Universidad
Carlos III de Madrid

Design organization
Generic design

INTEGRATED CIRCUITS AND MICROELECTRONICS

Index

- VHDL design units
  - Entity and architecture
  - Package and package_body
  - Configurations
- Libraries
  - Standard libraries: work, std, ieee
  - Library use: context clauses
- Packages
- Hierarchical design and generic parameters
- Iterative operations
- Subroutines: functions and procedures
- Generic design

VHDL design units

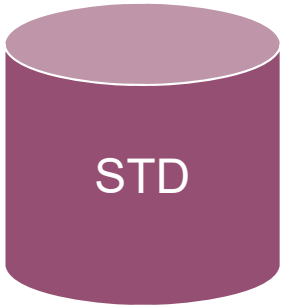
- Entity
  - Ports: in, out, inout
  - Generic parameters
- Architecture
  - Concurrent statements
  - Processes
  - Component instantiations
- Package
  - Declarations: Types, constants, subroutines
- Package body
  - Subroutine implementations
- Configurations
  - Entity-architecture binding

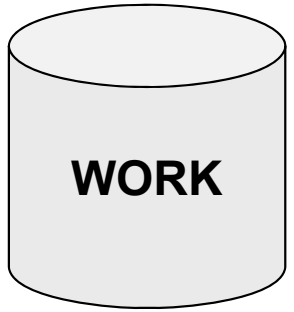
Libraries

- Definition:
  - Database where VHDL design units are stored
  - Usage is standard
  - Implementation is tool-dependent
- Usage through context clauses:
  - LIBRARY: select library
  - USE: select elements in a library
- Context clauses are placed before design unit declarations (entity, architecture, …)
- Predefined libraries:
  - STD, IEEE: standard types and operators
  - WORK: default user library

Standard libraries

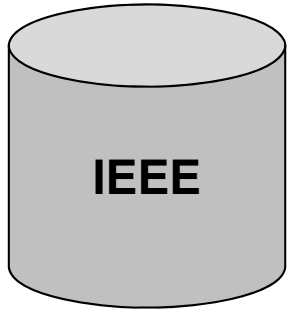
```
-- Predefined
library STD;
  use STD.standard.all;
library WORK;
```






```
library STD;
  use STD.textio.all;  -- Text and files

library IEEE;
  use IEEE.std_logic_1164.all;     -- Standard types
  use IEEE.numeric_std.all;        -- Arithmetic
  use IEEE.std_logic_textio.all;   -- Text and files
  use IEEE.numeric_bit.all;        -- Arithmetic
  use IEEE.std_logic_arith.all;    -- Arithmetic
  use IEEE.std_logic_signed.all;   -- Arithmetic
  use IEEE.std_logic_unsigned.all;-- Arithmetic
  use IEEE.math_real.all;          -- Real numbers
  use IEEE.math_complex.all;       -- Complex numbers
```



std.standard

```
    TYPE BOOLEAN IS (FALSE, TRUE);
    TYPE BIT IS ('0', '1');
    TYPE CHARACTER IS
(NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT,
FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
CAN, EM, SUB, ESC, FSP, GSP, RSP, USP, ' ', '!', '"',
'#', '$', '%', '&', ''', '(', ')', '*', '+', ',', '-',
'.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8',
'9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C',
'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z', '[', '\', ']', '^', '_', '`', 'a', 'b', 'c', 'd',
'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'{', '|', '}', '~', DEL);
    TYPE SEVERITY_LEVEL IS
      (NOTE, WARNING, ERROR, FAILURE);
    TYPE INTEGER IS
      RANGE -2147483648 TO 2147483647;
    TYPE REAL IS
      RANGE -1.7014110e+038 TO 1.7014110e+038;
```

```
TYPE TIME IS
       RANGE -9223372036854775808 TO 9223372036854775807
        UNITS fs;
            ps = 1000 fs;
            ns = 1000 ps;
            us = 1000 ns;
            ms = 1000 us;
            sec = 1000 ms;
            min = 60 sec;
            hr = 60 min;
        END UNITS;
    FUNCTION NOW RETURN TIME;
    SUBTYPE NATURAL IS
      INTEGER RANGE 0 TO INTEGER'HIGH;
    SUBTYPE POSITIVE IS
       INTEGER RANGE 1 TO INTEGER'HIGH;
    TYPE STRING IS
      ARRAY (POSITIVE RANGE <>) OF CHARACTER;
    TYPE BIT_VECTOR IS
      ARRAY (NATURAL RANGE <>) OF BIT;
END STANDARD;
```

