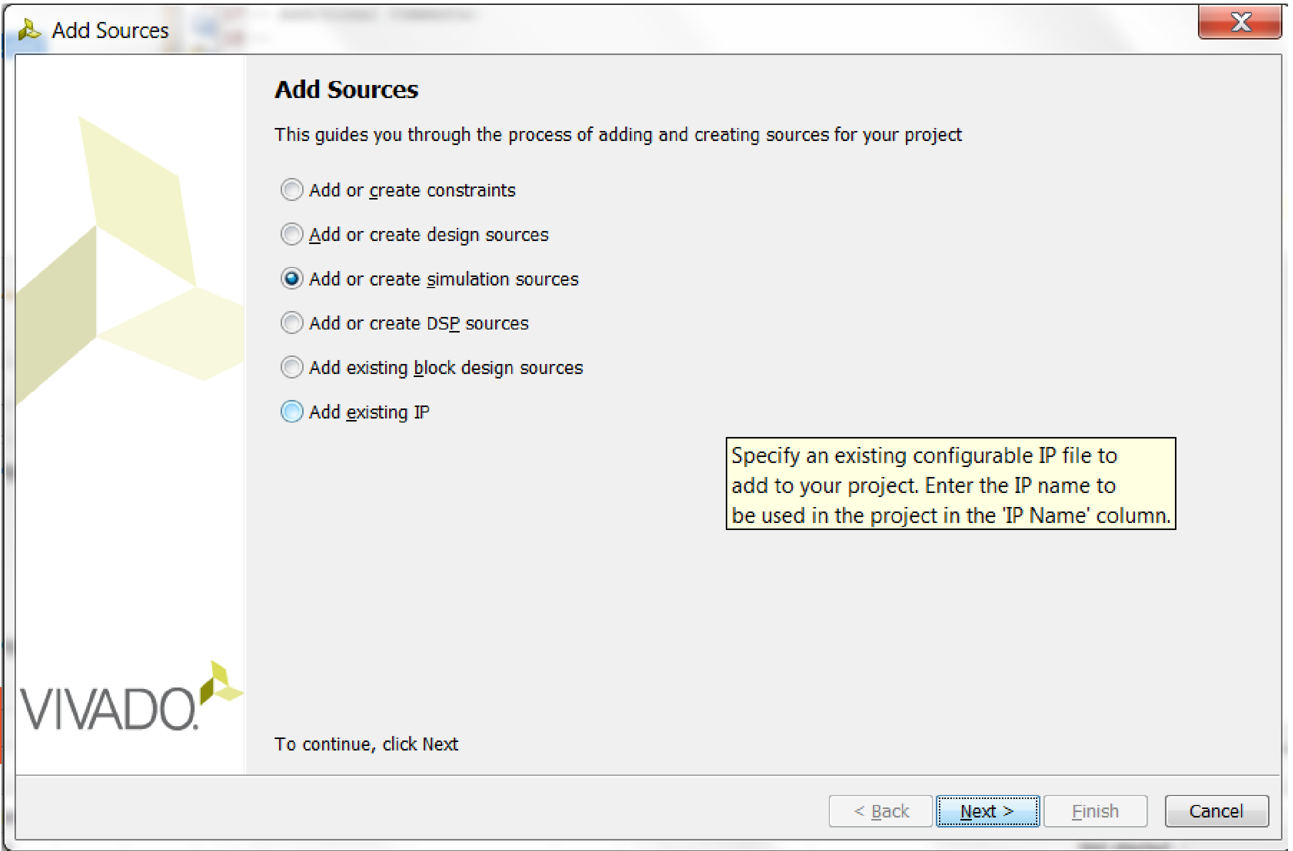
Source File Properti

labo1_1.vhd

Location:	C:/Use
Type:	VHDL
Library:	xil_de
Size:	1.0 KB
Modified:	Today
Copied to:	C:/Use
Read-only:	No

Project set up

- Simulate your code
    - o Add testbench



Add Sources

Add Sources

This guides you through the process of adding and creating sources for your project

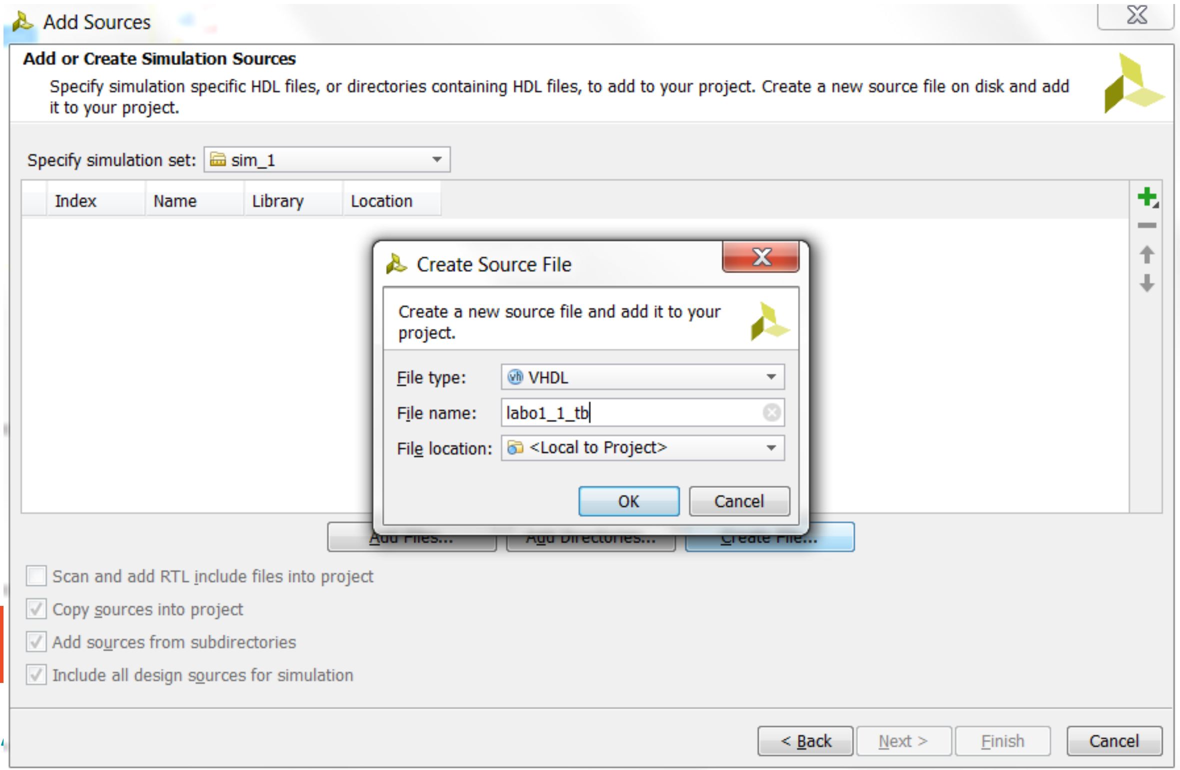
- ( ) Add or create constraints
- ( ) Add or create design sources
- [✓ Add or create simulation sources]
- ( ) Add or create DSP sources
- ( ) Add existing block design sources
- ( ) Add existing IP

Specify an existing configurable IP file to add to your project. Enter the IP name to be used in the project in the 'IP Name' column.

To continue, click Next

Project set up

- Simulate your code
  - Add testbench
    - Create file


Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Index	Name	Library	Location

Create Source File

Create a new source file and add it to your project.

File type: VHDL

File name: labo1_1_tb

File location: <Local to Project>

☐ Scan and add RTL include files into project

☑ Copy sources into project

☑ Add sources from subdirectories

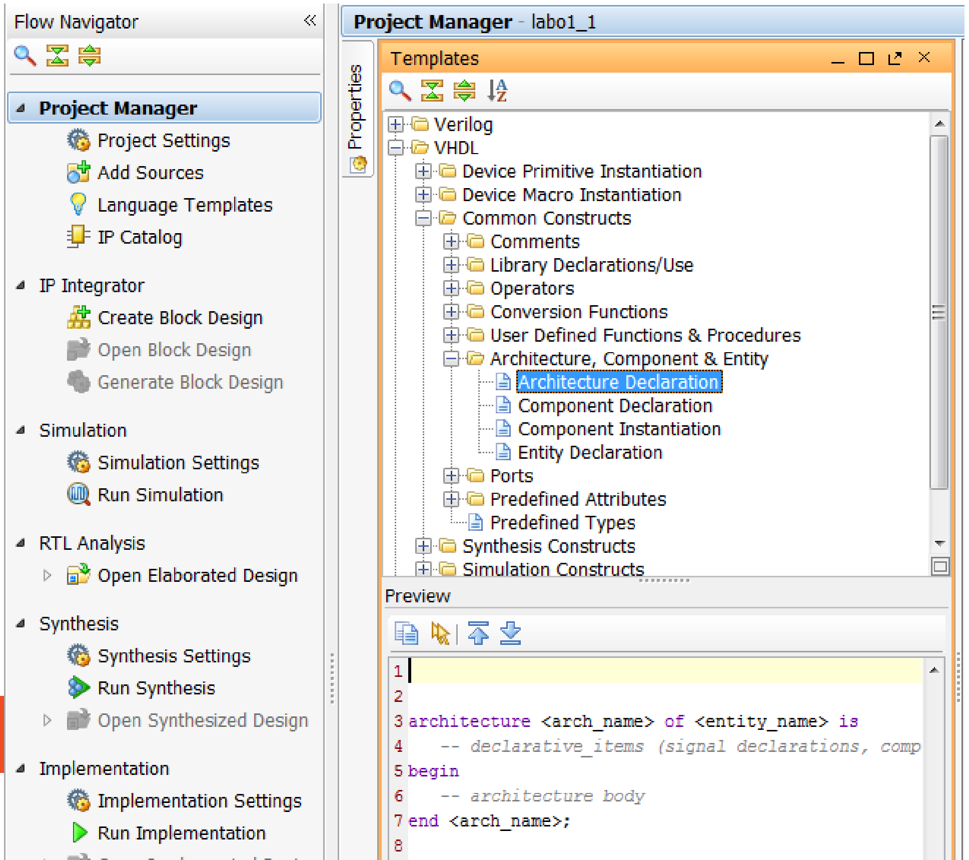
☑ Include all design sources for simulation

Project set up

- Simulate your code
  - Add testbench
    - Add libraries (templates => common constructs => libraries => commonly used)
    - Write empty entity (Language Templates => Common Constructs => Architecture, Component & Entity => Entity Declaration)
    - Write architecture (Language Templates => Common Constructs => Architecture, Component & Entity => Architecture Declaration)

Project set up

- Simulate your code
  - Add testbench



Flow Navigator

Project Manager
- Project Settings
- Add Sources
- Language Templates
- IP Catalog

IP Integrator
- Create Block Design
- Open Block Design
- Generate Block Design

Simulation
- Simulation Settings
- Run Simulation

RTL Analysis
- Open Elaborated Design

Synthesis
- Synthesis Settings
- Run Synthesis
- Open Synthesized Design

Implementation
- Implementation Settings
- Run Implementation

Project Manager - labo1_1

Templates

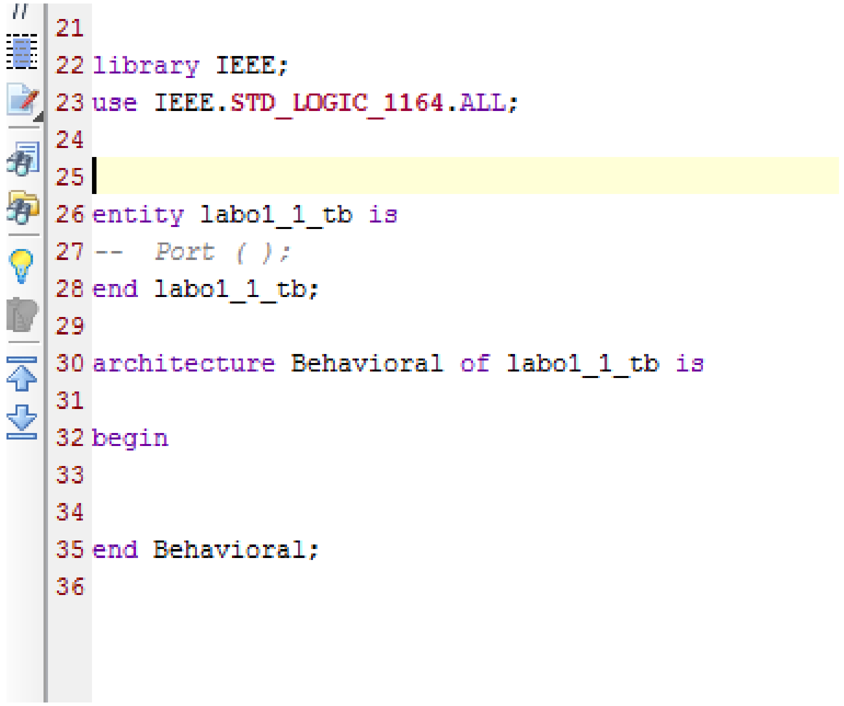
- Verilog
- VHDL
  - Device Primitive Instantiation
  - Device Macro Instantiation
  - Common Constructs
    - Comments
    - Library Declarations/Use
    - Operators
    - Conversion Functions
    - User Defined Functions & Procedures
    - Architecture, Component & Entity
      - Architecture Declaration
      - Component Declaration
      - Component Instantiation
      - Entity Declaration
    - Ports
    - Predefined Attributes
    - Predefined Types
  - Synthesis Constructs
  - Simulation Constructs

Preview

```

architecture <arch_name> of <entity_name> is
   -- declarative_items (signal declarations, comp
begin
   -- architecture body
end <arch_name>;

```

Project set up

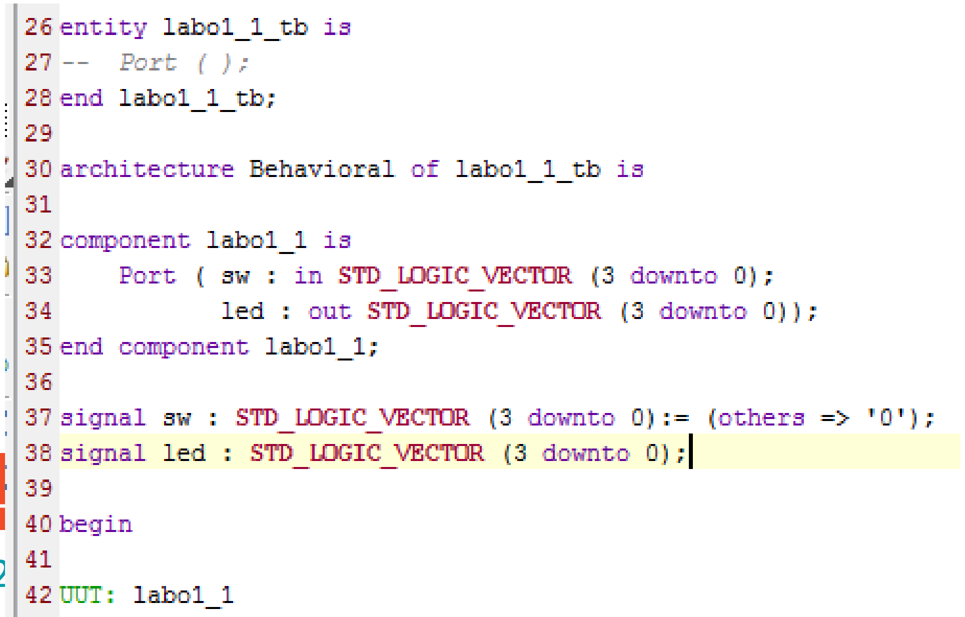
- Simulate your code
    - Add testbench



```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity labo1_1_tb is
--   Port ( );
end labo1_1_tb;

architecture Behavioral of labo1_1_tb is

begin


end Behavioral;

```

Project set up

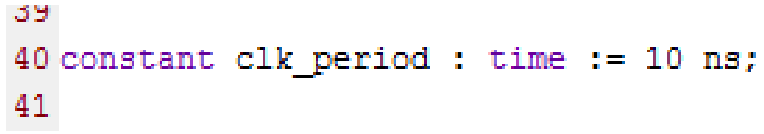
- Simulate your code
  - Add testbench
    - Add, declare component to test
    - Add, declare input and output signals for stimulating the Unit Under Test (UUT)



```vhdl
entity labo1_1_tb is
--   Port ( );
end labo1_1_tb;

architecture Behavioral of labo1_1_tb is

component labo1_1 is
    Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
           led : out STD_LOGIC_VECTOR (3 downto 0));
end component labo1_1;

signal sw : STD_LOGIC_VECTOR (3 downto 0):= (others => '0');
signal led : STD_LOGIC_VECTOR (3 downto 0);

begin

UUT: labo1_1
```

Project set up

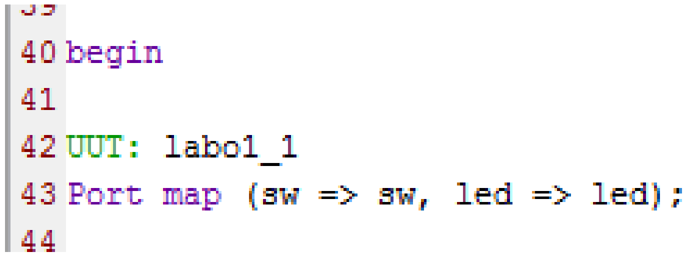
- Simulate your code
  - Add testbench
    - Add, declare constant clock period (not used in first exercises)



```
constant clk_period : time := 10 ns;
```

Project set up

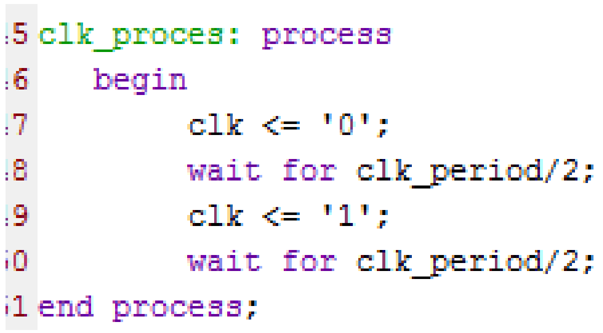
- Simulate your code
  - Add testbench
    - Use, instantiate the UUT, name & connect port to signals (port map)



```
begin

UUT: labo1_1
Port map (sw => sw, led => led);

```

Project set up

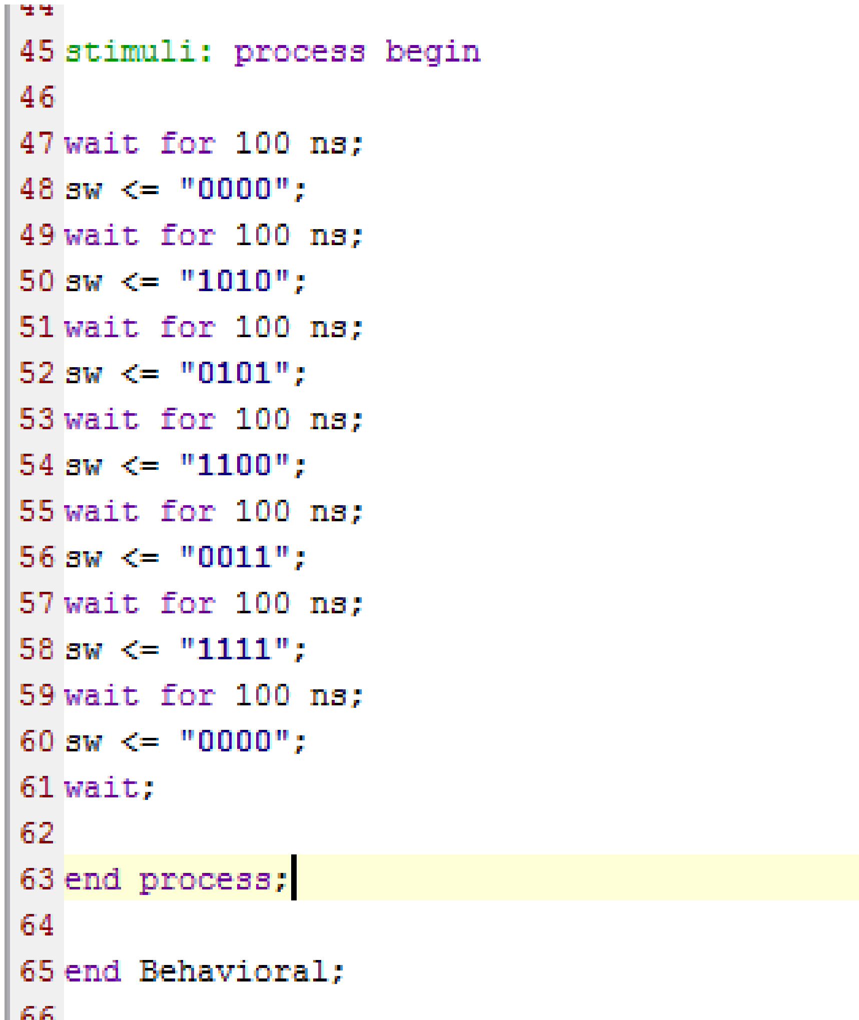
- Simulate your code
    - Add testbench
        - Write process for clock (not used in first exercises)



```
clk_proces: process
   begin
         clk <= '0';
         wait for clk_period/2;
         clk <= '1';
         wait for clk_period/2;
end process;
```

Project set up

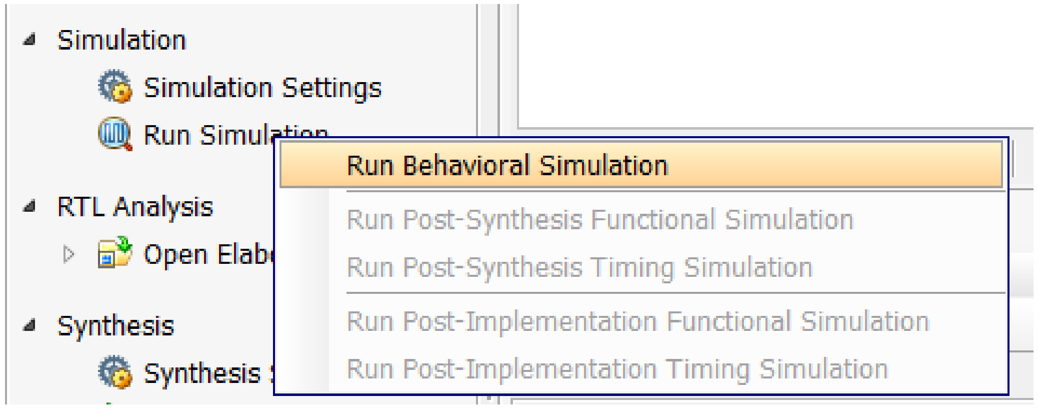
- Simulate your code
  - Add testbench
    - Write process for input stimuli



```
stimuli: process begin

wait for 100 ns;
sw <= "0000";
wait for 100 ns;
sw <= "1010";
wait for 100 ns;
sw <= "0101";
wait for 100 ns;
sw <= "1100";
wait for 100 ns;
sw <= "0011";
wait for 100 ns;
sw <= "1111";
wait for 100 ns;
sw <= "0000";
wait;

end process;

end Behavioral;
```

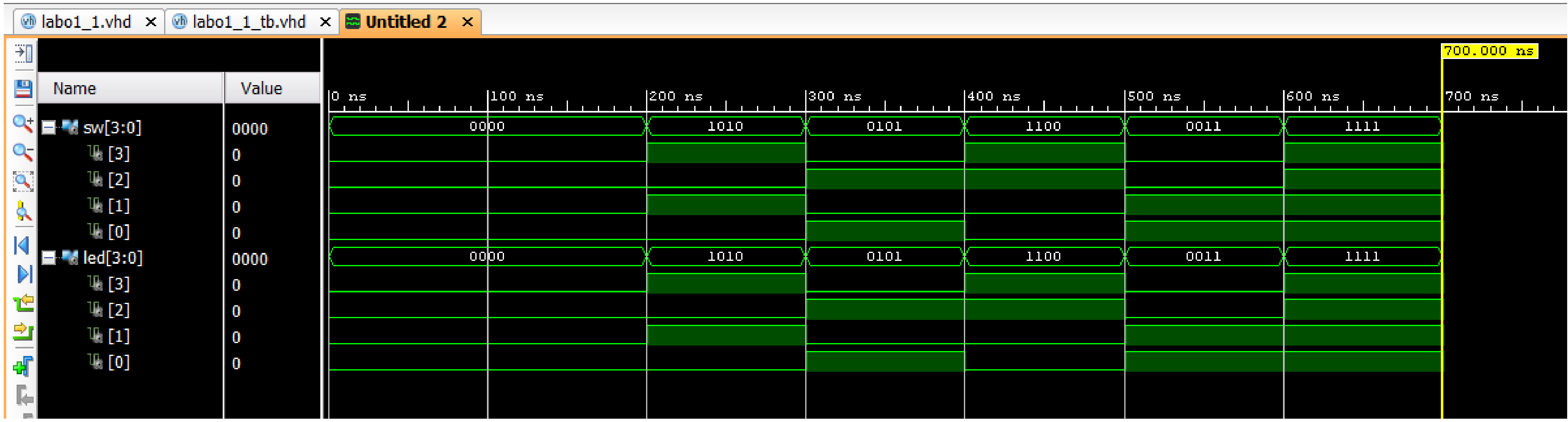
Project set up

- Simulate your code
  - Run simulation


Simulation
Simulation Settings
Run Simulation
RTL Analysis
Open Elab
Synthesis
Synthesis
Run Behavioral Simulation
Run Post-Synthesis Functional Simulation
Run Post-Synthesis Timing Simulation
Run Post-Implementation Functional Simulation
Run Post-Implementation Timing Simulation

Project set up

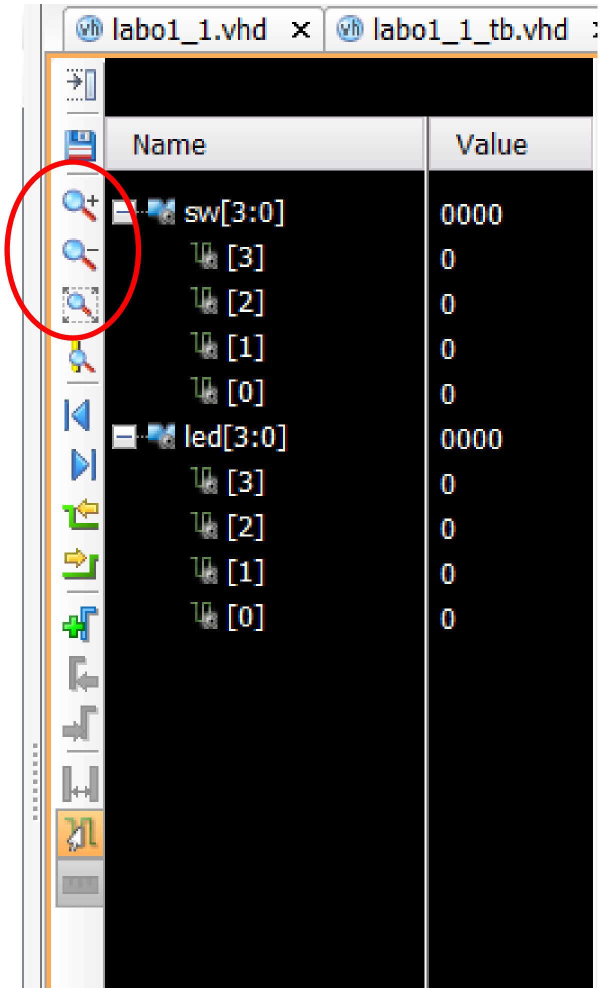
- Simulate your code
  - Check outputs in relation with the inputs



```
Name        Value   0000  1010  0101  1100  0011  1111
sw[3:0]     0000
  [3]       0
  [2]       0
  [1]       0
  [0]       0
led[3:0]    0000    0000  1010  0101  1100  0011  1111
  [3]       0
  [2]       0
  [1]       0
  [0]       0
```

Project set up

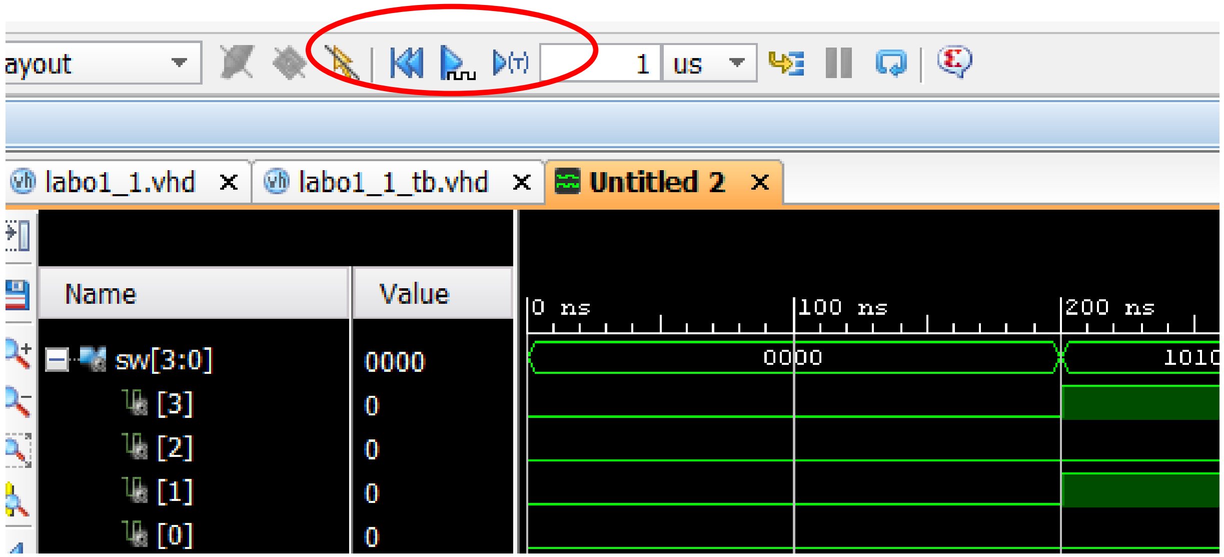
- Simulate your code
  - Zoom in & out



```
Name            Value
sw[3:0]         0000
  [3]           0
  [2]           0
  [1]           0
  [0]           0
led[3:0]        0000
  [3]           0
  [2]           0
  [1]           0
  [0]           0
```

Project set up

- Simulate your code
  - Restart / Run all / Run for 1us: test the features

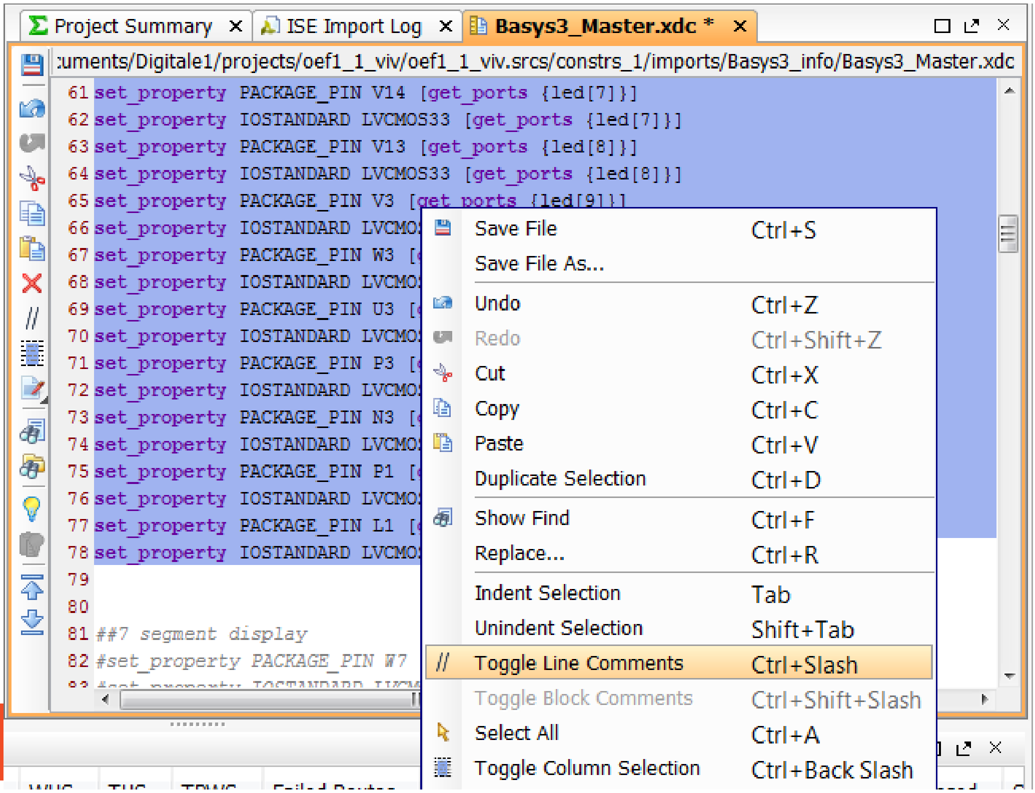


Project set up

- Activate the right properties in the xdc-file:
  - change naming
    - Names in entity and xdc should be the same
  - toggle line comments for using the ports from the entity declaration of your VHDL-file

Project set up

- Activate the right properties in the xdc-file:

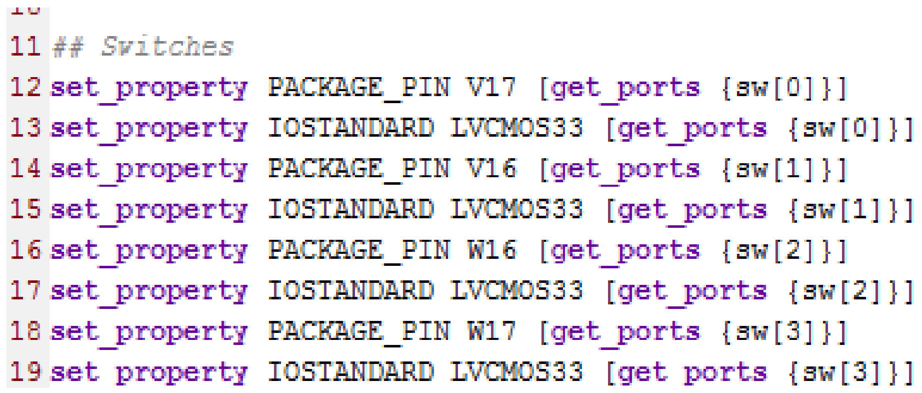


```
:uments/Digitale1/projects/oef1_1_viv/oef1_1_viv.srcs/constrs_1/imports/Basys3_info/Basys3_Master.xdc
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN W3 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN U3 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN P3 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN N3 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN P1 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN L1 [
set_property IOSTANDARD LVCMO


##7 segment display
#set_property PACKAGE_PIN W7
```

Save File Ctrl+S
Save File As...
Undo Ctrl+Z
Redo Ctrl+Shift+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Duplicate Selection Ctrl+D
Show Find Ctrl+F
Replace... Ctrl+R
Indent Selection Tab
Unindent Selection Shift+Tab
Toggle Line Comments Ctrl+Slash
Toggle Block Comments Ctrl+Shift+Slash
Select All Ctrl+A
Toggle Column Selection Ctrl+Back Slash

Project set up

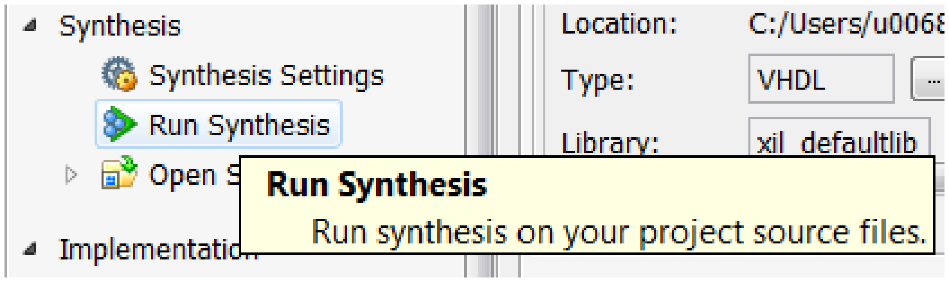
- Activate the right properties in the xdc-file:



```
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
```

Project set up

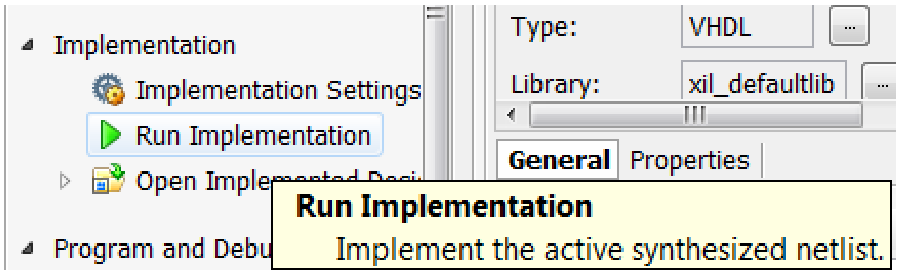
- Synthesize
    - o Translation of VHDL into hardware
    - o Result is a VHDL netlist (gates & connections)
        - – Next step: FPGA or ASIC



Run Synthesis
Run synthesis on your project source files.

Project set up

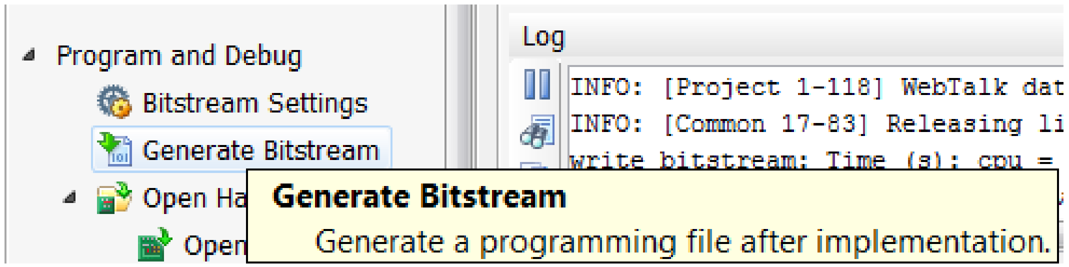
- Implement
  - Translate netlist into LUTs, flipflops, internal connections, IOBs



Run Implementation
Implement the active synthesized netlist.

Project set up

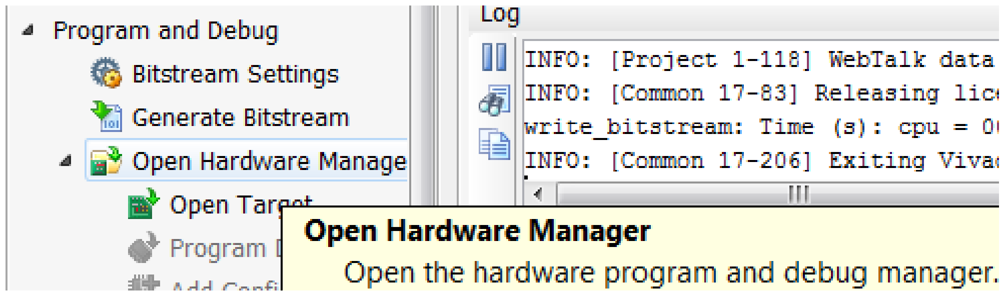
- Program & Debug
  - Generate Bitstream
    - Place ‘1 or ‘0’ in SRAM
    - Size is always the same for a given FPGA



```
INFO: [Project 1-118] WebTalk dat
INFO: [Common 17-83] Releasing li
write_bitstream: Time (s): cpu =
```

Project set up

- Program & Debug
  - o Open hardware manager



- Program and Debug
  - Bitstream Settings
  - Generate Bitstream
  - Open Hardware Manage
    - Open Target
    - Program

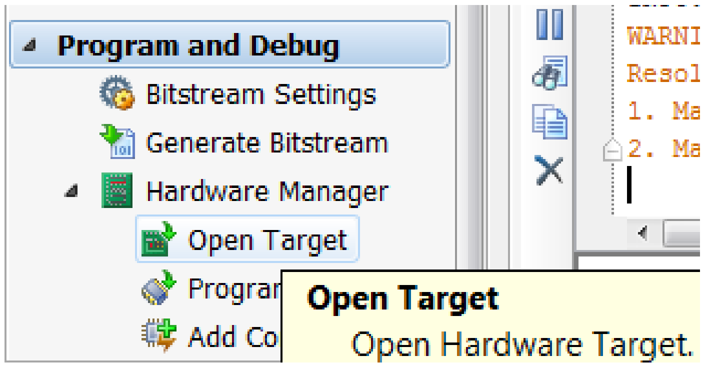
Log

```
INFO: [Project 1-118] WebTalk data
INFO: [Common 17-83] Releasing lic
write_bitstream: Time (s): cpu = 0
INFO: [Common 17-206] Exiting Viva
```

Open Hardware Manager
Open the hardware program and debug manager.

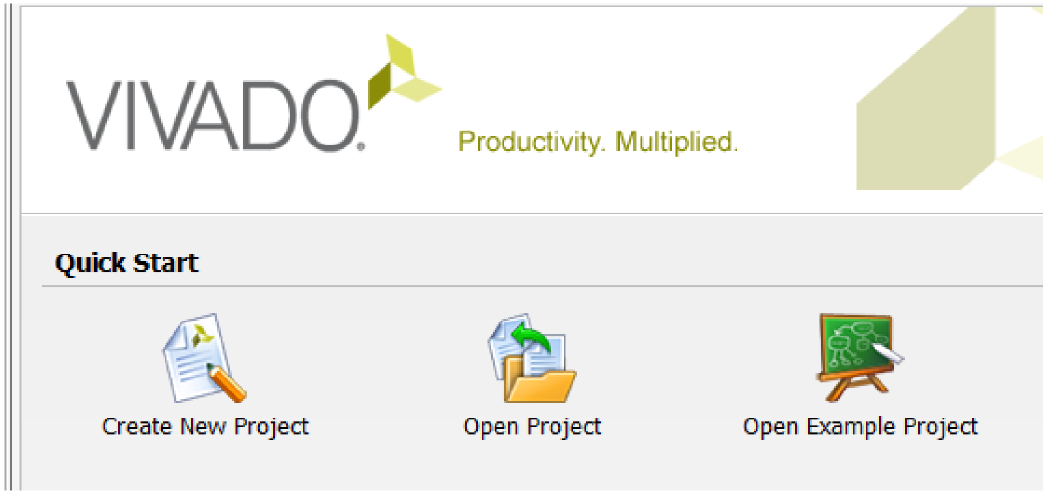
Project set up

- Program & Debug
  - Open target, with board connected & powered



Import ISE project

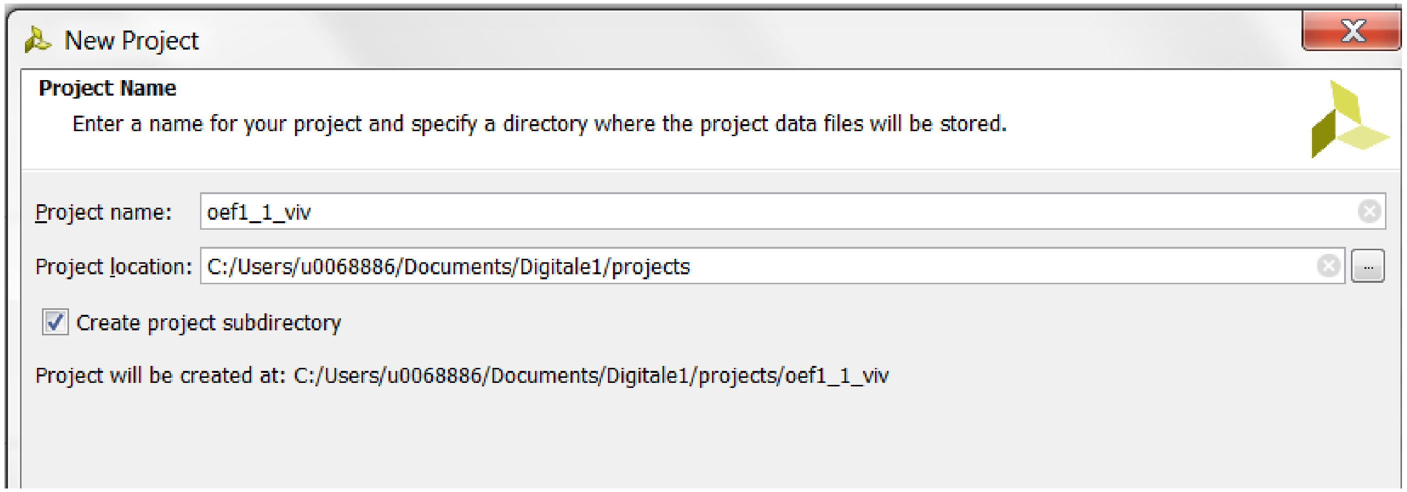
- Start a new project


VIVADO. Productivity. Multiplied.

Quick Start

Import ISE project

- Give the name of the imported ISE project with "_viv" to point out that it is a Vivado imported project.


New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: oef1_1_viv

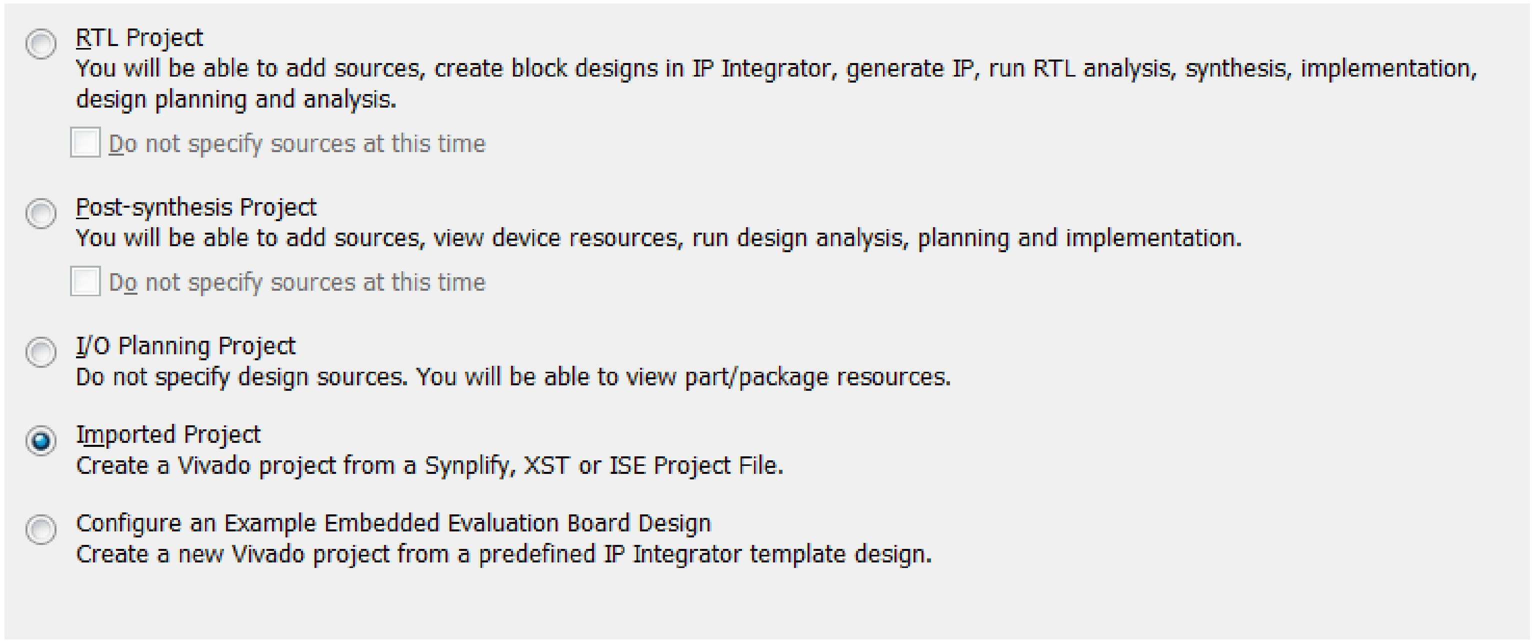
Project location: C:/Users/u0068886/Documents/Digitale1/projects

☑ Create project subdirectory

Project will be created at: C:/Users/u0068886/Documents/Digitale1/projects/oef1_1_viv

Import ISE project

- Choose imported project


( ) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
[ ] Do not specify sources at this time

( ) Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
[ ] Do not specify sources at this time

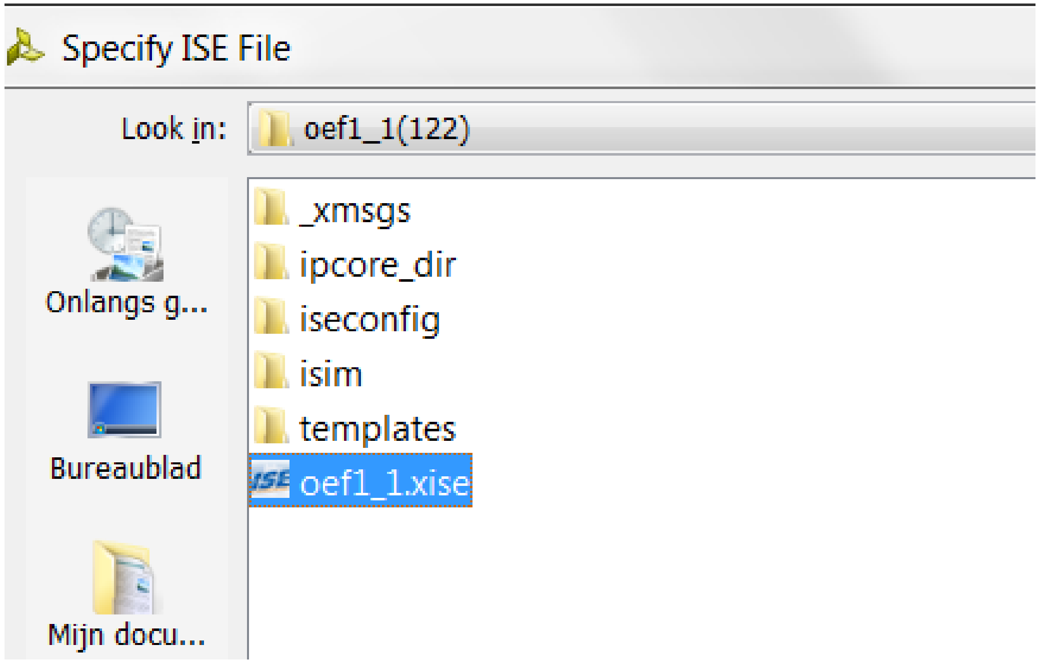
( ) I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

[✓ Imported Project]
Create a Vivado project from a Synplify, XST or ISE Project File.

( ) Configure an Example Embedded Evaluation Board Design
Create a new Vivado project from a predefined IP Integrator template design.

Import ISE project

- Specify the project



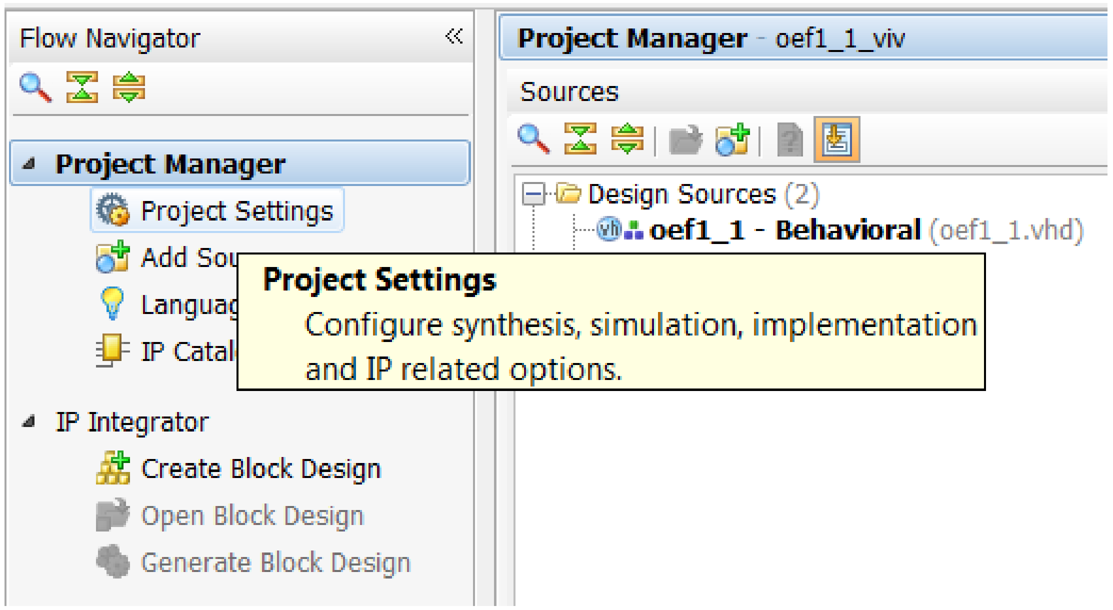
Specify ISE File

Look in: oef1_1(122)

_xmsgs
ipcore_dir
iseconfig
isim
templates
oef1_1.xise

Import ISE project

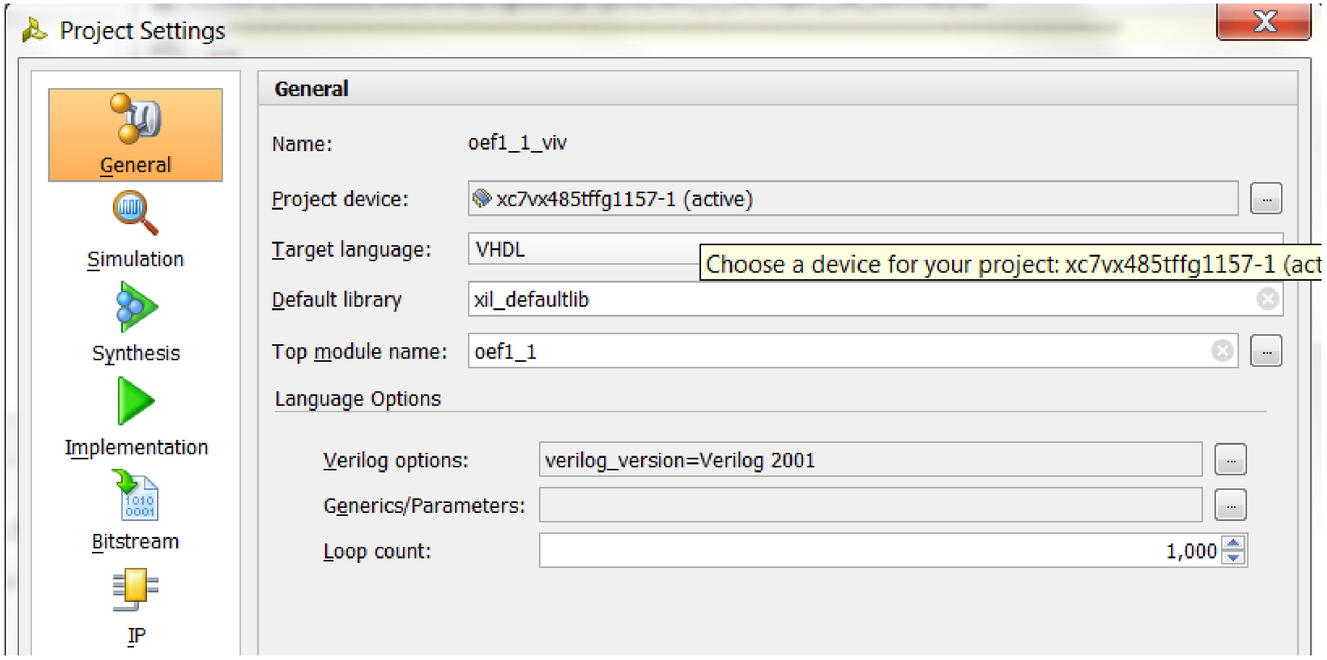
- Check & change if it is the proper FPGA you are using in project settings.



Project Settings
Configure synthesis, simulation, implementation and IP related options.

Import ISE project

- Check & change if it is the proper FPGA you are using in project settings.



