# Hardware Modeling [VU] (191.011)
# – WS25 –
## 9-Valued Logic and Resolution (IEEE 1164)

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

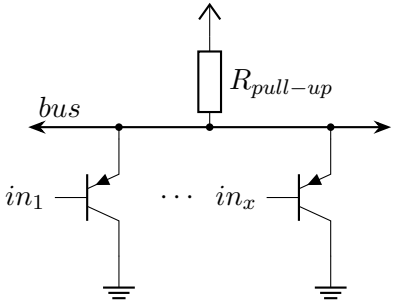
WS 2025/26

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

# Motivation | Wired-AND



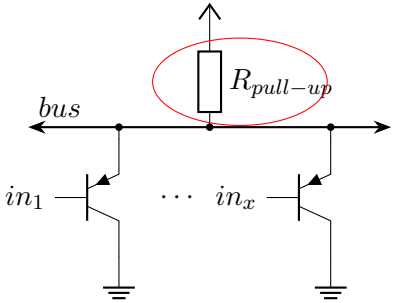
- Example: Wired-AND circuit



# Motivation | Wired-AND



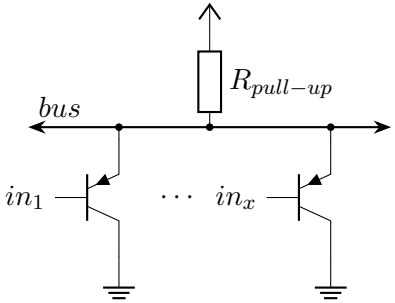
- Example: Wired-AND circuit
- A pull-up resistor pulls $bus$ to HIGH when none of the transistors is active



# Motivation | Wired-AND



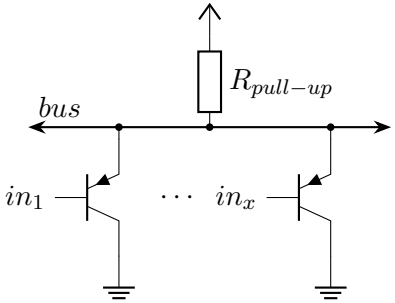
- Example: Wired-AND circuit
- A pull-up resistor pulls $bus$ to HIGH when none of the transistors is active



# Motivation | Wired-AND



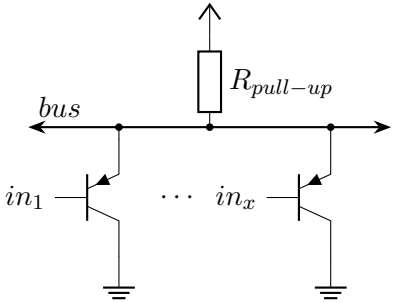
- Example: Wired-AND circuit
- A pull-up resistor pulls $bus$ to HIGH when none of the transistors is active
- Setting one of the inputs $in_1, \ldots, in_x$ to LOW overrides the pull-up



# Motivation | Wired-AND



- Example: Wired-AND circuit
- A pull-up resistor pulls $bus$ to HIGH when none of the transistors is active
- Setting one of the inputs $in_1, \ldots, in_x$ to LOW overrides the pull-up
  - How can we model this overriding behavior?

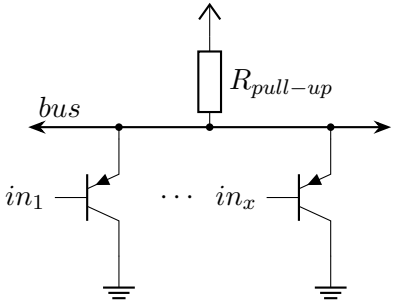


# Motivation | Wired-AND



- Example: Wired-AND circuit
- A pull-up resistor pulls $bus$ to HIGH when none of the transistors is active
- Setting one of the inputs $in_1, \ldots, in_x$ to LOW overrides the pull-up
  - How can we model this overriding behavior?

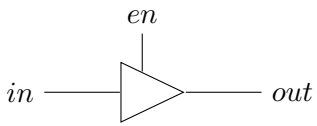
  $\Rightarrow$ We cannot model this with Boolean values alone!



