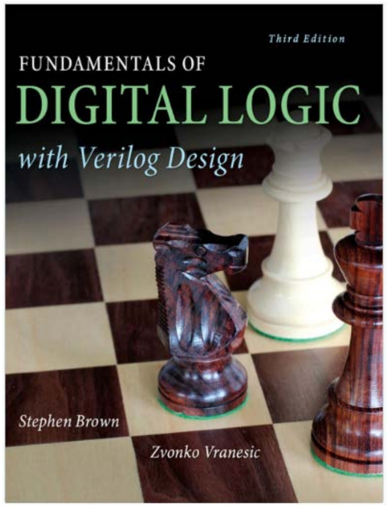
FPGAs and Verilog
Testing a Verilog design



Imperial College London: Digital Electronic 2
http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/index.html

Welcome to the world of ASIC
http://www.asic-world.com/

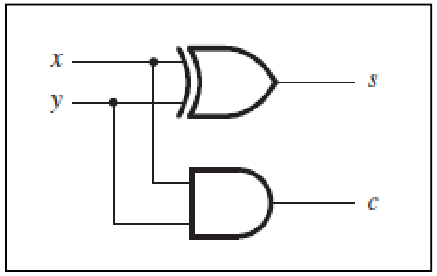
Brown and Vranesic, "Fundamentals of Digital Logic with Verilog Design"

EDA playground: online tool for writing and testing HDL code
https://www.edaplayground.com/

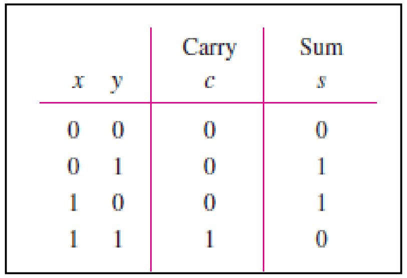
FPGAs and Verilog

Testing a Verilog design

- After writing Verilog code for a digital circuit, the next step is to test the functionality of the circuit

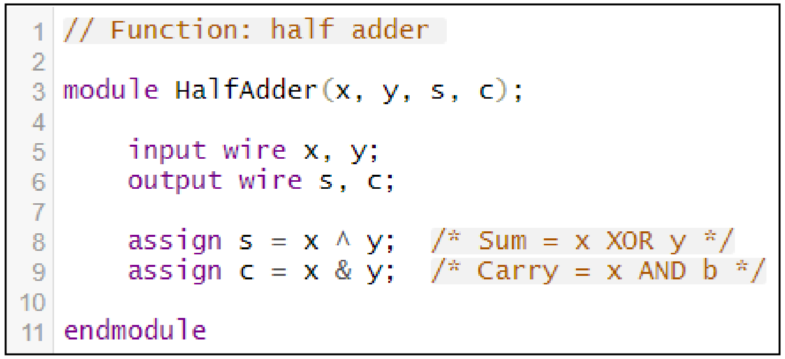


Circuit diagram of a half adder



x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder



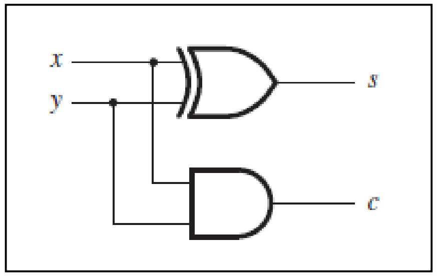
```
// Function: half adder

module HalfAdder(x, y, s, c);

    input wire x, y;
    output wire s, c;

    assign s = x ^ y;   /* Sum = x XOR y */
    assign c = x & y;   /* Carry = x AND b */

endmodule
```

Verilog code for a half adder

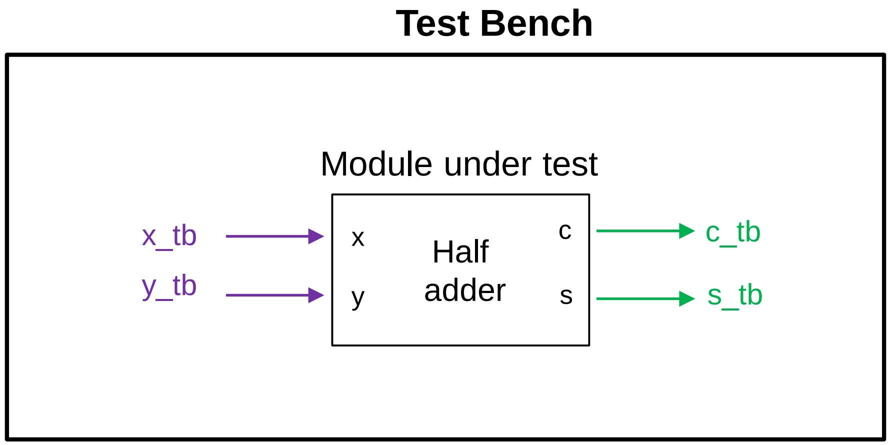
FPGAs and Verilog

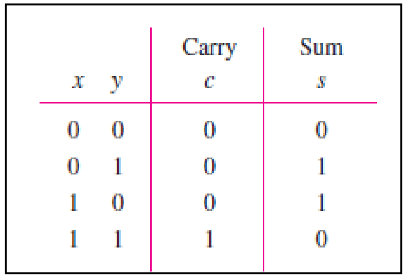
Testing a Verilog design

- After writing Verilog code for a digital circuit, the next step is to test the functionality of the circuit
- A test bench module can be written to generate input signals for the module under test and to observe the output signals



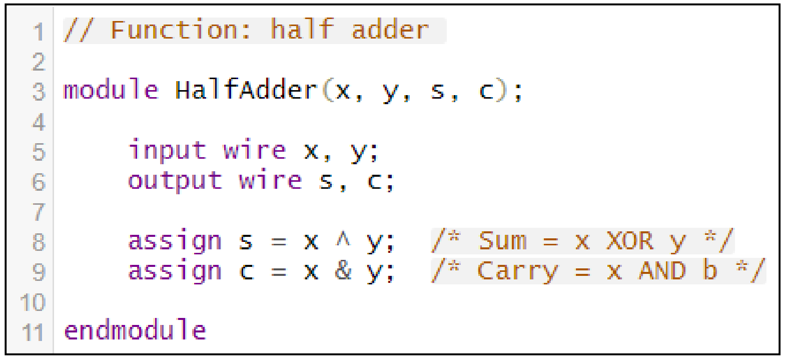
Circuit diagram of a half adder





x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder



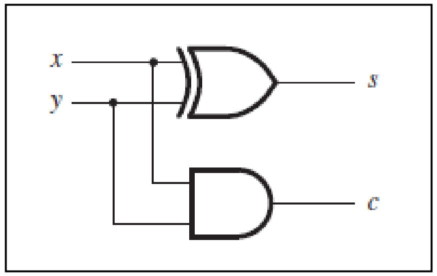
```
// Function: half adder

module HalfAdder(x, y, s, c);

    input wire x, y;
    output wire s, c;

    assign s = x ^ y;   /* Sum = x XOR y */
    assign c = x & y;   /* Carry = x AND b */

endmodule
```

Verilog code for a half adder

FPGAs and Verilog

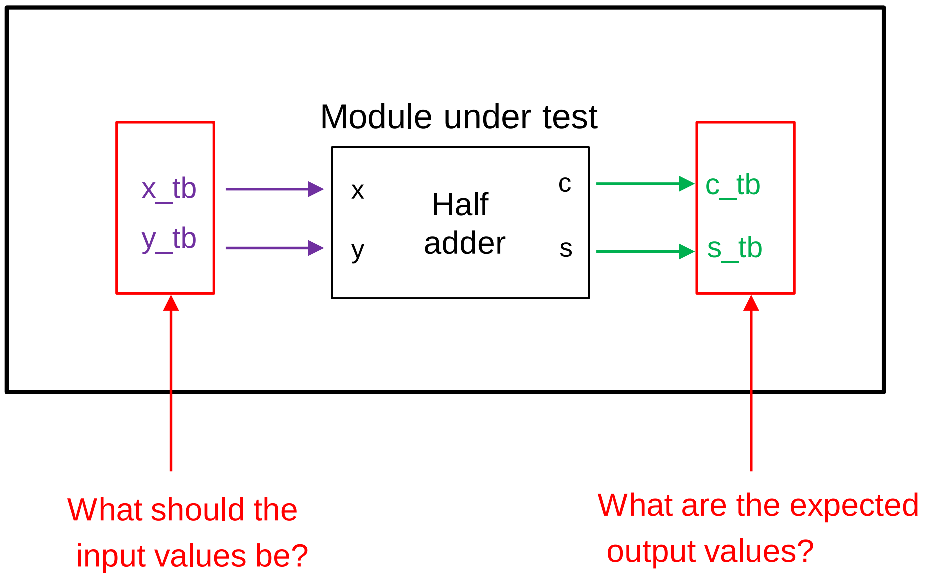
Testing a Verilog design

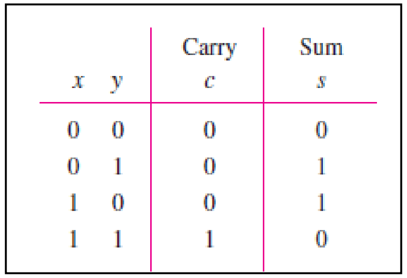
- After writing Verilog code for a digital circuit, the next step is to test the functionality of the circuit
- A test bench module can be written to generate input signals for the module under test and to observe the output signals



Circuit diagram of a half adder

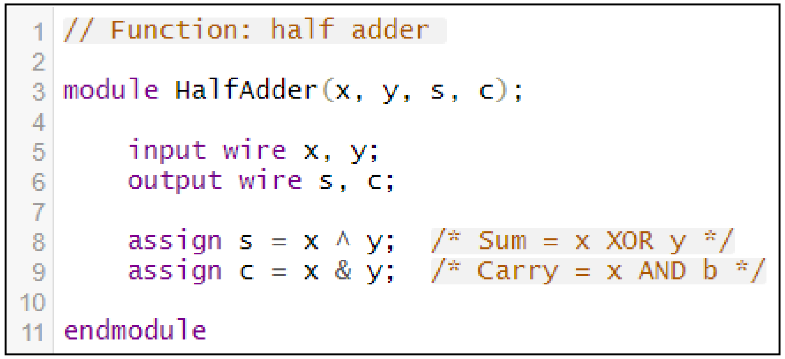
Test Bench


Module under test
x_tb
y_tb
x
y
Half adder
c
s
c_tb
s_tb
What should the input values be?
What are the expected output values?



x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder



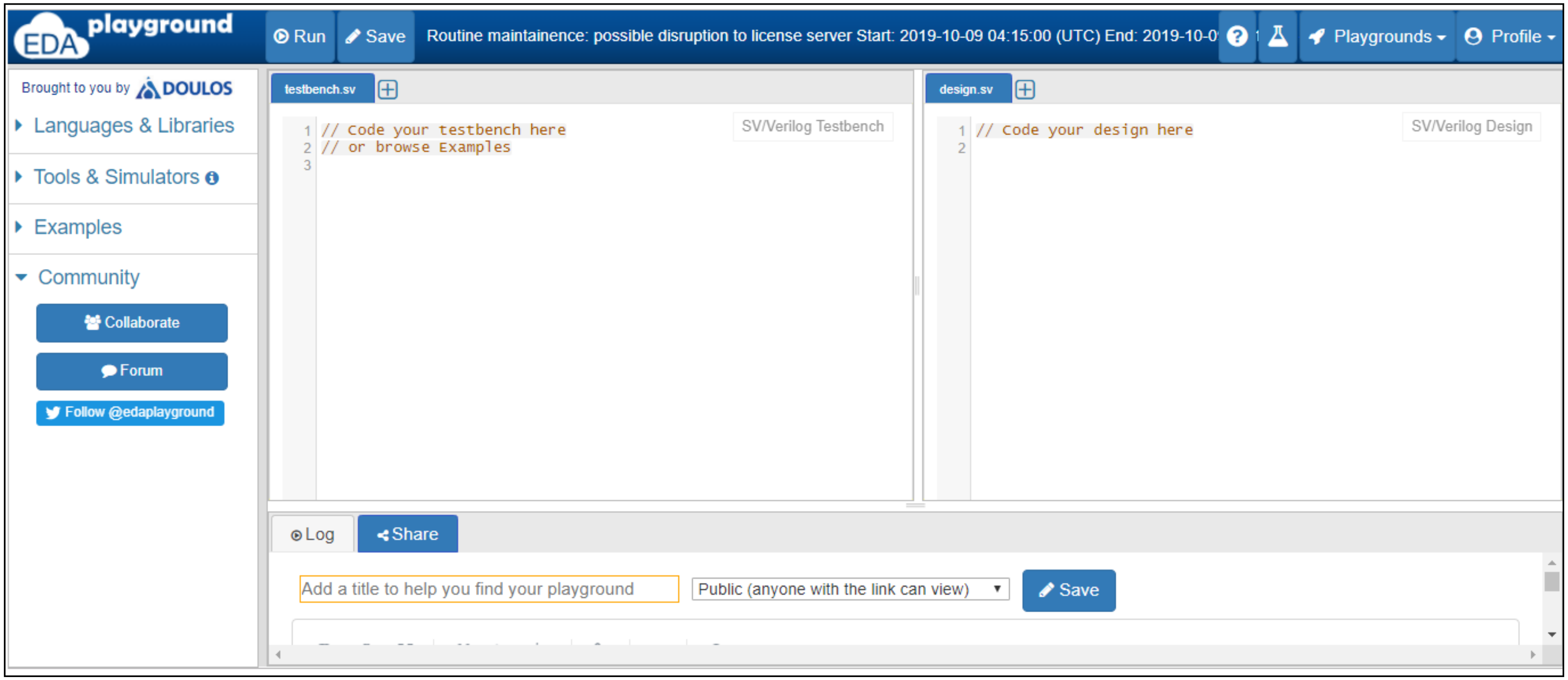
```verilog
// Function: half adder

module HalfAdder(x, y, s, c);

    input wire x, y;
    output wire s, c;

    assign s = x ^ y;   /* Sum = x XOR y */
    assign c = x & y;   /* Carry = x AND b */

endmodule
```

Verilog code for a half adder

FPGAs and Verilog

Testing a Verilog design

- EDA playground can be used to write Verilog code for both the digital circuit and the test bench: https://www.edaplayground.com/



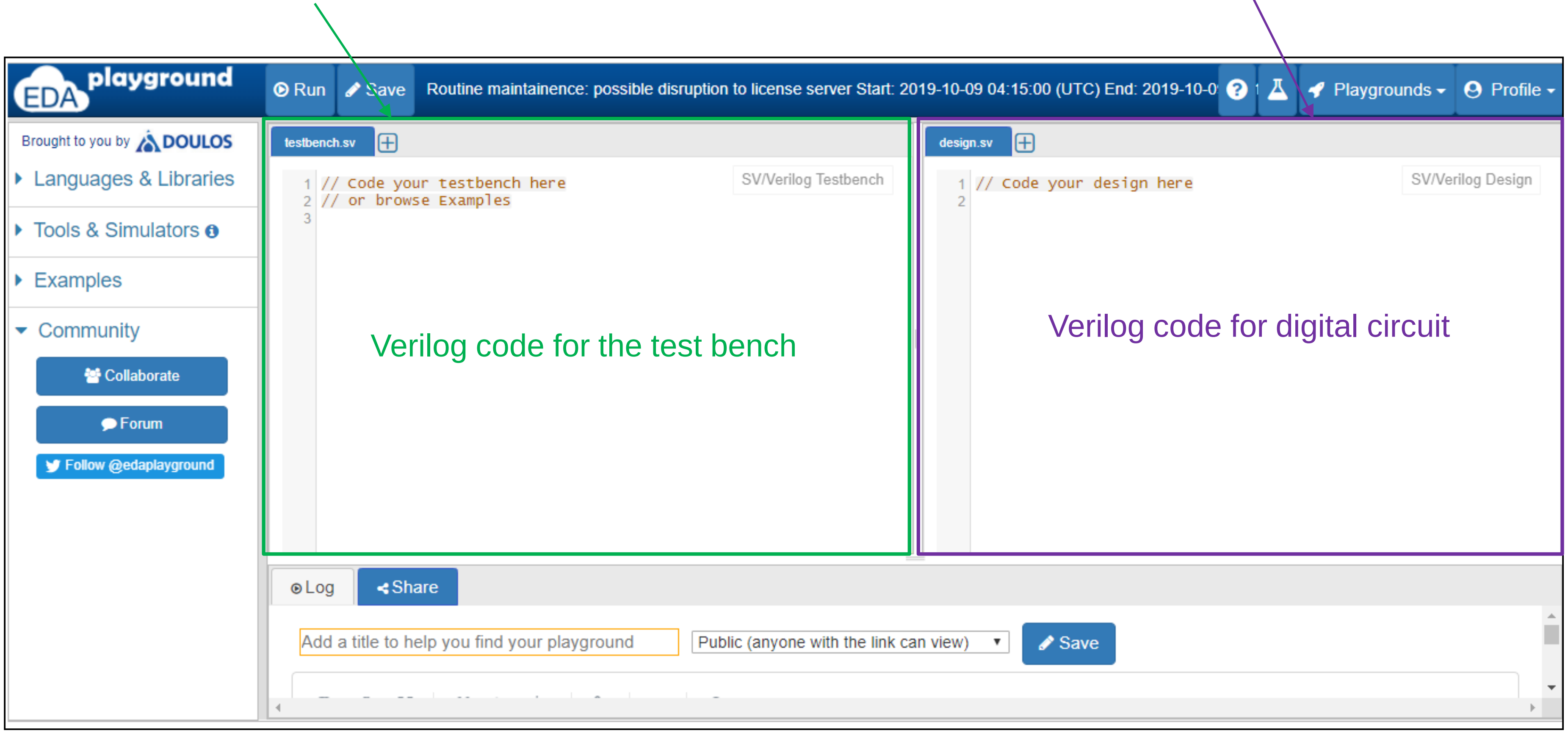
```
// Code your testbench here
// or browse Examples
```

```
// Code your design here
```

FPGAs and Verilog

Testing a Verilog design

- EDA playground can be used to write Verilog code for both the digital circuit and the test bench: https://www.edaplayground.com/



```
// Code your testbench here
// or browse Examples
```

```
// Code your design here
```