General

Name: oef1_1_viv

Project device: xc7vx485tffg1157-1 (active)

Target language: VHDL

Choose a device for your project: xc7vx485tffg1157-1 (act

Default library: xil_defaultlib

Top module name: oef1_1

Language Options

Verilog options: verilog_version=Verilog 2001

Generics/Parameters:

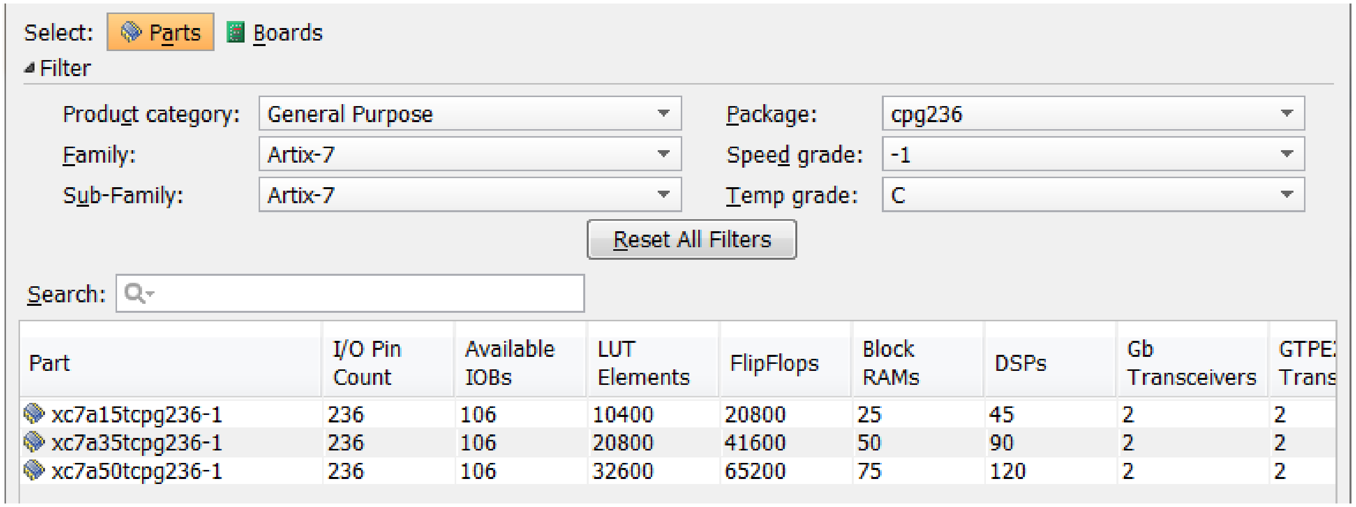
Loop count: 1,000

Import ISE project

- Check & change if it is the proper FPGA you are using in project settings.
  - The proper device FPGA is:
    - General purpose
    - Artix 7
    - Artix 7
    - CPG236
    - Speed grade -1
    - C
    - 35T

Import ISE project

- Check & change if it is the proper FPGA you are using in project settings.



Select: Parts Boards

Filter

Product category:	General Purpose	Package:	cpg236
Family:	Artix-7	Speed grade:	-1
Sub-Family:	Artix-7	Temp grade:	C

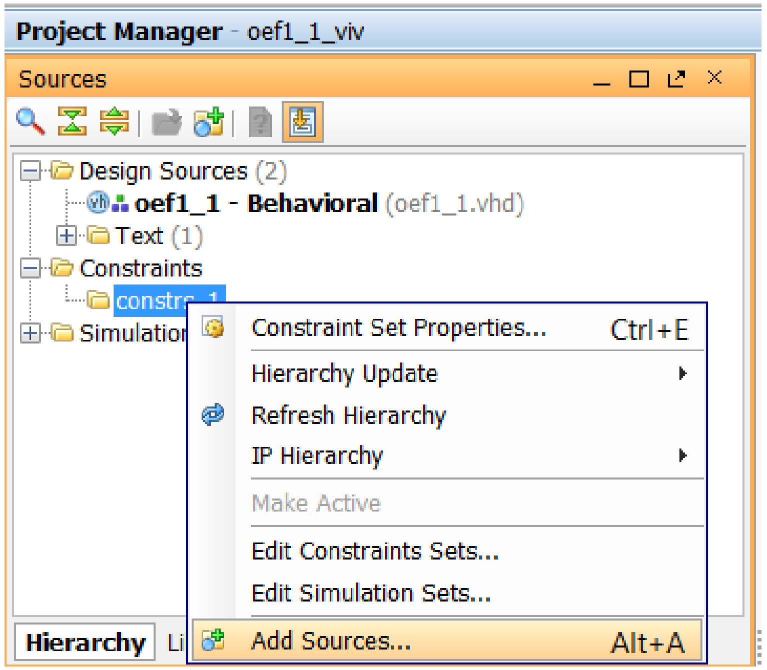
Reset All Filters

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE: Trans
xc7a15tcpg236-1	236	106	10400	20800	25	45	2	2
xc7a35tcpg236-1	236	106	20800	41600	50	90	2	2
xc7a50tcpg236-1	236	106	32600	65200	75	120	2	2

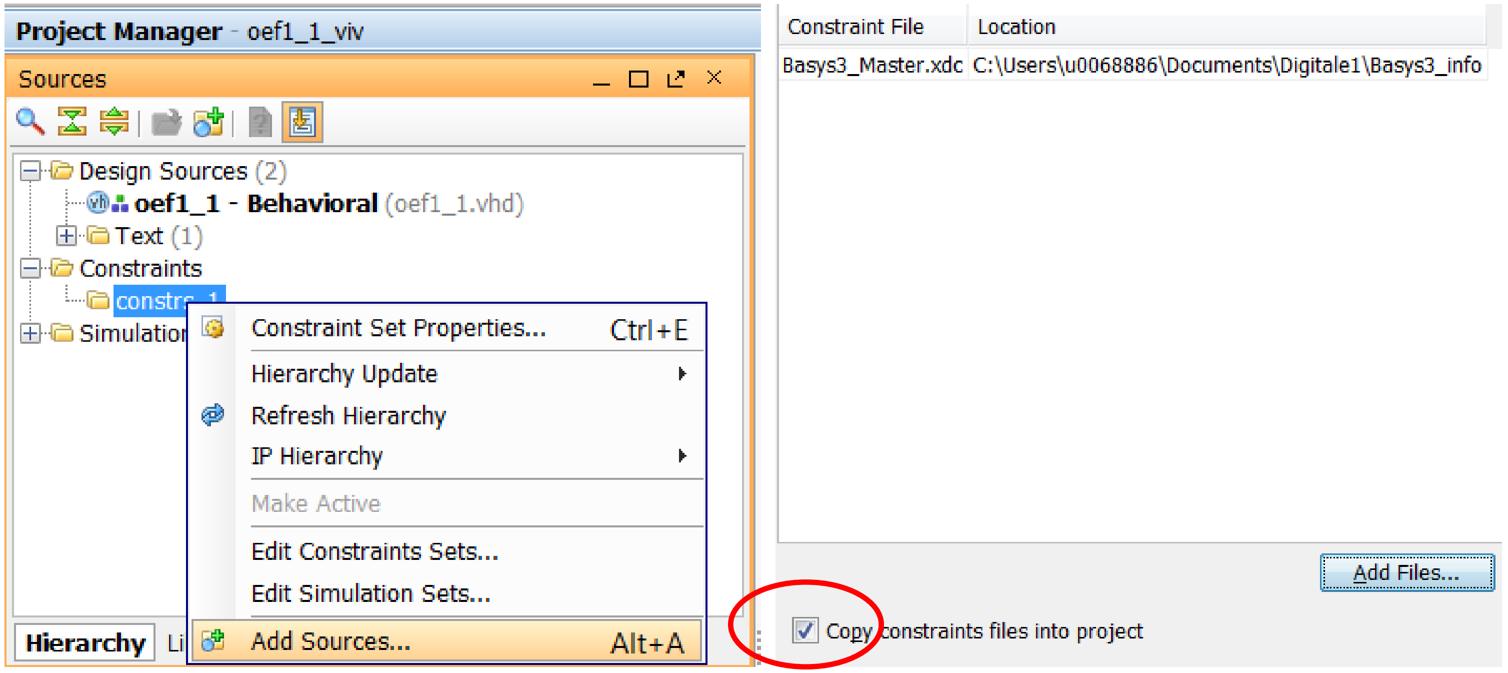
Import ISE project

- Add xdc-file
- Use copy of this file



Import ISE project

- Add xdc-file
- Use copy of this file



Constraint File	Location
Basys3_Master.xdc	C:\Users\u0068886\Documents\Digitale1\Basys3_info

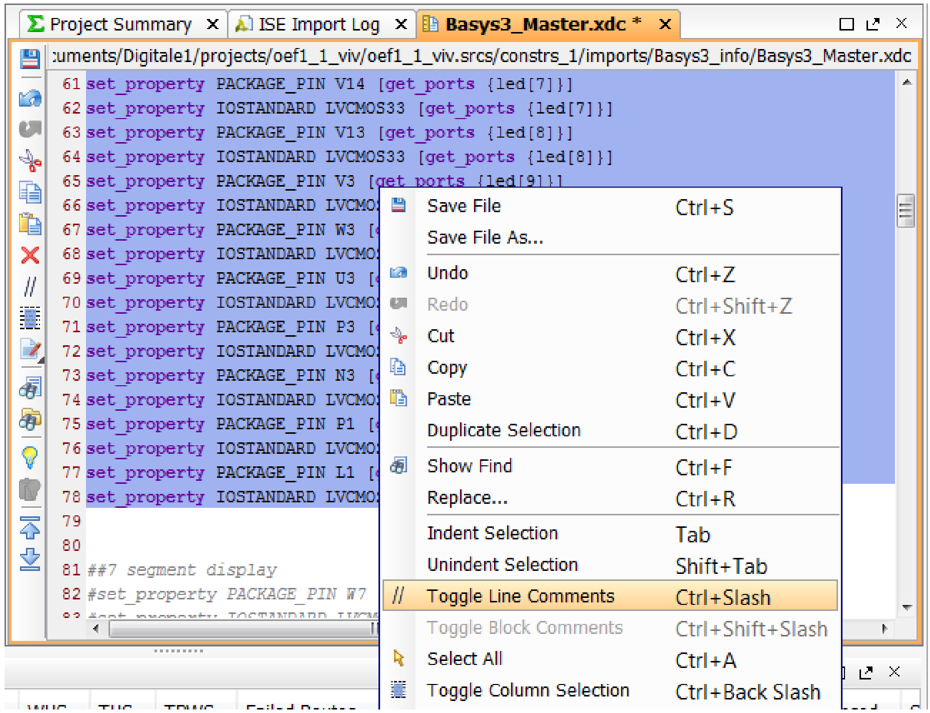
[✓] Copy constraints files into project

Import ISE project

- Add xdc-file
  - Activate the right properties in the xdc-file,
    - change naming,
    - toggle line comments for using the ports from the entity declaration of your VHDL-file

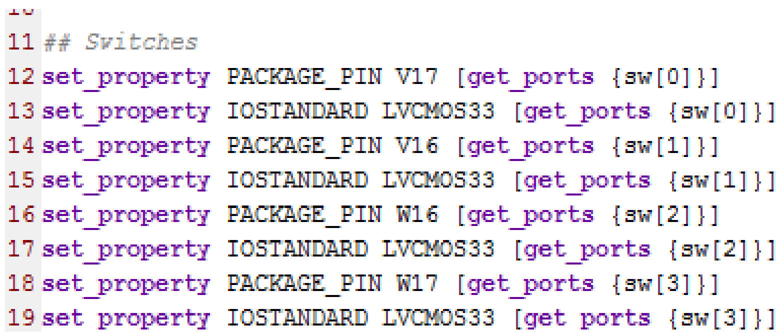
Import ISE project

- Add xdc-file



:uments/Digitale1/projects/oef1_1_viv/oef1_1_viv.srcs/constrs_1/imports/Basys3_info/Basys3_Master.xdc

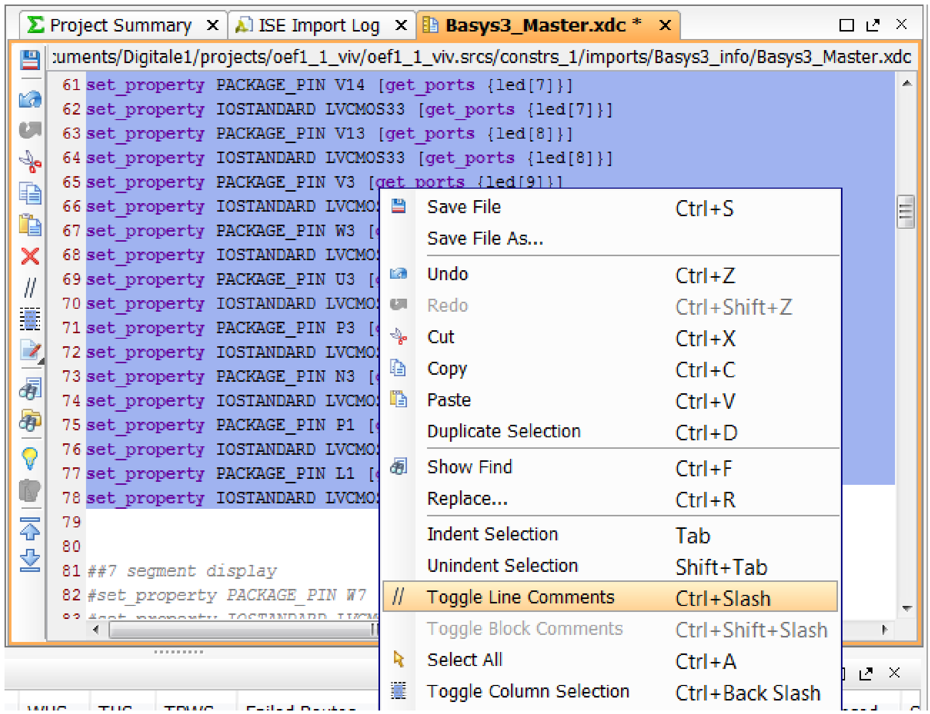
```
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN W3 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN U3 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN P3 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN N3 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN P1 [
set_property IOSTANDARD LVCMO
set_property PACKAGE_PIN L1 [
set_property IOSTANDARD LVCMO


##7 segment display
#set_property PACKAGE_PIN W7
```



```
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
```

Import ISE project

- Add xdc-file



```
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
```

Project Summary
ISE Import Log
Basys3_Master.xdc *
:uments/Digitale1/projects/oef1_1_viv/oef1_1_viv.srcs/constrs_1/imports/Basys3_info/Basys3_Master.xdc
Save File Ctrl+S
Save File As...
Undo Ctrl+Z
Redo Ctrl+Shift+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Duplicate Selection Ctrl+D
Show Find Ctrl+F
Replace... Ctrl+R
Indent Selection Tab
Unindent Selection Shift+Tab
Toggle Line Comments Ctrl+Slash
Toggle Block Comments Ctrl+Shift+Slash
Select All Ctrl+A
Toggle Column Selection Ctrl+Back Slash

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
```

Design Methods

There are three design methods in VHDL

1. Dataflow design
2. Behavioural design
3. Structural design

Dataflow design

```
C<=A xor B;
```

- Defines direct relation (single to couple of gates) between output and input
- Sequence not important (parallelism)
- Asynchronous
- Difficult for complex functions
- Difficult to understand

*Lab 0

Make a component which connects all switches to all LEDs

Lab 0 - solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
   Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
          led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

Begin

led <= sw;

end Behavioral;
```

Lab 0 – solution - xdc

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
```

Lab 1

Make a logic components to test all the gates

- Led 7 = SW1 and SW2 and SW3
- Led 6 = SW1 or SW2 or SW3
- Led 5 = SW1 and SW2
- Led 4 = SW1 nand SW2
- Led 3 = SW1 or SW2
- Led 2 = SW1 nor SW2
- Led 1 = SW1 xor SW2
- Led 0 = SW1 xnor SW2

Lab 1 - solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity gates is
			port(
					sw1 : in STD_LOGIC;
					sw2 : in STD_LOGIC;
					sw3 : in STD_LOGIC;
					led : out STD_LOGIC_VECTOR(7 downto 0)
				);
end gates;

architecture behavior of gates is
begin
			led(7) <= sw1 and sw2 and sw3;
			led(6) <= sw1 or sw2 or sw3;
			led(5) <= sw1 and sw2;
			led(4) <= sw1 nand sw2;
			led(3) <= sw1 or sw2;
			led(2) <= sw1 nor sw2;
			led(1) <= sw1 xor sw2;
			led(0) <= sw1 xnor sw2;
end behavior;
```

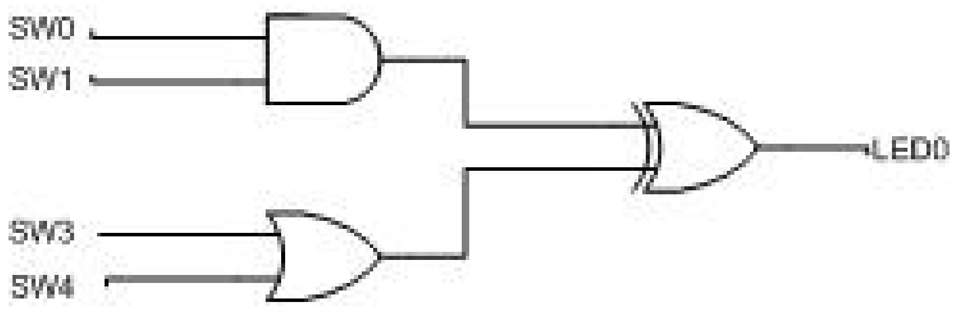
Parallellism:
Statements run concurrent to each other

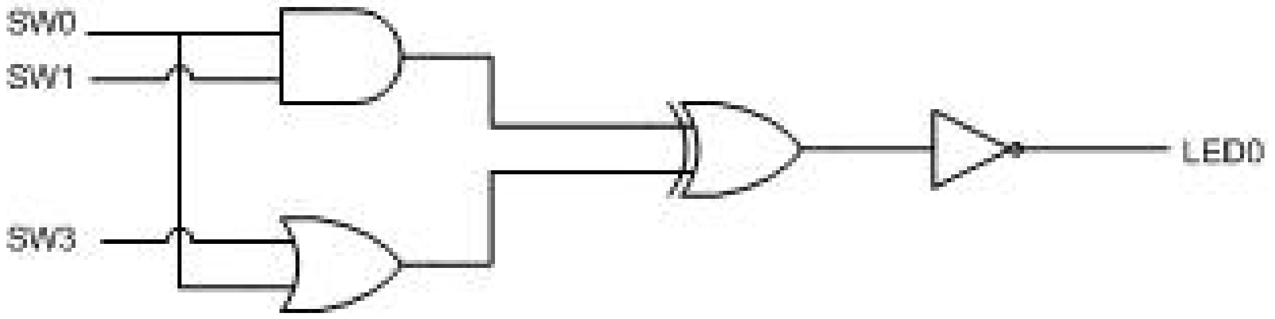
Lab 1 – solution - xdc

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw1}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw1}]
set_property PACKAGE_PIN V16 [get_ports {sw2}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw2}]
set_property PACKAGE_PIN W16 [get_ports {sw3}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw3}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
```

*Lab 2

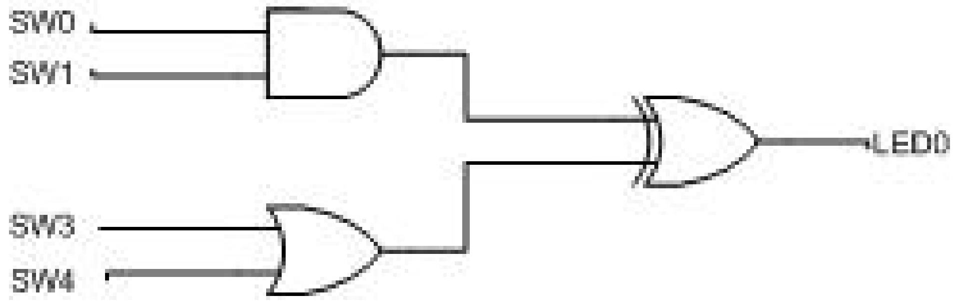
Make logic components to implement these schematics





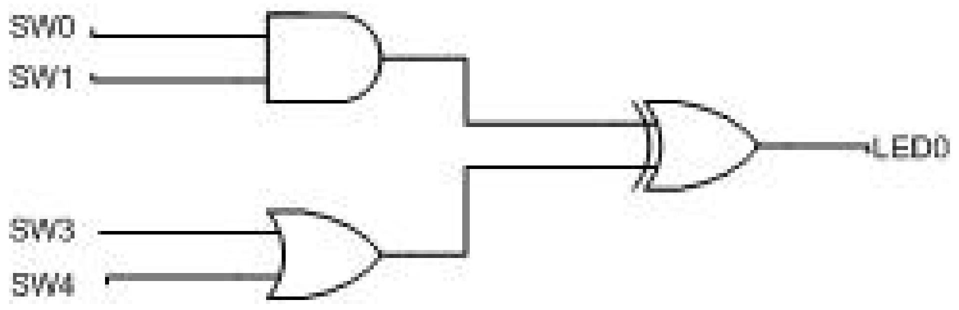
Lab 2 - solutions

```
entity lab2_1 is
   Port (        sw0 : in  STD_LOGIC;
                  sw1 : in  STD_LOGIC;
                 sw2 : in  STD_LOGIC;
                 sw3 : in  STD_LOGIC;
                 led0 : out  STD_LOGIC);
end lab2_1;

architecture Behavioral of lab2_1 is

begin
                  led0 <= (sw0 and sw1) xor (sw2 or sw3);

end Behavioral;
```



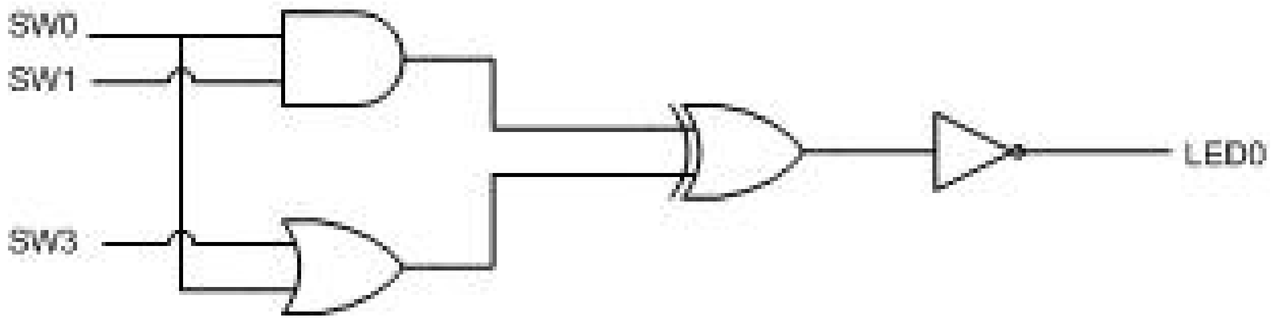
Lab 2 – solutions - xdc

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw0}]
                set_property IOSTANDARD LVCMOS33 [get_ports {sw0}]
set_property PACKAGE_PIN V16 [get_ports {sw1}]
                set_property IOSTANDARD LVCMOS33 [get_ports {sw1}]
set_property PACKAGE_PIN W16 [get_ports {sw2}]
                set_property IOSTANDARD LVCMOS33 [get_ports {sw2}]
set_property PACKAGE_PIN W17 [get_ports {sw3}]
                set_property IOSTANDARD LVCMOS33 [get_ports {sw3}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {led0}]
                set_property IOSTANDARD LVCMOS33 [get_ports {led0}]
```



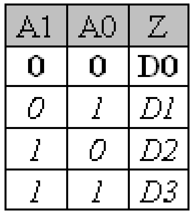
Lab 2 - solutions

```
entity lab2_2 is
   Port (       sw0 : in  STD_LOGIC;
                 sw1 : in  STD_LOGIC;
                sw3 : in  STD_LOGIC;
                led0 : out  STD_LOGIC);
end oef2_2;

architecture Behavioral of lab2_2 is

begin
                 led0 <= not ( (sw0 and sw1) xor (sw0 or sw3));

end Behavioral;
```

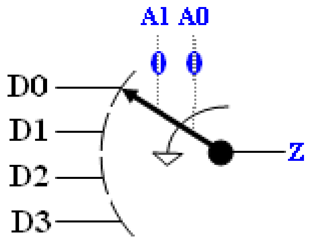


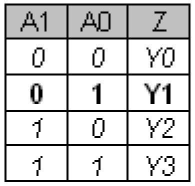
Combinatorial Logic

Multiplexer - principle

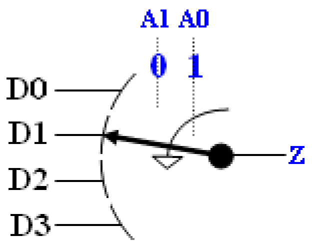


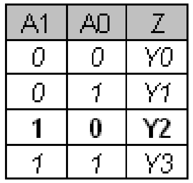
A1	A0	Z
0	**0**	**D0**
0	1	D1
1	0	D2
1	1	D3



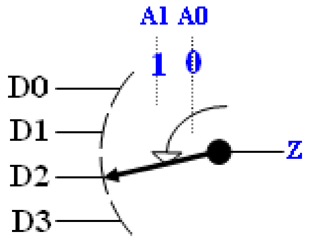


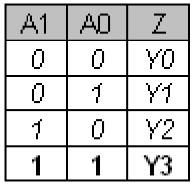
A1	A0	Z
0	0	Y0
0	**1**	**Y1**
1	0	Y2
1	1	Y3



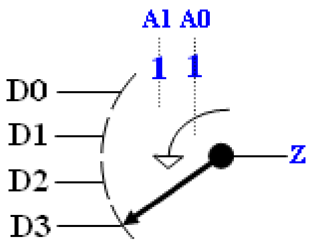


A1	A0	Z
0	0	Y0
0	1	Y1
1	**0**	**Y2**
1	1	Y3





A1	A0	Z
0	0	Y0
0	1	Y1
1	0	Y2
1	**1**	**Y3**



Lab 3

4 to 1 MUX

- D0 when A1A0 is “00”
- D1 when A1A0 is “01”
- D2 when A1A0 is “10”
- D3 when A1A0 is “11”

Lab 3 - solutions

4 to 1 MUX – with gates => increasingly complicated

```
entity mux41 is
            port(
                        d : in STD_LOGIC_VECTOR(3 downto 0);
                        a : in STD_LOGIC_VECTOR(1 downto 0);
                        z : out STD_LOGIC
              );
end mux41;

architecture behavior of mux41 is

begin
z <= (not a(1) and not a(0) and d(0))
  or (not a(1) and     a(0) and d(1))
  or (    a(1) and not a(0) and d(2))
  or (    a(1) and     a(0) and d(3));

end behavior;
```

Lab 3 - solutions

4 to 1 MUX – with gates => xdc

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {d[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {d[0]}]
set_property PACKAGE_PIN V16 [get_ports {d[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {d[1]}]
set_property PACKAGE_PIN W16 [get_ports {d[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {d[2]}]
set_property PACKAGE_PIN W17 [get_ports {d[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {d[3]}]
set_property PACKAGE_PIN W15 [get_ports {a[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
set_property PACKAGE_PIN V15 [get_ports {a[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {z}]
        set_property IOSTANDARD LVCMOS33 [get_ports {z}]
```

Design Methods

Behavioural design

Processes

```
process(sensitivity list: signal-names)
begin
        -- sequential code:
        e.g.:      if (condition) then
                           action1;
                   else
                           action2;
                   end if;
end process;
```

Behavioural design

Processes

- **If** (condition) **then** … **elsif** (condition) **then** … **else** … **end if**;
- **Case** (signal_name) **is**

  when condition 1 => …;

  when condition 2 => …;

  end case;
- **For** signal_name **in** start_value **to** stop_value **loop**

  …;

  end loop;

Tempus

Behavioural design

- **Sequence important**
- Asynchronous
- Divides complex functions in manageable parts
- Easier to understand
- For combinatorial: **all input signals in the sensitivity list**

*Lab 3 - solutions

4 to 1 MUX – with case statement

```
architecture behavior of mux41 is

begin
   p1: process(d, a) begin
               case a is
                              when "00" => z <= d(0);
                              when "01" => z <= d(1);
                              when "10" => z <= d(2);
                              when "11" => z <= d(3);
                              when others => z <= d(0);
               end case;
  end process;
end behavior;
```

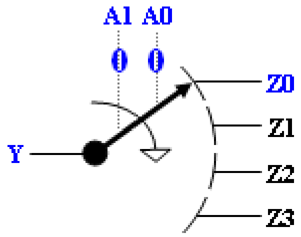
1 bit: ‘0’ or ‘1’:
single quotes

a vector “00” / “1001”:
double quotes

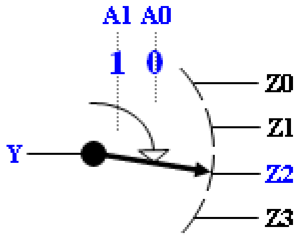
Combinatorial Logic

Demultiplexer - principle

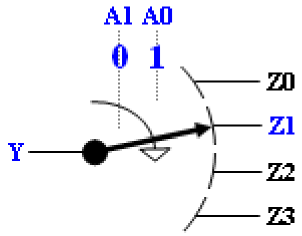
A1	A0	Z3	Z2	Z1	Z0
0	0	**0**	**0**	**0**	**Y**
0	1	0	0	Y	0
1	0	0	Y	0	0
1	1	Y	0	0	0



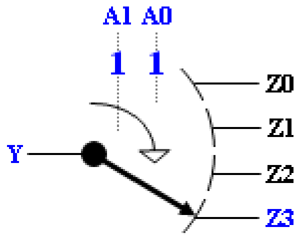
A1	A0	Z3	Z2	Z1	Z0
0	0	0	0	0	Y
0	1	0	0	Y	0
1	0	**0**	**Y**	**0**	**0**
1	1	Y	0	0	0



A1	A0	Z3	Z2	Z1	Z0
0	0	0	0	0	Y
0	1	**0**	**0**	**Y**	**0**
1	0	0	Y	0	0
1	1	Y	0	0	0



A1	A0	Z3	Z2	Z1	Z0
0	0	0	0	0	Y
0	1	0	0	Y	0
1	0	0	Y	0	0
1	1	**Y**	**0**	**0**	**0**



*Lab 4

1 to 4 DEMUX

- Y is Z0 when A1A0 is “00”
- Y is Z1 when A1A0 is “01”
- Y is Z2 when A1A0 is “10”
- Y is Z3 when A1A0 is “11”

⇒ Difficulty: what about the other outputs?

⇒ Do not leave them un-assigned: otherwise latching (= unintended memory synthesis)

Use concatenate: ‘&’ => binds different signals together
e.g. Z <= ‘0’ & ’0’ & ’0’ & Y;

Lab 4 - solutions

1 to 4 DEMUX

```
entity demux14 is
		port(
				y : in STD_LOGIC;
				a : in STD_LOGIC_VECTOR(1 downto 0);
				z : out STD_LOGIC_VECTOR(3 downto 0)
		  );
end demux14;
```

Tempus

Lab 4 – solutions - concatenate

1 to 4 DEMUX

```
architecture behavior of demux14 is

begin
p1: process(y, a) begin
        case a is
                when "00" => z <= ‘0’&’0’&’0’&y;
                when "01" => z <= ‘0’&’0’&y&’0’;
                when "10" => z <= ‘0’&y&’0’&’0’;
                when "11" => z <= y&’0’&’0’&’0’;
                when others => z <= ‘0’&’0’&’0’&’0’;
        end case;
 end process;

end behavior;
```

1 bit: ‘0’ or ‘1’:
single quotes
a vector “00” / “1001”:
double quotes

&: concatenate: binds signals together

Lab 4 – solutions - aggregate

1 to 4 DEMUX

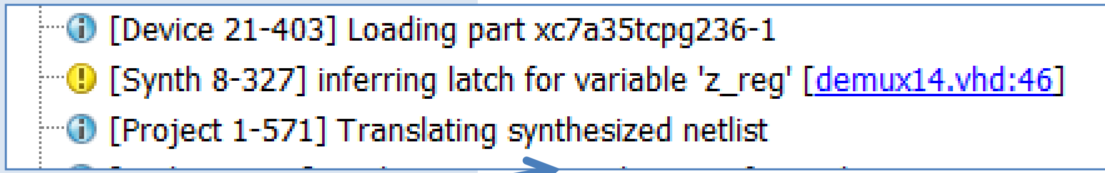
```
architecture behavior of demux14 is

begin
p1: process(y, a) begin
        case a is
                when "00" => z <= ('0','0','0',y);
                when "01" => z <= ('0','0',y,'0');
                when "10" => z <= ('0',y,'0','0');
                when "11" => z <= (y,'0','0','0');
                when others => z <= "0000";
        end case;
 end process;

end behavior;
```

(x,x,x,x): aggregate: binds signals together

Lab 4 – solutions - wrong

1 to 4 DEMUX

```
architecture Behavioral of demux14 is

begin

p1: process(y, a) begin
        case a is
                when "00" => z(0) <= y;
                when "01" => z(1) <= y;
                when "10" => z(2) <= y;
                when "11" => z(3) <= y;
                when others => z <= "0000";
        end case;
 end process;

end Behavioral;
```



```
[Device 21-403] Loading part xc7a35tcpg236-1
[Synth 8-327] inferring latch for variable 'z_reg' [demux14.vhd:46]
[Project 1-571] Translating synthesized netlist
```

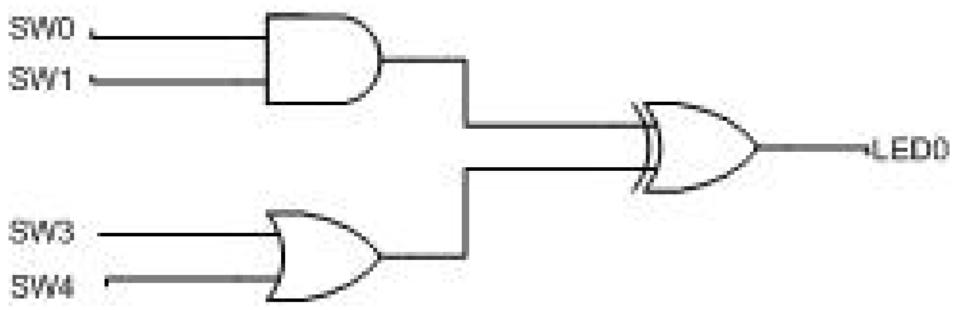
Design Methods

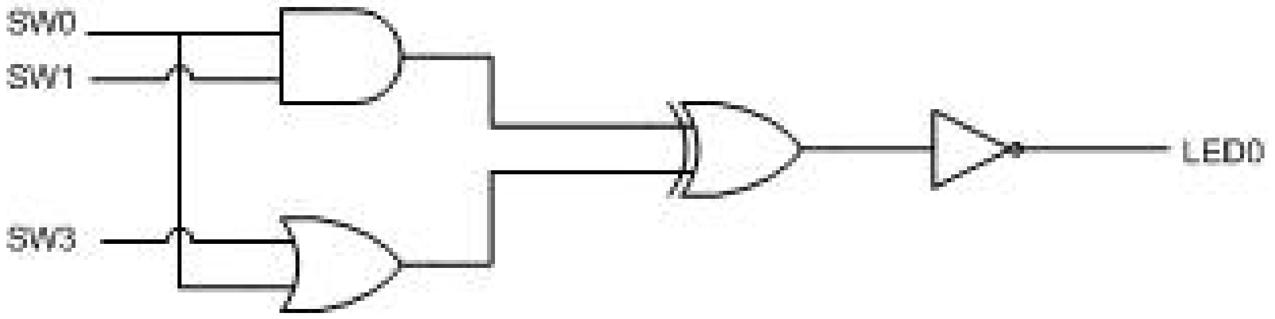
Behavioural design

- Inter-process communication is done with signals
- Signals are declared between “architecture” and “begin”
- Design can be structured with different processes, connected with signals

Lab 5

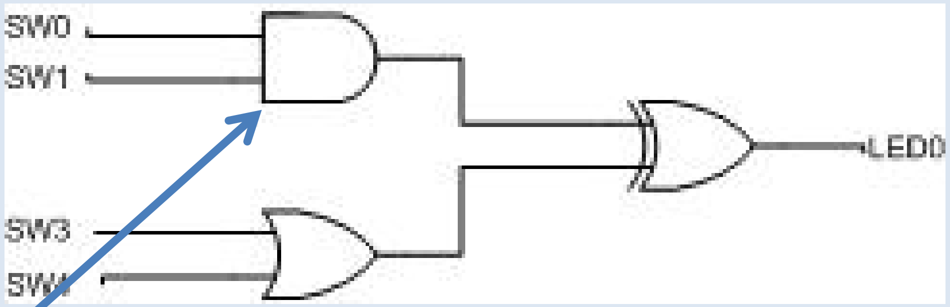
Make logic components to implement these schematics with the use of processes and signals





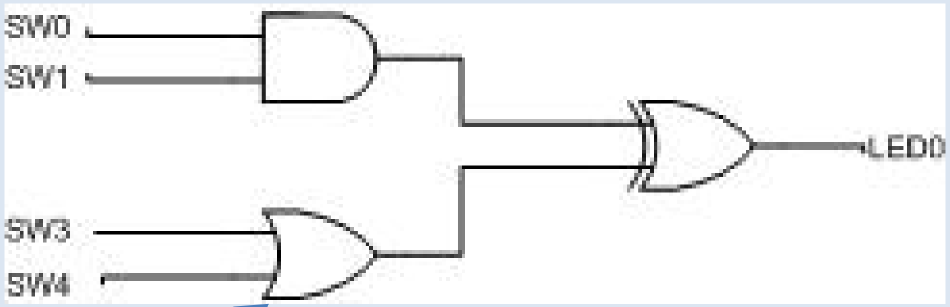
Lab 5 - solutions

```
entity lab5_1 is
Port (          sw0 : in STD_LOGIC;
                sw1 : in STD_LOGIC;
                sw2 : in STD_LOGIC;
                sw3 : in STD_LOGIC;
                led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
                signal and_sig: std_logic;
                signal or_sig: std_logic;
begin
                and_gt: process (sw0, sw1) begin
                                and_sig <= sw0 and sw1;
                end process;
                or_gt: process (sw2, sw3) begin
                                or_sig <= sw2 or sw3;
                end process;
                xor_gt: process (or_sig, and_sig) begin
                                led <= or_sig xor and_sig;
                end process;
end Behavioral;
```



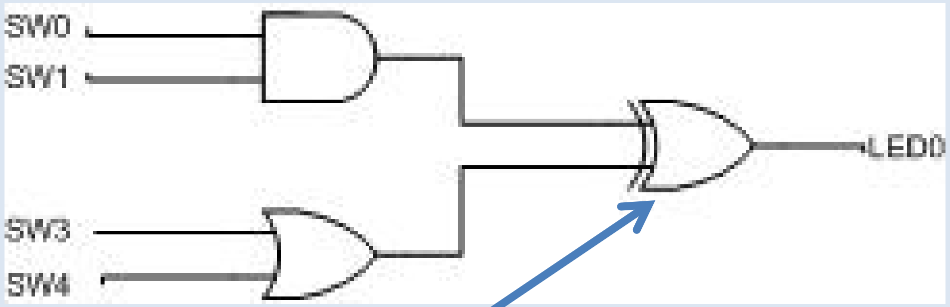
Lab 5 - solutions

```
entity lab5_1 is
Port (          sw0 : in STD_LOGIC;
                sw1 : in STD_LOGIC;
                sw2 : in STD_LOGIC;
                sw3 : in STD_LOGIC;
                led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
                signal and_sig: std_logic;
                signal or_sig: std_logic;
begin
                and_gt: process (sw0, sw1) begin
                                and_sig <= sw0 and sw1;
                end process;
                or_gt: process (sw2, sw3) begin
                                or_sig <= sw2 or sw3;
                end process;
                xor_gt: process (or_sig, and_sig) begin
                                led <= or_sig xor and_sig;
                end process;
end Behavioral;
```



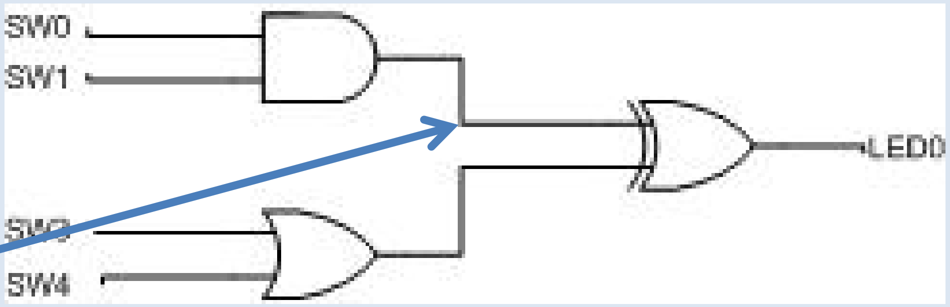
Lab 5 - solutions

```
entity lab5_1 is
Port (          sw0 : in STD_LOGIC;
                sw1 : in STD_LOGIC;
                sw2 : in STD_LOGIC;
                sw3 : in STD_LOGIC;
                led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
                signal and_sig: std_logic;
                signal or_sig: std_logic;
begin
                and_gt: process (sw0, sw1) begin
                                and_sig <= sw0 and sw1;
                end process;
                or_gt: process (sw2, sw3) begin
                                or_sig <= sw2 or sw3;
                end process;
                xor_gt: process (or_sig, and_sig) begin
                                led <= or_sig xor and_sig;
                end process;
end Behavioral;
```



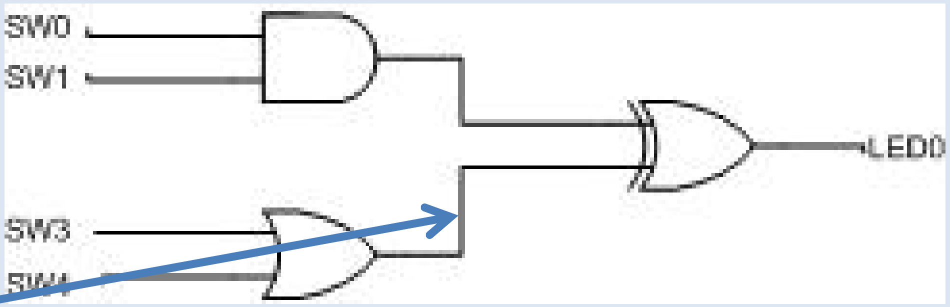
Lab 5 - solutions

```
entity lab5_1 is
Port (          sw0 : in STD_LOGIC;
                sw1 : in STD_LOGIC;
                sw2 : in STD_LOGIC;
                sw3 : in STD_LOGIC;
                led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
                signal and_sig: std_logic;
                signal or_sig: std_logic;
begin
                and_gt: process (sw0, sw1) begin
                                and_sig <= sw0 and sw1;
                end process;
                or_gt: process (sw2, sw3) begin
                                or_sig <= sw2 or sw3;
                end process;
                xor_gt: process (or_sig, and_sig) begin
                                led <= or_sig xor and_sig;
                end process;
end Behavioral;
```



Lab 5 - solutions

```
entity lab5_1 is
Port (          sw0 : in STD_LOGIC;
                sw1 : in STD_LOGIC;
                sw2 : in STD_LOGIC;
                sw3 : in STD_LOGIC;
                led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
                signal and_sig: std_logic;
                signal or_sig: std_logic;
begin
                and_gt: process (sw0, sw1) begin
                                and_sig <= sw0 and sw1;
                end process;
                or_gt: process (sw2, sw3) begin
                                or_sig <= sw2 or sw3;
                end process;
                xor_gt: process (or_sig, and_sig) begin
                                led <= or_sig xor and_sig;
                end process;
end Behavioral;
```



Design Methods

Structural design

- Design can be done with the use of components
- Different components are connected with signals
- Both components and signals are declared in the *declarative part* between "architecture" and "begin"
- In the *definition part* the actual component is used, *instantiated*
- Divide problems into sub-problems
- Reuse – one component can be used several times

  ⇒ the difference in actual component used is made in instance name and connected signals – naming of ports remain the same

Design Methods

Structural design

```
--component declarations
        component <entity-name>
        port (
                <signal-names : mode signal-type>
        );
        end component;
…

--signals to connect blocks
        signal-names : mode signal-type;
begin
<component-name> : <entity-name>
                port map (
                        <signal-mapping>
                );
…
```

Design Methods

Structural design

- Entity declaration of the component used and the component declaration in the top-module are very similar

⇒ The entity declaration of the sub-module can be used for the component declaration in the top-module

⇒ The component declaration of the top-module can be used for the entity declaration of the sub-module

- Copy – paste
- Change “component” to “entity” and visa versa

*Lab 6

Use a top-module to connect a MUX and a DEMUX

- Add VHDL-files with components you want to use
- Declare components
- Declare signals
- Instantiate – use components

Lab 6 - solution

```
entity muxdemux is
		port(
				d : in STD_LOGIC_VECTOR(3 downto 0);
				a : in STD_LOGIC_VECTOR(1 downto 0);
				z : out STD_LOGIC_VECTOR(3 downto 0)
		);
end muxdemux;
```

Lab 6 - solution

```
architecture behavior of muxdemux is
component mux41 is
          port(
                    d : in STD_LOGIC_VECTOR(3 downto 0);
                    a : in STD_LOGIC_VECTOR(1 downto 0);
                    z : out STD_LOGIC
            );
end component mux41;
component demux14 is
          port(
                    y : in STD_LOGIC;
                    a : in STD_LOGIC_VECTOR(1 downto 0);
                    z : out STD_LOGIC_VECTOR(3 downto 0)
            );
end component demux14;

signal  y : STD_LOGIC;
```

Copy – paste
Entity => component

Signal name *can* be the same as the port name, but is *not obliged*

Lab 6 - solution

```
Begin

Mux1: mux41
        port map(d => d, a => a, z => y);

Demux1: demux14
        port map(y => y, a => a, z => z);

end behavior;
```

- *Instantiate* – use component
- It is possible to use a second Mux2 if desired

- Connect ports, *left*, to signals, *right*
- Signals can be signals declared in the *entity declaration* **or** signals declared in the *architecture*
- Ports and signals can have the same name

Lab 6 – solution - xdc

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {d[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {d[0]}]
set_property PACKAGE_PIN V16 [get_ports {d[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {d[1]}]
set_property PACKAGE_PIN W16 [get_ports {d[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {d[2]}]
set_property PACKAGE_PIN W17 [get_ports {d[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {d[3]}]
set_property PACKAGE_PIN W15 [get_ports {a[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
set_property PACKAGE_PIN V15 [get_ports {a[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]
```

Lab 6 – solution - xdc

```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {z[0]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {z[0]}]
set_property PACKAGE_PIN E19 [get_ports {z[1]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {z[1]}]
set_property PACKAGE_PIN U19 [get_ports {z[2]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {z[2]}]
set_property PACKAGE_PIN V19 [get_ports {z[3]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {z[3]}]
```

Simulation

Testbench

How do you test a design?
Answer: implement the hardware on an FPGA-board and test all input options.
$\Rightarrow$ This process can be automated in a testbench

Why?

Synthesis (= the process of translating VHDL to the proper hardware) consumes a lot of time, up to hours.
$\Rightarrow$ Solution: simulate first, synthesize later

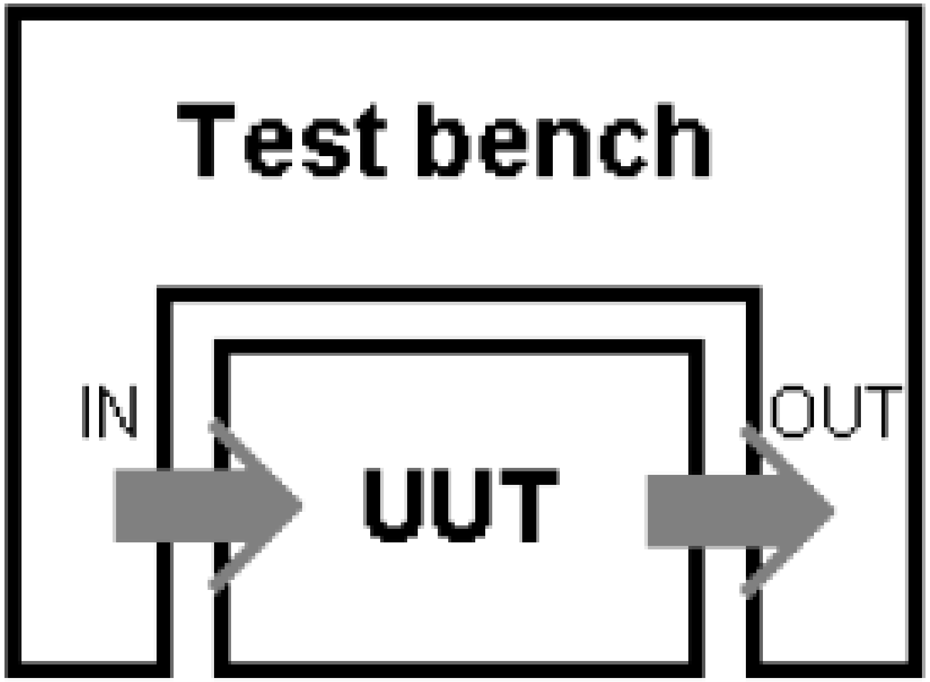
Simulation

Testbench

- A testbench is a VHDL module with one component, the component we want to test.
- It has no interface with the outside world: empty entity
  1. The component is declared
  2. The input and output signals are declared
  3. The component is *instantiated*
  4. The input and output signals are connected to the instantiated component
  5. In two processes all input signals are asserted at a certain moment of time
     - One process for the clock
     - One process for the other inputs
  6. The output is analyzed in a waveform or in a output txt-file.
- It is possible to look at internal signals

Simulation

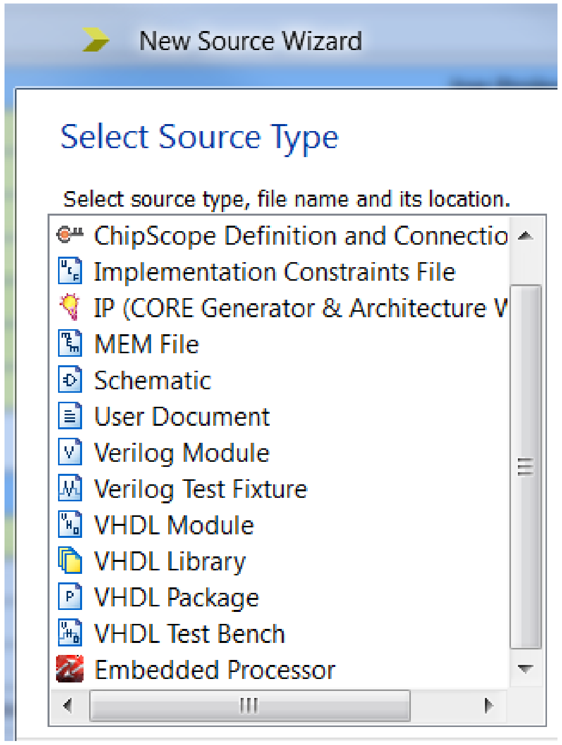
Testbench



Lab 7 - ISE

Simulate Lab 0 – 1 – 2 – 3 - 4

- Open the project
- Click on simulation in the left upper panel
- Add new source to the project
- Write a testbench to
  - simulate the clock
  - simulate all possible input combinations

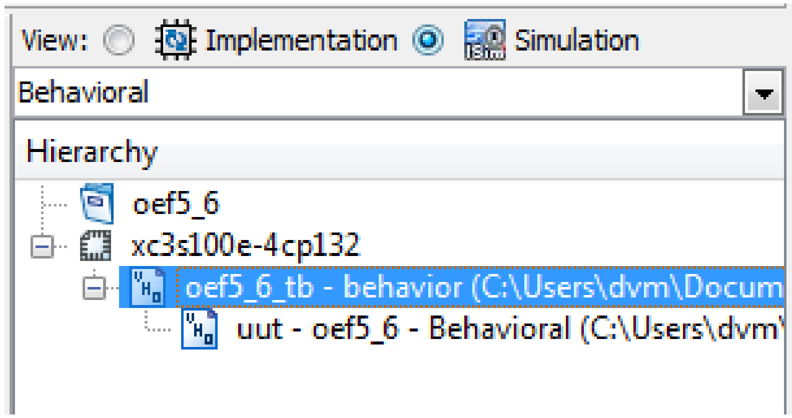


New Source Wizard

Select Source Type

Select source type, file name and its location.

- ChipScope Definition and Connectio
- Implementation Constraints File
- IP (CORE Generator & Architecture V
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor



View: ( ) Implementation [✓ Simulation]

Behavioral

Hierarchy

- oef5_6
- xc3s100e-4cp132
  - oef5_6_tb - behavior (C:\Users\dvm\Docum
    - uut - oef5_6 - Behavioral (C:\Users\dvm\

Tempus

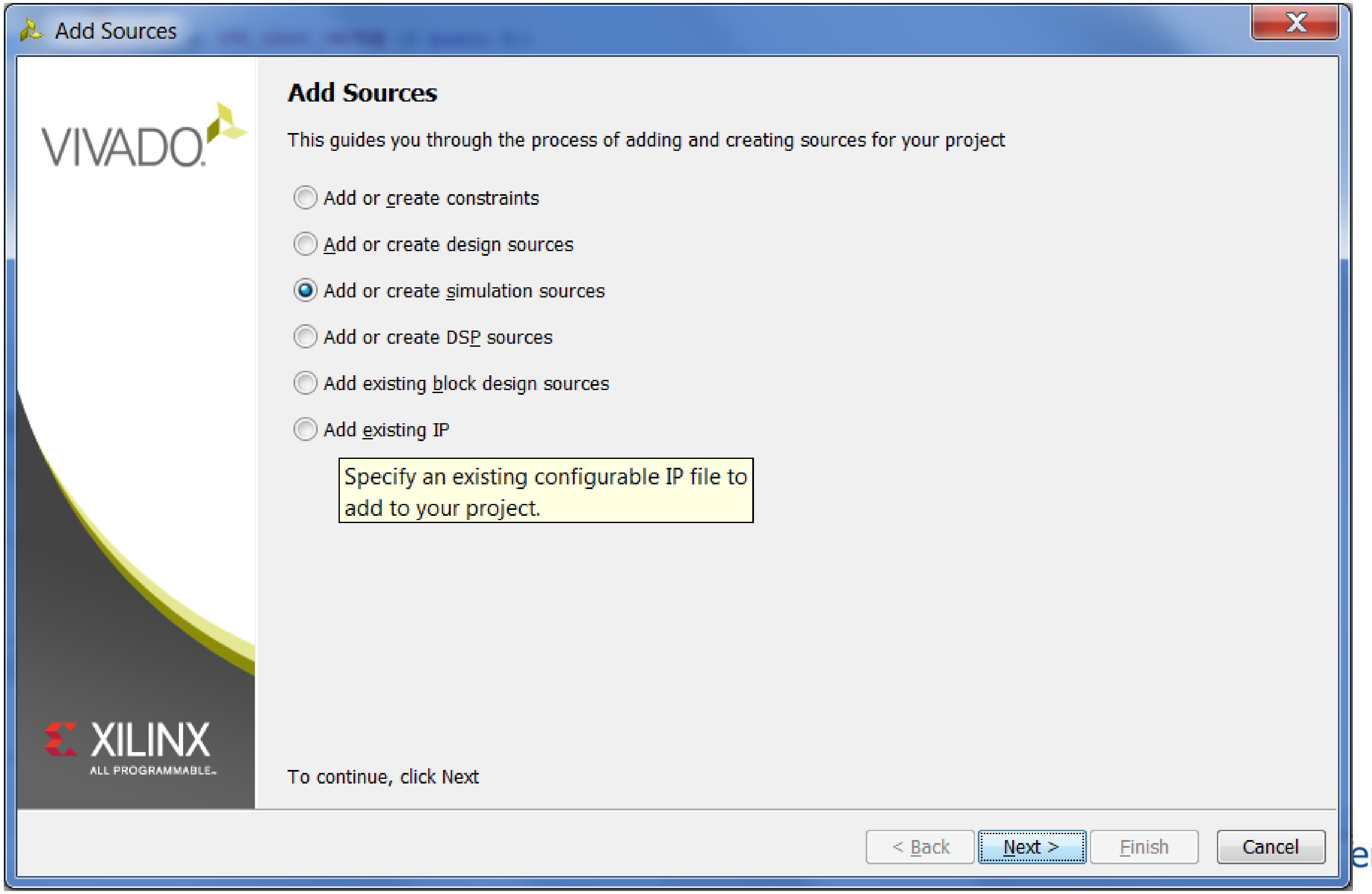
Lab 7 - Vivado

Simulate Lab 0 – 1 – 2 – 3 - 4

- Look at slide 80 and further
- Open the project
- Right-click on simulation in the left upper panel
- Add new source to the project
- Add or create simulation sources
- Write a testbench
  - Add the component to test
  - Add the signals to connect to the component
  - simulate the clock (not used now)
  - simulate all possible input combinations
- Push "Run Simulation"

Lab 7 - Vivado

Simulate Lab 0 – 1 – 2 – 3 - 4



Add Sources

This guides you through the process of adding and creating sources for your project

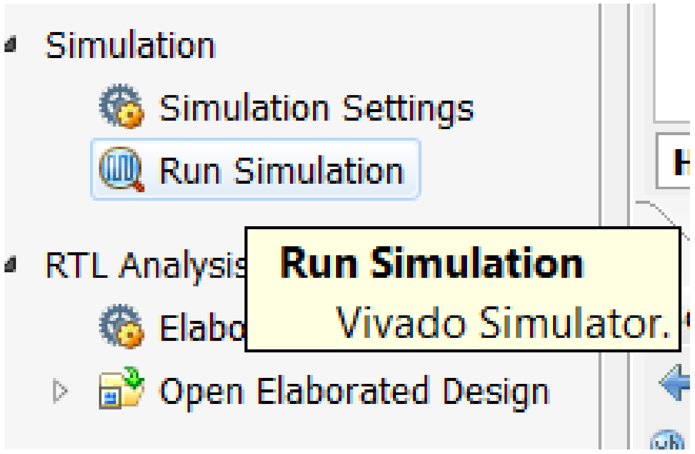
- ○ Add or create constraints
- ○ Add or create design sources
- ● Add or create simulation sources
- ○ Add or create DSP sources
- ○ Add existing block design sources
- ○ Add existing IP

Specify an existing configurable IP file to add to your project.

