Building hierarchical designs in Verilog

Opriţoiu Flavius
flavius.opritoiu@cs.upt.ro

October 16, 2024

Verilog hierarchies

Objectives:

- Learn how to instantiate a module
- Construct a design hierarchy

Hierarchical design

- Facilitates design of complex architectures
- Promote design reuse

Instance: a copy of a module used as component in a larger design.
An instances has:

- A *module*: provides the definition of the instance.
- A *container*: the design in which the instance is created.

Creating a new instance is referred to as *instantiation*.

Instantiation

An instance is constructed by providing:

1. the name of the *module* to be instantiated
2. the *name of the instance* (differentiate from other instances of the same module)
3. the *list of connections*

List of connections specifies which signals from container connects to which ports of the instance.

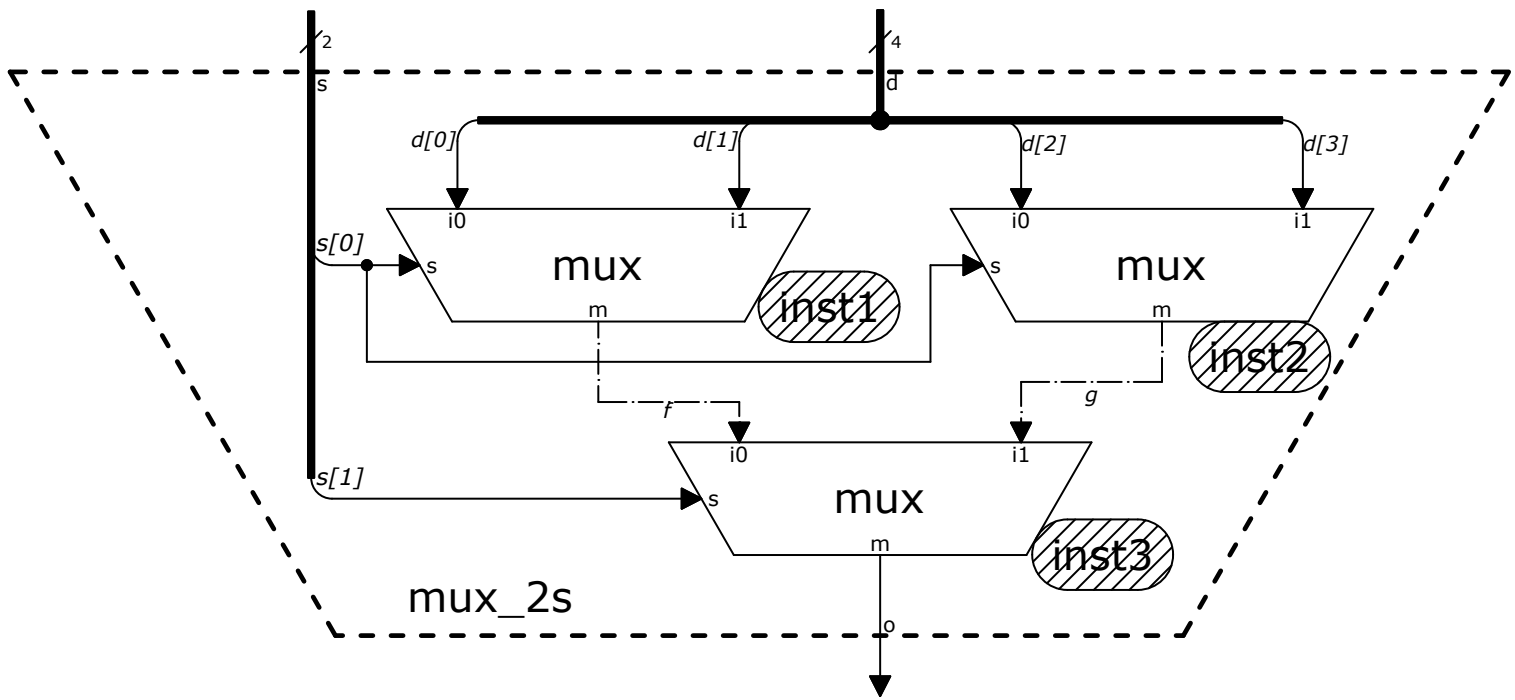
A connection is specified by:
`.<module_port>(<container_signal>)`, in which

- *module_port* is a port of the instance
- *container_signal* is a signal from container (can be a port of container)

4-to-1 multiplexer

Exercise: Build a 4-to-1 multiplexer out of 2-to-1 multiplexers.

