Teaching FPGA Image Processing with Remote-Lab and Video Lectures

Prof. Dr. Marco Winzker
Tutorial IEEE SiPS 2018, Cape Town

Content

• Education in image processing and hardware design
• Product development of signal processing applications
• Hands-on usage of remote-lab
• Teaching material and integration in lecture

This work by Marco Winzker, Hochschule Bonn-Rhein-Sieg is licensed under a Creative Commons Attribution 4.0 International License.
Education in Image Processing and Hardware Design

Conventional teaching material
• Textbooks for fundamentals of the subject
• Scientific paper for current research topics
• Product information for application information
• Hands-on laboratory for applying theoretical concepts

Digital teaching material
• Similar to existing formats
  ▪ Textbooks in PDF, remote-laboratory
  ➔ More flexibility, lower cost
• New formats
  ▪ Videos: Lecture, tool usage, applications, manufacturing, …
  ▪ Interactive learning: Quiz, …
  ➔ Additional information and activities
Approach to Use of Remote-Lab and Video Lectures

Course structure

- Teaching course often have lecture, exercises and hands-on labs
  - Exercises and hands-on labs start after some theory
  - Lecture and exercises can be combined or lecture includes interactivity

- Remote-lab and video lectures provide additional insight for students
  - Supplemental topic corresponding to 1 or 2 weeks of course (with 3 ~ 6 hours per week)
  - Optional content as elective or for advanced students
Scenarios for Usage in Course

- **optional for advanced students**
  - hands-on lab
  - exercises
  - lecture
  - remote-lab-lecture

- **optional content after semester**
  - hands-on lab
  - exercises
  - lecture
  - remote-lab-lecture

- **final design project**
  - hands-on lab
  - exercises
  - lecture
  - remote-lab-lecture
  - project 1, 2, 3, ...
  - students have choice of projects

remote-lab-lecture = remote-lab and video lectures
Application: Lane Detection

The lecture covers the design of a **lane detection algorithm for cars** and its implementation on an FPGA.

**Different learning objectives**

- **Signal processing:** Understand and optimize image processing algorithm
- **Digital design:** Modify FPGA design and reduce FPGA resources
- **Microelectronics:** Understand and reduce power consumption
Implementation: Remote-Lab-Lecture

**Approach for distance education**

- Video lectures for instructions
- Remote-lab for applying the knowledge
  ➔ Our name for this combination of formats: **Remote-lab-lecture**
- Open educational resource (CC-BY)

**Seven video lectures**

- Between 3 and 15 minutes
- English with subtitles

![Image of a lecture with a professor and a car, with text: Welcome at the Hochschule Bonn-Rhein-Sieg in Germany.]

![Image of a professor with a matrix for Sobel filter, with text: You have a matrix for horizontal and for vertical processing.]

M. Winzker, FPGA Remote-Lab, Slide 8–6
Students can implement the design on an FPGA remote-lab

- FPGA design software on student’s computer
- Upload of FPGA binary to remote server
- Result of experiments: output image and power consumption

- Currently 2 systems with 2 different FPGAs
  - Will be extended to 3 systems with 2 different FPGAs (and further according to demand)
  - Two different locations planned for high availability of experiment
Preparation for Hands-On Exercise

• Installation of FPGA-design Software Intel/Altera **Quartus Lite**
• Installation of **device files** for Cyclone IV and/or Cyclone V
  ▪ Remote-lab has 1 Cyclone IV and 1 Cyclone V experiment
  ▪ For this tutorial 1 additional system with Cyclone V

• If you have a Quartus Software installation with Cyclone IV or Cyclone V device files you can use it (independent of version)
• If you no Quartus Software and are interested in hands-on lab, download the software from [http://fpgasoftware.intel.com/?edition=lite](http://fpgasoftware.intel.com/?edition=lite)
  ▪ Version 18.1 plus device files Cyclone IV or Cyclone V
  ▪ Invoke setup.bat
Application Example – Lane Detection

- Modern cars have a camera in the windshield for
  - Driver assistance like lane departure warning
  - Autonomous driving

- We will look at the first stage in the image processing, the edge detection
  - The application example illustrates the algorithm development
Lane Detection – Steps of Product Development

- Several steps from
  - product idea and specification
  - digital circuit and product
- Division into development steps for all types of product development
  - Digital design
  - Software
  - Mechanical design
  - …

