

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

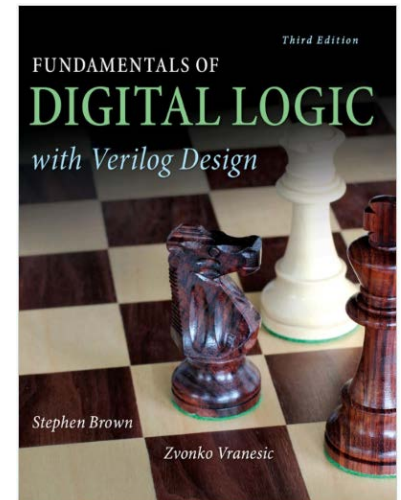
Welcome to the world of ASIC

<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

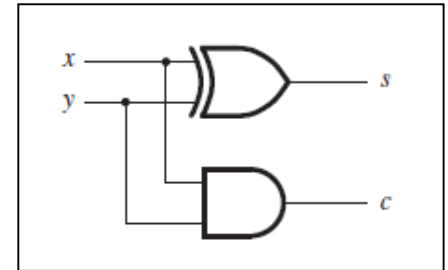
<https://www.edaplayground.com/>



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

		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

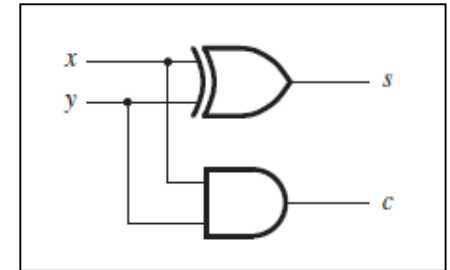
```
1 // Function: half adder
2
3 module HalfAdder(x, y, s, c);
4
5     input wire x, y;
6     output wire s, c;
7
8     assign s = x ^ y; /* Sum = x XOR y */
9     assign c = x & y; /* Carry = x AND y */
10
11 endmodule
```

Verilog code for a half adder

FPGAs and Verilog

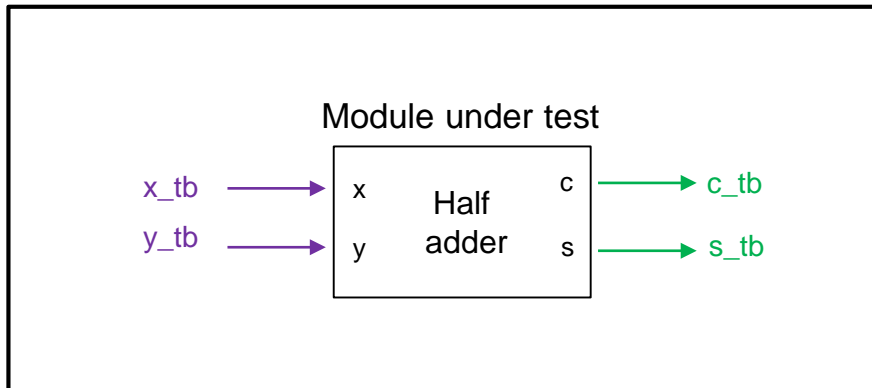
Testing a Verilog design

- 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



		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

```

1 // Function: half adder
2
3 module HalfAdder(x, y, s, c);
4
5     input wire x, y;
6     output wire s, c;
7
8     assign s = x ^ y; /* Sum = x XOR y */
9     assign c = x & y; /* Carry = x AND y */
10
11 endmodule

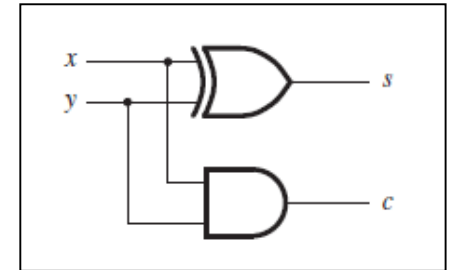
```

Verilog code for a half adder

FPGAs and Verilog

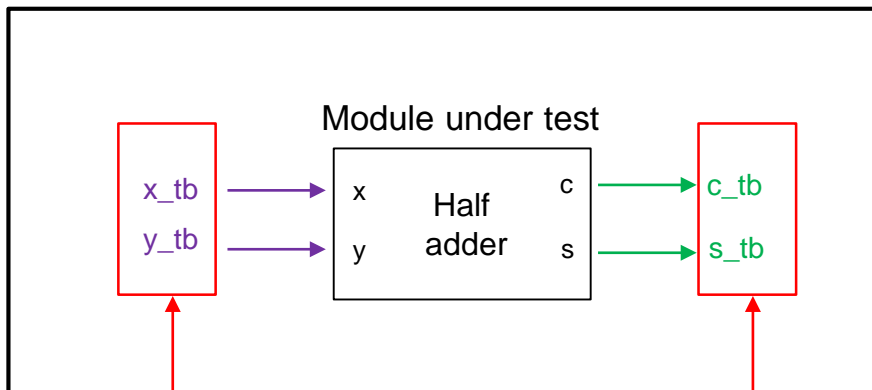
Testing a Verilog design

- 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



What should the input values be?

What are the expected output values?

		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

```

1 // Function: half adder
2
3 module HalfAdder(x, y, s, c);
4
5     input wire x, y;
6     output wire s, c;
7
8     assign s = x ^ y; /* Sum = x XOR y */
9     assign c = x & y; /* Carry = x AND y */
10
11 endmodule

```

Verilog code for a half adder

FPGAs and Verilog

Testing a Verilog design

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

The screenshot displays the EDA Playground web interface. At the top, a blue header bar contains the 'EDA playground' logo, 'Run' and 'Save' buttons, a maintenance notice, and navigation links for 'Playgrounds' and 'Profile'. On the left, a sidebar lists categories like 'Languages & Libraries', 'Tools & Simulators', and 'Examples', along with a 'Community' section featuring 'Collaborate', 'Forum', and 'Follow @edaplayground' buttons. The main workspace is divided into two side-by-side code editors. The left editor, titled 'testbench.sv', contains placeholder text: '1 // Code your testbench here', '2 // or browse Examples', and '3'. The right editor, titled 'design.sv', contains placeholder text: '1 // Code your design here' and '2'. Both editors have a tab icon in the top right corner. Below the editors, a footer bar includes 'Log' and 'Share' buttons, a title input field with the placeholder 'Add a title to help you find your playground', a visibility dropdown set to 'Public (anyone with the link can view)', and a 'Save' button.

