W5. Design of reusable modules

Implementing parameterized modules in Verilog

1. Objectives

O1. Learning how to separate *datapath* and *control path* instances

O2. Introduction of the concept of reusable mode

O3. Learn how to write parameterized modules

2. Theoretical Background

Datapath

The **Datapath** instance consists of elements that process the data: no decisions are made. Typical components here are:

- multiplexers (MUX)
- registers (REG)
- arithmetic-logical units (ALU)

In addition, the Datapath instance builds common buses using three-state logic gates.

Control path

The **Control path** involves decision making and is described in terms of machines with states.

Note: Components with validation states (such as registers, counters) can also be part of the datapath.

Reusable Modules

Reusable modules are defined as modules which integrate **parameters**, that can be redefined. In the Verilog 2001 standard, the parameters of the module are specified in a dedicated section, marked by the symbols # (s, i). The code below describes a parallel register, with parameterizable width (number of bits) and initialization vector (contents of the register after reset).

```
module rgst #(
   parameter w = 8, //register width parameter, with default value 8
   parameter iv  = {w{1'b0}} //initialization vector parameter
)(
input clk , //clock signal
input rst_b , // asynchronous reset; active on low
input [w-1:0] d , // input data on w bits
input ld , // synchronous  load; active on high
input clr , // synchronous clear; active on high
output reg [w-1:0] q // the contents of the register, on w bits
);
always  @  ( posedge clk, negedge rst_b )
begin
             if (! rst_b)
                q <= iv; // setting the content to the initialization value
           else if (clr)
                q <= iv; // setting the content to the initialization value
           else if ( ld )
            q <= d;
end
endmodule
```

3. Applications

Exercise: Build a 4 × 8 register file.

Solution: An $M \times N$ **register file** is a storage element organized as a vector of M registers, each register having N bits. Allows you to simultaneously read an internal register and write an internal register (possibly the same).

The interface of a register file includes the following connections:

- An N-bit data entry for writing internal registers (*wr_data*)
- N-bit data output for reading internal registers (*rd_data*)
- An input address, for selecting the register to be written (*wr_addr*)
- An output address, for selecting the register to be read (*rd_addr*)
- Write validation signal (*wr_e*)
- Read validation signal (*rd_e*)

The validation lines for the write/read port are optional, ***M*** generally takes the form of 2^k: the input/output addresses use ***k*** bits.

The interface of a 4 × N register file is shown below:

Fig. 1 – Structure of a register file

For the presented scenario, the interface consists of:

- Writing port (*wr_data, wr_addr, wr_e*)
- Read port (rd_data, rd_addr, rd_e)
- The clock signal (*clk*)
- The reset signal (*rst_b*)

A register file 4×8 without validation line on the output:

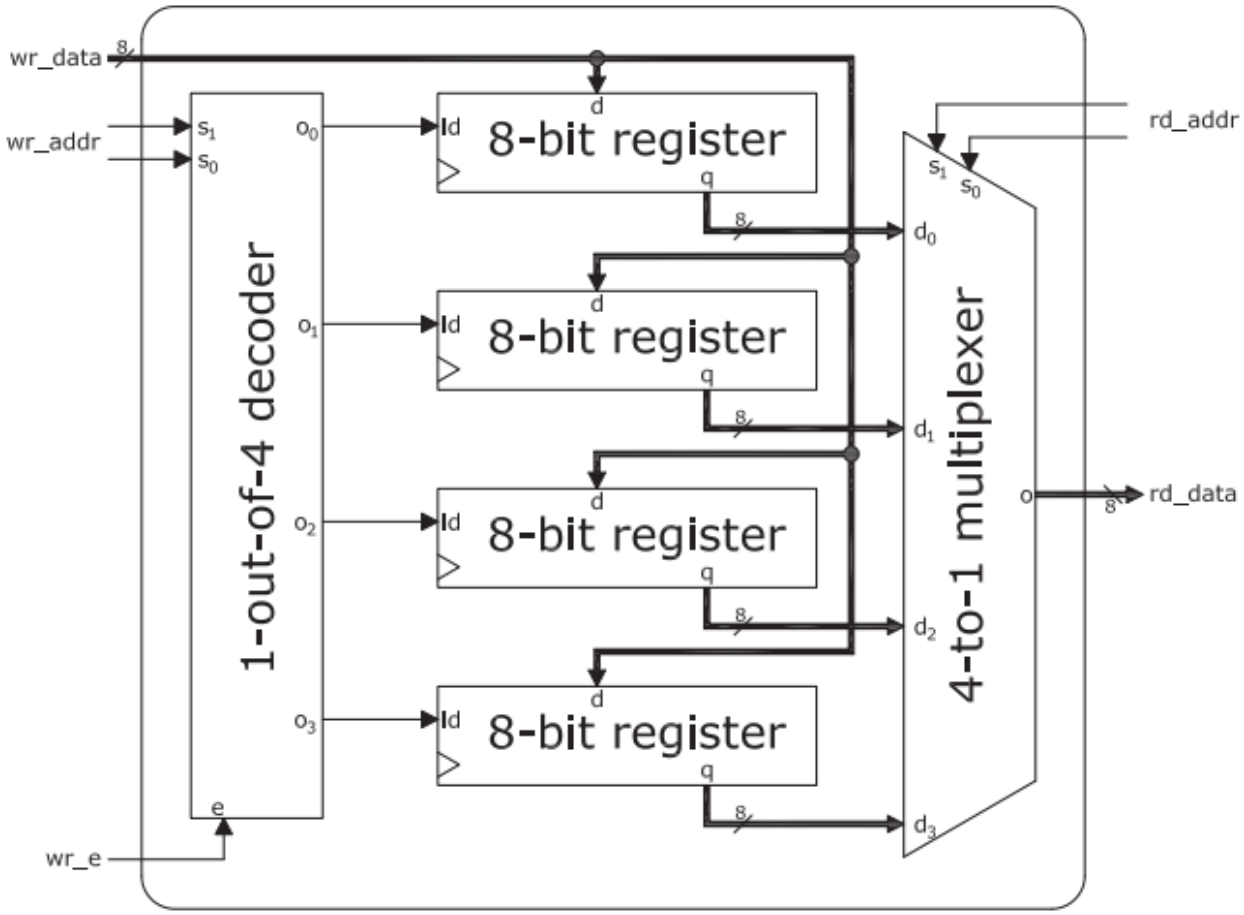
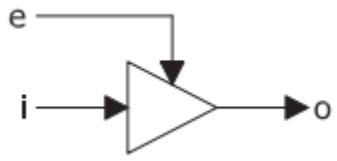


