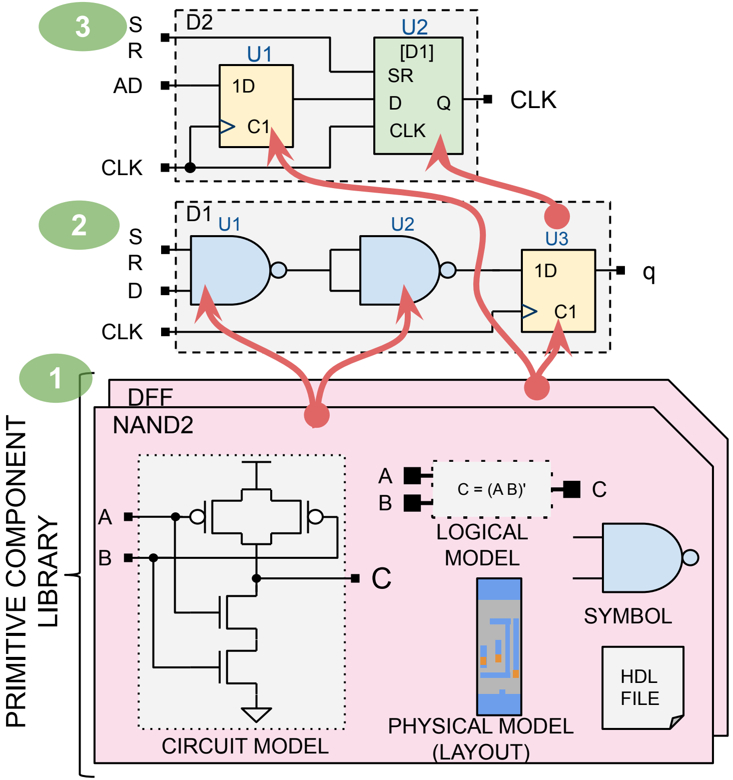
Digital Techniques 2

L2: Structural Modeling with SystemVerilog

UNIVERSITY OF OULU

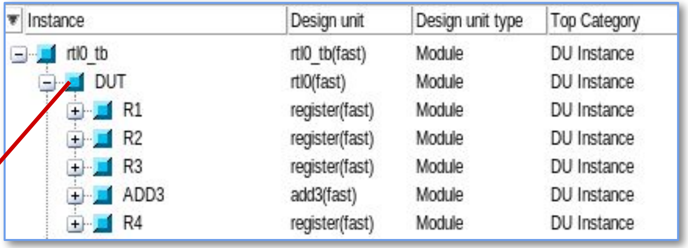
CIRCUITS and SYSTEMS

Hierarchical Design Concepts and Terminology



1. Designs are created using **primitive components** (NAND2,DFF) from a component **library** that contains **models** of the components for different EDA tools
2. To create a design (D1), primitive components are **instantiated** (U1,U2,U3) and connected with **nets** from their **ports** to ports of the design and other instances. Instantiated primitives are called **references**.
3. Designs can also be instantiated (D2/U2) to create **hierarchical designs** (D2).

