In-depth

STD_LOGIC VS STD_ULOGIC

Wednesday, Nov 21st, 2018

VHDL includes few built-in types but offers several additional types through extension packages. Two of the most widely used types are `std_logic` and `std_ulogic`. The difference between them is that the former is resolved while the latter isn't.

Before we go on to investigate what it means that a type is resolved, let's first look at the traits that the two types share in common.

`Bit` and `boolean` are part of the standard package, requiring no imports to use them. But `std_logic` and `std_ulogic` can only be used after importing the IEEE 1164 package. To import the IEEE 1164 package, simply add these lines to the top of the VHDL file:

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

'1'	Logic 1
'0'	Logic 0
'Z'	High impedance
'W'	Weak signal, can't tell if 0 or 1
'L'	Weak 0, pulldown
'H'	Weak 1, pullup
'-'	Don't care
'U'	Uninitialized
'X'	Unknown, multiple drivers

Most of the time, you will use `'1'` and `'0'` to indicate a logic high or low value. And `'U'` will be used for representing uninitialized values, such as RAM content at startup. Occasionally you will see the `'X'` value, indicating some sort of driver conflict.

STD_ULOGIC – THE UNRESOLVED TYPE

Let's first have a look at the `std_ulogic` type, the unresolved version of the two. Below is an excerpt of the type declaration taken from an implementation of the std_logic_1164 package.

```
TYPE std_ulogic IS ( 'U',  -- Uninitialized
                     'X',  -- Forcing  Unknown
                     '0',  -- Forcing  0
                     '1',  -- Forcing  1
                     'Z',  -- High Impedance
                     'W',  -- Weak     Unknown
                     'L',  -- Weak     0
                     'H',  -- Weak     1
                     '-'   -- Don't care
                   );
```

The `'U'` value is the first of the listed values. That's the reason why a signal of the `std_ulogic` type will get the value `'U'` if given no initial value. By default, an uninitialized signal will get the leftmost value from the type declaration. That's how VHDL works.

WHAT DOES IT MEAN THAT A TYPE IS *UNRESOLVED*?

If multiple drivers are driving different values onto a signal of the `std_ulogic` type, that's not going to work. Conflicting drivers is an error in VHDL. The simulation won't compile, or the design won't synthesize. Driver conflicts are not resolved with the `std_ulogic` type. The type doesn't have a built-in mechanism to determine what the signal value should be, so it's an error.

Consider this example where two processes attempt to drive a signal of `std_ulogic` type:

```vhdl
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3
 4  entity UnresolvedTb is
 5  end entity;
 6
 7  architecture sim of UnresolvedTb is
 8      signal Sig1 : std_ulogic := '0';
 9  begin
10
11      -- Driver A
12      Sig1 <= '0';
13
14      -- Driver B
15      Sig1 <= '1' after 20 ns;
16
17  end architecture;
```

If we try to compile this in ModeSim, it reports the following error:

```
# ** Error: C:/proj/tb.vhd(8): Nonresolved signal 'Sig1' has multiple sources.
#   Drivers:
#     C:/proj/tb.vhd(12):Conditional signal assignment line__12
```

STD_LOGIC – THE RESOLVED TYPE

The `std_logic` can represent the same values as the `std_ulogic`, but it's a resolved type. What does that mean? To find out, let's once again refer to the implementation of the standard. Below is the declaration taken from the IEEE 1164 package.

```
SUBTYPE std_logic IS resolved std_ulogic;
```

The `subtype` keyword in VHDL is normally used for declaring a type with a limited range from the base type. But it can also be used for specifying a *resolution function*. That's what the above statement is saying. The `std_logic` is a subtype of `std_ulogic`, and the name of the resolution function is "resolved".

Now, let's examine the "resolved" function:

```
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
            result := resolution_table(result, s(i));
        END LOOP;
    END IF;
    RETURN result;
END resolved;
```

It accepts one parameter, an array of `std_ulogic` representing all the simultaneous drivers. It then compares all the elements in the vector to each other, one by one, reducing them to a single `std_ulogic` value which is returned.

