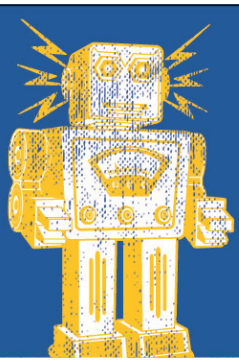
Elektronik



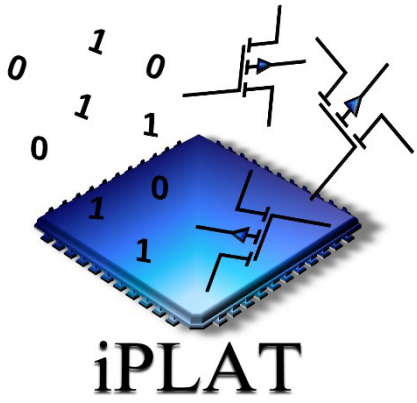
Digital System Design

Distance Learning Letter

VHDL: Entity, Architecture, Libraries

Gefördert von
MA23
Wirtschaft, Arbeit Statistik
StaDt+Wien

iPLAT – Competence Team for Innovative Platforms for Mixed Hardware/Software Systems
MA23 Project 18-06



Version: 0.5
Author: R. Höller, A. Puhm, C. Reisner

Copyright Notice

Introduction

This document will give you a short introduction to VHDL. Furthermore, it will examine the most important VHDL syntax and the library concept. With the help of simple examples, the basic concepts of VHDL will be explained. A small exercise using a VHDL simulator, ModelSim, concludes this document.

Introduction to VHDL

The abbreviation VHDL stands for Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (HDL). Even though it was developed in the United States, this language is mostly used in the European design community. There are other languages used for the description of digital systems, but only Verilog has about the same significance as VHDL. You will frequently encounter Verilog when designing ASICs (Application Specific Integrated Circuit).

History

During the early 1970s, there have been discussions about defining a hardware description language. The US Department of Defense, together with companies such as IBM or Texas Instruments, worked on the development of VHDL, which led to the IEEE Standard in 1987 (IEEE 1076-1987).
The EDA (Electronic Design Automation) tool support for VHDL quickly increased. A revision led to a new standard which was released in 1993 (IEEE 1076-1993). This standard is supported by virtually every EDA program today. This course will not go into detail about the differences between VHDL revisions (1987, 1993, 2002 or 2008).

Applications

VHDL is primarily used for the development of digital integrated circuits, of soft cores and of simulation models.

Development of Digital Integrated Circuits

The development of digital integrated circuits was revolutionized by the introduction of hardware description languages. The consistent and easy-to-read descriptions allowed engineers to design circuits with millions of gates. The possibility to verify the circuits with a testbench written in the same language as the design itself and to automate most of the steps in the design flow allowed the tremendous advancements in computer technology of the last 30 years.

Development of Soft Cores/Design Reuse

The growing complexity of today's circuits makes it impossible to design the whole chip from scratch. Therefore, parts of the chip are reused from older designs (design reuse) or bought from other companies as so-called soft cores or IP (Intellectual Property) cores. This is possible because VHDL has the ability to create configurable blocks (`GENERICs` or `CONSTANTs`). With this feature, it is possible to configure e.g. the word length of ALUs, UARTs or buses.

Development of Simulation Models

For verification purposes, it is often necessary to describe the behavior of a hardware unit (e.g. RAM). This way, the system can be verified before the first (cost intensive) prototype is manufactured. Furthermore, it is possible to describe the behavior of hardware units that cannot be designed with VHDL (e.g. ADC). This can also be done with C or C++ with the help of a PLI (Programming Language Interface).

VHDL Entity and Architecture

To get a valid and synthesizable hardware description that can also be simulated, two VHDL constructs have to be defined: entity and architecture. These constructs can be described in a single file or in two separate files.

Entity

The entity defines the external view of a VHDL module. It is similar to a symbol of an electronic circuit in a schematic. The name of the module, its inputs and outputs as well as some (optional) parameters (`GENERICs`) are declared here. The entity does not define the functional behavior of the module, with the exception of the port directions. Figure 1 shows the entity of a binary half adder as well as a diagram. The parts that are described in the entity are printed in red.

