Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

WORKING WITH THE FPGA VISION REMOTE LAB

Andrea Schwandt
Institute of Visual Computing, Bonn-Rhein-Sieg University

Topics

FPGA and VHDL Fundamentals – a Short Overview

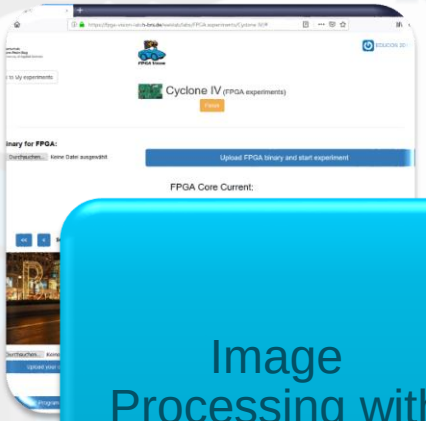
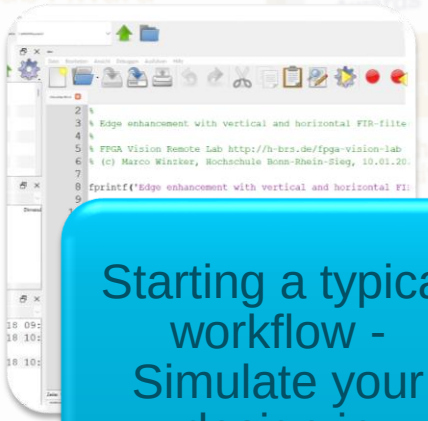
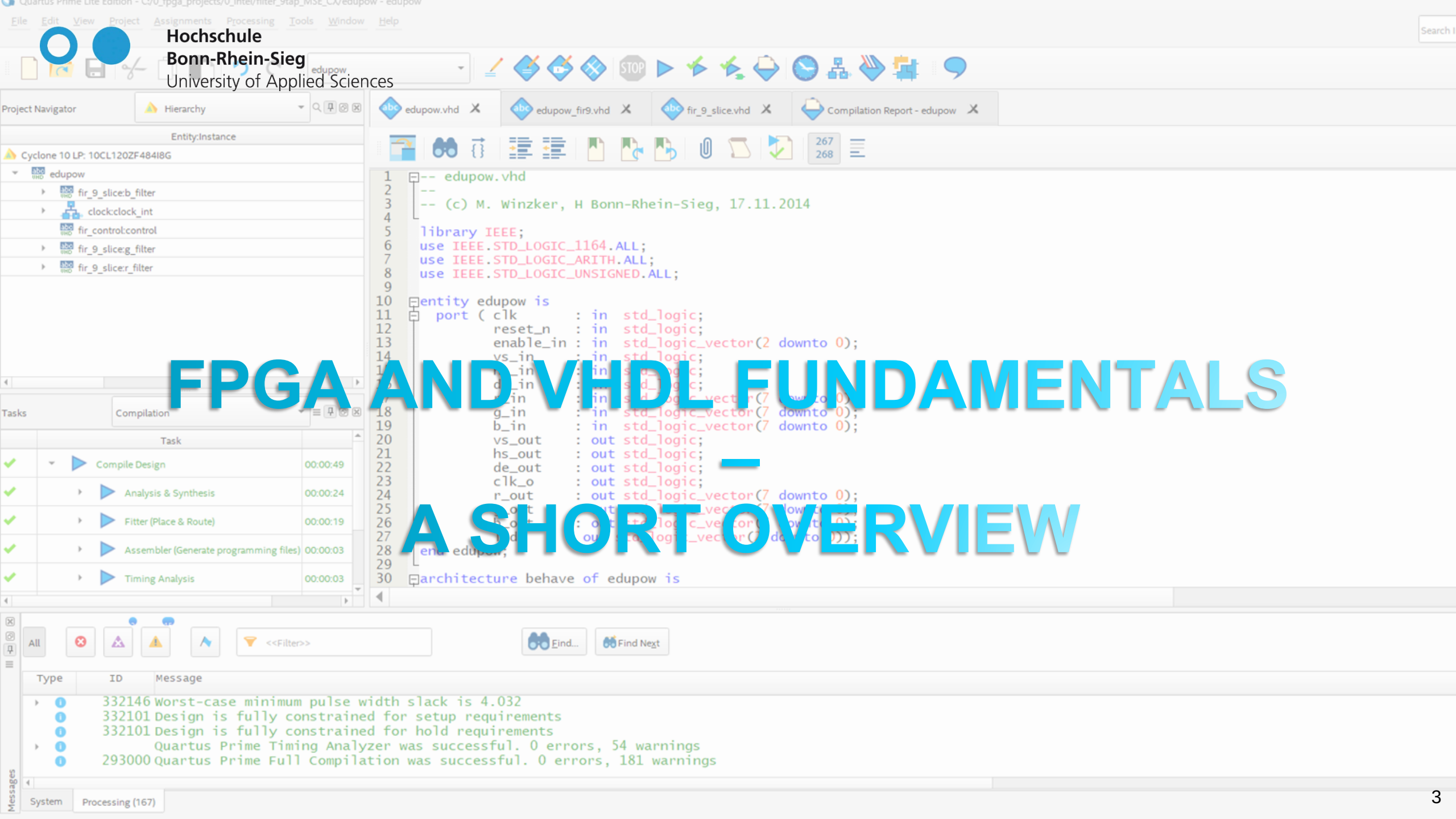


Image Processing with FPGAs



Starting a typical workflow - Simulate your design in Octave and Questa

Setting up a project in Quartus and test the design with the FPGA Vision Remote Lab


Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences
FPGA AND VHDL FUNDAMENTALS
–
A SHORT OVERVIEW

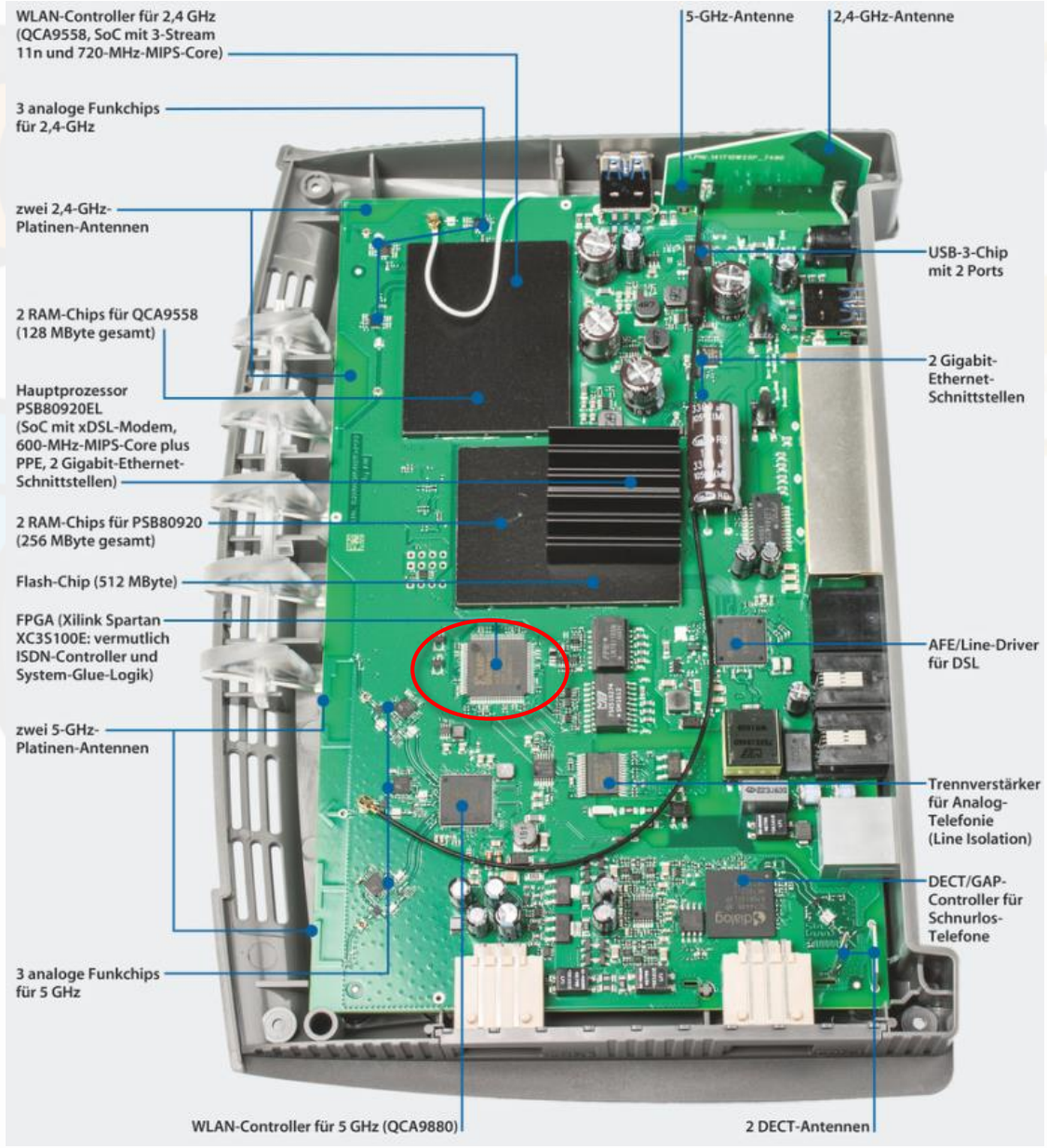
Where do we use FPGAs?

- as glue-logic
- for video processing
- for automotive applications
- …

Glue-Logic

AVM FritzBox 7490

from c't 17/2016, p. 162



Video processing

Nextera

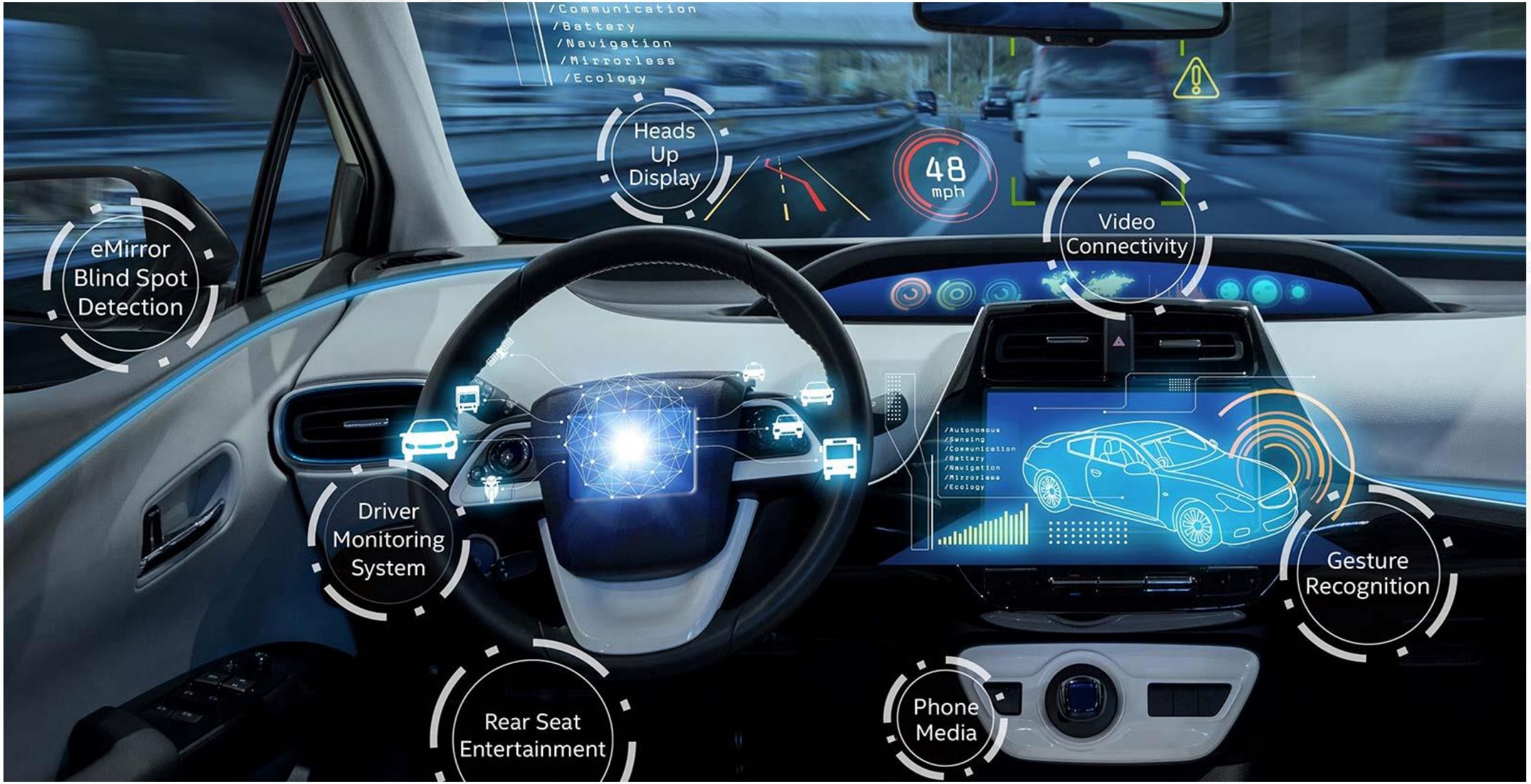
Video Solutions

Twitch

Live Video Streaming Powered by Xilinx FPGA Acceleration and NGCodec VP9 Transcoder

https://www.xilinx.com/about/xilinx-go/powered-by-xilinx.html

Automotive



source: https://www.intel.de/content/www/de/de/automotive/products/programmable/applications.html

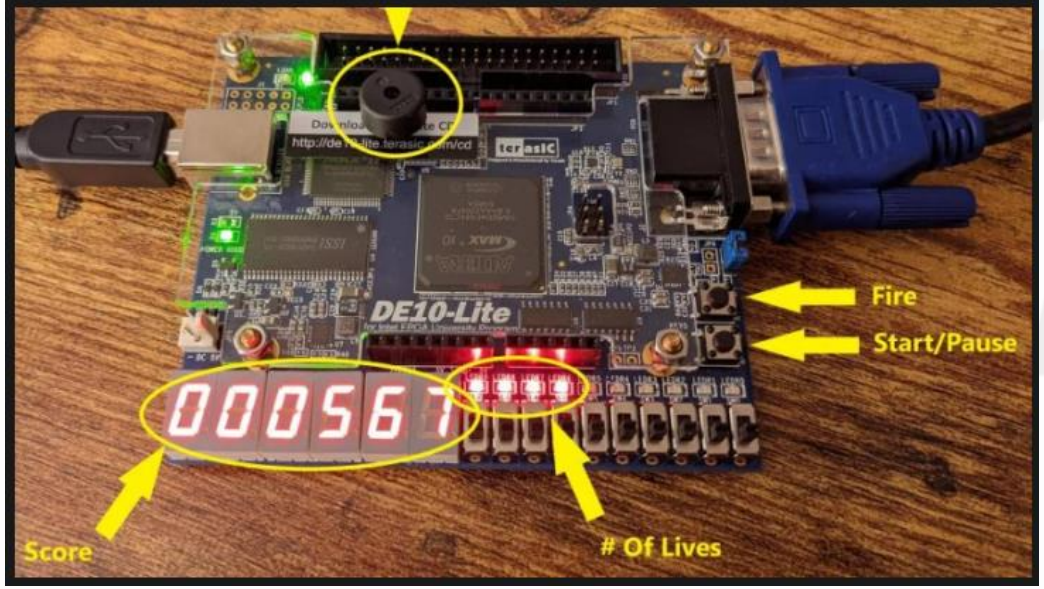
HACKADAY

DE10-LITE-FUL FPGA DEV BOARD HACK PLAYS THE 1981 CLASSIC DEFENDER

by: Ryan Flowers

6 Comments

January 8, 2022



VHDL Overview

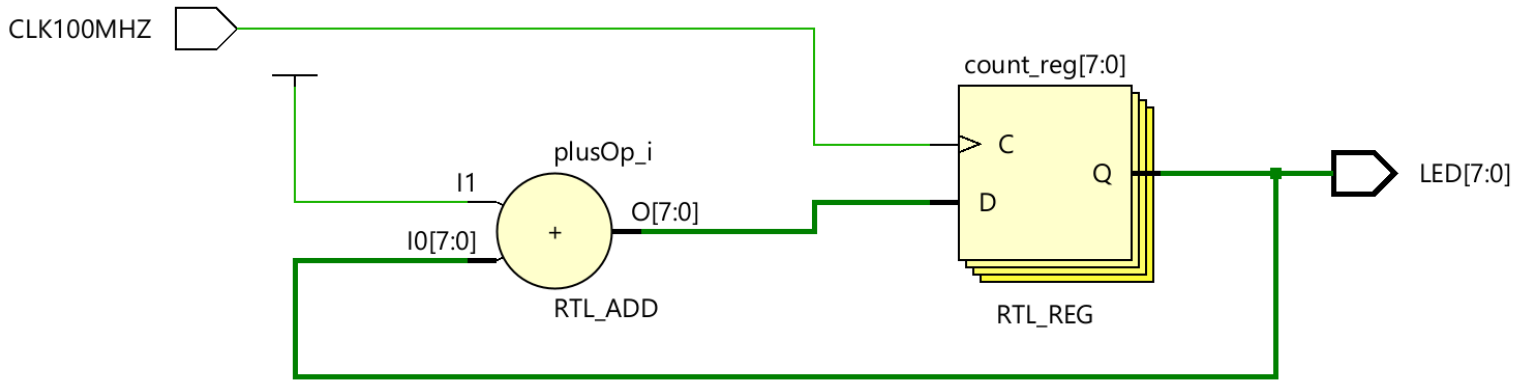
VHDL is a **hardware description language** for **design** and **verification** of digital circuits

