PLD Entwurf mit VHDL

Kombinatorische Logik

Funktionsbeschreibung kombinatorische Logik

ARCHITECTURE akomb OF ekomb IS

BEGIN

concurrent signal assignments

process statements

other concurrent assignments

END akomb

Funktionsbeschreibung kombinatorische Logik

Concurrent Signal Assignments Statements

- Simple Signal Assignments
- Conditional Signal Assignments
- Selected Signal Assignments

Kombinatorische Logik
kombinatorische Logik

Simple Signal Assignments

Bsp.:

```
y0<=x0 and not x1 or x2;
y1<=x2;
```

Kombinatorische Logik
Signal - Zuweisung

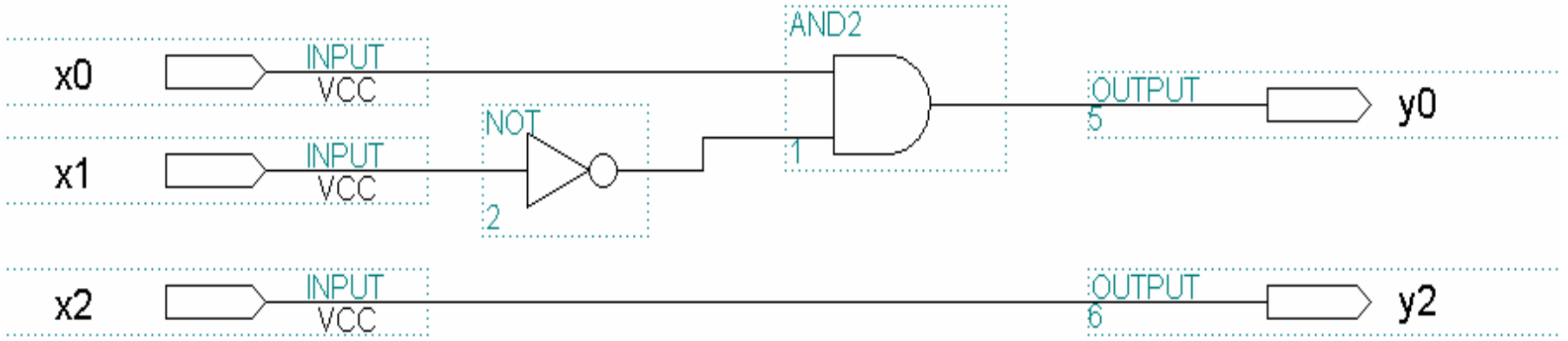
```
-- Simple Signal Assignment

ENTITY simp_sig IS
PORT
(
   x0,x1,x2          : IN        BIT;
   y0,y1             : OUT       BIT
);
END simp_sig;
```

Kombinatorische Logik
Signal - Zuweisung

```
-- Simple Signal Assignment
ARCHITECTURE asimp_sig OF simp_sig IS
BEGIN
   y0<= x0 and not x1;
   y1<= x2;
END asimp_sig;
```

