W3. Designing hierarchical architectures

Instantiating modules in Verilog language

1. Objectives

O1. Learning the technique of instantiating a module in Verilog language

O2. Implementation of a decoder device in Verilog HDL

O3. Building a hierarchical design

2. Theoretical Background

2.1 Hierarchical Design

We will emphasize that a hierarchical design::

- facilitates the design of complex architectures
- encourages the reuse of the designed design

2.1.1 Instance

The instance can be defined as a module used as a component in a more comprehensive module.

An instance has:

- A **module**: provides the definition for the instance.
- A **recipient**: the module in which the instance is created.

The procedure by which we create a new instance is called **instantiation**.

An instance is built by the following elements:

- the module to be instantiated, followed by
- the name of the instance (which differs from other instances of the same module), followed by a list of ports.

The list of ports specifies which signals from the container are linked to the instance ports.

A port connection is defined most simply:

`.<module_port>(<container_signal>),` in which:

- **module_port** is a port of the instantiated module.
- **container_signal** is an internal signal or a container port.

2.1.2 The architecture of a 4-to-1 multiplexer

Exercise:

Build a 4-to-1 multiplexer by using instances of 2-to-1 multiplexers.

Solution:

The architecture of a 4-to-1 multiplexer using three 2-to-1 multiplexer modules is illustrated below. The data input, ***d***, is on 4 lines and the selection line ***s***, on 2 bits, propagates 1 bit from input ***d*** to output ***o***.

For example, if ***s*** = (1,0), the line ***d [2]*** is propagated to ***o***.

