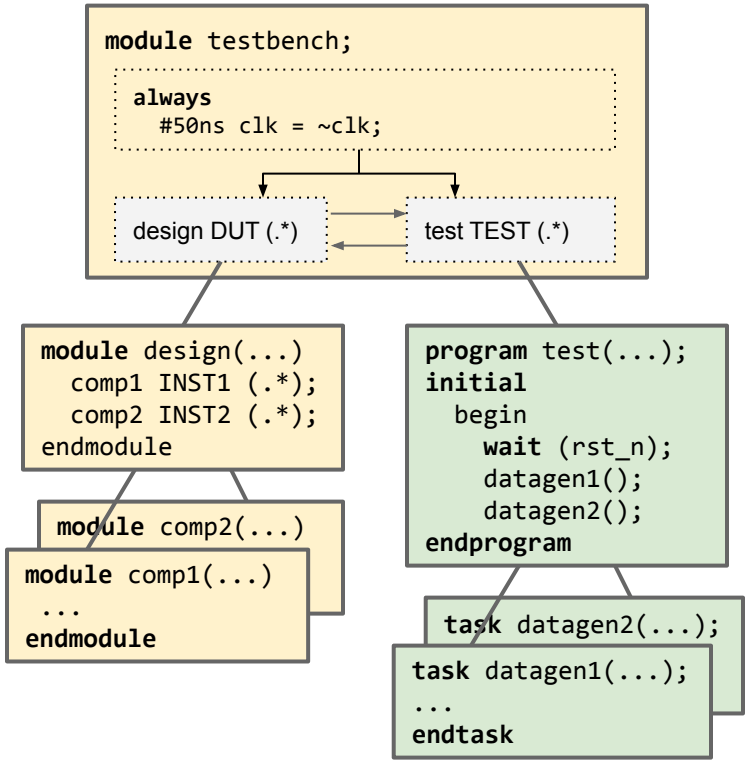
# Digital Techniques 2

## L9: Testbench Design

UNIVERSITY OF OULU
CIRCUITS and SYSTEMS

# SystemVerilog Simulation Testbench Structure

- Basic testbench **module**
  - Generates clock and reset signals in *always* and/or *initial* procedures
  - Instantiates the design's top module and a test program
  - Connects design's top module's ports to test programs ports
  - Connects clock and reset signals to the design and test program
- Design's top **module** instantiates the design's module hierarchy
- Test **program**
  - Contains an initial procedure that writes data to the design's inputs and reads data from its outputs
  - initial procedure can launch concurrent processes using the **fork** statement, and it can call **task** subroutines to generate test data, check results etc



# SystemVerilog **program** Block

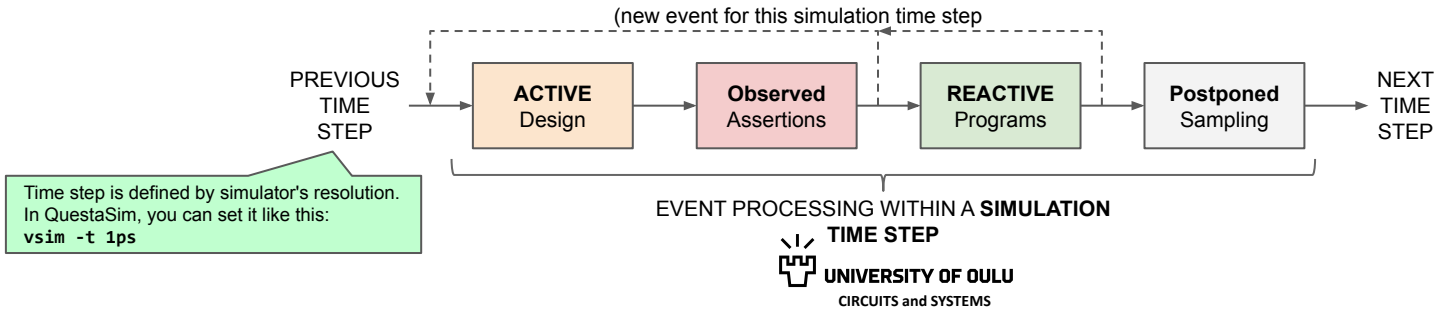
Race condition example: Any one of **blocking** assignments can be executed first!

```
always @(posedge clk)
  y = !y;

always @(posedge clk)
  x = y;
```

- **program** is much like a **module**, but it cannot instantiate modules or contain **always** procedures
- Usually has an **initial** procedure that calls **task** subroutines
- Statements inside modules and programs are executed in a different *regions* inside a simulation time step to prevent *race conditions* between testbench and DUT

- Modules execute in ACTIVE region while programs execute in REACTIVE region
  - *Modules* (DUT) first process all events that have been scheduled for the current time step by **=** or **<=** assignment statements
  - When the DUT has finished, control is passed to the *program* that reads data generated by the DUT and generates new data that is fed to the DUT



# SystemVerilog **initial** Procedure in Test Programs

- **initial**-procedure is started in the beginning of simulation, and it runs to its end and stops
- Timing and event controls can be used to control the execution, e.g.