# Motivation | Tri-State Buffer

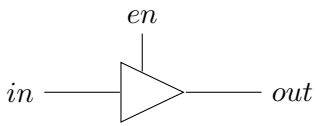


- Another example: *tri-state* buffer



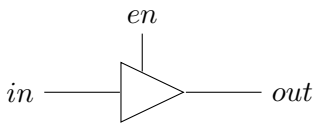
# Motivation | Tri-State Buffer

- Another example: *tri-state* buffer
- $en = 1 \Rightarrow$ buffer is *transparent*: $in$ propagated to $out$



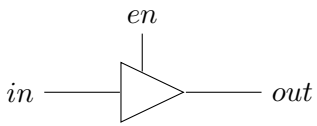
# Motivation | Tri-State Buffer

- Another example: *tri-state* buffer
- $en = 1 \Rightarrow$ buffer is *transparent*: $in$ propagated to $out$
- $en = 0 \Rightarrow$ buffer is *disabled*: high impedance at $out \Rightarrow$ overriding by active driver possible



# Motivation | Tri-State Buffer

- Another example: *tri-state* buffer
- $en = 1 \Rightarrow$ buffer is *transparent*: $in$ propagated to $out$
- $en = 0 \Rightarrow$ buffer is *disabled*: high impedance at $out \Rightarrow$ overriding by active driver possible
  - $\Rightarrow$ We cannot model this with Boolean values alone!



# The IEEE std_logic_1164 package





- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis

# The IEEE std_logic_1164 package





- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis

  $\Rightarrow$ IEEE 1164 standard for VHDL

# The IEEE std_logic_1164 package





- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis
  - $\Rightarrow$ IEEE 1164 standard for VHDL
- Implemented in the `std_logic_1164` package

# The IEEE std_logic_1164 package





- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis
  - $\Rightarrow$ IEEE 1164 standard for VHDL
- Implemented in the `std_logic_1164` package
- Must be imported via

# The IEEE std_logic_1164 package





- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis

  $\Rightarrow$ IEEE 1164 standard for VHDL
- Implemented in the `std_logic_1164` package
- Must be imported via

```
library ieee;
use ieee.std_logic_1164.all;
```

# The IEEE std_logic_1164 package



- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis

  $\Rightarrow$ IEEE 1164 standard for VHDL
- Implemented in the `std_logic_1164` package IEEE SA OPEN
- Must be imported via

```
library ieee;
use ieee.std_logic_1164.all;
```

$\Rightarrow$ Get access to two 9-valued logic (and related) types:
  - *Unresolved*: `std_ulogic` IEEE SA OPEN
  - *Resolved*: `std_logic` IEEE SA OPEN

# The IEEE std_logic_1164 package



- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis

  $\Rightarrow$ IEEE 1164 standard for VHDL
- Implemented in the `std_logic_1164` package IEEE SA OPEN
- Must be imported via

```
library ieee;
use ieee.std_logic_1164.all;
```

$\Rightarrow$ Get access to two 9-valued logic (and related) types:
  - *Unresolved*: `std_ulogic` IEEE SA OPEN
  - *Resolved*: `std_logic` IEEE SA OPEN

# IEEE 1164 Value System





The standard defines nine values and example use cases

| Value | Name | Example Use Case |
|---|---|---|

# IEEE 1164 Value System



The standard defines nine values and example use cases

| Value | Name | Example Use Case |
|---|---|---|
| `'U'` | Uninitialized State | Used as default value |

# IEEE 1164 Value System



The standard defines nine values and example use cases

| Value | Name | Example Use Case |
|---|---|---|
| `'U'` | Uninitialized State | Used as default value |
| `'X'` | Strong Unknown | Bus contention, error condition |

# IEEE 1164 Value System



The standard defines nine values and example use cases

| Value | Name | Example Use Case |
|---|---|---|
| `'U'` | Uninitialized State | Used as default value |
| `'X'` | Strong Unknown | Bus contention, error condition |
| `'0'` | Strong LOW | Active driver to LOW |
| `'1'` | Strong HIGH | Active driver to HIGH |

