Interface Section Modification Proposals - 20011217

Section 13.2.2 Interface example using a named wire bundle

Proposal: Convert the first /* */ multi-line comment into the first paragraph of section 13.2.2, with the following additional description (if the description is accurate). Also remove the inflated reference to 20 lines of savings due to the fact that the previous example declared many of the ports on different lines that were combined for the example of section 13.2.2 (not a fair comparison and engineers know it - reduces the credibility of the entire argument):

The simplest form of a SystemVerilog interface is a bundled collection of ports defined as simple data types. In the absence of assigned port directions, each of the ports is assumed to be an inout port. ~~This~~ The following interface example shows the basic syntax for defining, instantiating and connecting an interface. ~~Less text is needed than before (20 lines).~~Usage of the SystemVerilog interface capability can significantly reduce the amount of code required to model port connections.

Proposal: In the module **memMod**, name the interface "**a**" instead of "**b**."
Proposal: Also inside the **memMod** module, change all **b.req** and **b.gnt** references to **a.req** and **a.gnt**.

Reason: To insure that readers do not think that connecting interfaces between **memMod** and **cpuMod** require interface usage to use the same interface name in connected modules. To insure that readers do not think that the **b.req** and **b.gnt** references refer to a hierarchical reference in the **cpuMod** module (that also has a **simple_bus** interface named "**b**."

Proposal: In the top module, show named port and interface connections:

```
memMod mem (.a(sb_intf), .clk(clk));
cpuMod cpu (.b(sb_intf), .clk(clk));
```

Reason: The named connections more clearly show that the **sb_intf** instantiation of the **simple_bus** connects to the "**a**" and "**b**" interfaces in the sub-modules and that the two submodules are connected through the **sb_intf** interface complex-bus. This is also another reason not to use "**b**" for the interface names in both sub-modules, because someone reading this documentation might think that both **sb_intf** and "**b**" must be common to both connected modules, which is not true.

Proposal: Add the following clarification and example at the end of section 13.2.2:
NOTE: This proposal assumes passage of the implicit port connection proposals emailed on 12/13/01.

Note: If the same identifier, **sb_intf**, had been used to name the **simple_bus** interface in the **memMod** and **cpuMod** module headers, implicit port declarations also could have been used to instantiate the **memMod** and **cpuMod** modules into the **top** module, as shown below:

```
module memMod (simple_bus sb_intf, input bit clk);
  ... // the rest of the memMod code inserted here
endmodule

module cpuMod (simple_bus sb_intf, input bit clk);
  ... // the rest of the cpuMod code inserted here
endmodule

module top;
  logic clk = 0;

  simple_bus sb_intf;

  initial repeat(10) #10 clk++;
```

```
  memMod mem (.*); // implicit port connections
  cpuMod cpu (.*); // implicit port connections
endmodule
```

Section 13.2.3 Interface example using a generic wire bundle

Proposal: Convert the first /* */ multi-line comment into the first paragraph of section 13.2.3, with the following additional description (if the description is accurate):

A module header can be created with an unspecified interface instantiation as a place-holder for an interface to be selected when the module itself is instantiated. The unspecified interface is called a "generic" interface port. ~~This~~ The following interface example shows how to ~~use~~ specify a "generic" ~~"~~interface~~"~~ port in a module definition:

Proposal: Replace the internal logic assignments in the **memMod** and **cpuMod** modules with a comment of the form **"// RTL code for the XXX module**"

Reason: The internal code for these modules is pretty whimpy and is not important to the concept being presented in this section. The code was already shown once in the previous section. The **memMod** assignments reference ports that might not exist in a generic interface. No need to confuse the reader with these unimportant details.

Proposal: Place the interface code after the memMod and cpuMod code.

Reason: We don't want the reader to make the association that the generic interface is the interface that immediately preceded the generic reference in the code. Listing **memMod** and **cpuMod** first helps to show that they were defined without regard to any previously existing interface declaration.

Proposal: In the module **memMod**, name the interface "**a**" instead of "**b**."
Proposal: In the top module, again show named port and interface connections for clarification:

```
memMod mem (.a(sb_intf), .clk(clk));
cpuMod cpu (.b(sb_intf), .clk(clk));
```

Proposal: Implement the above code changes as shown below:

```
// memMod and cpuMod can use any interface
module memMod (interface a, input bit clk);
  // RTL code for the memMod module
endmodule

module cpuMod (interface b, input bit clk);
  // RTL code for the cpuMod module
endmodule

interface simple_bus; // Define the interface
  logic       req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic       start, rdy;
endinterface: simple_bus

module top;
  logic clk = 0;

  simple_bus sb_intf; // Instantiate the interface

  initial repeat(10) #10 clk++;
```

```
  // Connect the sb_intf instance of the simple_bus
  // interface to the generic interfaces of the
  // memMod and cpuMod modules
  memMod mem (.a(sb_intf), .clk(clk));
  cpuMod cpu (.b(sb_intf), .clk(clk));
endmodule
```

Proposal: Add the following clarification and example at the end of section 13.2.3:
NOTE: This proposal assumes passage of the implicit port connection proposals emailed on 12/13/01.