FPGAs and Verilog

Testing a Verilog design

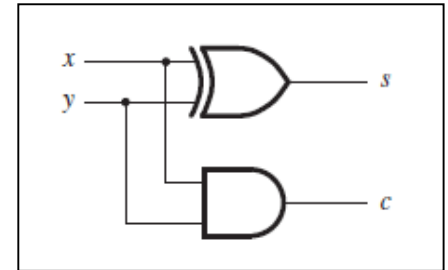
- EDA playground can be used to write Verilog code for both the digital circuit and the test bench: <https://www.edaplayground.com/>

The screenshot displays the EDA Playground web interface. At the top, a blue header bar contains the 'EDA playground' logo, a 'Run' button, a 'Save' button, and a status message: 'Routine maintenance: possible disruption to license server Start: 2019-10-09 04:15:00 (UTC) End: 2019-10-09 04:15:00 (UTC)'. To the right of the header are icons for help, a flask, 'Playgrounds', and a 'Profile' dropdown. On the left side, a sidebar lists navigation options: 'Languages & Libraries', 'Tools & Simulators', 'Examples', and 'Community' (with sub-links for 'Collaborate', 'Forum', and 'Follow @edaplayground'). The main area features two side-by-side code editors. The left editor, titled 'testbench.sv', is outlined in green and contains the text 'Verilog code for the test bench'. The right editor, titled 'design.sv', is outlined in purple and contains the text 'Verilog code for digital circuit'. Both editors have a line number indicator on the left and a tab label on the right. At the bottom, there is a 'Log' button, a 'Share' button, a text input field for a title, a dropdown menu for visibility settings (currently set to 'Public (anyone with the link can view)'), and a 'Save' button.

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	Sum
		c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

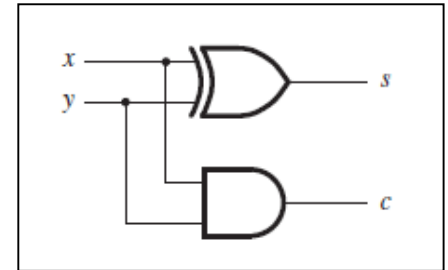
Truth table for a half adder

```
1 // Function: half adder
2
3 module HalfAdder(x, y, s, c);
4
5     input wire x, y;
6     output wire s, c;
7
8     assign s = x ^ y; /* Sum = x XOR y */
9     assign c = x & y; /* Carry = x AND y */
10
11 endmodule
```

FPGAs and Verilog

Testing a Verilog design: steps

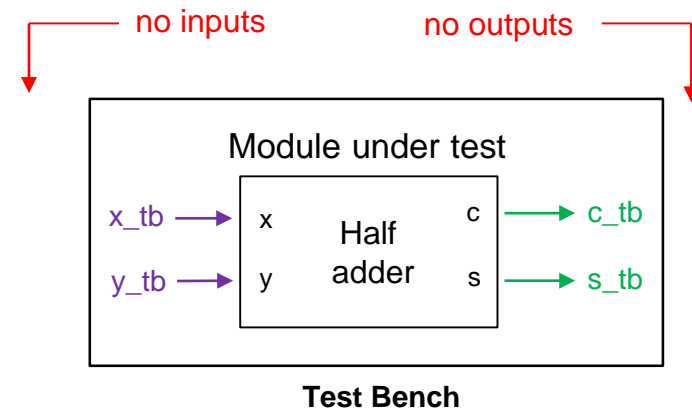
- **Step 1:** describe the expected operation of the digital circuit
- **Step 2:** write Verilog code for the test bench. Note that a test bench does not have any ports, ie. no inputs, no outputs



Circuit diagram of a half adder

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

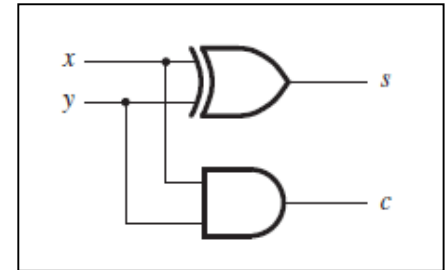
Truth table for a half adder



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	Sum
		c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

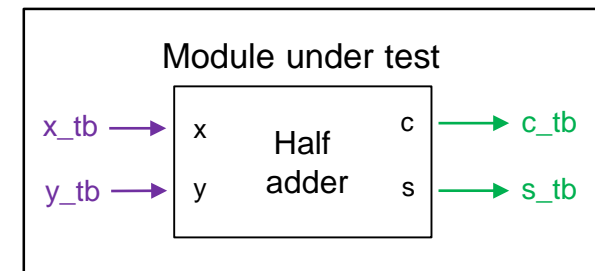
Truth table for a half adder

```

1 // Test Bench for half adder
2 module HalfAdder_tb;
3     reg x_tb, y_tb;
4     wire s_tb, c_tb;
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 endmodule
    
```

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

outputs for MUT: should be data type 'wire', since these are signals we want to observe

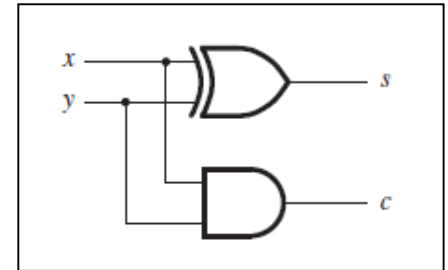


Test Bench

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

```

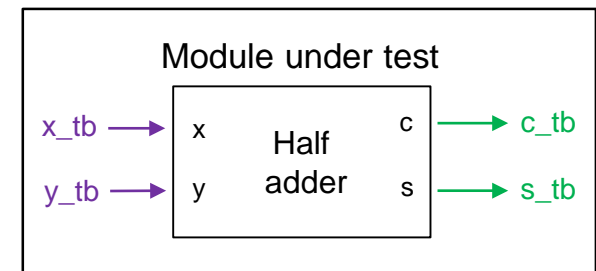
1 // Test Bench for half adder
2 module HalfAdder_tb;
3   reg x_tb, y_tb;
4   wire s_tb, c_tb;
5
6   // Instantiate design under test
7   HalfAdder MyHalfAdder (.x(x_tb), .y(y_tb), .s(s_tb), .c(c_tb));
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
    
```

format: .signal_MUT(signal_TestBench)

