Subject: PASSED - Implicit Ports Proposals

For your information, attached is the amended Positional Ports Proposal that passed in the April 15th SystemVerilog Conference call.

Cleanup and clarification wording is still permitted but not anticipated at this time.

PASSED #1 - Replace section 12.7 of draft 5 with the implicit ports proposal
PASSED #2 - Accept Cliff Cummings' December e-mail additions to section 13 as already incorporated in the draft 5 document.

Regards - Cliff

PROPOSAL #1: Replace section 12.7 of draft 5 with the implicit ports proposal as shown:

12.7 Module instances

BNF section A.4.1.1 - Module instantiation

```
module_instance ::= name_of_instance ( [ list_of_port_connections ] )

name_of_instance ::= module_instance_identifier { range }

list_of_port_connections ::=
        ordered_port_connection { , ordered_port_connection }
      | dot_named_port_connection { , dot_named_port_connection }
      | { named_port_connection , } dot_star_port_connection
        { , named_port_connection }

ordered_port_connection ::= { attribute_instance } [ expression ]

named_port_connection ::= { attribute_instance } .port_identifier ( [ expression ] )

dot_named_port_connection ::=
        { attribute_instance } .port_identifier
      | named_port_connection

dot_star_port_connection ::= { attribute_instance } .*
```

Syntax 12-20—Module instance syntax

A module can be used (instantiated) in two ways, hierarchical or top level. Hierarchical instantiation allows more than one instance of the same type. The module name can be a module previously declared or one declared later. Actual parameters can be named or ordered. Port connections can be named, ordered or implicitly connected. They can be nets, variables, or other kinds of interfaces, events, or expressions. See below for the connection rules.

Consider an alu-accumulator (**`alu_accum`**) example module that includes instantiations of an **`alu`** module, an accumulator register (**`accum`**) module and a sign-extension (**`xtend`**) module. The module headers for the three instantiated modules are shown in the following example code:

```
module alu (
  output reg [7:0] alu_out,
  output reg       zero,
  input      [7:0] ain, bin,
  input      [2:0] opcode);

  // RTL code for the alu module
```

```
endmodule

module accum (
  output reg [7:0] dataout,
  input      [7:0] datain,
  input            clk, rst_n);

  // RTL code for the accumulator module
endmodule

module xtend (
  output reg [7:0] dout,
  input            din,
  input            clk, rst_n);

  // RTL code for the sign-extension module
endmodule
```

12.7.1 Instantiation using positional port connections

Verilog has always permitted instantiation of modules using positional port connections as shown in the **alu_accum1** module example.

```
module alu_accum1 (
  output [15:0] dataout,
  input   [7:0] ain, bin,
  input   [2:0] opcode,
  input         clk, rst_n);

  wire    [7:0] alu_out;

  alu   alu   (alu_out, , ain, bin, opcode);

  accum accum (dataout[7:0], alu_out, clk, rst_n);

  xtend xtend (dataout[15:8], alu_out[7], clk, rst_n);

endmodule
```

As long as the connecting variables are ordered correctly and are the same size as the instance-ports that they are connected to, there will be no warnings and the simulation will work as expected.

12.7.2 Instantiation using named port connections

Verilog has always permitted instantiation of modules using named port connections as shown in the **alu_accum2** module example.

```
module alu_accum2 (
  output [15:0] dataout,
  input   [7:0] ain, bin,
  input   [2:0] opcode,
  input         clk, rst_n);

  wire    [7:0] alu_out;

  alu   alu   (.alu_out(alu_out), .zero(),
               .ain(ain), .bin(bin), .opcode(opcode));

  accum accum (.dataout(dataout[7:0]), .datain(alu_out),
               .clk(clk), .rst_n(rst_n));
```

```
  xtend xtend (.dout(dataout[15:8]), .din(alu_out[7]),
               .clk(clk), .rst_n(rst_n));

endmodule
```

Named port connections do not have to be ordered the same as the ports of the instantiated module. The variables connected to the instance ports must be the same size or a port-size mismatch warning will be reported.

12.7.3 Instantiation using implicit .name port connections

SystemVerilog adds the capability to implicitly instantiate ports using a **.name** syntax if the instance-port name and size match the connecting variable-port name and size. This enhancement eliminates the requirement to list a port name twice when the port name and signal name are the same, while still listing all of the ports of the instantiated module for documentation purposes.