ieee. std_logic_1164

```
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
    attribute ENUM_ENCODING of std_ulogic : type is "U D 0 1 Z D 0 1 D";
    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;
…
END std_logic_1164;
```

ieee. std_logic_1164

```
PACKAGE BODY std_logic_1164 IS
    -----------------------------------------------------------------
    -- local types
    -----------------------------------------------------------------
    --synopsys synthesis_off
    TYPE stdlogic_1d IS ARRAY (std_ulogic) OF std_ulogic;
    TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;
    -----------------------------------------------------------------
    -- resolution function
    -----------------------------------------------------------------
    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 |
…
END std_logic_1164;
```

Entities and architectures

- VHDL file

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

Context clauses

```
ENTITY entity_name IS
  [ GENERIC (...); ]
  [ PORT (...); ]
END entity_name ;
```

Entity

```
ARCHITECTURE architecture_name OF entity_name IS
  [ Declarations: types, components, signals…; ]
BEGIN
  [ Concurrent statements; ]
  [ Processes ]
  [ Instances ]
END [architecture_name ] ;
```

Architecture

Entity: ports and generics

- Ports are the circuit interface
- Modes
  - IN: Input port. It can only be read.
  - OUT: Output port. It can only be written.
  - INOUT: Input output port. It can be read and written.
- Generics are constants used to parameterize circuits. Typical use is size and range definition

Generics: N-bit counter

```
ENTITY n_count IS
  GENERIC (
    n: INTEGER := 8 );  -- 8-bit default
  PORT (
    clk, reset: IN STD_LOGIC;
    enable:     IN STD_LOGIC;
    count:      OUT UNSIGNED(n-1 downto 0));
END n_count;

ARCHITECTURE func OF n_count IS
  SIGNAL q: integer range 0 to 2**n-1;
BEGIN
  count <= to_unsigned(q,n)
  PROCESS (clk,reset)
  BEGIN
    if reset='1' then
      q <= 0;
    elsif rising_edge(clk) then
      if enable='1' then
        if q= 2**n-1 then
          q <= 0;
        else
          q <= q+1;
        end if;
      end if;
    end if;
  end PROCESS:
END func;
```

- Generic is declared in the entity and used in entity and architecture
- Generic actual value is given in instantiation
- If an actual value is not specified, default value in entity is taken

Hierarchy: components and instances

```
ENTITY counters IS
PORT (
  clk:      IN STD_LOGIC;
  reset:    IN STD_LOGIC;
  ena:      IN STD_LOGIC;
  count6:   OUT UNSIGNED( 5 downto 0);
  count12: OUT UNSIGNED(11 downto 0) );
END counters ;

ARCHITECTURE func OF counters IS

  COMPONENT n_count IS
   GENERIC (
    n: INTEGER := 8 );
   PORT (
    clk, reset: IN STD_LOGIC;
    enable:     IN STD_LOGIC;
    count:      OUT UNSIGNED(n-1 downto 0));
  END COMPONENT;

BEGIN
. . .
```

Component declaration

Instance name

Formal

Actual

```
. . . . .
BEGIN
   C6: n_count
      GENERIC MAP ( n => 6 );
      PORT MAP (
        clk    => clk,
        reset  => reset,
        enable => ena,
        count  => count12 );

   C2: n_count
      GENERIC MAP ( n => 12 );
      PORT MAP (
        clk => clk,
        reset => reset,
        enable => ena,
        count  => count12 );
END func;
```

- **Component declaration**: must match the entity

- Instantiation
  - **Instance name:** unique to every instance, even for the same component
  - **Formal**: generic/port name in the component
  - **Actual**: assigned in the architecture. Can be specified as “open” in output ports.