MyHalfAdder is an instant
of the module HalfAdder

```

1 // Function: half adder
2
3 module HalfAdder(x, y, s, c);
4
5   input wire x, y;
6   output wire s, c;
7
8   assign s = x ^ y; /* Sum = x XOR y */
9   assign c = x & y; /* Carry = x AND b */
10
11 endmodule
    
```

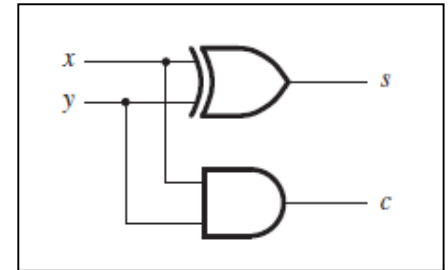


Test Bench

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	Sum
		c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

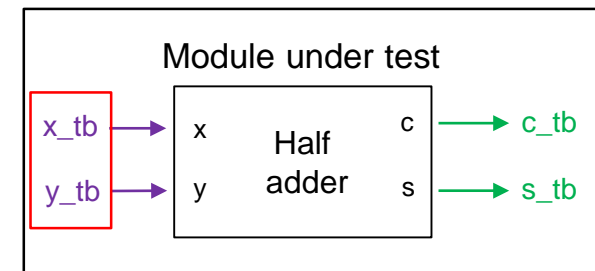
Truth table for a half adder

```

1 // Test Bench for half adder
2 module HalfAdder_tb;
3     reg x_tb, y_tb;
4     wire s_tb, c_tb;
5
6     // Instantiate design under test
7     HalfAdder MyHalfAdder(.x(x_tb), .y(y_tb), .s(s_tb), .c(c_tb));
8
9     initial begin
10
11
12
13
14
15
16
17
18     x_tb = 0; y_tb = 0; #5;
19     x_tb = 0; y_tb = 1; #5;
20     x_tb = 1; y_tb = 0; #5;
21     x_tb = 1; y_tb = 1; #5;
22 end
23 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

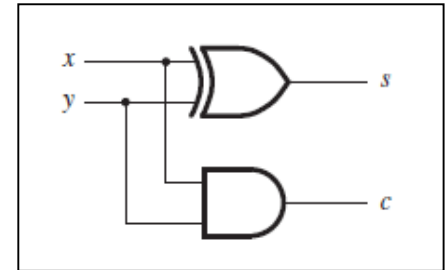


Test Bench

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	Sum
		c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

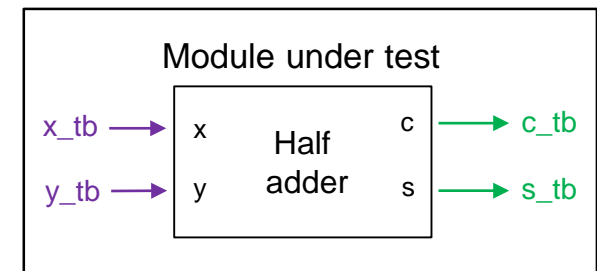
Truth table for a half adder

```

1 // Test Bench for half adder
2 module HalfAdder_tb;
3     reg x_tb, y_tb;
4     wire s_tb, c_tb;
5
6     // Instantiate design under test
7     HalfAdder MyHalfAdder(.x(x_tb), .y(y_tb), .s(s_tb), .c(c_tb));
8
9     initial begin
10         // Display the column headings
11         $display("\t\ttime,\tx_tb,\ty_tb,\ts_tb,\tc_tb"); // \t means tab
12
13         // Set up a monitor routine for pins of interest
14         $monitor("%d,\t\b,\t\b,\t\b,\t\b", $time, x_tb, y_tb, s_tb, c_tb);
15
16         x_tb = 0; y_tb = 0; #5;
17         x_tb = 0; y_tb = 1; #5;
18         x_tb = 1; y_tb = 0; #5;
19         x_tb = 1; y_tb = 1; #5;
20     end
21 endmodule
22
23 
```

\$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.

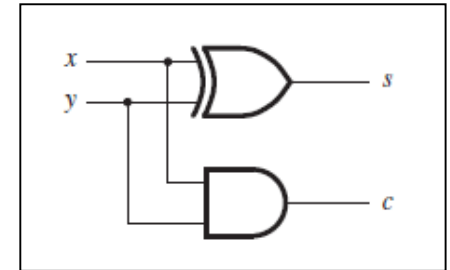


Test Bench

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	Sum
		c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder

```

1 // Test Bench for half adder
2 module HalfAdder_tb;
3     reg x_tb, y_tb;
4     wire s_tb, c_tb;
5
6     // Instantiate design under test
7     HalfAdder MyHalfAdder(.x(x_tb), .y(y_tb), .s(s_tb), .c(c_tb));
8
9     initial begin
10
11         // Display the column headings
12         $display("\t\ttime,\tx_tb,\ty_tb,\ts_tb,\tc_tb"); // \t means tab
13
14         // Set up a monitor routine for pins of interest
15         $monitor("%d,\tb,\tb,\tb,\tb", $time, x_tb, y_tb, s_tb, c_tb);
16
17         x_tb = 0; y_tb = 0; #5;
18         x_tb = 0; y_tb = 1; #5;
19         x_tb = 1; y_tb = 0; #5;
20         x_tb = 1; y_tb = 1; #5;
21     end
22 endmodule
23
    
```

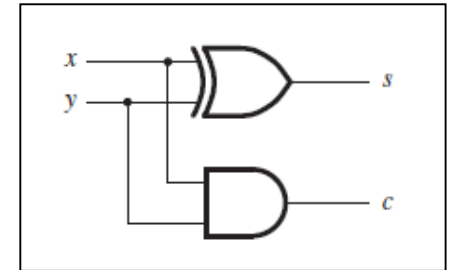
time,	x_tb,	y_tb,	s_tb,	c_tb
0,	0,	0,	0,	0
5,	0,	1,	1,	0
10,	1,	0,	1,	0
15,	1,	1,	0,	1

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	Sum
		c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table for a half adder

Equivalent
 \therefore Verilog code for HalfAdder was accurate

