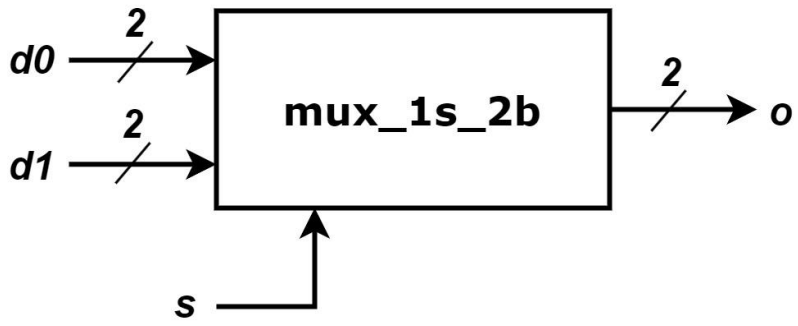
Week 2 - Introduction – Proposed Problems

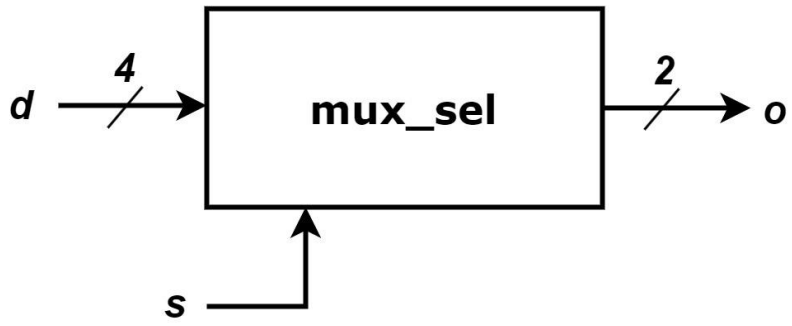
P.2.1 Design, using Verilog, a 2-to-1 multiplexer on 2 bits, having 3 inputs: ***s***, ***d0***, and ***d1***, and having one output, denoted by ***o***.



Solution:

```
module mux_1s_2b (
    input s,
    input [1:0] d0,
    input [1:0] d1,
    output [1:0] o
);
assign o = s ? d1:d0;
endmodule
```

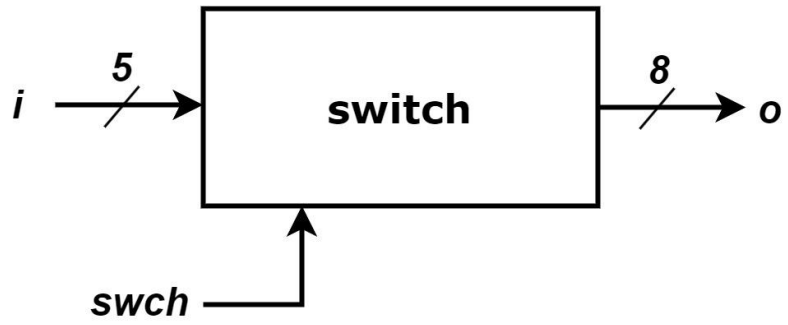
P.2.2 Design, using Verilog, a multiplexer on 4 bits, with 2 inputs: ***s*** and ***d***, and one output, ***o***. The multiplexer selects either the most significant half of the input ***d*** or the least significant half of the same input, if ***s*** has the value 1 or 0, respectively.



Solution:

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

P.2.3 Design, using Verilog, a module that attaches, depending on an input ***swch***, 3 bits of 1 on the most significant positions of the input ***i*** if ***swch*** is 1, or 3 bits of 0 on the least significant positions if ***swch*** is 0. The input ***i*** is declared as 5 bits, and the module's output is ***o***.



Solution:

```
module switch (
    input swch,
    input [4:0] i,
    output [7:0] o
);
assign o = swch ? {3'd7,i}:{i,3'd0};
endmodule
```

P.2.4 Build a module for reversing the order of the bits of a 5-bit value received at the input.

Solution:

```
module reverse_5b (
    input [4:0] i,
    output[4:0] o
);
assign o = {i[0],i[1],i[2],i[3],i[4]};
endmodule
```

P.2.5 Design, using Verilog, a module that separates the integer part from the fractional part of a number connected to the module's input. A 5-bit number is allocated for the ***integer part*** of the input, and a 3-bit number is reserved for the ***fractional part***.

Solution:

```
module separator_8b (
    input [7:0] i,
    output [4:0] in, // integer part
    output [2:0] fr // fractional part
);
assign in = i[7:3];
assign fr = i[2:0];
endmodule
```

P.2.6 Implement, using Verilog language, a module for testing the zero value of an 8-bit number represented in Sign-Magnitude without using the relational operator ***==***.

Solution:

```
module zero_tester (
    input [7:0] in,
    output zero
);
assign zero = | in[6:0];
endmodule
```

P.2.7 Implement, using Verilog language, a module that attaches a parity bit to a 7-bit input. The 8-bit output will have the parity bit placed in the least significant position.

Solution:

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

P.2.8 Design, using Verilog, a module to check the parity generated by the previous module. The module will have an 8-bit input ***i***, a 7-bit output ***o*** representing the most significant 7 bits of the input, and an active output ***err*** indicating if the parity bit was calculated incorrectly.

Solution:

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

P.2.9 Implement, using Verilog, a module for converting 8-bit numbers from Sign-Magnitude to One's Complement.

Solution:

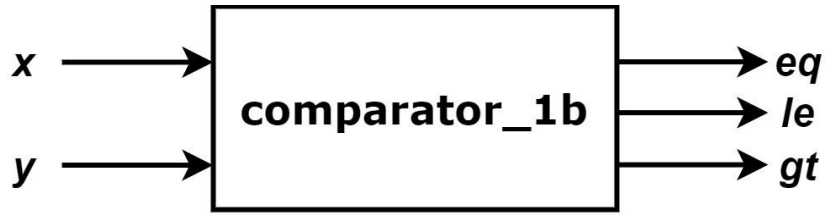
```
module converter_8b (
    input [7:0] i,
    output[7:0] o
);
assign o = i[7] ? {i[7],~i[6:0]}:i;
endmodule
```

P.2.10 Build, using Verilog, a module for detecting multiples of 3 connected to its input. The input ***i*** is declared on 4 bits, and the module's name is ***mul_3***.

Solution:

```
module mul_3 (
    input [3:0] i,
    output o
);
assign o = (i%3==0) ? 1 : 0;
endmodule
```

P.2.11 Design a module for comparing 2 numbers, ***x*** and ***y***, each on 1 bit. The module has the output ***eq*** active if the numbers ***x*** and ***y*** are equal, the output ***le*** active if ***x*** is less than ***y***, and the output ***gt*** active if ***x*** is greater than ***y***.



Solution:

```
module comparator_1b (
    input x,y,
    output eq,le,gt
);
assign eq = x ~^ y;
assign le = x & (~y);
assign gt = (~x) & y;
endmodule
```

P.2.12 Design, using Verilog, a module that displays the most significant decimal digit of an unsigned 4-bit input.

Solution:

```
module msd (
    input [3:0] i,
    output [3:0] o
);
assign o = i > 4'd9 ? 4'd1 : i;
endmodule
```