Kombinatorische Logik



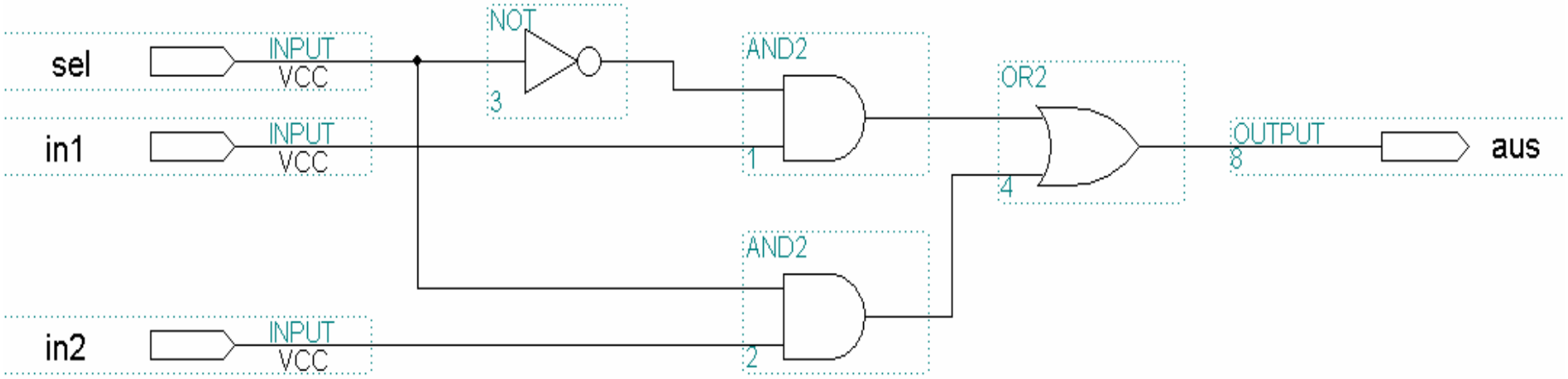
Kombinatorische Logik
Bedingte Signal - Zuweisung

```
ENTITY cond_sig IS
   PORT(
        in1,in2,sel          : IN   BIT;
        aus                         : OUT       BIT);
END cond_sig;
```

Kombinatorische Logik
Bedingte Signal - Zuweisung

```
ARCHITECTURE acond_sig OF cond_sig IS
BEGIN
   auswahl:
        aus <= in1 WHEN sel='0' ELSE in2;
END acond_sig;
```

Kombinatorische Logik
Bedingte Signal - Zuweisung



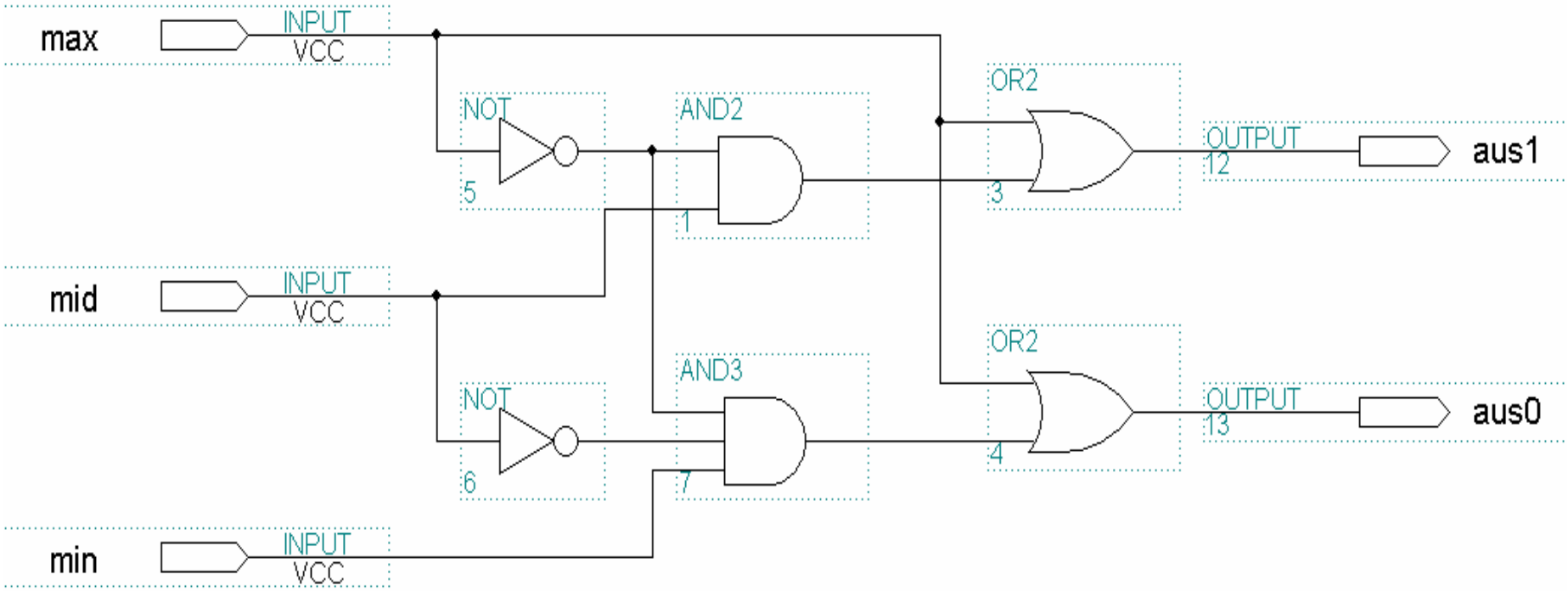
Kombinatorische Logik
Bedingte Signal - Zuweisung

```
-- Mehrfache Entscheidung
ENTITY cond_sigm IS
PORT
(
   max,mid,min  : IN   BIT;
   aus          : OUT INTEGER RANGE 0 TO 3
   );
END cond_sigm;
```