Small company
- 1 or 2 persons for all steps

Large company
- Individual persons for each step
Lane Detection – Specification

- Starting point for algorithm development are the principal approach and a specification including a set of test images
- Specification
  - Find edges in the image of an automotive windscreen camera
  - Output should enable detection of lanes on the street

Test images

Source: Youtube-User Nils169 „Driving from Brig to Luzern Switzerland/ Realspeed“
Quality, Cost, Time

- Requirements on a project or a product have a dependency to each other.
- The three most important project parameter form a "magical triangle":
  - **quality** and functionality,
  - **cost** for development and production,
  - **time** for development.
- Higher distance from the center mean better values (higher quality, lower cost, shorter time)
- The perimeter of the triangle stays the same:
  - Shorter project time increases cost and/or reduces quality or functionality.
Steps of Product Development

1. specification
2. algorithm development
3. fixed-point implementation
4. hardware architecture
5. VHDL design, simulation
6. FPGA synthesis
7. circuit
Lane Detection – Principal Approach

• Principal approach:
  - 3x3 Sobel-Filter
  - Use of greyscale image
    - Conversion with equation: \( Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \)

• Estimation of scaling values
  - Analysis of test image
  - Street has luminance \( Y \) of about 150
  - White lane has \( Y \) of about 220

\[
G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}
\]

\[
G = \sqrt{G_x^2 + G_y^2}
\]

\[
R \approx 150 \quad R \approx 220
\]

\[
G \approx 150 \quad G \approx 220
\]

\[
B \approx 150 \quad B \approx 220
\]
Lane Detection – Range of Values, Scaling

Case: Vertical edge

vertical edge

\[
\begin{bmatrix}
-1 \cdot 2.20 & 0 & 1 \cdot 1.50 \\
-2 \cdot 2.20 & 0 & 2 \cdot 1.50 \\
-1 \cdot 2.20 & 0 & 1 \cdot 1.50 \\
\end{bmatrix} = -280
\]

\[G_y = 0\]

\[G = \sqrt{G_x^2 + G_y^2} = 280\]

Exercise

Calculate G for the following cases.

diagonal edge

slope

slope with transient region

Filter region
Lane Detection – Algorithm Implementation in C

• With the algorithm and the scaling values a first C program is written
  ▪ All calculations are in floating point
  ▪ File lane_float.c

• Visualization of results
  ▪ Edge in test image has value of $G \approx 280$
  ▪ Divide $G$ by 2 and limit to range $[0;255]$
    $$g_{\text{int}} = \frac{g_{\text{root}}}{2};$$
    $$\text{if} \ (g_{\text{int}} > 255) \ g_{\text{int}} = 255;$$
  ▪ White background, edges in black
    $$\text{lum\_new} = 255 - g_{\text{int}};$$

Excerpt of test image
Lane Detection – Algorithm Implementation in C (II)

Names of variables

<table>
<thead>
<tr>
<th>variable</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel_{...}</td>
<td>RGB value</td>
</tr>
<tr>
<td>lum_{...}</td>
<td>luminance value of pixel</td>
</tr>
<tr>
<td>sum_{x;y}</td>
<td>X and Y filter matrix</td>
</tr>
<tr>
<td>g_square</td>
<td>sum_x^2 + sum_y^2</td>
</tr>
<tr>
<td>g_root</td>
<td>square root of g_square</td>
</tr>
<tr>
<td>g_int</td>
<td>mapping of g_root to range [0:255]</td>
</tr>
<tr>
<td>lum_new</td>
<td>luminance value to visualize G</td>
</tr>
</tbody>
</table>
Steps of Product Development

1. Specification
2. Algorithm development
3. Fixed-point implementation
4. Hardware architecture
5. VHDL design, simulation
6. FPGA synthesis
7. Circuit
Lane Detection – Fixed-Point Implementation

• An FPGA implementation will not use floating-point but **fixed-point**
  ▪ Hardware effort for floating-point is very high
  ▪ Floating-point not required, because range of values known in advance
• Floating-point and fixed-point algorithm will give slightly different results
  ▪ Hardware will give exactly the same results as fixed-point implementation
• Fixed-point implementation is called **bit-true**

• The translation from **theoretical algorithm** to a **bit-true implementation** is an important part of a system development
  ▪ Parameters, like scaling values need to be found
  ▪ Word width of fixed-point values have to be determined
  ▪ Modification of the algorithm might be required
    ▪ Different filter mask, e.g. Prewitt, Canny algorithm, …
    ▪ Additional elements, like adaptation to image brightness
Lane Detection – Fixed-Point Implementation (II)