In the following **alu_accum3** example, all of the ports of the instantiated **alu** module match the names of the variables connected to the ports, except for the unconnected **zero** port, which is listed using a named port connection, showing that the port is unconnected. Implicit **.name** port connections are made for all name and size matching connections on the instantiated module.

In the same **alu_accum3** example, the accum module has an 8-bit port called **dataout** that is connected to a 16-bit bus called **dataout**. Because the internal and external sizes of **dataout** do not match, the port must be connected by name, showing which bits of the 16-bit bus are connected to the 8-bit port. The **datain** port on the **accum** is connected to a bus by a different name (**alu_out**), so this port is also connected by name. The **clk** and **rst_n** ports are connected using implicit **.name** port connections.

Also in the same **alu_accum3** example, the **xtend** module has an 8-bit output port called **dout** and a 1-bit input port called **din**. Since neither of these port names match the names (or sizes) of the connecting variables, both are connected by name. The **clk** and **rst_n** ports are connected using implicit **.name** port connections.

```
module alu_accum3 (
  output [15:0] dataout,
  input   [7:0] ain, bin,
  input   [2:0] opcode,
  input         clk, rst_n);

  wire    [7:0] alu_out;

  alu   alu   (.alu_out, .zero(), .ain, .bin, .opcode);

  accum accum (.dataout(dataout[7:0]), .datain(alu_out), .clk, .rst_n);

  xtend xtend (.dout(dataout[15:8]), .din(alu_out[7]), .clk, .rst_n);

endmodule
```

Implicit **.name** port connections do not have to be ordered the same as the ports of the instantiated module.

The following rules apply to implicit **.name** port connections:

- For an implicit **.name** port connection to be legal, the connecting variable name must match the port name of the instantiated module.
- For an implicit **.name** port connection to be legal, the connecting variable size must match the port size of the instantiated module.

- For an implicit **.name** port connection to be legal, the connecting variable data type must be compatible to the port data type of the instantiated module. See section 12.7.5 for a description of compatible data types for implicit port connections.
- Implicit **.name** port connections cannot be used in the same instantiation with positional port connections.
- Implicit **.name** port connections may be used in the same instantiation with named port connections.
- Implicit **.name** port connections cannot be used in the same instantiation with implicit **.*** port connections.
- The order of the implicit **.name** port connections does not have to match the port-order of the instantiated module.
- All connecting variables must be explicitly declared, either as a port in the parent module (following the rules of Verilog-2001) or as an explicit net or variable of one or more bits.

12.7.4 Instantiation using implicit .* port connections

SystemVerilog adds the capability to implicitly instantiate ports using a **.*** syntax for all ports where the instance-port name and size match the connecting variable-port name and size. This enhancement eliminates the requirement to list any port where the name and size of the connecting variable match the name and size of the instance port. This implicit port connection style is used to indicate that all port names and sizes match the connections where emphasis is placed only on the exception ports. The implicit **.*** port connection syntax can greatly facilitate rapid block-level testbench generation where all of the testbench variables are chosen to match the instantiated module port names and sizes.

In the following **alu_accum4** example, all of the ports of the instantiated alu module match the names of the variables connected to the ports, except for the unconnected **zero** port, which is listed using a named port connection, showing that the port is unconnected. The implicit **.*** port connection syntax connects all other ports on the instantiated module.

In the same **alu_accum4** example, the accum module has an 8-bit port called **dataout** that is connected to a 16-bit bus called **dataout**. Because the internal and external sizes of **dataout** do not match, the port must be connected by name, showing which bits of the 16-bit bus are connected to the 8-bit port. The **datain** port on the **accum** is connected to a bus by a different name (**alu_out**), so this port is also connected by name. The **clk** and **rst_n** ports are connected using implicit **.*** port connections.

