

vdt specification

[Document Subtitle]

RyanHunter

[personal]

1. overview

this document describes the whole strategy for developing vdt-v1.x version tools.

TBD, omitting some redundant descriptions

1. usage&features

This chapter will list all possible usage/issues or features this tool are going to solve, it's kind of a very beginning state to collecting requirements and provide the features, And because of the lack of experience, this chapter may continue updating.

* 1. designs

A design here refers to a module in verilog, we can call 'design' with a bunch of code to declare a design module.

* 1. signals

There're two ways to publish signals to RTL code, one is explicitly declared by a global or module perspective signal method, another way is to implicitly inferred by the tool.

As for explicit declaration, users need to call with 'signal' method, with args such as type/name/width, however, it might be tedious to declare so many signals by calling that method, so we need a way to facilitating the signal declaration.

* + 1. signal pool

Signals can be declared within a signal pool, which is like a namespace, by importing that signal pool, a design module can then choose which of those signals to use.

Signal pool are much like to be used by declaring ports of a design.

* + - 1. reverse

the signal pool has a built-in method that can be called by design/component like: <sigPoolName>.reverse, which will revert the directions of all the signals within that pool, this action will affect only on <in/out> type of the signals, other type will be ignored.

* + 1. signal examples

```

sigpool :poolE {

signal :a,:in,10

signal :b,:out,2

}

<within a design>

poolE :sInst

```

* + 1. flexible signal operations

There might be two signals that have the same name, but different width or type. So we can first declare a signal in global scope, and then declared within a design with a succeeding method. (TBD)

* 1. components

a component is a bunch of logical operations that can be reused (by instance mechanism) in different modules, but it actually will not directly reflect on the target RTL file. It's simply a reusable block. Users can easily declare a new component similarly as declaring a design.

* + 1. declare a component

By calling component literally will trigger a declare action to create and register a new component in the global scope, every design can instantiate that component directly with some extra instantiating operations.

* + 1. supports

A component can support followings:

* design features, similar as in design, feature can be declared and used in a component
* signals, internally declare a signal of the component is more like a temporary place hold for instantiation, the name here might be changed when instantiated in a design, but the width and the type can be adopted.
* TBD
  + 1. feature in component

feature behaves all what it should have in a component, it has no specific actions we need to consider, so just refer to the feature section for details. <Link Here>

* + 1. signal in component

If we use signals just as how we do in design, it can be a bit more complicated for coding, so we need a way to shorten the complex of declaring a signal in a component.

**strategy** 1

Simply declare a signal name as a place holder that can be replaced by a real signal when in component instantiation.

* + 1. instantiate a component

If a component is declared before, then it can be instantiated everywhere within a design declaration. Instantiation can be done by calling the component name as a method, and with instance name and features as arguments. And signals in the design need to be filled into corresponding placeholders, like directly variable assignment within component instantiation.

`

aComponent inst {

iClk parent.iClk ## call local iClk as a method in aComponent

iReset parent.iReset

}

`

* + 1. examples

TBD

* 1. logics

The essence operations of a integrated circuit is kind of a logical operation, so logics are at the bottom layer of those component/design ... layers.

All logic operations should be pre-defined in the basic vdt flow, users don’t have to declare a logic, just to use it with variety configurations.

* + 1. conditional assignment

the conditional assignment is like, if one or more conditions matched, then assign an expression to a target signal, this logic can be sequential or combinational.

To use this logic, just call a method named **lcond**, with specific arguments like:

* circuit type: sequential or combinational
* relations of what condition is and it’s result if matched
* target signal to be assigned

one call of **lcond** only supports one target. For combinational circuit, use assign instead.

**example**:

`

lcond <seq/comb>,<target>, {“result”=>”condition”, “result”=>”condition”}

`

TBD, more logics need to be decided by real developing a design.

* 1. features

1. implementation&strategies

This chapter gives strategies and implementations for vdt development.

TBD