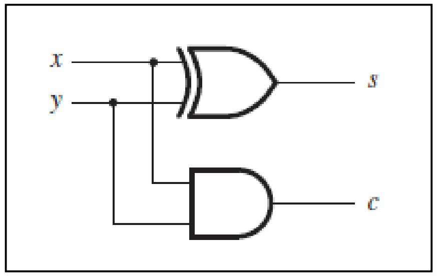
Verilog code for the test bench

Verilog code for digital circuit

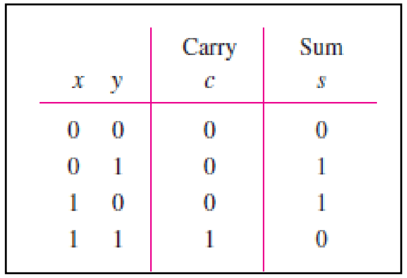
FPGAs and Verilog

Testing a Verilog design: steps

- Step 1: describe the expected operation of the digital circuit

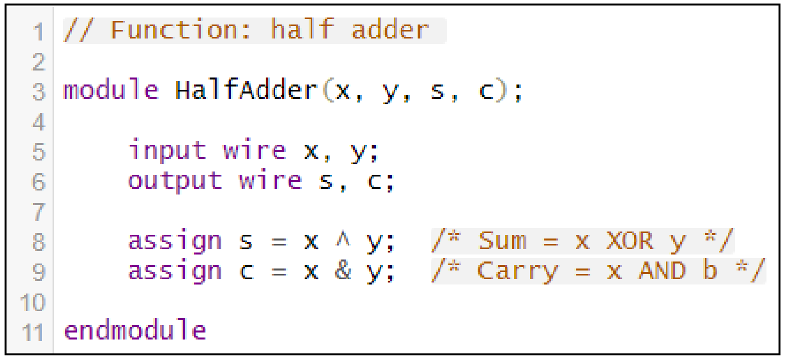


Circuit diagram of a half adder



x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder

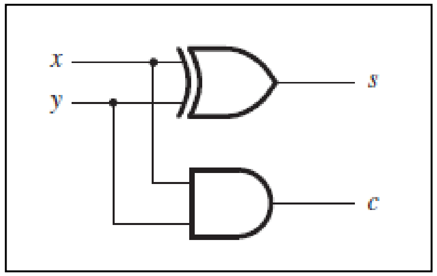


```verilog
// Function: half adder

module HalfAdder(x, y, s, c);

    input wire x, y;
    output wire s, c;

    assign s = x ^ y;   /* Sum = x XOR y */
    assign c = x & y;   /* Carry = x AND b */

endmodule
```

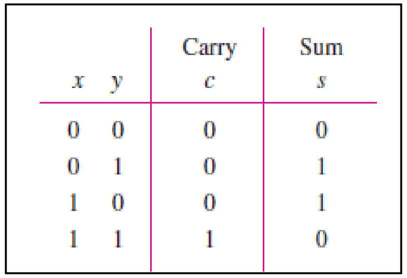
FPGAs and Verilog

Testing a Verilog design: steps

- Step 1: describe the expected operation of the digital circuit
- Step 2: write Verilog code for the test bench. Note that a test bench does not have any ports, ie. no inputs, no outputs

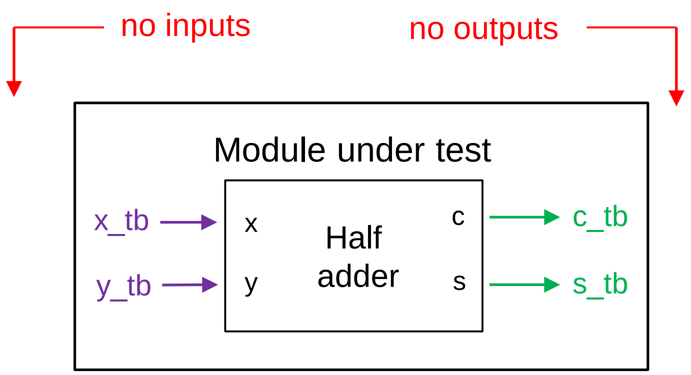


Circuit diagram of a half adder



		Carry	Sum
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder

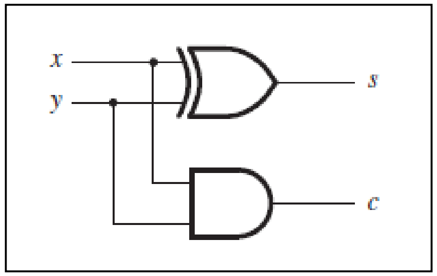


Test Bench

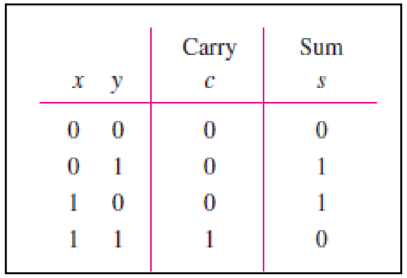
FPGAs and Verilog

Testing a Verilog design: steps

- Step 1: describe the expected operation of the digital circuit
- Step 2: write Verilog code for the test bench. Note that a test bench does not have any ports, ie. no inputs, no outputs
  - 2a) Define internal signals to connect with the inputs and outputs of the Module Under Test (MUT).

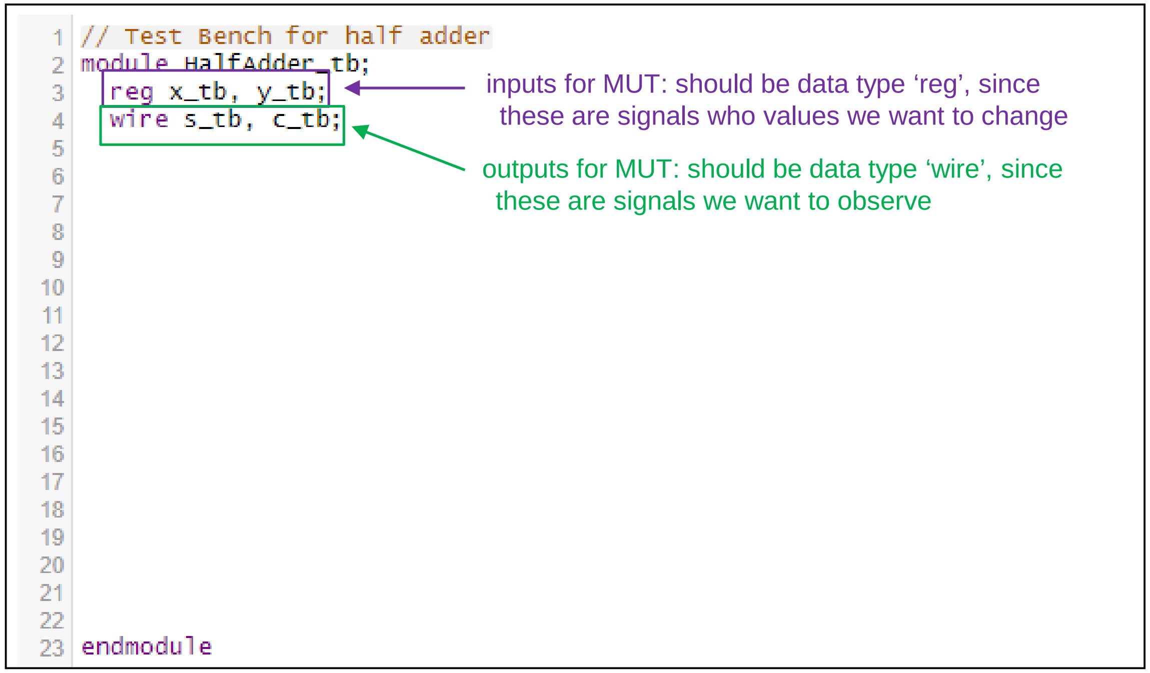


Circuit diagram of a half adder



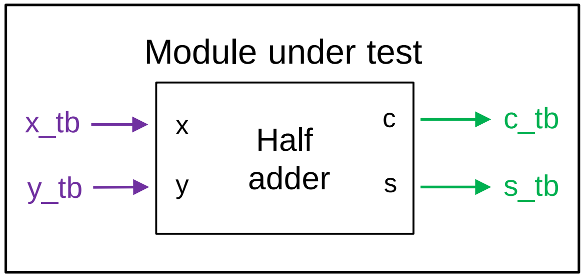
x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder



```
// Test Bench for half adder
module HalfAdder_tb;
  reg x_tb, y_tb;
  wire s_tb, c_tb;



















endmodule
```

inputs for MUT: should be data type 'reg', since these are signals who values we want to change
outputs for MUT: should be data type 'wire', since these are signals we want to observe

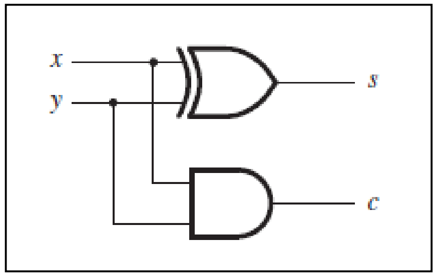


Test Bench

FPGAs and Verilog

Testing a Verilog design: steps

- Step 1: describe the expected operation of the digital circuit
- Step 2: write Verilog code for the test bench. Note that a test bench does not have any ports, ie. no inputs, no outputs
  - 2a) Define internal signals to connect with the inputs and outputs of the Module Under Test (MUT).
  - 2b) Instantiate the MUT in the test bench



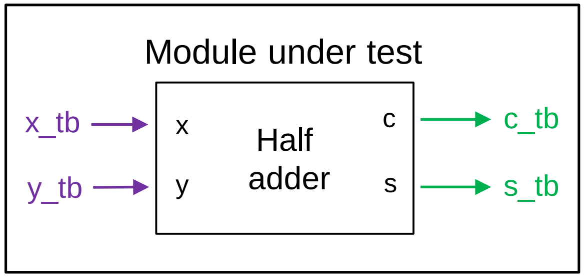
Circuit diagram of a half adder

```verilog
// Function: half adder

module HalfAdder(x, y, s, c);

    input wire x, y;
    output wire s, c;

    assign s = x ^ y;   /* Sum = x XOR y */
    assign c = x & y;   /* Carry = x AND b */

endmodule
```

```verilog
// Test Bench for half adder
module HalfAdder_tb;
  reg x_tb, y_tb;
  wire s_tb, c_tb;

  // Instantiate design under test
  HalfAdder MyHalfAdder(.x(x_tb), .y(y_tb), .s(s_tb), .c(c_tb));
















endmodule
```

format :   .signal_MUT(signal_TestBench)

MyHalfAdder is an instant of the module HalfAdder

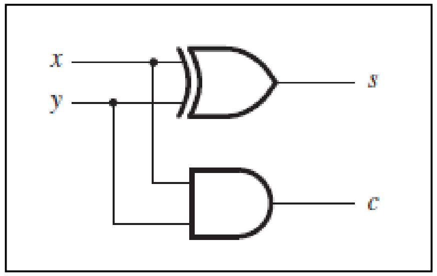


Test Bench

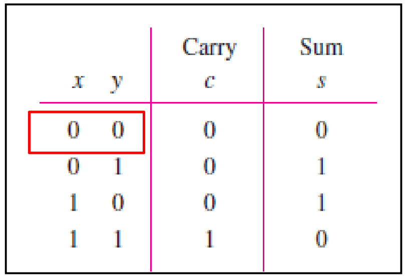
FPGAs and Verilog

Testing a Verilog design: steps

- Step 1: describe the expected operation of the digital circuit
- Step 2: write Verilog code for the test bench. Note that a test bench does not have any ports, ie. no inputs, no outputs
  - 2a) Define internal signals to connect with the inputs and outputs of the Module Under Test (MUT).
  - 2b) Instantiate the MUT in the test bench
  - 2c) Generate the input signals

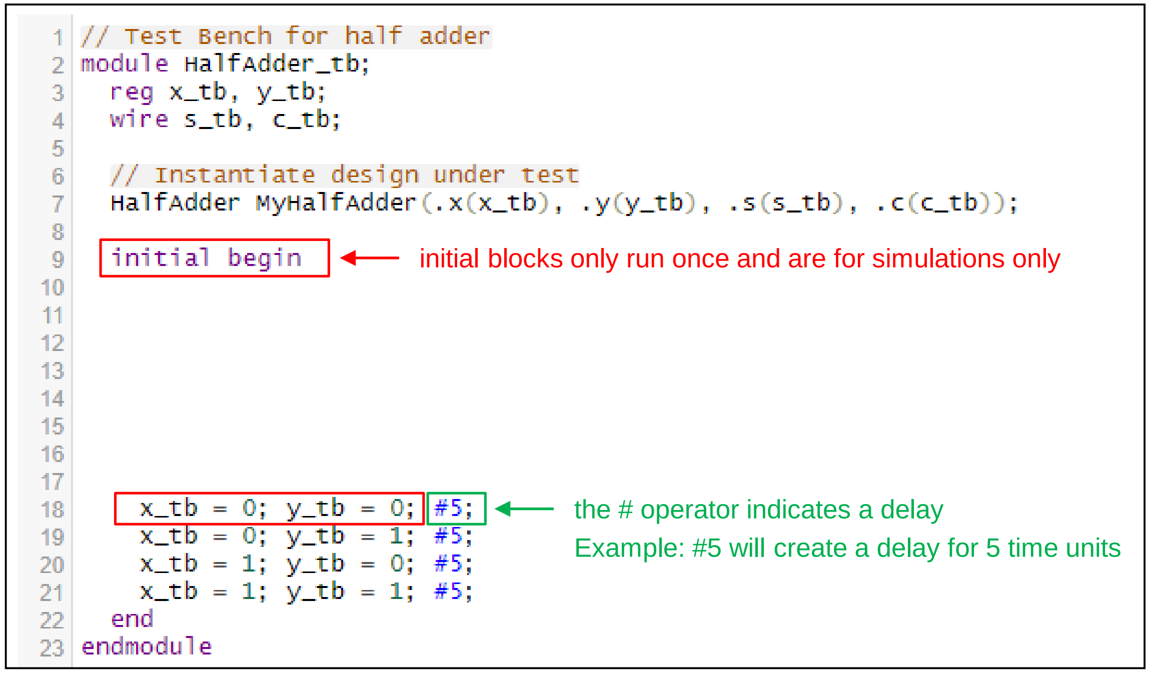


Circuit diagram of a half adder



x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

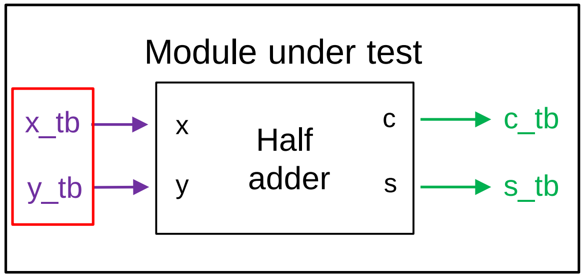
Truth table for a half adder



```verilog
// Test Bench for half adder
module HalfAdder_tb;
  reg x_tb, y_tb;
  wire s_tb, c_tb;

  // Instantiate design under test
  HalfAdder MyHalfAdder(.x(x_tb), .y(y_tb), .s(s_tb), .c(c_tb));

  initial begin









    x_tb = 0; y_tb = 0; #5;
    x_tb = 0; y_tb = 1; #5;
    x_tb = 1; y_tb = 0; #5;
    x_tb = 1; y_tb = 1; #5;
  end
endmodule
```

initial blocks only run once and are for simulations only

the # operator indicates a delay
Example: #5 will create a delay for 5 time units

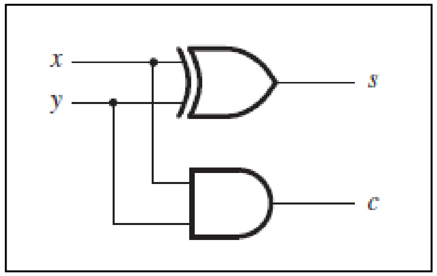


Test Bench

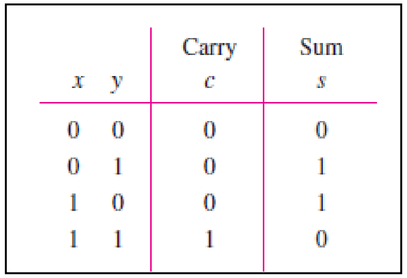
FPGAs and Verilog

Testing a Verilog design: steps

- Step 1: describe the expected operation of the digital circuit
- Step 2: write Verilog code for the test bench. Note that a test bench does not have any ports, ie. no inputs, no outputs
  - 2a) Define internal signals to connect with the inputs and outputs of the Module Under Test (MUT).
  - 2b) Instantiate the MUT in the test bench
  - 2c) Generate the input signals and observe the outputs

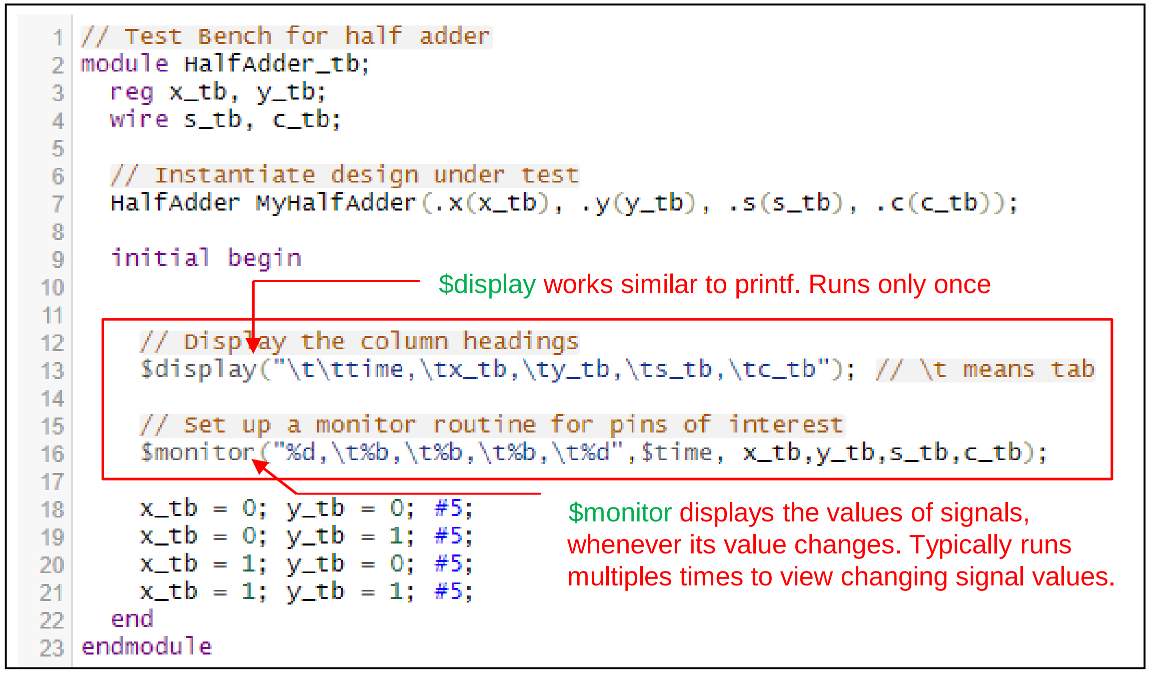


Circuit diagram of a half adder



x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

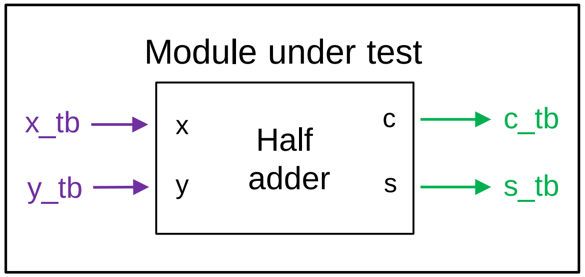
Truth table for a half adder



```
// Test Bench for half adder
module HalfAdder_tb;
  reg x_tb, y_tb;
  wire s_tb, c_tb;

  // Instantiate design under test
  HalfAdder MyHalfAdder(.x(x_tb), .y(y_tb), .s(s_tb), .c(c_tb));

  initial begin


    // Display the column headings
    $display("\t\ttime,\tx_tb,\ty_tb,\ts_tb,\tc_tb"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b,\t%b,\t%d",$time, x_tb,y_tb,s_tb,c_tb);

    x_tb = 0; y_tb = 0; #5;
    x_tb = 0; y_tb = 1; #5;
    x_tb = 1; y_tb = 0; #5;
    x_tb = 1; y_tb = 1; #5;
  end
endmodule
```

$display works similar to printf. Runs only once

$monitor displays the values of signals, whenever its value changes. Typically runs multiples times to view changing signal values.