Kombinatorische Logik
Bedingte Signal - Zuweisung

```
ARCHITECTURE acond_sigm  OF
   cond_sigm  IS
BEGIN
   merfach_auswahl:
        aus <=       3      WHEN max = '1' ELSE
                     2      WHEN mid = '1' ELSE
                     1      WHEN min = '1' ELSE
                     0;
END acond_sigm ;
```

Kombinatorische Logik
Bedingte Signal - Zuweisung



Kombinatorische Logik
Signal – Zuweisung mit SELECT

```
-- Multiplexer
ENTITY select_sig IS
PORT(
   d0,d1,d2,d3     : IN   BIT;
   sel             : IN   INTEGER RANGE 0
   TO 3;
   aus             : OUT       BIT );
END select_sig;
```

Kombinatorische Logik
Signal – Zuweisung mit SELECT

```
ARCHITECTURE aselect_sig  OF select_sig  IS
BEGIN
   select_auswahl:
WITH sel SELECT
   aus <=    d0     WHEN 0,
             d1     WHEN 1,
             d2     WHEN 2,
             d3     WHEN 3;
END aselect_sig;
```

Kombinatorische Logik
Signal – Zuweisung mit SELECT

```
-- Dekoder
ENTITY select_dek IS
PORT(
   sel   : IN INTEGER RANGE 0 TO 3;
   aus   : OUT      INTEGER RANGE 0 TO 15 );
END select_dek;
```

Kombinatorische Logik
Signal – Zuweisung mit SELECT

```
ARCHITECTURE aselect_dek  OF select_dek  IS
BEGIN
select_auswahl:
WITH sel SELECT
   aus <=    1       WHEN 0,
             2       WHEN 1,
             4       WHEN 2,
             8       WHEN 3;
END aselect_dek;
```

Kombinatorische Logik
Arithmetik

```
-- Addition und Subtraktion
ENTITY conc_arin IS
PORT(
    op1,op2    : IN INTEGER RANGE 0 TO 255;
    cin        : IN INTEGER RANGE 0 TO 1;
    opcod      : IN BIT;
    erg        : OUT INTEGER RANGE 0 TO 511);
END conc_arin;
```

Kombinatorische Logik - Arithmetik
Signal – Zuweisung mit SELECT

```
ARCHITECTURE aconc_arin  OF conc_arin  IS
BEGIN
   select_auswahl:
WITH opcod SELECT
   erg <=      op1+op2+cin         WHEN '0',
               op1-op2-cin         WHEN '1';
END aconc_arin;
```

Kombinatorische Logik - Arithmetik
Signal – Zuweisung mit SELECT

Bei der Realisierung eines n – Bit Addierers sind folgende Probleme zu lösen:

- n sollte ein variabler Parameter sein
- Variable Parameter werden in VHDL mit Hilfe von **generics** angegeben.
- Die Arithmetischen Operatoren sind in der Regel nur für Zahlen-Typen wie **unsigned, signed, integer** definiert. Während die Ein- und Ausgänge vom Typ **std_logic** bzw. **std_logic_vector** sind

Kombinatorische Logik - Arithmetik
Signal – Zuweisung mit SELECT

- in der Architektur sollte **signal** entsprechenden Typs verwendet werden
- Typumwandlungsfunktionen müssen verwendet werden gearbeitet werden.
- Die Vektoren für die Operanden haben eine andere Dimension als der Vektor für das Ergebnis
- arithmetischen Operatoren verlangen Vektoren gleicher Dimension.
- Operandenvektoren müssen mit Hilfe der Operation **&** und einer führenden ´**0**´ auf die Dimension des Ergebnisvektors gebracht werden.

