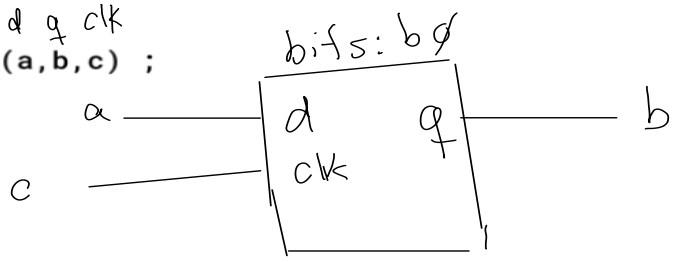
Hierarchical Design

Exercise 1:

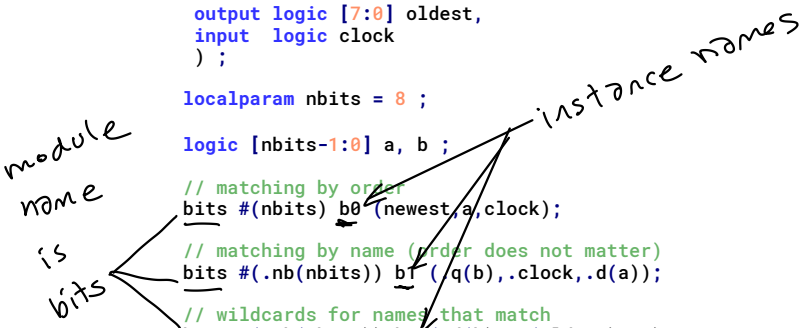
Draw a diagram for this instantiation of the **`bits`** module. Label the module, instance, signal and port names as in the diagram above.

```
bits #(4) b0 (a,b,c) ;
```

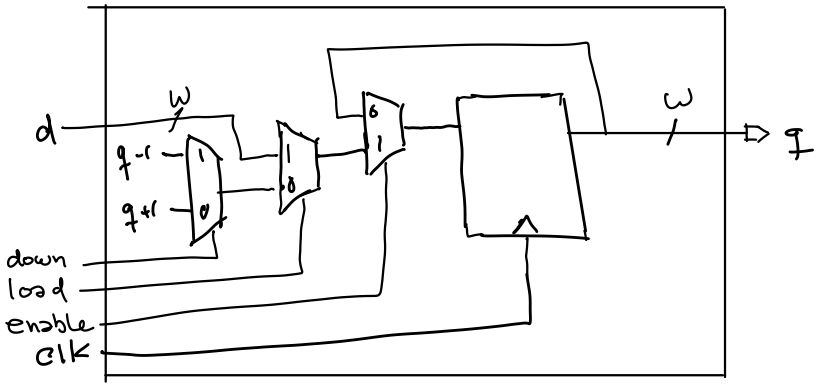


Exercise 2:

```
module sr3bytes
   (
    input  logic [7:0] newest,
    output logic [7:0] oldest,
    input  logic clock
    ) ;

   localparam nbits = 8 ;

   logic [nbits-1:0] a, b ;

   // matching by order
   bits #(nbits) b0 (newest,a,clock);

   // matching by name (order does not matter)
   bits #(.nb(nbits)) b1 (.q(b),.clock,.d(a));

   // wildcards for names that match
   bits #(.nb(nbits)) b2 (.d(b),.q(oldest),.*);

endmodule
```



Identify the module instantiation statements in the code above. For each one, what is the instantiated module's name? The instance name?



Exercise 3:

```
// traffic light controller

module ex70
  ( output logic [5:0] lights,
    input logic reset, clk ) ;

   logic [1:0] state ; // state register
   logic [4:0] count ; // delay counter
   logic [5:0] lut [4] = '{ 6'b100_001, 6'b100_010,
                            6'b001_100, 6'b010_100 } ;

   // lights state (counter)
   always @(posedge clk) state
      <= reset ? 2'b00 :
         state == 2'b11 && count == 0 ? 2'b00 :
         count == 0 ? state + 1'b1 :
         state ;

   // set output based on state
   assign lights = lut[state] ;

   // timer
   always @(posedge clk) count
      <= reset ? 29 :
         count != 0 ? count-1 :
         state == 2'b00 || state == 2'b10 ? 4 : 29 ;

endmodule
```

Write a **`counter`** module with a width parameter **`w`**, *w*-bit inputs and outputs **`d`** and **`q`**, **`enable`**, **`load`** and **`down`** control inputs, and a clock **`clk`**. The default counter width should be 4 bits, the default value of **`enable`** should be 1, and the default values of **`load`**, **`down`** and **`d`** should be **`0`**. Rewrite the traffic light controller module by instantiating two instances of this module.

```
// generic counter module

module counter #( parameter w=4)
   ( output logic [w-1:0] q,
     input logic [w-1:0] d,
     input logic enable, load, down, clk );

   always_ff @(posedge clk)
     q <= ! enable ? q :
          load ? d :
          down ? q-1 : q+1 ;

endmodule
```

```
// traffic light controller

module tlc
  ( output logic [5:0] lights,
    input logic reset, clk ) ;

   logic [1:0] state ; // state register
   logic [4:0] count ; // delay counter
   logic [5:0] lut [4] = '{ 6'b100_001, 6'b100_010,
                            6'b001_100, 6'b010_100 } ;

   // lights state (counter)
   // always @(posedge clk) state
   //    <= reset ? 2'b00 :
   //       state == 2'b11 && count == 0 ? 2'b00 :
   //       count == 0 ? state + 1'b1 :
   //       state ;

   counter #(.w(2)) c_state
     ( .q(state), .d(0),
       .enable(reset || count == 0), .load( reset || state == 2'b11 && count == 0 ),
       .down(0),
       .clk );

   // set output based on state
   assign lights = lut[state] ;

   // timer
   // always @(posedge clk) count
   //    <= reset ? 29 :
   //       count != 0 ? count-1 :
   //       state == 2'b00 || state == 2'b10 ? 4 : 29 ;

   counter #(.w(5)) c_timer
     ( .q(count), .d( reset ? 29 : ( state == 2'b00 || state == 2'b10 ) ? 4 : 29 ),
       .enable(1), .load(reset || count == 0), .down(1),
       .clk );


endmodule
```