Solution: The 4-to-1 multiplexer, *mux_2s*, uses the 2-to-1 multiplexer module, *mux*, having the inputs and outputs as depicted in the architecture bellow:



4-to-1 multiplexer (contd.)

The Verilog code implementing the previous architecture:

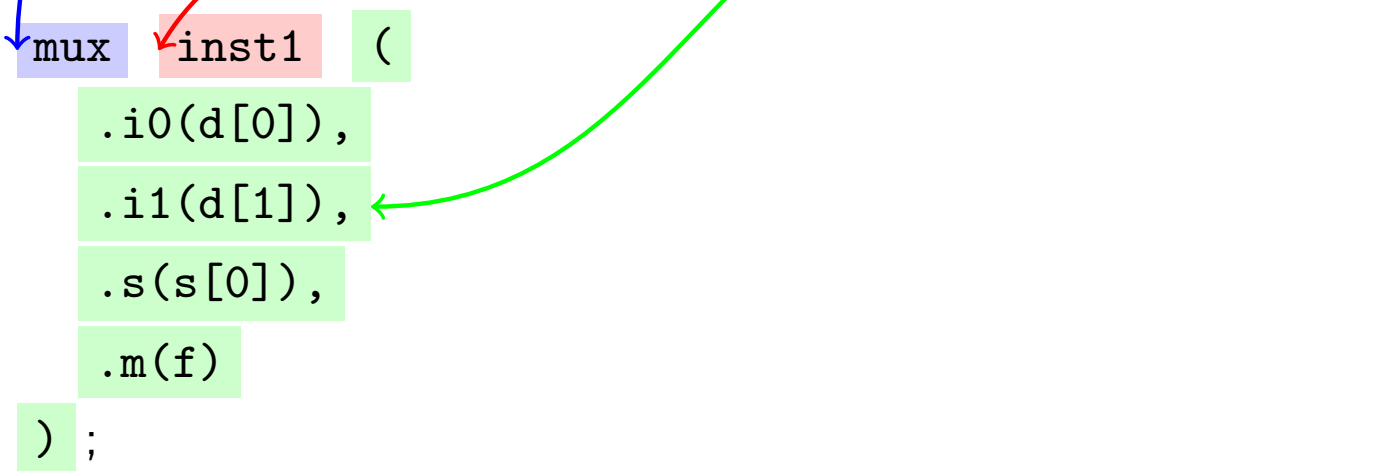
```
1  module mux (
2    input s, i1, i0,
3    output m
4  );

6    always @ (*)
7      if (s) m = i1;
8      else m = i0;
9  endmodule

11 module mux_2s (
12   input [3:0] d,
13   input [1:0] s,
14   output o
15 );

17   wire f;
18   wire g;
```

```
20     mux inst1 (
21       .i0(d[0]),
22       .i1(d[1]),
23       .s(s[0]),
24       .m(f)
25     );
26     mux inst2 (
27       .i0(d[2]),
28       .i1(d[3]),
29       .s(s[0]),
30       .m(g)
31     );
32     mux inst3 (
33       .i0(f),
34       .i1(g),
35       .s(s[1]),
36       .m(o)
37     );
38 endmodule
```

4-to-1 multiplexer (contd.)

The instance from line 20-25 of the code in previous slide has the following components:

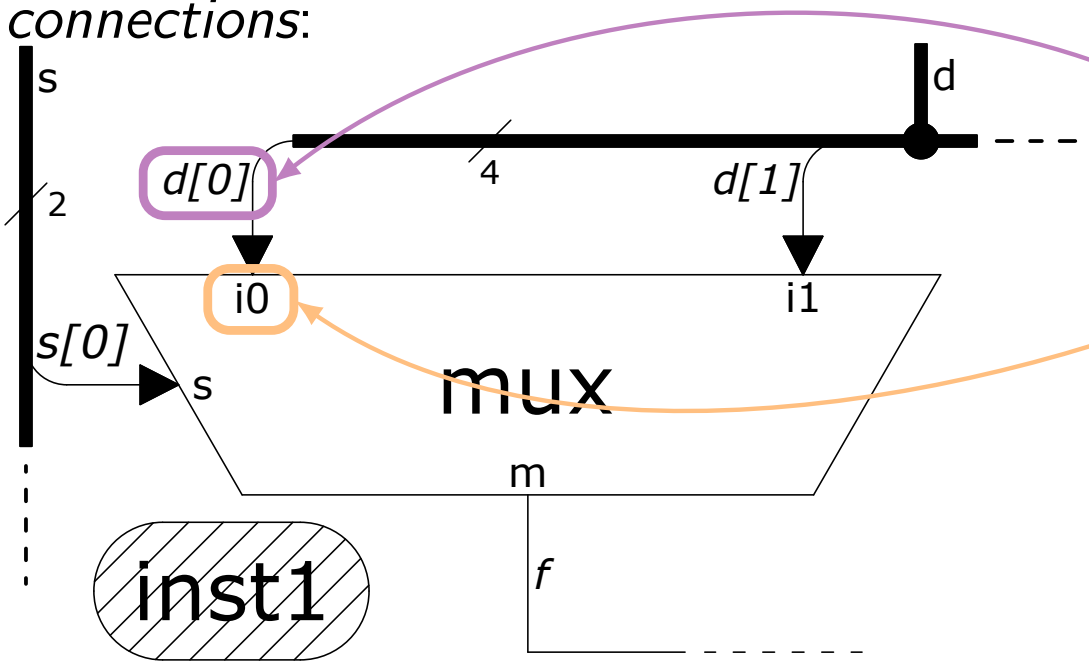
- the *module* to be instantiated is `mux` (it must be the name of an existing Verilog module)
- the *instance name* is *inst1*
- the *list of connections*, within round parentheses (more details in the next slide)



```
20  mux  inst1  (
21      .i0(d[0]),
22      .i1(d[1]),
23      .s(s[0]),
24      .m(f)
25  ) ;
```

4-to-1 multiplexer (contd.)

Block diagram details emphasizing the *inst1* instance and its *list of connections*:




```
20  mux inst1
21      (.i0(d[0]),
22       .i1(d[1]),
23       .s(s[0]),
24       .m(f)
25  );
```

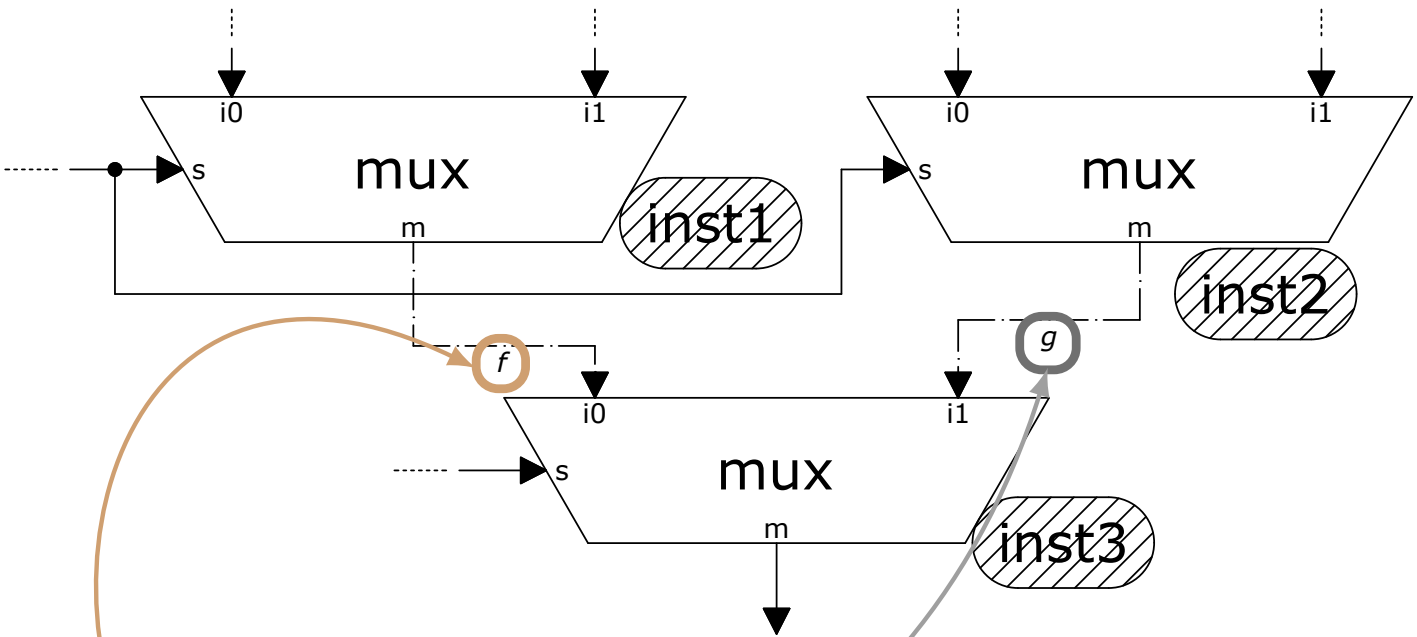
Elements of the first connection:

- i0 - port of the instantiated module
- d[0] - signal from container (d[0] is a port in container) to which port i0 is connected

4-to-1 multiplexer (contd.)

Internal wires connect internal instances

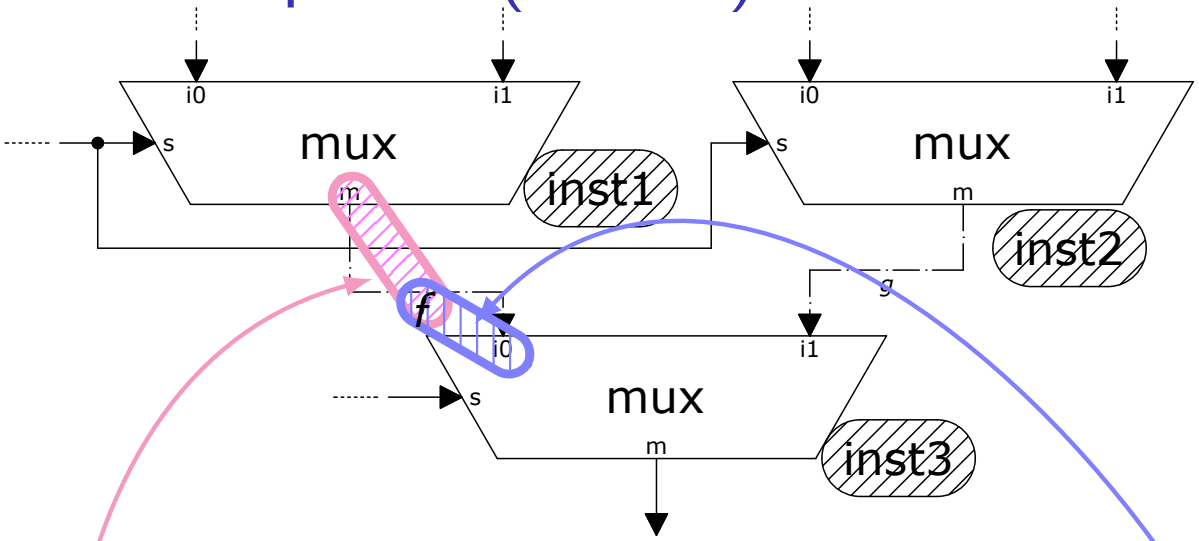
- connect one instance's output to another instance's input
- must be declared inside the container module as *wire*s



Internal wire declaration for the `mux_2s` architecture:

```
17  wire f;
18  wire g;
```

4-to-1 multiplexer (contd.)



Verilog code bellow

- connects *inst1*'s output, *m*, to wire *f* (left)
- connects *inst3*'s input, *i0*, to wire *f* (right)

Note: The widths of the port and connecting wire must match!

