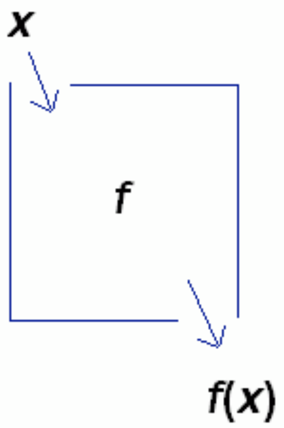
CPE 462
VHDL: Simulation and Synthesis

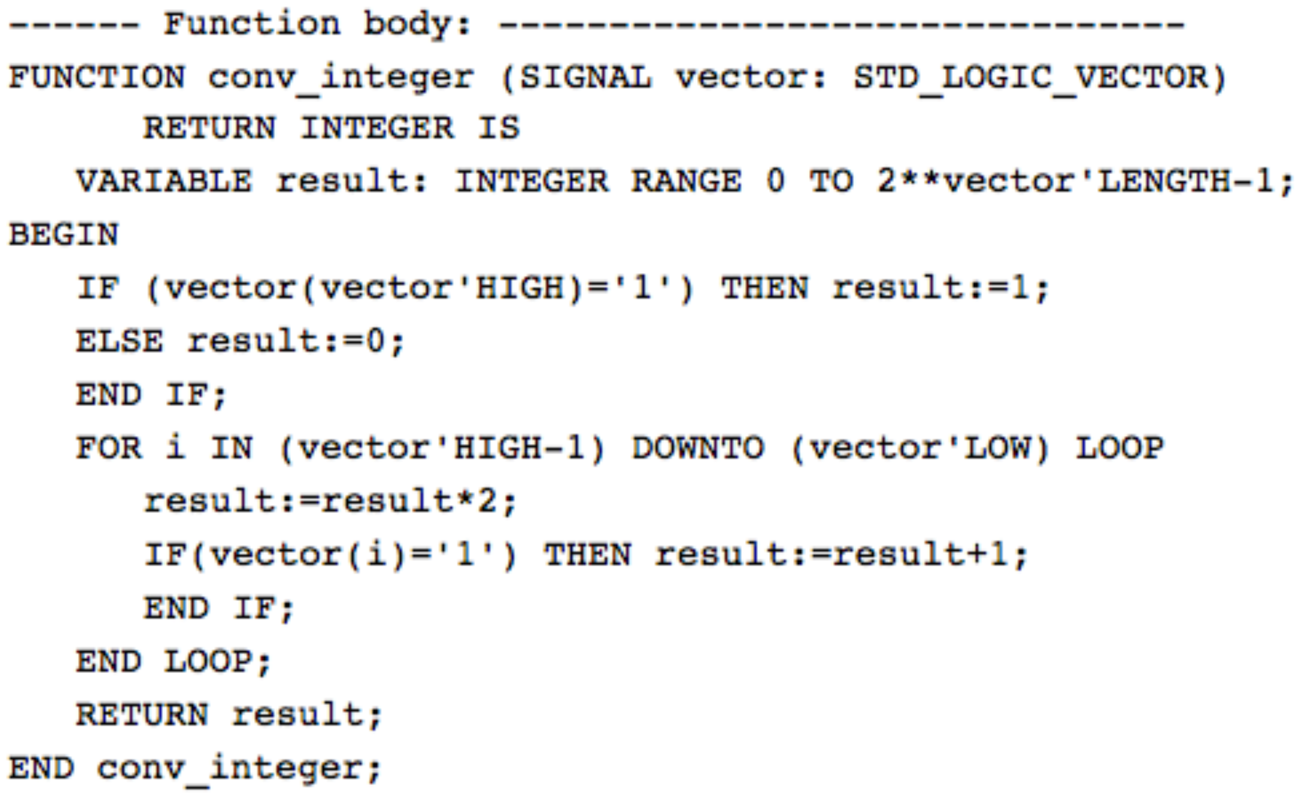
Topic #08 - d) Procedures

Review of VHDL functions



- In mathematics a function is just a box... You add some inputs, and you will get some modified outputs

