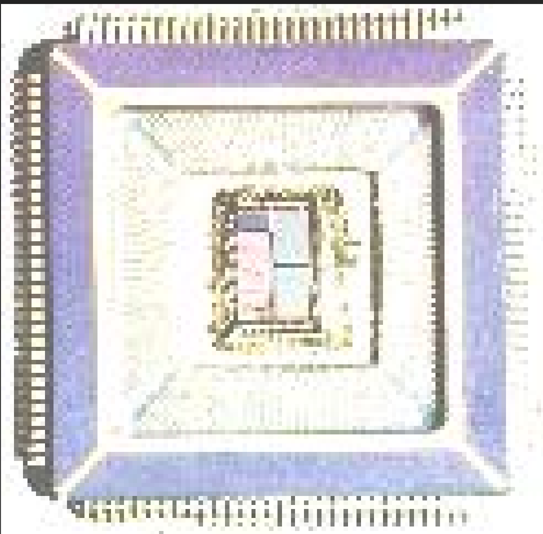


Hardware Modeling

Grundelemente

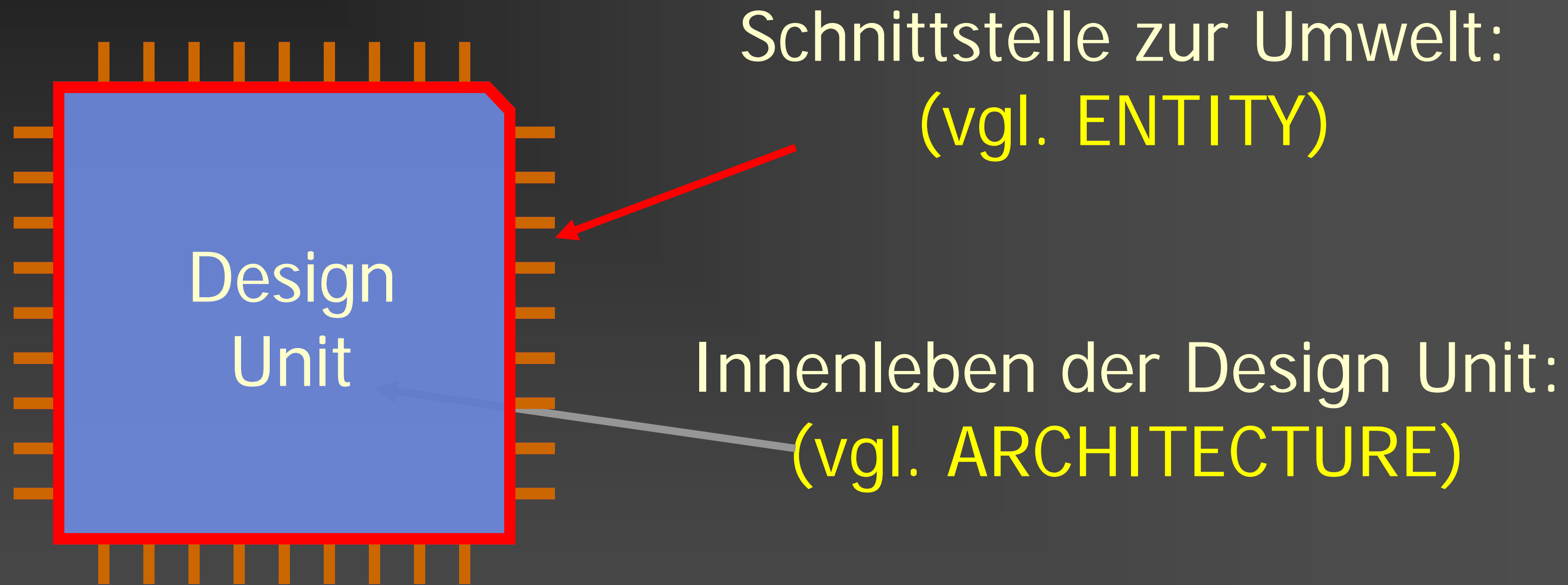
ECS Group, TU Wien

- Hardware Specification
  - Functional specification
  - High Level Requirements
  - Detailed Design Description
- Realisation
  - Hardware Description
  - Hardware Implementation
- Verification
  - Review
  - Formal verification
  - Simulation

Inhalt: Grundelemente

- Aufbau einer Design Unit
  - Entity
  - Architecture
  - Configuration
- Package
  - Package
  - Package Body
- Library

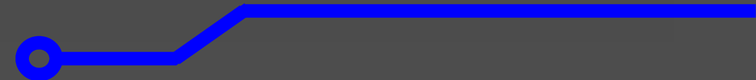
Wie bauen wir eine Design Unit ?



