VHDL Recapitulation

Henning Puttnies

University of Rostock
Institute of Applied Microelectronics and Computer Engineering

October 12, 2015

Outline

Outline

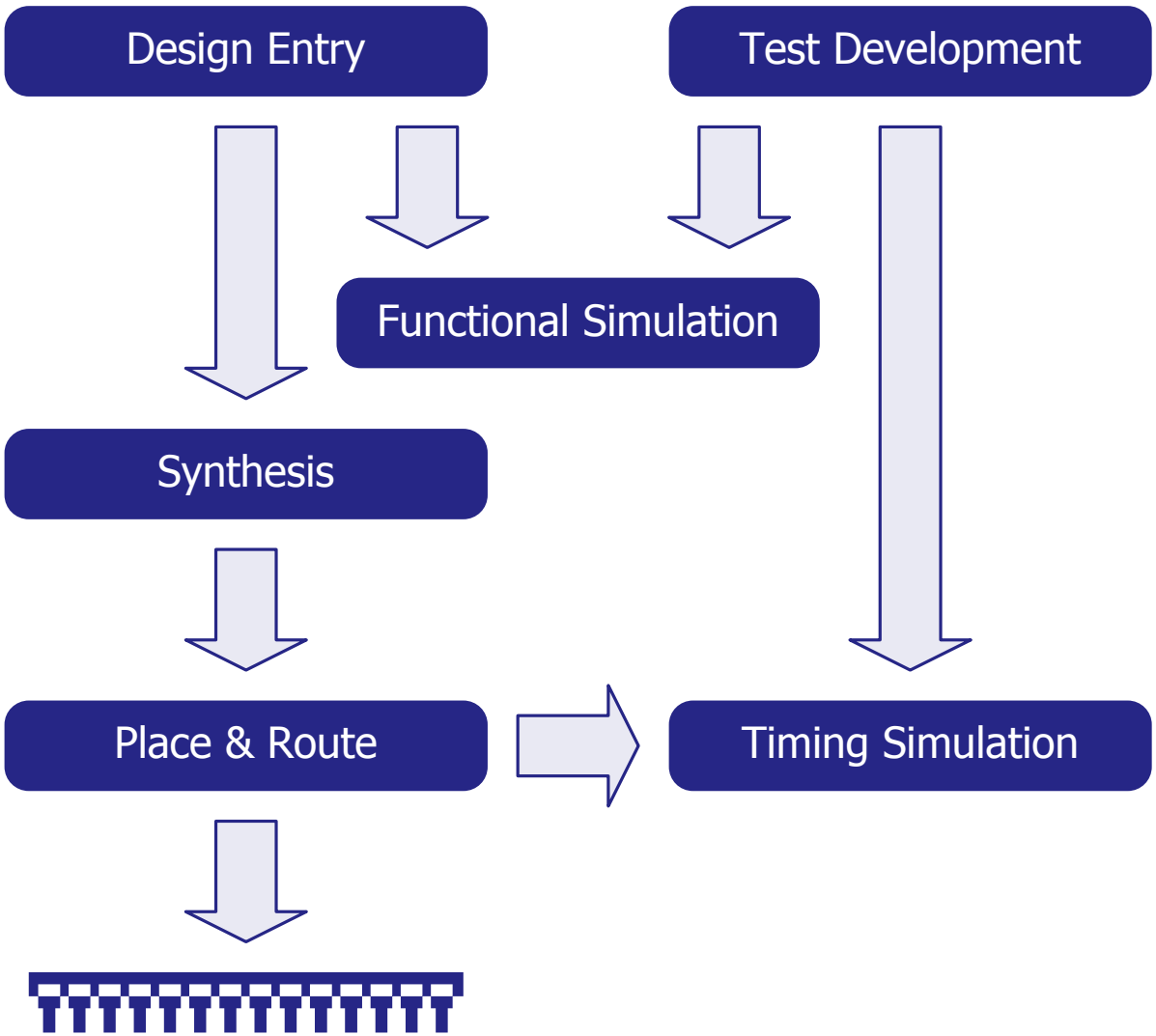
What are HDLs?