- HDL stands for **H**ardware **D**escription **L**anguage
- V stands for the initiative „VHSIC“ (Very High Speed Integrated Circuits), where VHDL originated

VHDL Design

```
process
begin
    wait until rising_edge(CLK100MHZ);

        count <= count + 1;

end process;

LED <= std_logic_vector(count(7 downto 0));
```

- VHDL describes the function of a circuit
- With the VHDL description the design/development suite creates the circuit

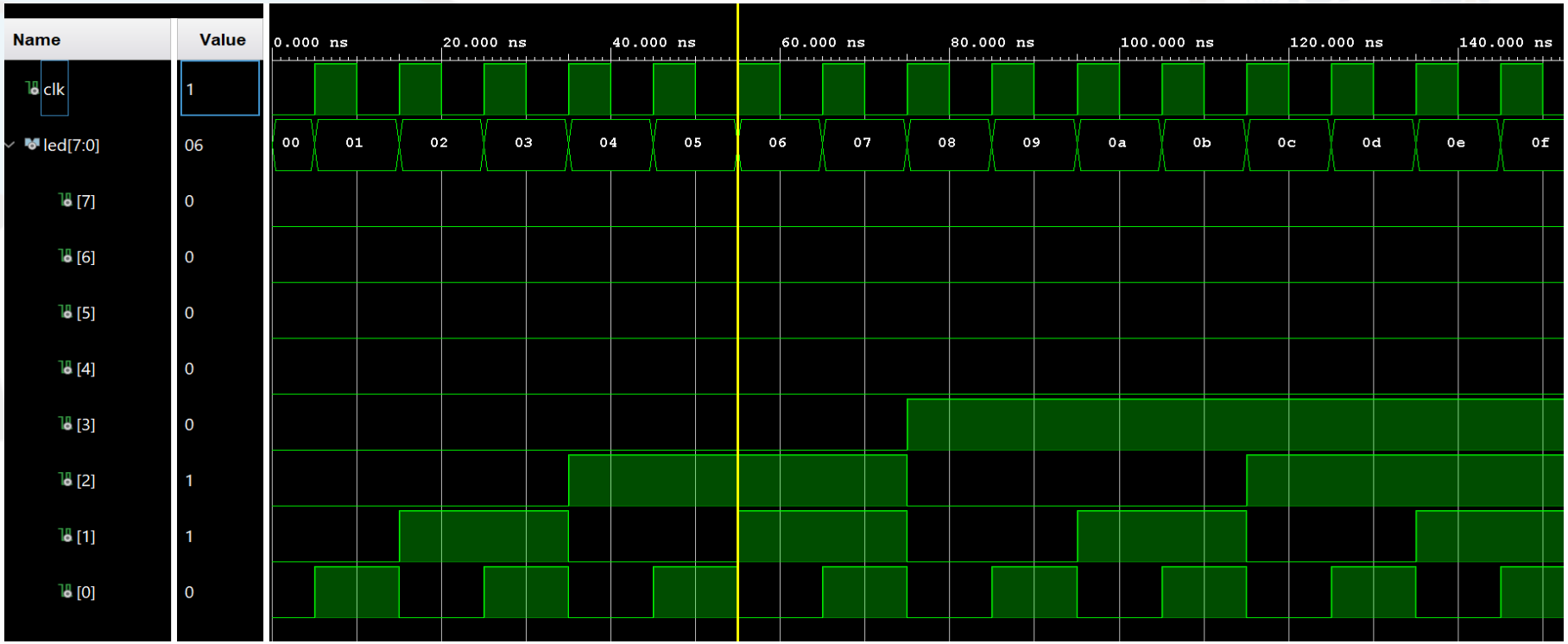


VHDL-Verification

```
-- clock generation
clk <= not clk after 5 ns;

count_1: entity work.zaehler
    port map (CLK100MHZ  => clk,
              LED        => led);
```

- Verifies the correct behavior of the circuit
- Testbench provides the I/O signals

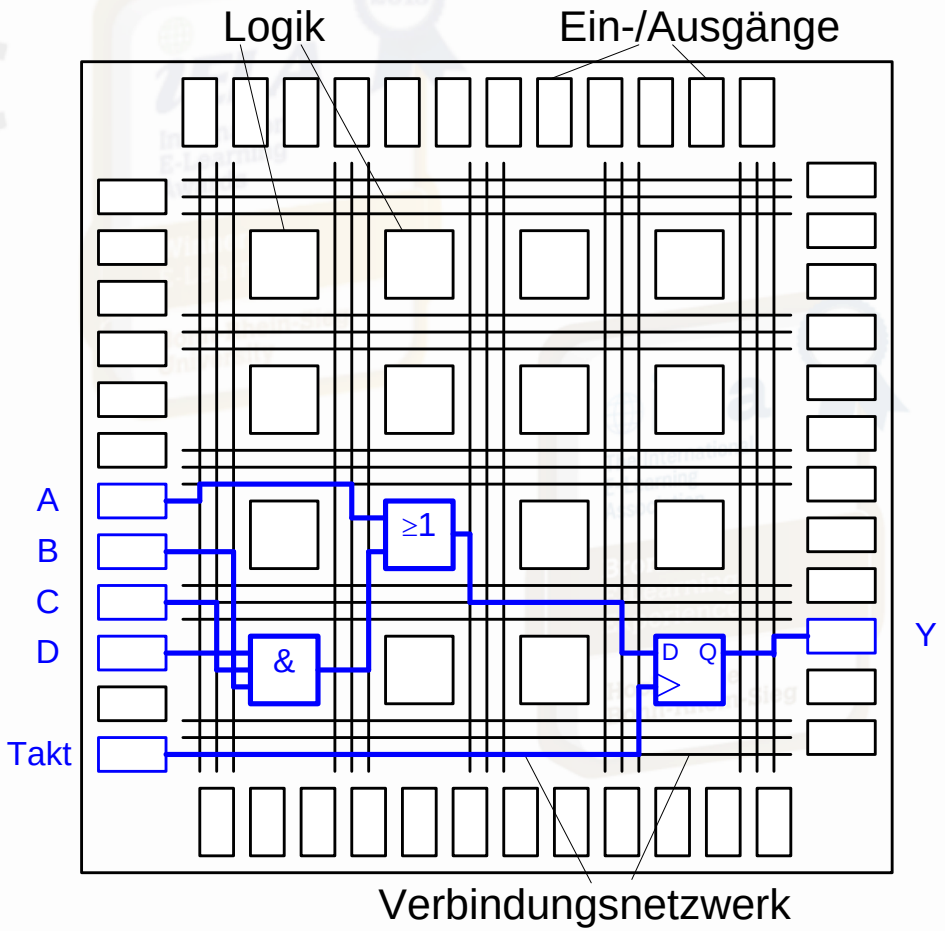
- Waveform viewer to verify the results



Programmable circuit - FPGA

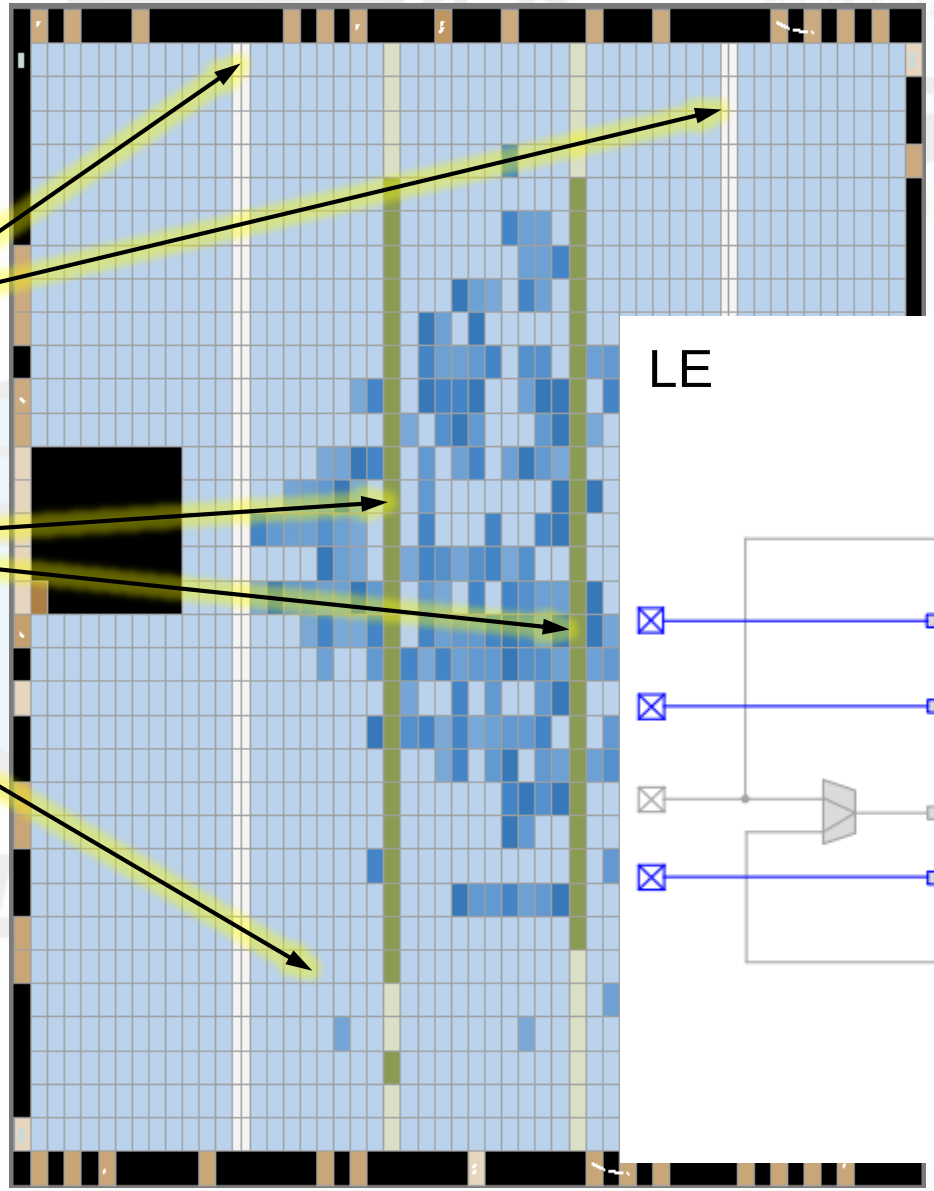
The „Field-Programmable-Gate-Array“ is a microchip containing a programmable circuit

- **field-programmable:** programmable while in use
- **gate-array:** array of gates with
  - some thousands logic functions (AND, OR, EXOR, ...)
  - some thousands memory elements (Registers, Flip-Flops, FFs)
  - connection network
- function and connections are programmable (in **blue**)

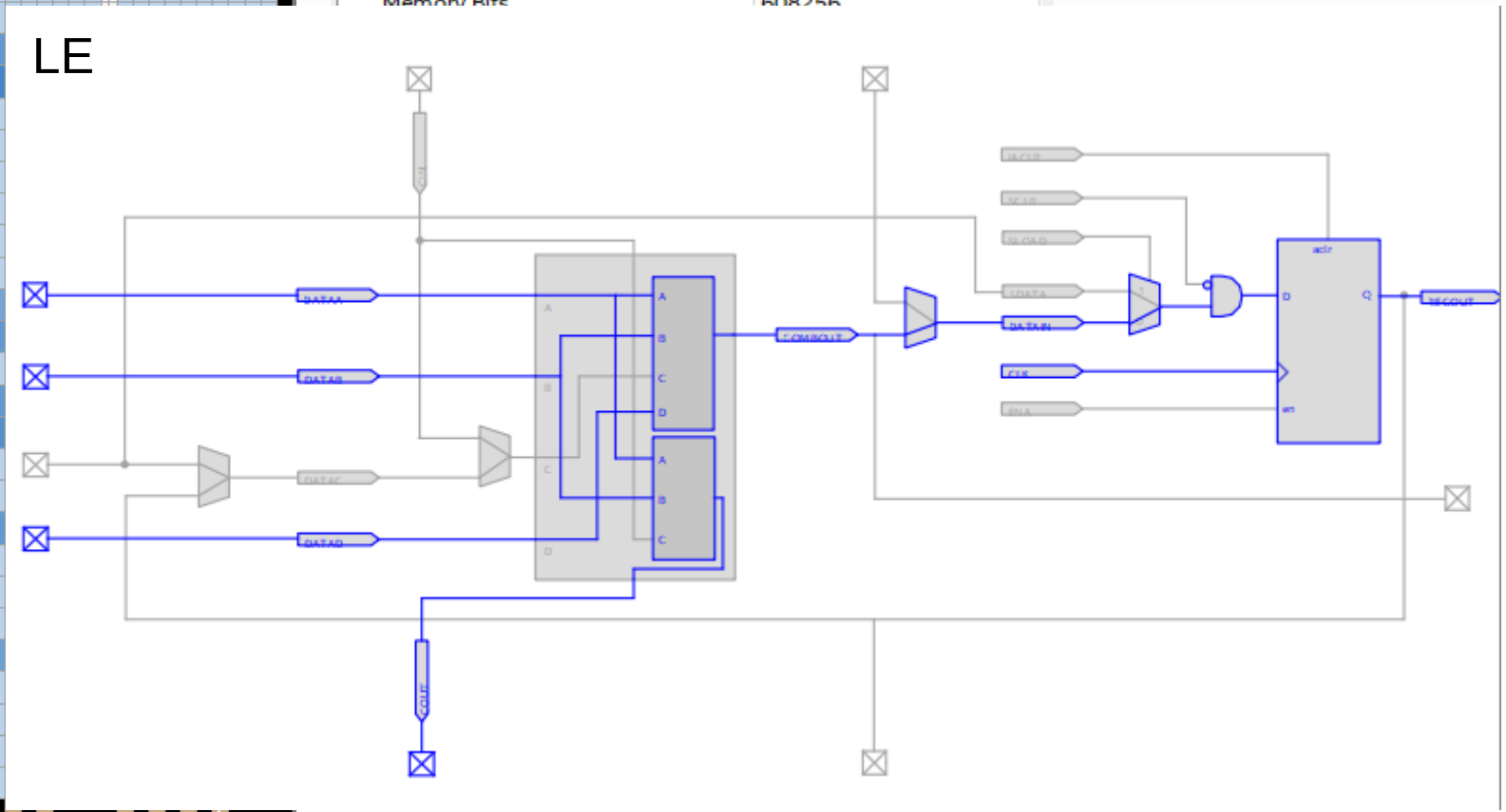


Simplified presentation: Real FPGAs consist of much more logicblocks and I/Os

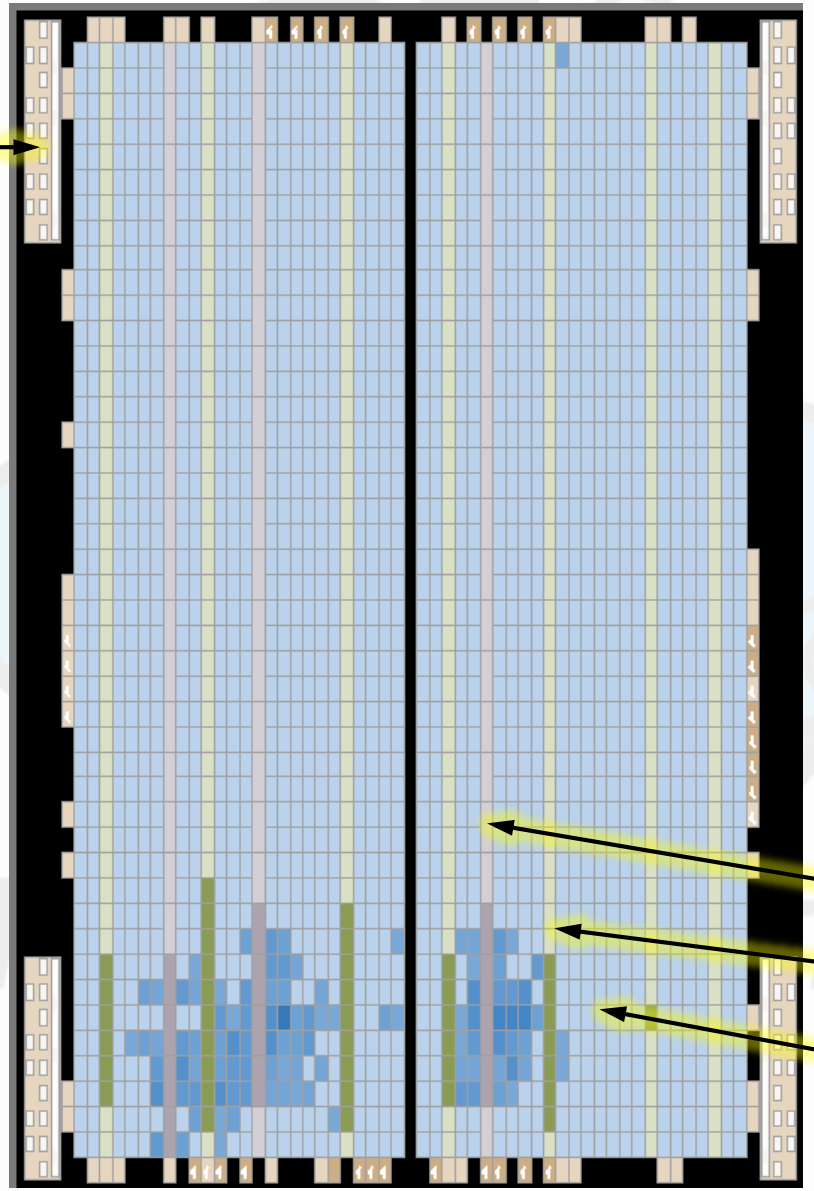
Elements of an FPGA (Cyclone IV)