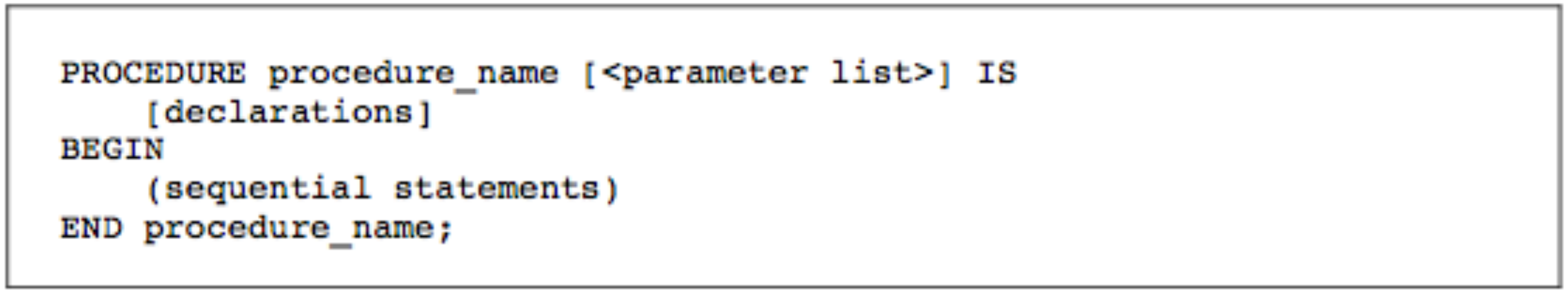
- A VHDL FUNCTION takes a set of inputs and returns a single output.



```
------ Function body: ------------------------------
FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
      RETURN INTEGER IS
   VARIABLE result: INTEGER RANGE 0 TO 2**vector'LENGTH-1;
BEGIN
   IF (vector(vector'HIGH)='1') THEN result:=1;
   ELSE result:=0;
   END IF;
   FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
      result:=result*2;
      IF(vector(i)='1') THEN result:=result+1;
      END IF;
   END LOOP;
   RETURN result;
END conv_integer;
```

VHDL PROCEDURE

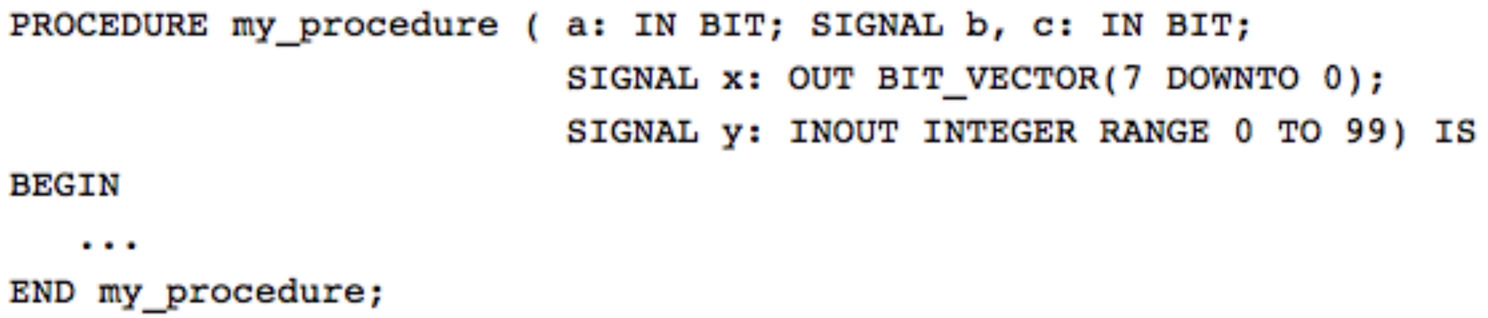
- A PROCEDURE is very similar to a FUNCTION and has the same basic purposes.
- However, a procedure can return more than one value.
- Like a FUNCTION, two parts are necessary to construct and use a PROCEDURE: the procedure itself (procedure body) and a procedure call.



```
PROCEDURE procedure_name [<parameter list>] IS
    [declarations]
BEGIN
    (sequential statements)
END procedure_name;
```