⇒ Jede Design Unit besteht aus einer ENTITY und keiner, einer oder mehreren ARCHITECTUREs

ENTITY Deklaration

- Schnittstelle des Blockes zur „Außenwelt“
- Wichtigste Elemente sind die **Eingänge** und **Ausgänge** (ports)
- Zusätzliche Deklaration von Parametern möglich (Generics)
- **Keine** Beschreibung der Funktion
- Vergleichbar mit dem Blockschaltbild eines Bauteiles



ENTITY Beispiel

Beispiel eines Halb-Addierers

entity name

generic parameter

```
entity HA is
    generic ( delay: time := 2,2 ns);
    port ( a,b          : in bit;
           carry, sum   : out bit);
end entity HA;
```

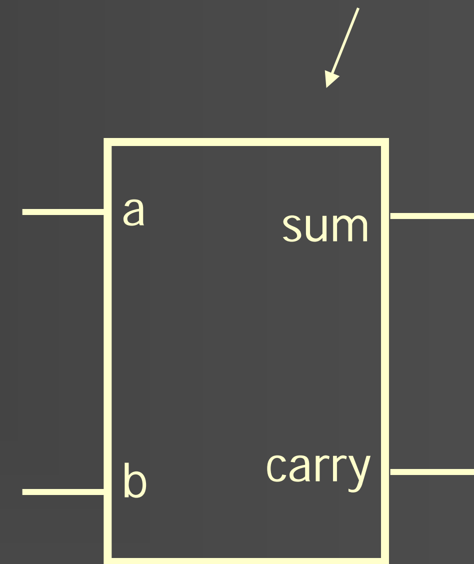
port type

port mode (direction)

reserved words

port name

Blockschaltbild



Beispiel Addierers mit einstellbarer Bitbreite

```
entity Adder is
    generic ( width: natural := 8);
    port ( a,b       : in bit_vector ( width-1 downto 0);
           sum       : out bit_vector ( width-1 downto 0);
           carry     : out bit);
end entity Adder;
```

Port-Typen

Eigensch. / Port-Typ	Lesen	Schreiben	Bemerkung
IN	Ja	Nein	-
OUT	Nein	Ja	-
INOUT	Ja	Ja	
BUFFER	Ja	Ja	Kann nur von einer Quelle beschrieben werden

- Enthält die Funktionalität der **entity**
- Die Beschreibung der Funktionalität kann auf der
  - *RTL Ebene*
  - *Logikebene*

  erfolgen
- Beide Ebenen können gemischt auftreten
- Eine **entity** kann *keine, eine* oder *mehrere* **architectures** haben

Architecture Beispiel (1)

```
architecture rtl of HA is
begin
 process(a,b)
 begin
    if a=`1` and b=`1` then
      carry <=`1`;
      sum  <=`0`;
    elsif a /= b then
      carry <=`0`;
      sum  <=`1`;
    else
       carry <=`0`;
      sum  <=`0`;
     end if;
 end process;
end;
```

keyword → architecture

architecture name → rtl

entity name → HA

sensitivity list → (a,b)

loop → process … end process

reserved words → end process;

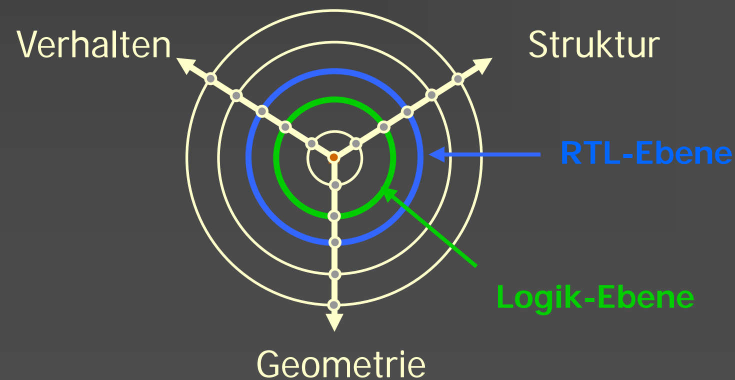
Architecture Beispiel (2)

Architecture „rtl“

```
architecture rtl of HA is
begin
 process(a,b)
     if a=`1` and b=`1` then
       carry <=`1`;
       sum  <=`0`;
     elsif a/= b then
       carry <=`0`;
       sum  <=`1`;
     else
        carry <=`0`;
       sum  <=`0`;
      end if;
 end process;
end;
```