Device	EP4CE22E22C7*
Family	Cyclone IV E
Device Extents	54 x 35
LEs	22320
Total I/Os	80
GPIOs	80
Memory Bits	608256



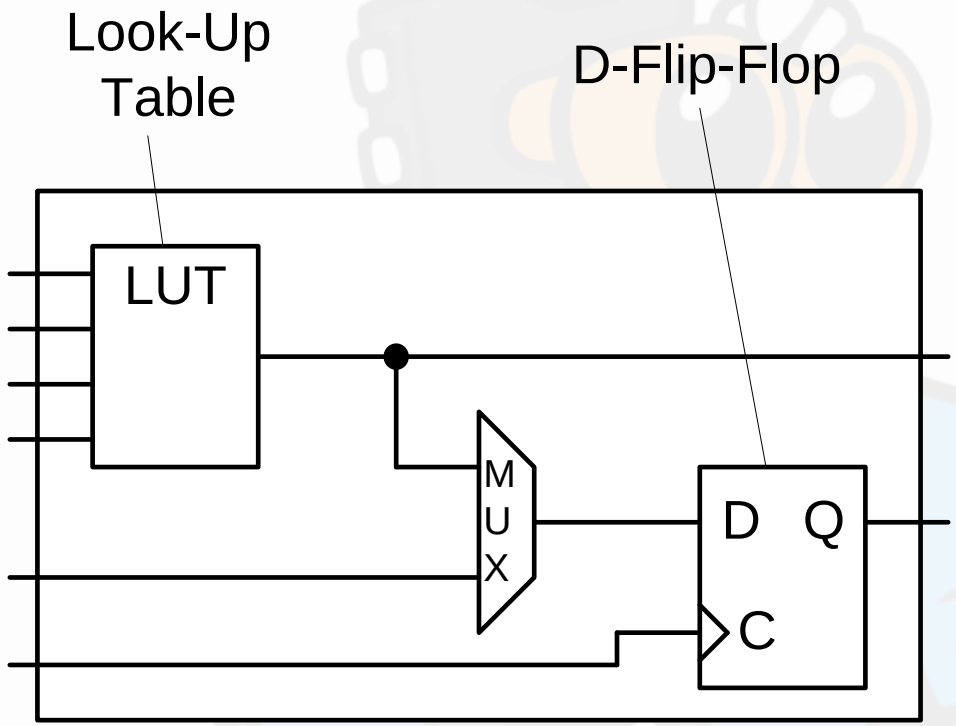
Elements of an FPGA (Cyclone V)



Device	5CEBA2F17C6*
Family	Cyclone V
Device Extents	55 x 46
ALMs	9430
Total I/Os	128
GPIOs	128
GXB Channel PMA	0
GXB Channel PCS	0
PCIe Hard IP Blocks	0
Memory Controllers	0
Memory Bits	1802240
DSP Blocks	25
Fractional PLLs	4
DLLs	4
Global Clocks	16
HPS CPU Cores	0

Flow Status	Successful - Thu Oct 08 18:41:07 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	edupow
Top-level Entity Name	edupow
Family	Cyclone V
Device	5CEBA2F17C6
Timing Models	Final
Logic utilization (in ALMs)	536 / 9,430 (6 %)
Total registers	1030
Total pins	63 / 128 (49 %)
Total virtual pins	0
Total block memory bits	393,216 / 1,802,240 (22 %)
Total DSP Blocks	12 / 25 (48 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 4 (0 %)
Total DLLs	0 / 4 (0 %)

Logikblocks/Logic elements (LE)



- Logic elements of an FPGAs consist of combinatorial logic
  and a register or Flip-Flop (FF)
- Name: LE („Logic Element“)
- The combinatorial logic is realized as a Look-Up-Table (LUT)

- Registers can save the results of an LUT or be used independent of the LUT

Example: Simple module in VHDL

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   -- Uncomment the following library declaration if using
5   -- arithmetic functions with Signed or Unsigned values
6   -- use IEEE.NUMERIC_STD.ALL;
7
8
9
10  entity taster_leds is
11      Port ( SW : in STD_LOGIC_VECTOR (9 downto 0);
12             LEDR : out STD_LOGIC_VECTOR (9 downto 0));
13  end taster_leds;
14
15  architecture Behavioral of taster_leds is
16
17  begin
18
19  LEDR <= SW;
20
21  end Behavioral;
22
```

call packages with definitions of datatypes

This is the call, not the definition of the package!

entity description of module „taster_leds“ with 10 inputs SW0 – SW9 and 10 outputs LEDR0 –LEDR9

concurrent instruction

architecture description of module „taster_leds“
Condition of switches SW0 – SW9 are written on LEDs LEDR0 – LEDR9

Keywords are printed in blue

VHDL describes parallel processing

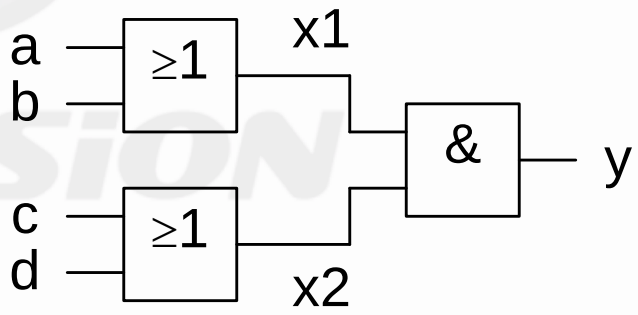
- Multiple concurrent commands are executed in parallel
- The **order** of the commands is **not relevant**!
- The following commands result in the **same behavior**:

```
x1 <= a or b;
x2 <= c or d;
y  <= x1 and x2;
```

```
y  <= x1 and x2;
x1 <= a or b;
x2 <= c or d;
```

```
x1 <= a or b;
y  <= x1 and x2;
x2 <= c or d;
```

- All three descriptions result into the same circuit:



Overview of VHDL commands

```
process
begin
   wait until rising_edge(clk);

   -- input FFs for control
    reset <= not reset_n;
    enable <= enable_in;
    -- input FFs for video signal
    vs_0  <= vs_in;
    hs_0  <= hs_in;
    de_0  <= de_in;
    r_0   <= to_integer(unsigned(r_in));
    g_0   <= to_integer(unsigned(g_in));
    b_0   <= to_integer(unsigned(b_in));
end process;
```

```
if (reset = '1') then
  wr_address := 0;
  rd_address := 1;
elsif (write_en = '1') then
  wr_address := rd_address;
  if (rd_address = 1279) then
    rd_address := 0;
  else
    rd_address := rd_address + 1;
  end if;
end if;
```

```
process
begin
  wait until rising_edge(clk);
  h_tap(0) <= v_out;
  for i in 0 to 7 loop
    h_tap(i+1) <= h_tap(i);
  end loop;
end process;
```

```
process(bin)
begin
    case bin is
        when "0000" => seg7_invert <= "0111111"; -- 0
        when "0001" => seg7_invert <= "0000110"; -- 1
        when "0010" => seg7_invert <= "1011011"; -- 2
        when "0011" => seg7_invert <= "1001111"; -- 3
        when "0100" => seg7_invert <= "1100110"; -- 4
        when "0101" => seg7_invert <= "1101101"; -- 5
        when "0110" => seg7_invert <= "1111101"; -- 6
        when "0111" => seg7_invert <= "0000111"; -- 7
        when "1000" => seg7_invert <= "1111111"; -- 8
        when "1001" => seg7_invert <= "1101111"; -- 9
        when "1010" => seg7_invert <= "1110111"; -- A
        when "1011" => seg7_invert <= "1111100"; -- B
        when "1100" => seg7_invert <= "0111001"; -- C
        when "1101" => seg7_invert <= "1011110"; -- D
        when "1110" => seg7_invert <= "1111001"; -- E
        when "1111" => seg7_invert <= "1110001"; -- F
        when others => seg7_invert <= "1000000"; -- '-' for rest
    end case;
end process;
```

„Golden Rules of VHDL“

VHDL is a hardware-design language

- When you are working with VHDL, you are not programming, you are “designing hardware". Your VHDL code should reflect this fact.
- If your VHDL code appears too similar to code of a higher-level computer language, it is probably bad VHDL code.

Have a general concept of what your hardware should look like

- Although VHDL is vastly powerful, if you do not understand basic digital constructs, you will probably be unable to generate efficient digital circuits.
- If you are not able to roughly envision the digital circuit you are trying to model in terms of basic digital circuits, you will probably misuse VHDL.
- VHDL is cool, but it is not as magical as it initially appears to be.

Aus: B. Mealy, F. Tappero, “Free Range VHDL,” http://freerangefactory.org

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Cyclone IV (FPGA experiments)

Finish

Binary for FPGA:

Durchsuchen... Keine Datei ausgewählt.

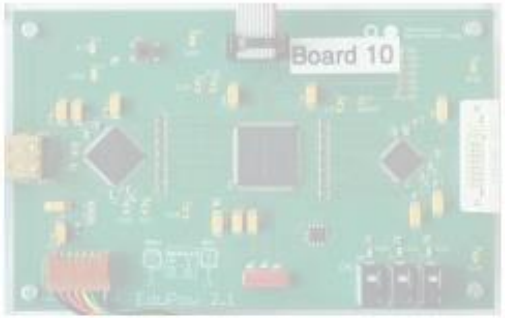
Upload FPGA binary and start experiment

IMAGE PROCESSING WITH FPGAS

96.6 mA

(Core Supply Voltage is 1.2 V)

<< < Select Input Image > >>

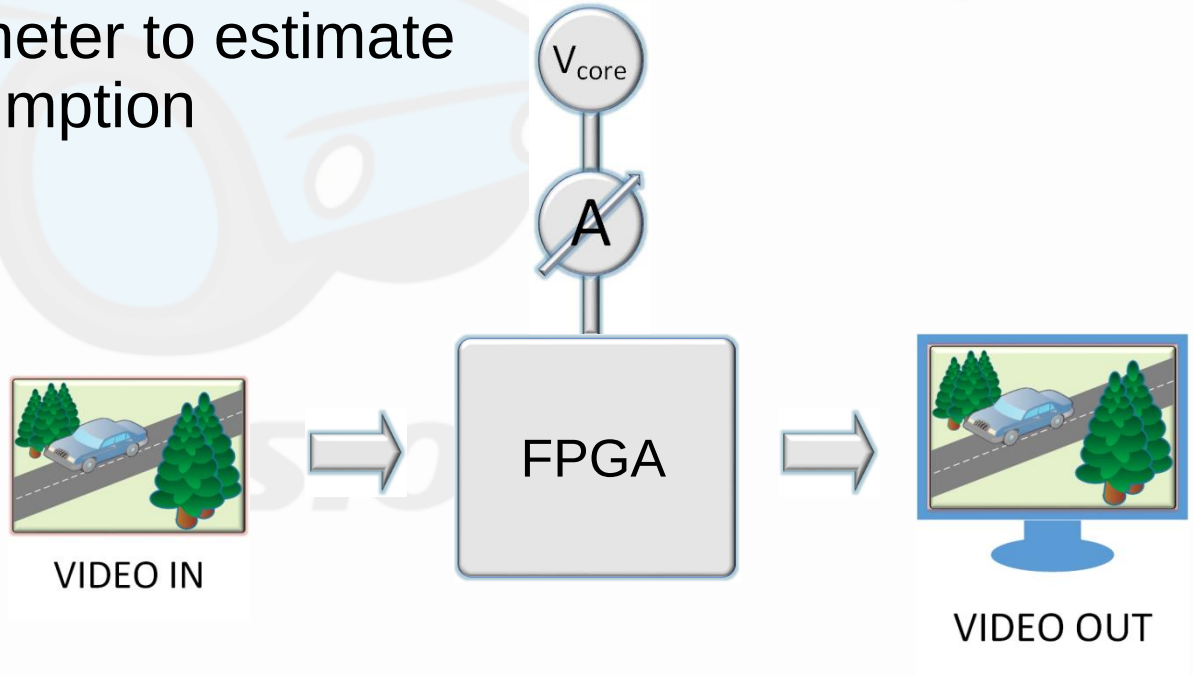


Introduction

- A typical setup for image processing could consist of
  - a CMOS device

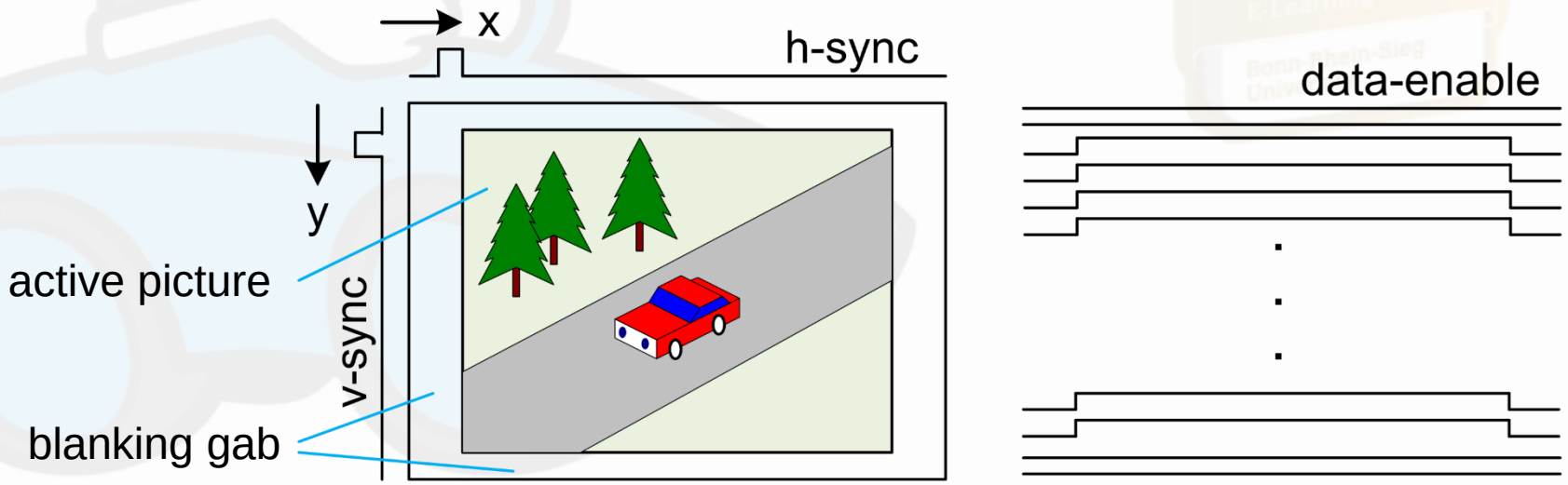
    e.g., an FPGA
  - video in- and output
  - and possible additional components

    ⇨ e.g., an ammeter to estimate power consumption



Video Signal

- Output of video signals row by row
- There is an inactive arial horizontally und vertically
  - Control signals v-sync, h-sync, data-enable

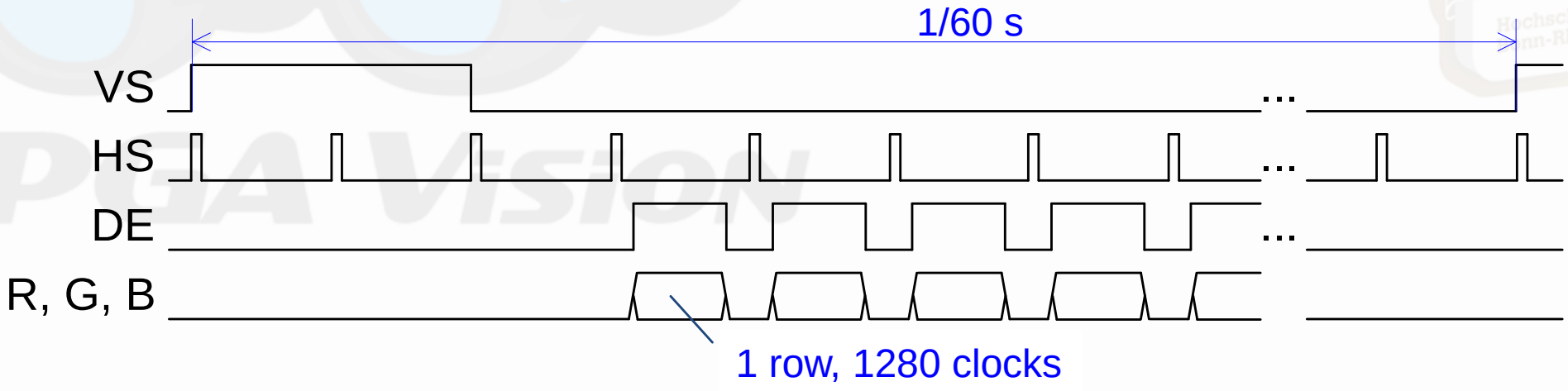


- Example: HDTV graphic format 720p
  - 1280*720 Pixel, 60 Hz image refresh rate
  - Vertical blanking gab 30 rows, total of 750 rows
  - Horizontal blanking gab 370 pixel, total of 1650 pixel

Video Signal II

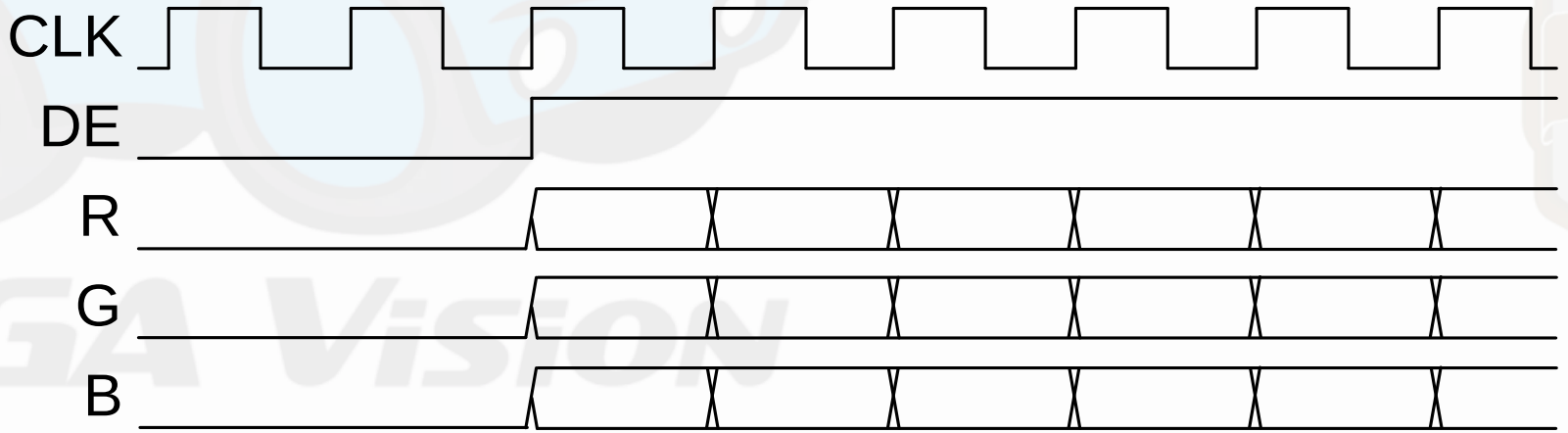
- 60 pictures/s * 750 rows * 1650 pixel result in clock frequency of 74,25 MHz
- Signalization of active pixel by three control signals
  - VS, v-sync: Start of picture
  - HS, h-sync: Start of row
  - DE, data-enable: Active pixel

- Color information R, G, B with 8 bit each



Video Signal III

- R, G, B every clock
  - R, G, B values as binary number between 0 and 255, respectively 8 bit per color
  - 0 = no color
  - 255 = maximum color



Structure of Templates

- There are VHDL-templates available
- **Inputs** are
  - CLK – clock of 74,25 MHz
  - RESET_N – Low-active reset, inverted in VHDL to the high-active RESET
  - ENABLE_IN – Three switches
  - VS_IN, HS_IN, DE_IN – control signals of input image
  - R_IN, G_IN, B_IN – Color information input image, 8 bit
- **Outputs** are
  - VS_OUT, HS_OUT, DE_OUT – control signals of output image
  - R_OUT, G_OUT, B_OUT – Color information output image, 8 bit
  - LED – Three LEDs, not implemented in remote lab

VHDL Entity

