Practical SystemVerilog Features

Instructor : Pei-Yun Tsai,
Meng-Yuan Huang

Digital System Design and Implementation

Outline

- Introduction
- SystemVerilog for RTL Design
- SystemVerilog for Verification
- References

Introduction

- Standard evolution: (IEEE Std. 1364) Verilog 95 -> Verilog 2001 -> Verilog 2005, (IEEE Std. 1800) SystemVerilog 2005 -> SystemVerilog 2009 -> SystemVerilog 2012 [1].
- Improvements from Verilog to SystemVerilog:
  - SystemVerilog for RTL Design: package, multidimensional array, …
    - Most supported by Xilinx Vivado 2017.04.
  - SystemVerilog for Verification: class, assertion, coverage, …
    - Some supported by Xilinx Vivado 2017.04.

File Extensions

- Verilog file extension – v. Examples:
  - Desgin.v
  - Testbench.v
- SystemVerilog file extension – sv. Examples:
  - Design.sv
  - Testbench.sv
- Verilog is **basically a subset** of SystemVerilog. Main exceptions are new keywords introduced in SystemVerilog, e.g., “logic.” Converting Verilog to SystemVerilog files by changing file extension usually works.

SYSTEMVERILOG FOR RTL DESIGN

Continuous Assignment to Variable

Verilog:

```
// Net
wire w;
// Continuous assignment
assign w = 1;

// Variable
reg r;
always(*) begin
   // Procedural assignment
   r = 1;
end
```

SystemVerilog:

```
// Variable
logic w;
// Continuous assignment
assign w = 1;

// Variable
logic r;
always(*) begin
   // Procedural assignment
   r = 1;
end
```

4-state integer type

always_comb

Verilog:

```
// Modeling combinational logic
always@(*) begin
   if(sel)
      out = in0;
   else
      out = in1;
end
```

SystemVerilog:

```
// Modeling combinational logic
always_comb begin
   if(sel)
      out = in0;
   else
      out = in1;
end
```

always_comb advantage:
Check “assignments from multiple always_comb” error during simulation time before synthesis time.

always_ff

Verilog:

```
// Modeling flip-flop-based sequential logic
always@(posedge clk) begin
   q <= d;
end
```

SystemVerilog:

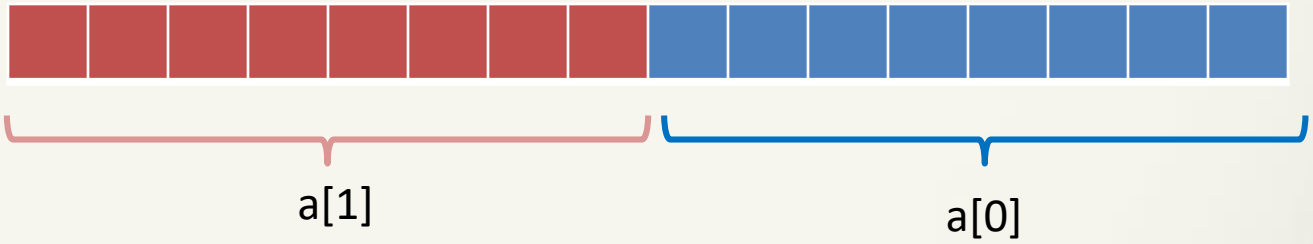
```
// Modeling flip-flop-based sequential logic
always_ff@(posedge clk) begin
   q <= d;
end
```

always_ff advantage:
Check “assignments from multiple always” error and “blocking assignment” error during simulation time before synthesis time.