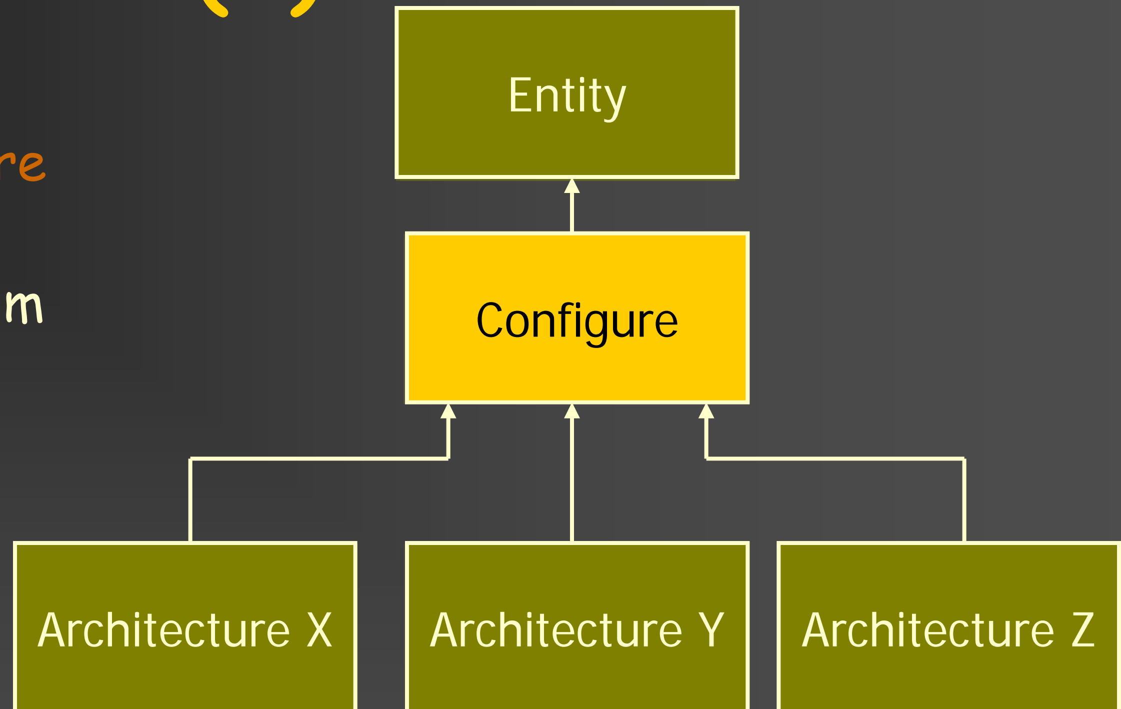
Architecture „dataflow“

```
architecture dataflow  of HA is
begin
    carry <=  a and b ;
    sum   <=  a xor b ;
end;
```



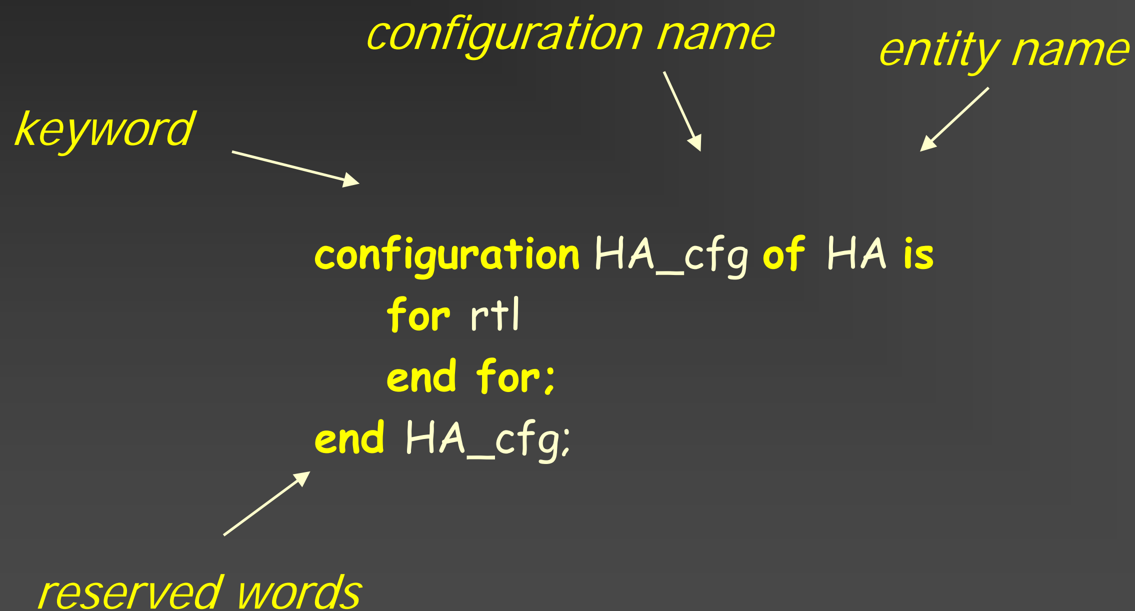
Configure-Teil (1)

