CSC34300
Lecture 01: VHDL Tutorial

Instructor: Zheng Peng

Computer Science Department

City College of New York

VHDL

- Initially sponsored by US Department of Defense (DoD) and IEEE
- A very capable hardware description language
- The behavior of a digital system can be described (specified) by writing a VHDL program.
- The VHDL description can be simulate to verify the systems behavior and it can then be synthesized and programmed into an FPGA

VHDL code basic structure

- Entity

  (I/O and generic declaration)

- Architecture

  structural (structure description)

  rtl (register transfer level description)

  behavioral (high-level description)

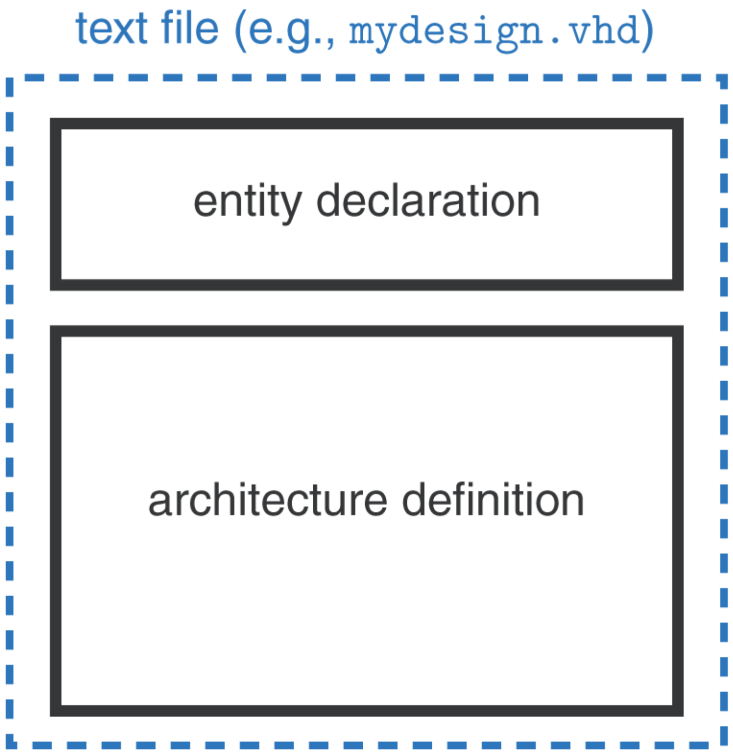
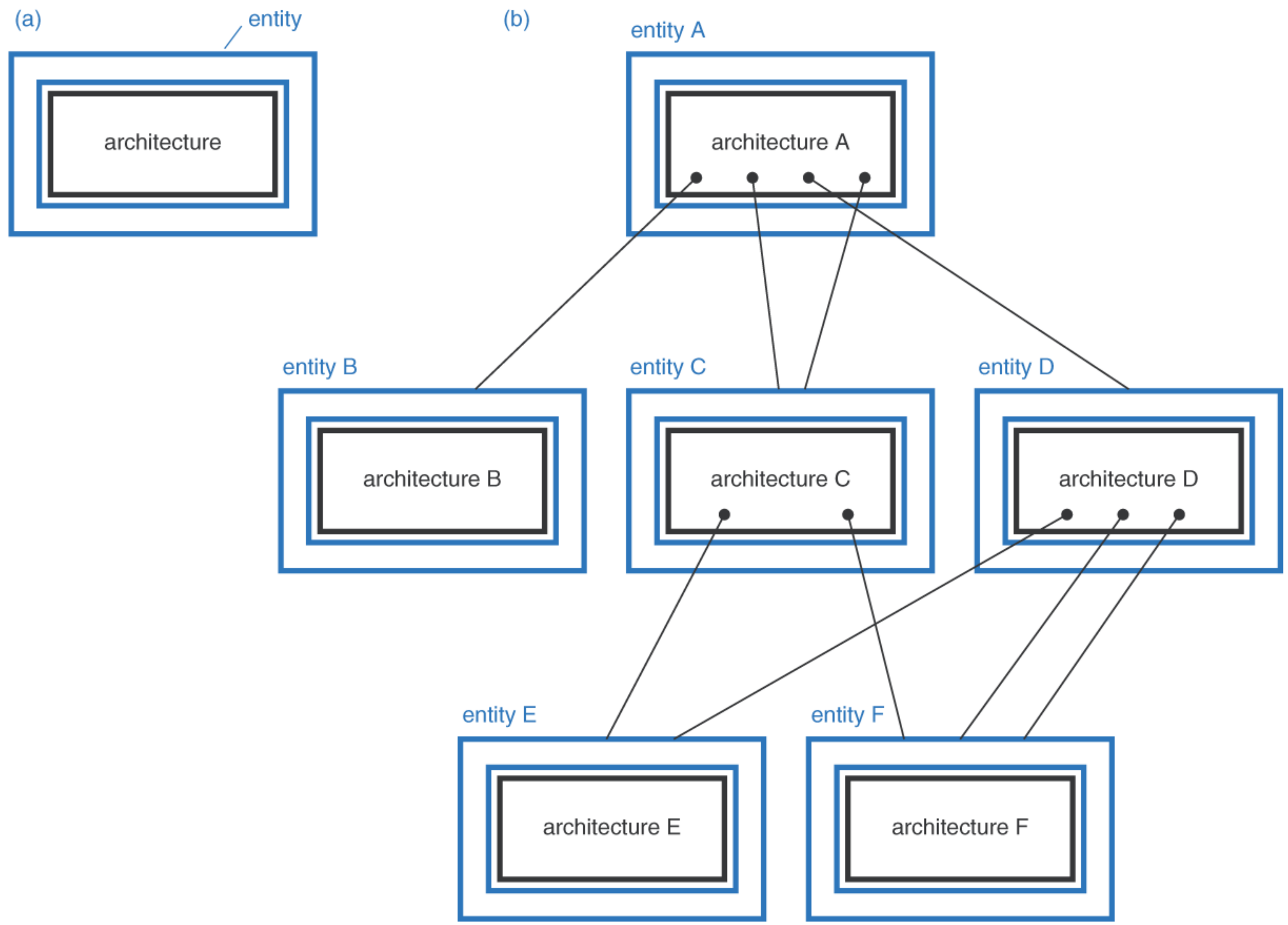


Figure 5-3

VHDL program file structure.