- Modeling language for electronic designs and systems
- VHDL, Verilog
- PALASM, ABEL
- Net list languages such as EDIF, XNF
- Test languages such as *e*, Vera
- SystemC for hardware/software co-design and verification

HDL for ...

- Formal description of hardware
- Specification on all abstraction layers
- Simulation of designs and whole systems
- Design entry for synthesis
- Standard interface between CAD tools
- Design reuse

Design methodology



Pros and Cons of HDLs

Pros

- Speeds up the whole design process
- Powerful tools
- Acts as an interface between tools
- Standardized
- Design reuse

Cons

- Learning curve
- Limited support for target architecture

Outline

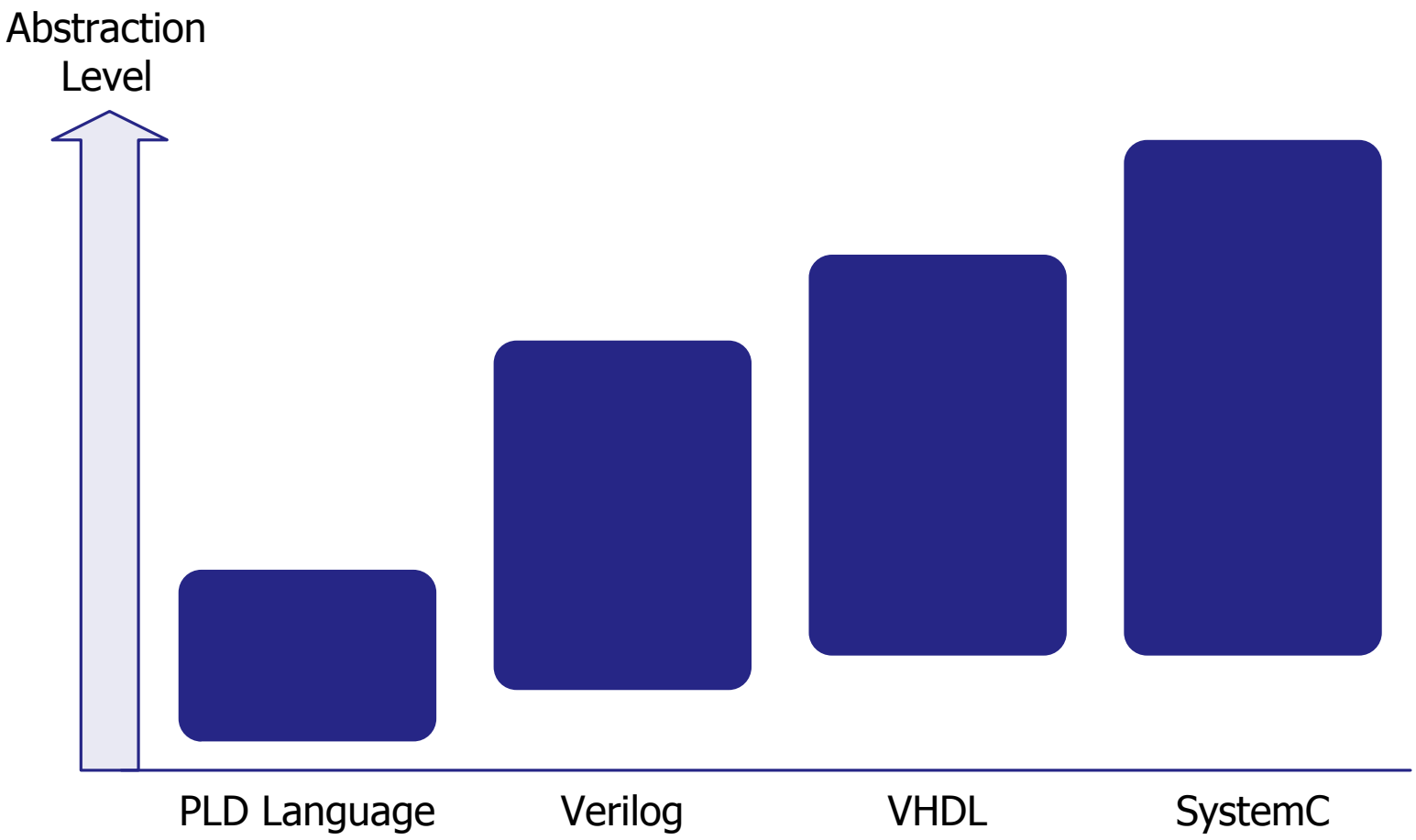
Design Abstraction Levels

Behavior Sequential statements, implicit registers

Data flow Register transfer level (RTL), Parallel statements, explicit registers

Structure Design hierarchy, Wiring components

HDL Abstraction Levels



Outline

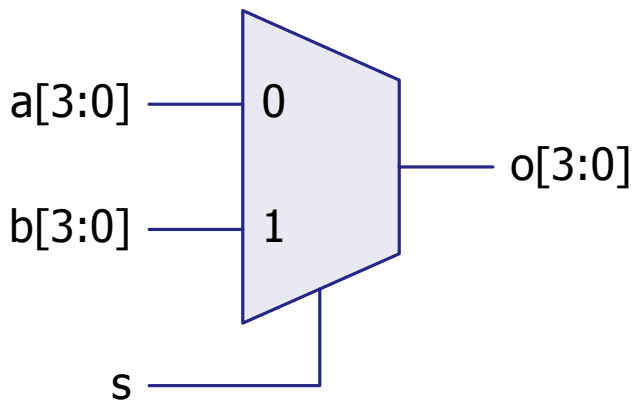
A First Example

Multiplexer

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
  port (
    a, b : in  std_logic_vector (3 downto 0);
    s    : in  std_logic;
    o    : out std_logic_vector (3 downto 0));
end mux;

architecture behavior of mux is
begin -- behavior
  o <= a when s = '0' else b;
end behavior;
```



Library

```
