Professur Schaltkreis- und Systementwurf

TECHNISCHE UNIVERSITÄT CHEMNITZ

Professur Schaltkreis- und Systementwurf

Operators

TECHNISCHE UNIVERSITÄT CHEMNITZ

Operators

logical	and	or	nand	nor	xor	xnor
	not					
relational	=	/=	<	<=	>=	>
shift	sll	srl	sla	sra	rol	ror
arithmetic	+	-				
	*	/	mod	rem		
	**	abs				

sorted on order of increasing precedence (top ⇒ down)

'93 **New operators: xnor, shift operators**

Logical Operators

```
entity LOGIC_OP is
  port (A, B, C, D : in bit;
        Z1         : out bit;
        EQUAL      : out boolean);
end LOGIC_OP;

architecture EXAMPLE of LOGIC_OP is
begin
  Z1 <= A and (B or (not C xor D));
  EQUAL <= A xor B;     -- wrong
end EXAMPLE;
```

id 020 csb

- **Priority**
  - **not (top priority)**
  - **and, or, nand, nor. xor, xnor (equal priority)**
- **Predefined for**
  - **bit, bit_vector**
  - **boolean**
- **Data types have to match**

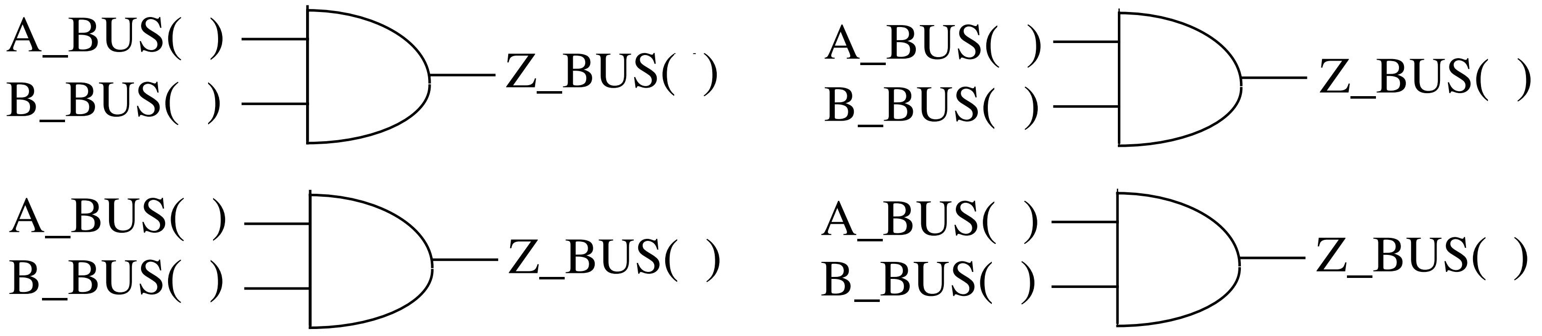
⚠ **Brackets must be used to define the order of evaluation**

Logical Operators with Arrays

```
architecture EXAMPLE of LOGICAL_OP is
  signal A_BUS : bit_vector (3 downto 0);
  signal B_BUS : bit_vector (3 downto 0);
  signal Z_BUS : bit_vector (4 to 7);
begin
  Z_BUS <= A_BUS and B_BUS;
end EXAMPLE;
```

id 021 csb

- Operands of the same length and type
- Assignment via the position of the elements (according to range definition)



Shift Operators: Examples

'93

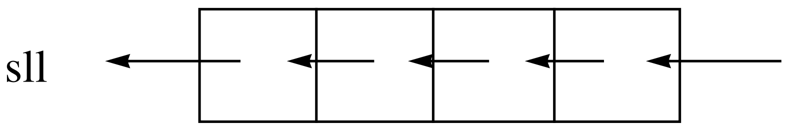
Defined only for one-dimensional arrays of a bit of boolean!