# IEEE 1164 Value System



The standard defines nine values and example use cases

| Value | Name | Example Use Case |
|---|---|---|
| `'U'` | Uninitialized State | Used as default value |
| `'X'` | Strong Unknown | Bus contention, error condition |
| `'0'` | Strong LOW | Active driver to LOW |
| `'1'` | Strong HIGH | Active driver to HIGH |
| `'Z'` | High Impedance | Tri-state buffer output |

# IEEE 1164 Value System



The standard defines nine values and example use cases

| Value | Name | Example Use Case |
|---|---|---|
| `'U'` | Uninitialized State | Used as default value |
| `'X'` | Strong Unknown | Bus contention, error condition |
| `'0'` | Strong LOW | Active driver to LOW |
| `'1'` | Strong HIGH | Active driver to HIGH |
| `'Z'` | High Impedance | Tri-state buffer output |
| `'W'` | Weak Unknown | Bus terminator |
| `'L'` | Weak LOW | Pull down resistor |
| `'H'` | Weak HIGH | Pull up resistor |

# IEEE 1164 Value System



The standard defines nine values and example use cases

| Value | Name | Example Use Case |
|---|---|---|
| `'U'` | Uninitialized State | Used as default value |
| `'X'` | Strong Unknown | Bus contention, error condition |
| `'0'` | Strong LOW | Active driver to LOW |
| `'1'` | Strong HIGH | Active driver to HIGH |
| `'Z'` | High Impedance | Tri-state buffer output |
| `'W'` | Weak Unknown | Bus terminator |
| `'L'` | Weak LOW | Pull down resistor |
| `'H'` | Weak HIGH | Pull up resistor |
| `'-'` | Don't care | Useful for synthesis and modeling |

# IEEE 1164 Value System



The standard defines nine values and example use cases

| Value | Name | Example Use Case |
|---|---|---|
| `'U'` | Uninitialized State | Used as default value |
| `'X'` | Strong Unknown | Bus contention, error condition |
| `'0'` | Strong LOW | Active driver to LOW |
| `'1'` | Strong HIGH | Active driver to HIGH |
| `'Z'` | High Impedance | Tri-state buffer output |
| `'W'` | Weak Unknown | Bus terminator |
| `'L'` | Weak LOW | Pull down resistor |
| `'H'` | Weak HIGH | Pull up resistor |
| `'-'` | Don't care | Useful for synthesis and modeling |

# IEEE 1164 std_ulogic Type



- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```


1

# IEEE 1164 std_ulogic Type



- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

1

# IEEE 1164 std_ulogic Type



- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```


1

# IEEE 1164 std_ulogic Type



- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

- *Unresolved*: Only supports signals with **single** driver

1

# IEEE 1164 std_ulogic Type



- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

- *Unresolved*: Only supports signals with **single** driver
- Multiple drivers are detected and reported (during elaboration)

1

# IEEE 1164 std_ulogic Type

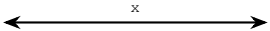


- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

- *Unresolved*: Only supports signals with **single** driver
- Multiple drivers are detected and reported (during elaboration)

```
1 signal x : std_ulogic;
```




1

# IEEE 1164 std_ulogic Type

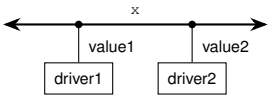


- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

- *Unresolved*: Only supports signals with **single** driver
- Multiple drivers are detected and reported (during elaboration)

```
1  signal x : std_ulogic;
2  [...]
3  driver1 : process(all) is
4    [...]
5    x <= value1;
6  end process;
7  driver2 : process(all) is
8    [...]
9    x <= value2;
10 end process;
```



# IEEE 1164 std_ulogic Type

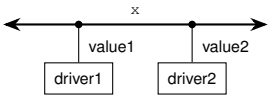


- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

- *Unresolved*: Only supports signals with **single** driver
- Multiple drivers are detected and reported (during elaboration)

```
1  signal x : std_ulogic;
2  [...]
3  driver1 : process(all) is
4    [...]
5    x <= value1;
6  end process;
7  driver2 : process(all) is
8    [...]
9    x <= value2;
10 end process;
```