Overview of PROCEDUREs

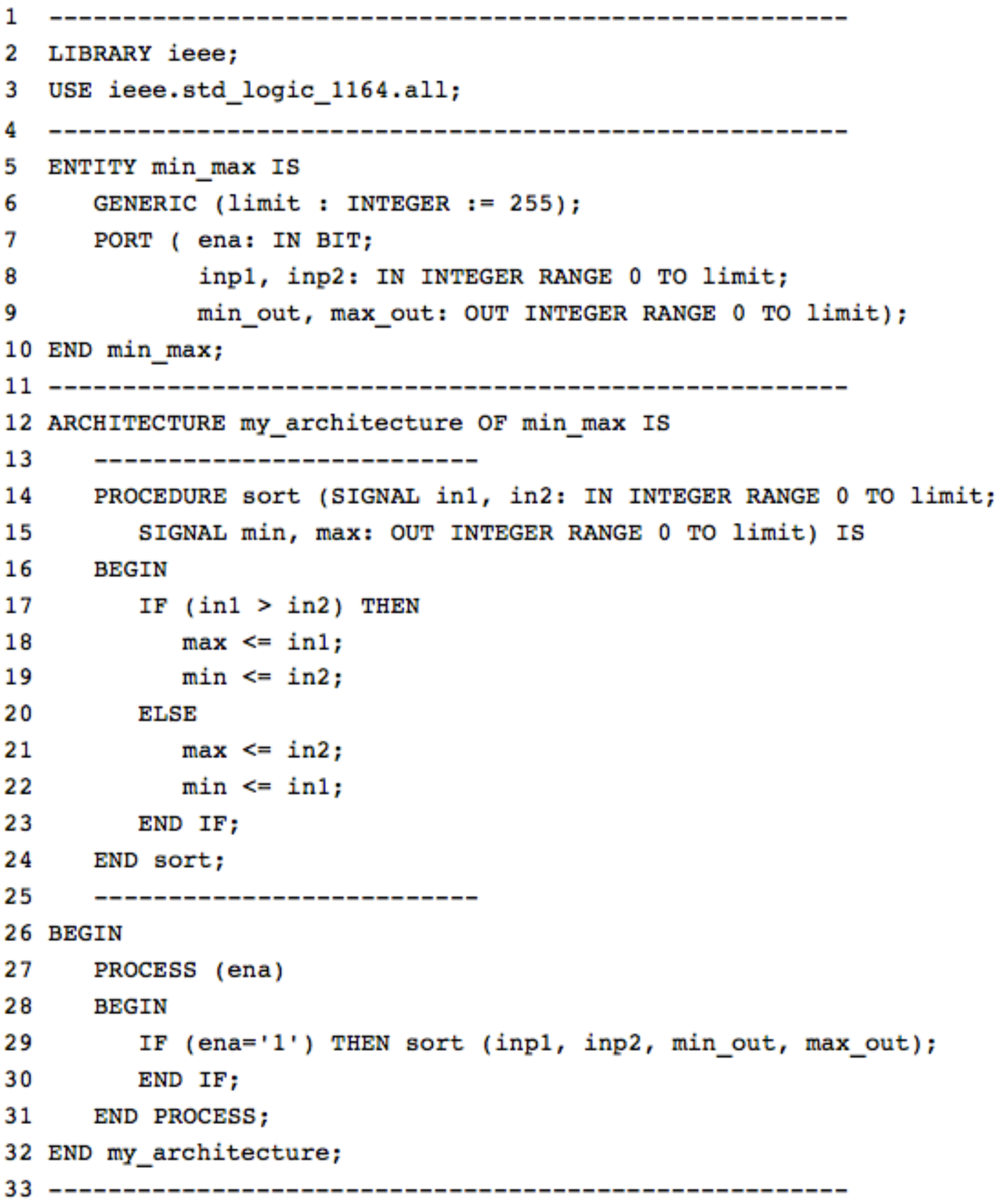
- A PROCEDURE can have any number of IN, OUT, or INOUT parameters, which can be SIGNALS,VARIABLES, or CONSTANTS.
- For input signals (mode IN), the default is CONSTANT, whereas for output signals (mode OUT or INOUT) the default is VARIABLE.
- A PROCEDURE like functions can be on a package or inside the main code
- Here is an example of the header section of a procedure...



