Hardware Modeling [VU] (191.011) – WS25 –

Vectors and Bit String Literals (IEEE 1164)

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

Modified: 2025-12-16, 18:01 (f8a58e9)

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

$\Rightarrow$ Nine valued logic and resolution of conflicts

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

$\Rightarrow$ Nine valued logic and resolution of conflicts

- Standardized in IEEE 1164 and implemented in `std_logic_1164` package

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

⇒ Nine valued logic and resolution of conflicts

- Standardized in IEEE 1164 and implemented in `std_logic_1164` package
  - *Unresolved*: `std_ulogic` IEEE SA OPEN
  - *Resolved*: `std_logic` IEEE SA OPEN

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

⇒ Nine valued logic and resolution of conflicts

- Standardized in IEEE 1164 and implemented in `std_logic_1164` package
  - *Unresolved*: `std_ulogic` IEEE SA OPEN Single driver per signal
  - *Resolved*: `std_logic` IEEE SA OPEN Multiple drivers per signal

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

⇒ Nine valued logic and resolution of conflicts

- Standardized in IEEE 1164 and implemented in `std_logic_1164` package
  - *Unresolved*: `std_ulogic` IEEE SA OPEN Single driver per signal
  - *Resolved*: `std_logic` IEEE SA OPEN Multiple drivers per signal
- What about...

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

⇒ Nine valued logic and resolution of conflicts

- Standardized in IEEE 1164 and implemented in `std_logic_1164` package
  - *Unresolved*: `std_ulogic` Single driver per signal
  - *Resolved*: `std_logic` Multiple drivers per signal
- What about...
  - operations on these types?

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

⇒ Nine valued logic and resolution of conflicts

- Standardized in IEEE 1164 and implemented in `std_logic_1164` package
  - *Unresolved*: `std_ulogic` Single driver per signal
  - *Resolved*: `std_logic` Multiple drivers per signal
- What about...
  - operations on these types?
  - multi-bit signals?

Recall: 9-valued Logic and Resolution Functions

- Boolean logic cannot express different driver strengths

⇒ Nine valued logic and resolution of conflicts

- Standardized in IEEE 1164 and implemented in `std_logic_1164` package
  - *Unresolved*: `std_ulogic` IEEE SA OPEN Single driver per signal
  - *Resolved*: `std_logic` IEEE SA OPEN Multiple drivers per signal
- What about...
  - operations on these types?
  - multi-bit signals?
- And when do you use which?

Logical Operators

Logical Operators

- Common logical operators are defined for `std_ulogic`

Logical Operators

- Common logical operators are defined for `std_ulogic`
  - NOT, AND, OR, XOR, NAND, NOR, XNOR

Logical Operators

- Common logical operators are defined for `std_ulogic`
  - NOT, AND, OR, XOR, NAND, NOR, XNOR
- Must respect different semantics of different values
  - Example: `'L' and '1'` must yield '0'

Logical Operators

- Common logical operators are defined for `std_ulogic`
  - NOT, AND, OR, XOR, NAND, NOR, XNOR
- Must respect different semantics of different values
  - Example: `'L' and '1'` must yield '0'
- Implemented by simple lookup tables

Logical Operators

- Common logical operators are defined for `std_ulogic`
  - NOT, AND, OR, XOR, NAND, NOR, XNOR
- Must respect different semantics of different values
  - Example: `'L' and '1'` must yield '0'
- Implemented by simple lookup tables

Logical Operators

- Common logical operators are defined for `std_ulogic`
  - NOT, AND, OR, XOR, NAND, NOR, XNOR
- Must respect different semantics of different values
  - Example: `'L' and '1'` must yield '0'
- Implemented by simple lookup tables

  Example: `and` operator

```
constant and_table : stdlogic_table := (
-- U    X    0    1    Z    W    L    H    -
---------------------------------------------
 ('U','U','0','U','U','U','0','U','U'), -- U
 ('U','X','0','X','X','X','0','X','X'), -- X
 ('0','0','0','0','0','0','0','0','0'), -- 0
 ('U','X','0','1','X','X','0','1','X'), -- 1
 ('U','X','0','X','X','X','0','X','X'), -- Z
 ('U','X','0','X','X','X','0','X','X'), -- W
 ('0','0','0','0','0','0','0','0','0'), -- L
 ('U','X','0','1','X','X','0','1','X'), -- H
 ('U','X','0','X','X','X','0','X','X')  -- -
);
```

