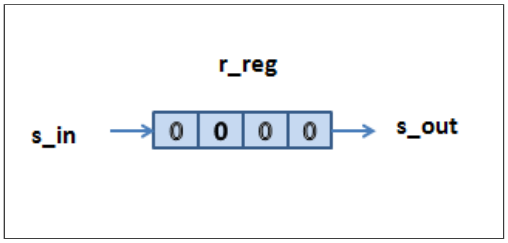
Reference Designer

Shift Register using verilog

We will now consider a shift register. Our shift register has an s_in input entering on its left hand side. At each clock cycle, the content of the register shifts to the right and s_in enters into the leftmost bit or the MSB bit. The whole design also has and output that we are c calling s_out. At each clock cyccle the right most bit of the register comes out.

The picture shows the scheme of the shift register.



Here is the verilog implemmentation of shift register.

```
1.  // referencedesigner.com
2.  // Example of shift register
3.  module free_run_shift_reg
4.     #(parameter N=8)
5.     (
6.      input wire clk, reset,
7.      input wire s_in,
8.      output wire s_out
9.     );
10.
11.    reg [N-1:0] r_reg;
12.    wire [N-1:0] r_next;
13.
14.
15.    always @(posedge clk, negedge reset)
16.    begin
17.       if (~reset)
18.          r_reg <= 0;
19.       else
20.          r_reg <= r_next;
21.        end
22.
23.        assign r_next = {s_in, r_reg[N-1:1]};
24.        assign s_out = r_reg[0];
25.
26.
27. endmodule
```

Explanation

Initially the reg value of undefined and hence we have placed 4'bxxxx in its value.

Because of the assign statement

```
assign s_out = r_reg[0];
```

the initial value of s_reg[0] is also 0.

When the reset pulse is applied the r_reg becomes 0000 at the next rising edge of clock. Note that the period of the negative level of the reset sould last at least to the next rising edge of the clock

At this stage, the value of s_out also becomes 0 (right after the rising edge of the clock).

Now the s_in value is supplied sometimes before the next rising edge of the clock. Now because of the assign statement

```
assign r_next = {s_in, r_reg[N-1:1]};
```

the wire r_next is driven by the value of s_in and [3:1] bits of r_reg.

And so, after the application of the s_in, at the next rising edge of the clock, the statement

```
r_reg <= r_next;
```

in the always loop takes effect. which essentially results in updating the r_reg value with its value shifted to right and s_in coming in at its MSB.

The testbech for the Serial shift register

```
1.  `timescale 1ns / 1ps
2.  module stimulus;
3.          // Inputs
4.          reg clk ;
5.          reg reset;
6.          // Outputs
7.          reg s_in;
8.          wire s_out;
9.          // Instantiate the Unit Under Test (UUT)
10.         free_run_shift_reg  #(2) s1  (
11.                 .clk(clk),
12.                 .reset(reset),
13.                 .s_in(s_in),
14.                 .s_out(s_out)
15.         );
16.
17.  integer i, j;
18.  initial
19.  begin
20.
21.  clk = 0;
22.  for(i =0; i<=40; i=i+1)
23.  begin
24.   #10 clk = ~clk;
25.  end
26.  end
27.
28. initial
29. begin
30.
31. $dumpfile("test.vcd");
32. $dumpvars(0,stimulus);
33.
34. s_in = 0; reset =1;
35. #2 s_in = 0 ; reset = 0;
36. #2 reset =1;
37.  for(i =0; i<=10; i=i+1)
38.  begin
39.   #20 s_in = ~s_in;
40. end
41. #20 s_in =1;
42. #20 s_in = 1;
43. #20 s_in =0;
44. #20 s_in =1;
45. #20 s_in = 1;
46. #20 s_in =0;
47. #20 s_in =1;
48. #20 s_in = 1;
49. #20 s_in =0;
50.  end
51.
52.                 initial begin
53.                  $monitor("clk=%d s_in=%d,s_out=%d",clk,s_in, s_out);
54.                  end
55.
56. endmodule
```

Exercizes

1. In test bench the shift register is instantiated with N=2. Verify that it behaves as expected. Repeat the testbench and verification for N=4

2. Write the above code for left shift in place of right shift. The data now comes out of the MSB. The data enters from LSB.

< Previous Next >