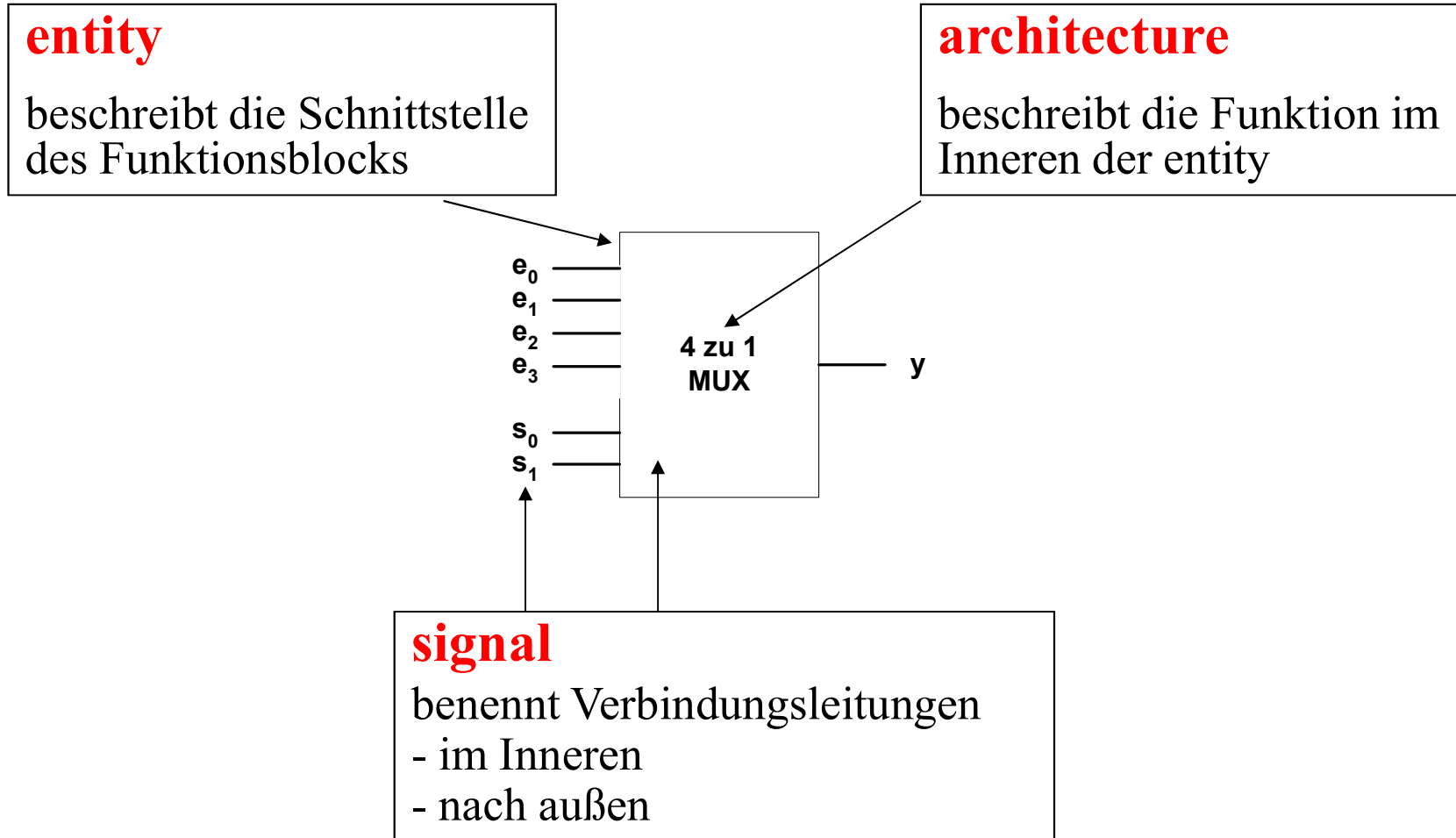


VHDL – Grundlagen



Vergleich VHDL - C

VHDL

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

entity segdecode is
port
(
  x: in std_logic_vector(3 downto 0);
  y: out std_logic_vector(0 to 6)
);
end segdecode;

