Proposal: Change the name of section 13 to:
Implicit Ports and Interfaces

Proposal: Changes to section 13.1 (highlighted):

The instantiation and 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 .name and .* implicit port instantiation capabilities of SystemVerilog greatly simplify the task of instantiating large logic blocks into a higher level model if the port name matches the name of the connecting net. 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 ~~such a useful language feature~~interface capabilities is one of the major advantages of SystemVerilog over Verilog-2001.

Proposal: Move the rest of section 13.1 to an interface-introduction section after a newly inserted section 13.2 to introduce and explain implicit port instantiation

Proposal: Insert a new section 13.2 to describe implicit port instantiations as shown below:
13.2 Implicit .name Port Instantiation Syntax

Proposal: Insert a new section 13.2 to describe implicit port instantiations as shown below:
13.2 Implicit .* Port Instantiation Syntax

Proposal: Change and show in section 13.2, the following BNF production (based on the Verilog-2001 BNF) for port_connections:

A.4.1 Module instantiation

```
module_instantiation ::=
        module_identifier [ parameter_value_assignment ]
           module_instance { , module_instance } ;
parameter_value_assignment ::= # ( list_of_parameter_assignments )
list_of_parameter_assignments ::=
        ordered_parameter_assignment { , ordered_parameter_assignment }
      | named_parameter_assignment { , named_parameter_assignment }
ordered_parameter_assignment ::= expression
named_parameter_assignment ::= .parameter_identifier ( [ expression ] )
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 }
      | named_port_connection { , named_port_connection }
      | dot_port_connection { { , named_port_connection }{ , dot_port_connection } }
      | dot_star_connection { , named_port_connection }
ordered_port_connection ::= { attribute_instance } [ expression ]
named_port_connection ::= { attribute_instance } .port_identifier ( [ expression ] )
dot_port_connection ::= { attribute_instance } .port_identifier
dot_star_connection ::= { attribute_instance } .*
```

Probably needs to include named_interface connection and ordered_interface_connection

Proposed wording for the new section 13.1 Implicit Port Instantiation Syntax:

(NOTE: the example and much of the wording for this part of the proposal came straight from one of my HDLCON-2002 papers - If there is a problem related to this material being published elsewhere, then we need to scrap this example and text and come up with an independent example and description).

SystemVerilog introduces the capability of instantiating modules with highly abbreviated and efficient implicit port connections. Implicit port connections are intended to facilitate the process of instantiating large sub-blocks into upper-level modules without having to type multiple lines of named port connections where the sub-blocks are instantiated. Implicit port connections reduce the verbose nature of most higher-level modules by limiting the number of named ports that actually have to be listed when a module is instantiated. At the same time, since only those nets or busses that do not match a port name must be listed in the module instantiation, only those port connections that must be made to a dissimilarly sized or named net or bus are emphasized and not hidden in a sea of unremarkable named port connections.

With a careful naming convention, instantiating large logic blocks into a higher level module can now be greatly facilitated by using SystemVerilog implicit port connections.

SystemVerilog introduces the new port connection token: **`.*`**

When a sub-block is instantiated into a module, and if the sub-block port names match the size and name of the module nets or busses that are connected to the sub-block ports, implicit port connections can be made using the **`.*`** token.

When using the **`.*`** implicit port connection token, any sub-block port that does not match in size or name to the module net or bus connected to the port, shall be connected using a named-port connection. It shall not be permitted to mix implicit port connections (**`.*`** connections) with positional port connections.

Implicit port connections shall follow these rules:

- Implicit ports shall not be used in an instantiated sub-block with positional ports.
- Implicit ports may be used in an instantiated sub-block with named ports. For the purposes of this document, the term "implicit port declarations" may or may not include named port connections.
- It is permitted to have sub-block instantiations with positional ports and sub-block instantiations with named ports and sub-block instantiations with implicit ports all instantiated in the same upper-level module.
- If implicit port connections are used to instantiate a sub-block, the **`.*`** token must be placed first in the instantiated port list, before any other named ports, if any, are listed.
- The port name on an instantiated sub-block must match the net or bus name of the connecting module.
- The port size on an instantiated sub-block must match the net or bus size of the connecting module.
- Any individual port in an implicitly instantiated module that does not match both size and name of the net or bus of the upper-level module, shall be instantiated by name.
- If a port on an instantiated sub-block is unconnected in the upper-level module, the port shall be explicitly listed as a named port with empty parentheses showing there is no connection to the port.
- All nets or busses in the upper-level module that connect to implicit ports must either be explicitly declared as a scalar-net, vector-net, or as a port on the upper-level module.

Consider the example of a Complex Arithmetic Logic Unit (CALU) as shown in figure XXX

Figure XXX - Block Diagram of a Complex Arithmetic Logic Unit (CALU)

(A black & white version of this gif diagram will be provided if we pass this enhancement and decide to keep this example)

This CALU design has nine instantiated sub-blocks. If the sub-block module port lists are carefully named so that the port names match the names of the top-level CALU module nets or busses that are connected to the instantiated ports, implicit port connections can be used when the sub-blocks are instantiated into the CALU.