Packages

- Groups of elements to use on several designs
- Packages may contain:
  - Declarations: types, subtypes, constants, signals, files, aliases, subroutines, attributes and components
- Packages may have a package body, containing:
  - Declarations: types, subtypes, constants, signals, files, aliases and subroutines
  - Subroutine bodies

Example: package declaration and usage

```
PACKAGE my_package IS
  COMPONENT n_count IS
    GENERIC (n: INTEGER := 8 );
    PORT (clk, reset: IN STD_LOGIC;
          enable:     IN STD_LOGIC;
          count:      OUT UNSIGNED(n-1 downto 0));
  END COMPONENT;
  TYPE byte IS std_logic_vector(7 downto 0);
  TYPE memory IS ARRAY (0 TO 511) OF byte;
  CONSTANT semicycle : TIME := 5 NS;
  PROCEDURE parity (a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC);
END my_package ;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.ALL;

ARCHITECTURE using_my_package OF ... IS
  SIGNAL a,b: byte;
BEGIN
  . . . . .
END using_my_package ;
```

Example: package_body

```
PACKAGE BODY my_package IS

   PROCEDURE parity (a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC) IS
       VARIABLE p: STD_LOGIC;
   BEGIN
       p := '0';
       FOR i IN a'RANGE  LOOP
           p:= p XOR a(i);
       END LOOP;
       s := p;
   END par_proc;

END my_package;
```

Sequential iterative statements: loops

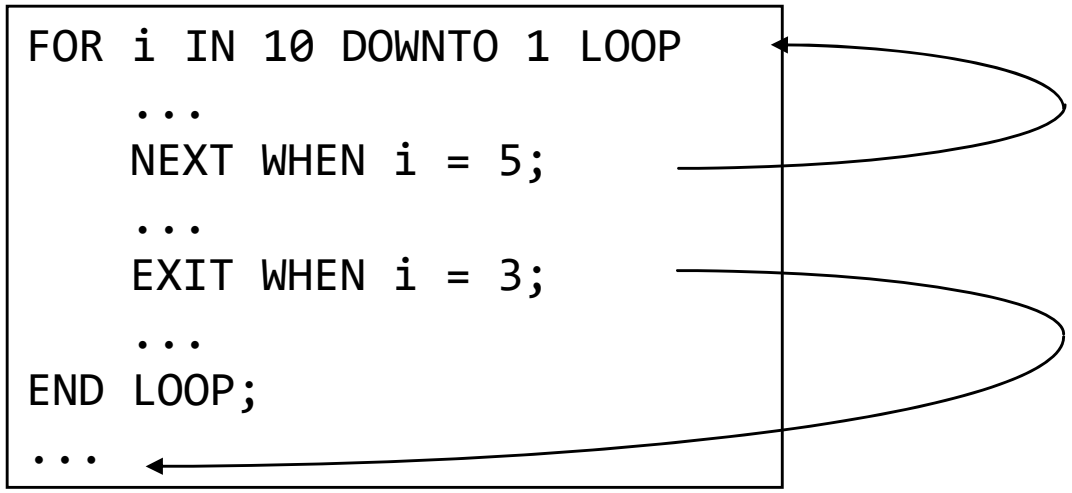
- Loops

```
-- Infinite loop
LOOP
    ...
END LOOP;
```

```
-- WHILE loop
WHILE a < b LOOP
    ...
END LOOP;
```

```
-- FOR loop
FOR i IN 10 DOWNTO 1 LOOP
    ...
END LOOP;
```

- NEXT ... [WHEN]: jumps to next loop iteration
- EXIT ... [WHEN]: exit loop, skip remaining iterations



Loop synthesis

- **Every loop iteration generates a circuit**
- For a loop to be synthesizable, the number of iterations must be **static** and determined at compilation time. The number of inferred circuits must be known.
- NEXT and EXIT are not synthesizable
- WHILE is admitted in some synthesizers, with a static number of iterations
- FOR loops must have constant range