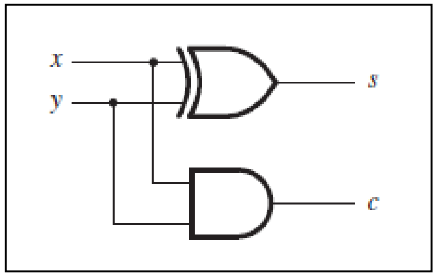


Test Bench

FPGAs and Verilog

Testing a Verilog design: steps

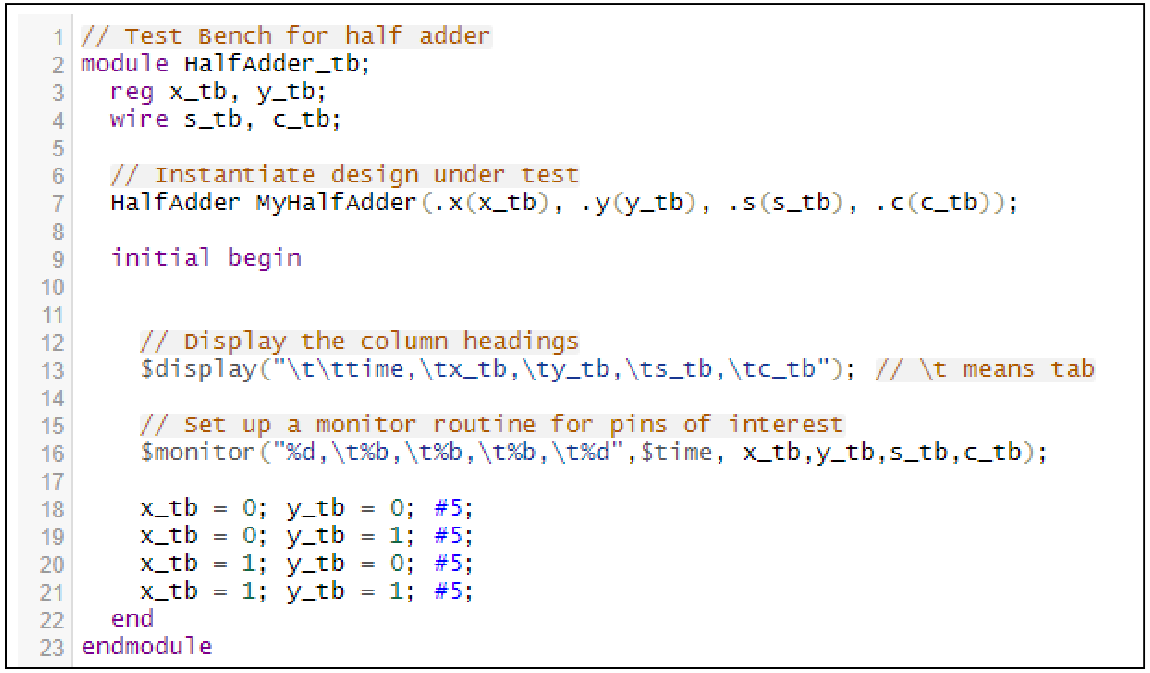
- Step 1: describe the expected operation of the digital circuit
- Step 2: write Verilog code for the test bench. Note that a test bench does not have any ports, ie. no inputs, no outputs
  - 2a) Define internal signals to connect with the inputs and outputs of the Module Under Test (MUT).
  - 2b) Instantiate the MUT in the test bench
  - 2c) Generate the input signals and observe the outputs



Circuit diagram of a half adder

x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder



```verilog
// Test Bench for half adder
module HalfAdder_tb;
  reg x_tb, y_tb;
  wire s_tb, c_tb;

  // Instantiate design under test
  HalfAdder MyHalfAdder(.x(x_tb), .y(y_tb), .s(s_tb), .c(c_tb));

  initial begin


    // Display the column headings
    $display("\t\ttime,\tx_tb,\ty_tb,\ts_tb,\tc_tb"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b,\t%b,\t%d",$time, x_tb,y_tb,s_tb,c_tb);

    x_tb = 0; y_tb = 0; #5;
    x_tb = 0; y_tb = 1; #5;
    x_tb = 1; y_tb = 0; #5;
    x_tb = 1; y_tb = 1; #5;
  end
endmodule
```

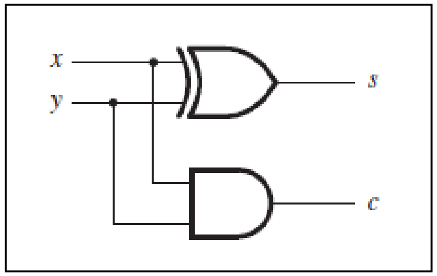
```
time,   x_tb,   y_tb,   s_tb,   c_tb
   0,   0,      0,      0,      0
   5,   0,      1,      1,      0
  10,   1,      0,      1,      0
  15,   1,      1,      0,      1
```

Display to the user after Verilog test bench code executes

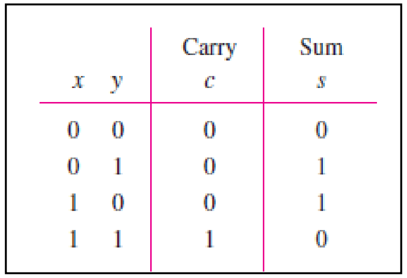
FPGAs and Verilog

Testing a Verilog design: steps

- Step 1: describe the expected operation of the digital circuit
- Step 2: write Verilog code for the test bench. Note that a test bench does not have any ports, ie. no inputs, no outputs
  - 2a) Define internal signals to connect with the inputs and outputs of the Module Under Test (MUT).
  - 2b) Instantiate the MUT in the test bench
  - 2c) Generate the input signals and observe the outputs

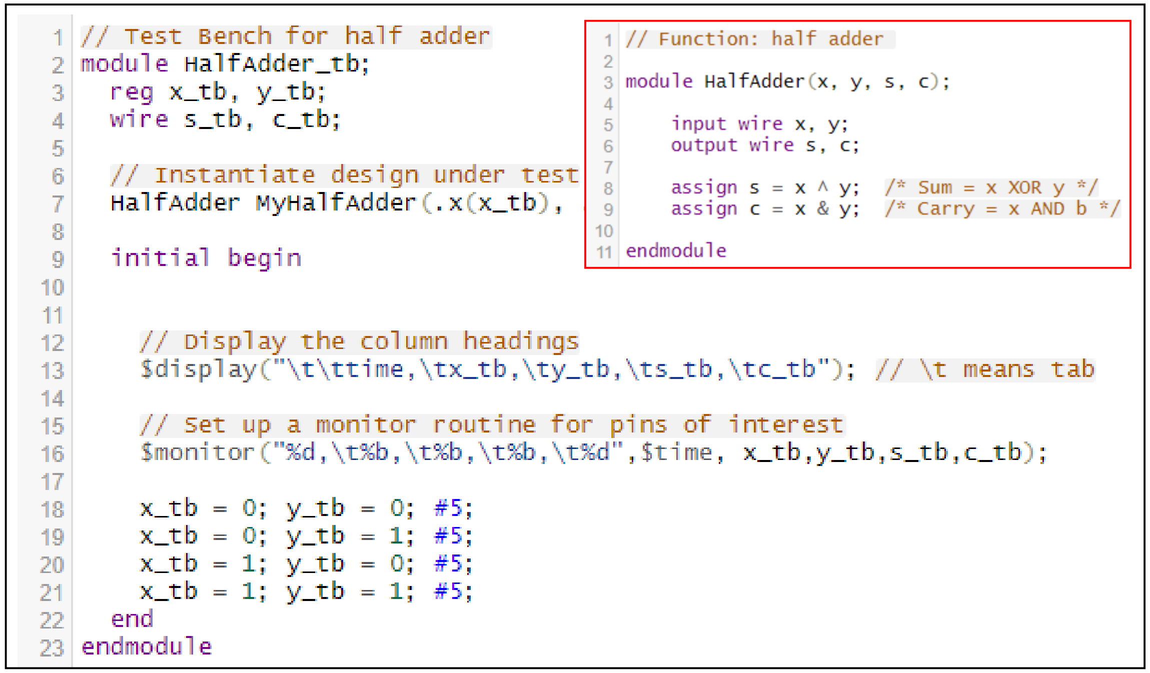


Circuit diagram of a half adder

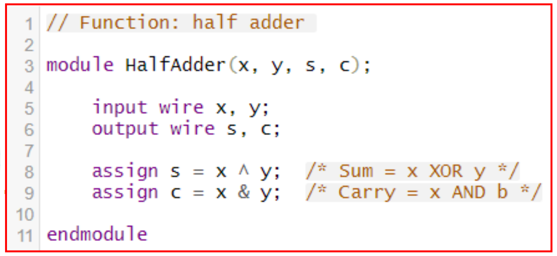


x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder

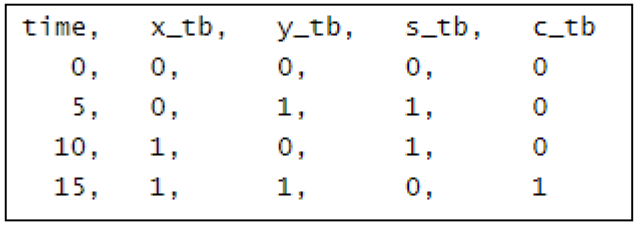


```verilog
// Test Bench for half adder
module HalfAdder_tb;
  reg x_tb, y_tb;
  wire s_tb, c_tb;

  // Instantiate design under test
  HalfAdder MyHalfAdder(.x(x_tb),

  initial begin


    // Display the column headings
    $display("\t\ttime,\tx_tb,\ty_tb,\ts_tb,\tc_tb"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b,\t%b,\t%d",$time, x_tb,y_tb,s_tb,c_tb);

    x_tb = 0; y_tb = 0; #5;
    x_tb = 0; y_tb = 1; #5;
    x_tb = 1; y_tb = 0; #5;
    x_tb = 1; y_tb = 1; #5;
  end
endmodule
```



```verilog
// Function: half adder

module HalfAdder(x, y, s, c);

    input wire x, y;
    output wire s, c;

    assign s = x ^ y;   /* Sum = x XOR y */
    assign c = x & y;   /* Carry = x AND b */

endmodule
```

Equivalent
∴ Verilog code for HalfAdder was accurate



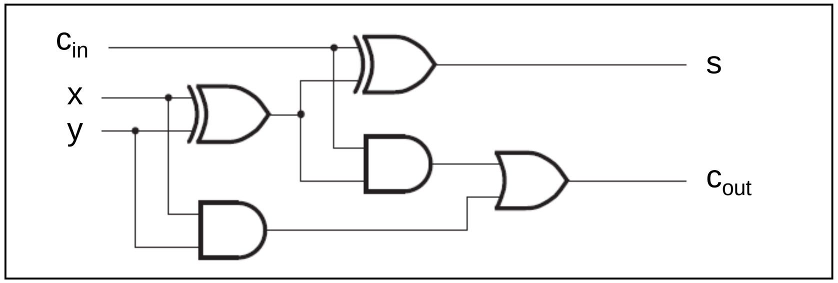
```
time,   x_tb,   y_tb,   s_tb,   c_tb
   0,   0,      0,      0,      0
   5,   0,      1,      1,      0
  10,   1,      0,      1,      0
  15,   1,      1,      0,      1
```

Display to the user after Verilog test bench code executes

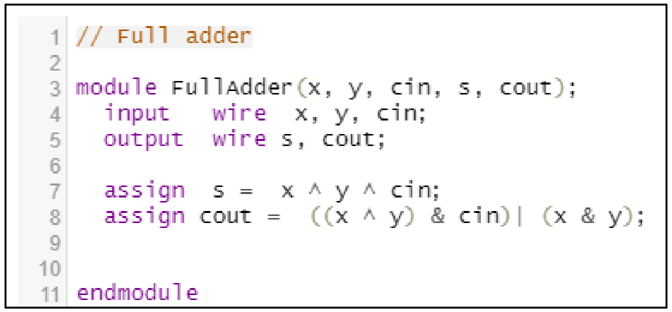
FPGAs and Verilog

Testing a Verilog design: full adder

- Write the Verilog code for the test bench



Full adder

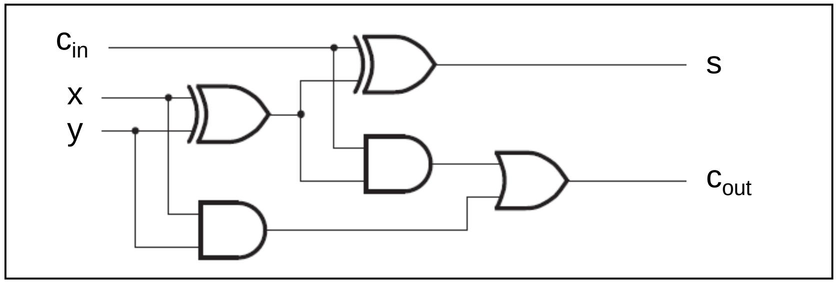


```
// Full adder

module FullAdder(x, y, cin, s, cout);
  input   wire  x, y, cin;
  output  wire s, cout;

  assign  s =  x ^ y ^ cin;
  assign cout =  ((x ^ y) & cin)| (x & y);


endmodule
```

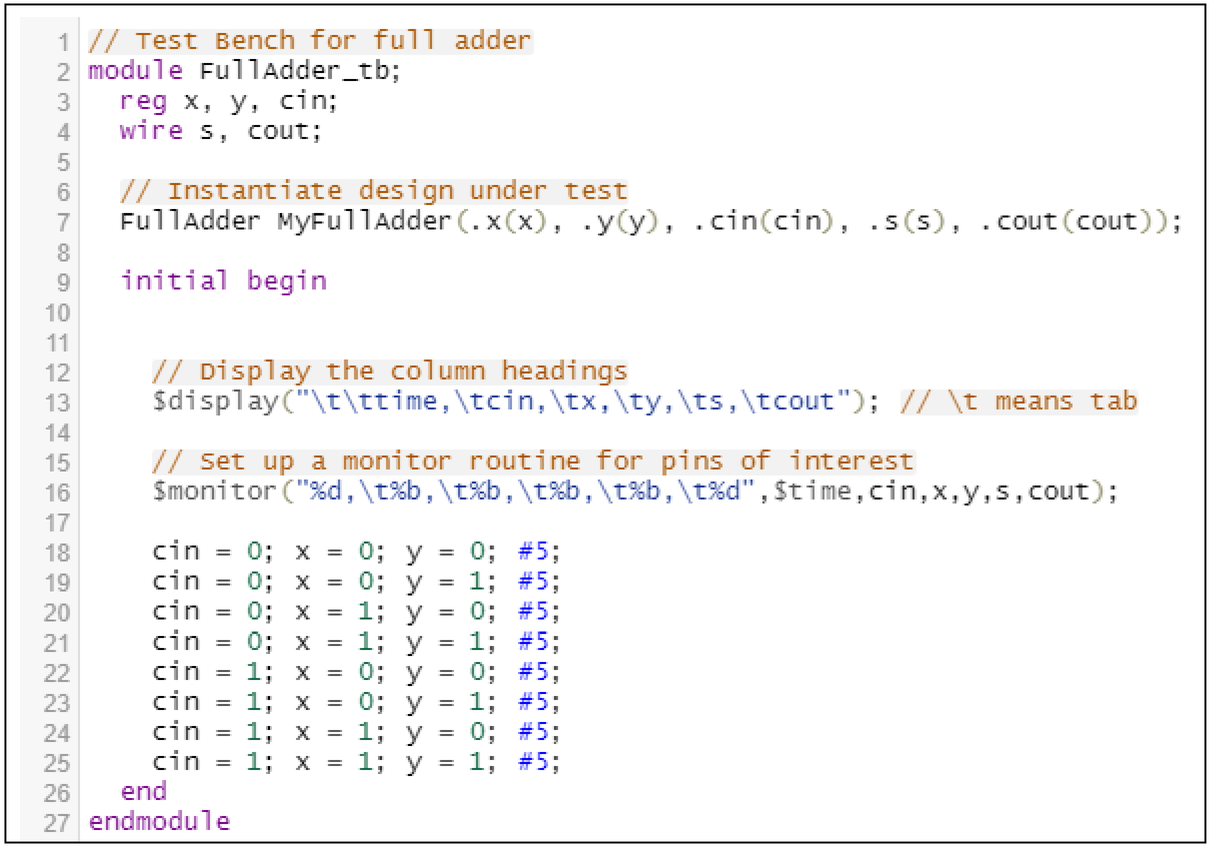
FPGAs and Verilog

Testing a Verilog design: full adder

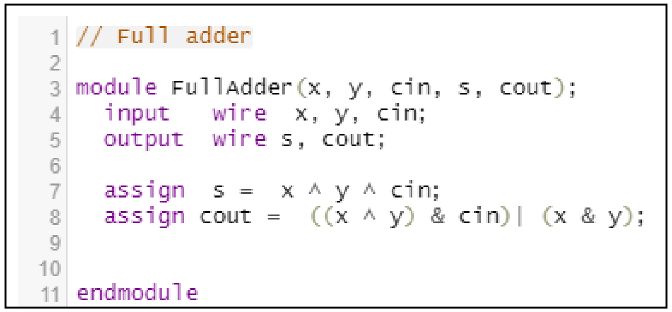
- Write the Verilog code for the test bench



