SystemVerilog vs Verilog in RTL Design

By Pong P. Chu
Last updated in May 2018

1 INTRODUCTION

"FPGA Prototyping by SystemVerilog Examples: Xilinx MicroBlaze MCS edition" is the successor edition of "*FPGA Prototyping by Verilog Examples*." As the title indicates, the new edition uses the SystemVerilog language for design and coding. The change reflects that Verilog is now "absorbed" by SystemVerilog. It may appear somewhat intimidating at first glance since SystemVerilog is a complex verification language. While SystemVerilog mainly expands the verification capability, it also enhances the RTL design and modeling. The new RTL design and modeling features alleviate some "nuisances" of Verilog-2001 and make the code more descriptive and less error-prone. The new edition of the text takes advantage of the enhancement and incorporates about a half-dozen new features. This note summarizes the new SystemVerilog features (i.e., those not existed in Verilog-2001) used in the book.

2 BRIEF HISTORY OF VERILOG AND SYSTEMVERILOG STANDARDS

Verilog is intended mainly for the gate- and register-transfer-level design and modeling. It became the IEEE Standard 1364-1995 in 1995 (referred to as Verilog-95), revised in 2001 (referred to as Verilog-2001) and again in 2005 (referred to as Verilog-2005). The 2005 revision just included some minor corrections and spec clarifications. When the term "Verilog" is used, it generally refers to Verilog-2001.

As digital systems grow bigger and more complex, more sophisticated verification features are needed. The original Verilog language constructs cannot accommodate the increasing demand. In 2005, a language extension covering the verification functionalities, known as *SystemVerilog*, was added and the extension became IEEE Standard 1800-2005.

In 2009, Verilog and SystemVerilog were merged into a single standard. For whatever reason, the merged language is called SystemVerilog. It became IEEE Standard 1800-2009 and was then revised in 2012.

In summary, the two standards are evolved as follows

- Before 2009:
  - Verilog is the main language and is for RTL design and modeling.
  - SystemVerilog is the extension of Verilog and is for verification.
- After 2009:
  - Verilog ceases to exist.
  - SystemVerilog is the main language and incorporates RTL design and modeling as well as verification.

"FPGA Prototyping by SystemVerilog Examples" uses some enhanced RTL design and modeling portion of SystemVerilog and does not cover any feature from the verification portion.

3 NEW SYSTEMVERILOG FEATURES

About a half-dozen new SystemVerilog RTL design and modeling features are used in the book and they are summarized in the following subsections.

3.1 `logic` DATA TYPE

Verilog-2001 divides the data types into a "net" group and a "variable" group. The former is used in the output of a continuous assignment and the **`wire`** type is the most commonly used type in the group. The latter is used in the output of a procedural assignment and the **`reg`** type is the most commonly used type in the group. Verilog-2001 has a specific set of rules and restrictions regarding the assignment and connection of signals from the different groups.

The names of the **`wire`** and **`reg`** types are misleading. A signal declared with the **`wire`** type can be a connection wire or a component with memory (e.g., a latch). A variable declared with the **`reg`** type may or may not infer a register. It can be a connecting wire, a register, a latch, or a "C-like software variable." Furthermore, the rules and restrictions imposed on assignment and connection of signals from different the groups are confusing and unnecessary.

SystemVerilog introduces the **logic** data type. It can be used in the variable group or net group, which is inferred automatically from context. The **logic** type can replace the **`wire`** and **reg** types in most codes. In addition, its name does not imply a specific hardware component and thus is more descriptive. The book uses the **logic** type.

3.2 ALWAYS BLOCKS

In addition to the "general-purpose" **always** block. SystemVerilog introduces three additional procedural blocks to describe the nature of the intended hardware:

- **always_comb.** It indicates that the block is intended to model a combinational circuit. Like **always @(*)** in Verilog-2001, the **always_comb** uses an implicit sensitivity list that includes all signals in right-hand-side expressions. In addition, the software will generate a warning if a latch (due to incomplete variable assignment) is inferred since the circuit is no longer a combinational circuit.
- **always_ff.** It indicates that the block is intended to model a register. The syntax of **always_ff** is similar to that of a general-purpose **always** block. It just informs the software that the block should infer a flip-flop FF or a register. The software will generate a warning if no FF or register is inferred.
- **always_latch.** It indicates that the block is intended to model a latch (i.e., a circuit with one or more "combinational loops" that forms internal memory).

The **always_comb** block prevents the incomplete sensitivity list for a combinational circuit. The suffixes of the new blocks clearly express the intended type of hardware. This helps the designer to better document the code and helps software to catch errors in an earlier stage. The book uses **always_comb** for combinational circuits and **always_ff** for FFs and registers.

3.3 ENUMERATE DATA TYPE FOR FSM

An FSM contains a set of symbolic states. When the FSM is realized in hardware, the symbolic states are mapped to binary representations in the *state assignment* process. In Verilog-2001, it is "emulated" using the **localparam** construct. For example, consider an FSM with states of {s0, s1, s2}. The code segment is

```
// FSM symbolic states
localparam [1:0] s0 = 2'b00,
                 s1 = 2'b01,
                 s2 = 2'b10;
// signal declaration
logic [1:0] state_reg, state_next;
```

It is cumbersome and may introduce subtle errors (such as duplicated constant values).

SystemVerilog introduces a new enumerate data type, which can explicitly list symbolic values of a set. For example, the previous FSM declaration can be rewritten as

```
// define an enumerate data type for FSM states
typedef enum {s0, s1, s2} state_type;
// signal declaration
state_type state_reg, state_next;
```

The is much clear and more descriptive. The book uses enumerate data type for FSM design.

3.4 **$log2c()** FUNCTION

SystemVerilog includes a new system function, **$log2c( )**, which performs $\lceil log_2 x \rceil$. It is a handy function to determine the number of bits needed to represent a value. For example, a parametrized mod-*M* counter needs a $\lceil log_2 M \rceil$-bit register and the code can be written as

```
module mod_m_counter
   #(parameter M=10) // mod-M
   (input  logic clk,
    output logic [$clog2(M)-1:0] q);

   //signal declaration
   logic [$clog2(M)-1:0] r_reg, r_next;

   // register
   always_ff @(posedge clk)
      r_reg <= r_next;
   // next-state logic
   assign r_next = (r_reg==(M-1)) ? 0 : r_reg + 1;
   // output logic
   assign q = r_reg;
endmodule
```

Verilog-2001 doesn't support this function. We need to define it as a user function or introduce a second parameter. The book uses this function as needed.

3.5 TWO-DIMENSIONAL PORT DECLARATION

Verilog-2001 supports two-dimensional array data type. However, the data type cannot be used in the port declaration. The workaround is to declare the port as a one-dimensional array and then reconstruct the two-dimensional signal internally, as shown in the following code segment:

```
module demo_verilog
(  . . .
   /* input logic [31:0] x_2d [63:0],  // not allowed in Verilog-2001 */
   input logic  [64*32-1:0] x_1d,
   . . .  );

// 2d ok in internal declaration
logic [31:0] x_2d [63:0];

// reconstruct 2d signal inside the module
genvar i;
generate
   for (i=0; i<64; i=i+1) begin:
      assign x_2d[i] = x_1d[(i+1)*32-1 : i*32];
   end
endgenerate
. . .
```

SystemVerilog supports two-dimensional array data type in the port declaration. The previous code segment can be rewritten as

```
module demo_sv
(  . . .
   input logic [31:0] x_2d [63:0], // ok in SystemVerilog
   . . .  );
```

The book uses two-dimensional array type in port declaration for the MMIO controller and video controller modules.