

Previously, you heard about VHDL types that closely resemble the ones you know from other programming languages. As we will discuss in this lecture though, they are not expressive enough to describe certain behavior of digital circuits. As a remedy, the IEEE defined additional types in the 1164 standard that are typically used when modeling hardware.

HWMoD  
WS25

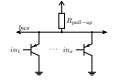
IEEE 1164

Motivation  
Standard  
VHDL Types  
Resolution

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

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

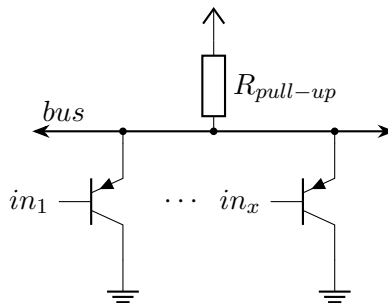
WS 2025/26

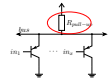


Before we start our discussion about these additional types, let us consider two examples that stress the need for them. The first one is a bus with wired AND topology, as shown on the slide. An example where such a topology can be found is the I<sup>2</sup>C protocol.

## Motivation | Wired-AND

### ■ Example: Wired-AND circuit

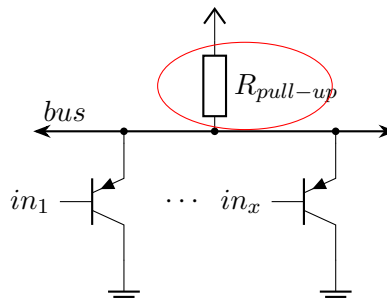


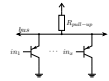


A crucial part of such a wired AND topology is the so-called pull-up resistor.

## Motivation | Wired-AND

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

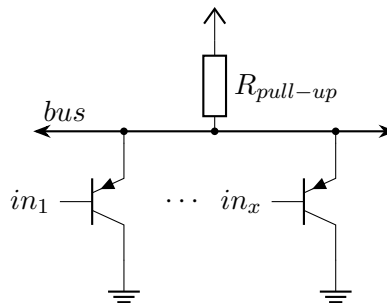


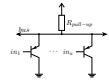


Its purpose is to ensure that the bus line exhibits a valid logical value at all times. This is achieved by connecting it to the supply voltage, which we assume to correspond to a logical HIGH.

## Motivation | Wired-AND

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

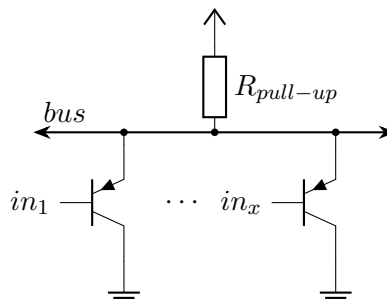


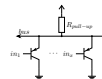


When a bus participant wants to transmit a logical LOW, it sets the input voltage of its bus driver to LOW. This will override the pull-up resistor and thus effectively pull the bus voltage to ground, which we assume to correspond to a logical LOW. This behavior, of any LOW bus driver resulting in a LOW bus value, is also the reason why this is referred to as wired AND 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  $in_1, \dots, in_x$  to LOW overrides the pull-up

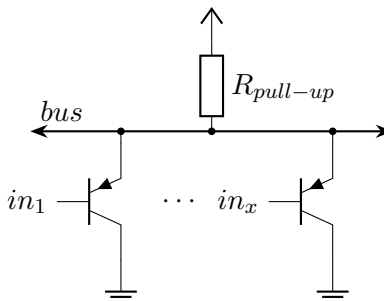




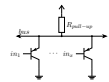
Now assume you are required to model this circuit using VHDL. *How* would you do that? Clearly we cannot use the basic `boolean` type and encode the desired circuit behavior. We would just end up describing a plain AND gate. The reason for this is that Boolean logic has no way to express the “overriding” that is so crucial for this circuit. If we had a way to also express the different *driver strengths*, we could model that the strong bus drivers override the weak pull-up resistor.