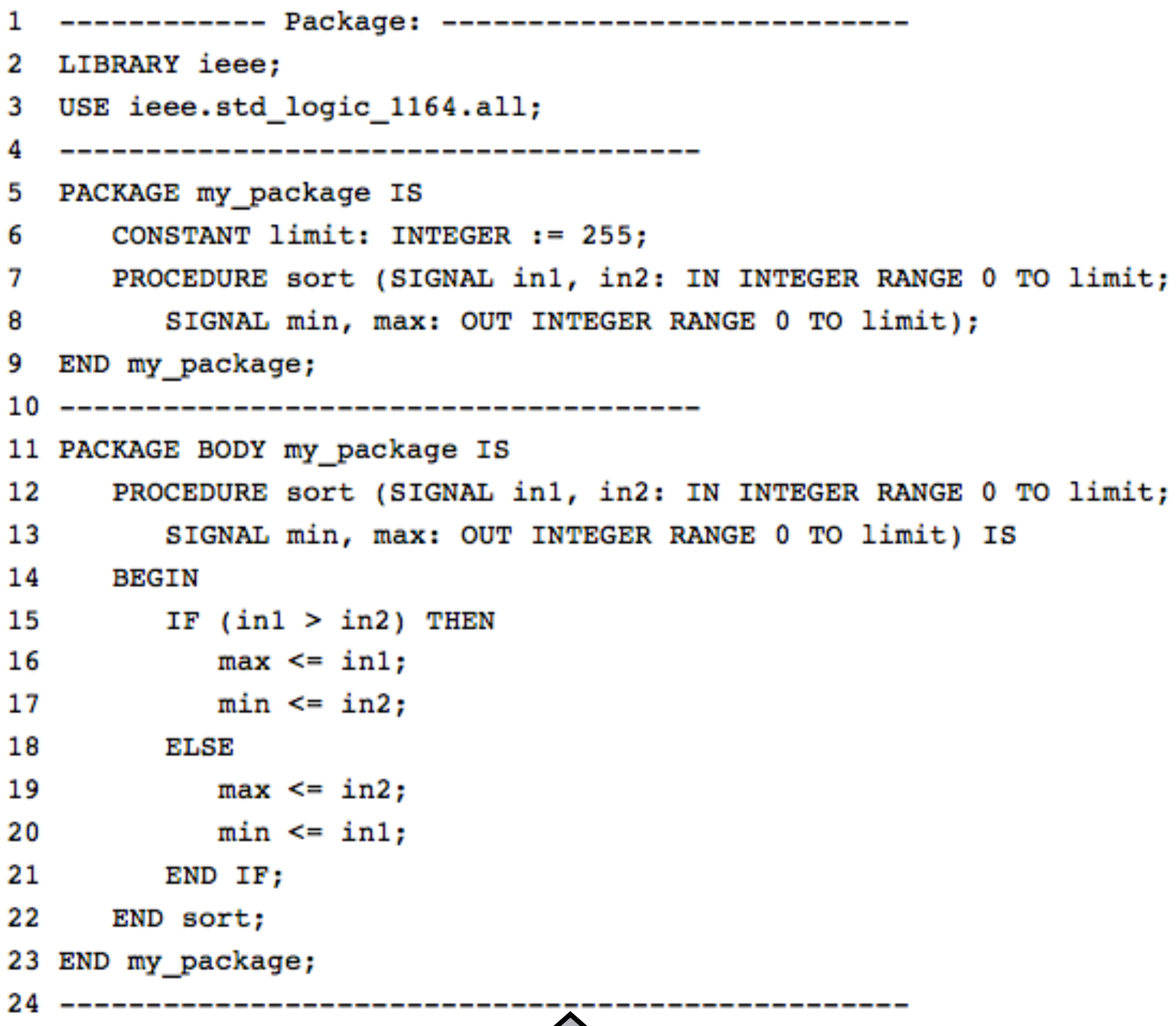
```
PROCEDURE my_procedure ( a: IN BIT; SIGNAL b, c: IN BIT;
                         SIGNAL x: OUT BIT_VECTOR(7 DOWNTO 0);
                         SIGNAL y: INOUT INTEGER RANGE 0 TO 99) IS
BEGIN
   ...
END my_procedure;
```