To continue, click Next

Lab 7 - Vivado

Simulate Lab 0 – 1 – 2 – 3 - 4



Run Simulation

Vivado Simulator.

Lab 7 – solution Lab 0

```
ENTITY sw_led_TB IS
END sw_led_TB;

ARCHITECTURE behavior OF sw_led_TB IS

-- Component Declaration for the Unit Under Test (UUT)

  COMPONENT sw_led
  PORT(
     sw : IN  std_logic_vector(3 downto 0);
     led : OUT  std_logic_vector(3 downto 0)
    );
  END COMPONENT;

--Inputs
  signal sw : std_logic_vector(3 downto 0) := (others => '0');

--Outputs
  signal led : std_logic_vector(3 downto 0);
```

Empty entity

Component declaration
Look at entity sw_led

Signals declaration
Same name as ports

Lab 7 – solution Lab 0

```
BEGIN

-- Instantiate the Unit Under Test (UUT)
 uut: sw_led PORT MAP (
       sw => sw,
       led => led
     );
```

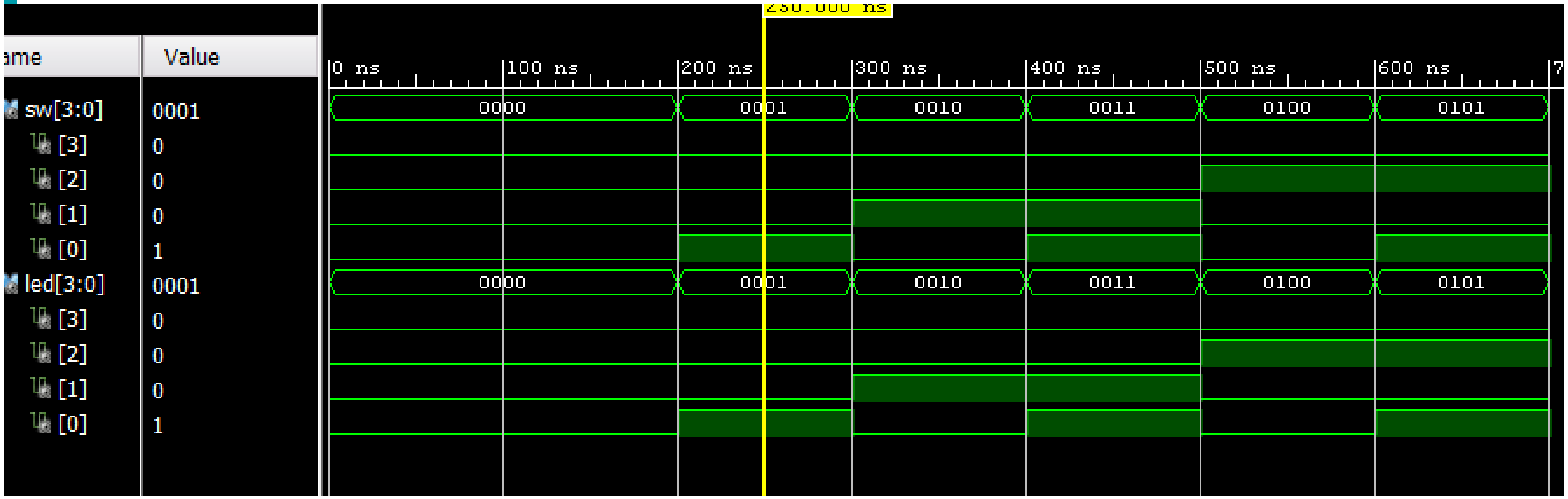
Instantiate component
uut= name
connect signals to ports

Lab 7 – solution Lab 0

```
-- Stimulus process
   stim_proc: process
   begin
              wait for 100 ns;
              sw <= “0000”;
              wait for 100 ns;
              sw <= “0001”;
              wait for 100 ns;
              sw <= “0010”;
              wait for 100 ns;
              sw <= “0011”;
              wait for 100 ns;
              sw <= “0100”;
              wait for 100 ns;
              sw <= “0101”;
              wait for 100 ns;
              sw <= “0110”;
              wait;
   end process;
```

Tempus

Lab 7 – solution Lab 0



```
sw[3:0]   0001   0000 | 0001 | 0010 | 0011 | 0100 | 0101
  [3]     0
  [2]     0
  [1]     0
  [0]     1
led[3:0]  0001   0000 | 0001 | 0010 | 0011 | 0100 | 0101
  [3]     0
  [2]     0
  [1]     0
  [0]     1
```

Lab 7 – solution Lab 0 - better

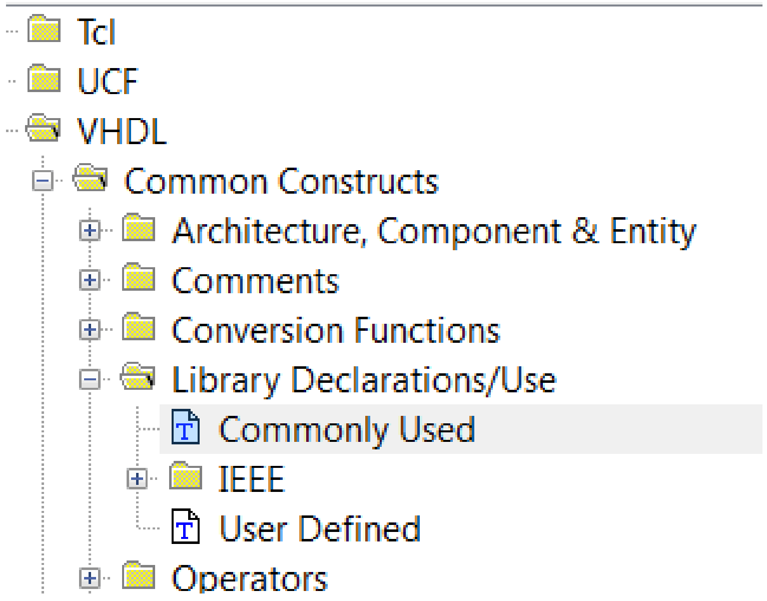
```
BEGIN

-- Instantiate the Unit Under Test (UUT)
   uut: sw_led PORT MAP (
          sw => sw,
          led => led
        );
 -- Stimulus process
   stim_proc: process
   begin
                 wait for 100 ns;
                 for i in 0 to 20 loop
                                 wait for 40 ns;
                                 sw <= sw + 1;
                 end loop;
                 wait;
   end process;
```

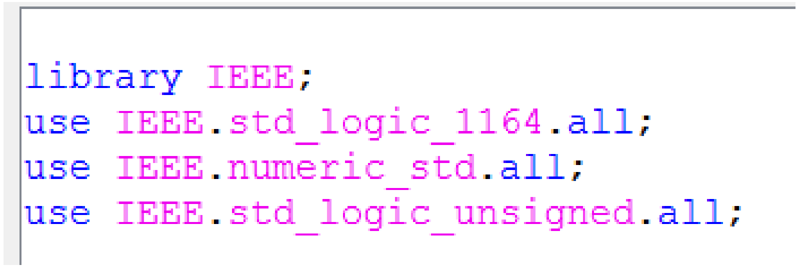
Problem:
+ - operant is not declared in the libraries
=> extra libraries needed

Lab 7 – solution Lab 0 - better

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
```

Edit => Language Templates…
VHDL => Common Constructs => Library Declarations/Use => Commonly Used





```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
```

Lab 7 – solution Lab 0 – variants

```
ARCHITECTURE behavior OF button_led_TB IS
…
--Inputs
   signal sw : std_logic_vector(3 downto 0) := (others => '0');

-- OR

   signal sw : std_logic_vector(3 downto 0);
…
BEGIN
…
```

What happens in simulation?

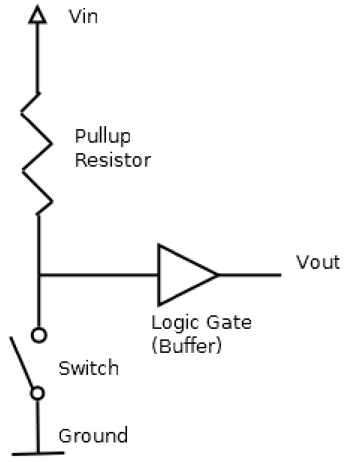
DesIRE

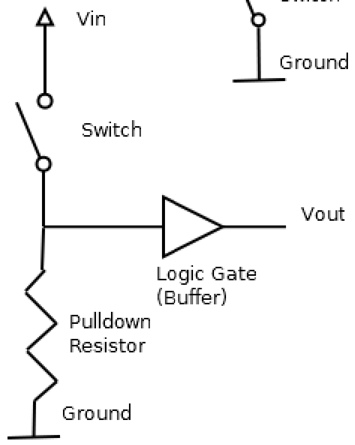
Std_ulogic

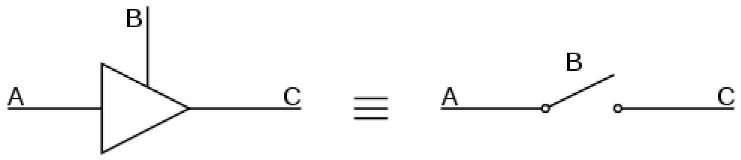
Possibilities for std_logic (not only ‘0’ and ‘1’)

```
type STD_ULOGIC is (
          ` U `,   -- uninitialized
          ` X `,   -- strong 0 or 1 (= unknown)
          ` 0 `,   -- strong 0
          ` 1 `,   -- strong 1
          ` Z `,   -- high impedance
          ` W `,   -- weak 0 or 1 (= unknown)
          ` L `,   -- weak 0
          ` H `,   -- weak 1
           ` - `,  -- don`t care);
```

By default first value in enumeration







Tempus

Lab 7 – solution Lab 1

```
entity gates_tb is
--  Port ( );
end gates_tb;

architecture Behavioral of gates_tb is

component gates is
    Port ( sw1 : in STD_LOGIC;
           sw2 : in STD_LOGIC;
           sw3 : in STD_LOGIC;
           led : out STD_LOGIC_VECTOR (7 downto 0));
end component gates;

signal led: std_logic_vector (7 downto 0);
signal sw: std_logic_vector (2 downto 0):= "000";
```

Empty entity

Component declaration

Signals declaration
sw is a vector for easy adding in the stimulation process

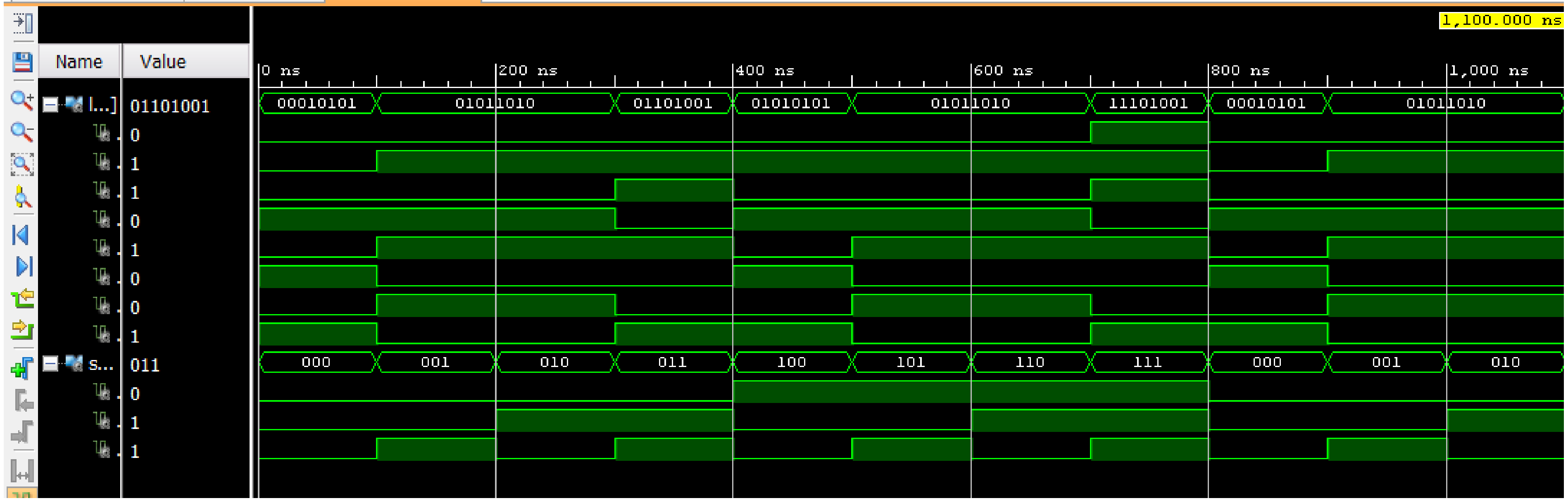
Lab 7 – solution Lab 1

```
begin

UUT: gates
Port map (sw1 => sw(0), sw2 => sw(1), sw3 => sw(2), led => led);

stim_proc: process begin

   for i in 0 to 10 loop
            wait for 100 ns;
            sw <= sw+1;
    end loop;

wait;


end process;

end Behavioral;
```

Connect individual elements of signal vector sw to ports sw1, sw2 en sw3

Only possible with sw a vector

Lab 7 – solution Lab 1



```
led(7) <= sw1 and sw2 and sw3;
led(6) <= sw1 or sw2 or sw3;
led(5) <= sw1 and sw2;
led(4) <= sw1 nand sw2;
led(3) <= sw1 or sw2;
led(2) <= sw1 nor sw2;
led(1) <= sw1 xor sw2;
led(0) <= sw1 xnor sw2;
```

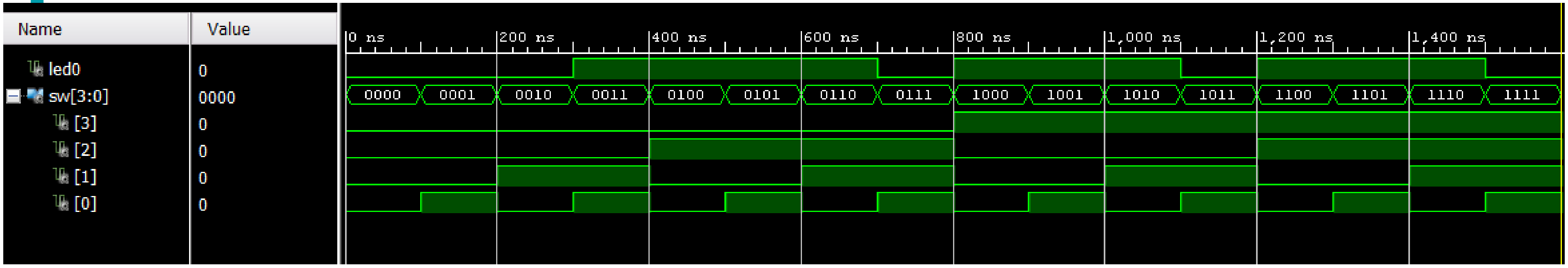
Lab 7 – solution Lab 2

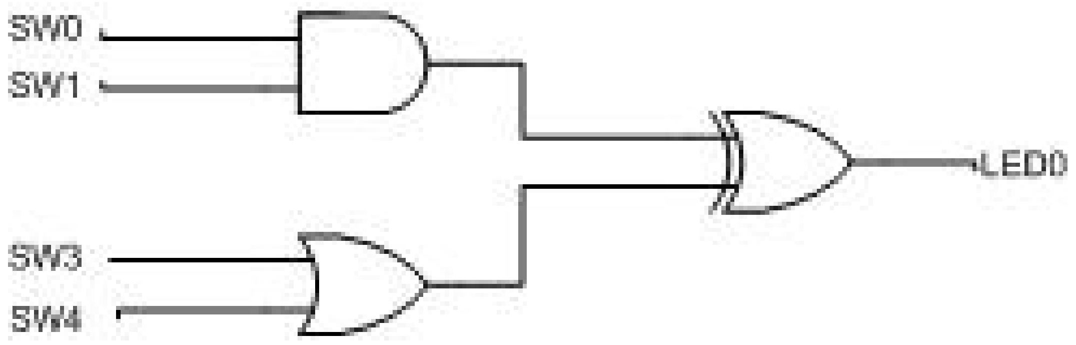
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity lab2_1_tb is
end lab2_1_tb;

architecture Behavioral of lab2_1_tb is

component Lab2_1 is
   Port ( sw0 : in STD_LOGIC;
          sw1 : in STD_LOGIC;
          sw2 : in STD_LOGIC;
          sw3 : in STD_LOGIC;
          led0 : out STD_LOGIC);
end component Lab2_1;

signal led0: std_logic;
signal sw: std_logic_vector (3 downto 0):= "0000";
```

Lab 7 – solution Lab 2

```
begin

UUT: Lab2_1
Port map (sw0 => sw(0), sw1 => sw(1), sw2 => sw(2), sw3 => sw(3), led0 => led0);

stim_process: process begin

   for i in 0 to 15 loop
                wait for 100 ns;
                sw <= sw+1;
    end loop;

wait;

end process;

end Behavioral;
```

Lab 7 – solution Lab 2





Lab 7 – solution Lab 3

```
entity mux41_tb is
--  Port ( );
end mux41_tb;

architecture Behavioral of mux41_tb is

component mux41 is
    Port ( d : in STD_LOGIC_VECTOR (3 downto 0);
           a : in STD_LOGIC_VECTOR (1 downto 0);
           z : out STD_LOGIC);
end component mux41;

signal d: STD_LOGIC_VECTOR (3 downto 0);
signal a: STD_LOGIC_VECTOR (1 downto 0);
signal z: STD_LOGIC;
```

Lab 7 – solution Lab 3

```
UUT: mux41
Port map (d => d, a => a, z => z);

stim_process: process begin

wait for 100ns;
d <= "0000";
a <= "00";

wait for 100ns;
d <= "0001";
a <= "00";

wait for 100ns;
d <= "0000";
a <= "00";
```

Address = 0
Change d(0) and see if this affects the output

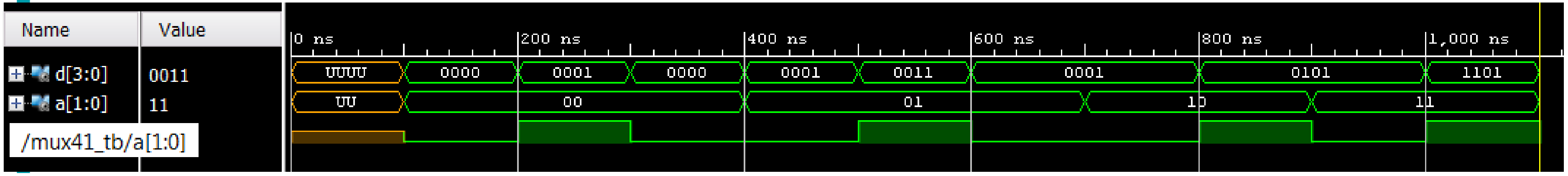
Lab 7 – solution Lab 3

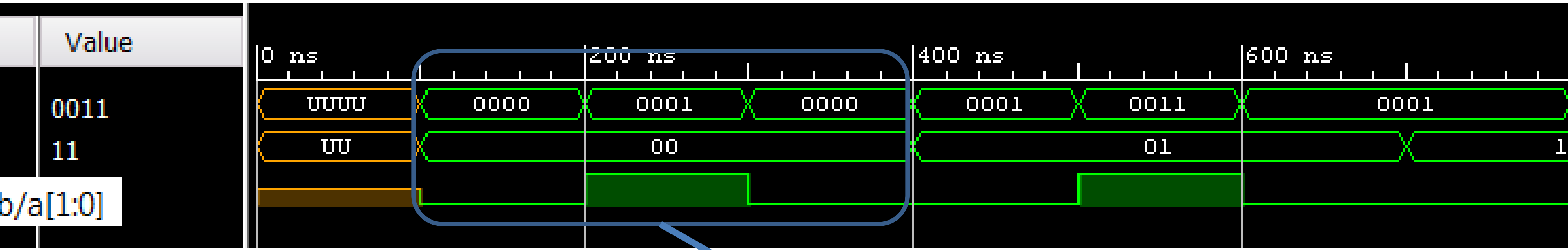
```
wait for 100ns;
d <= "0001";
a <= "01";

wait for 100ns;
d <= "0011";
a <= "01";

wait for 100ns;
d <= "0001";
a <= "01";

wait for 100ns;
d <= "0001";
a <= "10";

wait for 100ns;
d <= "0101";
a <= "10";
```

Lab 7 – solution Lab 3

```
wait for 100ns;
d <= "0101";
a <= "11";

wait for 100ns;
d <= "1101";
a <= "11";

wait for 100ns;
d <= "0011";
a <= "11";

wait;

end process;

end Behavioral;
```

Lab 7 – solution Lab 3





Address = 0
Change d(0) and see if this affects the output

Lab 7 – solution Lab 4

```
entity demux14_tb is
--  Port ( );
end demux14_tb;

architecture Behavioral of demux14_tb is

component demux14 is
    Port ( y : in STD_LOGIC;
           a : in STD_LOGIC_VECTOR (1 downto 0);
           z : out STD_LOGIC_VECTOR (3 downto 0));
end component demux14;

signal z: STD_LOGIC_VECTOR (3 downto 0);
signal a: STD_LOGIC_VECTOR (1 downto 0);
signal y: STD_LOGIC;
```

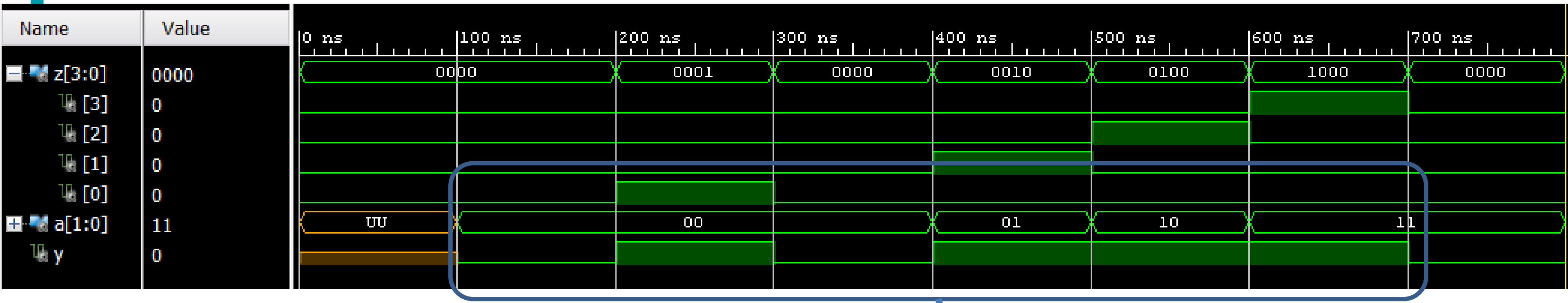
Lab 7 – solution Lab 4

```
begin

UUT: demux14
Port map (y => y, a => a, z => z);

stim_process: process begin

wait for 100 ns;
y <= '0';
a <= "00";
wait for 100 ns;
y <= '1';
a <= "00";
wait for 100 ns;
y <= '0';
a <= "00";
wait for 100 ns;
y <= '1';
a <= "01";
```

Lab 7 – solution Lab 4

```
wait for 100 ns;
y <= '1';
a <= "10";
wait for 100 ns;
y <= '1';
a <= "11";
wait for 100 ns;
y <= '0';
a <= "11";
wait for 100 ns;
wait;
end process;

end Behavioral;
```

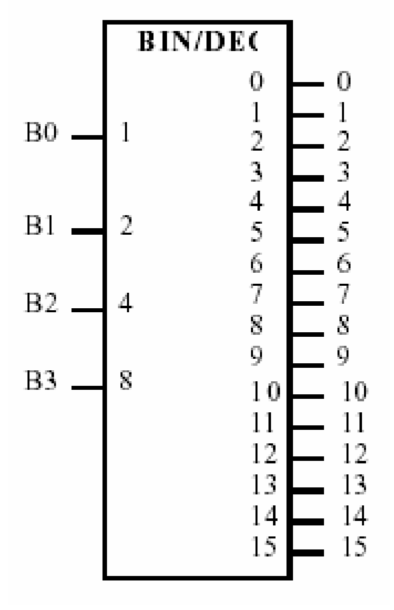
Lab 7 – solution Lab 4



Address changes
Change y and see if this affects the output.
Keep y = ‘1’ and change address.

Combinatorial Logic

BIN/DEC decoder



BIN / DEC

B3, B2 \ B1, B0	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$0 = \overline{B3}\,\overline{B2}\,\overline{B1}\,\overline{B0}$ $\quad 4 = \overline{B3}B2\overline{B1}\,\overline{B0}$ $\quad 8 = B3\overline{B2}\,\overline{B1}\,\overline{B0}$ $\quad 12 = B3B2\overline{B1}\,\overline{B0}$

$1 = \overline{B3}\,\overline{B2}\,\overline{B1}B0$ $\quad 5 = \overline{B3}B2\overline{B1}B0$ $\quad 9 = B3\overline{B2}\,\overline{B1}B0$ $\quad 13 = B3B2\overline{B1}B0$

$2 = \overline{B3}\,\overline{B2}B1\overline{B0}$ $\quad 6 = \overline{B3}B2B1\overline{B0}$ $\quad 10 = B3\overline{B2}B1\overline{B0}$ $\quad 14 = B3B2B1\overline{B0}$

$3 = \overline{B3}\,\overline{B2}B1B0$ $\quad 7 = \overline{B3}B2B1B0$ $\quad 11 = B3\overline{B2}B1B0$ $\quad 15 = B3B2B1B0$

Lab 8

Make a 3 to 8 Decoder in VHDL

- With gates
- With a for loop
- Simulate

Lab 8 – solution – gates

```
entity decode38 is
                port(
                                a : in STD_LOGIC_VECTOR(2 downto 0);
                                y : out STD_LOGIC_VECTOR(7 downto 0)
                   );
end decode38;

architecture behavior of decode38 is

begin
               y(0) <= not a(2) and not a(1) and not a(0);
               y(1) <= not a(2) and not a(1) and a(0);
               y(2) <= not a(2) and a(1) and not a(0);
               y(3) <= not a(2) and a(1) and a(0);
               y(4) <= a(2) and not a(1) and not a(0);
               y(5) <= a(2) and not a(1) and a(0);
               y(6) <= a(2) and a(1) and not a(0);
               y(7) <= a(2) and a(1) and a(0);
end decode38a;
```

Lab 8 – solution – for loop

```
architecture behavior of decode38 is
begin
  process(a)
  variable j: integer;
  begin
        j := conv_integer(a);
        for i in 0 to 7 loop
          if(i = j) then
                y(i) <= '1';
          else
                y(i) <= '0';
          end if;
        end loop;
  end process;
end behavior;
```

Converts binary to integer

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
```

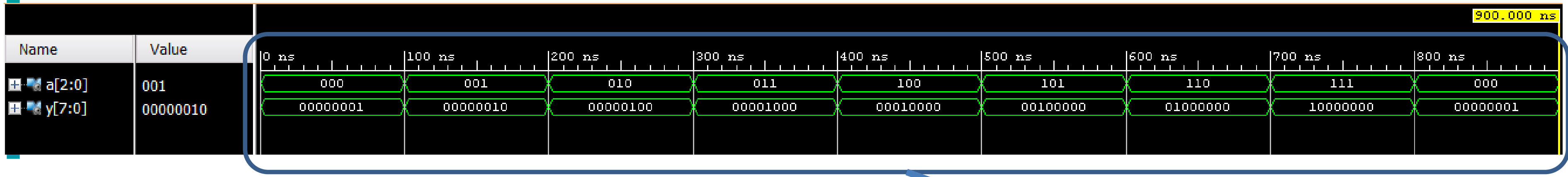
Lab 8 – solution – testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity decode38_tb is
end decode38_tb;

architecture Behavioral of decode38_tb is

component decode38 is
   Port ( a : in STD_LOGIC_VECTOR (2 downto 0);
          y : out STD_LOGIC_VECTOR (7 downto 0));
end component decode38;

signal a :  STD_LOGIC_VECTOR (2 downto 0):= "000";
signal y :  STD_LOGIC_VECTOR (7 downto 0);
```

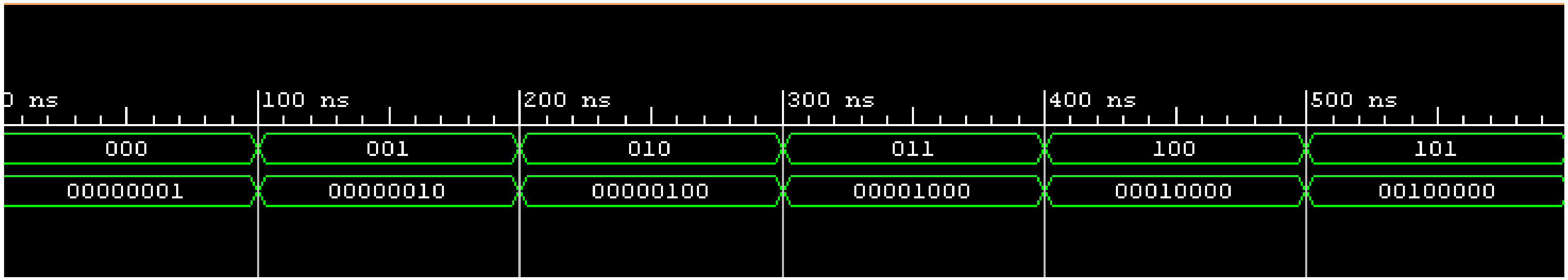
Lab 8 – solution – testbench

```
begin
UUT: decode38
Port map( a => a, y => y);

stim_process: process begin

for i in 0 to 8 loop
   wait for 100 ns;
   a <= a + 1;
end loop;

wait;

end process;
end Behavioral;
```

Lab 8 – solution – testbench



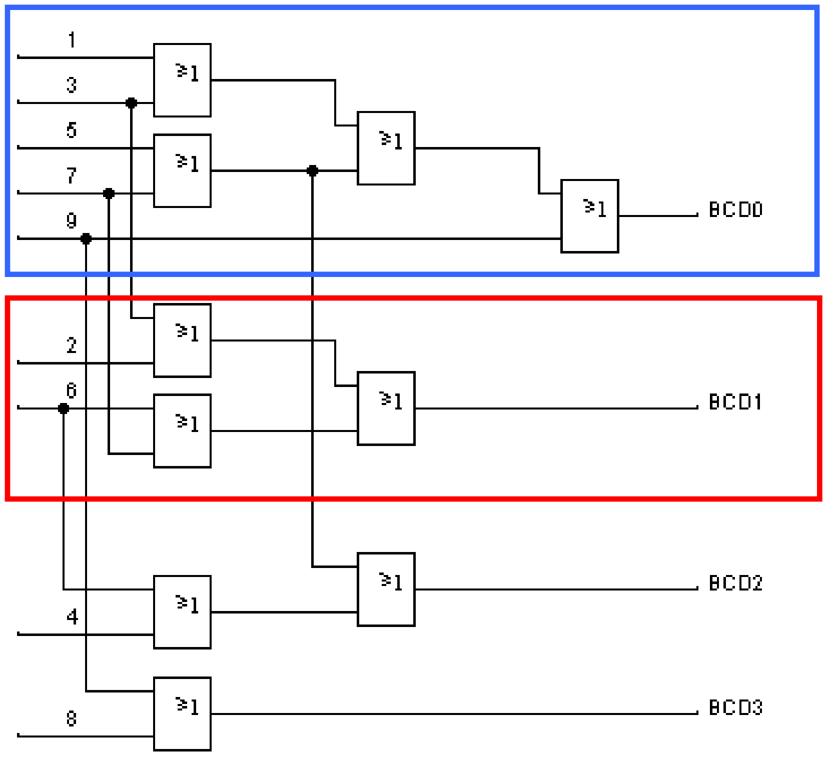
When input changes the corresponding output get '1'.

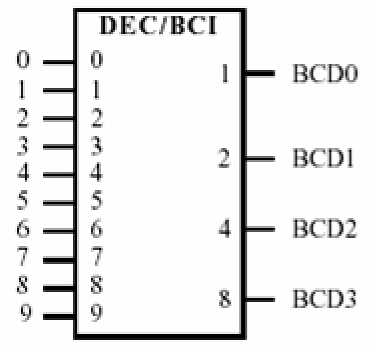


Combinatorial Logic

DEC/BCD encoder without priority

DEC	BCD3	BCD2	BCD1	BCD0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1





BCD0 = 1+3 ·+5+7+9

BCD1 = 2+3+6+7

BCD2 = 4+5+6+7

BCD1 = 8+9

Combinatorial Logic

DEC/BCD encoder with priority

- HIGH PRIORITY
  - The highest number appears on the output
- LOW PRIORITY
  - The lowest number appears on the output

Combinatorial Logic

DEC/BCD encoder with priority

0	1	2	3	4	5	6	7	8	9	BCD3	BCD2	BCD1	BCD0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	0	0	0	1
X	x	1	0	0	0	0	0	0	0	0	0	1	0
x	x	x	1	0	0	0	0	0	0	0	0	1	1
x	x	x	x	1	0	0	0	0	0	0	1	0	0
x	x	x	x	x	1	0	0	0	0	0	1	0	1
x	x	x	x	x	x	1	0	0	0	0	1	1	0
x	x	x	x	x	x	x	1	0	0	0	1	1	1
x	x	x	x	x	x	x	x	1	0	1	0	0	0
x	x	x	x	x	x	x	x	x	1	1	0	0	1

$$\mathrm{BCD0} = 1\overline{2}\,\overline{4}\,\overline{6}\,\overline{8} + 3\overline{4}\,\overline{6}\,\overline{8} + 5\overline{6}\,\overline{8} + 7\overline{8} + 9$$

$$\mathrm{BCD1} = 2\overline{4}\,\overline{5}\,\overline{8}\,\overline{9} + 3\overline{4}\,\overline{5}\,\overline{8}\,\overline{9} + 6\overline{8}\,\overline{9} + 7\overline{8}\,\overline{9}$$

$$\mathrm{BCD2} = 4\overline{8}\,\overline{9} + 5\overline{8}\,\overline{9} + 6\overline{8}\,\overline{9} + 7\overline{8}\,\overline{9}$$

$$\mathrm{BCD3} = 8 + 9$$

Lab 9

Make a 8 to 3 Encoder in VHDL

- With gates
- With a for loop
- Simulate

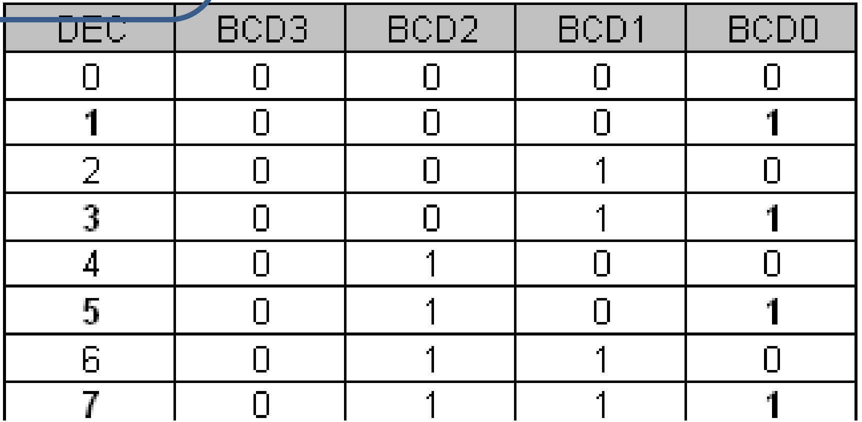
Lab 9 – solution - gates

```
entity encode83 is
		port(
				x : in STD_LOGIC_VECTOR(7 downto 0);
				y : out STD_LOGIC_VECTOR(2 downto 0);
				valid: out STD_LOGIC
		);
end encode83;
```

Lab 9 – solution - gates

```
architecture behavior of encode83 is
begin

   process(x)
   variable valid_var: STD_LOGIC;
   begin
                                y(2) <= x(7) or x(6) or x(5) or x(4);
                                y(1) <= x(7) or x(6) or x(3) or x(2);
                                y(0) <= x(7) or x(5) or x(3) or x(1);
                                valid_var := '0';
                                for i in 7 downto 0 loop
                                             valid_var := valid_var or x(i);
                                end loop;
                                valid <= valid_var;
              end process;

end behavior;
```

Check if there was an input



DEC	BCD3	BCD2	BCD1	BCD0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1

Lab 9 – solution – for loop

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_unsigned.all;

entity encode83 is
		port(
				x : in STD_LOGIC_VECTOR(7 downto 0);
				valid : out STD_LOGIC;
				y : out STD_LOGIC_VECTOR(2 downto 0)
		);
end encode83;
```

Arith is necessary for conv_std_logic_vector(j,3)

Lab 9 – solution – for loop

```
architecture behavior of encode83 is
begin
  process(x)
  variable j: integer;
  begin
     y <= "000";
     valid <= '0';
     for j in 0 to 7 loop
        if x(j) = '1' then
                          y <= conv_std_logic_vector(j,3);
             valid <= '1';
        end if;
     end loop;
  end process;
end behavior;
```

$\xrightarrow{}$ Converts integer to binary

Priority?   HIGH

LOW: for j in 7 <u>downto</u> 0

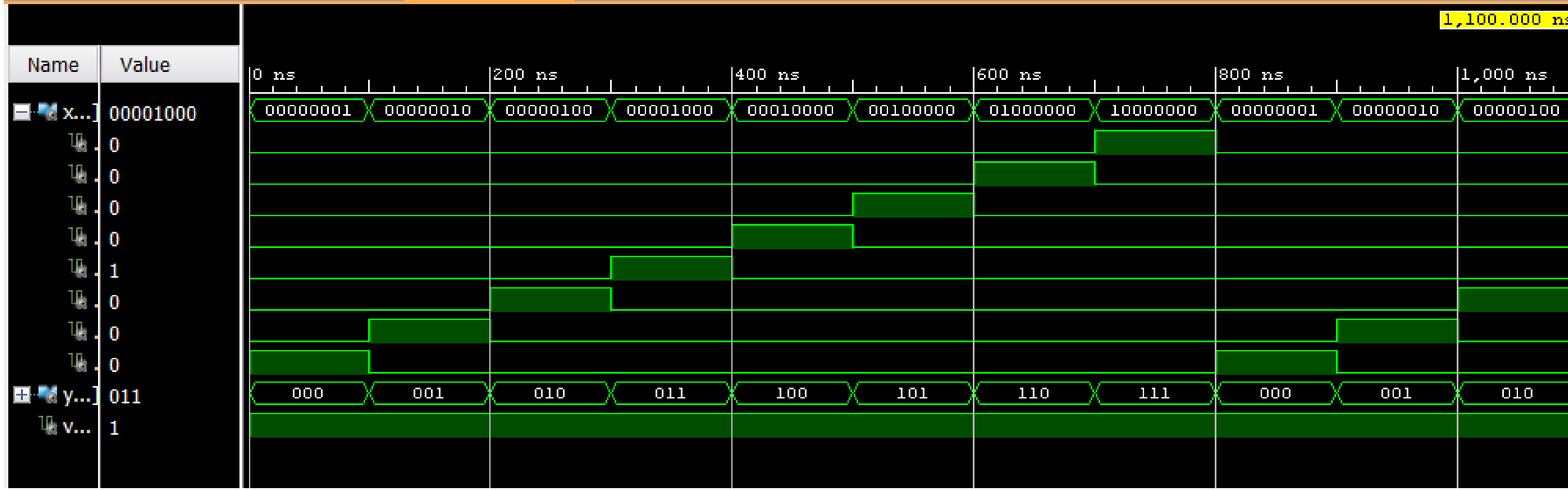
Lab 9 – solution – tb

```
entity encode83_tb is
--  Port ( );
end encode83_tb;

architecture Behavioral of encode83_tb is

component encode83 is
    Port ( x : in STD_LOGIC_VECTOR (7 downto 0);
           y : out STD_LOGIC_VECTOR (2 downto 0);
           valid : out STD_LOGIC);
end component encode83;

signal x : STD_LOGIC_VECTOR (7 downto 0):= "00000001";
signal y :  STD_LOGIC_VECTOR (2 downto 0);
signal valid :  STD_LOGIC;
```

Lab 9 – solution – tb

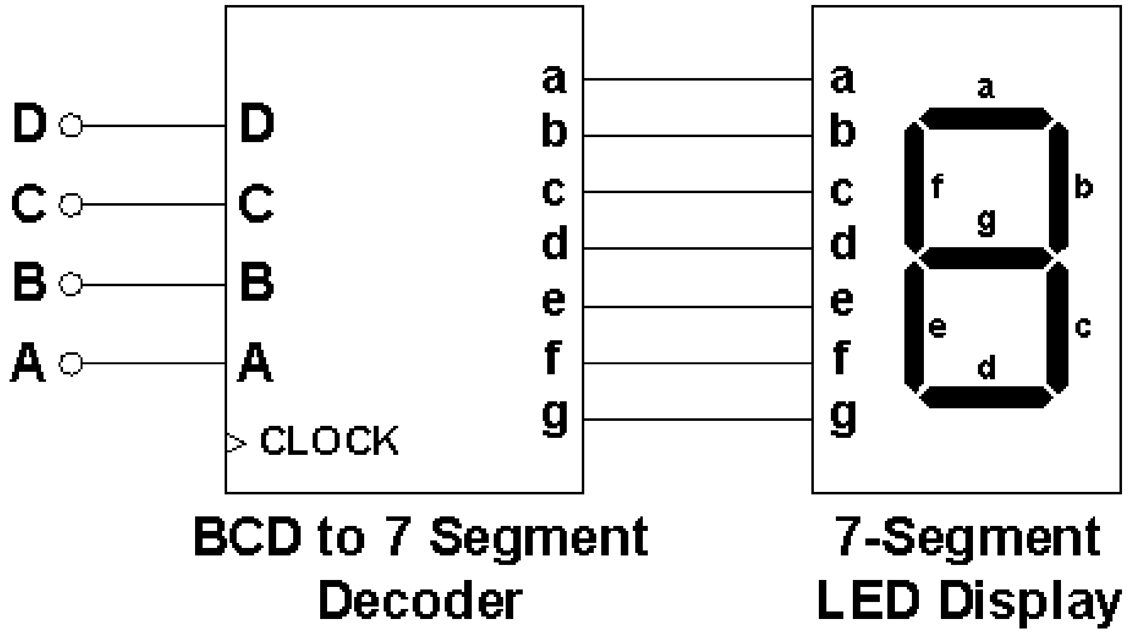
```
begin

UUT: encode83
    Port map( x => x, y => y, valid => valid);

stim_proc: process begin
for i in 0 to 10 loop
   wait for 100 ns;
   x(7 downto 1) <= x(6 downto 0);
   x(0) <= x(7);
end loop;
wait;

end process;
```

Lab 9 – solution – tb



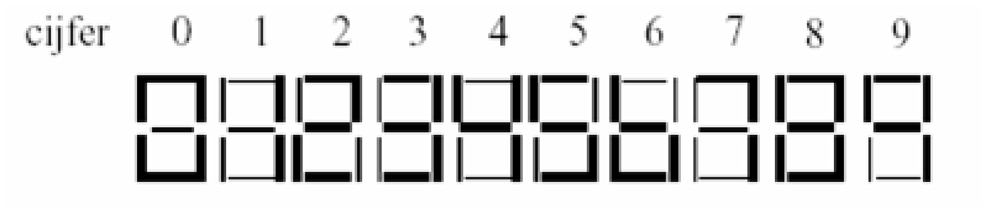
Combinatorial Logic

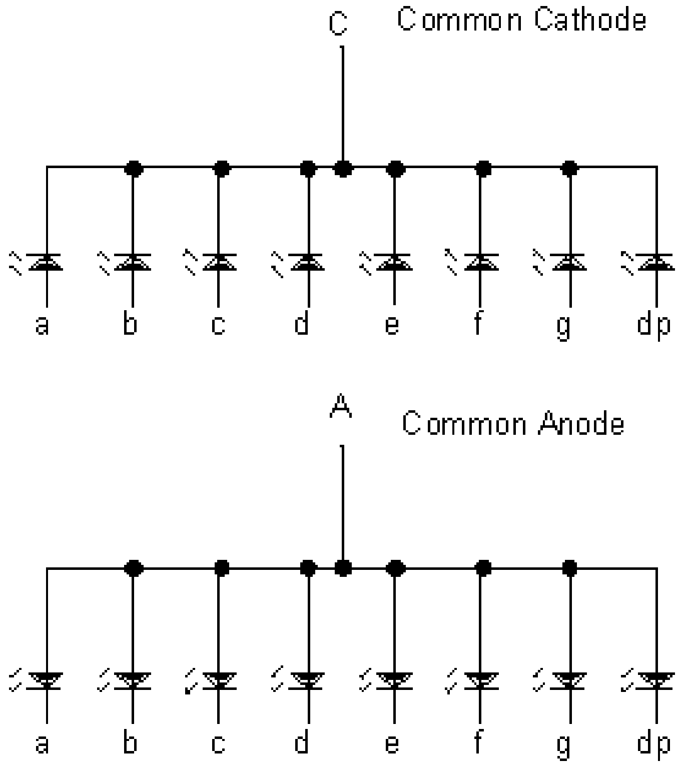
BCD/7-segment decoder

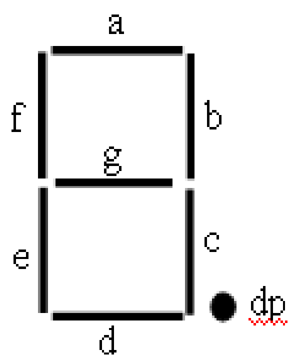


Combinatorial Logic

BCD/7-segment decoder





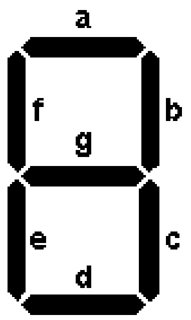


Combinatorial Logic

BCD/7-segment decoder

7-segment decoder active low outputs → **COMMON ANODE DISPLAY**

DEC	BCD3	BCD2	BCD1	BCD0	g	f	e	d	c	b	a
0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1	0	0	1
2	0	0	1	0	0	1	0	0	1	0	0
3	0	0	1	1	0	1	1	0	0	0	0
4	0	1	0	0	0	0	1	1	0	0	1
5	0	1	0	1	0	0	1	0	0	1	0
6	0	1	1	0	0	0	0	0	0	1	1
7	0	1	1	1	1	1	1	1	0	0	0
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	1	1	0	0	0



*Lab 10

Make a BCD/7-segment decoder

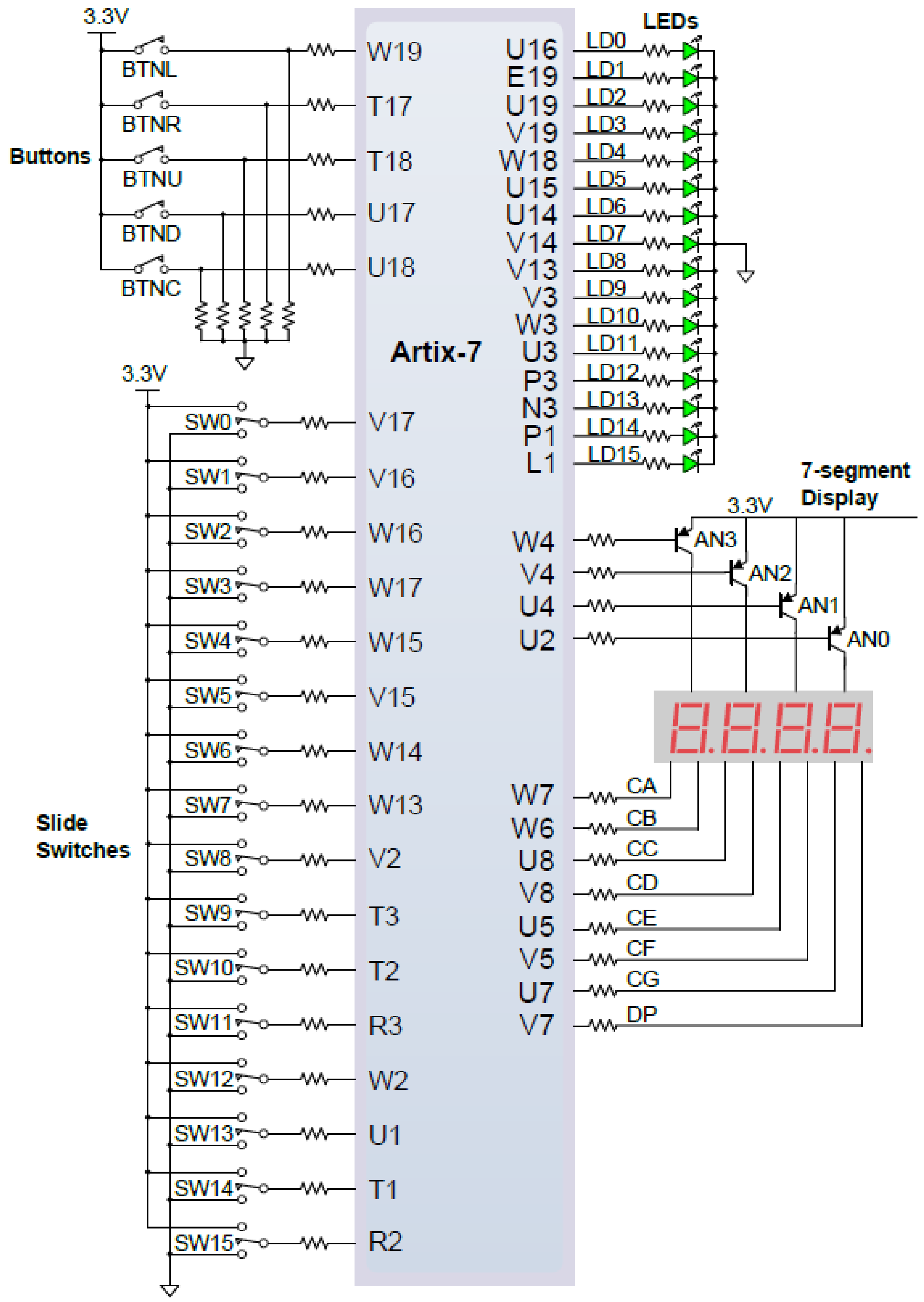
- Connect to the 7-segment display: active low
- Anodes are also connected active low
- Use 4 inputs for the binary number
- Use 4 inputs for the anodes input
- Simulate

Tempus

Lab 10

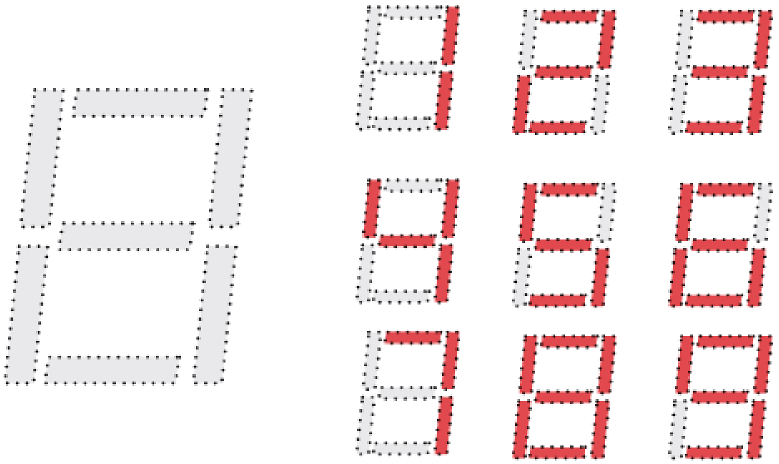
7-segment display: active low

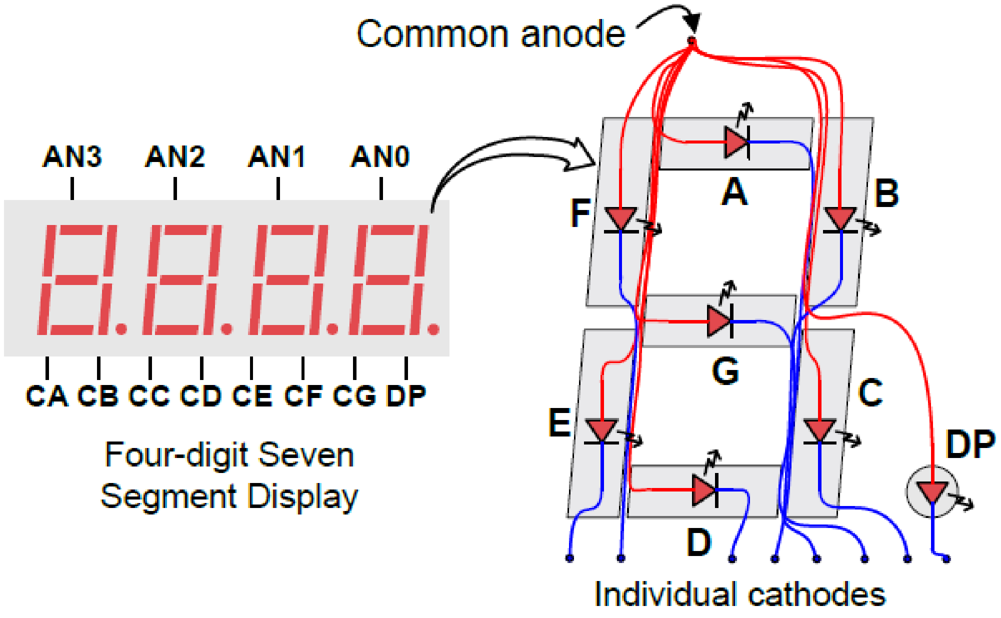
Anodes: active low



Lab 10

7-segment display: active low
Anodes: active low





Lab 10 - solution

```
entity hex7seg is
    Port ( x : in STD_LOGIC_VECTOR (3 downto 0);
           an_in : in STD_LOGIC_VECTOR (3 downto 0);
           g_to_a : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
end hex7seg;

architecture Behavioral of hex7seg is

begin
```

Lab 10 - solution

```
process(x)
begin
   case x is
      when "0000"    => g_to_a    <= "1000000";    --0
      when "0001"    => g_to_a    <= "1111001";    --1
      when "0010"    => g_to_a    <= "0100100";    --2
      when "0011"    => g_to_a    <= "0110000";    --3
      when "0100"    => g_to_a    <= "0011001";    --4
      when "0101"    => g_to_a    <= "0010010";    --5
      when "0110"    => g_to_a    <= "0000010";    --6
      when "0111"    => g_to_a    <= "1011000";    --7
      when "1000"    => g_to_a    <= "0000000";    --8
      when "1001"    => g_to_a    <= "0010000";    --9
      when "1010"    => g_to_a    <= "0001000";    --A
      when "1011"    => g_to_a    <= "0000011";    --b
      when "1100"    => g_to_a    <= "1000110";    --C
      when "1101"    => g_to_a    <= "0100001";    --d
      when "1110"    => g_to_a    <= "0000110";    --E
      when others    => g_to_a    <= "0001110";    --F
   end case;
end process;
an <= an_in;
end Behavioral;
```

Lab 10 - xdc

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {x[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {x[0]}]
set_property PACKAGE_PIN V16 [get_ports {x[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {x[1]}]
set_property PACKAGE_PIN W16 [get_ports {x[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {x[2]}]
set_property PACKAGE_PIN W17 [get_ports {x[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {x[3]}]
set_property PACKAGE_PIN W15 [get_ports {an_in[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an_in[0]}]
set_property PACKAGE_PIN V15 [get_ports {an_in[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an_in[1]}]
set_property PACKAGE_PIN W14 [get_ports {an_in[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an_in[2]}]
set_property PACKAGE_PIN W13 [get_ports {an_in[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an_in[3]}]
```

Lab 10 - xdc

```
##7 segment display
set_property PACKAGE_PIN W7 [get_ports {g_to_a[0]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {g_to_a[0]}]
set_property PACKAGE_PIN W6 [get_ports {g_to_a[1]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {g_to_a[1]}]
set_property PACKAGE_PIN U8 [get_ports {g_to_a[2]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {g_to_a[2]}]
set_property PACKAGE_PIN V8 [get_ports {g_to_a[3]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {g_to_a[3]}]
set_property PACKAGE_PIN U5 [get_ports {g_to_a[4]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {g_to_a[4]}]
set_property PACKAGE_PIN V5 [get_ports {g_to_a[5]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {g_to_a[5]}]
set_property PACKAGE_PIN U7 [get_ports {g_to_a[6]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {g_to_a[6]}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
		set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```

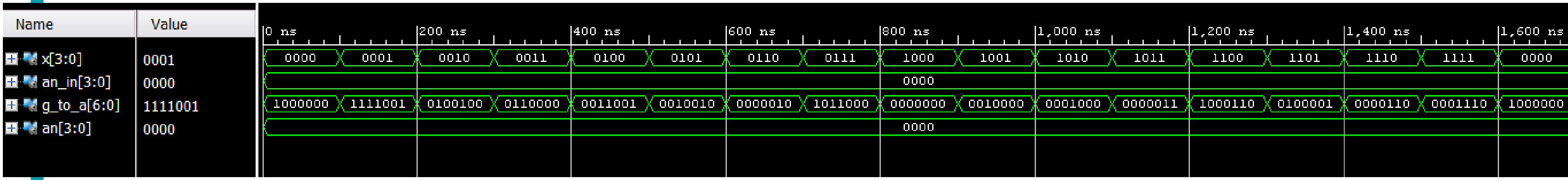
Lab 10 - testbench

