This DAC paper was an invited paper and is an abridged version of a SNUG 2007 - 28-page paper by the same title available on the www.sunburst-design.com web page.

World Class Verilog & SystemVerilog Training

SystemVerilog Implicit Port Enhancements Accelerate System Design & Verification

Clifford E. Cummings
Sunburst Design, Inc.
cliffc@sunburst-design.com

ABSTRACT
The IEEE Std 1800-2005 SystemVerilog Standard added new implicit port instantiation enhancements that help accelerate top-level composition of large ASIC & FPGA Designs. This paper details the new **.*** and **.name** implicit port instantiation capabilities, the rules related to the use of these new enhancements, and how these enhancements offer concise RTL coding styles while enforcing stronger port-type checking.

Categories and Subject Descriptors
B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids – *automatic synthesis, hardware description languages, simulation, verification.*

General Terms
Design, Verification, Documentation, Reliability, Languages.

Keywords
SystemVerilog, instantiation, implicit ports, .*, .name, Verilog EMACS mode, Verilog.

1. INTRODUCTION
For large ASIC and FPGA designs, the top-level design module or modules are 10's of pages of 100's of instantiations using 10,000's of named port connections.

The top-level design has very little value to design engineers.

There are so many port and signal names in a top-level module that it is nearly impossible to follow the intended design structure simply by looking at all the text. The only value of this top-level design is to establish connectivity for EDA tools. Since there is so little value in this top- level module, why should design engineers spend so much time piecing together a top-level design? Why can't we ask the tools to make the connection for us? That is the idea behind **.*** implicit port connections.

2. SAMPLE DESIGN
The IEEE Std 1800-2005[5], section 18.11, includes multiple instantiation examples of an **alu_accum** design modeled using Verilog[4] and SystemVerilog instantiation methods. Included in this paper is a slightly modified version of that same design (see Figure 1). In this design, the **alu** block has unconnected **zero** and **ones** output pins. The **alu_accum** model also has an unconnected **zero** output pin that is not connected to the unconnected **alu zero** output. This was done just to add an interesting scenario to the design.

3. PORT CONNECTION STYLES
In this section, the **alu_accum** model will be coded four different ways: (1) using positional port connections, (2) using named port connections, (3) using SystemVerilog **.*** implicit port connections, and (4) using SystemVerilog **.name** implicit port connections. The styles are compared for advantages, disadvantages, coding effort and efficiency.

3.1 Verilog positional port connections
Verilog has always permitted positional port connections. The Verilog code for the positional port connections for the **alu_accum** block diagram is shown in Example 1.

