Laborator S6 AC

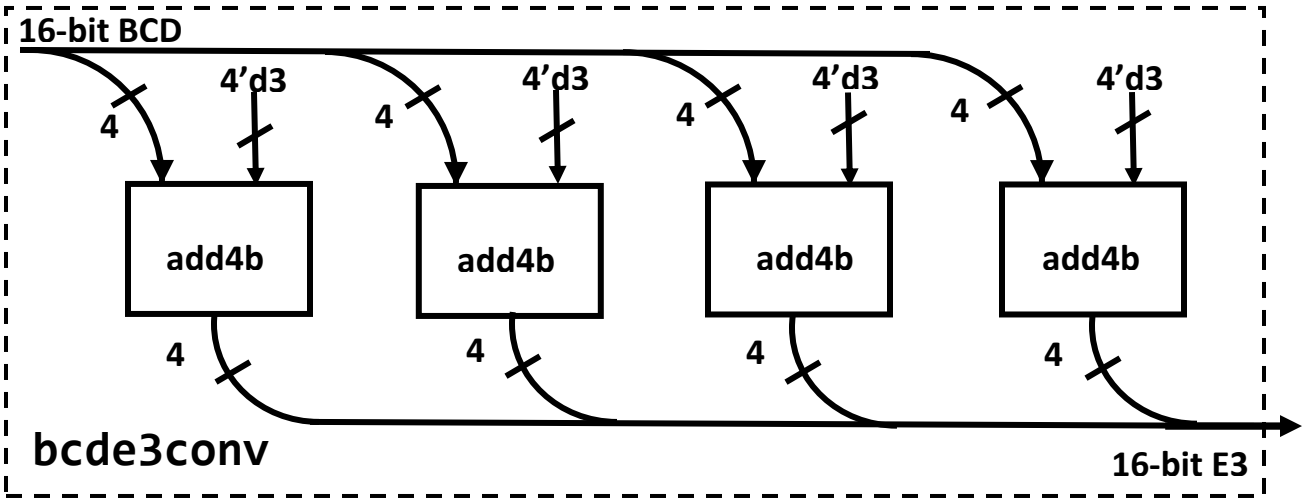
Construirea instanţelor multiple în limbajul Verilog

P.6.1 Să se proiecteze arhitectura unui convertor ***BCD8421*** la ***E3*** pentru numere cu ***k*** cifre, folosind ***k*** instanțe ale unui sumator pe 4 biți, numit ***add4b***. Implementați design-ul în limbajul Verilog, folosind:

a) ***vectori de instanțe***

b) ***un bloc generate***

Soluțiile la subpunctele a) și b) sunt furnizate în S6_Lab_AC din directorul Suport Laborator.

