

## 1. Introduction

This hypertext based VHDL teaching program is divided into several exercises. By writing typical VHDL programs you learn how to use this hardware description language.

In the beginning you are guided in a step by step manner, later on when you are used to the language and the tools you will solve the tasks on your own.

We are referring to the VHDL 87 standard, because the VHDL '93 is not yet supported by all tool manufacturers.

## 2. Structure

The tasks are divided into different sections:

- **GOAL:** In this section you learn about the goal of the task and the function of the model within the whole design.
- **DESCRIPTION:** This section tells you more about the function of the model.
- **HINTS:** In this section you are introduced to the problems which can be caused by VHDL. The help is supposed to assist you in solving the task.
- **GIVEN:** Here you find information how to name the files, entities, architecture etc. and which data type to use. You will also be told which description style to use, if necessary.
- **TO DO:** Here is the precise description of your job to solve the task.
- **VHDL Commands:** Here you find a list of the tools to use or a list of your files. You will learn at the end of this introduction how to use the tools.

## 3. Naming

In VHDL it is very important to have a consistent naming. The following naming conventions should be followed within this workshop:

**Tabelle 1: Naming conventions**

<b>File name:</b>	<b>mydesign.vhdl</b>	for the VHDL file
<b>Entity name:</b>	<b>MYDESIGN</b> <b>TB_MYDESIGN</b>	for the model for the testbench
<b>Architecture name:</b>	<b>RTL</b> <b>BEH</b> <b>TEST</b>	for RTL code for behavioural descriptions for the architecture of the testbench
<b>Configuration name:</b>	<b>CFG_TB_MYDESIGN</b>	for the configuration of the testbench
<b>Typedefinition name:</b>	<b>T_&lt;TYPE_NAME&gt;</b>	a leading T_ ...

## 4. The Design

Within this course you will design a camera control system. The objective is not to develop an optimal design, but to learn the hardware description language VHDL. At the beginning you will receive a lot of guidance. The guidance will become less and less until it eventually vanishes and you solve the tasks on your own.

The camera design consists of seven different modules to be developed:

1. **The decoder (decoder):** This unit will decode the signals from the input device in order to display the exposure time.

2. **The exposure latch (`exposure_latch`):** This latch stores the exposure time.
3. **The timer for the film transport (`timer_transport`):** If an error occurs while the film is transported (end of film or a tear in a film) this unit will report the error. (The maximum transport time is two seconds).
4. **The exposure timer (`timer_exposure`):** stores the exposure time selected via the input device.
5. **The photo controller (`photo_control`):** The control unit for the camera control.
6. **The display controller (`display_control`):** This unit controls the output device.
7. **The display driver (`display_driver`):** The driver for the output device.

The clock frequency of the camera control is 8192 Hz. It allows for exposure times according to the following formula:  $2 \exp(-i)$  (for  $i = 0..9$ ). You need this information later on in order to implement the required times. The camera control will be able to:

- determine the exposure time
- generate a control signal for the servo which transports the film
- taking a sequence of pictures
- generating the exposure control signal
- detection of end of film and tear in a film errors while transporting the film.

## 5. The text editor

The goal of this pre-exercise is to get used to the simulation and synthesis tools, and to learn the basic commands of the Synopsys tools. This exercise does not require any specific VHDL knowledge. The Synopsys VHDL simulation tool supports the IEEE 1076 standard, so you don't have to learn a VHDL subset only supported by Synopsys.

The first exercise is a AND gate. The file **und.vhd** contains a VHDL description of an AND gate with two inputs. To get used to the tools proceed through the following steps:

- Select the hyperlink **edit** in the section VHDL Commands for the **und.vhd** file. This will cause a text editor window to pop up displaying the VHDL code for the AND gate.



- **Important:**

**Hit the *Stop* key of your browser in order to interrupt the transmission.** (The browser daemon is waiting for a response of the application. The response is generated only after finishing the application. Hence, the daemon would wait until you finish the application and you would not be able to continue browsing until then.) If you are using a browser knowing about frames it is sufficient to select an other section in the sections frame.

- You can now use the editor. Now edit also the file `tb_und.vhd`. (Don't forget to press the Stop key...)
- Some questions to the testbench:  
Please write down the names of the entity and the component.  
Which other major design unit is used within this file? (Other than entity and architecture)  
Write down the type of the design unit and the logical VHDL name.

## 6. The VHDL Analyzer

- All the files of a VHDL design have to be analyzed in a given order. Which order is this for the two files?
- Use the hyperlinks in the appropriate order. The result of the analysis will be displayed in an extra page. Use the *back* button to get back to this page.
- Correct the errors you found with the text editor.

## 7. The VHDL Simulation

After you have analyzed all files in the correct order you can start the analyzer now by clicking on the hyperlink in the VHDL Commands section. A VHDL debugger select window should pop up:

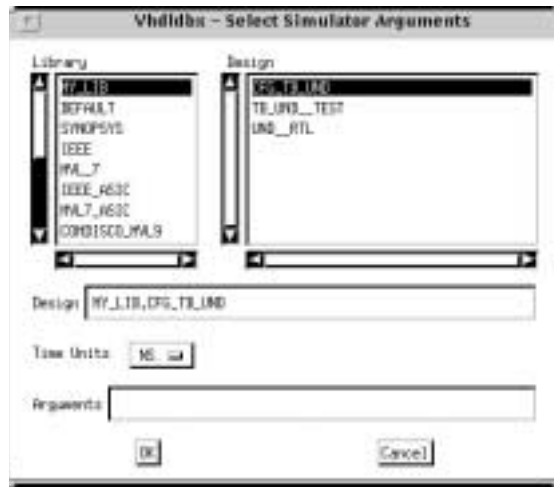


Figure 1: VHDL-Debugger-Select-Window

- The simulator is started via this window by selecting the name of the top level configuration. --> **The configuration is the object to be simulated.**
- After selecting the "top level configuration" for your "design hierarchy" by double clicking it the VHDL debugger simulator will pop up:

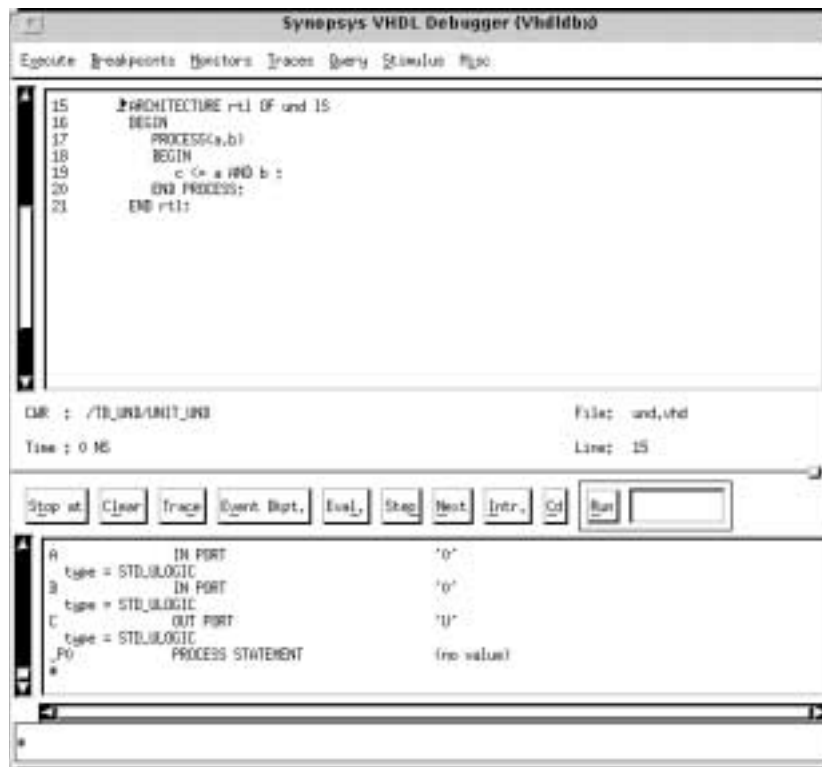


Figure 2: VHDL-Debugger-Simulator-Interface



- **Important:**

As with the editor you have to press the *Stop* button of your HTML browser to finish the service and be able to concurrently use your browser and the Synopsys tool.

- Again, you don't have to exit the simulator, it is okay to iconify it, although you are locking one of the limited licenses then.
- By using the *Execute->Restart* menu you can always simulate another analyzed design.
- **The simulator interface** is a debugger. Its "top window" displays the VHDL code and its "bottom window" is the command line interface to the simulator. Several commands can be executed via the menus, for other frequently used commands "quick buttons" are provided in the center of the main window.
- The signals and variables can be traced in different ways, we are using the following:
  - **Waveform Display:** the Synopsys "waveform display" is able to display "single bit" and "array type" signals, but it cannot display INTEGERS.
  - **Signal Evaluation:** here you can select arbitrary signals or VHDL expressions from the source code to be displayed and interactively changed during the simulation.

### 7.1. How to use the simulator (1)

- The text in the top window is the source code for the testbench. You can move the cursor within this window if it is selected (e.g. by clicking on it).
- You can move your design hierarchy using commands similar to the UNIX commands. You have to enter them in the command line window. You can use UNIX wild cards. Try the following commands:

```
cd /tb*
```

```
ls
```

```
ls *'sig    (to get a list of all signals within the current scope)
```

```
ls -t
```

```
ls -v
```

```
ls -t -v
```

```
pwd
```

```
cd unit_und    (also reachable via the quick buttons)
```

```
cd..
```

- to get the signals displayed in the waveform display:

```
cd /TB*
```

```
trace *'sig
```

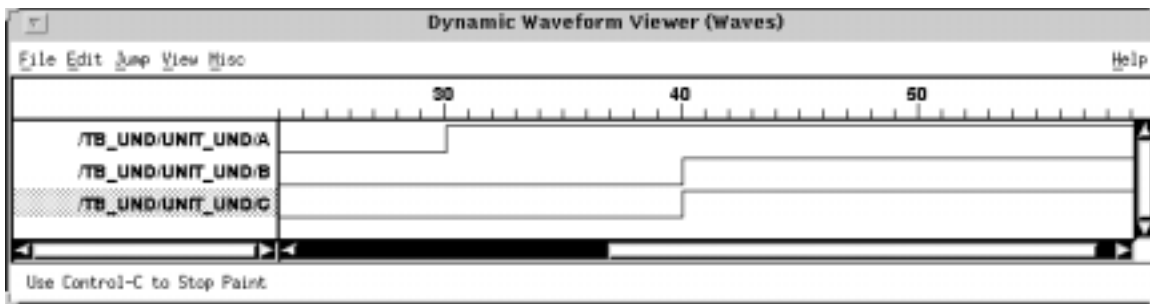


Figure 3: VHDL-Waveform-Viewer

### 7.2. How to use the simulator (2)

You can control the simulation in three ways:

- You can run a simulation for a certain amount of time,
- you can define breakpoints,
- and you can step through your code (Next-Button, Step-Button)!
- Note, however, if you use a break point (**Stop-at-Button**), the simulation is interrupted at

this very point, i.e. not all signals carry a current value for this simulation time step.

- Run the simulation as a sequence of 10ns steps. To do so type “10” in the window next to the **run-Button**. Every time you hit the run button the simulation time proceeds for the amount of time steps displayed aside:
  - Watch the behaviour of your design in the waveform display. Try to use the features of the waveform displays which you can reach via the menus. You have to choose a command and to select an object to be used by the command. You can interrupt a command by pressing the middle mouse button.
- Use the step buttons for a step by step execution. Find out when the signals are updated. Select a signal or a VHDL expression and evaluate it with the **Eval Expr** button.
- Set a breakpoint on the output signal **Value** by selecting the signal in the code window and pressing the **Event Bkpt** button. Clear the run-box window next to the run button (remove the 10). This will cause the simulation to run until it hits a breakpoint. Press the run button.
- Breakpoints can also be set to code lines: Move the cursor to the line you wish to set to breakpoint to and press the **Stop At** button.
- You can delete breakpoints via the **Breakpoints -> Delete Monitors** menu. Take care not to delete items, e.g. M, PP1, PP2....., because they are linked to the signal window.
- You might have noticed an error in the execution of the model. If you want to restart the simulation (restart at time “0”) use the **Execute -> Restart** menu. All commands you have executed before the first run command are then reexecuted.

## 8. VHDL Synthesis

After you have verified the function of your model with the simulator you can run the synthesis. Click on the synthesis hyperlink in the VHDL Commands section. The Synopsys Design-Analyzer will pop up.

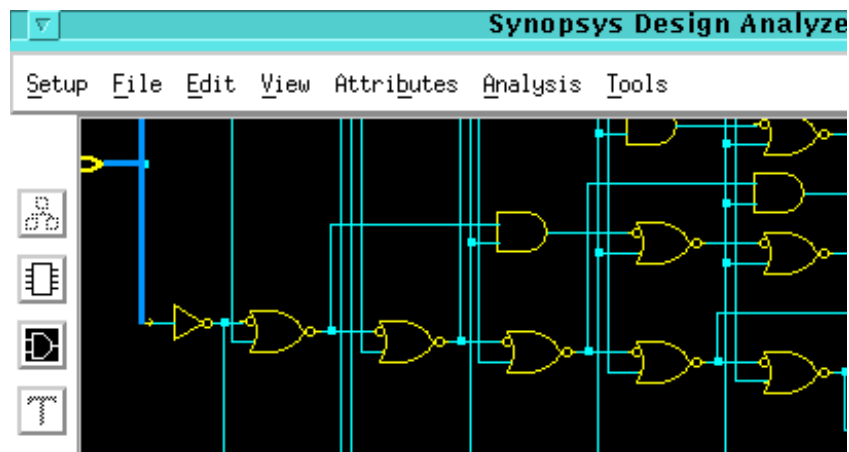


Figure 4: VHDL-Design-Analyzer



### Important:

- As with the other tools you have to press the **Stop** button of your HTML browsers.
  - You don't have to exit the tool, you can iconify it whenever you don't need it.
  - Take care not to have open more than one window of each of the synopsys tools, you would lock other people's licenses.
  - You can change the design to work on with the File->Read menu.
- In our configuration the design-analyzer uses a start-up-file to auto-load the **IEEE.Std\_Logic\_1164** package and to select the ASIC library (the "LSI Logic 10K series").
- Read the file “und.vhd” (via File -> Read). Skim through the tool messages to check whether

your design is conform to the synthesis subset which is supported by Synopsys. (synthesis tools in general do only support a tiny subset of the 1076-VHDL standards. This subset is not standardized.) Warnings such as "Design Library management not supported" and "Library clause not supported" can be ignored. Press the "Cancel" button to close the message window.

