

SYSTEM VERILOG

# A 4X1 MULTIPLEXER



# SYSTEM VERILOG

## A 4X1 MULTIPLEXER

*This tutorial builds a 4X1 multiplexer using 2X1 multiplexer modules in System Verilog.*

Two levels of functionality are included:

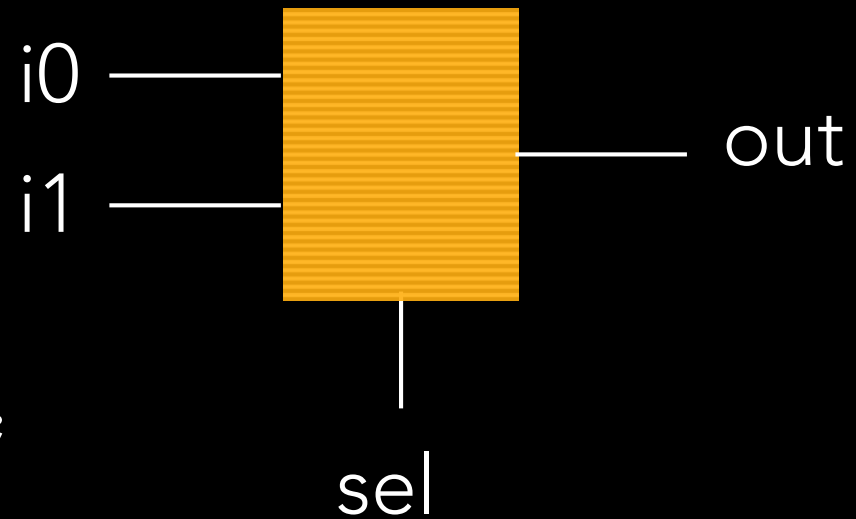
- Lower Level Module
  - 2X1 multiplexer (explained in detail in a previous tutorial)
- Higher Level Module
  - 4X1 multiplexer (constructed using 2X1 multiplexer modules)

*This tutorial also includes a testbench module which allows the 4X1 multiplexer to be simulated and its functionality verified in ModelSim.*

# SYSTEM VERILOG

## A 2X1 MULTIPLEXER

```
module mux2_1(out, i0, i1, sel);  
  
    output logic out;  
    input logic i0, i1, sel;  
  
    assign out = (i1 & sel) | (i0 & ~sel);  
  
endmodule
```



The mux2\_1 module has the functionality of a 2X1 multiplexer and is described in more detail in a previous tutorial. The 2X1 multiplexer has three inputs. If sel=0, the value of i0 is transferred to the output (out); If sel = 1, the value of i1 is transferred to the output (out).

# SYSTEM VERILOG

## A 4X1 MULTIPLEXER

```
module mux4_1(out, i00, i01, i1 1, i1 0, sel0, sel1);
```

```
output logic out;
```

```
input logic i00, i01, i1 1, i1 0, sel0, sel1;
```

```
wire out_0, out_1;
```

out\_0 and out\_1 are internal signals to the mux4\_1

```
mux2_1 a1(.out(out_0), .i0(i00), .i1(i01), .sel(sel0));
```

```
mux2_1 a2(.out(out_1), .i0(i1 0), .i1(i1 1), .sel(sel0));
```

```
mux2_1 a3(.out(out), .i0(out_0), .i1(out_1), .sel(sel1));
```

The functionality of the 4X1 multiplexer (mux4\_1) is generated by instantiating three 2X1 multiplexers (mux2\_1)

```
endmodule
```

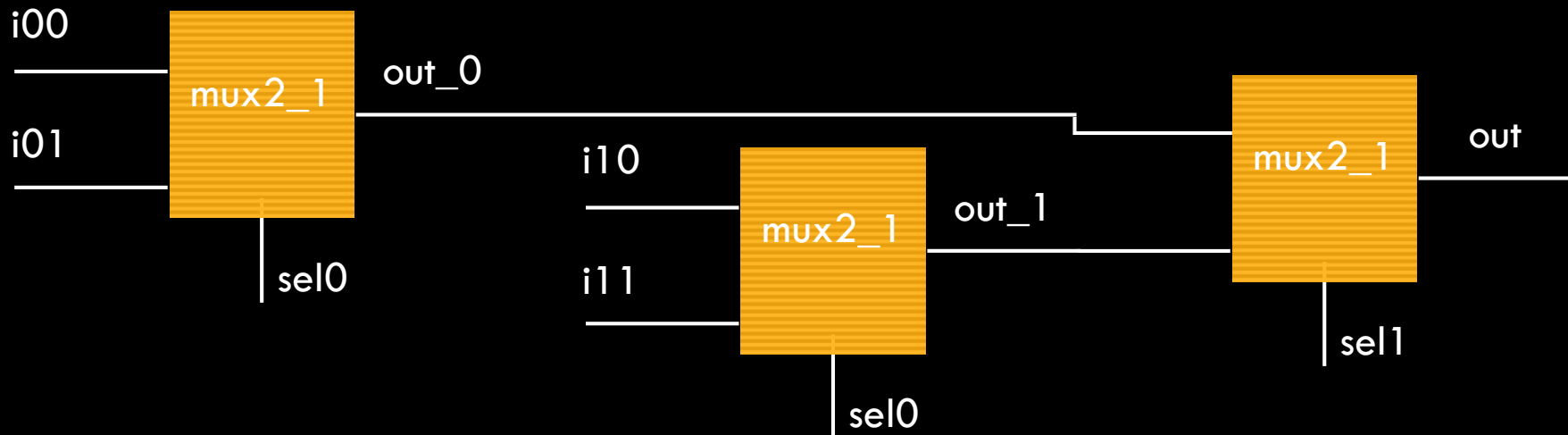
out is an output of the module mux4\_1. out is of the variable type logic and can have four possible values: z, x, 0, or 1