- In this application example, parameters are chosen as an “educated guess”
  - Optimization of parameters can be done as an exercise
  - Some suggestions will be given later

**Design Decision 1**

- Luminance Y shall be coded with 12 bit
  - R,G,B values (8 bit) are multiplied with integer factors that approximate
    \[ Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \]
  - Going from 8 bit (R,G,B) to 12 bit is a factor of \(2^4 = 16\)
    \[
    \begin{align*}
    0.299 \cdot 16 & = 4.78 \approx 5 \\
    0.587 \cdot 16 & = 9.39 \approx 9 \\
    0.114 \cdot 16 & = 1.82 \approx 2
    \end{align*}
    \]
  - Thus: \( Y = (5 \cdot R + 9 \cdot G + 2 \cdot B) / 16 \)
Design Decision 2

- Square-root shall be implemented with ROM-table
  - Cyclone IV and V have Block-RAMs with 13 bit word width
  - Input is limited to 13 bit
  - Required output for visualization is 8 bit
## Lane Detection – Fixed-Point Implementation (IV)

### Word width and range of signals

Fixed-point calculations in VHDL are best done by implementing integer and considering a factor, e.g. 1/16

<table>
<thead>
<tr>
<th>variable</th>
<th>range</th>
<th>format</th>
<th>factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel_{...}</td>
<td>0 : 255</td>
<td>unsigned, 8 bit</td>
<td>1</td>
</tr>
<tr>
<td>lum_{...}</td>
<td>0 : 4080</td>
<td>unsigned, 12 bit</td>
<td>1/16</td>
</tr>
<tr>
<td>sum_{x;y}</td>
<td>±16,320</td>
<td>signed, 15 bit</td>
<td>1/16</td>
</tr>
<tr>
<td>g_square</td>
<td>0 : 533 \cdot 10^6</td>
<td>unsigned, 29 bit</td>
<td>1/256</td>
</tr>
</tbody>
</table>

Multiply with filter matrix:

\[ \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \]

5 \cdot R + 9 \cdot G + 2 \cdot B
Lane Detection – Fixed-Point Implementation (V)

For implementation of square-root we calculate backwards

- \( g_{\text{int}} \) is limited to a range of \([0:255]\)
- Thus \( g_{\text{root}} \) can be \([0:510]\) and \( g_{\text{square}} \) \([0:260,100]\)
  - Corresponds to 18 bit unsigned with range \([0:262,143]\)
- Values of \( g_{\text{square}} \) larger than this range are limited to \( g_{\text{int}} = 255 \)

<table>
<thead>
<tr>
<th>variable</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_{\text{int}} )</td>
<td>( 0 : 255 )</td>
</tr>
<tr>
<td>divide by 2</td>
<td></td>
</tr>
<tr>
<td>( g_{\text{root}} )</td>
<td>( 0 : 510 )</td>
</tr>
<tr>
<td>square root</td>
<td></td>
</tr>
<tr>
<td>( g_{\text{square}} )</td>
<td>( 0 : 260,100 )</td>
</tr>
</tbody>
</table>
Handling of g_square

- g_square has 29 bit and a factor of 1/256
- Word width of ROM for square root can have 13 bit (design decision)
  - Divide g_square by 256
  - Limit g_square to 262,143
    - 18 bit accuracy
  - 13 bit accuracy for g_square by setting the last 5 bit to zero
    - Corresponds to rounding down to multiple of 32
Lane Detection – Implementation of Fixed-Point Algorithm

• The C program is refined with calculations in fixed point
  ▪ File lane_fixed.c
• Comparison of output images
  ➔ No difference visible

Detailed values for test image Position 690, 277

Floating-Point Algorithm

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sum_x</td>
<td>62.7</td>
</tr>
<tr>
<td>sum_y</td>
<td>170.8</td>
</tr>
<tr>
<td>G</td>
<td>181.9</td>
</tr>
<tr>
<td>lum_new</td>
<td>165</td>
</tr>
</tbody>
</table>

Fixed-Point Algorithm

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sum_x</td>
<td>62.8</td>
</tr>
<tr>
<td>sum_y</td>
<td>171.0</td>
</tr>
<tr>
<td>G</td>
<td>182.1</td>
</tr>
<tr>
<td>lum_new</td>
<td>164</td>
</tr>
</tbody>
</table>
Steps of Product Development