architecture behaviour of segdecode is
begin
  with x select
  y <= "0100000" when "0000",
       "1111100" when "0001",
       "0001010" when "0010",
       "1001000" when "0011",
       "1010100" when "0100",
       "1000001" when "0101",
       "0000001" when "0110",
       "1111000" when "0111",
       "0000000" when "1000",
       "1010000" when "1001",
       "0000000" when others;
end behaviour;
```

C

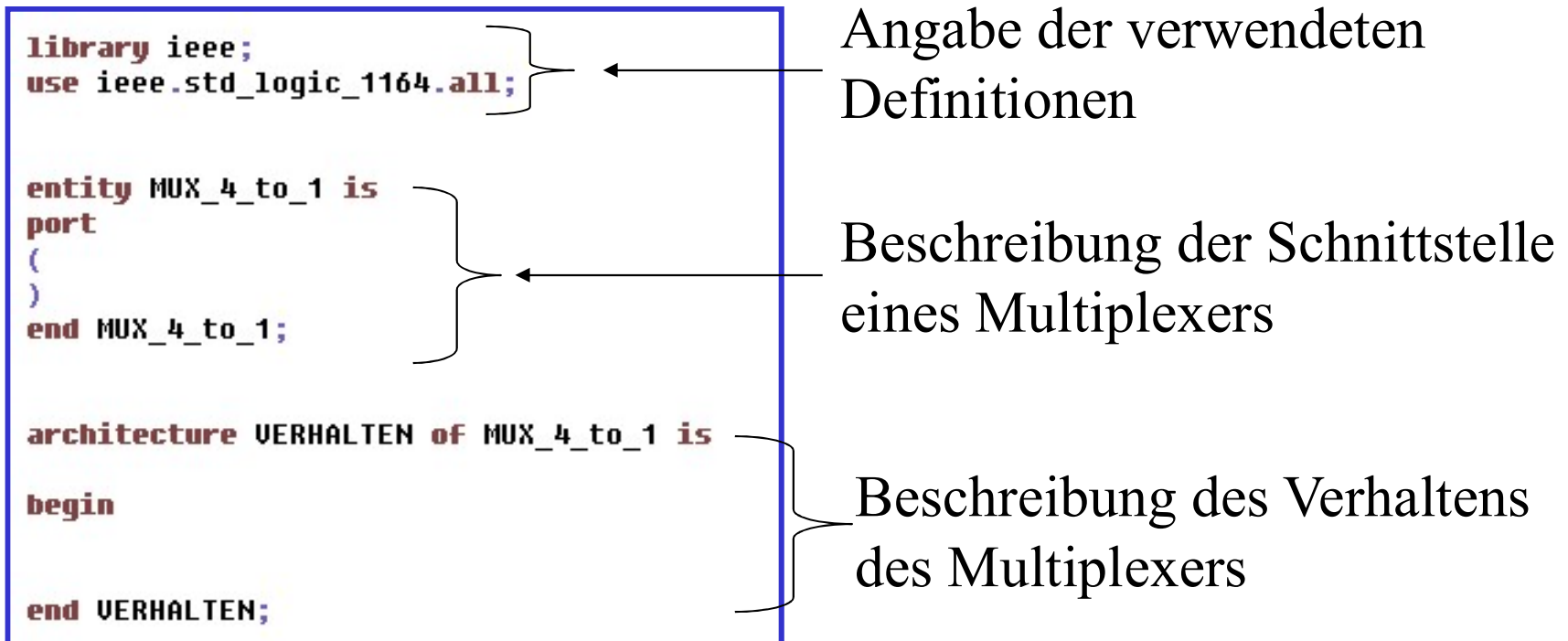
```
#include <stdio.h>

