

Hardware Modeling [VU] (191.011)

– WS25 –

Subprograms

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

- Functions
 - Call is **expression** returning a value
 - `pure`: No side effects, same parameters $\Rightarrow$ same return value
 - `impure`: Side effects possible, return value can vary for identical calls
- Procedures
 - Call is **statement** with side effects and no return value
- Subprogram call
 - Parameters passed in parentheses
`report to_string(x);` -- function call with one parameter
 - In case of zero parameters no parentheses
`stop;` -- procedure call without parameters

■ Simplified declaration syntax

```
1 [pure|impure] function designator [(parameter_list)] return TYPE_NAME is
2   [declarative_part]
3 begin
4   [statement part]  -- function body
5   return ...;
6 end function;
```

■ Type of the returned value can be scalar or composite

■ Default is `pure`

■ Primarily for computing values

⇒ Does not advance simulation time

■ Must not contain wait statements

- Must also hold for subprograms called inside the body

■ **Must** always end in `return`

- Declaration syntax
- Default parameter class is `constant`
 - Pass-by-copy
 - If signal attributes are required use `signal`
 - Not modifiable by function (in general)
- Type: scalar or composite (possibly unconstrained)
- Default value possible
- Examples

Pure Functions

HWMod
WS25

Subprograms

Functions

Overview

Pure

Impure

Recommendations

Procedures

Overloading

Packages

- **Always** return same value when passed same parameters (determinism)
⇒ Can only access its parameters and **constants** from outer scope
- Only computes value, no side effects
- Examples
 - Resolution functions (e.g., IEEE-1164's `resolved`) 

- Arithmetic and logic operators

```
1 function "+" (a, b: integer) return integer is
2 begin
3   return a+b;
4 end function;
```

Impure Functions

HWMod
WS25

Subprograms

Functions

Overview

Pure

Impure

Recommendations

Procedures

Overloading

Packages

- Might return different value for same parameters (nondeterminism)
- Can access signals and variables of outer scope
- Side effects possible $\Rightarrow$ function can be stateful
- Examples

```
1 impure function now return delay_length;
```

IEEE SA
OPEN

Recommendations

HWMod
WS25

Subprograms

Functions

Overview

Pure

Impure

Recommendations

Procedures

Overloading

Packages

- Use `pure` functions whenever possible
 - Easier to understand and debug
- Use `impure` functions for
 - nondeterministic behavior
 - stateful functions
 - file I/O
 - protected types
- Especially for `impure` functions with side effects: **use sensible names!**
- **Caveat:** File parameters can introduce non-determinism and side effects into pure-functions
 - ⇒ Recommendation: Do not use file parameters for pure functions

- Primarily to encapsulate sequential pieces of code (no return value)
- Do not return a value
 - However, `return` statements without value possible for termination
- Can contain wait statements
- Work via side effects
 - Access and modify outer scope
- Simplified declaration syntax

```
1 procedure identifier[(parameter_list)] is
2   [declarative_part]
3 begin
4   [statement part]  -- procedure body
5 end procedure;
```

- Simplified declaration syntax
- Similar to `entity` port declaration
 - Procedures can drive `out` and `inout` parameters
- Default parameter mode and class are `in` and `constant`

Example - Procedure

HWMod
WS25

Subprograms

Functions

Procedures

Overview

Parameters

Example

Overloading

Packages

Apply pulse to probe, wait and check if alive is set

■ probe, PULSE_WIDTH, alive, alive_cnt from outer scope

```
1 procedure probe_alive(wait_time : time) is
2 begin
3     probe <= '1';
4     wait for PULSE_WIDTH;
5     probe <= '0';
6     wait for wait_time;
7     if alive = '0' then
8         report "Module not alive";
9     else
10        alive_cnt := alive_cnt + 1;
11    end if;
12 end procedure;
```

- VHDL supports subprogram *overloading*
- Overloaded subprograms must differ in one of the following
 - Number of parameters
 - Sequence of parameter types (if any)
 - The result base type (for functions)
- Examples

```
1 function add (a, b: integer) return integer
2 function add (a: signed; b: integer) return integer
3 function add (a: integer; b: signed) return integer
4 function add (a: signed; b: integer) return signed
```

- Subprograms can be declared in the declaration sections of
 - entities and architectures
 - processes and subprograms
 - To facilitate reusability also possible in packages
 - The *package declaration* contains declarations (e.g., constants, types, components, subprograms etc.)
 - The *package body* contains the subprogram bodies
- ⇒ Similar to C function prototype and definition

Example - Package with Body

HWMod
WS25

Subprograms

Functions

Procedures

Overloading

Packages

Example

```
1 package math_pkg is
2     constant WIDTH, HEIGHT : natural := 100;
3     function max3(value1, value2, value3 : integer) return integer;
4 end package;
5
6 package body math_pkg is
7     -- package-local auxiliary function; must be declared before use in max3
8     function max(value1, value2 : integer) return integer is
9     begin
10         if value1 > value2 then
11             return value1;
12         end if;
13         return value2;
14     end function;
15
16     function max3(value1, value2, value3 : integer) return integer is
17     begin
18         return max(max(value1, value2), value3);
19     end function;
20 end package body;
```

Lecture Complete!