FH University of Applied Sciences TECHNIKUM WIEN

Chip Design 1

Verification and IC Design Flow

Version: 1.0
Author: R. Höller

Copyright Notice

Part 1: Verification

Introduction

This Distance Learning Letter examines the principle of a testbench as well as the closely related problem of verifying circuit designs.

Testbench

The term testbench refers to the code (VHDL or Verilog) that is used to drive the signals of a HDL design. These simulated signals are also called stimuli. In addition to generating the stimuli, the testbench can also be used to record and/or verify the output of the DUV (Design Under Verification). The testbench mostly consists of VHDL or Verilog files. However, due to the rising complexity of modern designs, text files for stimuli generation or checking and even C routines are increasingly used as a testbenches.
Figure 1 depicts the principle of a testbench and its interaction with the DUV.

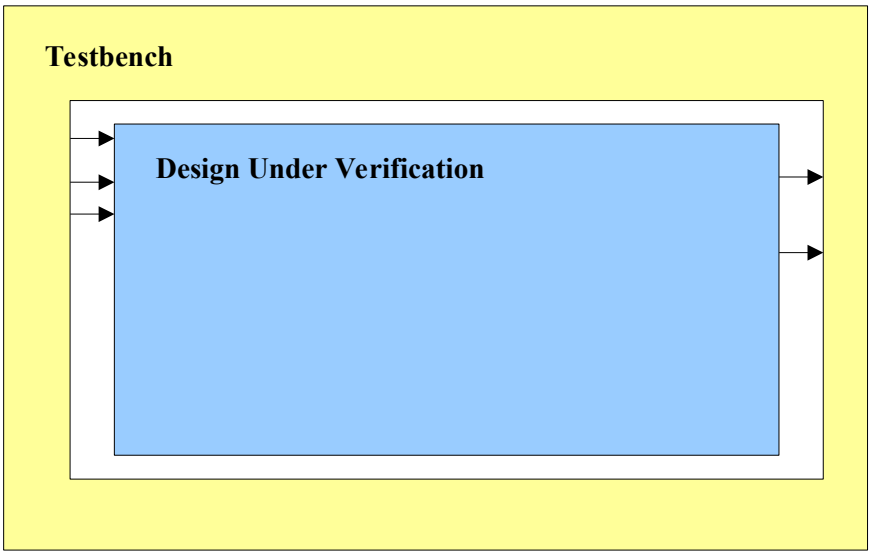

Figure 1: Principle of a testbench

The testbench generates all input signals and monitors all output signals of the DUV. The test environment is a closed system since the testbench has no in- or outputs and represents the whole environment of the DUV. The challenge of writing a testbench is to decide what input stimuli to generate as well as what output signals to expect and to consider as correct functional behavior of the DUV.

In industrial practice, the number of lines of code needed for the testbench often exceed the lines of code for the DUV. A proper verification process often takes up about 70 % of the total development time of a HDL design. This fact can be explained by the exploding costs of chip manufacturing. A mask set which is needed for a 90 nm process (today's leading processes use around 16 nm processes with even higher costs) has fixed costs of around a million Euros for a prototype. Therefore, the main aim of verification is to deliver such a highly expensive prototype without any errors.
For cost reasons, prototyping is often done on programmable logic (FPGAs), which can be reprogrammed almost indefinitely and costs about a few thousand euros in case of high-end FPGAs. It is important to mention that although FPGAs can be reprogrammed and tested over and over, a proper design process with appropriate verification effort will give much faster results than a trial-and-error approach.

Verification

The term verification includes all actions and processes whose common goal is to prove the correct functionality of the circuit design. Therefore, verification is more than just a single testbench or even a set of testbenches. Moreover, a desirable verification based on a formal mathematical proof is only possible in few exceptional cases, which is why it is common to set verification goals with regard to e.g. code coverage.