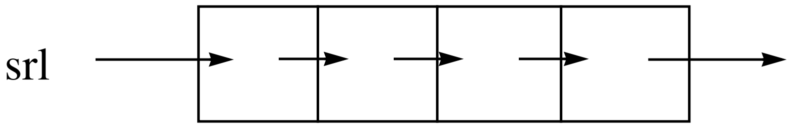
```
signal A_BUS, B_BUS, Z_BUS : bit_vector (3 downto 0);

Z_BUS <= A_BUS sll 2;
Z_BUS <= B_BUS sra 1;
Z_BUS <= A_BUS ror 3;
```

$\longleftarrow$ At the end, the first value of the type is used for filling up

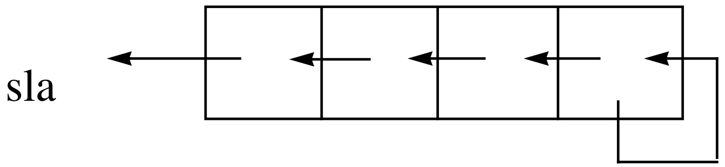
id 022 cs

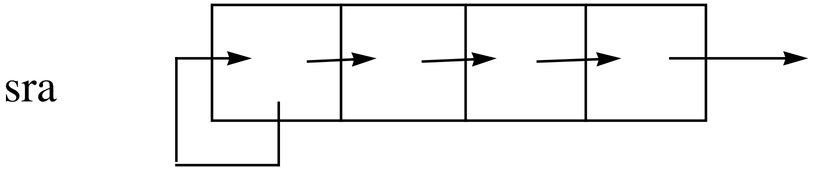
Logical shift



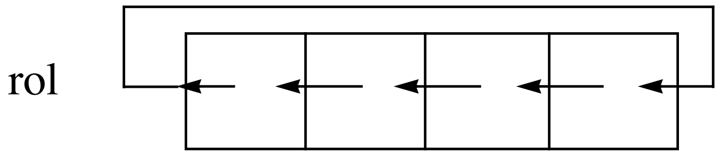


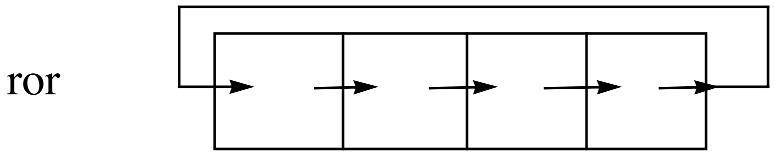
Arithmetic shift





Rotation





Relational Operators

```
architecture EXAMPLE of RELATIONAL_OP is

  signal NR_A, NR_B : integer;

  signal A_EQ_B1, A_EQ_B2 : bit;
  signal A_LT_B           : boolean;

begin
  -- A,B may be of any standard data type
  process (A, B)
  begin
    if (A = B) then
      A_EQ_B1 <= '1';
    else
      A_EQ_B1 <= '0';
    end if;
  end process;

  A_EQ_B2 <= A = B;        -- wrong

  A_LT_B <= B <= A;
end EXAMPLE;
```

id 023

- **Predefined for all standard data types**
- **Result: boolean type (true, false)**

<	less than
<=	less or equal
=	equal
/=	unequal
>=	greater or equal
>	greater

Arithmetic Operators

`+` addition	`–` subtraction
`*` multiplication	`**` exponentiation
`/` division	`mod` modulo
`abs` absolute value	`rem` remainder

```
...
  signal A, B, C : integer;
  signal RESULT : integer;
...
  RESULT <= -A + B * C;
...
```

- **Operands of the same type**
- **Predefined for**
  - **integer**
  - **real** (except mod and rem)
  - **physical types** (e.g. time)
- **Not defined for bit_vector** (undefined number format: unsigned, 2-complement, etc.)
- **Conventional mathematical meaning and priority**
- **'+' and '-' may also be used as unary operators (only one operand)**

Data types

Data Types

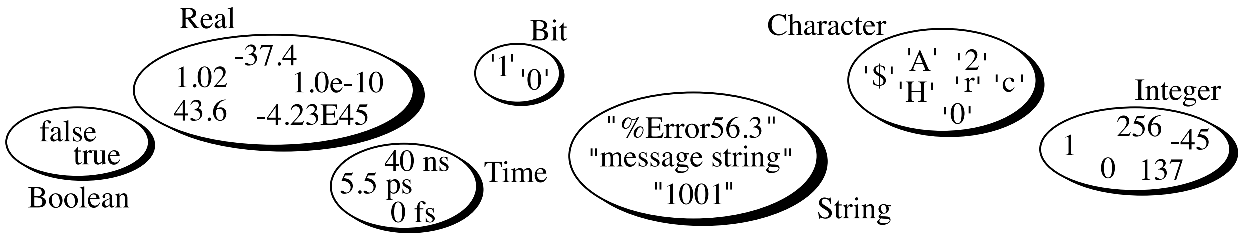
```
entity FULLADDER is
  port(A, B, CARRY_IN : in bit;
       SUM, CARRY     : out bit);
end FULLADDER;

architecture MIX of FULLADDER is
  component HALFADDER
    port(A, B        : in bit;
         SUM, CARRY : out bit);

  signal W_SUM, W_CARRY1, W_CARRY2 : bit;

begin
  HA1: HALFADDER
    port map(A, B, W_SUM, W_CARRY1);

  HA2: HALFADDER
    port map(CARRY_IN, W_SUM,
             SUM, W_CARRY2);

  CARRY <= W_CARRY1 or W_CARRY2;

end MIX;
```