Full adder



```
// Test Bench for full adder
module FullAdder_tb;
  reg x, y, cin;
  wire s, cout;

  // Instantiate design under test
  FullAdder MyFullAdder(.x(x), .y(y), .cin(cin), .s(s), .cout(cout));

  initial begin


    // Display the column headings
    $display("\t\ttime,\tcin,\tx,\ty,\ts,\tcout"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b,\t%b,\t%b,\t%d",$time,cin,x,y,s,cout);

    cin = 0;  x = 0;  y = 0;  #5;
    cin = 0;  x = 0;  y = 1;  #5;
    cin = 0;  x = 1;  y = 0;  #5;
    cin = 0;  x = 1;  y = 1;  #5;
    cin = 1;  x = 0;  y = 0;  #5;
    cin = 1;  x = 0;  y = 1;  #5;
    cin = 1;  x = 1;  y = 0;  #5;
    cin = 1;  x = 1;  y = 1;  #5;
  end
endmodule
```

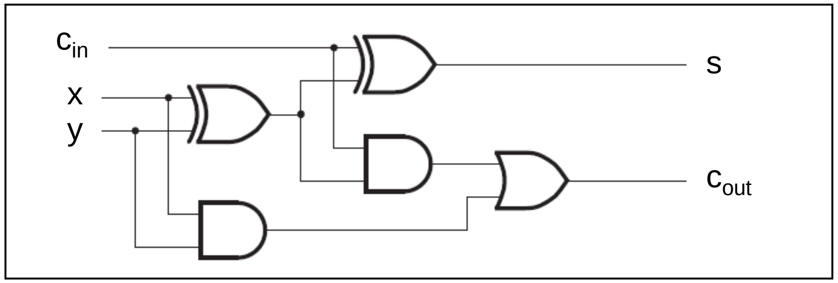


```
// Full adder

module FullAdder(x, y, cin, s, cout);
  input   wire  x, y, cin;
  output  wire s, cout;

  assign  s =  x ^ y ^ cin;
  assign cout =  ((x ^ y) & cin)| (x & y);


endmodule
```

FPGAs and Verilog

Testing a Verilog design: full adder

- Write the Verilog code for the test bench



Full adder

```verilog
// Test Bench for full adder
module FullAdder_tb;
  reg x, y, cin;
  wire s, cout;

  // Instantiate design under test
  FullAdder MyFullAdder(.x(x), .y(y), .cin(cin), .s(s), .cout(cout));

  initial begin


    // Display the column headings
    $display("\t\ttime,\tcin,\tx,\ty,\ts,\tcout"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b,\t%b,\t%b,\t%d",$time,cin,x,y,s,cout);

    cin = 0; x = 0; y = 0; #5;
    cin = 0; x = 0; y = 1; #5;
    cin = 0; x = 1; y = 0; #5;
    cin = 0; x = 1; y = 1; #5;
    cin = 1; x = 0; y = 0; #5;
    cin = 1; x = 0; y = 1; #5;
    cin = 1; x = 1; y = 0; #5;
    cin = 1; x = 1; y = 1; #5;
  end
endmodule
```

```verilog
// Full adder

module FullAdder(x, y, cin, s, cout);
  input   wire  x, y, cin;
  output  wire s, cout;

  assign  s =  x ^ y ^ cin;
  assign cout =  ((x ^ y) & cin)| (x & y);


endmodule
```

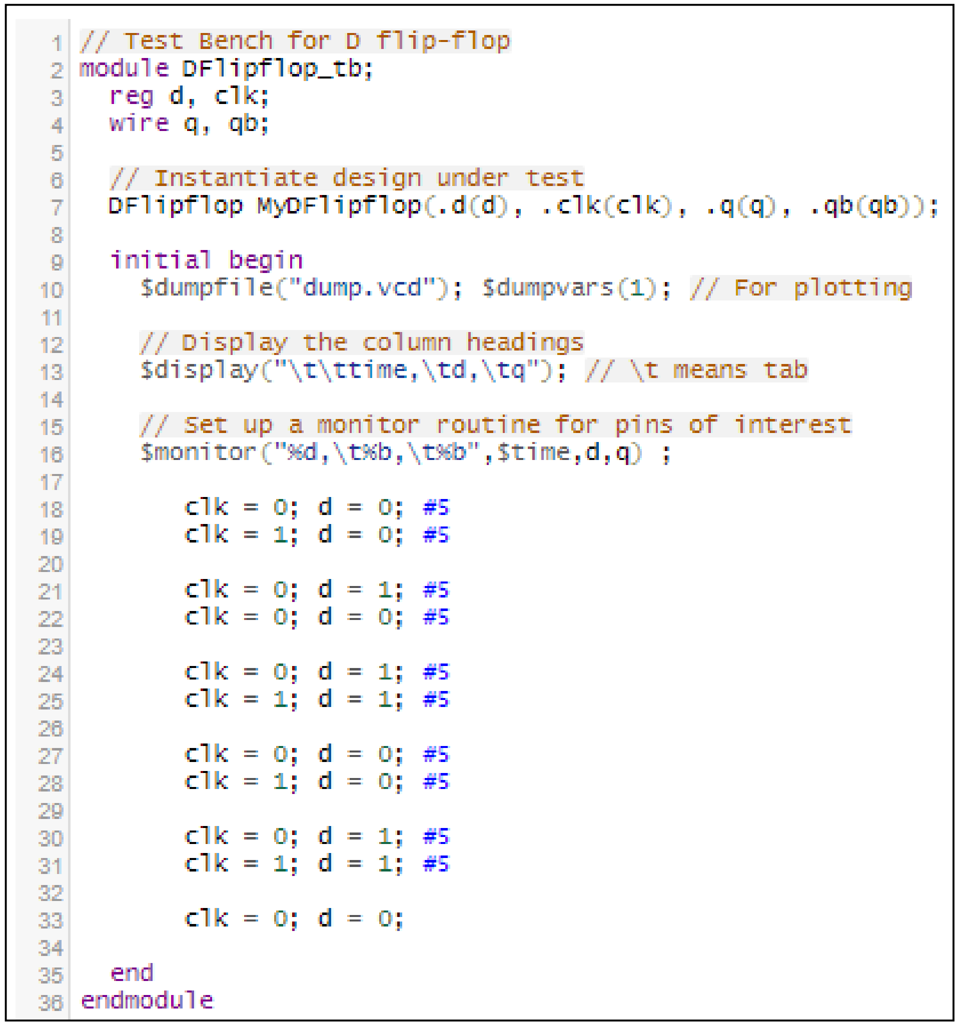
```
time,   cin,    x,      y,      s,      cout
   0,   0,      0,      0,      0,      0
   5,   0,      0,      1,      1,      0
  10,   0,      1,      0,      1,      0
  15,   0,      1,      1,      0,      1
  20,   1,      0,      0,      1,      0
  25,   1,      0,      1,      0,      1
  30,   1,      1,      0,      0,      1
  35,   1,      1,      1,      1,      1
```

After running code
∴ FullAdder was accurately coded in Verilog

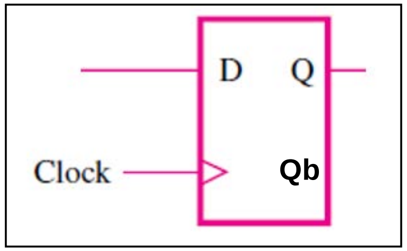
FPGAs and Verilog

Testing a Verilog design: D flip-flop

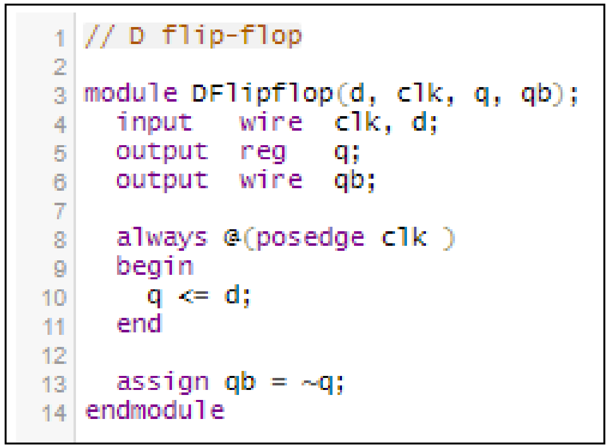
- Steps: follow the same steps as outlined previously



```
// Test Bench for D flip-flop
module DFlipflop_tb;
  reg d, clk;
  wire q, qb;

  // Instantiate design under test
  DFlipflop MyDFlipflop(.d(d), .clk(clk), .q(q), .qb(qb));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\td,\tq"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b",$time,d,q) ;

       clk = 0; d = 0; #5
       clk = 1; d = 0; #5

       clk = 0; d = 1; #5
       clk = 0; d = 0; #5

       clk = 0; d = 1; #5
       clk = 1; d = 1; #5

       clk = 0; d = 0; #5
       clk = 1; d = 0; #5

       clk = 0; d = 1; #5
       clk = 1; d = 1; #5

       clk = 0; d = 0;

  end
endmodule
```



Symbol of a D flip-flop

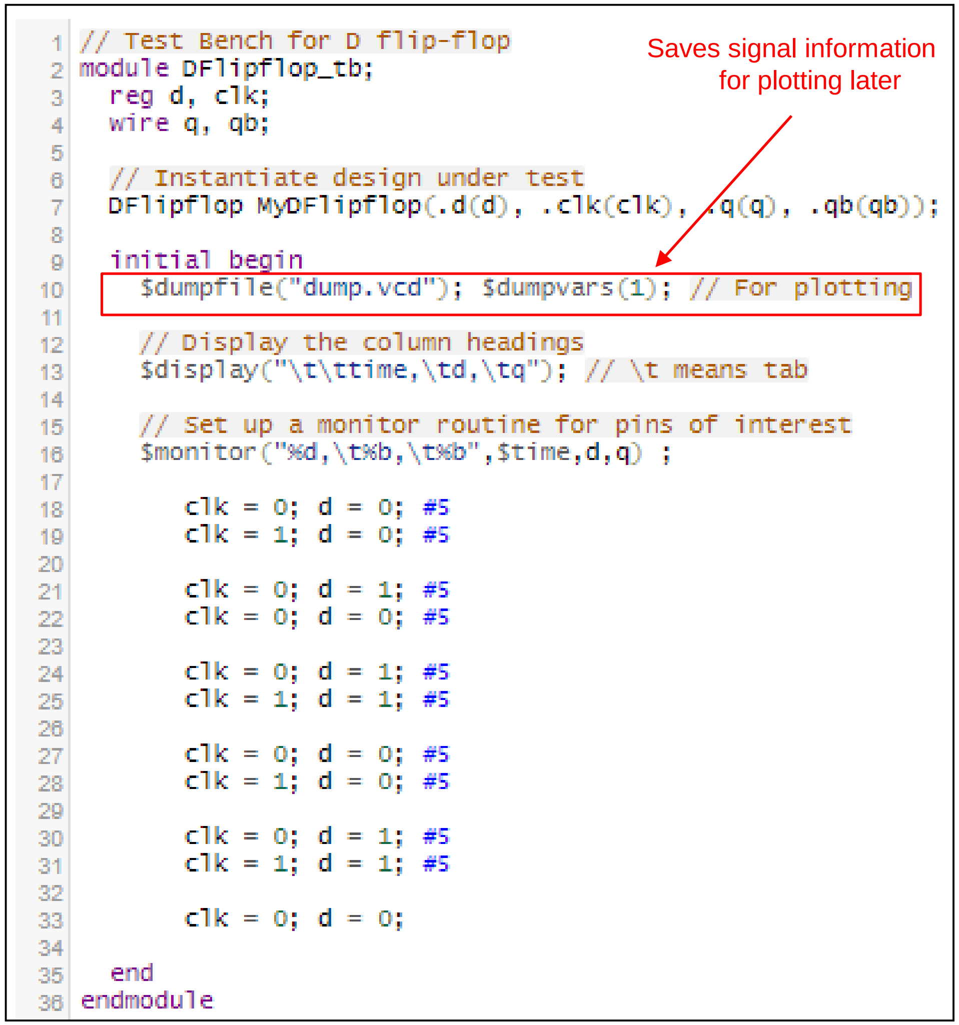


```
// D flip-flop

module DFlipflop(d, clk, q, qb);
  input   wire  clk, d;
  output  reg   q;
  output  wire  qb;

  always @(posedge clk )
  begin
    q <= d;
  end

  assign qb = ~q;
endmodule
```

FPGAs and Verilog

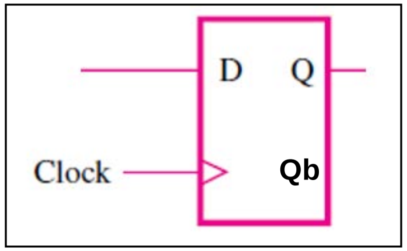
Testing a Verilog design: D flip-flop

- Steps: follow the same steps as outlined previously

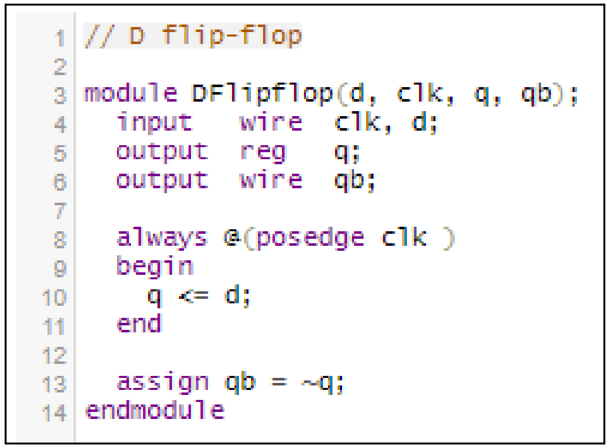


```verilog
// Test Bench for D flip-flop
module DFlipflop_tb;
  reg d, clk;
  wire q, qb;

  // Instantiate design under test
  DFlipflop MyDFlipflop(.d(d), .clk(clk), .q(q), .qb(qb));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\td,\tq"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b",$time,d,q) ;

      clk = 0; d = 0; #5
      clk = 1; d = 0; #5

      clk = 0; d = 1; #5
      clk = 0; d = 0; #5

      clk = 0; d = 1; #5
      clk = 1; d = 1; #5

      clk = 0; d = 0; #5
      clk = 1; d = 0; #5

      clk = 0; d = 1; #5
      clk = 1; d = 1; #5

      clk = 0; d = 0;

  end
endmodule
```

Saves signal information for plotting later



Symbol of a D flip-flop

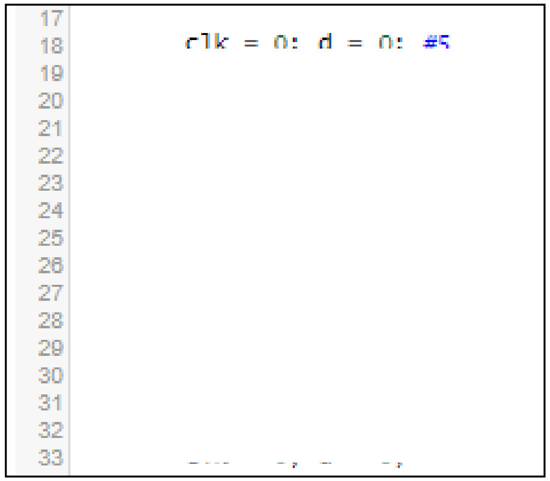


```verilog
// D flip-flop

module DFlipflop(d, clk, q, qb);
  input   wire  clk, d;
  output  reg   q;
  output  wire  qb;

  always @(posedge clk )
  begin
    q <= d;
  end

  assign qb = ~q;
endmodule
```

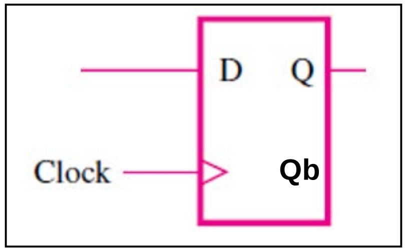
FPGAs and Verilog

Testing a Verilog design: D flip-flop

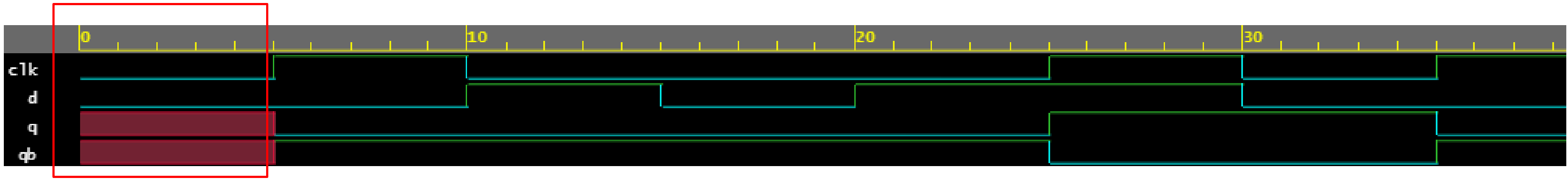
- Changing the inputs: clk and d
- Observing the outputs: q and qb



```
        clk = 0; d = 0; #5
```