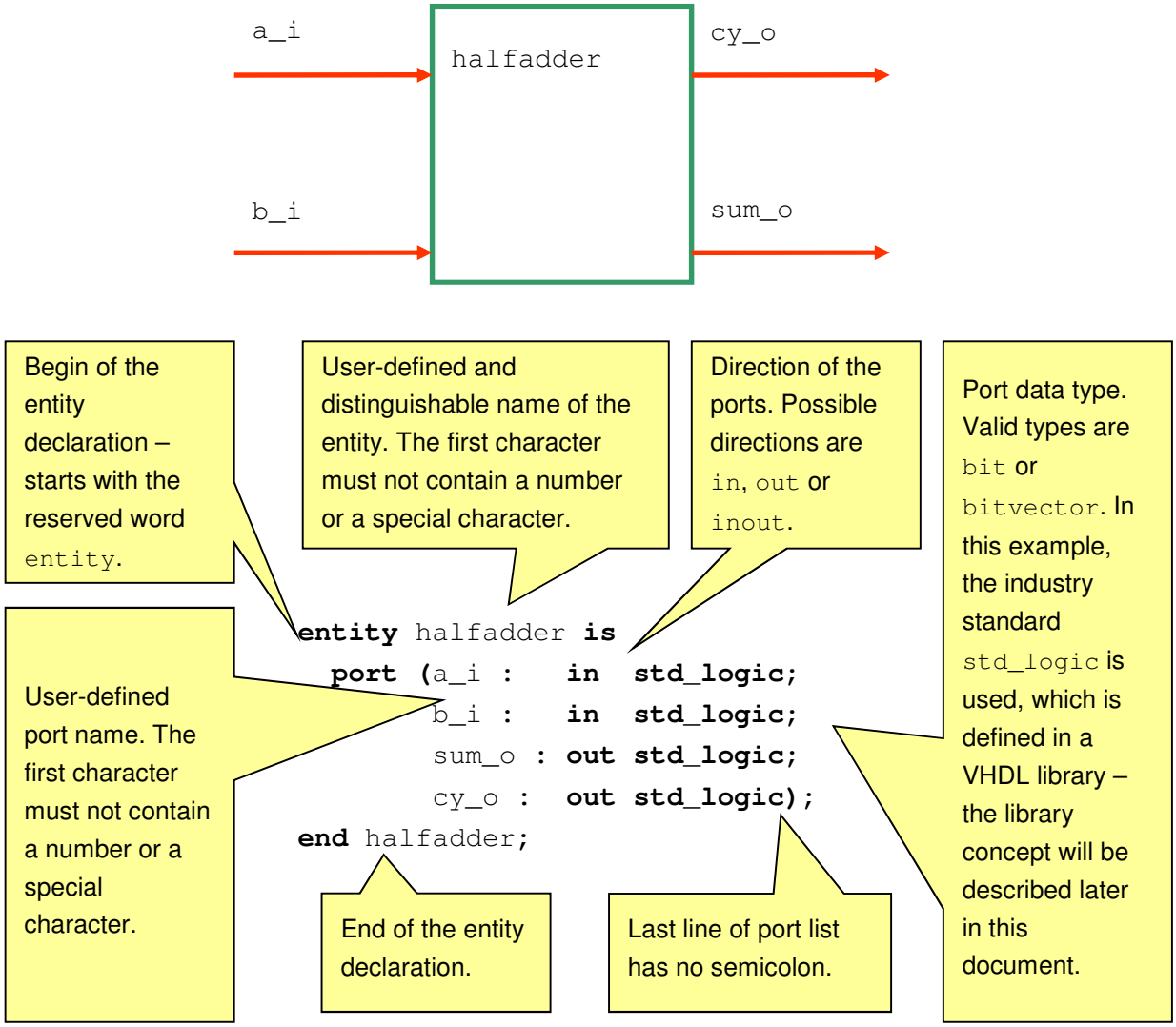


Figure 1: Entity of a binary half adder

As described before, it is also possible to parameterize VHDL modules. The widths of ports as well as the function of the module can be configured. This can be achieved with the usage of GENERICs. Figure 2 shows an example of how the data word width can be configured with the help of a GENERIC parameter.