- **Every signal has a type**
- **Type specifies possible values**
- **Types has to be defined at signal declaration ...**
- **... either in**
  - **entity: port declaration, or in**
  - **architecture: signal declaration**
- **Types have to match**

Standard Data Types

```
package STANDARD is
   type BOOLEAN is (FALSE,TRUE);
   type BIT is (‘0‘,‘1‘);
   type CHARACTER is (-- ascii set);
   type INTEGER is range
        -- implementation_defined
   type REAL is range
        -- implementation_defined
   -- BIT_VECTOR, STRING, TIME
end STANDARD;
```

- **Every type has a number of possible values**
- **Standard types are defined by the language**
- **User can define his own types**

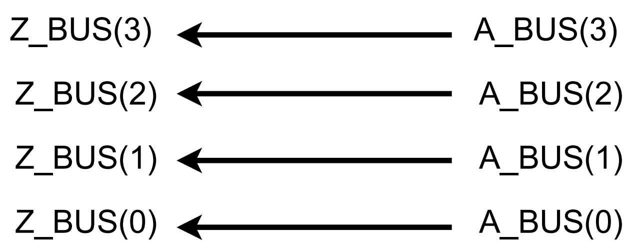


'93 **New types added to the “standard“ package, e.g. umlauts**

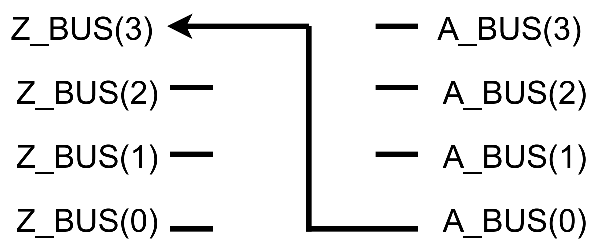
Definition of Arrays

```
Signal A_BUS, Z_BUS : bit_vector (3 downto 0);
```

```
Z_BUS <= A_BUS
```



```
Z_BUS(3) <= A_BUS(0)
```



- **Collection of signals of the same type**
- **Predefined arrays**
  - **bit_vector (array of bit)**
  - **string (array of character)**
- **Unconstrained arrays: definition of actual size during signal/port declaration**

‘integer’ and ‘bit’ Types

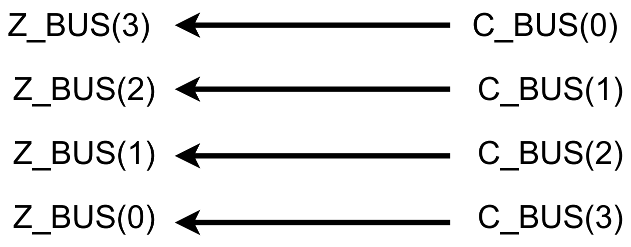
```
architecture EXAMPLE_1 of
DATATYPES
 is
  signal SEL :  bit;
  signal A, B, Z :
          integer range 0 to 3;

begin
  A <= 2;
  B <= 3;
  process(SEL,A,B)
  begin
    if SEL = ‘1’ then
      Z <= A;
    else
      Z <= B;
    end if;
  end process;
end EXAMPLE_1;
```

```
architecture EXAMPLE_2 of
DATATYPES
 is
  signal SEL : bit;
  signal A, B, Z :
          bit_vector(1 downto 0);

begin
    A <= "10";
    B <= "11";
    process(SEL,A,B)
    begin
       if SEL = ‘1’ then
          Z <= A;
       else
          Z <= B;
       end if;
    end process;
end EXAMPLE_2;
```

- Example for using ‘bit’ and ‘integer’

