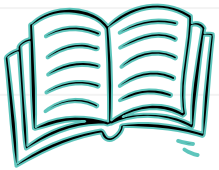
تصميم الدارات الإلكترونية بالحاسوب

Computer Design of Electronic Circuits



إعداد: د. علا جزماتي

السنة الرابعة قسم التحكم والأتمتة

العام الدراسي 2023-2024

المحاضرة الثالثة

محتويات المقرر

1. مدخل إلى أهمية تطوير أدوات التصميم باستخدام الحاسب (Introduction to The Need of Developing CAD Tools)
2. تصنيف عام لأنواع أدوات التصميم (General Classification of CAD Tools Used in Electronic Systems Design)
3. مدخل إلى اللغات المستخدمة في التصميم (Introduction to Design Languages VHDL, Verilog, Verilog System, ..)
4. مدخل إلى مراحل بناء النظم الرقمية (Introduction to Digital Systems Synthesis)
5. مرحلة البناء منخفض المستوى (Low Level Synthesis)
6. تصميم الدارات المتكاملة للنظم عالية التكامل (Layout Design for VLSI Systems)
7. تطبيقات تصميمية (Design Applications)
8. اتجاهات التطور الحديثة (Trends and New Directions)

تطبيقات حول التوصيف البنيوي:

```
// Half adder
module halfadder(
  input a,
  input b,
  output o,
  output c
);

  assign o = a ^ b;
  assign c = a & b;

endmodule
```

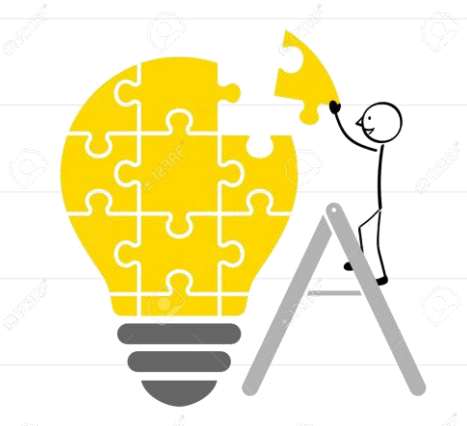
```
// Full adder
module fulladder(
  input a,
  input b,
  input d,
  output o,
  output c
);

  wire t, c1, c2;

  halfadder ha1(a, b, t, c1);
  halfadder ha2(t, d, o, c2);

  assign c = c1 | c2;

endmodule
```



توصيف بنيوي لدارة Full Adder :

```
module full_adder_4b ( input [3:0] a, input [3:0] b, input cin, output [3:0] sum, output cout );
wire [4:0] c;
wire [3:0] s;
genvar i;
generate for (i = 0; i < 4; i = i + 1)
begin :
label full_adder  fa (a[i], b[i], c[i], s[i], c[i+1]);
end
endgenerate
assign sum = s;
assign cout = c[4];
assign c[0] = cin ;
endmodule
```

```
module full_adder (  input a, input b, input
cin, output sum,  output cout );

assign sum = a ^ b ^ cin;
assign cout = (a & b)  |  (a & cin)  |  (b & cin);

endmodule
```

حاول توصيف هذا النظام
بطريقة أخرى وارسم
المخطط الصندوقي له

توصيف بنيوي لدارة Full Adder :

ملاحظات:

- يتم استخدام الأمر generate لإنشاء مجموعة من كائنات full_adder، حيث يتم إنشاء عدد من الكائنات يساوي عدد البتات (4 في هذا المثال)، وكل كائن يستخدم Full Adder لجمع البتات المقابلة في a و b و c.
- يتم توصيل الحمل الناتج من الكائن السابق إلى الكائن التالي بواسطة إشارة c[i+1] .
- يتم استخدام الأمر assign لإسناد قيمة الإشارات sum و cout إلى الإشارات الوسيطة s و c[4] على التوالي.
- genvar هي كلمة مفتاحية في لغة Verilog وهي اختصار لـ generate variable تستخدم لتعريف متغيرات تستخدم في الأوامر المرتبطة بالأمر generate، مثل for loops و if statements. يمكن استخدام genvar لتحديد متغير يتم تكرار قيمه في كل مرة يتم فيها تكرار الأمر المرتبط به، مثل إنشاء مجموعة من الكائنات. يمكن استخدام genvar في أي مكان داخل الأوامر المرتبطة بالأمر generate، ويجب تعيين قيمة ابتدائية للمتغير قبل استخدامه.

Setup I/O fulladder.sv

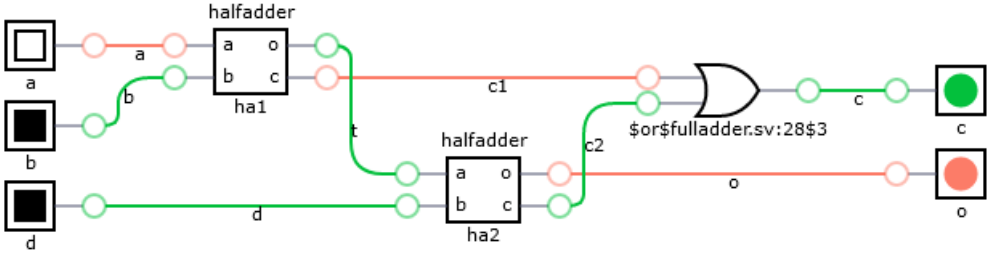
```
endmodule

// Full adder
module fulladder(
  input a,
  input b,
  input d,
  output o,
  output c
);

wire t, c1, c2;

  halfadder ha1(a, b, t, c1);
  halfadder ha2(t, d, o, c2);
```

Synthesize and simulate!



http://digitaljs.tilk.eu/#

scale 2 range 7566 – 7762

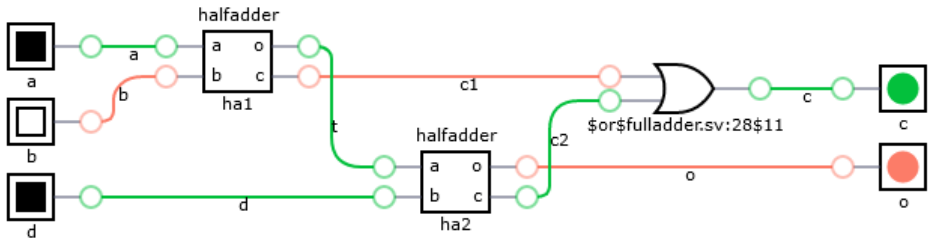
digitaljs.tilk.eu/#

10000

Setup I/O fulladder.sv fulladder0.sv

```
1  // Half adder
2  module halfadder(
3    input a,
4    input b,
5    output o,
6    output c
7  );
8
9    assign o = a ^ b;
10   assign c = a & b;
11
12 endmodule
13
14 // Full adder
15 module fulladder(
16   input a,
17   input b,
18   input d,
19   output o,
20   output c
21 );
```

Synthesize and simulate!



On Line

http://digitaljs.tilk.eu/#

digitaljs.tilk.eu/#

4904

Setup I/O fulladder.sv

DigitalJS Online

This is a demonstration app for the DigitalJS digital logic simulator and the yosys2digitaljs netlist format converter, by Marek Materzok, University of Wrocław. The source files are on Github. Contributions are welcome! This was made possible by the Yosys open-source hardware synthesis framework.