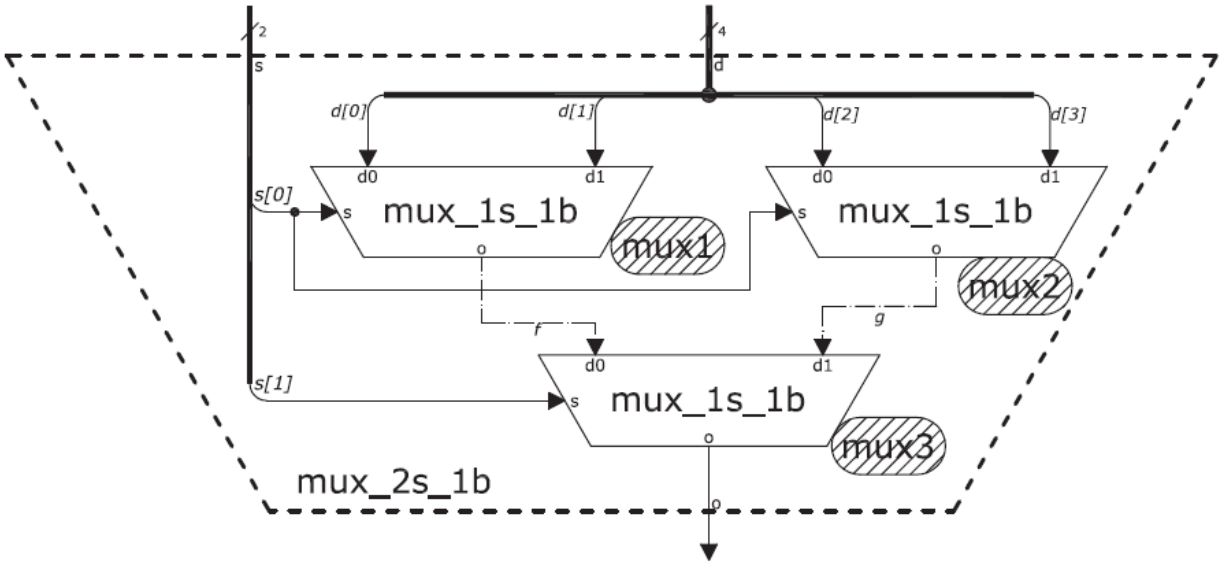

Fig. 1 – Hierarchical Design of a 4-to-1 multiplexer

The Verilog code that implements the architecture of a 4-to-1 multiplexer is presented below:

```
module mux_1s_1b (
input [3:0] d,
input [1:0] s,
output o
);

wire f;
wire g;

mux_1s_1b mux1 (
.d0(d[0]) ,
.d1(d[1]) ,
.s(s[0]) ,
.o(f)
);

mux_1s_1b mux2 (
.d0(d[2]) ,
.d1(d[3]) ,
.s(s[0]) ,
.o(g)
);

mux_1s_1b mux3 (
 .d0(f) ,
.d1(g) ,
.s(s[1]) ,
.o(o)
);
endmodule
```

By carefully analyzing the first instance created in the 4-to-1 multiplexer code, we can observe the components of an instantiation:

- the module to be instantiated is **mux_1s_1b** (must be the name of an existing Verilog module)
- the instance name is **mux1** (must be a valid Verilog name)
- list of connection ports, positioned between **round brackets** (more details below on the current page)

```
mux_1s_1b mux1 (
.d0(d[0]),
.d1(d[1]),
.s(s[0]),
.o(f)
);
```

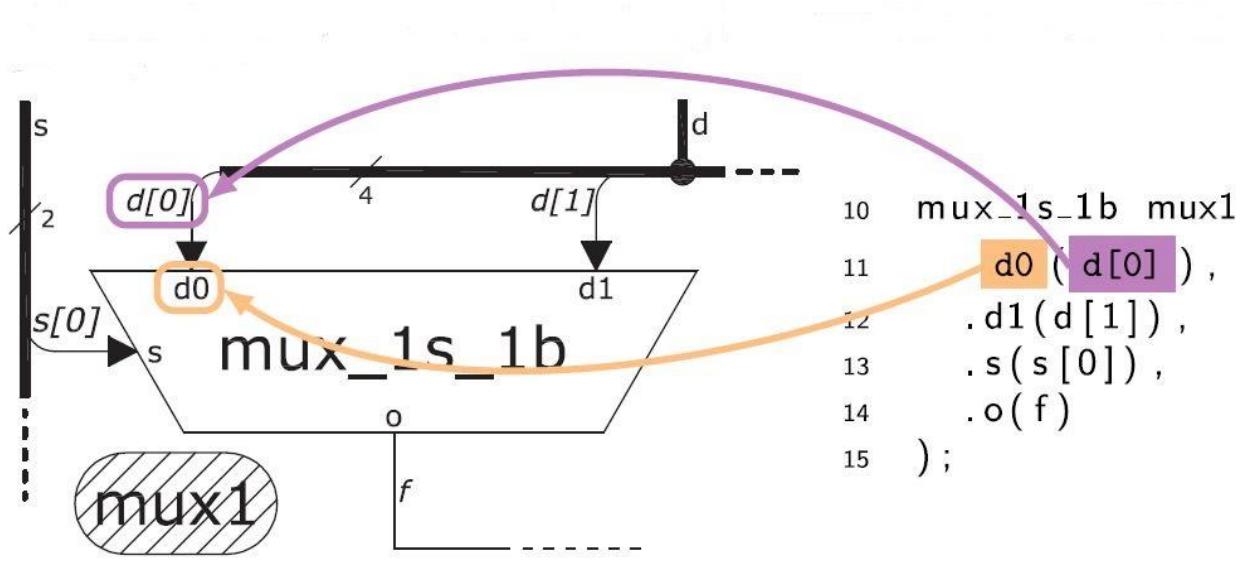

Fig. 2 – Module mux1 Instantiation

The elements of the first connection port are::

- **d0** – port of the mux_1s_1b module to be associated with
- **d[0]** – signal inside the container mux_2s_1 (d[0] is a port of a container, thus a signal in the container) to which d0 is connected.