Interpretation and Redundancy

The starting point for the verification processes is the question of what is to be verified and how the result of the verification can be found. Most often, a specification of the intended functionality of the design is written in a natural language and by different people with different communication abilities. Such a document is always open for interpretation.
Figure 2 shows a possible verification process. A single engineer creates and verifies a design based on his/her interpretation of the specification. Since the testbench and the RTL Code is based on the same perception, a wrong interpretation of the specification would not be noticed.

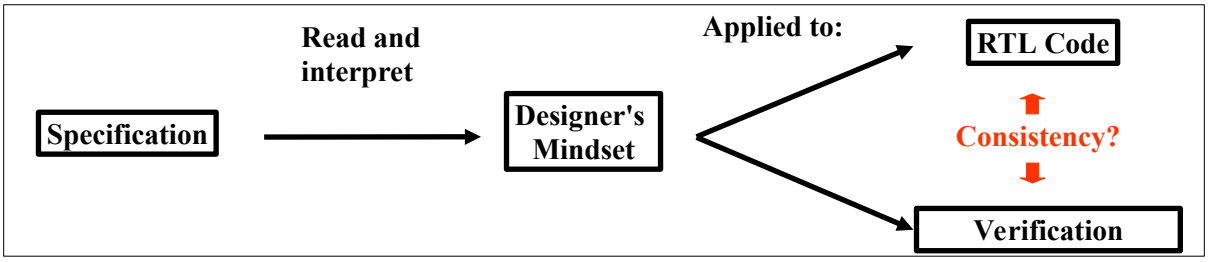

Figure 2: Verification process without redundancy

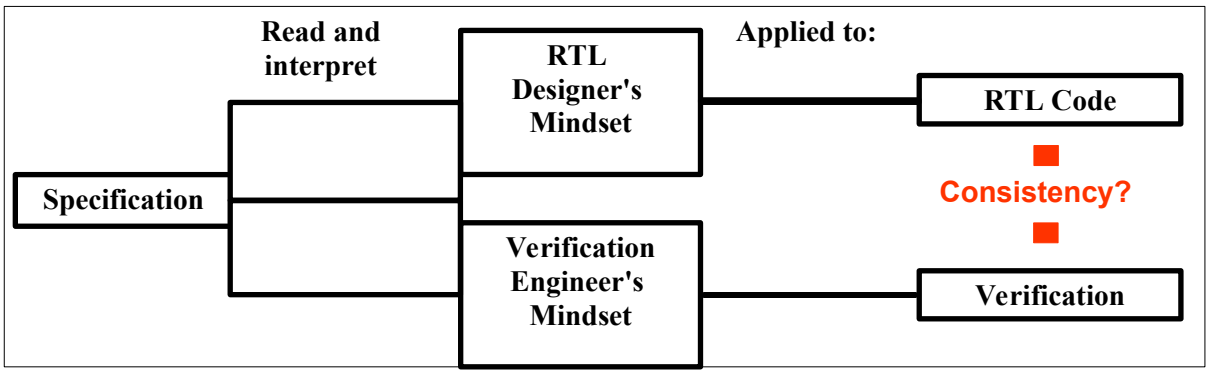

Figure 3: Verification process with redundancy

By introducing some kind of redundancy, it is possible to reduce the risk of such misinterpretation. In this case, the RTL Coding is done by one engineer, and the verification is done by another (or even two different teams of engineers).

Since the engineers separately gain an understanding of the functionality of the specification, it is possible that the verification process of digital integrated circuits will reveal differences in their interpretations. Figure 3 depicts this principle.
As already mentioned, the process shown in Figure 3 can reduce the risk of a wrong interpretation. However, this risk reduction comes with higher costs, especially because of the additional personnel. Moreover, it is important to mention that it is not possible to prove the absence of errors using the approaches mentioned above. Simulation can only prove the existence of errors, but not their absence.

Methods and Metrics

The following chapter describes the most important methods and metrics used in the verification process. The relevant literature might group these terms differently. However, the aim of the chapter is to briefly describe the most important keywords. The connections between the terms are intentionally left out.