Packed Arrays

- A packed array is a mechanism for subdividing a **vector** into subfields, which can be conveniently accessed as array elements.
- Allowed data types: single bit data types (logic, reg, wire), enum logic, …
- Dimension declarations are in the left of identifier names.
- Examples:

```
logic [1:0][7:0] a; // 2D packed array
logic [15:0] b; // 1D packed array
// Act like a one dimensional vector
assign a = 16'hABCD;
assign b = a;
// Act like an array.
assign a0 = a[0][0];
assign a9 = a[1][0];
```



Unpacked Arrays

- An array of elements with the same data type. Cannot be assigned like a vector as packed array.
- Allowed data types: any data type
- Dimension declarations are in the right of identifier names.
- Examples

integer b [0:7]; // unpacked array of 8 integers.

assign b[0] = 32'h01234567;

assign b[5] = 32'h76543210;

Multidimensional Array (MDA) (1/2)

- Verilog restricts the packed dimension to 1. SystemVerilog has no such restriction.
- Examples:

```
logic [2:0][1:0] arr; // 3*2 = 6 bits
assign arr[0] = 2'b10;
```

- In IO ports:

```
module MultiDimArray(
  output logic [2:0][1:0] o,
  input [2:0][1:0] i);
  assign o = ~i;
endmodule
```

Default data type: wire logic

- Multidimensional unpacked packed array: logic [2:0][2:0] arr [0:2][0:2];

Multidimensional Array (MDA) (2/2)

- MDAs will be expanded during synthesis -> RTL testbench can not be reuse.

```
// Synthesis netlist
module MultiDimArray
  (\o[2] , \o[1] , \o[0] ,
   \i[2] , \i[1] , \i[0] );
 output [1:0]\o[2] ;
 output [1:0]\o[1] ;
...
endmodule
```

- Workaround: RTL top module IOs don't use MDAs.

```
module MultiDimArray(
   output logic [1:0] o2, o1, o0,
   input [1:0] i2, i1, i0
   );
   logic [2:0][1:0] o;
   logic [2:0][1:0] i;
   assign i = {i2, i1, i0};
   assign o = ~i;
   assign {o2, o1, o0} = o;
endmodule
```

typedef and enum (1/2)

- typedef can define custom data types.
- Example:

```
typedef logic [2:0] Data;
Data d;
Data dReg;
always_ff@(posedge clk)
   dReg <= d;
```

The same as : logic [2:0] d;
logic [2:0] dReg;

- enum (enumeration) is used to declare a set of integral named constants.
- Example:

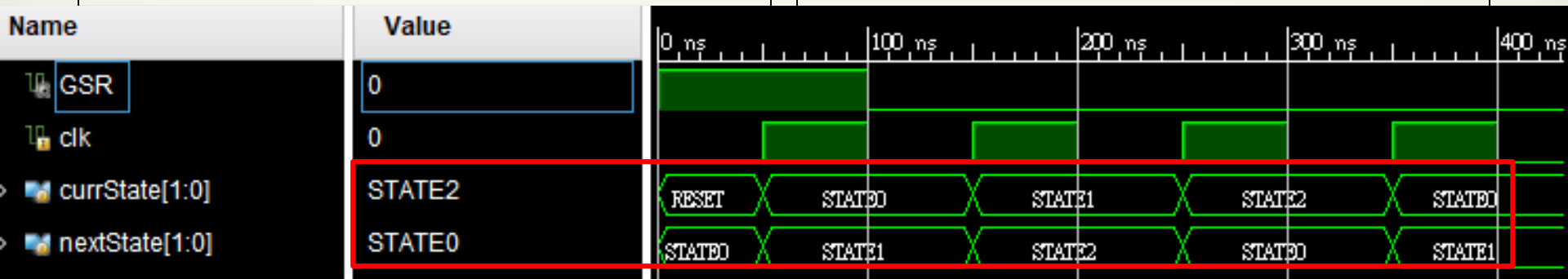
```
enum logic {FALSE, TRUE} truthValue;
// Equivalent to
enum logic {FALSE = 0, TRUE = 1} truthValue;
assign truthValue = TRUE;
```

typedef and enum (2/2)

```
typedef enum logic [1:0] {
    RESET, STATE0, STATE1, STATE2
} State; // State is data type.
State currState, nextState;

always_ff@(posedge clk or negedge nRst) begin
    if(!nRst) currState <= RESET;
    else currState <= nextState;
end
```

```
always_comb begin
    case(currState)
        RESET: nextState = STATE0;
        STATE0: nextState = STATE1;
        STATE1: nextState = STATE2;
        STATE2: nextState = STATE0;
    endcase
end
```