Logical Operators

- Common logical operators are defined for `std_ulogic`
  - NOT, AND, OR, XOR, NAND, NOR, XNOR
- Must respect different semantics of different values
  - Example: `'L' and '1'` must yield '0'
- Implemented by simple lookup tables

  Example: `and` operator

```vhdl
constant and_table : stdlogic_table := (
-- U    X    0    1    Z    W    L    H    -
------------------------------------------------
 ('U','U','0','U','U','U','0','U','U'), -- U
 ('U','X','0','X','X','X','0','X','X'), -- X
 ('0','0','0','0','0','0','0','0','0'), -- 0
 ('U','X','0','1','X','X','0','1','X'), -- 1
 ('U','X','0','X','X','X','0','X','X'), -- Z
 ('U','X','0','X','X','X','0','X','X'), -- W
 ('0','0','0','0','0','0','0','0','0'), -- L
 ('U','X','0','1','X','X','0','1','X'), -- H
 ('U','X','0','X','X','X','0','X','X')  -- -
);
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems

```
bit_string_literal::=[integer]base_specifier"[bit_value]"
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems

```
bit_string_literal::=[integer]base_specifier"[bit_value]"
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems

```
bit_string_literal::=[integer]base_specifier"[bit_value]"
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems

    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given ⇒ "signed" specifiers: **s**b, **s**x, **s**o

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    ```
    bit_string_literal::=[integer]base_specifier"[bit_value]"
    ```
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given ⇒ “signed” specifiers: **s**b, **s**x, **s**o

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given ⇒ “signed” specifiers: **s**b, **s**x, **s**o

```
variable u : std_ulogic_vector(7 downto 0) :=  8b"11_1111"; -- 00111111
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given ⇒ “signed” specifiers: **s**b, **s**x, **s**o

```
variable u : std_ulogic_vector(7 downto 0) :=  8b"11_1111"; -- 00111111
variable s : std_ulogic_vector(7 downto 0) := 8sb"11_1111"; -- 11111111
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given ⇒ “signed” specifiers: **s**b, **s**x, **s**o

```
variable u : std_ulogic_vector(7 downto 0) :=  8b"11_1111"; -- 00111111
variable s : std_ulogic_vector(7 downto 0) := 8sb"11_1111"; -- 11111111
variable e : std_ulogic_vector(7 downto 0) :=   b"11_1111"; -- error
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given ⇒ "signed" specifiers: **s**b, **s**x, **s**o

```
variable u : std_ulogic_vector(7 downto 0) :=  8b"11_1111"; -- 00111111
variable s : std_ulogic_vector(7 downto 0) := 8sb"11_1111"; -- 11111111
variable e : std_ulogic_vector(7 downto 0) :=   b"11_1111"; -- error
variable h : std_ulogic_vector(7 downto 0) :=   x"3W";      -- 0011WWWW
```

std_[u]logic_vector Types

- The standard also defines arrays of the new types, called vectors

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic_vector is (resolved) std_ulogic_vector;
```

- Vectors can be assigned *bit string literals*
  - Concise encoding of strings in different numeral systems
    `bit_string_literal::=[integer]base_specifier"[bit_value]"`
  - Base specifiers: **b**inary, he**x**adecimal, **o**ctal, **d**ecimal
  - Optional integer length can be given ⇒ "signed" specifiers: **s**b, **s**x, **s**o

```
variable u : std_ulogic_vector(7 downto 0) :=  8b"11_1111"; -- 00111111
variable s : std_ulogic_vector(7 downto 0) := 8sb"11_1111"; -- 11111111
variable e : std_ulogic_vector(7 downto 0) :=   b"11_1111"; -- error
variable h : std_ulogic_vector(7 downto 0) :=   x"3W";      -- 0011WWWW
```

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`
  - Length of both arguments and the return value are equal

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`
  - Length of both arguments and the return value are equal
  - Example: `"UX0011" and "01X0LW" = "0X000X"`