1. Specification
2. Algorithm development
3. Fixed-point implementation
4. Hardware architecture
5. VHDL design, simulation
6. FPGA synthesis
7. Circuit

Circuit development flows from specification to FPGA synthesis.
Lane Detection – Hardware Architecture

- Sobel filter needs 3×3 region
- Storage of two video lines gives: top, center, bottom

```
VIDEO-IN

24 (RGB)

tap_b

tap_c

tap_t

top
center
bottom

line = line-memory (1280 pixel)
```
Lane Detection – Hardware Architecture (II)

- Additional register stages (px) give 3·3 region
- Naming of pixel with: “lt”=left-top, “ct”=center-top, ...

```
24 (RGB)

VIDEO-IN

line

px

tap_t

tap_c

tap_b

px

px

tap_lt

tap_ct

tap_rt

tap_lc

tap_cc

tap_rc

tap_lb

tap_cb

tap_rb

line = line-memory (1280 pixel)
px = pixel-memory (register stage)
```

- top
- center
- bottom
- left
- center
- right
Lane Detection – Hardware Architecture (III)

- Submodule for Sobel-matrix used twice
- Different connection of 6 values out of 3·3 region

\[ \text{VIDEO-IN} \]

\[ \text{line} \rightarrow \text{tape}_t \rightarrow \text{tape}_c \rightarrow \text{tape}_b \]
\[ \text{line} \rightarrow \text{px} \rightarrow \text{px} \rightarrow \text{px} \rightarrow \text{px} \rightarrow \text{px} \rightarrow \text{px} \]
\[ \text{tape}_{lt} \rightarrow \text{tape}_{ct} \rightarrow \text{tape}_{rt} \]
\[ \text{tape}_{lc} \rightarrow \text{tape}_{cc} \rightarrow \text{tape}_{rc} \]
\[ \text{tape}_{lb} \rightarrow \text{tape}_{cb} \rightarrow \text{tape}_{rb} \]

\[ \text{OUTPUT} \]

line = line-memory (1280 pixel)
px = pixel-memory (register stage)
Lane Detection – Sobel-Matrix and Square-Root

- **in_p1a**
- **in_p2**
- **in_p1b**
- **in_m1a**
- **in_m2**
- **in_m1b**

**RGB-to-Y**

![Diagram](image)

- **from horizontal Sobel-matrix**
- **result of vertical Sobel-matrix**

- **limit**
- **√X**

- **red**
- **green**
- **blue**

**Operations**:
- *5
- *9
- *2

**Mathematical Expressions**:

- **24**
- **RGB-to-Y**
- ***2**
- **+**
- **15**
- **X²**
- **28**
- **limit**
- **13**
- **8**

**Notes**:
- **RGB-to-Y**
- **from horizontal Sobel-matrix**
- **result of vertical Sobel-matrix**
- **red**
- **green**
- **blue**

**Equations**:

- *5
- *9
- *2

**Diagram Details**:

- **Nodes and Connections**
- **Arrows**
- **Mathematical Expressions**
Potential Improvement for Hardware Architecture

- On purpose, the hardware architecture is not optimal

  - Twelve units for conversion from red, green, blue pixel to luminance (Y)
    - Can be changed to one conversion at video input
    - Line memories will only require 12 bit (not 24 bit)
      → One of many exercises for students
  
  - Also, need for register stage behind line memories can be investigated (*)

(*) Sobel-matrix, square

Sobel-matrix, square

square-root, limit

RGB-to-Y

line

line

px

px

px

px

px

px
Steps of Product Development

1. specification
2. algorithm development
3. fixed-point implementation
4. hardware architecture
5. VHDL design, simulation
6. FPGA synthesis
7. circuit
entity lane 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 lane;
Top-Down-Design with VHDL

*Top-Level:* lane.vhd, lane_sobel.vhd, lane_linemem.vhd, lane_g_matrix.vhd, lane_g_root_IP.vhd

- lane_linemem.vhd
- lane_g_matrix.vhd
- lane_g_root_IP.vhd

- lane_sync.vhd: h-sync, v-sync, data-enable delay
Square-Root as Look-up-Table

- Implementation of square-root in an FPGA Block-RAM
  - Cyclone-V has 176 blocks of M10K-RAM
  - M10K-Ram can be configured to 8K-1bit, 4K-2bit, ..., 256-40bit
  - Here the resolution of 8K (13 bit) input and 8 bit output is used
    - 8 of 175 Block-RAMs required (5%)
- For Cyclone IV
  - 66 blocks of M10K-RAM available
  - 8 Block-RAMs are 12% of available memory