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

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

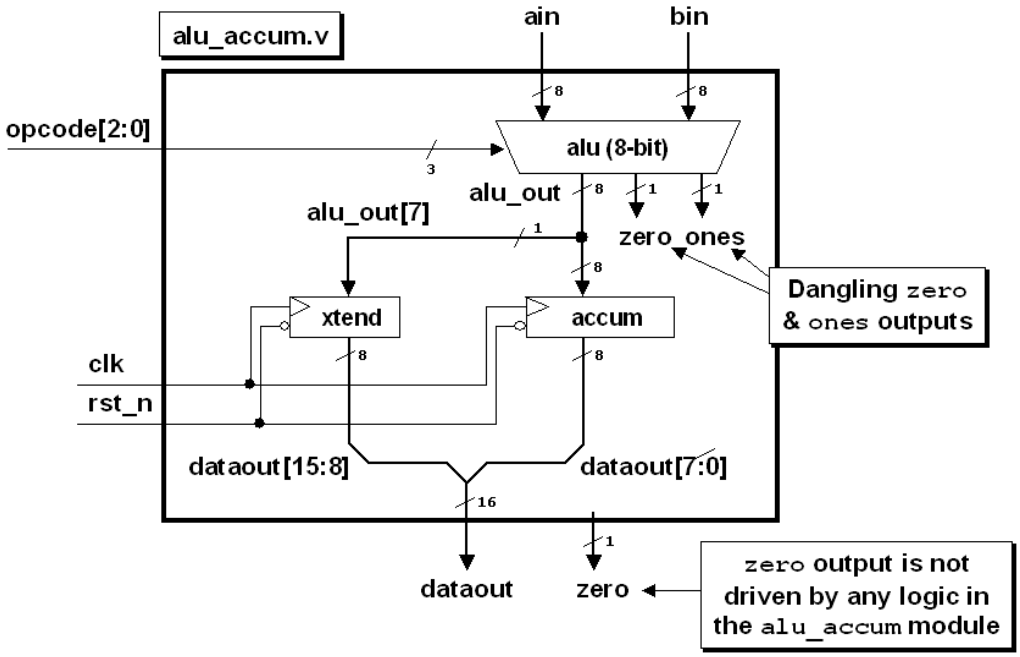

Figure 1 - alu_accum Block Diagram

```
  accum accum (dataout[7:0], alu_out,
               clk, rst_n);
  xtend xtend (dataout[15:8], alu_out[7],
               clk, rst_n);
endmodule
```

Example 1 - alu_accum model built using positional port connections

3.2 Verilog named port connections

Verilog has always permitted named port connections (also called explicit port connections). Any engineer who has ever assembled a top-level netlist for a large ASIC or FPGA is familiar with the tedious pattern of instantiating ports of the form:

```
mymodule u1 (.data(data), .address(address),
             ...  .BORING(BORING));
```

The top-level module description for a large ASIC or FPGA design may be 10-20 pages of tediously instantiated modules forming a collection of port names and net names that offer little value to the author or reviewer of the code. With net names potentially dispersed onto multiple pages of code, it is difficult for any engineer to comprehend the structure of such a design.

Most engineers agree that large top-level ASIC or FPGA netlists offer very little value aside from connecting modules together to simulate or synthesize. They are painful to assemble, painful to debug and sometimes painful to maintain when lower-level module port lists are modified, requiring top-level netlist modifications.

The problem with large top-level netlists is that there is too much information captured and the information is spread out over too many pages to allow easy visualization of the design structure. For all practical purposes, the top-level design becomes a sea of names and gates. The information is all there but it is in a largely unusable form!

The named port connections version of the Verilog code for the **alu_accum** block diagram is shown in Example 2.

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

  logic   [7:0] alu_out;

  alu   alu   (.alu_out(alu_out), .zero(),
               .ones(), .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
```

Example 2 - alu_accum model built using named port connections

3.3 The .* implicit port connection enhancement

SystemVerilog introduces the ability to do **.*** implicit port connections. Whenever the port name and size matches the connecting net or bus name and size, the port name can be

omitted and **.*** will make the connection automatically as shown in Example 3.

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

  logic   [7:0] alu_out;

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

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

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

Example 3 - - alu_accum model built using .* implicit port connections

3.4 The .name implicit port connection enhancement

SystemVerilog also introduces the ability to do **.name** implicit port connections. Just like the **.*** implicit port connection style, whenever the port name and size matches the connecting net or bus name and size, the port name can be listed just once with a leading period as shown in Example 4.

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

  logic   [7:0] alu_out;

  alu   alu   (.alu_out, .zero(), .ones(),
           .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
```

Example 4 - alu_accum **model built using** .name **implicit port connections**

4. .* ADVANTAGES

There are two strong advantages to using **.*** implicit port connections over Verilog positional or named port connections: (1) more concise designs, and (2) strong port type checking. These are two excellent reasons to use then new **.*** implicit port connections.

4.1 .* usage

According to IEEE Std 1800-2005, the **.*** may only be listed once per instantiation and may be placed anywhere in the instantiated port list.

Legal: **.*** at the beginning of the port list.

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

Legal: **.*** in the middle of the port list.

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

Recommended: **.*** at the end of the port list.

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

Illegal: **.*** in the port list twice.

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

To take advantage of the EMACS mode port collapsing and expansion, the **.*** must be placed last in the instantiated port list.

4.2 .* and .name rules

When instantiating modules using **.*** or **.name** implicit ports, the following rules apply:

(1) It is illegal to mix positional ports with new SystemVerilog **.*** or **.name** implicit port connections.

```
alu  alu (alu_out, , , .*); // ILLEGAL
```

In the above example, the first three ports are listed by position and the remaining ports are connected using SystemVerilog **.*** implicit ports. This is not legal in SystemVerilog.

```
// ILLEGAL
accum accum (dataout[7:0], alu_out, .*);
```

In the above example, the first two ports are listed by position and the remaining ports are connected using SystemVerilog **.name** implicit ports. This is not legal in SystemVerilog.