$clog2 System Function

- $clog2 can be used to obtain the minimum wordlength for representing a maximum number.
- Math:

$$\$clog2(n) = \lceil \log_2(n) \rceil, \text{ where } \lceil \cdot \rceil \text{ is ceiling function.}$$

- Example:

```
parameter maxCnt = 9;
// log2(9) = 3.17
// $clog2(9) = 4
// cnt wordlength is 4 bits.
logic [$clog2(maxCnt)-1:0] cnt;
```

package

- package is used to share typedef enum, parameters, functions and tasks to modules.
- Example:

```
package P;
   parameter wl = 3;
endpackage

module moduA(input [P::wl - 1:0] d);
// parameter wl=3
    moduB ub(d);
endmodule
```

```
module moduB(input [P::wl - 1:0] d);
// parameter wl=3
      moduC uc(d);
endmodule


// No need to pass parameter wl from module A to C hierarchically. More clean code.
module moduC(input [P::wl - 1:0] d);
// parameter wl=3
.....
endmodule
```

Implicit .* Connections

- .* implicitly connect IO ports to nets/variables with same names.
- Example:

```
module moduC(c2, c1, c0);
…
endmodule

module moduD(d2, d1, d0);
…
endmodule
```

```
module Test;
   logic c2, c1, c0;
   logic d2, d1, d0;
   // Connections by .*.
   moduC uc(.*);
   // Connections by mixing name and .*.
   moduD ud(.d1(d1), .*);
endmodule
```

Interface (1/2)

- Packs a set of signals by interface.

```
interface A_B_If();
   logic data;
endinterface

module moduA(A_B_If ios);
   assign ios.data = 1;
endmodule

module moduB(A_B_If ios);
   initial $display("%d", ios.data);
endmodule
```

```
module Top();
   // Connect A and B with A_B_If.
   A_B_If a_b_if();
   moduA a0(a_b_if);
   moduB b0(a_b_if);
endmodule
```

- Advantages:
  - No need to declare signals (e.g. data) in each module.
  - No need to connect modules signal by signal.

Interface (2/2)

- Another example:

```
interface memBus(input clk);
    logic [7:0] Add, D,Q;
    logic WE;
endinterface

module Array1(memBus bus);
logic [7:0] mem [0:255];
always_ff@(posedge bus.clk)
if (bus.WE)
   mem[bus.Add]=bus.D;

always_ff@(posedge bus.clk)
   bus.Q=mem[bus.Add];

endmodule
```