library ieee;
use ieee.std_logic_1164.all;
```

- Makes library ieee visible
- Use objects within the library
- Use package std_logic_1164
- Makes std_logic and std_logic_vector visible

Entity

```
entity mux is
  port (
    a, b : in  std_logic_vector (3 downto 0);
    s    : in  std_logic ;
    o    : out std_logic_vector (3 downto 0));
end mux;
```

- Defines the name of the module
- Describes the interface
  - Name of the ports
  - Direction
  - Type

Architecture

```
architecture behavior of mux is
begin  -- behavior
  o <= a when s = '0' else b;
end behavior;
```

- Implements the module
  - Behavior
  - Structure
  - RTL description
- Entity and architecture are linked by name (mux)
- Name of architecture (behavior)
- More than one architecture per entity allowed

Outline

Data Types

- Std_logic:
  - Represents electrical behaviour of circuit nodes
  - Can have 9 different states
- Std_logic_vector: array of std_logic values
- Signed and unsigned
  - Arrays similar to std_logic_vector
  - Main difference: arithmetic operators like + - * are defined
- Types like integer and natural often used with "range"
- No floating point types

logic vlaue		logic state		
		low	unknown	high
uninitialized			U	
strength	strong	0	X	1
	weak	L	W	H
	high-impedance	Z	Z	Z
don't care			-	

Outline

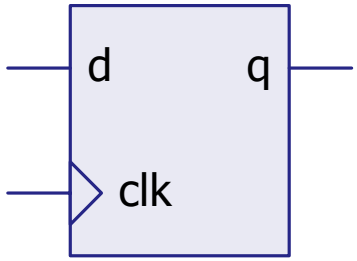
Register