- Eine **entity** kann mehrere **architectures** haben
- Die Zuordnung erfolgt im **Configuration-Teil** des Designs



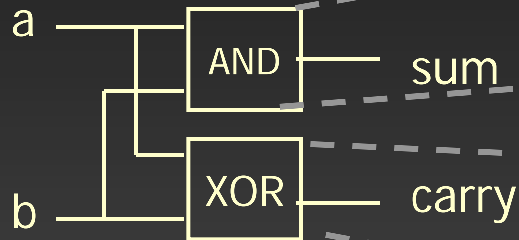
Vorteil:
Unterschiedliche Implementierungen einer Funktion können parallel zueinander existieren und lediglich durch Ändern des Configuration-Teils ausgetauscht werden

Configure-Teil (2)



Configure-Teil (3)

architecture structural



Architecture behaviour

Architecture structural

Architecture behaviour

Architecture structural

```
configuration HA_cfg of HA is
        for dataflow
                for and : and_gate use work.and_gate(behaviour);
                for xor  : xor_gate use work.xor_gate(structure);
        end for;
end HA_cfg;
```

additional configurations

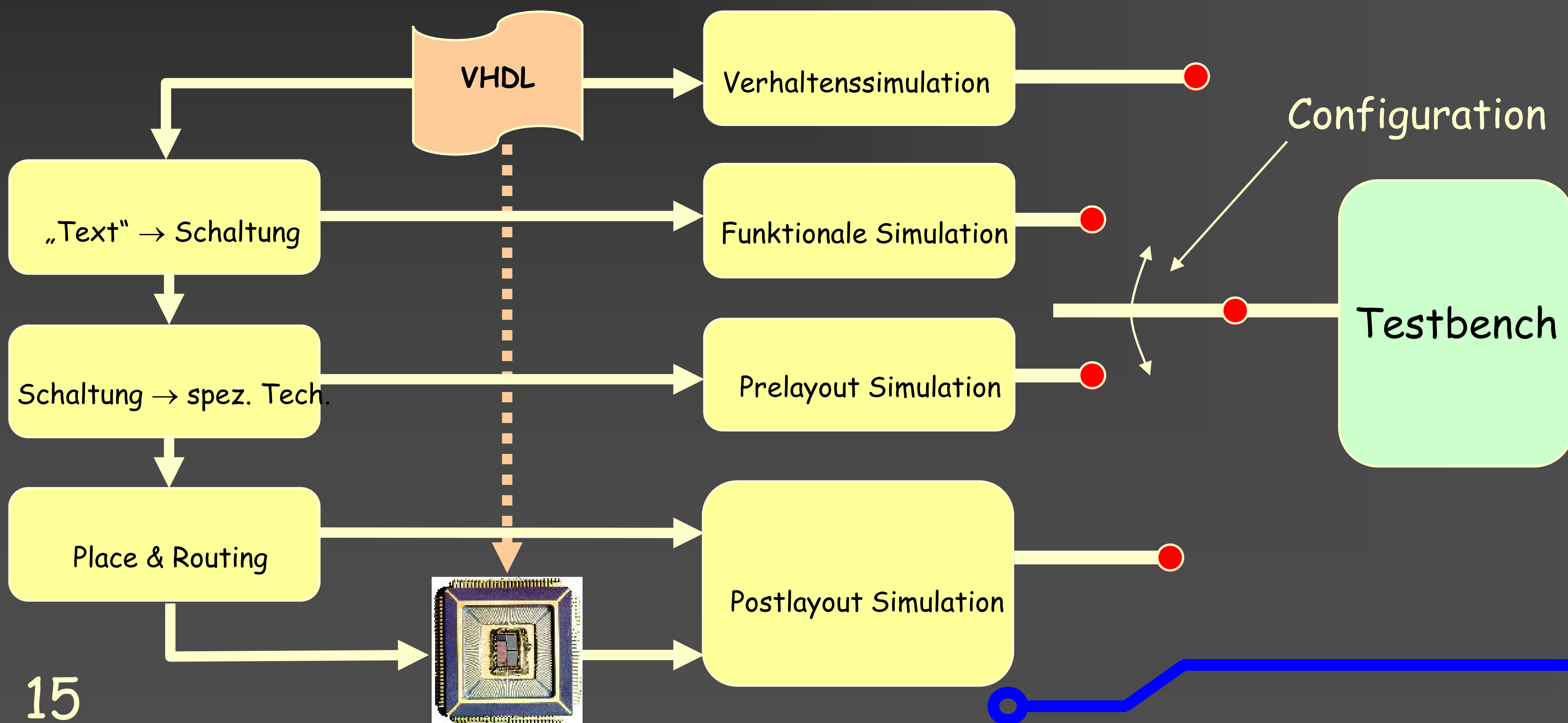
instance in structure

entity of instance

architecture of instance

Configure-Teil (4)

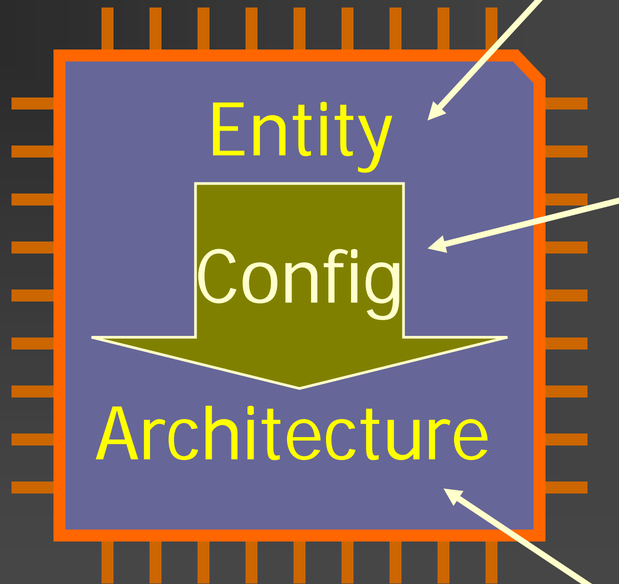
Anwendungsbeispiel einer Configuration



Zwischenbilanz

VHDL-Block
besteht aus

- einer **Entity,**
- Keine, einer oder mehreren **Achitectures**
- und einem **Konfigurationsteil**




```
entity HA is
   generic ( delay: time := 2,2 ns);
   port ( a,b            : in bit;
          carry, sum     : out bit);
end entity nand;
```

