FPGA-Tutorial

Jochen Steinmann

RWTH Aachen University

Lecture 01

28.10.2020

Organisation I

- **Date and Place**
  - Remote via ZOOM (link in Moodle)
  - Wednesday 14:30 – 16:00 obligatory
    16:00 – 17:30 optional
- **Structure**
  - Solution of the last Excercise
  - Presentation (FPGA / Electronics / VHDL)
  - Hand‘s on tutorial
  - „Homework“
- **Requirements for this course**
  - CIP Pool Login
  - X2Go connection
- Slides and Skeleton programs will be (hopefully) inside the Moodle before the tutorial

About myself

- Dr. Jochen Steinmann
  jochen.steinmann@physik.rwth-aachen.de

- **Duties:**
  JUNO: OSIRIS - Design of a Development of a PMT readout system
  consists of Microcontroller, FPGA, ADC etc.

  Responsible for hardware design.

Scope of this tutorial

- **What this tutorial cannot do?**
  - Improve your soldering skills
  - Make you an electrical engineer
  - Make you an FPGA „expert“

- **What this tutorial can do?**
  - Help you understanding electrical engineers talking about FPGA / Programmable Logic
  - Help you programming a basic firmware
  - Get an overview of programmable logic
    - Problems and their solutions
    - Features and how they can be used

Lecture
Overview about detectors

GEANT4
Simulation of detectors

FPGA
Readout of detectors

Motivation

Why does a Physicist needs FPGA-Knowledge?

- All „big“ Experiments use readouts / trigger based on FPGAs
- Understanding the processes inside a FPGA helps troubleshooting the experiment

- Typical distribution of work
  - Electronic workshop:
    - PCB layout
    - Production of the real hardware

  - Physicists
    - Firmware
    - Tests
    - Commissioning

Why programmable Logic

State of the art

- Designing Prototype electronics
- Allow „easy“ reconfiguration
  - If you need a counter, just implement it
  - If you don‘t need it – remove it

  - Decisions can be done, when the PCB is produced
    - You can make connections inside the Chip – no need to solder a wire

Proper PCB layout and design is mandatory!

- Firmware has to match the experiment
  - Physicists should know all about their experiment
  - They know all the timing issues etc.

- If you want to build a custom chip (ASIC), the design flow is almost the same!
  - Same language

Why programmable IC`s?

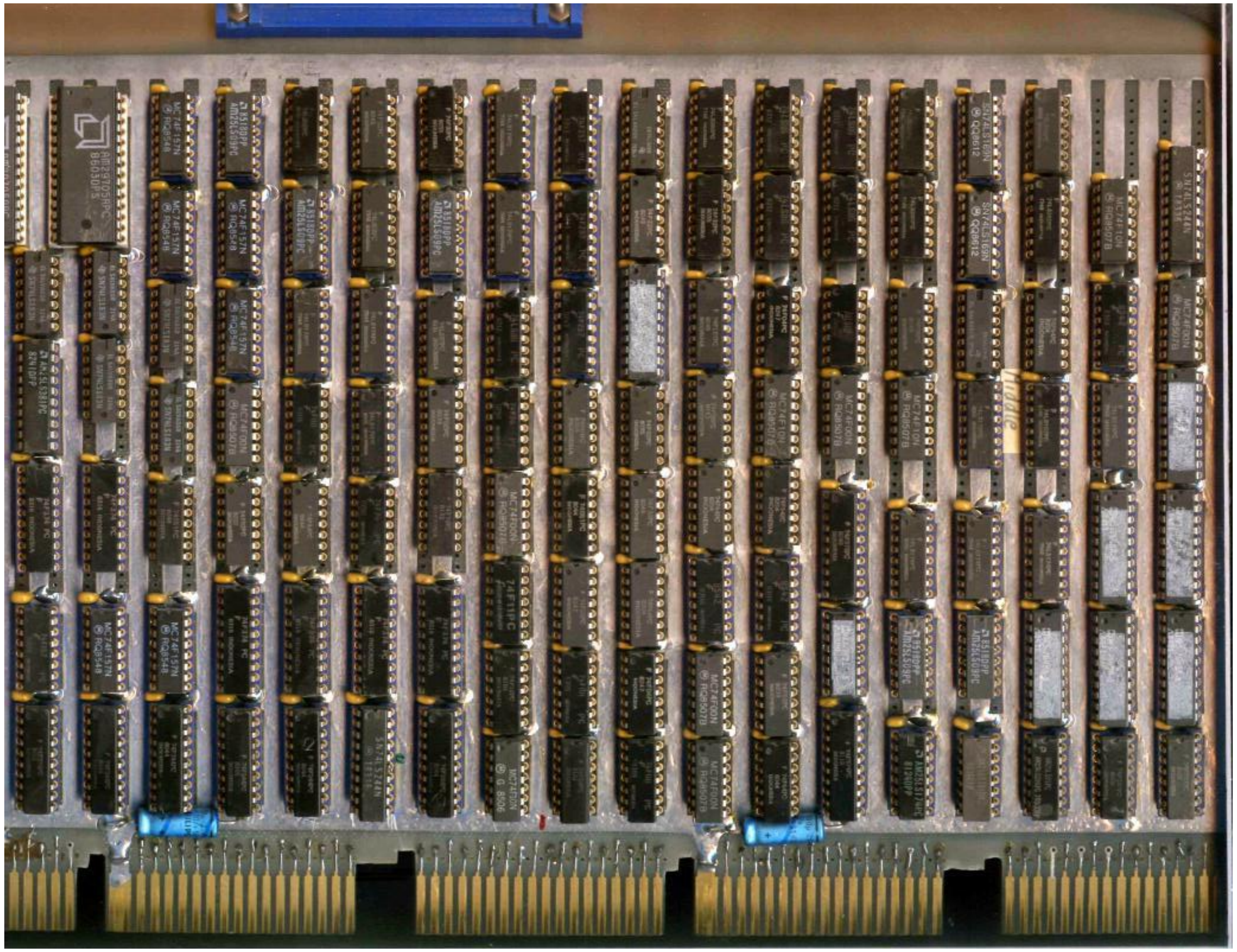
- Producing a chip costs much more than producing a PCB
  - Using the same chip on many PCBs increases number of chips
  - Reduces costs per chip