This restriction is not surprising. The new SystemVerilog implicit port styles are alternate forms of named port connections and it has never been legal in Verilog to mix positional ports with named ports in the same instantiation.

```
// ILLEGAL Verilog-2001 instantiation
xtend xtend (dataout[15:8], alu_out[7], clk,
             .rst_n(rst_n));
```

In the above example, the first three ports are listed by position and the last port is connected using a Verilog-2001 named port connection. This is not legal in Verilog or SystemVerilog.

(2) It is legal to mix SystemVerilog **.*** or **.name** implicit port connections with Verilog named port connections. As a matter of fact, it is required to mix SystemVerilog **.*** or

.name implicit port connections with named ports if rules 3 - 6 apply (see below).

The following rules apply if the individual module instantiation includes SystemVerilog **.*** or **.name** implicit port connections.

(3) If a port name does not match a connecting net name, then Verilog named port connections must be used.

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

In the above example, two ports are connected to buses that have different names. In this same example, the upper 8 bits of a 16-bit **dataout** bus is connected to an 8-bit **dout** port and the MSB of an 8-bit **alu_out** bus is connected to a 1-bit **din** input.

(4) If a port size is smaller than a connecting bus size, then Verilog named port connections must be used to show which bits of the connecting bus are connected to the port. Port sizes and connecting nets or buses must match.

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

In the above example, the lower 8 bits of a 16-bit **dataout** bus are connected to an 8-bit **dataout** port. Although the names matched, the sizes did not match so a named port connection was required. In this same example an 8-bit **alu_out** bus is connected to an 8-bit port with a different name (**datain**).

(5) If a port size is larger than a connecting bus or net size, then Verilog named port connections must be used and a concatenated bus of matching size (using constants, nets and or buses) must be connected to the port. Port sizes and connecting nets or buses must match.

```
mymod1 u1 (.din({4'b0,nibble}), .*);
```

In the above example, a 4-bit **nibble** bus is connected to an 8-bit **din** port so the 4 MSBs are connected to **0**'s through the use of a concatenation of **4'b0** and the 4-bit **nibble** bus.

(6) All unconnected ports must be shown as named empty port connections. Empty ports cannot be omitted, unlike in Verilog-2001 instantiations.

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

In the preceding example, the **zero** and **ones** ports are left unconnected so they must be explicitly listed.

(7) All internal 1-bit nets must be declared. In Verilog-2001 designs, undeclared identifiers automatically turned into 1-bit wires. For SystemVerilog, 1-bit nets connected to **.*** or **.name** implicit ports must be declared. Empty ports cannot be omitted, unlike in Verilog-2001 instantiations.

(8) According to IEEE Std 1800-2005, it is legal to mix SystemVerilog **.*** and **.name** implicit port connections in the same instantiation. The need to do this is rare and is only required when SystemVerilog packages are used to make declarations.

4.3 Comparing Verilog-2001 to SystemVerilog .* implicit ports

The comparisons between Verilog-1995/2001 positional or named port connections to SystemVerilog **.*** implicit port connections are shown in the following table.

Table 1 - Verilog port connections compared to SV implicit port connections

Verilog-2001 Port Connections	SystemVerilog .* Ports
If the port size does not match the connecting net or bus • port-size mismatch WARNING • simulations run without modification (this is almost always an error)	If the port size does not match the connecting net or bus • port-size mismatch ERROR • sizes must be corrected before simulations will run
Omitting ports from the instantiated module • LEGAL !! - no warning • simulations run without modification and treat the unlisted port as an unconnected port	Omitting ports from the instantiated module • ERROR • must list all unconnected ports before simulations will run
Undeclared 1-bit wires in top module • LEGAL !! - no warning • implicitly creates 1-bit wires	Undeclared 1-bit wires in top module • ERROR • all implicitly connected ports require an upper-level net declaration, including 1-bit wires

What does this mean? It means that using **.*** implicit ports permits the creation of more concise top-level designs with stronger port type checking. This is a win-win enhancement!

So why not add the additional checking to Verilog-2001 ports? The stronger port type checking cannot be added to Verilog-2001 style port connections for reasons of backward compatibility. The stronger port type checking would break too many existing designs.