```
entity edupow is
    port ( clk       : in  std_logic;
           reset_n   : in  std_logic;
           enable_in : in  std_logic_vector(2 downto 0);
           vs_in     : in  std_logic;
           hs_in     : in  std_logic;
           de_in     : in  std_logic;
           r_in      : in  std_logic_vector(7 downto 0);
           g_in      : in  std_logic_vector(7 downto 0);
           b_in      : in  std_logic_vector(7 downto 0);
           vs_out    : out std_logic;
           hs_out    : out std_logic;
           de_out    : out std_logic;
           r_out     : out std_logic_vector(7 downto 0);
           g_out     : out std_logic_vector(7 downto 0);
           b_out     : out std_logic_vector(7 downto 0);
           clk_o     : out std_logic;
           led       : out std_logic_vector(2 downto 0));
end edupow;
```

control input

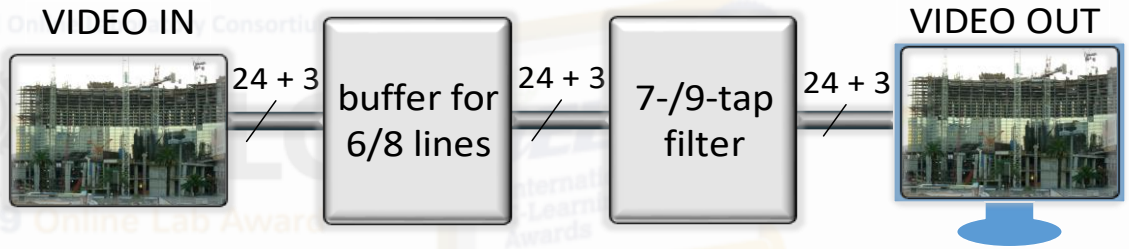
video input

video output

control output

Basics FIR Filter

- Block diagramm:



- Filter equation (discrete convolution):

$$P(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{n} p(x+i-a, y+j-a) \cdot h_{ij}$$

p(x,y) : Pixel at (x,y)
n : No taps
x : horizontal position
y : vertical position

i : horizontal matrix index
j : vertical matrix index
a : n/2 +1
h_{ij} : Element of convolution matrix at position (i,j)

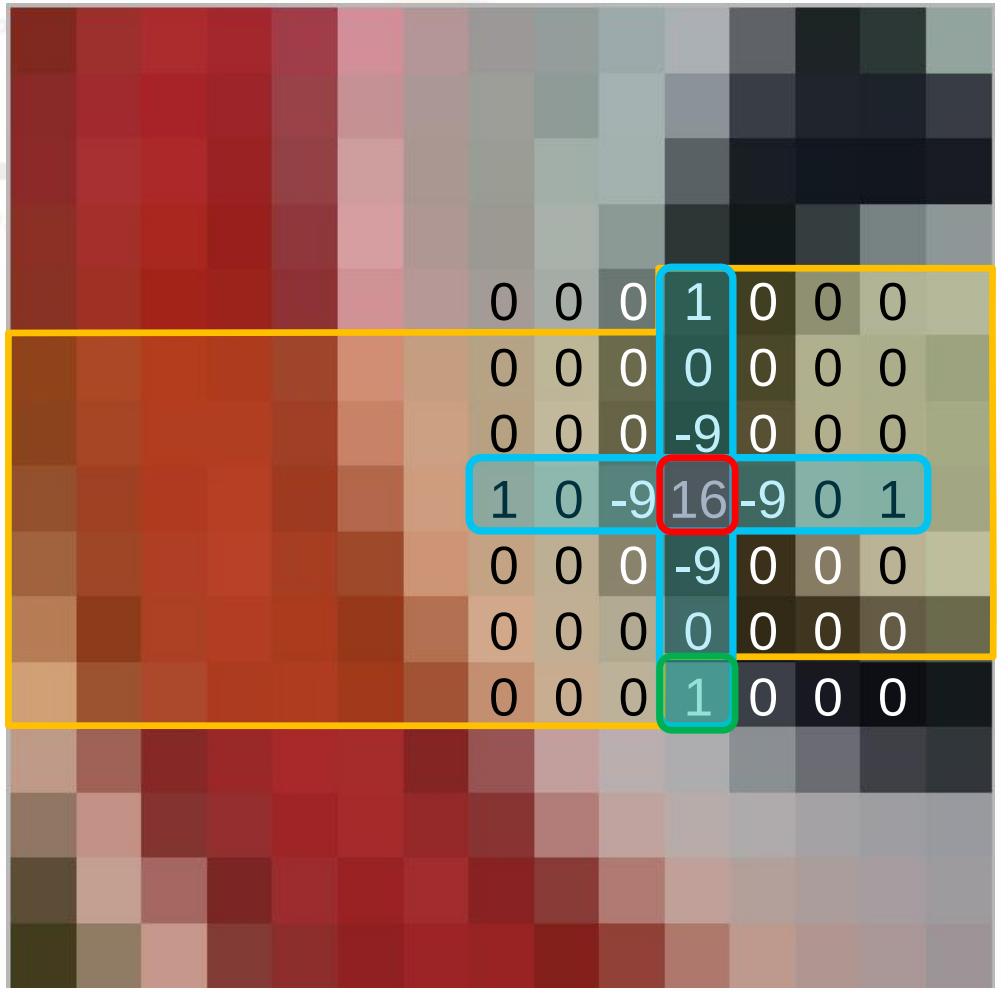
- Filter matrix (7-tap filter):

$$h_{7-tap} = \frac{1}{32}\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -9 & 0 & 0 & 0 \\ 1 & 0 & -9 & 16 & -9 & 0 & 1 \\ 0 & 0 & 0 & -9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

