Procedural Blocks in Verilog and System Verilog

Dev's VLSI Training Institute

PRESENTED BY
ANU NEELI

Contents:

Procedural block:

A procedural block is a block of code that runs sequentially to describe behavior of hardware — how signals change over time.

Procedural Assignment:

Procedural assignment refers to assigning values to variables inside procedural blocks

Procedural Blocks in Verilog:

- Initial
- Always

Procedural Blocks in System Verilog:

- Initial
- Always
- Final

1. Initial block

This procedural block is executed only once when the Verilog simulation begins at time at 0 time units.

a. It is a non synthesizable block and it can note contribute to building design schematic (RTL) development and for FPGA it can able to synthesizable.

- This block is mainly used in text bench.
- 'Initial' keyword is used to declare an initial procedural block

Syntax:

```
initial
statement;
```

- If the block consist of more than one statement are closed with in begins end statement.

Syntax:

```
initial  begin
statement;
statements;
end
```

If more than one initial block declared in a module, all the basically will run concurrently.. We will run parallelly and will all start at the same time at 0 time. Every individual block may finish their execution independently.

Code:

```
module tb;
reg a, b;
  wire y;
and_gate dut (a, b, y);
initial begin
    a = 0; b = 0; #10;
    a = 0; b = 1; #10;
    a = 1; b = 0; #10;
   a = 1; b = 1; #10;
end
initial begin
     $monitor(“a=%0b,b=%0b,y=%0b", $time, a, b, y);
end
endmodule
```

Definition: An always block is one of the *procedural* blocks in Verilog. Statements inside an always block are executed sequentially and performs some action when a signal within the sensitivity list becomes active.
Sensitivity list is represented within the parentheses().
To trigger the execution of always block on some event we use @() and inside parentheses we write the name of variables or net upon which execution will depend.
Multiple always blocks are valid in a module and each block independently.
Nested always blocks are not valid

Syntax:

1.always @ (event)
[statement]

2. always @ (event) begin
[multiple statements]
end

An always block can be used to realize combinational or sequential elements.
a combinational block becomes active when one of its input values change.
A sequential element like flip flop becomes active when it is provided with a clock and reset.

Code:

```
module mux2to1 (input a, b, sel,output reg y);
always @(*)
//always@(a, b, c)
//always@(a or b or c)
//always@(a)
begin
   if (sel == 0)
     y = a;
   else
     y = b;
 end
endmodule
```

Code:

```
module d_ff ( input clk, reset, d,output reg q);
 always @(posedge clk) // Active-high
synchronous
always @(posedge clk,posedge reset) //Active
high asynchronous
 begin
   if (reset)
     q <= 0;
   else
     q <= d;
 end
endmodule
```

Design code:

```
module behav(a, b, s, y );
input a, b, s;
output reg y;
 always@(*)
begin
     y=s?a:b;
    end
endmodule
```

Test Bench Code:

```
module tb;
  reg a,b,s;
  wire y,x;
  behav dut(.*);
  initial
    begin
      a=0;b=1;s=0; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
       a=1;b=0;s=1;#1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
        a=1;b=0;s=0; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
       a=0;b=1;s=1;#1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
    end
endmodule
```

Output:

```
1a=0,b=1,s=0,y=1
 2a=1,b=0,s=1,y=1
3a=1,b=0,s=0,y=0
4a=0,b=1,s=1,y=0
```

Design code:

```
module behav(a, b, s, y );
input a, b, s;
output reg y;
 always@(a)
//a changes only o/p change
otherwise prev o/p will occurs
begin
      y=s?a:b;
\\s=1 for a , s=0 for b
    end
endmodule
```

Test Bench Code:

```
module tb;
  reg a,b,s;
  wire y,x;
  behav dut(.*);
  initial
    begin
      a=0;b=1;s=0; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
       a=1;b=0;s=1;#1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
        a=1;b=0;s=0; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
       a=0;b=1;s=1;#1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
    end
endmodule
```

Output:

```
1a=0,b=1,s=0,y=1
2a=1,b=0,s=1,y=1
3a=1,b=0,s=0,y=1
4a=0,b=1,s=1,y=0
```

What happens if there is no sensitivity list ?

