

Hardware Modeling [VU] (191.011)

– WS25 –

VHDL Type System

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

- Strongly-typed language
 - Static type checks at compile time
 - Explicit type conversions
- Type declaration syntax

```
type TYPE_NAME is [...];
```
- Possible type declaration locations
 - Packages
 - Declarative parts of entities, architectures, processes, blocks, functions, procedures

- Built-in types

- Defined in `standard` package IEEE SA
OPEN

- Some Examples

- `boolean` IEEE SA
OPEN

- `integer` IEEE SA
OPEN

- `real` IEEE SA
OPEN

- `time` IEEE SA
OPEN

- Five classes of types

- scalar
- composite
- file
- access
- protected

- Keep synthesizability in mind!

■ Declaration Syntax

```
type TYPE_NAME is range_constraint;  
range_constraint ::= range expr to expr  
                  | range expr downto expr  
                  | range range_attribute_name
```

- Left/right expressions must be integers
- Ranges are either ascending or descending
 - Ascending (to): $\text{left} \leq \text{right}$
 - Descending (downto): $\text{left} \geq \text{right}$

Integer Types (cont'd)

HWMod
WS25

Types

Scalar Types

Integers

Floats

Enums

Physical Types

Attributes

Subtypes

■ Examples

- 8 bit unsigned integer

```
type uint8_t is range 0 to 255;
```

- 16 bit signed integer

```
type int16_t is range -2**15 to 2**15-1;
```

- Integer that can only hold values between 42 and 13

```
type descending_int_t is range 42 downto 13;
```

- Only built-in integer type: `integer` 

- Ascending range

- Range boundaries are implementation dependent

- VHDL-2008: at least $-2^{31} - 1$ to $2^{31} - 1$ (i.e., 32 bit)
- VHDL-2019: at least -2^{63} to $2^{63} - 1$ (i.e., 64 bit)

Range Constraints

HWMod
WS25

Types

Scalar Types

Integers

Floats

Enums

Physical Types

Attributes

Subtypes

■ Range constraints

- are checked statically (i.e., at compile time)
- are checked dynamically during simulation (i.e., runtime)
- not checked in hardware

■ Example: static range errors

```
1      constant A : uint8_t := -1; -- range error: -1 < 0
2      variable b : uint8_t := 16*16; -- range error: 256 > 255
```

■ Example: dynamic range error

```
1      process
2          variable x : uint8_t := 254;
3      begin
4          x := x + 1;
5          report "everything fine!";
6          x := x + 1;
7          report "not fine" -- never executed because of the range error
8      end process;
```

■ Declaration Syntax

```
type TYPE_NAME is range_constraint;
```

■ Example

```
type my_fp_t is range 0.0 to 1.0;
```

- Only built-in floating-point type: `real` IEEE 54 OPEN
 - range is implementation dependent
 - most likely a 64 bit (i.e., double precision) floating-point
- Range checks as with integers
- Cannot be used in synthesizable code!

■ Declaration Syntax

```
type TYPE_NAME is (enum_literal {, enum_literal});
```

■ Examples

```
■ type fsm_state_t is (INIT, WAIT_FOR_DATA, TRANSMIT);
```

```
■ type color_t is (RED, GREEN, BLUE);
```

```
■ type my_char_t is ('A', '1', '*');
```

■ Predefined enumeration types (standard package)

```
■ type boolean is (false, true); IEEE SA  
OPEN
```

```
■ type direction is (ASCENDING, DESCENDING); IEEE SA  
OPEN
```

```
■ type bit is ('0', '1'); IEEE SA  
OPEN
```

```
■ type character is (NUL, SOH, [...], 'A', 'B', [...]); IEEE SA  
OPEN
```


■ Declaration Syntax

```
type TYPE_NAME is range_constraint units
    primary_unit
    {secondary_unit}
end units;
```

■ Example

```
type distance_t is range 0 to 1_000_000_000 units
    nm; -- primary unit
    um = 1000 nm; mm = 1000 um; -- secondary units
    cm = 10 mm; m = 1000 mm; -- secondary units
end units;
```

■ Predefined physical type (standard Package): `time`

- VHDL Attributes
 - represent a powerful language feature
 - are comparable to attributes in C# or C++
 - come in the form of user- and predefined attributes
- Type attributes allow to
 - obtain meta-information about a type
 - perform operations with values of a type
- Syntax
 - Accessing attribute a of type T : $T' a$
Example: `integer' low`
 - Invoking attribute function f of type T with argument x : $T' f(x)$
Example: `integer' image(x)`

Example: low, high, left, right, ascending

HWMod
WS25

Types

Scalar Types

Attributes

Example I

Example II

Subtypes

Example code

```
1  process
2    type int_t is [...];
3  begin
4    report "low/high: " &
5      to_string(int_t'low) & "/" &
6      to_string(int_t'high);
7    report "left/right: " &
8      to_string(int_t'left) & "/" &
9      to_string(int_t'right);
10   report "asc: " & to_string(int_t'ascending);
11   wait;
12 end process;
```

Output

```
■ range 3 downto -4
[...]: low/high: -4/3
[...]: left/right: 3/-4
[...]: asc: false

■ range -4 to 3
[...]: low/high: -4/3
[...]: left/right: -4/3
[...]: asc: true
```

Note

For ascending types: $T' \text{ low} = T' \text{ left}$ and $T' \text{ high} = T' \text{ right}$

For descending types: $T' \text{ low} = T' \text{ right}$ and $T' \text{ high} = T' \text{ left}$

Example: image, value

HWMod
WS25

Example code

```
1 process
2   variable str : string(1 to 2) := "42";
3   variable val : integer;
4 begin
5   report str;
6   val := integer'value(str); -- convert string to integer
7   report integer'image(val); -- convert integer to string and print it
8   wait;
9 end process;
```

Note

For all scalar types T and values of that type V : $V = T' \text{ value}(T' \text{ image}(V))$

Types

Scalar Types

Attributes

Example I

Example II

Subtypes

Subtypes

HWMod
WS25

Types
Scalar Types
Attributes
Subtypes

■ Subtypes

- can be used to further constrain an existing type (base type)
- inherit operators defined on the base type
- can associate a resolution function with the type

■ Declaration syntax

```
subtype SUBTYPE_NAME is  
    [resolution_indication] TYPE_NAME range_constraint;
```

■ Predefined Subtypes (standard package)

- `subtype delay_length is time range 0 fs to time'high;` IEEE SA OPEN
- `subtype natural is integer range 0 to integer'high;` IEEE SA OPEN
- `subtype positive is integer range 1 to integer'high;` IEEE SA OPEN

Lecture Complete!