## 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  $in_1, \dots, in_x$  to LOW overrides the pull-up
  - How can we model this overriding behavior?



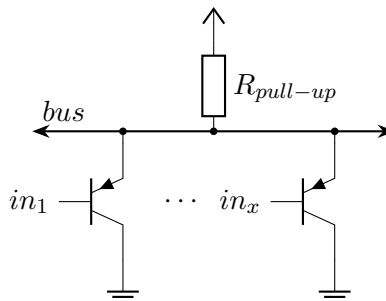
- Example: Wired-AND circuit
- A pull-up resistor pulls *bus* to HIGH when none of the transistors is active
- Setting one of the inputs  $in_1, \dots, in_x$  to LOW overrides the pull-up
- How can we model this overriding behavior?
- ... We cannot model this with Boolean values alone!



In particular, we also need values for the *weak* logical HIGH of the pull-up resistor, and the *strong* logical LOW of the bus drivers.

## 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  $in_1, \dots, in_x$  to LOW overrides the pull-up
    - How can we model this overriding behavior?
- ⇒ We cannot model this with Boolean values alone!





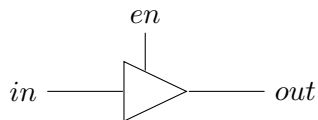
Our second example is a *tri-state* buffer. This special kind of buffer circuit, shown on the slide, has two states, where its current state is determined by the enable signal  $en$ .

## Motivation | Tri-State Buffer

HWMoD  
WS25

IEEE 1164  
Motivation  
Standard  
VHDL Types  
Resolution

- Another example: *tri-state* buffer





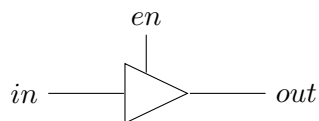
If the enable signal is HIGH, the buffer will be *transparent*. This means that it will simply propagate the logical value applied at its input to its output.

## Motivation | Tri-State Buffer

HWMoD  
WS25

IEEE 1164  
Motivation  
Standard  
VHDL Types  
Resolution

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





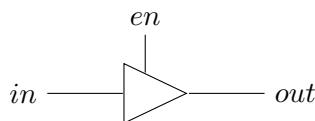
However, if the enable signal is LOW, the buffer is disabled. This is referred to as its *high-impedance* state. In this state the buffer's input is *not* forwarded to its output. Instead, the output is disconnected from the input and not actively driven. This allows other drivers to actively drive the wire connected to the buffer's output without causing a short circuit.

## Motivation | Tri-State Buffer

HWMMod  
WS25

IEEE 1164  
Motivation  
Standard  
VHDL Types  
Resolution

- 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



- 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!



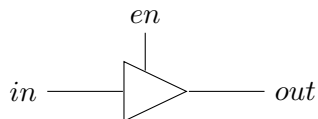
Similar to the wired AND topology, this can be used to build a bus that is shared by multiple participants. However, as before, we face the problem that Boolean logic is not expressive enough to model this additional high-impedance state.

## Motivation | Tri-State Buffer

HWMoD  
WS25

IEEE 1164  
Motivation  
Standard  
VHDL Types  
Resolution

- 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!



## 9-Valued Logic and Resolution (IEEE 1164)

### Standard

### 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

At this point, you might recall the nine-valued logic from the Digital Design lecture. Instead of the Boolean LOW and HIGH values, this special logic comes with values for different driver strengths and impedance. Furthermore, it also contains values useful for simulation and synthesis.

## The IEEE std\_logic\_1164 package

327

HWMoD  
WS25

IEEE 1164

Motivation

Standard

Package

Value System

VHDL Types

Resolution

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

## 9-Valued Logic and Resolution (IEEE 1164)

### Standard

### 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
- ⇒ IEEE 1164 standard for VHDL

Due its usefulness for describing hardware, the IEEE standardized this special logic for VHDL in 1193. The respective standard is called IEEE 1164.