VHDL entities and architectures



VHDL program example

- VHDL program for an “inhibit” gate
  - The output of the gate is “1” only when X=1 and Y=0, otherwise the output is “0”

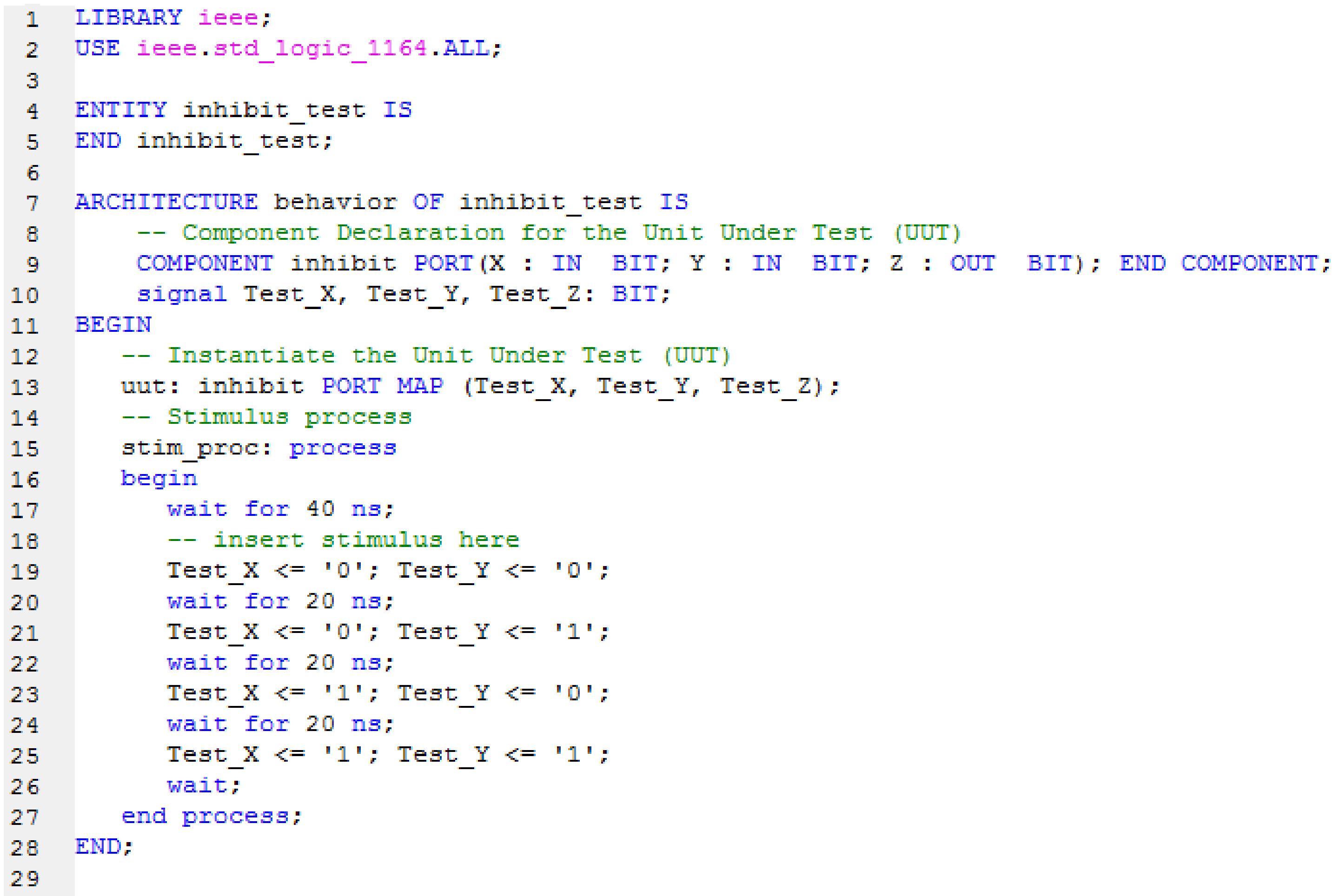
```
entity Inhibit is     -- also known as 'BUT-NOT'
  port (X,Y: in BIT;      --   as in 'X but not Y'
        Z:   out BIT);   --   (see [Klir, 1972])
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

VHDL test benches

- A test bench specifies a sequence of inputs to be applied by the simulator to a VHDL entity
- The entity being tested is often called the unit under test (UUT)
- Basic parts of a VHDL test-bench program:
  - There is an entity declaration, but without inputs or outputs
  - The architecture definition makes a component declaration for the UUT
  - A process with no sensitivity list starts at simulation time 0
  - Different input combinations are applied to the UUT at specified time points

VHDL testbench example

- Inhibit-gate test bench



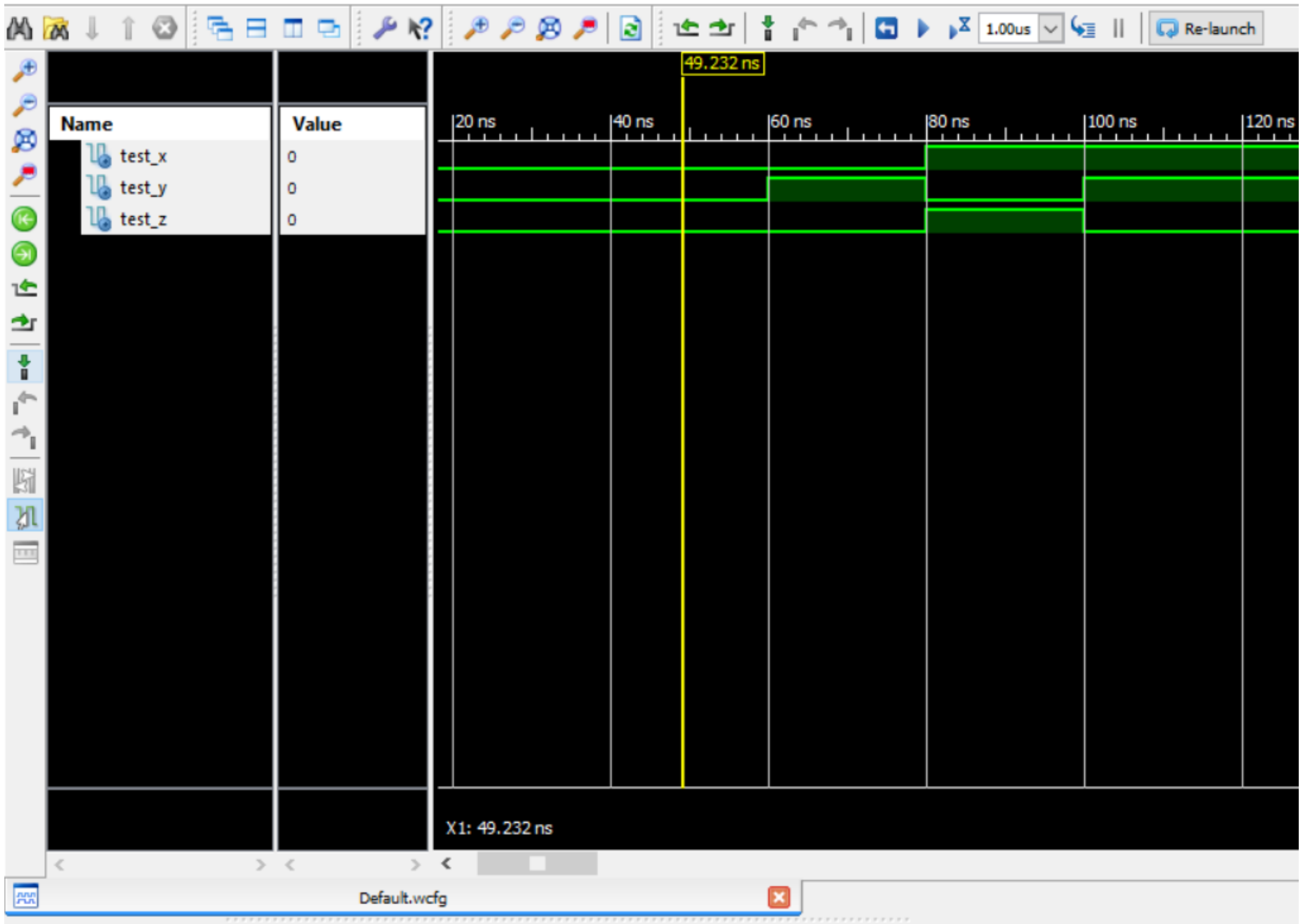
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY inhibit_test IS
END inhibit_test;

ARCHITECTURE behavior OF inhibit_test IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT inhibit PORT(X : IN  BIT; Y : IN  BIT; Z : OUT  BIT); END COMPONENT;
    signal Test_X, Test_Y, Test_Z: BIT;
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: inhibit PORT MAP (Test_X, Test_Y, Test_Z);
   -- Stimulus process
   stim_proc: process
   begin
      wait for 40 ns;
      -- insert stimulus here
      Test_X <= '0'; Test_Y <= '0';
      wait for 20 ns;
      Test_X <= '0'; Test_Y <= '1';
      wait for 20 ns;
      Test_X <= '1'; Test_Y <= '0';
      wait for 20 ns;
      Test_X <= '1'; Test_Y <= '1';
      wait;
   end process;
END;
```

VHDL simulation

- A VHDL simulator is used to observe the operation of a VHDL program
- Simulator operation begins at time zero
- Signals and variables need to be initialized
- The simulator then begins the execution of all processes and concurrent statements