# IEEE 1164 std_ulogic Type

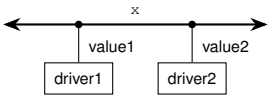


- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

- *Unresolved*: Only supports signals with **single** driver
- Multiple drivers are detected and reported (during elaboration)

```
1  signal x : std_ulogic;
2  [...]
3  driver1 : process(all) is
4    [...]
5    x <= value1;
6  end process;
7  driver2 : process(all) is
8    [...]
9    x <= value2;
10 end process;
```




```
[...]: error: too many drivers for signal "x"
```

# IEEE 1164 std_ulogic Type

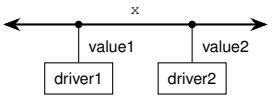


- Simple enumeration type with nine values

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

- *Unresolved*: Only supports signals with **single** driver
- Multiple drivers are detected and reported (during elaboration)

```
1  signal x : std_ulogic;
2  [...]
3  driver1 : process(all) is
4    [...]
5    x <= value1;
6  end process;
7  driver2 : process(all) is
8    [...]
9    x <= value2;
10 end process;
```




```
[...]: error: too many drivers for signal "x"
```

# IEEE 1164 std_logic Type



- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic;
```

# IEEE 1164 std_logic Type



- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic;
```

- `std_logic` has the same nine values as `std_ulogic`

# IEEE 1164 std_logic Type



- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic;
```

- `std_logic` has the same nine values as `std_ulogic`
- Allows **multiple** drivers (e.g., wired-AND, tri-state bus)

# IEEE 1164 std_logic Type



- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic;
```

- `std_logic` has the same nine values as `std_ulogic`
- Allows **multiple** drivers (e.g., wired-AND, tri-state bus)
- Changing the type of signal `x` in the previous example does not result in the observed error

# IEEE 1164 std_logic Type



- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic;
```

- `std_logic` has the same nine values as `std_ulogic`
- Allows **multiple** drivers (e.g., wired-AND, tri-state bus)
- Changing the type of signal `x` in the previous example does not result in the observed error
  - There are still multiple drivers ⇒ What value will `x` exhibit?

# IEEE 1164 std_logic Type



- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic;
```

- `std_logic` has the same nine values as `std_ulogic`
- Allows **multiple** drivers (e.g., wired-AND, tri-state bus)
- Changing the type of signal `x` in the previous example does not result in the observed error
  - There are still multiple drivers ⇒ What value will `x` exhibit?
- Uses a *resolution function* (`resolved`)

# IEEE 1164 std_logic Type



- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic;
```

- `std_logic` has the same nine values as `std_ulogic`
- Allows **multiple** drivers (e.g., wired-AND, tri-state bus)
- Changing the type of signal `x` in the previous example does not result in the observed error
  - There are still multiple drivers $\Rightarrow$ What value will `x` exhibit?
- Uses a *resolution function* (`resolved`)

# IEEE 1164 std_logic Type



- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic;
```

- `std_logic` has the same nine values as `std_ulogic`
- Allows **multiple** drivers (e.g., wired-AND, tri-state bus)
- Changing the type of signal `x` in the previous example does not result in the observed error
  - There are still multiple drivers ⇒ What value will `x` exhibit?
- Uses a *resolution function* (`resolved`)

# Resolution Function





- Defines resolution of multiple drivers' values into single *resolved value*

# Resolution Function





- Defines resolution of multiple drivers' values into single *resolved value*
- Single parameter: Array of all values assigned to signal

# Resolution Function





- Defines resolution of multiple drivers' values into single *resolved value*
- Single parameter: Array of all values assigned to signal
- Invoked during the simulation, no real meaning for synthesis

# Resolution Function



- Defines resolution of multiple drivers' values into single *resolved value*
- Single parameter: Array of all values assigned to signal
- Invoked during the simulation, no real meaning for synthesis
- Resolution functions can be associated to subtypes or signals
  - For subtypes: All signals of this subtype are resolved

# Resolution Function





- Defines resolution of multiple drivers' values into single *resolved value*
- Single parameter: Array of all values assigned to signal
- Invoked during the simulation, no real meaning for synthesis
- Resolution functions can be associated to subtypes or signals
  - For subtypes: All signals of this subtype are resolved
  - Arrays and records of subtypes are also supported