```

1 // Test Bench for half adder
2 module HalfAdder_tb;
3     reg x_tb, y_tb;
4     wire s_tb, c_tb;
5
6     // Instantiate design under test
7     HalfAdder MyHalfAdder(.x(x_tb),
8
9     initial begin
10
11
12     // Display the column headings
13     $display("\t\ttime,\tx_tb,\ty_tb,\ts_tb,\tc_tb"); // \t means tab
14
15     // Set up a monitor routine for pins of interest
16     $monitor("%d,\t%b,\t%b,\t%b,\t%d", $time, x_tb, y_tb, s_tb, c_tb);
17
18     x_tb = 0; y_tb = 0; #5;
19     x_tb = 0; y_tb = 1; #5;
20     x_tb = 1; y_tb = 0; #5;
21     x_tb = 1; y_tb = 1; #5;
22     end
23 endmodule
    
```

```

1 // Function: half adder
2
3 module HalfAdder(x, y, s, c);
4
5     input wire x, y;
6     output wire s, c;
7
8     assign s = x ^ y; /* Sum = x XOR y */
9     assign c = x & y; /* Carry = x AND y */
10
11 endmodule
    
```

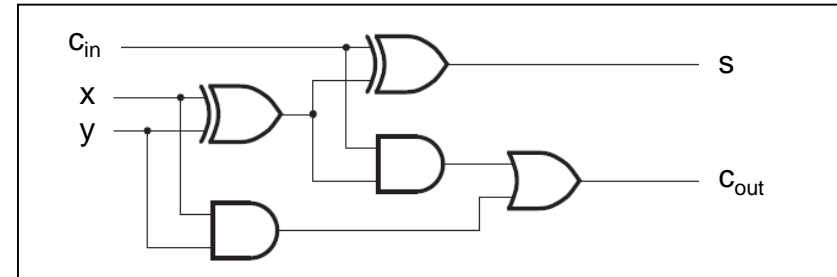
time,	x_tb,	y_tb,	s_tb,	c_tb
0,	0,	0,	0,	0
5,	0,	1,	1,	0
10,	1,	0,	1,	0
15,	1,	1,	0,	1

Display to the user after Verilog test bench code executes

FPGAs and Verilog

Testing a Verilog design: full adder

- Write the Verilog code for the test bench



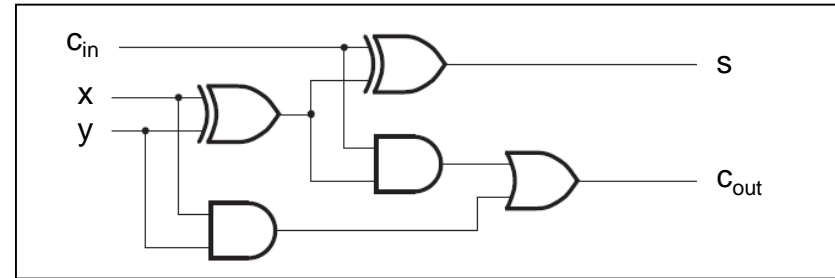
Full adder

```
1 // Full adder
2
3 module FullAdder(x, y, cin, s, cout);
4   input wire x, y, cin;
5   output wire s, cout;
6
7   assign s = x ^ y ^ cin;
8   assign cout = ((x ^ y) & cin) | (x & y);
9
10
11 endmodule
```

FPGAs and Verilog

Testing a Verilog design: full adder

- Write the Verilog code for the test bench



Full adder

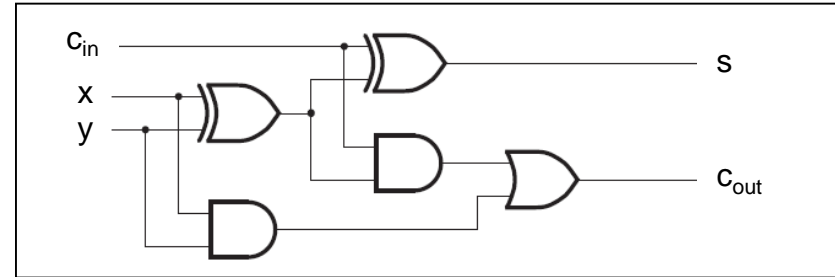
```
1 // Test bench for full adder
2 module FullAdder_tb;
3     reg x, y, cin;
4     wire s, cout;
5
6     // Instantiate design under test
7     FullAdder MyFullAdder(.x(x), .y(y), .cin(cin), .s(s), .cout(cout));
8
9     initial begin
10
11         // Display the column headings
12         $display("\t\ttime,\tcin,\tx,\ty,\ts,\tcout"); // \t means tab
13
14         // Set up a monitor routine for pins of interest
15         $monitor("%d,\t\b,\t\b,\t\b,\t\b,\t\b", $time, cin, x, y, s, cout);
16
17         cin = 0; x = 0; y = 0; #5;
18         cin = 0; x = 0; y = 1; #5;
19         cin = 0; x = 1; y = 0; #5;
20         cin = 0; x = 1; y = 1; #5;
21         cin = 1; x = 0; y = 0; #5;
22         cin = 1; x = 0; y = 1; #5;
23         cin = 1; x = 1; y = 0; #5;
24         cin = 1; x = 1; y = 1; #5;
25     end
26 endmodule
```

```
1 // Full adder
2
3 module FullAdder(x, y, cin, s, cout);
4     input wire x, y, cin;
5     output wire s, cout;
6
7     assign s = x ^ y ^ cin;
8     assign cout = ((x ^ y) & cin) | (x & y);
9
10
11 endmodule
```


FPGAs and Verilog

Testing a Verilog design: full adder

- Write the Verilog code for the test bench



Full adder

```

1 // Test bench for full adder
2 module FullAdder_tb;
3   reg x, y, cin;
4   wire s, cout;
5
6   // Instantiate design under test
7   FullAdder MyFullAdder(.x(x), .y(y), .cin(cin), .s(s), .cout(cout));
8
9   initial begin
10
11
12     // Display the column headings
13     $display("\t\ttime,\tcin,\tx,\ty,\ts,\tcout"); // \t means tab
14
15     // Set up a monitor routine for pins of interest
16     $monitor("%d,\t\b,\t\b,\t\b,\t\b,\t\b", $time, cin, x, y, s, cout);
17
18     cin = 0; x = 0; y = 0; #5;
19     cin = 0; x = 0; y = 1; #5;
20     cin = 0; x = 1; y = 0; #5;
21     cin = 0; x = 1; y = 1; #5;
22     cin = 1; x = 0; y = 0; #5;
23     cin = 1; x = 0; y = 1; #5;
24     cin = 1; x = 1; y = 0; #5;
25     cin = 1; x = 1; y = 1; #5;
26   end
27 endmodule
  
```