```
entity reg is
  port (
    d, clk : in  std_logic ;
    q      : out std_logic );
end reg;

architecture rtl of reg is
begin -- rtl

  reg: process (clk)
  begin -- process reg
    if rising_edge (clk) then
      q <= d;
    end if;
  end process reg;

end rtl ;
```



- Sensitivity list only contains the clock
- Assignment on the rising edge of the clock
- Not transparent

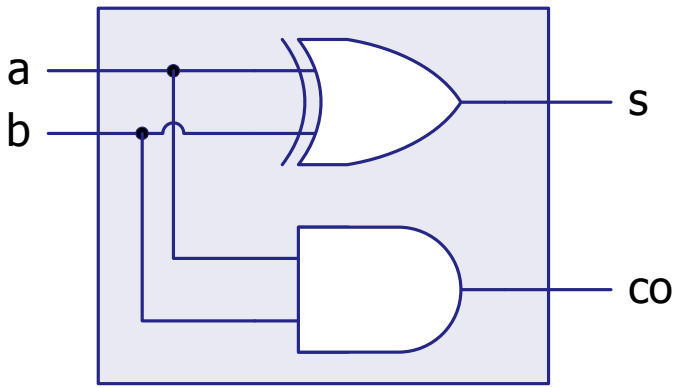
Parallel Processing

Half Adder

```
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
  port (
    a, b  : in  std_logic;
    s, co : out std_logic );
end half_adder;

architecture rtl of half_adder is
begin -- rtl
  s  <= a xor b;
  co <= a and b;
end rtl;
```

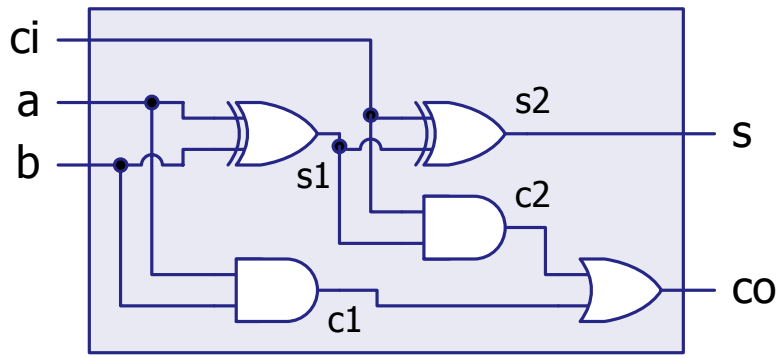


- Implicit gates
- Signals are used to wire gates
- Computes signals s and co from a and b in parallel

Full Adder

```
entity full_adder is
  port (
    a, b, ci : in  std_logic ;
    s, co    : out std_logic );
end full_adder ;

architecture beh_par of full_adder is
  signal s1, s2, c1, c2 : std_logic ;
begin -- behavior
  -- half adder 1
  s1 <= a xor b;
  c1 <= a and b;
  -- half adder 2
  s2 <= s1 xor ci;
  c2 <= s1 and ci;
  -- evaluate s and co
  s <= s2;
  co <= c1 or c2;
end beh_par;
```



- All statements are processed in parallel
- A signal must not be driven at two places
- Order of statements is irrelevant

Sequential Processing – Processes

```
architecture beh_seq of full_adder is
begin  -- beh_seq

  add: process (a, b, ci)
    variable s_tmp, c_tmp : std_logic;
  begin  -- process add
    -- half adder 1
    s_tmp := a xor b;
    c_tmp := a and b;
    -- half adder 2
    c_tmp := c_tmp or (s_tmp and ci);
    s_tmp := s_tmp xor ci;
    -- drive signals
    s <= s_tmp;
    co <= c_tmp;
  end process add;