Formal Verification

Formal verification is the mathematical proof of the correct functionality of a design. Even state-of-the-art formal verification is only possible for very simple, rather academic examples. In the industry, this form of verification is only used in the form of equivalence checking or model checking (see below). Formal verification needs an executable (computer readable) version of the specification which has to be derived from the original (human readable) specification.

Automation

For complex designs and for safety related designs, it is not possible to determine the correct functional behavior with the help of visual control of the signals and signal transitions in a simulator. It goes without saying that in industrial practice, the complete verification flow is automated (simulation, checking of the results, finding differences to previous versions…). This provides more confidence in the verification and relieves the DAVE (Design and Verification Engineer).

Equivalence Checking

Equivalence checking is a formal verification method. It compares the same design on two different abstraction levels and checks whether they are functionally equivalent. Such tools are mostly used to compare the VHDL code of a designer with the gate-level netlist that is produced by a synthesis tool.

Model Checking

Model checking is a formal verification method. It is used to e.g. determine if there are unreachable states in a state machine.

Automatic Testbench Generation

Some parts of the testbench can be generated automatically, e.g. to cover specific test cases. However, it is still necessary to check if the generated testbench drives the design with the correct stimuli. Some IP Cores include something like a testbench generator. It can generate parts of the testbench as well as simulation models depending on the use of the IP Core.

Black-Box Verification

The functionality of the design is verified without knowledge of the internal structure and of the exact implementation of the design. The verification only has access to the (top-level) interfaces of the design. It has no access to the internal structures. A downside of this approach is that black-box verification has less control over the design due to the lack of knowledge of the internal structures. On the other side, this approach allows the verification to be independent from the implementation.

White-Box Verification

The verification of the design is done with full knowledge of the structure. A downside of this approach is that the verification is implementation dependent, which means that it has to be redone if the implementation changes (e.g. a different FPGA is used). On the other side, it is far easier to identify errors in the code since the internal structure is visible.

Design for Verification

Since the verification of the circuit typically takes more time than its design, it makes sense to write HDL code with verification in mind. For example, a counter can be realized with a loading functionality in order to enable the simulation to set the counter to a specific value. This way, e.g. the overflow of a counter can be simulated elegantly.
It is important to mention that it is an advantage to consider verification issues already during the specification phase.

Reuse

Since it takes a substantial amount of time and effort to create a proper testbench for a specific project, it makes sense to try to use at least parts of it for other projects.
If, for example, a company uses a specific CPU core for most of its products, a simulation model of this core will be needed in more than one development project. Therefore, it makes perfect sense to put more effort into the development of the simulation model in order for it to be easily portable to other projects.

Code Coverage

State-of-the-art simulation tools allow to record if and how often HDL code lines, statements or specific if-branches are executed during the simulation. This is called code coverage. In industrial practice, a code coverage of 100 % is required, which means that every line of HDL code gets executed at least once. Although a certain quality of the testbench is ensured, the existence of errors in the design cannot be ruled out even with 100 % code coverage.

Code Review

During the code review, the finished, but normally not yet simulated HDL code is discussed by the designer, project manager and colleagues. Additionally, an engineer who is not involved in the project should participate. The review covers the code line by line. Such discussion can reveal not only problems and errors in the design, but also in the specification. It is of utmost importance that all participants participate in the code review with an open mindset. Even the best HDL designer makes mistakes. The goal of the code review is not to embarrass the designer, but to find as many errors as possible.

Linting Tool

A linting tool checks the HDL code for specific, user configurable rules. Possible rules could be that all clocks are named “clk”, that every file contains a CVS Header, or that the code is compliant with a specific coding style. Furthermore, problems with the syntax get reported.

Verification during the course of the project

As mentioned before, verification needs to be taken into account at every step of the project flow, starting with the specification. Figure 4 shows an exemplary project flow and the questions regarding verification that are associated with the individual steps.