Total memory requirements, including line memories
- 1280 pixel occupy 2048 memory cells
- 24 blocks for RGB storage
  - Cyclone IV: 32 of 66 blocks is about 50% usage
- Only 12 blocks if changed to luminance
Square-Root as Look-up-Table (II)

- Memory content is calculated with a spreadsheet (here OpenOffice Calc)
  - ROM also contains subtraction “lum_new = 255 – g_int;”
    \[ \text{MAX}(0; 255 - \text{ROUNDDOWN} (\text{SQRT} (A7 \times 32) / 2; 0)) \]
  - Text with header can be used as Altera MIF-file
Circuit Simulation

- A circuit simulation is performed with a **testbench**.
- The circuit description is the **design under verification**.
- The testbench generates input signals (**stimuli**) for the circuit and receive the output signals (**response**).

- Testbench and circuit design are simulated in the computer.
Example for a Circuit Simulation

Addition of two bytes

- A and B are 8 bit values
- S is the 9 bit sum of A and B
  - S is generated by the circuit
  - S_EXP is the expected value generated by the testbench
Evaluation of Simulation Results

- For simple circuits, the simulation results can be evaluated and checked in the „Waveform-Viewer“ of the EDA-tool.
- However, this is unreliable, because routine leads to less attention of the human observer.

- A **self checking testbench** generates the stimuli and evaluates the response.

**Example:** Edge detection for images

- The testbench reads a test-image and gives it as a stimuli to the design.
- The testbench compares the response of the design with the reference from the “C”-code.
Detailed Design-Flow for VHDL Simulation

- VHDL can not read image files
- Images are converted to a text file
  - One line is one pixel: Hex values for R,G,B

```
C prog
fixed-point
reference
stimuli
```

```
C prog.bmp2sim.c
bmp files
txt files

# comment
15 28 3F
27 36 A1
A0 49 9C
...
```
Detailed Design-Flow for VHDL Simulation (II)

- Simulation with VHDL files
- Comparison of simulation result with reference

Diagram:
- Stimuli
- Testbench
- Design
- VHDL files
- VHDL simulator (ModelSim)
- Console message: “Simulation OK”
- C program for fixed-point reference stimuli
- BMP files
- BMP2SIM.C
- TXT files

Specifications:
- 15 28 3F
- 27 36 A1
- A0 49 9C
- ...
Detailed Design-Flow for VHDL Simulation (III)

- Simulation result also as a text file
- Can be converted to image format for error analysis
  - If no error occurs, output is identical to reference

![Diagram showing the design flow]

- VHDL files
- VHDL simulator (ModelSim)
- Console message: "Simulation OK"
- Simulation result also as a text file
  - Can be converted to image format for error analysis
  - If no error occurs, output is identical to reference

- C prog fixed-point
- bmp files
- C prog bmp2sim.c
- txt files
- # comment
  15 28 3F
  27 36 A1
  A0 49 9C
  ...

- C prog sim2bmp.c
Steps of Product Development

- Specification
- Algorithm development
- Fixed-point implementation
- Hardware architecture
- VHDL design, simulation
- FPGA synthesis
- Circuit
Quartus Design Software

![Quartus Design Software Interface]

- Quartus Prime Lite Edition
- Project Navigator
- Entity Instance
- Tasks
- Task Progress
  - Compile Design
  - Analysis & Synthesis
  - Filter (Place & Route)
  - Assembler (Generate programming files)
  - Timing Analysis
  - EDA Netlist Writer
  - Edit Settings
  - Program Device (Open Programmer)

Table of Contents:
- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Filter
- Flow Messages
- Flow Suppressed Messages
- Assembler

Flow Summary:
- Flow Status: In progress - Mon Oct 15 16:38:21 2018
- Quartus Prime Version: 18.00 Build 614 04/24/2018 SJ Lite Edition
- Revision Name: lane
- Top-level Entity Name: lane
- Family: Cyclone V
- Device: SCEBA2F17C6
- Timing Models: Final
- Logic utilization (in ALMs): 183 / 9,430 (2 %)
- Total registers: 410
- Total pins: 63 / 128 (49 %)
- Total virtual pins: 0
- Total block memory bits: 125,976 / 1,802,240 (7 %)
- Total DSP Blocks: 2 / 25 (8 %)
- Total HSSI RX PCS: 0
- Total HSSI PMA RX Deserializers: 0
- Total HSSI TX PCS: 0