Kombinatorische Logik - Arithmetik

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

entity adder is
  generic(n: NATURAL := 4);   --Parameter n
  Port (op1 :     in std_logic_vector(n-1 downto 0);
      op2 :       in std_logic_vector(n-1 downto 0);
      cin :       in std_logic;
      sum :       out std_logic_vector(n-1 downto 0);
      cout  : out std_logic);
end adder;
```

Kombinatorische Logik - Arithmetik

```
architecture Behavioral of adder is
 signal summe : unsigned(n downto 0);
 signal carry : unsigned(n downto 0);
  constant nulvek : unsigned(n-1 downto 0)
  :=(others => '0');
begin
   carry<=(nulvek&cin);  -- Vektor für cin
-- Typumwandlung std_logic_vector nach unsigned
   summe<= ('0'&unsigned(op1)) +
  ('0'&unsigned(op2))+carry;
-- Typumwandlung unsigned nach  std_logic_vector
 sum<=std_logic_vector(summe(n-1 downto 0));
 cout<=summe(n);
end Behavioral;
```

Kombinatorische Logik - Arithmetik

In einigen Entwurfssystemen gelten numerische Operatoren auch für std_logic bzw std_logic_vector.

Das nächste Beispiel ist mit den Xilinx – Tools und Quartus von Altera realisierbar.

Das liegt daran, dass Operatoren überladen werden können. Das Überladen kann in einer Library definiert werden.

Kombinatorische Logik - Arithmetik

```
package std_logic_unsigned is
Bsp.:
    function "+"(L: STD_LOGIC_VECTOR; R:
  STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
    function "+"(L: STD_LOGIC_VECTOR; R:
  INTEGER) return STD_LOGIC_VECTOR;
    function "+"(L: INTEGER; R:
  STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
    function "+"(L: STD_LOGIC_VECTOR; R:
  STD_LOGIC) return STD_LOGIC_VECTOR;
    function "+"(L: STD_LOGIC; R:
  STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
```

Kombinatorische Logik - Arithmetik

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity add_nx is
    Port ( opa : in std_logic_vector(3
  downto 0);
           opb : in std_logic_vector(3
  downto 0);
           cin : in std_logic;
           sum : out std_logic_vector(3
  downto 0);
           cout : out std_logic);
end add_nx;
```

Kombinatorische Logik - Arithmetik

```
architecture Behavioral of add_nx is
  signal summe : std_logic_vector(4
                                  downto 0);
begin
-- Vektorerweiterung für Simulator
  erforerlich
  summe<=('0'&opa)+('0'&opb)+
         ("0000"&cin);
  cout<=summe(4);
  sum<=summe(3 downto 0);
end Behavioral;
```

Kombinatorische Logik - Arithmetik

Der 1 Bit Addierer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ADD_1BIT is
    Port ( opa : in std_logic;
           opb : in std_logic;
           cin : in std_logic;
           sum : out std_logic;
           cout : out std_logic);
end ADD_1BIT;
```

Kombinatorische Logik - Arithmetik

```
architecture Behavioral of ADD_1BIT is

begin
  sum<=opa xor opb xor cin;
  cout<= (opa and opb) or (opa and
  cin) or (opb and cin);
end Behavioral;
```

Kombinatorische Logik - Arithmetik

Die Testbench für den 1 Bit Addierer