Representation of a Hierarchical Design in QuestaSim GUI

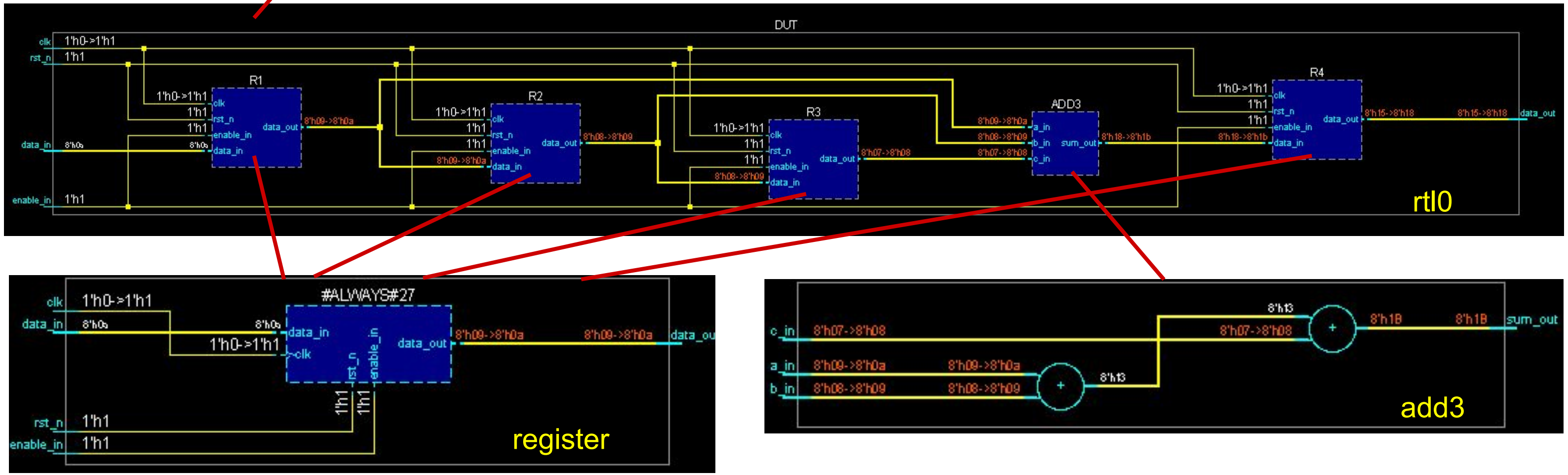


Instance	Design unit	Design unit type	Top Category
rtl0_tb	rtl0_tb(fast)	Module	DU Instance
DUT	rtl0(fast)	Module	DU Instance
R1	register(fast)	Module	DU Instance
R2	register(fast)	Module	DU Instance
R3	register(fast)	Module	DU Instance
ADD3	add3(fast)	Module	DU Instance
R4	register(fast)	Module	DU Instance

← Testbench module instance

← Design module instance (hierarchical)

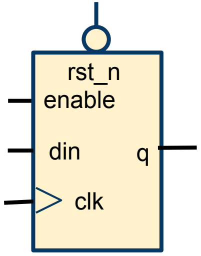
Module instances (non-hierarchical)



SystemVerilog **module**

- **module** is the basic design building block in SystemVerilog
- module defines a **design's name**, its **input and output port names and types**, its **parameters**, and its function or hierarchical structure
- You create a design by creating its **module hierarchy** from the top-level module down to lowest level modules
- Typically, the lowest level modules that do not instantiate other modules contain the functional RTL code
- Statements inside a module are **concurrent** (executed all at the same time)

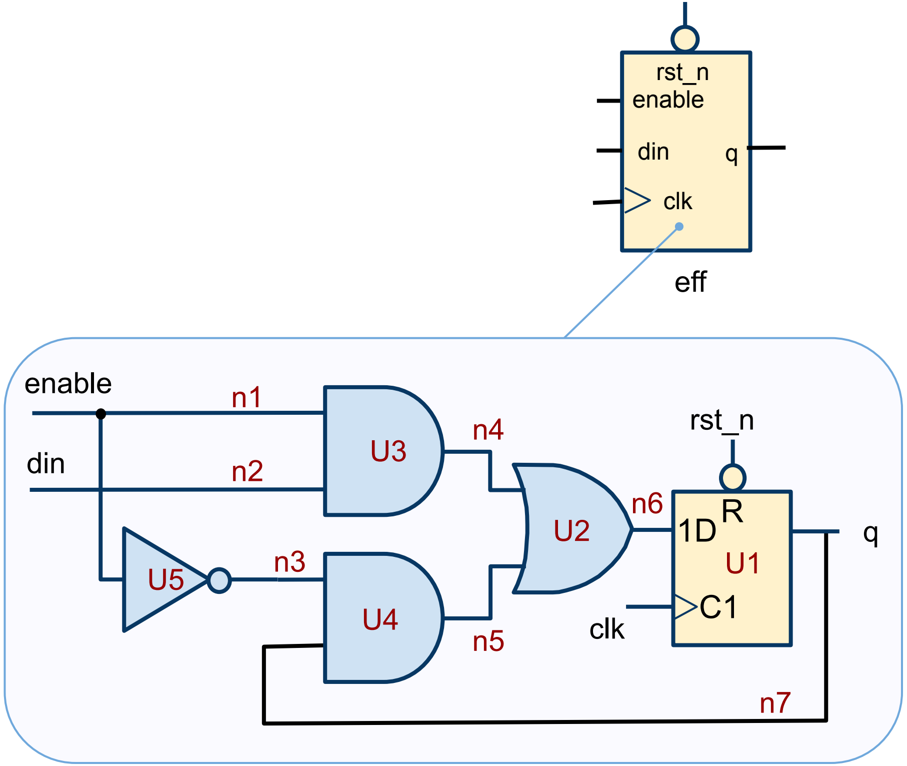
```
module eff
     (input logic clk,
      input logic rst_n,
      input logic din,
      input logic enable,
      output logic q);

     // Concurrent statements (not shown)

endmodule
```



