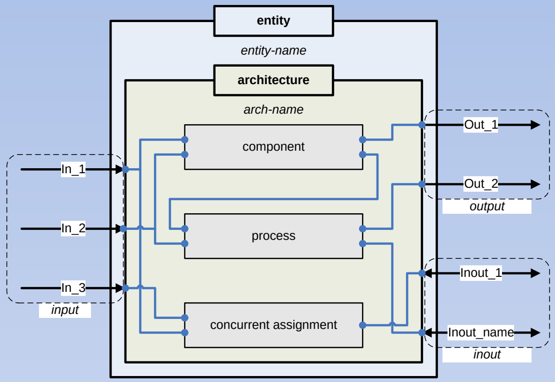
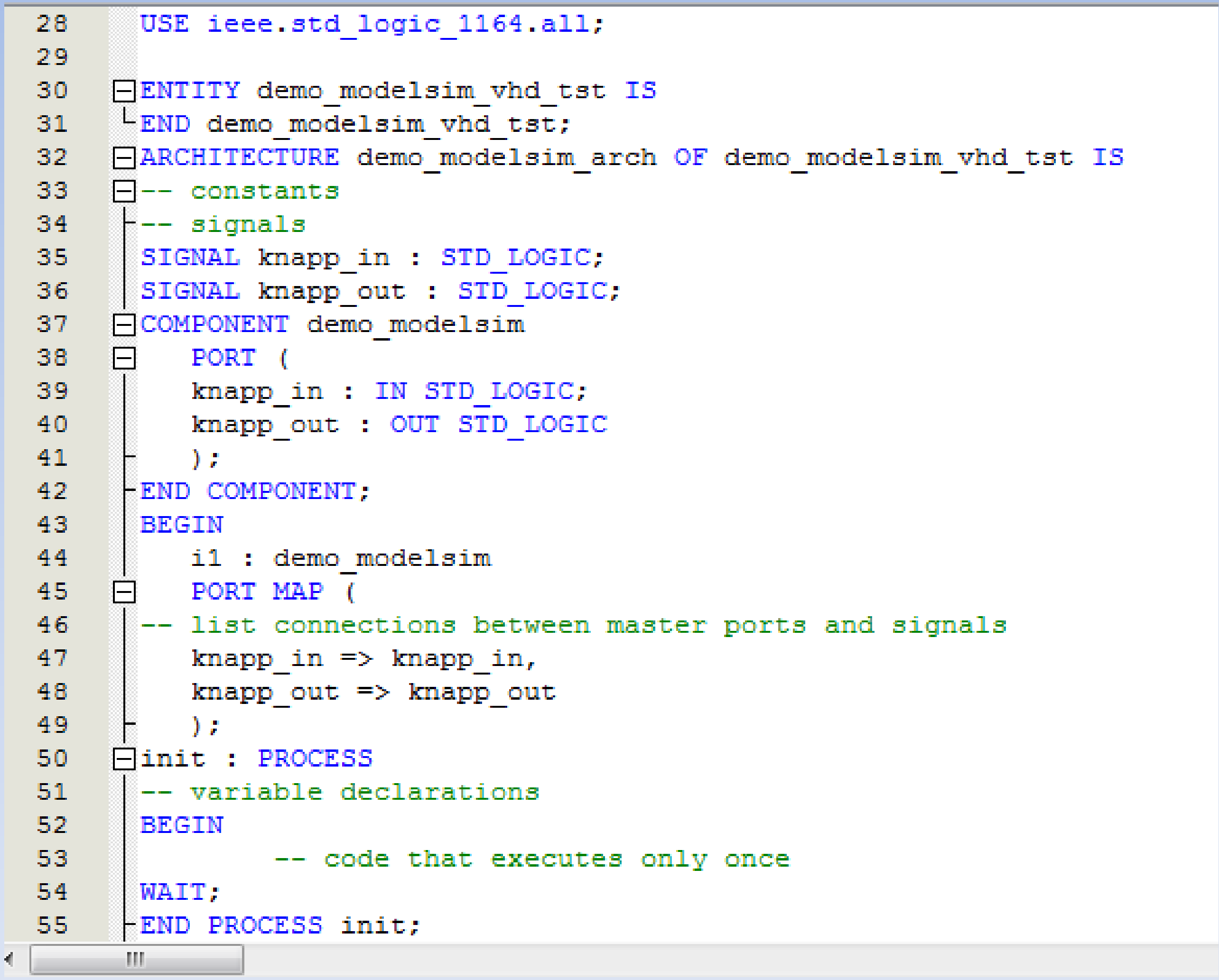
VHDL- programmering för inbyggda system
Välkommen

