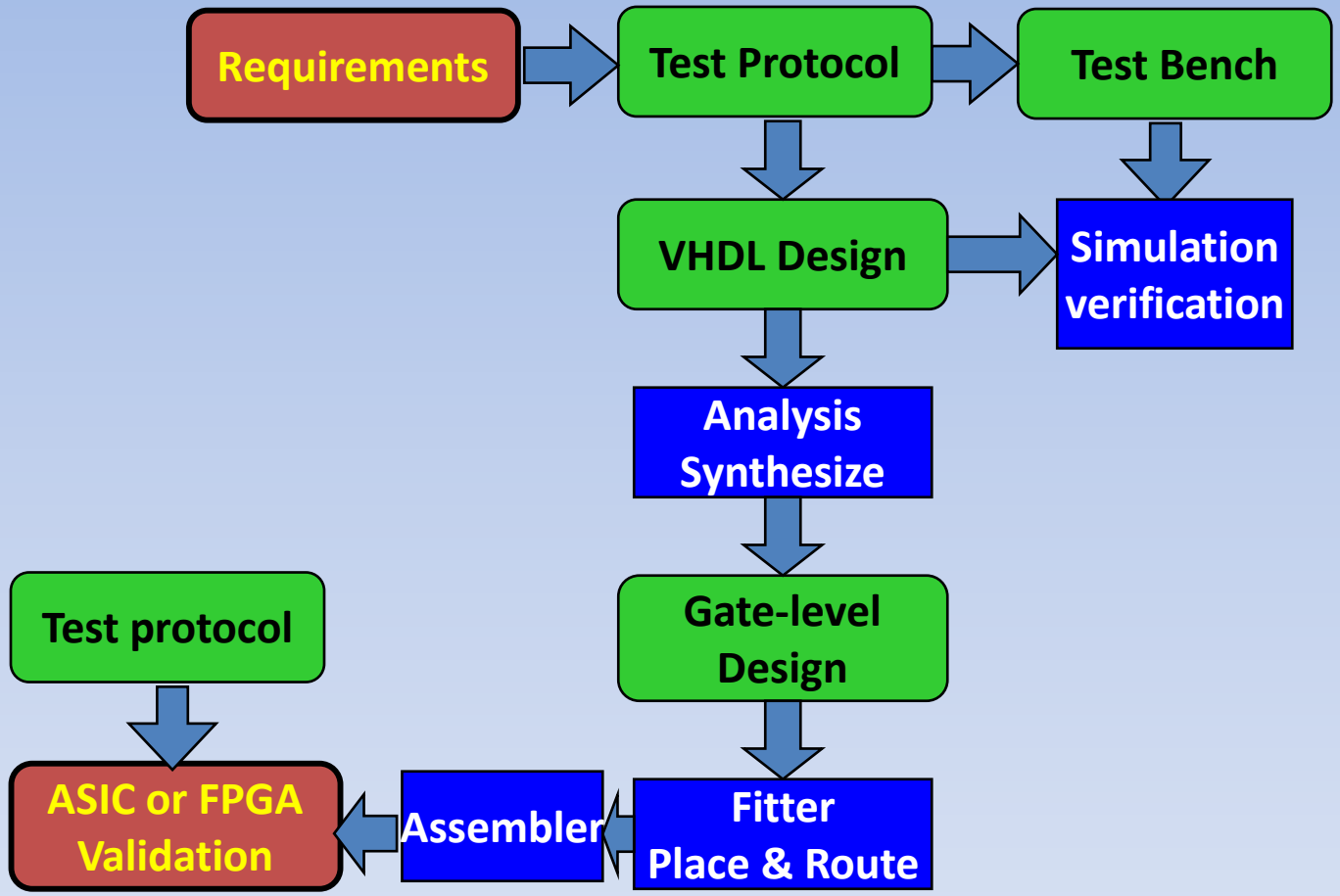
AGSTU
Arbete Genom STUdier
Utbildning

VHDL- programmering för inbyggda system
Välkommen

- Design Methodology
- Terminology
- Behavioral model/component
- Structural model/component
- Complexity Reduction
  - Language abstractions
  - Design hierarchy
- Configuration
- Sub Program
  - Function and Procedure

AGSTU
Arbete Genom STUdier
Utbildning

Basic Design Methodology



AGSTU
Arbete Genom STUdier
Utbildning

Terminology

- HDL – **H**ardware **D**escription **L**anguage is a software programming language that is used to model/design a piece of hardware