eff

SystemVerilog Model of a Simple Hierarchical Design

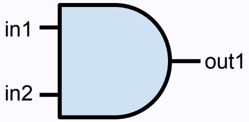
- We develop a hierarchical model for an enabled flip-flop **eff** using primitive components defined as modules that contain functional code
- We learn how to:
  - Create and name instances of other modules inside a module
  - Create variables that represent nets
  - Connect the nets to the ports of the module instances
  - Connect nets to the ports of the module
- We need models of AND, OR and NOT gates and a D-flip-flop



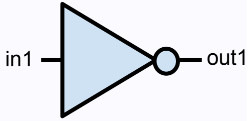
UNIVERSITY OF OULU
CIRCUITS and SYSTEMS

RTL Models of the Primitive Components

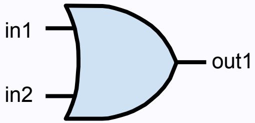
```
module and2 (input logic in1, in2,
            output logic out1);

   assign out1 = in1 & in2;

endmodule
```



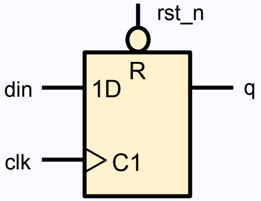
```
module inv (input logic in1,
           output logic out1);

   assign out1 = ~in1;

endmodule
```



```
module or2 (input logic in1, in2,
           output logic out1);

   assign out1 = in1 | in2;

endmodule
```



```
module dff (input logic clk, rst_n, din,
           output logic q);

   always_ff @ (posedge clk or negedge rst_n)
     if (!rst_n)
       q <= 1'b0;
     else
       q <= din;
endmodule
```



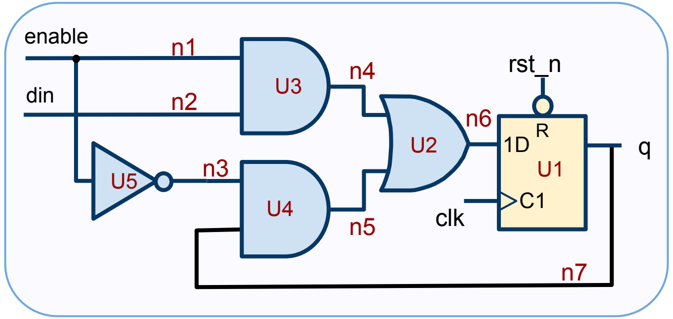
Notice! Only look at the **module** declarations at this stage. Internal functional code will be covered later.

