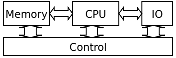
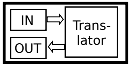
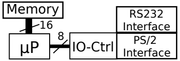
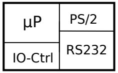
Hardware Modeling [VU] (191.011)
– WS25 –
Structural Modeling

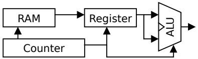
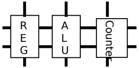
Florian Huemer & Sebastian Wiedemann & Dylan Baumann

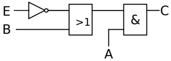
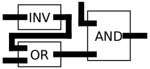
WS 2025/26

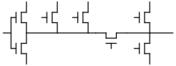
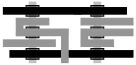
Modified: 2025-12-16, 16:05 (f8a58e9)

Introduction

	Behavior	Structure	Geometry
System Level	Inputs : Keyboard Output: Display Function:		
Algorithmic Level	while input read English text translate to German output German Text	 Memory 16 8 µP IO-Ctrl RS232 Interface PS/2 Interface	

Register Transfer Level (RTL)	if A=`1` then
	B:= B+1
	else
	B:= B
	end if
			

Logic Level	D = NOT E
	C = (D OR B) AND A
			

Circuit Level	$\frac{dU}{dt} = R\frac{dI}{dt} + \frac{I}{C} + L\frac{d^2I}{dt^2}$		

Introduction (cont’d)

Structual Modeling

Create complex modules by combining and interconnecting (simpler) sub-modules.

Introduction (cont'd)

Structual Modeling

Create complex modules by combining and interconnecting (simpler) sub-modules.

Top-Level Design/Entity

The design unit (entity) that sits on highest layer of a hierarchical hardware design.

Introduction (cont'd)

Structual Modeling

Create complex modules by combining and interconnecting (simpler) sub-modules.

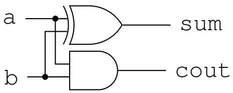
Top-Level Design/Entity

The design unit (entity) that sits on highest layer of a hierarchical hardware design.

Unit Under Test (UUT)

The module instantiated in a testbench and whose behavior is verified.

Creating Instances – Example: Half Adder

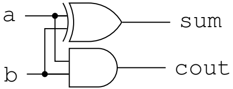


```
entity xor_gate is
 port (
  a, b : in std_ulogic;
  x : out std_ulogic
 );
end entity;

architecture arch of xor_gate is
begin
 x <= a xor b;
end architecture;
```

```
entity and_gate is
 port (
  a, b : in std_ulogic;
  x : out std_ulogic
 );
end entity;

architecture arch of and_gate is
begin
 x <= a and b;
end architecture;
```

Creating Instances – Example: Half Adder

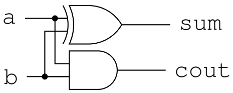
```
entity ha is
 port (
  a, b : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;

architecture arch of ha is
begin
 and_gate_inst : entity work.and_gate
 port map (a, b, cout);

 xor_gate_inst : entity work.xor_gate
 port map (a, b, sum);
end architecture;
```



- Instances → statement part
- Two instantiation statements
- Positional association
- Inputs must be readable, outputs writable

Creating Instances – Example: Half Adder

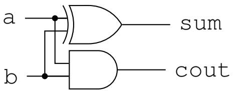
```
entity ha is
 port (
  a, b : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;

architecture arch of ha is
begin
 and_gate_inst : entity work.and_gate
 port map (a, b, cout);

 xor_gate_inst : entity work.xor_gate
 port map (a, b, sum);
end architecture;
```



- Instances → statement part
- Two instantiation statements
- Positional association
- Inputs must be readable, outputs writable

Creating Instances – Example: Half Adder

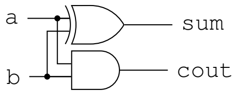
```
entity ha is
 port (
  a, b : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;

architecture arch of ha is
begin
 and_gate_inst : entity work.and_gate
 port map (a, b, cout);

 xor_gate_inst : entity work.xor_gate
 port map (a, b, sum);
end architecture;
```



- Instances → statement part
- Two instantiation statements
- Positional association
- Inputs must be readable, outputs writable