```

1 // Full adder
2
3 module FullAdder(x, y, cin, s, cout);
4   input wire x, y, cin;
5   output wire s, cout;
6
7   assign s = x ^ y ^ cin;
8   assign cout = ((x ^ y) & cin) | (x & y);
9
10
11 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

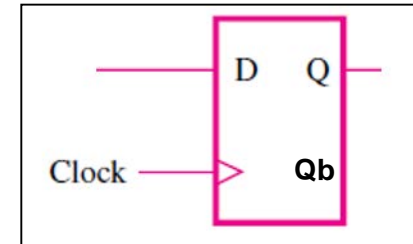
After running code
 \therefore FullAdder was accurately coded in Verilog

FPGAs and Verilog

Testing a Verilog design: D flip-flop

- **Steps:** follow the same steps as outlined previously

```
1 // Test Bench for D flip-flop
2 module DFlipflop_tb;
3     reg d, clk;
4     wire q, qb;
5
6     // Instantiate design under test
7     DFlipflop MyDFlipflop(.d(d), .clk(clk), .q(q), .qb(qb));
8
9     initial begin
10         $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
11
12         // Display the column headings
13         $display("\t\ttime,\td,\tq"); // \t means tab
14
15         // Set up a monitor routine for pins of interest
16         $monitor("%d,\t\b,\t\b", $time, d, q) ;
17
18         clk = 0; d = 0; #5
19         clk = 1; d = 0; #5
20
21         clk = 0; d = 1; #5
22         clk = 0; d = 0; #5
23
24         clk = 0; d = 1; #5
25         clk = 1; d = 1; #5
26
27         clk = 0; d = 0; #5
28         clk = 1; d = 0; #5
29
30         clk = 0; d = 1; #5
31         clk = 1; d = 1; #5
32
33         clk = 0; d = 0;
34     end
35 endmodule
```



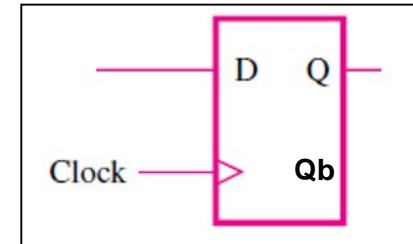
Symbol of a D flip-flop

```
1 // D flip-flop
2 module DFlipflop(d, clk, q, qb);
3     input wire clk, d;
4     output reg q;
5     output wire qb;
6
7     always @(posedge clk )
8     begin
9         q <= d;
10    end
11
12    assign qb = ~q;
13 endmodule
```

FPGAs and Verilog

Testing a Verilog design: D flip-flop

- Steps: follow the same steps as outlined previously



Symbol of a D flip-flop

```
1 // Test Bench for D flip-flop
2 module DFlipflop_tb;
3   reg d, clk;
4   wire q, qb;
5
6   // Instantiate design under test
7   DFlipflop MyDFlipflop(.d(d), .clk(clk), .q(q), .qb(qb));
8
9   initial begin
10    $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
11
12    // Display the column headings
13    $display("\t\ttime,\td,\tq"); // \t means tab
14
15    // Set up a monitor routine for pins of interest
16    $monitor("%d,\t\b,\t\b", $time, d, q) ;
17
18    clk = 0; d = 0; #5
19    clk = 1; d = 0; #5
20
21    clk = 0; d = 1; #5
22    clk = 0; d = 0; #5
23
24    clk = 0; d = 1; #5
25    clk = 1; d = 1; #5
26
27    clk = 0; d = 0; #5
28    clk = 1; d = 0; #5
29
30    clk = 0; d = 1; #5
31    clk = 1; d = 1; #5
32
33    clk = 0; d = 0;
34  end
35 endmodule
```

Saves signal information for plotting later

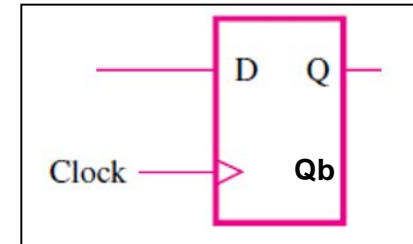
```
1 // D flip-flop
2
3 module DFlipflop(d, clk, q, qb);
4   input wire clk, d;
5   output reg q;
6   output wire qb;
7
8   always @(posedge clk )
9   begin
10    q <= d;
11  end
12
13  assign qb = ~q;
14 endmodule
```

FPGAs and Verilog

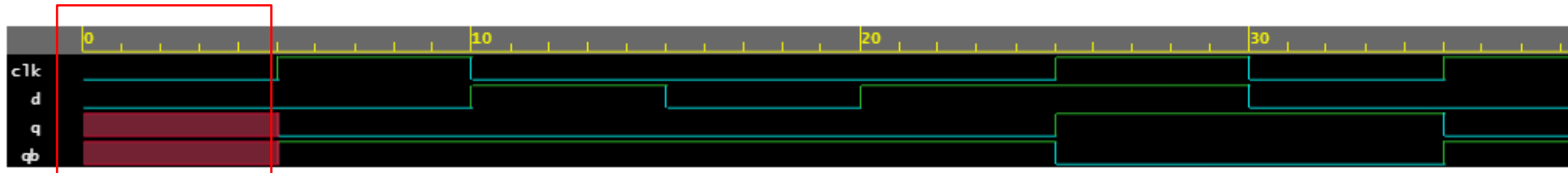
Testing a Verilog design: D flip-flop

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

```
17
18      clk = 0; d = 0; #5
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```



Symbol of a D flip-flop

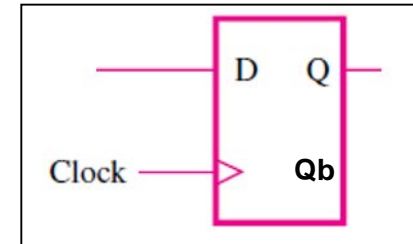


FPGAs and Verilog

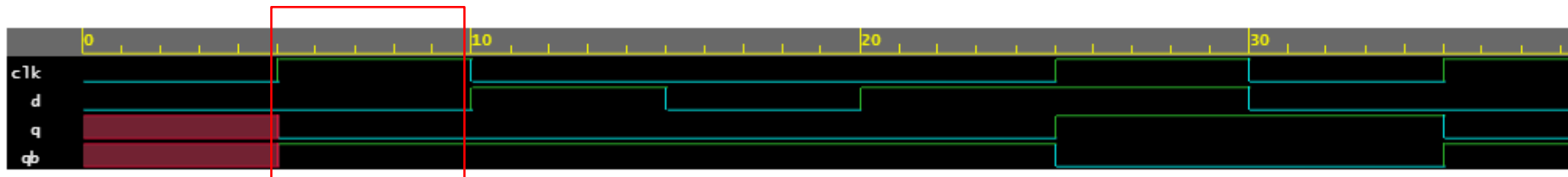
Testing a Verilog design: D flip-flop

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