In future tasks/exercises we will define our own packages. The synthesis tool requires that packages are read before they are used or referenced.

**The most frequent errors with the synthesis are missing, not read packages!**

- Double click on the symbol representing your design and see the first generic netlist. (Translation of the VHDL code in a set of tool specific basic gates).
- Switch off the optimization "mapping to gates" in the Tools -> Design menu. The tools are optimizing for area if no specific synthesis constraints are set. Use the defaults in the Design->Optimisation menu, just press "OK".
- You will now get a netlist which is optimized for the target library (LSI10k).
- Double click the symbol when the synthesis is finished to see the gate level implementation. Zoom commands are available via the popup menu linked to the right mouse button. First select the zoom command, then the area to be zoomed.
- Execute the command Analysis -> Report to generate different reports.

**Well done! You have managed to survive!**

# 1. Exercise: Multiplexer

## 1.1 GOAL

- Your first task is to write the VHDL description of a multiplexer. The testbench is given, but you might have to add some stimuli to verify your design. Subsequently, you will simulate and synthesize your model.

## 1.2 DESCRIPTION

- You need a display for the camera. The display will either show the number of pictures of the exposure time. This is the schematic of the multiplexer:

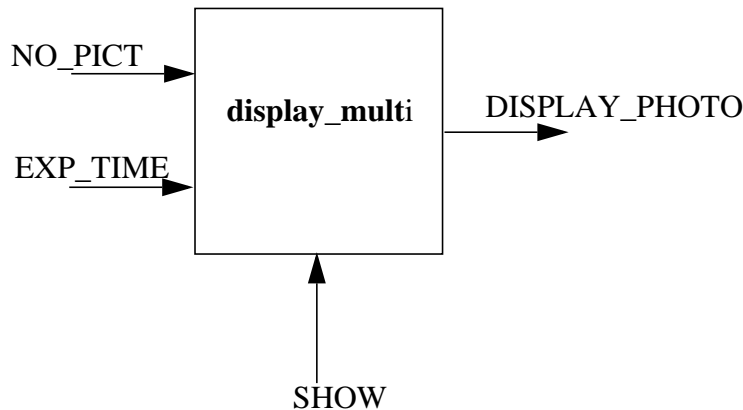


Figure 5: The Multiplexer

- The algorithm for the multiplexer: If  $SHOW = 0$  then show the number of pictures else show the exposure time.

## 1.3 HINTS

- The configuration is the “hook” for the simulation. For the sake of simplicity the testbench and the configuration are located in the same file. It is not just possible, but for larger designs recommended to use different files for design units (entity, architecture, package, package body and configuration are the five design units). We use the default configuration. In this case component and entity objects with identical names are wired by the position (not the name) of the port signals.

## 1.4 GIVEN

- Data types:** Use the type Integer for the exposure time, number of pictures, and display. For the signal SHOW use std\_ulogic.
- Names of the design units:** Name the entity DISPLAY\_MULTI, and the architecture BEHAVE.
- File names:** Name the file **display\_multi.vhd** and the file for the testbench **tb\_display\_multi.vhd**.

## 1.5 TO DO

- Describe the multiplexer in VHDL with the editor (use the hyperlink edit).
- Edit and analyze your program until it analyzes successfully (hyperlink analyze)
- Extend the testbench with additional test vectors (stimuli).
- Edit and analyze the testbench until you analyze it successfully.
- Synthesize the model. The generated signals have a width of 32 bit. Using a RANGE declaration can avoid this waste. You will see this in the following exercise.

1.6



## 2. Exercise: Extending the Multiplexer

### 2.1 GOAL

- In this exercise you will extend the functionality of the multiplexer, adapt the testbench accordingly, and simulate the design again.

### 2.2 DESCRIPTION

- If an error occurs during normal operation (e.g. the servo is defect, a tear on the film) an error message should appear on the display. Therefore the new input signal ERR is introduced. ERR is '1' if an error has occurred. The multiplexer changes as follows:

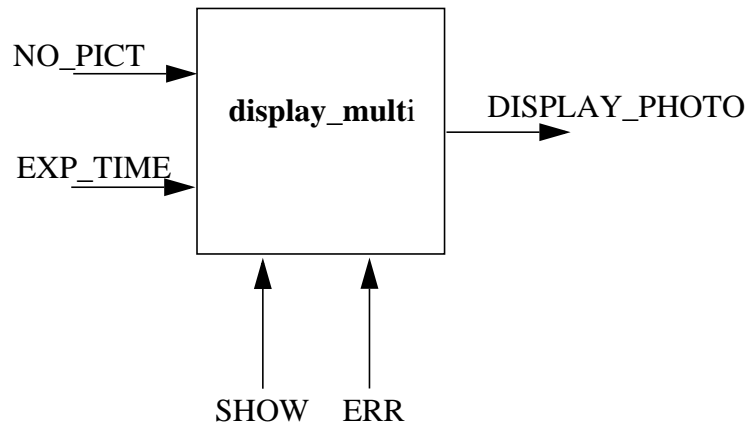


Figure 6: Der Multiplexer

- If an error has occurred DISPLAY\_PHOTO is assigned a '10' else the function as described in exercise one applies, i.e. the result depends on SHOW.

### 2.3 GIVEN

- Data types:** Use std\_ulogic for the signal ERR.
- Names:** The Names remain as in exercise one.

### 2.4 TO DO:

- Apply the required modifications.
- Extend the testbench so that it verifies also the new functionality.
- Analyze the testbench and simulate the new model.
- Synthesize your new model.

### 3. Exercise: A Seven Segment Display

#### 3.1 GOAL

- In this exercise you will develop a VHDL description for a seven segment display. Later on you will extend this display in order to get a three digit seven segment display. You can use the available package `p_digit`.