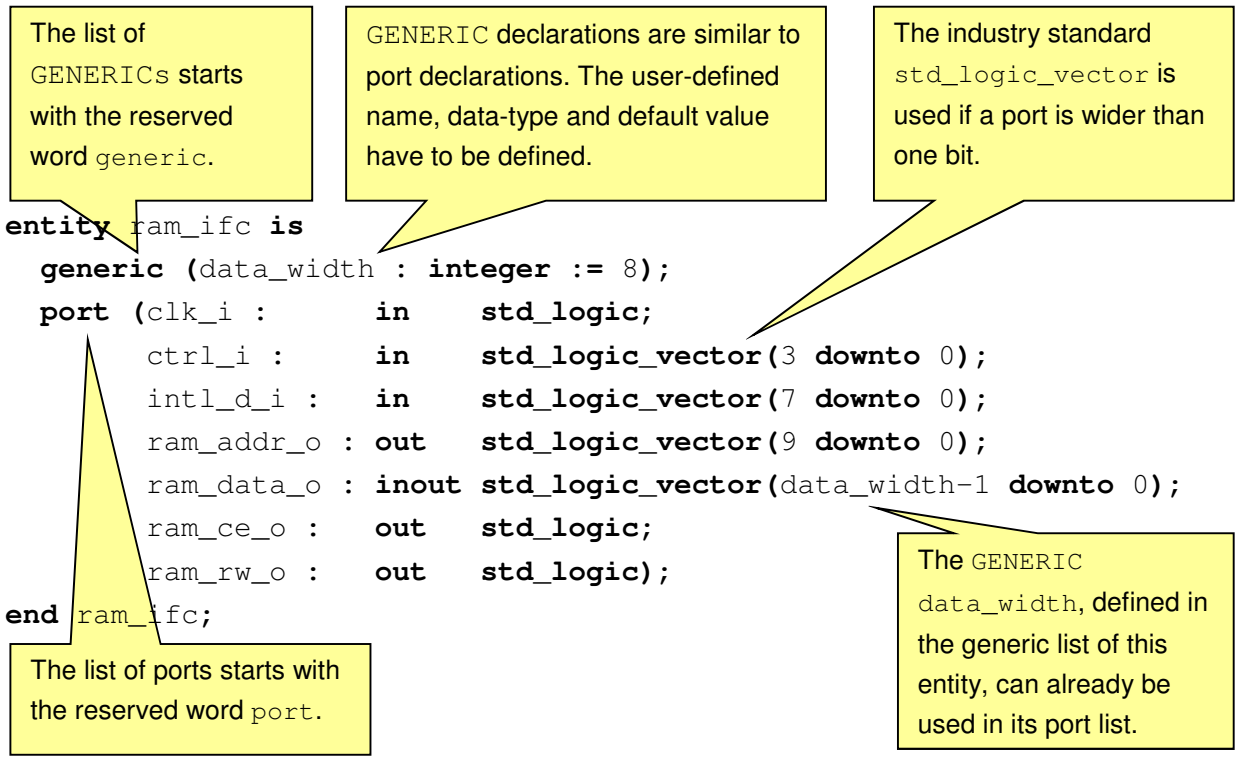


Figure 2: Entity with a GENERIC

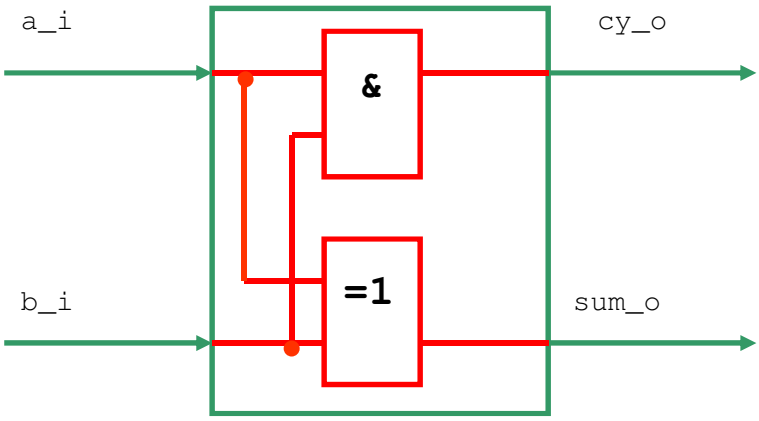
Figure 3 shows the same entity declaration without the optional generic list.

```
entity ram_ifc is
  port (clk_i :      in    std_logic;
        ctrl_i :     in    std_logic_vector(3 downto 0);
        intl_d_i :   in    std_logic_vector(7 downto 0);
        ram_addr_o : out   std_logic_vector(9 downto 0);
        ram_data_o : inout std_logic_vector(7 downto 0);
        ram_ce_o :   out   std_logic;
        ram_rw_o :   out   std_logic);
end ram_ifc;
```

