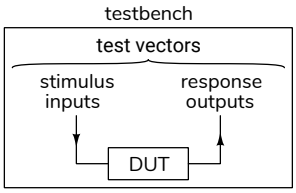
Simulation

This lecture describes how to use Verilog to simulate designs.
After this lecture you should be able to write a testbench that can: set initial values, generate clocks, read test vectors from a file, display values, and terminate on a condition.
Version 2: changed to $stop and printf format in example testbench.

Simulation

Verilog can be used to test HDL designs by simulating their operation. A simulation consists of the module being tested (called the Design Under Test or DUT) that is instantiated in another module called a testbench. The testbench applies inputs to the DUT and checks its output:



Test Vectors

The inputs to the DUT and the corresponding expected outputs are called test vectors. These can be generated by the testbench itself or they can be read from a file. Test vectors should generally include:

1. typical inputs,
2. minimum and maximum valid inputs,
3. invalid inputs, and
4. randomly-chosen values.

Exercise 1: Give examples of appropriate test inputs for each of the above categories if you were testing a circuit that computed the square root of a 16-bit signed number.

Verilog for Verification

The following Verilog features are useful for simulation but are not synthesizable (cannot be implemented in hardware).

initial blocks

Initial blocks execute once at the start of the simulation and are used to initialise signals. **`begin`** and **`end`** are used to group statements (as with `{` and `}` in C).

if/else

If/else statements can be used in **`initial`** or **`always`** blocks.

Delays

Placing *#number* before a statement delays[1] execution by *number* simulation time. The suffixes **`ns`** and **`us`** can be used for nano- and micro-seconds.

The syntax `@(`*event*`)` where *event* can be **`posedge`** or **`negedge`** before a signal name or just a signal name delays execution until that signal edge or a change in that signal value.

wait

The **`wait(`***expression***`)`** statement pauses until the expression is non-zero.

Exercise 2: What's the difference between `wait(x) y='1;` and `@(x) y='1;`?

System Tasks

Functions beginning with **`$`** are called system tasks. Useful ones include:

- **`$display()`** similar to C's **`printf()`**, can be used to print values during a simulation;
- **`$dumpfile`** and **`$dumpvars`** record changes in signals to a `.vcd` file for subsequent viewing with a waveform viewer.

[1] Delays are not synthesizable because delays cannot be easily implemented in hardware.

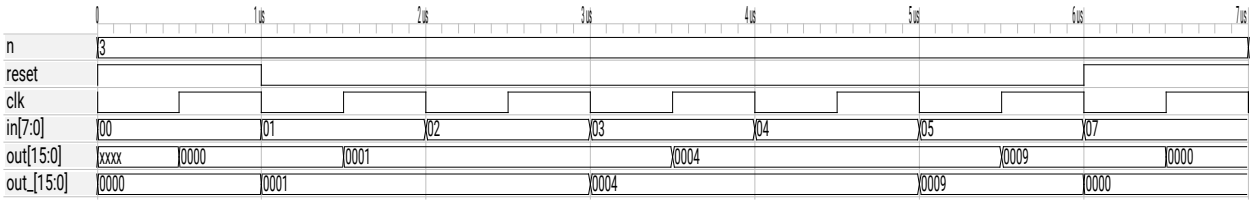


Figure 1: Simulation waveforms (from **ex66.vcd**).

- **$fopen()** and **$fscanf()**, similar to the C library functions **fopen()** and **fscanf()**, can open and read from text files.
- **$finish** and **$stop** terminate or suspend a simulation.

Example

The testbench below demonstrates the language features described above.

The DUT is a module with an 8-bit input that outputs a 16-bit sum of all the odd-valued inputs since a reset input was asserted. On each falling edge of the clock the testbench checks the current output and reads another test vector from a file. Each line of the file contains the (reset and input) input values and the expected output values.

```
// output cumulative sum of odd-valued inputs

module ex66
  ( input logic reset, clk,
    input logic [7:0] in,
    output logic [15:0] out ) ;

   always_ff @(posedge clk)
     out <= reset ? '0 :
            in & 1 ? out+in : out ;

endmodule


// example testbench

module ex66_tb ;

   // DUT inputs and outputs
   logic reset, clk ;
   logic [7:0] in ;
   logic [15:0] out, out_ ;

   // instantiate DUT
   ex66 ex66_0 (.*) ;

   // file descriptor and number of values read
   integer fd, n=0 ;

   initial begin

      // record all signals in a .vcd file
      $dumpfile("ex66.vcd") ;
      $dumpvars ;

      // initialize signals
      {reset, clk} = '0 ;

      // open the test vector file
      fd = $fopen("ex66data.csv","r") ;

      // wait for error or end of file
      wait (n < 0) $stop() ;
   end

   // 1 MHz clock (2x500ns period)
   always #0.5us clk = ~clk ;

   // check outputs and change inputs on
   // each falling clock edge
   always @(negedge clk) begin
      if ( out != out_ )
        $display("Error: %d %d %d", in, out, out_) ;
      n = $fscanf(fd,"%d,%d,%d",reset,in,out_) ;
   end

endmodule
```

The test vectors are read from the file **ex66data.csv** containing the following lines[2]:

```
1,0,0
0,1,1
0,2,1
0,3,4
0,4,4
0,5,8
1,7,0
```

Running this testbench using the Modelsim simulation program creates the waveform files shown in Figure 1 and, since the test vector contains an error, the following line is printed showing the input, actual output and the expected output given in the test vector:

```
# Error:  5    9    8
```

Exercise 3: How could you:

(a) terminate the simulation if a test vector failed?

(b) change the clock frequency to 10 MHz?

(c) print each test vector as it's read?

(d) assert the reset input for two clock cycles?

[2]This file is in Comma Separated Values (.csv) format.