Assignments with Array Types

```
architecture EXAMPLE of ARRAYS is
  signal Z_BUS : bit_vector (3 downto 0);
  signal C_BUS : bit_vector (0 to 3);
begin
  Z_BUS <= C_BUS;
end EXAMPLE;
```



Elements are assigned according to their position, not their number

The direction of arrays should always be defined the same way

2.3.6 Bit String Literals

```
architecture EXAMPLE of ASSIGNMENT is

signal Z_BUS : bit_vector (3 downto 0);
  signal BIG_BUS :bit_vector (15 downto 0);

begin
  -- legal assignments:
  Z_BUS(3)    <= '1';
  Z_BUS       <= "1100";

  Z_BUS       <= b"1100";
  Z_BUS       <= x"c";
  Z_BUS       <= X"C";
  BIG_BUS     <= B"0000_0001_0010_0011";

end EXAMPLE;
```

- **Single bit values are enclosed in '.'**
- **Vector values are enclosed in "..."**
  - **optional base specification (default: binary)**
  - **Values may be separated by underscores to improve readability**

Different specification of single bits an bit vectors

’93 **Valid assignments for the data type ‘bit’ are also valid for all character arrays, e.g. ‘std_(u)logic_vector’**

- **Concatenation operator: &**

```
architecture EXAMPLE_1 of CONCATENATION is
  signal BYTE : bit_vector (7 downto 0);
  signal A_BUS, B_BUS :bit_vector (3 downto 0);

begin

  BYTE   <= A_BUS & B_BUS;

end EXAMPLE_1;
```

```
architecture EXAMPLE_1 of CONCATENATION is
  signal Z_BUS : bit_vector (3 downto 0);
  signal A_BIT, B_BIT, C_BIT, D_BIT :bit;

begin

  Z_BUS   <= A_BIT & B_BIT & C_BIT & D_BIT;

end EXAMPLE_2;
```

- **Resulting signal assignment:**

$$\begin{tikzcd}
\text{BYTE(7)} & \text{A\_BUS(3)} \arrow[l] \\
\text{BYTE(6)} & \text{A\_BUS(2)} \arrow[l] \\
\text{BYTE(5)} & \text{A\_BUS(1)} \arrow[l] \\
\text{BYTE(4)} & \text{A\_BUS(0)} \arrow[l] \\
\text{BYTE(3)} & \text{B\_BUS(3)} \arrow[l] \\
\text{BYTE(2)} & \text{B\_BUS(2)} \arrow[l] \\
\text{BYTE(1)} & \text{B\_BUS(1)} \arrow[l] \\
\text{BYTE(0)} & \text{B\_BUS(0)} \arrow[l]
\end{tikzcd}$$

The Concatenation operator '&' is allowed on the right side of the signal assignment operator '<=' , only

Slices of Array

```
architecture EXAMPLE of SLICES is
  signal BYTE : bit_vector (7  downto 0);
  signal A_BUS : bit_vector (3 downto 0);
  signal Z_BUS : bit_vector (3 downto 0);
  signal A_BIT : bit;
begin

  BYTE (5 downto 2) <= A_BUS;
  BYTE (5 downto 0) <= A_BUS;      -- wrong

  Z_BUS (1 downto 0) <= '0' & A_BIT;
  Z_BUS <= BYTE (6 downto 3);
  Z_BUS (0 to 1) <= '0' & A_BIT;   -- wrong

  A_BIT   <= A_BUS (0);

end EXAMPLE;
```

id 011

- Slices select elements of arrays

Arrays: Size must match on both sides of an assignment

The direction of the 'slice" and of the array must match

Professur Schaltkreis- und Systementwurf

VHDL Language and Syntax

- **Extended Data Types - Multi valued Types - std_logic data type**

TECHNISCHE UNIVERSITÄT CHEMNITZ

Multi Valued Types - BIT Type Issues

```
Type bit is (‘0’, ‘1’);
```

- Values ‘0’ and ‘1’, only
  - Default value ‘0’
- Additional requirements for simulation and synthesis
  - Uninitialized
  - High impedance
  - Undefined
  - ‘don’t care’
  - Different driver strengths

Multi Valued Types

- **Multi valued logic systems are declared via new data types**
  - **Uninitialized**
  - **Unknown**
  - **High impedance**
  - **...**
- **IEEE-standard**
  - **9-valued logic-system defined and accepted by the IEEE**
  - **Standard IEEE 1164 (STD_LOGIC_1164)**