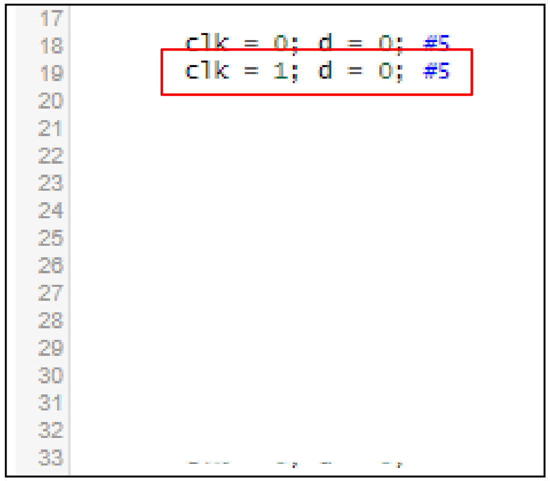
Symbol of a D flip-flop



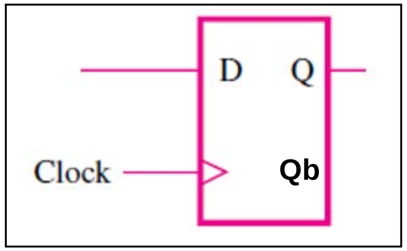
FPGAs and Verilog

Testing a Verilog design: D flip-flop

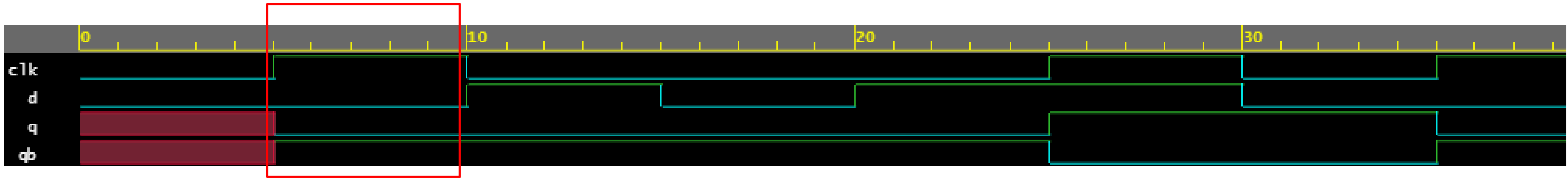
- Changing the inputs: clk and d
- Observing the outputs: q and qb



```
        clk = 0; d = 0; #5
        clk = 1; d = 0; #5
```



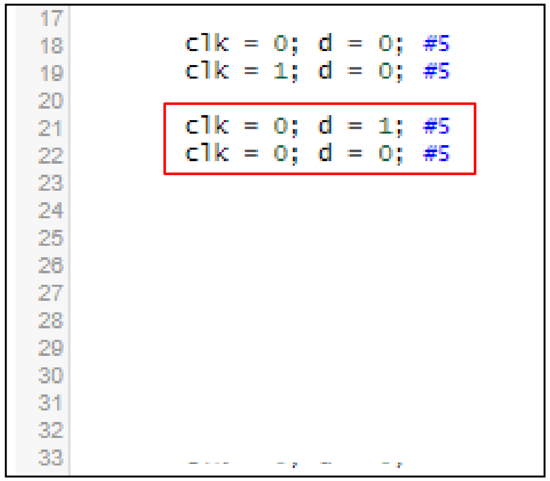
Symbol of a D flip-flop



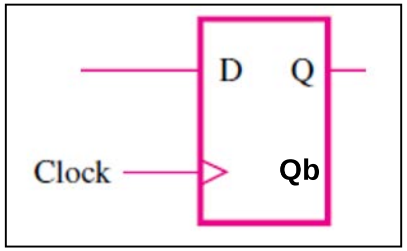
FPGAs and Verilog

Testing a Verilog design: D flip-flop

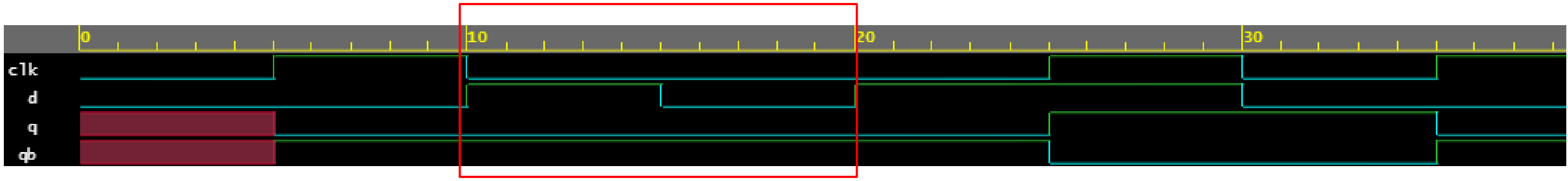
- Changing the inputs: clk and d
- Observing the outputs: q and qb



```
        clk = 0; d = 0; #5
        clk = 1; d = 0; #5

        clk = 0; d = 1; #5
        clk = 0; d = 0; #5
```



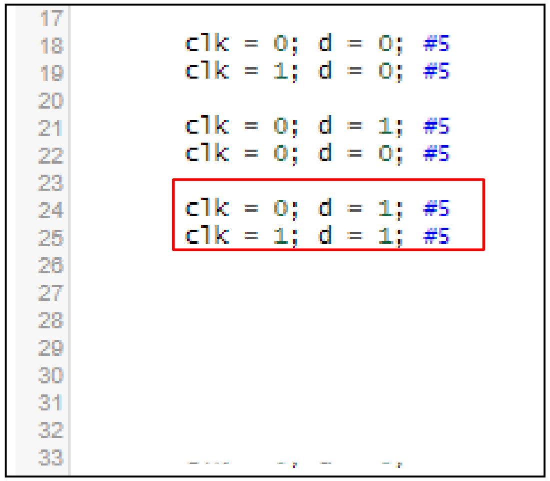
Symbol of a D flip-flop



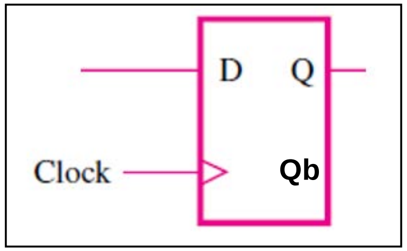
FPGAs and Verilog

Testing a Verilog design: D flip-flop

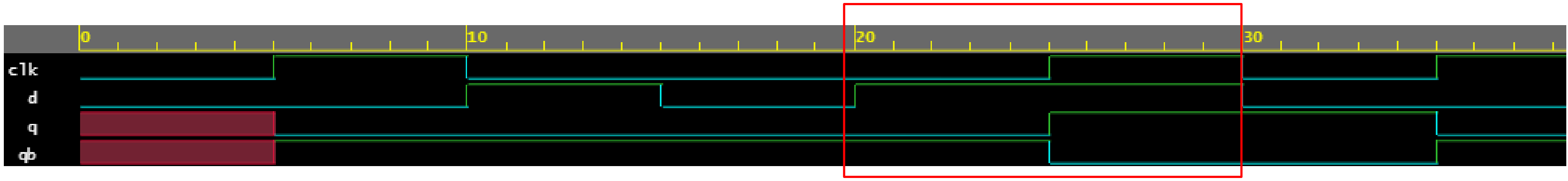
- Changing the inputs: clk and d
- Observing the outputs: q and qb



```
        clk = 0; d = 0; #5
        clk = 1; d = 0; #5

        clk = 0; d = 1; #5
        clk = 0; d = 0; #5

        clk = 0; d = 1; #5
        clk = 1; d = 1; #5
```



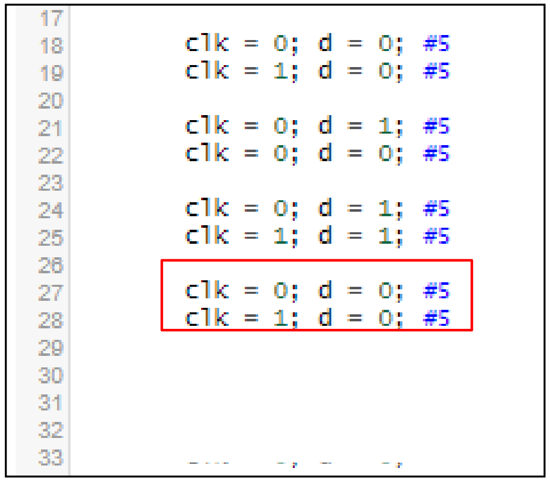
Symbol of a D flip-flop



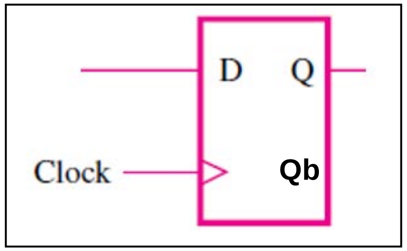
FPGAs and Verilog

Testing a Verilog design: D flip-flop

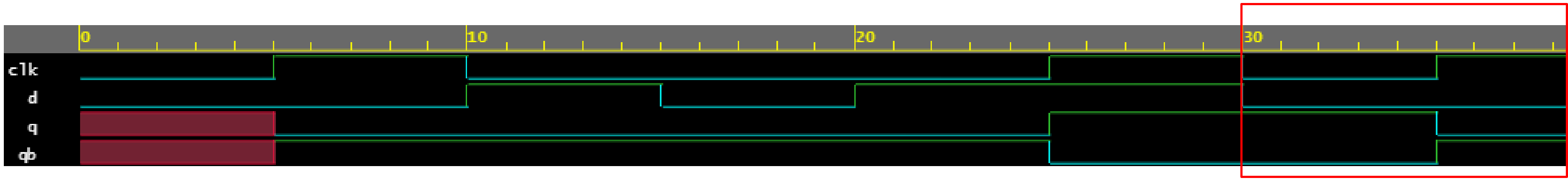
- Changing the inputs: clk and d
- Observing the outputs: q and qb



```
        clk = 0; d = 0; #5
        clk = 1; d = 0; #5

        clk = 0; d = 1; #5
        clk = 0; d = 0; #5

        clk = 0; d = 1; #5
        clk = 1; d = 1; #5

        clk = 0; d = 0; #5
        clk = 1; d = 0; #5
```



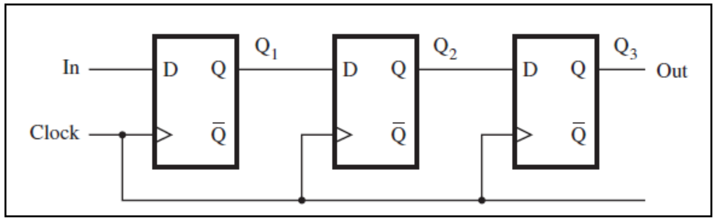
Symbol of a D flip-flop



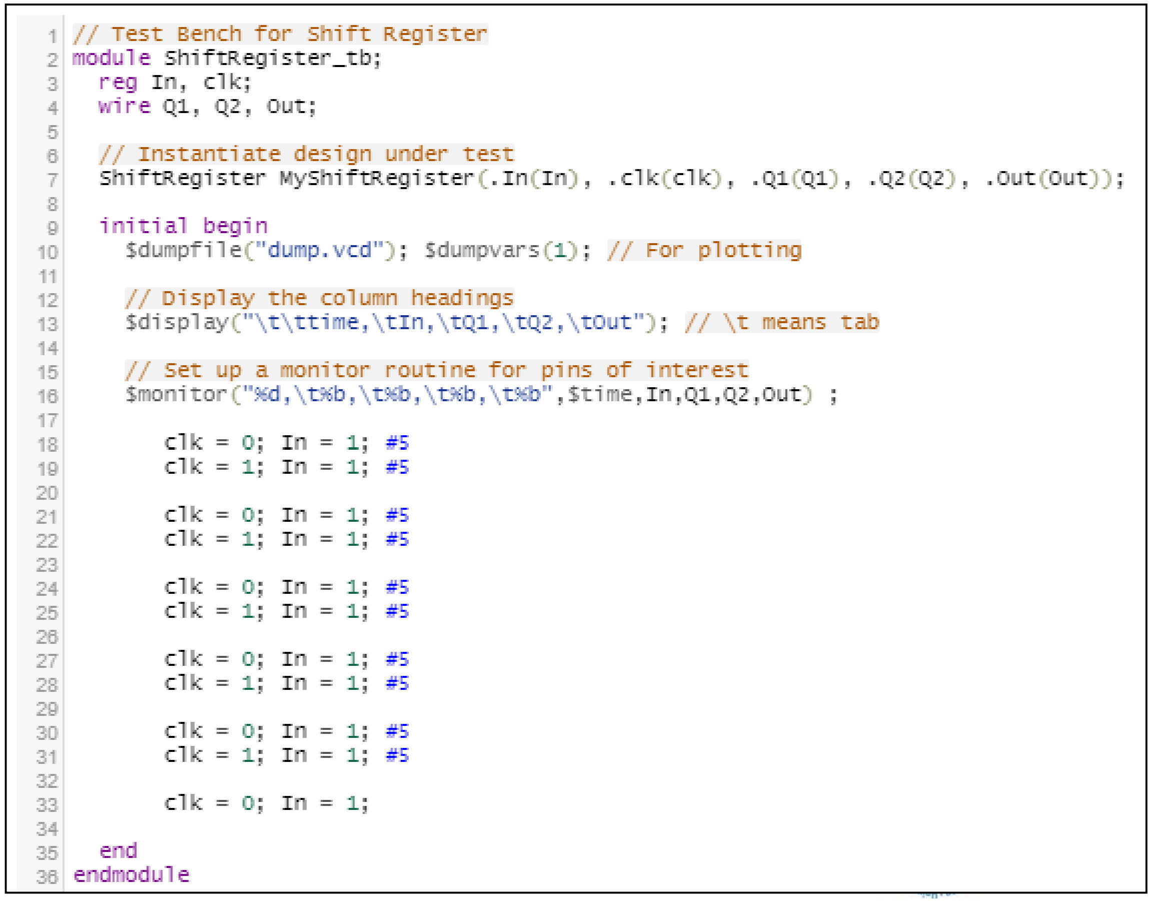
FPGAs and Verilog

Testing a Verilog design: shift register

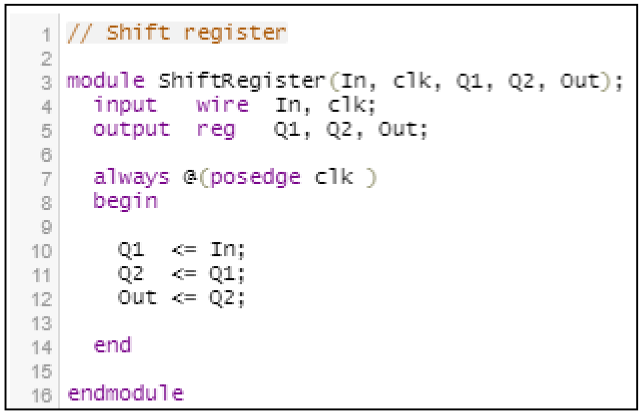
- Steps: follow the same steps as outlined previously



Circuit diagram of a 3-bit shift register



```
// Test Bench for Shift Register
module ShiftRegister_tb;
  reg In, clk;
  wire Q1, Q2, Out;

  // Instantiate design under test
  ShiftRegister MyShiftRegister(.In(In), .clk(clk), .Q1(Q1), .Q2(Q2), .Out(Out));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\tIn,\tQ1,\tQ2,\tOut"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b,\t%b,\t%b",$time,In,Q1,Q2,Out) ;

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1;

  end
endmodule
```

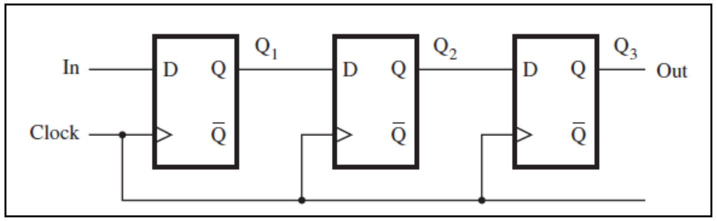


```
// Shift register

module ShiftRegister(In, clk, Q1, Q2, Out);
  input   wire  In, clk;
  output  reg   Q1, Q2, Out;

  always @(posedge clk )
  begin

    Q1  <= In;
    Q2  <= Q1;
    Out <= Q2;

  end

endmodule
```