Two PROCEDURE Examples

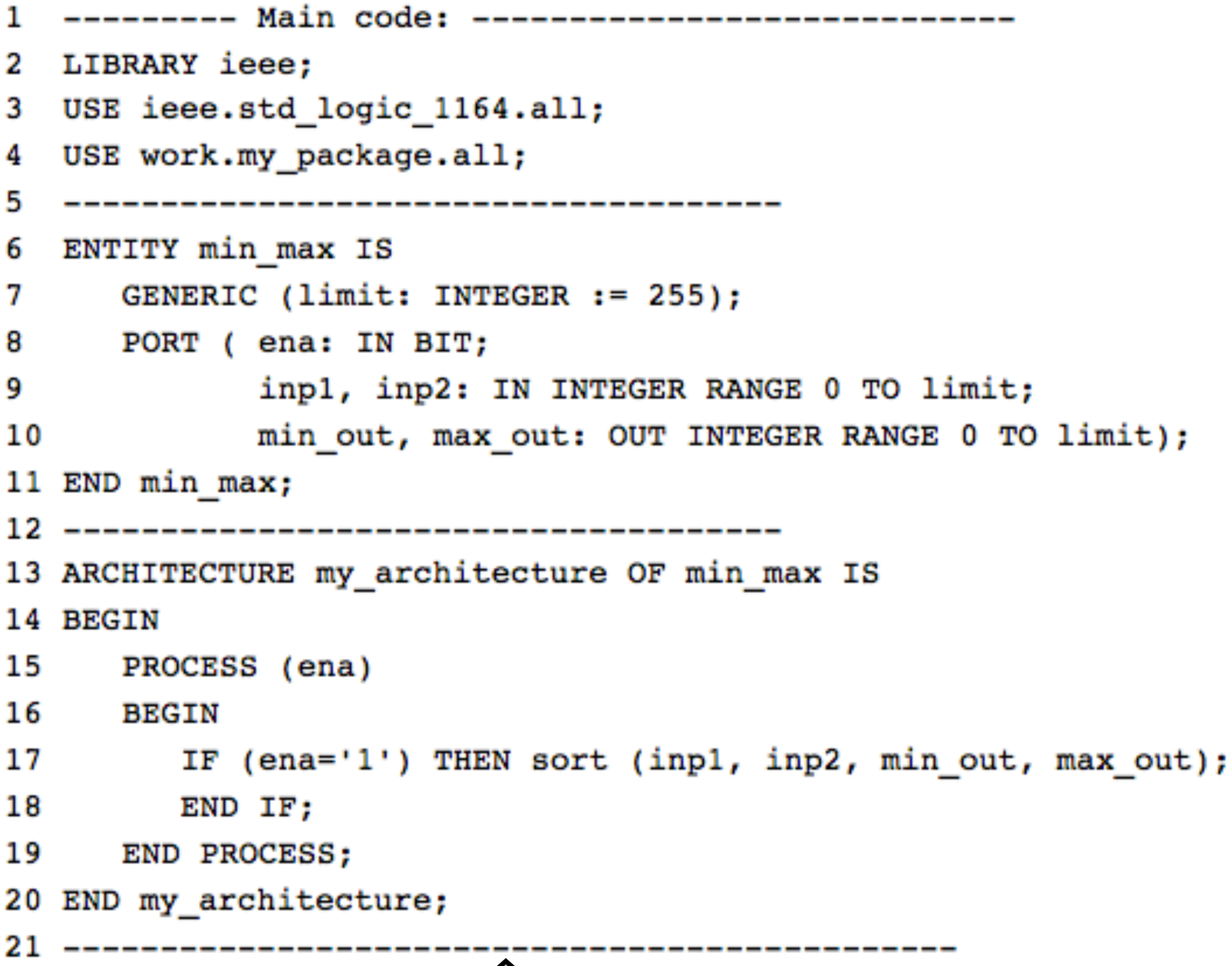
- The min_max code makes use of a PROCEDURE called sort.
- It takes two 8-bit unsigned integers as inputs (inp1, inp2), sorts them, then outputs the smaller value at min_out and the higher value at max_out.
- The PROCEDURE is located in the declarative part of the ARCHITECTURE (main code).
- The PROCEDURE call, sort(inp1,inp2,min_out,max_out), is a statement on its own.



```
-----------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------------------
ENTITY min_max IS
   GENERIC (limit : INTEGER := 255);
   PORT ( ena: IN BIT;
          inp1, inp2: IN INTEGER RANGE 0 TO limit;
          min_out, max_out: OUT INTEGER RANGE 0 TO limit);
END min_max;
-----------------------------------------------------
ARCHITECTURE my_architecture OF min_max IS
   --------------------------
   PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
      SIGNAL min, max: OUT INTEGER RANGE 0 TO limit) IS
   BEGIN
      IF (in1 > in2) THEN
         max <= in1;
         min <= in2;
      ELSE
         max <= in2;
         min <= in1;
      END IF;
   END sort;
   --------------------------
BEGIN
   PROCESS (ena)
   BEGIN
      IF (ena='1') THEN sort (inp1, inp2, min_out, max_out);
      END IF;
   END PROCESS;
END my_architecture;
-----------------------------------------------------
```



```
------------ Package: ---------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------
PACKAGE my_package IS
   CONSTANT limit: INTEGER := 255;
   PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
      SIGNAL min, max: OUT INTEGER RANGE 0 TO limit);
END my_package;
-------------------------------------
PACKAGE BODY my_package IS
   PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 TO limit;
      SIGNAL min, max: OUT INTEGER RANGE 0 TO limit) IS
   BEGIN
      IF (in1 > in2) THEN
         max <= in1;
         min <= in2;
      ELSE
         max <= in2;
         min <= in1;
      END IF;
   END sort;
END my_package;
-------------------------------------------------
```



```
--------- Main code: ---------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-------------------------------------
ENTITY min_max IS
   GENERIC (limit: INTEGER := 255);
   PORT ( ena: IN BIT;
          inp1, inp2: IN INTEGER RANGE 0 TO limit;
          min_out, max_out: OUT INTEGER RANGE 0 TO limit);
END min_max;
-------------------------------------
ARCHITECTURE my_architecture OF min_max IS
BEGIN
   PROCESS (ena)
   BEGIN
      IF (ena='1') THEN sort (inp1, inp2, min_out, max_out);
      END IF;
   END PROCESS;
END my_architecture;
-------------------------------------------
```