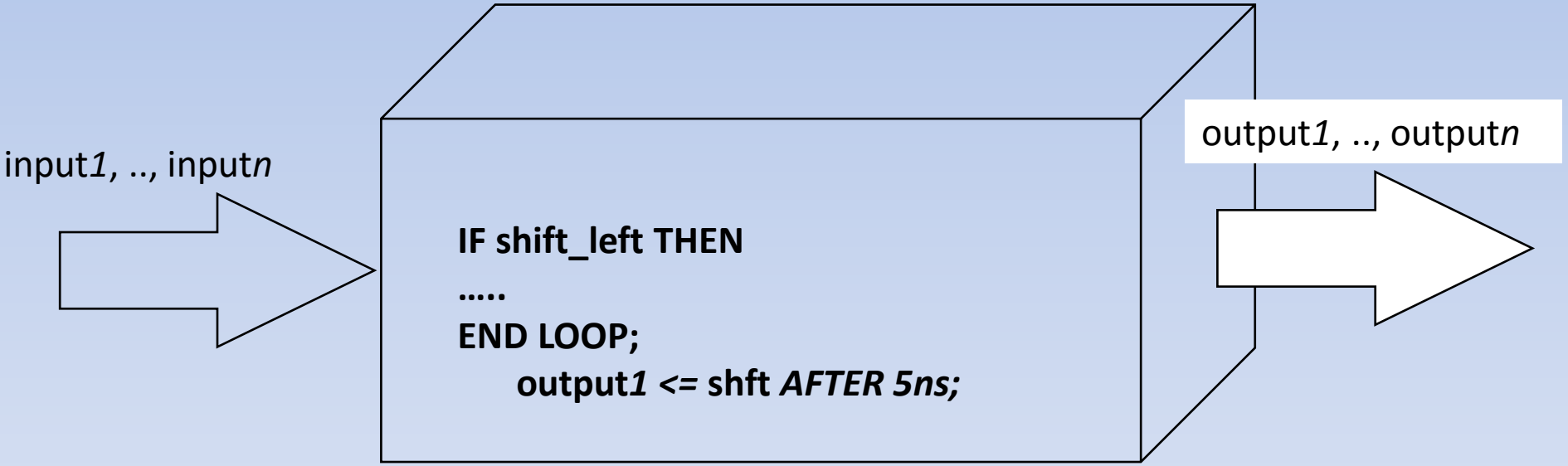
- Behavioral modeling - A component is described by its input/output response
  - It describes algorithmically (if a>23 then ..)

- Structural modeling - A component is described by interconnecting components/ primitives
  - It is basically schematic

AGSTU
Arbete Genom STUdier
Utbildning

Behavioral Modeling

Only the functionality of the circuit

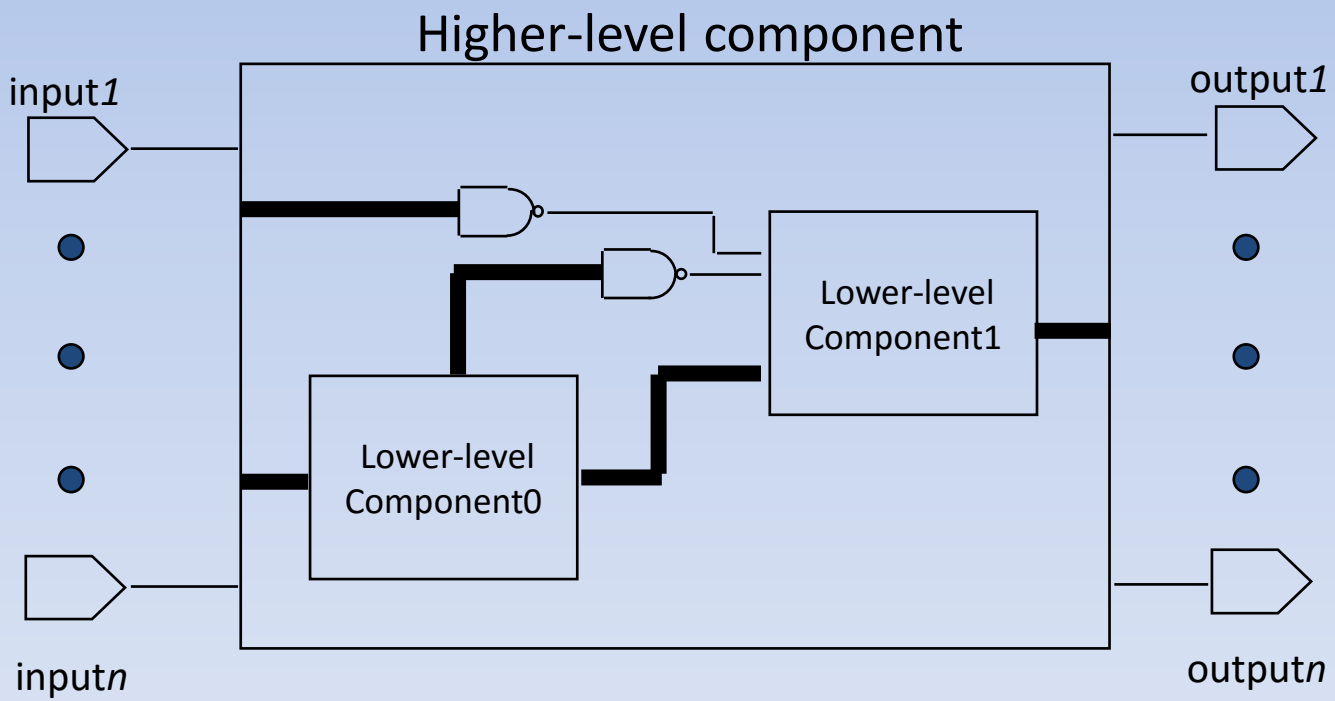


Left bit shifter

AGSTU
Arbete Genom STUdier
Utbildning

Structural Modeling

Functionality and structure of the circuit



Terminology

- **R**egister **T**ransfer **L**evel (RTL) - A type of behavioral modeling, for the purpose of synthesis
  - Restrictions on coding style