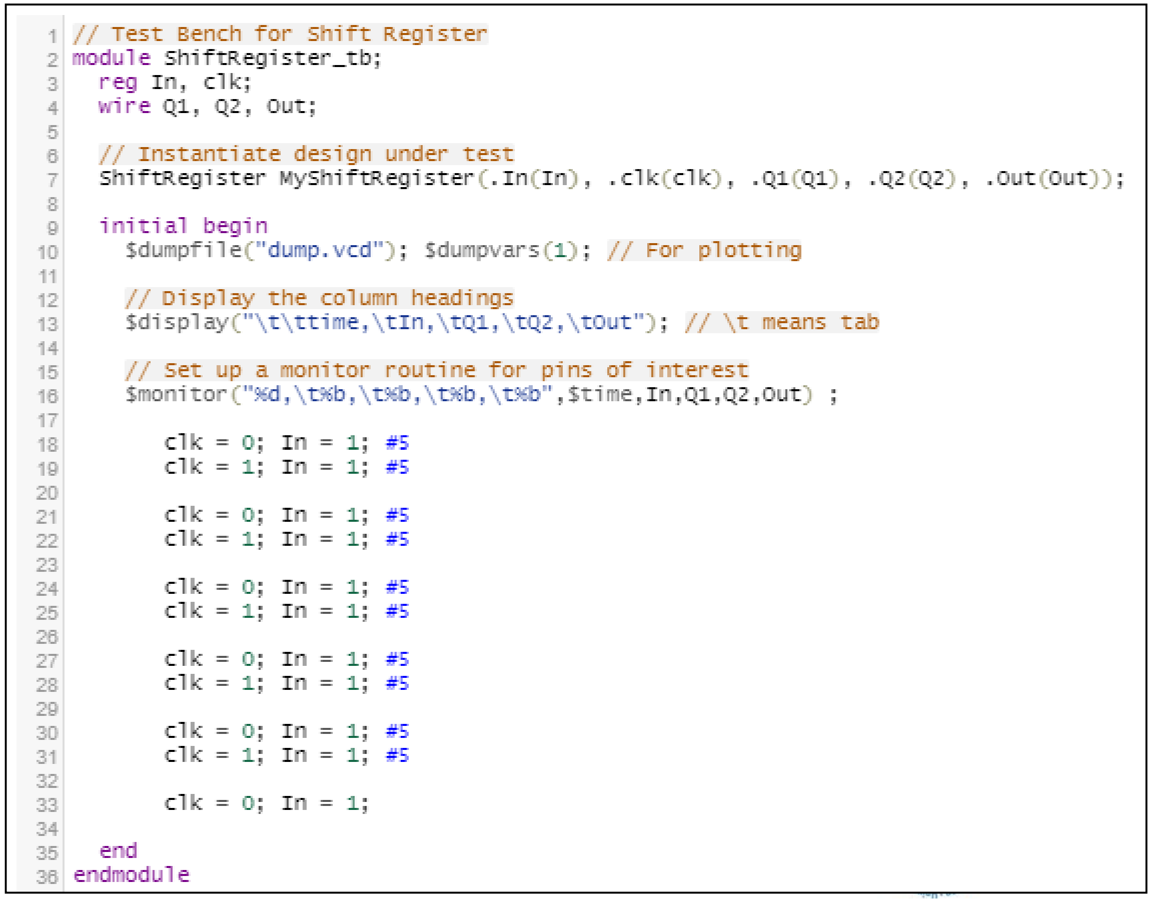
FPGAs and Verilog

Testing a Verilog design: shift register

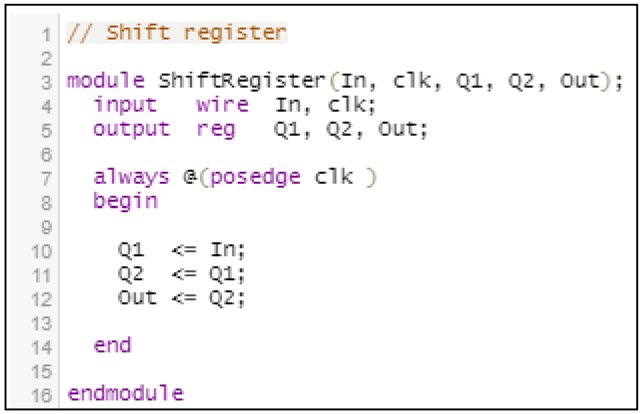
- Steps: follow the same steps as outlined previously



Circuit diagram of a 3-bit shift register



```
// Test Bench for Shift Register
module ShiftRegister_tb;
  reg In, clk;
  wire Q1, Q2, Out;

  // Instantiate design under test
  ShiftRegister MyShiftRegister(.In(In), .clk(clk), .Q1(Q1), .Q2(Q2), .Out(Out));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\tIn,\tQ1,\tQ2,\tOut"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b,\t%b,\t%b",$time,In,Q1,Q2,Out) ;

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1; #5
       clk = 1; In = 1; #5

       clk = 0; In = 1;

  end
endmodule
```



```
// Shift register

module ShiftRegister(In, clk, Q1, Q2, Out);
  input   wire  In, clk;
  output  reg   Q1, Q2, Out;

  always @(posedge clk )
  begin

    Q1  <= In;
    Q2  <= Q1;
    Out <= Q2;

  end

endmodule
```

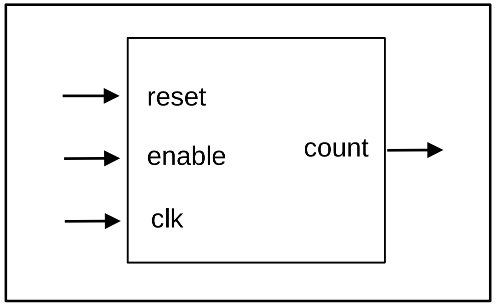
```
time,   In,    Q1,    Q2,    Out
   0,   1,     x,     x,     x
   5,   1,     1,     x,     x
  15,   1,     1,     1,     x
  25,   1,     1,     1,     1
```

After running code
∴ ShiftRegister was accurately coded in Verilog

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1

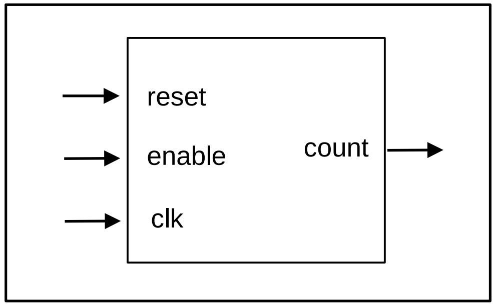


4-bit counter

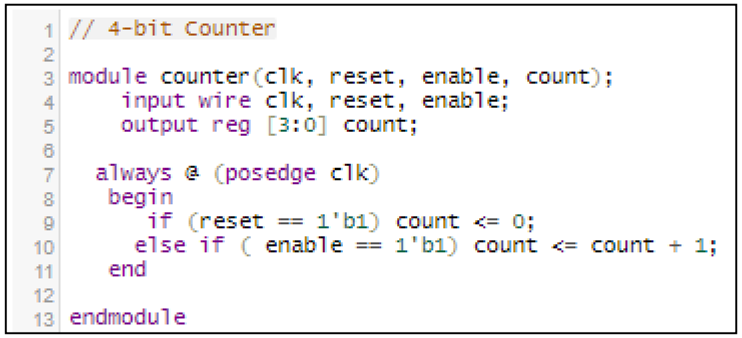
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter

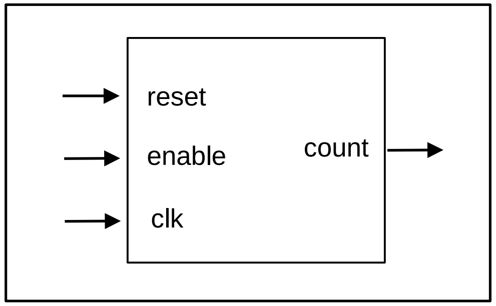


```
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
      if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

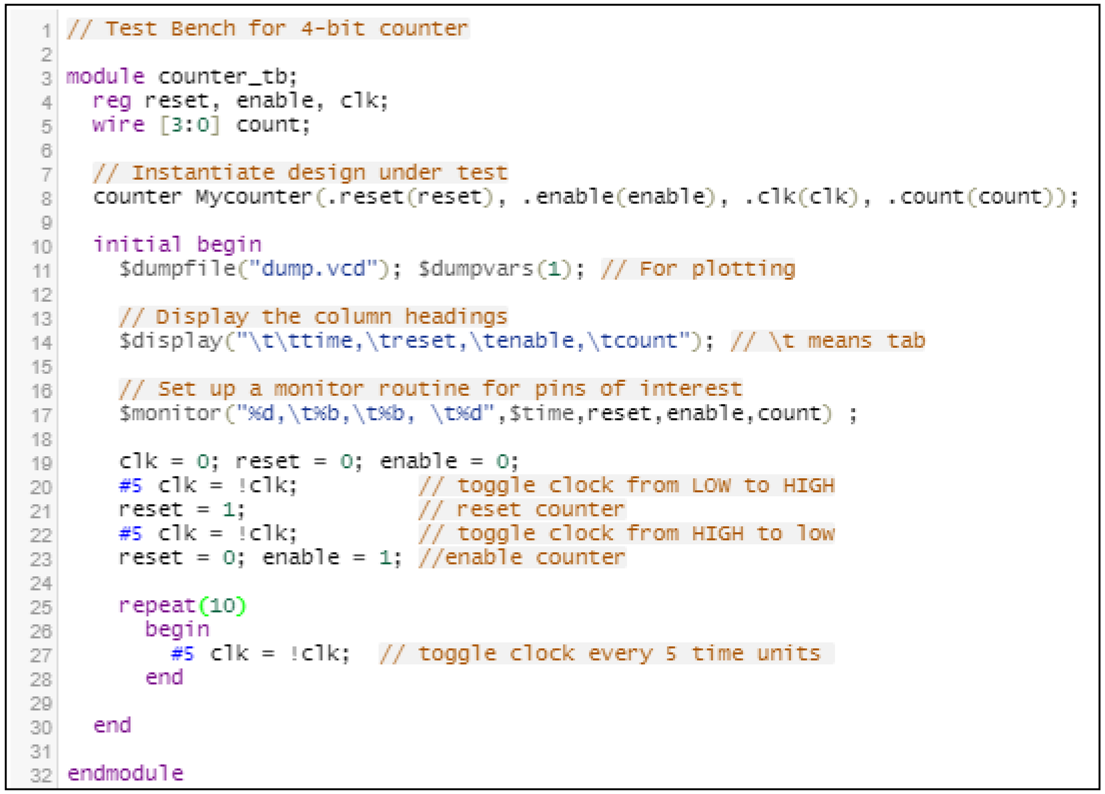
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

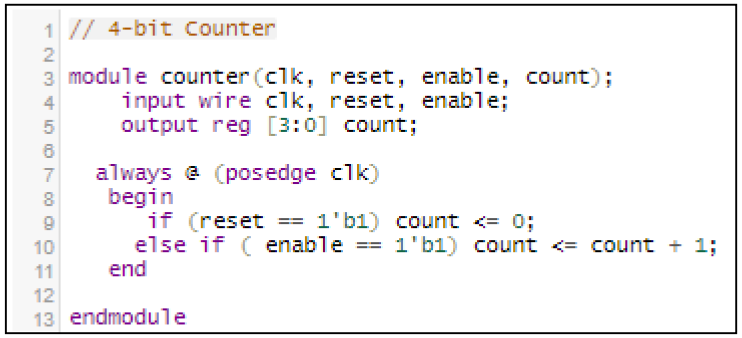
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter



```verilog
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;         // toggle clock from LOW to HIGH
    reset = 1;             // reset counter
    #5 clk = !clk;         // toggle clock from HIGH to low
    reset = 0; enable = 1; //enable counter

    repeat(10)
      begin
        #5 clk = !clk;  // toggle clock every 5 time units
      end

  end

endmodule
```

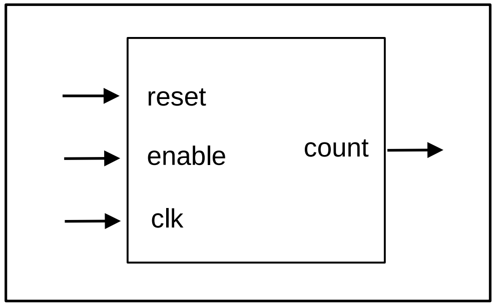


```verilog
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
      if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

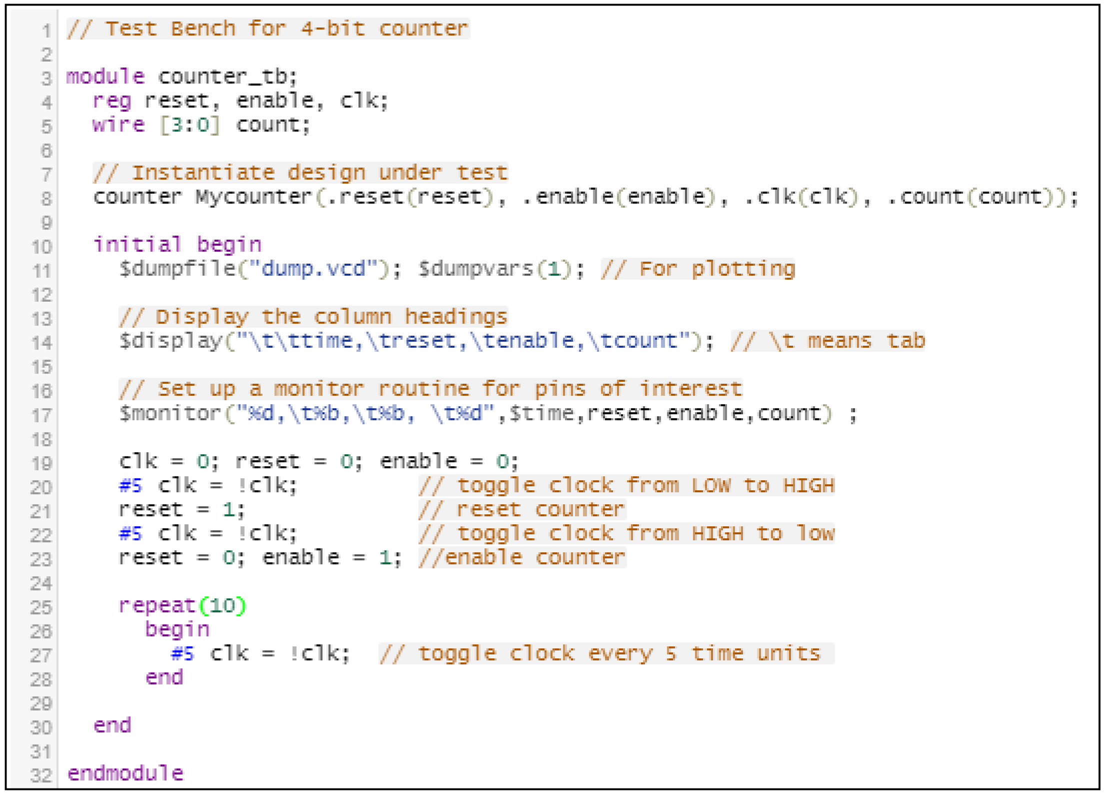
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

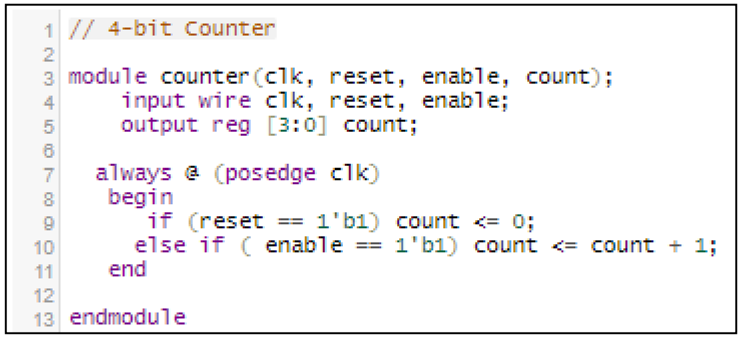
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



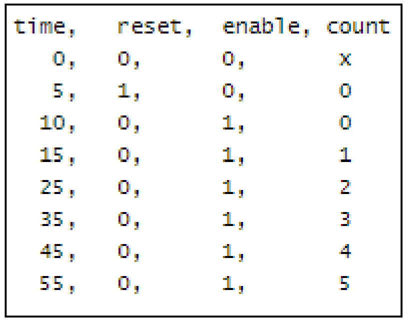
4-bit counter



```verilog
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;          // toggle clock from LOW to HIGH
    reset = 1;              // reset counter
    #5 clk = !clk;          // toggle clock from HIGH to low
    reset = 0; enable = 1;  //enable counter

    repeat(10)
      begin
        #5 clk = !clk;   // toggle clock every 5 time units
      end

  end

endmodule
```



```verilog
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
      if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

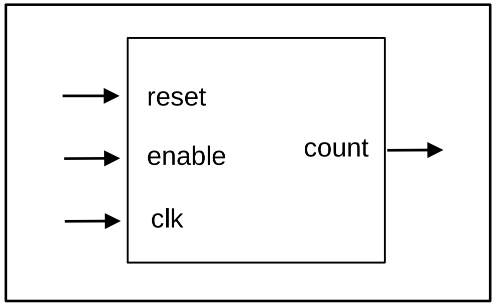


```
time,    reset,   enable,  count
   0,    0,       0,           x
   5,    1,       0,           0
  10,    0,       1,           0
  15,    0,       1,           1
  25,    0,       1,           2
  35,    0,       1,           3
  45,    0,       1,           4
  55,    0,       1,           5
```

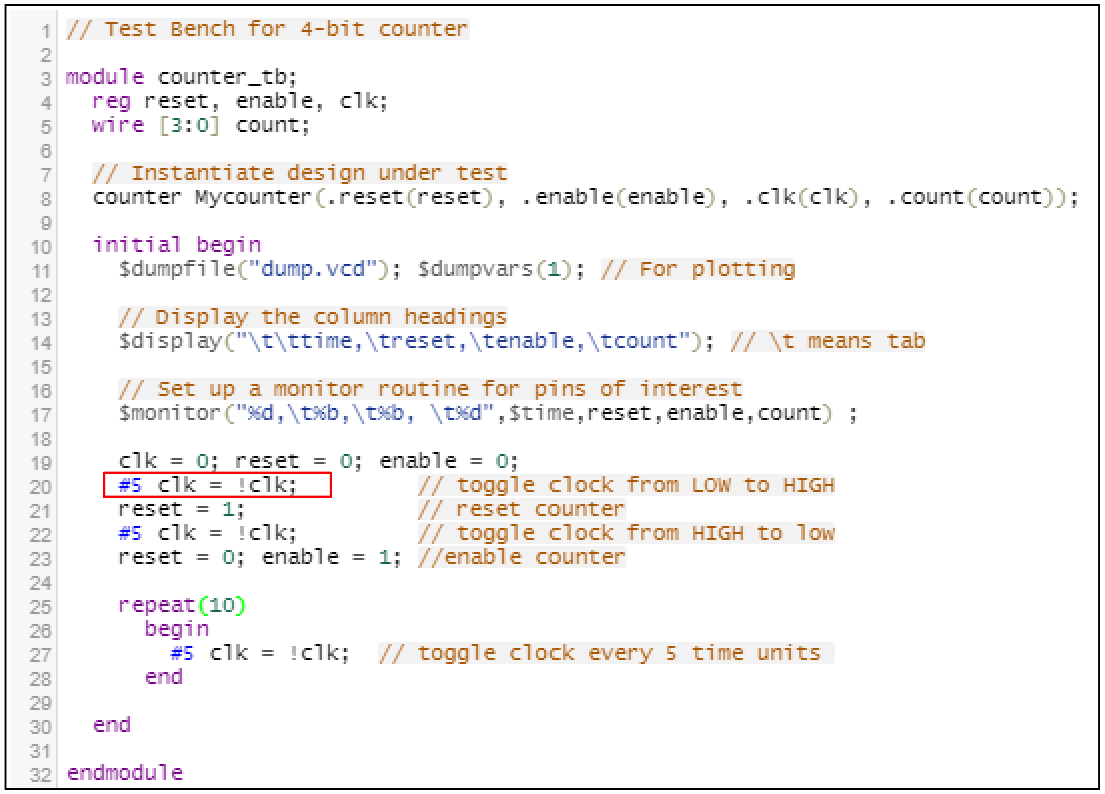
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

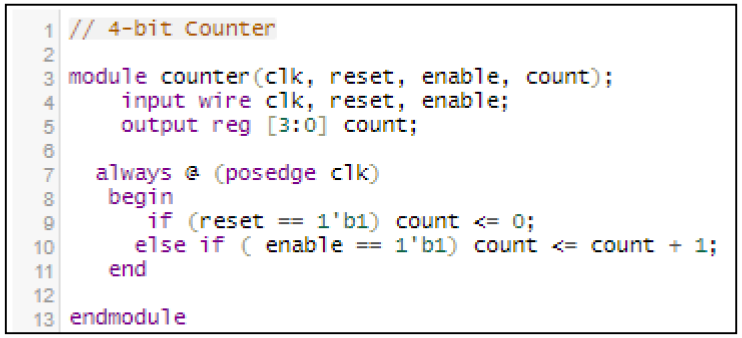
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



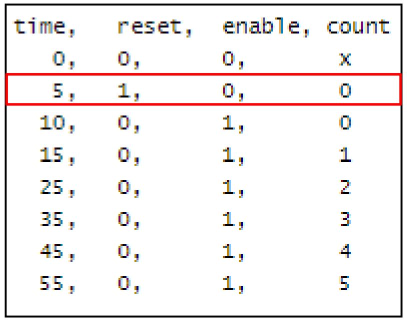
4-bit counter