- Repetition Types
- Generics
- Package and library
- Operator Overloading

Copyright © AGSTU AB. All rights reserved

Repetion Structural design





```
USE ieee.std_logic_1164.all;

ENTITY demo_modelsim_vhd_tst IS
END demo_modelsim_vhd_tst;
ARCHITECTURE demo_modelsim_arch OF demo_modelsim_vhd_tst IS
-- constants
-- signals
SIGNAL knapp_in : STD_LOGIC;
SIGNAL knapp_out : STD_LOGIC;
COMPONENT demo_modelsim
   PORT (
   knapp_in : IN STD_LOGIC;
   knapp_out : OUT STD_LOGIC
   );
END COMPONENT;
BEGIN
   i1 : demo_modelsim
   PORT MAP (
-- list connections between master ports and signals
   knapp_in => knapp_in,
   knapp_out => knapp_out
   );
init : PROCESS
-- variable declarations
BEGIN
        -- code that executes only once
WAIT;
END PROCESS init;
```

AGSTU
Arbete Genom STUdier
Utbildning

Copyright © AGSTU AB. All rights reserved

Types

- VHDL has built-in data types to model hardware
- VHDL also allows creation types for declaring objects (i.e. constants, signals, variables)
- Subtype
- Enumerated Data Type
- Array

Copyright © AGSTU AB. All rights reserved

Subtype

- A constrained type
- Synthesizable if base type is synthesizable
- Use to make code more readable and flexible
  - Place in package to use throughout design

```
ARCHITECTURE logic OF subtype_test IS

        SUBTYPE word IS std_logic_vector (31 DOWNTO 0);
        SIGNAL mem_read, mem_write : word;

        SUBTYPE dec_count IS INTEGER RANGE 0 TO 9;
        SIGNAL ones, tens : dec_count;

BEGIN
```

Copyright © AGSTU AB. All rights reserved

Enumerated Data Type

- Allows user to create data type ***name*** and ***values***
  - Must create constant, signal or variable of that type to use
- Used in
  - Making code more readable
  - Finite state machines
- Enumerated type declaration

```
TYPE <your_data_type> IS
        (data type items or values separated by commas);
```

```
TYPE enum IS (idle, fill, heat_w, wash, drain);
SIGNAL dshwshr_st : enum;
        …
drain_led <= ‘1’ WHEN dshwsher_st = drain ELSE ‘0’;
```

Array

- Creates multi-dimensional data type for storing values
  - Must create constant, signal or variable of that type
- Used to create memories and store simulation vectors
- Array type Declaration

array depth

```
TYPE <array_type_name> IS ARRAY (<integer_range>) OF
      <data_type>;
```

what can be stored in each array address

Array Example

```
ARCHITECTURE logic OF my_memory IS

    -- Creates new array data type named mem which has 64
    --   address locations each 8 bits wide
    TYPE mem IS ARRAY (0 to 63) OF std_logic_vector (7 DOWNTO 0);

    -- Creates 2 - 64x8-bit array to use in design
    SIGNAL mem_64x8_a, mem_64x8_b : mem;

BEGIN
    …
    mem_64x8_a(12) <= x“AF”;
    mem_64x8_b(50) <= “11110000”;
    …
END ARCHITECTURE logic;
```

Copyright © AGSTU AB. All rights reserved

Generic Declaration

```
ENTITY  <entity_name>  IS
      Generic declarations
      Port Declarations
END ENTITY <entity_name> ; (1076-1993 version)
```