Package file

Main code

- Same procedure as the previous slide, but inside a package
- This means, we now need two files to run the procedure.

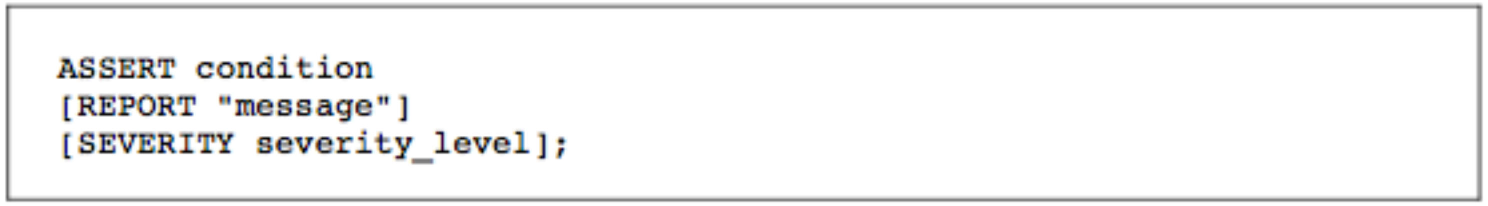
Differences between FUNCTION and PROCEDURES

- A FUNCTION has zero or more input parameters and a single return value. The input parameters can only be CONSTANTS (default) or SIGNALS (VARIABLES are not allowed).
- A PROCEDURE can have any number of IN, OUT, and INOUT parameters, which can be SIGNALS, VARIABLES, or CONSTANTS. For input parameters the default is CONSTANT, whereas for output parameters the default is VARIABLE.
- A FUNCTION is called as part of an expression, while a PROCEDURE is a statement on its own.
- In both, WAIT and COMPONENTS are not synthesizable.
- Both FUNCTIONS and PROCEDURES can be placed beneath the architecture or inside PACKAGES

Assert Operation

ASSERT

- Assert is a very useful operation for simulation!
- This is not synthesizable
- ASSERT is a non-synthesizable statement whose purpose is to write out messages (on the screen, for example) when problems are found during simulation.
- Its syntax is the following:

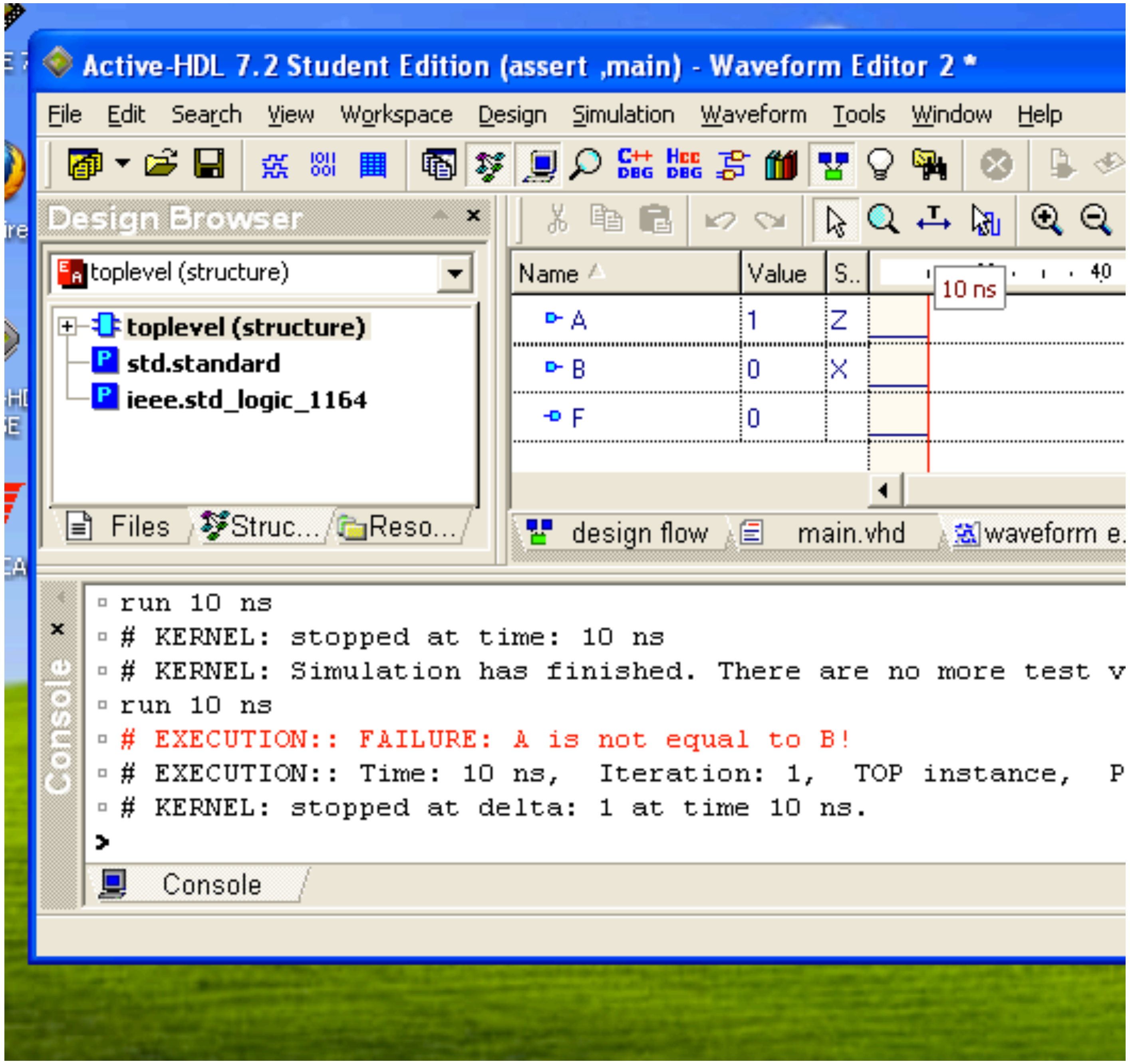


```
ASSERT condition
[REPORT "message"]
[SEVERITY severity_level];
```

The severity level can be: Note, Warning, Error (default), or Failure. The message is written when the condition is FALSE.