end beh_seq;
```

- All statements are processed sequential
- Sensitivity list (a, b, ci)
- Multiple variable assignments are allowed
- Order of statements is relevant
- Variables are updated immediately
- Signals are updated at the end of a process
- Try to avoid multiple signal assignments

Parallel versus Sequential Processing

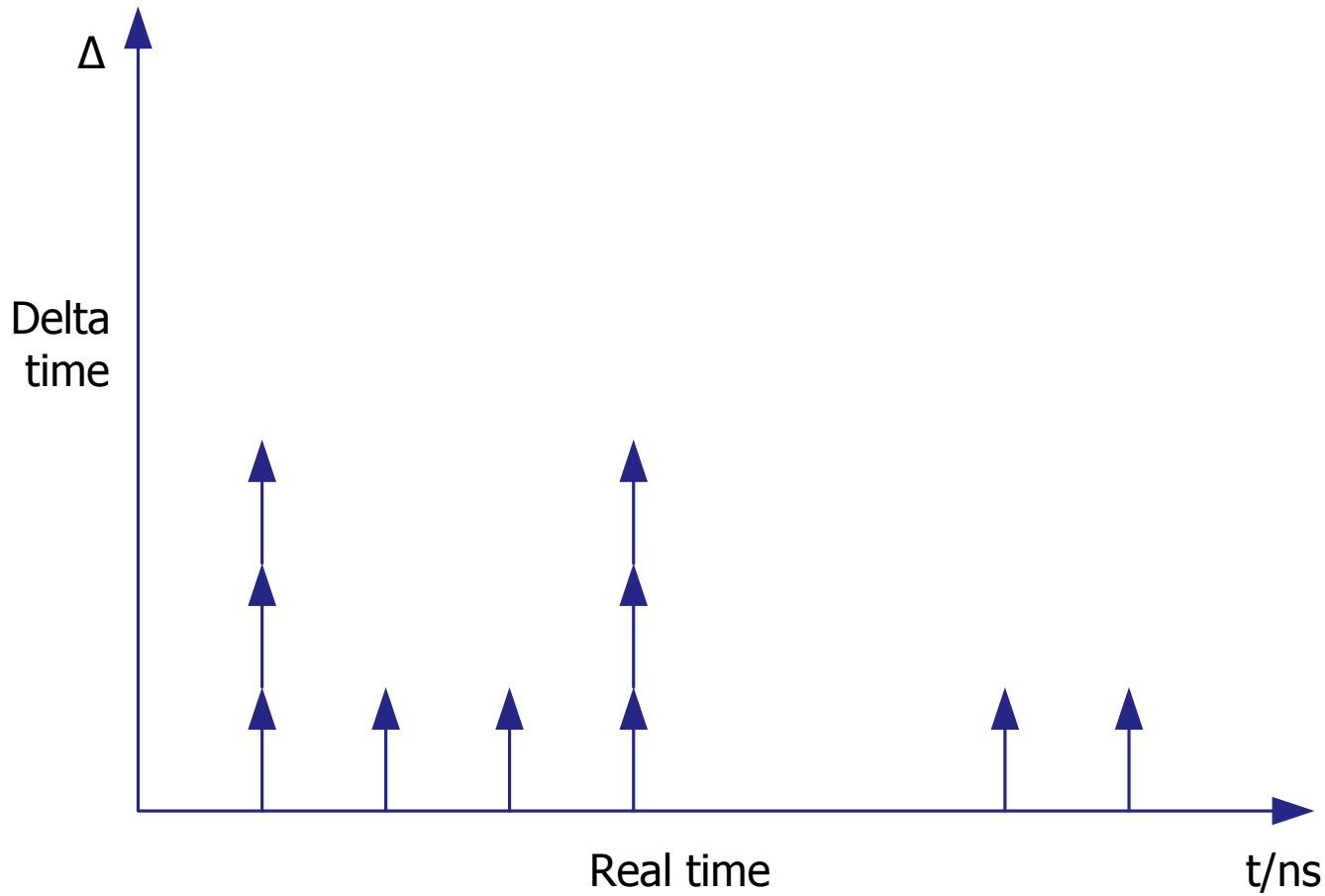
```
p1: process (a, b)
begin  -- process p1
  -- sequential statements
  -- ...
  -- drive process outputs
  x <= ...
end process p1;

p2: process (c, x)
begin  -- process p2
  -- sequential statements
  -- ...
  -- drive process outputs
  y <= ...
end process p2;

