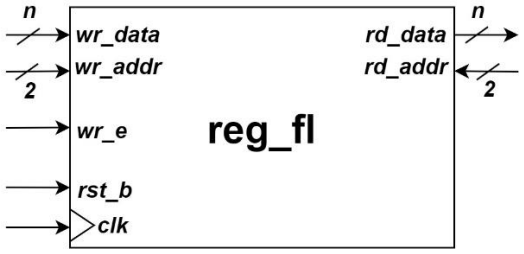
Week 5 – Design of reusable modules

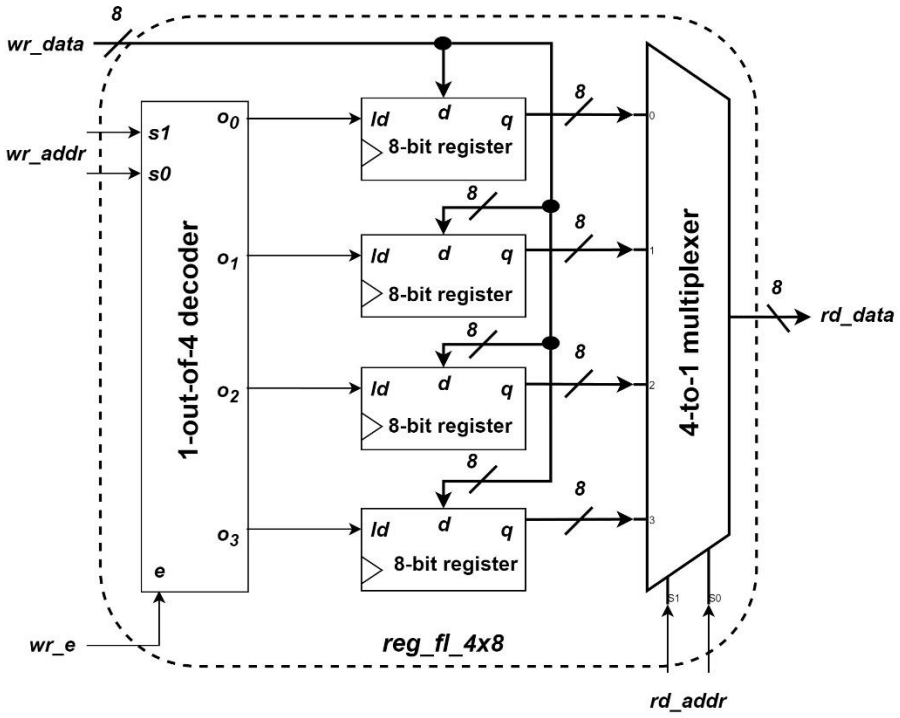
Parameterized modules in Verilog

The Datapath

Consider the architecture of a Register File, denoted as ***reg_fl***, with the ports ***wr_data*** and ***rd_data*** on ***n*** bits, the ***wr_addr*** and ***rd_addr*** inputs on 2 bits, respectively the 1-bit signals ***ld***, ***clr***, ***rst***, and ***clk***. The module's graphical representation is illustrated below:



The detailed architecture of the Register File is provided as follows:



P.5.1. Design, using Verilog language, the decoder module depicted on the previous page, which has a 2-bit selection line (***s***), one enable line (***e***) on 1-bit, and an output on 4 bits, denoted by ***o***. The module's name will be ***dec_2x4***.

a) Draw the truth table of the **1-out-of-4 decoder** module.

b) Implement the module using a ***continuous assignment*** command.

c) Construct the same module using a ***combinational always*** block with ***case statement***.

Solution:

a)

Enable	***Selection***		***Output***			
e	**s[1]**	**s[0]**	**o[3]**	**o[2]**	**o[1]**	**o[0]**
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0
0	d	d	0	0	0	0

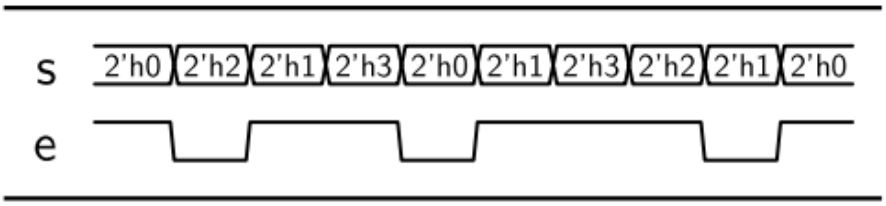
b)

```
module dec_2x4 (
    input [1:0] s,
    input e,
    output [3:0] o
);
assign o[0] = e & (~s[1]) & (~s[0]);
assign o[1] = e & (~s[1] & s[0];
assign o[2] = e & s[1] & (~s[0]);
assign o[3] = e & s[1] & s[0];
endmodule
```

c)

```
module dec_2x4 (
    input [1:0] s,
    input e,
    output reg [3:0] o
);
always @ (*) begin
case {e,s}
3'b100 : o = 4'b0001;
3'b101 : o = 4'b0010;
3'b110 : o = 4'b0100;
3'b111 : o = 4'b1000;
3'b0?? : o = 4'b0000;
encase
end
endmodule
```