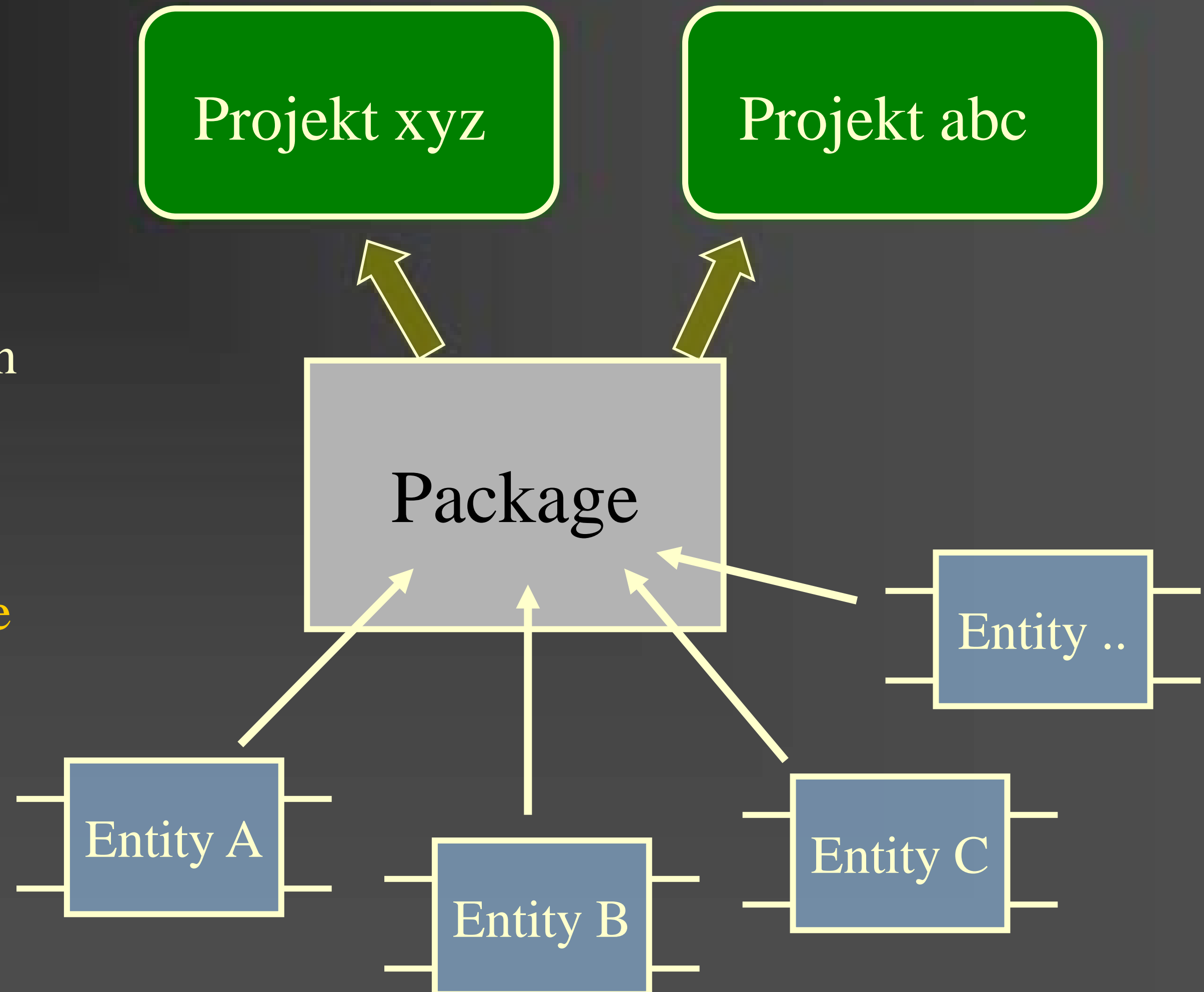
```
configuration HA_cfg of HA is
      for rtl
      end for;

end HA_cfg;
```

```
architecture rtl of HA is
begin
 process(a,b)
      if a=1 and b=1 then
        carry <=`1; sum  <=`0`;
      elsif a<> b then
        carry <=`0`; sum  <=`1`;
      else
        carry <=`0`; sum  <=`0`;
       end if;