ASSERT example: severity failure will abort simulation

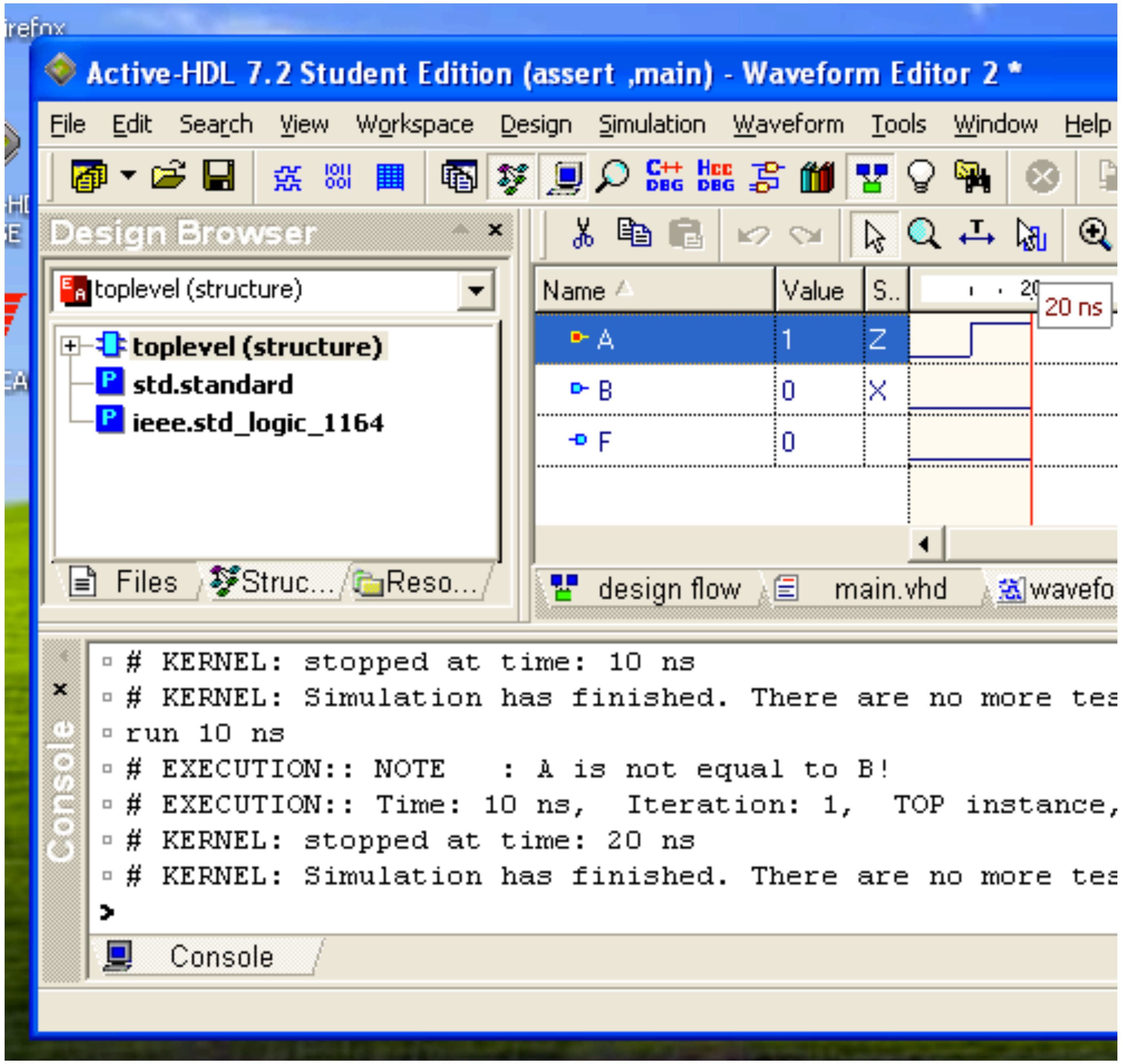
```
--entry_point.vhd
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity topLevel is
  port (A, B : in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is
begin

    assert (a=b)
    report "A is not equal to B!"
    severity failure;

    F <= A AND B;

end architecture;
```



```
run 10 ns
# KERNEL: stopped at time: 10 ns
# KERNEL: Simulation has finished. There are no more test v
run 10 ns
# EXECUTION:: FAILURE: A is not equal to B!
# EXECUTION:: Time: 10 ns,  Iteration: 1,  TOP instance,  P
# KERNEL: stopped at delta: 1 at time 10 ns.
>
```

ASSERT example: severity note will continue simulation

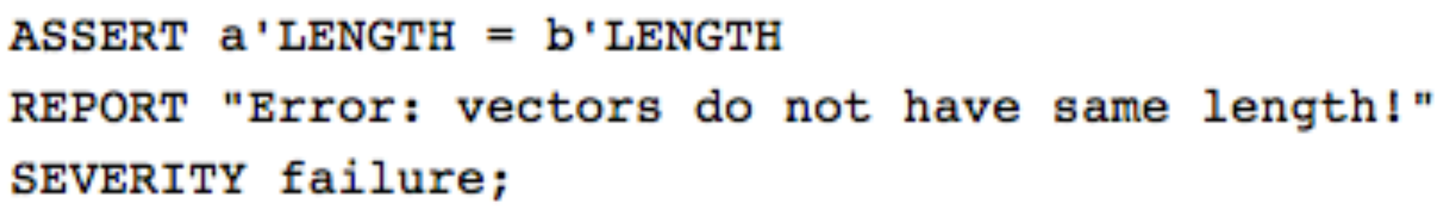
```
--entry_point.vhd
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity topLevel is
  port (A, B : in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is
begin

    assert (a=b)
    report "A is not equal to B!"
    severity note;

    F <= A AND B;

end architecture;
```



```
# KERNEL: stopped at time: 10 ns
# KERNEL: Simulation has finished. There are no more tes
run 10 ns
# EXECUTION:: NOTE   : A is not equal to B!
# EXECUTION:: Time: 10 ns,  Iteration: 1,  TOP instance,
# KERNEL: stopped at time: 20 ns
# KERNEL: Simulation has finished. There are no more tes
>
```

When to use assert

Example: Say that we have written a function to add two binary numbers, where it was assumed that the input parameters must have the same number of bits. In order to check such an assumption, the following ASSERT statement could be included in the function body:



```
ASSERT a'LENGTH = b'LENGTH
REPORT "Error: vectors do not have same length!"
SEVERITY failure;
```