```
-- VHDL Test Bench Created from source file
    add_1bit.vhd -- 19:53:22 09/12/2004
-- Notes:
-- This testbench has been automatically generated
    using types std_logic and
-- std_logic_vector for the ports of the unit under test.
    Xilinx recommends
-- that these types always be used for the top-level I/O of
    a design in order
-- to guarantee that the testbench will bind correctly to
    the post-implementation
-- simulation model.
```

Kombinatorische Logik - Arithmetik

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

-- Entity der Testeinrichtung

ENTITY add_1bit_add_1bitt_vhd_tb IS
END add_1bit_add_1bitt_vhd_tb;
```

Kombinatorische Logik - Arithmetik

```
ARCHITECTURE behavior OF
   add_1bit_add_1bitt_vhd_tb IS

   COMPONENT add_1bit
   PORT(
        opa : IN std_logic;
        opb : IN std_logic;
        cin : IN std_logic;
        sum : OUT std_logic;
        cout : OUT std_logic
        );
   END COMPONENT;
```

Kombinatorische Logik - Arithmetik

Tester und zu testende Einheit verbinden

```
SIGNAL opa :  std_logic;
SIGNAL opb :  std_logic;
SIGNAL cin :  std_logic;
SIGNAL sum :  std_logic;
SIGNAL cout :  std_logic;

BEGIN
   uut: add_1bit PORT MAP(
         opa => opa,
         opb => opb,
         cin => cin,
         sum => sum,
         cout => cout
   );
```

Kombinatorische Logik - Arithmetik

```
-- *** Test Bench - User Defined Section ***
 tb : PROCESS
 BEGIN
  opa<='0';
  opb<='0';
  cin<='0';
  wait for 50 ns;
  opa<='1';
  opb<='0';
  cin<='0';
  wait for 50 ns;
  opa<='0';
  opb<='1';
             wait; -- will wait forever
  END PROCESS;
END;
```

Kombinatorische Logik - Arithmetik

N Bit Adder mit generate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

entity ADD_nBIT is
generic (n: NATURAL := 4);
    Port ( opa : in std_logic_vector(n-1 downto 0);
           opb : in std_logic_vector(n-1 downto 0);
           cin : in std_logic;
           sum : out std_logic_vector(n-1 downto 0);
           cout : out std_logic);
end ADD_nBIT;
```

Kombinatorische Logik - Arithmetik

```
 architecture Behavioral of ADD_nBIT is
  signal carry : std_logic_vector(0 to n);
begin
gen : for i in 0 to n-1 generate
  ind_0 : if i = 0 generate
      add_0 : entity WORK.ADD_1BIT port map
            (opa(i),opb(i),cin,sum(i),carry(i+1));
  end generate ind_0;
  indn_1: if i = n-1 generate
      add_n_1 : entity WORK.ADD_1BIT port map
            (opa(i),opb(i),carry(i),sum(i),cout);
  end generate indn_1;
  ind_m: if i >0 and i<n-1 generate
      add_m : entity WORK.ADD_1BIT port map
      (opa(i),opb(i),carry(i),sum(i),carry(i+1)
  end generate ind_m;
end generate gen;
end Behavioral;
```

Kombinatorische Logik - Arithmetik

Bemerkungen:

- **if generate hat keinen else Zweig**
- **Instanzen und generate Anweisungen müssen immer ein Label haben**

Kombinatorische Logik
mit Process Statement

```
__process_label:
PROCESS (signalname, signalname, signalname)
   VARIABLE __variable_name : Typ;
BEGIN
   --  Signal Assignment Statement
   --  Variable Assignment Statement
   --  Procedure Call Statement
   --  If Statement
   --  Case Statement
   --  Loop Statement
END PROCESS __process_label;
```

Kombinatorische Logik
mit Process Statement

```
-- Multiplexer 4 zu 1 mit enable
ENTITY process_kom_mux IS
PORT(
 adr                 : IN INTEGER Range 0 TO 3;
 muxen               : IN BIT;
 di0,di1,di2,di3     : IN BIT;
 maus                : OUT       BIT);
END process_kom_mux;
```