Hierarchical Model of **eff** Module

```
module eff (input logic clk, rst_n, din, enable,
                 output logic q);
  logic n1, n2, n3, n4, n5, n6, n7;

  assign n1 = enable;
* assign n2 = din;
  assign q  = n7;

  dff  U1 (.clk(clk), .rst_n(rst_n), .din(n6),  .q(n7));
  or2  U2 (.in1(n4),  .in2(n5),      .out1(n6)       );
  and2 U3 (.in1(n1),  .in2(n2),      .out1(n4)       );
  and2 U4 (.in1(n3),  .in2(n7),      .out1(n5)       );
  inv  U5 (.in1(n1),  .out1(n3)                      );

endmodule
```

.port_of_instantiated_module **(**variable_or_port_in _this_module**)**

1. Variables represent nets
2. Connections between ports and variables can be defined with a "continuous" **assign** statement
3. Module instantions and port connections ("port mappings")



[*] Notice: You could also have connected **enable**, **din** and **q** to the ports of the components directly without creating the nets **n1**, **n2** and **n7**. This example just shows how you can use the **assign** statement to connect nets,

UNIVERSITY OF OULU
CIRCUITS and SYSTEMS

Module Instantiation Syntax Summary

- Ports in the module to be instantiated must be connected to variables or ports of the instantiating module
- Parts of instantiation
  - module name
  - module parameters (see later slides)
  - unique instance name
  - connection list

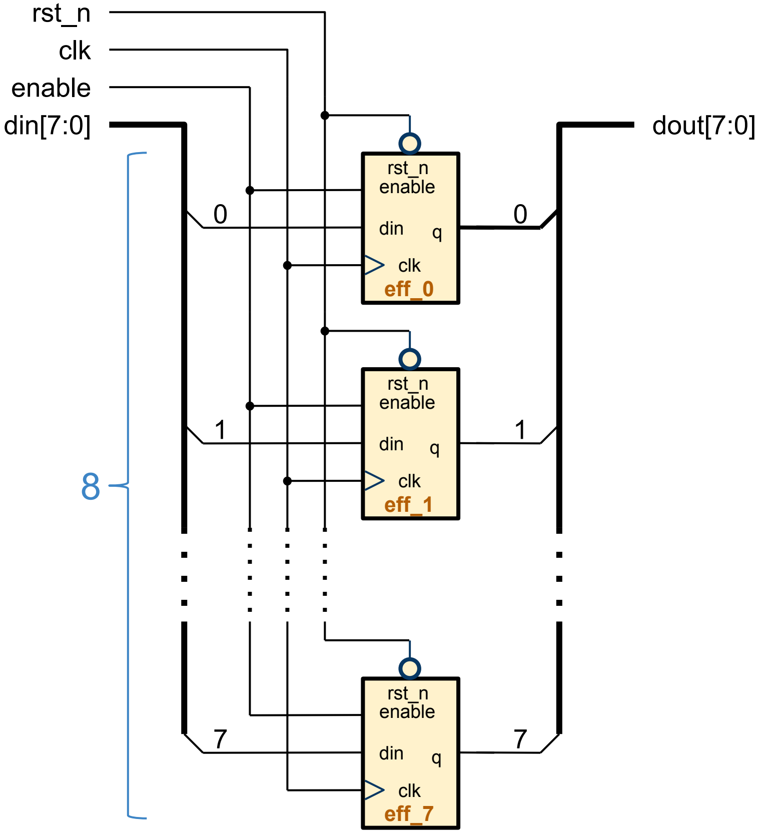
    ```
    inv U5 (.in1(n1), .out1(n3) );
    ```

- Connection list formats
  - Order-based (don't use!!!!)
  - Name-based
  - Implicit: connect ports with wildcard `.*` to variables/ports with a matching name (sometimes useful)

Implicit Port Connection Example:

```
module inv (
   input logic data_in,
   output logic data_out);
   assign data_out = ~data_in;
endmodule

module buf (
   input logic buf_in,
   output logic buf_out);
   assign buf_out = buf_in;
endmodule

module tb;
   logic data_in;
   logic data_out;
   logic buf_out;

   inv U1 (.*);
   buf U2 (.buf_in(data_in), .*); // Port buf_in must be
                                   // connected by name
endmodule
```