# Resolution Function



- Defines resolution of multiple drivers' values into single *resolved value*
- Single parameter: Array of all values assigned to signal
- Invoked during the simulation, no real meaning for synthesis
- Resolution functions can be associated to subtypes or signals
  - For subtypes: All signals of this subtype are resolved
  - Arrays and records of subtypes are also supported
  - For signals: Only respective signal resolved
  - Example: `signal x : resolved std_ulogic;`

# Resolution Function



- Defines resolution of multiple drivers' values into single *resolved value*
- Single parameter: Array of all values assigned to signal
- Invoked during the simulation, no real meaning for synthesis
- Resolution functions can be associated to subtypes or signals
  - For subtypes: All signals of this subtype are resolved
  - Arrays and records of subtypes are also supported
  - For signals: Only respective signal resolved
  - Example: `signal x : resolved std_ulogic;`

# The std_ulogic Resolution Function



- Resolves multiple `std_ulogic` values into a single one

# The std_ulogic Resolution Function



- Resolves multiple `std_ulogic` values into a single one

```
1  function resolved (s : std_ulogic_vector) return std_ulogic is
2
3  begin
4
5
6
7
8
9
10
11
12 end function;
```

# The std_ulogic Resolution Function



- Resolves multiple `std_ulogic` values into a single one

```
1  function resolved (s : std_ulogic_vector) return std_ulogic is
2    variable result : std_ulogic := 'Z';
3  begin
4
5
6
7
8
9
10
11
12 end function;
```

# The std_ulogic Resolution Function



- Resolves multiple `std_ulogic` values into a single one

```
1  function resolved (s : std_ulogic_vector) return std_ulogic is
2    variable result : std_ulogic := 'Z';
3  begin
4    if (s'length = 1) then
5      return s(s'low);
6    else
7      for i in s'range loop
8        res := RES_TABLE(result, s(i));
9      end loop;
10   end if;
11   return result;
12 end function;
```

# The std_ulogic Resolution Function



- Resolves multiple `std_ulogic` values into a single one

```
1  function resolved (s : std_ulogic_vector) return std_ulogic is
2    variable result : std_ulogic := 'Z';
3  begin
4    if (s'length = 1) then
5      return s(s'low);
6    else
7      for i in s'range loop
8        res := RES_TABLE(result, s(i));
9      end loop;
10   end if;
11   return result;
12 end function;
```

# The std_ulogic Resolution Function



- Resolves multiple `std_ulogic` values into a single one

```
1  function resolved (s : std_ulogic_vector) return std_ulogic is
2    variable result : std_ulogic := 'Z';
3  begin
4    if (s'length = 1) then
5      return s(s'low);
6    else
7      for i in s'range loop
8        res := RES_TABLE(result, s(i));
9      end loop;
10   end if;
11   return result;
12 end function;
```

# The std_ulogic Resolution Function



- Resolves multiple `std_ulogic` values into a single one

```
1  function resolved (s : std_ulogic_vector) return std_ulogic is
2    variable result : std_ulogic := 'Z';
3  begin
4    if (s'length = 1) then
5      return s(s'low);
6    else
7      for i in s'range loop
8        res := RES_TABLE(result, s(i));
9      end loop;
10   end if;
11   return result;
12 end function;
```