```
ARCHITECTURE aprocess_kom_mux OF process_kom_mux IS
BEGIN
process_mpx:
PROCESS
BEGIN
 IF muxen='1' THEN
  CASE adr IS
     WHEN 0 => maus<=di0;
     WHEN 1 => maus<=di1;
     WHEN 2 => maus<=di2;
     WHEN 3 => maus<=di3;
     WHEN OTHERS => maus<='0';
  END CASE;
 ELSE maus<='0';
 END IF;
END PROCESS process_mpx;
END aprocess_kom_mux;
```

Kombinatorische Logik
mit Process Statement

```
-- Arithmetik
ENTITY pro_k_ari IS
 PORT(
   op1,op2 : IN INTEGER RANGE 0 to 255;
   code    : IN BIT;
   erg     : OUT  INTEGER RANGE 0 to 255);
END pro_k_ari;
```

Kombinatorische Logik
mit Process Statement

```
ARCHITECTURE apro_k_ari OF pro_k_ari IS
BEGIN
 pro_add_sub:
 PROCESS (op1,op2,code)
 BEGIN
  IF code='1' THEN
        erg <= op1+op2;
  ELSE
        erg <= op1-op2;
  END IF;
 END PROCESS pro_add_sub;
END apro_k_ari;
```

Kombinatorische Logik mit Process Statement

```
-- ADDER mit cin und cout
ENTITY pro_k_ari IS
 PORT(
   op1,op2 : IN INTEGER RANGE 0 TO 255;
   cin     : IN INTEGER RANGE 0 TO 1;
   code    : IN BIT;
   cout    : OUT BIT;
   erg     : BUFFER INTEGER RANGE 0
   TO 511);
END pro_k_ari;
```

Kombinatorische Logik mit Process Statement

```
ARCHITECTURE apro_k_ari OF pro_k_ari IS
BEGIN
 pro_add_sub:
 PROCESS (op1,op2,code)
 BEGIN
  IF code='1' THEN erg <= op1+op2+ cin;
  ELSE        erg <= op1-op2-cin;
  END IF;
    IF erg>255 THEN      cout<='1';
    ELSE cout<='0';
    END IF;
 END PROCESS pro_add_sub;
END apro_k_ari;
```

Kombinatorische Logik
mit Process Statement

```
ENTITY pro_k_par IS
 PORT(
   d          : IN  BIT_VECTOR ( 7 DOWNTO
   0);
   zahl       : OUT INTEGER RANGE 0 TO 8;
   parb       : OUT INTEGER RANGE 0 TO 1);
END pro_k_par;
```

```
ARCHITECTURE apro_k_par OF pro_k_par IS
BEGIN
PROCESS
VARIABLE zvar : INTEGER;
BEGIN
    zvar := 0;
bit_zahl:
 FOR i IN d'RANGE LOOP
  IF d(i)='1' THEN
    zvar:=zvar+1;
  END IF;
 END LOOP bit_zahl;
zahl<=zvar;
parb<=zvar;
END PROCESS;
END apro_k_par;
```

Kombinatorische Logik Paritätsgenerator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;
entity parit_gen is
    generic(n: NATURAL := 8);
    Port ( dawo : in std_logic_vector(n-1 downto 0);
           pabit : out std_logic;
                   bitza          :out std_logic_vector(3
    downto 0));
end parit_gen;
```

Kombinatorische Logik Paritätsgenerator

```
architecture Behavioral of parit_gen is
begin
process (dawo)
    variable even : std_logic;
    variable bitzav          : std_logic_vector(3 downto 0);
   begin
          even := '0';        -- Wertzuweisung an Variable
          bitzav:="0000";
    schleife:   for I in dawo'range loop
          if dawo(i) = '1' then
                even := not even;
                bitzav:=bitzav+1;
          end if;
    end loop;
    pabit<=even;
    bitza<=bitzav;
end process;
end Behavioral;
```

Kombinatorische Logik
Process

```
process_label:
PROCESS (signal_name, signal_name, signal_name)
BEGIN
    --  Signal Assignment Statement
    --  Variable Assignment Statement
    --  Procedure Call Statement
    --  If Statement
    --  Case Statement
    --  Loop Statement
END PROCESS process_label;
```