7-Tap FIR Filter

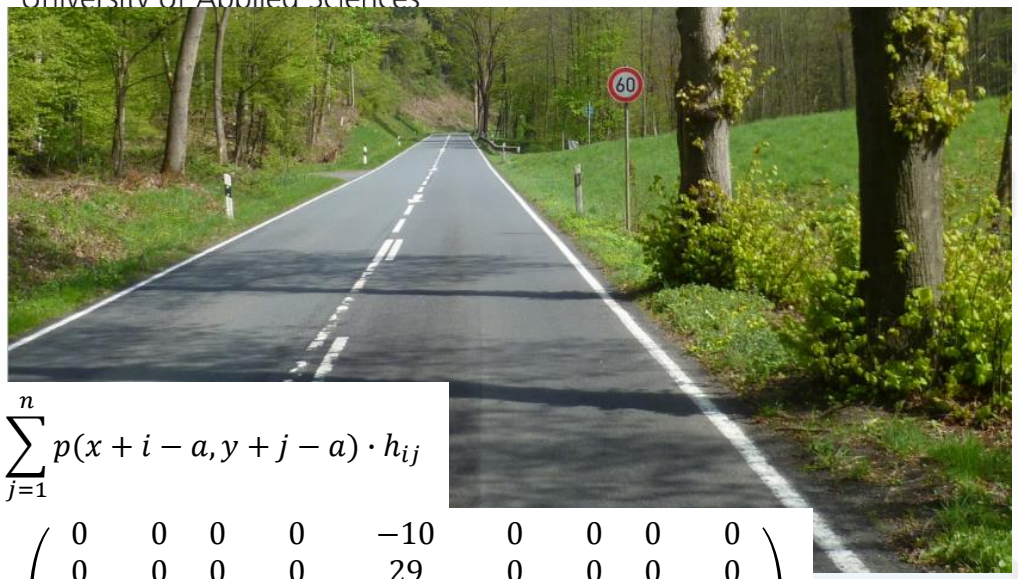
$$P(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{n} p(x+i-a, y+j-a)\cdot h_{ij}$$

$$h_{7-tap} = \frac{1}{32}\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -9 & 0 & 0 & 0 \\ 1 & 0 & -9 & 16 & -9 & 0 & 1 \\ 0 & 0 & 0 & -9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$



- current pixel p(x,y+3) - 3 rows under filtered pixel p(x,y)
- buffer 6 rows for 7-tap filter
- Apply filter matrix on p(x,y)
- Apply effective matrix coefficients

9-Tap FIR Filter



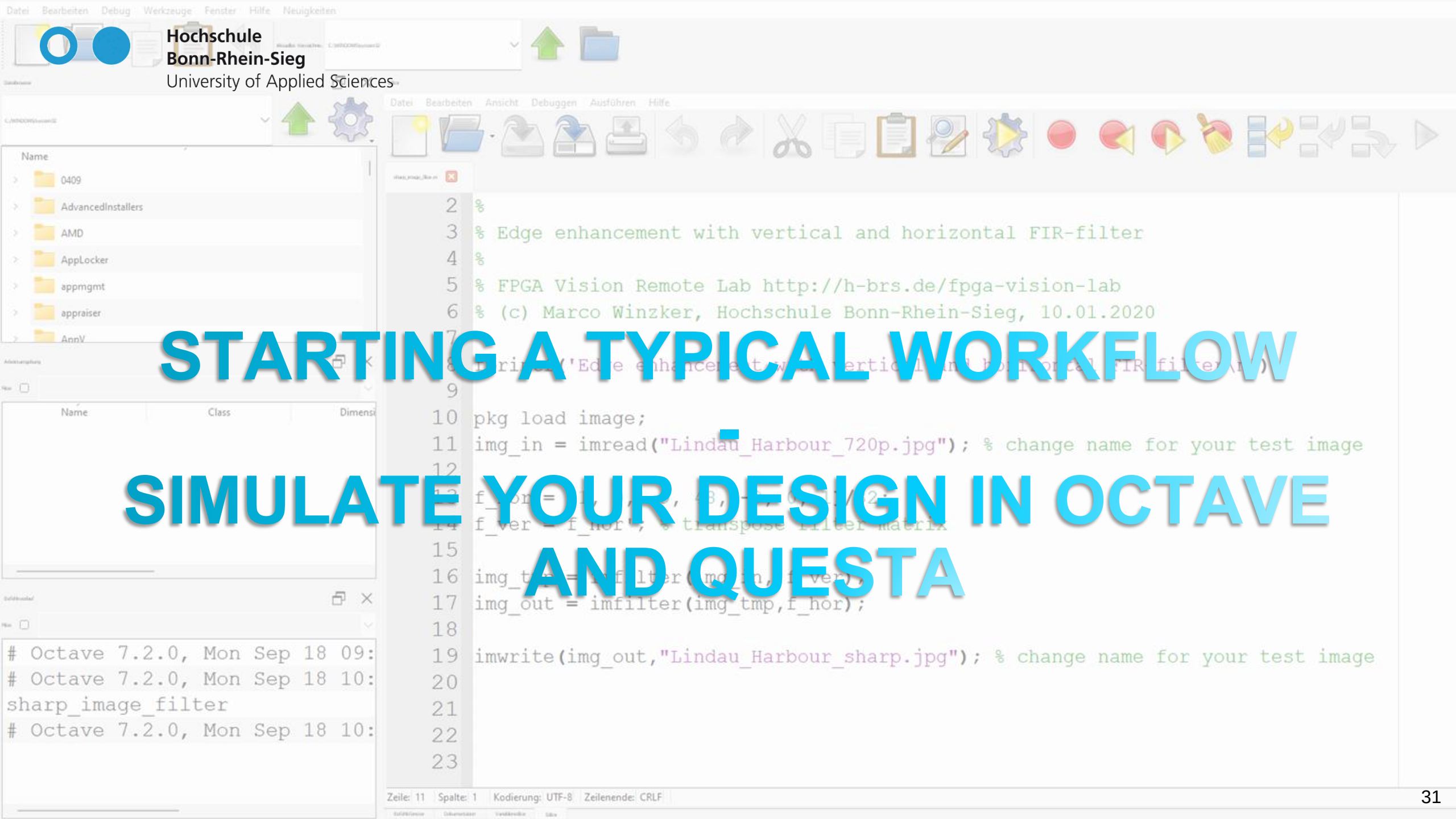
$$P(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{n} p(x+i-a, y+j-a)\cdot h_{ij}$$

$$h_{9-tap} = \frac{1}{1024}\begin{pmatrix} 0 & 0 & 0 & 0 & -10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 29 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -285 & 0 & 0 & 0 & 0 \\ -10 & 29 & 9 & -285 & 514 & -285 & 9 & 29 & -10 \\ 0 & 0 & 0 & 0 & -285 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 29 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -10 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- current pixel p(x,y+4) - 4 rows under filtered pixel p(x,y)
- buffer 8 rows for 9-tap filter
- Apply filter matrix on p(x,y)
- Apply effective matrix coefficients

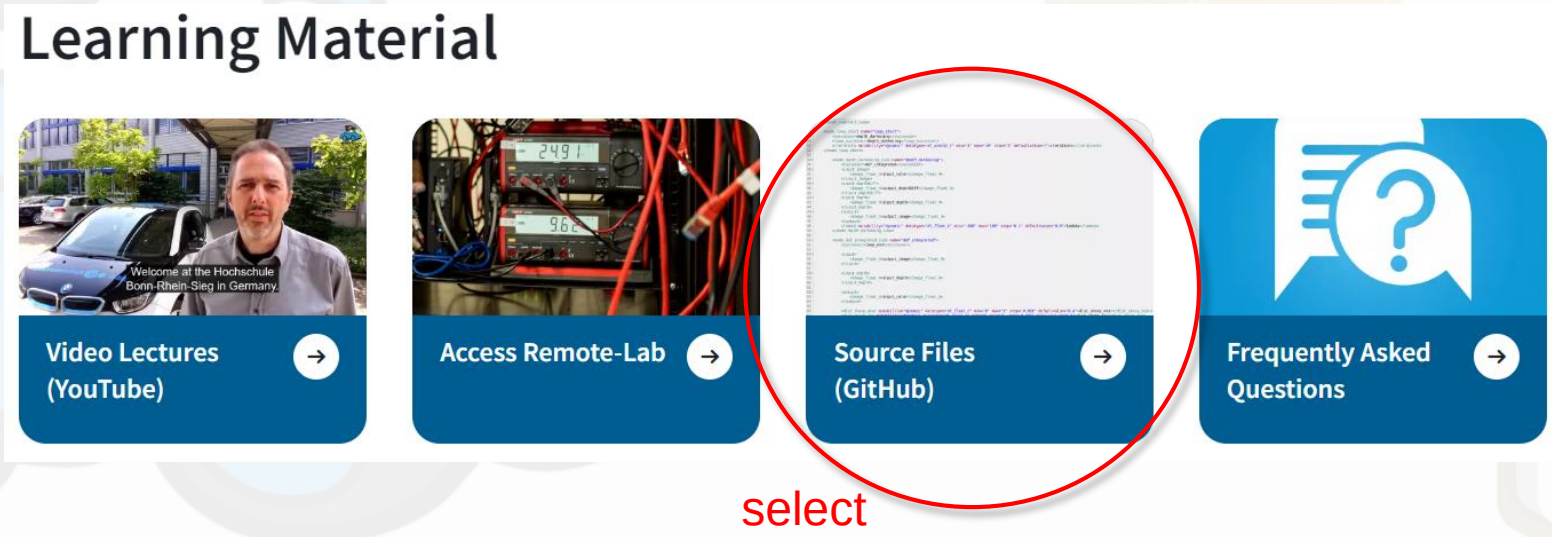
Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

STARTING A TYPICAL WORKFLOW - SIMULATE YOUR DESIGN IN OCTAVE AND QUESTA