- Synthesis - Translating HDL to a circuit and then optimizing the represented circuit
  - Translates register-transfer-level (RTL) design into gate-level netlist

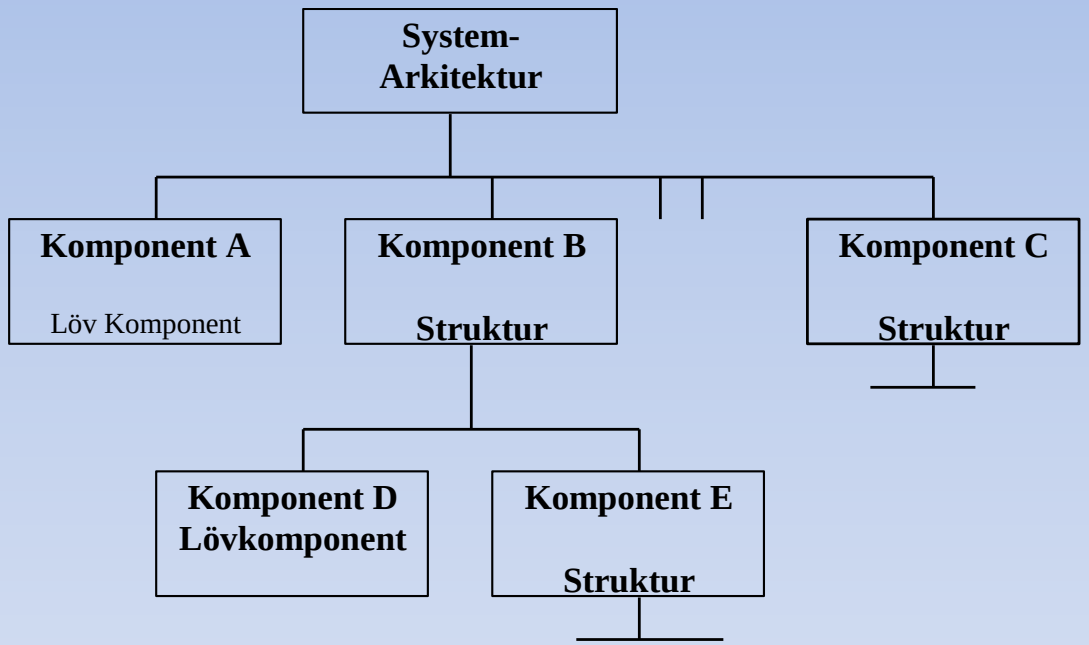
AGSTU
Arbete Genom STUdier
Utbildning

Reduce complexity

- **Language abstractions** use the language to describe complex matters without having to describe small details
  - Functions and procedures are parts of the language in order to handle complexity
- **Design hierarchy** uses components in order to conceal details - the black box principle
  - The term black box means that only inputs/outputs of a component are visible at a certain level