Figure 3: Entity without a GENERIC

Architecture

The architecture describes the behavior as well as the function of the component. The description can be done on different levels. For example, a behavioral description will define the function of the module on a very abstract level. This kind of description can e.g. use time delays to model a VHDL module. Such a system cannot be synthesized – it is not possible to manufacture a chip that is described in this way. This description is usually used for verification (testbench). On the other hand, a description on RTL (Register Transfer Level) is fully synthesizable. The allowed VHDL syntax for synthesizable code is standardized. Figure 4 shows the architecture of a half adder. Its entity was shown in Figure 1. The parts that are described in the architecture are printed in red.
It is usually not feasible to describe a whole digital system in only a single entity and architecture.



Begin of the architecture – starts with the reserved word `architecture`.

User-defined name of the architecture. Meaningful names are e.g. rtl, struc or sim.

Name of the corresponding entity. An entity can have multiple architectures.

The reserved keyword `begin` divides the architecture into two parts. Signals or components are declared before the begin statement (More details follow later).

```
architecture rtl of halfadder is
begin
   sum_o <= a_i xor b_i;
   cy_o  <= a_i and b_i;
end rtl;
```

The actual function of the circuit is described after the keyword `begin`.

The reserved keyword `end` followed by the name of the architecture marks its end.

Figure 4: Architecture

This is why VHDL offers the possibility to divide a system into smaller subunits (each with their separate architectures and entities). These subunits are then instantiated and interconnected in a separate entity and architecture which is called the top-level module. This is called a hierarchical design. Figure 5 shows how this is done in VHDL.