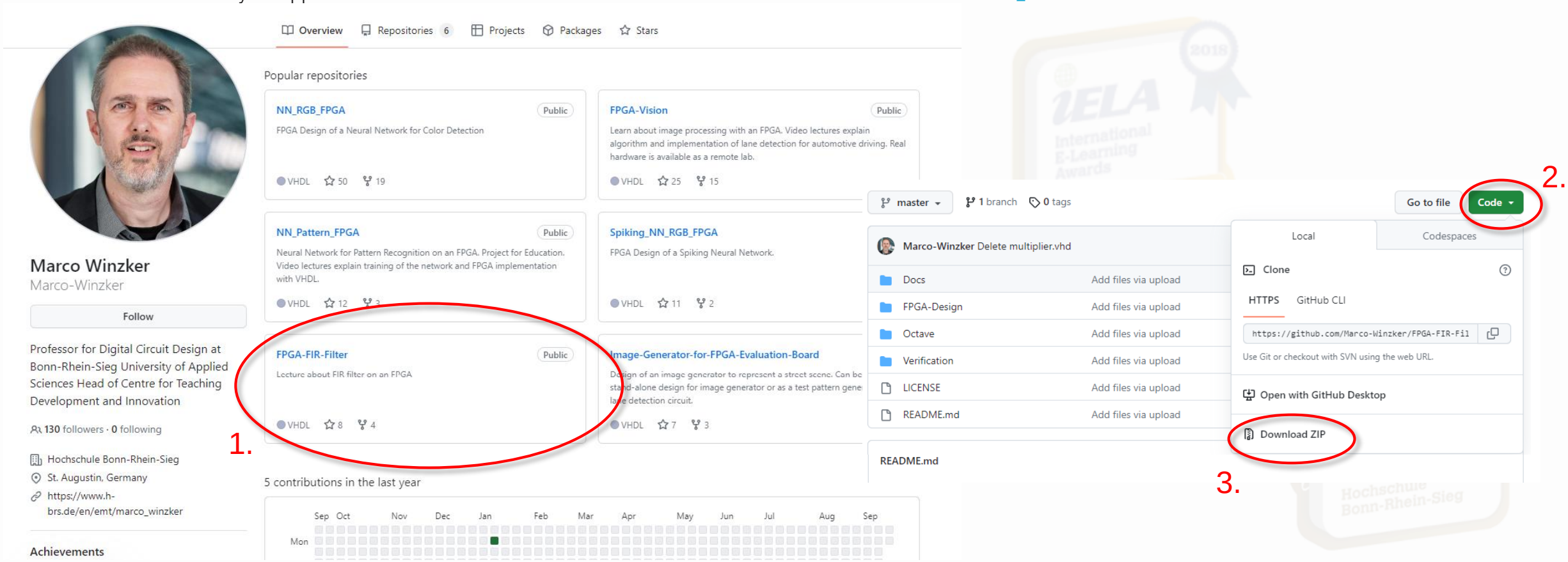
First step: get the templates

- Visit https://www.h-brs.de/fpga-vision-lab
- scroll down to

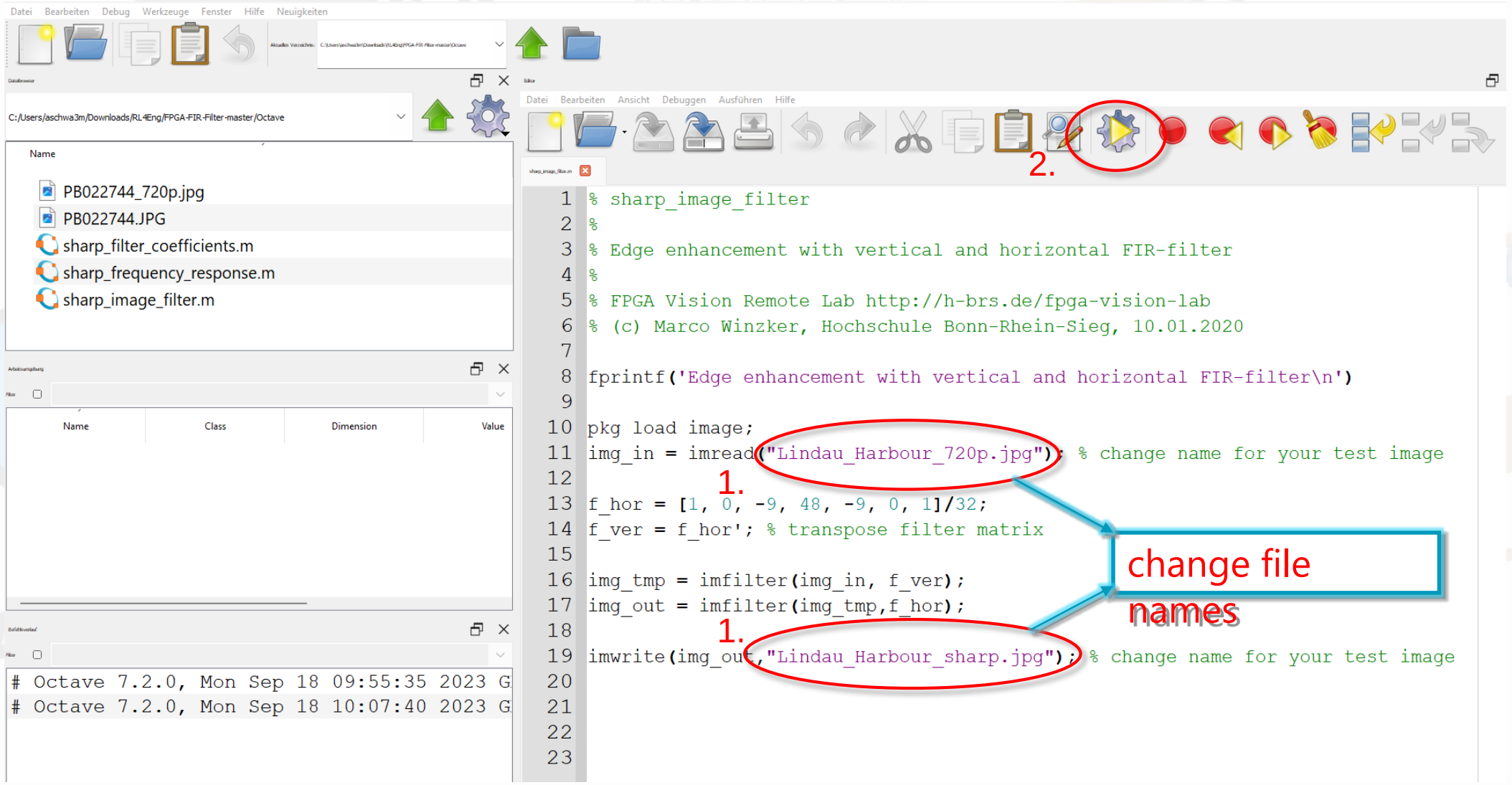


Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Get the Templates from Github



Simulate in Octave

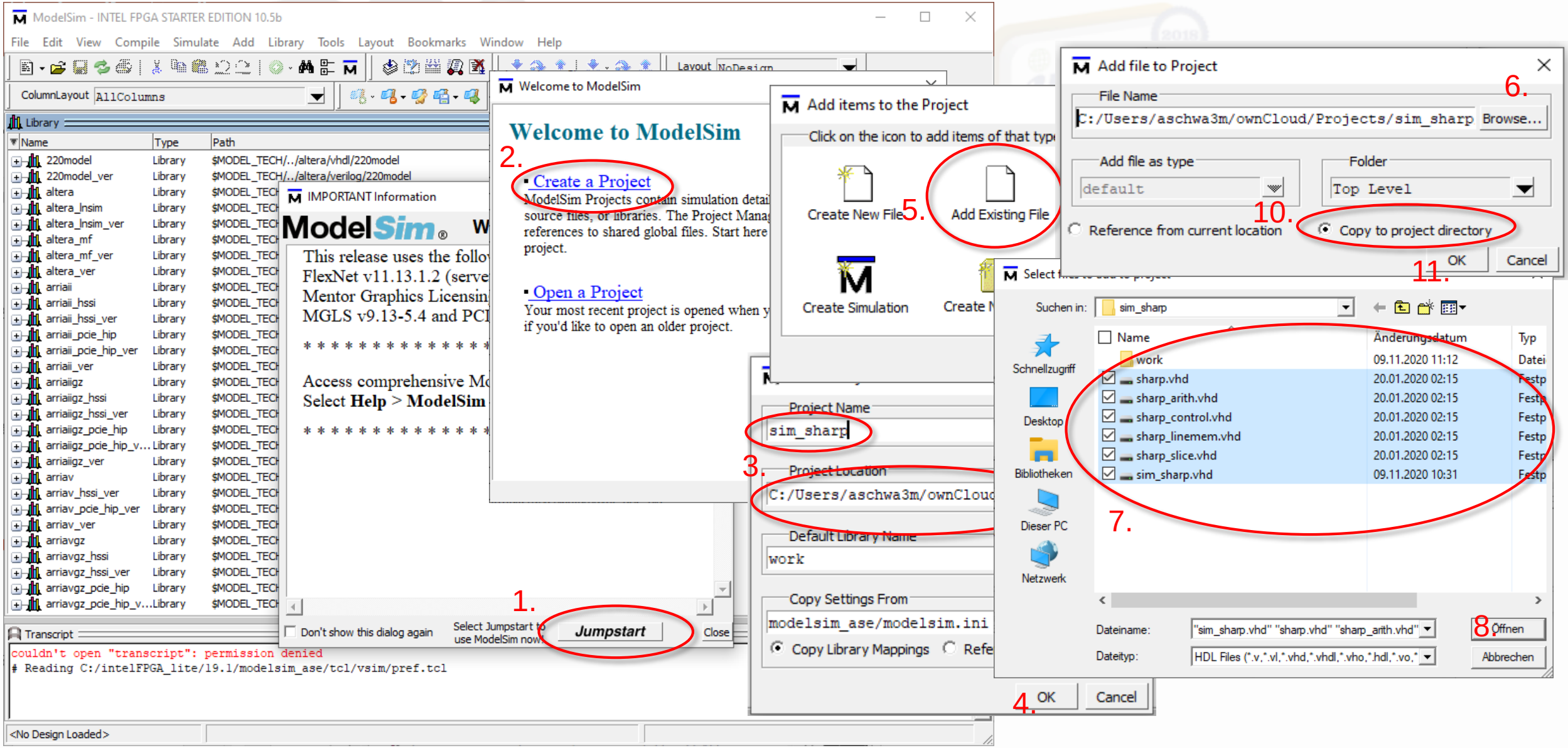


Prepare files for simulation

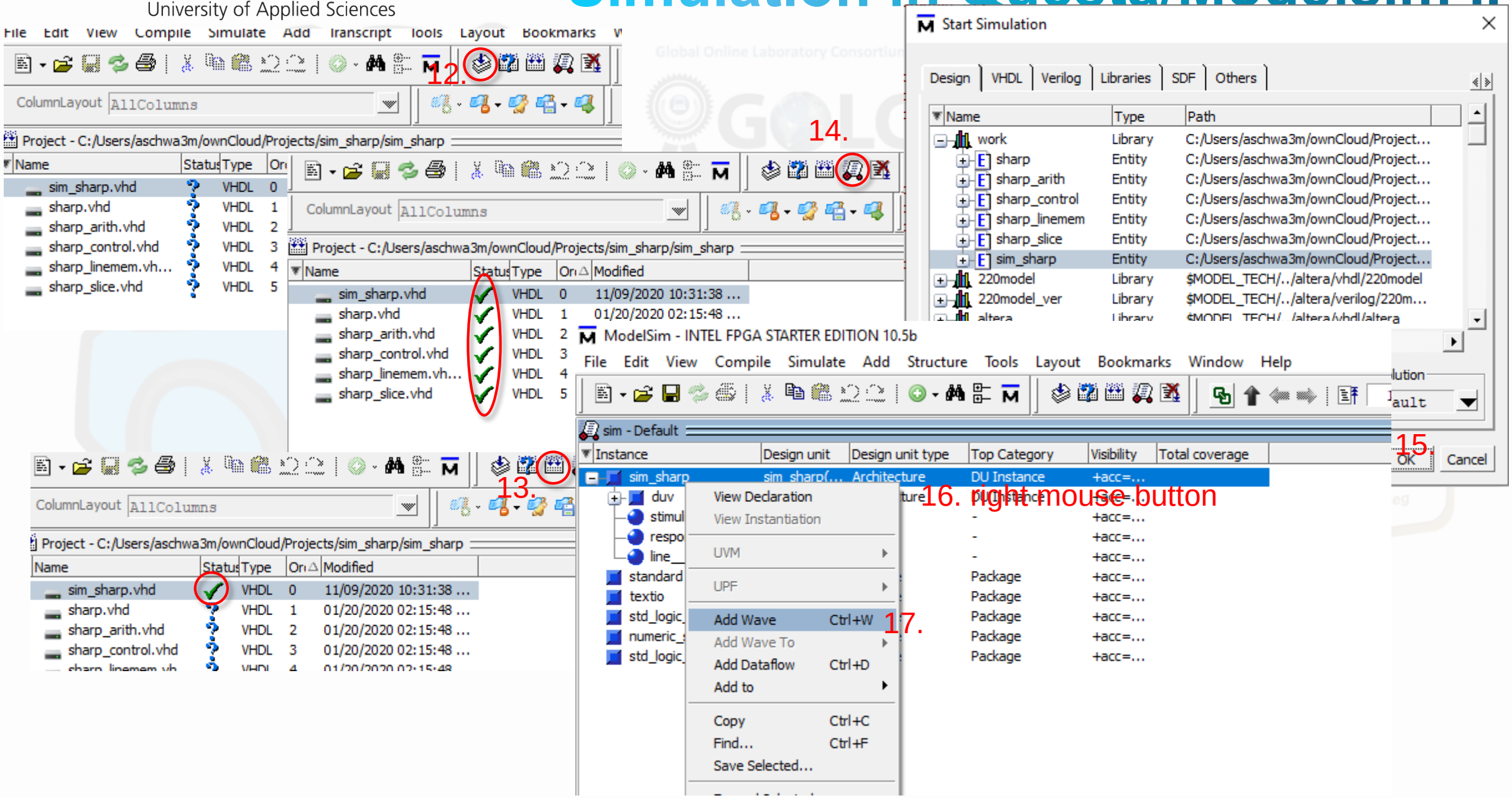
- copy converted 720p image into the verification folder
- open „sharp_generate_testbench_images.m“ in Octave (ensure write_ascii_ppm.m is in the same folder)
- adjust rows 12, 21 und 22 to own file names, run Skript and wait a bit until files are created

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

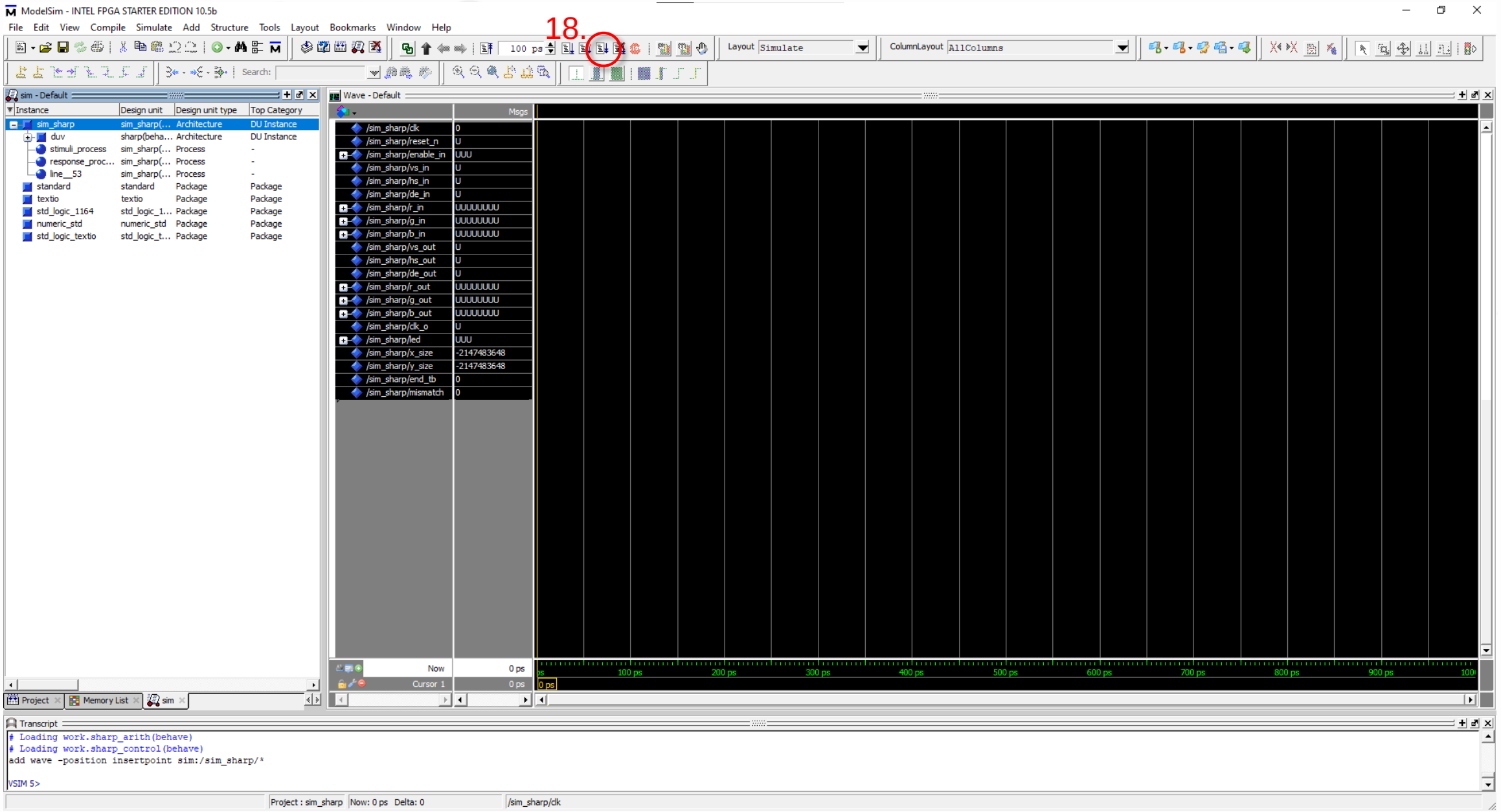
Simulation in Questa/Modelsim



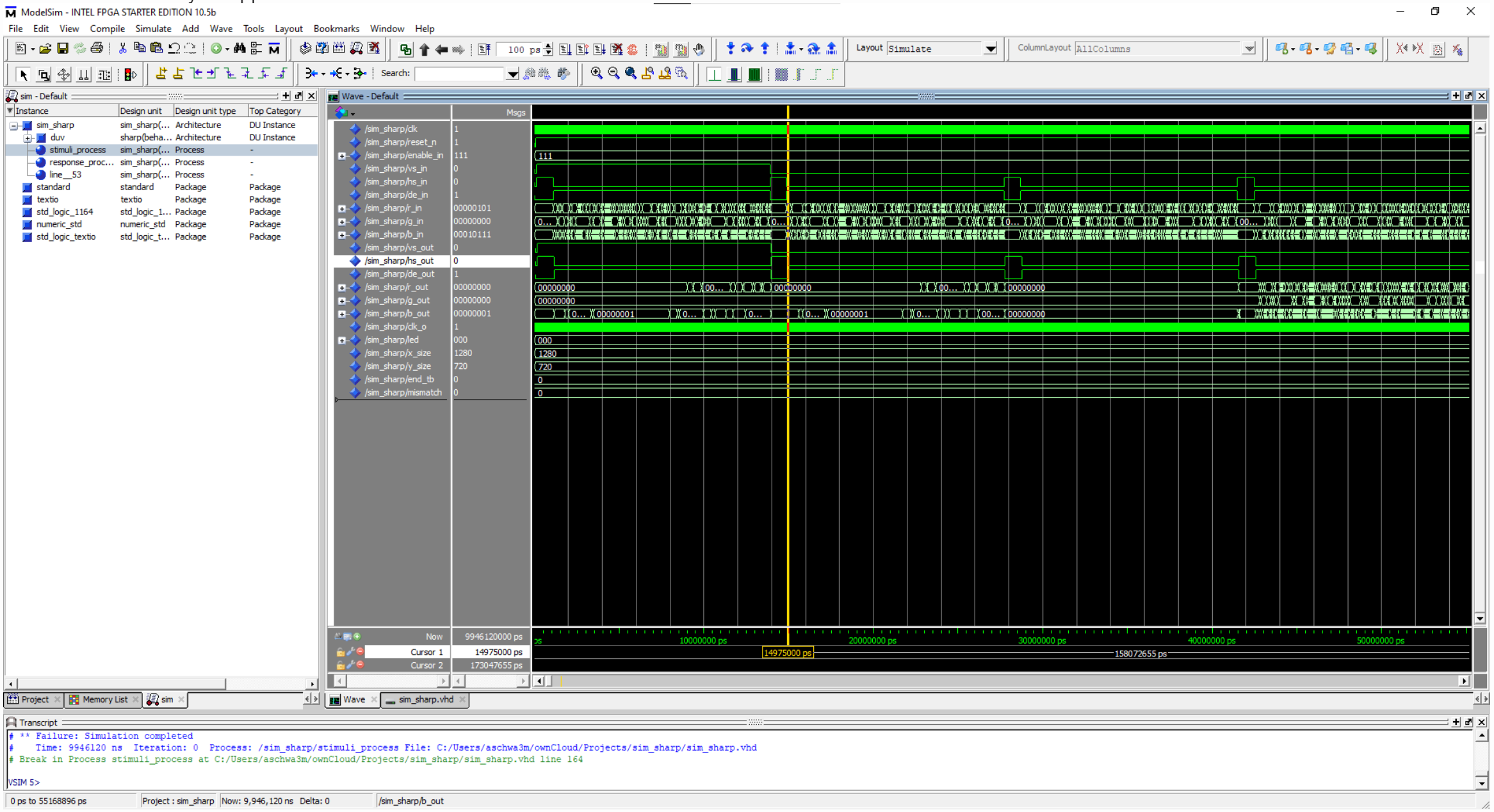
Simulation in Questa/Modelsim II



Simulation in Questa/Modelsim III



Simulation in Questa/Modelsim IV



Simulate with self checking testbench

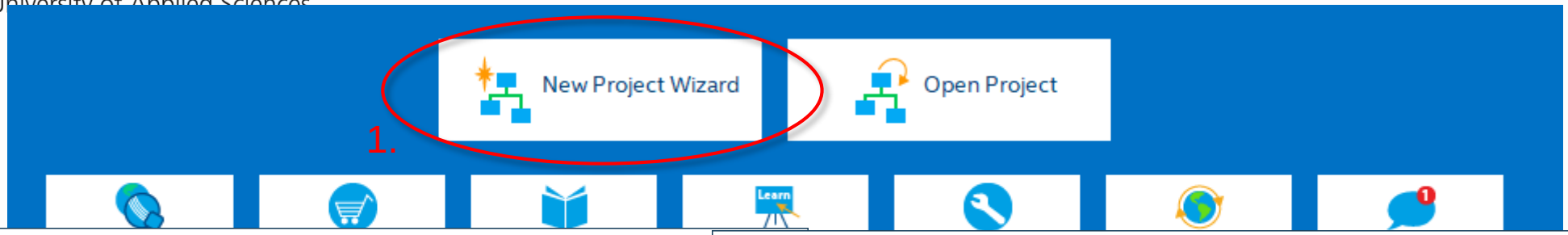
- create project in Modelsim, compile and simulate it (see slides 35ff)
- watch messages:

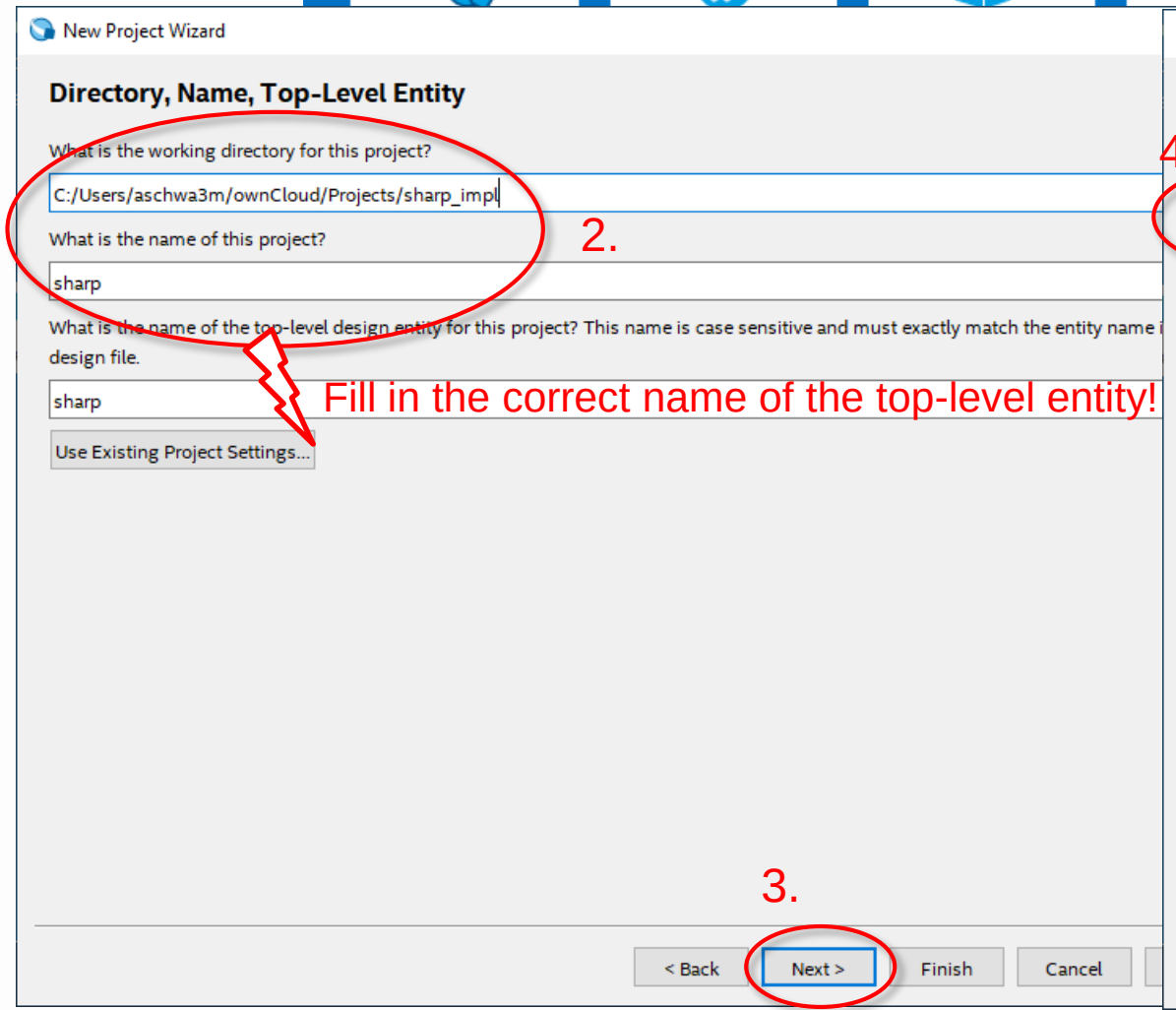
```
VSIM 3> run -all
# ** Failure: Simulation completed, EVERYTHING OK
#    Time: 9946120 ns  Iteration: 0  Process: /sim_sharp/stimuli_process File: C:/Users/aschwa3m/ownCloud/Projects/sim_sct_sharp/sim_sharp_self-checking.vhd
# Break in Process stimuli_process at C:/Users/aschwa3m/ownCloud/Projects/sim_sct_sharp/sim_sharp_self-checking.vhd line 167