AGSTU
Arbete Genom STUdier
Utbildning

Konstruktions Hierarki



Högst upp: System-nivån,

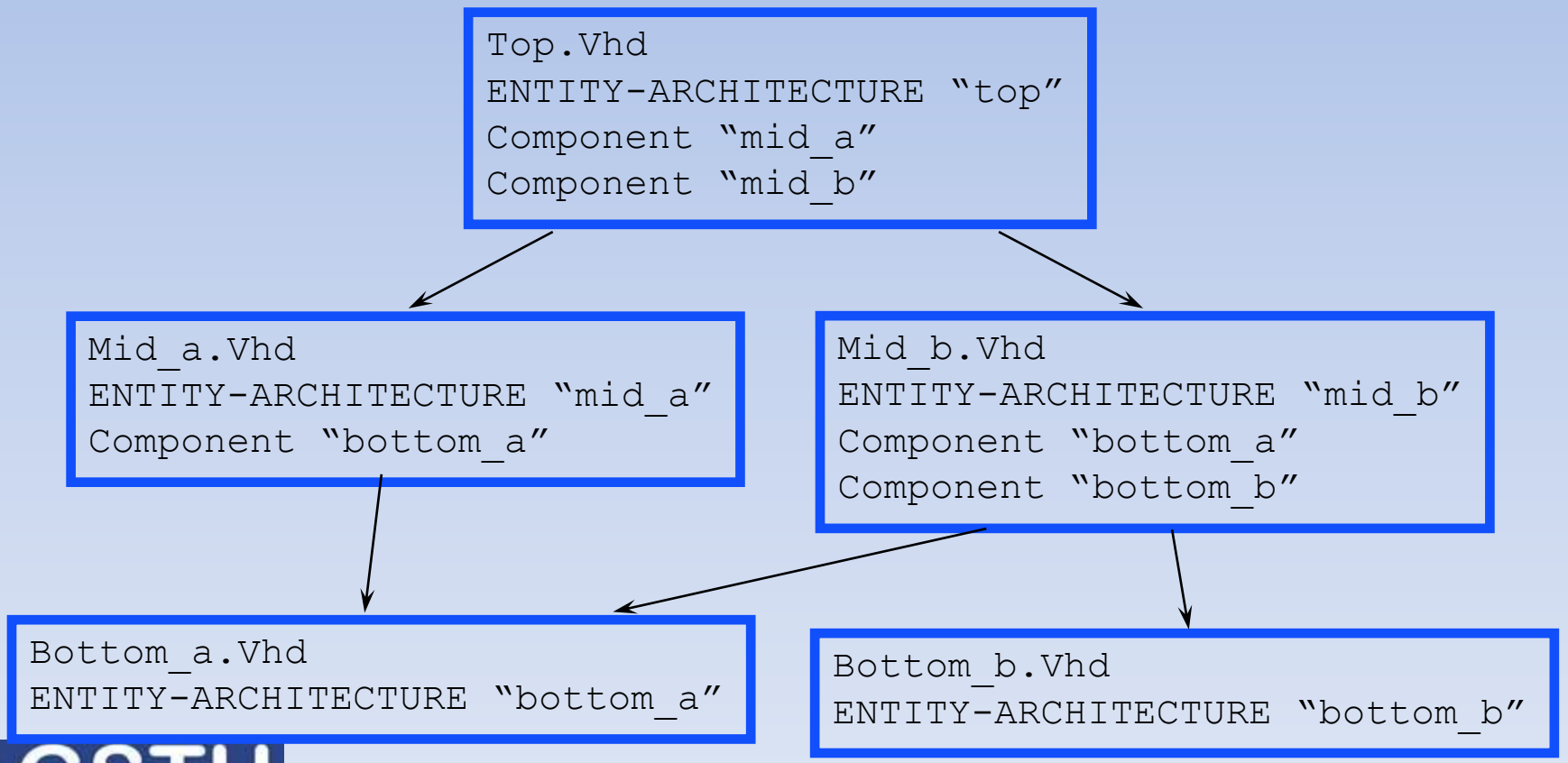
Top-komponenten: Tex RDY_Top

Nivå X: Sista nivån kallas ibland lövkomponent.

AGSTU
Arbete Genom STUdier
Utbildning

Design Hierarchically - Multiple Design Files

- VHDL hierarchical design requires component declarations and component instantiations



AGSTU
Arbete Genom STUdier
Utbildning

Component Declaration and Instantiation

- **Component declaration** - used to declare the *port types* and the *data types* of the ports for a lower-level design

```
COMPONENT <lower-level_design_name>
        PORT  (
                        <port_name>  :  <port_type> <data_type>;
                                ...
                        <Port_name>  :  <port_type> <data_type>
        );
END COMPONENT;
```

- **Component instantiation** - used to map the ports of a lower-level design to that of the current-level design

```
<Instance_name> : <lower-level_design_name>
        PORT MAP(<lower-level_port_name>  => <current_level_port_name>,                    ...,
<lower-level_port_name> => <current_level_port_name>);
```

Component Declaration and Instantiation (1)

```
LIBRARY IEEE;
USE  IEEE. STD_LOGIC_1164.ALL;

ENTITY tolleab IS
      PORT (
            tclk, tcross, tnickel, tdime, tquarter : IN STD_LOGIC;
            tgreen, tred : OUT STD_LOGIC
      );
END ENTITY tolleab;

ARCHITECTURE tolleab_arch OF tolleab IS
      COMPONENT tollv
            PORT(
                  clk, cross, nickel, dime, quarter : IN STD_LOGIC;
                  green, red : OUT STD_LOGIC;
            );
      END COMPONENT;
BEGIN
      U1 :  tollv PORT MAP (clk => tclk, cross => tcross,
            nickel => tnickel, dime => tdime, quarter => tquarter,
            green => tgreen, red => tred);

END ARCHITECTURE tolleab_arch;
```

Upper-level of hierarchy design must have a component declaration for a lower-level design before it can be instantiated

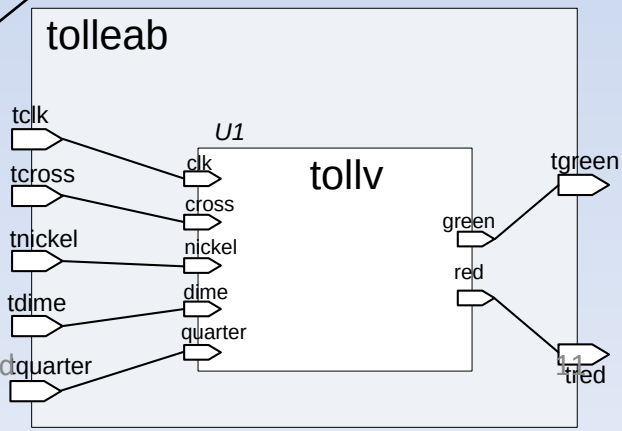
Component declaration

Lower-level port

Dime => tdime

Current-level port

Named Association



Instance label/name

Component instantiation

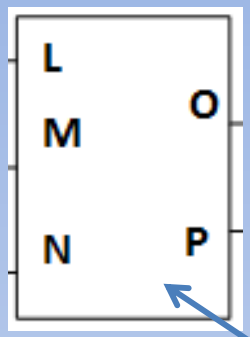
Component Declaration and Instantiation (2)

- Component instantiation using **positional association**
  - Order of ports in declaration maps to order of ports in instantiation
  - **Not recommended**

