Laborator AC – Săptămâna 2

P.2.1 Proiectați, utilizând Verilog, un multiplexor 2-la-1 pe 2 biți, având 3 intrări: ***s***, ***d0*** și ***d1*** și având o ieșire, notată cu ***o***.

Soluție:

```
module mux_2_to_1  (
       input        s,
       input [1:0] d0,
       input [1:0] d1,
       output [1:0] o
           );
    assign o = s ? d1:d0;
endmodule
```

P.2.2 Proiectați, utilizând Verilog, un multiplexor 2-la-1 pe 4 biți cu 2 intrări: **s** și ***d*** și o ieșire ***o***. Multiplexorul selectează fie cea mai semnificativă jumătate a intrării ***d***, fie cea mai puțin semnificativă jumătate ai aceleași intrări, dacă ***s*** are valoarea 1 sau 0, respectiv.

Soluție:

```
module mux_2_to_1_sel  (
       input        s,
       input [3:0] d,
       output [1:0] o
           );
    assign o = s ? d[3:2]:d[1:0];
endmodule
```

P.2.3 Proiectați, utilizând Verilog, un modul care selectează, în funcție de o intrare ***swch***, cei mai semnificativi 16 biți ai intrării ***i*** sau cei mai puțin semnificativi 16 biți ai același intrări. Intrarea ***i*** este pe 24 de biți, iar ieșirea modulului se numește ***o***.

Soluție:

```
module switch  (
       input  [23:0] i,
       input swch,
       output [15:0] o
           ) ;
assign o = swch? i[23:8] : i[15:0];
endmodule
```

P.2.4 Construiți un modul pentru inversarea ordinii biților a unei valori pe 6 biți primită la intrare.

Soluție:

```
module reverse_6b  (
       input  [5:0] i,
       output [5:0] o
           );
assign o = {i[0],i[1],i[2],i[3],i[4],i[5]};
endmodule
```

P.2.5 Proiectați, utilizând Verilog, un modul care calculează rezultatul înmulțirii numărului de pe 8 biți la intrare cu 16, fără a utiliza operatorul Verilog de deplasare la stânga **<<**.

Soluție:

```
module multiplier  (
       input  [7:0] in,
       output [11:0] out
           );
assign out = {in,4'd0};
endmodule
```

P.2.6 Implementați, utilizând Verilog, un modul pentru testarea valorii 0 a unui număr pe 8 biți reprezentat în Semn-Mărime fără a utiliza operatorul relațional **==**.

Soluție:

```
module zero_tester  (
       input  [7:0] in,
       output zero
           );
assign zero = | in;
endmodule
```

P.2.7 Implementați, utilizând Verilog, un modul care atașează bitul de paritate la o intrare de 7 biți. Ieșirea pe 8 biți va avea bitul de paritate plasat în poziția cea mai puțin semnificativă.

Soluție:

```
module parity_bit  (
       input  [6:0] in,
       output [7:0] out
           );
assign out = {in, ^in};
endmodule
```

P.2.8 Proiectați, utilizând Verilog, un modul pentru a verifica paritatea generată de modulul anterior. Modulul va avea intrarea ***i*** pe 8 biți, o ieșire de 7 biți, reprezentând cei mai semnificativi 7 biți ai intrării și o ieșire ***err*** activă de dacă bitul de paritate a fost greșit calculat.

Soluție:

```
module parity_bit_checker  (
        input  [7:0] i,
        output [6:0] o,
        output err
            );
assign o = i[7:1];
assign err = ^i;
endmodule
```

P.2.9 Implementați, utilizând Verilog, un modul pentru conversia numerelor pe 8 biți din Semn-mărime în Complement de 1.

Soluție:

```
module sign_magnitude_converter (
       input  [7:0] i,
       output [7:0] o
          );
assign o = i[7] ? {i[7],~i[6:0]} : i;
endmodule
```

P.2.10 Proiectați un modul pentru compararea a 2 numere pe 1 bit, ***x*** și ***y***. Modulul are ieșirile ***eq*** activă dacă numerele x și y sunt egale, ieșirea ***le*** activă dacă ***x*** este mai mic decât ***y*** , iar ieșirea ***gt*** este activă în celelalte cazuri.

Soluție:

```
module compare_1bit_numbers (
        input x,y,
        output eq, le, gt
            );
assign eq = x ~^ y;
assign le = x & (~y);
assign gt = (~x) & y;
endmodule
```