



TECHNISCHE  
UNIVERSITÄT  
DRESDEN

Faculty of Computer Science Institute of Computer Engineering, Chair of VLSI Design, Diagnostics & Architecture

# VHDL – BASICS

**Dr. Thomas B. Preußner**

*thomas.preusser@tu-dresden.de*

Dresden, July 1, 2014



DRESDEN  
concept  
für die Zukunft  
von Wissenschaft  
und Kultur

# Goals of this Lecture

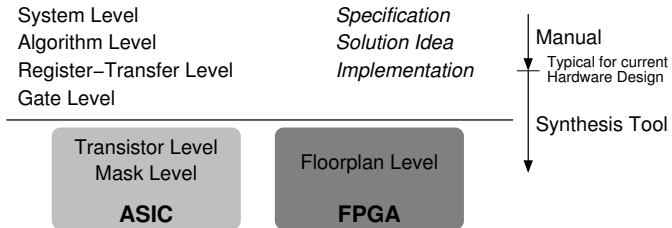
- Answer the questions:
  - Where does VHDL come from?
  - What are its fundamental paradigms?
- Prepare a (concurrent) HelloWorld.

# History

VHDL – Very High Speed Integrated Circuit (VHSIC)  
Hardware Description Language

- 1981 Initiated by US Department of Defense (DoD) for simplifying the reproduction of hardware in new technologies: “Hardware Life Cycle Crisis”
- 1983 Intermetrics, IBM and TI to design an ADA-based HDL.
- 1985 Completion of core VHDL in version 7.2.
- 1986 DoD transfers all rights on VHDL to the IEEE.
- 1987 VHDL becomes IEEE-Standard 1076-1987.
- 1987 DoD requires VHDL models for all purchased ASICs.
- 1988 VHDL becomes ANSI Standard.
- 1993 IEEE-Standard 1076-1993: still most widely used.
- 2008 IEEE-Standard 1076-2008: latest major language revision.

# Abstractions Levels



- VHDL is suitable for descriptions from the system to the gate level.
- The automated (technology-specific) synthesis typically starts at the Register-Transfer Level.
- The synthesis from higher abstraction levels is an active research domain.

# Fundamental Paradigms

## Structure

- Designs are built from *components*.

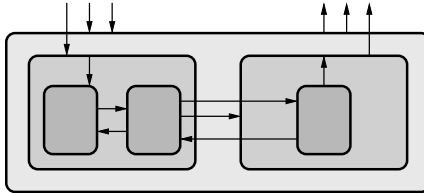
## Concurrency

- Implementations are composed of *concurrent* statements.

# Fundamental Paradigms

## Structure

- Designs are built from *components*.



- typically: wide, bidirectional interfaces (of *wires*).
- Hierarchical instantiation tree of components.
- Wiring within or through instantiating module.

## Concurrency

- Implementations are composed of *concurrent* statements.

# Fundamental Paradigms

## Structure

- Designs are built from *components*.

## Concurrency

- Implementations are composed of *concurrent* statements.
  - Order of statements does *not* matter:

`p <= a xor b;` and `s <= p xor c;` are equivalent!  
`s <= p xor c;` and `p <= a xor b;`

- Dependencies are defined by signal connections.

## Structure through Modules

VHDL was designed after ADA → similarities with Pascal

- Rather wordy syntax.
- Strict separation between interface and implementation:

— *Interface*

```
entity FA is  
  port (  
    a, b, c : in   bit;  
    s, cout : out bit;  
  );  
end FA;
```

— *(An) Implementation*

```
architecture FA_1 of FA is  
begin  
  s      <= a xor b xor c;  
  cout <= (a and b) or (a and c)  
          or (b and c);  
end FA_1;
```

- Both are stored in *one* .vhd1-file.
- Comments start with — and extend to the end of the line.
- VHDL is *case-insensitive*.



# Concurrency through Processes

- A process is a *timed sequential* model of some behavior.
- A process constitutes an implicit infinite loop.
- Processes run concurrently.
- They are synchronized by events.

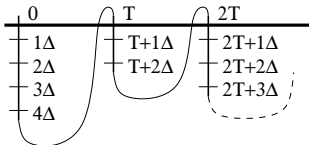
```
process  
begin  
  report "A_says: „Hello „World!" severity note;  
  wait for 5 ns;  
  report "A_says: „Good_bye." severity note;  
  wait; — forever: allow termination  
end process;
```

## Time Model

1. Set simulation time  $t = 0$ .
2. Collect all processes into the set of executable processes.
3. Remove and execute every process from the set of executable processes up to the next **wait** statement. Unless this statement lacks a delay clause, schedule the process to resume execution at:  
$$\begin{array}{ll} t + \Delta & \text{if } \mathbf{wait\ for\ } 0\text{ ns; and} \\ \lfloor t \rfloor + d & \text{if } \mathbf{wait\ after\ } d; \end{array}$$
4. Advance the simulation time up to the time of the next resumption event. If no such event exists, terminate.
5. Collect all processes to resume at the advanced simulation time into the set of executable processes.
6. Goto step 3.

# What are Deltas?

Deltas realize a two-dimensional time model:



Think of them as very small time slice, which:


- never interferes with the order of events on the real axis but
- creates an order of events occurring at the same real time.

**wait for** 0 ns; is *not* a no-op!

# Lab

Introduce a second (concurrent) **process** to `hello.vhdl` and produce different output orders.

# Getting Started with ModelSim

1. **File** → **New** > → **Project...**
2. Project Name: `hello`  
Project Location: `/home/xxx/hello`  
Copy Library Mappings: Reference Library Mappings
3. Add new file `hello.vhdl`, edit, save, and ...
4.  – Compile, Compile All, Simulate, Break.  
*Your design `hello` is compiled to the user library `work` by default.*