Create a new tab and enter your SystemVerilog code, or load an example by using the dropdown menu, and press the Synthesize button to start the simulation.

file name .sv Create file

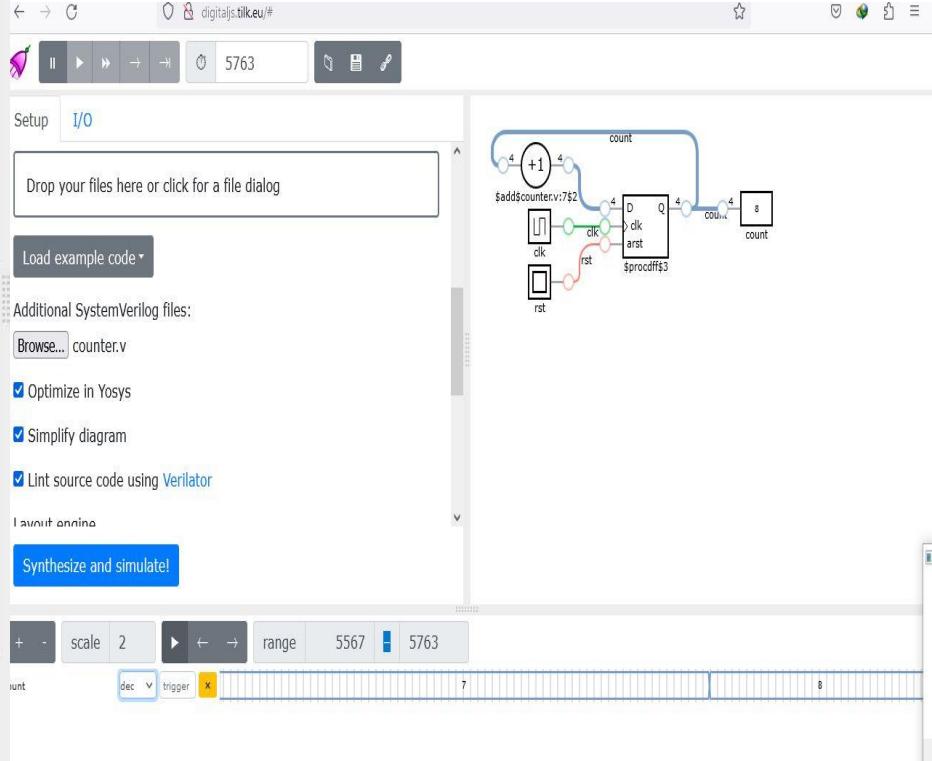
Drop your files here or click for a file dialog

Load example code

Additional SystemVerilog files:

Synthesize and simulate!

http://digitaljs.tilk.eu/#



always

- **Basic syntax:**

Signal list - change activates block

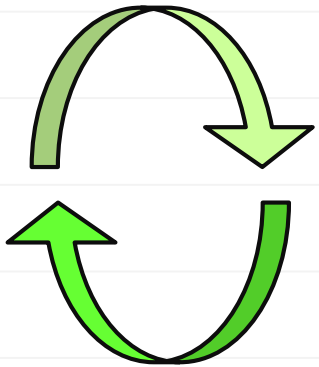
```
always (sensitivity-list)
    statement
```

Sequential statement (=, if/else, etc.)

or

```
always (sensitivity-list)
begin
    statement-sequence
end
```

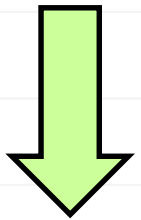
Compound Statement - sequence of sequential statements



initial

Syntax

```
1  initial
2      [single statement]
3
4  initial begin
5      [multiple statements]
6  end
```



Delays in Behavioral Verilog

- Blocking Delay

- **Delay control statement -**

  `#n sequential_statement`

  - Simulation effect: suspends simulation for n time units before simulating following statement

- **Example: clock generator**

```
always
  begin
        clk = 0;
  #50 clk = 1;
  #50 ;
  end
```