Example

```
SIGNAL a: STD_LOGIC_VECTOR (0 TO 7);
SIGNAL z: STD_LOGIC;
...
z <= a(0) AND
     a(1) AND
     a(2) AND
     a(3) AND
     a(4) AND
     a(5) AND
     a(6) AND
     a(7);
```

```
SIGNAL a: STD_LOGIC_VECTOR (0 TO 7);
SIGNAL z: STD_LOGIC;
...
PROCESS (a)
     VARIABLE aux: STD_LOGIC;
BEGIN
     aux := a(0);
     FOR i IN 1 to 7 LOOP
          aux := aux AND a(i);
     END LOOP;
     z <= aux;
END PROCESS;
```

Concurrent iterative statements : GENERATE

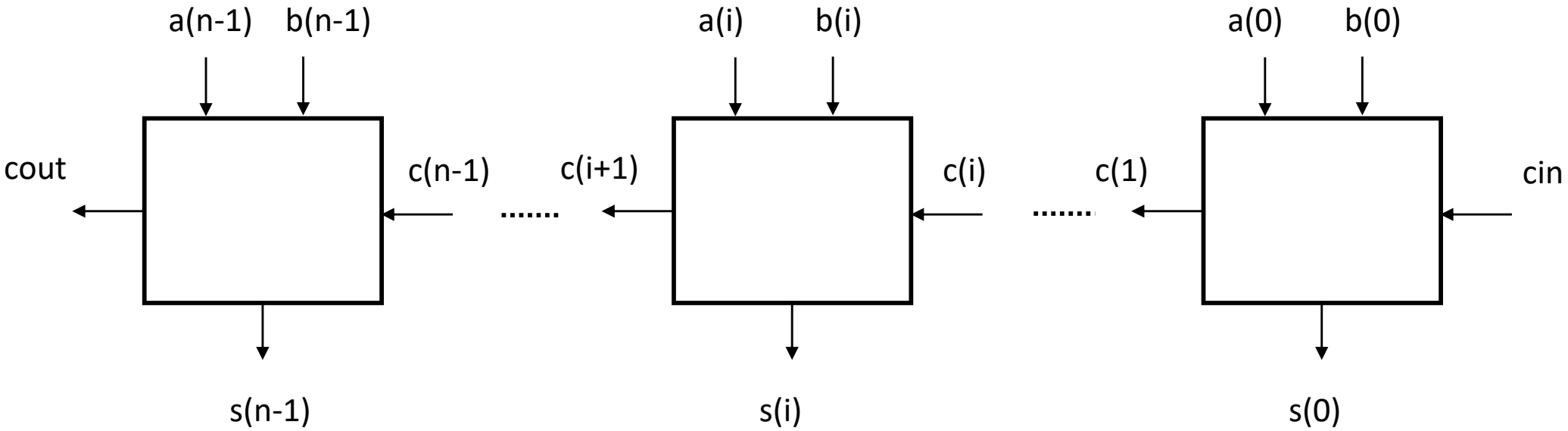
- Allows easy circuit replication based on parameters
- Two forms:
  - Generation of several circuits: FOR … GENERATE

```
Label: FOR name IN discrete_range GENERATE
  ... – concurrent statements
END GENERATE [ Label ];
```

  - Conditional circuit generation: IF … GENERATE

```
Label: IF condición GENERATE
  ... -- concurrent statements
END GENERATE [ Label];
```

FOR … GENERATE Example



FOR … GENERATE Example

```
ENTITY n_bit_adder IS
      GENERIC (n: INTEGER);
      PORT (     a:    IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
                 b:    IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
                 cin:  IN STD_LOGIC;
                 s:    IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
                 cout:       OUT STD_LOGIC );
END n_bit_adder ;

ARCHITECTURE generated OF n_bit_adder IS
      COMPONENT full_adder
      PORT ( x: IN STD_LOGIC; y: IN STD_LOGIC; cin: IN STD_LOGIC;
            s: OUT STD_LOGIC; cout: OUT STD_LOGIC);
      END COMPONENT;
      SIGNAL c : STD_LOGIC_VECTOR (n DOWNTO 0);
BEGIN
            c(0) <= cin;
fors:       FOR i IN 0 to n-1 GENERATE
sumx:             full_adder  PORT_MAP (a(i), b(i), c(i), s(i), c(i+1));
            END GENERATE;
            cout <= c(n);
END generated ;
```