## The IEEE std\_logic\_1164 package

327

HWMoD  
WS25

IEEE 1164

Motivation

Standard

Package

Value System

VHDL Types

Resolution

- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis
- ⇒ IEEE 1164 standard for VHDL

In addition to that, the IEEE also provides an open-source implementation of this standard in the form of the `std_logic_1164` package. You can have a look at this implementation by clicking the icon on the slide.

## The IEEE std\_logic\_1164 package

327

HWMoD  
WS25

IEEE 1164

Motivation

Standard

Package

Value System

VHDL Types

Resolution

- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis
  - ⇒ IEEE 1164 standard for VHDL
- Implemented in the `std_logic_1164` package [IEEE SA !\[\]\(d9a15d31e7c1d692ffdd153240283bf0\_img.jpg\)](#)

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

To make use of a package in VHDL, you have to import it first though. This is similar to including a header file in C, or a module in Java. You can do this in VHDL by first importing the library containing the package, and then the particular package itself using the `use` statement.

## The IEEE std\_logic\_1164 package

327

HWMoD  
WS25

IEEE 1164

Motivation


Standard

Package

Value System

VHDL Types

Resolution

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

# 9-Valued Logic and Resolution (IEEE 1164)

## Standard

### 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
- 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 slide shows how this looks like for the `std_logic_1164` package.

## The IEEE std\_logic\_1164 package

327

HWMoD  
WS25

IEEE 1164

Motivation


Standard

Package

Value System

VHDL Types

Resolution

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

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

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

```
library ieee;  
use ieee.std_logic_1164.all;
```
- ⇒ Get access to two 9-valued logic (and related) types:
  - *Unresolved*: `std_ulogic`
  - *Resolved*: `std_logic`

After importing this package, you essentially gain access to two new types: `std_ulogic` and `std_logic`, as well as to some related types and operators. And while the two types are named similarly, we will see shortly that they can behave quite differently. We coin these different behaviors as *unresolved* and *resolved*.

## The IEEE std\_logic\_1164 package

327

HWMod  
WS25

IEEE 1164

Motivation

Standard

Package

Value System

VHDL Types

Resolution

- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis
- ⇒ 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;
```
- ⇒ Get access to two 9-valued logic (and related) types:
  - *Unresolved*: `std_ulogic` IEEE SA OPEN
  - *Resolved*: `std_logic` IEEE SA OPEN

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

```
library ieee;  
use ieee.std_logic_1164.all;
```
- ⇒ Get access to two 9-valued logic (and related) types:
  - *Unresolved*: `std_ulogic`
  - *Resolved*: `std_logic`

Before we get into the specifics of the two types, let us briefly introduce the nine values which `std_ulogic` and `std_logic` share.

## The IEEE std\_logic\_1164 package

327

HWMoD  
WS25

IEEE 1164

Motivation

Standard

Package

Value System

VHDL Types

Resolution

- Recall from *Digital Design* lecture: 9-valued logic
  - Contains values for different driver strength / impedance
  - Also values useful for simulation and synthesis⇒ 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;
```
- ⇒ Get access to two 9-valued logic (and related) types:
  - *Unresolved*: `std_ulogic` IEEE SA OPEN
  - *Resolved*: `std_logic` IEEE SA OPEN

In addition to that, we will also briefly motivate the need for these values. You can also find use cases for the values in the standard.

## IEEE 1164 Value System

562

HWMoD  
WS25

IEEE 1164

Motivation

Standard

Package

Value System

VHDL Types

Resolution

The standard defines nine values and example use cases

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

## 9-Valued Logic and Resolution (IEEE 1164)

### Standard

### 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

Per default, all instances of the `std_ulogic` and `std_logic` types are uninitialized. This is done by setting them to the value `'U'`. This can be used by simulators to detect signals that have not been changed since the simulation started.