Waveforms

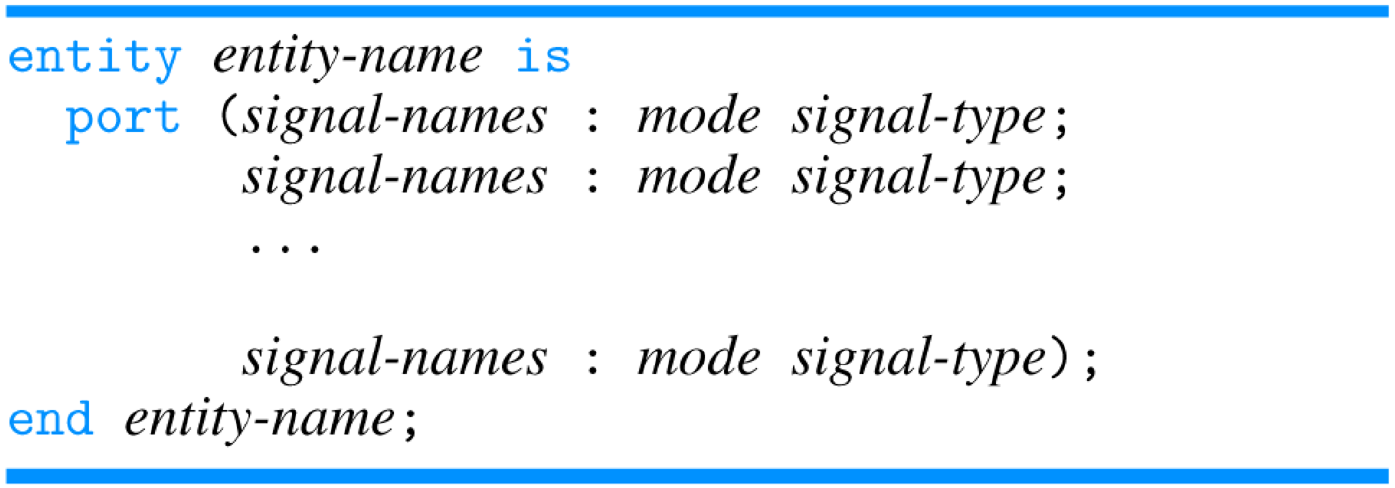
- Inhibit-gate simulation waveforms



Name	Value
test_x	0
test_y	0
test_z	0

X1: 49.232 ns

VHDL entity declaration (1)



```
entity entity-name is
   port (signal-names : mode signal-type;
         signal-names : mode signal-type;
         ...

         signal-names : mode signal-type);
end entity-name;
```

- ***entity-name***: a user-selected identifier to name to entity
- ***signal-names***: a comma-separated list of one or more user-selected identifiers to name external-interface signals

VHDL entity declaration (2)

```
entity entity-name is
   port (signal-names  :  mode  signal-type;
         signal-names  :  mode  signal-type;
         ...

         signal-names  :  mode  signal-type);
end entity-name;
```

- ***mode***: one of four reserved words, specifying the signal direction
  - in: the signal is an input to the entity
  - out: the signal is an output of the entity
  - buffer: the signal is an output of the entity, and its value can also be read inside the entity
  - inout: the signal can be used as an input or an output of the entity
- ***signal-type***: a built-in or user-defined signal types

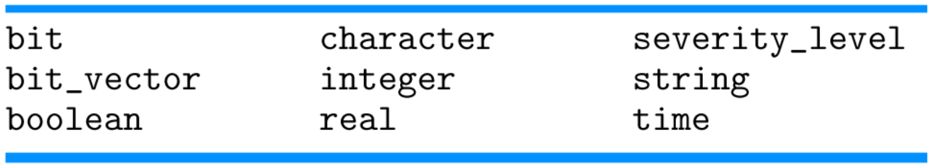
VHDL architecture declaration

- Syntax of a VHDL architecture definition

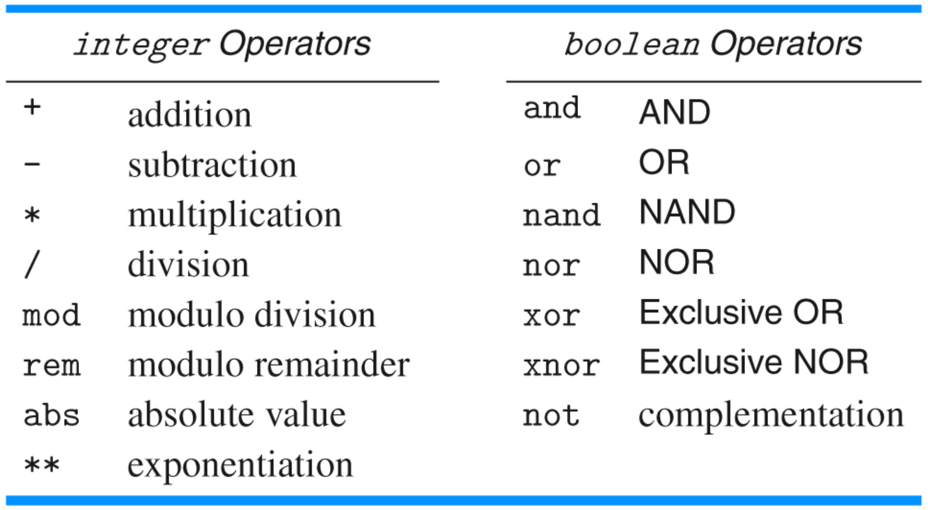
```
architecture architecture-name of entity-name is
   type declarations
   signal declarations
   constant declarations
   function definitions
   procedure definitions
   component declarations
begin
   concurrent-statement
   . . .
   concurrent-statement
end architecture-name;
```

VHDL types, constants and arrays

- Every signal, variable and constant in a VHDL program must have an associated type
- VHDL has predefined types
- VHDL predefined operators for integer and boolean types



`bit`	`character`	`severity_level`
`bit_vector`	`integer`	`string`
`boolean`	`real`	`time`