- If a chip is programmable, it can be used on many PCBs
  - Different „firmware“ for different situations

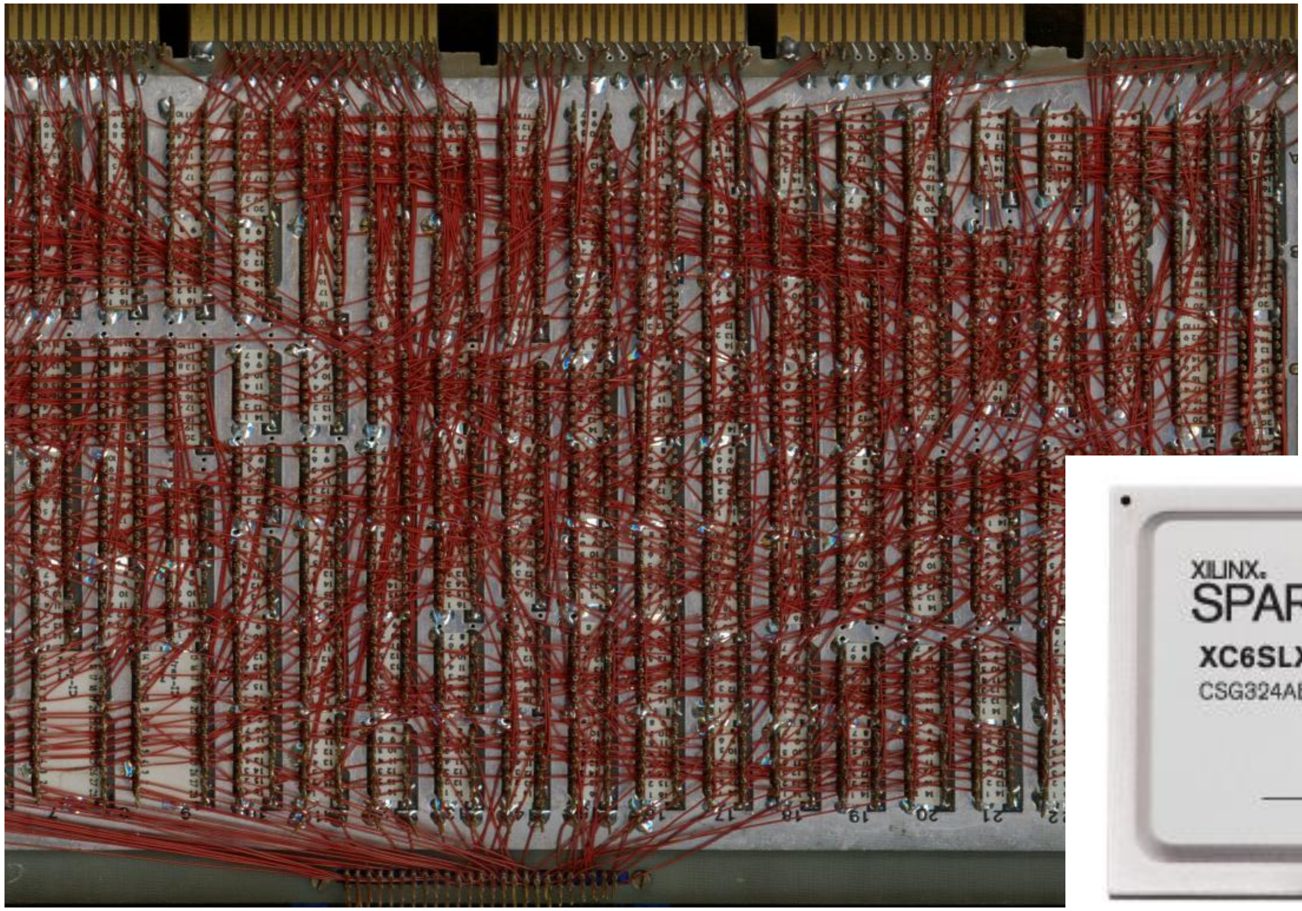
- Fast and cheap turnaround times
  - FPGA:
    - Firmware test takes several hours
    - Easy debugging (using on chip debug features)
    - No new PCB design needed

