INTRODUCTION TO FPGA PROGRAMMING

Lesson 08: Packages, Libraries and Parametrisation

Dr. Davide Cieri[1]

[1]Max-Planck-Institut für Physik, Munich

September 2024

MAX-PLANCK-INSTITUT FÜR PHYSIK

INTRODUCTION TO VHDL PACKAGES

- **Definition**
  - A VHDL package is a collection of related declarations that can be shared across multiple design units.
  - Packages provide a way to organize and encapsulate common code, such as constants, types, subprograms, and components.
- **Purpose of VHDL Packages**
  - Promotes code reuse by centralizing commonly used definitions.
  - Enhances readability and maintainability by organizing related code.
  - Facilitates modular design by separating interface and implementation.

STRUCTURE OF VHDL PACKAGES

- **Package Declaration**
  - Specifies the interface of the package.
  - Includes declarations of constants, types, subprograms, and components.
- **Package Body**
  - Contains the implementation of the subprograms declared in the package.
  - Optional if the package only contains type and constant declarations.

```
package MyPackage is
  constant MAX_SIZE : integer := 1024;
  type DataArray is array (0 to MAX_SIZE
      -1) of std_logic_vector(7 downto
      0);
  function Add(a, b: integer) return
      integer;
end MyPackage;

package body MyPackage is
  function Add(a, b: integer) return
      integer is
  begin
    return a + b;
  end function;
end MyPackage;
```

FUNCTIONS AND PROCEDURES IN PACKAGES

- Both functions and procedure should have a declaration and a body
- Return declaration of a function should be unconstrained
  - OK: `return signed;`. Not OK: `return signed(4 downto 0);`

```
package MyPackage is
function Add_Two_Numbers(
  a: integer;
  b: integer) return integer;
end package MyPackage;
```

```
package body MyPackage is
  function Add_Two_Numbers(
    a: integer;
    b: integer) return integer is
  begin
    return a + b;
  end function Add_Two_Numbers;
end package body MyPackage;
```

FUNCTIONS AND PROCEDURES IN PACKAGES

- Both functions and procedure should have a declaration and a body

```
package MyPackage is
  procedure Initialize_Signal(
    signal sig: out std_logic);
end package MyPackage;
```

```
package body MyPackage is
  procedure Initialize_Signal(
    signal sig: out std_logic) is
  begin
    sig <= '0';
  end procedure Initialize_Signal;
end package body MyPackage;
```

USING VHDL LIBRARIES

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.MyPackage.all;

entity Example is
    port (
        a, b : in  integer;
        sum  : out integer
    );
end Example;

architecture Behavioral of Example is
begin
    process(a, b)
    begin
        sum <= Add(a, b);
    end process;
end Behavioral;
```

- **Including Packages**
  - Use the `library` and `use` clauses to include a package in your design.

```
library mylib;
use mylib.MyPackage.all;
```

  - Reminder: `work` is the current library of the importing file
  - Libraries and mapping is done by the simulator or synthesizer software

DESIGN PARAMETRISATION

- Parametrisation in VHDL to instantiate similar blocks using parameters to adapt to the different requirements
  - Different data widths, sizes, and functionalities.
- *Reusability*: allows the same code to be used across different projects.
- *Maintainability*: easier to update and modify Design
- *Scalability*: Facilitates the creation of scalable designs with different requirements

VHDL GENERICS

- Parameters defined at the entity level, that enable instance specific constants
- They can be used in the port definitions (e.g. to define the data widths)
- They can be used like any other constants in the entity architecture
- If no default value is given, generics must be defined when instantiating (synthesis/simulation error)

```
entity Adder is
generic (
  WIDTH  : integer := 8;
  WIDTH2 : integer := 9
);
port (
  a, b : in  std_logic_vector(WIDTH-1 downto 0);
  sum  : out std_logic_vector(WIDTH2-1 downto 0)
);
end Adder;
```

```
adder_i : adder generic map (WIDTH => 32,
                             WIDTH2 => 33)
                port map ( a => a,
                           b => b,
                           sum => sum);
```

GENERICS USAGE (CONSTANTS)

- You can define constants from the a generic to use a common value
- Constants will always be related to the value of the generic
- Cannot be separate from the value of the generic (holds always a valid value)
- N.B. Constants cannot be used in the port declarations

