

SysVeri Labs

LAB NO:	01	TOPIC:	Introduction to SystemVerilog	↓
---------	----	--------	-------------------------------	---

Compiled by: Shahzaib Kashif

Lab 01: Introduction to SystemVerilog

Objective:

This laboratory session will be an introduction to SystemVerilog (SV) programming. The objective is to enable the reader to write simple SV programs, model basic hardware like gates, and implement simple modules.

1.1 Introduction to SystemVerilog

SystemVerilog is a hardware description and verification language that extends Verilog. It combines procedural, object-oriented, and constraint-based programming for designing and verifying digital systems. SV code is compiled and simulated using tools like ModelSim or Verilator, and can be synthesized to hardware (e.g., FPGA).

We will start with primitive data types in SV followed by an introduction to modules and procedural programming.

1.1.1 SystemVerilog Data Types

In SystemVerilog, signals and variables have data types. Common types are treated as 4-state (0,1,X,Z) for simulation realism. Variables can be static or dynamic.

Key data types are listed in Table 1.1. Use **logic** for most signals (it's a variable type). Nets like **wire** are for connections.

Table 1.1: SystemVerilog data types.

Data Type	Description
logic	4-state bit (0,1,X,Z); default for signals
bit	2-state bit (0,1)
byte	8-bit signed integer
shortint	16-bit signed integer
int	32-bit signed integer
longint	64-bit signed integer
real	Double-precision float
string	Sequence of characters
wire	Net for connections (can't store in always blocks)

Data types follow a loose hierarchy: scalars (bit/logic) vs. vectors (e.g., logic [31:0]), and variables vs. nets.

1.1.2 SystemVerilog Modules

A module defines a hardware block. It's like a class but for hardware.

Example: A simple counter module.

Counter.sv

```

1  module Counter #(parameter BITS = 8) (
2      input logic clk,
3      input logic reset,
4      output logic [BITS-1:0] count
5  );
6      always @(posedge clk or posedge reset) begin
7          if (reset) begin
8              count <= 0;
9          end else if (count == (1 << BITS) - 1) begin
10             count <= 0;
11          end else begin
12             count <= count + 1;
13          end
14      end
15      initial $display("Counter created with bits %d", BITS);
16  endmodule

```

- Instantiate: In another module, e.g., `Counter #(.BITS(16)) my_counter(.clk(clk), ...);`

1.1.3 SystemVerilog Type Casting

Use apostrophe for static casting or \$cast for dynamic.

Example: Numeric casting.

```
Casting.sv

1 byte b = 8'hFF; // -1 signed
2 int i = int'(b); // Cast to int (-1)
3 logic [7:0] u = 8'hFF;
4 logic signed [7:0] s = signed'(u); // Cast to signed (-1)
5 $display("Byte: %d, Int: %d", b, i);
```

For modules/interfaces (OOP-like), use \$cast for polymorphism.

1.2 Basic Gates in SystemVerilog

Model gates using primitives or assign statements.

Example:

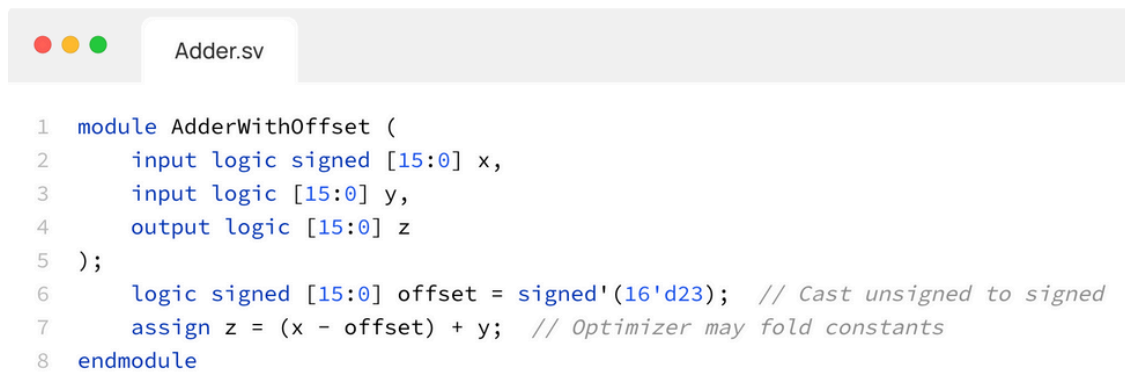
```
Gates.sv

1 module BasicGates (
2     input logic a, b,
3     output logic and_out, or_out, xor_out, not_out
4 );
5     assign and_out = a & b; // Behavioral
6     assign or_out = a | b;
7     assign xor_out = a ^ b;
8     assign not_out = ~a;
9
10    // Or primitives
11    and g_and(and_out_prim, a, b);
12 endmodule
```