Creating Instances – Example: Half Adder

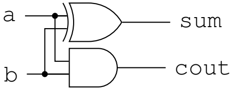
```
entity ha is
 port (
  a, b : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;

architecture arch of ha is
begin
 and_gate_inst : entity work.and_gate
 port map (a, b, cout);

 xor_gate_inst : entity work.xor_gate
 port map (a, b, sum);
end architecture;
```



- Instances → statement part
- Two instantiation statements
- Positional association
- Inputs must be readable, outputs writable

Creating Instances – Example: Half Adder

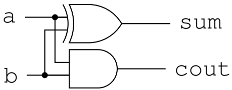
```vhdl
entity ha is
 port (
  a, b : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;

architecture arch of ha is
begin
 and_gate_inst : entity work.and_gate
 port map (a, b, cout);

 xor_gate_inst : entity work.xor_gate
 port map (a, b, sum);
end architecture;
```



- Instances → statement part
- Two instantiation statements
- Positional association
- Inputs must be readable, outputs writable

Creating Instances – Example: Half Adder

```vhdl
entity ha is
 port (
  a, b : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;

architecture arch of ha is
begin
 and_gate_inst : entity work.and_gate
 port map (a, b, cout);

 xor_gate_inst : entity work.xor_gate
 port map (a, b, sum);
end architecture;
```



- Instances → statement part
- Two instantiation statements
- Positional association
- Inputs must be readable, outputs writable

Port Map

- Port map syntax

```
port map(association_list)
```

- Part in parentheses → Association list ◇110

Port Map

- Port map syntax

```
port map(association_list)
```

- Part in parentheses → Association list 110
- Named and positional associations possible

Port Map

- Port map syntax

```
port map(association_list)
```

- Part in parentheses → Association list ◇110
- Named and positional associations possible
- **Named association** oftentimes preferable over positional association because of better
  - clarity
  - maintainability
  - flexibility
  - robustness (w.r.t. connection bugs)

Port Map

- Port map syntax

```
port map(association_list)
```

- Part in parentheses → Association list ◇110
- Named and positional associations possible
- **Named association** oftentimes preferable over positional association because of better
  - clarity
  - maintainability
  - flexibility
  - robustness (w.r.t. connection bugs)
- Association list syntax

```
association_list ::= assoc_elem {, assoc_elem}
assoc_elem ::= [formal_part => ]actual_part
```

Port Map

- Port map syntax

```
port map(association_list)
```

- Part in parentheses → Association list ◇110
- Named and positional associations possible
- **Named association** oftentimes preferable over positional association because of better
  - clarity
  - maintainability
  - flexibility
  - robustness (w.r.t. connection bugs)
- Association list syntax

```
association_list ::= assoc_elem {, assoc_elem}
assoc_elem ::= [formal_part => ]actual_part
```

- Don’t mix association styles in port maps

Port Map

- Port map syntax

```
port map(association_list)
```

- Part in parentheses → Association list ◇110
- Named and positional associations possible
- **Named association** oftentimes preferable over positional association because of better
  - clarity
  - maintainability
  - flexibility
  - robustness (w.r.t. connection bugs)
- Association list syntax

```
association_list ::= assoc_elem {, assoc_elem}
assoc_elem ::= [formal_part => ]actual_part
```

- Don’t mix association styles in port maps

Port Map

- Port map syntax

  ```
  port map(association_list)
  ```

- Part in parentheses → Association list
- Named and positional associations possible
- **Named association** oftentimes preferable over positional association because of better
  - clarity
  - maintainability
  - flexibility
  - robustness (w.r.t. connection bugs)
- Association list syntax

  ```
  association_list ::= assoc_elem {, assoc_elem}
  assoc_elem ::= [formal_part => ]actual_part
  ```

- Don’t mix association styles in port maps

Port Map

- Port map syntax

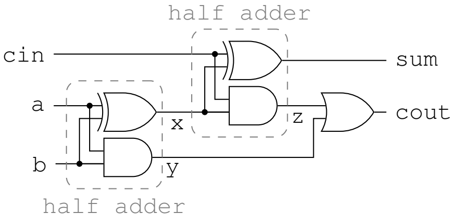
```
port map(association_list)
```

- Part in parentheses → Association list ◇110
- Named and positional associations possible
- **Named association** oftentimes preferable over positional association because of better
  - clarity
  - maintainability
  - flexibility
  - robustness (w.r.t. connection bugs)
- Association list syntax

```
association_list ::= assoc_elem {, assoc_elem}
assoc_elem ::= [formal_part => ]actual_part
```