P.5.2. Design, using Verilog language, a ***testbench*** for non-exhaustive checking of the ***dec_2x4*** module implemented in ***P.5.1***. Generate the signals according to the timing diagram offered below as a support.




```
module dec_2x4 (
    output reg [1:0] s,
    output reg e,
    output [3:0] o
);
```

```
// Testbench instantiation
dec_2x4 DUT ( .s(s), .e(e), .o(o) );
// Selection signal generation
initial begin
s = 2'h0;
# 20 s = 2'h2;
# 20 s = 2'h1;
# 20 s = 2'h3;
# 20 s = 2'h0;
# 20 s = 2'h1;
# 20 s = 2'h3;
# 20 s = 2'h2;
# 20 s = 2'h1;
# 20 s = 2'h0;
end
// Enable signal generation
initial begin
e = 1'b1;
# 20 e = 1'b0;
# 20 e = 1'b1;
# 40 e = 1'b0;
# 20 e = 1'b1;
# 60 e = 1'b0;
# 20 e = 1'b1;
end
endmodule
```

P.5.3. Design, using Verilog language, a **parameterized module** for the register depicted in the initial architecture, having an 8-bit input and output, respectively the ***load***, ***clear***, ***clock***, and ***reset*** signals on 1 bit.

Solution:

```
module rgst # (
parameter w = 8,
parameter iv = {w{1'b0}}
)(
input clk, rst_b, clr, ld,
input [w-1:0] d,
output reg [w-1:0] q
);
always @ ( posedge clk, negedge rst_b )
begin
if (! rst_b || clr)
q <= iv;
else if ( ld )
q <= d;
end
endmodule
```

P.5.4. Construct a parameterized 4-to-1 multiplexer, called ***mux_2s***. The module will be parameterizable by its width, having the following interface:

```
module mux_2s #(
        parameter w = 4
)(
        input [w-1:0] d0,d1,d2,d3,
        input [1:0] s,
        output [w-1:0] o
);
```

Solution:

```
module mux_2s # (
parameter k = 8
)(
input [1:0] s,
input [k-1:0] d0, d1, d2, d3,
output [w-1:0] o
);
assign o = s[1] ? (s[0] ? d3 : d2) : (s[0] ? d1 : d0);
endmodule
```

Alternative:

```
module mux_2s # (
parameter k = 8 )(
input [1:0] s,
input [k-1:0] d0, d1, d2, d3,
output reg [w-1:0] o );
always @ (*) begin
 if (s==2'd3)
      o = d3;
else if (s==2'd2)
      o = d2;
else if (s==2'd1)
      o = d1;
else o = d0;
end
endmodule
```

P.5.5. Design the Register File named ***reg_fl*** by instantiating the ***dec_2x4*** module, the ***rgst*** components, and the ***mux_2s*** unit, accordantly.

Solution:

```
module reg_fl (
input [7:0] wr_data,
input [1:0] wr_addr, rd_addr,
input wr_e, clk, rst_b,
output [7:0] rd_data
);
wire [3:0] w;
wire [7:0] q0, q1, q2, q3;
// Decoder Instantiation
dec_2x4 decoder ( .s(wr_addr), .e(wr_e), .o(w) );
// Registers Instantiation
rgst # ( .w(8) ) register1 ( .d(wr_data), .ld(w[0]),
.clk(clk), .rst_b(rst_b), .clr(1'b0), .q(q0) );
rgst # ( .w(8) ) register2 ( .d(wr_data), .ld(w[1]),
.clk(clk), .rst_b(rst_b), .clr(1'b0), .q(q1) );
rgst # ( .w(8) ) register3 ( .d(wr_data), .ld(w[2]),
.clk(clk), .rst_b(rst_b), .clr(1'b0), .q(q2) );
rgst # ( .w(8) ) register4 ( .d(wr_data), .ld(w[3]),
.clk(clk), .rst_b(rst_b), .clr(1'b0), .q(q3) );
// Multiplexer Instantiation
mux # ( .k(8) ) multiplexer ( .s(rd_addr), .d0(q0),
.d1(q1), .d2(q2), .d3(q3), .o(rd_data) );

endmodule
```