IEEE Standard Logic Type

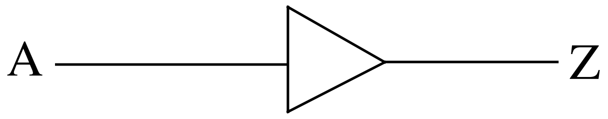
```
type STD_ULOGIC is (
  'U',   -- uninitialized
  'X',   -- strong 0 or 1 (= unknown)
  '0',   -- strong 0
  '1',   -- strong 1
  'Z',   -- high impedance
  'W',   -- weak 0 or 1 (= unknown)
  'L',   -- weak 0
  'H,',  -- weak 1
  '-',   -- don't care);
```

- **9 different signal states**
- **Superior simulation results**
- **Bus modeling**
- **"ASCII-characters"**

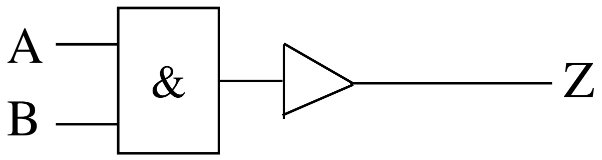
- **Defined in package 'IEEE.std_logic_1164'**
- **Similar data type 'std_logic' with the same values**
- **Array types available: 'std_(u)logic_vector', similar to 'bit_vector'**
- **All 'bit' operators available**

The IEEE standard should be used in VHDL designs

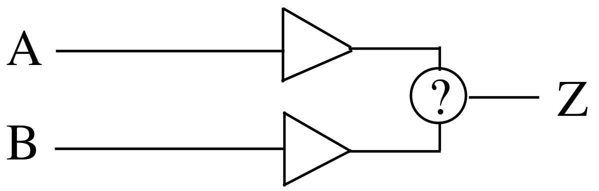
Resolved and Unresolved Types



Z <= A;



Z <= A and B;



Z <= A;
Z <= B;

= **resolution function**

```
architecture EXAMPLE of ASSIGNMENT is
  signal A, B, Z : bit;
  signal INT     : integer;
begin
  Z <= A;
  Z <= B;
  Z <= INT;  -- wrong
end EXAMPLE;
```

- **Signal assignments are represented by drivers**
- **Unresolved data type: only one driver**
- **Resolved data type: possibly several drivers per signal**
- **Conditions for valid assignments**
  - **types have to match**
  - **Resolved type, if more than 1 concurrent assignment**

Std_Logic_1164 Package

```
PACKAGE std_logic_1164 IS
-----------------------------------
-- logic state system (unresolved)
-----------------------------------
TYPE STD_ULOGIC IS (
   'U',     -- uninitialized
   'X',     -- Forcing Unknown
   '0',     -- Forcing 0
   '1',     -- Forcing 1
   'Z',     -- High Impedance
   'W',     -- Weak Unknown
   'L',     -- Weak 0
   'H',     -- Weak 1
   '-',     -- don't care);

-----------------------------------
unconstrained array of std_ulogic
  -- for use with the resolution
  -- function
-----------------------------------
 TYPE std_ulogic_vector IS
    ARRAY(NATURAL RANGE <>) OF
    std_ulogic;
```

```
-----------------------------------
-- resolution function
-----------------------------------
FUNCTION resolved
 (s : std_ulogic_vector )
 RETURN std_ulogic;
-----------------------------------
-- ** industry standard logic type **
-----------------------------------
SUBTYPE std_logic IS resolved
   std_ulogic;
-----------------------------------
-- unconstrained array of std_logic
-- for use in declaring signal arrays
-----------------------------------
TYPE std_logic_vector IS
ARRAY(NATURAL RANGE <>)OF std_logic;

END std_logic_1164;
```

Resolution Function

```
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;
CONSTANT resolution_table : stdlogic_table :=(
  -- U   X   0   1   Z   W   L   H   -   ------
  -- ----------------------------------------
    ('U','U','U','U','U','U','U','U','U'), -- U
    ('U','X','X','X','X','X','X','X','X'), -- X
    ('U','X','0','X','0','0','0','0','X'), -- 0
    ('U','X','X','1','1','1','1','1','X'), -- 1
    ('U','X','0','1','Z','W','L','H','X'), -- Z
    ('U','X','0','1','W','W','W','W','X'), -- W
    ('U','X','0','1','L','W','L','W','X'), -- L
    ('U','X','0','1','H','W','W','H','X'), -- H
    ('U','X','X','X','X','X','X','X','X')  -- -
    );
FUNCTION resolved(s : std_ulogic_vector) RETURN std_ulogic IS
    VARIABLE result : std_ulogic := 'Z'; -- weakest state default
BEGIN
  IF (s'LENGTH = 1) THEN
    RETURN s(s'LOW);
  ELSE
    FOR i IN s'RANGE LOOP
      result := resolution_table(result, s(i));
    END LOOP;
  END IF;
  RETURN result;
END resolved;
```