```
entity hex7seg_tb is
--  Port ( );
end hex7seg_tb;

architecture Behavioral of hex7seg_tb is

component hex7seg is
   Port ( x : in STD_LOGIC_VECTOR (3 downto 0);
          an_in : in STD_LOGIC_VECTOR (3 downto 0);
          g_to_a : out STD_LOGIC_VECTOR (6 downto 0);
          an : out STD_LOGIC_VECTOR (3 downto 0));
end component hex7seg;

signal x : STD_LOGIC_VECTOR (3 downto 0):="0000";
signal an_in :  STD_LOGIC_VECTOR (3 downto 0):="0000";
signal g_to_a : STD_LOGIC_VECTOR (6 downto 0);
signal an :  STD_LOGIC_VECTOR (3 downto 0);
```

Lab 10 - testbench

```
begin

UUT: hex7seg
Port map( x => x,
          an_in => an_in ,
          g_to_a => g_to_a,
          an => an);
stim_process: process begin

for i in 0 to 16 loop
  wait for 100 ns;
  x <= x + 1;
end loop;

wait;
end process;

end Behavioral;
```

Lab 10 - testbench



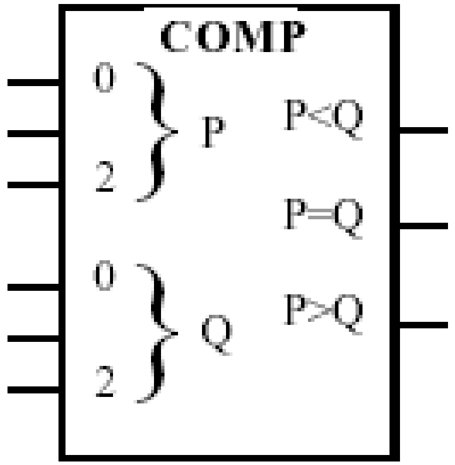
```
Name          Value      |0 ns      |200 ns    |400 ns    |600 ns    |800 ns    |1,000 ns  |1,200 ns  |1,400 ns  |1,600 ns
x[3:0]        0001       0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000
an_in[3:0]    0000       0000
g_to_a[6:0]   1111001    1000000 1111001 0100100 0110000 0011001 0010010 0000010 1011000 0000000 0010000 0001000 0000011 1000110 0100001 0000110 0001110 1000000
an[3:0]       0000       0000
```

```
case x is
    when "0000"    => g_to_a    <= "1000000";    --0
    when "0001"    => g_to_a    <= "1111001";    --1
    when "0010"    => g_to_a    <= "0100100";    --2
    when "0011"    => g_to_a    <= "0110000";    --3
    when "0100"    => g_to_a    <= "0011001";    --4
    when "0101"    => g_to_a    <= "0010010";    --5
    when "0110"    => g_to_a    <= "0000010";    --6
    when "0111"    => g_to_a    <= "1011000";    --7
    when "1000"    => g_to_a    <= "0000000";    --8
    when "1001"    => g_to_a    <= "0010000";    --9
    when "1010"    => g_to_a    <= "0001000";    --A
    when "1011"    => g_to_a    <= "0000011";    --b
    when "1100"    => g_to_a    <= "1000110";    --C
    when "1101"    => g_to_a    <= "0100001";    --d
    when "1110"    => g_to_a    <= "0000110";    --E
    when others    => g_to_a    <= "0001110";    --F
```

Combinatorial Logic

Comparator

A	B	A < B	A > B	A = B
0	0	0	0	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1



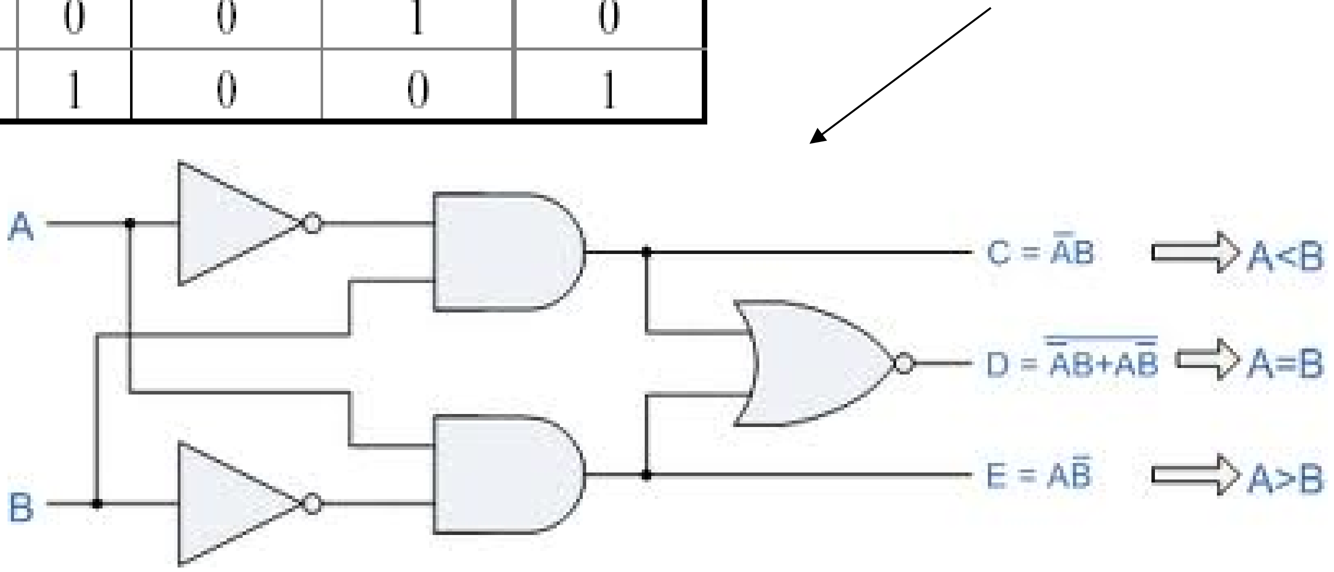
Combinatorial Logic

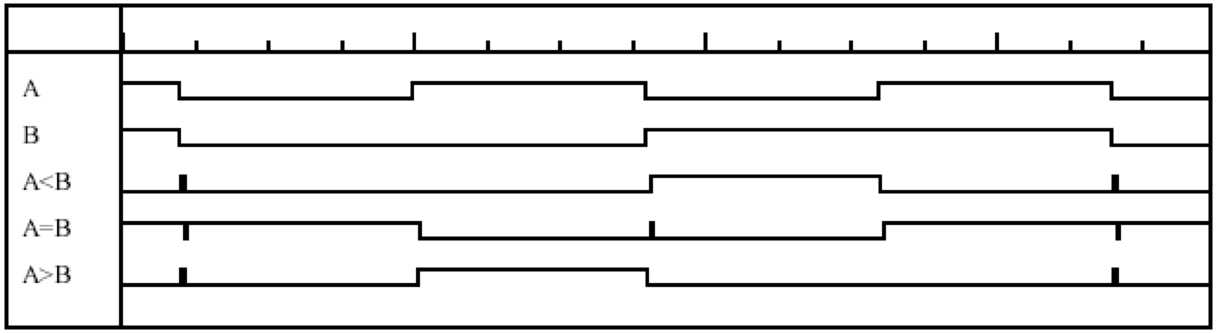
A	B	A < B	A > B	A = B
0	0	0	0	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

$$(A < B) = \overline{A}B$$

$$(A > B) = A\overline{B}$$

$$(A = B) = \overline{\overline{A}B + A\overline{B}} = \overline{A \oplus B}$$


A
B
C = $\overline{A}B$ ⇒ A<B
D = $\overline{\overline{A}B+A\overline{B}}$ ⇒ A=B
E = $A\overline{B}$ ⇒ A>B



Combinatorial Logic

2-bit comparator

A_1 , B_1	A_0 , B_0	$A > B$	$A < B$	$A = B$
$A_1 > B_1$	x	1	0	0
$A_1 < B_1$	x	0	1	0
$A_1 = B_1$	$A_0 > B_0$	1	0	0
$A_1 = B_1$	$A_0 < B_0$	0	1	0
$A_1 = B_1$	$A_0 = B_0$	0	0	1

4-bit comparator

A_3 , B_3	A_2 , B_2	A_1 , B_1	A_0 , B_0	$A > B$	$A < B$	$A = B$
$A_3 > B_3$	x	x	x	1	0	0
$A_3 < B_3$	x	x	x	0	1	0
$A_3 = B_3$	$A_2 > B_2$	x	x	1	0	0
$A_3 = B_3$	$A_2 < B_2$	x	x	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	x	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	x	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1

Lab 11

Make a comparator

- Simulate

Lab 11 - solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity comp is
        generic (N:integer := 8);
        port(
                x : in STD_LOGIC_VECTOR(N-1 downto 0);
                y : in STD_LOGIC_VECTOR(N-1 downto 0);
                gt : out STD_LOGIC;
                eq : out STD_LOGIC;
                lt : out STD_LOGIC
         );
end comp;
```

Can be used to make other comparators when inserted as component in a separate top module:

- generic = 4 creates a 4-bit comp
- generic = 16 creates a 16-bit comp

Lab 11 - solution

```
architecture behavior of comp is
begin
        process(x, y)
        begin
                gt <= '0';
                eq <= '0';
                lt <= '0';
                if (x > y) then
                        gt <= '1';
                elsif (x = y) then
                        eq <= '1';
                elsif (x < y) then
                        lt <= '1';
                end if;
        end process;
end behavior;
```

Lab 11 – solution - testbench

```
entity comp_tb is
end comp_tb;

architecture Behavioral of comp_tb is
        component comp is
         generic (N:integer := 8);
         port(
                x : in STD_LOGIC_VECTOR(N-1 downto 0);
                y : in STD_LOGIC_VECTOR(N-1 downto 0);
                gt : out STD_LOGIC;
                eq : out STD_LOGIC;
                lt : out STD_LOGIC);
        end component comp;
        signal x :  STD_LOGIC_VECTOR (7 downto 0);
        signal y :  STD_LOGIC_VECTOR (7 downto 0);
        signal gt : STD_LOGIC;
        signal eq : STD_LOGIC;
        signal lt :  STD_LOGIC;
begin
    UUT: comp
    Port map (x => x, y => y, gt => gt, eq => eq, lt => lt);
```

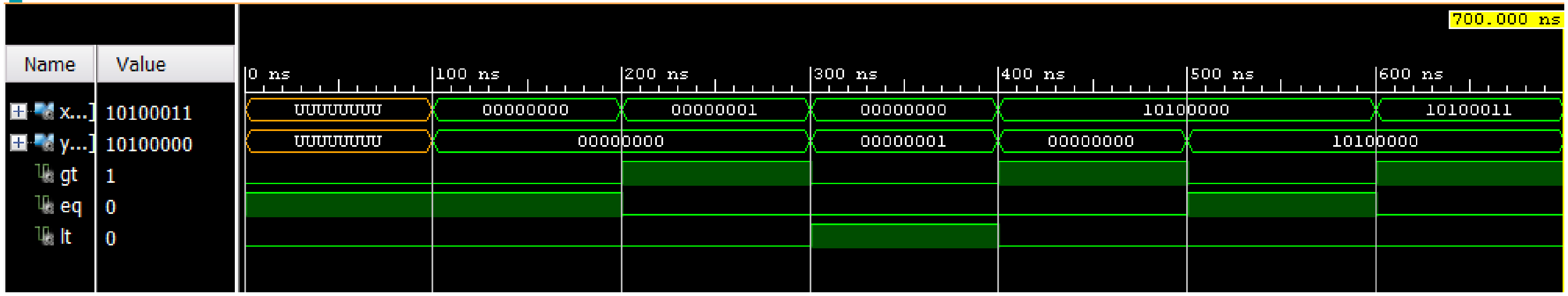
```
stim_process: process begin
        wait for 100 ns;
        x <= X"00";
        y <= X"00";
        wait for 100 ns;
        x <= X"01";
        y <= X"00";
        wait for 100 ns;
        x <= X"00";
        y <= X"01";
        wait for 100 ns;
        x <= X"A0";
        y <= X"00";
        wait for 100 ns;
        x <= X"A0";
        y <= X"A0";
        wait for 100 ns;
        x <= X"A3";
        y <= X"A0";
        wait for 100 ns;
        wait;
end process;
end Behavioral;
```

X”01”: hexadecimal
= B”00000001”

X”A3”: hexadecimal
= B”10100011”

Tempus

Lab 11 – solution - testbench



```
x...]  10100011   UUUUUUUU | 00000000 | 00000001 | 00000000 | 10100000 | 10100011
y...]  10100000   UUUUUUUU | 00000000 | 00000001 | 00000000 | 10100000
gt     1
eq     0
lt     0
```

```
if (x > y) then
            gt <= '1';
elsif (x = y) then
            eq <= '1';
elsif (x < y) then
            lt <= '1';
end if;
```

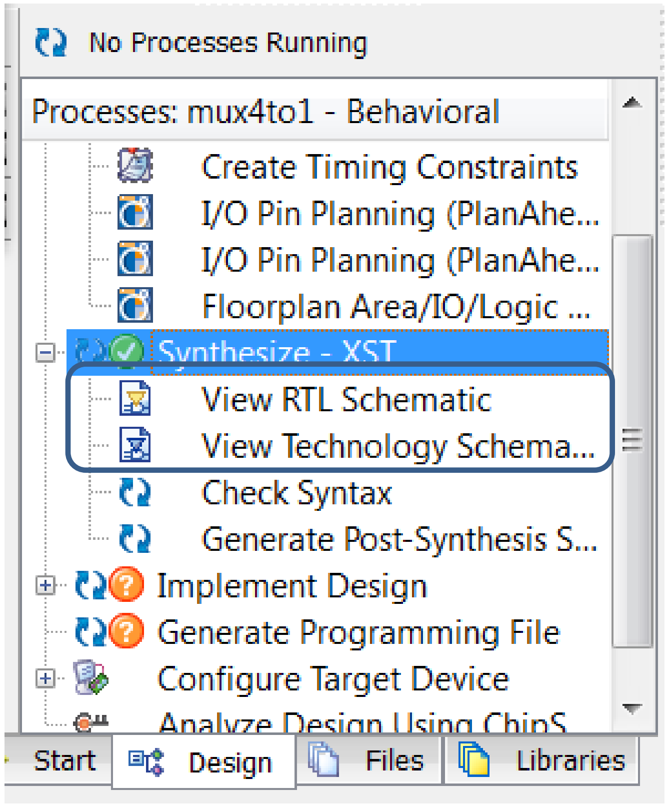
Analytical tools - ISE

- RTL
  - Gives the logical build-up of your circuit
- Technology
  - Gives the internal logic & resources used (LUT, FF’s, MUX, Buffers)

$\Rightarrow$Both after synthesis

Analytical tools - ISE

- RTL - Technology



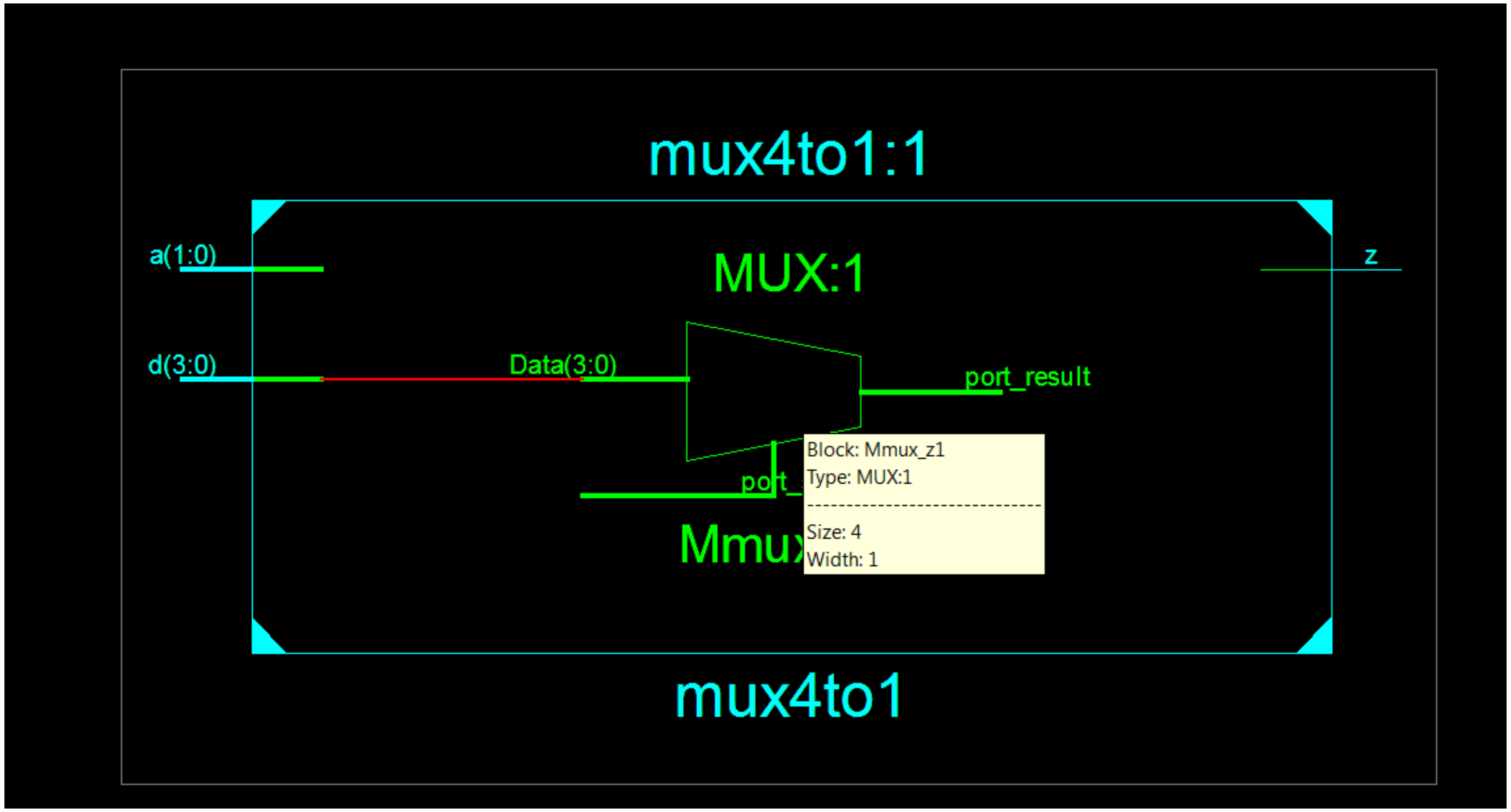
No Processes Running

Processes: mux4to1 - Behavioral

- Create Timing Constraints
- I/O Pin Planning (PlanAhe...
- I/O Pin Planning (PlanAhe...
- Floorplan Area/IO/Logic ...
- Synthesize - XST
  - View RTL Schematic
  - View Technology Schema...
  - Check Syntax
  - Generate Post-Synthesis S...
- Implement Design
- Generate Programming File
- Configure Target Device
- Analyze Design Using ChipS

Analytical tools - ISE

- RTL – Technology – mux4to1

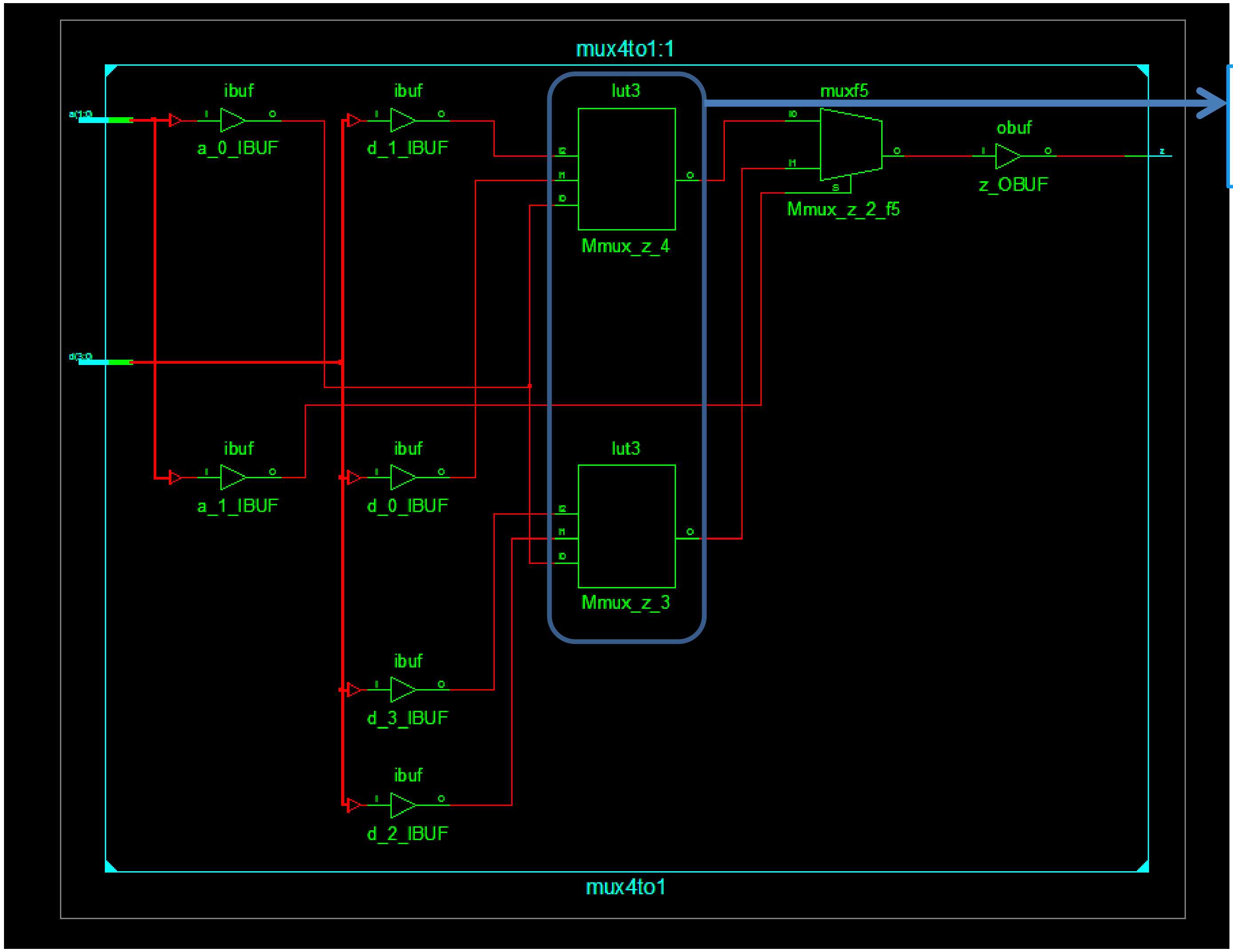


Block: Mmux_z1
Type: MUX:1

Size: 4
Width: 1

Analytical tools - ISE

- RTL – Technology – mux4to1



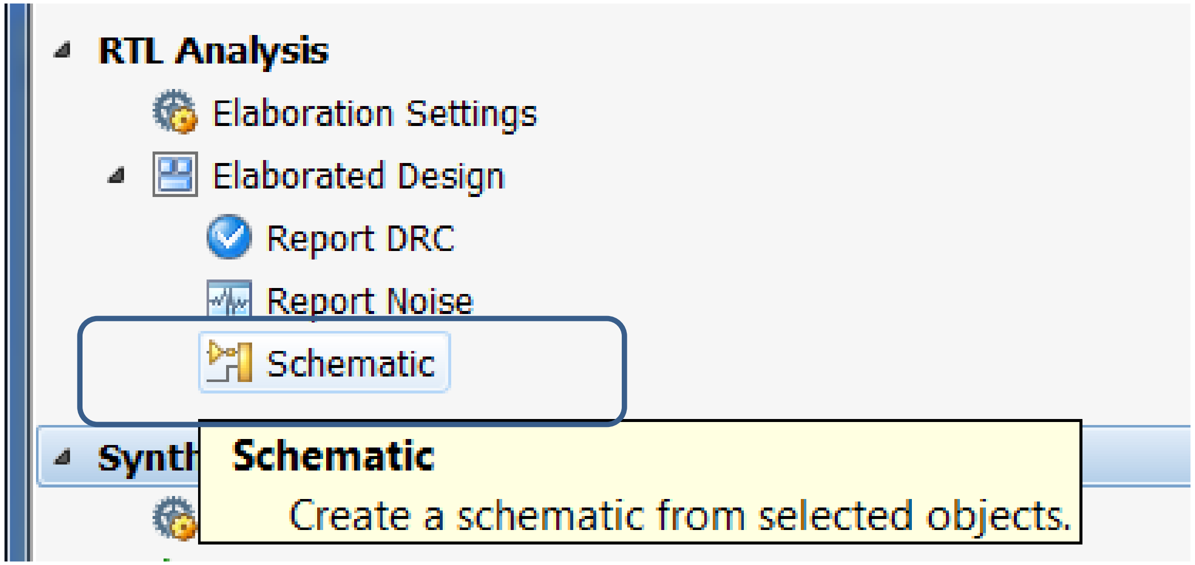
Basys2: max 4-input LUT for Spartan

Analytical tools - Vivado

- RTL
  - Gives the logical build-up of your circuit
- Synthesized design => schematic
  - Gives the internal logic used

Analytical tools - Vivado

- RTL

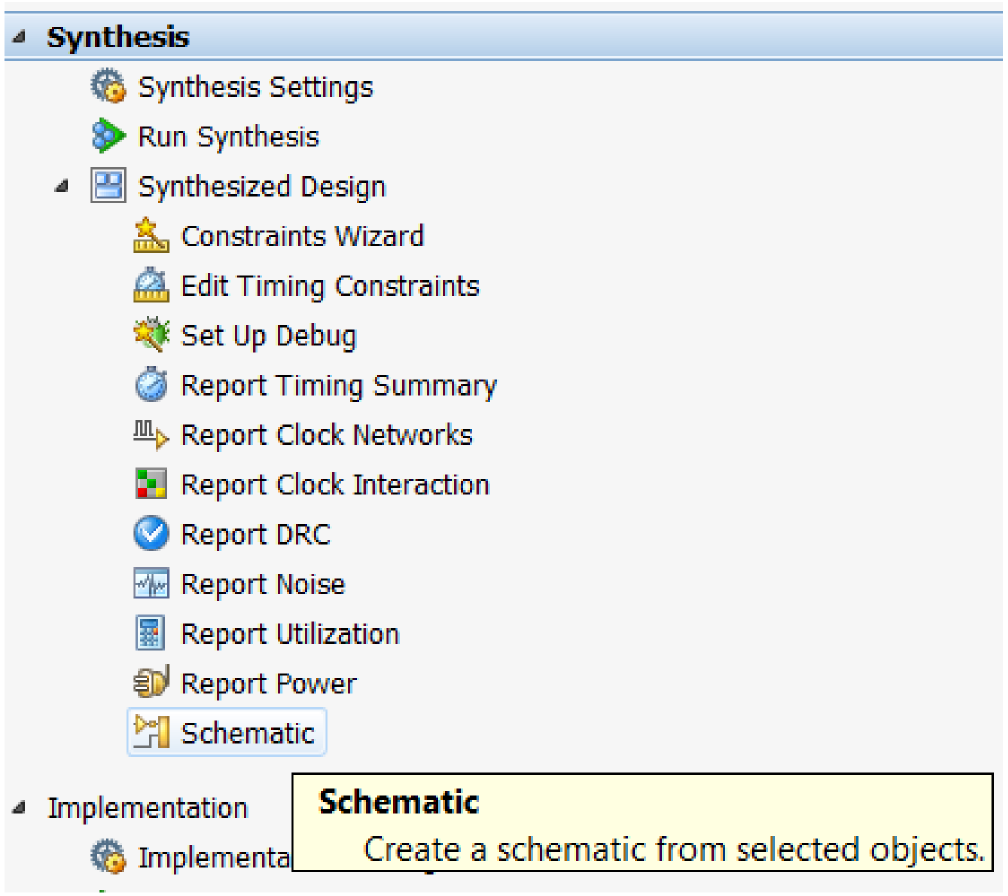


Schematic
Create a schematic from selected objects.

Tempus

Analytical tools - Vivado

- Synthesized design => schematic

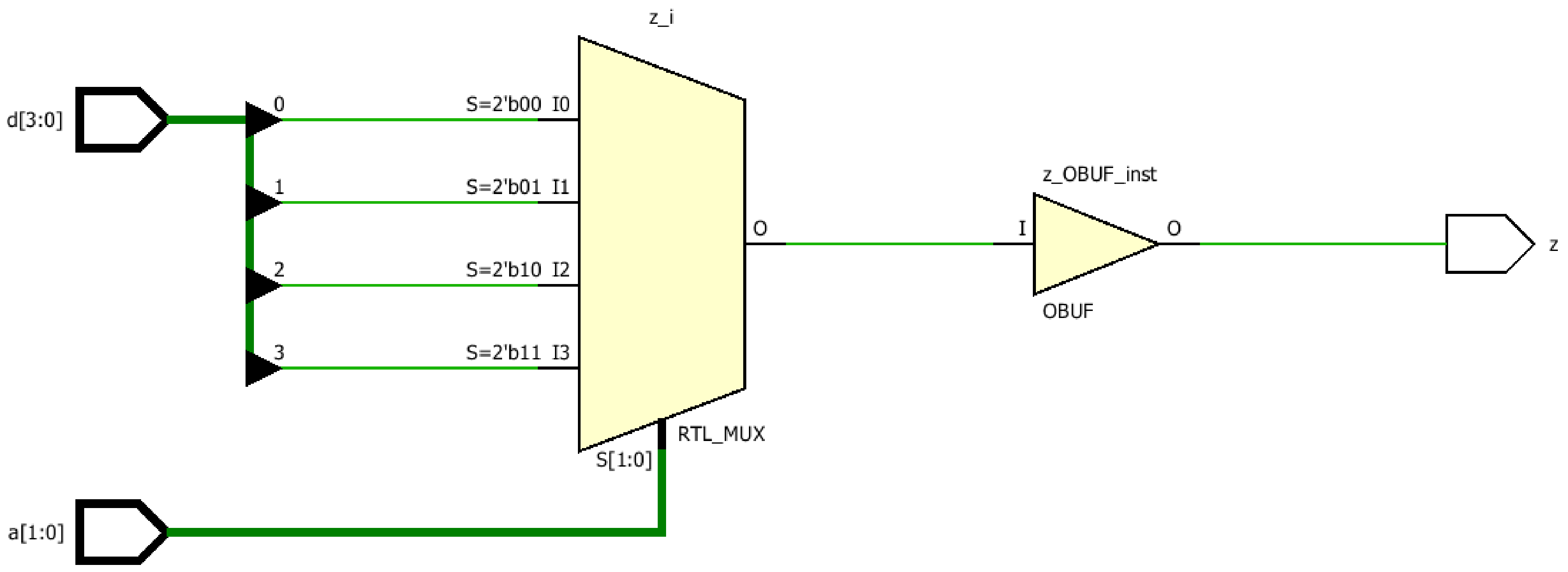


Synthesis
Synthesis Settings
Run Synthesis
Synthesized Design
Constraints Wizard
Edit Timing Constraints
Set Up Debug
Report Timing Summary
Report Clock Networks
Report Clock Interaction
Report DRC
Report Noise
Report Utilization
Report Power
Schematic
Implementation
Implementa

Schematic
Create a schematic from selected objects.

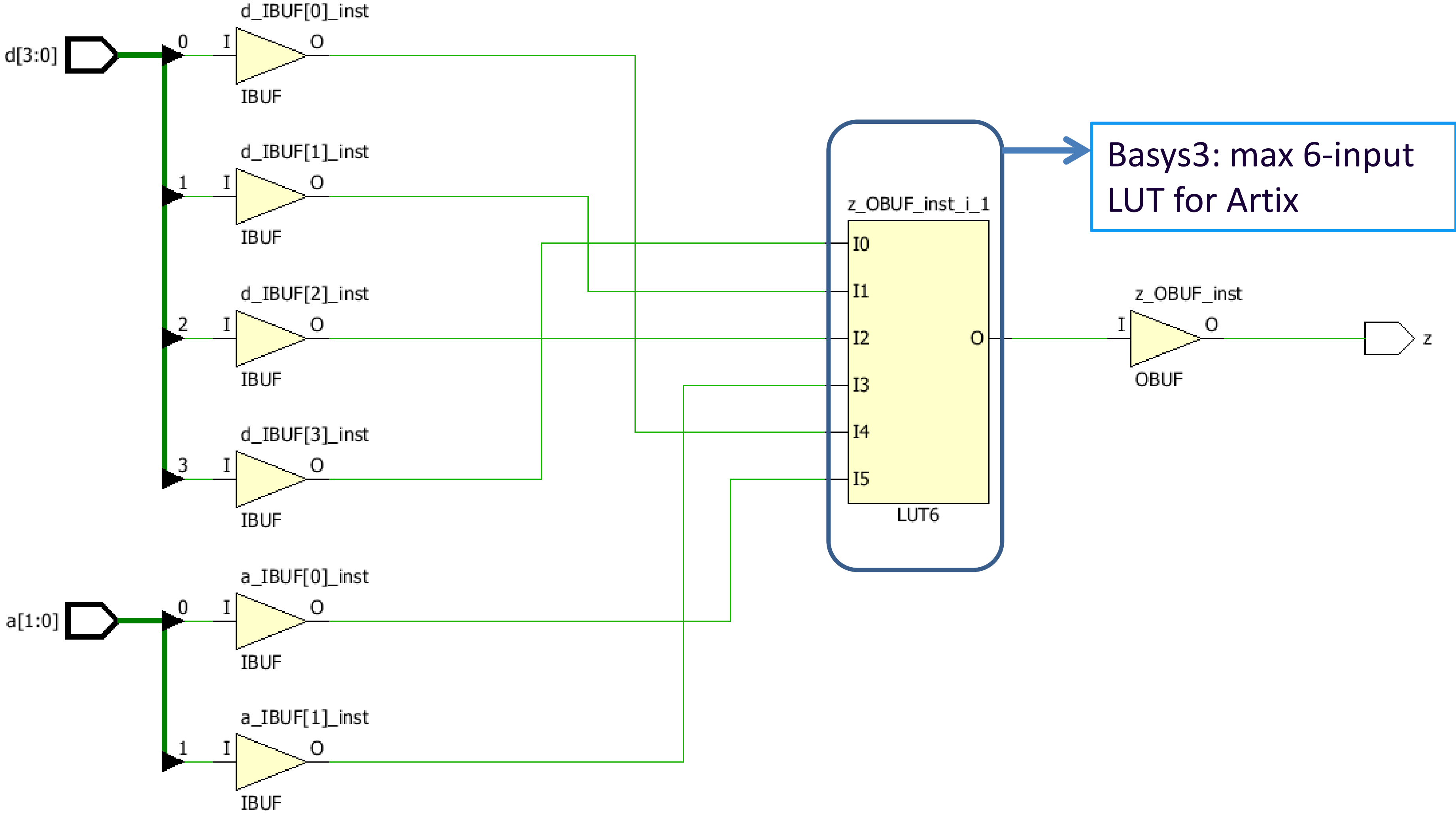
Analytical tools - Vivado

- RTL - mux4to1 – lab3_2



Analytical tools - Vivado

- Synthesized design - mux4to1

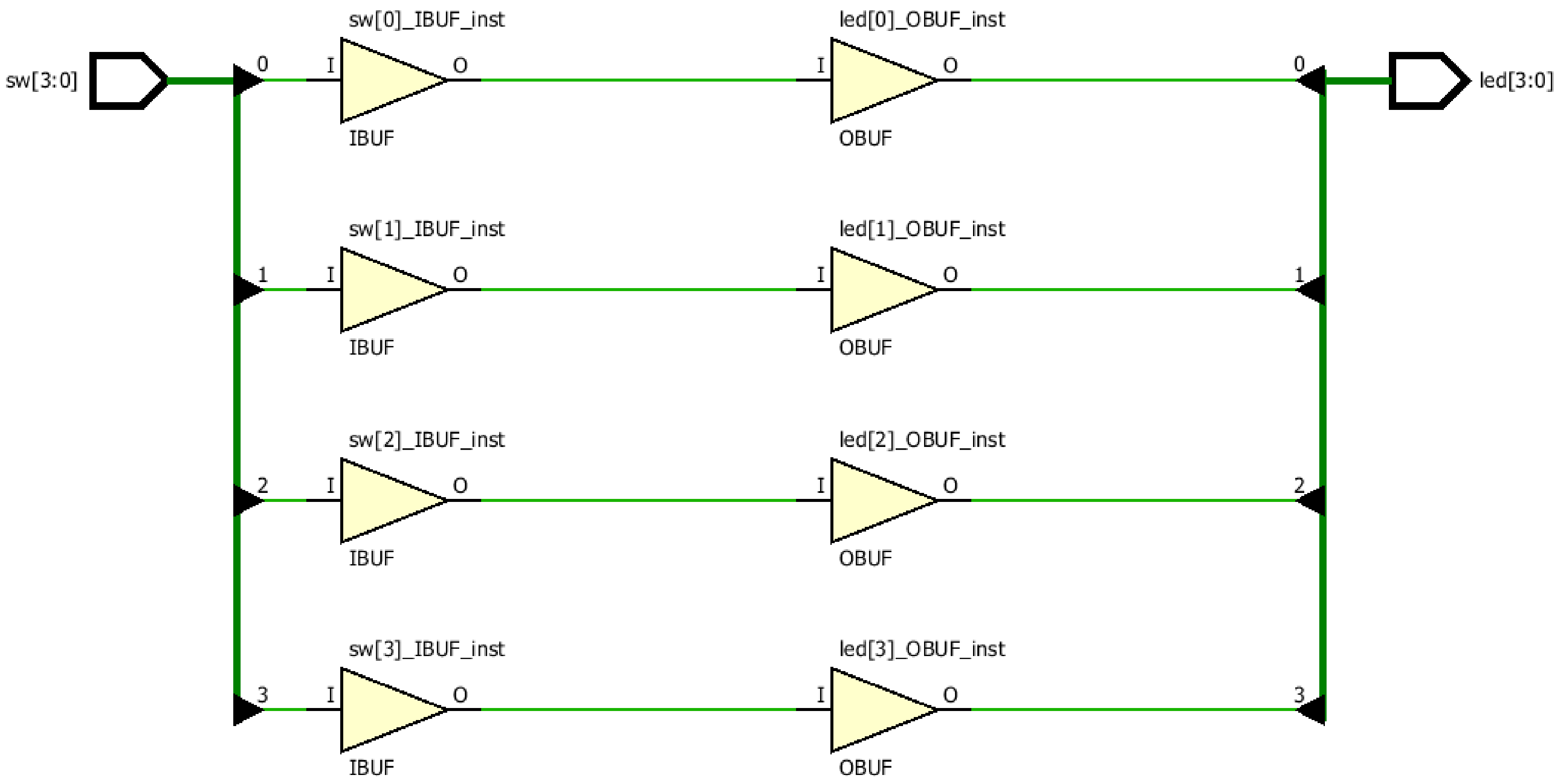


Basys3: max 6-input LUT for Artix

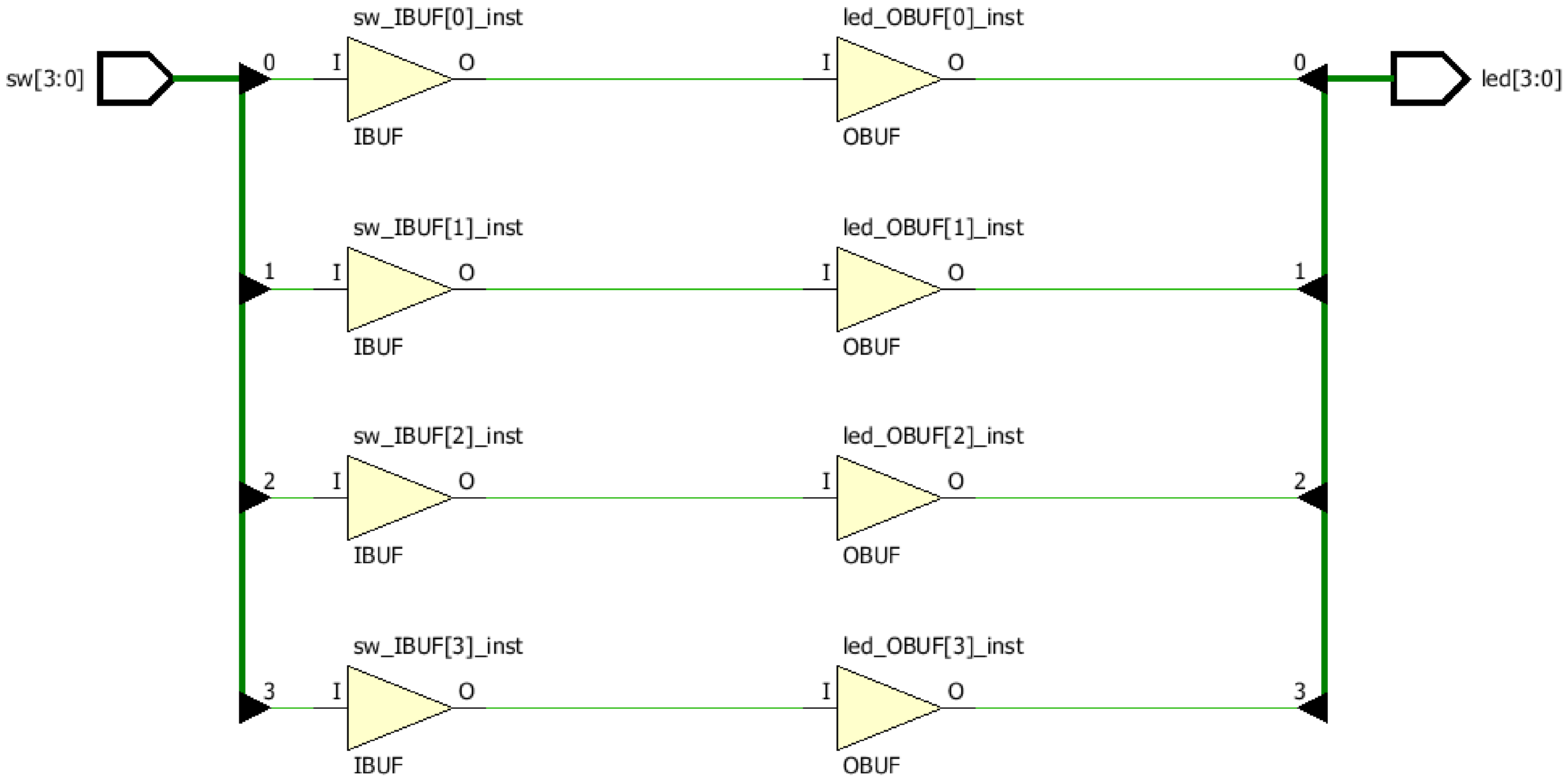
Lab 12

- Analyze RTL and Technology of
  Lab 0 – 1 – 2 – 4 – 6 – 8 – 9 – 11
- Explain the results