## IEEE 1164 Value System

562

HWMoD  
WS25

IEEE 1164

Motivation

Standard

Package

Value System

VHDL Types

Resolution

The standard defines nine values and example use cases

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

## 9-Valued Logic and Resolution (IEEE 1164)

### Standard

### 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

As you already heard before, we need different driver strengths. But what if, for example, two opposite values are driven with equal strength? In such cases of *conflicting* drivers, there might not exist a sensible logic value as result. Simulators will then use the *strong unknown* value, 'X', to express that they cannot determine a result among the other eight values.

## IEEE 1164 Value System

562

HWMoD  
WS25

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  
Motivation  
Standard  
Package  
Value System  
VHDL Types  
Resolution

# 9-Valued Logic and Resolution (IEEE 1164)

## Standard

### 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

The values '0' and '1' are the "classical" Boolean logic values. These are the values you will typically use and encounter.

## IEEE 1164 Value System

562

HWMoD  
WS25

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  
Motivation  
Standard  
Package  
Value System  
VHDL Types  
Resolution

## 9-Valued Logic and Resolution (IEEE 1164)

### Standard

### 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

For modeling a high impedance, like in the tri-state buffer example, the value 'Z' is used.

## IEEE 1164 Value System

562

HWMMod  
WS25

IEEE 1164

Motivation

Standard

Package

Value System

VHDL Types

Resolution

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

# 9-Valued Logic and Resolution (IEEE 1164)

## Standard

### 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

Furthermore, there are the weakly driven UNKOWN, LOW, and HIGH, associated with the values depicted on the slide. In combination with their strongly driven cousins, these values allow expressing the overriding behavior we encountered in the wired AND example.

## IEEE 1164 Value System

562

HWMoD  
WS25

IEEE 1164  
Motivation  
Standard  
Package  
Value System  
VHDL Types  
Resolution

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

# 9-Valued Logic and Resolution (IEEE 1164)

## Standard

### 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

Finally, the standard also defines a *don't care* value. This is referred to via the '-' symbol. The values can be used in cases where not all inputs, or states, of the modelled circuit can actually occur. Synthesis tools can then use this information to perform optimizations.

## IEEE 1164 Value System

562

HWMoD  
WS25

IEEE 1164  
Motivation  
Standard  
Package  
Value System  
VHDL Types  
Resolution

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

# 9-Valued Logic and Resolution (IEEE 1164)

## Standard

### 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

Let us now look at the types provided by the `std_logic_1164` package.

## IEEE 1164 Value System

562

HWMoD  
WS25

IEEE 1164  
Motivation  
Standard  
Package  
Value System  
VHDL Types  
Resolution

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

We begin with the `std_ulogic` type. Under the hood, this type is simply an enumeration type consisting of the previously mentioned nine values. You can find the definition of the type on the slide.

## IEEE 1164 std\_ulogic Type

### ■ Simple enumeration type with nine values

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

[IEEE 1164](#)  
[OPEN](#)

Note that all instances of this type are automatically initialized to 'U' due to the value being the first one in the list.

## IEEE 1164 std\_ulogic Type

- Simple enumeration type with nine values

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

[IEEE 1164](#)  
[OPEN](#)

Recall that we previously stated that `std_ulogic` differs from `std_logic` due to it being *unresolved*. This is also the reason for the *u* in its name. But what do we mean by it being unresolved?

## IEEE 1164 std\_ulogic Type

- Simple enumeration type with nine values

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

[IEEE 1164](#)  
[OPEN](#)

Essentially, what makes a type unresolved is that signals of this type only permit a **single** driver.

## IEEE 1164 std\_ulogic Type

- Simple enumeration type with nine values

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

[IEEE 5A](#)  
[OPEN](#)

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

Violating this, meaning a single signal is modelled to be driven by multiple drivers, will be detected by simulators and reported when elaborating the VHDL code. To illustrate this, let us consider a simple example.

## IEEE 1164 std\_ulogic Type

- Simple enumeration type with nine values

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

[IEEE 54 OPEN](#)

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

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



We start by declaring a signal `x` of type `std_ulogic`. The declaration is shown on the slide. Furthermore, next to the code we will also draw an abstract version of the circuit we model. Let us now add multiple drivers for our signal.