1.The always block repeats continuously throughout the duration of a simulation.
2.The sensitivity list brings along a certain sense of timing i.e. whenever any signal in the sensitivity list changes, the always block is triggered.
3.If there are no timing control statements within an always block, the simulation will hang because of a zero-delay infinite loop .
Syntax:
always clk = ~clk;
Note:
1.The statement is executed after every 0 time units. Hence, it executes forever because of the absence of a delay in the statement.

2. Even if the sensitivity list is empty, there should be some other form of time delay. Simulation time is advanced by a delay statement within the always construct as shown below. Now, the clock inversion is done after every 10 time units.
always #10 clk = ~clk;

Drawbacks using always block in Verilog:

Verilog uses the same always block for both combinational and sequential logic. This can lead to unintended latches(it can hold previous value only, if we change input too) of logic if sensitivity lists or control logic are incomplete.

designers had to manually list all signals in the sensitivity list. Missing one causes simulation vs. synthesis mismatch.

Representation of always block in different ways:

System Verilog:

For combinational circuit we use to represent in

always_comb

For sequential circuit we used to represent in two ways

1.always_latch

2.always_ff

But for flipflop we used write in code as

always_ff(posedge clk)

Overcome problem using System Verilog:

Automatically includes all RHS signals in sensitivity list → avoids bugs.

always_ff must have requires all paths to assign the output — if not, it flags errors to avoid latches a clock edge. If not, compiler throws an error.

We can't use comb functionality in sequential always and vice versa, if we do like that it throws a error.

Design code:

```
module behav(input a,b,s,
output reg y );
//o/p change only for a
  always_comb
   begin
    y=s?a:b;
   end
endmodule
```

Test Bench Code:

```
module tb;
 reg a,b,s;
 wire y,x;
 behav dut(.*);
 initial
  begin
   a=0;b=1;s=0; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
    a=1;b=0;s=1; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
     a=1;b=0;s=0; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
    a=0;b=1;s=1; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
  end
endmodule
```

Output:

```
1a=0,b=1,s=0,y=1
2a=1,b=0,s=1,y=1
 3a=1,b=0,s=0,y=0
4a=0,b=1,s=1,y=0
```

Design Code:

```
module behav(input a,b,s, output reg y );
 function reg fn(reg a );
  return s?a:b;
 endfunction
 always_comb
begin
   y=fn(a);
  end
endmodule
```

Test Bench Code:

```
module tb;
 reg a,b,s;
 wire y,x;
 behav dut(.*);
 initial
  begin
   a=0;b=1;s=0; #1;
$display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
    a=1;b=0;s=1; #1;
 $display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
     a=1;b=0;s=0; #1;
 $display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
   a=0;b=1;s=1; #1;
 $display($time,"a=%b,b=%b,s=%b,y=%b",a,b,s,y);
  end
endmodule
```

Output:

```
1a=0,b=1,s=0,y=1
2a=1,b=0,s=1,y=1
 3a=1,b=0,s=0,y=0
4a=0,b=1,s=1,y=0
```

- always@(*) and always_comb can’t be used in the same module block.

Design Code:

```
module behav(input a,b,s, output reg y );
always_comb   //comb wont work with same variable at LHS in both blocks
   begin
    y=s?a:b;
   end
 always@(*)
begin
    y=s?a:b;
   end
 endmodule
```

Output:

```
ERROR VCP2675 "Cannot write to a variable 'y' that is also driven by an always_comb/always_latch/always_ff procedural block. Use ""-err VCP2675 W1"" to suppress this error." "design.sv" 11 8
MESSAGE_SP VCP2674 " ... see 'y' drive in always_comb/always_latch/always_ff procedural block." "design.sv" 6 8
FAILURE "Compile failure 1 Errors 0 Warnings Analysis time: 0[s]."
Exit code expected: 0, received: 255
```

- always_comb and always_comb can't be used in the same module block.