 end process;
end;
```

! Diese drei Teile können getrennt oder in einer gemeinsamen Datei abgelegt werden

Package (1)

- Komponenten können in einem Package zusammengefasst werden
- Durch Einbinden des Packages können unterschiedliche Projekte diese Komponenten verwenden
- vgl. C-Header-Datei

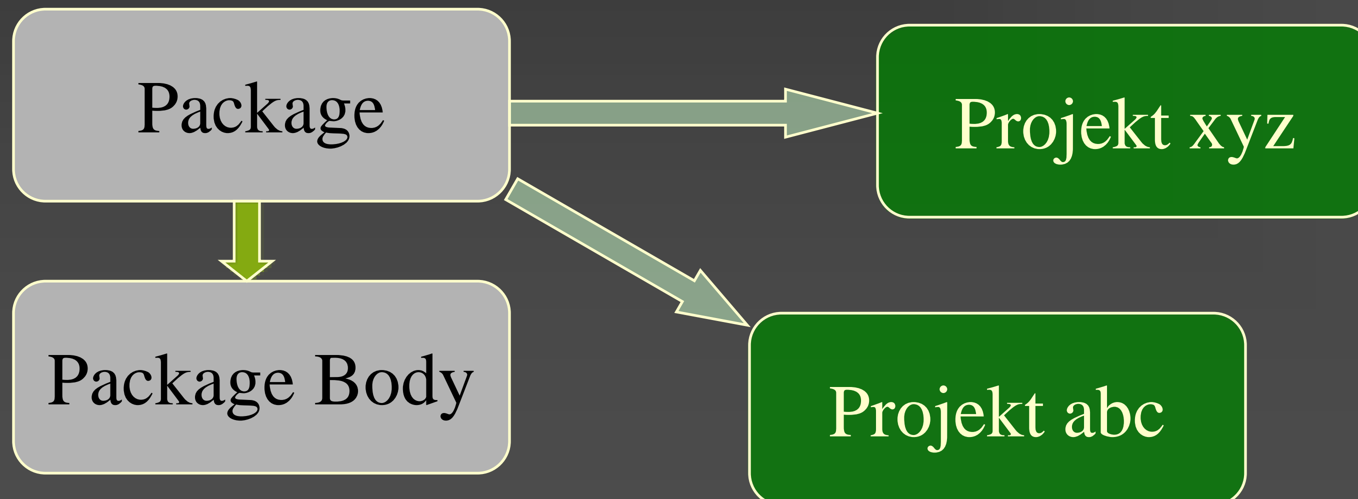


Package (2)