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`
  - Length of both arguments and the return value are equal
  - Example: `"UX0011" and "01X0LW" = "0X000X"`
- Shift operators: `sll, srl`

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`
  - Length of both arguments and the return value are equal
  - Example: `"UX0011" and "01X0LW" = "0X000X"`
- Shift operators: `sll, srl`
  - Examples: `"1101" sll 2 = "0100"`, `"1101" srl 2 = "0011"`

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`
  - Length of both arguments and the return value are equal
  - Example: `"UX0011" and "01X0LW" = "0X000X"`
- Shift operators: `sll, srl`
  - Examples: `"1101" sll 2 = "0100"`, `"1101" srl 2 = "0011"`
- Rotate operators: `rol, ror`

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`
  - Length of both arguments and the return value are equal
  - Example: `"UX0011" and "01X0LW" = "0X000X"`
- Shift operators: `sll, srl`
  - Examples: `"1101" sll 2 = "0100"`, `"1101" srl 2 = "0011"`
- Rotate operators: `rol, ror`
  - Example: `"1101" rol 2 = "0111"`, `"1101" ror 2 = "1110"`

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`
  - Length of both arguments and the return value are equal
  - Example: `"UX0011" and "01X0LW" = "0X000X"`
- Shift operators: `sll, srl`
  - Examples: `"1101" sll 2 = "0100"`, `"1101" srl 2 = "0011"`
- Rotate operators: `rol, ror`
  - Example: `"1101" rol 2 = "0111"`, `"1101" ror 2 = "1110"`
- Conversion functions
  - From and to `bit_vector`
  - To differently encoded strings: `to_bstring, to_ostring, to_hstring`

std_[u]logic_vector Logical Operators

- Common logical operators are defined in a bit-wise manner for `std_ulogic_vector` / `std_logic_vector`
  - Length of both arguments and the return value are equal
  - Example: `"UX0011" and "01X0LW" = "0X000X"`
- Shift operators: `sll, srl`
  - Examples: `"1101" sll 2 = "0100"`, `"1101" srl 2 = "0011"`
- Rotate operators: `rol, ror`
  - Example: `"1101" rol 2 = "0111"`, `"1101" ror 2 = "1110"`
- Conversion functions
  - From and to `bit_vector`
  - To differently encoded strings: `to_bstring, to_ostring, to_hstring`

std_ulogic vs. std_logic

- When should which type be used?

std_ulogic vs. std_logic

- When should which type be used?
- Use unresolved types whenever modelled circuit has a single driver

std_ulogic vs. std_logic

- When should which type be used?
- Use unresolved types whenever modelled circuit has a single driver
  - Allow tools to detect undesired multiple drivers

std_ulogic vs. std_logic

- When should which type be used?
- Use unresolved types whenever modelled circuit has a single driver
  - Allow tools to detect undesired multiple drivers
- Most tool generated code and resources use the resolved types

std_ulogic vs. std_logic

- When should which type be used?
- Use unresolved types whenever modelled circuit has a single driver
  - Allow tools to detect undesired multiple drivers
- Most tool generated code and resources use the resolved types

std_ulogic vs. std_logic

- When should which type be used?
- Use unresolved types whenever modelled circuit has a single driver
  - Allow tools to detect undesired multiple drivers
- Most tool generated code and resources use the resolved types
  - VHDL 1993 required unpleasant type casts

std_ulogic vs. std_logic

- When should which type be used?
- Use unresolved types whenever modelled circuit has a single driver
  - Allow tools to detect undesired multiple drivers
- Most tool generated code and resources use the resolved types
  - VHDL 1993 required unpleasant type casts
  - Just using `std_logic_vector` hides undesired multiple drivers

std_ulogic vs. std_logic

- When should which type be used?
- Use unresolved types whenever modelled circuit has a single driver
  - Allow tools to detect undesired multiple drivers
- Most tool generated code and resources use the resolved types
  - VHDL 1993 required unpleasant type casts
  - Just using `std_logic_vector` hides undesired multiple drivers
  - `std_logic_vector` subtype of `std_ulogic_vector` since VHDL 2008

Lecture Complete!