```
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;         // toggle clock from LOW to HIGH
    reset = 1;             // reset counter
    #5 clk = !clk;         // toggle clock from HIGH to low
    reset = 0; enable = 1; //enable counter

    repeat(10)
      begin
        #5 clk = !clk;  // toggle clock every 5 time units
      end

  end

endmodule
```



```
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
     if (reset == 1'b1) count <= 0;
    else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

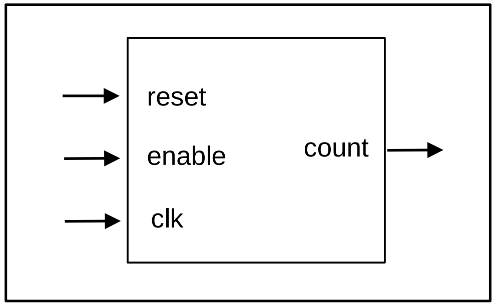


```
time,   reset,   enable,  count
   0,   0,       0,       x
   5,   1,       0,       0
  10,   0,       1,       0
  15,   0,       1,       1
  25,   0,       1,       2
  35,   0,       1,       3
  45,   0,       1,       4
  55,   0,       1,       5
```

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter

```
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;              // toggle clock from LOW to HIGH
    reset = 1;                  // reset counter
    #5 clk = !clk;              // toggle clock from HIGH to low
    reset = 0; enable = 1;      //enable counter

    repeat(10)
      begin
        #5 clk = !clk;   // toggle clock every 5 time units
      end

  end

endmodule
```

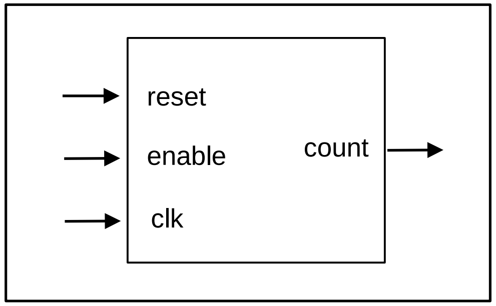
```
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
     if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

```
time,   reset,   enable,  count
   0,   0,       0,           x
   5,   1,       0,           0
  10,   0,       1,           0
  15,   0,       1,           1
  25,   0,       1,           2
  35,   0,       1,           3
  45,   0,       1,           4
  55,   0,       1,           5
```

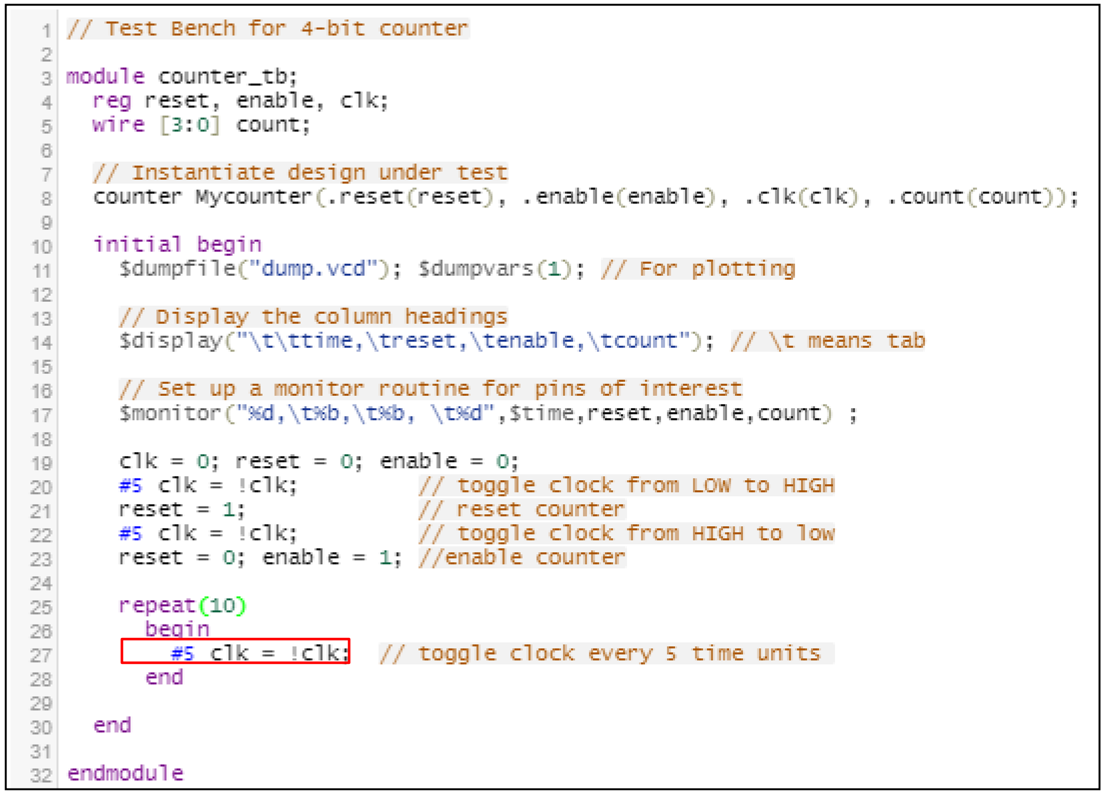
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

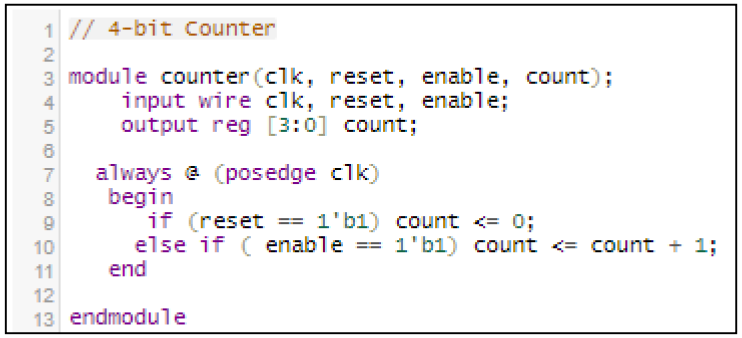
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter

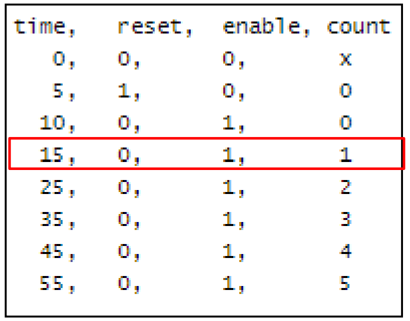


```
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;          // toggle clock from LOW to HIGH
    reset = 1;              // reset counter
    #5 clk = !clk;          // toggle clock from HIGH to low
    reset = 0; enable = 1; //enable counter

    repeat(10)
     begin
       #5 clk = !clk;   // toggle clock every 5 time units
     end

  end

endmodule
```



```
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
      if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

repeat = 1

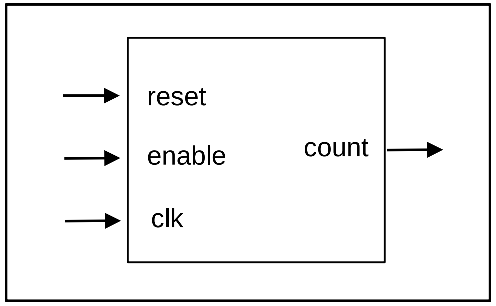


```
time,   reset,  enable, count
   0,   0,      0,         x
   5,   1,      0,         0
  10,   0,      1,         0
  15,   0,      1,         1
  25,   0,      1,         2
  35,   0,      1,         3
  45,   0,      1,         4
  55,   0,      1,         5
```

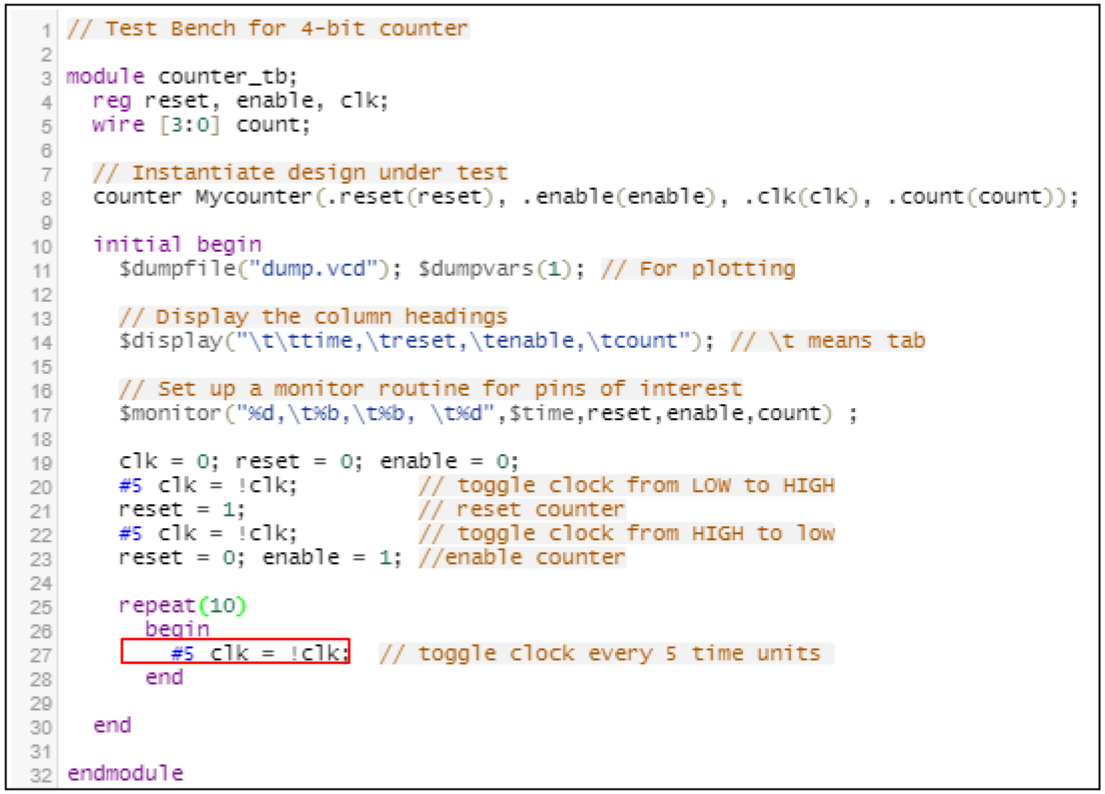
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

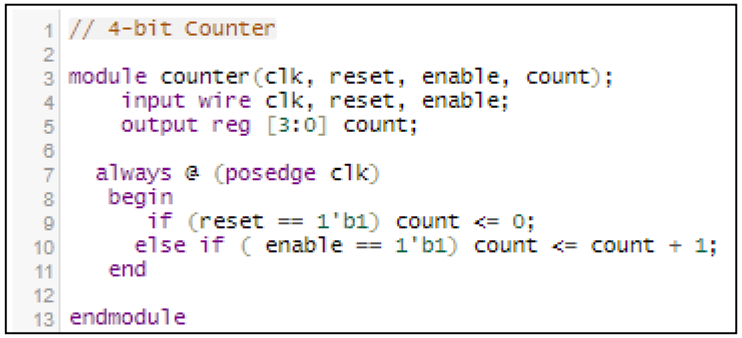
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter

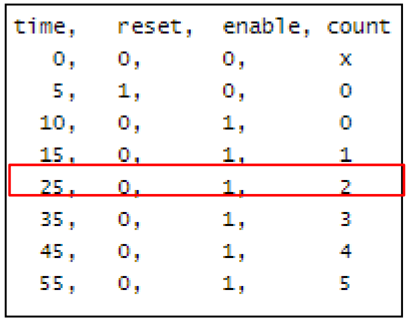


```verilog
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;          // toggle clock from LOW to HIGH
    reset = 1;              // reset counter
    #5 clk = !clk;          // toggle clock from HIGH to low
    reset = 0; enable = 1;  //enable counter

    repeat(10)
      begin
        #5 clk = !clk;   // toggle clock every 5 time units
      end

  end

endmodule
```



```verilog
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
     if (reset == 1'b1) count <= 0;
    else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

repeat = 3

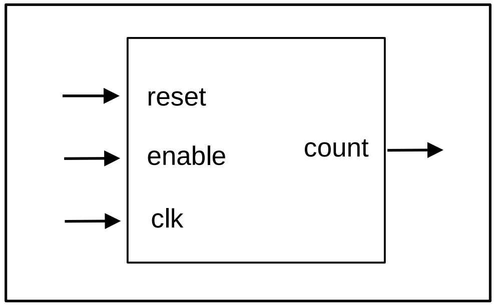


```
time,   reset,   enable,  count
   0,   0,       0,           x
   5,   1,       0,           0
  10,   0,       1,           0
  15,   0,       1,           1
  25,   0,       1,           2
  35,   0,       1,           3
  45,   0,       1,           4
  55,   0,       1,           5
```

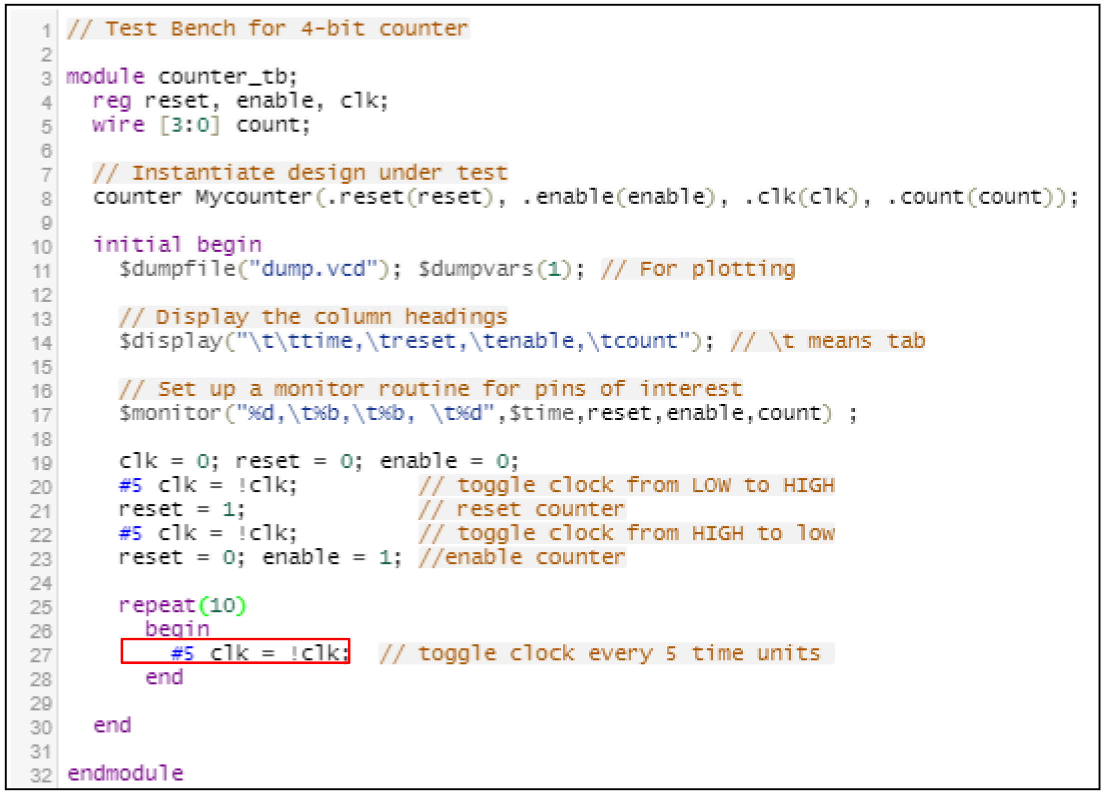
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

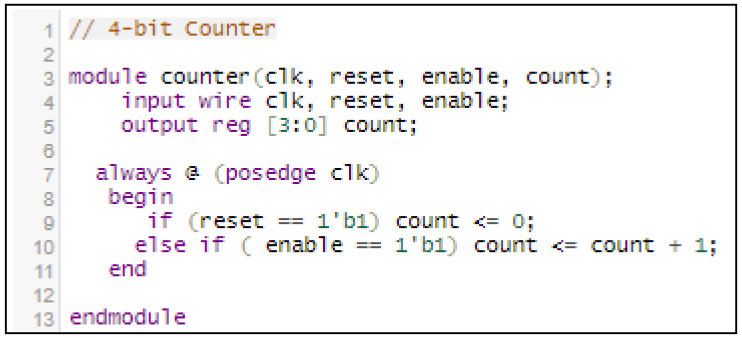
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter

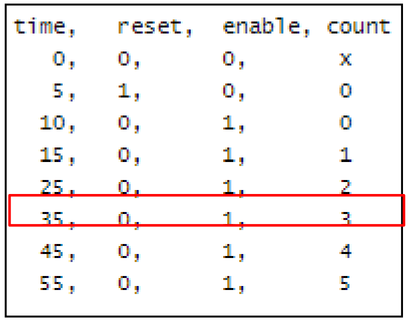