- Ein Package kann in mehreren VHDL-Projekten verwendet werden
- Es enthält:
  - Datentypen,
  - Komponenten,
  - Funktionen / Prozeduren,
  - Deklarationen, ...
- VHDL unterscheidet
  - Package (Deklarationen)
  - Package Body (*Definitionen und Funktionen)*

Package vs. Package Body (1)

- Package: Schnittstelle für Funktionen, Typen, Komponenten, ... (vgl. *include.h* -Datei in C)
- Package Body: Funktionskörper, Wertzuweisungen für Konstanten, ... (vgl. *include.c* -Datei in C)



Package vs. Package Body (2)

package name

```
package my_pack is
    type tristate is (`0`,`1`,`Z`);
    constant std_delay : time;

    -- Funktionskopf
    function bit2int (val : bit) return
    integer;

    -- Komponenten Deklaration
    component HA
       generic ( delay: time :=
    std_delay);
       port  ( a,b          : in bit;
               carry, sum   : out bit);
    end component;
end my_pack;
```

```
package body my_pack is
    constant std_delay : time := 2ns;

    -- Funktionskörper
    function bit2int (val: bit) return
    integer;
      begin
         if value=`1` then
             return 1;
         else
             return 0;
         end if;
    end bit2int;

end my_pack
```

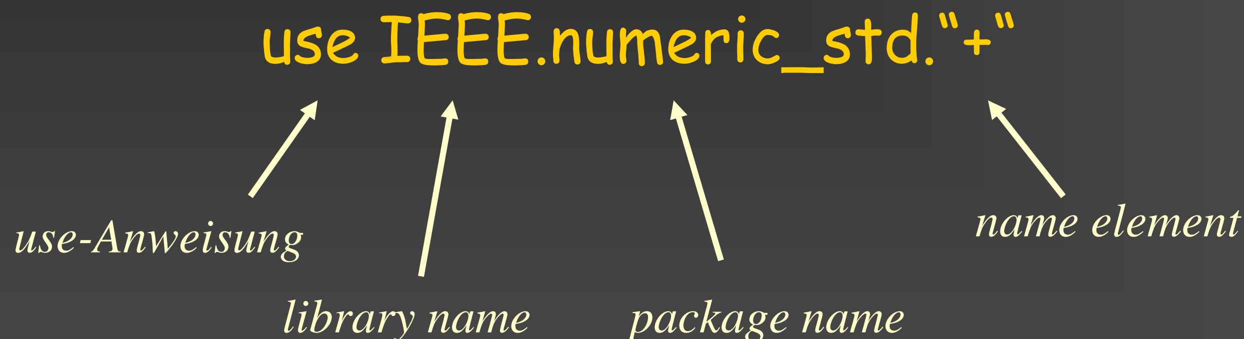
- Libraries sind vor-compilierte VHDL Units
- Library Name entspricht somit einer Pfadangabe
- Mit der folgenden Anweisung werden Bibliotheken in einem Design sichtbar gemacht

```
LIBRARY lib_name;
```

- Die Library-Anweisung kann vor einer *Entity,* vor einer *Architecture,* vor einer *Configuration,* vor einem *Package* und vor einem *Package Body* stehen

USE - Anweisung

- Mit der USE-Anweisung werden Bibliothekselemente bekannt gemacht:



- Wenn statt dem „*.element name*" „*.all*" steht, so sind alle Elemente der Library bzw. des Packages sichtbar.

- Festgelegt durch IEEE-1076
- Enthält die Package *standard* und *textio*

- Package standard (automatisch sichtbar, vgl. work)
  - Grundlegende Typen:
    bit, boolean, integer, natural, ...
  - Operatoren:
    - and, or, not, ...
    - =, /=,<, ...,

- Package textio (nicht autom. sichtbar)
  - Dateitypen
  - Ein-/ Ausgabeoperatoren

- Package std_logic_1164:
  - Typen
    - std_logic, std_ulogic, ...
  - Operatoren
    - and, or, not, ...
  - Konvertierungsfunktionen:
    - z.B.: To_bit, To_stdlogic, ...
  - Funktionen
    - rising_edge, falling_edge, ...