-- drive module outputs
o <= x or y;
```

- process p1, process p2 and o are processed in parallel
- Statements within processes are sequential
- Inter-process communication with signals
- Signal values are evaluated recursively in zero time
- Simulator uses delta cycles

Real time and delta cycles



Signals and Variables

Signals

- Inside and outside processes
- Communication between parallel statements and processes
- Only the last assignment within a process is evaluated
- Signal is updated at the end of a process
- Signals are wires

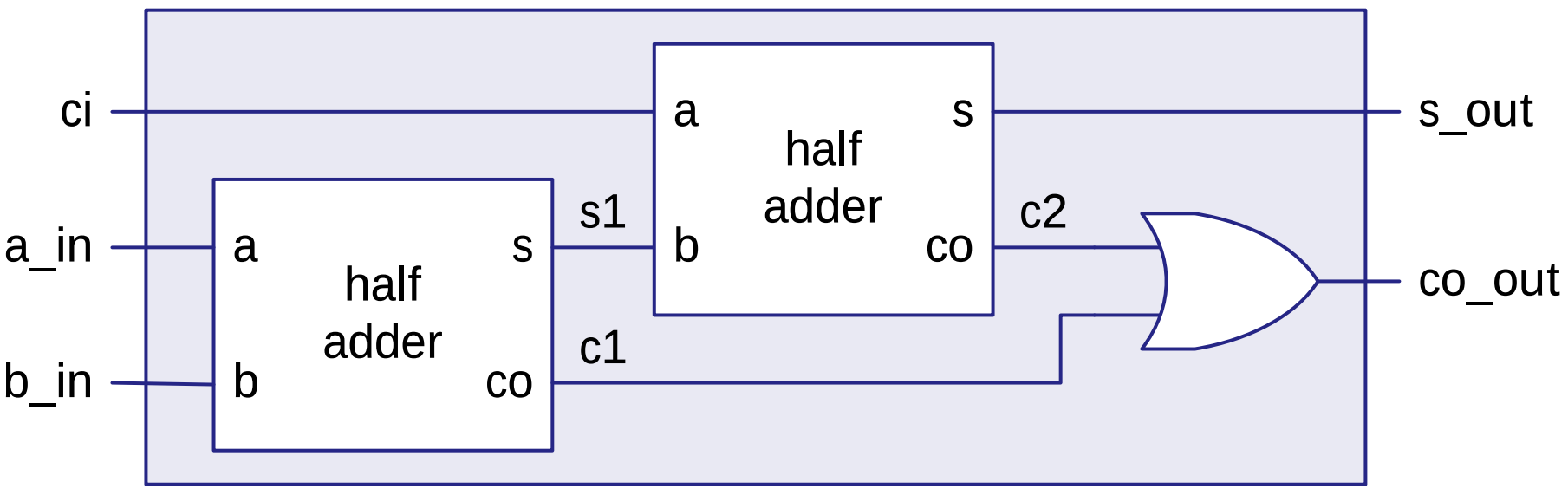
Variables

- Inside processes only
- For intermediate results
- Multiple assignments allowed
- Immediate update

Outline

Structural Description

- Module composition
- Wiring of Modules
- Design Hierarchy
- “Divide and conquer”



```
architecture structural of full_adder is
  component half_adder
    port (
      a, b  : in  std_logic ;
      s, co : out std_logic );
  end component;
  signal s1, c1, c2 : std_logic ;
begin  -- structural
  half_adder_1 : half_adder
    port map (
      a  => a_in,  b  => b_in,
      s  => s1, co => c1);

  half_adder_2 : half_adder
    port map (
      a  => ci, b  => s1,
      s  => s_out,  co => c2);

  co_out <= c1 or c2;

end structural ;
```

- Make module half_adder known with component declaration
- Module instantiation
- Connect ports and signals

Outline

Additional Information

Syntax Reference

- Vivado VHDL templates: Project Manager → Templates
- Wikibooks: https://de.wikibooks.org/wiki/VHDL-Tutorium

Literature

- Hubert Keaslin: Digital Integrated Circuit Design - From VLSI Architecture to CMOS Fabrication
- Paul Molitor, Jörg Ritter: Kompaktkurs VHDL