- Generic declarations
  - Used to pass information into a model/component

ENTITY : Generic Declaration

```
ENTITY  <entity_name>  IS
        GENERIC (
                CONSTANT tplh , tphl            : time := 5 ns;
                -- Note CONSTANT is assumed and is not required
                tphz, tplz                      : TIME := 3 ns;
                default_value                   : INTEGER := 1;
                cnt_dir                         : STRING := "up"
        );
        PORT declarations
END ENTITY <entity_name> ; ( 1076-1993 version)
```

- Generic values can be overwritten during compilation
  - i.e. Passing in parameter information
- Generic must resolve to a constant during compilation

Copyright © AGSTU AB. All rights reserved

Packages

- Packages are a convenient way of storing and using information throughout an entire model
- Packages consist of:
  - Package declaration (required)
    - Type declarations
    - Subprograms declarations
  - Package body (optional)
    - Subprogram definitions

Copyright © AGSTU AB. All rights reserved

Packages

```
PACKAGE <package_name> IS
    Constant declarations
    Type declarations
    Signal declarations
    Subprogram declarations
    Component declarations
    --There are other declarations
END PACKAGE <package_name> ; (1076-1993)
```

```
PACKAGE BODY <package_name> IS
    Constant declarations
    Type declarations
    Subprogram body
END PACKAGE BODY <package_name> ; (1076-1993)
```