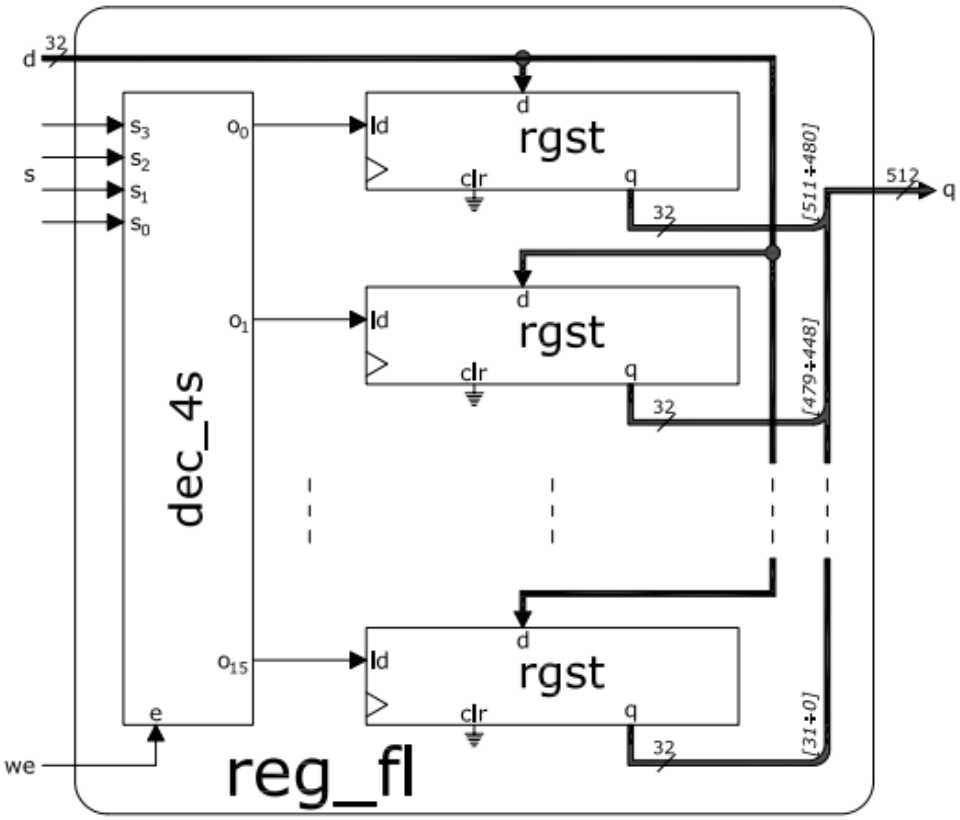


P.6.2 Să se proiecteze arhitectura unui ***Register File*** *16 x 32*, așa cum este ilustrată mai jos. Modulul ***reg_fl*** va primi la intrare pachete de date pe 32 biți iar la ieșirea sa va livra un bloc de date pe 512 biți format din concatenarea tuturor ieșirilor registrelor interne ale architecturii.

a) Implementați modulul *dec_4s*, folosind limbajul Verilog.

b) Implementați modulul parametrizat *rgst*, folosind limbajul Verilog.

c) Folosind metoda instanțierii multiple, construiți *reg_fl*.



a) Modulul pentru dec_4s

```
module dec_4s ( input [3:0] s,
                input e,
                output reg [15:0] o );
  always @(*) begin
    case ({e, s})
    5'b10000: o = {15'b0, 1'b1};
    5'b10001: o = {14'b0, 1'b1, 1'b0};
    5'b10010: o = {13'b0, 1'b1, 2'b0};
    5'b10011: o = {12'b0, 1'b1, 3'b0};
    5'b10100: o = {11'b0, 1'b1, 4'b0};
    5'b10101: o = {10'b0, 1'b1, 5'b0};
    5'b10110: o = {9'b0, 1'b1, 6'b0};
    5'b10111: o = {8'b0, 1'b1, 7'b0};
    5'b11000: o = {7'b0, 1'b1, 8'b0};
    5'b11001: o = {6'b0, 1'b1, 9'b0};
    5'b11010: o = {5'b0, 1'b1, 10'b0};
    5'b11011: o = {4'b0, 1'b1, 11'b0};
    5'b11100: o = {3'b0, 1'b1, 12'b0};
    5'b11101: o = {2'b0, 1'b1, 13'b0};
    5'b11110: o = {1'b0, 1'b1, 14'b0};
    5'b11111: o = {1'b1, 15'b0};
    5'b0????: o = 16'b0;
    endcase
  end
endmodule
```

b) Modulul parametrizat pentru registru

```
module rgst # (
        parameter w = 8, // parametrul de latime a registrului, valoarea implicita 8
        parameter iv = {w{1'b0}} // initializarea valorii parametrului
)(
        input clk, // semnalul de tact
        input rst_b,//reset asincron, activ pe low
        input [w-1:0] d, //datele de intrare pe w biti
        input ld, //load sincron, activ pe high
        input clr,//clear sincron, activ pe high
        output reg [w-1:0] q//continutul registrului, pe w biti
);


always @ ( posedge clk, negedge rst_b)
        if(! rst_b)
                q <= iv;
        else if(clr)
                q <= iv;
        else if(ld)
                q <= d;
endmodule
```