## IEEE 1164 std\_ulogic Type

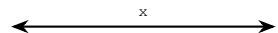
- Simple enumeration type with nine values

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

[IEEE 1164](#) [OPEN](#)

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

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



- Simple enumeration type with nine values
- 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;

```

One way to implement them would be via distinct processes that each assign a value to x somewhere in their body.

## IEEE 1164 std\_ulogic Type

HWMMod  
WS25

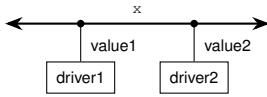
IEEE 1164  
Motivation  
Standard  
VHDL Types  
std\_ulogic  
std\_logic  
Resolution

- Simple enumeration type with nine values
- 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 5A  
OPEN

# 9-Valued Logic and Resolution (IEEE 1164)

## VHDL Types

### IEEE 1164 std\_ulogic Type

- Simple enumeration type with nine values
- Unresolved: Only supports signals with single driver
- Multiple drivers are detected and reported (during elaboration)

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



Due to both processes driving `x`, we have modelled conflicting drivers.

## IEEE 1164 std\_ulogic Type

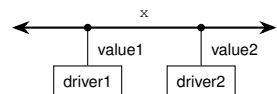
- Simple enumeration type with nine values

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

IEEE 58  
OPEN

- 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;
```



- Simple enumeration type with nine values
- 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"

If you would now try simulating this circuit, you would observe that the simulator produces an error like the one shown on the bottom of the slide.

## IEEE 1164 std\_ulogic Type

- Simple enumeration type with nine values

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

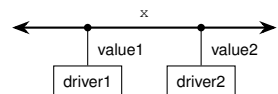
IEEE 58  
OPEN

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

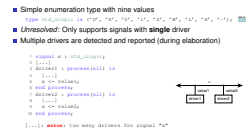
```

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"



Let us now continue with `std_logic`, the resolved pendant of the `std_ulogic` type.

## IEEE 1164 std\_ulogic Type

- Simple enumeration type with nine values

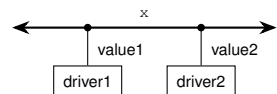
```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
```

- 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"

As before, you can find the type definition on the slide. However, note that instead of `std_logic` being defined as enumeration type, it is a particular subtype of `std_ulogic`. We will pick up on the `resolved` in the definition shortly.

## IEEE 1164 std\_logic Type

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

std\_ulogic

std\_logic

Resolution

### ■ Special subtype of `std_ulogic`

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

IEEE 1164  
9.2.1.1

But first, observe that due to it being a subtype, `std_logic` must comprise all values of `std_ulogic`. This is an important detail we will pick up again later, but let us now get to `std_logic` being resolved. Previously, we stated that `std_ulogic` is unresolved because its instances only allow a single driver.

## IEEE 1164 std\_logic Type

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

`std_ulogic`

`std_logic`

Resolution

- Special subtype of `std_ulogic`

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

IEEE 1164-2001

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

- Special subtype of `std_ulogic`  
`subtype std_logic is resolved std_ulogic;` IEEE 1164-1993
- `std_logic` has the same nine values as `std_ulogic`
- Allows multiple drivers (e.g., wired-AND, tri-state bus)

It won't be much of a surprise then when we reveal that resolved types have no such restriction, and allow multiple drivers for a single signal. This is, for example, needed for the wired AND and tri-state example circuits, since both circuits rely on drivers overriding each other.

## IEEE 1164 std\_logic Type

HWMoD  
WS25

IEEE 1164  
Motivation  
Standard  
VHDL Types  
std\_ulogic  
std\_logic  
Resolution

- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic; IEEE 1164-1993
```

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

- Special subtype of `std_ulogic`  
`subtype std_logic is resolved std_ulogic;` IEEE 1164-2001 3.10.1
- `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

For illustration, if we would change the type of the shared signal `x` in the previous example to `std_logic`, the simulator would not produce the error we saw before.