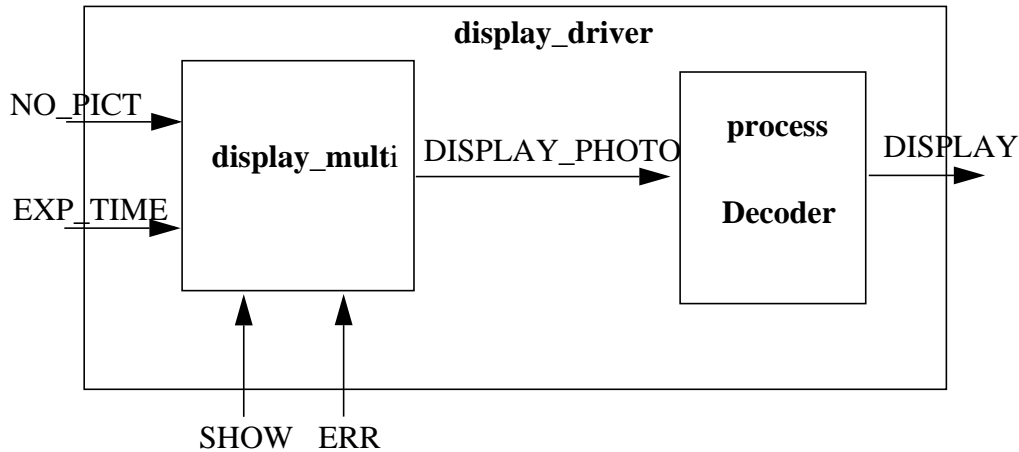


Figure 7: The Display-Driver

#### 3.2 DESCRIPTION

- Now we rename the design to `DISPLAY_DRIVER`. It provides basically the same functionality, the difference is that the output signal is now called `DISPLAY` and its width is now seven bit in order to drive the seven segment display. The former signal `DISPLAY_PHOTO` will be used as an internal signal from now on. This is necessary, because assignments to the new output signal `DISPLAY` are not allowed since they have different data types. The internal signal `DISPLAY_PHOTO` keeps its data type.
- You have to write a decoder which transforms the binary information in `DISPLAY_PHOTO` into the signals driving the seven segments. The package `p_digit` contains the definitions `ZERO_SEG` till `NINE_SEG`. They define which segment to drive for which number. The additional definition `E_SEG` can be used, if an error occurs and should be assigned to the number 10.
- Since you are operating on a single digit in this exercise, the range of the input signal (`INTEGER`) has to be limited from zero to ten (including the E to signal an error).

#### 3.3 GIVEN

- Data types:** use `std_ulogic`, `std_ulogic_vector(6 DOWNTO 0)` for the new output signal `DISPLAY` and `INTEGER RANGE 0 TO 10` for the number of pictures, exposure time, and `DISPLAY_PHOTO`.
- Files:** Copy the files from exercise one.
- Names of the design units:** Change the names of the units as follows.  
Model: Entity `DISPLAY_DRIVER`, architecture `BEHAVE`  
Testbench: Entity `TB_DISPLAY_DRIVER`, architecture `TEST`, configuration `CFG_DISPLAY_DRIVER`
- Package:** Use the package `p_digit`. To make the package available within an entity you have to insert the following line before the entity declaration:

```
USE work.p_digit.all
```

The package has to be analyzed before the design.

- **Style:** Use a separate process (sensitive to DISPLAY\_PHOTO) to describe the coding of your seven segment display. Use the case statement to distinguish the numbers 0 to 9 and the error E.
- **Seven segment display:** As mentioned already you can use the package p\_digit. This package allows for displaying the seven segments graphically in a window. Use a concurrent statement

```
DISPLAY_DIGIT(DISPLAY);
```

in your testbench. This procedure is defined in the package and takes care of the proper updating of the seven segment display window.

Open the window by clicking on **display\_digit**.

### 3.4 TO DO:

- Change the names as described above.
- Write the process to drive the seven segment display.  
Before you analyze and edit your design make sure the package is analyzed.
- Adapt the testbench. Don't forget to insert the statement used to update your seven segment display.
- Open the window for the seven segment display and simulate your model.
- Synthesize your model and optimize it for minimum area size. Don't forget to synthesize the package before synthesizing your model. Write down the required area size.

## 4. Exercise: A Three Digit Seven Segment Display

### 4.1 GOAL

- Extend your DISPLAY\_DRIVER so that it can drive a three digit display. You will learn how to use a package containing data types common to several different design units.

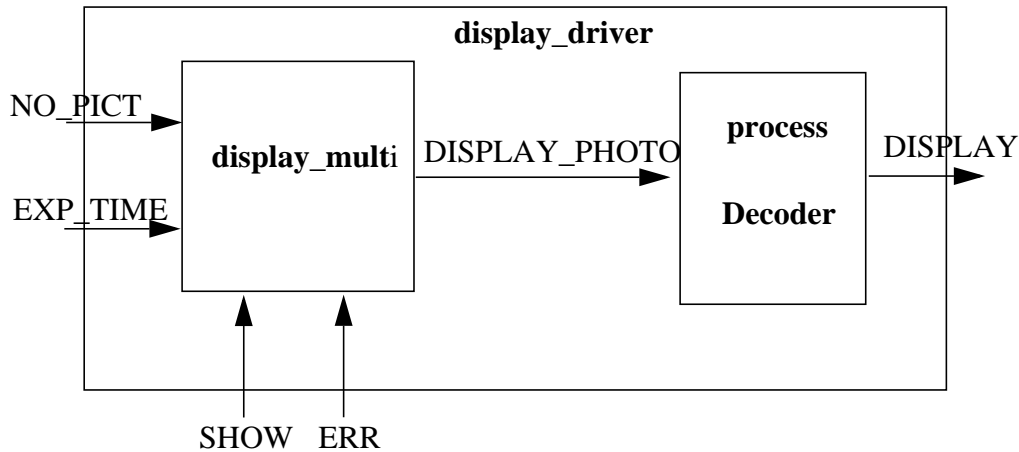


Figure 8: The Display-Driver

### 4.2 HINTS

- Advantage of a package:** A (large) VHDL design is usually developed by a team. Therefore it is sensible to define common data types for common signals. This is done by defining one package which is then used by all designers. In our case the two signals NO\_PICT and EXP\_TIME shall be of type T\_ANZEIGE. This data type is defined in the package **showdisplay** (file name is p\_showdisplay.vhd). The output signal DISPLAY shall be of type T\_DISPLAY.
- Arrays:** The data type T\_ANZEIGE is a three dimensional array containing INTEGERS within the range 0 to 10 (10 for E). The type T\_DISPLAY is a three dimensional array of 7 bit signals to drive the three seven segment displays.
- Integer with range:** VHDL allows for limiting the range of a INTEGER number. If the range is not limited the synthesis of integer signals will result in a 32 bit representation. A RANGE 0 TO 64 limited integer will result in a 7 bit representation.
- Loop:** Use the FOR-LOOP statement to drive the different seven segment displays. During each loop iteration one integer value is to be transformed into the driving signals.

### 4.3 DESCRIPTION

- The function of the DISPLAY\_DRIVER remains identical, however the data types change. The new data type for the input signals NO\_PICT and EXP\_TIME is T\_ANZEIGE, the new data type for the output signal DISPLAY is T\_DISPLAY. Both are defined in the package showdisplay.

```
TYPE T_ANZEIGE IS ARRAY (2 DOWNTO 0) OF integer RANGE 0 TO 10;
```

```
TYPE T_DISPLAY IS ARRAY (2 DOWNTO 0) OF std_ulogic_vector(6 DOWNTO 0);
```

### 4.4 GIVEN

- Data types:** Use the predefined types T\_ANZEIGE und T\_DISPLAY.
- File names:** No changes with the file names.
- Names of the design units:** No changes with the names of the design units.
- Testbench:** The stimuli of your testbench have to be changed according to the new data

types. You can use the FOR-LOOP statement to describe your stimuli.