For comparing the values, what seems to be a different function named "resolution_table" is called. But it's actually a two-dimensional array:

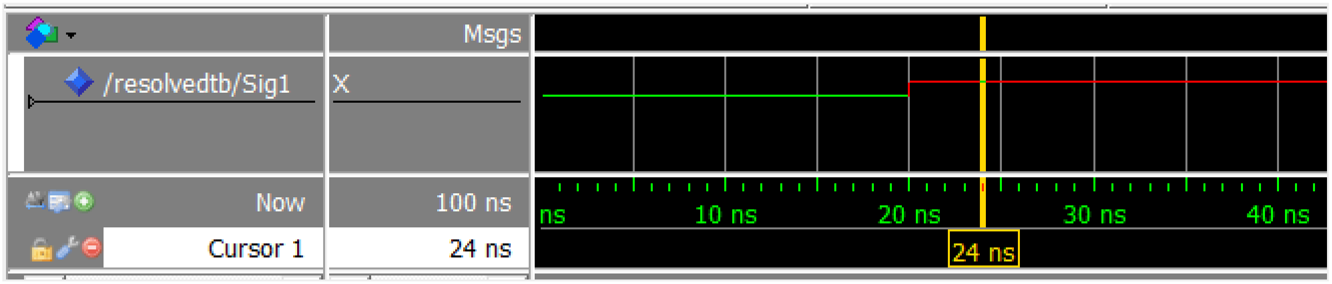
```
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;

CONSTANT resolution_table : stdlogic_table := (
--      ---------------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -        |   |
        ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
        ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
        ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
        ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
    );
```

The resolution function governs the final value that a signal will have in case of multiple drivers.

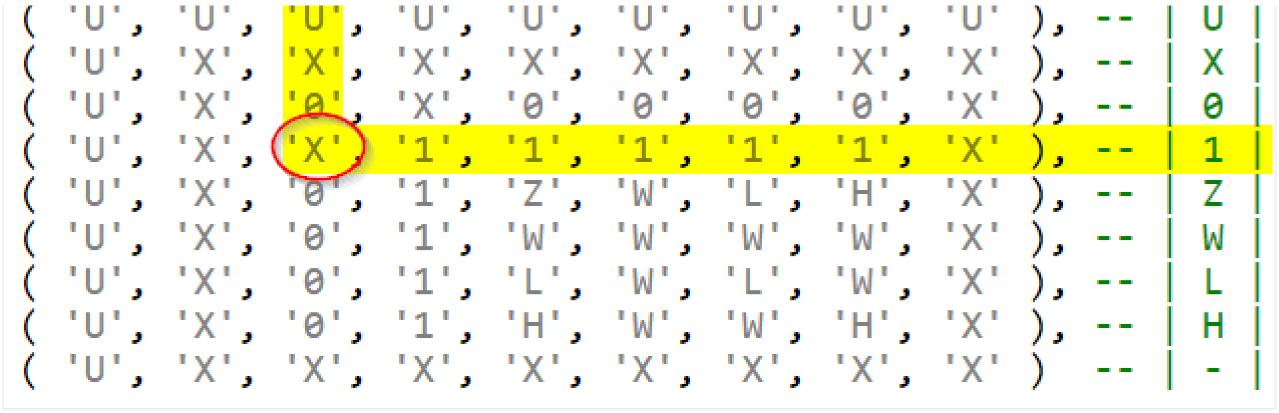
This is the same example code as before, but with the `std_ulogic` converted to `std_logic`:

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

entity ResolvedTb is
end entity;

architecture sim of ResolvedTb is
    signal Sig1 : std_logic := '0';
begin

    -- Driver A
    Sig1 <= '0';