```
17  
18   clk = 0; d = 0; #5  
19   clk = 1; d = 0; #5  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33
```



Symbol of a D flip-flop

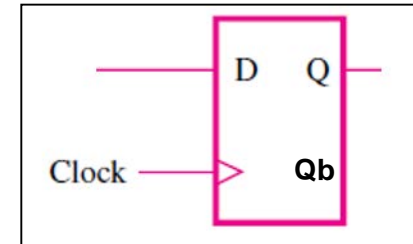


FPGAs and Verilog

Testing a Verilog design: D flip-flop

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

```
17
18     clk = 0; d = 0; #5
19     clk = 1; d = 0; #5
20
21     clk = 0; d = 1; #5
22     clk = 0; d = 0; #5
23
24
25
26
27
28
29
30
31
32
33
```



Symbol of a D flip-flop

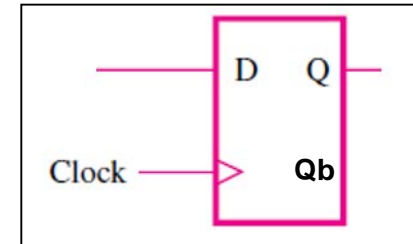


FPGAs and Verilog

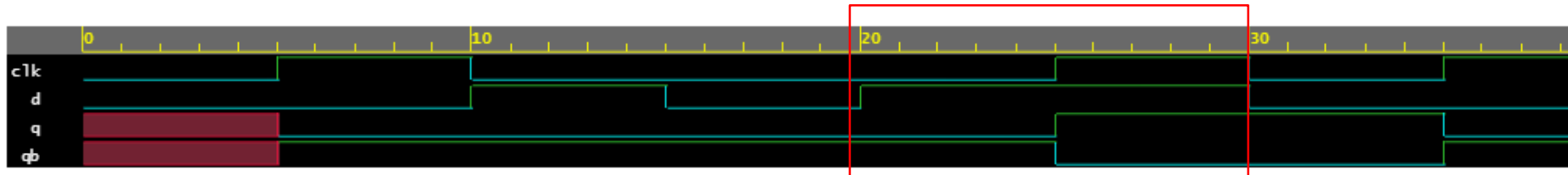
Testing a Verilog design: D flip-flop

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

```
17
18     clk = 0; d = 0; #5
19     clk = 1; d = 0; #5
20
21     clk = 0; d = 1; #5
22     clk = 0; d = 0; #5
23
24     clk = 0; d = 1; #5
25     clk = 1; d = 1; #5
26
27
28
29
30
31
32
33
```



Symbol of a D flip-flop

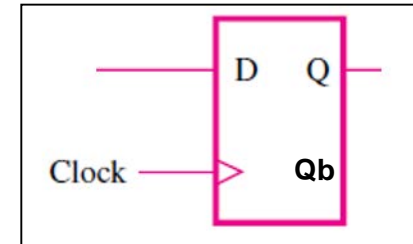


FPGAs and Verilog

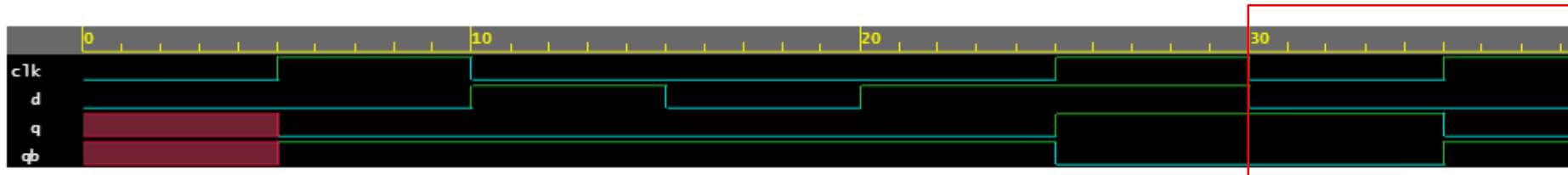
Testing a Verilog design: D flip-flop

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

```
17
18     clk = 0; d = 0; #5
19     clk = 1; d = 0; #5
20
21     clk = 0; d = 1; #5
22     clk = 0; d = 0; #5
23
24     clk = 0; d = 1; #5
25     clk = 1; d = 1; #5
26
27     clk = 0; d = 0; #5
28     clk = 1; d = 0; #5
29
30
31
32
33
```



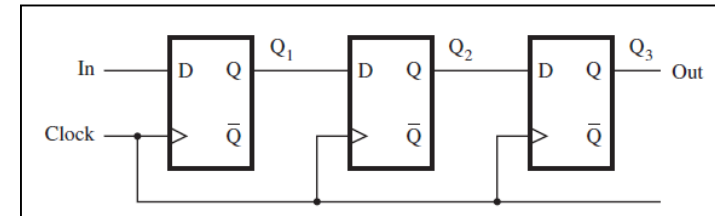
Symbol of a D flip-flop



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

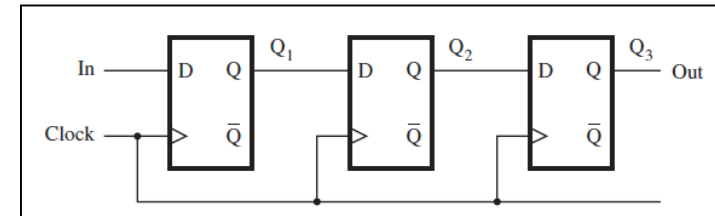
```
1 // Test Bench for Shift Register
2 module ShiftRegister_tb;
3     reg In, clk;
4     wire Q1, Q2, Out;
5
6     // Instantiate design under test
7     ShiftRegister MyShiftRegister(.In(In), .clk(clk), .Q1(Q1), .Q2(Q2), .Out(Out));
8
9     initial begin
10         $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
11
12         // Display the column headings
13         $display("\t\ttime,\tIn,\tQ1,\tQ2,\tout"); // \t means tab
14
15         // Set up a monitor routine for pins of interest
16         $monitor("%d,\t\b,\t\b,\t\b,\t\b", $time, In, Q1, Q2, Out) ;
17
18         clk = 0; In = 1; #5
19         clk = 1; In = 1; #5
20
21         clk = 0; In = 1; #5
22         clk = 1; In = 1; #5
23
24         clk = 0; In = 1; #5
25         clk = 1; In = 1; #5
26
27         clk = 0; In = 1; #5
28         clk = 1; In = 1; #5
29
30         clk = 0; In = 1; #5
31         clk = 1; In = 1; #5
32
33         clk = 0; In = 1;
34     end
35 endmodule
```