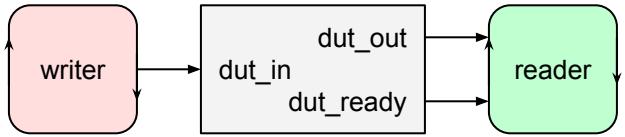
  `#10ns;` Wait for 10 ns

  `@(posedge clk);` Stop until *clk* edge

  `wait (rst_n);` Wait for *rst_n* to to be true
- **fork - join*** statements can be used to spawn concurrent processes e.g. to handle different, concurrently running DUT interfaces

  * = **join**, **join_any**, **join_none**, also see **disable**

```
initial
  begin
    wait (rst_n);
    fork
      begin : writer
        repeat(10)
          @(negedge clk) dut_in = $urandom;
      end : writer
      begin : reader
        forever
          @(posedge clk)
            if (dut_ready)
              $display("%b", dut_out);
      end : reader
    join_any
    $finish;
  end
```

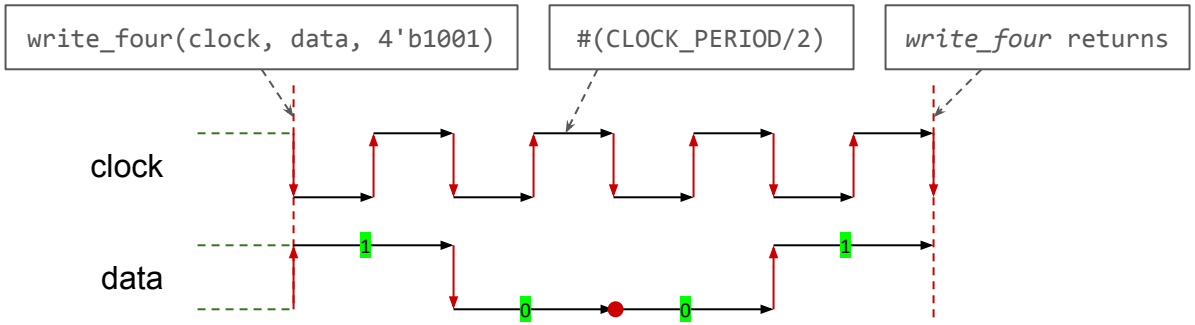


# SystemVerilog Subroutines

- **Functions** are used to compute values and they are executed in zero time
- **Tasks** can consume time → can be used to generate test data and to read responses

```
task automatic write_four
    ( ref logic clock,
      ref logic data,
      input logic [3:0] value
      );
     for (int i=3; i >= 0; --i)
     begin
        clock = '0;
        if (value[i] == '1)
            data = '1;
        else
            data = '0;
        #(CLOCK_PERIOD/2);
        clock = '1;
        #(CLOCK_PERIOD/2);
     end
  endtask;
```

A call to task **write_four** generates the waveforms of **clock** and **data** shown below. Arguments **clock** and **data** are **references** to variables in the calling program (updated immediately). Tasks can contain timing controls, such as **#**.




automatic: internal variables are automatic (default is static for tasks)



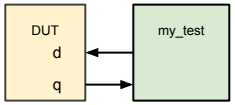
See: pass by reference type arguments

# SystemVerilog System Tasks

- Simulation control tasks
  `$finish $stop $exit`
- Simulation time functions
  `$realtime $stime $time`
- Severity tasks for printing messages
  `$fatal $error $warning $info`
- Probabilistic distribution functions
  `$urandom $urandom_range $random`
- Data display tasks
  `$display $write $monitor`
- File I/O tasks and functions $fclose $fopen
  `$fdisplay $fwrite`
- Real math functions
  `$ln(x), log(x), $sin(x) etc.`

See IEEE Std 1800-2012, chapter 20 for more.