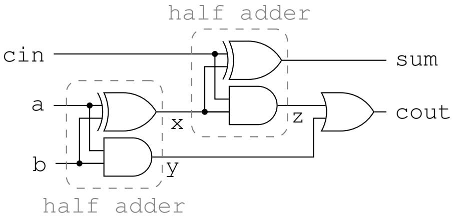
- Don’t mix association styles in port maps

Port Map - Example: Full Adder



```
entity fa is
 port (
  a, b, cin  : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;
```

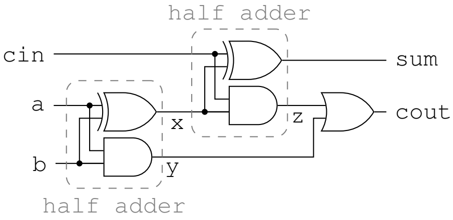
Port Map - Example: Full Adder



```vhdl
entity fa is
 port (
  a, b, cin   : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;
```

```vhdl
architecture arch of fa is
 signal x, y, z : std_ulogic;
begin
 ha1 : entity work.ha
 port map(a, b, x, y);

 ha2 : entity work.ha
 port map(
  a => cin,
  b => x,
  cout => z,
  sum => sum
 );

 cout <= y or z;
end architecture;
```

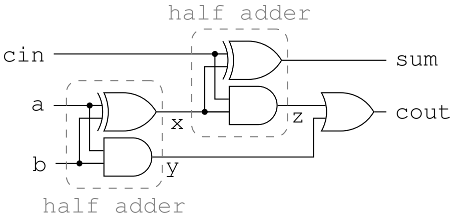
Port Map - Example: Full Adder



```vhdl
entity fa is
 port (
  a, b, cin  : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;
```

```vhdl
architecture arch of fa is
 signal x, y, z : std_ulogic;
begin
 ha1 : entity work.ha
 port map(a, b, x, y);

 ha2 : entity work.ha
 port map(
  a => cin,
  b => x,
  cout => z,
  sum => sum
 );

 cout <= y or z;
end architecture;
```

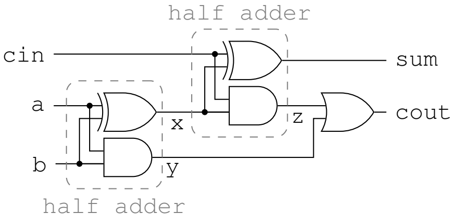
Port Map - Example: Full Adder



```
entity fa is
 port (
  a, b, cin   : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;
```

```
architecture arch of fa is
 signal x, y, z : std_ulogic;
begin
 ha1 : entity work.ha
 port map(a, b, x, y);

 ha2 : entity work.ha
 port map(
  a => cin,
  b => x,
  cout => z,
  sum => sum
 );

 cout <= y or z;
end architecture;
```

Port Map - Example: Full Adder



```vhdl
entity fa is
 port (
  a, b, cin  : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;
```

```vhdl
architecture arch of fa is
 signal x, y, z : std_ulogic;
begin
 ha1 : entity work.ha
 port map(a, b, x, y);

 ha2 : entity work.ha
 port map(
  a => cin,
  b => x,
  cout => z,
  sum => sum
 );

 cout <= y or z;
end architecture;
```

Unused Ports

Unused Ports

Unused outputs

- Use `open` keyword
- Don’t leave unconnected!

```
 [...]
 signal a, b, c, m : std_ulogic;
begin
 majority : entity work.fa
 port map (
  a => a,
  b => b,
  c => c,
  sum => open, -- not connected
  cout => m
 );
 [...]
```

Unused Ports

Unused outputs

- Use `open` keyword
- Don’t leave unconnected!

```
 [...]
 signal a, b, c, m : std_ulogic;
begin
 majority : entity work.fa
 port map (
  a => a,
  b => b,
  c => c,
  sum => open, -- not connected
  cout => m
 );
 [...]
```