4.4 Abstraction and fear

Many engineers are very enthusiastic about **.*** (myself included) while others are terribly afraid of the abbreviated syntax because they cannot see the port lists of instantiated modules.

When SystemVerilog **.*** implicit port connections were first introduced to engineers, about half of the engineers I taught enthusiastically embraced the use of **.*** implicit ports while the other half expressed fear related to debugging a design where the port names were not visible. For this reason, a large number of engineers elected to use **.name** implicit port connections instead of the more concise **.*** implicit ports. I was frustrated that I could

not convince more engineers to use the simple, elegant and more concise **.*** implicit ports.

I questioned how was I going to get more engineers excited about using the new **.*** implicit ports over using **.name** implicit ports.

5. VERILOG EMACS MODE PORT EXPANSION

At the urging of my colleague and friend, Erich Whitney[3], I approached Wilson Snyder to see if he could offer support for **.*** implicit ports in his Verilog EMACS mode[7]. Wilson's EMACS mode already supported the insertion of **/*AUTOINST*/**, which offered support for the exact same capability that we wanted with **.***. The request was made to Wilson in early November, 2005 and the first version of the EMACS mode with **.*** implicit port-expansion support was ready by the end of the same month.

5.1 EMACS basic setup

There were two main issues that had to be addressed to make **.*** support within EMACS a reality: (1) automated port expansion and collapsing of implicit ports, and (2) finding all the files of the instantiated modules so they could be used to match and check the port expansions. The second requirement could be most easily satisfied if the Verilog EMACS mode could recognize common Verilog command line options and simple Verilog command files.

If all of the design files are in the same directory, the EMACS mode can expand and collapse the ports automatically with no additional information provided to the EMACS Verilog mode.

The EMACS command sequence to expand all the ports for debugging is ctl-c ctl-a (often abbreviated as (C-cC-a) by EMACS users).

Note: the EMACS mode cannot expand and collapse **.name** implicit ports.

When the **alu_accum3** module of Example 3 is edited using the new EMACS Verilog mode, and when the **.*** implicit ports are expanded by issuing the command-key sequence (C-cC-a), the design is expanded as shown in Example 5

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

  logic   [7:0] alu_out;

  alu   alu   (
     .zero(), .ones(), .*,
     // Outputs
     .alu_out  (alu_out[7:0]), //Implicit .*
     // Inputs
     .ain      (ain[7:0]),     //Implicit .*
     .bin      (bin[7:0]),     //Implicit .*
     .opcode   (opcode[2:0])); //Implicit .*

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

  xtend xtend (
     .dout(dataout[15:8]),
     .din(alu_out[7]), .*,
     // Inputs
     .clk      (clk),          //Implicit .*
     .rst_n    (rst_n));       //Implicit .*
endmodule
```

Example 5 - alu_accum3 design with EMACS expanded ports

After debugging the **alu_accum3** module and when there is no more need to see all of the additional named ports, the expanded ports can be easily collapsed. The EMACS command sequence to collapse all of the ports is: (C-cC-k)

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

  logic   [7:0] alu_out;

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

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

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

Example 6 - alu_accum3 design with ports collapsed by EMACS

Note that the EMACS mode will automatically collapse the ports when the EMACS editor buffer is saved.

5.2 EMACS mode file-trailer comments

In reality, it is rare to find all of the necessary files for a large design in a single directory. Most Verilog power-users use command files with command options to designate the locations of all of the design files. Assuming that the command file is named **run.f** and that the command file is in the same directory as the top-level file that uses **.*** implicit ports, then it is a simple matter to add the following four lines of comment-code to the bottom of the top-level design module after the **endmodule** statement:

```
// Local Variables:
// mode: Verilog
// verilog-library-flags:("-f run.f")
// End:
```

With this EMACS directive explicitly placed at the bottom of the top-level design, the EMACS Verilog mode will now search all of the necessary design files to match and expand all of the ports for the top-level instantiations.

The expansion and collapsing of ports is done for all instances in the active module scope, which makes it easy to debug the design. When the debugging task is complete executing the command sequence (C-cC-k), or saving the design buffer will automatically collapse all of the expanded ports to save the very concise **.*** version of the design.

5.3 .* Adoption

As noted earlier, about one half of he engineers that I used to train would not use **.*** because they were afraid that **.*** designs would be difficult to debug. Now that I can use the EMACS Verilog mode to demonstrate an easy way to expand and collapse ports to help debug designs, I no longer hear objections to using the **.*** implicit ports.

5.4 EMACS -vs- vi

I am a **vi** and **vim** user and have not seriously used EMACS in over 10 years. The intent of this paper is not to convince you that you should abandon all other editors and exclusively use EMACS. Even though I continue to primarily use **vi** and **vim** for almost all file editing, when I need to do automated port expansion for design debug, I now open the **.*** version of the design using Wilson's EMACS mode and with a couple of keystrokes I can easily expand the ports for debugging purposes, and then collapse the ports for storage and normal use.

6. CONCLUSIONS

Early experimentation using the **.*** implicit port connections indicates that the amount of code required to build large top-level ASIC and FPGA designs can be reduced by 40-70% and that the more concise designs are debugged much quicker due to the strong port checking enforced by the **.*** implicit port rules.

It should be noted that a recent clarification by the SystemVerilog Standards Group has ruled that the **.name** implicit port connections do not require listing of unconnected ports. Although the original intent was to enforce listing of unconnected ports using the **.name** implicit port style, the IEEE 1800-2005 Standard was somewhat ambiguous on this point and later attempts to add clarification notes to enforce the restriction were rejected because vendors had already permitted users to omit the unconnected ports and the clarification would have imposed a potential backward compatibility issue for some users. For this reason, I now consider this to be another good reason to adopt **.*** implicit ports over the **.name** implicit port style (stronger port checking using **.***).

Wilson Snyder's enhancements to the Verilog EMACS mode makes it easy to expand and collapse ports when debugging is necessary, so there is no compelling reason not to employ the new, more concise, and more strongly checked **.*** implicit port connection style.

Also note that the Verilog EMACS mode cannot expand and collapse the **.name** implicit port connection style.

7. ACKNOWLEDGMENTS

Supreme kudos to Wilson Snyder for implementing **.*** port expansion and collapsing in the EMACS Verilog mode. Also a special thanks to Heath Chambers for reviewing and recommending improvements to this paper.

8. REFERENCES

[1] Clifford E. Cummings, “SystemVerilog Implicit Port Enhancements Accelerate System Design & Verification,” SNUG (Synopsys Users Group) September 2007 (Boston, MA), September 2007. Also available at http://www.sunburst-design.com/papers

[2] Clifford E. Cummings, “SystemVerilog Implicit Port Connections - Simulation & Synthesis,” DesignCon 2005 (Santa Clara, CA), February 2005. Also available at http://www.sunburst-design.com/papers

[3] Erich Whitney, personal communication.

[4] "IEEE Standard Verilog Hardware Description Language," IEEE Computer Society, IEEE, New York, NY, IEEE Std 1364-2001

[5] "IEEE Standard For SystemVerilog - Unified Hardware Design, Specification and Verification Language," IEEE Computer Society, IEEE, NY, NY, IEEE Std 1800-2005

[6] SystemVerilog 3.1a Language Reference Manual, Accellera’s Extensions to Verilog, Accellera Organization, Inc., Napa, CA, 2004. Available at www.eda.org/sv

[7] Veripool web site: http://www.veripool.com

9. Author & Contact Information

Cliff Cummings, President of Sunburst Design, Inc., is an independent EDA consultant and trainer with 26 years of ASIC, FPGA and system design experience and 16 years of SystemVerilog, synthesis and methodology training experience.

Mr. Cummings has presented more than 80 SystemVerilog seminars and training classes in the past five years and was the featured speaker at the world-wide SystemVerilog NOW! seminars.

Mr. Cummings has participated on every IEEE & Accellera SystemVerilog, SystemVerilog Synthesis, SystemVerilog committee, and has presented more than 40 papers on SystemVerilog & SystemVerilog related design, synthesis and verification techniques.

Mr. Cummings holds a BSEE from Brigham Young University and an MSEE from Oregon State University.

Sunburst Design, Inc. offers World Class Verilog & SystemVerilog training courses. For more information, visit the www.sunburst-design.com web site.

Email address: cliffc@sunburst-design.com

An updated version of this paper can be downloaded from the web site: www.sunburst-design.com/papers

(Last updated November 21, 2008)