Internal **wire** type links connect internal instances:

- connects an exit of the instance to an input of another instance.
- must be declared in the module container as ***wire*** elements

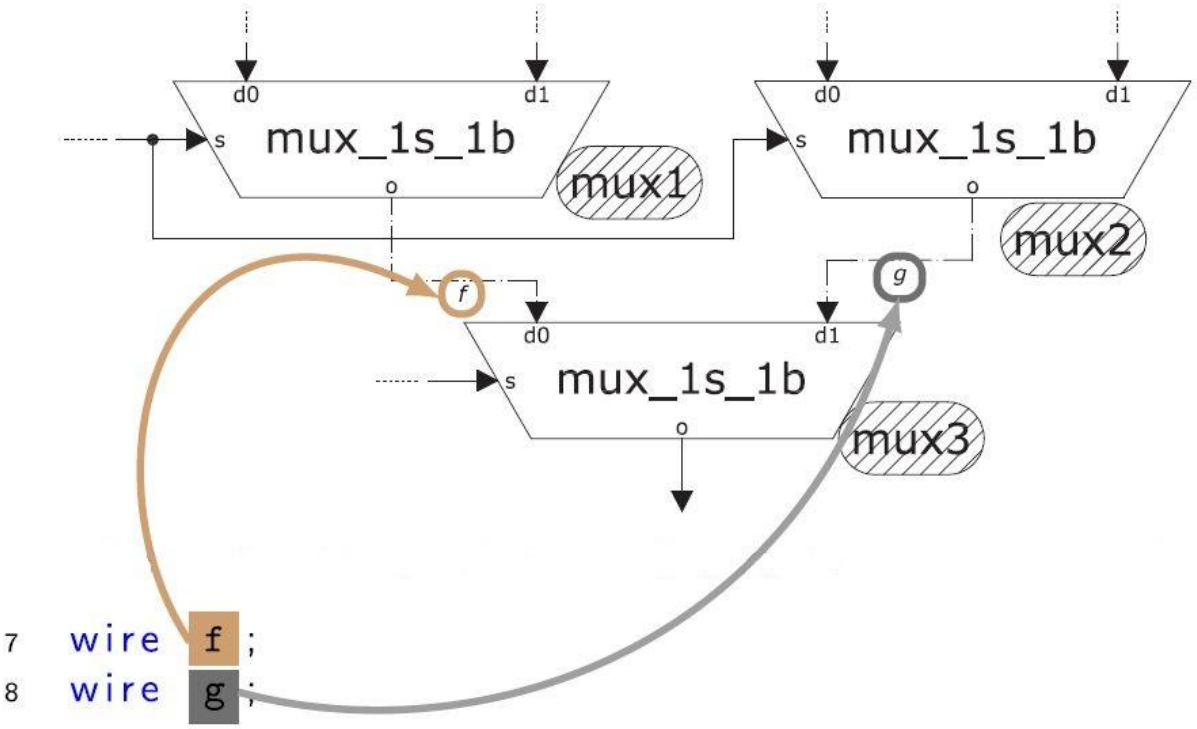

Fig. 3 – Declaring the internal physical wires of the architecture mux_2s_1b

Important: When connecting a wire to a port, their widths (dimensions) must be identical!

If the two ports to be connected have different sizes, choose a width of one of them for the size of the physical wire to be connected. To connect the wire to the desired port:

- if the physical wire has fewer lines and the other port is an output, some of the output lines will remain unconnected
- if the physical wire has fewer lines and the port to which it connects is an input, use the concatenation operation to provide the required input lines
  ex: **`.x({4'd0, data}),`**
- • if the wire has multiple lines and the opposite port is an input, use part-select to provide only the required input lines
  ex: **`.s(addr[15:14]),`**
- if the wire has multiple lines and the port it connects to is an output, use part-select to provide the required output lines
  ex: **`.z(next_addr[7:0]),`**