- **Package:** Note that you have to use the package **showdisplay** now. Open the display window by clicking on

**display\_photo**

Use the

**display\_photo(DISPLAY);**

statement in your testbench in order to get the window updated.

#### 4.5 TO DO

- Apply the necessary changes to your design and your testbench. Analyze and edit them until they work error free.
- Simulate your design and verify the funktion.
- Synthesize your design and optimize it for minimum area size.

## 5. Exercise: A Decoder

### 5.1 GOAL

- An entity architecture couple for the input from the keyboard.

### 5.2 DESCRIPTION



Figure 9: The Decoder

- The keyboard has ten buttons for the different exposure times. The decoder will transform the input signals into the output signal KEY of the type T\_ANZEIGE. The input signal is of the type std\_ulogic\_vector(1 TO 10) and is delivered by a keypad unit. It consists of nine zeros and one one. The position of the one contains the information how long the exposure time should be. The signal KEY shall get the decimal number of the exposure time, i.e. keypad "1000000000" will result in one second exposure time and KEY will be assigned (0,0,1), keypad "0100000000" will result in half a second exposure time and (0,0,2) will be assigned to KEY and so on (exposure times are calculated as follows:  $KEY = 2 \exp(-i)$ , for  $i = 0 \dots 9$ ). If an other key is pressed, (0,0,0) is assigned to KEY. Use a case statement.

### 5.3 GIVEN

- **Data types:** use std\_ulogic\_vector, and T\_ANZEIGE which is defined in the package show-display.
- **File names:** use **decoder.vhd** for the design and **tb\_decoder.vhd** for the testbench.
- **Names of the design units:** use DECODER for the entity, RTL for the architecture, and CFG\_DECODER for the configuration.

### 5.4 TO DO:

- Write the decoder. Analyze and edit until it is error free.
- Write a testbench and verify the function of your design. Simulate the decoder.
- Synthesize your design Design and optimize it for minimum area size.

## 6. Exercise: A Register

### 6.1 GOAL

- The goal is to write a register which stores the exposure time.

### 6.2 DESCRIPTION

- When a exposure time is selected by the user it has to be stored. The register is keeping its value or storing a new value with the rising edge of the clk signal. The register has a reset input. The value is set to one second if the register is reset. The model is as follows:

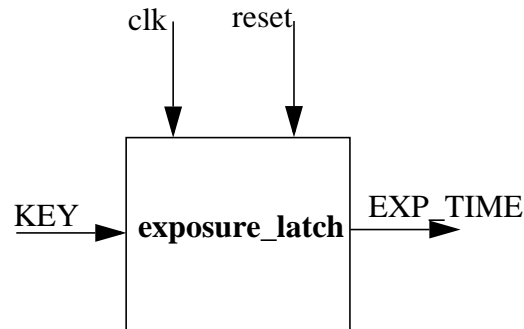


Figure 10: The exposure time register

- The algorithm is the following:

```

IF reset = '1' THEN
  reset design
ELSIF rising clock
  IF KEY pressed
    load it
  END IF
END IF
  
```

### 6.3 GIVEN

- **Data types:** use Std\_ulogic and Typ T\_ANZEIGE.
- **File names:** use exposure\_latch.vhd for the design and tb\_exposure\_latch.vhd for the testbench.
- **Names of the design units:** use EXPOSURE\_LATCH for the entity, RTL for the architecture, and CFG\_EXPOSURE\_LATCH for the configuration.
- **Package:** use the package showdisplay.
- **Style:** Use a process. It has to be sensitive to the clk and reset signals.

### 6.4 TO DO:

- Write the exposure time register, analyze and edit it until it is error free.
- Write a testbench to verify your design and simulate the exposure time register.
- Synthesize your design and optimize it for minimum area size.

## 7. Exercise: A Counter

### 7.1 GOAL

- A counter to watch the maximum transport duration.

### 7.2 DESCRIPTION

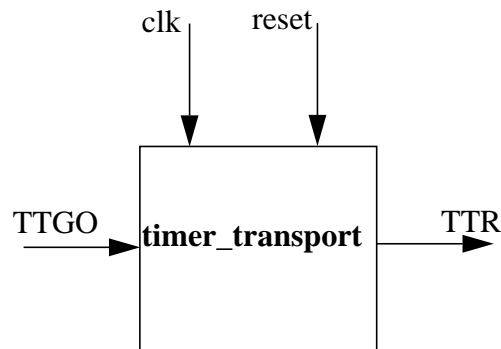


Figure 11: The counter for the transport duration

- Two seconds after the signal TTGO (Timer Transport Go) is switching to '0', the signal TTR has to switch from '0' to '1'. TTGO remains at '0' for one clock cycle (this is granted by the module photo\_control). Since the delay is two seconds and the clock frequency is 8192Hz, the counter has to count from 0 to 16383. After counting to 16383 the counter has to produce a output signal.

### 7.3 HINTS

- Advantages of variables: Signals are globally available and are updated only at the end of a process, i.e. if a signal is assigned a '2' and a '4' within one process, the signal is assigned the '4' at the end of the process and the '2' is discarded. Variables are local to a process and an assignment is immediately valid (VHDL 93 allows also global variables).
- In this design the output signal TTR shall switch to '1' for one clock cycle only when the time has expired. The problem can be solved with variables. Using variables causes the counter to start immediately. The variable storing the counter value shall be called counter. The counting algorithm follows:

```

IF reset ...
ELSIF clk....
  TTR is assigned '0'
  IF TTGO.....
    var is set to '1'
    counter is set to '0'
  END..
  IF var = '1' ...
    IF counter /= ...
      ...
    ELSE
      TTR ...
      var is set to '0'
    END..
  END..
END...
END...

```

### 7.4 GIVEN

- **Data types:** use INTEGER with a range for counter and std\_ulogic for the other signals.
- **File names:** use **timer\_transport.vhd** for the desitgn, and **tb\_timer\_transport.vhd** for the

testbench.

- **Names of the design units:** use `TIMER_TRANSPORT` for the entity, `BEHAVE` for the architecture, `TB_TIMER_TRANSPORT` for the testbench, and `CFG_TIMER_TARNSPORT` for the configuration.
- **Package:** No package is required for this design.

### 7.5 TO DO:

- Write the counter and a testbench.
- Simulate your design and verify its funktion.
- How many flip flops are required for the counter?
- Synthesize your counter and optimize it for minimum area size. Verify your answer regarding the flip flops.

## 8. Exercise: A Timer

### 8.1 GOAL

- The timer for the exposure has to be designed now. It will be realized by a counter counting up to a certain number. The picture counter shall be sensitive to the input signal TEGO.