`integer` *Operators*		*`boolean`* *Operators*	
`+`	addition	`and`	AND
`-`	subtraction	`or`	OR
`*`	multiplication	`nand`	NAND
`/`	division	`nor`	NOR
`mod`	modulo division	`xor`	Exclusive OR
`rem`	modulo remainder	`xnor`	Exclusive NOR
`abs`	absolute value	`not`	complementation
`**`	exponentiation		

VHDL user-defined types (1)

- Syntax of VHDL type and constant declarations

```
type type-name is (value-list);

subtype subtype-name is type-name start to end;
subtype subtype-name is type-name start downto end;

constant constant-name: type-name := value;
```

- For example

```
type traffic_light_state is (reset, stop, wait, go);
```

VHDL user-defined types (2)

- Another example

```
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
                   );
subtype STD_LOGIC is resolved STD_ULOGIC;
```

VHDL subtypes and constants

- VHDL allows users to create subtypes
  - Examples:

```
subtype twoval_logic is std_logic range ‘0’ to ‘1’;
subtype fourval_logic is std_logic range ‘X’ to ‘Z’;
subtype negint is integer range -2147483647 to -1;
subtype bitnum is integer range 31 downto 0;
subtype natural is integer range 0 to highest-integer;
subtype positive is integer range 1 to highest-integer;
```

- VHDL constant
  - Examples:

```
constant BUS_SIZE: integer := 32;
constant MSB: integer := BUS_SIZE - 1;
constant Z: character := ‘Z’;
```

VHDL arrays (1)

- VHDL array is an ordered set of elements of the same type, where each element is selected by an array index
  - Syntax of VHDL array declarations

```
type type-name is array (start to end) of element-type;

type type-name is array (start downto end) of element-type;

type type-name is array (range-type) of element-type;

type type-name is array (range-type range start to end) of element-type;

type type-name is array (range-type range start downto end) of element-type;
```

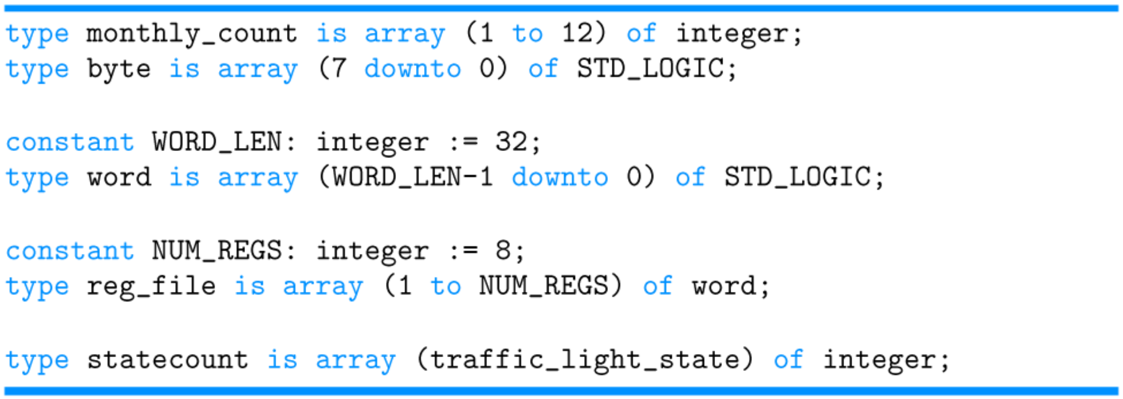
  - Examples of VHDL array declarations

```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;

constant WORD_LEN: integer := 32;
type word is array (WORD_LEN-1 downto 0) of STD_LOGIC;

constant NUM_REGS: integer := 8;
type reg_file is array (1 to NUM_REGS) of word;

type statecount is array (traffic_light_state) of integer;
```

VHDL arrays (2)

- Array initialization examples

```
B := (‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’);
W := (0=>’0’, 8=>’0’, 16=>’0’, 24=>’0’, others=>’1’);
B := “11111111”;
W := “11111110111111101111111011111110”;
```



```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;

constant WORD_LEN: integer := 32;
type word is array (WORD_LEN-1 downto 0) of STD_LOGIC;

constant NUM_REGS: integer := 8;
type reg_file is array (1 to NUM_REGS) of word;

type statecount is array (traffic_light_state) of integer;
```

VHDL functions and procedures

- A VHDL function accepts a number of arguments and returns a result
- A VHDL procedure is similar to a function, except it does not return a result

VHDL function Syntax

```
function function-name (
        signal-names  :  signal-type;
        signal-names  :  signal-type;
        ...
        signal-names  :  signal-type
) return return-type is
    type declarations
    constant declarations
    variable declarations
    function definitions
    procedure definitions
begin
    sequential-statement
    ...
    sequential-statement
end function-name;
```

VHDL function example

```
architecture Inhibit_archf of Inhibit is

function ButNot (A, B: bit) return bit is
begin
  if B = '0' then return A;
  else return '0';
  end if;
end ButNot;

begin
  Z <= ButNot(X,Y);
end Inhibit_archf;
```

VHDL libraries and packages (1)

- A VHDL **library** is a place where VHDL compiler stores information about a particular design project

- For a given VHDL design the compiler automatically creates and uses a library named “work”

- A developer can specify the name of the library using a library clause at the beginning of the design file, for example:

```
library ieee;
```

VHDL libraries and packages (2)

- A VHDL **package** is a file containing definitions of objects that can be used in other programs
- A design can "use" a package by including a use clause at the beginning of the design file
- For example, to use all of the definitions in the IEEE standard 1164 package, we could write:

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

- One can also write the name of a particular object to use its definition, for example:

```
use ieee.std_logic_1164.std_ulogic;
```