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ----------------------------------------------
4    ('U','U','U','U','U','U','U','U','U'),  -- U
5    ('U','X','X','X','X','X','X','X','X'),  -- X
6    ('U','X','0','X','0','0','0','0','X'),  -- 0
7    ('U','X','X','1','1','1','1','1','X'),  -- 1
8    ('U','X','0','1','Z','W','L','H','X'),  -- Z
9    ('U','X','0','1','W','W','W','W','X'),  -- W
10   ('U','X','0','1','L','W','L','W','X'),  -- L
11   ('U','X','0','1','H','W','W','H','X'),  -- H
12   ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ------------------------------------------------
4    ('U','U','U','U','U','U','U','U','U'), -- U
5    ('U','X','X','X','X','X','X','X','X'), -- X
6    ('U','X','0','X','0','0','0','0','X'), -- 0
7    ('U','X','X','1','1','1','1','1','X'), -- 1
8    ('U','X','0','1','Z','W','L','H','X'), -- Z
9    ('U','X','0','1','W','W','W','W','X'), -- W
10   ('U','X','0','1','L','W','L','W','X'), -- L
11   ('U','X','0','1','H','W','W','H','X'), -- H
12   ('U','X','X','X','X','X','X','X','X')  -- -
13 );
```

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ----------------------------------------------
4    ('U','U','U','U','U','U','U','U','U'),  -- U
5    ('U','X','X','X','X','X','X','X','X'),  -- X
6    ('U','X','0','X','0','0','0','0','X'),  -- 0
7    ('U','X','X','1','1','1','1','1','X'),  -- 1
8    ('U','X','0','1','Z','W','L','H','X'),  -- Z
9    ('U','X','0','1','W','W','W','W','X'),  -- W
10   ('U','X','0','1','L','W','L','W','X'),  -- L
11   ('U','X','0','1','H','W','W','H','X'),  -- H
12   ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ----------------------------------------------
4    ('U','U','U','U','U','U','U','U','U'),  -- U
5    ('U','X','X','X','X','X','X','X','X'),  -- X
6    ('U','X','0','X','0','0','0','0','X'),  -- 0
7    ('U','X','X','1','1','1','1','1','X'),  -- 1
8    ('U','X','0','1','Z','W','L','H','X'),  -- Z
9    ('U','X','0','1','W','W','W','W','X'),  -- W
10   ('U','X','0','1','L','W','L','W','X'),  -- L
11   ('U','X','0','1','H','W','W','H','X'),  -- H
12   ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ----------------------------------------------
4    ('U','U','U','U','U','U','U','U','U'),  -- U
5    ('U','X','X','X','X','X','X','X','X'),  -- X
6    ('U','X','0','X','0','0','0','0','X'),  -- 0
7    ('U','X','X','1','1','1','1','1','X'),  -- 1
8    ('U','X','0','1','Z','W','L','H','X'),  -- Z
9    ('U','X','0','1','W','W','W','W','X'),  -- W
10   ('U','X','0','1','L','W','L','W','X'),  -- L
11   ('U','X','0','1','H','W','W','H','X'),  -- H
12   ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

# The std_ulogic Resolution Function (cont'd)

- The `RES_TABLE` defines how two values are resolved into one
  Example: pull-up (`'H'`), active (`'0'`) and inactive (`'Z'`) driver

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ----------------------------------------------
4   ('U','U','U','U','U','U','U','U','U'),  -- U
5   ('U','X','X','X','X','X','X','X','X'),  -- X
6   ('U','X','0','X','0','0','0','0','X'),  -- 0
7   ('U','X','X','1','1','1','1','1','X'),  -- 1
8   ('U','X','0','1','Z','W','L','H','X'),  -- Z
9   ('U','X','0','1','W','W','W','W','X'),  -- W
10  ('U','X','0','1','L','W','L','W','X'),  -- L
11  ('U','X','0','1','H','W','W','H','X'),  -- H
12  ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

`resolve("H0Z"):`

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one
  Example: pull-up (`'H'`), active (`'0'`) and inactive (`'Z'`) driver

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ---------------------------------------------
4   ('U','U','U','U','U','U','U','U','U'),  -- U
5   ('U','X','X','X','X','X','X','X','X'),  -- X
6   ('U','X','0','X','0','0','0','0','X'),  -- 0
7   ('U','X','X','1','1','1','1','1','X'),  -- 1
8   ('U','X','0','1','Z','W','L','H','X'),  -- Z
9   ('U','X','0','1','W','W','W','W','X'),  -- W
10  ('U','X','0','1','L','W','L','W','X'),  -- L
11  ('U','X','0','1','H','W','W','H','X'),  -- H
12  ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