### 8.2 DESCRIPTION

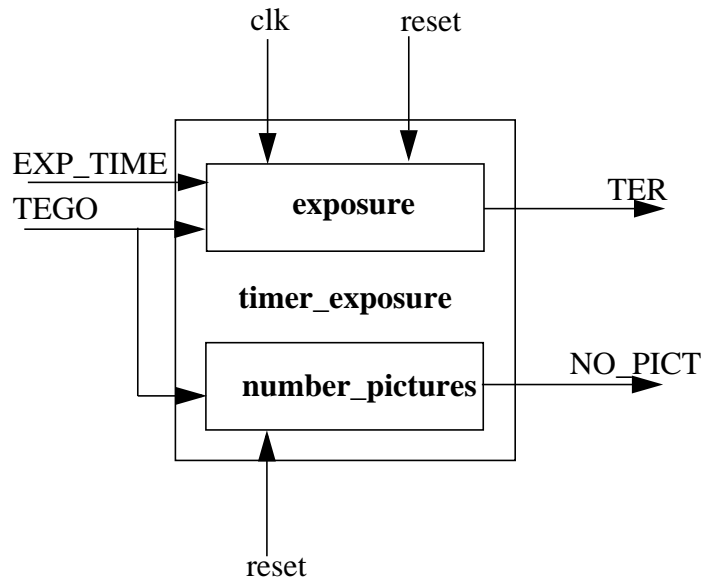


Figure 12: The counter for the exposure time

- The design consists of two modules. The first is the timer for the exposure time, the second is the picture counter. The exposure timer shall start immediately after TEGO switches to '0'. After the exposure time is expired the output signal TVR has to be set. You hit the same problem as in the previous exercise: the output signal TVR has to be set to '1' for one clock cycle only. The counter can be reset to zero (reset input). When the counter has reached its final value it has to be set to zero, too.
- The picture counter shall be incremented by one with the falling edge of TEGO. The reset signal shall reset the picture counter to zero.

### 8.3 HINTS

- Reading of output signals: It is not possible in VHDL to assign values to signals with the mode OUT. In this case you have to define a temporary signal/variable which can be read within the unit and which is assigned to the output signal concurrently.
- The output signal of the picture counter shall be of type T\_ANZEIGE. This counter is described with an other algorithm. The counting variable/signal shall also be of the type T\_ANZEIGE. You can define variables for the different digits of T\_ANZEIGE, and build a counter chain. At the end of the process you have to assign your internal variables/signals to the output signals.

### 8.4 GIVEN

- Data types:** use INTEGER with RANGE for the counter signal, std\_ulogic and T\_ANZEIGE for the other Signals.
- File names:** use **timer\_exposure.vdh** for the design and **tb\_timer\_exposure.vdh** for the testbench.

- **Names of the design units:** use `TIMER_EXPOSURE` for the entity, `BEHAVE` for the architecture, `TB_TIMER_EXPOSURE` for the testbench, and `CFG_TIMER_EXPOSURE` for the configuration.
- **Package:** use the package described in `p_showdisplay.vhd`.
- **Style:** You need two processes. One for the exposure time and one for the picture counter. Note that the input signal is of type `T_ANZEIGE`. You have to convert the signal to an integer. Design an algorithm for the exposure timer.

### 8.5 TO DO:

- Write both modules and a testbench.
- Simulate your design and verify its function.
- How many flip flops are required by the counter?
- Synthesize your timer and optimize it for minimum area size. Verify your answer regarding the flip flops.

## 9. Exercise: A State Machine for the Camera Control

### 9.1 GOAL

- The design of the state machine for the camera control.

### 9.2 DESCRIPTION

- All internal signals are generated by the camera control.

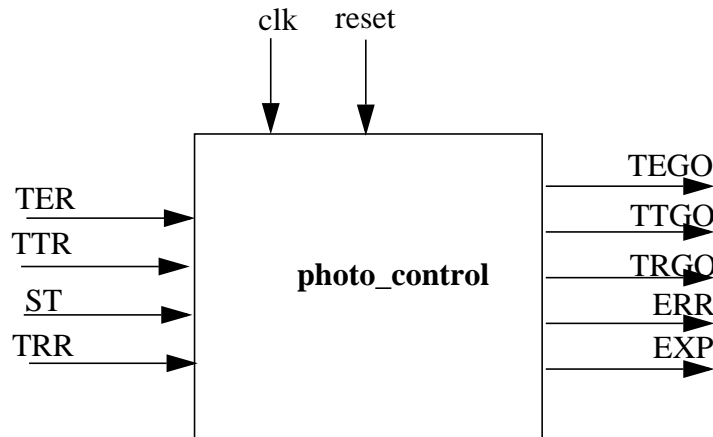


Figure 13: The camera control

- The inputs are:**

TER: Timer for the exposure time has expired (Signal is high for one clock cycle)

TTR: Timer for the maximum transport time has expired (Signal is high for one clock cycle)

ST: The Shutter release (active = high)

TRR: Film successfully transported (Signal is high for one clock cycle)

- The outputs (control signals) are:**

TEGO: Start timer for the exposure time (Signal is low for one clock cycle)

TTGO: Start timer for the maximum transport time (Signal is low for one clock cycle)

TRGO: Servo (on = high, off = low)

ERR: error display (on = high, off = low)

EXP: exposure (open = high, close = low)

- The camera control shall open the shutter after pressing the shutter release, wait for the selected exposure time, and finally close the shutter and transport the film concurrently.
- Keeping the shutter release pressed shall result in a picture sequence.
- Successful transportation of the film is acknowledged by the synchronous signal TRR (transport ready, high for one clock cycle).
- At the end of the film or if the film has become stuck, the servo shall stop after two seconds and an error shall be displayed. Pressing the shutter release after an error has occurred (e.g. if the film has been changed) results in a new attempt to transport the film and a reseted display.
- The maximum transport time is modeled by the **timer\_transport** module.
- The exposure time is modeled by the **timer\_exposure** module.
- The two modules **timer\_transport** and **timer\_exposure** are started by the signals TTGO and TEGO respectively. If a timer expires it sets the signals TTR or TER to high for one clock cycle, respectively.
- The camera control shall be implemented as a Moore machine.

### 9.3 HINTS

- **State machine:** There are different kinds of state machines (Mealy/Moore), and several ways to implement them. A good implementation consists of one process for the state transitions including the calculation of the following state, and a second process for the calculation of the output. If the complete state machine is described within one process sensitive to clock and reset then flip flops are not only generated to store the state, but also for the output signals. The output signals are then updated one clock cycle later than specified.

For a good implementation in VHDL one defines a `state_type` defining all possible states. With a signal of the type `state_type` and a case statement the actions corresponding to a certain state are executed properly. In this exercise you will write two processes, one of them containing the transition logic.

The camera control has some don't care conditions. For the synthesis '-' (don't cares) have to be described explicitly. One uses an AND operation to mask out the don't care bits, e.g.

```
IF (input AND "1000") = "0000" THEN ...
```

does check input for ("0---").

The other process implements the calculation of the output by means of the SELECT statement ("|" is an OR operation), e.g.:

```
WITH state SELECT
    TVGO <=
        '0' WHEN ...
        '1' WHEN state2 | state3 | ....
```

## 9.4 GIVEN

- **Data types:** use `std_ulogic` and `std_ulogic_vector` for your signals. Define your own type called e.g. `state_type`. This type contains the allowed states of the machine. Define a signal state of your type.
- **Names:** Choose your own names, use selfexplaining names.