## IEEE 1164 std\_logic Type

HWMoD  
WS25

IEEE 1164  
Motivation  
Standard  
VHDL Types  
`std_ulogic`  
`std_logic`  
Resolution

- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic; IEEE 1164-2001 3.10.1
```

- `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

- Special subtype of `std_ulogic`  
`subtype std_logic is resolved std_ulogic;` IEEE 1164-2001 3.10.1
- `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?

However, the issue of multiple drivers obviously still exists. So, how is this handled with `std_logic`? And what value will `x` have in case of conflicting drivers?

## IEEE 1164 std\_logic Type

HWMod  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

`std_ulogic`

`std_logic`

Resolution

- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic; IEEE 1164-2001 3.10.1
```

- `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?

- Special subtype of `std_ulogic`  
`subtype std_logic is resolved std_ulogic;` IEEE 1164-2001 3.10.1
- `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`)

The way this is handled is by using a so-called *resolution function*.

## IEEE 1164 std\_logic Type

HWMod  
WS25

IEEE 1164  
Motivation  
Standard  
VHDL Types  
std\_ulogic  
std\_logic  
Resolution

- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic; IEEE 1164-2001 3.10.1
```

- `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`)

- 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`)

In the case of the `std_logic_1164` package this function is called `resolved`. This function is so integral to resolved types, that it is even part of their respective subtype declaration.

## IEEE 1164 std\_logic Type

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

`std_ulogic`

`std_logic`

Resolution

- 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`)

We will now discuss resolution functions in more detail.

## IEEE 1164 std\_logic Type

HWMod  
WS25

IEEE 1164  
Motivation  
Standard  
VHDL Types  
`std_ulogic`  
`std_logic`  
Resolution

- Special subtype of `std_ulogic`

```
subtype std_logic is resolved std_ulogic; IEEE 1164-2001 3.10.1
```

- `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`)

In a nutshell, the resolution function must define the outcome of multiple drivers assigning a value to a shared signal. We refer to the result of this function as the *resolved value*.

## Resolution Function

44

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

Resolution

Overview

std\_logic

RES\_TABLE

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

In general such a resolution function is nothing special. It is merely a function that takes an array of all values being assigned to a signal, and returns the resolved value.

## Resolution Function

44

HWMoD  
WS25

IEEE 1164

Motivation  
Standard  
VHDL Types  
Resolution  
Overview  
std\_logic  
RES\_TABLE

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

During the simulation, this function is invoked to determine a single, effective, signal value resulting from the ones applied by multiple drivers. We want to stress that this has no real meaning for the synthesis in a physical circuit, as the resolution of conflicting drivers is in reality a result of physics and cannot simply be defined. Indeed, the sole purpose of the resolution function is to coarsely approximate the physical behavior of such conflicts.

## Resolution Function

44

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

Resolution

Overview

std\_logic

std\_logic

- 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

- 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

Note that in general such resolution functions cannot only be part of a subtype declaration, where the resolution function it is applied to all signals of this type.

## Resolution Function

44

HWMoD  
WS25

IEEE 1164

Motivation  
Standard  
VHDL Types  
Resolution  
Overview  
std\_logic  
RES\_TABLE

- 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

- 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

In particular, also subtypes of arrays and records can be declared to be resolved. For details, we refer you to the VHDL standard though.

## Resolution Function

44

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

Resolution

Overview

std\_logic

std\_logic

std\_logic

- 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

- 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`

In addition to that, it is also possible to only resolve a single signal by inserting a resolution function name before the signal declaration's type. The slide contains an example declaration of a signal `x` using a resolution function called `resolved`.

## Resolution Function

44

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

Resolution

Overview

`std_ulogic`

`RES_TABLE`

- 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;`

- 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;`

Let us now turn our attention to the resolution function defined in the `std_logic_1164` package, which you can find by following the link to the IEEE repository.

## Resolution Function

44

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

Resolution

Overview

`std_ulogic`

`RES_TABLE`

- 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;`

Naturally, the purpose of this resolution function is to resolve multiple `std_ulogic` values into a single one.

## The std\_ulogic Resolution Function

- Resolves multiple `std_ulogic` values into a single one

# 9-Valued Logic and Resolution (IEEE 1164)

## Resolution

### The `std_ulogic` Resolution Function