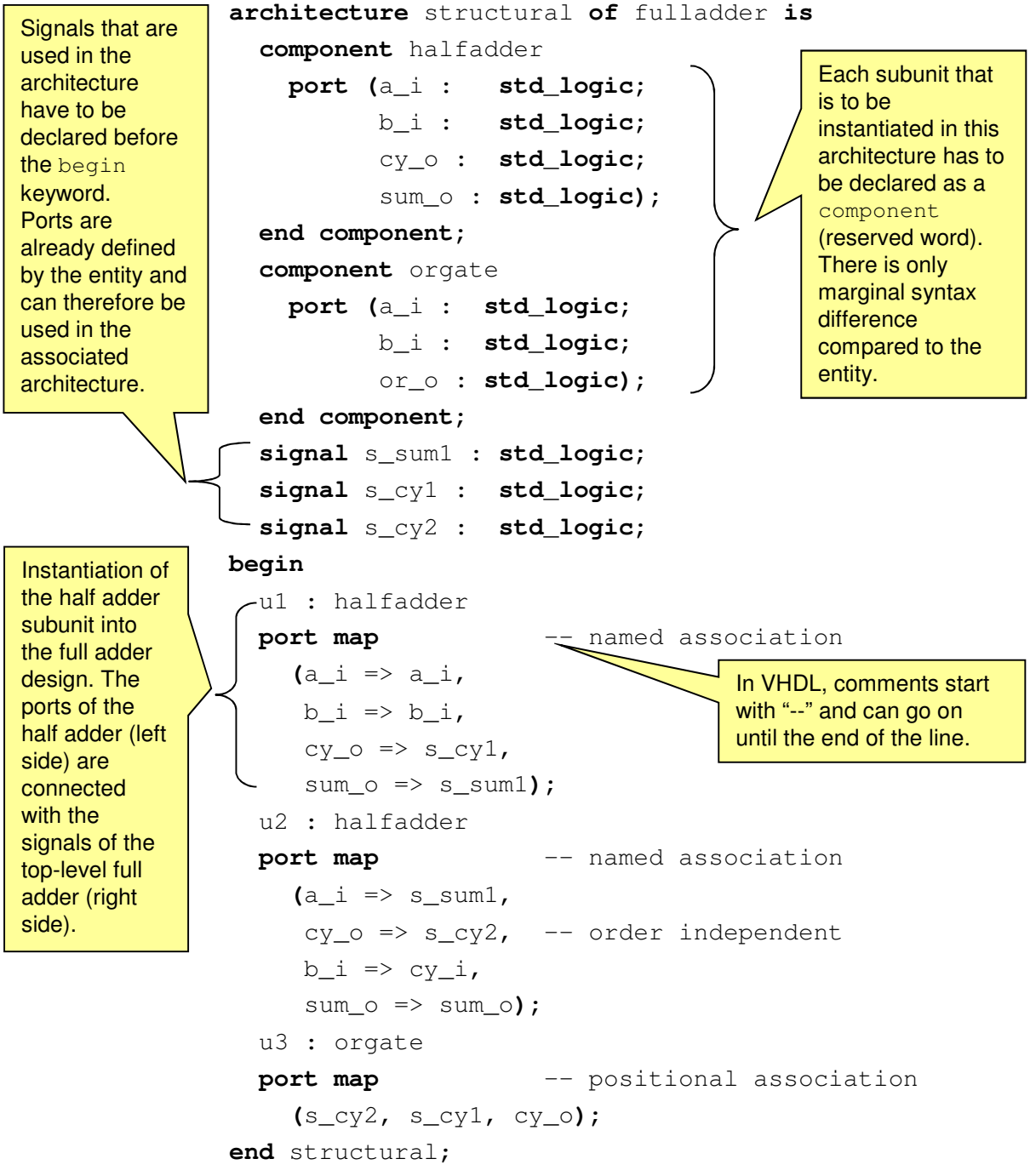


Figure 5: Top-level module

Please note that port names in entity and component declaration must be identical! When instantiating subunits, it might help to think of physically wiring of components on a PCB (Printed

Circuit Board), since in reality, the signals and connections used in VHDL get translated into copper or aluminum wires. There are two different ways of instantiation in VHDL: “named association” and “positional association”. Generally, because of its better readability, only the “named association” is used.
As mentioned before, it is possible to have more than one architecture for a single entity. To decide which architecture will be used, a VHDL configuration is necessary. Since there is only one entity and architecture in the example code shown in Figure 5, a configuration is not necessary. More information on this subject can be found in the lecture slides.

VHDL Libraries

Just like programming languages (C, C++, Java ...), VHDL uses libraries to manage already compiled units. Figure 6 depicts this principle. First, a VHDL file (containing entity and architecture) is compiled (e.g. with the `vcom` command in ModelSim). Then, the file is typically associated with the library `work`, which has to be available for every simulation. Finally, the compiled design can be accessed via this library during the simulation.