    -- Driver B
    Sig1 <= '1' after 20 ns;

end architecture;
```

The code now compiles in ModelSim without errors. When we run it, it produces this waveform:



At first, only Driver A is driving a `'0'` value onto `Sig1`. After 20 nanoseconds, Driver B starts driving `'1'` onto the same signal. The conflicting values were resolved according to the resolution table, resulting in the signal getting the value `'X'`:


```
( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
```

WHY USE STD_LOGIC?

The `std_logic` is generally preferred over the built-in `bit` or `boolean` types in VHDL. This is because they give us more information than the simple `'1'` or `'0'`, on or off. If you prefer to have the simulator treat multiple drivers as hard errors and stop at the first occurrence, you could, of course, use `std_ulogic`.

Unintended multiple drivers probably won't go undetected, even when using the `std_logic` type. You will likely see the "metavalue detected" warnings in the simulator transcript window as the `'X'` values propagate through the datapath.

`std_logic` has become the most used type for internal and external interfaces because it accurately models a digital electrical signal's behaviors. The `std_logic` type offers a number of benefits and use cases, including the following.

UNINITIALIZED VALUES

Tracking of uninitialized values is useful for detecting missing resets and the use of uninitialized data. Since block RAM cannot have reset values, they will contain only `'U'` values at simulation start. This makes it easy for us to spot uninitialized RAM content being used in the simulation waveform. With a two-value type like `bit` or `boolean`, such errors would go undetected.

Implementing tri-state logic is best done with a resolved signal like `std_logic`. The high impedance state `'Z'` is normally used for releasing the bus while another driver has control of it. In VHDL, there is no undo when a process begins to drive a value onto a signal. There exists no keyword to stop driving. The only way to release the signal is by driving a weaker value which can by overridden by another driver.

MODELING EXTERNAL INTERFACES

To accurately model an external signal, a boolean value is not enough. External digital interfaces can exhibit behaviors like pull-up or pull-down logic. Without the `'H'` and `'L'` values, it would be difficult to simulate interfaces that rely on pulling like I^{2}C or SPI.

PROPAGATING ERRORS DOWNSTREAM

A strategy that will make erroneous usage of modules easier to detect is setting the outputs to `'X'` when the data is invalid, for example, when the `valid` output is `'0'`. Downstream modules that sample the outputs at the wrong time will receive the `'X'` value. This will cause a simulation error or metavalue warning, rather than an invalid `'0'` or `'1'` silently being used. It can prevent errors that are otherwise difficult to detect.

We did this when implementing a multiplexer in the Case-When tutorial.

Author: Jonas Julian Jensen

I'm from Norway, but I live in Bangkok, Thailand. Before I started VHDLwhiz, I worked as an FPGA engineer in the defense industry. I earned my master's degree in informatics at the University of Oslo.

Existing customer? Click here to log in: MY LIBRARY

ABOUT CONTACT

« Delta cycles explained How to create a Linked List in VHDL »

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

COMMENT

To add a block of VHDL code to your comment:

```
[vhdl]
signal my_slv : std_logic_vector(7 downto 0);
[/vhdl]
```

NAME *

EMAIL *

WEBSITE

✓ NOTIFY ME OF REPLIES TO MY COMMENT VIA EMAIL

POST COMMENT

This site uses Akismet to reduce spam. Learn how your comment data is processed.

6 THOUGHTS ON "STD_LOGIC VS STD_ULOGIC"

Lawrence Nwaogo

REPLY

Jonas Julian Jensen - https://jonasjulianjensen.com/

Thanks, Lawrence. That's nice to hear!

Posted on June 4, 2020 at 6:15 pm

REPLY

Martinho Mussamba

Great job. Good explanation.

Posted on December 3, 2020 at 10:38 am

REPLY

Jonas Julian Jensen - https://jonasjulianjensen.com/

Thanks, Martinho. I'm glad you liked it.

Posted on December 4, 2020 at 2:35 pm

REPLY

Philip Abbey

> "Unintended multiple drivers probably won't go undetected"

So VHDL is a strongly typed language intended to prevent silly mistakes. So I find it bizarre that std_logic has become so popular when using std_ulogic absolutely does detect unintended multiple drivers. Therefore you could use std_ulogic all the time except for when you intend to code a tri-state buffer, and this feels fitting with VHDL's strongly typed nature too.

More an observation than a question, do you agree popularity within the industry went the wrong way on this?

Xilinx IP Cores are all coded with the resolved type. This makes conversion between std_logic_vector and std_ulogic_vector a pain worth avoiding. A pain that would not have existed if the IP Cores were originally coded with the unresolved type instead. And this may have driven the general adoption the other way.

Feels like a missed opportunity to have the compiler help us.

Posted on July 10, 2021 at 5:38 pm

REPLY

Jonas Julian Jensen - https://jonasjulianjensen.com/

I can see your point because I've had the same thought from time to time.