```
■ Resolves multiple std_ulogic values into a single one IEEE 1164  
1 function resolved (s : std_ulogic_vector) return std_ulogic is  
2  
3 begin  
4  
5  
6  
7  
8  
9  
10  
11  
12 end resolved;
```

And while we have not covered functions yet, the function declaration shown on the slide is simple enough to grasp it intuitively. In essence, it needs to resolve an array of `std_ulogic` values, which is called a vector, into a single `std_ulogic` value.

## The `std_ulogic` Resolution Function

HWMMod  
WS25

### ■ Resolves multiple `std_ulogic` values into a single one [IEEE 1164](#)

```
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;
```

IEEE 1164  
Motivation  
Standard  
VHDL Types  
Resolution  
Overview  
`std_ulogic`  
RES\_TABLE

```
■ Resolves multiple std_ulogic values into a single one SEE SA OPEN  
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 resolved;
```

The first thing we can observe in the function is a temporary variable being declared which will hold the final resolved value. This variable is initialized to the high impedance value, as this is the weakest driving state. You will understand why it is initialized to this value soon. Let us now consider the body of the resolution function.

## The `std_ulogic` Resolution Function

HWMMod  
WS25

### ■ Resolves multiple `std_ulogic` values into a single one [SEE SA OPEN](#)

```
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;
```

IEEE 1164  
Motivation  
Standard  
VHDL Types  
Resolution  
Overview  
`std_ulogic`  
`RES_TABLE`

```
■ Resolves multiple std_ulogic values into a single one SEE SA  
function resolved (s : std_ulogic_vector) return std_ulogic is  
    variable result : std_ulogic := 'Z';  
begin  
    if s'length = 1 then  
        return s(s'low);  
    else  
        for i in s'range loop  
            res := RES_TABLE(result, s(i));  
        end loop;  
    end if;  
    return result;  
end resolved;
```

## The std\_ulogic Resolution Function

- Resolves multiple `std_ulogic` values into a single one [SEE SA](#) [OPEN](#)

```
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;
```

```

■ Resolves multiple std_ulogic values into a single one
function resolved (s : std_ulogic_vector) return std_ulogic is
  variable result : std_ulogic := 'Z';
begin
  if (s'length = 1) then
    return s(s'low);
  else
    for i in s'range loop
      res := RES_TABLE(result, s(i));
    end loop;
  end if;
  return result;
end resolved;

```

First, the function handles an edge case. In case of a single driving value, this is already the resolved value and thus returned.

## The std\_ulogic Resolution Function

HWMoD  
WS25

### ■ Resolves multiple std\_ulogic values into a single one [SEE SA](#) [OPEN](#)

```

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;

```

IEEE 1164  
Motivation  
Standard  
VHDL Types  
Resolution  
Overview  
std\_ulogic  
RES\_TABLE

```

■ Resolves multiple std_ulogic values into a single one SEE SA
function resolved (s : std_ulogic_vector) return std_ulogic is
    variable result : std_ulogic := 'Z';
begin
    if s'length = 1 then
        return s(s'low);
    else
        for i in s'range loop
            res := RES_TABLE(result, s(i));
        end loop;
    end if;
    return result;
end resolved;

```

In the general case of multiple drivers, the function instead iterates over the driving values and applies a special look-up table. Essentially, this results in the resolution being done for pairs of conflicting values consecutively.

## The std\_ulogic Resolution Function

- Resolves multiple `std_ulogic` values into a single one [SEE SA](#) [OPEN](#)