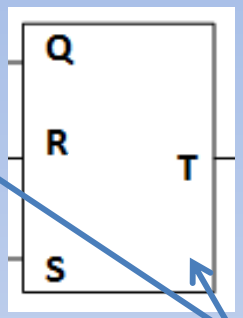
```
ARCHITECTURE tolleab_arch OF tolleab IS
    COMPONENT tollv
        PORT(
            clk : IN STD_LOGIC;
            cross, nickel, dime, quarter : IN  STD_LOGIC;
            green, red : OUT  STD_LOGIC;
        );
    END COMPONENT;
BEGIN
    U1 :  tollv  PORT MAP (tclk, tcross, tnickel, tdime,
        tquarter, tgreen, tred);
```

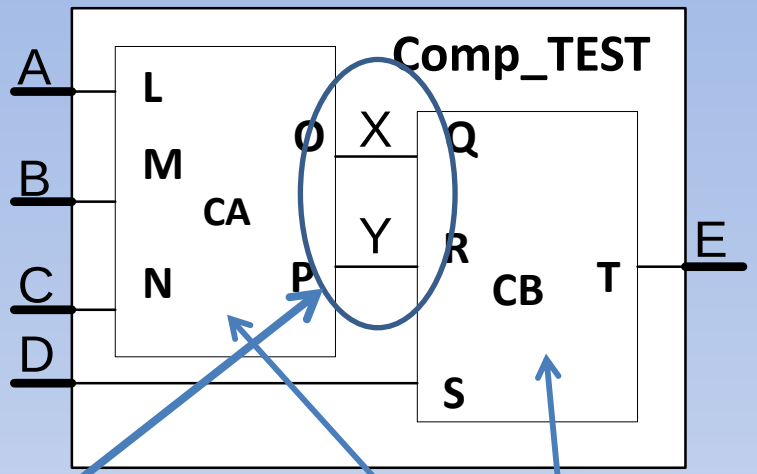
Positional association

Example



Component library






```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;


ENTITY Comp_TEST IS
PORT (A,B,C,D  :  IN STD_LOGIC;
       E        : OUT STD_LOGIC);
END Comp_TEST;


ARCHITECTURE BEHAVIOR OF
       Comp_TEST IS

SIGNAL X,Y : STD_LOGIC;
```

Declaration

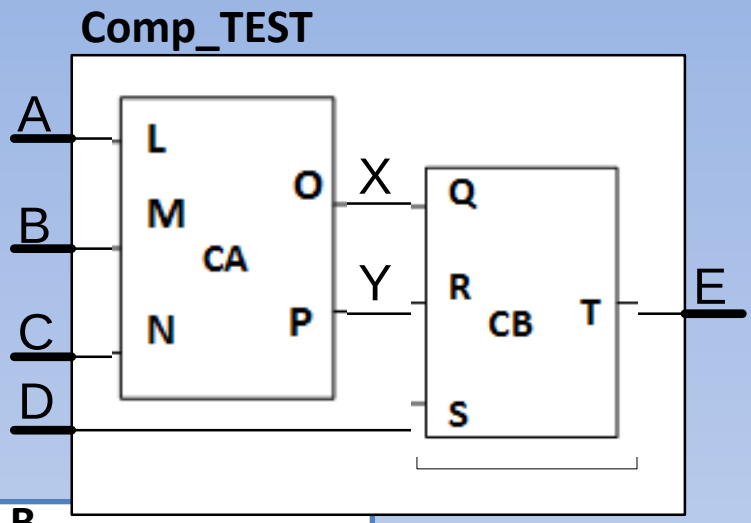
```
COMPONENT Comp_A
PORT (
L,M,N   : IN STD_LOGIC;
O,P     : OUT STD_LOGIC);
END COMPONENT;


COMPONENT Comp_B
PORT (
Q,R,S :  IN STD_LOGIC;
T  : OUT STD_LOGIC);
END COMPONENT;
```

Instantiation

```
BEGIN

CA : Comp_A
PORT MAP (A,B,C,X,Y);

CB : Comp_B
PORT MAP (
Q => X,
R => Y,
S => D,
T => E);

END BEHAVIOR;
```