Running Quartus Prime Assembler

```
Command: quartus_asm --read_settings_files=off --write_settings_files=off lane -c lane
18236 Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment.
```

System (1) Processing (82)
Lane Detection – Implementation Results

For 720P images (1280x720) the edge detection requires the following FPGA resources of the Altera/Intel Cyclone V 5CEBA2F17C6:

- Logic utilization (in ALMs) 183 / 9,430 (2 %)
- Total registers 505
- Total pins 63 / 128 (49 %)
- Total block memory bits 126,976 / 1,802,240 (7 %)
- Total DSP Blocks 1 / 25 (4 %)

Several versions of VHDL code have been tested, resources may vary slightly, depending on version.

- Remark: Number of ALMs is different in data sheet (9,434) and Quartus software (9,430)
Hands-On Exercise

- Website of our project: http://www.h-brs.de/fpga-vision-lab
  - Or Google: fpga vision lab

Available on Website
- Video Lectures
  - YouTube
- Access Remote-Lab
  - Registration and login
  - Demonstration experiment without registration available
- Source Files
  - C-Code of image processing algorithm
  - VHDL code for synthesis and simulation
Hands-On Exercise: VHDL-Synthesis

- **Download VHDL-files** from website (Google: fpga vision lab)
- **Perform FPGA synthesis** on your computer
  - New Project Wizard “lane” with Cyclone IV or V FPGA: EP4CE22E22C7 / 5CEBA2F17C6
  - Add all source files
  - Assignments -> Import Assignments (Cyclone IV or V)
  - Compile Design
- **Access Remote-Lab**
  - Registration at “Get an Account” and “Sign Up”
  - Use your name or stay anonymous
- **Perform Experiment**
  - Reserve Cyclone IV or V lab
  - Upload the FPGA binary: <project directory>/output_files/lane.sof
  - Change input image with slides, observe output and power consumption
Hands-On Exercise: Further Experiments

FPGA Experiment: VHDL Code for Image Inversion (Image Negative)

- Modify lane.vhd
- Invert all input signals and give them to the output
- Option: Use input switches "enable_in" to select inversion
- Option: Change RGB to luminance (see lane_g_matrix.vhd for conversion)

VHDL Simulation

- If you have installed Modelsim, perform simulation of the testbench sim_lane.vhd

C-Simulation of Image Processing Algorithm

- Download C-code from website
- Compile lane_float.c
- Modify code, e.g. change accuracy of luminance to 8 bit integer
Integration into your Course

Special properties of the FPGA remote-lab

- **Advanced experiment**: Supplement to regular course
- **Exclusivity**: Standard FPGA boards not equipped for image processing
- **Real input required**: Students have to perform programming
- **Authentic feedback**: Result of experiment depends on input and is not trivial to predict

Remote-Lab is open for wide range of experiments

- Experiments for **lane detection** or other **image processing algorithms**
- Experiments can be: **Easy, Moderate, Advanced, Complex**
Lecture Videos

Technical Background

- **Introduction** (2:24 minutes)
  scope and overview of the lectures

- **Lane Detection** (9:58 minutes)
  algorithm and verification
  implementation as C program

- **Low-Power Design** (15:48 minutes)
  CMOS power consumption
digital design for low-power

Digital Design

- **Circuit Design** (14:18 minutes)
  architecture of lane detector
  fixed-point implementation

- **VHDL Simulation** (10:13 minutes)
  verification concept
design flow for simulation

Experiment on Remote Lab

- **FPGA Remote Lab** (8:44 minutes)
  FPGA design flow
  access to remote lab

- **Next Steps** (2:39 minutes)
  suggestions for further experiments

- Subtitles in English, Arabic, Spanish
  - You can contribute subtitles in your language
- Students select which videos are of interest for them
Exercises

Lecture slides with exercises available
• Website -> “Source Files”
• Documents: Lecture-Slides_FPGA-Vision.pdf and .pptx

Introductory experiments like in hands-on lab
• Synthesis of available VHDL-code to set up design flow
• Image inversion

Further Experiments with Motivation
• See next slides
Optimize Position of RGB-to-Y Conversion

- RGB input pixel are stored in two line memories for vertical edge detection. For processing they are converted to luminance (Y).
- It is better to first convert RGB to Y and save memory.
- Also this reduces resources for processing, as only one conversion is required (as discussed earlier).