VSIM 4>
```

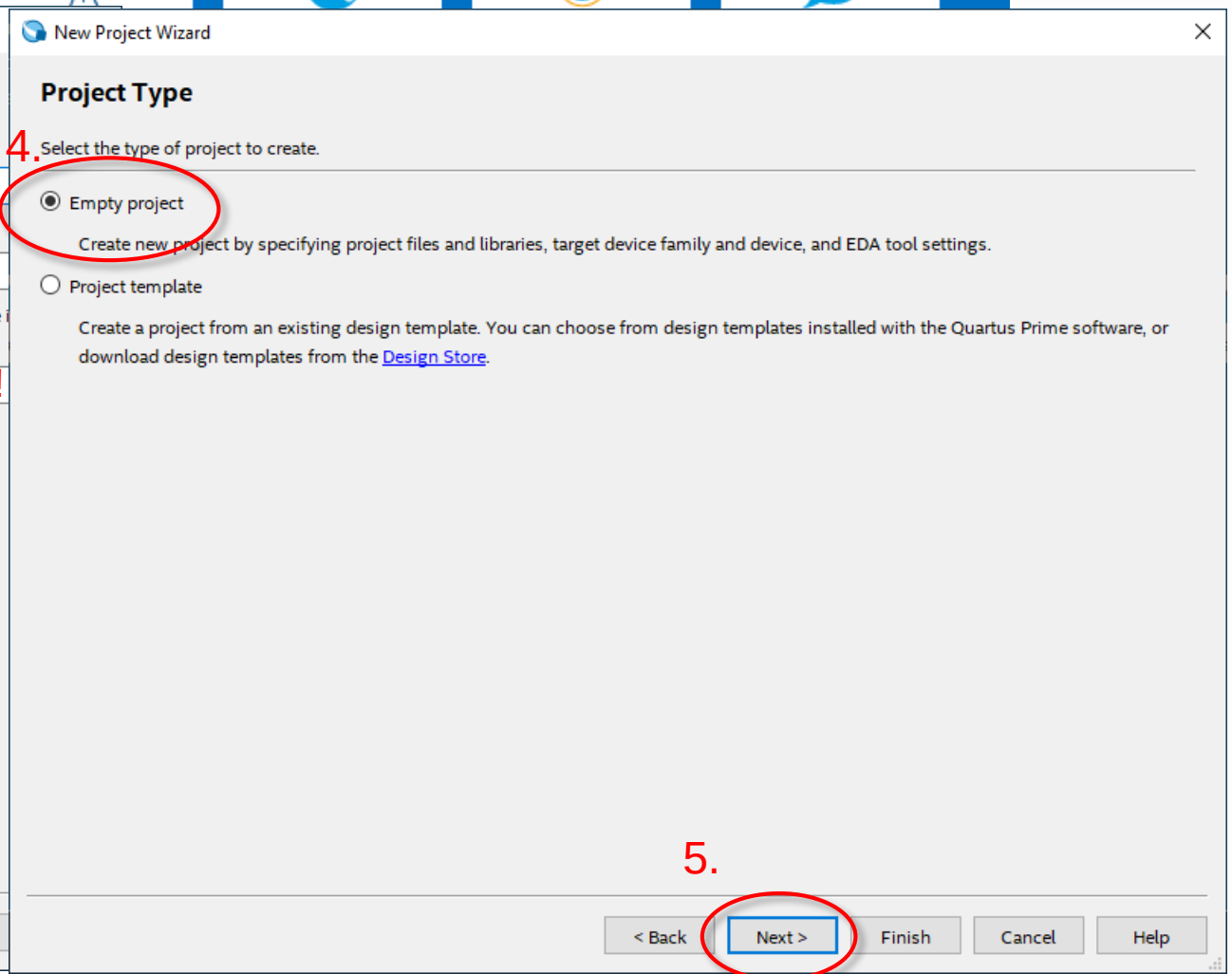
Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

SETTING UP A PROJECT IN QUARTUS AND TEST THE DESIGN WITH THE FPGA VISION REMOTE LAB

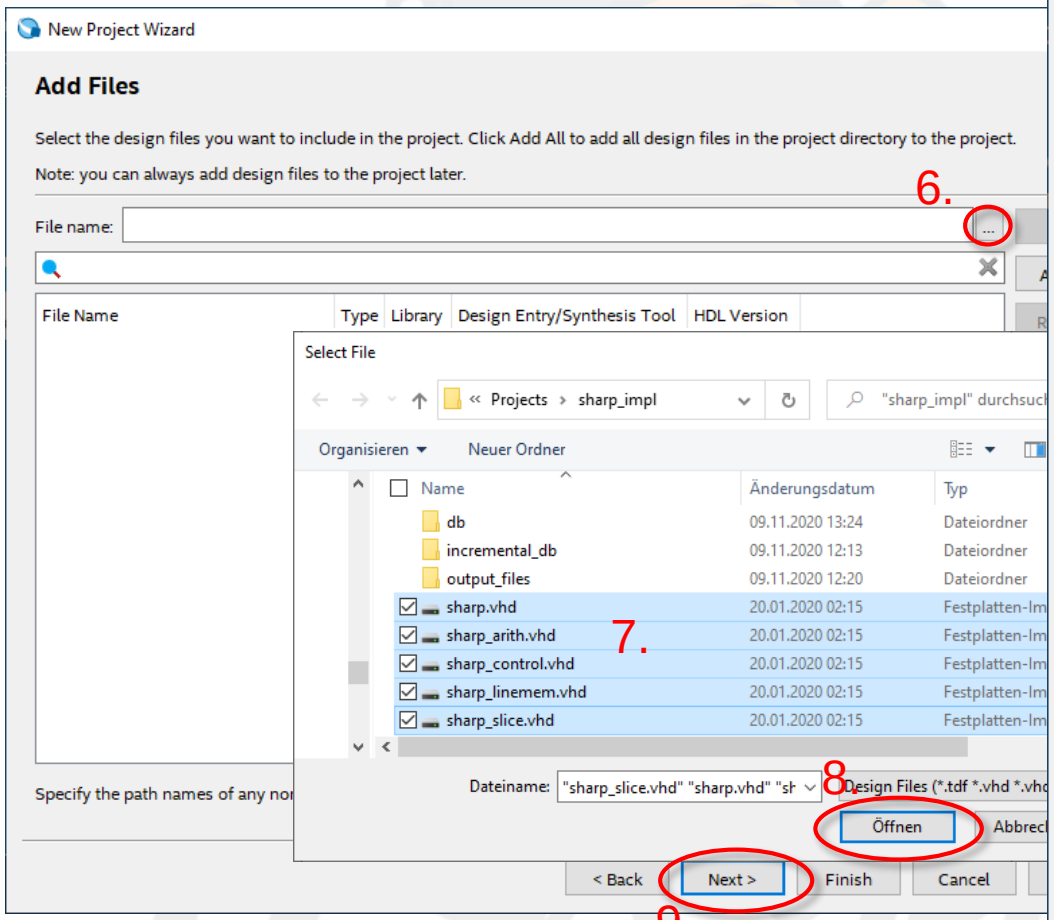
Starting with the Project Wizzard I

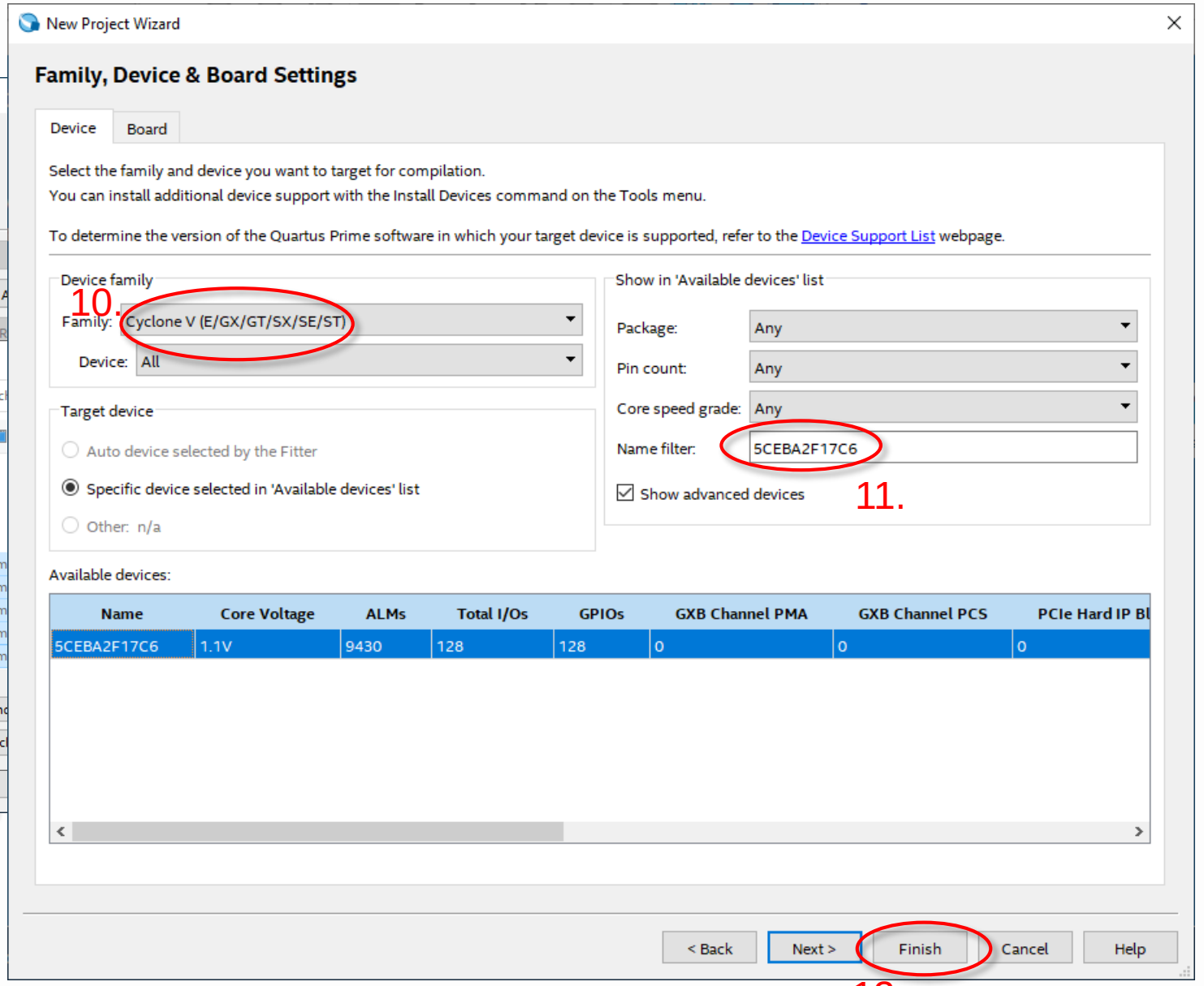






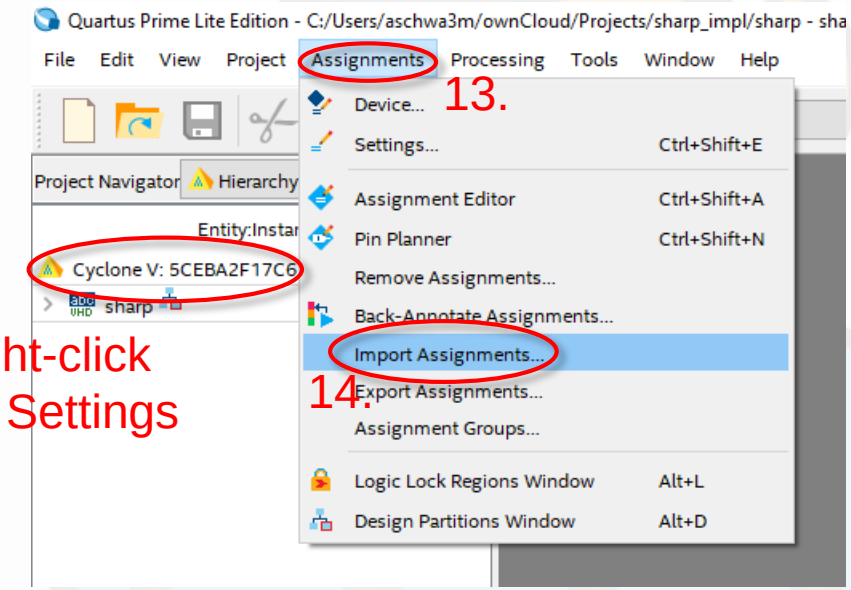
Starting with the Project Wizzard II





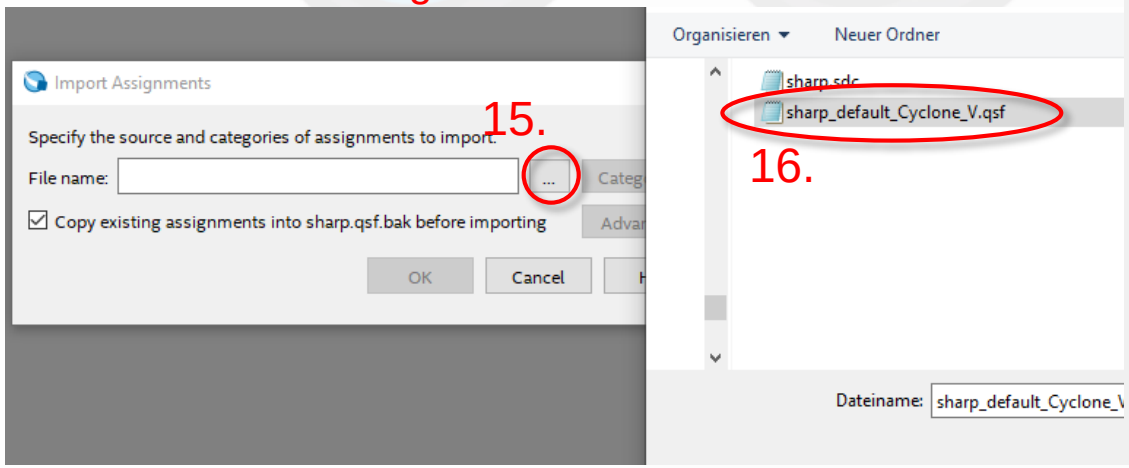
Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

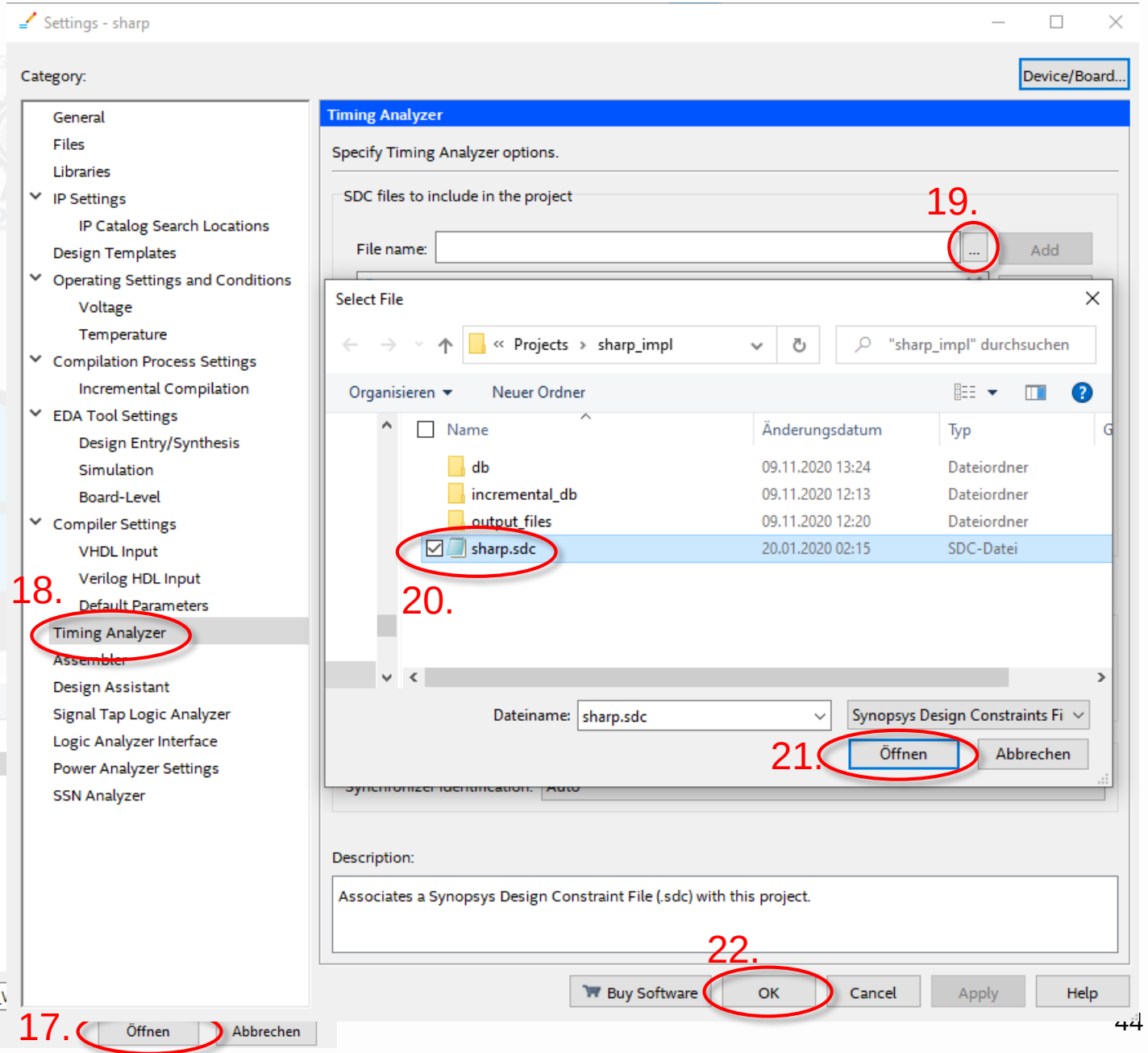
Import Assignments and Adding Constraints



