Intelligent Architecture Lab Manual

Lorenzo Pes, Zhanbo Shen
l.pes@tue.nl, z.shen1@tue.nl

December 12, 2024

Neuromorphic Edge Computing Systems Lab

Contents

1 SystemVerilog (for those who are new)

SystemVerilog provides new features on top of Verilog. Some of you may not come across this tool before, which is fine because we will have a brief tutorial about the features that we will use in this lab and hope that will be enough to get you started. For those who are familiar with SystemVerilog, we meet you at the next section!

SystemVerilog is a hardware description and hardware verification language used to model, design, simulate, test and implement electronic systems. It is based on Verilog and some extentions, commonly used in the semiconductor and electronic design. If you are familiar with Verilog, you will find SystemVerilog easy to learn, if not, we also provide you a brief journey to the hardware description world.

1.1 Data Types

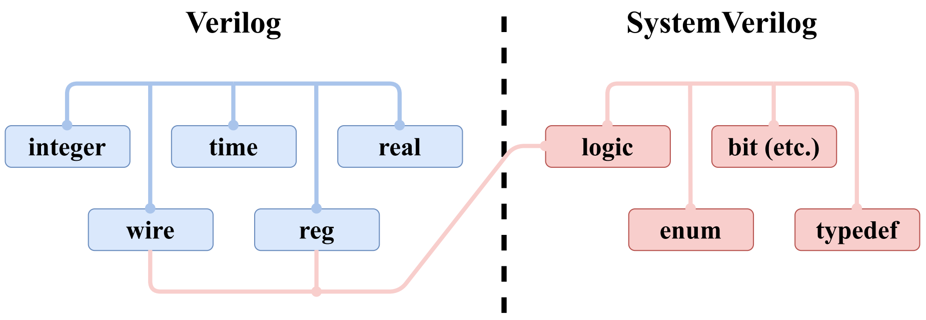

Figure 1: Data Types in Verilog & SystemVerilog

Verilog:

- **integer:** 32-bit signed integer, only for simulation and non-synthesis purpose.

```
integer i;
for (i=0; i<10; i=i+1) begin
    $display("i = %d", i);
end
```

- **time:** 64-bit unsigned integer, for presenting time.

```
time t_start, t_end;
```

- **real:** floating number, for accurate time calculation and simulation.

```
real voltage;
```

- **wire:** net type, for connecting modules and continuous assignment (combinational logic).

```
wire a, b;
assign a = b & 1'b1;
```

- **reg:** memory or register type, for storing values and sequential logic.

```
reg clk;
always @(posedge clk) begin
    clk <= ~clk;
end
```

SystemVerilog:

- **logic:** to replace wire and reg, for both combinational and sequential logic.

```
logic clk, reset;
```

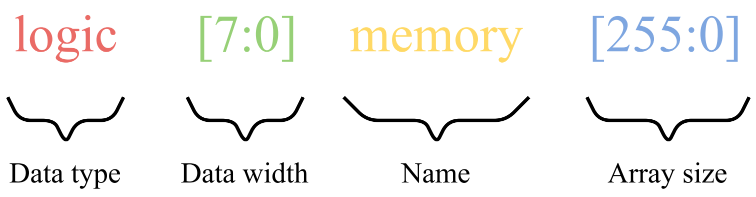

Figure 2: Array in SystemVerilog

- **bit (etc.):** offers more flexibility in bit manipulation.

```
bit [7:0] data;
shortint value;
```

- **enum:** enumeration type, for defining a set of named values.

```
typedef enum logic [1:0] {IDLE, BUSY, DONE}
    state_t;
state_t current_state;
```

- **typedef:** to define new data types.

```
typedef logic [7:0] byte_t;
byte_t data;
```

SystemVerilog also offers some advanced data types, such as **struct, union, array, queue**, etc. You will see them if they are used in the lab, we do not cover them here.

1.2 Combinational Logic

Combinational logic refers to circuits or logic blocks whose outputs depend entirely on their current inputs, without any form of internal storage or memory.

Operation	bit-wise	logic-wise
AND	&	&&
OR	|	||
NOT	~	!
XOR	^	
XNOR	^~	

Table 1: Combinational Logic Operations

```
logic [3:0] a = 4'b1010;
logic [3:0] b = 4'b1100;
logic [3:0] and_result, or_result, not_result;
logic and_z, or_z, not_z;

assign and_result = a & b; // result = 4'b1000
assign or_result = a | b; // result = 4'b1110
assign not_result = ~a; // result = 4'b0101
assign and_z = a && b; // z = 1'b1
assign or_z = a || b; // z = 1'b1
assign not_z = !a; // z = 1'b0
```

1.3 Sequential Logic

Sequential logic differs from combinational logic in that the output of a sequential circuit depends not only on its current inputs, but also on its past inputs or state. And sequential logic always requires a clock signal to control the timing of the circuit.

combinational logic:

```
module mux4to1 (
    input logic [3:0] data,
    input logic [1:0] sel,
    output logic out
);
    always @(*) begin
        case (sel)
            2'b00: out = data[0];
            2'b01: out = data[1];
            2'b10: out = data[2];
            2'b11: out = data[3];
        endcase
    end
endmodule
```

sequential logic:

```
module counter (
    input logic clk, reset,
    output logic [3:0] count
);
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            count <= 4'b0000;
        end else begin
            count <= count + 1;
        end
    end
endmodule
```