unsigned char siebensegment(unsigned int zahl)
{
  unsigned char segment;

  switch (zahl)
  {
    case 0:      segment = 0x20; break;
    case 1:      segment = 0x7c; break;
    case 2:      segment = 0x0a; break;
    case 3:      segment = 0x48; break;
    case 4:      segment = 0x54; break;
    case 5:      segment = 0x41; break;
    case 6:      segment = 0x01; break;
    case 7:      segment = 0x78; break;
    case 8:      segment = 0x00; break;
    case 9:      segment = 0x50; break;
    default:     break;
  }

  return segment;
}
```

VHDL - Dateiaufbau



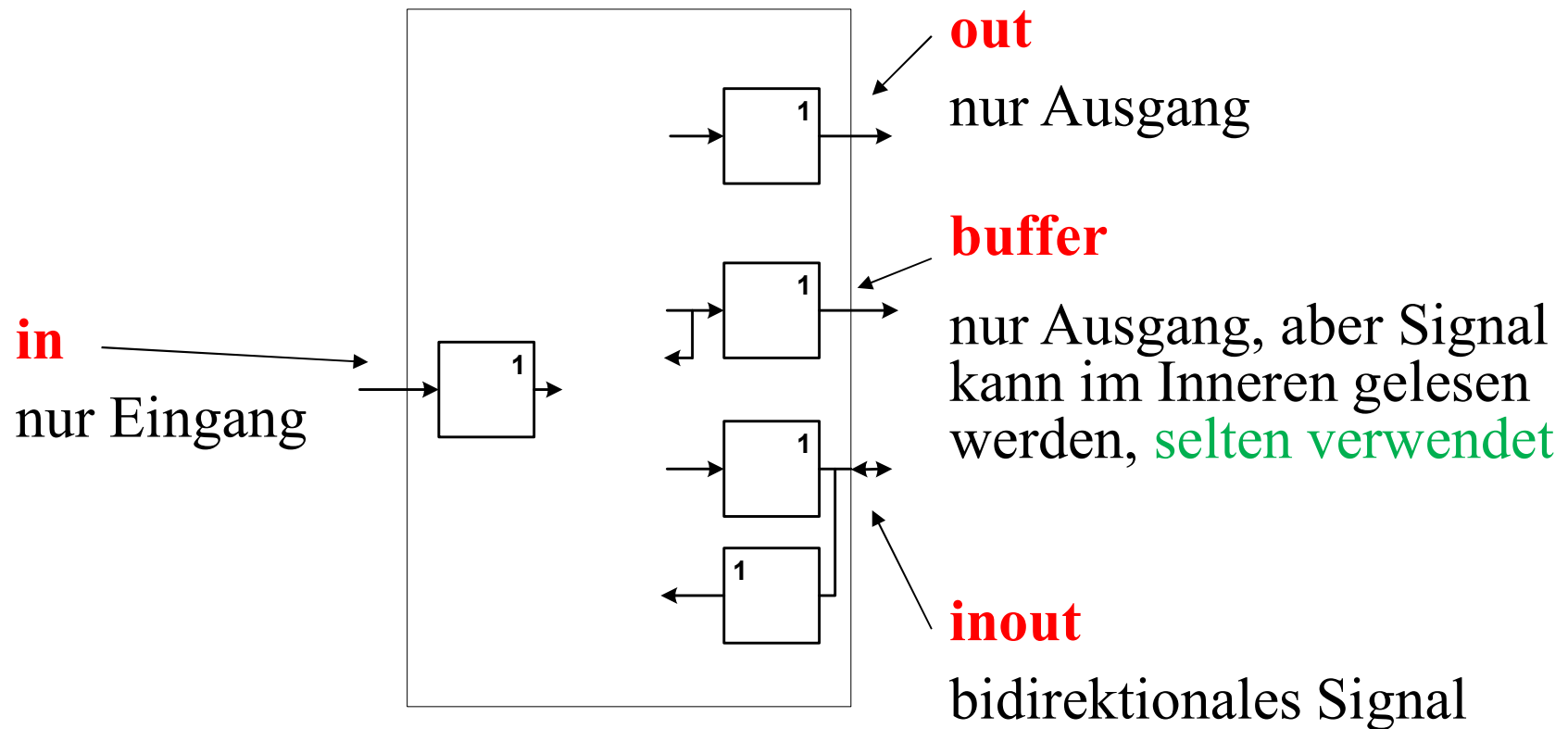
Entity – Aufbau und Syntax