Also in the same **alu_accum4** example, the **xtend** module has an 8-bit output port called **dout** and a 1-bit input port called **din**. Since neither of these port names match the names (or sizes) of the connecting variables, both are connected by name. The **clk** and **rst_n** ports are connected using implicit **.*** port connections.

```
module alu_accum4 (
  output [15:0] dataout,
  input   [7:0] ain, bin,
  input   [2:0] opcode,
  input         clk, rst_n);

  wire    [7:0] alu_out;

  alu   alu   (.*, .zero());

  accum accum (.*, .dataout(dataout[7:0]), .datain(alu_out));

  xtend xtend (.*, .dout(dataout[15:8]), .din(alu_out[7]));

endmodule
```

The following rules apply to implicit **.*** port connections:

- For an implicit **.*** port connection to be legal, all implicitly connected ports must have a connecting variable name to match the port name of the instantiated module.
- For an implicit **.*** port connection to be legal, all implicitly connected ports must have a connecting variable size to match the port size of the instantiated module.
- For an implicit **.*** port connection to be legal, the connecting variable data type must be compatible to the port data type of the instantiated module. See section 12.7.5 for a description of compatible data types for implicit port connections.
- Implicit **.*** port connections cannot be used in the same instantiation with positional port connections.
- Implicit **.*** port connections may be used in the same instantiation with named port connections.
- Implicit **.*** port connections cannot be used in the same instantiation with implicit **.name** port connections.
- If implicit **.*** port connections are used in an instantiation, all unconnected ports must be shown using named port connections.
- When the implicit **.*** port connection is mixed in the same instantiation with named port connections, the implicit **.*** port connection token can be placed anywhere in the port list.
- All connecting variables must be explicitly declared, either as a port in the parent module (following the rules of Verilog-2001) or as an explicit net or variable of one or more bits.

Modules may be instantiated into the same parent module using any combination of legal positional, named, implicit .name connected and implicit **.*** connected instances as shown in **alu_accum5** example.

```
module alu_accum5 (
  output [15:0] dataout,
  input   [7:0] ain, bin,
  input   [2:0] opcode,
  input         clk, rst_n);

  wire    [7:0] alu_out;

  // mixture of named port connections and
  // implicit .name port connections
  alu   alu   (.ain(ain), .bin(bin), .alu_out, .zero(), .opcode);

  // positional port connections
  accum accum (dataout[7:0], alu_out, clk, rst_n);

  // mixture of named port connections and
  // implicit .* port connections
  xtend xtend (.dout(dataout[15:8]), .*, .din(alu_out[7]));

endmodule
```

12.7.5 Compatible data types for implicit port connections

Implicit port connections are permitted between any two data types that are allowed by SystemVerilog port connection rules, as long as the SystemVerilog simulator is not required to report a warning about the connection. Any SystemVerilog instantiation that would cause a warning to be issued must be connected by name if other ports of the instance are instantiated using an implicit port connection style.

If for example a top-level module connects a signal named **net1** of any data type to an instantiated submodule with a port also named **net1** of same data type, SystemVerilog will run this simulation without warning because the data types are the same across ports. It is legal to make this type of connection using an implicit port connection style.

If for example a top-level module connects a signal named **net2** of type **wire** to an instantiated submodule with a port also named **net2** of type **reg**, Verilog simulators run this simulation without

warning because the data types are compatible across ports. It is legal to make this type of connection using an implicit port connection style.

If for example a top-level module connects a signal named **`net3`** of type **`tri1`** to an instantiated submodule with a port named **`net3`** of type **`tri0`**, Verilog simulators issue a warning and the top-level data type (**`tri1`**) is used during simulation, as described in the IEEE Verilog-2001 Standard. It is legal to make this type of connection using named port connections but it shall be a syntax error to make this connection using an implicit port connection style. Any port connection that results in a required warning message shall not be permitted to be instantiated using an implicit port connection style.

A top-level module shall not implicitly connect a signal of any data type to a port by the same name of another data type if connecting the data types is illegal as defined by this SystemVerilog standard.

PROPOSAL #2 - Accept Cliff Cummings' December e-mail additions to section 13 as already incorporated in the draft 5 document.
Modify the first paragraph of section 13.1, page 56 of draft 5 as shown:
Keep the implicit port explanation and example shown in section 13.2.2 on page 59 of Draft 5.
Keep the implicit port explanation and example shown in section 13.2.3 on page 60 of Draft 5.
Keep the implicit port explanation shown in section 13.3 on page 61 of Draft 5.

The communication between blocks of a digital system is a critical area that can affect everything from ease of RTL coding, to hardware-software partitioning, to performance analysis, to bus implementation choices and protocol checking. The interface construct in SystemVerilog was created specifically to encapsulate the communication between blocks, allowing a smooth migration from abstract system-level design through successive refinement down to lower-level register-transfer and structural views of the design. By encapsulating the communication between blocks, the interface construct also facilitates design re-use. The inclusion of interface capabilities is one of the major advantages of SystemVerilog over Verilog-2001.