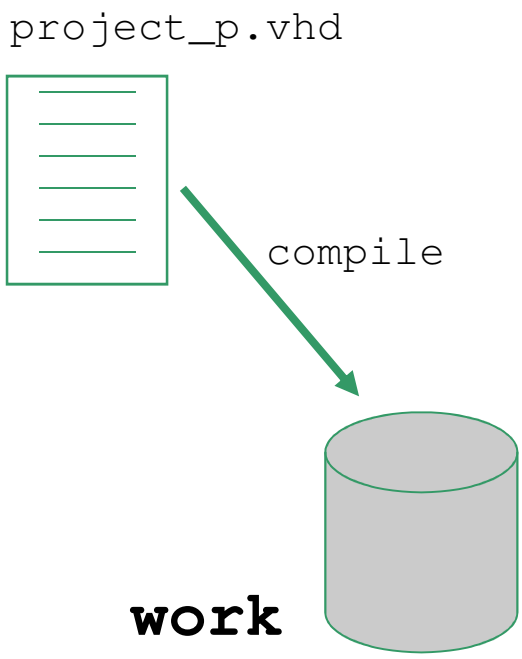


Figure 6: Principle of libraries

Of course, there are commonly used libraries which provide special data types as well as VHDL functions. In the previous entity declarations, it has been assumed that the library `IEEE` and the package `std_logic_1164` were included. This library allows to use the industry standard data types `std_logic` and `std_logic_vector`. The example in Figure 7 shows how to include two commonly used packages from the IEEE library. Finally, the fact that the library `work` and the package `standard` are both automatically included in every entity/architecture may contribute to the understanding of the VHDL library concept.

```
library work;
use work.project_p.all;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

Figure 7: Use of libraries

Abbreviations

ADC	Analog-to-Digital Converter
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
EDA	Electronic Design Automation
HDL	Hardware Description Language
IP	Intellectual Property
IEEE	Institute of Electrical and Electronics Engineers
PCB	Printed Circuit Board
PLI	Programming Language Interface
RAM	Random Access Memory
RTL	Register Transfer Level
UART	Universal Asynchronous Receiver Transmitter
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

References

There are numerous websites on this topic, e.g http://www.vhdl-online.de/.

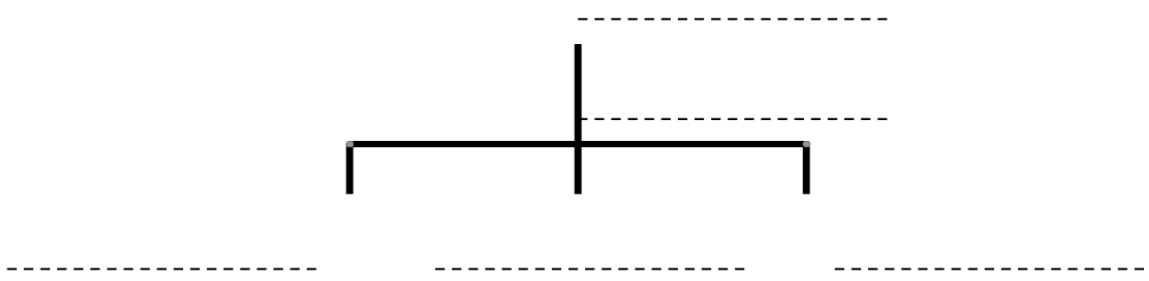
The library of UAS Technikum Vienna has several books about VHDL in English and German. Worth noting are the following:

- Digital Design and Modeling with VHDL and Synthesis, K. C. Chang. Los Alamitos, CA, USA. Wiley – IEEE Computer Soc. Pr., 1997. ISBN 0-8186-7716-3
- Essential VHDL. RTL Synthesis Done Right, Sundar Rajan. USA, 1998. ISBN 0-966959-0-0
- HDL Programming Fundamentals. VHDL and Verilog, N. M. Botros. Hingham, MA, USA. Da Vinci Engineering Press, 2006. ISBN 1-58450-855-8.

Questions

1. Examine the files of the Full Adder VHDL example from the directory `design`. Can you identify the structure as well as the function of the circuit?
2. Open the files found in the `tb` folder of the Full Adder example which is located in the directory `design`. Can you identify the structure as well as the function of these files?

3. Insert the hierarchy of the Full Adder VHDL example from the directory `design` into the figure shown below. Where in ModelSim can you see the hierarchy of the loaded design?



4. When was VHDL standardized for the first time?
5. Which construct describes the external view (inputs and outputs) of a VHDL module?
6. Which construct describes the internal function of a VHDL module?
7. Does VHDL use a library concept? If it does, how does it work?
8. Where in an architecture are signals declared?
9. Where in an architecture are hierarchical subunits declared? Where are they instantiated?
10. Which ways of instantiation are there? Which one is used primarily and why?

Version

Version 0.1, 2006-08-21	Dokument angelegt.
Version 0.2, 2011-06-03	Document translated and updated.
Version 0.3, 2015-08-15	New template.
Version 0.4, 2016-02-22	Document updated.
Version 0.5, 2016-08-17	Proofreading.

If you find errors or inconsistencies, please report them to the supervisors of this lecture via e-mail. Thank you!