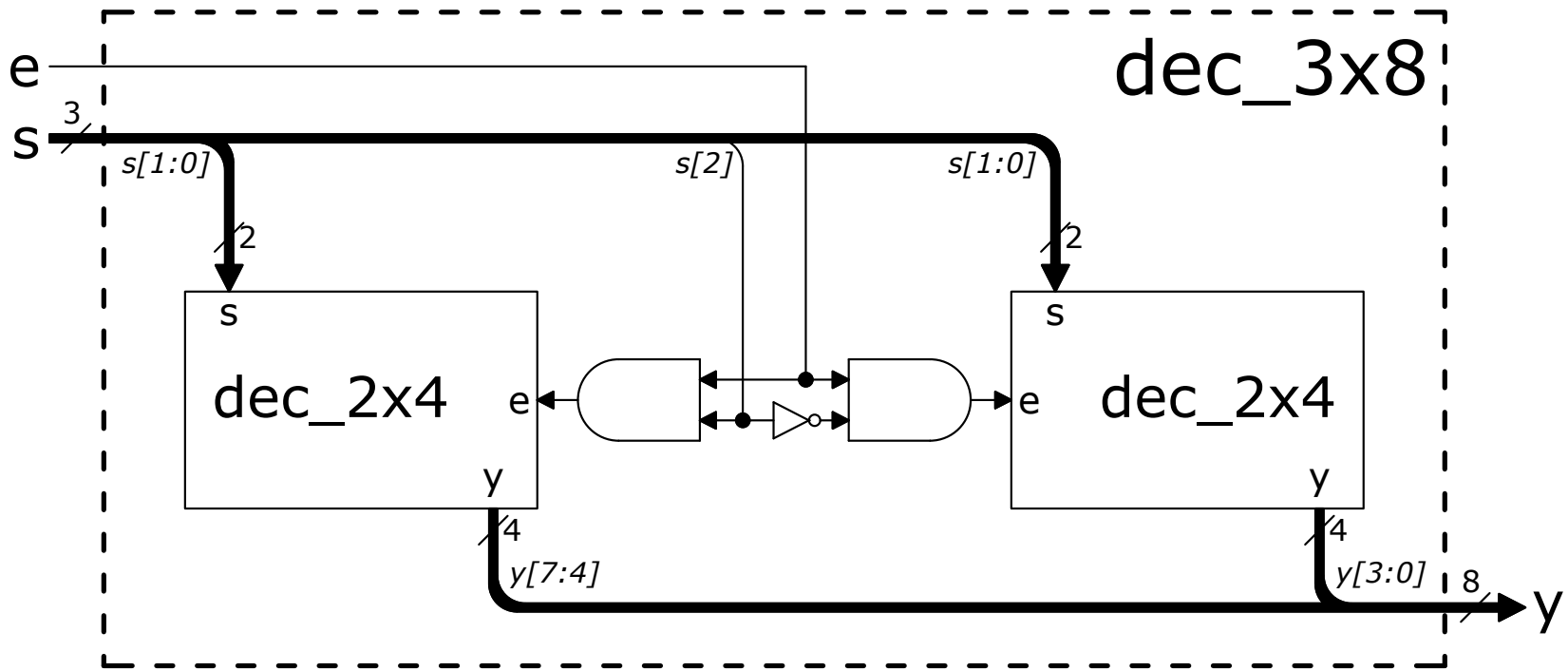
```
20  mux inst1 (
21      .i0(d[0]),
22      .i1(d[1]),
23      .s(s[0]),
24      .m(f)
25  );
```

```
32  mux inst3 (
33      .i0(f),
34      .i1(g),
35      .s(s[1]),
36      .m(o)
37  );
```

Three-to-eight-lines decoder with enable input

Exercise: Build a 3-selection lines decoder with enable input and active low outputs using the *dec_2x4* module from here (slide 12).

Solution: The architecture is depicted bellow.



Three-to-eight-lines decoder with enable input (contd.)

Verilog code implementing the architecture from previous slide:

```
1  module dec_3x8 (
2    input e,
3    input [2:0] s,
4    output [7:0] y
5  );

7    dec_2x4 i1 (
8      .s(s[1:0]),
9      .e(e & s[2]),
10     .y(y[7:4])
11   );

13   dec_2x4 i2 (
14     .s(s[1:0]),
15     .e(e & (~s[2])),
16     .y(y[3:0])
17   );
18 endmodule
```