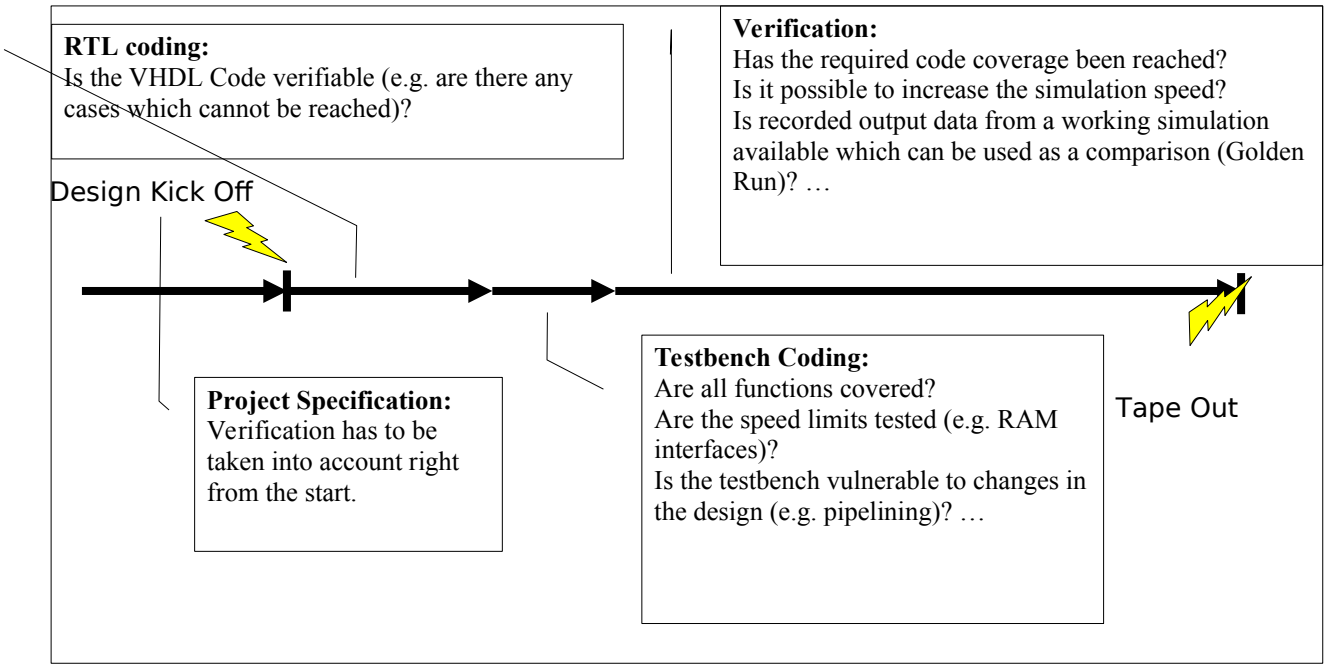

Figure 4: Verification during the course of the project

Abbreviations

CPU	Central Processing Unit
CVS	Concurrent Version System
DAVE	Design and Verification Engineer
DUV	Design Under Verification
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
IP	Intellectual Property
RTL	Register Transfer Level
VHDL	Very High Speed Integrated Circuit Hardware Description Language

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. Use the basic considerations about testbenches and verification that have been presented in this document in order to evaluate your VHDL testbenches. Are there possible improvements?
2. What is a testbench?
3. Why is a thorough verification necessary?
4. What does verification mean in the context of digital system design?
5. What is formal verification?
6. What is equivalence checking?
7. What does automation mean in the context of digital system design verification?
8. What is model checking?
9. What is automatic testbench generation?
10. What is black-box verification?
11. What is white-box verification?
12. What does design reuse mean in the context of digital system design?
13. What is code coverage? What can be measured by it?
14. What is a code review?
15. What is a linting tool?
16. What problems and questions with regard to verification can arise during a typical digital system design project?