```
entity Adder is
generic (
  WIDTH  : integer := 8
);
...
end Adder;

architecture rtl of Adder is
  constant W2 : integer := WIDTH*2;
begin
  ...
end rtl;
```

GENERICS USAGE (GENERATE STATEMENTS)

- The `generate` statement is used in VHDL to generate hardware design
- It can be combined with generics or constants to allow configurability
- Two `generate constructs available`
  - `if generate`: For conditional instantiations
  - `for generate`: For repeated instantiations
  - `case generate`: Conditional instantiations (VHDL-2008)
- `generate` can also be used to generate multiple processes or continuous assignments
- They must be placed in the module architecture after the `begin` statement

IF GENERATE STATEMENT

- Each `if generate` statement is evaluated at elaboration time, to determine whether the code is generated or not
- Generate condition must be deterministic at elaboration time
  - Use constants or generics
- VHDL-2008 introduced usage of `else`/`elseif` constructs (in 93 use separate `if generate` statements)

```
<GENERATE_LABEL> : if condition generate
-- statements
elsif condition2 generate
-- statements
else generate
-- statements
end generate;
```

```
GEN0 : if A=0 generate
  a1 : adder1
  generic map(...) port map(...);
else
  a2 : adder2
  generic map(...) port map(...);
end generate;
```

- N.B. In the same generic statement, you can instantiate as many module as you wish.

IF GENERATE STATEMENT

- You can also add continuous assignments or processes inside the generate statement
- Only useful to add "glue logic" to generated instantiations
- Standalone continuous assignment can be implemented using simpler VHDL code

```
GEN0 : if MOD = 0 generate
  sum <= a + b;
elsif MOD = 1 generate
  sum <= a + b + 1;
else
  sum <= a + b + 2;
end if;
```

```
GEN1 : if MOD = 0 generate
  process(a,b)
  begin
    sum <= a + b;
  end process;
else
  process(a,b)
  begin
    sum <= a + b + 1;
  end process;
end generate;
```

CASE GENERATE STATEMENT

- `case generate` introduced in VHDL-2008
- Useful when you have multiple conditions
- Remember to set a default (`others`)

```
<GENERATE_LABEL> : case expressions generate
when condition =>
-- statements
when condition2 =>
-- statements
when others =>
end generate;
```

```
GEN0 : case A generate
  when 1 =>
    a1 : adder1
    generic map(...) port map(...);
  when 2 =>
    a2 : adder2
    generic map(...) port map(...);
  when others =>
    add : adder
    generic map(...) port map(...);
end generate;
```

FOR GENERATE STATEMENT

- `for generate` are used to parallise block instantiations
- For expanded at elaboration time
- N.B. Always check the mapping of port-signals

```
GEN0 : for i in 0 to N_ADDERS generate
  add : adder
  port map(
    a => a(i), b => b(i), sum (i)
  );
end generate;
```

PARAMETRISE MEMORIES AND COUNTERS

- Parametrise by address width (`AWIDTH`)
  - Easy to determine memory depth (`DEPTH=2**AWIDTH`)
  - Only allows $2^{\texttt{AWIDTH}}$ memory locations
- Parameterise by memory depth (`DEPTH`)
  - Define precise memory sizes
  - Address width easy to calculate
    - From `ieee.math_real.all`
    - `AWIDTH=integer(ceil(log2(real(DEPTH))));`
  - Counters parametrised in a similar fashion
    - Take values from 0 to N (N+1 values in total)
    - Counter width is ceil(log2(N+1))

LAB 12: PACKAGES AND LIBRARIES

LAB 13: PARAMETERS AND PARAMETRISED GENERATION

The figures in these slides are taken from:

- Digital Design: Principles and Practices, Fourth Edition, John F. Wakerly, ISBN 0-13- 186389-4. ©2006, Pearson Education, Inc, Upper Saddle River, NJ. All rights reserved
- allaboutfpga.com
- nandland.com
- docs.amd.com
- https://www.symmetryelectronics.com/
- https://www.edn.com/
- Stephen A. Edwards, Columbia University, Fundamentals of Computer Systems, Spring 2012
- https://medium.com/well-red/state-machines-for-everyone-part-1-introduction-b7ac9aaf482e