AGSTU
Arbete Genom STUdier
Utbildning

Sammandrag



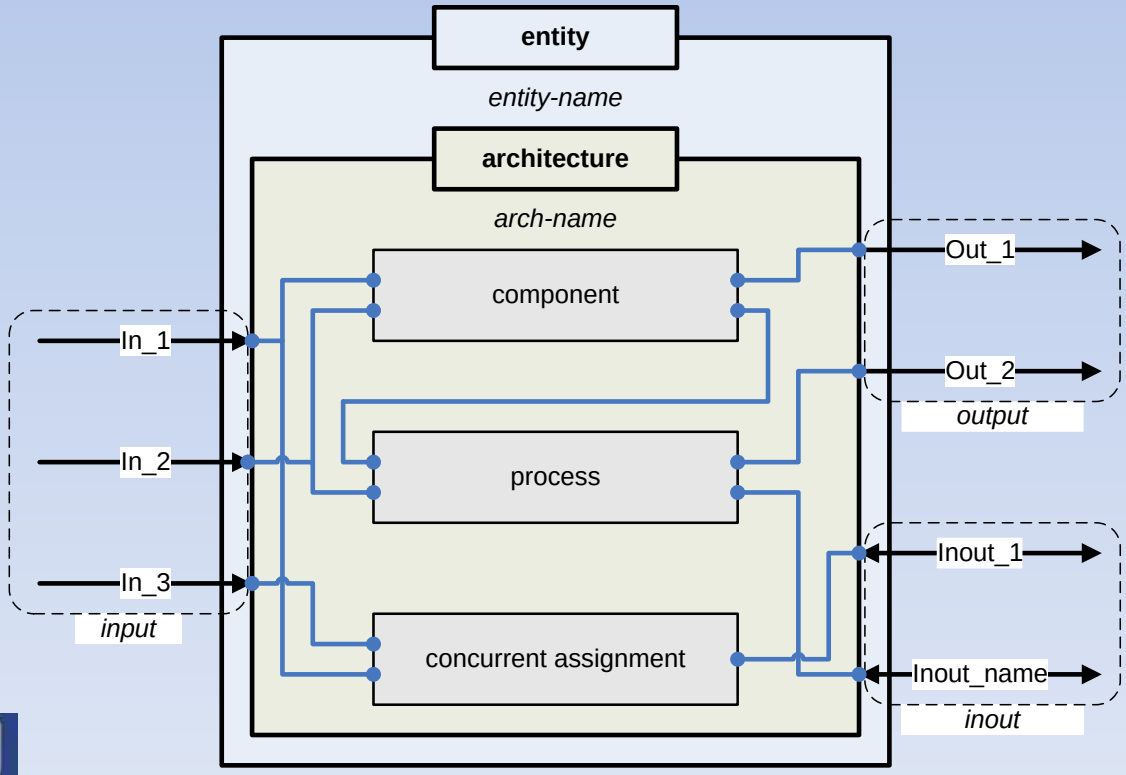

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;


ENTITY Comp_TEST IS
PORT (A,B,C,D  :  IN STD_LOGIC;
      E        : OUT STD_LOGIC);
END Comp_TEST;


ARCHITECTURE BEHAVIOR OF
      Comp_TEST IS

SIGNAL X,Y : STD_LOGIC;

COMPONENT Comp_A
PORT (
L,M,N   :  IN STD_LOGIC;
O,P     : OUT STD_LOGIC);
END COMPONENT;


-- next page
```

```
COMPONENT Comp_B
PORT (
Q,R,S :  IN STD_LOGIC;
T  : OUT STD_LOGIC);
END COMPONENT;
BEGIN

CA : Comp_A
PORT MAP (A,B,C,X,Y);

CB : Comp_B
PORT MAP (
Q => X,
R => Y,
S => D,
T => E);

END BEHAVIOR;
```

AGSTU
Arbete Genom STUdier
Utbildning

A general VHDL design

- An architecture can contain both behavioral and structural parts



Mixed Example

```
…
multiplier_sr : entity work.shift_reg(behavior)
    port map ( d => multiplier,  q => mult_bit,
                load => mult_load,  clk => clk );
product <= full_product;

control_section : process is
    -- variable declarations for control_section
    -- …
begin
    -- sequential statements to assign values to control signals
    -- …
    wait on clk, reset;
end process control_section;
end architecture mixed;
```

ARCHITECTURE

ARCHITECTURE <identifier> **OF** <entity_identifier> **IS**

--ARCHITECTURE declaration section

SIGNAL temp **:** INTEGER **:=** 1**;** -- signal declarations with optional default values

CONSTANT load **:** boolean **:=** true**;** --constant declarations

--Type declarations
--Component declarations
--Subprogram declarations
--Subprogram body
--Subtype declarations
--Attribute declarations
--Attribute specifications

BEGIN

PROCESS statements
Concurrent procedural calls
Concurrent signal assignment
Component instantiation statements
Generate statements

END ARCHITECTURE <architecture_identifier>**;**

Configuration

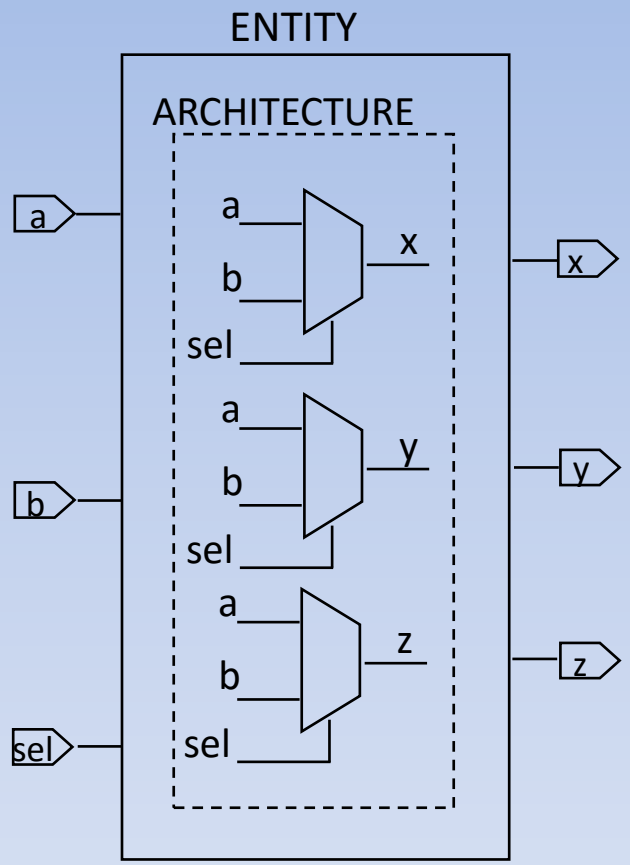
```
CONFIGURATION <identifier> OF <entity_name> IS
    FOR <architecture_name>
        FOR <instance_name> : <component_name> USE <entity>(<architecture>)
        END FOR;
        FOR <instance_name> : <component_name> USE <configuration_name>
        END FOR;
    END FOR;
END CONFIGURATION <identifier>; ( 1076-1993 version)
```

- Makes associations (bindings) within models
  - Associate an entity and an architecture
  - Associate an instance to another entity-architecture (component) hierarchically
- Uses
  - In simulation environments to execute different sets of vector stimulus
  - To provide flexible and fast path to design alternatives
    - e.g. Behavioral vs. synthesizable model; FPGA vs. ASIC model
  - Not required as most tools have ability to do bindings automatically

AGSTU Arbete Genom STUdier Utbildning

Putting It All Together