Part 2: IC Design Flow Introduction

This Distance Learning Letter focuses on the different steps of a digital system design project. It starts with the project definition and concludes with the implementation in hardware.

Design Flow

At the beginning of a design stands the vision of a new product or a problem that needs to be solved. The first and most important thing is the creation of a specification document. Typically, this document is formulated in a natural language. It is also possible to formulate the specification in an executable way, e.g. in the programming language C. In order to prevent later difficulties, it is absolutely recommended to create clear specifications; otherwise, the project is at risk with respect to external time restrictions, technical constraints and requirements.

The next step is an evaluation of the system by writing simple models and simulating the whole system on an abstract layer (e.g. in VHDL, Verilog, System C, System Verilog or in another programming language). This way, questions regarding the size of the needed memory, clock frequency to be used, controllers or DSPs to be used, etc. can be answered. Often, a more detailed specification (design specification) is created with the output of this evaluation. The coding of the design in the HDL language (VHDL, Verilog) on the Register Transfer Level (RTL) is kept in mind when writing this design specification. Here, the design is also usually separated into different sub-parts to be distributed among the different engineers. In order to verify the design, testbenches and simulators are used. After completing the HDL coding on RTL level, the design is typically verified and at the same time synthesized. Synthesis handles the transformation of the HDL code as written by the designer to a gate-level netlist.

The result of this design step is verified again. In the industrial context, there are tools called equivalence checkers for the comparison of the HDL design and the generated netlist. In the last step, the place-and-route (also called layout), the different gates, storage elements and larger logic blocks are placed on the chip (mapped to similar logic in the FPGA) and connected with metal wires. Again, the design is verified against the chosen clock frequency, and, if necessary, it can be optimized so that the chosen frequency can be used. The output of this process contains the information necessary to produce the chip or to load the FPGA with the design.

The following figure shows the ideal path of the design flow. It is often necessary to run certain steps iteratively as e.g. the design cannot be used for the targeted clock frequency and therefore has to be redesigned. If an error is found late in the design, it will have a much larger impact on the additional workload (and further additional costs) needed to fix it than if it is discovered early in the flow.