  - ASIC development
    - Time between Tape Out and Chip back 3 - 4 month
    - If you have a mistake in your silicon, you will notice it about 6 month later

Early times of digital electronics



Early times of digital electronics



History of Programmable Logic Devices

1960th
- Fuse Configurable Diode Matrix

1971
- Programmable ROM

1978
- Programmable Array Logic **PLA**

PLA Programmable Logic Array
every boolean operation
fixed logic gates and programmable interconnections (matrix)
OTP (one time programmable)

- Gate Array Logic **GAL** reprogrammable

- Complex Programmable Logic Device **CPLD** In circuit programmable
  Configured during programming → keep configuration during power cycle

1989
- Field Programmable Gate Array **FPGA**

Development process CLPD & FPGA is almost the same

What are FPGAs?

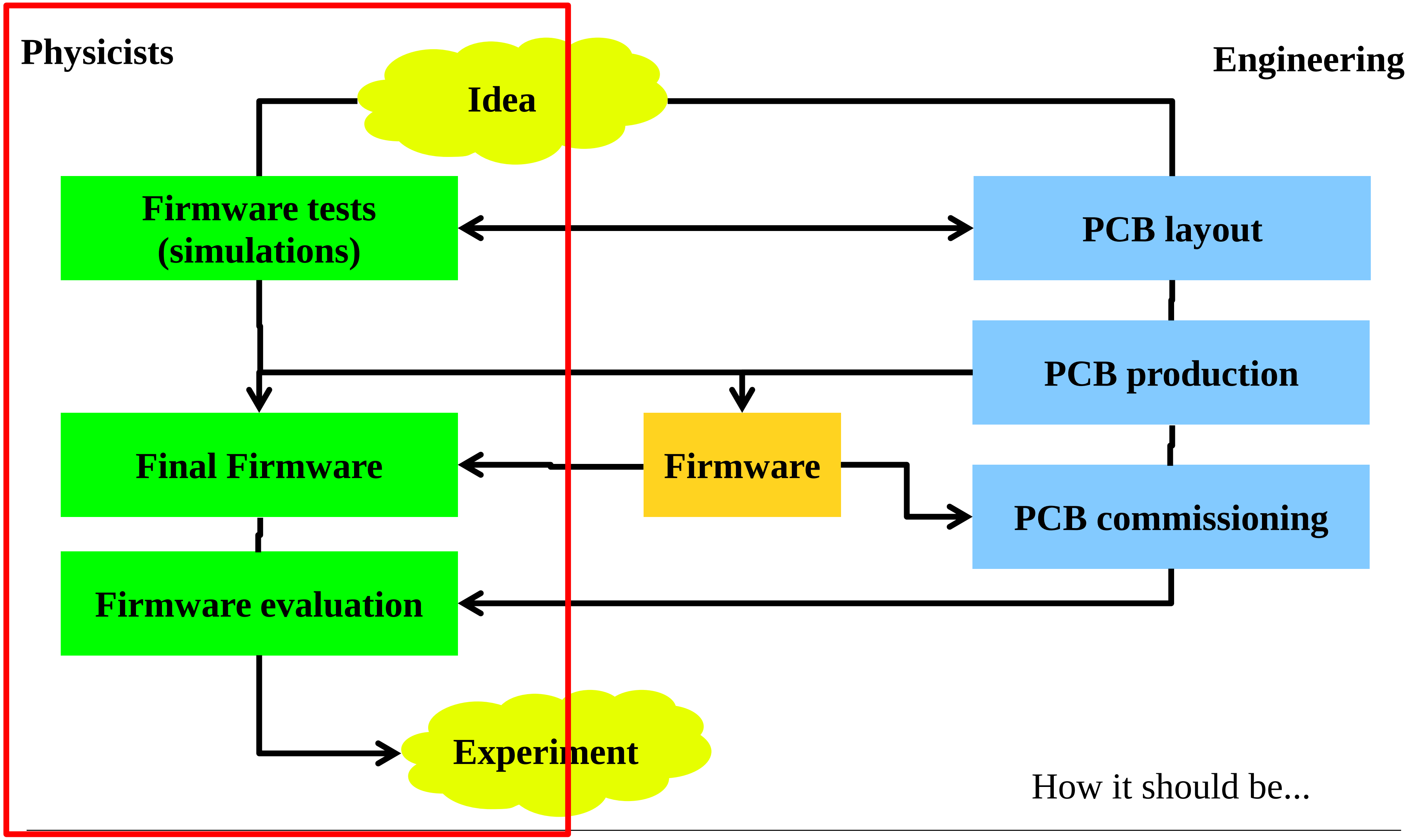
Programmable Logic Devices

- More detailed
  - Lot of additional function-blocks
    - RAM, DSP
  - Flexible configuration, recofiguration in the experiment
  - Are able to host mikrocontrollers

Usage of FPGAs:

- Deterministic calculation
- Well defined timing between input and output
- High speed data analysis & manipulation

- Easy interface to
  - Memory different types SRAM, DDR3, DDR4
  - Ethernet 100Mbit/s, 1000Mbit/s
  - PCI / PCIexpress
  - High Speed serial transceiver (Virtex7 up to 2.8 Tb/s in conmbined mode)



How it should be...

Programming an FPGA

Better describe the functions …

- **Everything the FPGA should do, is described using a HDL**
  - Hardware Description Language (we use VHDL)
  - No programming and compiling
- Use GHDL as a simulation tool
- Why VHDL?
  - Alternative: Verilog
  - Very structured
  - Highly used in European hardware development projects
- Origin of HDL in development of Chips