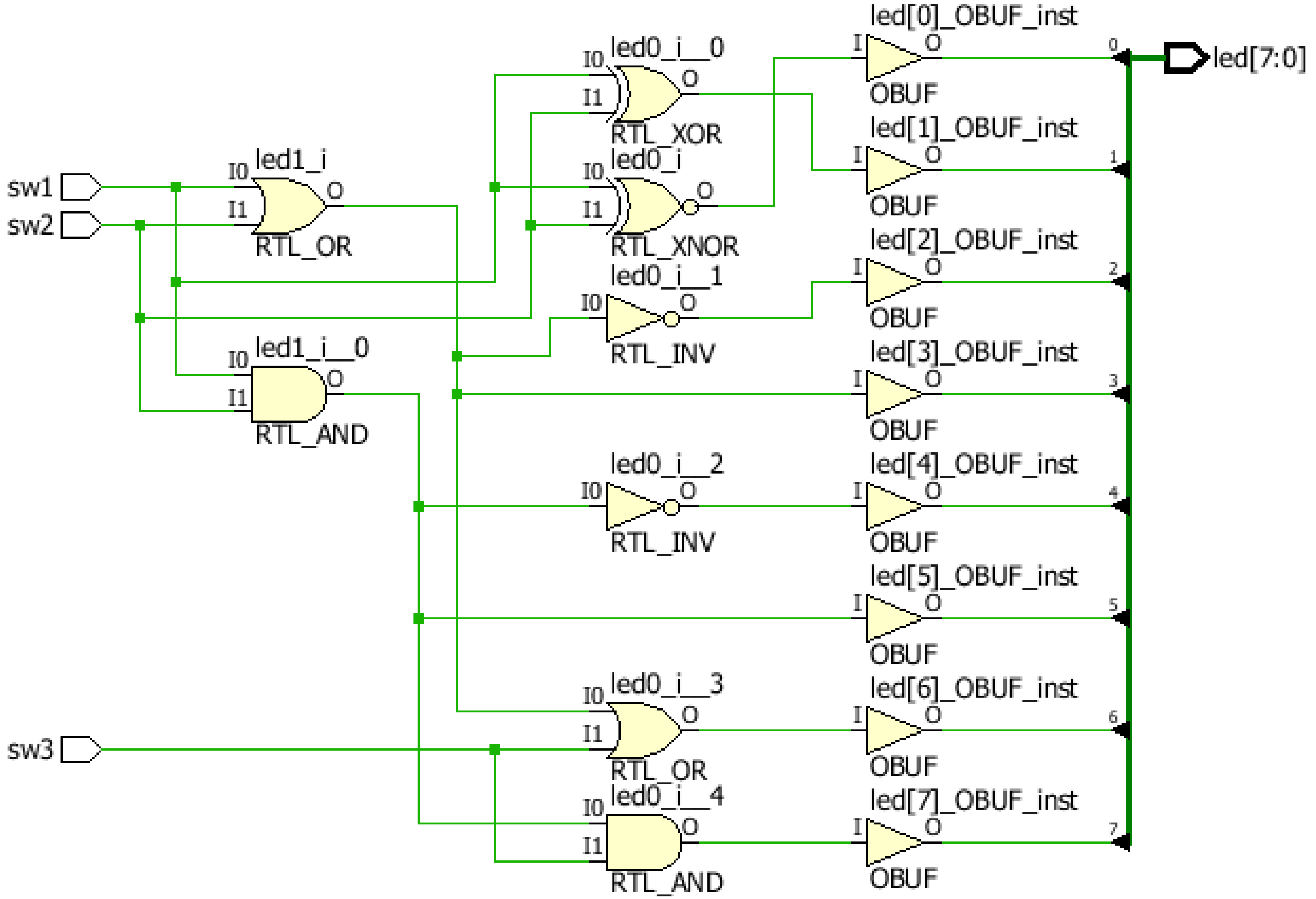
Lab 0 – solution - RTL



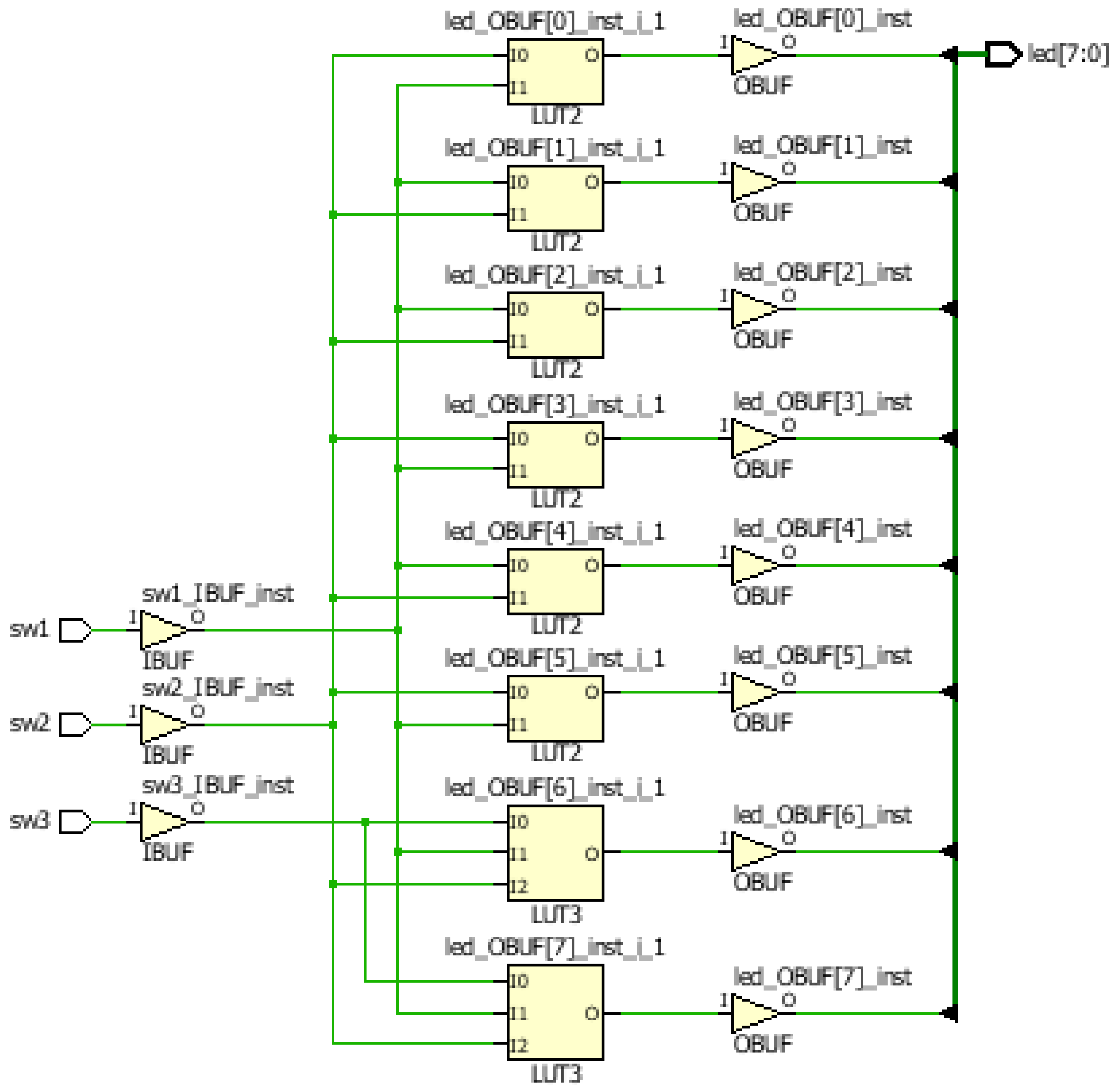
Lab 0 – solution - synth



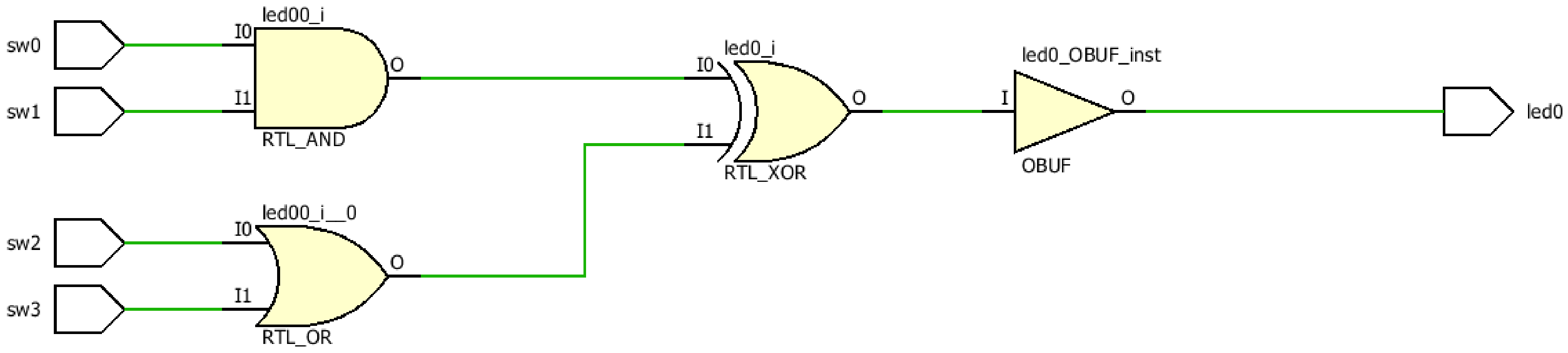
Lab 1 – solution - RTL



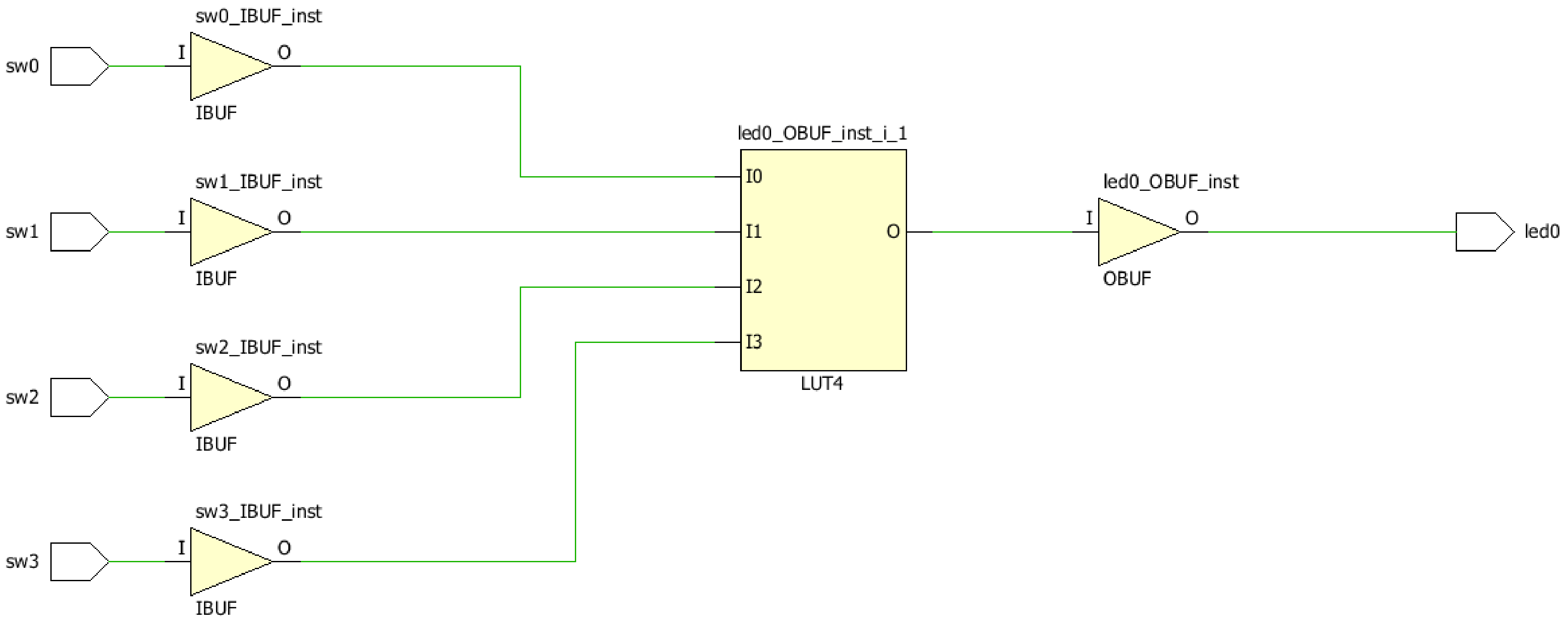
Lab 1 – solution - synth



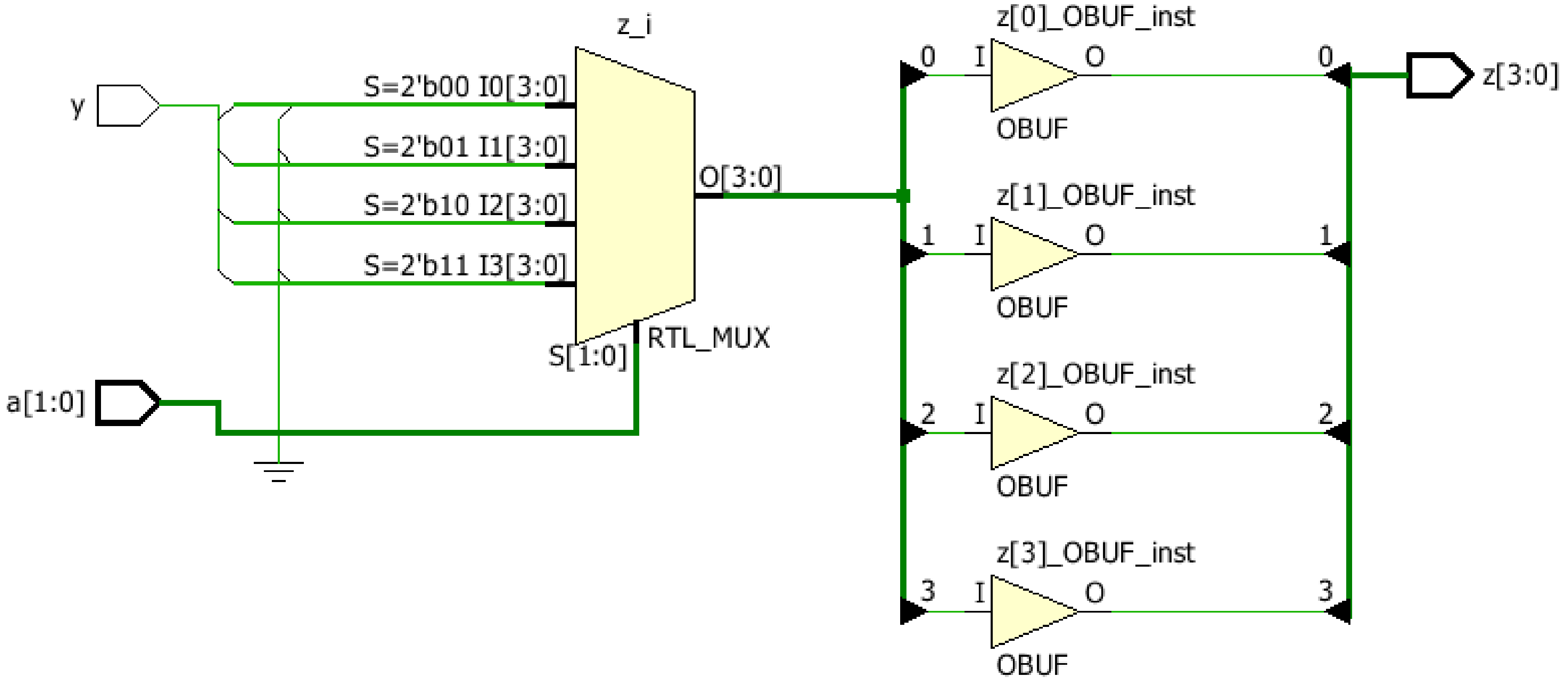
Lab 2 – solution - RTL



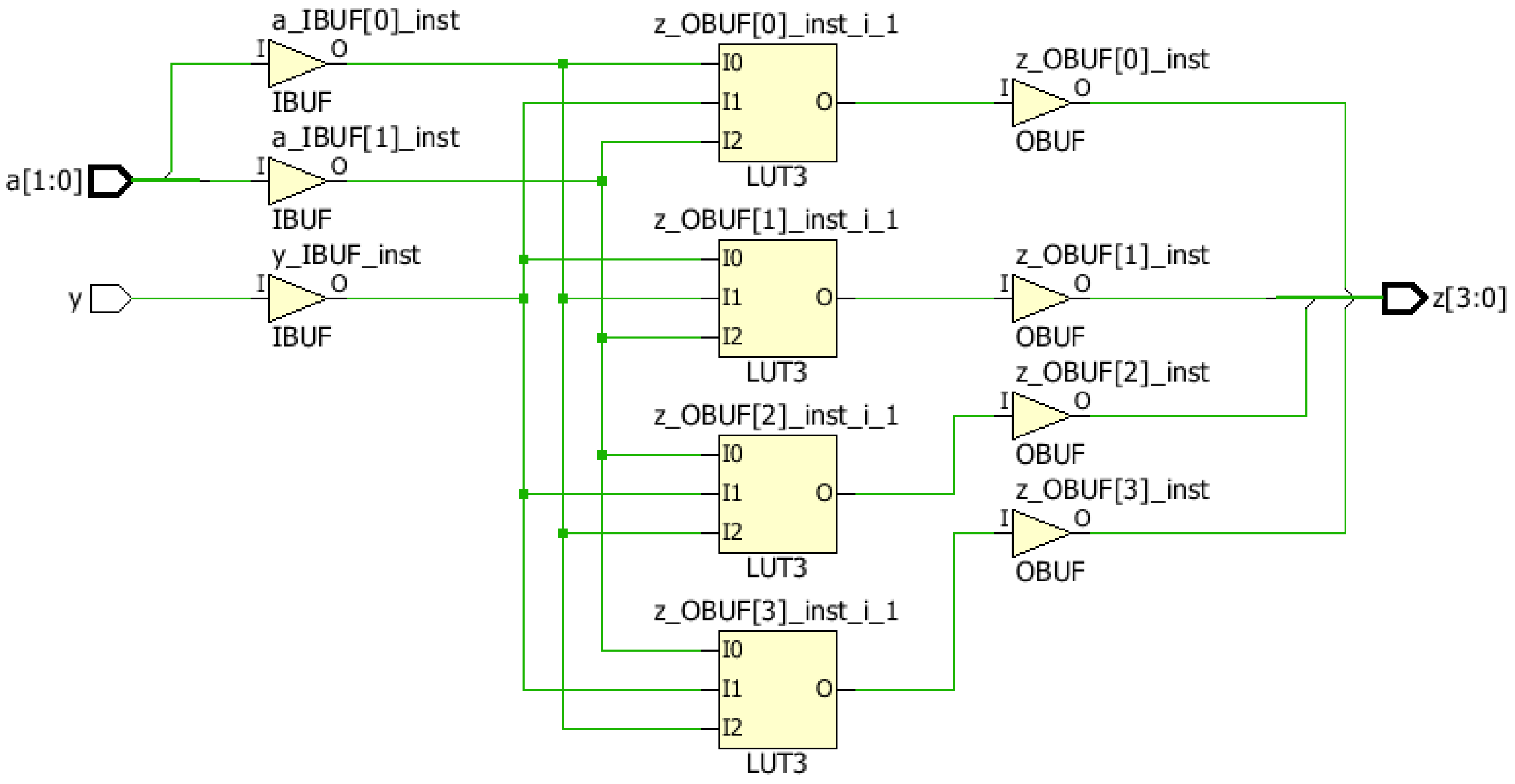
Lab 2 – solution - synth



Lab 4 – solution - RTL

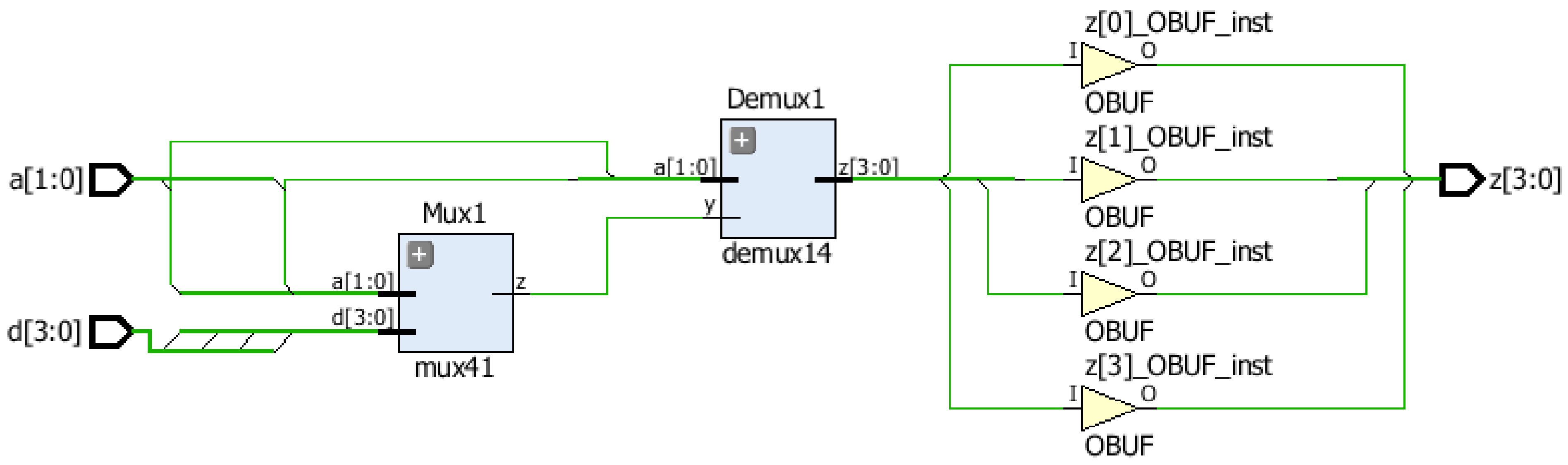


Lab 4 – solution - synth

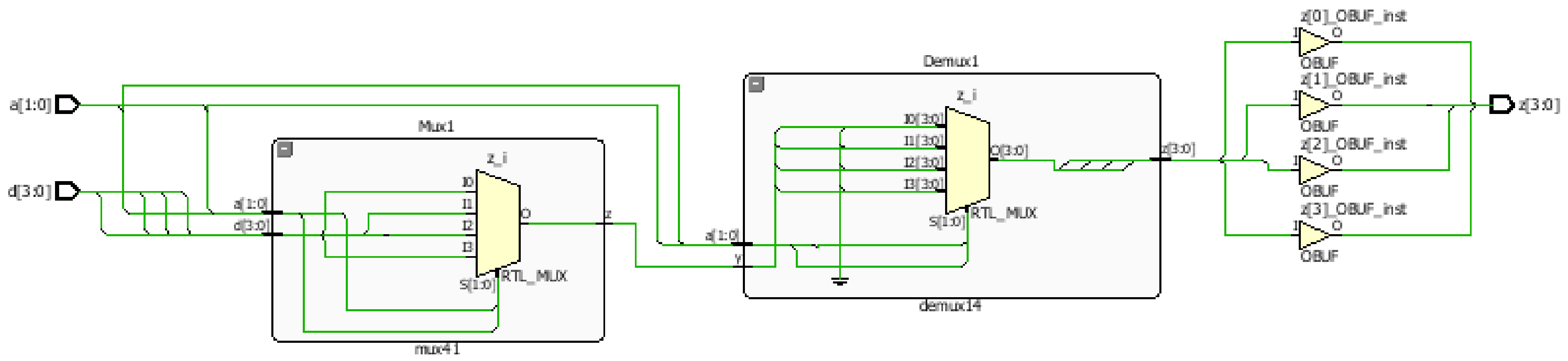


Tempus

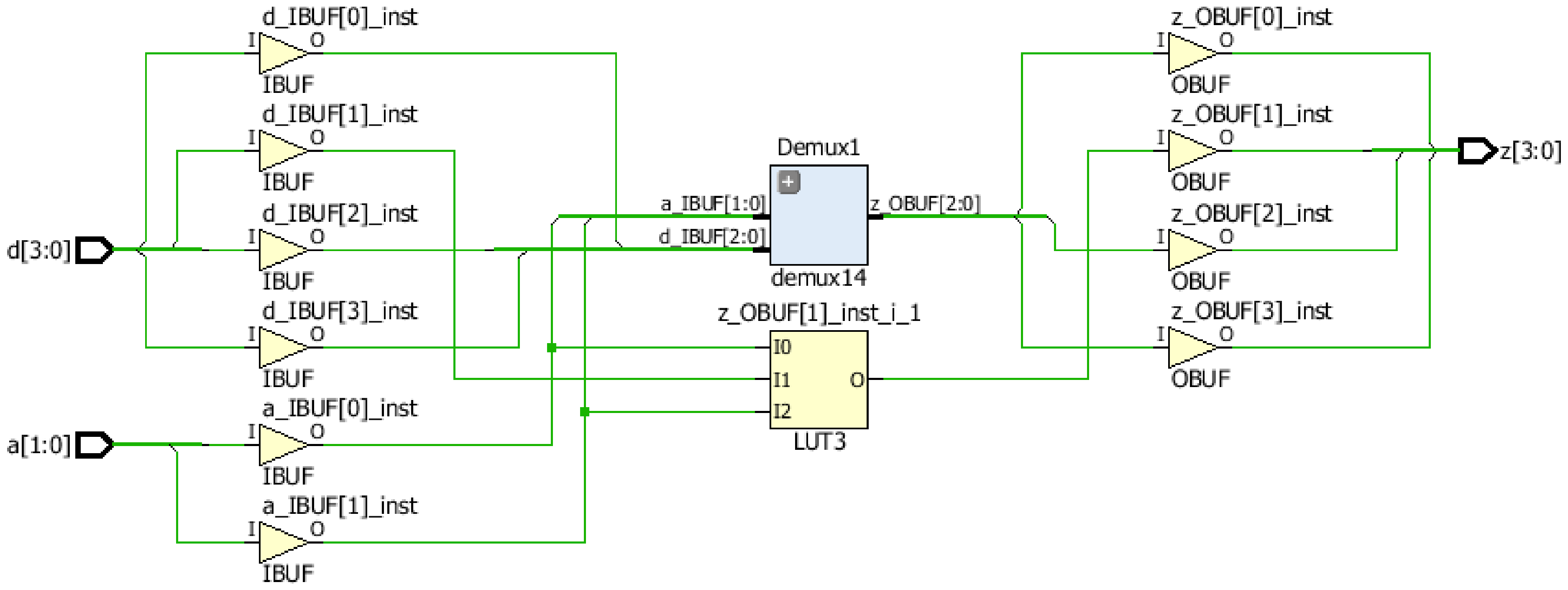
Lab 6 – solution - RTL



Lab 6 – solution - RTL

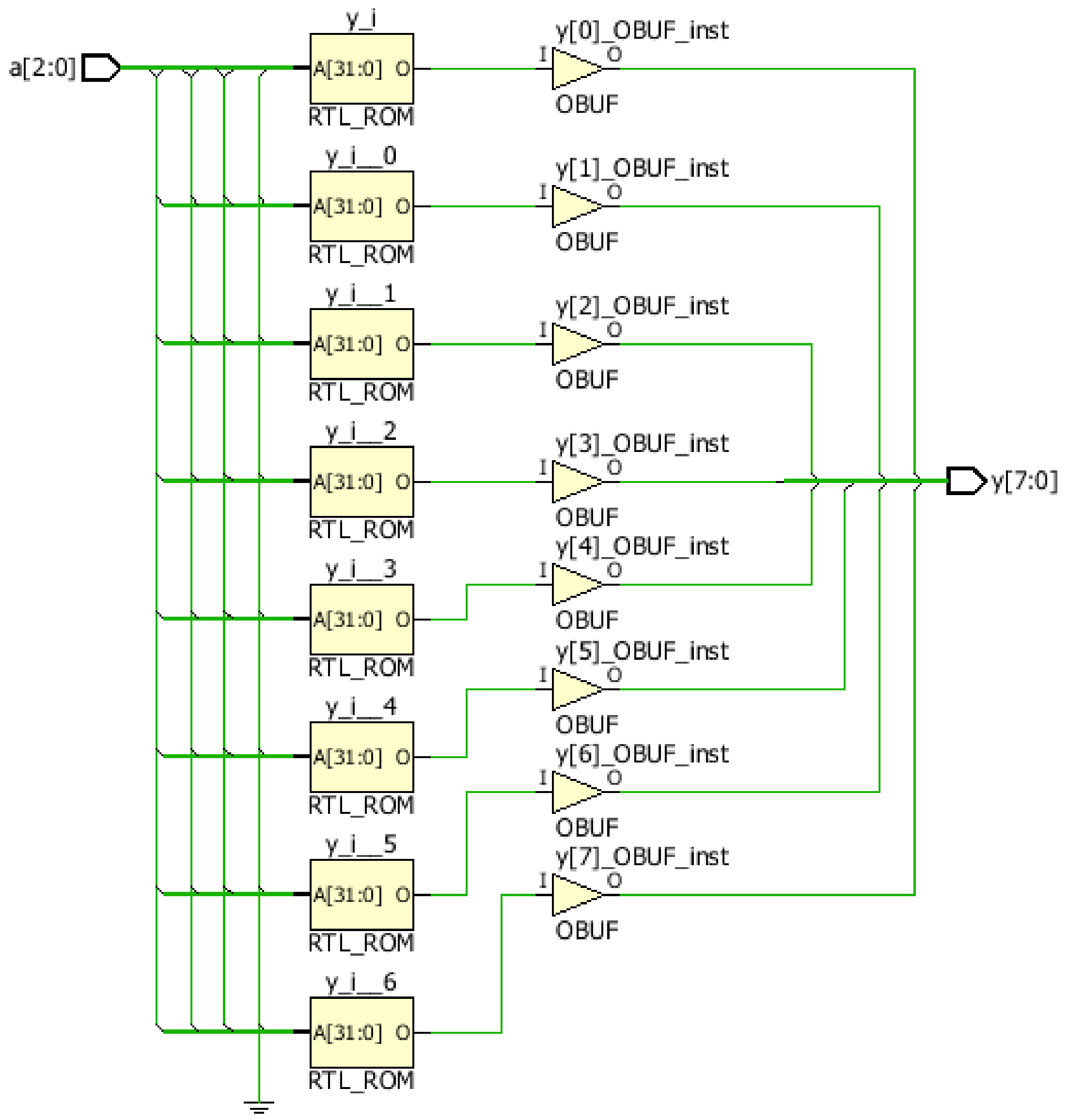


Lab 6 – solution - synth



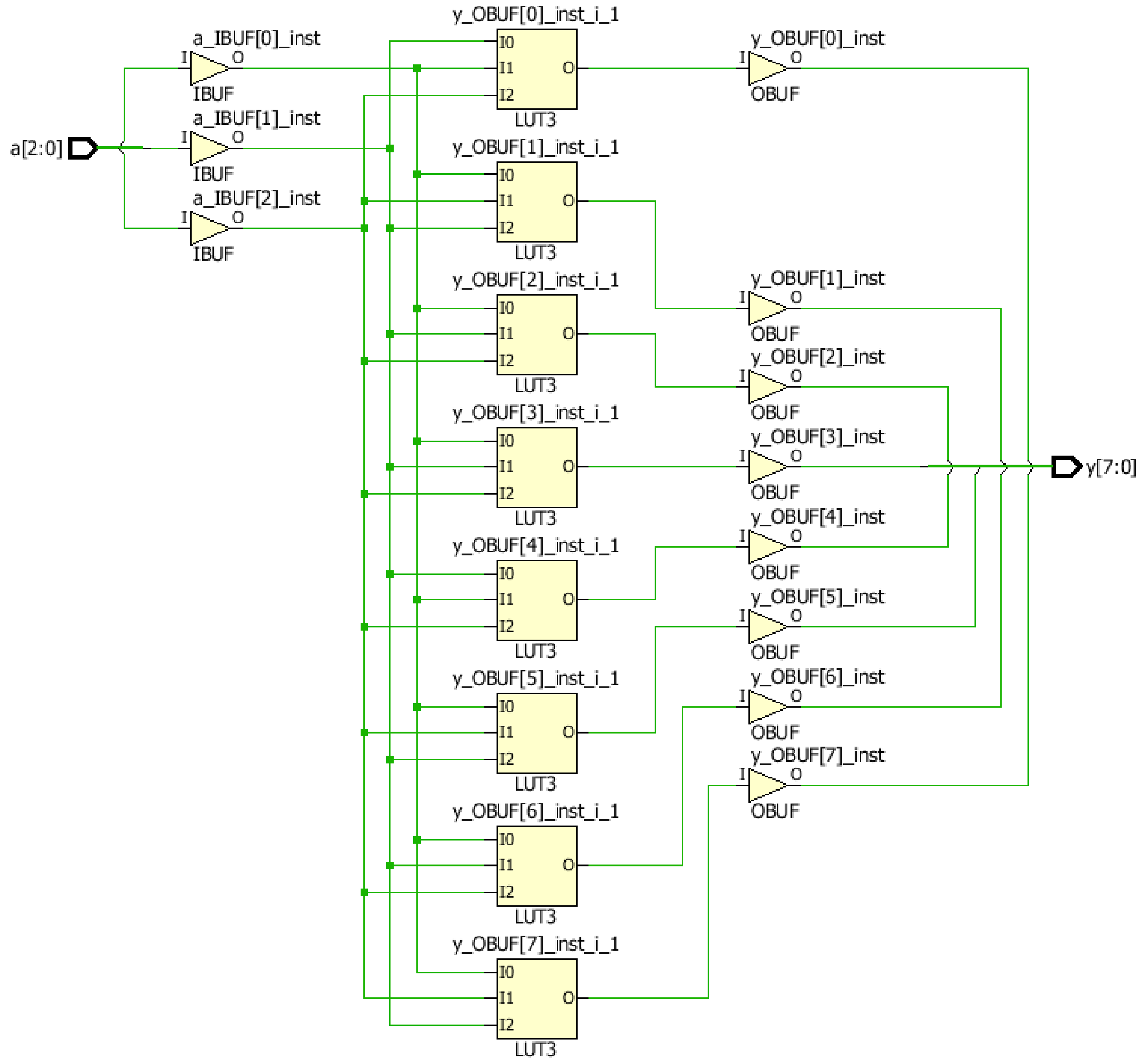
Tempus

Lab 8 – solution - RTL

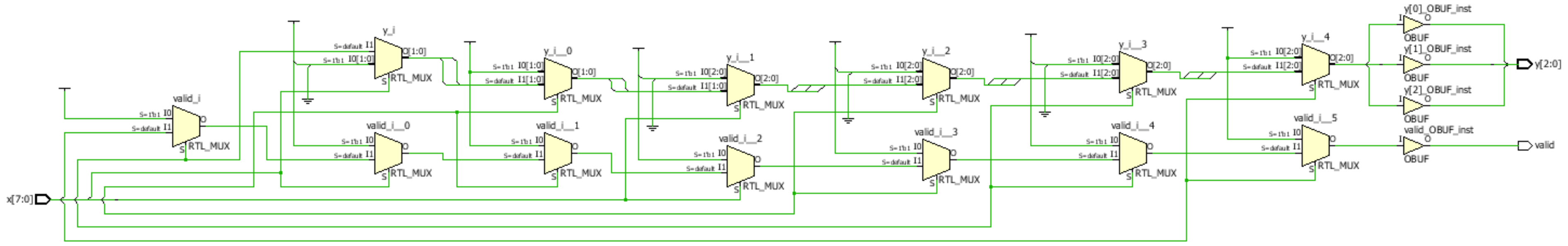


Tempus

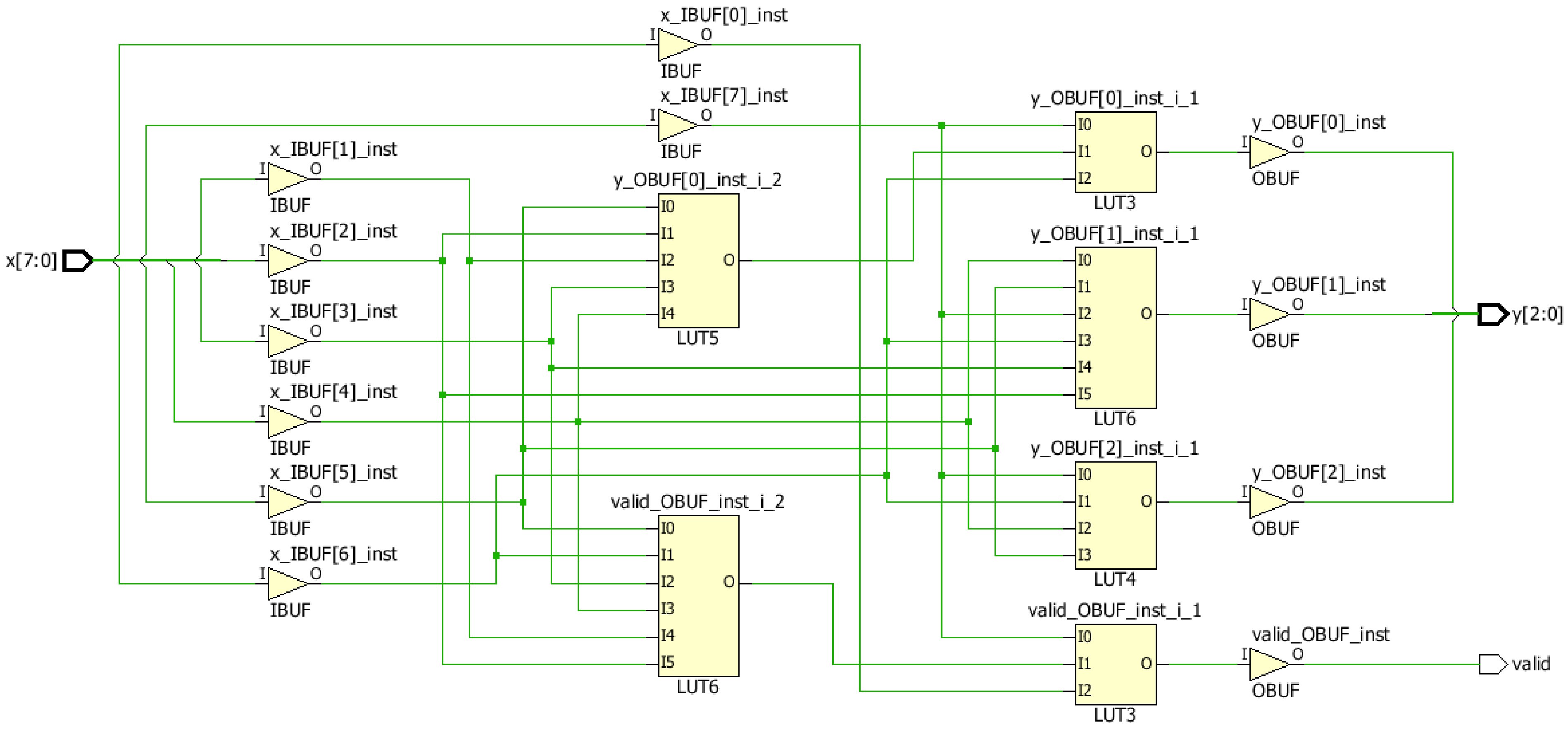
Lab 8 – solution - synth



Lab 9 – solution - RTL

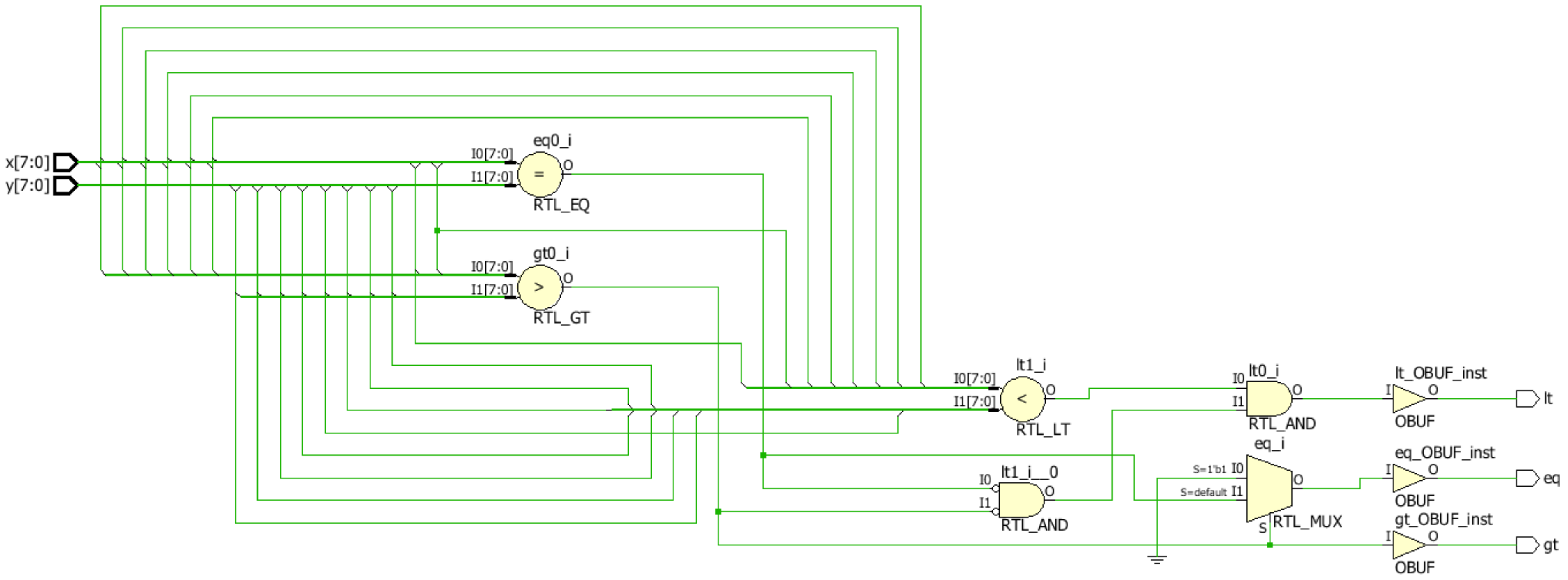


Lab 9 – solution - synth

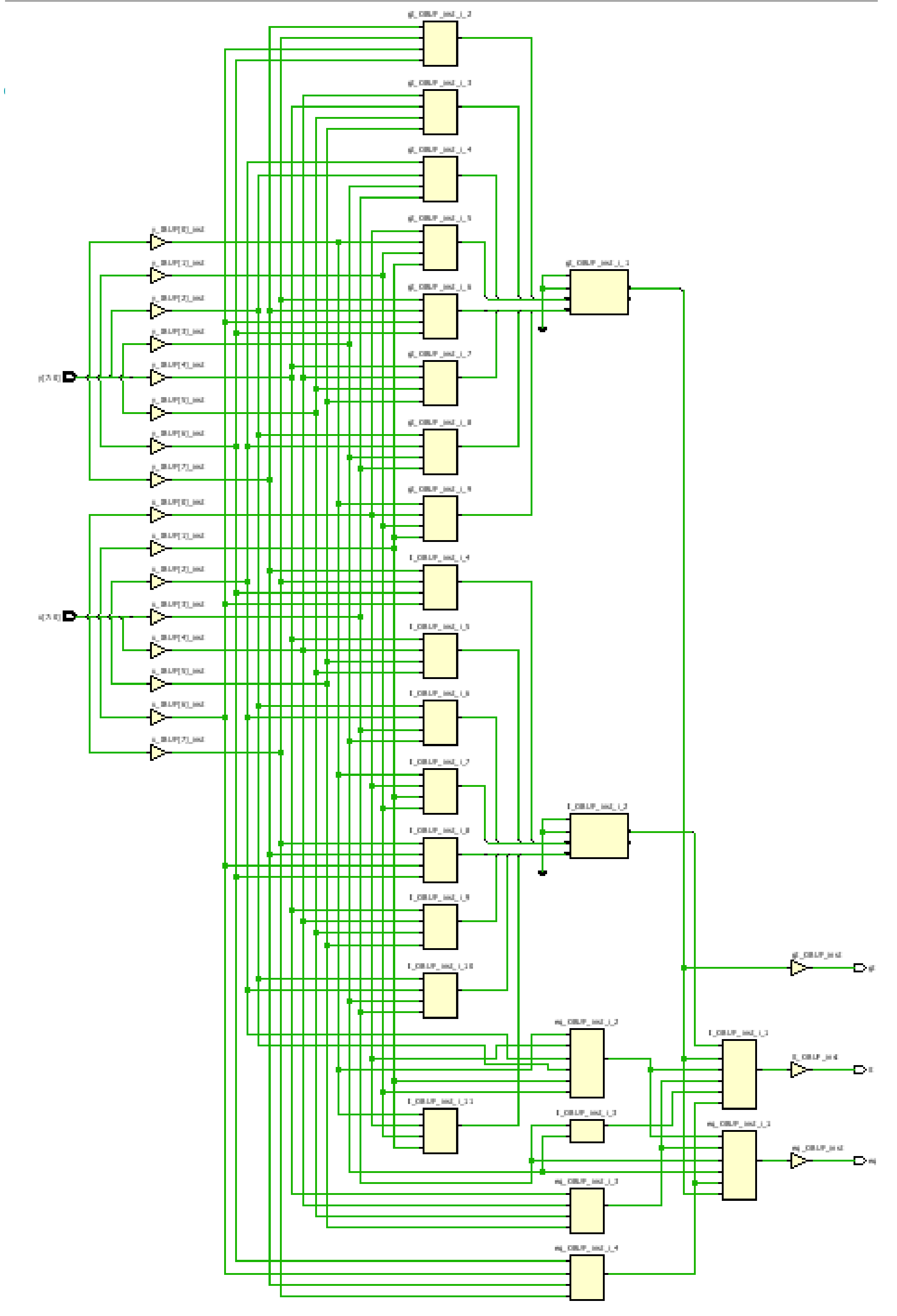


Tempus

Lab 11 – solution - RTL

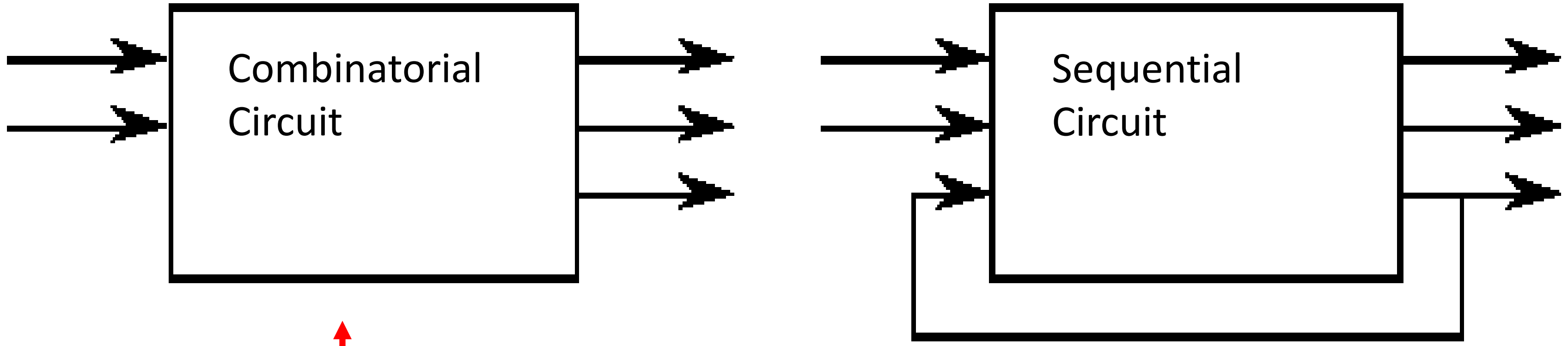


Lab 11 – solution synth



Sequential Logic

Sequential vs Combinatorial



New state determined by:

- A number of independent inputs

New state determined by:

- A number of independent inputs
- The current state of the system (through a memory element)

Sequential Logic

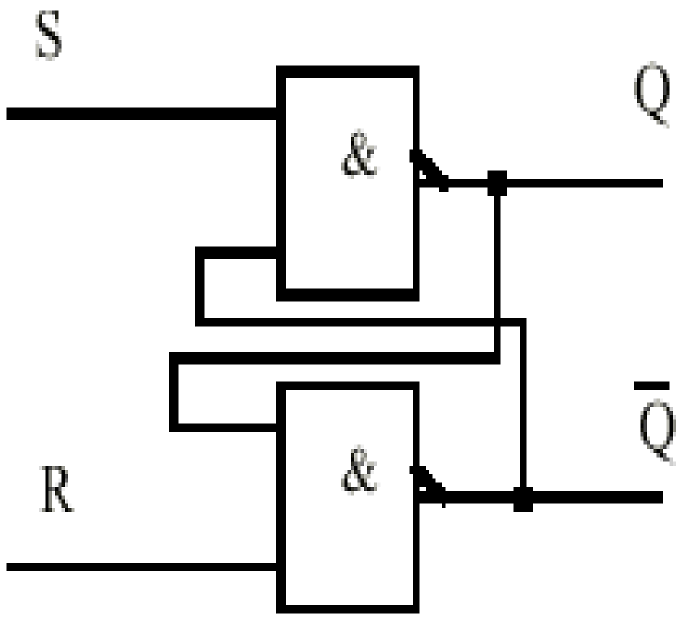
Sequential vs Combinatorial

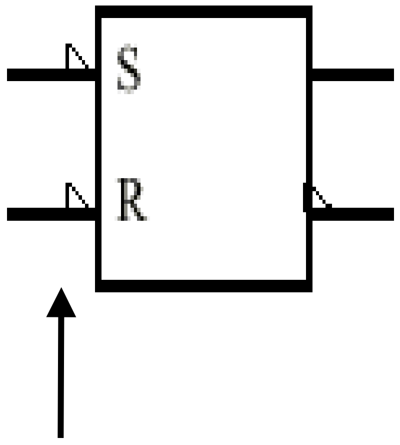
- In a combinatorial system: <u>all inputs</u> in the sensitivity list of a process
- In a sequential system: <u>only clock and asynchronous inputs</u> in the sensitivity list of a process

FLIPFLOP

- FLIPFLOP = one bit memory = base element of sequential circuits
- Amount of FF’s is determined by amount of states
  - One lamp on / of: one FF
  - Four bit binairy counter: 4 FFs.

SR-FLIPFLOP with NAND





Active low inputs!

SR-FLIPFLOP with NAND

S	R	Q_T	Q_{NT}	$/Q_{NT}$
0	0	x	1	1
0	1	x	1	0
1	0	x	0	1
1	1	0	0	1
1	1	1	1	0

Memory

Set

Reset

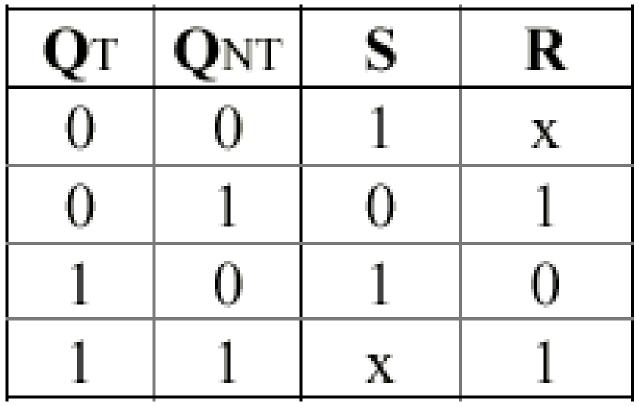
Forbidden state

S	R	Q_{NT}	$/Q_{NT}$	
0	0	1	1	Forbidden state
0	1	1	0	Set
1	0	0	1	Reset
1	1	Q_T	$/Q_T$	Memory

SR-FLIPFLOP with NAND

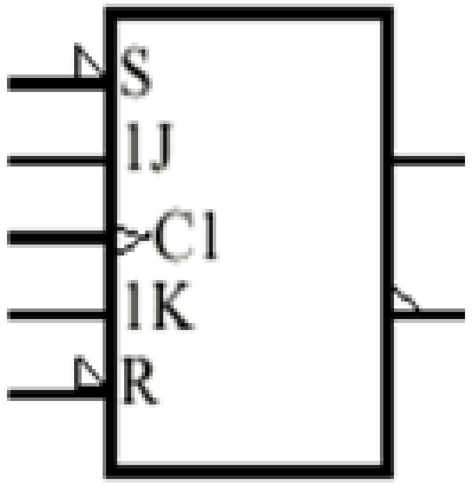
- Exitation table
  - Which inputs is needed to go from one state to the other?
  - For design!

Excitatietabel.

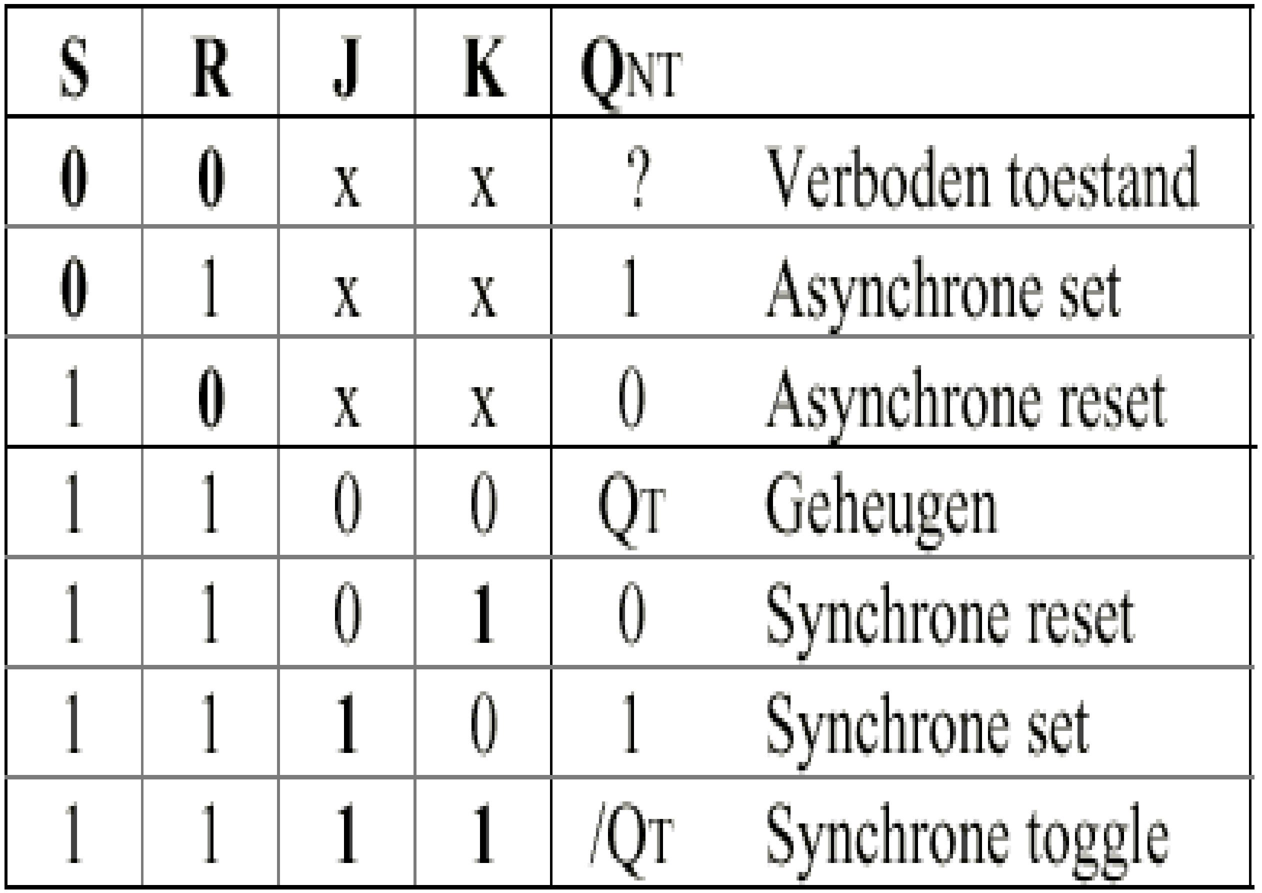


Q_T	Q_{NT}	S	R
0	0	1	x
0	1	0	1
1	0	1	0
1	1	x	1

JK-FLIPFLOP with asynchronous set and reset



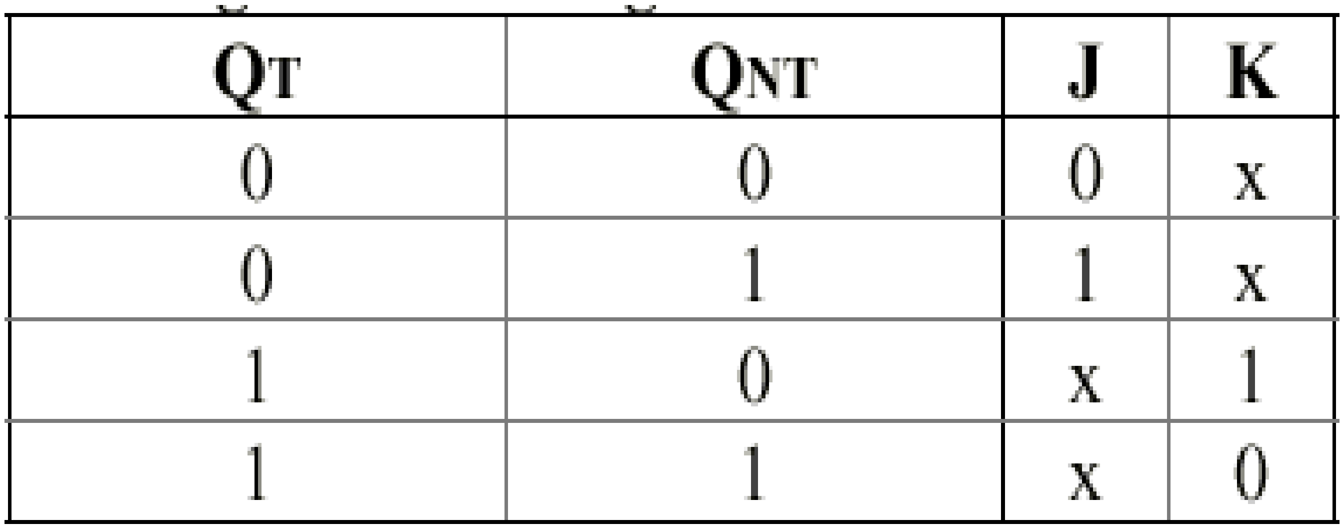
JK-FLIPFLOP with asynchronous set and reset



S	R	J	K	Q_{NT}	
0	0	x	x	?	Verboden toestand
0	1	x	x	1	Asynchrone set
1	0	x	x	0	Asynchrone reset
1	1	0	0	Q_T	Geheugen
1	1	0	1	0	Synchrone reset
1	1	1	0	1	Synchrone set
1	1	1	1	$/Q_T$	Synchrone toggle

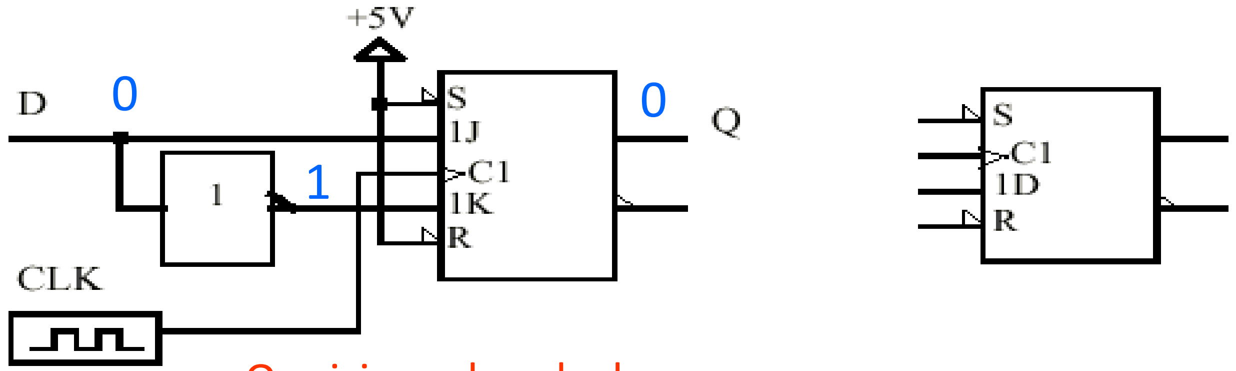
JK-FLIPFLOP

- Exitation table

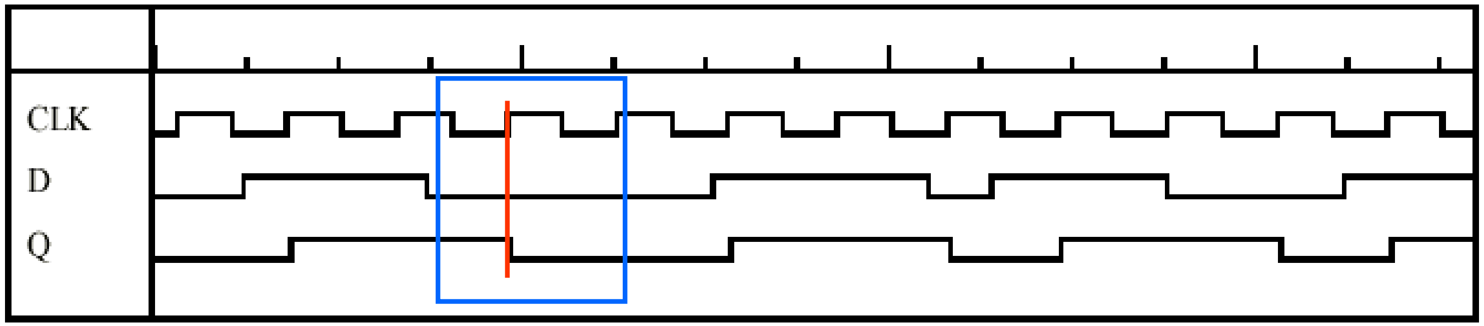


Q_T	Q_{NT}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

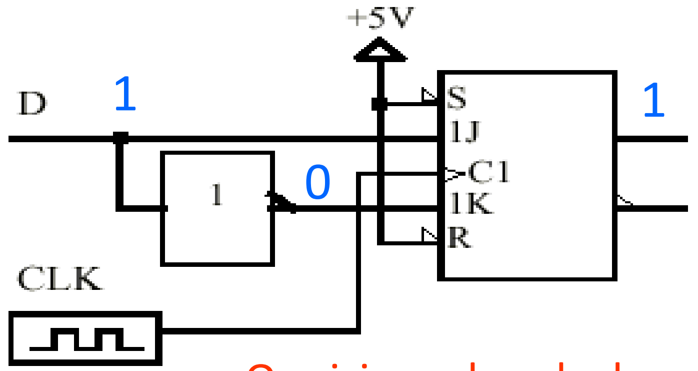
D-FLIPFLOP

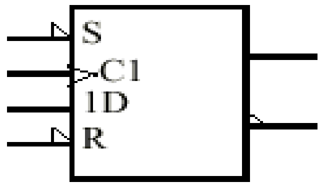


On rising edge clock

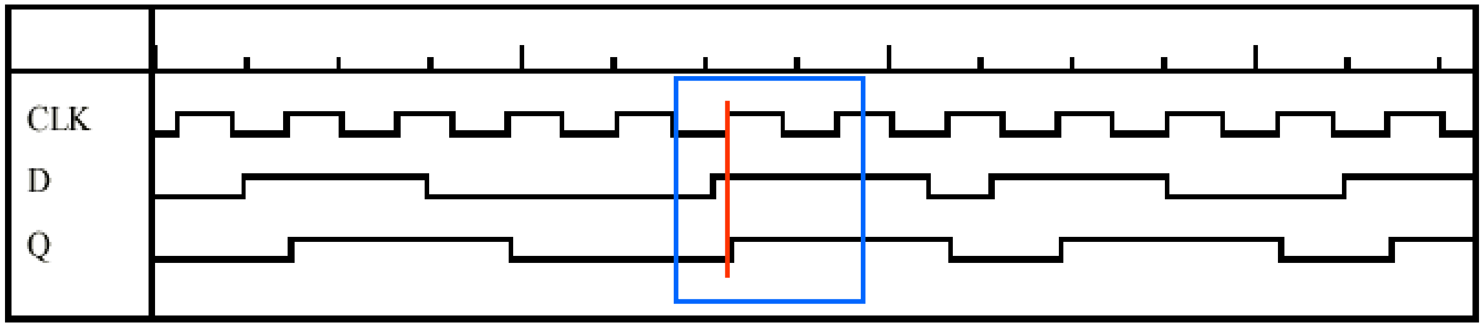


D-FLIPFLOP

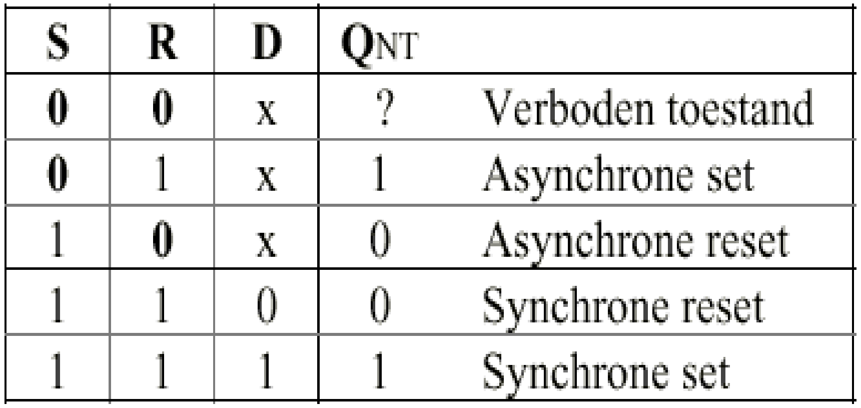




On rising edge clock

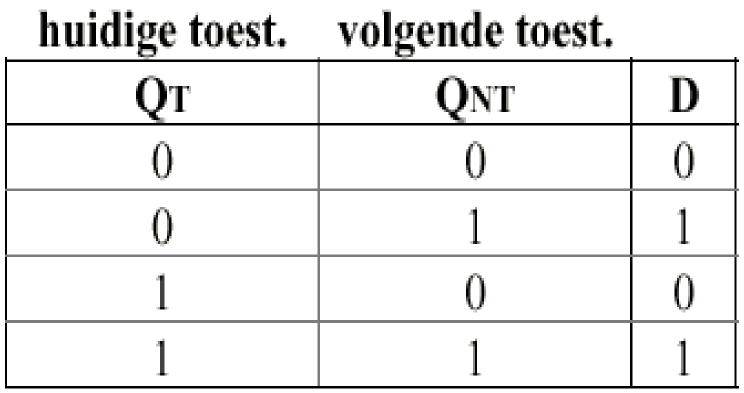


D-FLIPFLOP



S	R	D	Q_{NT}	
0	0	x	?	Verboden toestand
0	1	x	1	Asynchrone set
1	0	x	0	Asynchrone reset
1	1	0	0	Synchrone reset
1	1	1	1	Synchrone set

Exitation table



huidige toest.	volgende toest.	
Q_T	Q_{NT}	D
0	0	0
0	1	1
1	0	0
1	1	1

D = 0: => J = 0 and K = 1

D = 1: => J = 1 and K = 0

Lab 13

Make a edge-triggered D-flipflop

- Simulate: use clock process

```
-- Clock period definitions
  constant clk_period : time := 10 ns;

 -- Clock process definitions
   clk_process :process
   begin
                  clk <= '0';
                  wait for clk_period/2;
                  clk <= '1';
                  wait for clk_period/2;
   end process;
```

Lab 13

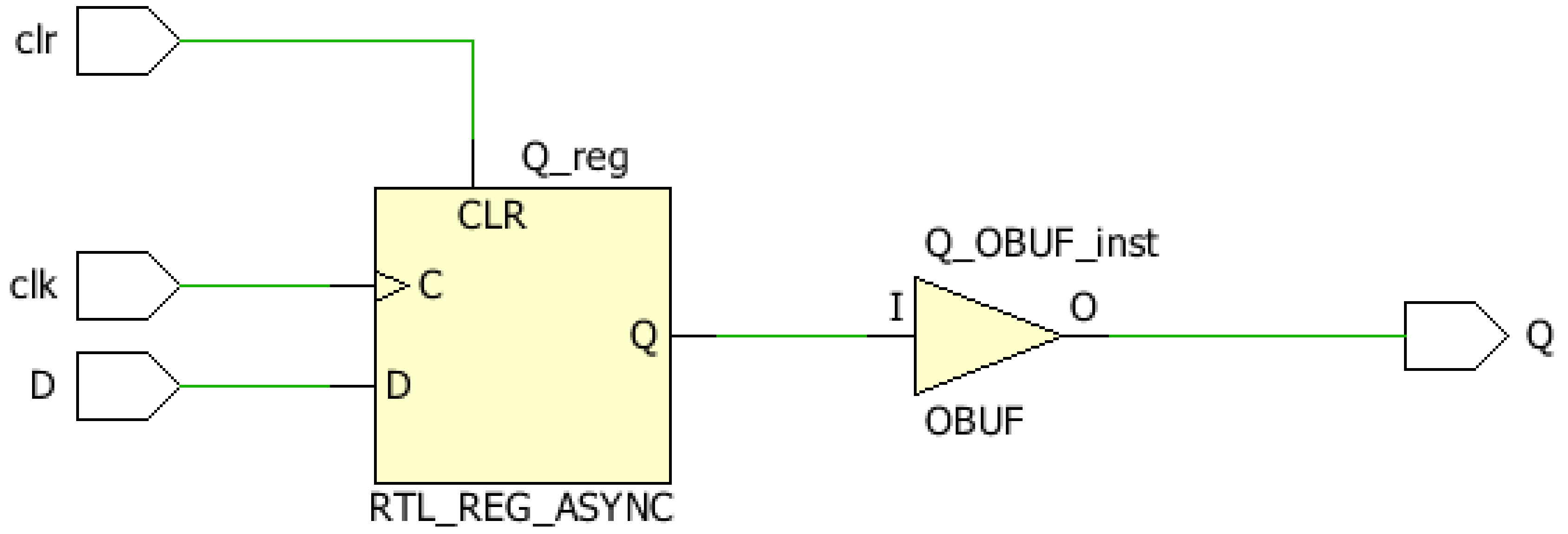
Make a edge-triggered D-flipflop

- Check RTL and Technology
- Explain

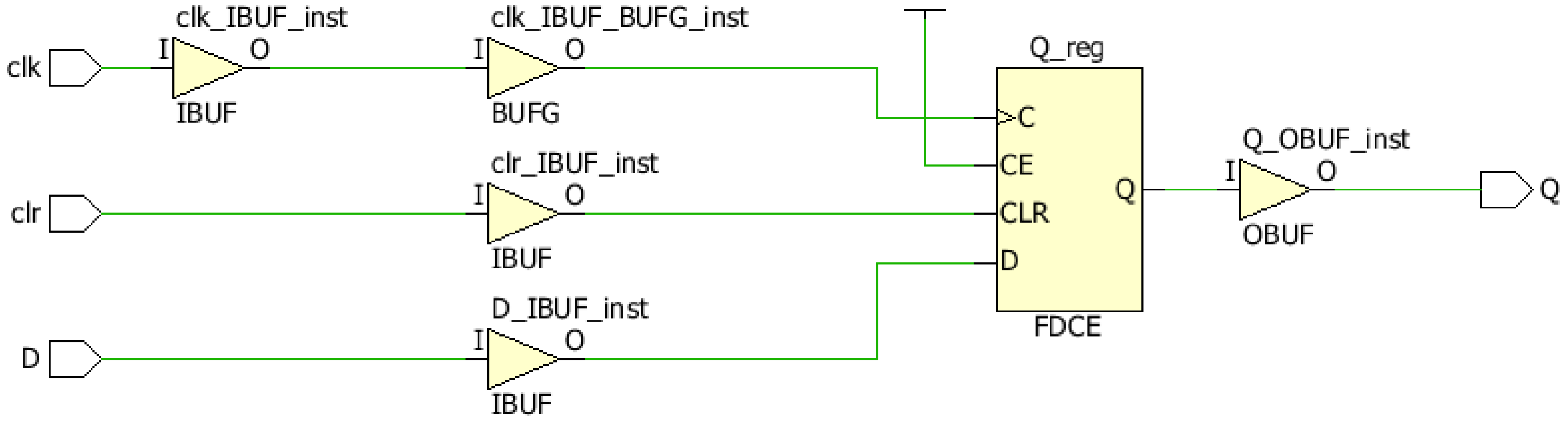
Lab 13 - solution

```
entity Dff is
          port(
                    clk : in STD_LOGIC;
                    clr : in STD_LOGIC;
                    D : in STD_LOGIC;
                    q : out STD_LOGIC
            );
end Dff;

architecture behavior of Dff is
begin
  process(clk, clr)
  begin
          if(clr = '1') then
                    q <= '0';
          elsif(rising_edge(clk)) then
                    q <= D;
          end if;
  end process;
end behavior;
```

