Hi, All -

I could support an extern module proposal if it is clear that tools that allow separate compilation may require extern modules if the submodules have not been previously compiled.

For example:

```
module dff_wrapper
  (output q,
   input  d, clk, rst_n);

  dff a (.*);
endmodule

module dff
  (output reg q,  // note the ugly reg declaration!
   input      d, clk, rst_n);

  always @(posedge clk or negedge rst_n)
    if (!rst_n) q <= 0;
    else        q <= d;
endmodule
```

For the above example, any tool that is asked to compile the entire design (not separately) should be able to compile this design without declaring extern modules.

For a tool that allows (or requires) separate compilation, as long as the dff module is compiled first, the dff_wrapper should be permitted to compile without an extern module declaration (Karen's proposal does not seem to permit this - I would like this to be allowed and clarified in the proposal).

For a tool that allows (or requires) separate compilation, if the dff_wrapper is compiled first, it is reasonable for a tool vendor, like Synopsys, to require an extern module declaration. The point is, I don't ever want to use extern modules. A reasonable restriction is to at least force me to create empty modules with headers and compile them first before I can compile the top-level module with .* instantiation. I could vote for this. In real designs, I anticipate that I will have all of my modules coded and simulated using VCS before I go to Synopsys synthesis and I will create a Makefile that reads the files into Synopsys in the correct order without extern modules. We know that this can be done. Intel's IHDL tools have been doing this without extern modules for years.

Karen and I talked about a methodology where and engineer might build an extern module and then include it in both the top-level design and the instantiated module. One of my chief complaints about included files is that the included code is usually not stand-alone code (you cannot compile the included code separately, typically because it lacks module-endmodule and the rest of the supporting declarations). About the only thing I include in my designs are `define macro definitions, because they can be tested separately from the rest of the design. I'm not sure if this extern-module proposal permits separate compilation of the extern module itself, or if it requires a supporting module to surround it.

If we pass this enhancement, I will teach engineers how extern modules work and then generally advise them to code the submodule headers first as opposed to playing with the extern modules (they are there if you really think you need them, but you really don't need them!)

More comments below.

Regards - Cliff

===== Karen's proposal =====

Extern Module Proposal

Issue: .* cannot be implemented in a separate compilation scenario like
DC uses because the downside ports cannot be known.
The ports can be known if the submodules with appropriate module headers are compiled first.

The issue is more complicated by the fact that if the downside
contains a generic interface, the upside is needed to know the
type of the downside port.
This may be true. Using extern modules may be required here.

Solution: The idea for this proposal is to allow extern declarations of
modules. An extern module declaration of module x will include
the ports of the module. The module x can have .* as its
portlist, which will result in the use of the ports from the
extern declaration.

Standard changes in Section 12.7.4 Instantiation using implicit .* port connections

At the end of 12.7.4 ADD:

To support separate compilation, extern declarations of a
module ~~can~~ may be used to declare the ports on a module without
defining the module itself. An extern module declaration
consists of the keyword extern followed by the module name,
and the list of ports for the module. Both list of ports
syntax (possibly with parameters), and original Verilog
style port declarations may be used. Note that the potential
existence of defparams precludes the checking of the
port connection information prior to elaboration time (have we defined elaboration time in the SystemVerilog standard?) even for
list of ports style declarations.
Make as many restrictions on defparams as you would like. We deprecated them and another reason to not use them can only be a good thing!

The following example demonstrates the usage of extern
module declarations.

```
extern module m (a,b,c,d);
extern module a #(parameter size= 8, parameter type TP = logic[7:0])
       (input [size-1:0] a, output TP b);

module top ();
   wire [8:0] a;
   logic [7:0] b;

   m m (.*);
   a a (.*);
endmodule
```

Modules m and a are then assumed to be instantiated as:

```
// Where possible, showing the named port connections helps remove confusion
module top ();
```

```
    m m (.a(a), .b(b), .c(c) ,.d(d));
    a a (.a(a), .b(b));
endmodule
```

If an extern declaration exists for a module, it is possible to use .* as the ports of the module. (too strong - implies that extern modules are required) This usage will be equivalent to placing the ports (and possibly parameters) of the extern declaration on the module. For example,

```
extern module m (a,b,c,d);
extern module a #(parameter size= 8, parameter type TP = logic[7:0])
        (input [size-1:0] a, output TP b);

module m (.*);
    input a,b,c;
    output d;
endmodule

module a (.*);
endmodule
```

is equivalent to writing:

```
module m (a,b,c,d);
    input a,b,c;
    output d;
endmodule

module a #(parameter size= 8, parameter type TP = logic[7:0])
        (input [size-1:0] a, output TP b);
endmodule
```

Extern module declarations can appear at any level of the instantiation hierarchy, but are visible only within the level of hierarchy in which they are declared. (This seems to preclude separate compilation of extern modules that are then `include-ed) It shall be an error for the module definition to not exactly match the extern module declaration.

The proposal needs to make clear that only tools that can perform separate compilation, might reasonably require extern modules if the top-level module is compiled before the instantiated submodules; otherwise, I would oppose this proposal (Cliff).