```verilog
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;          // toggle clock from LOW to HIGH
    reset = 1;              // reset counter
    #5 clk = !clk;          // toggle clock from HIGH to low
    reset = 0; enable = 1;  //enable counter

    repeat(10)
      begin
        #5 clk = !clk;   // toggle clock every 5 time units
      end

  end

endmodule
```



```verilog
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
     if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

repeat = 5

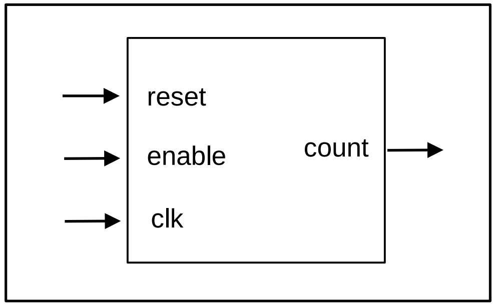


```
time,   reset,  enable, count
   0,   0,      0,        x
   5,   1,      0,        0
  10,   0,      1,        0
  15,   0,      1,        1
  25,   0,      1,        2
  35,   0,      1,        3
  45,   0,      1,        4
  55,   0,      1,        5
```

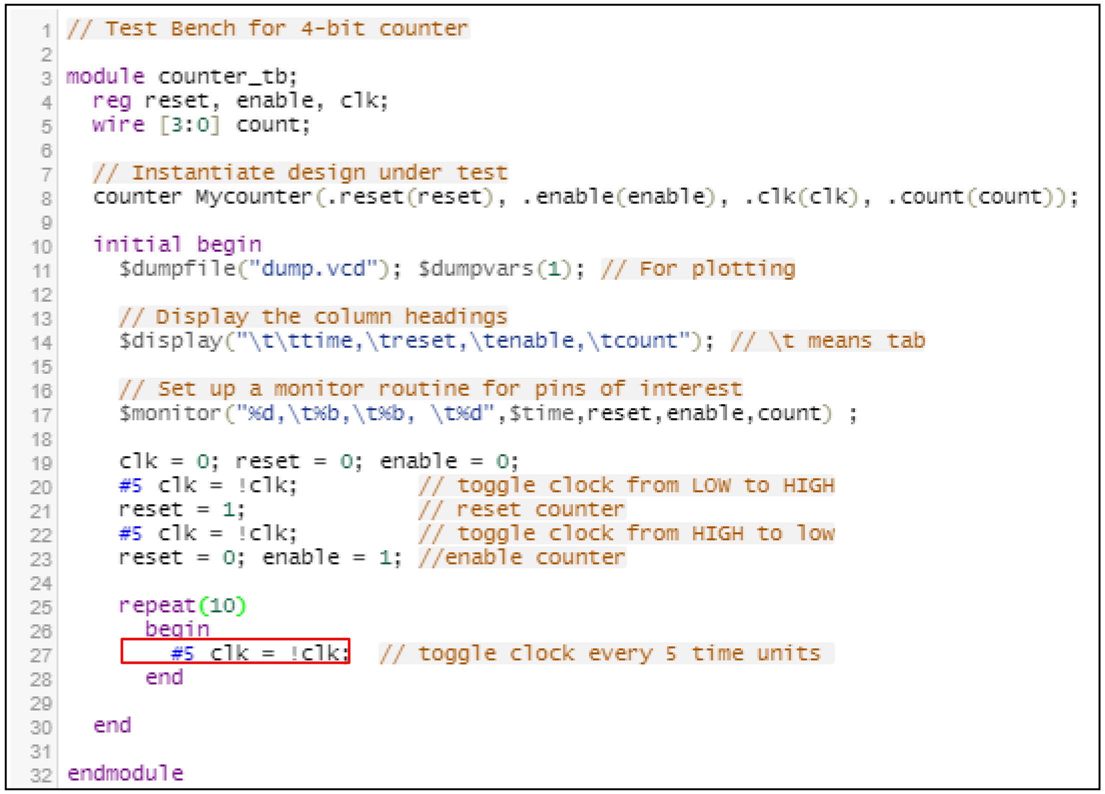
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

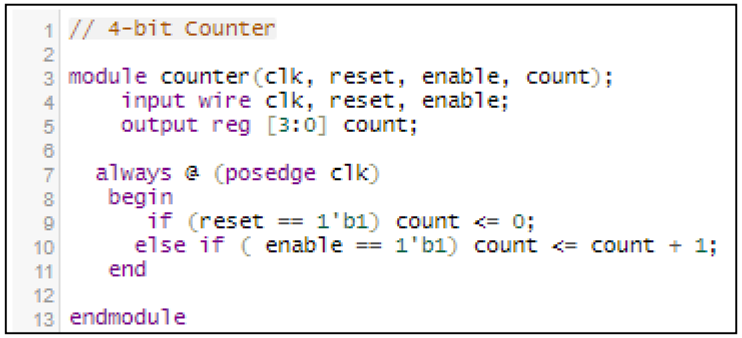
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter

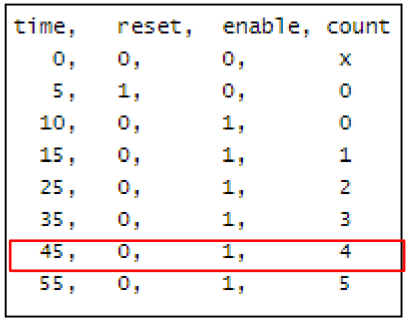


```
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;          // toggle clock from LOW to HIGH
    reset = 1;              // reset counter
    #5 clk = !clk;          // toggle clock from HIGH to low
    reset = 0; enable = 1;  //enable counter

    repeat(10)
      begin
        #5 clk = !clk;   // toggle clock every 5 time units
      end

  end

endmodule
```



```
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
     if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

repeat = 7

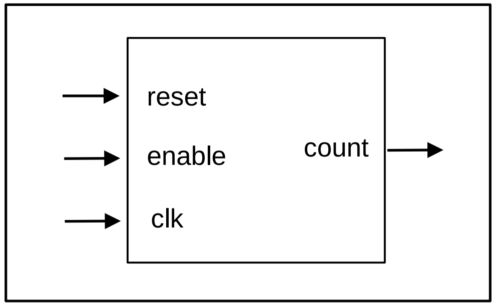


```
time,   reset,  enable, count
   0,   0,      0,         x
   5,   1,      0,         0
  10,   0,      1,         0
  15,   0,      1,         1
  25,   0,      1,         2
  35,   0,      1,         3
  45,   0,      1,         4
  55,   0,      1,         5
```

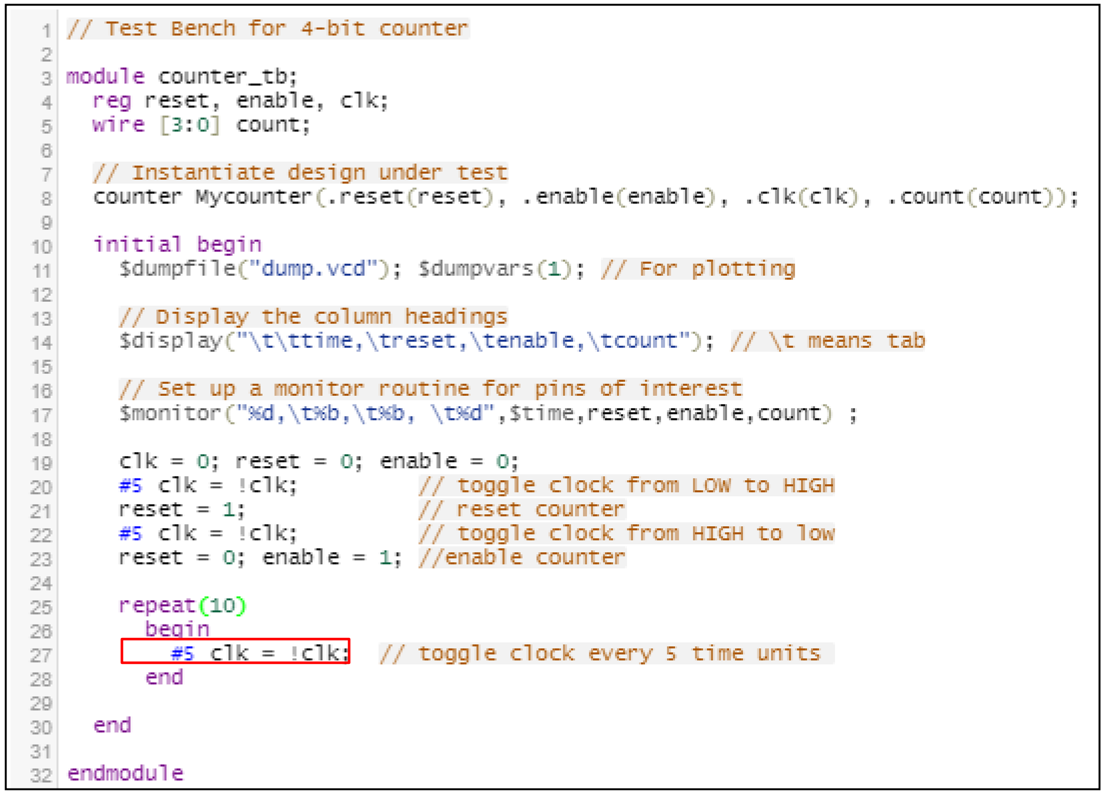
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

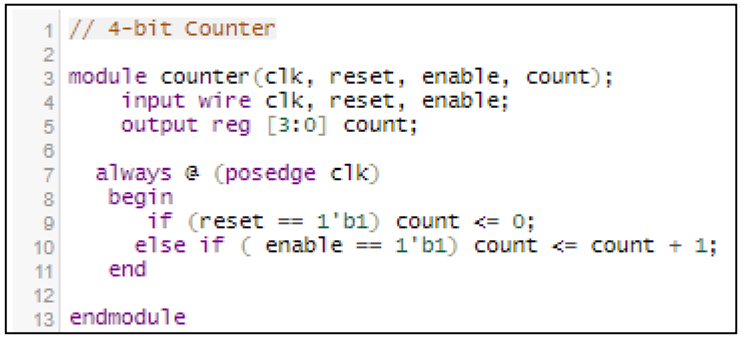
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter

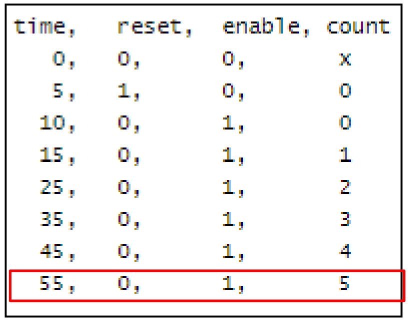


```verilog
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;          // toggle clock from LOW to HIGH
    reset = 1;              // reset counter
    #5 clk = !clk;          // toggle clock from HIGH to low
    reset = 0; enable = 1;  //enable counter

    repeat(10)
      begin
        #5 clk = !clk;    // toggle clock every 5 time units
      end

  end

endmodule
```



```verilog
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
     if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

repeat = 9

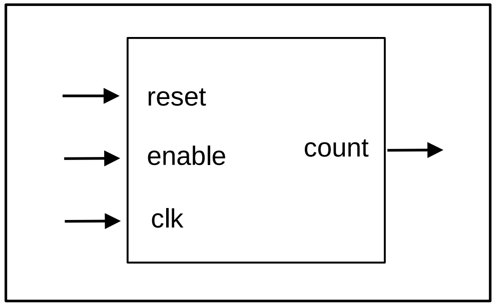


```
time,   reset,  enable,  count
   0,   0,      0,         x
   5,   1,      0,         0
  10,   0,      1,         0
  15,   0,      1,         1
  25,   0,      1,         2
  35,   0,      1,         3
  45,   0,      1,         4
  55,   0,      1,         5
```

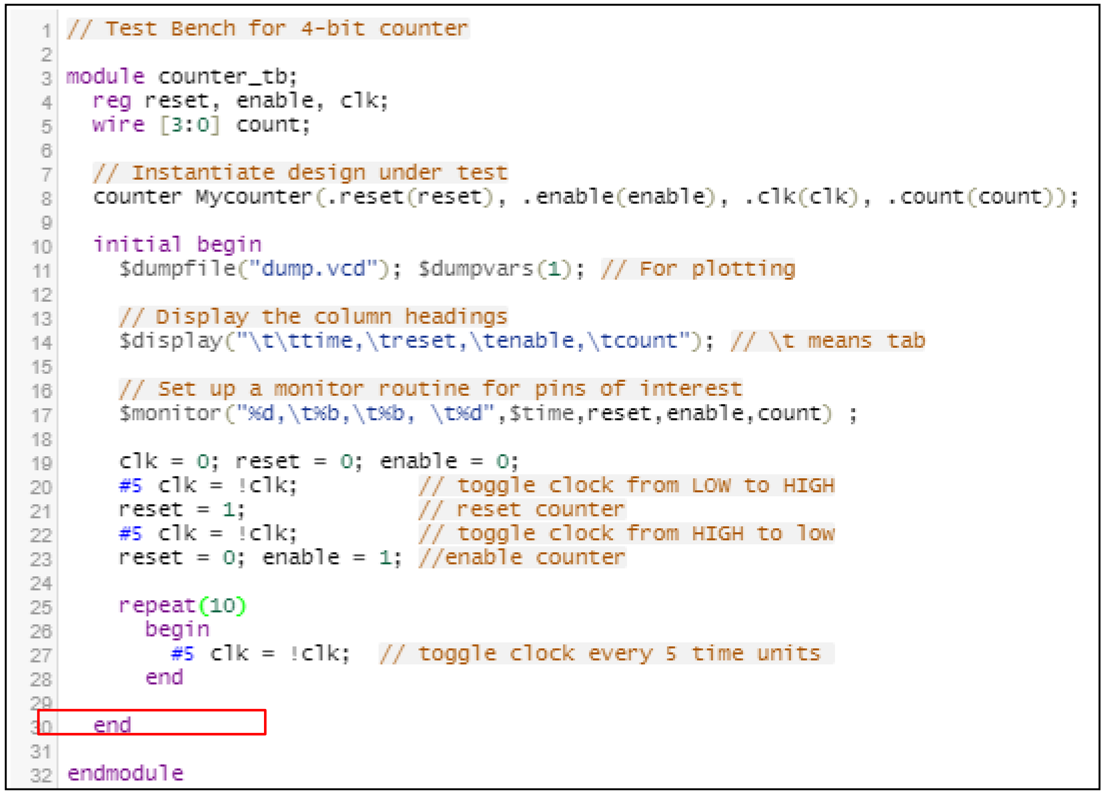
FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

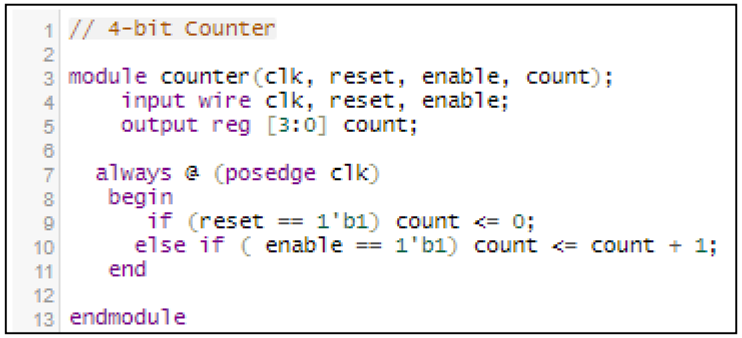
- Functionality of the counter:
  - When reset line = 1: count = 0
  - When reset line = 0, enable = 1: count = count + 1



4-bit counter

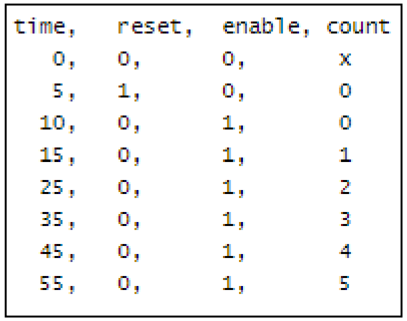


```verilog
// Test Bench for 4-bit counter

module counter_tb;
  reg reset, enable, clk;
  wire [3:0] count;

  // Instantiate design under test
  counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));

  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting

    // Display the column headings
    $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab

    // Set up a monitor routine for pins of interest
    $monitor("%d,\t%b,\t%b, \t%d",$time,reset,enable,count) ;

    clk = 0; reset = 0; enable = 0;
    #5 clk = !clk;          // toggle clock from LOW to HIGH
    reset = 1;              // reset counter
    #5 clk = !clk;          // toggle clock from HIGH to low
    reset = 0; enable = 1;  //enable counter

    repeat(10)
      begin
        #5 clk = !clk;   // toggle clock every 5 time units
      end

  end

endmodule
```



```verilog
// 4-bit Counter

module counter(clk, reset, enable, count);
    input wire clk, reset, enable;
    output reg [3:0] count;

  always @ (posedge clk)
   begin
     if (reset == 1'b1) count <= 0;
     else if ( enable == 1'b1) count <= count + 1;
   end

endmodule
```

repeat = 10



```
time,   reset,   enable,  count
   0,   0,       0,         x
   5,   1,       0,         0
  10,   0,       1,         0
  15,   0,       1,         1
  25,   0,       1,         2
  35,   0,       1,         3
  45,   0,       1,         4
  55,   0,       1,         5
```

FPGAs and Verilog

Testing a Verilog design: summary of commands

- Constructs that apply to test benches only:
  - #n: delay for n time units in simulation.
  - initial: similar to an always block. Does not have a sensitivity list. Only runs once.
  - $display: syntax and operation is similar to printf. Runs only once.
  - $monitor: monitors changes in signals and displays values when they change
  - repeat(n): repeats a block, *n* times