## 9.5 TO DO:

- Write the state machine and the testbench.
- Simulate your design and verify its function.
- How many flip flops does your design require?
- Synthesize your design and optimize it for minimum area size. Verify your answer regarding the flip flops.

## 10. Exercise: A State Machine for the Display

### 10.1 GOAL

- A design of the state machine for the display.

### 10.2 DESCRIPTION

- The schematic of the display controller.

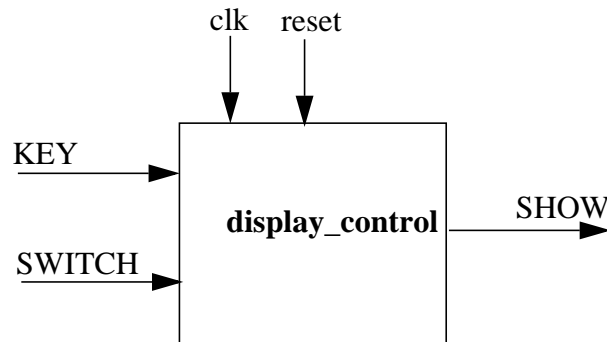


Figure 14: The display controller

- **The inputs are:**

**KEY:** A button to set the exposure time was pressed

**SWITCH:** The button to switch the display from number of pictures to exposure time and vice versa.

**The output (control signal) is:**

**SHOW:** If SHOW = '0' then display the number of pictures else if SHOW = '1' then show the exposure time.

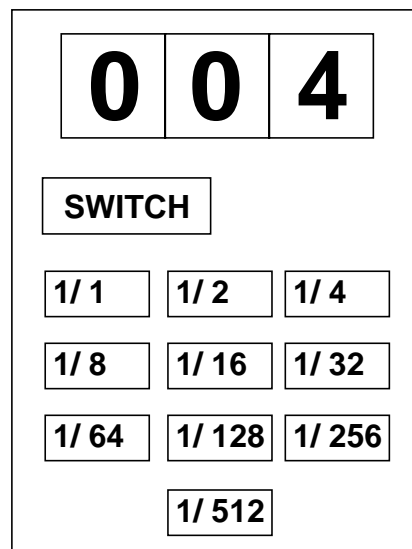


Figure 15: The input control unit

- If one of the exposure time selection buttons is pressed, the display has to show the exposure time immediately.
- If SWITCH is pressed the display switches between the number of pictures and the exposure time.
- The state machine shall be a Moore machine.

### 10.3 GIVEN

- Since you have already written a state machine in the previous exercise, you are on your own

this time ;-)

**10.4 TO DO:**

- Write the state machine and the testbench.
- Simulate your design and verify its funktion.
- How many flip flops does your design require?
- Synthesize your design and optimize it for minimum area size. Verify your answer regarding the flip flops.

## 11. Exercise: The Camera

### 11.1 GOAL

- Merging all modules into one design, the complete camera control.

### 11.2 DESCRIPTION

- All you have to do is to connect all your modules and to simulate the whole system.

### 11.3 HINTS

- Structural modelling: structural modelling means the use (instantiation) and wiring of components resulting in a net list. VHDL provides the following means for structural modelling:
  - Component declaration
  - Component instantiation
  - Component configuration
- Use this means for your camera control.

### 11.4 GIVEN

- You are on your own again. The following figure shows the camera control, but most of the signal names are missing. Draw your own figure and label all signals.

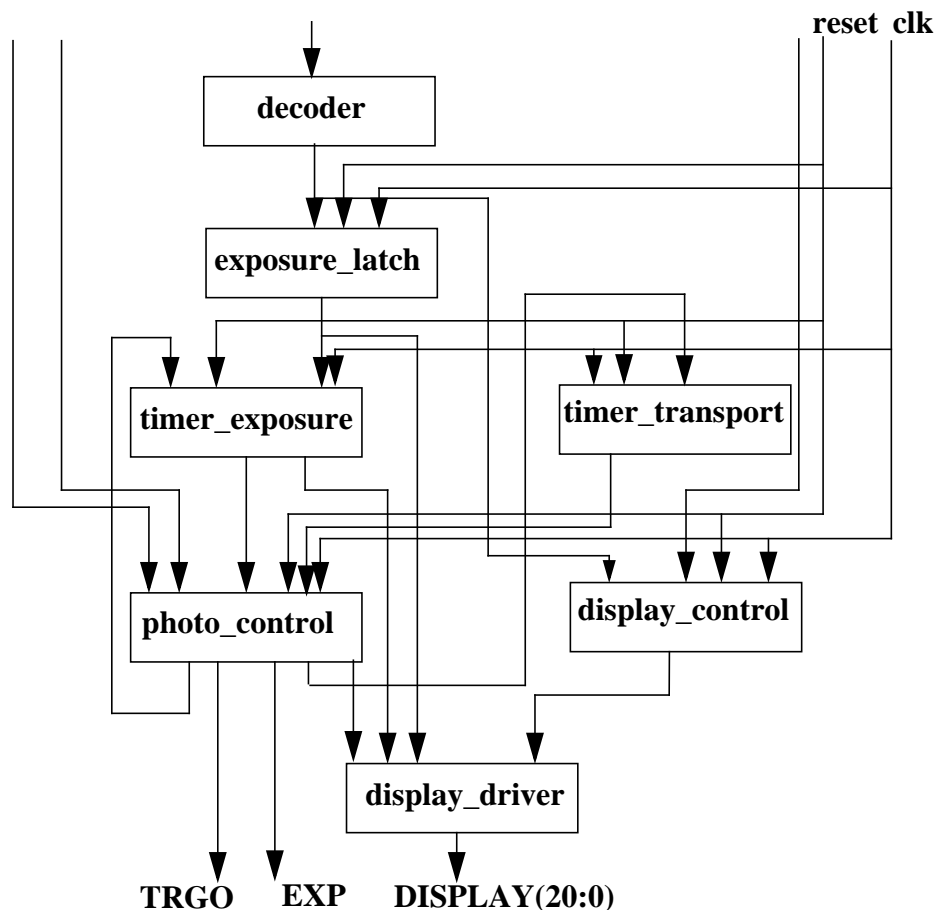


Figure 16: The complete camera control

### 11.5 TO DO:

- Combine your modules. Simulate and synthesize the camera control.

- Write down the area size and the number of gates of the whole design.
- Is the result equal to the sum of the area/gates of the modules?  
(The synthesis tool could optimize the whole desing better and thus achive better results).
- Write down the area and the delays.
- Select all modules on the top level and flatten the hierarchy (Edit -> Ungroup (all Levels)).
- Save the result of the synthesis as a VHDL file (photo\_gates.vhd) for the simulation on the gate level (next exercise).

## 12. Exercise: Simulation on Gate Level

### 12.1 GOAL

- The camera control has to be verified on the gate level.

