Subject: Proposal: Default Interface Ports & Ref Port Modifications

Hi, All -

Stu Sutherland, Dave Rich, Karen Pieper and myself met at Boston SNUG and discussed in great detail interface default ports and ref ports. The following summary and proposals were discussed.

(1) Specifying that non-modport interface variables default to ref-ports will be confusing to new and existing Verilog users.
(2) Users (like Intel) do not want to be forced to declare modports early in the early stages of a design.

The following proposed changes would imply the following:

(1) The non-modport variables shall now default to inputs.
(2) All variables can be changed to the existing ref-port behavior by adding the keywords "default ref;" anywhere inside of an interface (reasoning explained below).
(Note: Dave Rich and Stu Sutherland did not necessarily like the "default ref;" keywords because users could become confused and think that all variable and net ports both inside and outside of modports, would be changed to ref-ports, but neither one could think of better simple wording - if a better interface command is discovered, they will submit it and I do not have any particular allegiance to "default ref;"). Second option: "default ref logic;" (Logic variables default to ref ports).
(3) modport port declarations override the default implicit variable-inputs or "default ref" variable ref-ports for the specified variable ports within the specified modport.
(4) Stu proposed and Dave and I liked the idea of allowing optional default direction declarations on each variable and net in an interface (reasoning explained below).
Since most interfaces will be built with at least two modports, one to specify "receiver" port directions and another to specify "sender" port directions, allowing optional port directions on the interface variables themselves could reduce the number of required modports by one. Now a user could define the interface to have "receiver" defaults and then define a "sender" modport.

Included below:
Ref ports are introduced in section 5.6 (pg. 42 - 3rd paragraph) - no changes required.
Proposal to change wording in section 9.5 (pg. 64 - 2nd paragraph)
Proposal to change wording in section 18.8.1 (pg. 201 - 5th bullet)
Proposal to change wording in section 19.2.2 (pg. 206 - 1st paragraph)
Proposal to change wording in section 19.4 (pg. 210 - paragraph just before section 19.4.1)
Proposal to change the example in section 19.4.1 (pg. 210)
Proposal to change the example in section 19.4.2 (pg. 211)
Proposal to change the example in section 19.4.3 (pg. 212)
Proposal to change the example in section 19.5.2 (pg. 214)
Proposal to change the example in section 19.5.3 (pg. 215)
Proposal to change the example in section 19.5.4 (pg. 217)
Proposal to change the example in section 19.6 (pg. 219)
Proposal to augment the index with additional "ref" page references.

I would like to discuss and vote on these proposals at an upcomong sv-bc meeting.

Regards - Cliff

Proposal to change wording in section 9.5 (pg. 64 - 2nd paragraph)

Reason: *typo* and ref-ports are new, their behavior is unknown and unusual to new and existing Verilog users, so the ref-port behavior needs to be more explicitly and unambiguously described.

Section 5.6 Continuous Assignments - 2nd paragraph.

WAS: SystemVerilog removes this restriction, and permits continuous assignments to drive nets any type of variable. Nets can be driven by multiple continuous assignments, or a mixture of primitives and continuous assignments. Variables can only be driven by one continuous assignment or one primitive output. It shall be an error for a variable driven by a continuous assignment or primitive output to have an initializer in the declaration or any procedural assignment. See also Section 5.6.

PROPOSED: SystemVerilog removes this restriction, and permits continuous assignments to drive nets ***and*** any type of variable. Nets can be driven by multiple continuous assignments, or a mixture of primitives and continuous assignments. Variables can only be driven by one continuous assignment or one primitive output. ***This also means that a variable cannot be driven by two or more continuous assignments from ref ports through two or more different modules to a common variable, nor can the common variable be driven by a continuous assignment through a ref port and by any other procedural assignment, including a procedural assignment through another ref port.*** It shall be an error for a variable driven by a continuous assignment or primitive output to have an initializer in the declaration or any procedural assignment. See also Section 5.6.

Proposal to change wording in section 18.8.1 (pg. 201 - 5th bullet)

Reason: *typo* and ref-ports are new, their behavior is unknown and unusual to new and existing Verilog users, so I believe the following clarifications are required to disambiguate the continuous assignment behavior to a ref port.

Section 18.8.1 Port connection rules for variables

WAS: — A ref port shall be connected to an equivalent variable data type. References to the port variable shall be treated as hierarchal references to the variable it is connected to in its instantiation. This kind of port can not be left unconnected.

PROPOSED: — A ref port shall be connected to an equivalent variable data type. References to the port variable shall be treated as ***hierarchical*** references to the variable it is connected to in its instantiation. ***If a continuous assignment is made to a ref-port connected variable data type, no other continuous or procedural assignment can be made to the variable data type, even if the assignments were made through other ref ports.*** This kind of port can not be left unconnected ***because a ref port is an implicit hierarchical connection to a variable at a higher scope***.