IF … GENERATE Example

- Similar to conditional compilation
- Example:

```
ENTITY example IS
      GENERIC (reset_active_level: STD_LOGIC := ‘0’);
      PORT ( reset: STD_LOGIC;
           …);
END example ;

ARCHITECTURE a OF example IS
      SIGNAL rst: std_logic;
BEGIN

gen_low: IF reset_active_level = '0' GENERATE
      rst <= NOT reset; -- Change level
END GENERATE;

gen_high: IF reset_active_level = '1' GENERATE
      rst <= reset;
END GENERATE;
…
END a;
```

Subroutines: functions and procedures

- As any other programming language, VHDL supports subroutines to improve structure and modularity
- Two different types:
  - Functions: return a value and cannot modify input parameters
  - Procedures: used like statements; they can modify its parameters
- Recursive subroutines are not synthesizable
- Can be declared in architectures, but are typically placed in packages

Function example

```
FUNCTION parity (a: IN STD_LOGIC_VECTOR) RETURN STD_LOGIC IS
        VARIABLE p: STD_LOGIC;
BEGIN
        p := '0';
        FOR i IN a'RANGE  loop
                p:= p XOR a(i);
        END LOOP;
        RETURN p;
END parity;
```

```
-- Function call
        r <= parity (“00101101”);
```

Procedure example

```
PROCEDURE parity(a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC) IS
        VARIABLE p: STD_LOGIC;
BEGIN
        p := '0';
        FOR i IN a'RANGE LOOP
                p:= p XOR a(i);
        END LOOP;
        s := p;
END parity;
```

```
-- Procedure call
parity (“00101101”, r);
```

Subroutine parameter association

- Parameter asssociation as in component instantiation:
  - By position
  - Explicit
- Functions can only have parameters of mode IN
- Procedures can have parameters of mode IN, OUT or INOUT
- IN parameters are passed by value
- OUT parameters are passed by reference
  - Can be VARIABLE (default) or SIGNAL
  - The actual parameter must match the declaration of the formal parameter

Generic Design

- Designing with generic sizes: describing circuits without using explicit constant numbers.
- Examples (N,P and Q are generics):
  - Ports with generic size

```
X: in  unsigned(P-1 downto 0);
Y: out unsigned(Q-1 downto 0) );
```

  - Generic size data

```
type t_X is array (0 to N-1) of unsigned(P-1 downto 0);
signal sX : t_X;
```

  - Generic number of operations

```
for i in sX'range loop
  tmp := tmp+ sX(i)*A(i);  -- New result
end loop;
```

  - Generic number of assignments

```
sX(0) <= X;      -- Store incoming X
for i in sX'low+1 to sX'high loop
  sX(i) <= sX(i-1); -- Shift X samples
end loop;
```

Generic design: FIR filter, order N

Y(n)=a(0)*x(0)+a(1)*x(1)+…+a(N-1)*x(N-1)

N = order — a(i)=coefficients

P = X input data width — x(i)=input samples

Q = Y output data width — y(n)=output

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

package constants is
  type t_coefs is array (0 to 9) of integer;
  constant A: t_coefs :=
    (123,45,67,-112,561,-122,09,12,2,334);
end constants;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.constants.all;

entity FIR is
  generic (N,P,Q: integer);
  port ( clk:    in std_logic;
         enable: in std_logic;
         X: in  unsigned(P-1 downto 0);
         Y: out unsigned(Q-1 downto 0) );
end FIR;
```

```
architecture Func of FIR is
  type t_X is array (0 to N-1) of unsigned(P-1 downto 0);
  signal sX : t_X;
  signal result: unsigned (P+Q-1 downto 0);
begin

  process(sX)
    variable tmp: unsigned (P+Q-1 downto 0);
  begin
    for i in sX'range loop
      tmp := tmp+ sX(i)*A(i);  -- New result
    end loop;
    result <= tmp;
  end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if enable='1' then
        sX(0) <= X;      -- Store incoming X
        for i in sX'low+1 to sX'high loop
          sX(i) <= sX(i-1); -- Shift X samples
        end loop;
        Y <= result(P+Q-1 downto P); -- Store Q bits
      end if;
    end if;
  end process;

end Func;
```