### 12.2 DESCRIPTION

- The result of the synthesis has to be verified, i.e. it has to be compared if the gate level description has the same behaviour as the original design. To do so you have to analyze the gate level description and simulate it using the old testbench.

### 12.3 HINTS

- For a simulation on gate level you need an appropriate library. In this exercise the FullTimingGateSimulation (FTGS) library of the lsi\_10k library will be used.
- A gate level simulation after the RTL Synthesis is also called 'pre-layout simulation'. See also Figure 17:

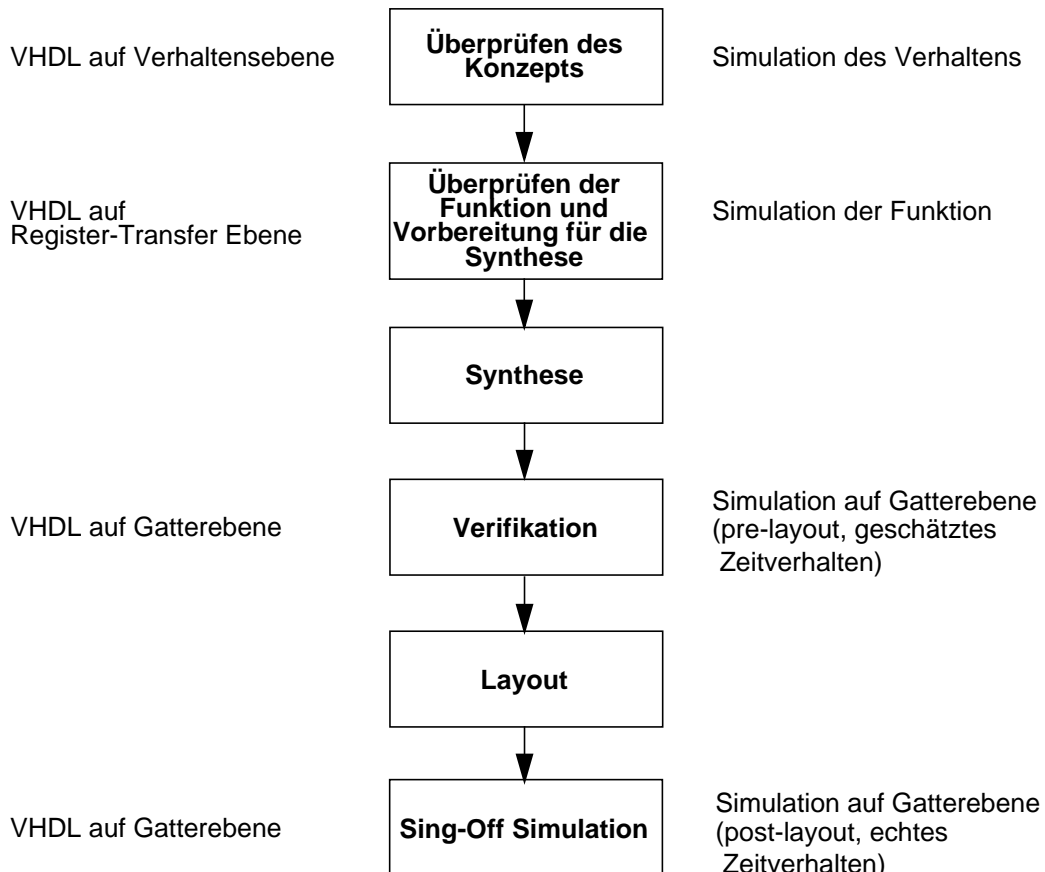


Figure 17: The Placement of the gate level simulation within all design steps.

### 12.4 GIVEN

- Again, you are on your own.

### 12.5 TO DO:

- Delete the attribute and type declarations from the package header CONV\_PACK\_photo in the gate net list (photo\_gates.vhd).
- Verify the functionality of the gate level description.

## 13. Exercise: Synthesis with Objective Functions

### 13.1 GOAL

- The camera control shall be synthesized and verified under consideration of different objective functions.

### 13.2 DESCRIPTION

- In this exercise you will define different constraints for the synthesis. After the synthesis you will simulate your design again and you will notice differences in the timing.

### 13.3 HINTS

- In order to get the best synthesis result you have to take the environment of the camera control into consideration. I.e. not only area, timing and power consumption constraints, but also working conditions have to be defined before the synthesis.
- In general Synopsys provides the following classes of attributes:
  - *Clocks* -> to mark a clock signal.
  - *Operating Environment* -> characterisation of the environment.
  - *Optimization Constraints* -> definition of constraints.
  - *Optimization Directives* -> additional attributes.
- A FullTimingGateSimulation (FTGS) library has default values for the different constants. In order to overwrite these with the current values from the synthesis you have to write them into a file. The format of this file is the Standard Delay Format, SDF. The saved file can be read before the synthesis.

### 13.4 GIVEN

- This is the final exercise, of course you are on your own here. Read the whole design, configure the attributes and simulate on the gate level with respect to the new delays.

### 13.5 TO DO:

- Read the whole design into the Design-Analyzer.
- Flatten the hierarchy (Edit -> Ungroup (all Levels)).
- Set the attributes to reasonable values: ( Menu *Attributes* ->)
  - Clock -> Period, Dont Touch Network, Skew and Uncertainty (the clock has to be selected in advance!)
  - Operating Environment -> Operating Conditions (The top entity has to be selected in advance!)
  - Operating Environment -> Wire Load (with *report\_lib lsi\_10k* in Setup -> Commanda-Window you can obtain information about wire loads)
  - Operating Environment -> Timing Ranges are not defined for the *lsi\_10k* library.
  - Optimization Constraints -> Design Constraints -> Max Area, Max Fanout (for all outputs).
  - Optimization Constraints -> Timing Constraints (here not necessary)
  - Optimization Directives -> Design (top entity!) -> Ungroup, Boundary Optimization, Flip Flop, Port is Pad, Design Pad Attributes (e.g. 2,3,0,5), Flatten Logic (Effort, Flatten Minimize, Flatten Phase), Structure Logic (both).
- Additional settings for the inputs (select!) of the top level entity:
  - Operating Environment -> Input Delay (e.g. 2 ns off the clock signal each)
  - Operating Environment -> Drive Strength (e.g. 0.02; the bigger the value the more power-

ful the external drivers will become; 0 is the default).

- Additional settings for the outputs (select!) of the top level entity:
  - Operating Environment -> Output Delay (e.g. 2 ns off the clock signal each)
- Select all modules on the top level and flatten the hierarchy (Edit -> Ungroup (all Levels)).
- Synthesize the design.
- Compare the area size and the delays with the results of the previous synthesis.
- Save the result of the synthesis as a VHDL file (photo\_gates.vhd) for the following simulation on the gate level.
- Save the timing information in a SDF file (File -> Save Info -> Design Timing -> photo\_gates.sdf) for the following gate level simulation.
- Remove the attribute and type declarations from the CONV\_PACK\_photo package header in the net list (photo\_gates.vhd).
- Simulate once more.
- ***That's it! Congratulations! You have designed, simulated and synthesized your first VHDL design!***