AGSTU Arbete Genom STUdier Utbildning

Copyright © AGSTU AB. All rights reserved

Package Example

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

PACKAGE filt_cmp IS
        TYPE state_type IS (idle, tap1, tap2, tap3, tap4);
        COMPONENT acc
                PORT (
                        xh        : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                      clk, first   : IN STD_LOGIC;
                      yn     : OUT STD_LOGIC_VECTOR (11 DOWNTO 4)
                );
        END COMPONENT;
        FUNCTION compare (SIGNAL a , b : INTEGER) RETURN BOOLEAN;
END PACKAGE filt_cmp;
PACKAGE BODY filt_cmp IS
        FUNCTION compare (SIGNAL a , b : INTEGER) RETURN BOOLEAN IS
        VARIABLE temp : BOOLEAN;
        BEGIN
                IF a < b THEN
                        temp := true;
                ELSE
                        temp := false;
                END IF;
                RETURN temp;
        END compare;
END PACKAGE BODY filt_cmp;
```

Package declaration

Package body

AGSTU
Arbete Genom STUdier
Utbildning

Copyright © AGSTU AB. All rights reserved

Libraries

- A **LIBRARY** is a directory that contains a package or a collection of packages
- Two types of libraries
  - Working library
    - Current project directory
  - Resource libraries
    - **STANDARD** package
    - **IEEE** developed packages
    - Altera component packages
    - Any **LIBRARY** of design units that is referenced in a design

Copyright © AGSTU AB. All rights reserved

Model Referencing of LIBRARY and PACKAGE

- All packages must be compiled before being referenced
- Implicit libraries
  - **WORK**
  - **STD**
  - ⇨Note: items in these packages do not need to be referenced, they are implied
- Referencing a package requires 2 clauses
  - **LIBRARY** clause
    - Defines the library name that can be referenced
    - Is a symbolic name to path/directory
    - Defined in the compiler tool
  - **USE** clause
    - Specifies the package and object in the library that you have specified in the library clause

Copyright © AGSTU AB. All rights reserved

Example

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

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

- **LIBRARY** <name>, <name> ;
  - Name is symbolic and defined by compiler tool
  - ⇨ Note: Remember that WORK and STD do not need to be defined.
- **USE** lib_name.pack_name.object;
  - **ALL** is a reserved word for object name
- Placing the library/use clause first will allow all following design units to access it

AGSTU Arbete Genom STUdier Utbildning

Libraries

- **LIBRARY STD;**
  - Contains the following packages
    - **STANDARD**
      - Pre-defined data types
      - Operator functions to support pre-defined data types
    - **TEXTIO**
      - File operations
      - Not discussed
  - An implicit library (built_in)
    - Does not need to be explicitly referenced in VHDL design

Copyright © AGSTU AB. All rights reserved

Types Defined in STANDARD Package

- Type **BIT**
  - 2 logic value system ('0', '1')

    SIGNAL a_temp : BIT;
  - Bit_vector array of bits

    SIGNAL temp : **BIT_VECTOR** (3 **DOWNTO** 0);

    SIGNAL temp : **BIT_VECTOR** (0 **TO** 3);
- Type **BOOLEAN**
  - (False, true)
- Type **INTEGER**
  - Positive and negative values in decimal

    SIGNAL int_tmp   : **INTEGER**; -- 32-bit number

    SIGNAL int_tmp1 : **INTEGER RANGE** 0 **TO** 255; --8 bit number

Copyright © AGSTU AB. All rights reserved

Other Types Defined in Standard Package

- Type **NATURAL**
  - Integer with range 0 to 2^{32}
- Type **POSITIVE**
  - Integer with range 1 to 2^{32}
- Type **CHARACTER**
  - ASCII characters
- Type **STRING**
  - Array of characters
- Type **TIME**
  - Value includes units of time (e.g. ps, us, ns, ms, sec, min, hr)
- Type **REAL**
  - Double-precision floating point numbers

AGSTU Arbete Genom STUdier Utbildning

Copyright © AGSTU AB. All rights reserved

Libraries

- **LIBRARY IEEE**
  - Contains the following packages
    - **STD_LOGIC_1164** (**STD_LOGIC** types & related functions)
    - **NUMERIC_STD** (unsigned arithmetic functions using standard logic vectors defined as SIGNED and UNSIGNED data type)
    - **STD_LOGIC_ARITH*** (arithmetic functions using standard logic vectors as SIGNED or UNSIGNED)
    - **STD_LOGIC_SIGNED*** (signed arithmetic functions directly using standard logic vectors)
    - **STD_LOGIC_UNSIGNED*** (unsigned arithmetic functions directly using standard logic vectors)
    - **STD_LOGIC_TEXTIO*** (file operations using std_logic)

** Packages actually developed by Synopsys but accepted and supported as standard VHDL by most synthesis and simulation tools*

Copyright © AGSTU AB. All rights reserved

Types Defined in STD_LOGIC_1164 Package

- Type **STD_LOGIC**
  - 9 logic value system (‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘-’)
    - ‘1’: Logic high
    - ‘0’: Logic low
    - ‘X: Unknown
    - ‘Z’: (not ‘z’) Tri-state
    - ‘-’: Don’t Care
    - ‘U’: Undefined
    - ‘H’: Weak logic high
    - ‘L’: Weak logic low
    - ‘W’: Weak unknown
  - Resolved type: supports signals with multiple drivers
    - Driving multiple values onto same signal results in known value

AGSTU Arbete Genom STUdier Utbildning

Copyright © AGSTU AB. All rights reserved

User-Defined Libraries/Packages

- User-defined packages in the same directory as the design

LIBRARY WORK; *--optional*
USE WORK.*<Package name>***.ALL**;

- User-defined packages in a different directory
  – Library name must be mapped to directory path via tool settings

LIBRARY *<any_name>***;**

USE *<any_name>.<Package_name>***.ALL;**

Copyright © AGSTU AB. All rights reserved

VHDL packages

- The package standard is predefined in the compiler. Types defined include: bit bit_vector typical signals integer natural positive typical variables boolean string character typical variables real time delay_length typical variables Click on standard to see the functions defined Note: This package must be provided with compiler, do not use this one.
- The package textio provides user input/output. Click on textio to see how to call the functions
- The package std_logic_1164 provides enhanced signal types .Types defined include: std_logic std_logic_vector. Click on std_logic_1164 to see available functions
- The package std_logic_textio provides input/output. Click on std_logic_textio to see how to call the functions
- The package std_logic_arith provides numerical computation Click on std_logic_arith_syn to see the functions defined std_logic_arith_ex.vhd has arithmetic functions that operate on signal types std_logic_vector and std_ulogic_vector Click on std_logic_arith_ex to see the functions defined
- The package numeric_bit provides numerical computation. Click on numeric_bit to see the functions defined
- The package numeric_std provides numerical computation. Click on numeric_std to see the functions defined
- The package std_logic_signed provides signed numerical computation on type std_logic_vector. Click on std_logic_signed to see the functions defined
- The package std_logic_unsigned provides unsigned numerical computation. Click on std_logic_unsigned to see the functions defined
- The package math_real provides numerical computation. Click on math_real to see the functions defined This declaration and body are in mathpack
- The package math_complex provides numerical computation. Click on math_complex to see the functions defined This declaration and body are in mathpack
- Reference: http://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html

Copyright © AGSTU AB. All rights reserved

Arithmetic Function