```
entity name is
  port( Liste der Anschlüsse );
  generic( Liste der Parameter); -- optional
end name;                       -- name ist hier optional
```

Anschlüsse, ein Listenelement (Reihenfolge egal)
Name, Name: Richtung Typ;

```
entity demo is
port
(
  x3, x2, x1, x0: in  std_logic;
  y, z:           out std_logic -- kein ; (bis VHDL2008)
);
end;
```

Ports - Richtungen



Architecture – Aufbau und Syntax

```
architecture name of entity-name is
  -- Deklarationen für diese architecture, kann leer sein

begin
  -- Nebenläufige Umgebung (Reihenfolge der Anweisungen ist egal)
  -- Signale aus der entity (ports) sind hier sichtbar
end name; -- name ist hier optional
```

```
architecture mfm of demo is
begin

  y <= x0 and x1;
  z <= x2 and x3;

end;
```

Signale und Konstanten

Signale sind Verbindungsleitungen **mit Zeitverlauf**
Sie können sich also ändern.

Konstanten sind unveränderliche Werte

```
signal name: typ;           -- ohne Startwert  
signal name: typ := startwert; -- mit Startwert  
constant name: typ := fester_wert;
```

```
signal zwischensignal: std_logic;  
constant HIGH: bit := '1'; -- die 1 in Hochkommas (Literal)
```

Konstante Werte

Es gibt

- **Literale** (einzelne Zeichen)
- **Strings** (aus Literalen)
- **Zahlen**
- **Symbole** (wie Namen)

```
y <= '1';      -- y ist ein einzelnes Signal, z.B. ein bit
y <= 'Z';      -- y ist ein einzelnes Signal, z.B. ein std_logic
Y <= "0Z";     -- y ist ein Feld mit zwei Elementen
Y <= 1;        -- y ist eine Zahl
Y <= true;     -- y ist ein einzelnes Signal, Typ boolean
```


Vektoren (Sonderform eines Feldes)

- **Ist ein vordefiniertes Feld:**
 - `bit_vector`, `std_logic_vector`,
 - VHDL 2008: `boolean_vector`, `integer_vector`, `real_vector`
- **Hat **Startindex**, **Endindex** und **Richtung****
 - `signal x, y: bit_vector(0 to 7) ;`
 - `signal z: std_logic_vector(15 downto 4);`
- **Kann wie **ein** Signal verwendet werden**
 - `x <= x(0 to 3) and x(4 to 7); x <= "10010010";`
 - VHDL 2008: `r <= and x;` -- alle Elemente in x werden mit *und* verknüpft, r ist dann ein **einzelnes** Signals

Vektoren

- Der Name alleine ist der gesamte Vektor
- Einzelne Elemente werden mit **name (index)** herausgegriffen
- **Zusammenhängende Teile** können herausgegriffen werden
 - name (**index1 to index2**)
 - name (**index1 downto index2**)
- Mit **&** können Teile „zusammengeklebt“ werden
 - v3 <= v1 & v2; -- v3 ist ein Vektor, v1 und v2 können es auch sein**
- Eine Zerlegung in ein Aggregat **()** ist möglich
 - (a, b, c, d) <= name; -- name ist ein Vektor**