i00, i01, i1 1, i1 0, sel0, and sel1 are all inputs to mux4X1 and also are of the variable type logic

# SYSTEM VERILOG

## A 4X1 MULTIPLEXER

```
module mux4_1(out, i00, i01, i11, i10, sel0, sel1);  
    output logic out;  
    input logic i00, i01, i11, i10, sel0, sel1;  
    wire out_0, out_1;  
  
    mux2_1 a1(.out(out_0), .i0(i00), .i1(i01), .sel(sel0));  
    mux2_1 a2(.out(out_1), .i0(i10), .i1(i11), .sel(sel0));  
    mux2_1 a3(.out(out), .i0(out_0), .i1(out_1), .sel(sel1));  
endmodule
```





# SYSTEM VERILOG

## A 4X1 MULTIPLEXER

```
module mux4_1_testbench;
```

```
    logic out;
```

```
    logic i00, i01, i11, i10, sel0, sel1;
```

```
    mux4_1 dut (.out, .i00, .i01, .i11, .i10, .sel0, .sel1);
```

```
    initial begin
```

```
        sel0 = 0; sel1 = 0; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
        sel0 = 1; sel1 = 1; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
        sel0 = 1; sel1 = 0; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
        sel0 = 1; sel1 = 1; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
        sel0 = 1; sel1 = 1; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
        sel0 = 1; sel1 = 0; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
        sel0 = 0; sel1 = 1; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
        sel0 = 0; sel1 = 0; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
    end
```

```
endmodule
```

The testbench establishes the conditions by which the mux4\_1 that was created (functionalized) in the previous slide (via the module mux4\_1) can be tested in ModelSim; the testbench module is typically included in the same file as the functionalized module (which would be mux4\_1 in this case).

# SYSTEM VERILOG

## A 4X1 MULTIPLEXER

```
module mux4_1_testbench;
```

```
    logic out;  
    logic i00, i01, i11, i10, sel0, sel1;
```

Sets up the mux4\_1 for testing/simulating and names it dut (device under test). Unlike the previous slide, each input and output signal does not need a separate wire name (e.g. .out (out\_0)) because this module only tests mux4\_1 rather than instantiating it.

```
    mux4_1 dut (.out, .i00, .i01, .i11, .i10, .sel0, .sel1);
```

```
    initial begin
```

```
        sel0 = 0; sel1 = 0; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 1; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 0; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 1; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 1; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 0; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 0; sel1 = 1; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 0; sel1 = 0; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
    end
```

```
endmodule
```

# SYSTEM VERILOG

## A 4X1 MULTIPLEXER

```
module mux4_1_testbench;
```

```
    logic out;  
    logic i00, i01, i11, i10, sel0, sel1;
```

```
    mux4_1 dut (.out, .i00, .i01, .i11, .i10, .sel0, .sel1);
```


```
    initial begin
```

```
        sel0 = 0; sel1 = 0; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 1; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 0; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 1; i00 = 1; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 1; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 1; sel1 = 0; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 0; sel1 = 1; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;  
        sel0 = 0; sel1 = 0; i00 = 0; i01 = 1; i10 = 0; i11 = 1; #10;
```

```
    end
```

```
endmodule
```

This initial block sets up the testing of 8 of the 64 possible input (sel0, sel1, i00, i01, i10, i11) combinations to mux4\_1. The outputs of this testing sequence can be viewed in ModelSim. A delay of 10 time units occurs between each input combination (as indicated by #10 after each input combination).





# SYSTEM VERILOG

## A 4X1 MULTIPLEXER

In this tutorial, we have created a 4X1 multiplexer in System Verilog using 2X1 multiplexers and created a testbench to simulate a subset of input combinations in ModelSim.