# SystemVerilog **clocking** Block

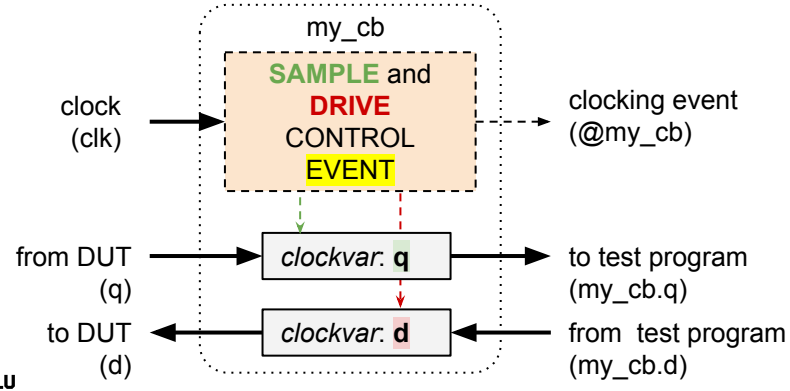



- A **clocking** block is used to synchronize *sampling* of DUT output values and *driving* DUT input values to a clocking event
- Sampled values are stored in "**clockvars**" that can be read and written in the test program: these values are therefore all synchronous to the clock
- Clockvars are assigned using **<=**
- By defining a clocking block as **default**, **cycle-based** delays can be used. e.g.:
  `##10;` // wait for 10 clock cycles
- **Skews** (#) can be defined inside a clocking block to delay driving of outputs, or to sample inputs earlier

```
program my_test
 (input logic clk, q,
  output logic d);

 default clocking my_cb @(posedge clk);
   input        q;
   output  #1ns d; // 1 ns skew (delay)
  endclocking

 initial begin
   my_cb.d <= '0;
   ##2; // wait for 2 clock cycles
   if (my_cb.q == '1)
```



# SystemVerilog configuration

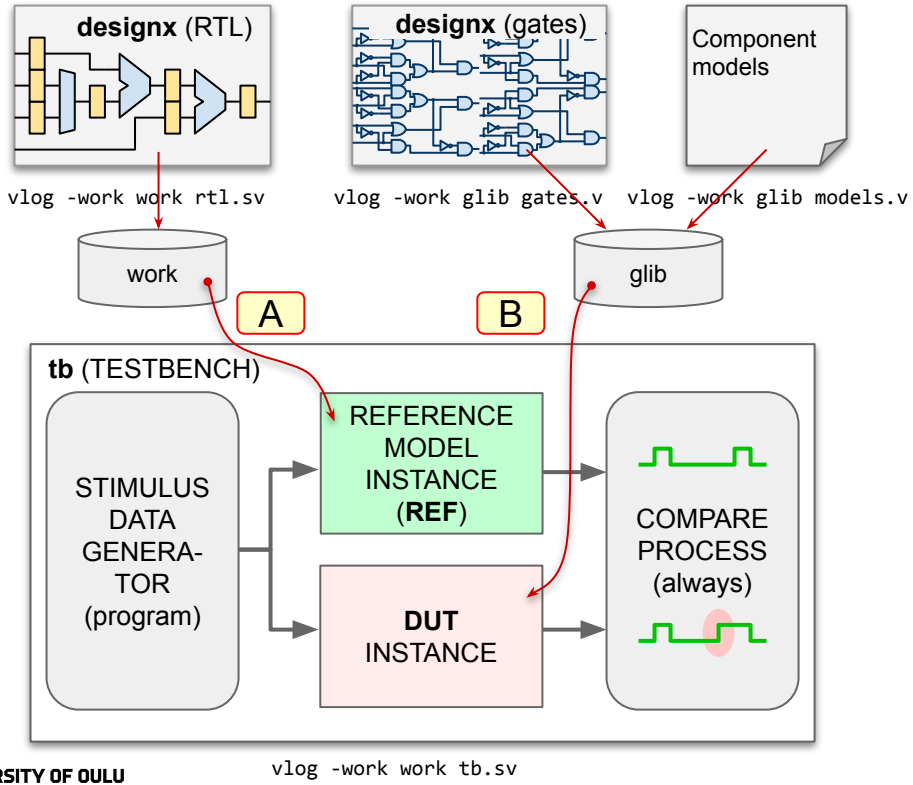
- Configurations can be used to control **module binding** from **design libraries**
- Example: Instantiation of gate-level (DUT) and RTL (reference) models in the same testbench

```
config glhier_cfg;        // Configuration for
   design glib.designx;   //  whole gl-hierarchy
   default liblist glib; //  (designx + components)
endconfig

config tb_cfg;  // Configuration for testbench
   design work.tb;
   default liblist work;
   instance tb.REF use work.designx; // A
   instance tb.DUT use work.glhier_cfg:config; // B
endconfig
```

- By changing the configuration, different simulations can be run in the same testbench (RTL alone, RTL-vs-post-layout etc)

RTL and gate-level modules **have the same name** and must therefore be compiled to separate libraries.



# SystemVerilog Verification Tools Covered in DT3

- Interfaces (typically a functional model of a bus: signals + tasks for reading and writing data)
- Assertions (see *_svamod.sv files in project)
- Functional coverage constructs
- Constrained random stimulus generation
- Classes
- Universal Verification Methodology (UVM)

# References


1. IEEE Standard for SystemVerilog:
   24. Programs
   14. Clocking blocks
   33. Configuring the contents of a design