Proposal to change wording in section 19.2.2 (pg. 206 - 1st paragraph)

Reason: Change the default behavior of an interface variable that is not included in a modport. Again, ref ports are confusing to most hardware design engineers, so changing the default to be an input will force engineers to either use the interface variables as inputs, make the proposed "default ref;" declaration inside of an interface (the engineer has to intentionally shoot themselves in the foot), or add a modport declaration to allow procedural or driver output assignments.

WAS: The simplest form of a SystemVerilog interface is a bundled collection of variables or nets. When an interface is used as a port, the variables and nets in it are assumed to be **`ref`** and **`inout`** ports, respectively. The following interface example shows the basic syntax for defining, instantiating and connecting an interface. Usage of the SystemVerilog interface capability can significantly reduce the amount of code required to model port connections.

PROPOSED: The simplest form of a SystemVerilog interface is a bundled collection of variables or nets. When an interface is used as a port, the variables and nets in it are assumed to be **`input`** and **`inout`** ports, respectively. The following interface example shows the basic syntax for defining, instantiating and connecting an interface. Usage of the SystemVerilog interface capability can significantly reduce the amount of code required to model port connections.

Proposal to change wording in section 19.4 (pg. 210 - paragraph just before section 19.4.1)

Reason:

WAS: Note that if no **modport** is specified in the module header or in the port connection, then all the nets and variables in the interface are accessible with direction **inout** or **ref**, as in the examples above.

PROPOSED: Note that if no **modport** is specified in the module header or in the port connection, then all the nets and variables in the interface are accessible with direction **inout** or **input**, as in the examples above.

Proposal to change the example in section 19.4.1 (pg. 210)

Reason: It is more likely that a data bus would be a synthesizable bi-directional port. By Dave Rich's own admission, Dave made a global substitution from inout to ref in the SystemVerilog document. I believe the previous inout ports are better.

WAS:

```
interface simple_bus (input bit clk); // Define the interface
  logic req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic start, rdy;

  modport slave (input req, addr, mode, start, clk,
                 output gnt, rdy,
                 ref data);

  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start,
                 ref data);
endinterface: simple_bus
```

PROPOSED:

```
interface simple_bus (input bit clk); // Define the interface
  logic      req, gnt;
  logic [7:0] addr;
  logic [1:0] mode;
  logic      start, rdy;
  wire  [7:0] data;

  modport slave (input req, addr, mode, start, clk,
                 output gnt, rdy,
                 inout data);

  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start,
                 inout data);
endinterface: simple_bus
```

Proposal to change the example in section 19.4.2 (pg. 211)

Reason: It is more likely that a data bus would be a synthesizable bi-directional port

WAS:

```
interface simple_bus (input bit clk); // Define the interface
  logic req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic start, rdy;

  modport slave (input req, addr, mode, start, clk,
                 output gnt, rdy,
                 ref data);
```

```
  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start,
                 ref data);
endinterface: simple_bus
```

PROPOSED:

```
interface simple_bus (input bit clk); // Define the interface
  logic      req, gnt;
  logic [7:0] addr;
  logic [1:0] mode;
  logic      start, rdy;
  wire  [7:0] data;

  modport slave (input req, addr, mode, start, clk,
                 output gnt, rdy,
                 inout data);

  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start,
                 inout data);
endinterface: simple_bus
```

Proposal to change the example in section 19.4.3 (pg. 212)

Reason: It is more likely that a data bus would be a synthesizable bi-directional port

WAS:

```
interface simple_bus (input bit clk); // Define the interface
  logic      req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic      start, rdy;

modport slave (input req, addr, mode, start, clk,
               output gnt, rdy,
               ref data);

modport master(input gnt, rdy, clk,
               output req, addr, mode, start,
               ref data);
endinterface: simple_bus
```

PROPOSED:

```
interface simple_bus (input bit clk); // Define the interface
  logic      req, gnt;
  logic [7:0] addr;
  logic [1:0] mode;
  logic      start, rdy;
  wire  [7:0] data;

modport slave (input req, addr, mode, start, clk,
               output gnt, rdy,
               inout data);

modport master(input gnt, rdy, clk,
               output req, addr, mode, start,
               inout data);
endinterface: simple_bus
```

Proposal to change the example in section 19.5.2 (pg. 214)

Reason: It is more likely that a data bus would be a synthesizable bi-directional port

WAS:

```
interface simple_bus (input bit clk); // Define the interface
  logic req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic start, rdy;

  modport slave (input req, addr, mode, start, clk,
              output gnt, rdy,
              ref data,
              import task slaveRead(),
                    task slaveWrite());
          // import into module that uses the modport

  modport master(input gnt, rdy, clk,
              output req, addr, mode, start,
              ref data,
              import masterRead,
                    masterWrite);
          // import into module that uses the modport
```

…

PROPOSED:

```
interface simple_bus (input bit clk); // Define the interface
  logic      req, gnt;
  logic [7:0] addr;
  logic [1:0] mode;
  logic      start, rdy;
  wire  [7:0] data;

  modport slave (input req, addr, mode, start, clk,
              output gnt, rdy,
              inout data,
              import task slaveRead(),
                    task slaveWrite());
          // import into module that uses the modport

  modport master(input gnt, rdy, clk,
              output req, addr, mode, start,
              inout data,
              import masterRead,
                    masterWrite);
          // import into module that uses the modport
```