VHDL component

- In VHDL each concurrent statement executes simultaneously
- The most basic of VHDL's concurrent statements is the component statement
- Syntax of a VHDL component statement:

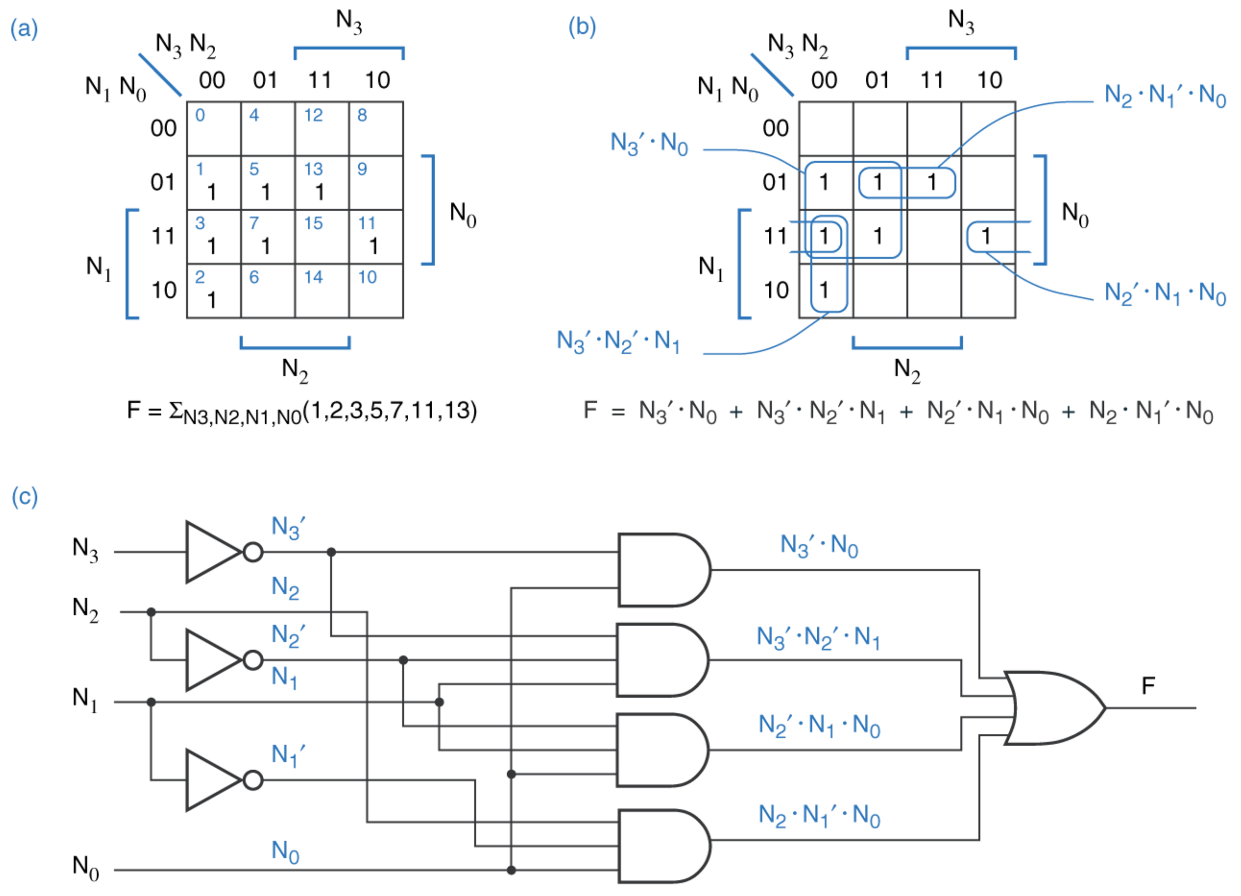
```
label: component-name port map(signal1, signal2, ..., signaln);

label: component-name port map(port1=>signal1, port2=>signal2, ..., portn=>signaln);
```

- Syntax of a VHDL component declaration:

```
component component-name
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end component;
```

Example: Prime-number detector (1)

- Gate-level circuit implementation


(a)
N3 N2
N1 N0
00 01 11 10
N3
N2
N1
N0
$F = \Sigma_{N3,N2,N1,N0}(1,2,3,5,7,11,13)$
(b)
$N_3' \cdot N_0$
$N_3' \cdot N_2' \cdot N_1$
$N_2 \cdot N_1' \cdot N_0$
$N_2' \cdot N_1 \cdot N_0$
$F = N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 + N_2' \cdot N_1 \cdot N_0 + N_2 \cdot N_1' \cdot N_0$
(c)
N_3
N_2
N_1
N_0
N_3'
N_2'
N_1'
F

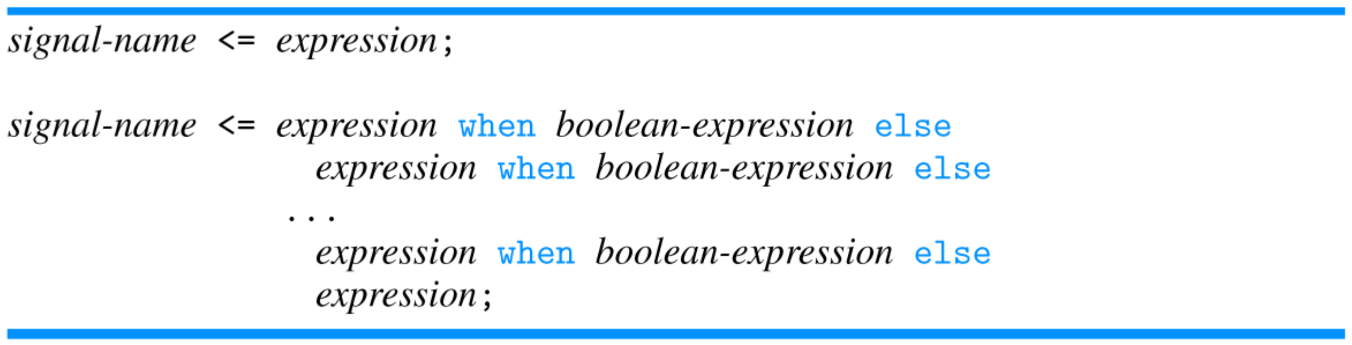
Example: Prime-number detector (2)