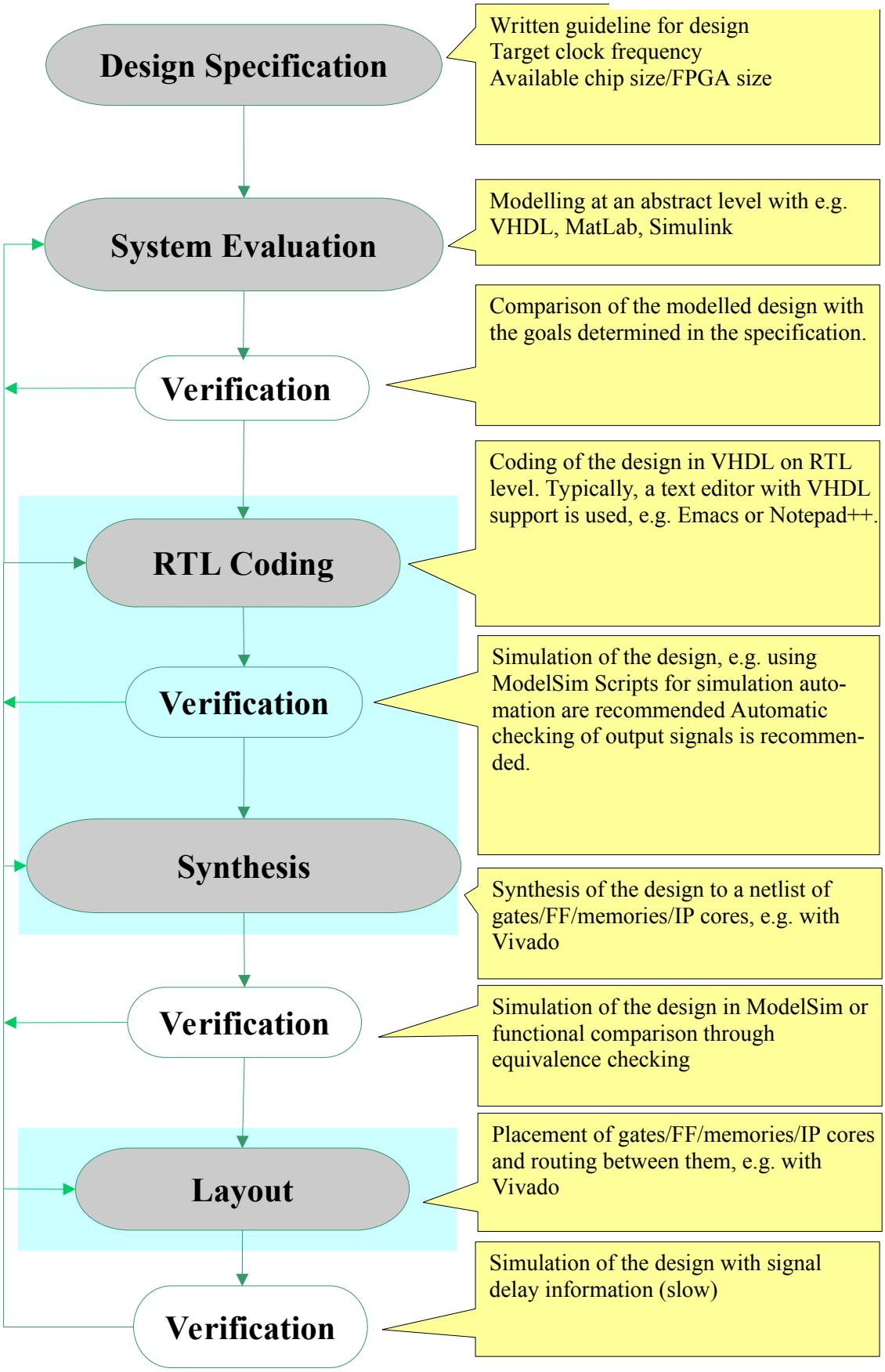

Figure 5: Design flow

The highlighted parts in Figure 5 show the design steps which have been covered in the previous lecture notes or will be covered in the future: VHDL as a hardware description language which is used for modelling respectively description of digital circuits; the ModelSim simulator which is used to simulate the circuit design and the corresponding testbench; and finally synthesis and place-and-route which is done with the help of Vivado.

Abstraction Levels

During the design flow from specification to production data, the design is described on different abstraction levels. The descriptions of these abstraction levels are coded by the designer (behavioral level and RTL) or are generated by CAD/EDA tools (gate level and layout level) as netlists. The following figure gives an overview of how the design is represented on these abstraction levels.