```
ENTITY cmpl_sig IS
      PORT (
            a, b, sel          : IN    BIT;
            x, y, z            : OUT BIT
      );
END ENTITY cmpl_sig;

ARCHITECTURE logic OF cmpl_sig IS
BEGIN
        -- simple signal assignment
      x <= (a AND NOT sel) OR (b AND sel);
        -- conditional signal assignment
      y <= a WHEN sel='0' ELSE
           b;
        -- selected signal assignment
      WITH sel SELECT
            z <= a WHEN '0',
                 b WHEN '1',
                '0' WHEN OTHERS;
END ARCHITECTURE logic;

CONFIGURATION cmpl_sig_conf OF cmpl_sig IS
      FOR logic
      END FOR;
END CONFIGURATION cmpl_sig_conf;
```



Test bänken för demo_modelsim

```
28   USE ieee.std_logic_1164.all;
29
30   ENTITY demo_modelsim_vhd_tst IS
31   END demo_modelsim_vhd_tst;
32   ARCHITECTURE demo_modelsim_arch OF demo_modelsim_vhd_tst IS
33   -- constants
34   -- signals
35   SIGNAL knapp_in : STD_LOGIC;
36   SIGNAL knapp_out : STD_LOGIC;
37   COMPONENT demo_modelsim
38      PORT (
39      knapp_in : IN STD_LOGIC;
40      knapp_out : OUT STD_LOGIC
41      );
42   END COMPONENT;
43   BEGIN
44      i1 : demo_modelsim
45      PORT MAP (
46   -- list connections between master ports and signals
47      knapp_in => knapp_in,
48      knapp_out => knapp_out
49      );
50   init : PROCESS
51   -- variable declarations
52   BEGIN
53           -- code that executes only once
54   WAIT;
55   END PROCESS init;
```

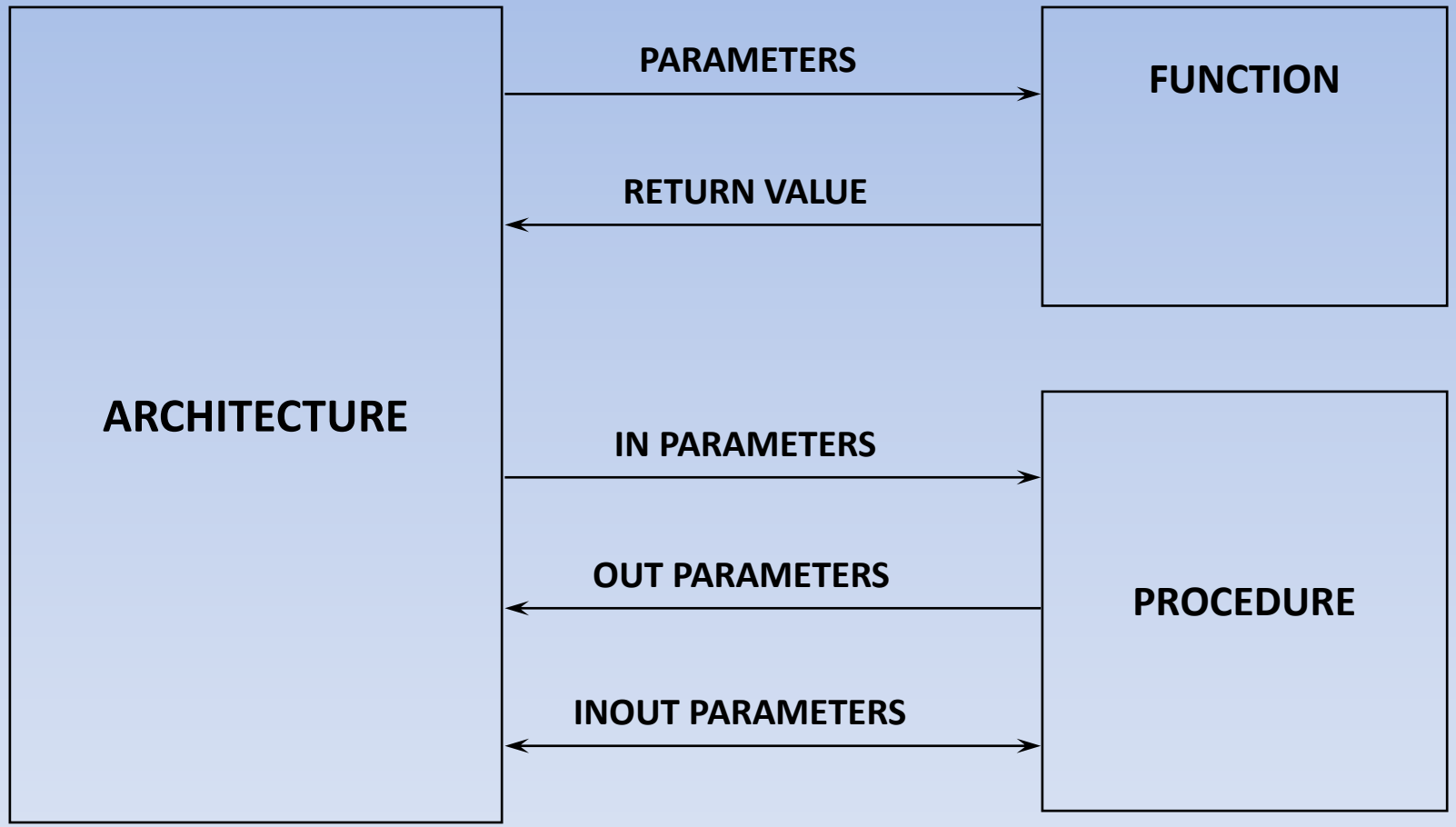
AGSTU Arbete Genom STUdier Utbildning

Subprograms

- VHDL has 2 subprograms
  - FUNCTION
    - Performs calculation and returns value
  - PROCEDURE
    - Performs sequence of defined sequential statements
- Uses
  - Replacing repetitive code
  - Enhancing readability
  - Break processes into executable sections
- Defined by means of subprogram declaration (optional) and subprogram body
  - Subprogram declarations required if subprogram is called before subprogram body is read
- May be declared in process, architecture or package
  - Determines visibility
  - When placed in package, subprogram declaration goes in package declaration and subprogram body goes in package body (see earlier package example)
- Synthesis places restrictions on use of subprograms

AGSTU
Arbete Genom STUdier
Utbildning

Subprogram Diagram



AGSTU
Arbete Genom STUdier
Utbildning

Function Definition & Call

Function Declaration