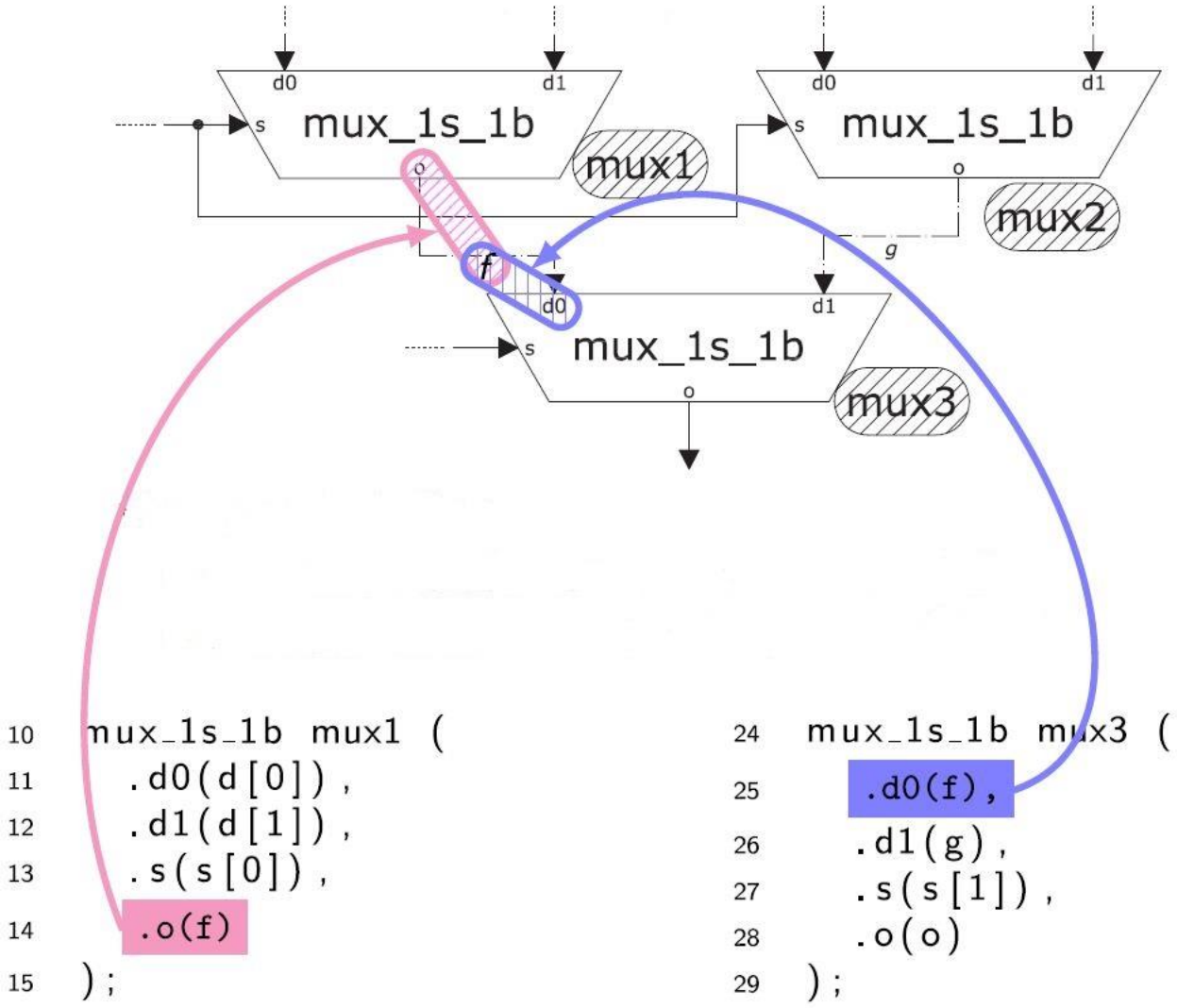

Fig. 4 – Connecting inputs and outputs to declared physical wires

The Verilog code illustrated above:

- connects the output o of ***mux1*** to wire ***f*** (see left)
- connects the ***d0*** input of ***mux3*** to wire ***f*** (see right side)