Below are the Verilog-1995-style module headers and port names for the nine sub-blocks that are instantiated into the CALU module:

```
module multop1 (mop1, data, ld_multop1, clk, rst_n);
  output [15:0] mop1;
  input  [15:0] data;
  input         ld_multop1, clk, rst_n;
  // RTL code for the multiplier operand1 register
endmodule

module multiplier (mult, mop1, data);
  output [31:0] mult;
  input  [15:0] mop1, data;
  // RTL code for the multiplier output register
endmodule

module multoutreg (multout, mult, ld_multout, clk, rst_n);
  output [31:0] multout;
  input  [31:0] mult;
  input         ld_multout, clk, rst_n;
  // RTL code for the multiplier output register
endmodule

module barrel_shifter (bs, data, bs_lshft, ld_bs, clk, rst_n);
  output [31:0] bs;
  input  [15:0] data;
  input  [ 4:0] bs_lshft;
  input         ld_bs, clk, rst_n;
  // RTL code for the barrel shifter
endmodule

module mux2 (y, i1, i0, sel1);
  output [31:0] y;
  input  [31:0] i1, i0;
  input         sel1;
  // RTL code for a 2-to-1 mux
endmodule

module alu (alu_out, zero, neg, alu_in, acc, alu_op);
  output [31:0] alu_out;
  output        zero, neg;
  input  [31:0] alu_in, acc;
  input  [ 2:0] alu_op;
  // RTL code for the ALU
endmodule

module accumulator (acc, alu_out, ld_acc, clk, rst_n);
  output [31:0] acc;
  input  [31:0] alu_out;
  input         ld_acc, clk, rst_n;
  // RTL code for the accumulator register
endmodule

module shifter (data, acc, shft_lshft, ld_shft, en_shft, clk, rst_n);
  output [15:0] data;
  input  [31:0] acc;
  input  [ 1:0] shft_lshft;
  input         ld_shft, en_shft, clk, rst_n;
  // RTL code for the shifter
endmodule

module tribuf (data, acc, en_acc);
  parameter SIZE=16;
```

```
  output [SIZE-1:0] data;
  input  [SIZE-1:0] acc;
  input             en_acc;
  // RTL code for the tristate buffer
endmodule
```

When these modules are instantiated into a Verilog-1995-style or Verilog-2001-style CALU module with named port connections, the correct CALU module code is shown below:

```
module calu1 (data, bs_lshft, alu_op, shft_lshft, calu_muxsel,
              en_shft, ld_acc, ld_bs, ld_multop1, ld_multout,
              ld_shft, en_acc, clk, rst_n);
  inout  [15:0] data;
  input  [ 4:0] bs_lshft;
  input  [ 2:0] alu_op;
  input  [ 1:0] shft_lshft;
  input         calu_muxsel, en_shft, ld_acc, ld_bs;
  input         ld_multop1, ld_multout, ld_shft, en_acc;
  input         clk, rst_n;
  wire   [31:0] acc, alu_in, alu_out, bs, mult, multout;
  wire   [15:0] mop1;

  multop1        multop1        (.mop1(mop1), .data(data),
                                  .ld_multop1(ld_multop1),
                                  .clk(clk), .rst_n(rst_n));
  multiplier     multiplier     (.mult(mult), .mop1(mop1),
                                  .data(data));
  multoutreg     multoutreg     (.multout(multout),
                                  .mult(mult),
                                  .ld_multout(ld_multout),
                                  .clk(clk), .rst_n(rst_n));
  barrel_shifter barrel_shifter (.bs(bs), .data(data),
                                  .bs_lshft(bs_lshft),
                                  .ld_bs(ld_bs),
                                  .clk(clk), .rst_n(rst_n));
  mux2           mux            (.y(alu_in),
                                  .i0(multout),
                                  .i1(acc),
                                  .sel1(calu_muxsel));
  alu            alu            (.alu_out(alu_out),
                                  .zero(), .neg(), .alu_in(alu_in),
                                  .acc(acc), .alu_op(alu_op));
  accumulator    accumulator    (.acc(acc), .alu_out(alu_out),
                                  .ld_acc(ld_acc), .clk(clk),
                                  .rst_n(rst_n));
  shifter        shifter        (.data(data), .acc(acc),
                                  .shft_lshft(shft_lshft),
                                  .ld_shft(ld_shft),
                                  .en_shft(en_shft),
                                  .clk(clk), .rst_n(rst_n));
  tribuf         tribuf         (.data(data), .acc(acc[15:0]),
                                  .en_acc(en_acc));
endmodule
```