eff Can Now in Turn Be Used to Create a Register

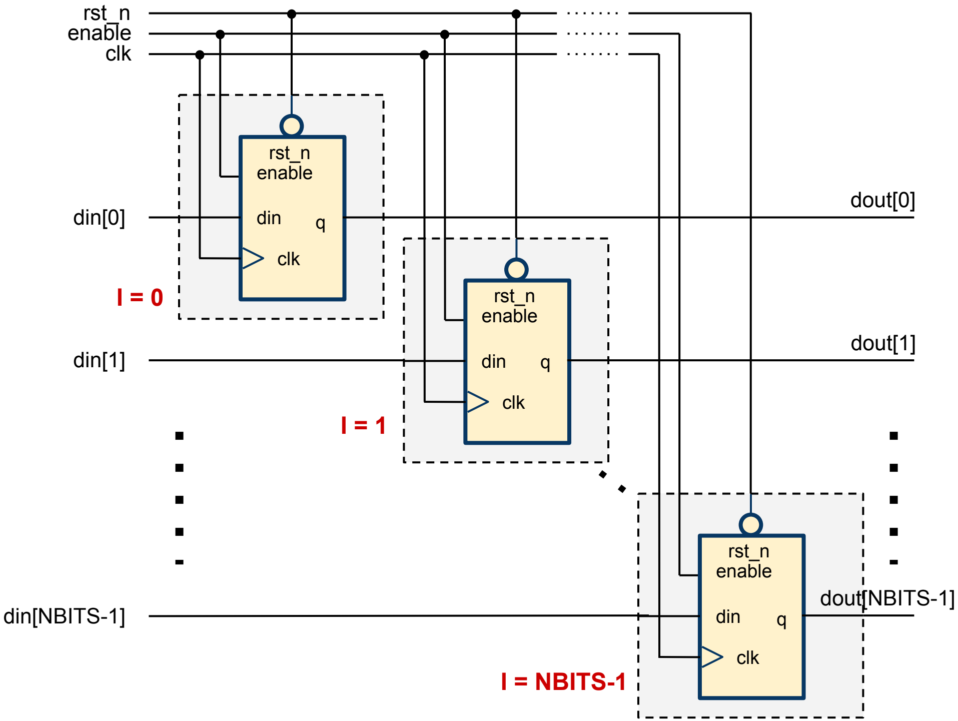


```
module effreg8
  (input logic          clk,
   input logic          rst_n,
   input logic          enable,
   input logic [7:0]    din,
   output logic [7:0]   dout);

  eff eff_0 (.clk(clk), .rst_n(rst_n), .din(din[0]),  .enable(enable), .q(dout[0]));
  eff eff_1 (.clk(clk), .rst_n(rst_n), .din(din[1]),  .enable(enable), .q(dout[1]));
  eff eff_2 (.clk(clk), .rst_n(rst_n), .din(din[2]),  .enable(enable), .q(dout[2]));
  eff eff_3 (.clk(clk), .rst_n(rst_n), .din(din[3]),  .enable(enable), .q(dout[3]));
  eff eff_4 (.clk(clk), .rst_n(rst_n), .din(din[4]),  .enable(enable), .q(dout[4]));
  eff eff_5 (.clk(clk), .rst_n(rst_n), .din(din[5]),  .enable(enable), .q(dout[5]));
  eff eff_6 (.clk(clk), .rst_n(rst_n), .din(din[6]),   .enable(enable), .q(dout[6]));
  eff eff_7 (.clk(clk), .rst_n(rst_n), .din(din[7]),   .enable(enable), .q(dout[7]));

endmodule
```

Common Errors in Structural Models

- If you forget to declare an internal variable, or type a variable's name incorrectly, SystemVerilog assumes this referenced name to be an **implicitly declared 1-bit variable**, which leads to hard-to-detect bugs (e.g. `.reset(rest)` → `rest is` assumed to exist)
- If you forget to connect a port, you will get a *warning* of a **missing connection**, but no *error*
- If you connect to a port a variable that has a wrong number of bits, you will get a *warning* of **wrong connection size**, but no *error*
- Even if the model compiles, check out these potential errors from simulator log!!! (Log = "Transcript" in QuestaSim)