Note: An implicit port cannot be used to connect to a generic interface. A named port must be used to connect to a generic interface as shown below:

```
module memMod (interface a, input bit clk);
  ... // the rest of the memMod code inserted here
endmodule

module cpuMod (interface b, input bit clk);
  ... // the rest of the cpuMod code inserted here
endmodule

module top;
  logic clk = 0;

  simple_bus sb_intf;

  initial repeat(10) #10 clk++;

  memMod mem (.*, .a(sb_intf)); // partial implicit port connections
  cpuMod cpu (.*, .b(sb_intf)); // partial implicit port connections
endmodule
```

Section 13.3 Ports in interfaces

Proposal: Clarify the beginning of section 13.3, with the following additional description (if the description is accurate):

One limitation to simple interfaces is that the nets and variables declared within the interface are only used to connect to another interface with the same nets and variables. To share an external wire or variable, one that makes a connection from outside of the interface and forms a common connection to all modules that instantiate the interface, an interface port declaration is required. ~~To share a wire or variable (e.g. between two interface instances), there are ports to the interface declaration, like a module declaration.~~ The difference between wires in the interface port list and other wires within the interface is that only the wires in the port list can be connected externally by name or position when the interface is instantiated.

```
interface i1 (input a, output b, inout c);
  wire d;
endinterface
```

The wires a, b and c can be individually connected to the interface and thus shared with other interfaces.

Proposal: Convert the /* */ multi-line comment for the example in section 13.3 into a text paragraph with minor changes:

~~This interface~~The following example shows how to specify an interface with inputs, allowing a wire to be shared between two instances of the interface.

Proposal: In the module **memMod**, name the interface "**a**" instead of "**b**."

Proposal: Also inside the **memMod** module, change all **b.req** and **b.gnt** references to **a.req** and **a.gnt**.
Proposal: Replace the internal logic assignments in the **memMod** and **cpuMod** modules with a comment of the form **"// RTL code for the XXX module**"

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

module memMod(simple_bus a); // Uses just the interface
  always @(posedge a.clk)    // the clk port from the interface
  // RTL code for the memMod module
endmodule

module cpuMod(simple_bus b);
endmodule

module top;
  logic clk = 0;

  // instantiate two copies of the simple_bus interface
  // with a common connection to the top.clk varible
  simple_bus sb_intf1(.clk(clk));
  simple_bus sb_intf2(.clk(clk));

  initial repeat(10) #10 clk++;

  memMod mem1(.a(sb_intf1)); // Connect bus 1 to memory 1
  cpuMod cpu1(.b(sb_intf1));
  memMod mem2(.a(sb_intf2)); // Connect bus 2 to memory 2
  cpuMod cpu2(.b(sb_intf2));
endmodule
```

Proposal: Add the following clarification concerning implicit port connections to the end of section 13.3:
NOTE: This proposal assumes passage of the implicit port connection proposals emailed on 12/13/01.

Note: Because the instantiated interface names do not match the interface names used in the **memMod** and **cpuMod** modules, implicit port connections cannot be used for this example.

Section 13.4 Modports

Proposal: Modify the first paragraph of section 13.4, as follows:

To bundle module ports there are **modport** lists with directions declared within the interface. The keyword **modport** ~~shows~~ indicates that the directions are ~~viewed from~~declared inside the module.

```
interface i2;
  wire a, b, c, d;
  modport master (input a, b, output c, d);
  modport slave (output a, b, input c, d);
endinterface
```

The **modport** list name (master or slave) can be specified in the module header, where the **modport** name acts as a direction and the interface name as a type:

Proposal: In the top module (add top modules to reduce confusion), again show named port and interface connections for clarification:

```
module m (i2.master i);
endmodule

module s (i2.slave i);
endmodule

module top;
  i2 i;

  m u1(.i(i));
  s u2(.i(i));
endmodule
```

The **modport** list name (master or slave) can also be specified in the port connection with the module instance, where the **modport** name is hierarchical from the interface instance:

```
module m (i2 i);
endmodule

module s (i2 i);
endmodule

module top;
  i2 i;

  m u1 (.i(i.master));
  s u2 (.i(i.slave));
endmodule
```

The syntax of `interface_name.modport_name instance_name` is really a hierarchical type followed by an instance. Note that this can be generalized to any interface with a given **modport** name by writing **interface.**–`modport_name instance_name`.