1.3 Optimization of Signals and Parametrized Hardware Generation

SV optimizes constants during compilation. Parameters allow configurable hardware.

Example: Offset adder with casting.



```

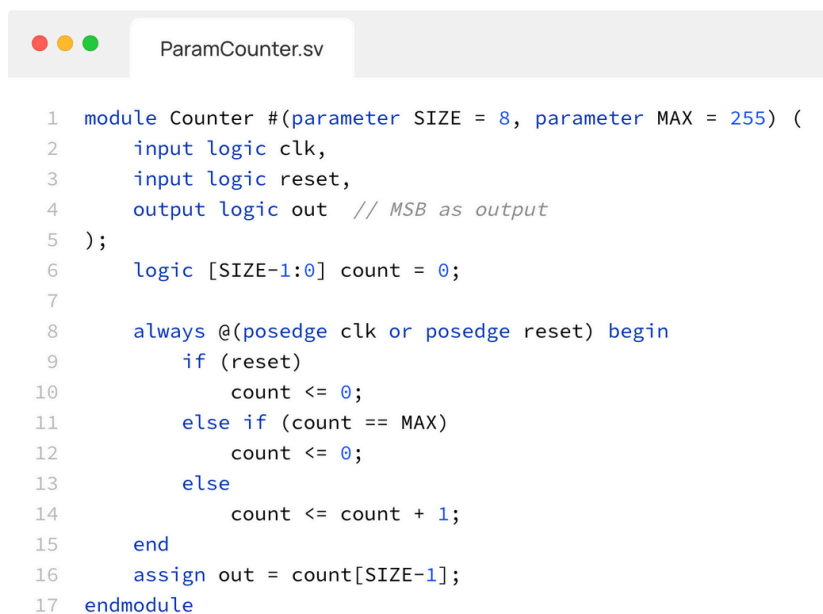
1 module AdderWithOffset (
2     input logic signed [15:0] x,
3     input logic [15:0] y,
4     output logic [15:0] z
5 );
6     logic signed [15:0] offset = signed'(16'd23); // Cast unsigned to signed
7     assign z = (x - offset) + y; // Optimizer may fold constants
8 endmodule

```

1.3.1 Parametrized Hardware Generation

Use parameters and generate blocks for configurable designs.

Example: Parametrized counter.



```

1 module Counter #(parameter SIZE = 8, parameter MAX = 255) (
2     input logic clk,
3     input logic reset,
4     output logic out // MSB as output
5 );
6     logic [SIZE-1:0] count = 0;
7
8     always @(posedge clk or posedge reset) begin
9         if (reset)
10             count <= 0;
11         else if (count == MAX)
12             count <= 0;
13         else
14             count <= count + 1;
15     end
16     assign out = count[SIZE-1];
17 endmodule

```

1.4 Exercises

Exercise 1: Modify the counter to use signed int type for count.

Exercise 2: Reset the counter when MSB flips from 0 to 1.

Exercise 3: Change MAX to int type and cast it to logic vector.

1.5 Assignments

Task 1: Determine which type castings are possible (similar table as in Chisel lab, adapted to SV types like logic, int, wire, etc.).

Task 2: Implement an up-down counter module. Starts at 0, counts up to a parameter, then down to 0. Sets output high for one cycle at max/min.

Skeleton:

```
1 module counter_up_down #(parameter N = 8) (  
2     input logic clk,  
3     input logic reset,  
4     input logic [N-1:0] data_in,  
5     input logic reload,  
6     output logic out  
7 );  
8     logic [N-1:0] counter = 0;  
9     logic [N-1:0] max_count = 6;  
10    // Your code here  
11 endmodule
```