- VHDL implementation (structural design)

```
library IEEE;
use IEEE.std_logic_1164.all;
library unisim;
use unisim.vcomponents.all;

entity prime is
    port ( N: in STD_LOGIC_VECTOR (3 downto 0);
           F: out STD_LOGIC );
end prime;

architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
  U1: INV port map (N(3), N3_L);
  U2: INV port map (N(2), N2_L);
  U3: INV port map (N(1), N1_L);
  U4: AND2 port map (N3_L, N(0), N3L_N0);
  U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
  U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
  U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
  U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

VHDL signal assignment (1)

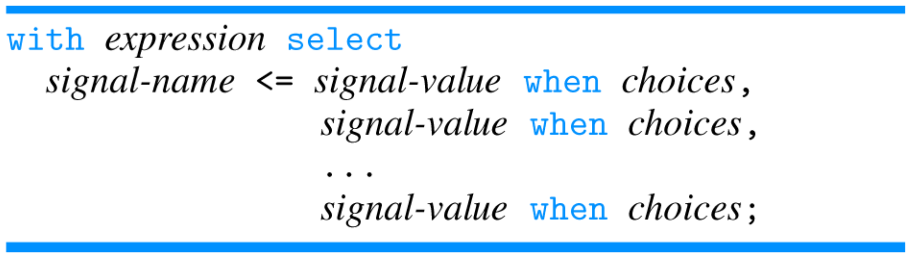
- Concurrent signal-assignment statement



```
signal-name <= expression;

signal-name <= expression when boolean-expression else
                 expression when boolean-expression else
                . . .
                 expression when boolean-expression else
                 expression;
```

VHDL signal assignment (2)

- Selected signal-assignment statement



```
with expression select
   signal-name <= signal-value when choices,
                  signal-value when choices,
                  . . .
                  signal-value when choices;
```

Signal assignment examples (1)

- Prime-number detector VHDL implementation (dataflow designs)
  - Using concurrent signal assignments

```
architecture prime2_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= not N(3)                           and N(0);
  N3L_N2L_N1 <= not N(3) and not N(2) and     N(1)         ;
  N2L_N1_N0  <=              not N(2) and     N(1) and N(0);
  N2_N1L_N0  <=                  N(2) and not N(1) and N(0);
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime2_arch;
```

Signal assignment examples (2)

- Prime-number detector VHDL implementation (dataflow designs)
  - Using conditional signal assignments

```
architecture prime3_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= '1' when N(3)='0' and N(0)='1' else '0';
  N3L_N2L_N1 <= '1' when N(3)='0' and N(2)='0' and N(1)='1' else '0';
  N2L_N1_N0  <= '1' when N(2)='0' and N(1)='1' and N(0)='1' else '0';
  N2_N1L_N0  <= '1' when N(2)='1' and N(1)='0' and N(0)='1' else '0';
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime3_arch;
```

Signal assignment examples (3)

- Prime-number detector VHDL implementation (dataflow designs)
  - Using selected signal assignments

```
architecture prime4_arch of prime is
begin
  with N select
    F <= '1' when "0001",
         '1' when "0010",
         '1' when "0011" | "0101" | "0111",
         '1' when "1011" | "1101",
         '0' when others;
end prime4_arch;
```

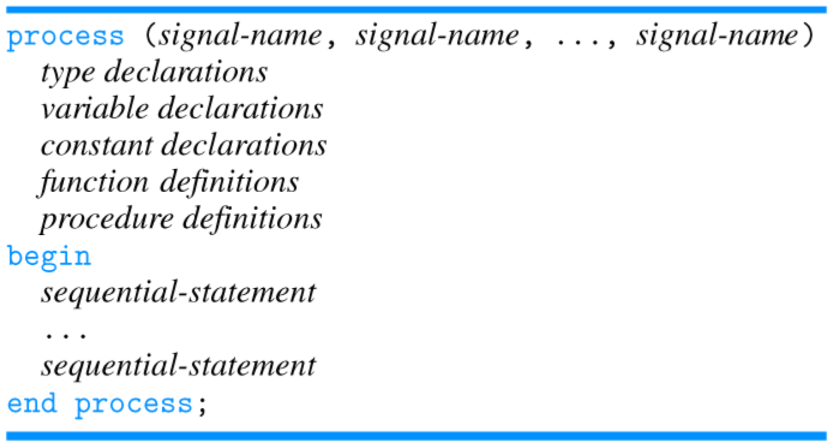
Signal assignment examples (4)

- Prime-number detector VHDL implementation (behavioral design)
    - A more behavioral description of the prime-number detector