Design Code:

```
module behav(input a,b,s, output reg y );
always_comb//comb wont work with same variable at LHS in both blocks
   begin
    y=s?a:b;
   end
always_comb
   begin
    y=s?a:b;
   end
 endmodule
```

Output:

```
ERROR VCP2675 "Cannot write to a variable 'y' that is also driven by an
always_comb/always_latch/always_ff procedural block. Use ""-err VCP2675 W1"" to suppress this
error." "design.sv" 11 8
MESSAGE_SP VCP2674 " ... see 'y' drive in always_comb/always_latch/always_ff procedural block."
"design.sv" 6 8
FAILURE "Compile failure 1 Errors 0 Warnings Analysis time: 0[s]."
Exit code expected: 0, received: 255
```

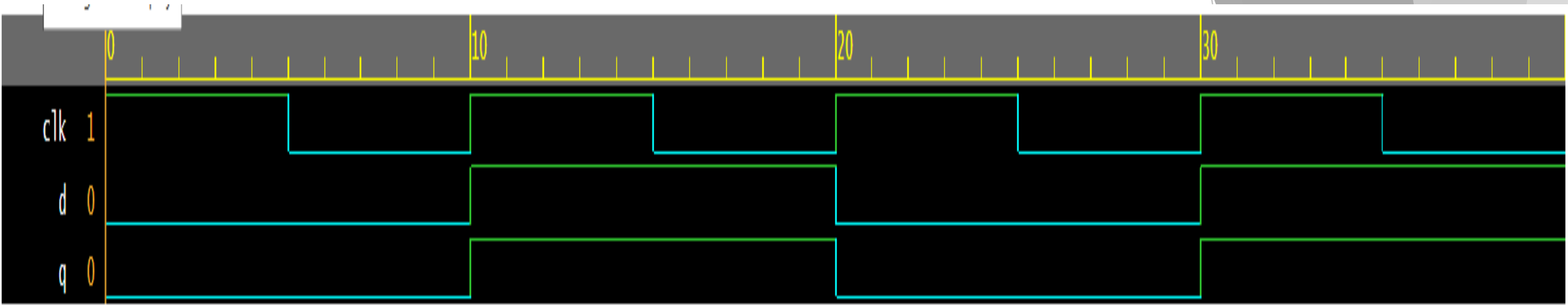
- This code is related to Latch:

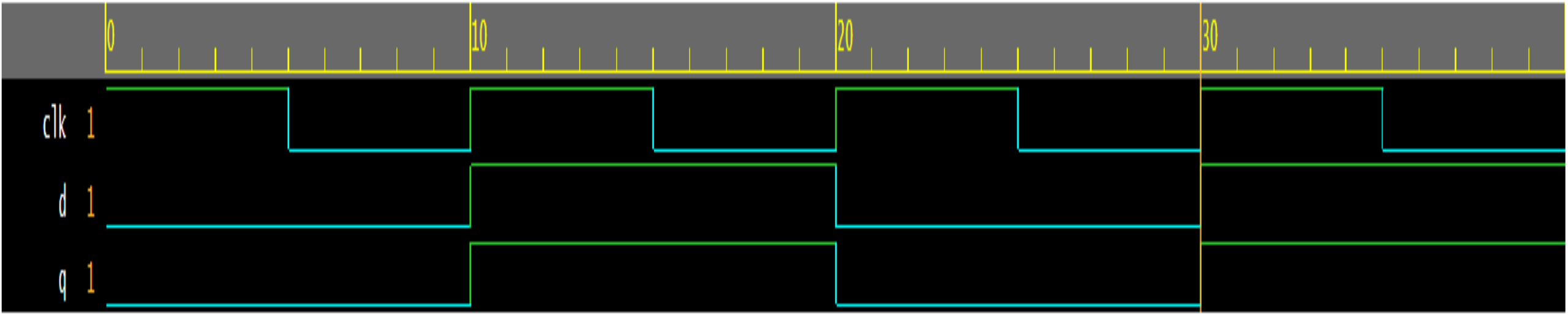
Design Code:

```
module dlatch(d,clk,q);
 input d,clk;
 output reg q;
 always_latch
  begin
   if(d)
     q<=1;
   else
     q<=0;
  end
endmodule
```

Test Bench:

```
module tb;
 reg d,clk;
 wire q;
 dlatch dut(.*);
 initial begin
    $dumpfile("dump.vcd"); $dumpvars;
    clk=1;
    forever #5 clk=~clk;
  end
 initial begin
     d=0; #10;
    d=1; #10;
    d=0;  #10;
    d=1; #10;
    $finish;
  end
endmodule
```