```
1 // Shift register
2
3 module ShiftRegister(In, clk, Q1, Q2, Out);
4     input wire In, clk;
5     output reg Q1, Q2, Out;
6
7     always @(posedge clk)
8     begin
9
10         Q1 <= In;
11         Q2 <= Q1;
12         Out <= Q2;
13
14     end
15
16 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

```

1 // Test Bench for Shift Register
2 module ShiftRegister_tb;
3     reg In, clk;
4     wire Q1, Q2, Out;
5
6     // Instantiate design under test
7     ShiftRegister MyShiftRegister(.In(In), .clk(clk), .Q1(Q1), .Q2(Q2), .Out(Out));
8
9     initial begin
10         $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
11
12         // Display the column headings
13         $display("\t\ttime,\tIn,\tQ1,\tQ2,\tout"); // \t means tab
14
15         // Set up a monitor routine for pins of interest
16         $monitor("%d,\t%b,\t%b,\t%b,\t%b", $time, In, Q1, Q2, Out) ;
17
18         clk = 0; In = 1; #5
19         clk = 1; In = 1; #5
20
21         clk = 0; In = 1; #5
22         clk = 1; In = 1; #5
23
24         clk = 0; In = 1; #5
25         clk = 1; In = 1; #5
26
27         clk = 0; In = 1; #5
28         clk = 1; In = 1; #5
29
30         clk = 0; In = 1; #5
31         clk = 1; In = 1; #5
32
33         clk = 0; In = 1;
34     end
35 endmodule

```

```

1 // Shift register
2 module ShiftRegister(In, clk, Q1, Q2, Out);
3     input wire In, clk;
4     output reg Q1, Q2, Out;
5
6     always @(posedge clk)
7     begin
8
9
10        Q1 <= In;
11        Q2 <= Q1;
12        Out <= Q2;
13
14    end
15
16 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

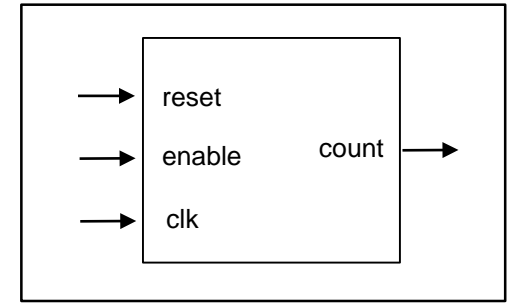
∴ ShiftRegister was accurately coded in Verilog

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



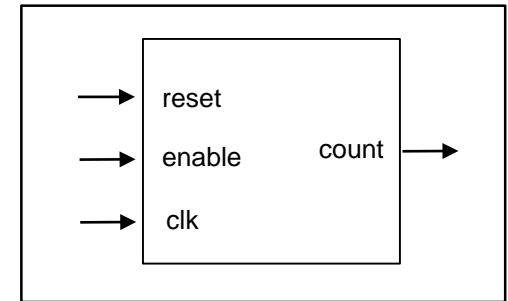
4-bit counter

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

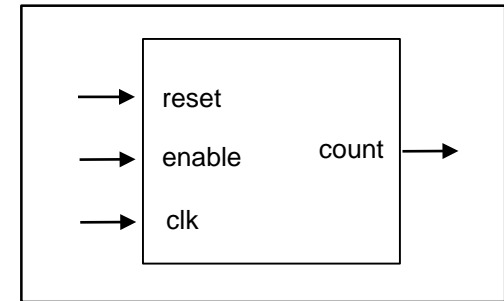
```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule
```

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29
30    end
31
32 endmodule
```

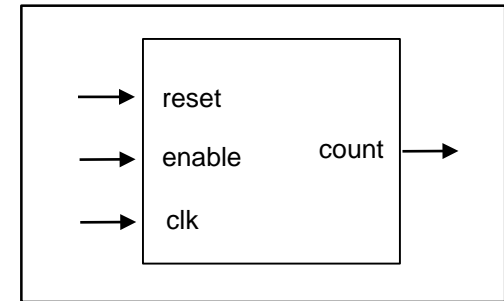
```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule
```

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\tretset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29    end
30
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule
14
```

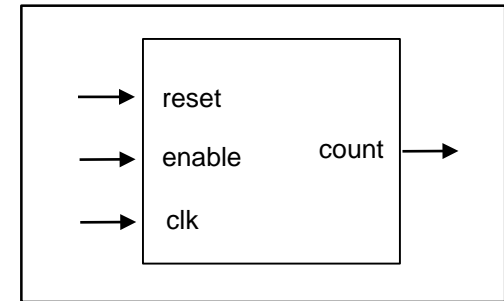
time,	reset,	enable,	count
0,	0,	0,	x
5,	1,	0,	0
10,	0,	1,	0
15,	0,	1,	1
25,	0,	1,	2
35,	0,	1,	3
45,	0,	1,	4
55,	0,	1,	5

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\tretset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29    end
30
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule
14
```

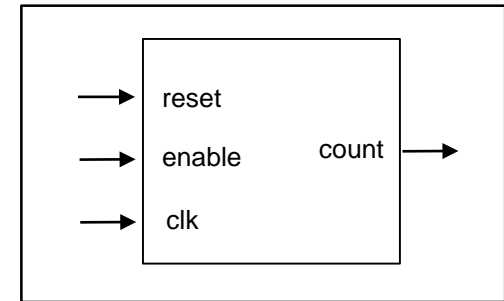
time,	reset,	enable,	count
0,	0,	0,	x
5,	1,	0,	0
10,	0,	1,	0
15,	0,	1,	1
25,	0,	1,	2
35,	0,	1,	3
45,	0,	1,	4
55,	0,	1,	5