```
architecture prime5_arch of prime is
begin
  with CONV_INTEGER(N) select
    F <= '1' when 1 | 2 | 3 | 5 | 7 | 11 | 13,
         '0' when others;
end prime5_arch;
```

VHDL process (1)

- A VHDL process is a collection of sequential statements
  - Execute in parallel with other concurrent statements and other processes
- Syntax of a VHDL process statement



```
process (signal-name, signal-name, ..., signal-name)
   type declarations
   variable declarations
   constant declarations
   function definitions
   procedure definitions
begin
   sequential-statement
   ...
   sequential-statement
end process;
```

- A process may not declare signals, only variables

```
variable variable-names : variable-type;
```

VHDL process (2)

- How does a VHDL process work?
  - A process is always either running or suspended;
  - The list of signals, called sensitivity list, determines when the process runs;
  - A process initially is suspended;
  - When any signal in its sensitivity list changes value, the process resumes execution;
  - If any signal in the sensitivity list changes value as a result of the process, the process runs again;

VHDL process example

- Prime-number detector implementation
  - Using VHDL process

```
architecture prime6_arch of prime is
begin
  process(N)
    variable N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
  begin
    N3L_N0     := not N(3)                               and N(0);
    N3L_N2L_N1 := not N(3) and not N(2) and     N(1)          ;
    N2L_N1_N0  :=              not N(2) and     N(1) and N(0);
    N2_N1L_N0  :=                  N(2) and not N(1) and N(0);
    F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
  end process;
end prime6_arch;
```

VHDL if statement

- Syntax of a VHDL **if** statement

```
if boolean-expression then sequential-statements
end if;

if boolean-expression then sequential-statements
else sequential-statements
end if;

if boolean-expression then sequential-statements
elsif boolean-expression then sequential-statements
...
elsif boolean-expression then sequential-statements
end if;

if boolean-expression then sequential-statements
elsif boolean-expression then sequential-statements
...
elsif boolean-expression then sequential-statements
else sequential-statements
end if;
```

If statement example

- Prime-number detector implementation
  - Using VHDL **if** statement

```
architecture prime7_arch of prime is
begin
  process(N)
    variable NI: INTEGER;
  begin
    NI := CONV_INTEGER(N);
    if NI=1 or NI=2 then F <= '1';
    elsif NI=3 or NI=5 or NI=7 or NI=11 or NI=13 then F <= '1';
    else F <= '0';
    end if;
  end process;
end prime7_arch;
```

VHDL case statement

- Select among multiple alternatives based on the value of just one signal or expression
- Syntax of a VHDL **case** statement

```
case expression is
  when choices => sequential-statements
  ...
  when choices => sequential-statements
end case;
```

Case statement example

- Prime-number detector implementation
  - Using VHDL **case** statement

```
architecture prime8_arch of prime is
begin
  process(N)
  begin
    case CONV_INTEGER(N) is
      when 1 => F <= '1';
      when 2 => F <= '1';
      when 3 | 5 | 7 | 11 | 13 => F <= '1';
      when others => F <= '0';
    end case;
  end process;
end prime8_arch;
```

VHDL loop statements

- Syntax of a basic VHDL **loop** statement

```
loop
   sequential-statement
   . . .
   sequential-statement
end loop;
```

- Syntax of a VHDL **for** loop

```
for identifier in range loop
   sequential-statement
   . . .
   sequential-statement
end loop;
```

- Syntax of a VHDL **while** loop

```
while boolean-expression loop
   sequential-statement
   . . .
   sequential-statement
end loop;
```

Loop statement example

- Prime-number detector implementation
  - Using VHDL **for** statement

```
entity prime9 is
    port ( N: in STD_LOGIC_VECTOR (15 downto 0);
           F: out STD_LOGIC );
end prime9;

architecture prime9_arch of prime9 is
begin
  process(N)
  variable NI: INTEGER;
  variable prime: boolean;
  begin
    NI := CONV_INTEGER(N);
    prime := true;
    if NI=1 or NI=2 then null; -- take care of boundary cases
    else for i in 2 to 253 loop
           if (NI mod i = 0) and (NI /= i) then
             prime := false; exit;
           end if;
         end loop;
    end if;
    if prime then F <= '1'; else F <= '0'; end if;
  end process;
end prime9_arch;
```

VHDL delay statements

- VHDL allows users to specify a time delay using the keyword *after* in any signal-assignment, including sequential, concurrent, conditional and selected assignments
  - For example

```
Z <= ‘1’ after 4 ns when X=‘1’ and Y=‘0’ else ‘0’
  after 3 ns;
```

- Another way to introduce time delay is with *wait*, a sequential statement.
  - Can be used to suspend a process

wait statement & sensitivity list

- A process with sensitivity is functionally equivalent to a process statement with a *wait* statement

```
process(clk)
begin
   clk <= not(clk) after 50 ns;
end process;
```

```
process
begin
   clk <= not(clk) after 50 ns;
   wait on clk;
end process;
```

VHDL *wait* statement example

- Using the VHDL *wait* statement to generate input waveforms in a test-bench program

```
entity InhibitTestBench is
end InhibitTestBench;

architecture InhibitTB_arch of InhibitTestBench is
component Inhibit port (X,Y: in BIT; Z: out BIT); end component;
signal XT, YT, ZT: BIT;
begin
  U1: Inhibit port map (XT, YT, ZT);
  process
  begin
    XT <= '0'; YT <= '0';
    wait for 10 ns;
    XT <= '0'; YT <= '1';
    wait for 10 ns;
    XT <= '1'; YT <= '0';
    wait for 10 ns;
    XT <= '1'; YT <= '1';
    wait; -- this suspends the process indefinitely
  end process;
end InhibitTB_arch;
```