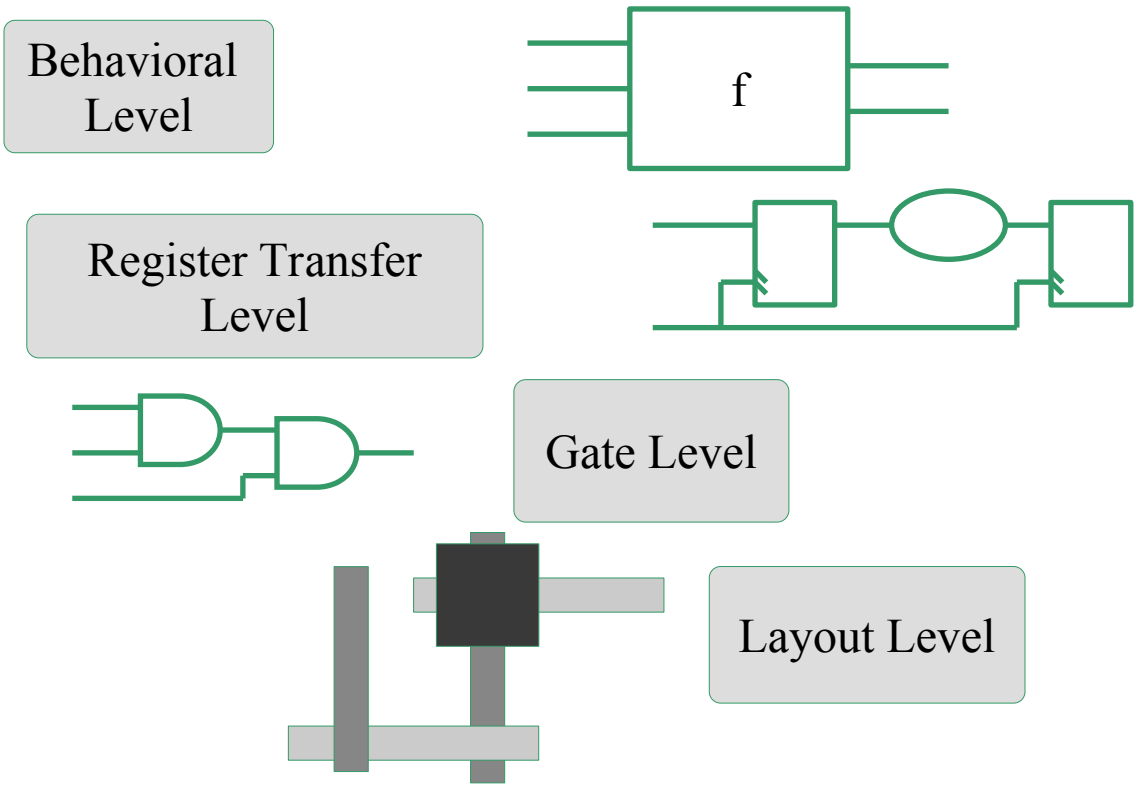

Figure 6: Abstraction levels

On the behavioral level, the details of the implementation are not modelled. The design is characterized as a black box: It is modelled what the design should do, but not how.
On the Register Transfer Level, the design is coded with VHDL or Verilog. The goal is to write the code so it is usable both by the simulator as well as by the synthesis tool.

After synthesis, the VHDL code is transformed to an optimized gate netlist (gate level). The gates used in the netlist depend both on the target process (e.g. 90 nm process from TSMC or UMC) and on the device (Altera FPGA, Xilinx FPGA, …).
On the layout level, all necessary files that are needed for chip manufacturing or FPGA initialization are produced.

Abbreviations

CAD	Computer Aided Design
DSP	Digital Signal Processor
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
RTL	Register Transfer Level
TSMC	Taiwan Semiconductor Manufacturing Company
UMC	United Microelectronics Corporation
VHDL	Very High Speed Integrated Circuit Hardware Description Language

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
- Writing Testbenches. Functional Verification of HDL Models, Janick Bergeron. Norwell, Massachusetts, 2000. ISBN 0-7923-7766-4
- Application-Specific Integrated Circuits (ASICs... the book), Michael John Sebastian Smith. Addison-Wesley, VLSI Systems Series, 1997. ISBN 0-201-50022-1
- The Design Warrior's Guide to FPGAs. Devices, Tools and Flows, C. Maxfield. Burlington, MA, USA, Elsevier 2004. ISBN 0-7506-7604-3
- Reuse methodology manual for system-on-a-chip designs, Michael Keating and Pierre Bricaud. Boston, MA, USA: Kluwer Acad. Publ., 1999. ISBN 0-7923-8175-0

Questions

1. Sketch a typical design flow for a digital system.
2. What abstraction levels are there in the design flow?
3. Which hardware description languages do you know?
4. What does the term synthesis mean in the context of digital system design?
5. What does the term place-and-route, or layout, mean in the context of digital system design?
6. What does the term constraint mean in the context of digital system design?

Version

Version 1.0	Update to entire document for CHIP1

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