Subject: **`-dangles`** Proposal

Hi, All -

Proposal idea - I would like to raise this issue in the BC and based on feedback, possibly take an action item to work out the proposal.

There has been much recent debate on ESNUG (with more feedback coming to the next ESNUG issue) regarding the purpose, value and efficiency of using the Verilog-2001 **`default_nettype none** compiler directive.

The intended purpose was to find mistakes through declarations, but the **`default_nettype** compiler directive falls short in many ways (I would be happy to provide examples of multiple problems that will not be caught by **`default_nettype** and indeed can be hidden by **`default_nettype**).

After pinging a number of users, we tend to agree (with some dissention) that connectivity is a superior means to catch design bugs. The ability to check for single-ended connections or unconnected nets would find many more bugs than forcing declarations that can simply satisfy a compiler by making a declaration for a single-ended wire.

Note Jonathan and Shalom have raised good questions and concerns (in a separate private email thread) but after much consideration, I believe I have a reasonable approach to their concerns and would be very open to modifications to my proposal below.

I will first address the most common issues and how they are resolved by this proposal, then I will address issues raised by Jonathan and Shalom (exception conditions)

I would like to explore the following proposal that would significantly help design engineers:

Add the compiler directives (can be embedded into code):
`report_dangles on
`report_dangles off

Add a requirement that vendors add a command line switch to check the compilation unit for the same potential "dangles" problems and recommend that vendors use the command line switch-name:
`-dangles`
Command line switch names cannot be enforced but there is precedence to recommend switch names (such as **`-lib`**)

If added to the code, the compiler directives would have precedence over the command line switch.

When the dangle-checker command line switch is activated from the command line or when activated through a compiler directive, compilers shall reports errors when:

- a scalar net or variable has driver(s) but no receiver
- a vector net or variable has driver(s) but no receiver on one or more bits in the vector
- a scalar net or variable has receiver(s) but no driver
- a vector net or variable has receiver(s) but no driver on one or more bits in the vector
- a scalar or vector nets or variables have no driver(s) OR receiver(s) (declared net or variable is not in use - remove the clutter from the design)

This would catch the majority of the bugs that engineers typically try to detect, but Jonathan and Shalom have raised a few good exceptions:

First Shalom's:
[SB] For example, very often, signals are declared as placeholders. A common example is of scan signals, which are often connected only after synthesis.

I had already considered this potential and had proposed making Z-assignments to the bits but I was never very comfortable with that idea. I was trying to find something along the lines of empty named port connections
(e.g. -
```
dff u1 (.q(q), .qb(), .d(d), .clk(clk));
    // unconnected qb output )
```
And I think we could do something similar with an enhancement and no new keywords:
```
module mymod1 (output scanout, input scanin, ... );
  `ifndef SCAN
    assign scanout = (); // empty driver on scanout
    assign () = scanin;  // scanin drives empty receiver
  `else
    // ... actual scan logic code - add later
  `end
 ...
```
The empty parentheses are already used for an empty connection and this code shows designer intent.

[SB] Another example is of vectors where often you declare an entire vector but only use part of it. The rest of it is just a placeholder, or is used elsewhere.

Of course, it would be great if we could declare an identifier more than once with a different range to satisfy this requirement, but the alternative would be:
```
// In this module,
// only addr[31:24] and addr[7:0] are in use
module mymod1 (input [31:0] addr, ... );
  assign () = addr[23:8]; // addr[23:8] drives empty
                          // receiver bits
  ...
```

In both of these examples, the designer has indicated to all tools that the designer is aware of bits without drivers or receivers and that those bits should be ignored in compilation. It might also be useful (??) to have compilers treat the empty-driver and receiver bits as unresolved bits (a la **`logic`** bits) and flag errors if any other drivers or receivers are attached to the assignment-declared empty bits (??)

In both cases, I believe synthesis tools could simply ignore these constructs. The constructs are put into place to catch mistakes early in the design cycle.

For most of the RTL designs that I work on, I will never need the empty driver or receiver assignments. These just cover reasonable exceptions raised by Shalom.

Second, Jonathan's:
[JB] for RTL I want the checking local to a module, but for general Verilog you can have sources and sinks on a signal by cross-module reference (CMR) ... I'm not sure how to deal with that.

A very good point! CMR's are not allow in synthesizable RTL, but they appear frequently in Verilog and SystemVerilog testbenches.

One could surround the non-RTL code with **`report_dangles off`** - **`report_dangles on`** and ignore the issue, but I believe there is value in doing the checking in the testbenches too.

The problem surfaces in separate compilation where external CMR drivers and CMR receivers might not be present in the compilation unit.

Can we tell the compiler that CMR drivers and receivers exist?
Can we tell the elaborator to check for external CMR drivers and receivers?

I believe we can do both (says the guy who has not written any of the tools ☺)

A possible proposal, one that again does not use any new keywords, would be to use an empty extern driver or receiver assignment.

```
// assumes that sig1 will be a CMR-referenced driver-signal
// elsewhere in the elaborated design. Do not check during
// compilation. Only check during elaboration.
extern assign () = sig1;

// assumes that sig2 will be a CMR-referenced receiver-
signal elsewhere in the elaborated design. Do not check
during compilation. Only check during elaboration.
extern assign sig2 = ();
```

- Additional restriction: This also raises the point that CMR signals in the code should only be checked for potential-driver and potential receiver and verified during elaboration.

Thoughts and feedback welcome. I have tried to choose coding proposals that would make sense but I am very open to alternate syntax.

Regards - Cliff