Module Parameterization: Making Code Reusable



```
module effreg  #(parameter NBITS = 8)
  (input logic clk,
   input logic rst_n,
   input logic enable,
   input logic [NBITS-1:0] din,
   output logic [NBITS-1:0] dout
   );

  generate
    for (genvar I = 0; I < NBITS; ++I)  begin
            eff eff_inst (.clk(clk),  .rst_n(rst_n),
                          .din(din[I]),  .enable(enable),
                          .q(dout[I]));
      end
  endgenerate
endmodule
```

generate statement creates many instances of a component using a loop

We Can Instantiate effreg in a Testbench Module

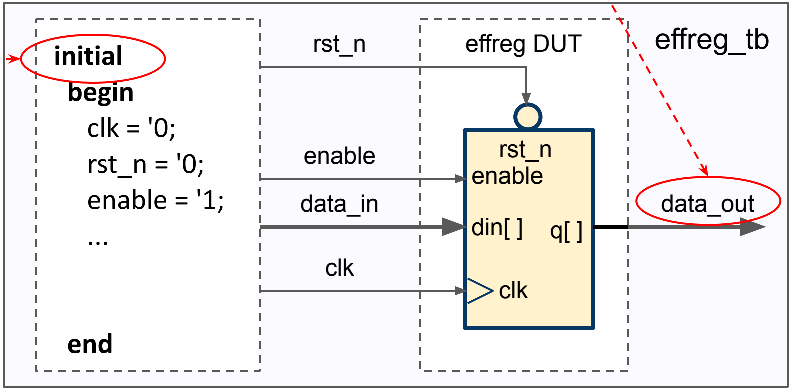
```
module effreg_tb;
  logic clk, rst_n, enable;
  logic [3:0] data_in, data_out;

  initial
     begin
         clk = '0;
         rst_n = '0;
         enable = '1;
         data_in = 4'b1010;
         #50ns;  // wait for 50 ns
         clk = '1;
         #50ns;
         clk = '0;
         rst_n = '1;
         #50ns;
         clk = '1;
         #50ns;
     end
```

#(.parameter_name (parameter value))

```
 effreg #(.NBITS(4)) DUT
    (.clk(clk),
     .rst_n(rst_n),
     .din(data_in),
     .enable(enable),
     .dout(data_out));
endmodule
```

Testbench models the design environment. You can view testbench signals in the simulators GUI.

initial procedure is used to execute a test program that generates test data



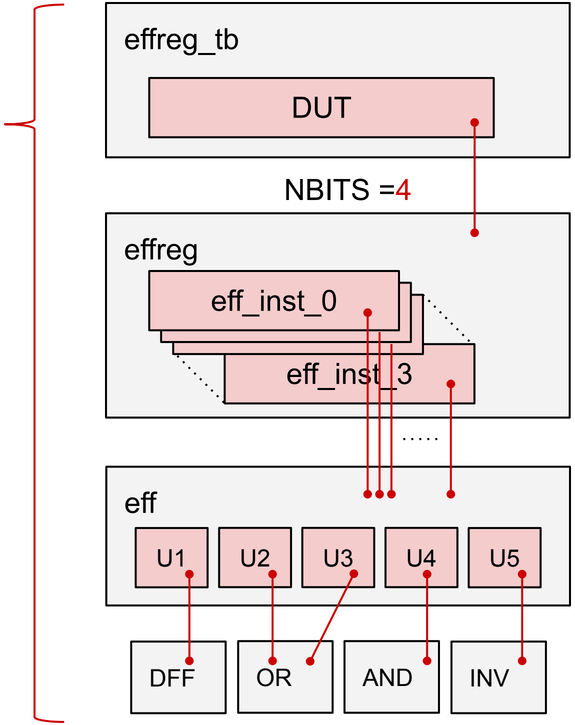
Summary of Module Hierarchy Creation

1. Write module "headers" for all modules
    a. Module name
    b. Ports
    c. Parameters
2. In hierarchical modules that instantiate other modules, write
    a. Variable declarations to represent module-to-module connections
    b. Module instantiation statements that define
        i. Name of module to be instantiated
        ii. Optionally: Parameter value assignments
        iii. Unique name for the module instance
        iv. Port mappings: Connections between instance ports and parent module's ports or variables
3. Instantiate top module in testbench