**Complexity: Moderate**
Reduce Word Width of Luminance Values

- RGB input pixel are converted to 12 bit luminance. Reduce the word width of luminance values to save resources.
- Reducing the word width of luminance values also reduces the word width of the consecutive processing steps (see diagram for an example).
- Modification of C-Code is required for verification

Complexity: Advanced
Power Consumption of different FPGAs

- Implement a design on two different FPGAs and compare resource usage and power consumption.
- Understand the influence of CMOS technology for power consumption

**Complexity: Moderate**

See video lecture “Comparing CMOS Technologies with FPGA Experiments”
https://youtu.be/7hSjqMc742A
Reduce Power Consumption by “Sleep Mode”

• Lane detection is active for the complete image. However, lanes are not present in the top region of the image (see diagram).
• Processing can be switched off for the top of the image to save power.

Complexity: Advanced
### Results for Cyclone IV FPGA

<table>
<thead>
<tr>
<th>Design</th>
<th>Logic Elements</th>
<th>Memory Bits</th>
<th>Power [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>910</td>
<td>126,976</td>
<td>47.6</td>
</tr>
<tr>
<td>RGB-to-Y before line memories</td>
<td>462</td>
<td>96,256</td>
<td>35.3</td>
</tr>
<tr>
<td>as above, 8 bit for luminance Y</td>
<td>398</td>
<td>86,016</td>
<td>30.8</td>
</tr>
<tr>
<td>as above, 6 bit for luminance Y</td>
<td>354</td>
<td>80,896</td>
<td>27.0</td>
</tr>
<tr>
<td>as above, ROM for square root with 8 bit input (instead of 13 bit)</td>
<td>337</td>
<td>17,408</td>
<td>20.2</td>
</tr>
<tr>
<td>as above, „sleep mode“ using fixed value for Y at top third of image</td>
<td>361</td>
<td>17,408</td>
<td>19.0</td>
</tr>
<tr>
<td>as above, stop line memories at top 1/3</td>
<td>377</td>
<td>17,408</td>
<td>17.4</td>
</tr>
</tbody>
</table>

- Changes in FPGA design are **clearly observable in resource usage**
- Students can check quality of images after change of algorithm

Values might vary slightly with temperature of FPGA and implementation details (like in every experiment)
Image Enhancement with Sharpening Filter

You can also use the remote lab for general experiment about image processing.

The following experiments are not specific for lane detection:

- Improve perceived image quality by sharpening the image with an FIR filter.
- Research literature about sharpening filter. Different filter functions are possible.

**Complexity: Advanced**
Brightness Adjustment

- When you have images that are dark or bright, it can be useful to adjust the brightness.
- Also the contrast can be enhanced.
  - Develop an algorithm using C code.
  - Implement the algorithm with VHDL, simulate it and check functionality on the remote-lab

**Complexity: Complex**
CMOS Technology and Power Consumption

- Implement a design on two different FPGAs and compare resource usage and power consumption.
- Implement a shift register with different numbers of flip-flops and compare resource usage and power consumption.
- See video lecture “Comparing CMOS Technologies with FPGA Experiments” [https://youtu.be/7hSjqMc742A](https://youtu.be/7hSjqMc742A)

**Complexity: Moderate**
Further Steps in Lane Detection

And you can use the edge detection as a starting point for further processing steps

• The current design detects edges in the input image.
• Next step in a lane detection system are identification of lane boundaries, tracking of lanes, warning when a vehicle crosses lanes.
• Superimpose lane boundaries on the original or edge detection image.

Complexity: Very Complex
Summary

Remote-lab and video lectures are an open educational resource

- Algorithm and FPGA implementation for lane detection
- Supplement to a course and for individual students

http://www.h-brs.de/fpga-vision-lab

Rework, Reuse, Remix

- Use the material and inform your colleagues
- Rate us at merlot.org (keyword FPGA)
- Contribute subtitles in your national language

Reference


Acknowledgement

- Ministry for Innovation, Science, Research and Technology of the State of North Rhine-Westphalia, Germany
- Stifterverband, Germany