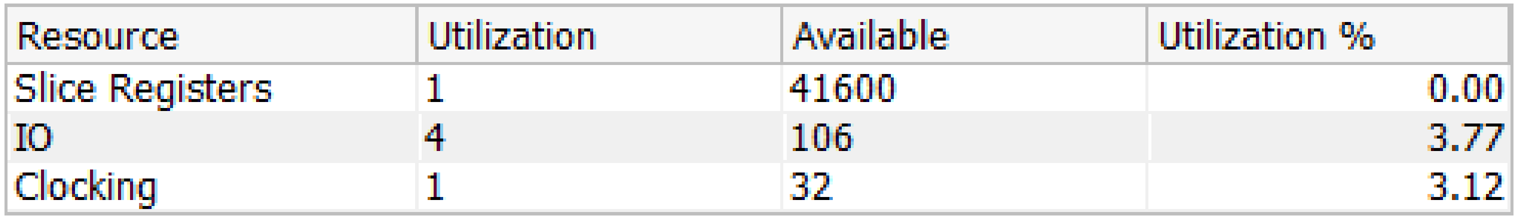
Lab 13 – solution - RTL



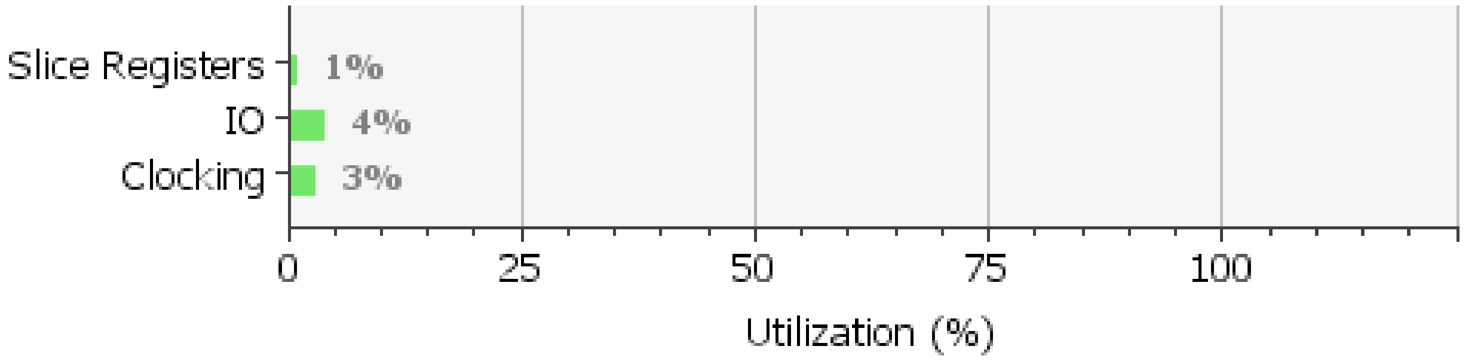
Lab 13 – solution - Synth



Lab 13 – solution - Utilization



Resource	Utilization	Available	Utilization %
Slice Registers	1	41600	0.00
IO	4	106	3.77
Clocking	1	32	3.12



Lab 13 – solution tb

```
entity Lab13_tb is
end Lab13_tb;

architecture Behavioral of Lab13_tb is

component Lab13 is
   Port ( clk : in STD_LOGIC;
          D : in STD_LOGIC;
          clr : in STD_LOGIC;
          Q : out STD_LOGIC);
end component Lab13;

signal clk, D, clr, Q: std_logic;
constant clk_period: time := 100 ns;
```

Lab 13 – solution tb

```
begin

UUT: Lab13
Port map (clk => clk, D => D, clr => clr, Q => Q);

clk_process: process
begin
   clk <= '0';
   wait for clk_period/2;
   clk <= '1';
   wait for clk_period/2;
end process;
```

Lab 13 – solution tb

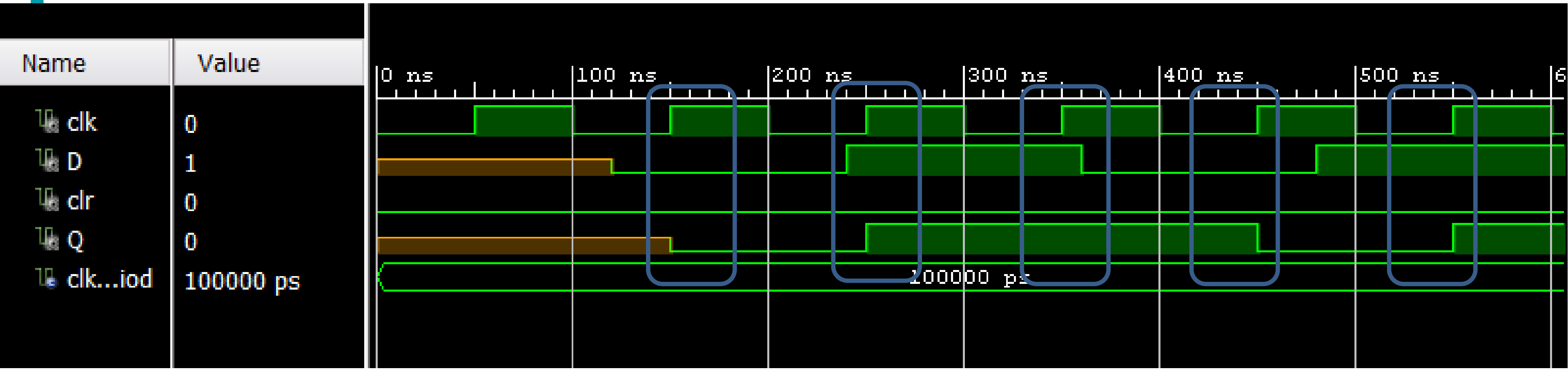
```
stim_process: process
begin
    clr <= '0';
    wait for 120ns;
    D <= '0';
    wait for 120ns;
    D <= '1';
    wait for 120ns;
    D <= '0';
    wait for 120ns;
    D <= '1';
```

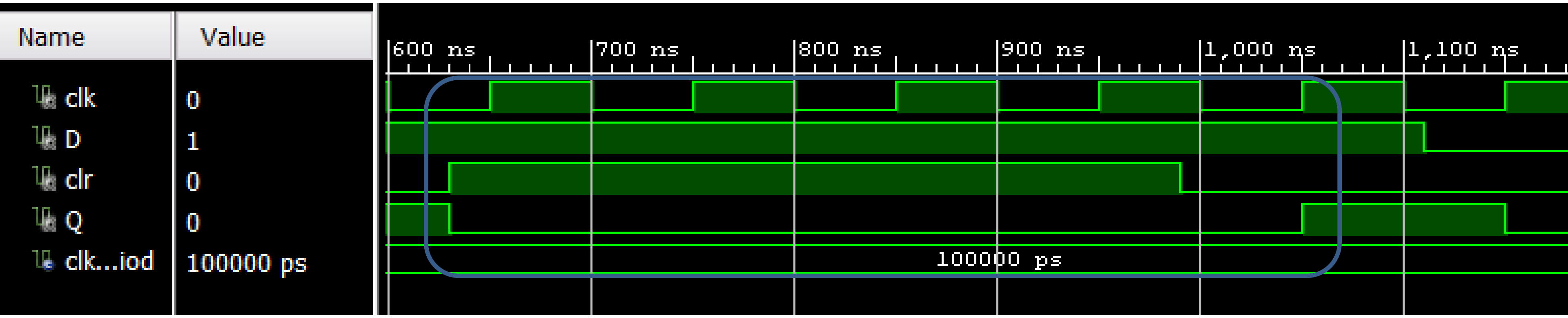
Lab 13 – solution tb

```
    wait for 120ns;
    D <= '1';
    wait for 30ns;
    clr <= '1';
    wait for 120ns;
    D <= '1';
    wait for 120ns;
    D <= '1';
    wait for 120ns;
    D <= '1';
end process;
end Behavioral;
```

Tempus

Lab 13 – solution tb

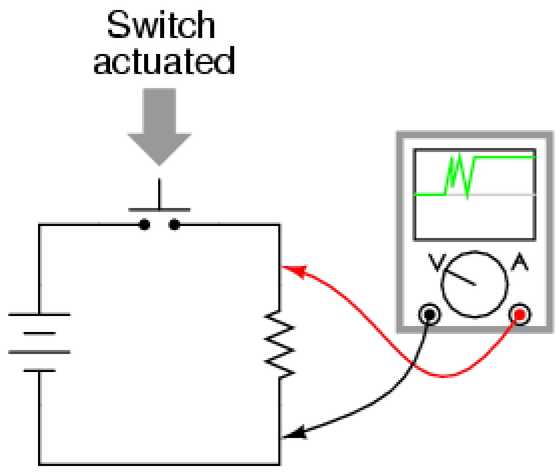


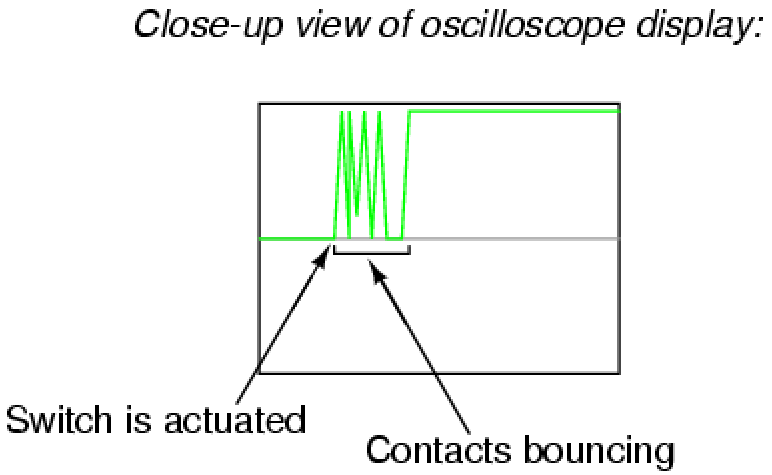


Signal conditioning

- Debouncing
  - Often sequential circuits are depended on signal edges.
  - Mechanical switches and buttons are prone to bounce: this produces more than one edge

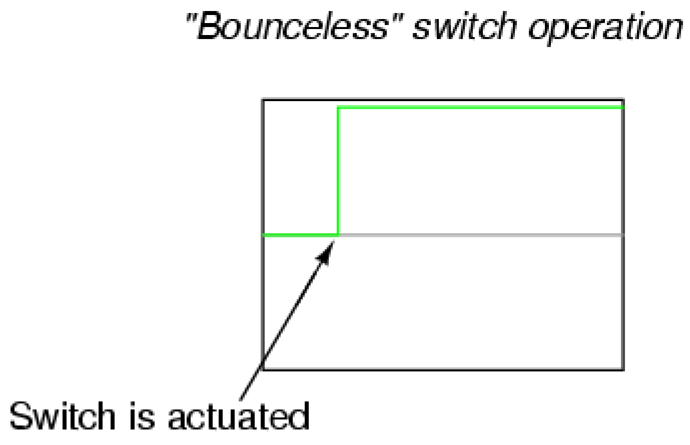
contacts will "bounce" upon closure for a period of milliseconds before coming to a full rest and providing unbroken contact





Signal conditioning

- Debouncing
  - For proper operation we need one clean edge



Signal conditioning

- Debouncing – VHDL solutions

```
entity debouncer is
		Port (	sig_in : in  STD_LOGIC;
				clk : in  STD_LOGIC;
				sig_out : out  STD_LOGIC);
end debouncer;

architecture debounce of debouncer is
		signal Q1, Q2, Q3 : std_logic;
begin
process(clk) begin
		if (clk'event and clk = '1') then
				Q1 <= sig_in;
				Q2 <= Q1;
				Q3 <= Q2;
		end if;
end process;
sig_out <= Q1 and Q2 and Q3;
end debounce;
```

Signal conditioning

- Debouncing – one-shot pulse

```
entity debouncer is
            Port (      sig_in : in  STD_LOGIC;
                        clk : in  STD_LOGIC;
                        sig_out : out  STD_LOGIC);
end debouncer;

architecture debounce of debouncer is
            signal Q1, Q2, Q3 : std_logic;
begin
process(clk) begin
            if (clk'event and clk = '1') then
                        Q1 <= sig_in;
                        Q2 <= Q1;
                        Q3 <= Q2;
            end if;
end process;
sig_out <= Q1 and Q2 and (not Q3);
end debounce;
```

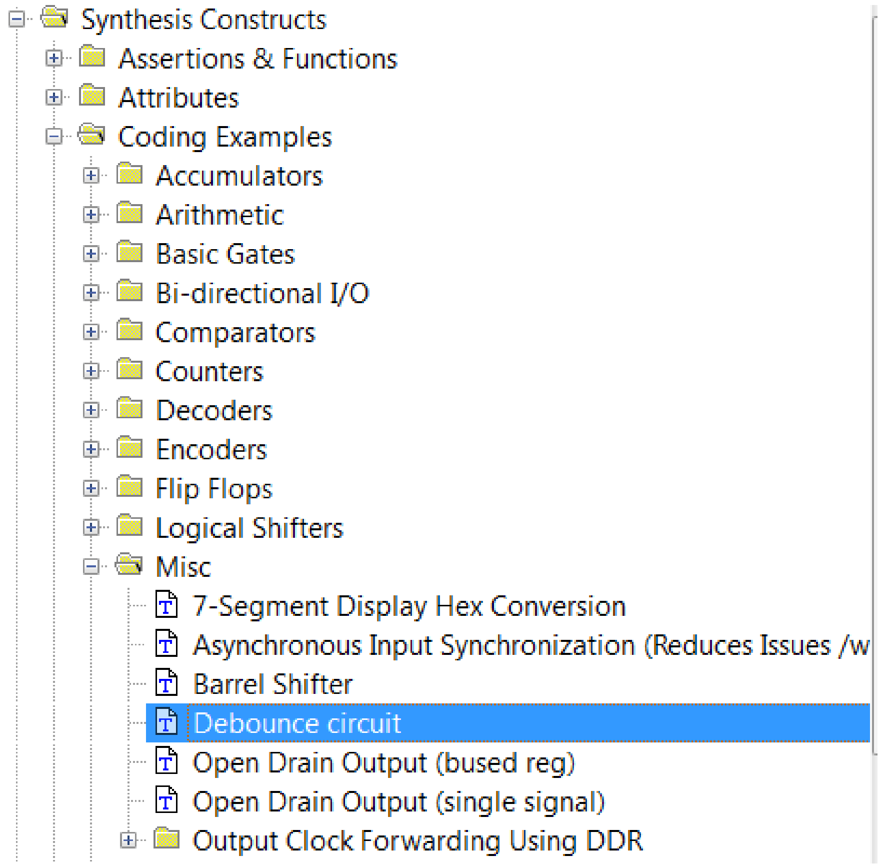
Signal conditioning

- Debouncing – active low

```
entity debouncer is
        Port (    sig_in : in  STD_LOGIC;
                  clk : in  STD_LOGIC;
                  sig_out : out  STD_LOGIC);
end debouncer;

architecture debounce of debouncer is
        signal Q1, Q2, Q3 : std_logic;
begin
process(clk) begin
        if (clk'event and clk = '1') then
                  Q1 <= sig_in;
                  Q2 <= Q1;
                  Q3 <= Q2;
        end if;
end process;
sig_out <= Q1 or Q2 or Q3;
end debounce;
```

Signal conditioning

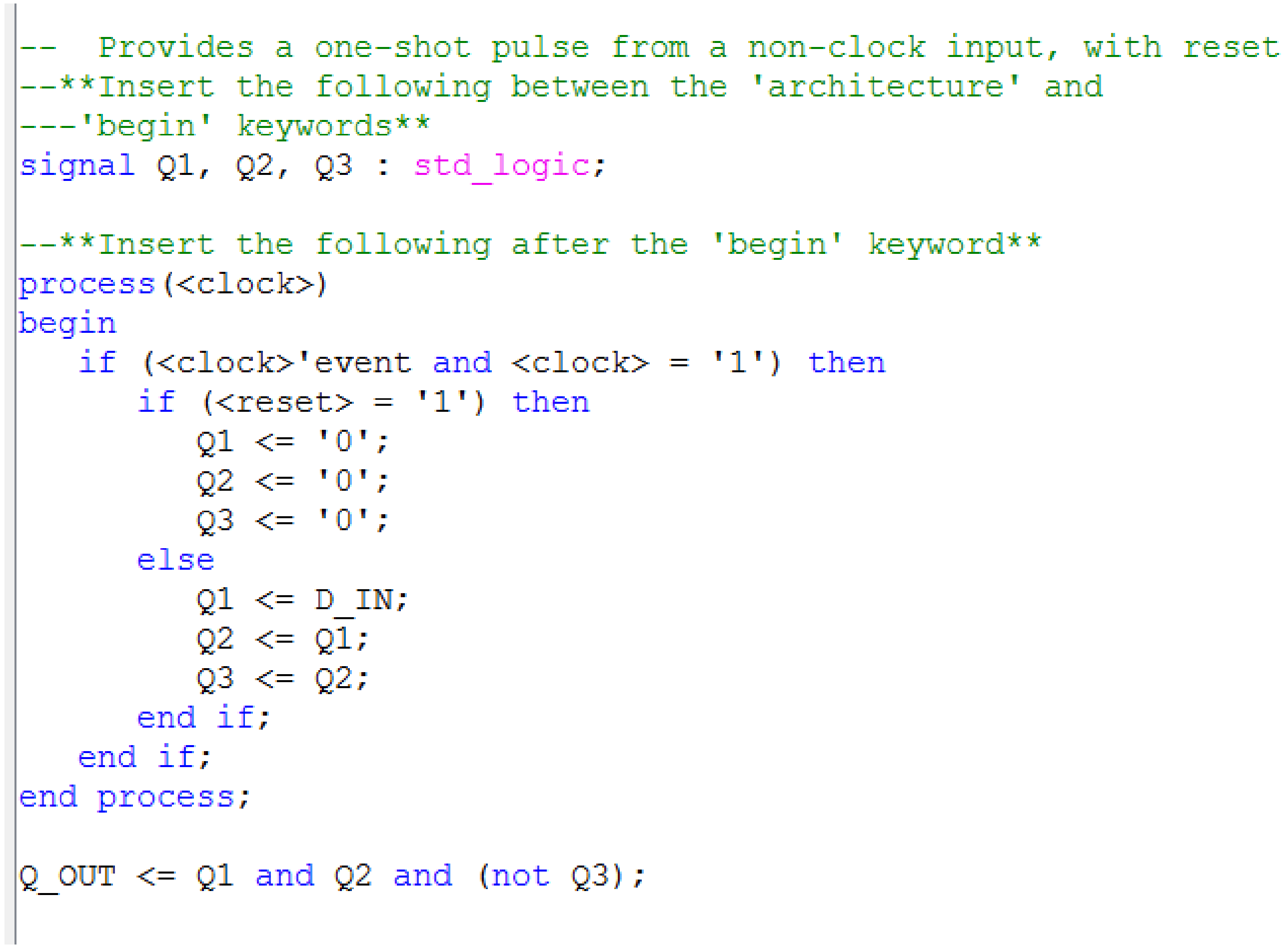
- Debouncing – language templates
  - Synthesis Constructs => Coding Examples => Misc



Synthesis Constructs
- Assertions & Functions
- Attributes
- Coding Examples
  - Accumulators
  - Arithmetic
  - Basic Gates
  - Bi-directional I/O
  - Comparators
  - Counters
  - Decoders
  - Encoders
  - Flip Flops
  - Logical Shifters
  - Misc
    - 7-Segment Display Hex Conversion
    - Asynchronous Input Synchronization (Reduces Issues /w
    - Barrel Shifter
    - Debounce circuit
    - Open Drain Output (bused reg)
    - Open Drain Output (single signal)
    - Output Clock Forwarding Using DDR

Signal conditioning

- Debouncing – language templates



```
--  Provides a one-shot pulse from a non-clock input, with reset
--**Insert the following between the 'architecture' and
---'begin' keywords**
signal Q1, Q2, Q3 : std_logic;

--**Insert the following after the 'begin' keyword**
process(<clock>)
begin
   if (<clock>'event and <clock> = '1') then
      if (<reset> = '1') then
         Q1 <= '0';
         Q2 <= '0';
         Q3 <= '0';
      else
         Q1 <= D_IN;
         Q2 <= Q1;
         Q3 <= Q2;
      end if;
   end if;
end process;

Q_OUT <= Q1 and Q2 and (not Q3);
```

Signal conditioning

- Clock divider
  - Often the system clock is to fast to test the output of the VHDL module
  - We divide the clock to make it slower by counting pulses

Signal conditioning

- Clock divider – VHDL solution

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity clk_div is
    Port (              clk : in  STD_LOGIC;
                        div : in std_logic_vector (1 downto 0);
                        clk_slow : out  STD_LOGIC);
end clk_div;
```

Signal conditioning

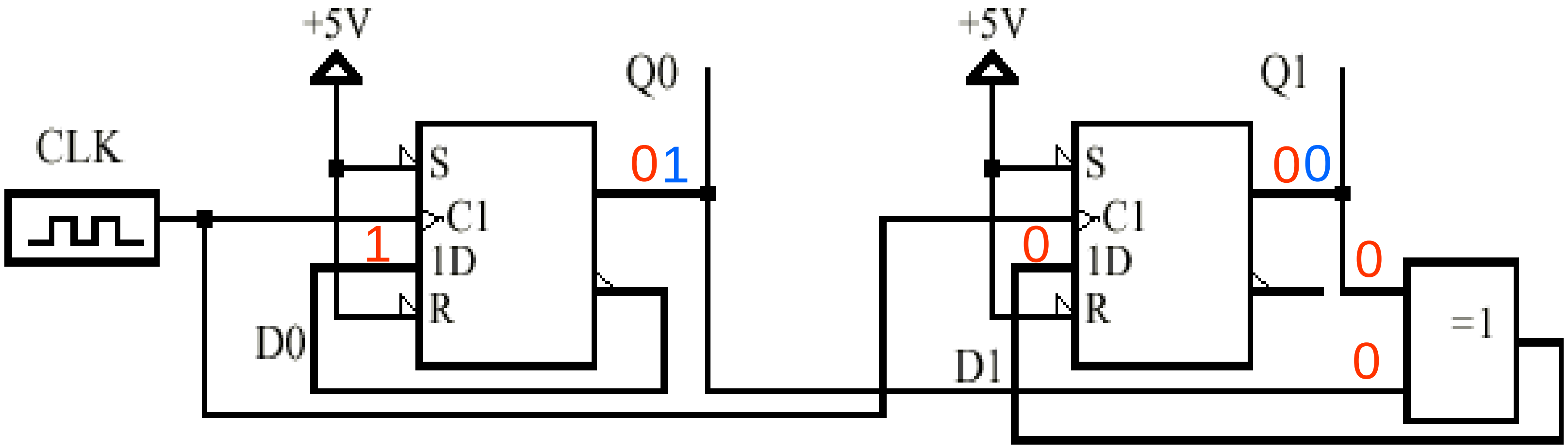
- Clock divider – VHDL solution

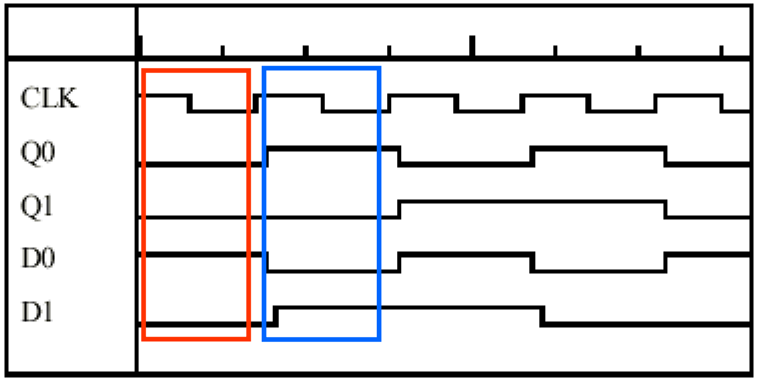
```
architecture Behavioral of clk_div

signal i: integer range 0 to 50000000:=0;
signal int_clk: std_logic:= '0';
signal base: integer range 0 to 25000000:= 50000000;
```

Signal conditioning

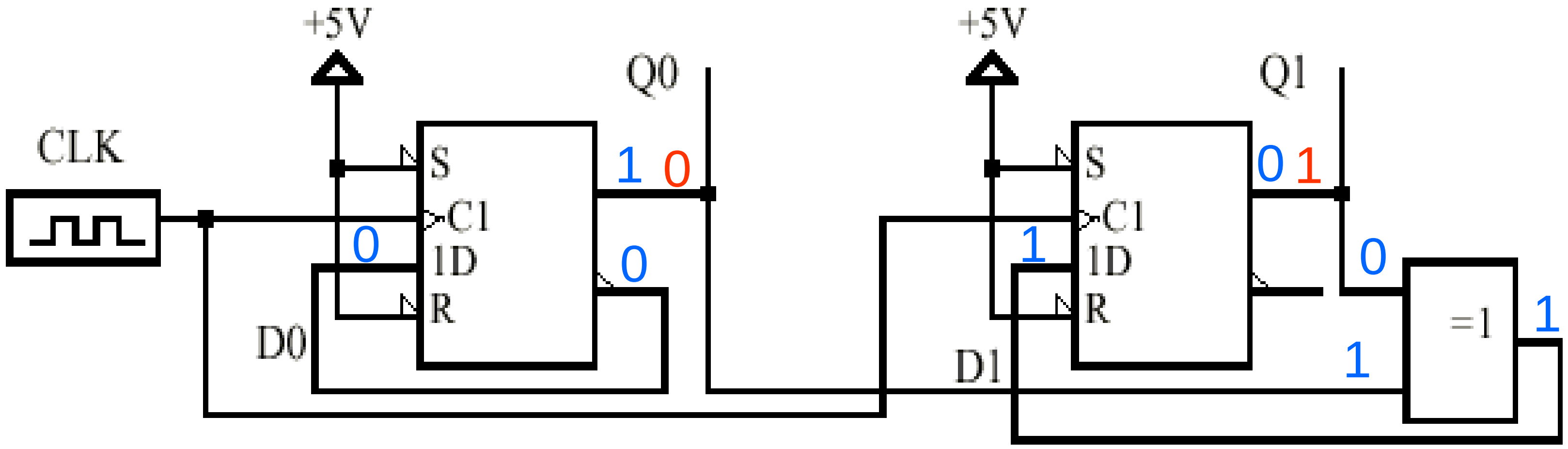
```
begin
process (clk, div) begin
        if rising_edge (clk) then
                i <= i + 1;
                if (i = base) then
                        i <= 0;
                        int_clk <= not int_clk;
                end if;
                case (div) is
                        when "00" => base <= 50000000;
                        when "01" => base <= 25000000;
                        when "10" => base <= 5000000;
                        when others => base <= 500000;
                end case;
        end if;
end process;

clk_slow <= int_clk;

end Behavioral;
```

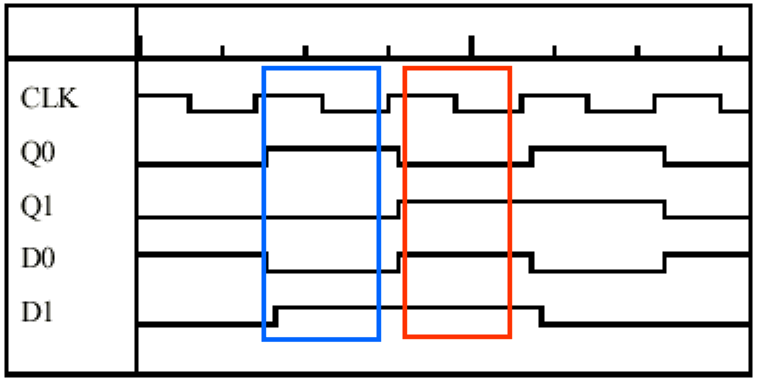
2-bit synchronous counter



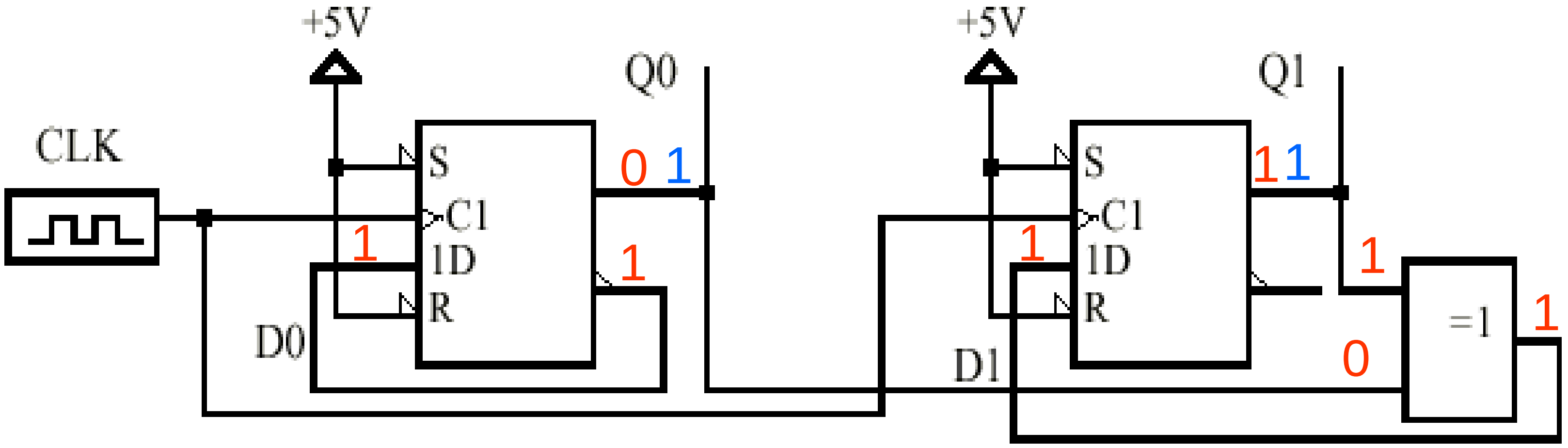


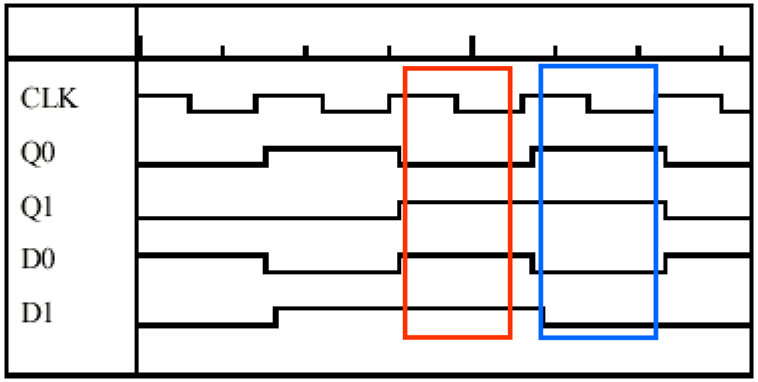
2-bit synchronous counter





2-bit synchronous counter





Lab 14

Make a Divide-by 2 Counter

- Simulate
- Add a clock divider to slow down the clock

Lab 14 - solution

```
entity div2cnt is
		port(
			clk : in STD_LOGIC;
			clr : in STD_LOGIC;
			q0 : out STD_LOGIC
		 );
end div2cnt;
```

Lab 14 - solution

```
architecture behavior of div2cnt is
signal D, q: STD_LOGIC;
begin

D <= not q;

  -- D Flip-flop
  process(clk, clr)
  begin
                  if(clr = '1') then
                                  q <= '0';
                  elsif (clk'event and clk = '1') then
                                  q <= D;
                  end if;
  end process;

  q0 <= q;
end behavior;
```

Lab 14 - tb

```
entity div2cnt_tb is
--  Port ( );
end div2cnt_tb;

architecture Behavioral of div2cnt_tb is

component div2cnt is
    Port ( clk : in STD_LOGIC;
           clr : in STD_LOGIC;
           q0 : out STD_LOGIC);
end component div2cnt;

signal clk : STD_LOGIC;
signal clr :  STD_LOGIC;
signal q0 :  STD_LOGIC;
constant clk_period: time := 10 ns;
```

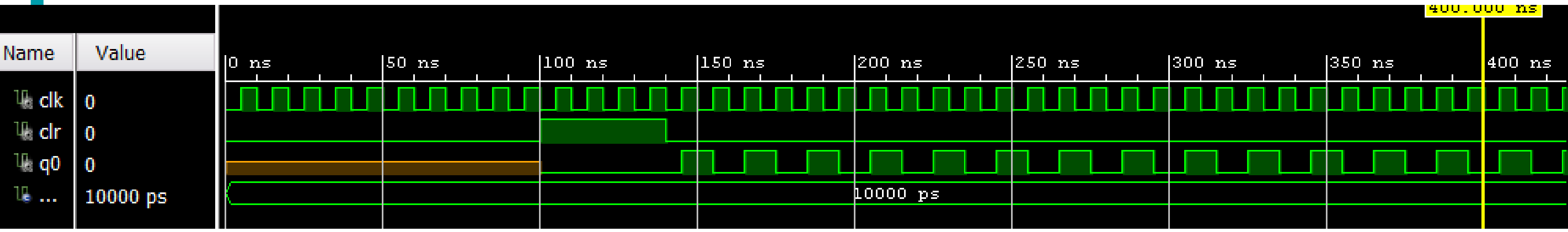
Lab 14 - tb

```
begin

UUT: div2cnt
Port map ( clk => clk, clr => clr, q0=> q0);

clk_proc: process begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
end process;
```

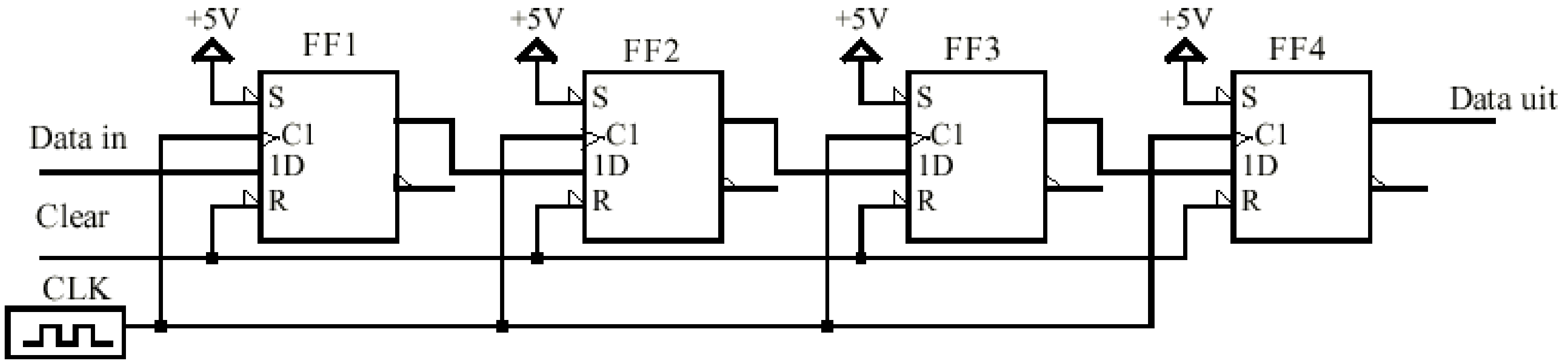
Lab 14 - tb

```
stim_proc: process begin
clr <= '0';
wait for 100 ns;
clr <= '1';
wait for 40 ns;
clr <= '0';
wait;
end process;

end Behavioral;
```

Lab 14 - tb


400.000 ns
Name
Value
clk
0
clr
0
q0
0
...
10000 ps
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
10000 ps

4-bit shift register



Clear	act. flank	Data in	Q_{FF1}	Q_{FF2}	Q_{FF3}	Data out
0	x	x	0	0	0	0
1	1^e	**1**	**1**	0	0	0
1	2^e	*0*	*0*	**1**	0	0
1	3^e	***0***	***0***	*0*	**1**	0
1	4^e	1	1	***0***	*0*	**1**
1	5^e	1	1	1	***0***	*0*
1	6^e	0	0	1	1	***0***
1	7^e	1	1	0	1	1

*Lab 15

Make a 4-bit shift register with ‘clr’

- Simulate
- Check RTL and Technology

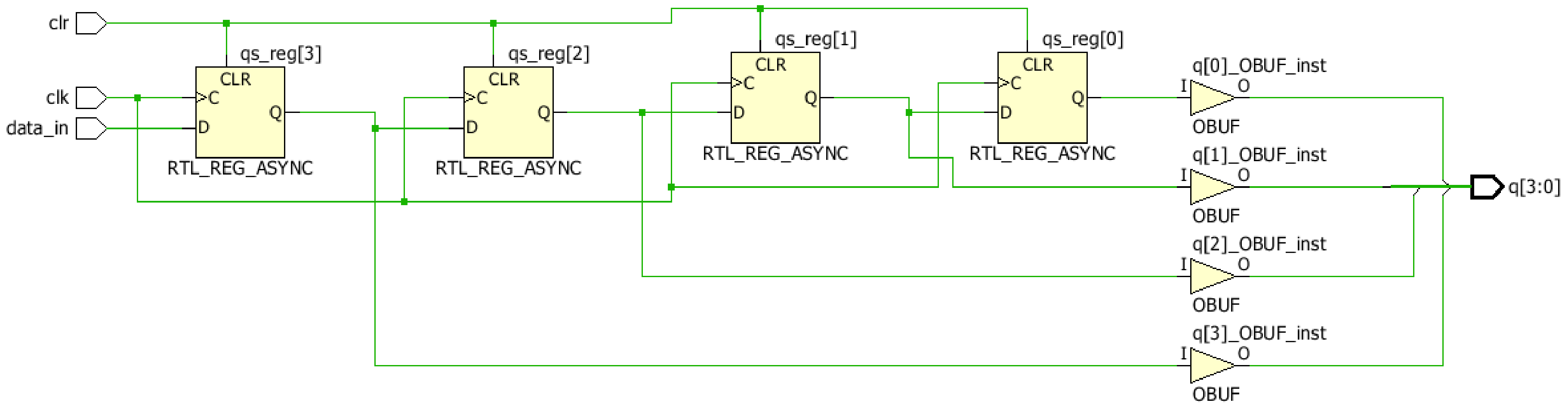
Lab 15 - solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ShiftReg is
            port(
                        clk : in STD_LOGIC;
                        clr : in STD_LOGIC;
                        data_in : in STD_LOGIC;
                        q : out STD_LOGIC_VECTOR(3 downto 0)
              );
end ShiftReg;
```

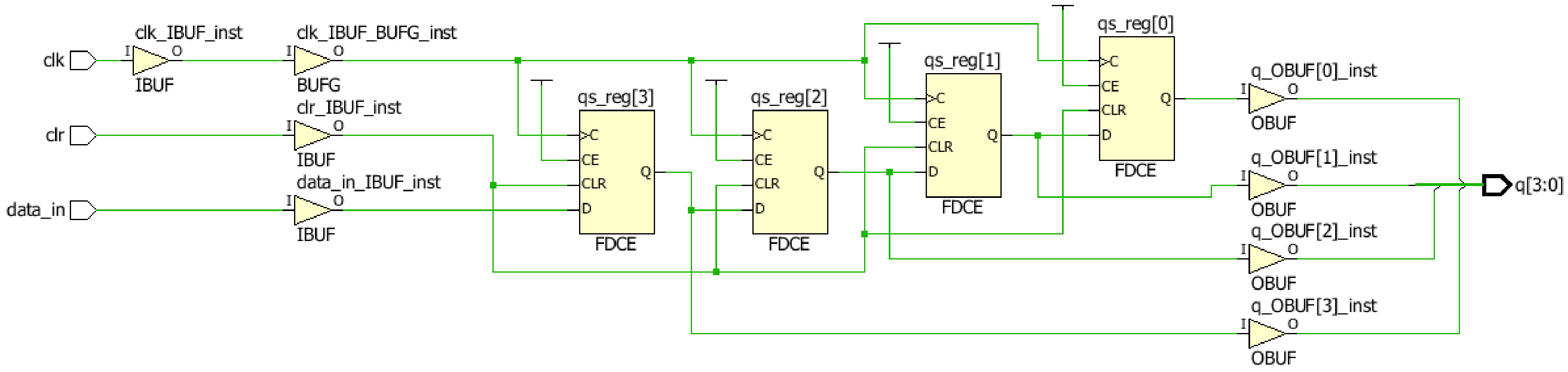
Lab 15 - solution

```
architecture behavior of ShiftReg is
signal qs: STD_LOGIC_VECTOR(3 downto 0);
begin
        -- 4-bit shift register
        process(clk, clr)
        begin
                if clr = '1' then
                        qs <= "0000";
                elsif clk'event and clk = '1' then
                        qs(3) <= data_in;
                        qs(2 downto 0) <= qs(3 downto 1);
                end if;

        end process;
        q <= qs;
end behavior;
```

Lab 15 – solution - RTL



Lab 15 – solution - Synth



Lab 15 – solution - tb

```
entity ShiftReg_tb is
--  Port ( );
end ShiftReg_tb;

architecture Behavioral of ShiftReg_tb is

component ShiftReg is
    Port ( clk : in STD_LOGIC;
           clr : in STD_LOGIC;
           data_in : in STD_LOGIC;
           q : out STD_LOGIC_VECTOR (3 downto 0));
end component ShiftReg;

signal clk, clr, data_in: STD_LOGIC;
signal q: STD_LOGIC_VECTOR (3 downto 0);
constant clk_period: time := 100 ns;
```

Lab 15 – solution - tb

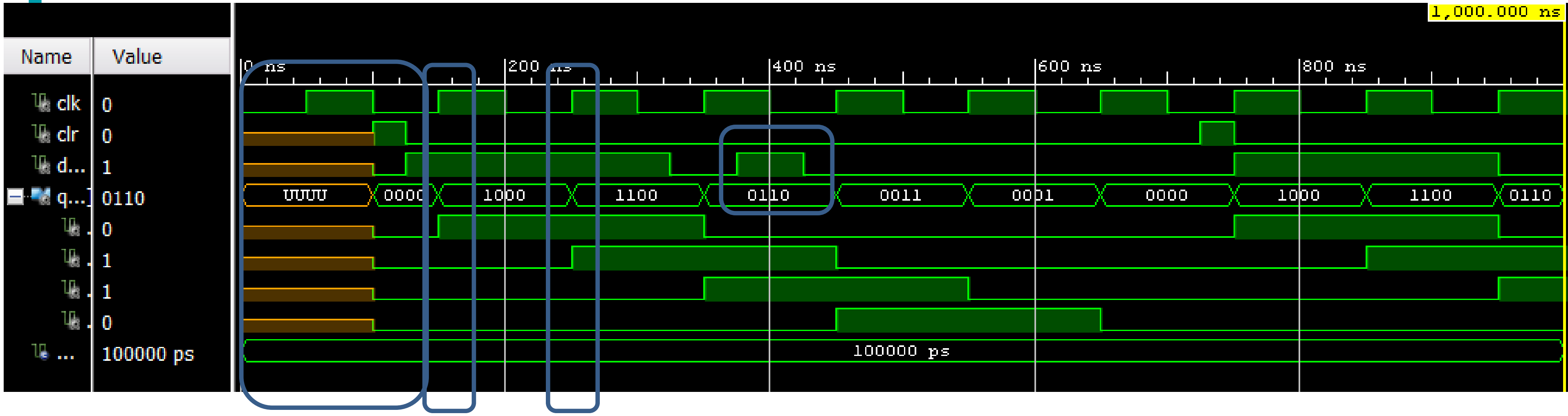
```
begin

UUT: ShiftReg
Port map( clk => clk, clr => clr, data_in => data_in, q => q);

clk_proc: process begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
end process;
```

```
stim_proc: process begin
wait for clk_period;
data_in <= '0';
clr <= '1';
wait for clk_period/4;
data_in <= '1';
clr <= '0';
wait for clk_period/2;
data_in <= '1';
wait for clk_period/2;
data_in <= '1';
wait for clk_period/2;
data_in <= '1';
wait for clk_period/2;
data_in <= '0';
```

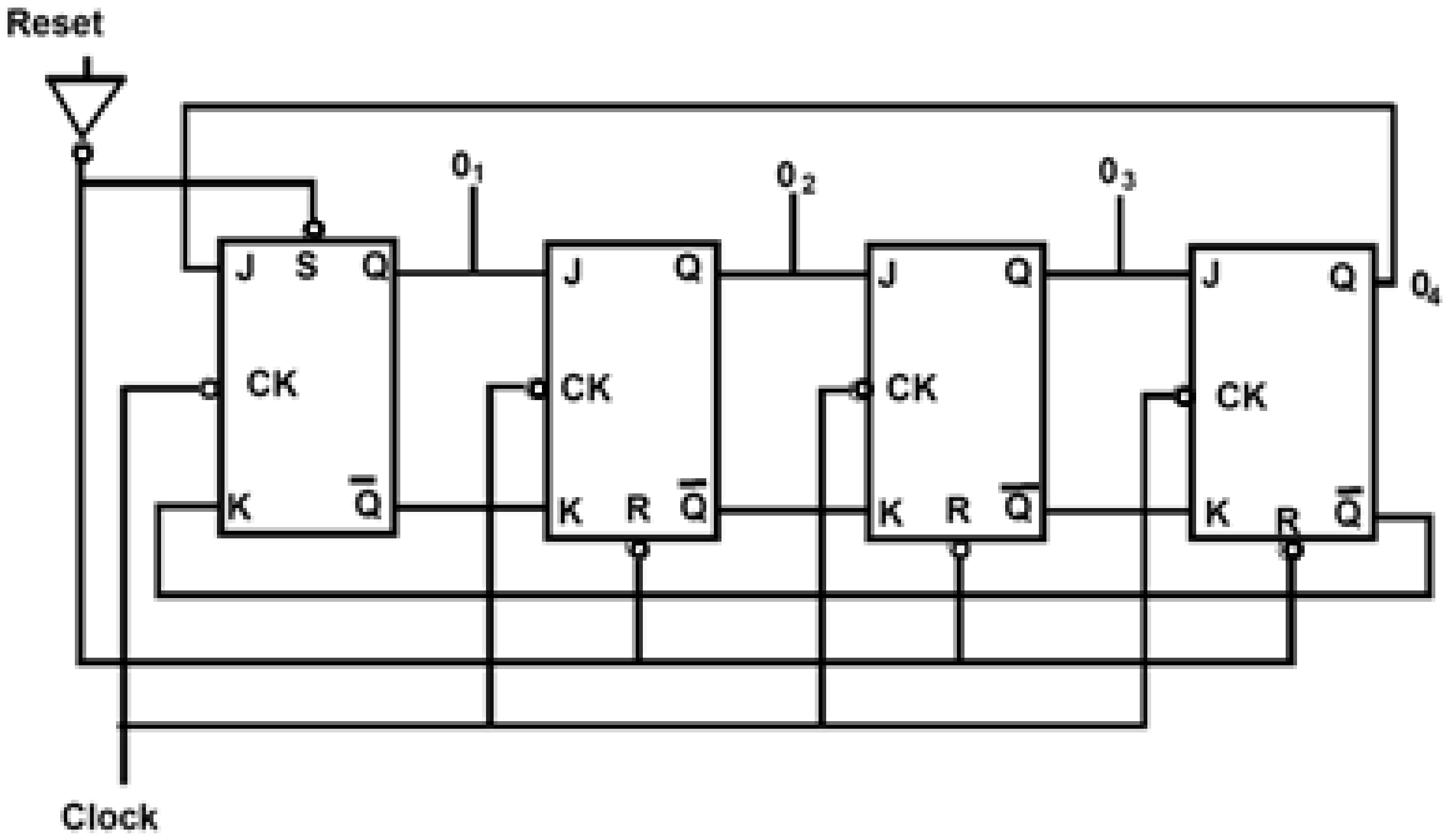
Lab 15 – solution - tb

```
wait for clk_period/2;
data_in <= '1';
wait for clk_period/2;
data_in <= '0';
wait for clk_period/2;
data_in <= '0';
wait for clk_period/2;
data_in <= '0';
wait for clk_period/2;
data_in <= '0';
wait for clk_period/2;
end process;
end Behavioral;
```

Lab 15 – solution - tb



Ring counter



Ring counter

Clock	O_1	O_2	O_3	O_4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0

Lab 16

Make a ring counter

- Simulate
- Check RTL and Technology

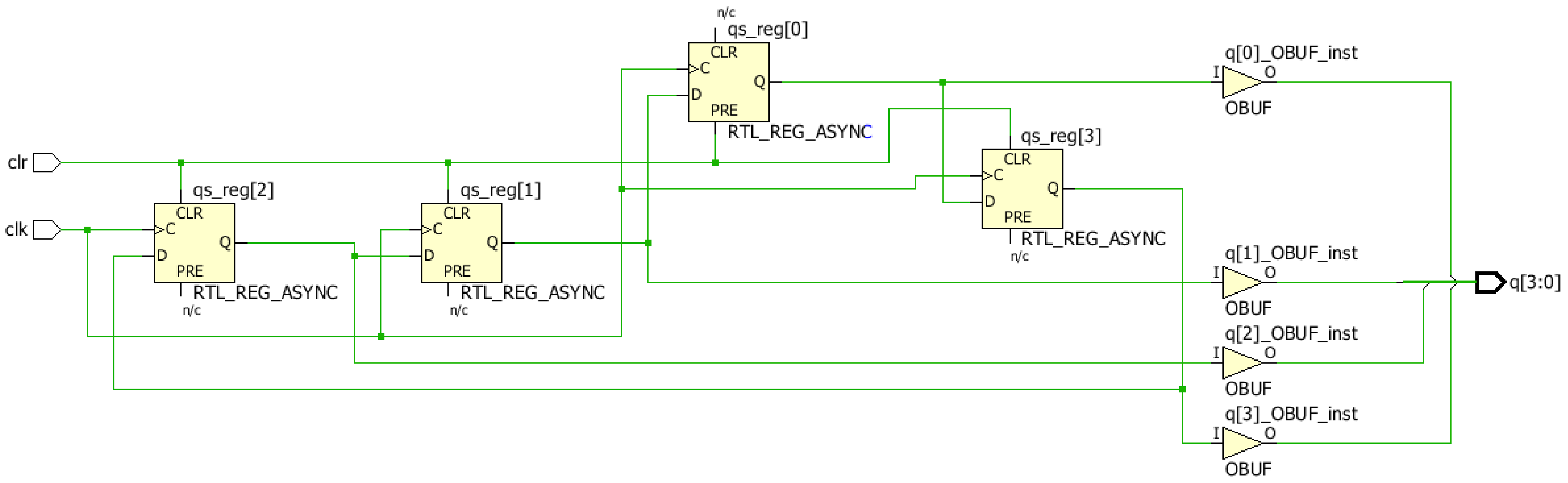
Lab 16 - solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ring4 is
		port(
				clk : in STD_LOGIC;
				clr : in STD_LOGIC;
				q : out STD_LOGIC_VECTOR(3 downto 0)
		  );
end ring4;
```

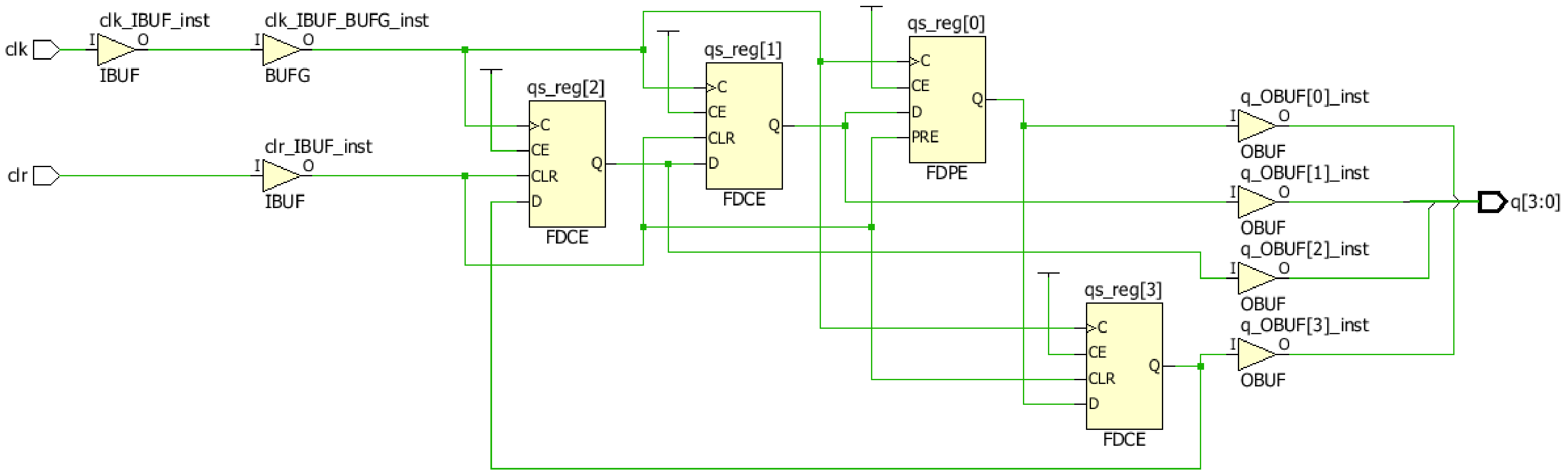
Lab 16 - solution

```
architecture behavior of ring4 is
signal qs: STD_LOGIC_VECTOR(3 downto 0);
begin
		-- 4-bit ring counter
		process(clk, clr)
		begin
				if clr = '1' then
						qs <= "0001";
				elsif clk'event and clk = '1' then
						qs(3) <= qs(0);
						qs(2 downto 0) <= qs(3 downto 1);
				end if;

		end process;
		q <= qs;
end behavior;
```

Lab 16 – solution - RTL



Lab 16 – solution - Synth



Lab 16 – solution - tb

```
entity ring4_tb is
--  Port ( );
end ring4_tb;

architecture Behavioral of ring4_tb is

component ring4 is
    Port ( clk : in STD_LOGIC;
           clr : in STD_LOGIC;
           q : out STD_LOGIC_VECTOR (3 downto 0));
end component ring4;
signal clk , clr :  STD_LOGIC;
signal q : STD_LOGIC_VECTOR (3 downto 0);
constant clk_period: time := 10 ns;
```

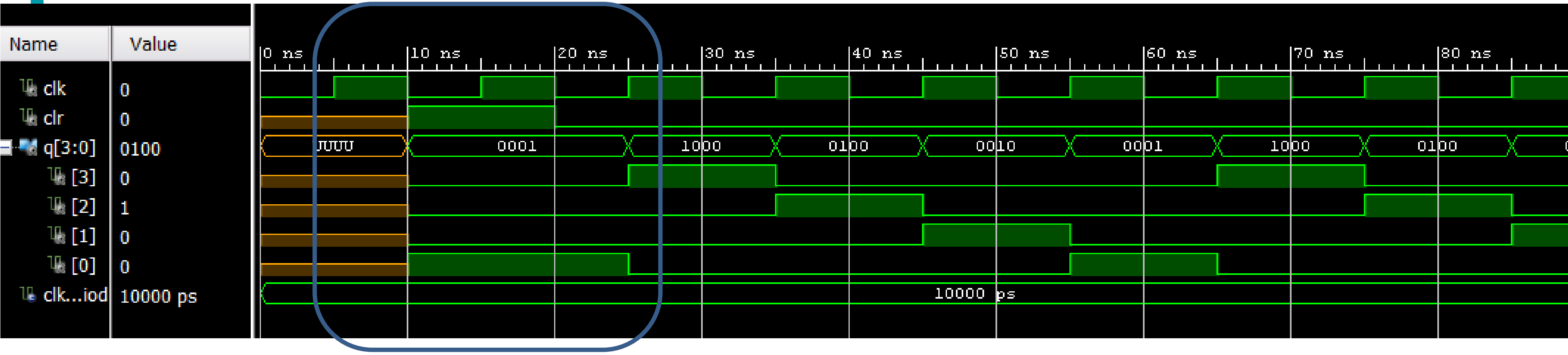
Lab 16 – solution - tb