A situation of unintended multiple drivers will smack you in the face as an error at compile time when using `std_ulogic`. But with `std_logic`, you must detect it during simulation, or it will show up at synthesis time.

I should add that VHDL-2008 and above allows automatic casting between `std_logic` and `std_ulogic` types. If all vendors could start supporting it, that would be great.

Posted on July 10, 2021 at 5:59 pm

REPLY

VHDLwhiz helps you understand advanced concepts within FPGA design without being overly technical.

Learn VHDL the easier way.

You can do this! 😎

Search …

Existing customer? Click here to log in: MY LIBRARY

ABOUT CONTACT

News

Opinion

Tips and tricks

Tutorial

FILTER BY KNOWLEDGE LEVEL

Beginner

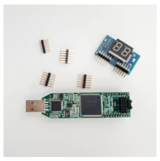
Intermediate

Advanced

PREMIUM COURSES

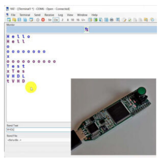
Course: FPGA and VHDL Fast-Track

$ 27



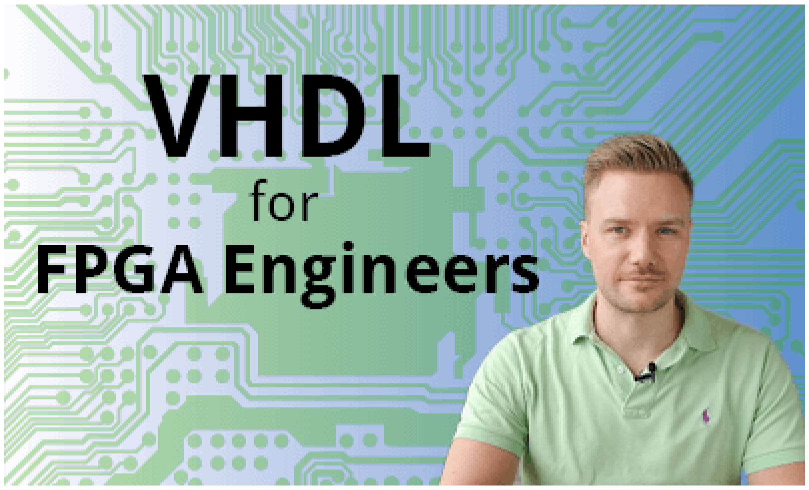
Course: Run-length encoding in VHDL

$ 89



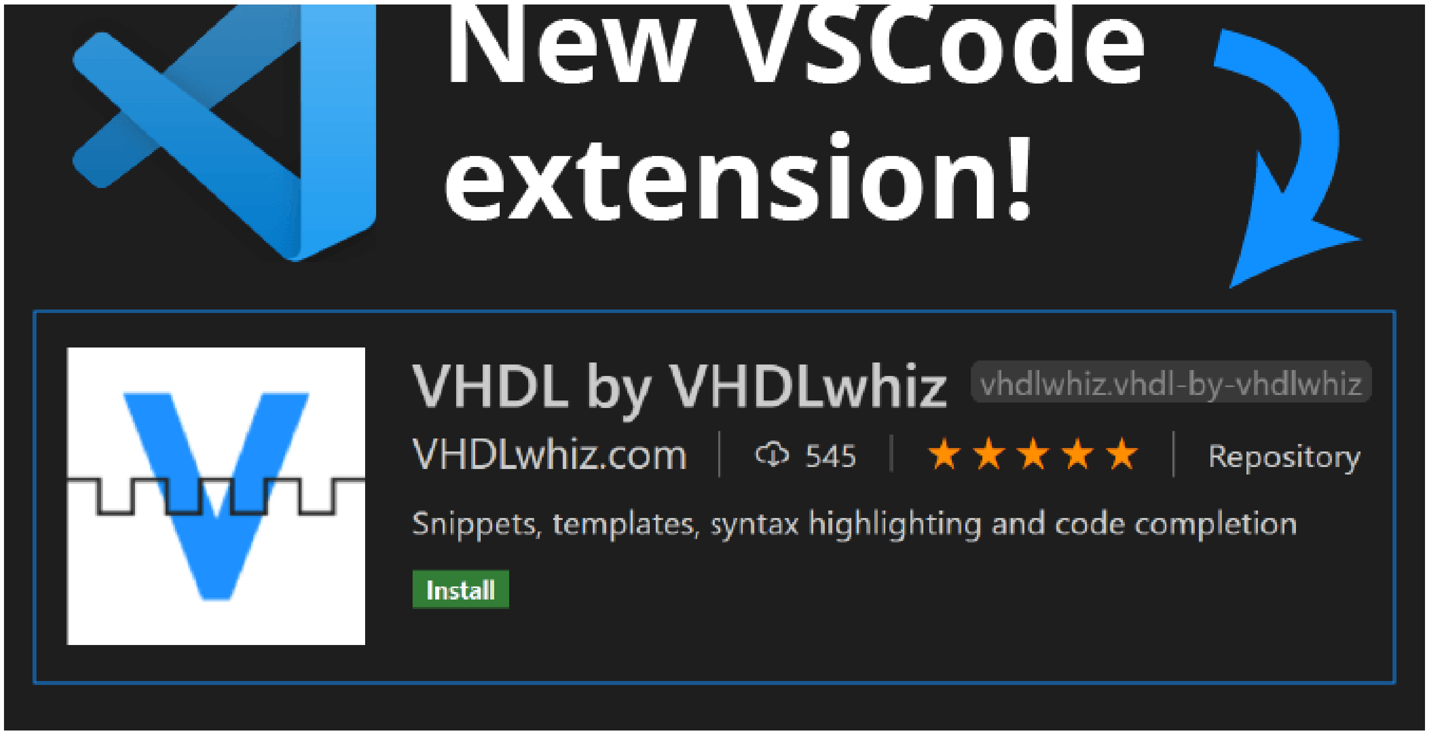
FACEBOOK GROUP

Join the private Facebook group! Participate in discussions and post your questions about VHDL and FPGAs.



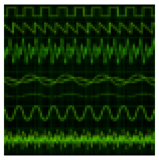
Existing customer? Click here to log in: MY LIBRARY

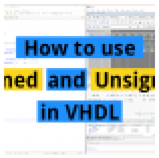
ABOUT CONTACT


New VSCode extension!
VHDL by VHDLwhiz
vhdlwhiz.vhdl-by-vhdlwhiz
VHDLwhiz.com
545
Repository
Snippets, templates, syntax highlighting and code completion
Install

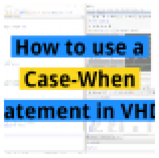
Click here to get it

POPULAR POSTS

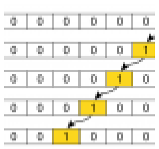

Free VHDL simulator alternatives



How to use Signed and Unsigned in VHDL



How to use a Case-When statement in VHDL


8 ways to create a shift register in VHDL

How to create a Finite-State Machine in VHDL

Existing customer? Click here to log in: MY LIBRARY

ABOUT **CONTACT**

PRIVACY POLICY TERMS OF SERVICE COOKIE POLICY