```
FUNCTION ones_count (SIGNAL a : STD_LOGIC_VECTOR) RETURN VARIABLE;
```

- Must return a single value based on zero or more inputs
- Must be called in an expression
- Can be passed classes **CONSTANT** (default), **SIGNAL** or **FILE**
- Class for internal objects must be **VARIABLE**

Function Body

```
FUNCTION ones_count
        (SIGNAL a : STD_LOGIC_VECTOR) IS
    VARIABLE r : INTEGER;
BEGIN
   r := 0;
    FOR i IN  a'RANGE LOOP
        IF a(i) /= '0' THEN
            r := r + 1 ;
        END IF;
    END LOOP;
    RETURN r; -- Required
END FUNCTION ones_count;
```

Invoking a Function

```
total_ones <= ones_count (input) WHEN test_ones = '1';
```

Note: 'RANGE is a VHDL attribute which returns the range of the object it is applied to (e.g. 7 DOWNTO 0)

AGSTU Arbete Genom STUdier Utbildning

Procedure Definition & Call

Procedure Declaration

```
PROCEDURE incr_comp (
        SIGNAL cnt_sig : INOUT STD_LOGIC_VECTOR;
        CONSTANT max : IN INTEGER;
        SIGNAL maxed_out : OUT BOOLEAN
);
```

- May have inputs, inouts and outputs
- May return zero or multiple outputs
- Must be called as a separate sequential statement
- Parameters may be any class
  - Inputs are **CONSTANT** by default
  - Outputs/inouts are **VARIABLE** by default

Invoking a Procedure

```
incr_comp (err_cnt, 12, err_cnt_maxed);
incr_comp (code_cnt, 144, code_cnt_maxed);
```

Procedure Declaration

```
PROCEDURE incr_comp (
        SIGNAL cnt_sig : INOUT STD_LOGIC_VECTOR;
        CONSTANT max : IN INTEGER;
        SIGNAL maxed_out : OUT BOOLEAN
        ) IS
        -- declare any local objects (i.e. constants,
        --                            variables,…)
BEGIN
        IF cnt_sig >= max THEN
                maxed_out <= TRUE;
        ELSE
                maxed_out <= FALSE;
                cnt_sig <= cnt_sig + 1;
        END IF;
END PROCEDURE incr_comp;
```

AGSTU Arbete Genom STUdier Utbildning

Functions vs. Procedures

Functions

- Always execute in zero time
  - Cannot pause their execution
  - Can not contain any delay, event, or timing control statements
- Must have at least one input argument
  - Inputs may not be affected by function
- Arguments may not be outputs and inouts
- Always return a single value

Procedures

- May execute in non-zero simulation time
  - May contain delay, event, or timing control statements
- May have zero or more input, output, or inout arguments
- Modify zero or more values
- Return values by means of parameter arguments

AGSTU
Arbete Genom STUdier
Utbildning


SLUT
AGSTU
Arbete Genom STUdier
Utbildning