```
begin
UUT: ring4
Port map( clk => clk, clr => clr,q => q);

clk_proc: process begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
end process;
```

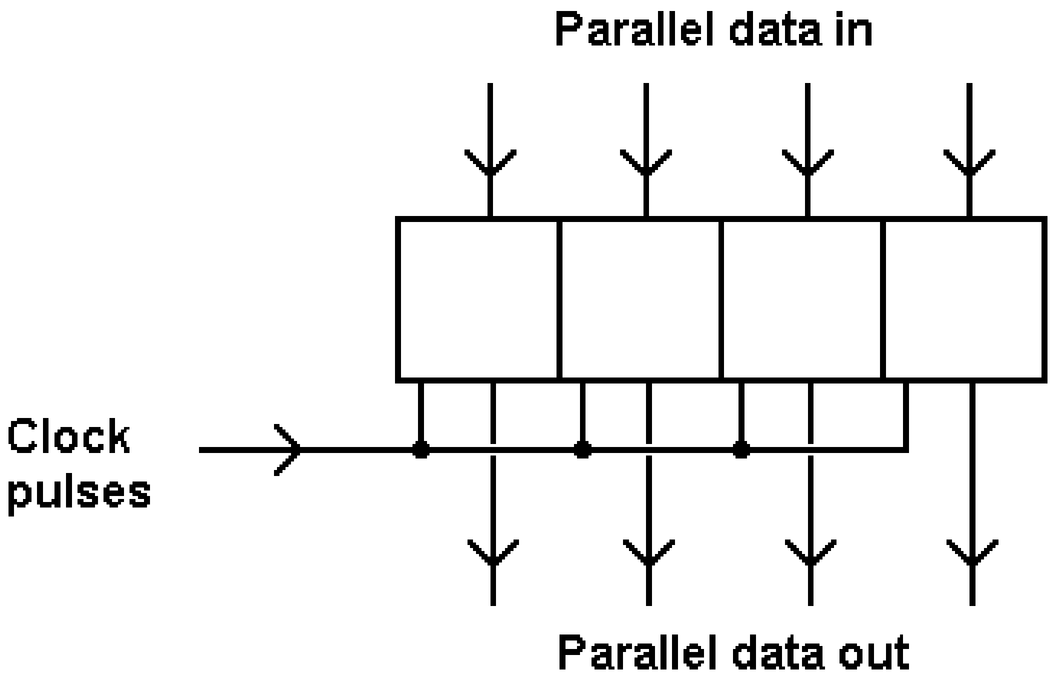
Lab 16 – solution - tb

```
stim_proc: process begin
wait for clk_period;
clr <= '1';
wait for clk_period;
clr <= '0';
wait;

end process;
end Behavioral;
```

Lab 16 – solution - tb



4-bit register



Lab 17

Make a 4-bit register with ‘clr’ and ‘load’

- Simulate
- Check RTL and Technology

Lab 17 - solution

```
entity reg4bit is
		port(
			load : in STD_LOGIC;
			inp0 : in STD_LOGIC_VECTOR(3 downto 0);
			clk : in STD_LOGIC;
			clr : in STD_LOGIC;
			q0 : out STD_LOGIC_VECTOR(3 downto 0)
		 );
end reg4bit;
```

Lab 17 - solution

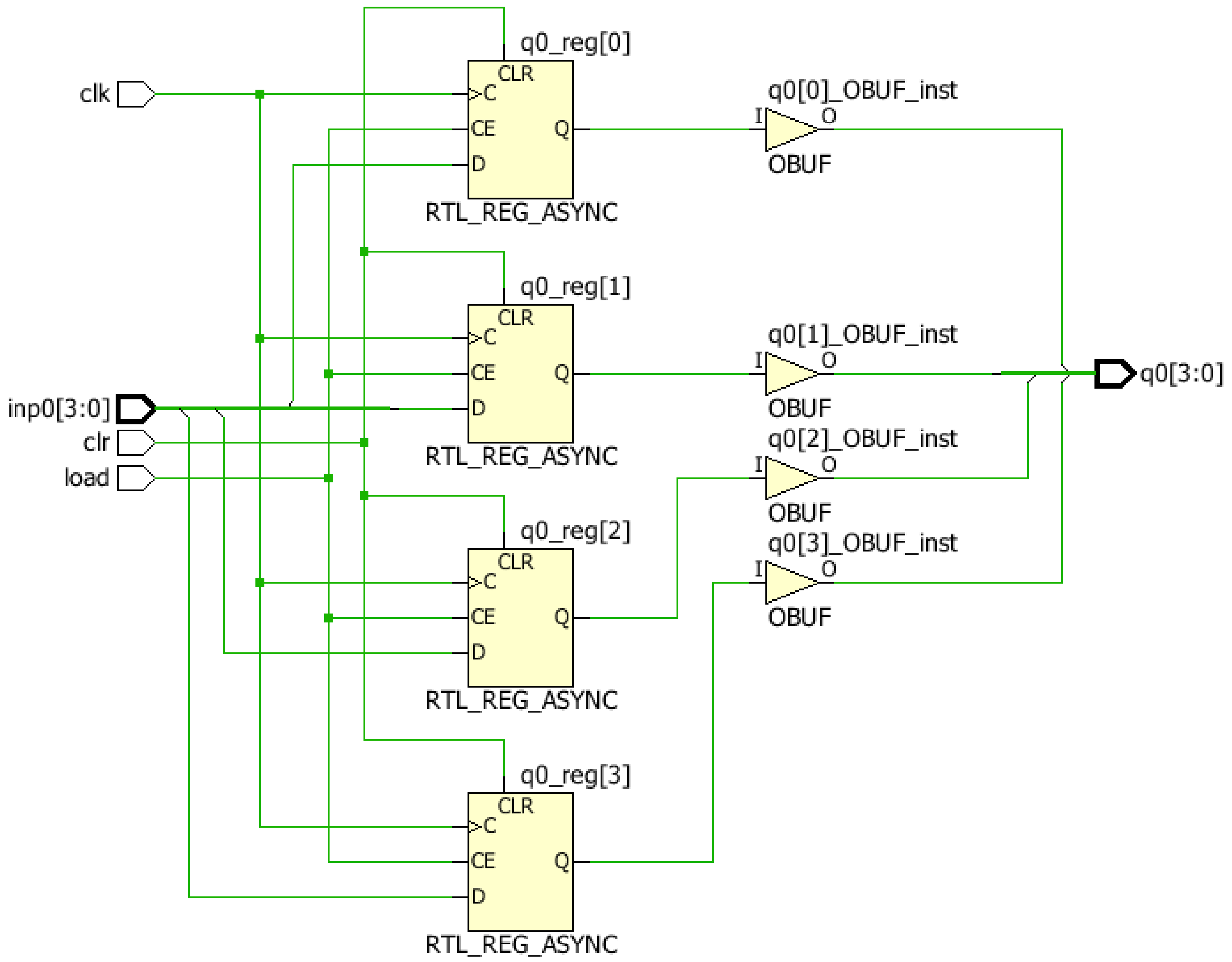
```
architecture behavior of reg4bit is
begin
        -- 4-bit register with load
        process(clk, clr)
        begin
                if clr = '1' then
                        q0 <= "0000";
                elsif clk'event and clk = '1' then
                        if load = '1' then
                                q0 <= inp0;
                        end if;
                end if;

        end process;
end behavior;
```

Lab 17: N-bit register

```
entity reg is
          generic(N:integer := 8);
          port(
                    load : in STD_LOGIC;
                    clk : in STD_LOGIC;
                    clr : in STD_LOGIC;
                    d : in STD_LOGIC_VECTOR(N-1 downto 0);
                    q : out STD_LOGIC_VECTOR(N-1 downto 0)
             );
end reg;
```

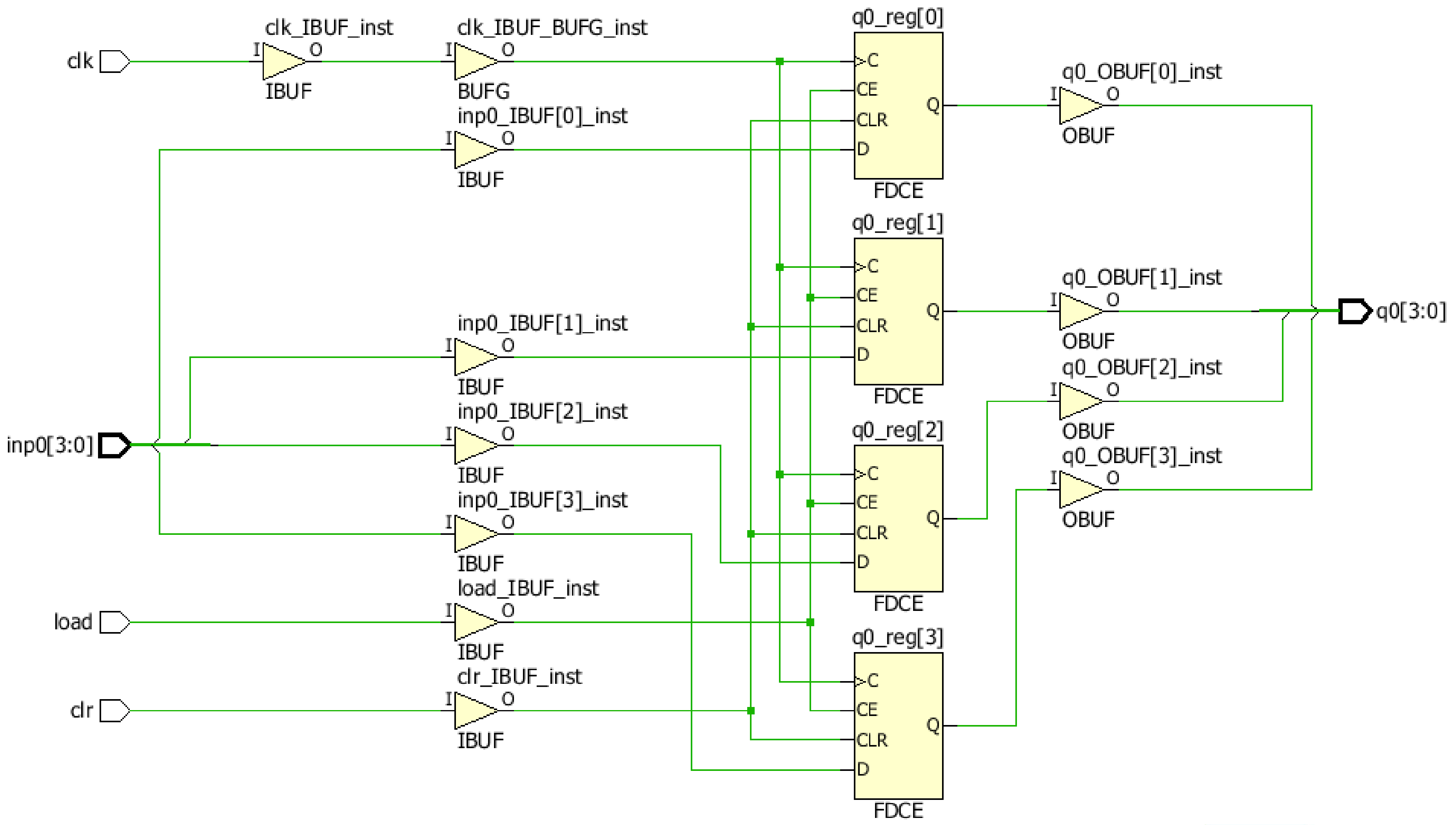
Generic gets definitive value in component declaration, when used as component

Lab 17 – solution - RTL



Tempus

Lab 17 – solution - Synth



Lab 17 – solution - tb

```
entity reg4bit_tb is
end reg4bit_tb;

architecture Behavioral of reg4bit_tb is

component reg4bit is
   Port ( load : in STD_LOGIC;
          inp0 : in STD_LOGIC_VECTOR (3 downto 0);
          clk : in STD_LOGIC;
          clr : in STD_LOGIC;
          q0 : out STD_LOGIC_VECTOR (3 downto 0));
end component reg4bit;

signal load :  STD_LOGIC;
signal inp0 :  STD_LOGIC_VECTOR (3 downto 0);
signal clk :  STD_LOGIC;
signal clr :  STD_LOGIC;
signal q0 :  STD_LOGIC_VECTOR (3 downto 0);

constant clk_period: time := 10 ns;
```

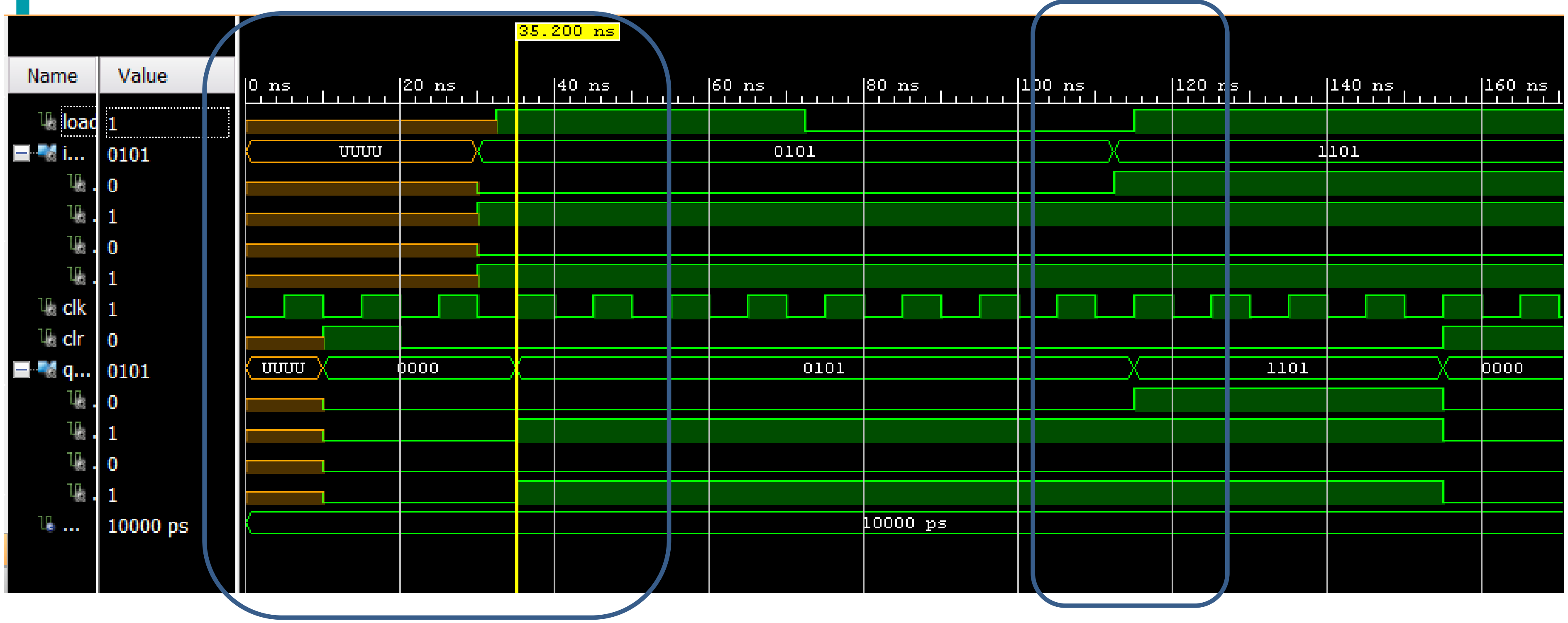
Lab 17 – solution - tb

```
begin

UUT: reg4bit
 Port map( load => load, inp0 => inp0, clk => clk,clr => clr, q0 => q0);

clk_proc: process begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
end process;
```

Lab 17 – solution - tb

```
stim_proc: process begin
wait for clk_period;
clr <= '1';
wait for clk_period;
clr <= '0';
wait for clk_period;
inp0 <= "0101";
wait for clk_period/4;
load <= '1';
wait for clk_period*4;
load <= '0';
wait for clk_period*4;
inp0 <= "1101";
wait for clk_period/4;
load <= '1';
wait for clk_period*4;
clr <= '1';
wait;
end process;
end Behavioral;
```

Lab 17 – solution - tb

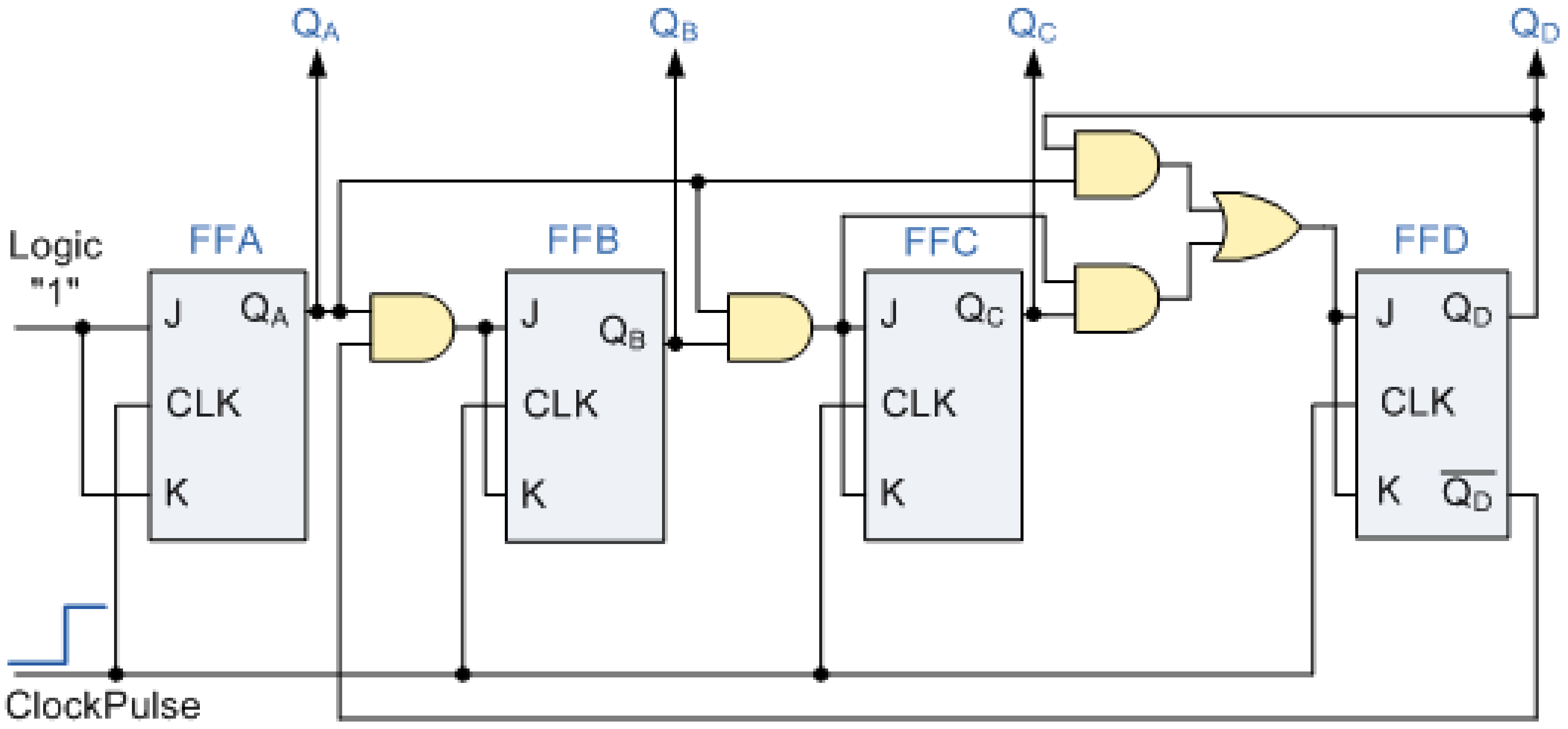


Synchronous Counter

- Counts number of clock pulses
- The value appears on the outputs
- All outputs change on the same clock edge

Synchronous Counter

Synchronous BCD counter with T-FF's: (J&K connected)



*Lab 18

Make a 10-counter with ‘en’, ‘rst’ & ‘co’ (carry out)

- Simulate
- Check RTL and Technology

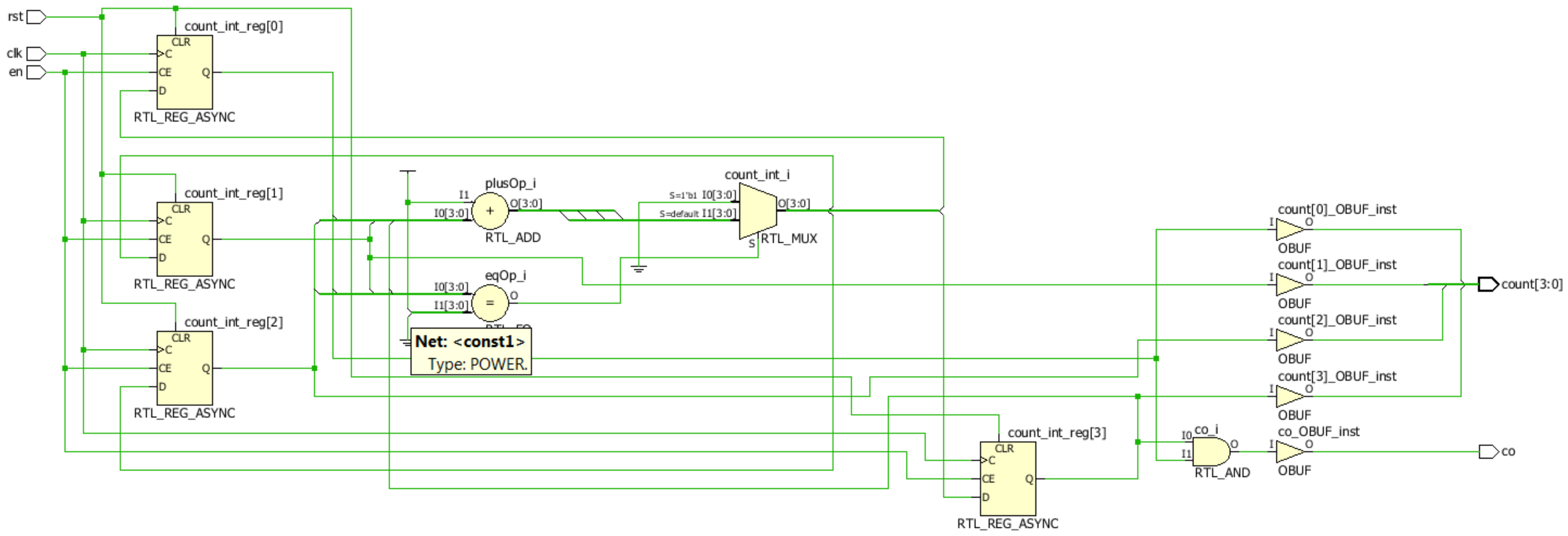
Lab 18 - solution

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity count10 is
Port       ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           en : in STD_LOGIC;
           co : out STD_LOGIC;
           count : out STD_LOGIC_VECTOR (3 downto 0));
end count10;
```

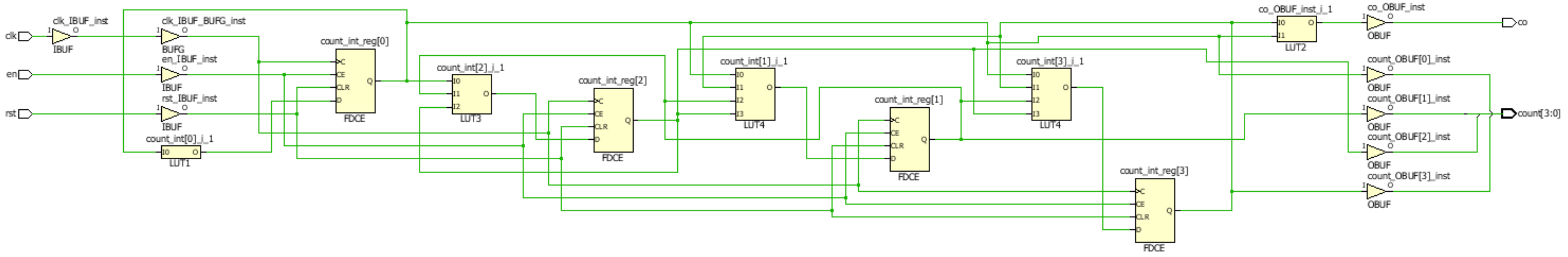
Lab 18 - solution

```
architecture Behavior of count10 is
        signal count_int: STD_LOGIC_VECTOR (3 downto 0);
begin
        process (clk, rst) begin
                if (rst = '1') then
                        count_int <= "0000";
                elsif (rising_edge (clk) and en = '1') then
                        count_int <= count_int + 1;
                        if count_int = "1001" then
                                count_int <= "0000";
                        end if;
                end if;
        end process;
count <= count_int;
co <= count_int(3) and count_int(0);
end Behavior;
```

Lab 18 – solution - RTL



Lab 18 – solution - Synth



Lab 18 – solution - tb

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity count10_tb is
end count10_tb;

architecture Behavioral of count10_tb is
component count10 is
   Port ( clk : in STD_LOGIC;
          rst : in STD_LOGIC;
          en : in STD_LOGIC;
          co : out STD_LOGIC;
          count : out STD_LOGIC_VECTOR (3 downto 0));
end component count10;
```

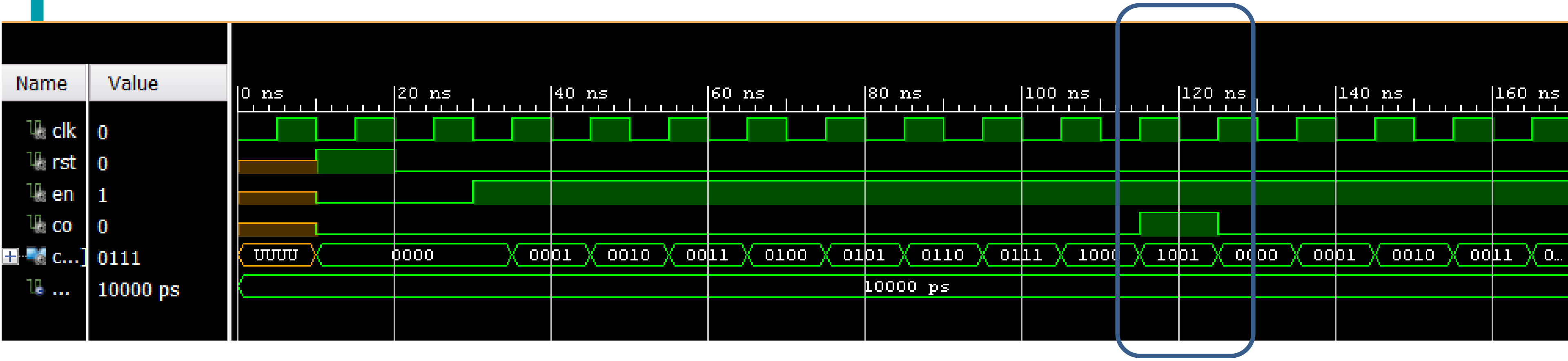
Lab 18 – solution - tb

```
signal clk :  STD_LOGIC;
signal rst :  STD_LOGIC;
signal en :  STD_LOGIC;
signal co :  STD_LOGIC;
signal count :  STD_LOGIC_VECTOR (3 downto 0);

constant clk_period: time := 10 ns;

begin

UUT: count10
    Port map(   clk => clk, rst => rst , en => en,  co => co, count => count);

clk_proc: process begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
end process;
```

Lab 18 – solution - tb

```
stim_proc: process begin
wait for clk_period;
rst <= '1';
en <= '0';
wait for clk_period;
rst <= '0';
wait for clk_period;
en <= '1';
wait;
end process;

end Behavioral;
```

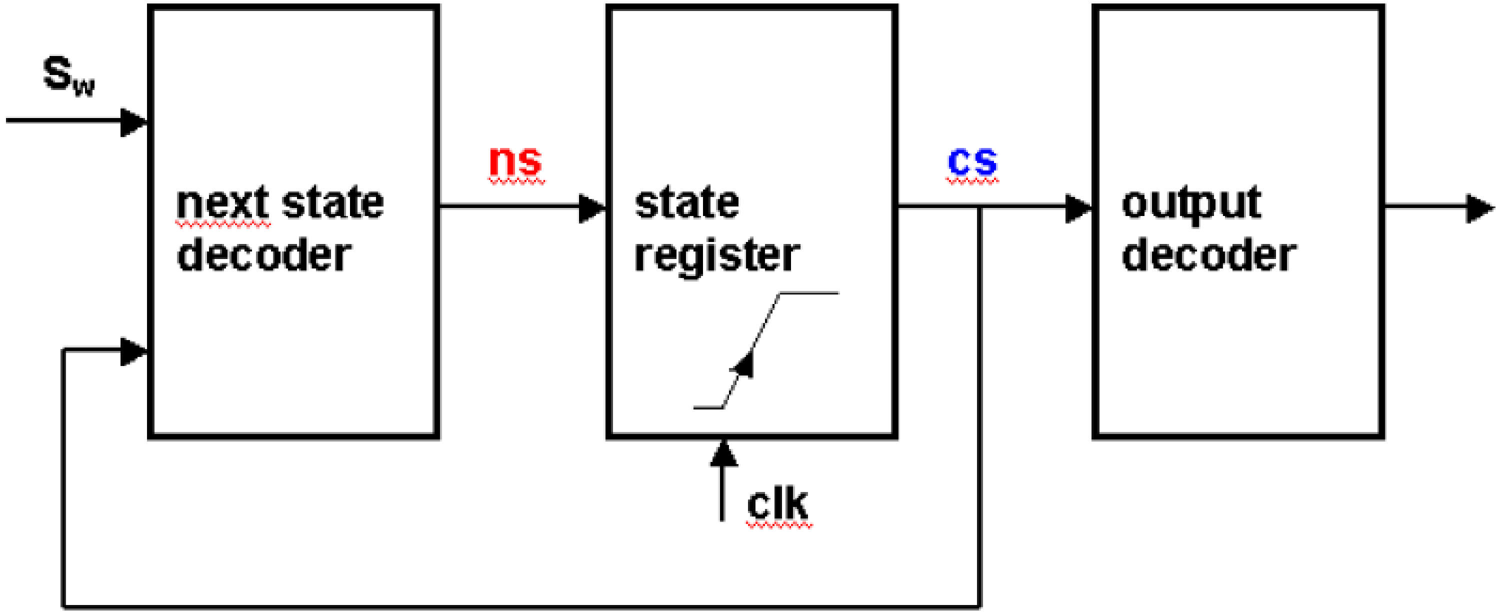
Lab 18 – solution - tb



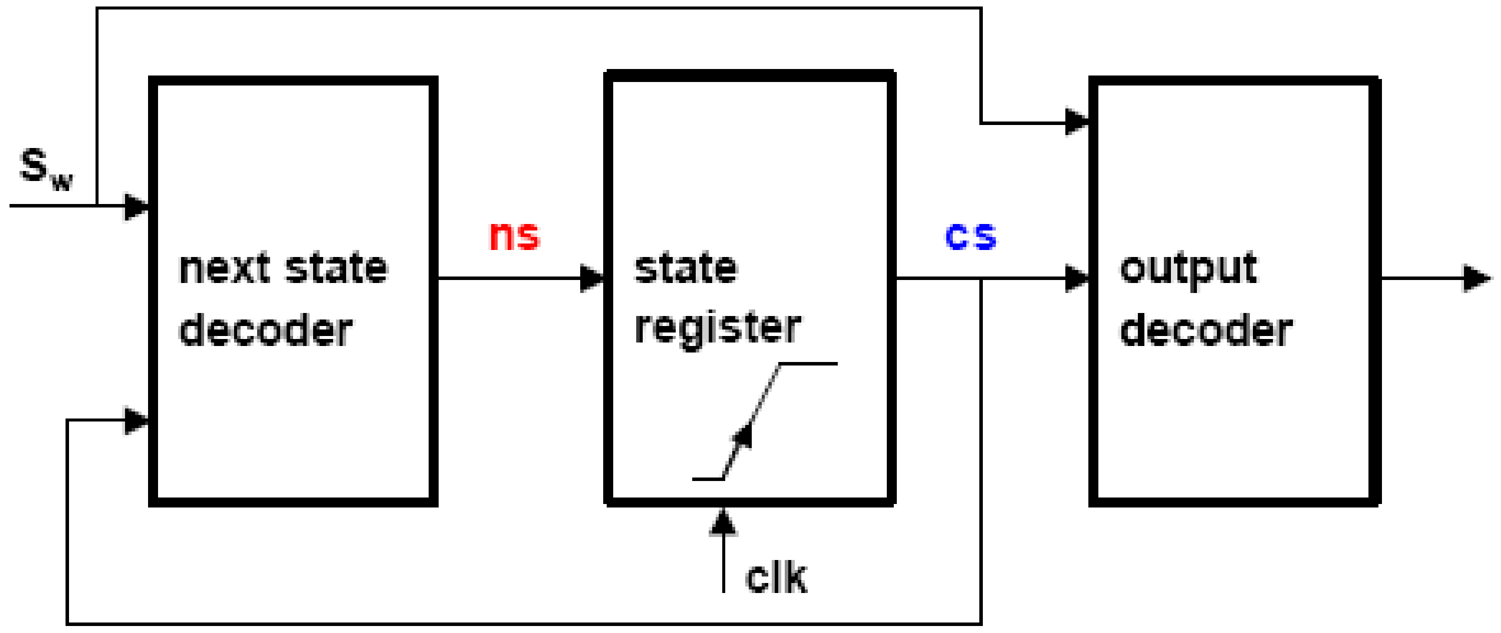
Finite State Machines

- Runs through a finite amount of states
- E.g. BCD-counter – Traffic light
- Next state is determined by the present state, the clock and possibly the value on the inputs

FSM: block diagram Moore

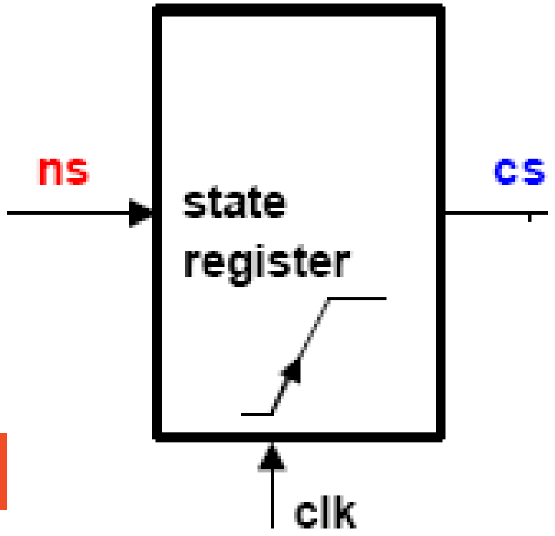


FSM: block diagram Mealy



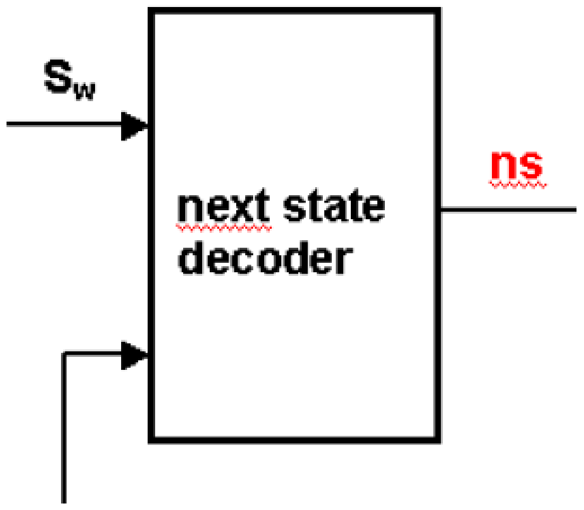
FSM: block diagram

- State register
  - A certain amount of flip-flops (D or JK), triggered by the clock
  - Amount FFs is determined by the amount of states



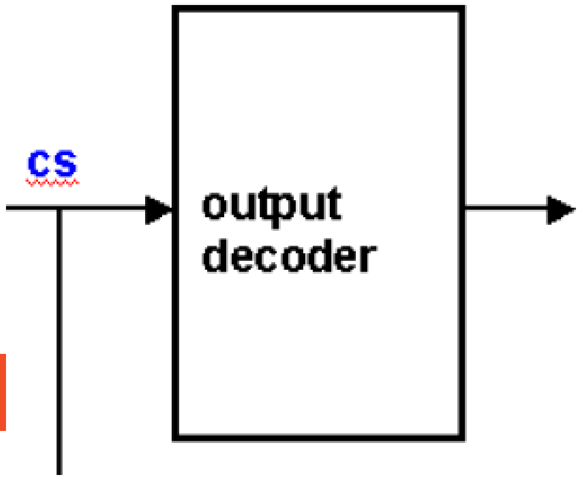
FSM: block diagram

- Next state decoder
  - Next state is determined by the current state (fed back) and possibly an input



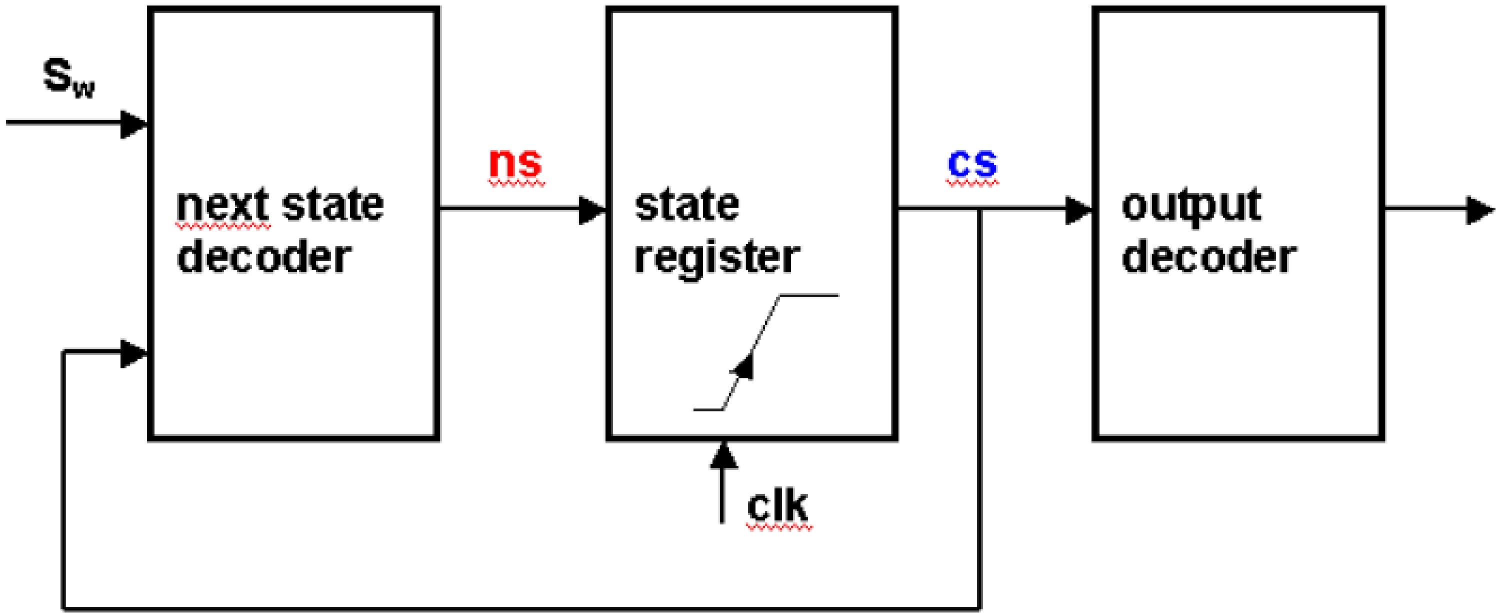
FSM: block diagram

- Output decoder
  - o Output is determined by current state and possibly the inputs (Mealy)
  - o By involving inputs in the output decoder, the system is not synchronous anymore (Mealy)

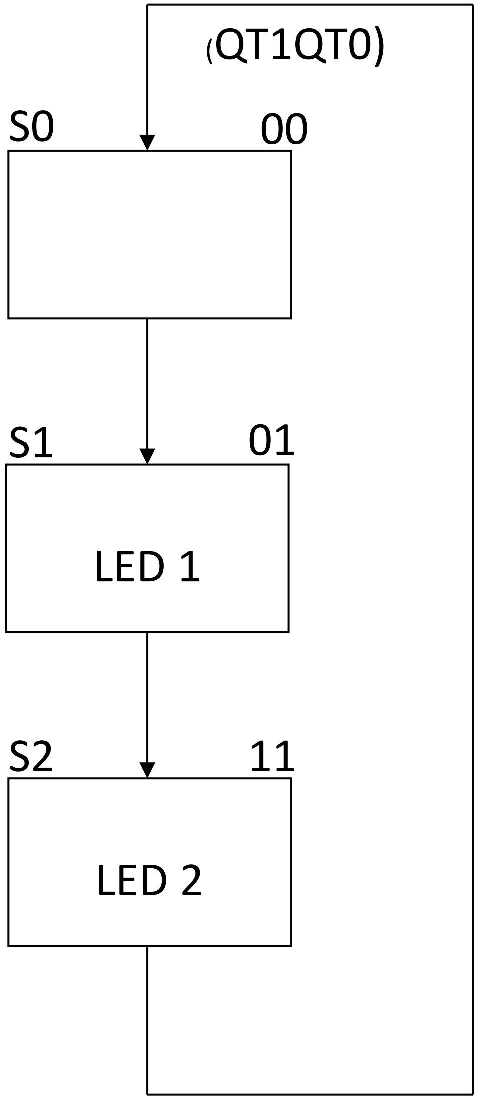


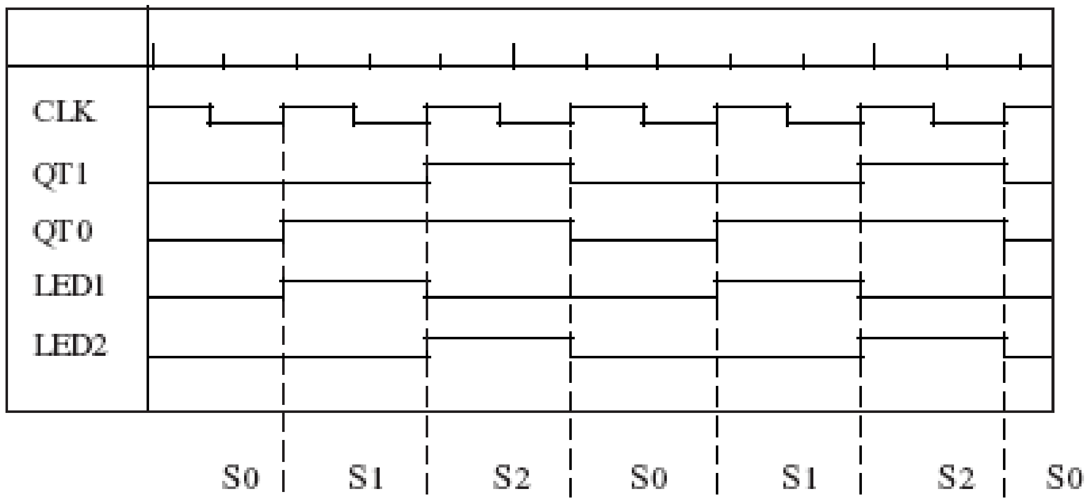
FSM: block diagram

- Next state and output decoder are pure combinatorial components
- Synchronisation is in state register

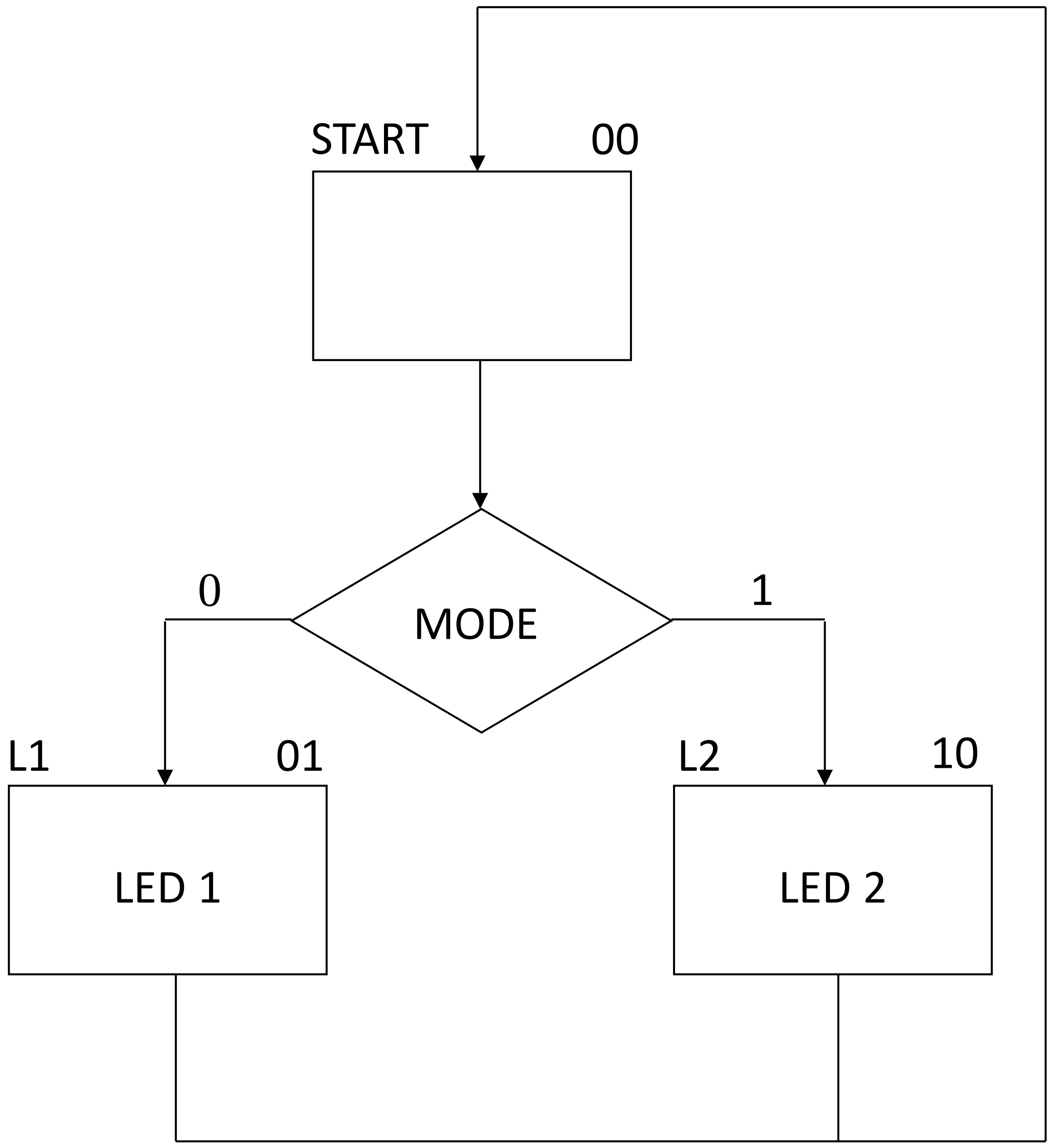


FSM: state diagram



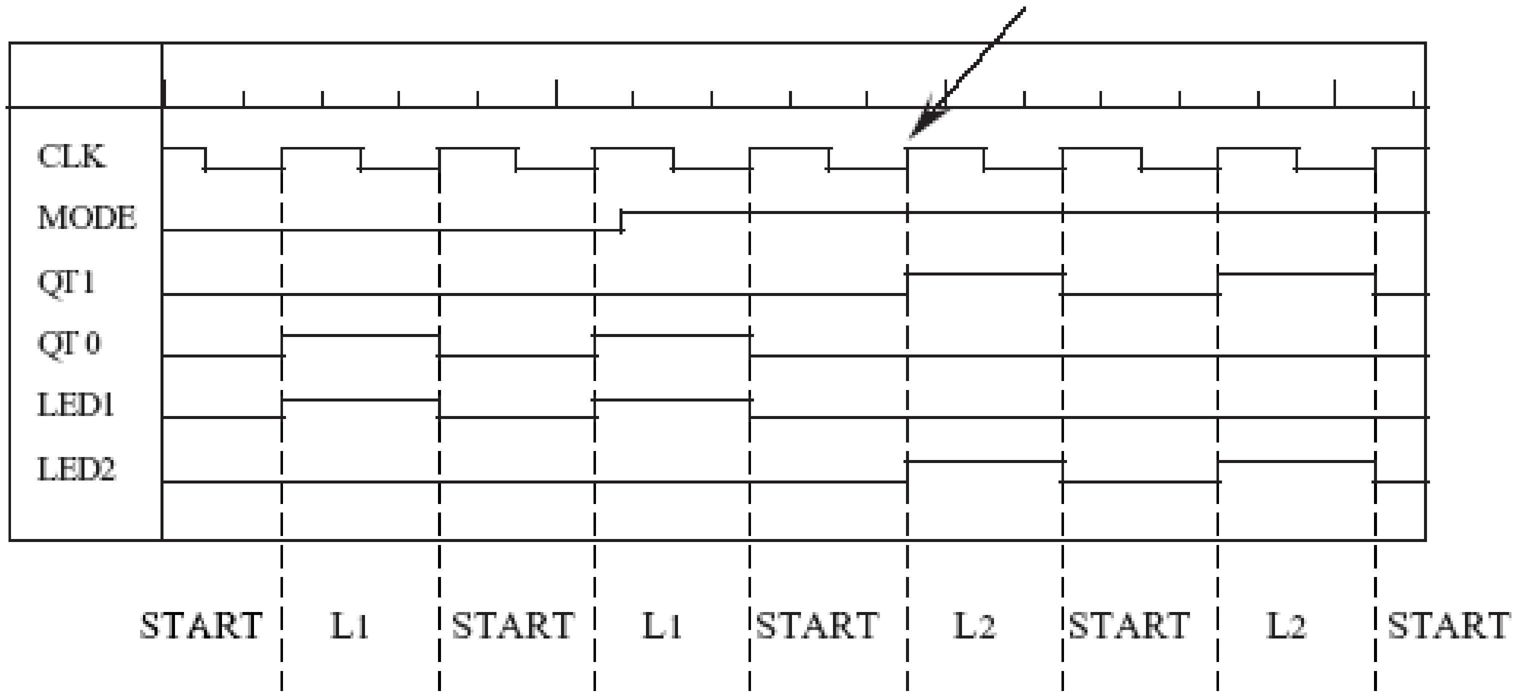


FSM: state diagram

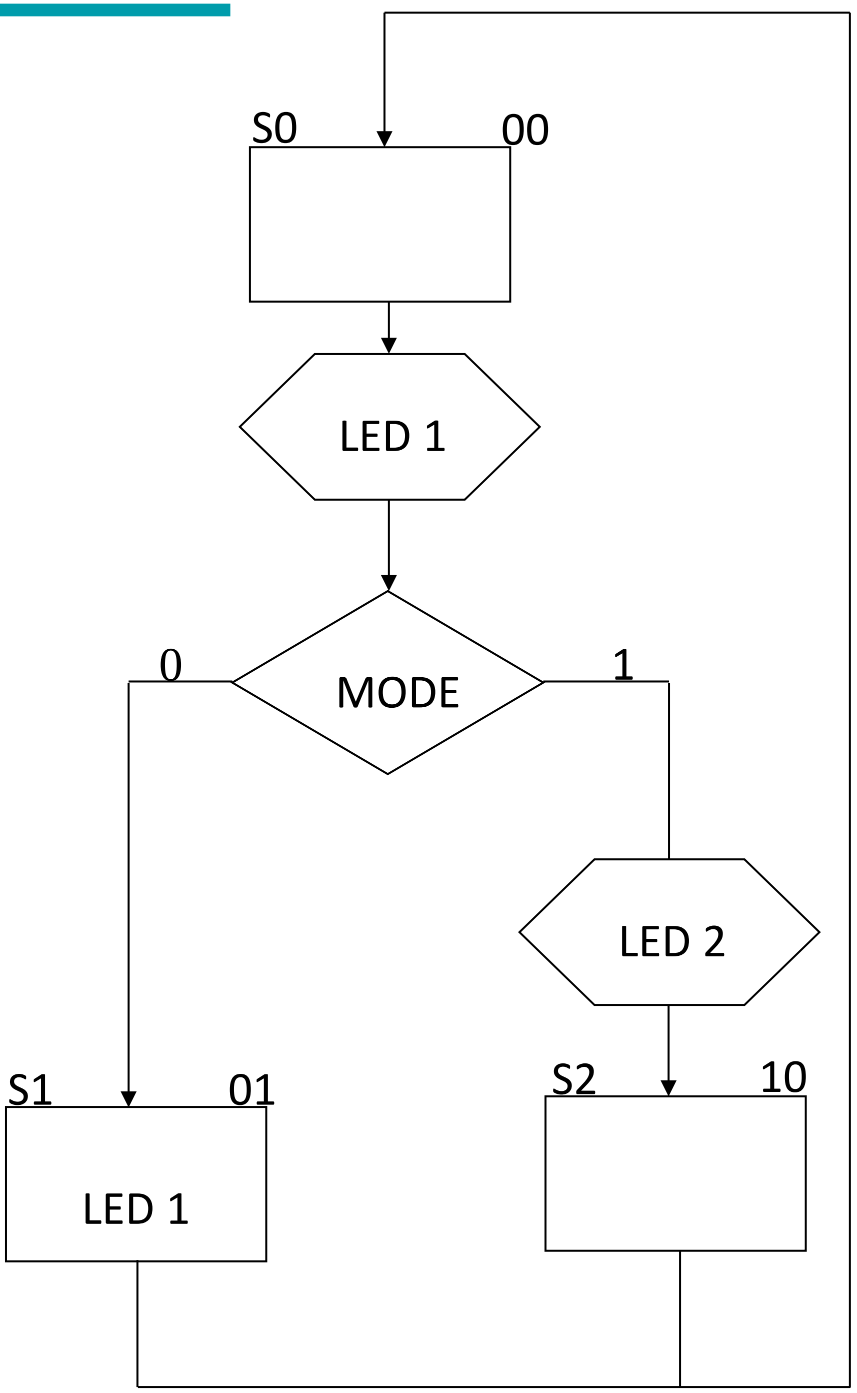


FSM: state diagram

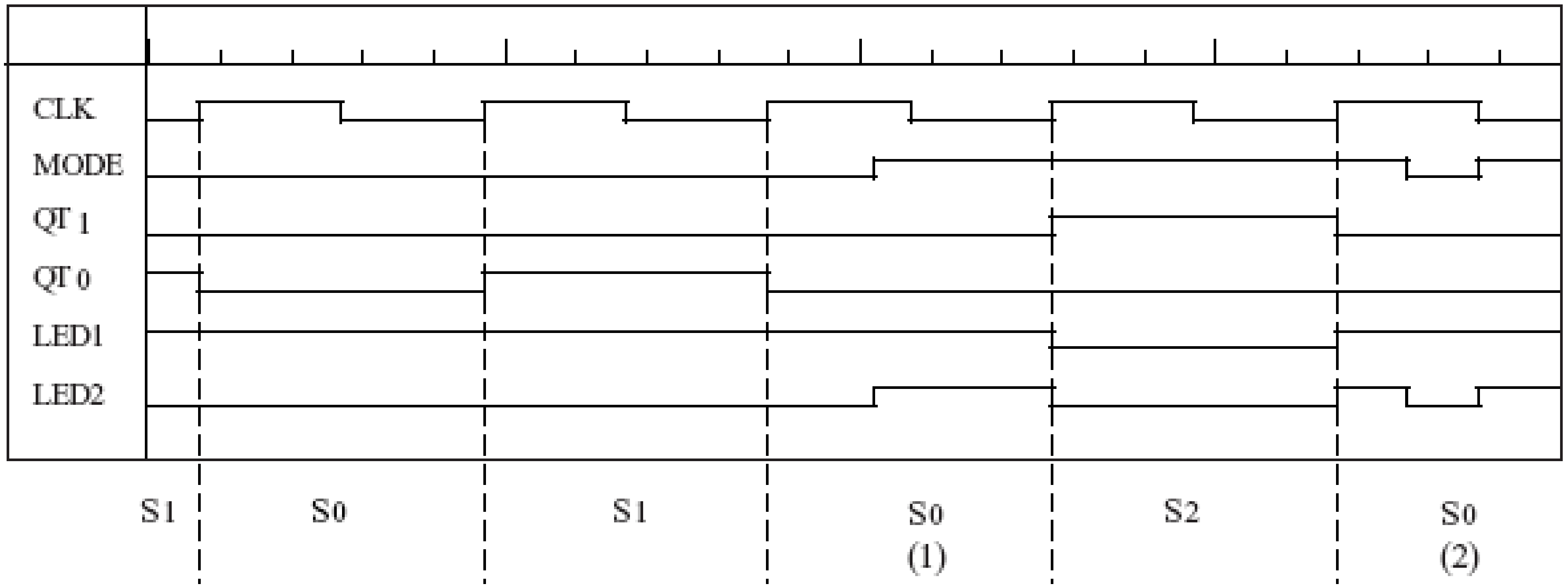
De verandering van "MODE" wordt nu pas verwerkt!



FSM:

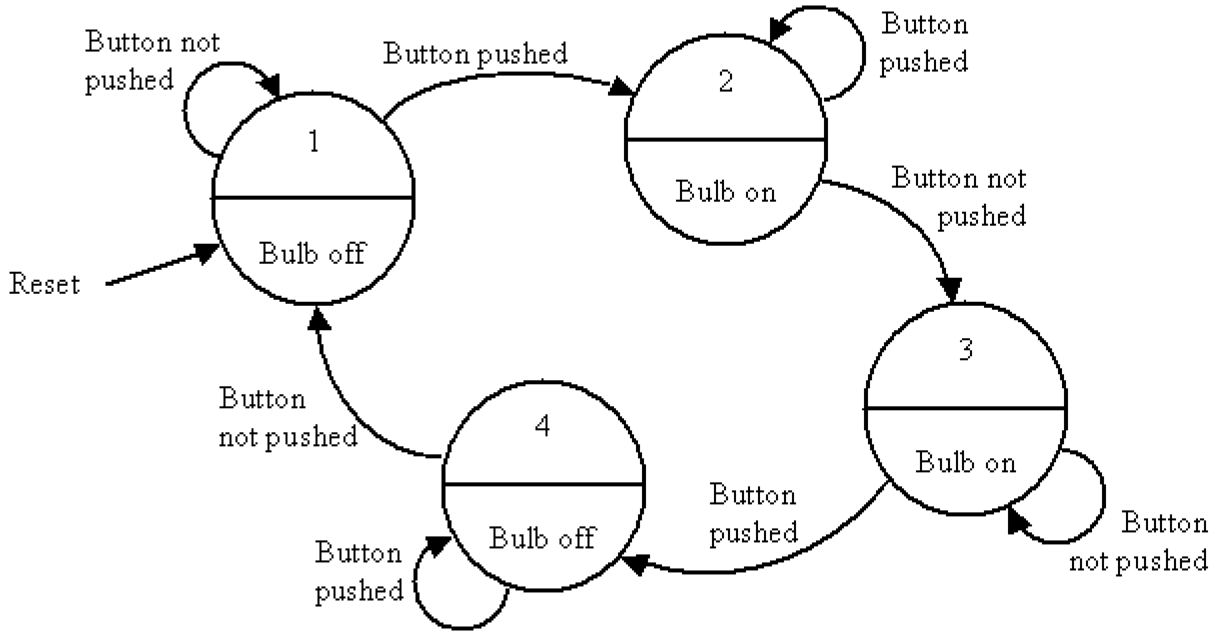


FSM: state diagram



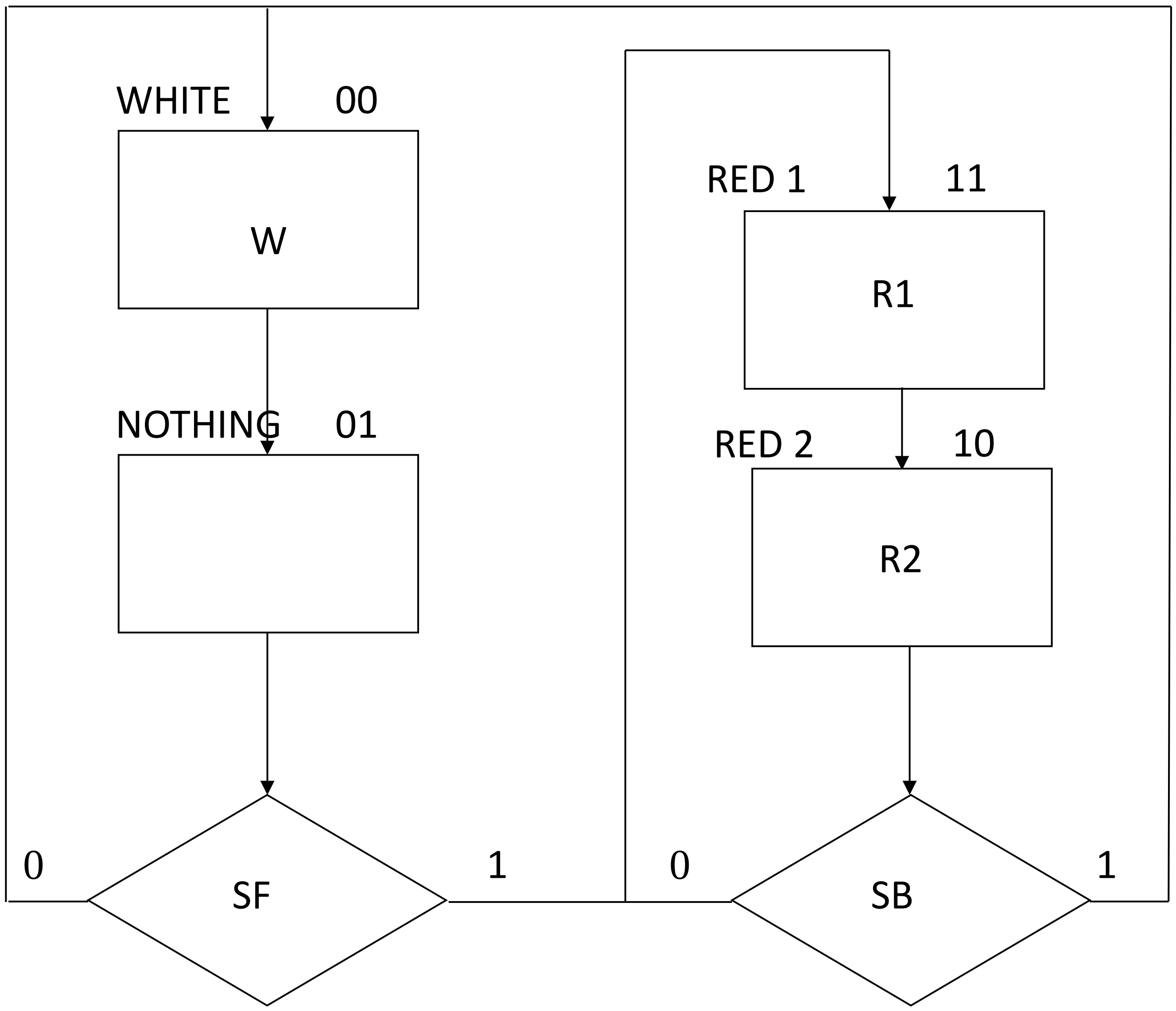
FSM: state diagram

- Other representation:
  - o Circles with name and state
  - o Coding of the state
  - o Arrows give the change of state



FSM: example

- Railway signalization
  - Two sensors, one before and one behind the railway crossing.
  - When the train arrives, the first sensor (SF) gives an impulse to activate the red light.
  - This state remains until the train passes the next sensor (SB) and activates the white light.


WHITE
00
W
NOTHING
01
RED 1
11
R1
RED 2
10
R2
0
SF
1
0
SB
1

*Lab 19

Make a railway crossing system

- Simulate
- Check RTL & Technology

Lab 19 - solution