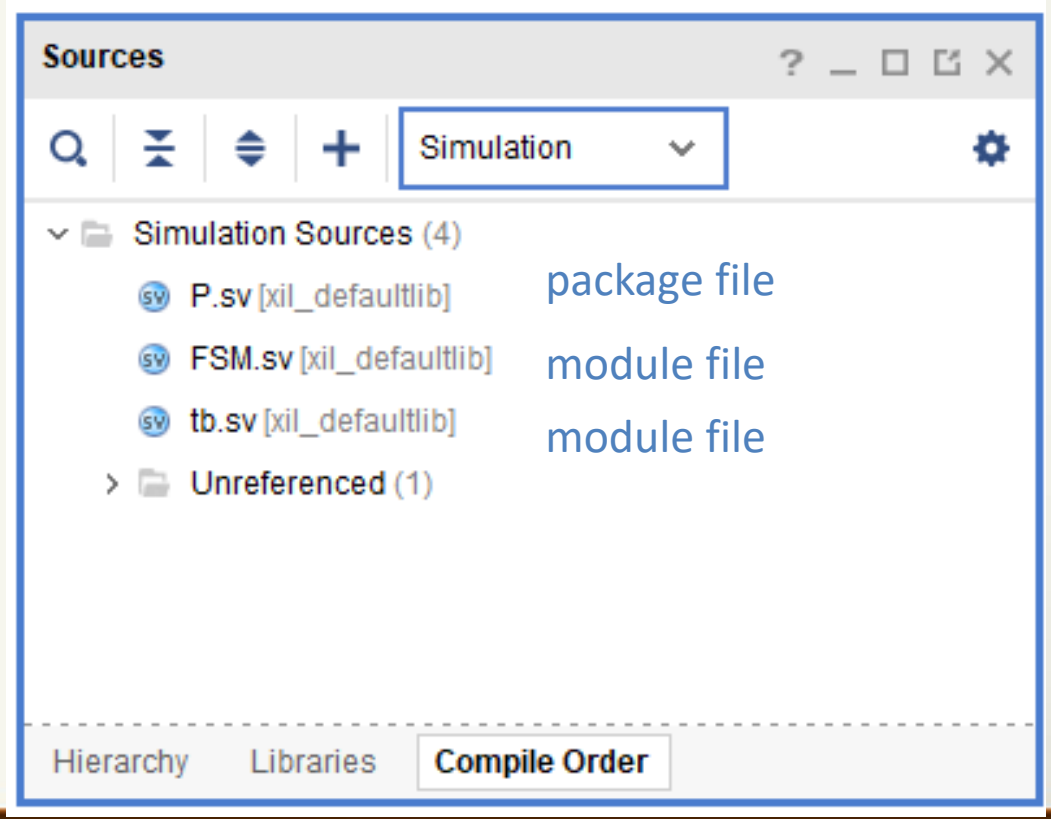
```
module Testbench;
reg clock;
// interface
memBus Tbus(.clk(clock));

// device under test
Array1 U1(.bus(Tbus));

initial
begin
…
end
endmodule
```

Compile Order

- Verilog modules have arbitrary compile orders.
- Suggested SystemVerilog file compile order:
  - package files
  - interface files
  - module files
- Xilinx Vivado **automatically** update compile order. If the manual compile order is necessary, please refer: https://www.xilinx.com/support/answers/64112.html



Automatically updated compile order in Vivado

SYSTEMVERILOG FOR VERIFICATION

$sformatf System Function

- $sformatf returns a formatted string.
- Syntax:

  $sformatf (format_string [, list_of_arguments])
- Specifiers (e.g., %d (decimal), %x (hexadecimal), %b (binary), %s (string)) in format_string will be replaced with list_of_arguments.
- Example:

```
integer fid;
initial begin
   // Create files, File0.txt, File1.txt and File2.txt.
   for(int i = 0; i < 3; i++)
      fid = $fopen($sformatf("File%0d.txt", i));
end
```

string data type

- “string” is a convenient data type to store string without specifying its length.
- Examples:

```
string str0 = "ThisIsA";
string str1 = "LongFileName";
// Combine strings.
string str2 = {str0, str1};
initial $display($sformatf("%s%0d.txt", str2, 0));
// Print "ThisIsALongFileName0.txt"
initial $display($sformatf("%s%0d.txt", str2, 1));
// Print "ThisIsALongFileName1.txt"
```

SEE MORE VIVADO SUPPORTED SYSTEMVERILOG FEATURES IN [2].

References

[1] “IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language,” IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009), pp. 1–1315, Feb. 2013.

[2] "Vivado Design Suite User Guide Logic Simulation UG900 (v2017.4)," Xilinx, pp. 167, December 20, 2017 [Online] http://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug900-vivado-logic-simulation.pdf

[3] “How Do I Manually Set Simulation Compile Order?,” Xilinx Vivado [Online] https://www.xilinx.com/support/answers/64112.html