```
resolve("H0Z"):
 1  RES_TABLE('Z', 'H') = 'H'
```

# The std_ulogic Resolution Function (cont'd)

- The `RES_TABLE` defines how two values are resolved into one
  Example: pull-up (`'H'`), active (`'0'`) and inactive (`'Z'`) driver

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ------------------------------------------------
4   ('U','U','U','U','U','U','U','U','U'),  -- U
5   ('U','X','X','X','X','X','X','X','X'),  -- X
6   ('U','X','0','X','0','0','0','0','X'),  -- 0
7   ('U','X','X','1','1','1','1','1','X'),  -- 1
8   ('U','X','0','1','Z','W','L','H','X'),  -- Z
9   ('U','X','0','1','W','W','W','W','X'),  -- W
10  ('U','X','0','1','L','W','L','W','X'),  -- L
11  ('U','X','0','1','H','W','W','H','X'),  -- H
12  ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

```
resolve("H0Z"):
  1  RES_TABLE('Z', 'H') = 'H'
  2  RES_TABLE('H', '0') = '0'
```

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one
  Example: pull-up (`'H'`), active (`'0'`) and inactive (`'Z'`) driver

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ---------------------------------------------
4   ('U','U','U','U','U','U','U','U','U'),  -- U
5   ('U','X','X','X','X','X','X','X','X'),  -- X
6   ('U','X','0','X','0','0','0','0','X'),  -- 0
7   ('U','X','X','1','1','1','1','1','X'),  -- 1
8   ('U','X','0','1','Z','W','L','H','X'),  -- Z
9   ('U','X','0','1','W','W','W','W','X'),  -- W
10  ('U','X','0','1','L','W','L','W','X'),  -- L
11  ('U','X','0','1','H','W','W','H','X'),  -- H
12  ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

```
resolve("H0Z"):
1  RES_TABLE('Z', 'H') = 'H'
2  RES_TABLE('H', '0') = '0'
3  RES_TABLE('0', 'Z') = '0'
```

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one
  Example: pull-up (`'H'`), active (`'0'`) and inactive (`'Z'`) driver

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ----------------------------------------------
4   ('U','U','U','U','U','U','U','U','U'),  -- U
5   ('U','X','X','X','X','X','X','X','X'),  -- X
6   ('U','X','0','X','0','0','0','0','X'),  -- 0
7   ('U','X','X','1','1','1','1','1','X'),  -- 1
8   ('U','X','0','1','Z','W','L','H','X'),  -- Z
9   ('U','X','0','1','W','W','W','W','X'),  -- W
10  ('U','X','0','1','L','W','L','W','X'),  -- L
11  ('U','X','0','1','H','W','W','H','X'),  -- H
12  ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

`resolve("H0Z"):`

1. `RES_TABLE('Z', 'H') = 'H'`
2. `RES_TABLE('H', '0') = '0'`
3. `RES_TABLE('0', 'Z') = '0'`

$\Rightarrow$ `resolve("H0Z") = '0'`

# The std_ulogic Resolution Function (cont'd)



- The `RES_TABLE` defines how two values are resolved into one
  Example: pull-up (`'H'`), active (`'0'`) and inactive (`'Z'`) driver

```
1  constant RES_TABLE: stdlogic_table := (
2  -- U    X    0    1    Z    W    L    H    -
3  ----------------------------------------------
4   ('U','U','U','U','U','U','U','U','U'),  -- U
5   ('U','X','X','X','X','X','X','X','X'),  -- X
6   ('U','X','0','X','0','0','0','0','X'),  -- 0
7   ('U','X','X','1','1','1','1','1','X'),  -- 1
8   ('U','X','0','1','Z','W','L','H','X'),  -- Z
9   ('U','X','0','1','W','W','W','W','X'),  -- W
10  ('U','X','0','1','L','W','L','W','X'),  -- L
11  ('U','X','0','1','H','W','W','H','X'),  -- H
12  ('U','X','X','X','X','X','X','X','X')   -- -
13 );
```

`resolve("H0Z"):`

1. `RES_TABLE('Z', 'H') = 'H'`
2. `RES_TABLE('H', '0') = '0'`
3. `RES_TABLE('0', 'Z') = '0'`

$\Rightarrow$ `resolve("H0Z") = '0'`

# Lecture Complete!