Hi, All -

For the past six years, I have been trying to fix the annoying requirement that forces engineers to declare procedural variables or change declarations when procedural variables are moved to continuous assignments or vice versa. I even wrote an HDLCON 2000 conference paper on the topic titled*: "A Proposal To Remove Those Ugly Register Data Types From Verilog"* (downloadable from my web site).

Every EDA tool developer should be required to teach Verilog syntax and semantics to a class of bright VHDL engineers who have been forced by company policy to learn and use Verilog and listen to their complaints about the separate **`reg-wire`** declaration requirements. Every EDA tool developer should be required to teach Verilog syntax and semantics to a class of new Verilog engineers and explain the reason for this requirement.

Engineers are not interested in the fact that the declarations make EDA tool development easier and engineers see no value in making these different declarations just because the language requires it.

I would like to try one more time to fix this in SystemVerilog.

I talked about the **`reg-wire`** proposal to Stu Sutherland and Karen Pieper after the P1800 meeting yesterday to gauge acceptance or opposition. Stu and Karen suggested that Steve Sharp and Francoise and Brad be brought into the discussion for their technical insight into potential tool issues and limitations. I would like to invite comment from all interested parties.

Let me list the goals for this enhancement:
(1) Fully backward compatible with all existing Verilog and SystemVerilog designs.
(2) Should not require an extra keyword to be added to **`output`**, **`input`** and **`inout`** port declarations to use the enhancement.
(3) Should have a common keyword to declare **`wire`**-equivalent and **`reg`**-equivalent vectors within a module.
(4) Should be first-usage context sensitive. If the first usage is that of an assigned variable within a procedural block, the semantics should be identical to a declared **`reg`**-variable (one nice, optional, possible exception noted below). If the first usage is that of a driven net (**`output`** of a continuous assignment or connected to the **`output`** or **`inout`** port of an instantiated module or Verilog primitive), the semantics should be identical to a declared **`wire`** or **`wire`**-vector (including multi-driver resolution, unlike the **`logic`** data type which only allows single driver-assignment).
(5) It shall be illegal to make both procedural and driver assignments to the same identifier (this is enforced in Verilog today because it is illegal to declare **`wire`**s and **`reg`**s using the same identifier name).
(6) OPTIONAL: My own preference is that this new data type shall only be assigned from a single procedural block and any attempt to make assignments to the variable from more than one procedural block would be a syntax error (just like

`always_comb`, `always_latch` and `always_ff` procedural blocks). This would enforce good hardware design practices that are already enforced by synthesis tools. If this requirement proved to be a stumbling block, I could easily forego this capability to get the rest.

My existing proposal overloaded the `wire` key-word, so no new additional keyword would be required. Using `wire` is also very intuitive to hardware design engineers because it just connects logic, and RTL designers do not distinguish between combinational logic coded as an `always` block and combinational logic coded as a continuous assignment. `wire`s are also used connect the outputs of flip-flops to other logic, so there really is no need for a `reg` variable as far as hardware engineers are concerned (some companies do not even allow RTL designers to code RTL flip-flops - they must be instantiated from a company library).

Stu's objection to the `wire`-overloading proposal was primarily related to the PLI. Stu noted that internal vectors declared as `wire`s but assigned with procedural assignments could be de-compiled by the PLI and the de-compiled code would show that the internal "`wire`"-vector in the elaborated design was indeed a `reg`, which could be confusing.

Stu suggested an alternative that would meet all of my enhancement goals but would require a new keyword. For now let me call the new keyword `TBD` (the to-be-determined keyword).

(1) `TBD` would be the new universal implicit declaration type.

(2) `TBD` types would be first-usage sensitive and behave exactly like `wire`s or `reg`s, depending on first usage.

(3) It shall be illegal to make driver assignments and procedural assignments to the same variable.

(4) To make it possible to still have implicit data type port declarations, the `TBD` type would have to be the new default data type for SystemVerilog designs. This is still 100% backward compatible with existing SystemVerilog designs because undeclared port types would default to the `TBD` type and per first-usage requirements, the `TBD` ports would turn into `wire`-ports, the same as SystemVerilog today.

(5) De-compiled listings using the PLI would still show the individual identifier types to be `wire` or `reg`, but Stu said this would be reasonable compared to declaring a `wire`-identifier and having it show up in the de-compiled PLI as a `reg`-identifier..

NOTE: when some of us first joined the Superlog Work Group, the Co-Design guys wanted `logic` to be the default data type, but we pointed out that that would break existing Verilog designs that had undeclared multi-driver nets. That is why `logic` was rejected as the new SystemVerilog data type. The `TBD` data type would not have this problem.

Karen tried to find examples where an implicit, first usage type would fail. Karen gave the example of a pass-through module where the data types of the ports for both modules were undeclared. These are easily determined to be `TBD` `wire`-type ports.

Verilog already connects **regs** to **wires** through instantiated module ports, so a **TBD-reg-output** port connected to a **TBD**-external-**wire** works fine. Similarly, a **TBD-reg**-variable connected to an instantiated **TBD-wire-input** port also works fine.

Stu, myself and Karen are trying to figure out if there are some unresolved corner cases that we have not considered.

I believe Stu and I would like to know what the interest-level and opposition-level is to this type of proposal.

I personally believe that SystemVerilog users are not as likely to use the PLI (even though tool vendors will still use the PLI) and that overloading the **wire** keyword for the **reg-wire** proposal would be fine. The tradeoff is measuring the confusion-level that might be experienced by PLI users against the problems of finding a good descriptive keyword to use for the **TBD** variable and how many designs the new keyword would break, but I could be persuaded to use either.

This type of enhancement offers the following advantages to the average design and verification engineer:

(1) Finally get rid of the confusing net-variable separate declaration requirements.
(2) Eliminate the need to change declarations when the type of assignment changes within a module.
(3) More concise.
  - *Enhanced SystemVerilog:* For simple designs that use the **TBD** type, the only required **TBD** declarations would be internal buses (net or **reg**) and internal scalar declarations when using either the **.name** or **.*** SystemVerilog implicit port connection styles.
  - *Existing SystemVerilog & Verilog-2001:* Require additional variable declarations for ports that are driven using procedural assignments and internal scalar variables, but not **wire** ports or internal scalar **wires**.
  - *To force SystemVerilog & Verilog-2001 declarations:* If you want to force engineers to declare all port and internal identifiers, you can always use the Verilog-2001 **`default_nettype none** compiler directive.

(4) Remove the verbose and annoying need to add **`ifdef** for declarations that change according to usage context.

Please, for a little inconvenience on the part of the EDA tool vendors, we achieve a lot of simplification and eliminate a nagging Verilog annoyance.

All feedback comments and potential-problem examples welcome.

Regards - Cliff