- **All driving values are collected in a vector**
- **The result is calculated element by element according to the table**
- **Resolution function is called whenever signal assignments involving resolved types are carried out**

STD_LOGIC vs. STD_ULOGIC

	Benefit
STD_ULOGIC **STD_ULOGIC_VECTOR**	• **Error messages in case of multiple concurrent signal assignments**
	Benefit
STD_LOGIC **STD_LOGIC_VECTOR**	• **Common industry standard** • **Gate level netlists** • **Mathematical functions** • **Required for tristate busses**

STD_LOGIC(_VECTOR) is recommended for RT level designs

Use port mode ‘buffer’ to avoid multiple signal assignments

Professur Schaltkreis- und Systementwurf

Counter

TECHNISCHE UNIVERSITÄT CHEMNITZ

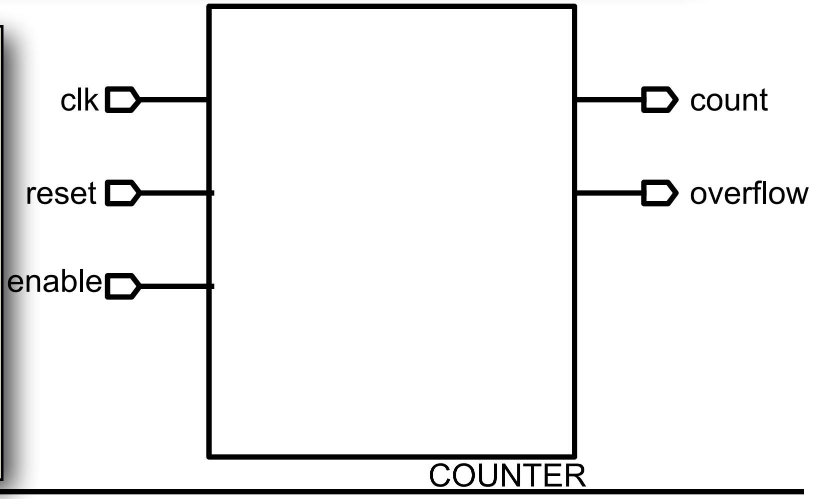
10 bit Counter with enable and asynchronous reset

A module named COUNTER shall be designed with a reset and enable input
In case the reset port is HIGH all the outputs shall be set to Zero.
In case the enable port is LOW the current value of count should be kept on the count output.
In case the enable port is HIGH the output count shall be increased by 1 with every rising clock edge.
In case the highest count value is reached the output shall be set to Zero and the overflow output shall be set to ‘1‘ for one clock cycle

all input ports are of data type: std_logic
the counter output shall be data type integer
the overflow output shall be data type std_logic

Inputs: clk, reset, enable

Outputs: count, overflow



10 bit Counter with enable and asynchronous reset

```
ENTITY COUNTER IS
PORT (
  clk      : in  std_logic;
  reset    : in  std_logic;
  enable   : in  std_logic;
  count    : out integer range 0 to 1023;
  overflow: out std_logic);
END ENTITY COUNTER;
```

```
ARCHITECTURE RTL OF COUNTER IS
signal count_i: integer range 0 to 1023;
BEGIN  -- ARCHITECTURE RTL
  P_COUNT:PROCESS (clk, reset)
   BEGIN
     if reset = '1' then
           count_i  <= 0;
           overflow <= '0';
     elsif clk'event and clk = '1' then
       if enable = '1' then
          if count_i < 1023 then
          overflow <= '0';
          count_i <= count_i + 1;
          else
          overflow <= '1';
          count_i <= 0;
          end if;
       end if;
     end if;
    END PROCESS;
    count <= count_i;
  END ARCHITECTURE RTL;
```

Attention!

Professur Schaltkreis- und Systementwurf

TECHNISCHE UNIVERSITÄT CHEMNITZ