Simulation

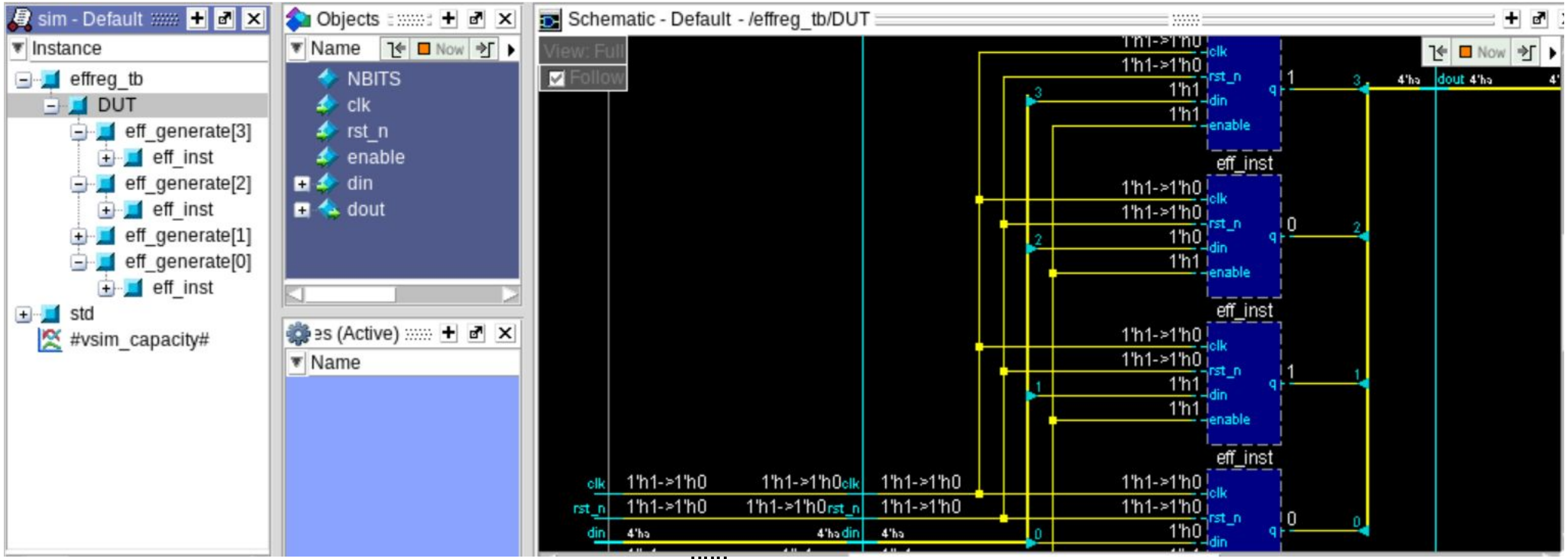
1. Source code is **analyzed** into work library:
   `vlog -sv effreg.sv effreg_tb.sv`
2. Simulation model is **elaborated** starting from top module in the library
   `vsim work.effreg_tb`
   Elaboration tasks include:
   a. Binding of module instances
   b. Computation of parameter values
   c. Creation of port connections
3. Elaborated simulation model is executed
   `run -all`

Tip: Before examining simulation results, check that elaboration went well! (Read the *Transcript* window in QuestaSim from the beginning)



Lab: Simulation and Analysis of a Hierarchical Model

- The aim of this lab is to learn to understand how a hierarchical design is analyzed (compiled), elaborated and simulated, and how it is presented in the simulator's graphical user interface (GUI).



References

1. IEEE Std 1800-2012 IEEE STANDARD FOR SYSTEMVERILOG, Ch 23 Modules and Hierarchy