  - HDL were used to describe the behavior of the chip and the simulation stimulus
  - another language was used to create the netlists needed for production
  - At some point it has been decided to use one language for both processes

Use VHDL

How to program a FPGA?

No programming, just description!

Hardware Description HDL	Describe **what** the hardware should do.
Syntax Check	Check Syntax
Synthesis	“compiles” the design to transform HDL source into an architecture-specific design netlist
Translate	
Map	Fits the design into the available resources on the target device, and optionally, places the design
Place & Route	Places and routes the design to the timing constrains
Programming	Write configuration into the FPGA or configuration memory

III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY

VHDL

<u>V</u>ery High Speed Integrated Circuit <u>H</u>ardware <u>D</u>escription <u>L</u>anguage

- The logic inside FPGAs are described
  - If you call it correctly we are describing the hardware inside the FPGA
  - It‘s not software or firmware – it‘s **hardware**!

Naming convention

- Filenames

.m.vhd	modules
.tb.vhd	testbenches

- Inputs/Outputs

i_sl_<name>	in std_logic
o_sl_<name	out std_logic

VHDL

```
---------------------------------------
-- lecture 01 simple gate
-- just forwarding the signal
---------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------

entity lecture01 is
port(
            i_sl_x : in  std_logic;
            o_sl_F: out std_logic
);
end lecture01;

----------------------------------------

architecture behaviour of lecture01 is
begin

    process(i_sl_x)
    begin
        -- compare to truth table
        if (i_sl_x = '1') then
            o_sl_F <= '1';
        else
            o_sl_F <= '0';
        end if;
    end process;
end behaviour;

------------------------------------------
```