```

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;

```

```

■ Resolves multiple std_ulogic values into a single one SEE SA
function resolved (s : std_ulogic_vector) return std_ulogic is
    variable result : std_ulogic := 'Z';
begin
    if s'length = 1 then
        return s(s'low);
    else
        for i in s'range loop
            res := RES_TABLE(result, s(i));
        end loop;
    end if;
    return result;
end resolved;

```

This table is fittingly called the *resolution table* and we will now talk about it in more detail.

## The std\_ulogic Resolution Function

- Resolves multiple `std_ulogic` values into a single one [SEE SA](#) [OPEN](#)

```

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;

```

As we just saw, the purpose of the `std_ulogic` resolution table is to act as a look-up table that provides a resolved value for each *pair* of `std_ulogic` values. We will now look at the content of this table as defined in the `std_logic_1164` package.

## 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 );

```

Observe that this table is really nothing more than a two-dimensional array of `std_ulogic` values. However, what makes it special is that each entry in the table follows a particular reasoning to arrive at a sensible resolved value.

## 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 );

```

```

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'), -- U
5   ('U','X','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','X') -- -
13 );

```

For example, consider the content of the first row. This defines the resolved values for all conflicts with the uninitialized value 'U'. And since nothing is known about the uninitialized value, nothing can be inferred about the outcome of a conflict. Hence, the resolved value is uninitialized as well. And while you might argue that one could also just define the resolution to take the other value in such cases, the existing definition is designed to propagate 'U' values in simulations. This simplifies spotting them.

## 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'), -- U
5   ('U','X','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','X') -- -
13 );

```

```

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 );

```

Now consider the row for the weak unknown value. For reasons of symmetry we already know the resolved value for a conflict with 'U'. Hence, let us look at the second entry.

## 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 );

```

```

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 );

```

In this case, a conflict between the strong and the weak unknown value must be resolved. However, since the strong unknown is associated with a stronger driver strength, it will dominate the weaker driver.

## 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 );

```

## 9-Valued Logic and Resolution (IEEE 1164)

### Resolution

### 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 );
```

Similarly, the weak unknown will dominate a high impedance value since this corresponds to the weakest driver strength. Let us now consider an example, to illustrate how multiple driving values are resolved using this table.

## 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 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 );

```

For that, we will return to the wired AND example, where we had multiple drivers and a pull-up resistor connected to a shared bus. Now, let's assume that we have two drivers, one active and one inactive. We can model the drivers using the values for high impedance, 'Z', and for the strong LOW, '0'. For the pull-up resistor we use the weak HIGH, '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           resolve("H0Z"):
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 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 );

```

In a first step, we must resolve the values 'H' and 'Z' due to the function initializing the temporary variable to 'Z'. Since 'H' is associated with a stronger driver strength, it is the resolved value for this pair of values. Due to the choice for the variables initial value, the first resolved value will always be the first driving one.

## 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 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 );

```

Next, this intermediate result must be resolved with '0'. The result is of course also '0' due to the stronger driver.

## 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 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 );

```

Finally, the pair '0' and 'Z' is resolved. As mentioned before, with 'Z' being associated to the weakest driver, the result will again be '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 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 );

```

The resolution is now done and the final value the resolved value. Hence, the shared bus exhibits the value of '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'
⇒ resolve("H0Z") = '0'

```

# 9-Valued Logic and Resolution (IEEE 1164)

## Resolution

### 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 );
```

With this example we want to end this introduction to the nine-valued and resolution functions provided by the `std_logic_1164` package. In the next lecture we will pick up where we left and discuss the array types of `std_ulogic` and `std_logic`, as well as the operators defined on them.

## 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'
⇒ resolve("H0Z") = '0'
```

Thank you for listening! We recommend you to immediately take the self-check test in TUWEL, to see if you understood the material presented in this lecture.

HWMoD  
WS25

IEEE 1164

Motivation

Standard

VHDL Types

Resolution

Overview

std\_logic

RES\_TABLE

# Lecture Complete!