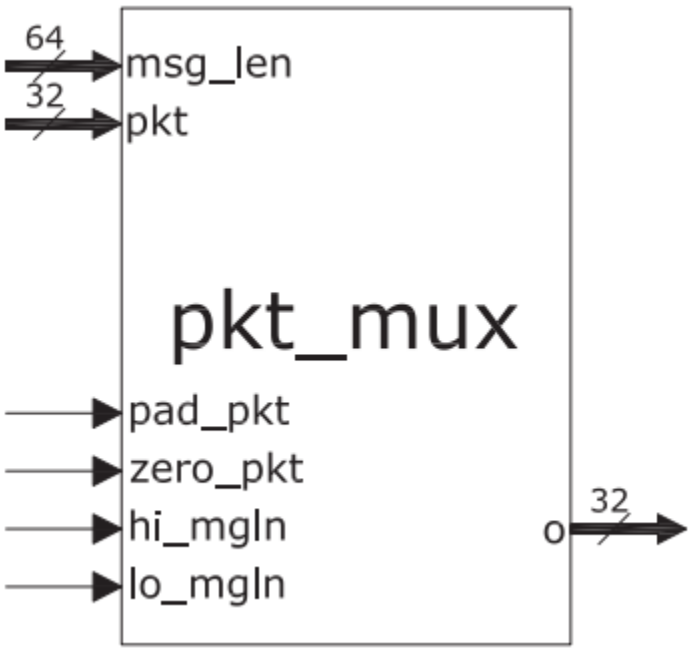
c) Modulul pentru Register File 16x32 cu bloc generate

```
module reg_fl( input clk, rst_b,
                input [31:0] d,
                input [3:0] s,
                input we,
        output [511:0] q );
wire [15:0] r_ld;
dec_4x16 u_dec ( .s(s), .e(we), .o(r_ld) );
generate
        genvar i;
                for( i=0; i<16; i=i+1 )
        begin: arr
        rgst #(
                .w(32)
        ) u_rgst(
                .clk(clk),
                .rst_b(rst_b),
                .d(d),
                .clr(1'd0),
                .ld(r_ld[i]),
                .q(q[i*32+31 : i*32])
        );
        end
endgenerate
endmodule
```

P.6.3 Pachetul de date pe 32 de biți de la ***P.6.2*** este furnizat de un multiplexor dedicat ***pkt_mux***, așa cum este ilustrat mai jos. Intrările sale de date sunt ***pkt*** (pentru pachetele mesajului) și ***msg_len*** (pentru lungimea mesajului inițial).

Ieşirea ***o*** este controlată de următoarele linii de selecţie:

- ***pad_pkt*** – furnizează un bit de 1 urmat de 31 de biți de 0.
- ***zero_pkt*** – furnizează un pachet de zero.
- ***hi_msgln*** – furnizează jumătatea mai semnificativă a lungimii mesajului (primită la intrarea ***msg_len***)
- ***lo_mgln*** – furnizează jumătatea mai puţin semnificativă a lungimii mesajului



Nu pot fi simultan active două sau mai multe intrări de selecţie. Dacă niciuna din cele 4 intrări de selecţie nu este activă, la ieşire se furnizează pachetul de mesaj curent, primit la intrarea ***pkt***.

```
// Varianta 1
module pkt_mux(
        input [63:0]msg_len,
        input [31:0]pkt,
        input pad_pkt, zero_pkt, hi_mgln, lo_mgln,
        output [31:0]o
);

assign o = pad_pkt? {1'b1,31'b0} : ( zero_pkt? {32'b0} : (hi_mgln ? msg_len[63:32] :
(lo_mgln?msg_len[31:0] : pkt )));
endmodule

// Varianta 2
module pkt_mux(
        input [63:0]msg_len,
        input [31:0]pkt,
        input pad_pkt, zero_pkt, hi_mgln, lo_mgln,
        output reg [31:0]o );
always @(*) begin
if (pad_pkt) o = {1'b1,31'b0} ;
else if (zero_pkt) o = 32'b0;
else if (hi_mgln) o = msg_len[63:32];
else if (lo_mgln) o = msg_len[31:0];
else o = pkt;
end
endmodule
```