```
ENTITY opr IS
     PORT (
          a   : IN  INTEGER RANGE 0 TO 16;
          b   : IN  INTEGER RANGE 0 TO 16;
          sum : OUT  INTEGER RANGE 0 TO 32
     );
END ENTITY opr;

ARCHITECTURE example OF opr IS
BEGIN
     sum <= a + b;
END ARCHITECTURE example;
```

*The VHDL compiler can understand this operation because an arithmetic operation is defined for the built-in data type **INTEGER***

⇨ Note: remember the library **STD** and the package **STANDARD** do not need to be referenced

AGSTU Arbete Genom STUdier Utbildning

Copyright © AGSTU AB. All rights reserved

Operator Overloading

- VHDL defines arithmetic & boolean functions only for data types defined in STANDARD package

- How do you use arithmetic & boolean functions with other data types?
  - ***Operator Overloading*** - defining Arithmetic & Boolean functions with other data types

- Operators are overloaded by defining a function whose name is the same as the operator itself
  - Because the operator and function name are the same, the function name must be enclosed within double quotes to distinguish it from the actual VHDL operator
  - The function is normally declared in a package so that it is globally visible for any design

Operator Overloading Function/Package

- Packages that define operator overloading functions can be found in the **LIBRARY IEEE**
  - **STD_LOGIC_ARITH** (arithmetic functions)
  - **STD_LOGIC_SIGNED** (signed arithmetic functions)
  - **STD_LOGIC_UNSIGNED** (unsigned arithmetic functions)
  - **NUMERIC_STD** (signed & unsigned arithmetic)
- For example, the package **STD_LOGIC_UNSIGNED** defines some of the

```
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;

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;
```

Copyright © AGSTU AB. All rights reserved

Operator Overloading

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

ENTITY adder16 IS
              PORT (
X, Y          : IN          SIGNED(15 DOWNTO 0) ;
S             : OUT         SIGNED(15 DOWNTO 0) ) ;
END adder16 ;
ARCHITECTURE Behavior OF adder16 IS
BEGIN
              S <= X + Y;
END Behavior ;
```

Include these statements at the beginning of a design file

This allows us to perform arithmetic on non-built-in data types

Copyright © AGSTU AB. All rights reserved

Operator Overloading

- function "+" (L, R: signed) return signed;
- function "+" (L, R: unsigned) return unsigned ;

AGSTU
Arbete Genom STUdier
Utbildning

Copyright © AGSTU AB. All rights reserved

Examples Overloading

- Deklaration
  - Signal A_unsigned, B_unsigned, C_unsigned : unsigned(7 downto 0) ;
  - Signal R_signed, S_signed, T_signed : signed(7 downto 0) ;
  - Signal J_std_logic_vector, K_std_logic_vector, L_std_logic_vector : std_logic_vector(7 downto 0) ;
  - signal Y_signed : signed(8 downto 0) ;
- Permitted
  - A_unsigned <= B_unsigned + C_unsigned ; -- Unsigned + Unsigned = Unsigned
  - A_unsigned <= B_unsigned + 1 ; -- Unsigned + Integer = Unsigned
  - A_unsigned <= 1 + C_unsigned; -- Integer + Unsigned = Unsigned
  - R_signed <= S_signed + T_signed ; -- Signed + Signed = Signed
  - R_signed <= S_signed + 1 ; -- Signed + Integer = Signed
  - R_signed <= 1 + T_signed; -- Integer + Signed = Signed
  - J_std_logic_vector <= K_std_logic_vector + L_std_logic_vector ; -- if using std_logic_unsigned
- Illegal
  - R_signed <= A_unsigned - B_unsigned ;
    - Solution type conversions

Copyright © AGSTU AB. All rights reserved

SLUT

AGSTU
Arbete Genom STUdier
Utbildning