…

Proposal to change the example in section 19.5.3 (pg. 215)

Reason: It is more likely that a data bus would be a synthesizable bi-directional port

WAS:

```
interface simple_bus (input bit clk); // Define the interface
  logic req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic start, rdy;

  modport slave( input req, addr, mode, start, clk,
              output gnt, rdy,
              ref data,
              export task Read(),
                    task Write());
             // export from module that uses the modport
```

```
  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start,
                 ref data,
                 import task Read(input logic [7:0] raddr),
                        task Write(input logic [7:0] waddr));
              // import requires the full task prototype
endinterface: simple_bus
```

PROPOSED:

```
interface simple_bus (input bit clk); // Define the interface
  logic      req, gnt;
  logic [7:0] addr;
  logic [1:0] mode;
  logic      start, rdy;
  wire  [7:0] data;

  modport slave( input req, addr, mode, start, clk,
                 output gnt, rdy,
                 inout data,
                 export task Read(),
                        task Write());
                // export from module that uses the modport

  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start,
                 inout data,
                 import task Read(input logic [7:0] raddr),
                        task Write(input logic [7:0] waddr));
              // import requires the full task prototype
endinterface: simple_bus
```

Proposal to change the example in section 19.5.4 (pg. 217)

Reason: It is more likely that a data bus would be a synthesizable bi-directional port

WAS:

```
interface simple_bus (input bit clk); // Define the interface
  logic req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic start, rdy;
  int slaves = 0;

   // tasks executed concurrently as a fork/join block
  extern forkjoin task countSlaves();
  extern forkjoin task Read (input logic [7:0] raddr);
  extern forkjoin task Write (input logic [7:0] waddr);

  modport slave  (input req,addr, mode, start, clk,
                  output gnt, rdy,
                  ref data, slaves,
                  export Read, Write, countSlaves);
               // export from module that uses the modport

  modport master (input gnt, rdy, clk,
                  output req, addr, mode, start,
                  ref data,
                  import task Read(input logic [7:0] raddr),
                  task Write(input logic [7:0] waddr));
               // import requires the full task prototype
...
```

PROPOSED:

```
interface simple_bus (input bit clk); // Define the interface
  logic      req, gnt;
  logic [7:0] addr;
  logic [1:0] mode;
  logic      start, rdy;
  wire  [7:0] data;
  int slaves = 0;

  // tasks executed concurrently as a fork/join block
  extern forkjoin task countSlaves();
  extern forkjoin task Read (input logic [7:0] raddr);
  extern forkjoin task Write (input logic [7:0] waddr);

  modport slave  (input req,addr, mode, start, clk,
                  output gnt, rdy,
                  inout data, slaves,
                  export Read, Write, countSlaves);
             // export from module that uses the modport

  modport master (input gnt, rdy, clk,
                  output req, addr, mode, start,
                  inout data,
                  import task Read(input logic [7:0] raddr),
                  task Write(input logic [7:0] waddr));
             // import requires the full task prototype
...
```

Proposal to change the example in section 19.6 (pg. 219)

Reason: It is more likely that a data bus would be a synthesizable bi-directional port

WAS:

```
interface simple_bus #(parameter AWIDTH = 8, DWIDTH = 8;)
                      (input bit clk); // Define the interface
  logic req, gnt;
  logic [AWIDTH-1:0] addr;
  logic [DWIDTH-1:0] data;
  logic [1:0] mode;
  logic start, rdy;

  modport slave (input req, addr, mode, start, clk,
                 output gnt, rdy,
                 ref data,
                 import task slaveRead(),
                        task slaveWrite());
            // import into module that uses the modport

  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start,
                 ref data,
                 import task masterRead(input logic [AWIDTH-1:0] raddr),
                        task masterWrite(input logic [AWIDTH-1:0] waddr));
            // import requires the full task prototype
...
```

PROPOSED:

```
interface simple_bus #(parameter AWIDTH = 8, DWIDTH = 8;)
                      (input bit clk); // Define the interface
  logic           req, gnt;
  logic [AWIDTH-1:0] addr;
  wire  [DWIDTH-1:0] data;
  logic       [1:0] mode;
```

```
  logic            start, rdy;

  modport slave (input req, addr, mode, start, clk,
                 output gnt, rdy,
                 inout data,
                 import task slaveRead(),
                        task slaveWrite());
            // import into module that uses the modport

  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start,
                 inout data,
                 import task masterRead(input logic [AWIDTH-1:0] raddr),
                        task masterWrite(input logic [AWIDTH-1:0] waddr));
            // import requires the full task prototype
...
```

Proposal to augment the index with additional "ref" page references.

Reason: The index needs to reference the ref port sections better.

WAS: ref **74**

PROPOSED: ref **42, 64, 74, 201, 206, 210**

==========

For the record, we have defined a ref-port without really describing why or how they should be used. No real good examples of ref-port usage is shown in the standard.