SystemVerilog Features - Cheat Sheet

Answers to Frequently Asked Questions

SystemVerilog features & syntax can be quite confusing, thanks to historic reasons such as maintaining backward compatibility for 39 years. Explaining those details would confuse beginners. Therefore, When you code, just follow the following best practices. For anyone curious, the full picture and real reasons are given at the end of this document.

1. **Datatypes**
   - **Use logic everywhere** for wires and registers in RTL design, except for one-line combinational assignments
     - e.g: logic x = 0; will throw synthesis error in ASIC tools
   - ONLY in FPGAs, only to set initial values for registers during power-up time, use
     - output logic x=0;
     - initial logic x=0;
   - Use wires only for shorthand (optional) one-line combinational assignments
     - e.g: wire x = y; ties x to y.
   - Use bit types only in simulation
   - Use genvar or int in for loop variables in RTL design
   - Do not use “reg”

2. **Blocking (=) vs Non-Blocking (<=) assignments**
   - = is for designing combinational circuits (assign x=, wire x=, always_comb x=)
   - <= is for designing sequential circuits (always_ff @(posedge clk) x<= y)
   - DO NOT MIX / SWAP THEM
   - In the testbench, (=) get assigned one after the other (blocking) like software, while (<=) get assigned at the same time step, like driving hardware signals (non-blocking)
     - Use blocking (=) for software variables. (i=i+1)
     - Use non-blocking (<=) to assign to hardware signals

1-bit Full Adder

Design Features

module

- Template (like a class) for "instances" (like objects)
- Can be parameterized
- Instantiation
    - port order: adder(a,b,ci,s,co) - error-prone, don't do this
    - named: adder(.a(a),.b(b),.ci(ci),.s(s),.co(co)
    - dot-star: adder(.*) - if port names, sizes match with variables/nets in your code.

wire

- Represents a "physical wire"
- 4-state datatype, net type
- Used mostly for single line assignments: wire x = a & b;

logic

- Represents a "physical wire"
- 4-state datatype, variable type
- Used almost everywhere else except single line assignments.

always_comb

- "Procedure" for combinational assignments
- Will be helpful in decoders
- Synonym to old verilog: always @(*)

logical vs bitwise operators

- 2'b10 & 2'b11: 10 // bitwise and - result is a vector of same width
- 2'b10 && 2'b11: 1 // logical and - boolean (one bit) result (1,0)

Simulation only

timeunit, timeprecision

- defines the unit of time used in delays
- If timeunit is 1ns, #10 will cause 10ns delay
- timeprecision defines the smallest scale of time the simulator should simulate. Today it is irrelevant. Just set it to 1ps or something.

dumpfile, dumpvars - to get vcd waveform

assert

- assert (condition) pass; fail;
- Checks if the condition is true. If true, does the pass statement (optional). Else doesn't fail statement (optional).
- Used to automatically compare the output of the design with expected output

$display - System task. Prints the string in terminal

$error - similar to display, but causes simulation failure. (It shows fail at the end)

$finish - stop the simulation

N-bit Full Adder

Design features

vector

- collection of physical wires
- logic [3:0] x; // little endian, 4-bit vector of logic type
- Can reference a single bit: x[1], or a range of bits: x[2:1] (2 bits)

packed vs unpacked array:

- logic [3:0] x; // packed
- logic y [3:0]; // unpacked
- Both can be used in RTL design, but assigning one to another is hard (needs streaming)
- Better to use only packed arrays in RTL design - easy to assign to one another.
- Indexing:
  The following are all indexed as arr[a][b][c][d][e] to get a 1-bit value:
  - logic [A-1:0][B-1:0][C-1:0][D-1:0][E-1:0] arr; //packed
  - logic arr [A-1:0][B-1:0][C-1:0][D-1:0][E-1:0]; //unpacked
  -
  - logic [C-1:0][D-1:0][E-1:0] arr [A-1:0][B-1:0]; // mixed
- Vectors vs arrays:
  - In C programming, int arr [5]; is an array of 5 integers. An integer is either 2 or 4 bytes, depending on your processor.
  - SV is a hardware language. An integer in SV is shorthand for logic [31:0] (see datatypes)
  - Therefore, logic [4:0][31:0] x; represents an array of 5 words, each of 32 bits. x[0] gives a 32-bit word and x[0][0] gives a 1-bit wire.
  - Similarly [5:0][4:0][3:0][2:0] x is a multidimensional array.

parameter vs localparam -

- module parameter can be changed from outside (during instantiation).
- Local parameter is calculated inside the module based on other parameters. Cannot be directly changed from outside.

Combinational ALU

RTL Design

signed vs unsigned

- Vectors can be either signed or unsigned (default).
- Used ONLY for sign extensions. signed vectors will be sign extended (first bit copied) when assigning to a bigger vector

if - else

- Suggests a priority encoder (first if has high priority).
- Tool might understand there's no priority and might optimise it into parallel decoder (multiplexer)

case

- Multiplexers
- unique case - tells the tool to make everything parallel
- priority case - priority decoder

Simulation only

class

- Template to create objects
- new ()
    - constructor, available by default (hidden) in every class
    - We can override it for initialization of values
- rand attribute - can specify constraints: range of allowed values
- randomize() - will randomize all rand attributes as per constraints

Full Picture

Datatypes

Full, confusing explanation is given here, for the sake of completion. SystemVerilog datatypes have two properties: types and state-types.

- Types
  - net
    - Can be driven by multiple drivers - Don't do it, unless you know what you are doing
    - wire is one of the net types
  - var - Only one driver - safe
- state-type
  - logic: 4 state
    - 0 - low voltage
    - 1 - high voltage
    - x - undefined (initially not driven or multiple drivers
    - z - high impedance (not connected)
    - Use 4 state types for RTL design
  - bit: 2 state
    - 0, 1
    - Used for variables in simulation

Some example synthesizable datatypes and their properties are:

- logic
  - shorthand for “var logic”
  - Except in module input & inout ports, where “wire logic” is inferred
- wire - net, 4 state
- bit - shorthand for “var bit”
- reg - remnant from old Verilog, equivalent to “var logic”
- integer - equivalent to “var logic [31:0]”
- int - equivalent to “var bit [31:0]”
- byte - equivalent to “var bit [8:0]”
- shortint - equivalent to “var bit [15:0]”
- longint - equivalent to “var bit [63:0]”
- genvar - Not a datatype. It is an integer used only for synthesis (for loops…etc)

Some example non-synthesizable (simulation only) datatypes and their properties are:

- real - 64-bit floating point
- shortreal - 32-bit floating point
- string - dynamic array of byte types that store 8-bit ASCII characters

Why is this so complicated?
(System)Verilog is 39 years old. They kept old features for backwards compatibility.

Blocking (=) & Non Blocking (<=)

Do not mix blocking & non-blocking in same always block. Use = for combinational, <= for sequential circuits.

Blocking statements (=)

- Execute one after the other (they block each other), like a programming language
- In the following example, x is calculated first, and its value is used to calculate y in every time step.

```
always_comb begin
   x = a & b; // and
   y = x | c; // or
end
```

- Therefore, this is equivalent to assign `y = (a & b) | c;`
- That is why we use this for combinational assignments

Non-blocking statements (<=)

- Execute all at the same time step
- In the following example, x and y are flip-flops, and they get updated at the same clock cycle.

```
always_ff @(posedge clk) begin
   x <= a & b;
   y <= x | c;
end
```

- Value of y in this clock cycle will be value of x in the previous clock cycle, OR'd with value of c in previous clock cycle. Because both x & y update in same time step (clock edge).
- That is why we use this for sequential assignments.