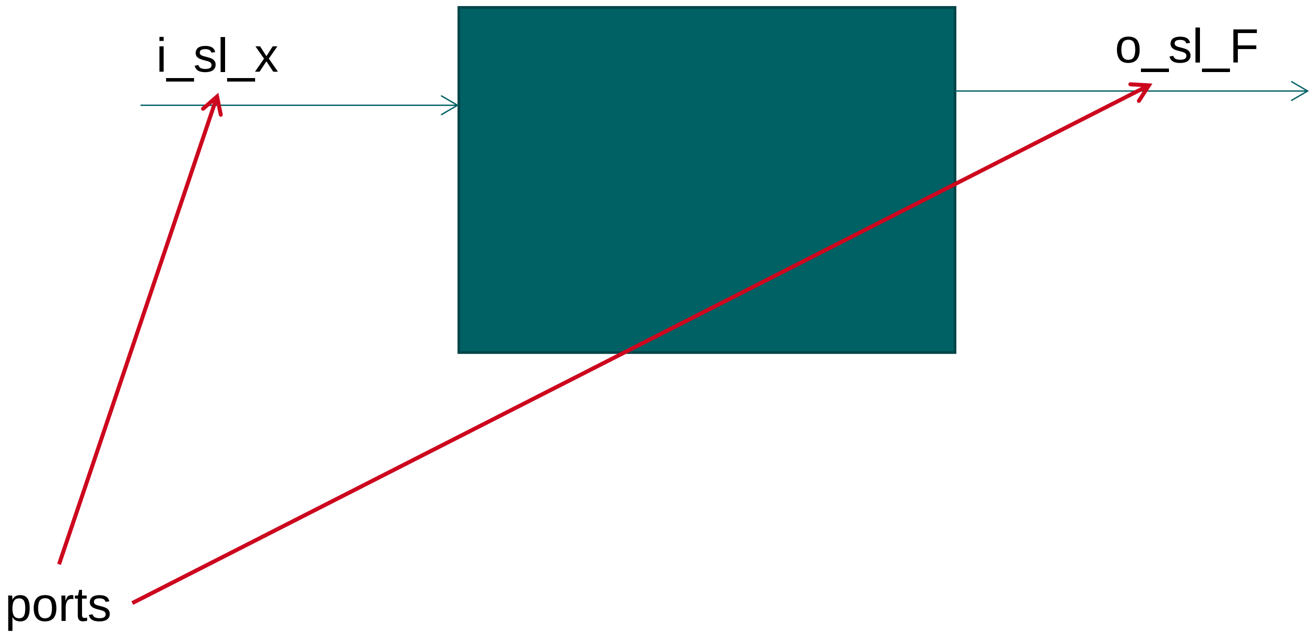
last line in port list without ;

<= assign a value to **o_sl_F**

VHDL introduction

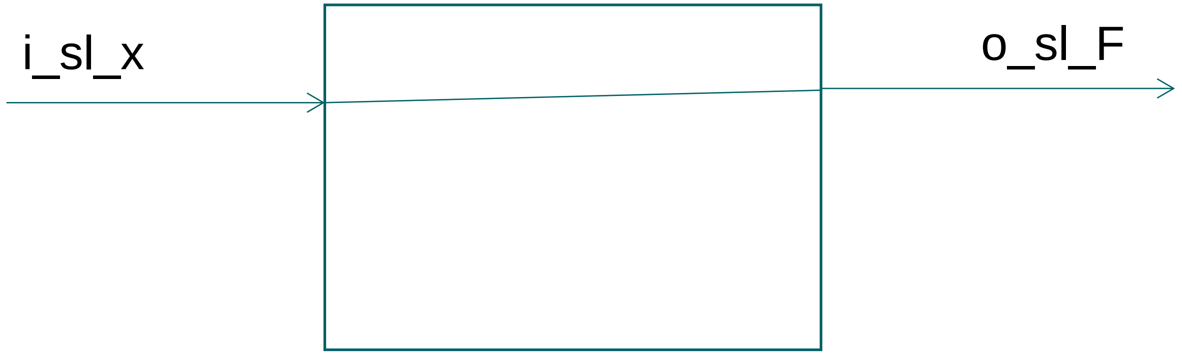
entity

- Describes our module from outside



architecture

- Describes what is inside our module



VHDL operators

Logic operators:

and, or, nand, nor, xor, not

Equality operators:

```
=  -- equal to
/= -- not equal to
```

Relational operators:

```
<   -- less than
>   -- greater than
<= -- less than or equal to
>= -- greater than or equal to
```

Variable Assignment (we will use it later as well)

:=

Signal Assignment

<=

```
signal s1 : std_logic;
variable v1 : std_logic;

s1 <= '0';
v1 := '1';
```

difference signal / variable later …

First exercise

- Modify the skeleton lecture_01, that it includes
  - All logic operators at the same time
  - e.g. implement an additional o_sl_and <= x and y;

Thank you!