Below are the Verilog-2001-style module headers and port names for the nine sub-blocks that are instantiated into the CALU module (note that the Verilog-1995-style sub-blocks would also work with implicit port instantiation):

```
module multop1 (
  output [15:0] mop1,
  input  [15:0] data,
  input         ld_multop1, clk, rst_n);
```

```
  // RTL code for the multiplier operand1 register
endmodule

module multiplier (
  output [31:0] mult,
  input  [15:0] mop1, data);
  // RTL code for the multiplier output register
endmodule

module multoutreg (
  output [31:0] multout,
  input  [31:0] mult,
  input         ld_multout, clk, rst_n);
  // RTL code for the multiplier output register
endmodule

module barrel_shifter (
  output [31:0] bs,
  input  [15:0] data,
  input  [ 4:0] bs_lshft,
  input         ld_bs, clk, rst_n);
  // RTL code for the barrel shifter
endmodule

module mux2 (
  output [31:0] y,
  input  [31:0] i1, i0,
  input         sel1);
  // RTL code for a 2-to-1 mux
endmodule

module alu (
  output [31:0] alu_out,
  output        zero, neg,
  input  [31:0] alu_in, acc,
  input  [ 2:0] alu_op);
  // RTL code for the ALU
endmodule

module accumulator (
  output [31:0] acc,
  input  [31:0] alu_out,
  input         ld_acc, clk, rst_n);
  // RTL code for the accumulator register
endmodule

module shifter (
  output [15:0] data,
  input  [31:0] acc,
  input  [ 1:0] shft_lshft,
  input         ld_shft, en_shft, clk, rst_n);
  // RTL code for the shifter
endmodule

module tribuf #(parameter SIZE=16)
  (output [SIZE-1:0] data,
   input  [SIZE-1:0] acc,
   input             en_acc);
  // RTL code for the tristate buffer
endmodule
```

When these modules are instantiated into a SystemVerilog-style CALU module with implicit port connections, the correct CALU module code is shown below:

```
module calu2 (
  inout  [15:0] data,
  input  [ 4:0] bs_lshft,
  input  [ 2:0] alu_op,
  input  [ 1:0] shft_lshft,
  input         calu_muxsel, en_shft, ld_acc, ld_bs,
  input         ld_multop1, ld_multout, ld_shft, en_acc,
  input         clk, rst_n);

  wire   [31:0] acc, alu_in, alu_out, bs, mult, multout;
  wire   [15:0] mop1;

  multop1        multop1        (.*);
  multiplier     multiplier     (.*);
  multoutreg     multoutreg     (.*);
  barrel_shifter barrel_shifter (.*);
  mux2           mux            (.y(alu_in), .i0(multout),
                                 .i1(acc), .sel1(calu_muxsel));
  alu            alu            (.*, .zero(), .neg());
  accumulator    accumulator    (.*);
  shifter        shifter        (.*);
  tribuf         tribuf         (.*, .acc(acc[15:0]));
endmodule
```

In the example code for the **calu2** module, note that all of the busses and nets that are connected to the ports of both the **multop1** and **multiplier** modules have names and sizes that match the port names and sizes on the instantiated modules. There is a 16-bit bus named **mop1** that is driven by the **multop1** register into the **multiplier** module. Since this bus is not a declared port on the **calu2** module, it must be explicitly declared in the **calu2** module in order to take advantage of the **.*** notation. Similarly, the 32-bit **mult** bus, driven by the multiplier module is also an internal bus and must be explicitly declared in the **calu2** module. All of the other ports that are connected to the **multop1** register and the **multiplier** instance are connected to busses and nets that are explicitly declared as ports on the **calu2** module and therefore they do not require a separate explicit net declaration.

In the example code for the **calu2** module, note that a generic 32-bit-wide, 2-to-1 **mux** has been instantiated. Since none of the ports on this sub-block match the net or bus names of the **calu2** module, all of the ports must either be connected by name or by position.

In the example code for the **calu2** module, note that the **alu** has two unused outputs, **zero** and **neg**. The unused ports must be listed with empty connections when using the implicit port connection token (**.***) to make the rest of the connections.

In the example code for the **calu2** module, note that the **tribuf** module has a 16-bit input port named **acc** but it is connected to a 32-bit bus also named **acc**. Since the port and bus sizes do not match, a named connection is required to show which bits of the 32-bit **acc** bus are connected to the 16-bit **acc** port. SystemVerilog does not assume that the low-order bits of a same-named port and bus are connected. That information must be provided in the named port connection.

For all of the other sub-blocks in the example code for the **calu2** module, any instance port that is connected to a **calu2** module port requires no additional explicit net declaration, while all of the instance ports of sub-blocks that are connected to **calu2** internal busses require explicit net declarations within the **calu2** module.

Implicit port connections are not very useful for a gate-level netlist where multiple copies of library gate primitives are instantiated and connected to nets with names that do not match the primitive port names, but implicit port instantiations are exceptionally useful for instantiating large logic sub-blocks into a higher-level model.