```
entity Railway is
   Port ( clk : in STD_LOGIC;
          clr : in STD_LOGIC;
          SF : in STD_LOGIC;
          SB : in STD_LOGIC;
          W : out STD_LOGIC;
          R1 : out STD_LOGIC;
          R2 : out STD_LOGIC);
end Railway;
```

Lab 19 – state declaration

```
architecture Behavioral of Railway is

type state_type is ( Nothing,  White, Red1, Red2);
signal present_state, next_state: state_type;
```

Type definition:
type <name_type> is (<all possible states>);

present_state and next_state can have the values:
Nothing, White, Red1, Red2
Possible statements:
- present_state <= Nothing;
- present_stare <= next_state;

Lab 19 – state register

```
begin

sreg: process(clk, clr)
begin
    if clr = '1' then
        present_state <= Nothing;
    elsif clk'event and clk = '1' then
        present_state <= next_state;
    end if;
end process;
```

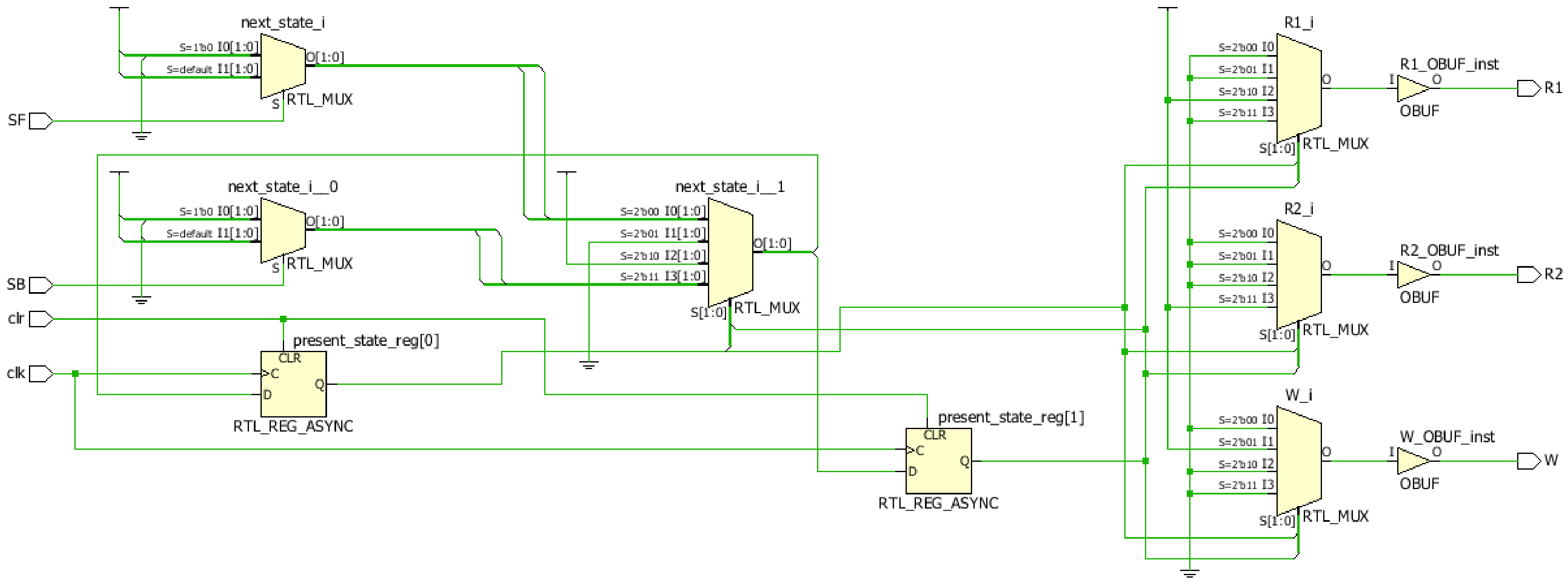
Lab 19 – next state decoding

```
ns_dec: process (present_state, SF, SB)
begin
case present_state is
   when Nothing =>             if SF = '0' then
                                        next_state <= White;
                               else
                                        next_state <= Red1;
                               end if;
   when White =>               next_state <= Nothing;
   when Red1 =>                next_state <= Red2;
   when Red2 =>                if SB = '0' then
                                        next_state <= Red1;
                               else
                                        next_state <= White;
                                end if;
end case;
end process;
```

Lab 19 – output decoding

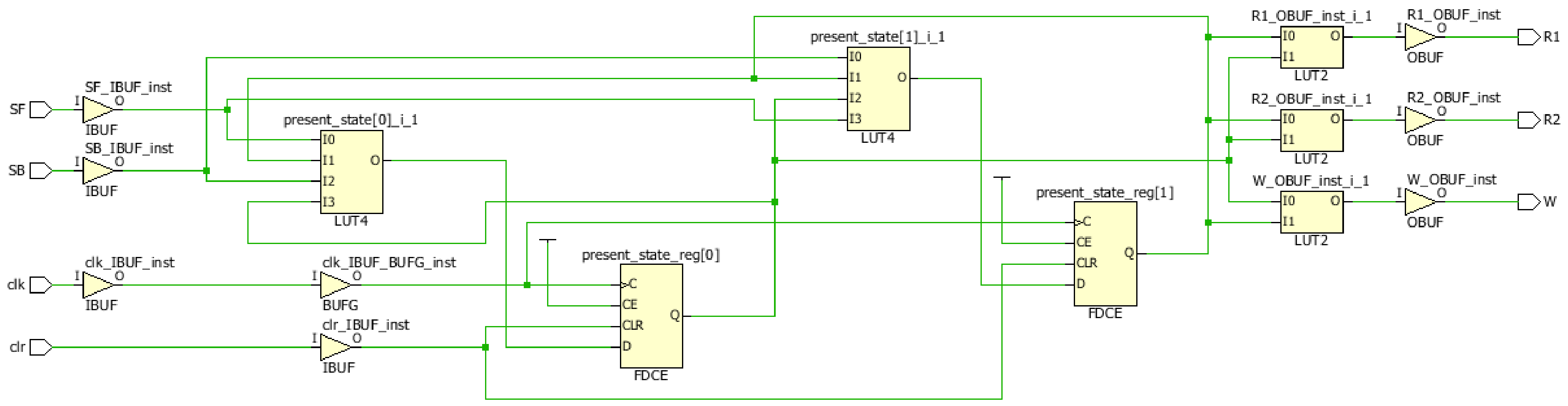
```
out_dec: process (present_state)
begin
case present_state is
   when Nothing =>              W <= '0';
                                R1 <= '0';
                                R2 <= '0';
   when White =>                W <= '1';
                                R1 <= '0';
                                R2 <= '0';
   when Red1 =>                 W <= '0';
                                R1 <= '1';
                                R2 <= '0';
   when Red2 =>                 W <= '0';
                                R1 <= '0';
                                R2 <= '1';
end case;
end process;
end Behavioral;
```

Lab 19 – solution - RTL



Tempus

Lab 19 – solution - Synth



Lab 19 – solution - tb

```
entity Railway_tb is
--  Port ( );
end Railway_tb;

architecture Behavioral of Railway_tb is
component Railway is
    Port ( clk : in STD_LOGIC;
           clr : in STD_LOGIC;
           SF : in STD_LOGIC;
           SB : in STD_LOGIC;
           W : out STD_LOGIC;
           R1 : out STD_LOGIC;
           R2 : out STD_LOGIC);
end component Railway;

signal clk,clr, SF, SB, W, R1,  R2: STD_LOGIC;

constant clk_period: time := 10 ns;
```

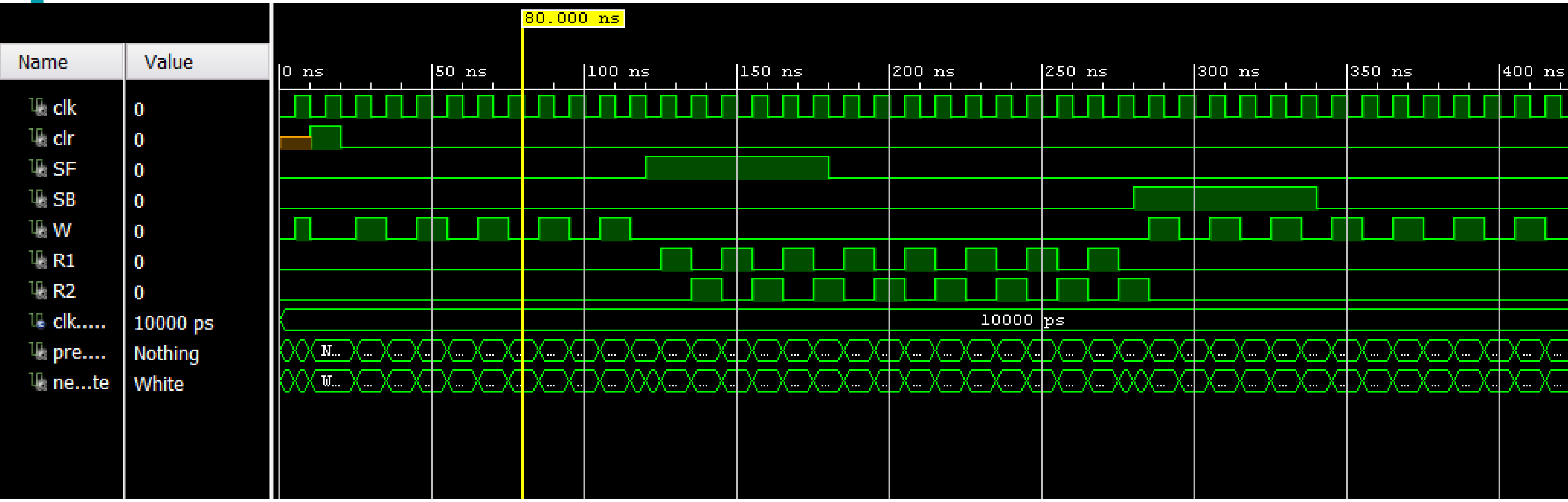
Lab 19 – solution - tb

```
begin
UUT: Railway
    Port map ( clk => clk, clr => clr, SF => SF, SB => SB, W => W, R1 => R1, R2 =>
R2);

clk_proc: process begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
end process;
```

Lab 19 – solution - tb

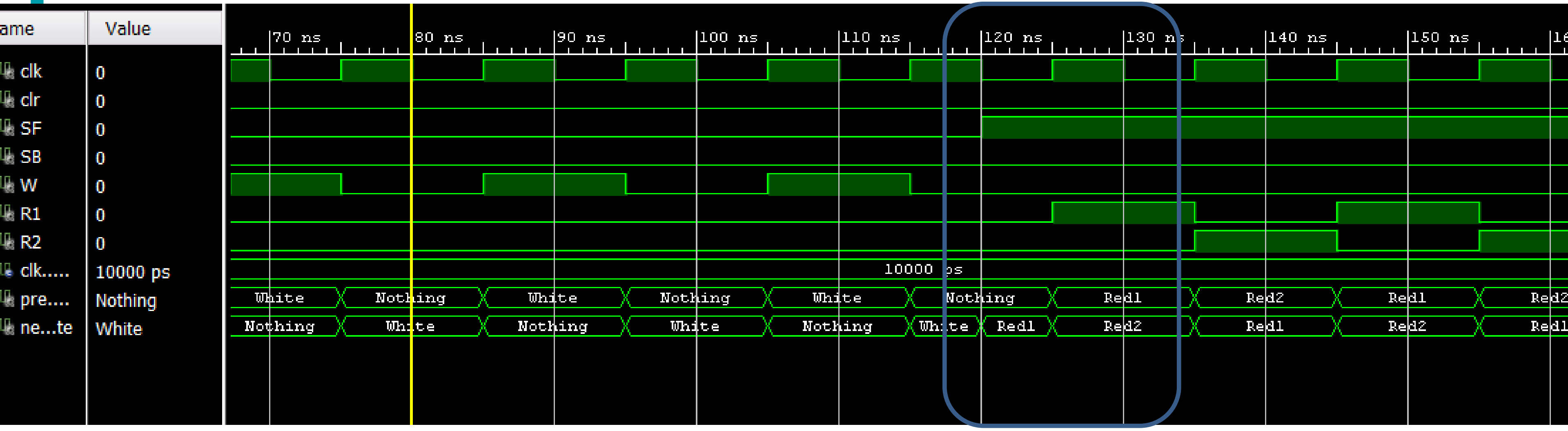
```
stim_proc: process begin
SB <= '0';
SF <= '0';
wait for clk_period;
clr <= '1';
wait for clk_period;
clr <= '0';
wait for clk_period*10;
SF <= '1';
wait for clk_period*6;
SF <= '0';
wait for clk_period*10;
SB <= '1';
wait for clk_period*6;
SB <= '0';
wait;
end process;
```

Lab 19 – solution - tb



Name	Value
clk	0
clr	0
SF	0
SB	0
W	0
R1	0
R2	0
clk.....	10000 ps
pre....	Nothing
ne...te	White

Lab 19 – solution - tb



*Final lab

Make a stopwatch

- Use the following components
  - Debouncers for Start – Stop – Reset
  - SR Flipflop for Start – Stop
  - Two 10 Counters for 1's and 10's
  - A clock divider for slowing down the clock from 50 MHz to 1 Hz

Solution – debounce – active high

```
entity deb is
    Port (                      sig : in  STD_LOGIC;
                                clk: in std_logic;
                                 sig_db : out  STD_LOGIC);
end deb;
architecture Behavioral of deb is
                signal Q1, Q2, Q3 : std_logic;
begin
                process(clk) begin
                                if rising_edge (clk) then
                                                Q1 <= sig;
                                                Q2 <= Q1;
                                                Q3 <= Q2;
                                end if;
                end process;
                sig_db <= Q1 and Q2 and  Q3;
end Behavioral;
```

Solution – SR-flipflop

```
entity SRFF is
   Port ( S : in  STD_LOGIC;
          R : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          Q : out  STD_LOGIC;
          Q_inv : out  STD_LOGIC);
end SRFF;
```

Solution – SR-flipflop

```
architecture Behavioral of SRFF is
begin
process (clk) begin
        if rising_edge (clk) then
                if (S = '1') then
                        Q <= '1';
                        Q_inv<= '0';
                elsif (R = '1') then
                        Q <= '0';
                        Q_inv <= '1';
                end if;
        end if;
end process;
end Behavioral;
```

Solution – Clock divider

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity ckdiv is
    Port (              clk : in  STD_LOGIC;
                        div : in std_logic_vector (1 downto 0);
                        clk_slow : out  STD_LOGIC);
end ckdiv;
```

Solution – Clock divider

```
architecture Behavioral of ckdiv is

signal i: integer range 0 to 25000000:=0;
signal int_clk: std_logic:= '0';
signal base: integer range 0 to 25000000:= 25000000;
signal prev_div: std_logic:= '0';
```

Solution – Clock divider

```
begin
process (clk, div) begin
        if rising_edge (clk) then
                i <= i + 1;
                if (i = base) then
                        i <= 0;
                        int_clk <= not int_clk;
                end if;
                case (div) is
                        when "00" => base <= 25000000;
                        when "01" => base <= 12500000;
                        when "10" => base <= 2500000;
                        when others => base <= 250000;
                end case;
        end if;
end process;

clk_slow <= int_clk;

end Behavioral;
```

empus

Solution – 10 counter

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity count10 is
    Port (    clk : in  STD_LOGIC;
              rst : in  STD_LOGIC;
              en : in  STD_LOGIC;
              co : out  STD_LOGIC;
              count : out  STD_LOGIC_VECTOR (3 downto 0));
end count10;
```

Solution – 10 counter

```
architecture Behavioral of count10 is
signal count_int: STD_LOGIC_VECTOR (3 downto 0);
begin
process (clk, rst) begin
        if (rst = '1') then
                count_int <= "0000";
        elsif (rising_edge (clk) and en = '1') then
                count_int <= count_int + 1;
                if count_int = "1001" then
                        count_int <= "0000";
                end if;
        end if;
end process;

count <= count_int;
co <= count_int(3) and count_int(0);

end Behavioral;
```

Solution – Quad 2 to 1 MUX

```
entity QMUX21 is
    Port ( sw : in  STD_LOGIC_VECTOR (7 downto 0);
           sel : in  STD_LOGIC;
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end QMUX21;

architecture Behavioral of QMUX21 is

begin
process (sw, sel) begin
                case (sel) is
                                when '0' => led <= sw (3 downto 0);
                                when '1' => led <= sw (7 downto 4);
                                when others => led <= "0000";
                end case;
end process;
end Behavioral;
```

Solution – 7 seg decoder

```
entity seg7 is
    Port (    sw : in  STD_LOGIC_VECTOR (3 downto 0);
              an : out  STD_LOGIC_VECTOR (3 downto 0);
              seg : out  STD_LOGIC_VECTOR (6 downto 0));
end seg7;
```

Solution – 7 seg decoder

```
architecture Behavioral of seg7 is
alias GFEDCBA: std_logic_vector (6 downto 0) is seg;
begin
process (sw) begin
        case (sw) is
                when "0000" => GFEDCBA <= "1000000";
                when "0001" => GFEDCBA <= "1111001";
                when "0010" => GFEDCBA <= "0100100";
                 when "0011" => GFEDCBA <= "0110000";
                when "0100" => GFEDCBA <= "0011001";
                when "0101" => GFEDCBA <= "0010010";
                 when "0110" => GFEDCBA <= "0000010";
                when "0111" => GFEDCBA <= "1111000";
                 when "1000" => GFEDCBA <= "0000000";
                when others => GFEDCBA <= "0010000";
        end case;
an <= "0000";
end process;
end Behavioral;
```

Solution – top module

```
entity top is
   Port ( strt : in  STD_LOGIC;
             stp : in  STD_LOGIC;
             rst : in  STD_LOGIC;
             clk : in  STD_LOGIC;
             seg : out  STD_LOGIC_VECTOR (6 downto 0);
        an : out  STD_LOGIC_VECTOR (3 downto 0));
end top;
```

Solution – top module

```
architecture Behavioral of top is

component seg7 is
   Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
              an : out  STD_LOGIC_VECTOR (3 downto 0); -- anodes of 7-segment
              seg : out  STD_LOGIC_VECTOR (6 downto 0)); -- 7 segements
end component seg7;

component QMUX21 is
   Port ( sw : in  STD_LOGIC_VECTOR (7 downto 0);
              sel : in  STD_LOGIC;
              led : out  STD_LOGIC_VECTOR (3 downto 0));
end component QMUX21;

component SRFF is
   Port ( S : in  STD_LOGIC;
          R : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          Q : out  STD_LOGIC;
          Q_inv : out  STD_LOGIC);
end component SRFF;
```

Solution – top module

```
component count10 is
    Port (    clk : in  STD_LOGIC;
              rst : in  STD_LOGIC;
              en : in  STD_LOGIC;
              co : out  STD_LOGIC;
              count : out  STD_LOGIC_VECTOR (3 downto 0));
end component count10;

component ckdiv is
    Port (    clk : in  STD_LOGIC;
              div : in std_logic_vector (1 downto 0);
              clk_slow : out  STD_LOGIC);
end component ckdiv;

component deb is
    Port (    sig : in  STD_LOGIC;
              clk: in std_logic;
              sig_db : out  STD_LOGIC);
end component deb;
```

Solution – top module

```
signal strt_db, stp_db, rst_db, en, co, clk_2Hz, clk_100Hz: std_logic;
signal cnt_10, cnt_1, dec_in: std_logic_vector (3 downto 0);
signal cnt: std_logic_vector (7 downto 0);
```

Solution – top module

```
begin

cnt <= cnt_10 & cnt_1;

start_db: deb
            Port map(            sig => strt,
                                  clk => clk,
                                 sig_db => strt_db);
stop_db: deb
            Port map(            sig => stp,
                                 clk => clk,
                                 sig_db => stp_db);
reset_db: deb
            Port map(            sig => rst,
                                 clk => clk,
                                 sig_db => rst_db);
```

Solution – top module

```
SR_flfl: SRFF
        Port map(       S => strt_db,
                        R => stp_db,
                        clk => clk,
                        Q => en,
                        Q_inv => open);
clk_tel: ckdiv
        Port map(       clk => clk,
                        div => "01",
                        clk_slow => clk_2Hz);
clk_mux_an: ckdiv
        Port map(       clk => clk,
                        div => "11",
                        clk_slow => clk_100Hz);
```

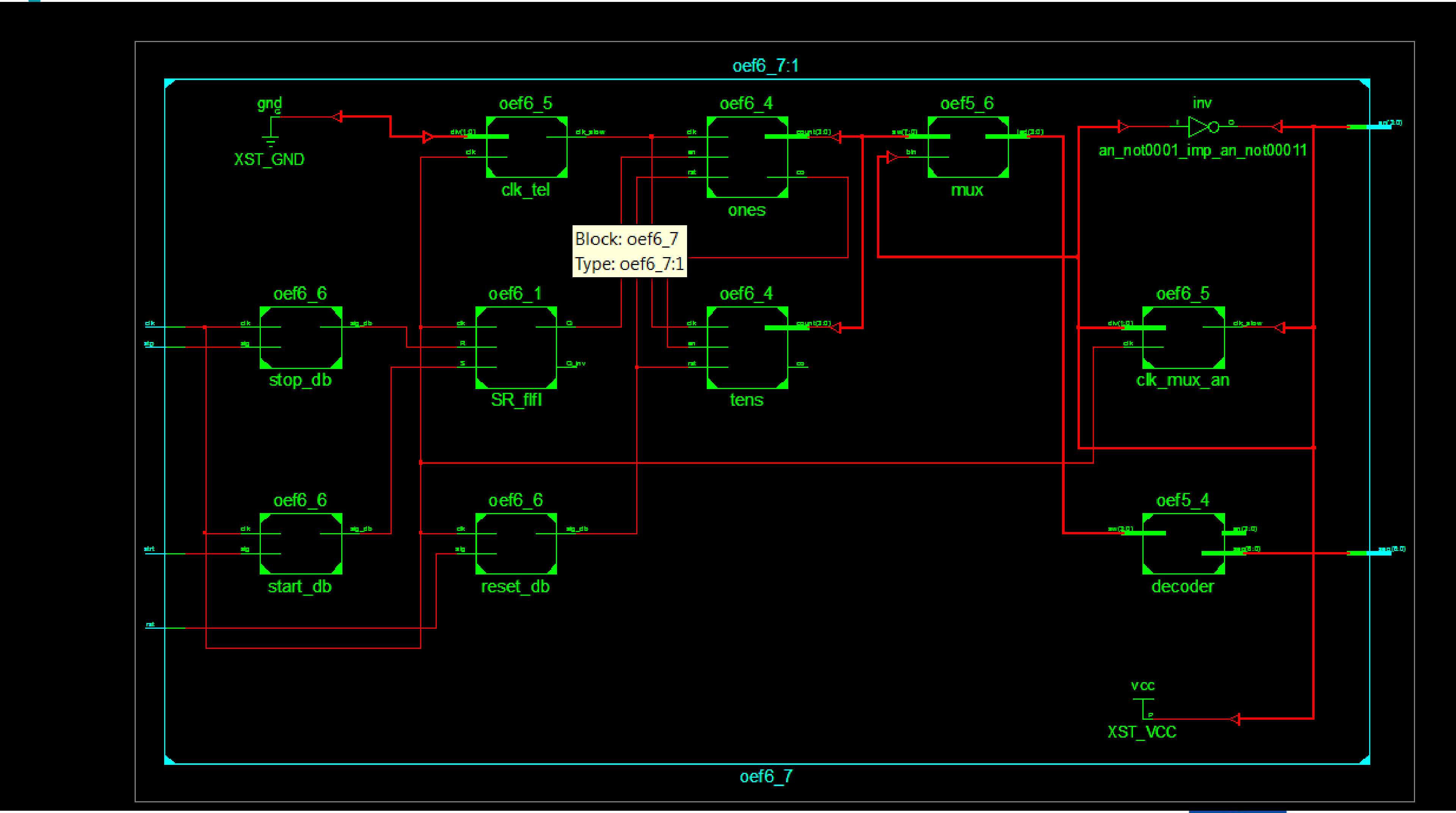
Solution – top module

```
ones: count10
		Port map(		clk => clk_2Hz,
					rst => rst_db,
					 en => en,
					co => co,
					count => cnt_1);
tens: count10
		Port map(		clk => clk_2Hz,
					rst => rst_db,
					en => co,
					co => open,
					count => cnt_10);
mux: QMUX21
		Port map(		sw => cnt,
					sel => clk_100Hz,
					led => dec_in);
decoder: seg7
		Port map(		sw => dec_in,
					an => open,
					seg => seg);
```

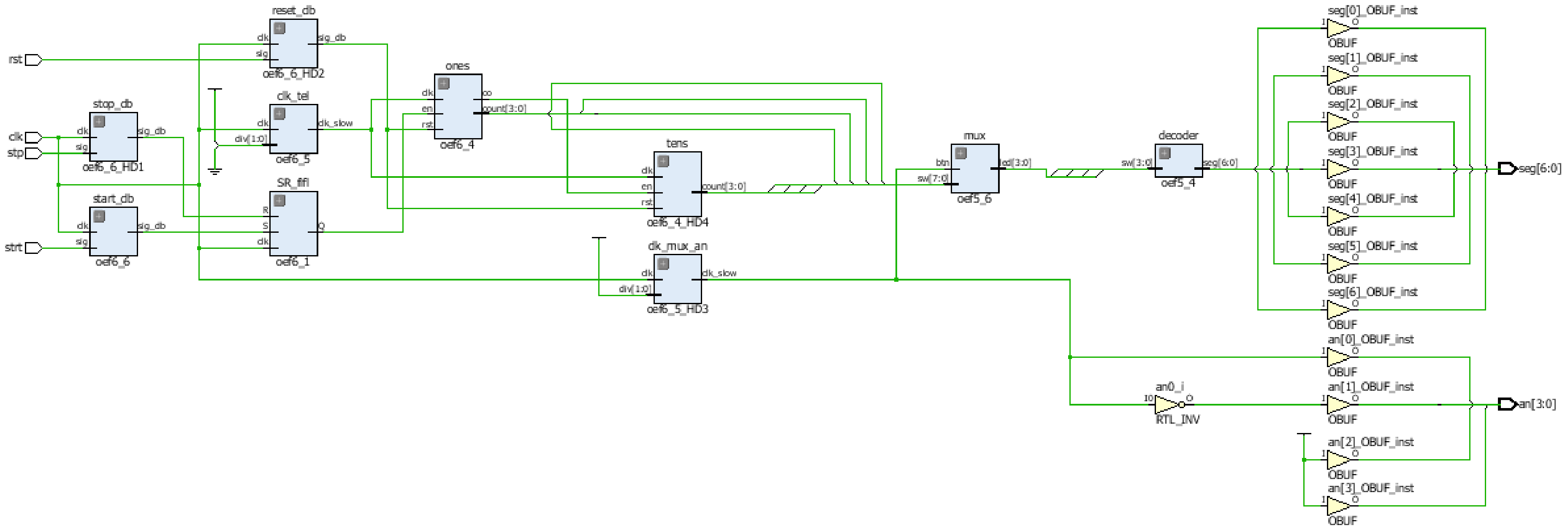
Solution – top module

```
an <= "11" & not clk_100Hz & clk_100Hz;
end Behavioral;
```

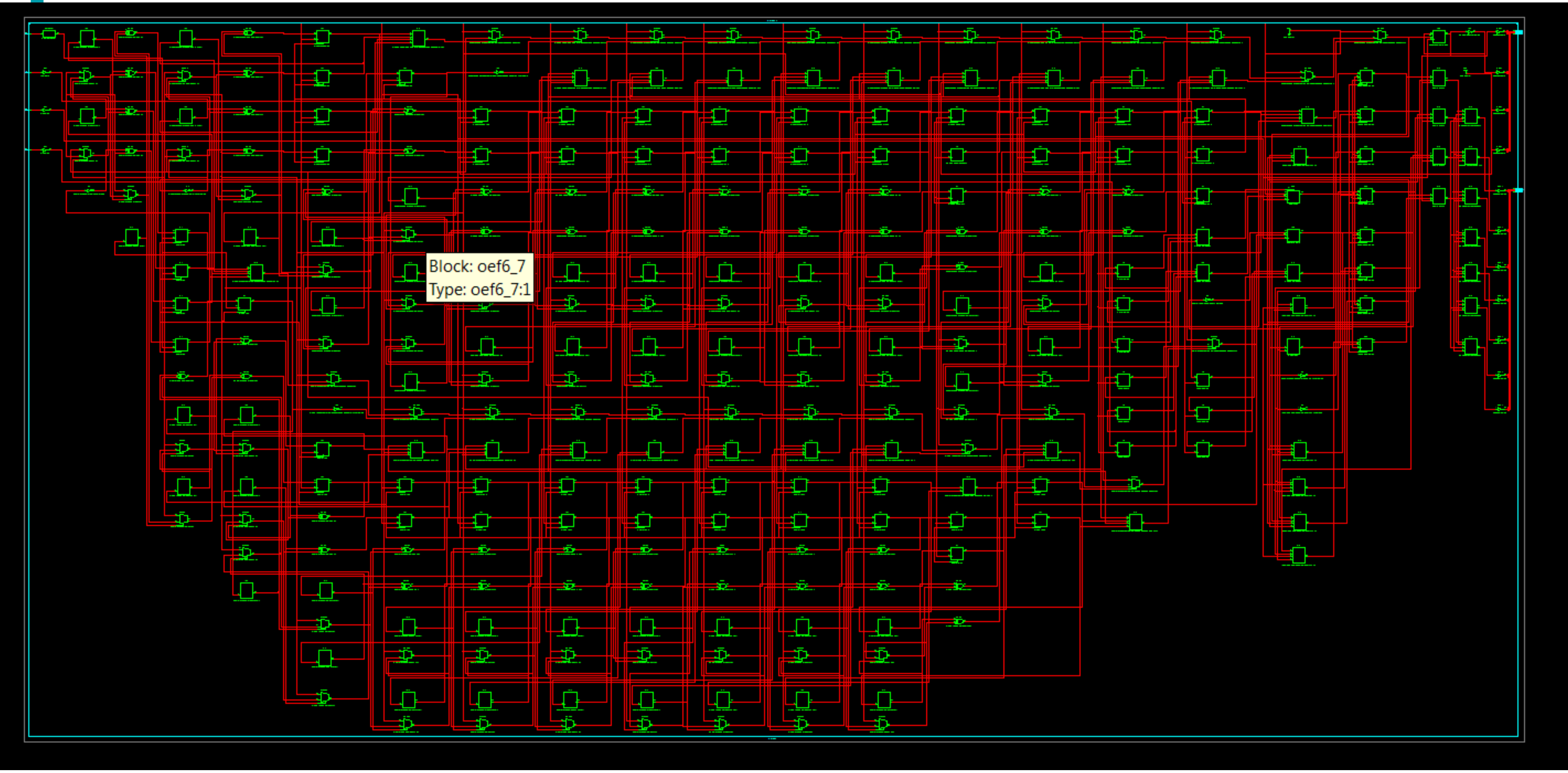
Solution – RTL



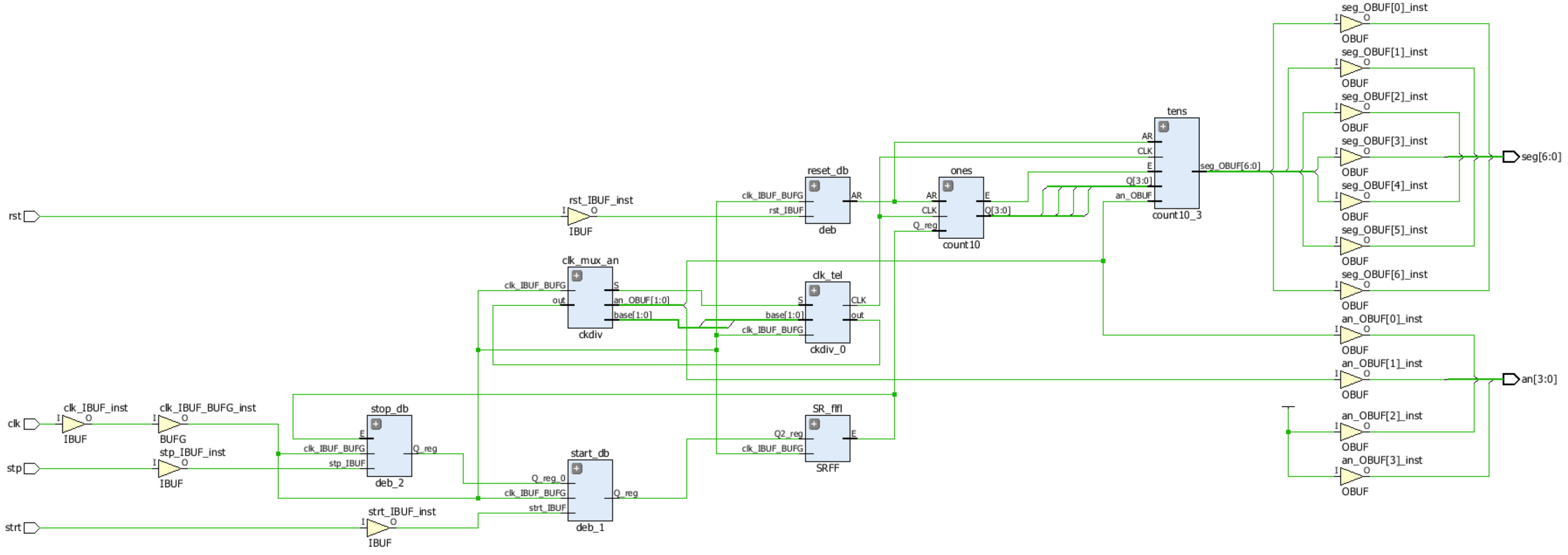
Solution – RTL



Solution – Technology



Solution – Synthesis



Solution

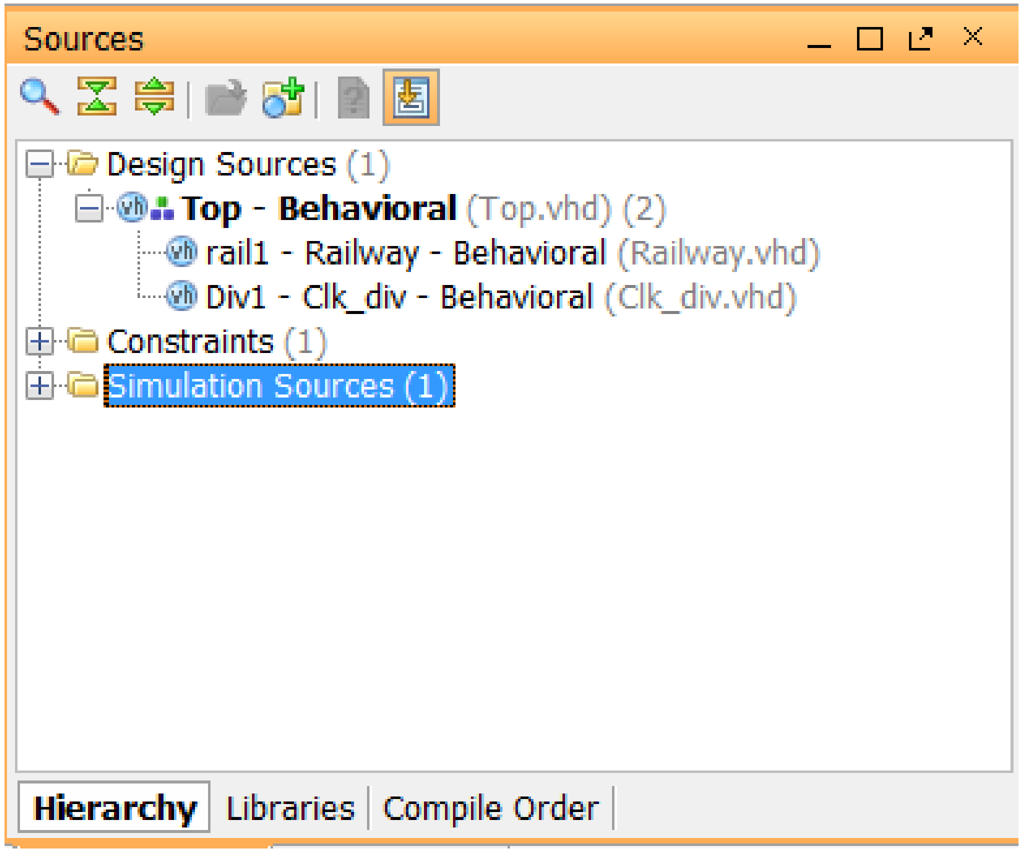
Where is the bug?

- What happens if you push stop at “09”
- How to solve this?

Extra

Because of the fact that the system clock is to fast it is difficult to control the correct operation of your design.
This is solved by adding a clock divider.
Do this for 13 – 14 – 15 – 16 – 18 – 19

19_2



Sources

Design Sources (1)
Top - Behavioral (Top.vhd) (2)
rail1 - Railway - Behavioral (Railway.vhd)
Div1 - Clk_div - Behavioral (Clk_div.vhd)
Constraints (1)
Simulation Sources (1)

Hierarchy | Libraries | Compile Order

19_2 - Top

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Top is
   Port ( clk : in STD_LOGIC;
          clr : in STD_LOGIC;
          SF : in STD_LOGIC;
          SB : in STD_LOGIC;
          W : out STD_LOGIC;
          R1 : out STD_LOGIC;
          R2 : out STD_LOGIC;
          clk_slow : out STD_LOGIC;
          div : in STD_LOGIC_VECTOR (1 downto 0));
end Top;
```

19_2 - Top

```
architecture Behavioral of Top is
component Railway is
    Port ( clk : in STD_LOGIC;
           clr : in STD_LOGIC;
           SF : in STD_LOGIC;
           SB : in STD_LOGIC;
           W : out STD_LOGIC;
           R1 : out STD_LOGIC;
           R2 : out STD_LOGIC);
end component Railway;

component Clk_div is
    Port ( clk : in STD_LOGIC;
           div : in std_logic_vector (1 downto 0);
           clk_slow : out STD_LOGIC);
end component Clk_div;

signal clk_slw: std_logic;
```

19_2 - Top

```
begin

rail1: Railway
Port map (clk => clk_slw, clr => clr, SF => SF, SB => SB, W => W, R1 => R1, R2 => R2);

Div1: Clk_div
Port map (clk => clk, div => div, clk_slow => clk_slw);

clk_slow <= clk_slw;

end Behavioral;
```

19_2 - Top

```
begin

rail1: Railway
Port map (clk => clk_slw, clr => clr, SF => SF, SB => SB, W => W, R1 => R1, R2 => R2);

Div1: Clk_div
Port map (clk => clk, div => div, clk_slow => clk_slw);

clk_slow <= clk_slw;

end Behavioral;
```

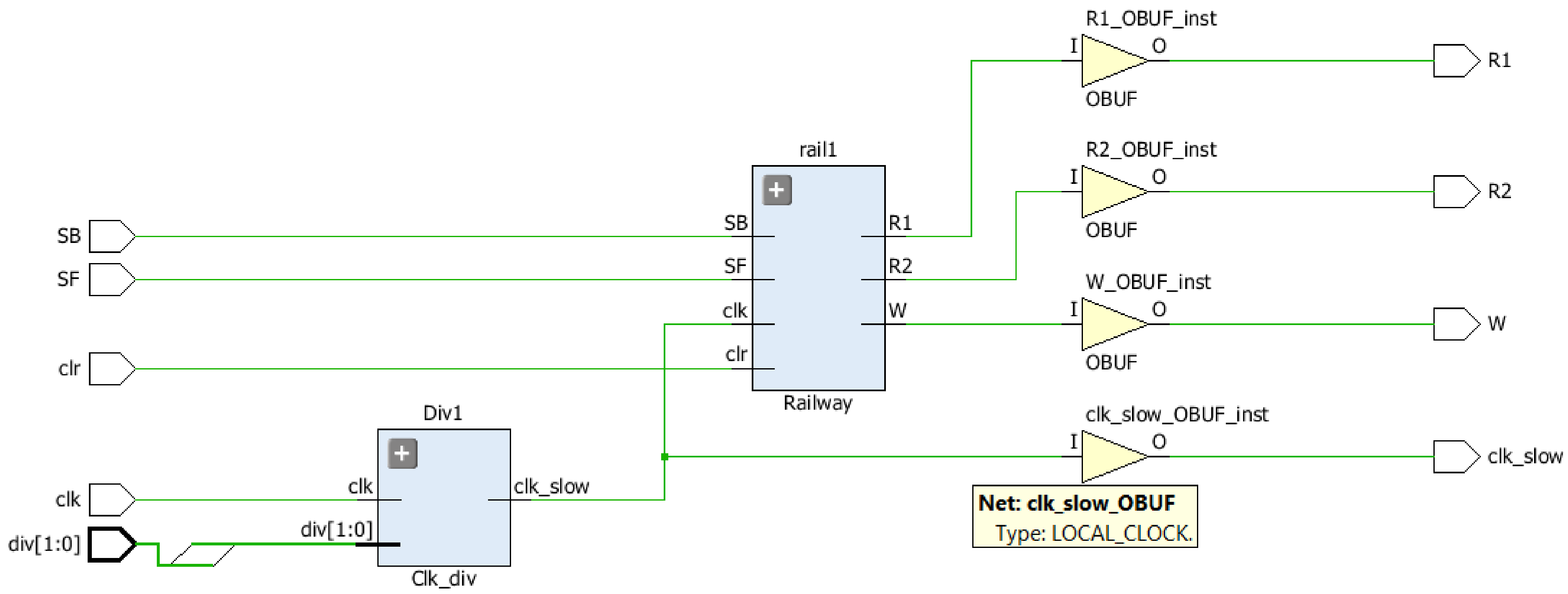
19_2 - XDC

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]

        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {clr}]
        set_property IOSTANDARD LVCMOS33 [get_ports {clr}]
set_property PACKAGE_PIN V16 [get_ports {SF}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SF}]
set_property PACKAGE_PIN W16 [get_ports {SB}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SB}]
set_property PACKAGE_PIN W17 [get_ports {div[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {div[0]}]
set_property PACKAGE_PIN W15 [get_ports {div[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {div[1]}]
```

19_2 - XDC

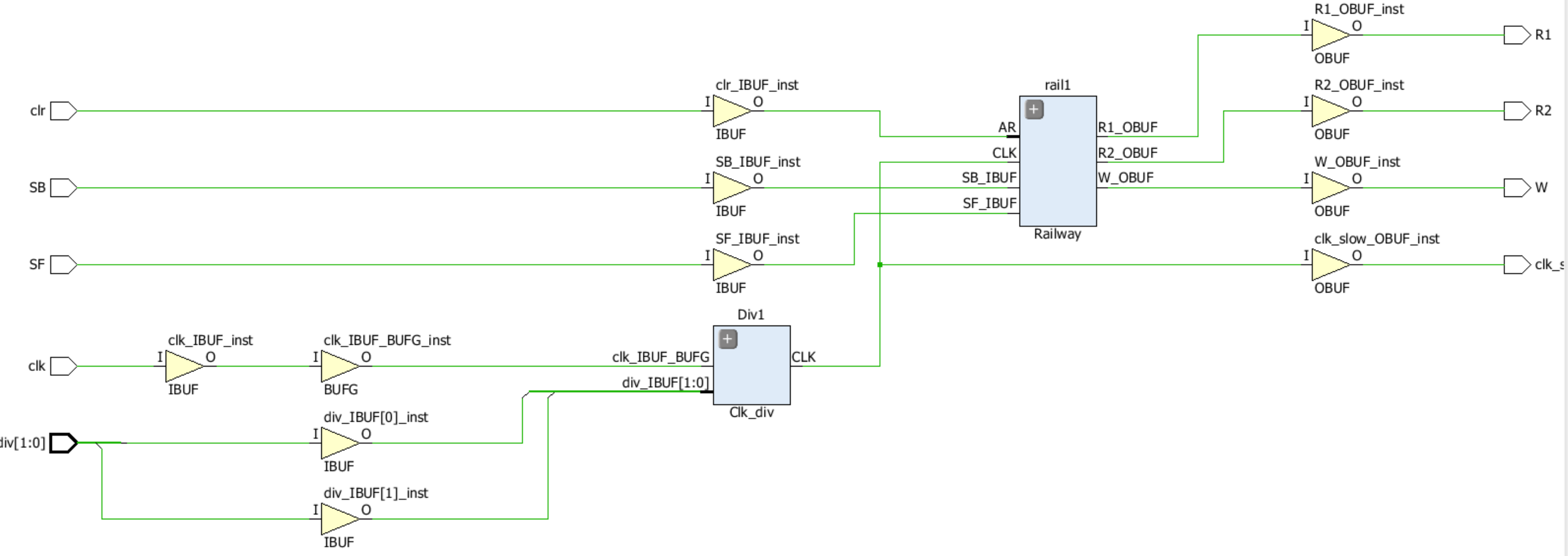
```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {W}]
	set_property IOSTANDARD LVCMOS33 [get_ports {W}]
set_property PACKAGE_PIN E19 [get_ports {R1}]
	set_property IOSTANDARD LVCMOS33 [get_ports {R1}]
set_property PACKAGE_PIN U19 [get_ports {R2}]
	set_property IOSTANDARD LVCMOS33 [get_ports {R2}]
set_property PACKAGE_PIN V19 [get_ports {clk_slow}]
	set_property IOSTANDARD LVCMOS33 [get_ports {clk_slow}]
```

19_2 - RTL

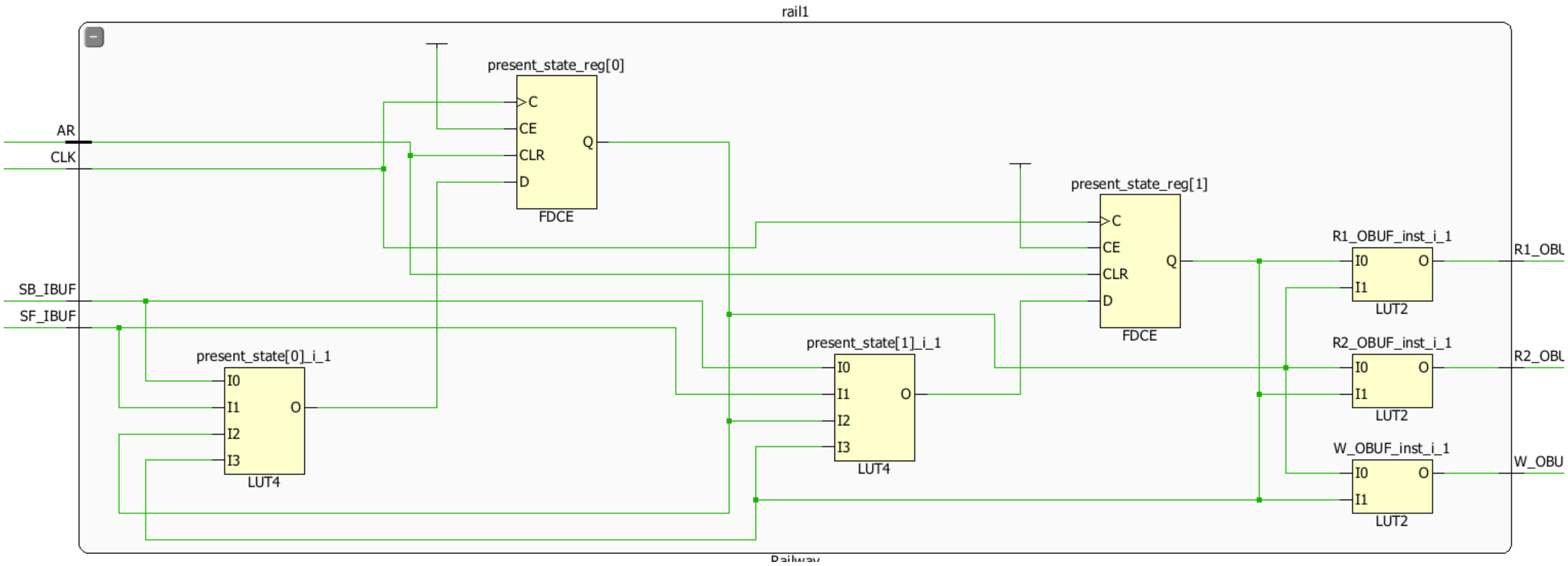


Net: clk_slow_OBUF
Type: LOCAL_CLOCK.

19_2 - Synthesis



19_2 - Synthesis



Contact

Ing. Dirk Van Merode MSc.
Project Coordinator DESIRE

Thomas More | Campus De Nayer
Technology & Design
J. P. De Nayerlaan 5

2860 Sint-Katelijne-Waver

Belgium
Tel. + 32 15 31 69 44
Gsm + 32 496 26 84 15

dirk.vanmerode@thomasmore.be

Skype dirkvanmerode

www.thomasmore.be

Dr. Ing. Peter Arras MSc.
International Relations Officer

KU Leuven | Campus De Nayer
Faculty of engineering technology
J. P. De Nayerlaan 5

2860 Sint-Katelijne-Waver

Belgium
Tel. + 32 15 31 69 44
Gsm + 32 486 52 81 96

peter.arras@kuleuven.be

Skype pfjlarras

www.iiw.kuleuven.be

Sources

http://www.allaboutcircuits.com/textbook/digital/chpt-4/contact-bounce/

Digital Design Using Digilent FPGA Boards
Richard E. Haskell
Darrin M. Hanna
LBE Books – Third Edition, 2014