0
10
20
30
clk 1
d 0
q 0


0
10
20
30
clk 1
d 1
q 1

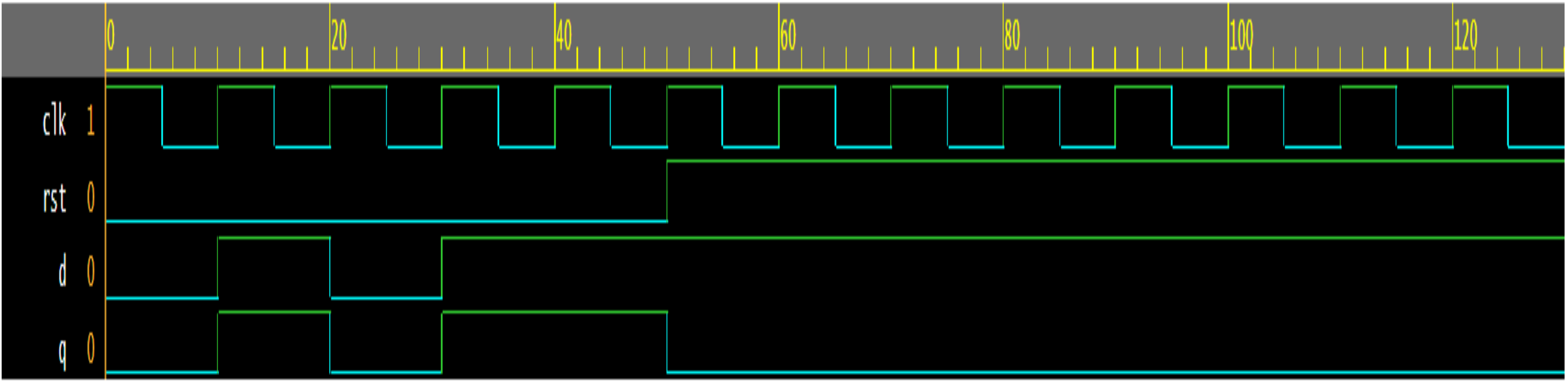
- This code is related to Flip Flop:

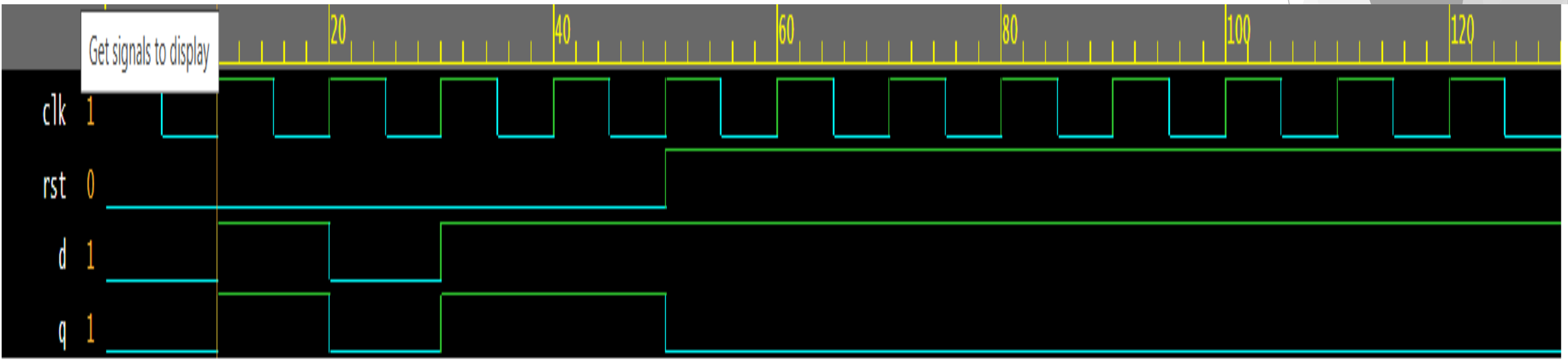
Design Code:

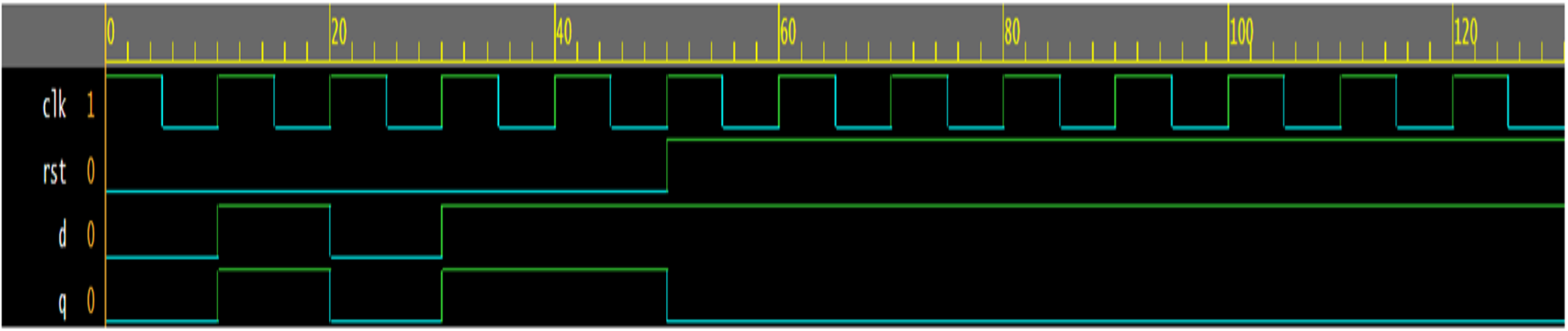
```
module dff(d,clk,rst,q);
 input d,clk,rst;
 output reg q;
 always_ff@(posedge
clk)
   begin
    if(rst)
    q<=0;
     else
       q<=d;
  end
endmodule
```

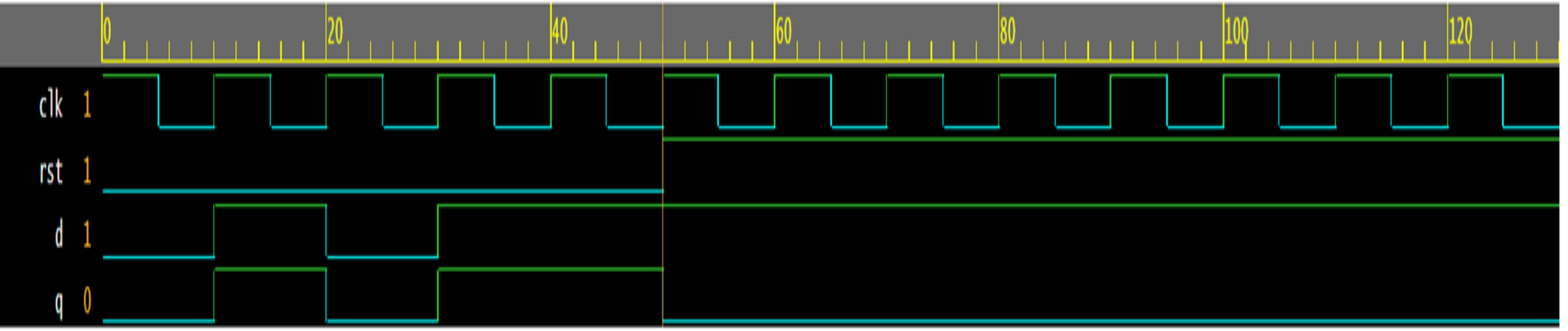
Test Bench Code:

```
module tb;
 reg d,clk;
 wire q;
 dlatch dut(.*);
 initial begin
    $dumpfile("dump.vcd"); $dumpvars;
    clk=1;
    forever #5 clk=~clk;
  end
 initial begin
     d=0; #10;
    d=1; #10;
    d=0;  #10;
    d=1; #10;
    $finish;
  end
endmodule
```


0
20
40
60
80
100
120
clk 1
rst 0
d 0
q 0


Get signals to display
20
40
60
80
100
120
clk 1
rst 0
d 1
q 1


0
20
40
60
80
100
120
clk 1
rst 0
d 0
q 0


0
20
40
60
80
100
120
clk 1
rst 1
d 1
q 0

Final Block: The final block is a special procedural block introduced in System Verilog. It runs once, just like initial, but at the very end of simulation (after $finish, or when time ends).

Not for hardware, used only in **testbenches**

Test Bench Code:

```
module tb;
  initial
    begin
      $display("before finish");
    #5;
      $display("just before finish");
    $finish;
  end
  final
  begin
    $display("final block,%0t", $time);
  end
endmodule
```

Output:

```
#Time: 5 ns
 stopped at time: 5 ns
 Simulation has finished. There are no more
test vectors to simulate.
# final block,5
```


Thank
You