Fig. 2 – Architecture of a register file

Note: The *clock* and *reset* lines have been omitted for clarity.

Three-state gates



It is used to connect several components on a common line or bus.

Output ***o*** is set to ***i*** when the validation line is active, and on **high impedance** in the opposite scenario. An output set to high impedance (symbolized ***z***, in Verilog) allows another component connected to the same line (or bus) to set the line value.

The code fragment below exemplifies the control of a signal to high impedance, using a control line ***e***:

```
wire [15:0] data, data_hiz ;

assign data_hiz = (e) ? data : 16'bz
```

Since the high impedance symbol z represents the most significant bit of the constant on line 2, it will extend by 16 bits z.

Construction of a multiplexer using three-state gates

A 4-to-1 ***n***-bit multiplexer is implemented using three-state logic gates:

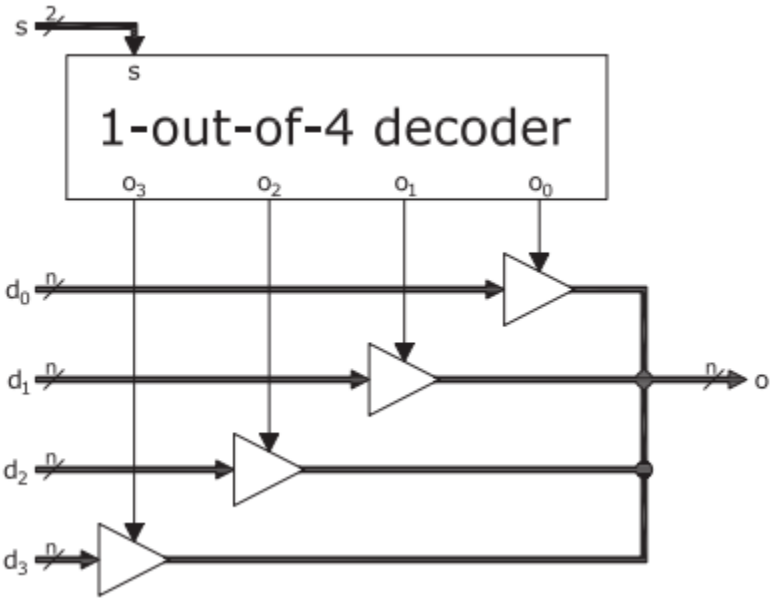


Fig. 3 – The architecture of a multiplexer **implemented with trivalent gates** and a decoder

Observation:
In general, explicitly redefining the parameters of a module in Verilog 2001 uses the following format:

```
module-name #( .parameter-name(value) , ... )
  instance_name ( .port-name(signal) ,  ... )
```

The code on the next page instantiates a 16-bit register, with an initialization value set to 0.

```
rgst #(
    .w(16)
) registru1 (
    .clk(clk), ...
);
```

Another code below instantiates a 4-bit register with an initialization value set to 15.

```
rgst #(
    .w(4)
    .iv(4'd15)
) registru2 (
    .clk(clk), ...
);
```