Unused inputs

- Connect to constant
- If not connected → default value

```
 [...]
 signal a, b, sum, cout : std_ulogic;
begin
 half_adder : entity work.fa
 port map (
  a => a,
  b => b,
  c => '0', -- constant
  sum => sum,
  cout => cout
 );
 [...]
```

Generic Map

- Generic map syntax

  ```
  generic map(association_list)
  ```

- Must appear before port map

Generic Map

- Generic map syntax

  ```
  generic map(association_list)
  ```
- Must appear before port map
- Use named association and avoid the positional style

Generic Map

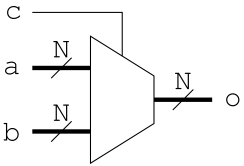
- Generic map syntax

  ```
  generic map(association_list)
  ```

- Must appear before port map
- Use named association and avoid the positional style
- Formal parts must be compile-time constants (generics cannot be connected to signals)

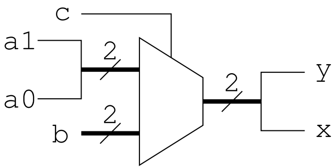
Generic Map – Example: Multiplexer

```vhdl
entity mux is
 generic (
  N : positive
 );
 port (
  c : in std_ulogic;
  a : in std_ulogic_vector(N-1 downto 0);
  b : in std_ulogic_vector(N-1 downto 0);
  o : out std_ulogic_vector(N-1 downto 0)
 );
end entity;

architecture arch of mux is
begin
 o <= a when c = '0' else b;
end architecture;
```



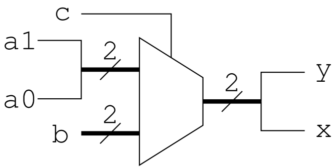
Generic Map – Example: Multiplexer (cont’d)

```
 [...]
 signal c : std_ulogic;
 signal a0, a1 : std_ulogic;
 signal b : std_ulogic_vector(1 downto 0);
 signal x, y : std_ulogic;
begin
 mux_inst : entity work.mux
 generic map (N => 2)
 port map (
  c => c,
  a => a1 & a0,
  b => b,
  o(0) => x,
  o(1) => y
 );
 [...]
```



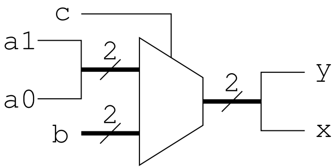
Generic Map – Example: Multiplexer (cont’d)

```
 [...]
 signal c : std_ulogic;
 signal a0, a1 : std_ulogic;
 signal b : std_ulogic_vector(1 downto 0);
 signal x, y : std_ulogic;
begin
 mux_inst : entity work.mux
 generic map (N => 2)
 port map (
  c => c,
  a => a1 & a0,
  b => b,
  o(0) => x,
  o(1) => y
 );
 [...]
```



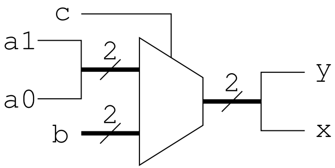
Generic Map – Example: Multiplexer (cont'd)

```
 [...]
 signal c : std_ulogic;
 signal a0, a1 : std_ulogic;
 signal b : std_ulogic_vector(1 downto 0);
 signal x, y : std_ulogic;
begin
 mux_inst : entity work.mux
 generic map (N => 2)
 port map (
  c => c,
  a => a1 & a0,
  b => b,
  o(0) => x,
  o(1) => y
 );
 [...]
```



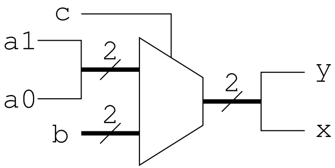
Generic Map – Example: Multiplexer (cont'd)

```
 [...]
 signal c : std_ulogic;
 signal a0, a1 : std_ulogic;
 signal b : std_ulogic_vector(1 downto 0);
 signal x, y : std_ulogic;
begin
 mux_inst : entity work.mux
 generic map (N => 2)
 port map (
  c => c,
  a => a1 & a0,
  b => b,
  o(0) => x,
  o(1) => y
 );
 [...]
```