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\tretset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29    end
30
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule
14
```

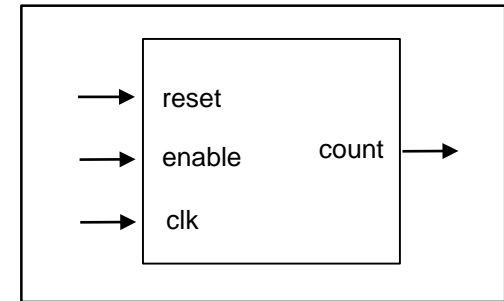
time,	reset,	enable,	count
0,	0,	0,	x
5,	1,	0,	0
10,	0,	1,	0
15,	0,	1,	1
25,	0,	1,	2
35,	0,	1,	3
45,	0,	1,	4
55,	0,	1,	5

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\tretset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29
30    end
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12 endmodule
13
```

repeat = 1

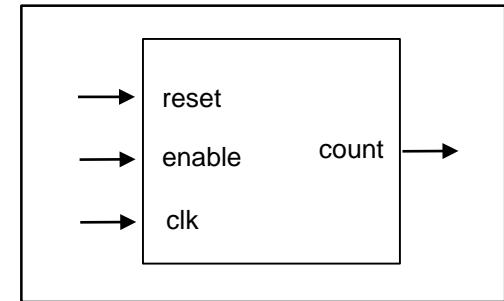
time,	reset,	enable,	count
0,	0,	0,	x
5,	1,	0,	0
10,	0,	1,	0
15,	0,	1,	1
25,	0,	1,	2
35,	0,	1,	3
45,	0,	1,	4
55,	0,	1,	5

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\tretset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29    end
30
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule
14
```

repeat = 3

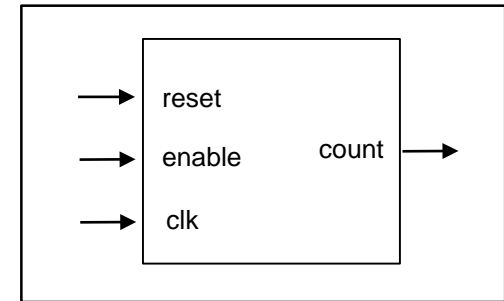
time,	reset,	enable,	count
0,	0,	0,	x
5,	1,	0,	0
10,	0,	1,	0
15,	0,	1,	1
25,	0,	1,	2
35,	0,	1,	3
45,	0,	1,	4
55,	0,	1,	5

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29    end
30
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule

```

repeat = 5

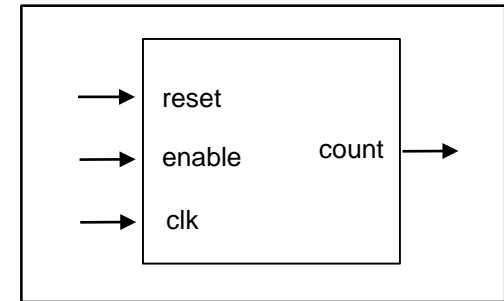
time,	reset,	enable,	count
0,	0,	0,	x
5,	1,	0,	0
10,	0,	1,	0
15,	0,	1,	1
25,	0,	1,	2
35,	0,	1,	3
45,	0,	1,	4
55,	0,	1,	5

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29    end
30
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule
14
```

repeat = 7

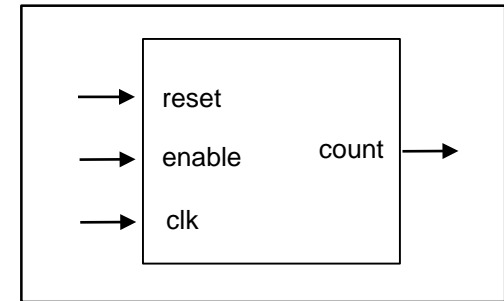
time,	reset,	enable,	count
0,	0,	0,	x
5,	1,	0,	0
10,	0,	1,	0
15,	0,	1,	1
25,	0,	1,	2
35,	0,	1,	3
45,	0,	1,	4
55,	0,	1,	5

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29    end
30
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12
13 endmodule
14
```

repeat = 9

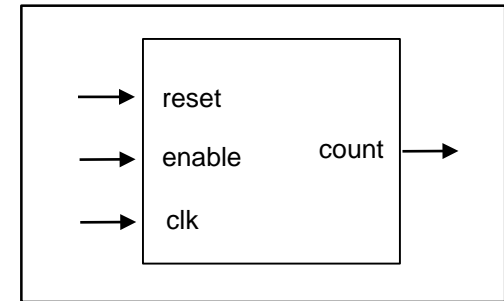
time,	reset,	enable,	count
0,	0,	0,	x
5,	1,	0,	0
10,	0,	1,	0
15,	0,	1,	1
25,	0,	1,	2
35,	0,	1,	3
45,	0,	1,	4
55,	0,	1,	5

FPGAs and Verilog

Design and test a digital circuit: 4-bit counter

- Functionality of the counter:

- When reset line = 1: $\text{count} = 0$
- When reset line = 0, enable = 1: $\text{count} = \text{count} + 1$



4-bit counter

```
1 // Test Bench for 4-bit counter
2
3 module counter_tb;
4     reg reset, enable, clk;
5     wire [3:0] count;
6
7     // Instantiate design under test
8     counter Mycounter(.reset(reset), .enable(enable), .clk(clk), .count(count));
9
10    initial begin
11        $dumpfile("dump.vcd"); $dumpvars(1); // For plotting
12
13        // Display the column headings
14        $display("\t\ttime,\treset,\tenable,\tcount"); // \t means tab
15
16        // Set up a monitor routine for pins of interest
17        $monitor("%d,\t%d,\t%d,\t%d", $time, reset, enable, count);
18
19        clk = 0; reset = 0; enable = 0;
20        #5 clk = !clk; // toggle clock from LOW to HIGH
21        reset = 1; // reset counter
22        #5 clk = !clk; // toggle clock from HIGH to low
23        reset = 0; enable = 1; //enable counter
24
25        repeat(10)
26            begin
27                #5 clk = !clk; // toggle clock every 5 time units
28            end
29    end
30 end
31 endmodule
32
```

```
1 // 4-bit Counter
2
3 module counter(clk, reset, enable, count);
4     input wire clk, reset, enable;
5     output reg [3:0] count;
6
7     always @ (posedge clk)
8     begin
9         if (reset == 1'b1) count <= 0;
10        else if (enable == 1'b1) count <= count + 1;
11    end
12 endmodule
13
```

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