Library IEEE (2)

- Package numeric_std:
  - Typen
    - SIGNED
    - UNSIGNED
  - Operatoren
    - +, -, *, /, ABS, ... (Achtung „Synthese“)
    - <, >, <=, ...
  - Konvertierungsfunktionen

    TO_INTEGER(arg: UNSIGNED) return INTEGER

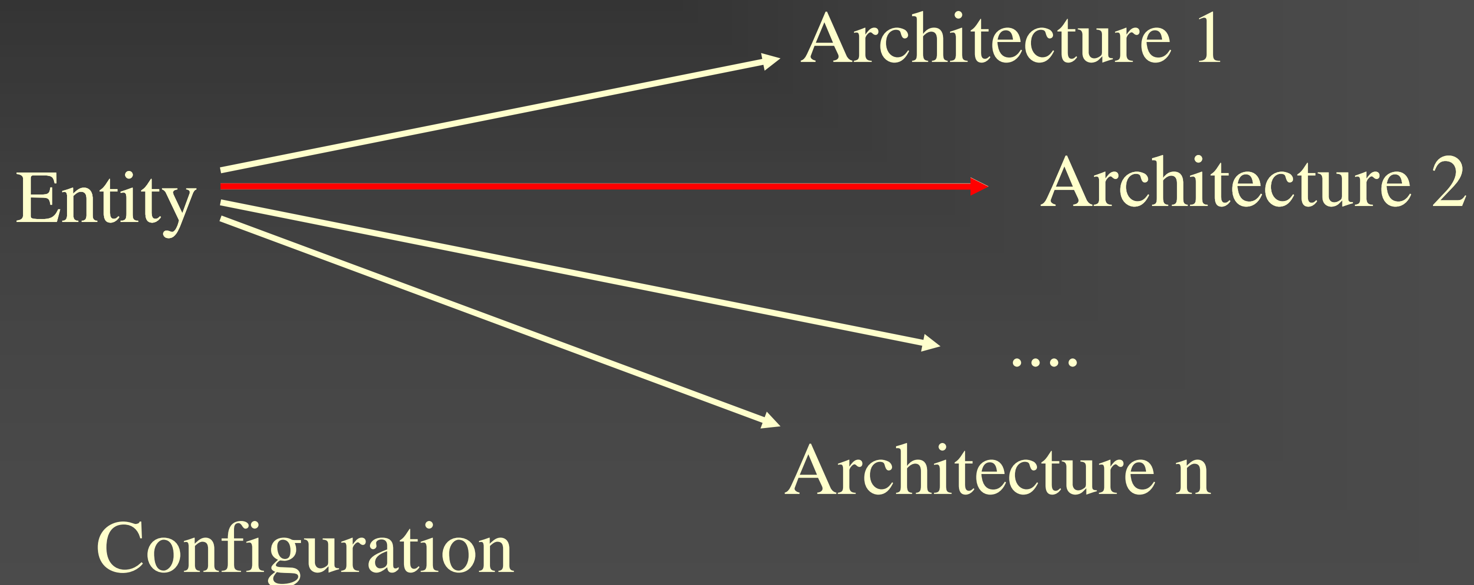
    ....
- Synopsys Equivalent: std_logic_arith

Library „work“

- Standardmäßig legt der Compiler alle compilierten Design-Einheiten in der Library work ab
- Die Library work muss nicht explizit angegeben werden
- Compilierte Packages werden auch in work abgelegt ⇒ vgl. Library
- Packages müssen mit der use Anweisung sichtbar gemacht werden

Zusammenfassung (1)

- Design Unit besteht aus:



- Design Units können in Packages zusammengefasst werden (Komponentendeklarationen)
- Packages können auch Konstanten, Deklarationen, Prozeduren oder Funktionen enthalten
- Packages werden in Libraries zusammengefasst

Zusammenfassung (3)

- Eine Library muss sichtbar gemacht werden:

  LIBRARY my_Lib

- Elemente einer Library müssen bekannt gemacht werden:

  USE my_Lib.HA

- Spezialfälle
  - Library work (automatisch sichtbar)
  - Package std.standard (automatisch sichtbar)
  - Element .all (alle Elemente einer Library/Packages)