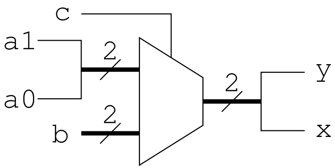
Generic Map – Example: Multiplexer (cont’d)

```
 [...]
 signal c : std_ulogic;
 signal a0, a1 : std_ulogic;
 signal b : std_ulogic_vector(1 downto 0);
 signal x, y : std_ulogic;
begin
 mux_inst : entity work.mux
 generic map (N => 2)
 port map (
  c => c,
  a => a1 & a0,
  b => b,
  o(0) => x,
  o(1) => y
 );
 [...]
```



Generic Map – Example: Multiplexer (cont’d)

```
 [...]
 signal c : std_ulogic;
 signal a0, a1 : std_ulogic;
 signal b : std_ulogic_vector(1 downto 0);
 signal x, y : std_ulogic;
begin
 mux_inst : entity work.mux
 generic map (N => 2)
 port map (
  c => c,
  a => a1 & a0,
  b => b,
  o(0) => x,
  o(1) => y
 );
 [...]
```



Components

Components

- Components are *“entity prototype”*

Components

- Components are *“entity prototype”*
- Component declaration syntax

```
component NAME is
[ generic ( {generic_element;} generic_element ); ]
[ port ( {port_element;} port_element ); ]
end component;
```

Components

- Components are *“entity prototype”*
- Component declaration syntax

```
component NAME is
[ generic ( {generic_element;} generic_element ); ]
[ port ( {port_element;} port_element ); ]
end component;
```

- Components can be put in
  - packages
  - declarative part of architectures (blocks and generate statements)

Components (cont'd)

- Instantiation example
  - Entity: `i : entity work.fa port map (...);`
  - Component: `i : fa port map (...);`
  - Component (explicit): `i : component fa port map (...);`
- Modularity, abstraction and separation of concerns
- Compilation order
- Entity may not always be available
- Mixed-language designs

Architecture Selection

- Multiple different architectures possible for a single entity
  $\Rightarrow$ How to select one?
- Two possibilities
  - Specify architecture, when an **entity** is instantiated
    Caution: This does not work for components!
  - Configurations

Entity Instantiations

```
entity mystery is
 port (
  a, b : in std_ulogic;
  x : out std_ulogic
 );
end entity;

architecture a1 of mystery is
begin
 x <= a and b;
end architecture;

architecture a2 of mystery is
begin
 x <= a xor b;
end architecture;
```

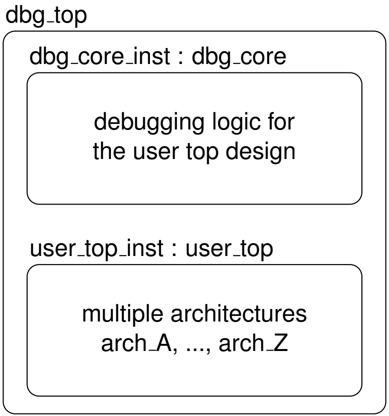
Entity Instantiations

```vhdl
entity mystery is
 port (
  a, b : in std_ulogic;
  x : out std_ulogic
 );
end entity;

architecture a1 of mystery is
begin
 x <= a and b;
end architecture;

architecture a2 of mystery is
begin
 x <= a xor b;
end architecture;
```

```vhdl
entity ha is
 port (
  a, b : in std_ulogic;
  sum, cout : out std_ulogic
 );
end entity;

architecture arch of ha is
begin
 and_gate : entity work.mystery(a1)
 port map (a, b, cout);

 xor_gate : entity work.mystery(a2)
 port map (a, b, sum);
end architecture;
```

Configurations

```
configuration dbg_top_conf of dbg_top is
 for dbg_top_arch
  for user_top_inst : user_top
   use entity user_top(arch_A);
  end for;
 end for;
end configuration;
```



dbg_top

dbg_core_inst : dbg_core

debugging logic for the user top design

user_top_inst : user_top

multiple architectures arch_A, ..., arch_Z

Lecture Complete!