right-click
-> Settings

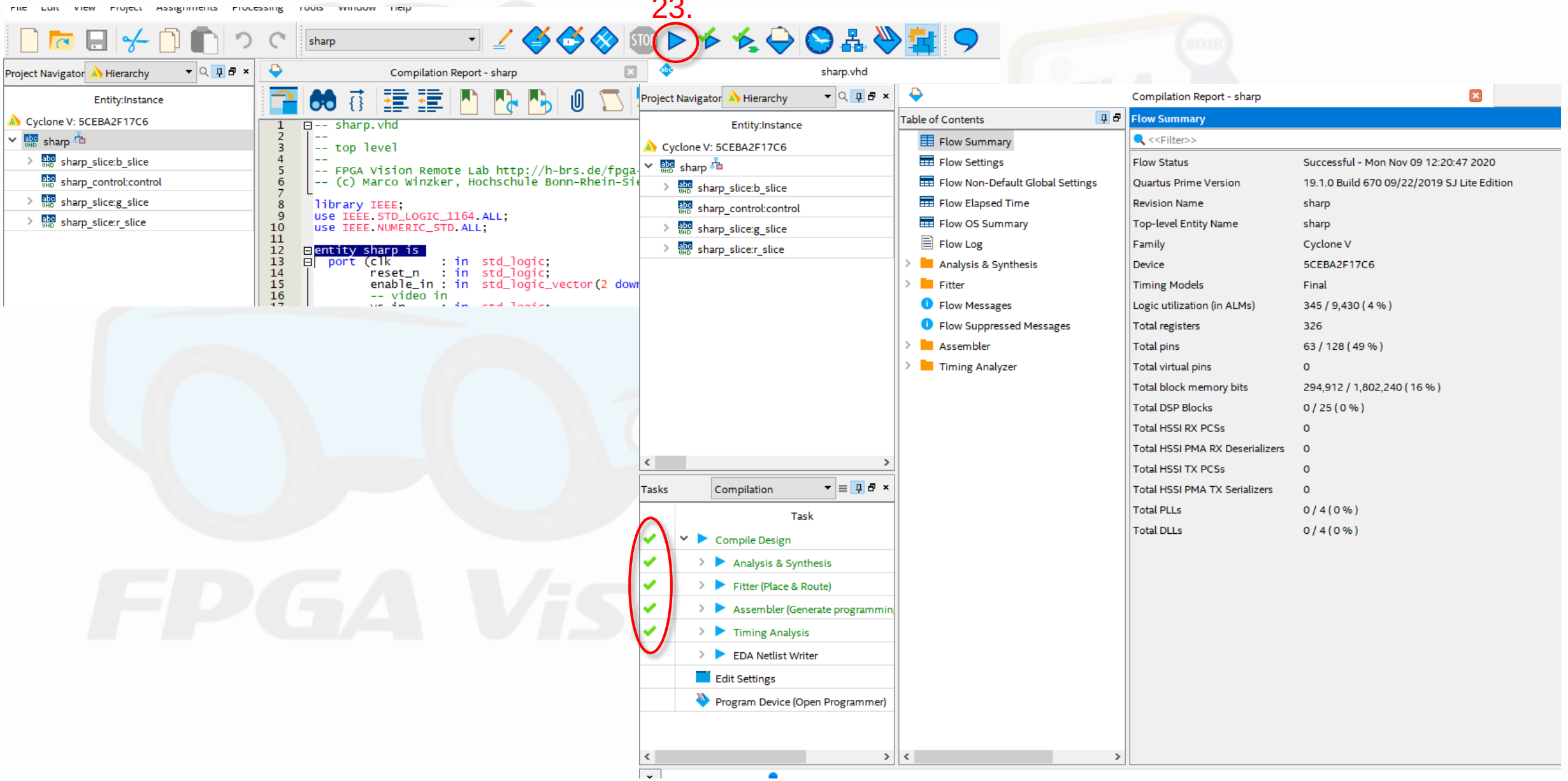
Without the assignments the software doesn't know the location of the pins and you won't have video signals!



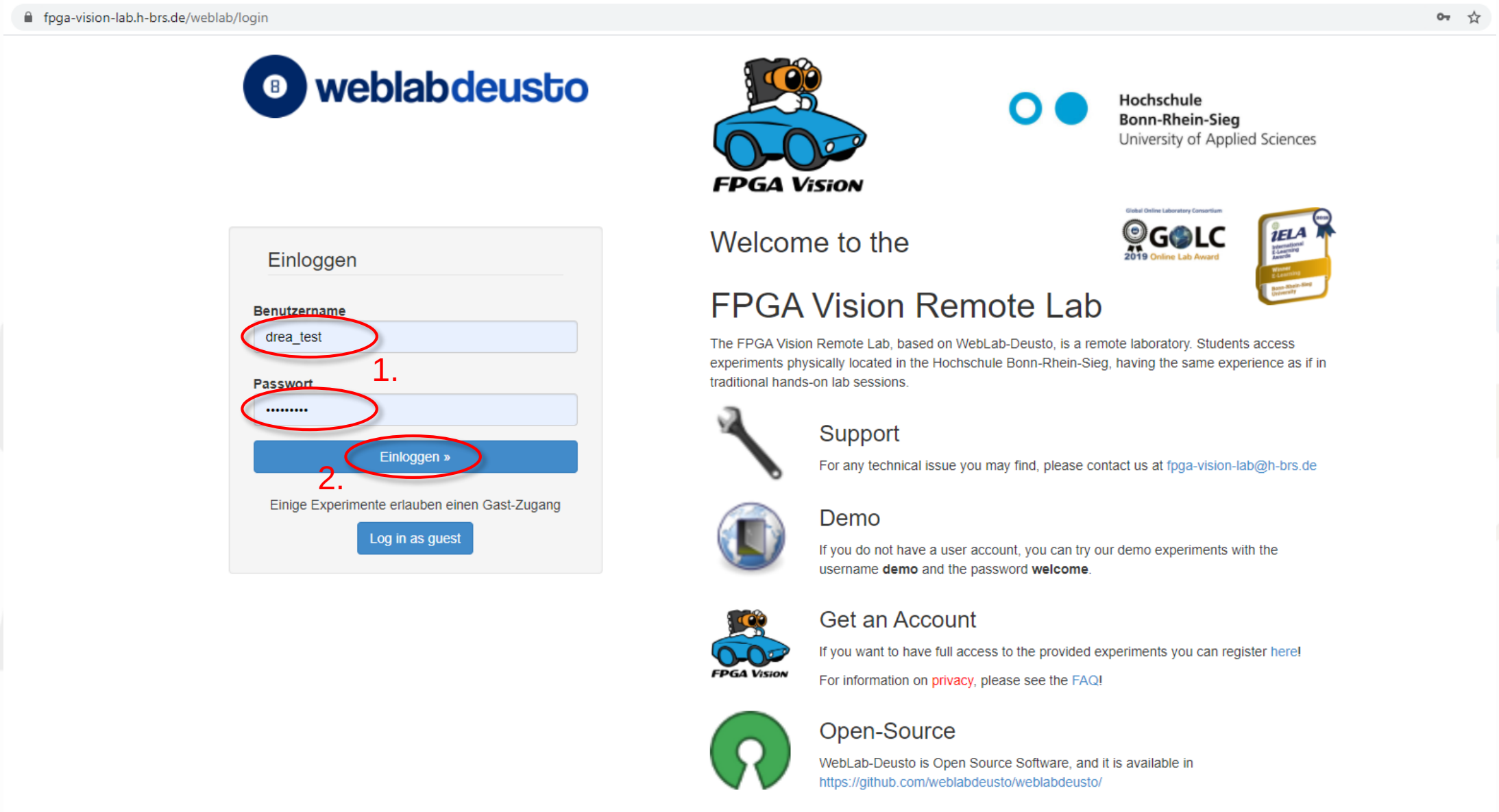


Design Flow

23.

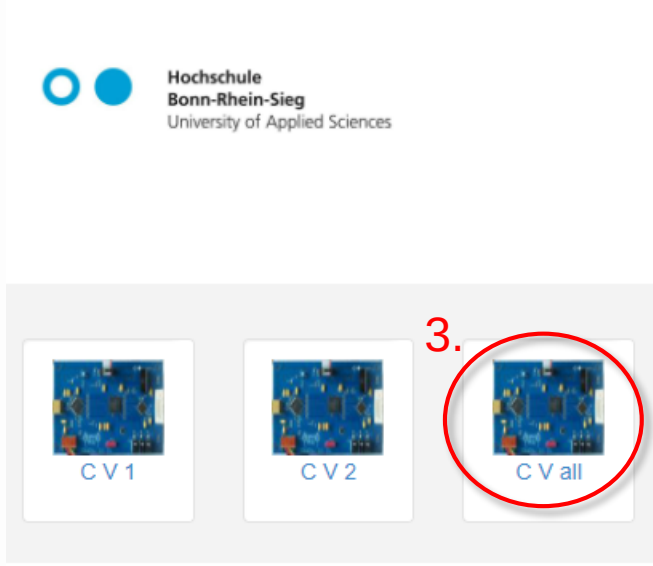


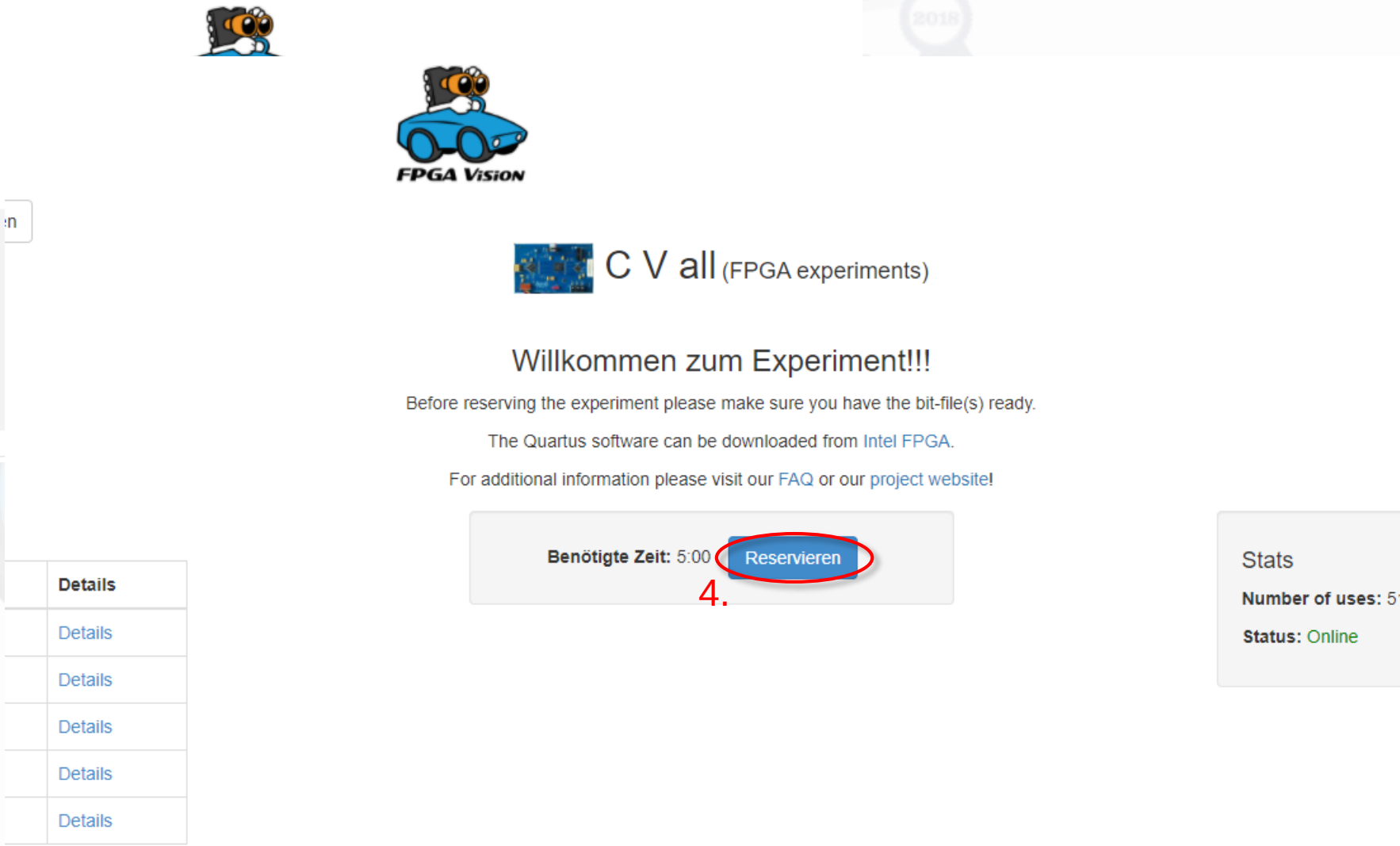
Login into the Remote Lab



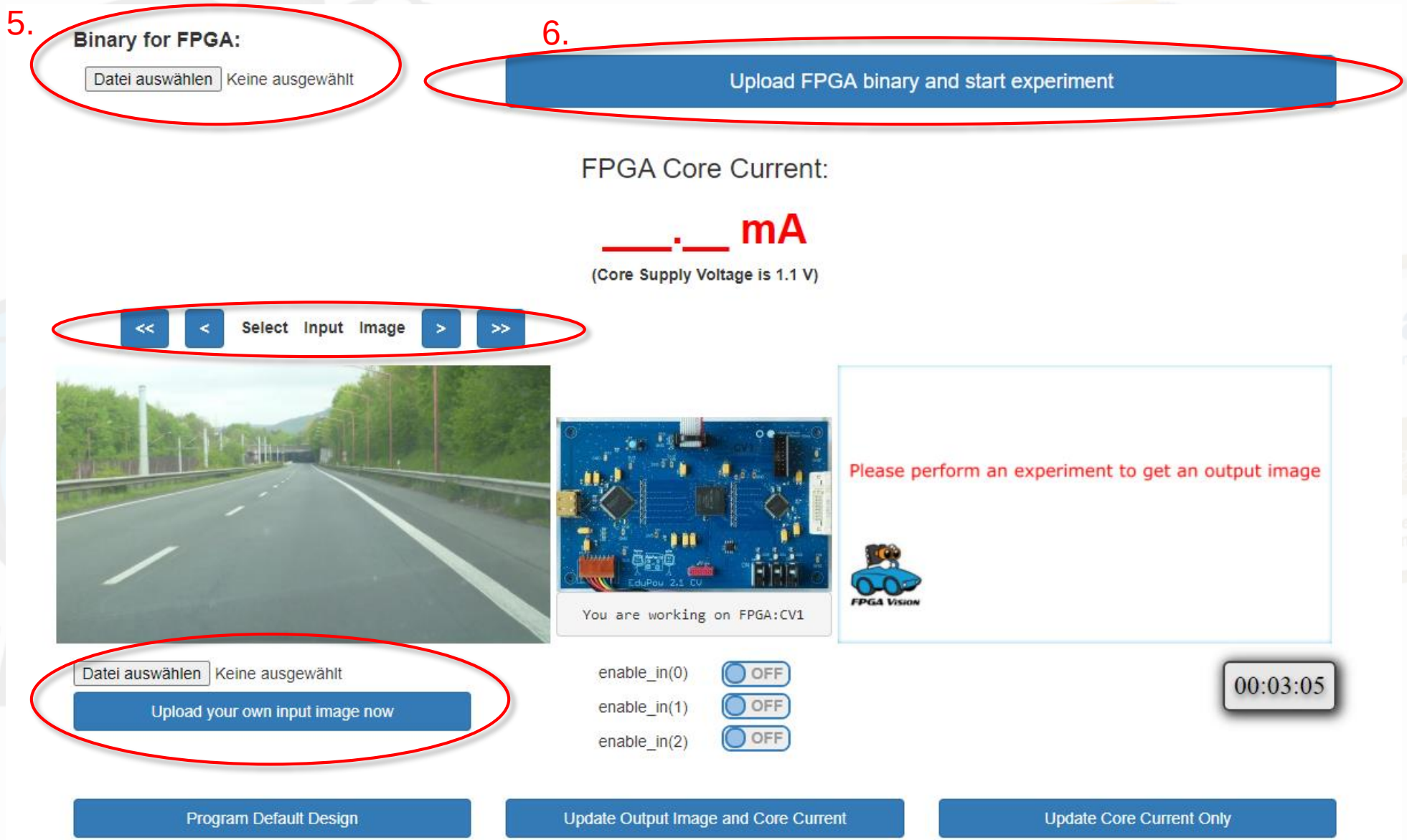
Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Choose and Reserve your Experiment





Test your Design



AND NOW?

ADJUST THE FILTER, TRY OUT OTHER TEMPLATES, …