Kombinatorische Logik
Wahrheitstabelle

```
ENTITY table IS
   PORT(
        x       : IN  BIT_VECTOR(2 DOWNTO
   0);
        y       : OUT      BIT);
END table;
```

Kombinatorische Logik
Wahrheitstabelle

```
ARCHITECTURE atable OF table IS
BEGIN
tafel:
WITH x SELECT
   y <=      '1' WHEN "001",
             '1' WHEN "011",
             '1' WHEN "111",
             '0' WHEN OTHERS;
END atable;
```

Kombinatorische Logik
Tristate

```
library IEEE;
USE IEEE.std_logic_1164.all;


ENTITY bidirekt IS
   PORT(
        bidirein      : IN   STD_LOGIC;
        outen         : IN   STD_LOGIC;
        bidiraus      : OUT       STD_LOGIC;
        treiber             : INOUT    STD_LOGIC);
END bidirekt;
```

Kombinatorische Logik
Tristate

```
ARCHITECTURE abidirekt OF bidirekt IS
BEGIN
-- Zuweisung von 'Z' an einen Ausgang erzeugt
   einen tri-state Ausgang
   treiber <= bidirein WHEN outen = '0'
                       ELSE 'Z';
   bidiraus<= treiber WHEN outen = '1'
                       ELSE 'Z';
END abidirekt;
```

Kombinatorische Logik
Zusammenfassung

Libraries

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Kombinatorische Logik
Zusammenfassung

concurrent signal assignments

Simple Signal Assignments

```
y0<= x0 and not x1;
```

Conditional Signal Assignments

merfach_auswahl:

```
aus <= 3   WHEN max = '1' ELSE
       2   WHEN mid = '1' ELSE
       1   WHEN min = '1' ELSE
       0;
```

Kombinatorische Logik
Zusammenfassung

concurrent signal assignments
Selected Signal Assignments

```
select_auswahl:
        WITH sel SELECT
        aus <=      d0    WHEN 0,
                    d1    WHEN 1,
                    d2    WHEN 2,
                    d3    WHEN 3;
```

Kombinatorische Logik
Zusammenfassung

process statements

generic Parameter

```
entity adder is
generic(n: NATURAL := 4);  --Parameter n
Port ...
```

Kombinatorische Logik
Zusammenfassung

Bildung von zusammengesetzten
- **Signalvektoren**
- **(nulvek & cin)**
- **carry<=(nulvek&cin);**

Bildung von Konstanten im Architekturteil und Initialisierung der Konstanten mit 0
- **constant nulvek : unsigned(n-1 downto 0)**
- **:=(others => '0');**

Kombinatorische Logik
Zusammenfassung

Typumwandlung durch Funktionen (castoperator)

```
-- Typumwandlung std_logic_vector nach unsigned
   summe<= ('0'& unsigned(op1)) + ('0'&
   unsigned(op2))

-- Typumwandlung unsigned nach  std_logic_vector
   sum<=std_logic_vector(summe(n-1 downto 0));
```

Kombinatorische Logik
Zusammenfassung

Die Testbench
Die Testbench für den 1 Bit Addierer

Kombinatorische Logik
Zusammenfassung

Nutzung einer Komponente im Architekturteil

```
COMPONENT add_1bit
PORT(
    opa : IN std_logic;
    opb : IN std_logic;
    cin : IN std_logic;
    sum : OUT std_logic;
    cout : OUT std_logic
    );
END COMPONENT;
```

Kombinatorische Logik
Zusammenfassung

for generate

Funktionsbeschreibung in einem Process

Kombinatorische Logik
Zusammenfassung

Funktionsbeschreibung in einem Process

```
PROCESS (signalname, signalname, signalname)
   VARIABLE __variable_name : Typ;
BEGIN
   --  Signal Assignment Statement
   --  Variable Assignment Statement
   --  Procedure Call Statement
   --  If Statement
   --  Case Statement
   --  Loop Statement
END PROCESS __process_label;
```

Kombinatorische Logik
Zusammenfassung

Variablen in einem Process

Signalattribut range
dawo'range