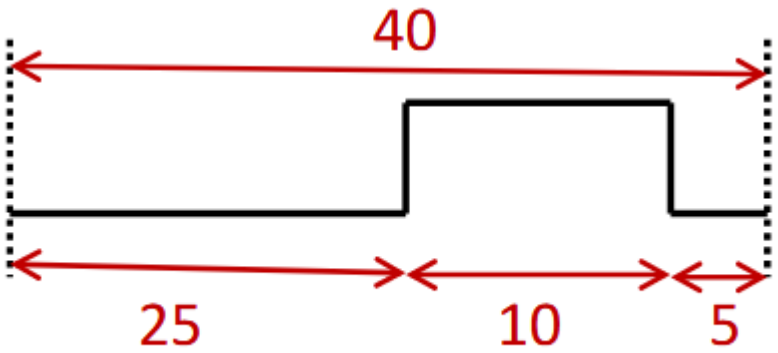
null statement

(suspends simulation 50 time units)

Delays in Behavioral Verilog

- Blocking Delay

```
initial  x = 0;
initial   begin
          #25  x = 1;
          #10  x = 0;
          #5;
end
```



Simulation Time in Verilog:

`timescale

- `timescale controls simulation time

```
`timescale time_unit time_precision

`timescale 1ns 100ps
```

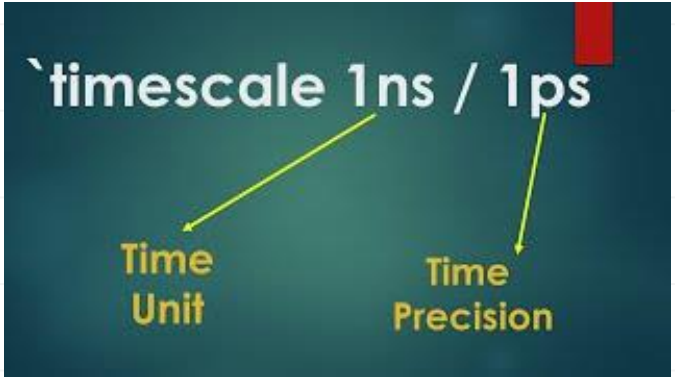
- # operator specifies delay in terms of time units

```
`timescale 1ns 100ps
#5      // delays 5*1ns = 5ns;


`timescale 4ns 1ns
#3     // delays 3*4ns = 12ns
```

Simulation Time in Verilog:

`timescale



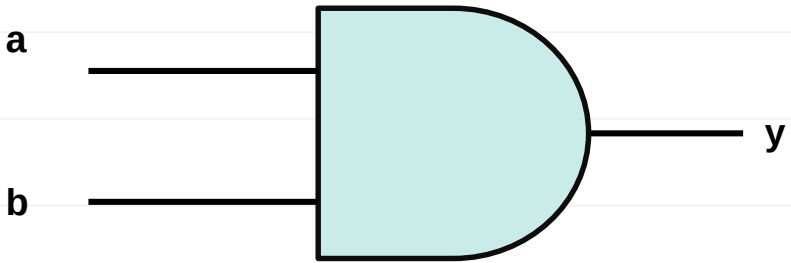
Character	Unit
s	seconds
ms	milliseconds
us	microseconds
ns	nanoseconds
ps	picoseconds
fs	femtoseconds

Verilog code for and_gate

```
module and_gate(input a, input b, output y);

assign y = a & b;

endmodule
```

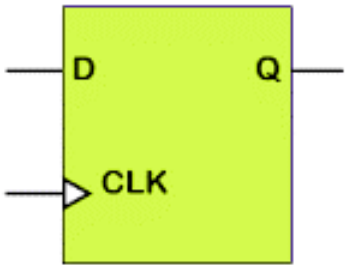


Verilog Testbench code to simulate and_gate

```
module test ();
reg a1, b1;
wire y1;
and_gate uut ( .a(a1), .b(b1), .y(y1) );
Initial begin
 a1 = 0; b1 = 0;
 #10 a1 = 0; b1 = 1;
 #10 a1 = 1; b1 = 0;
 #10 a1 = 1; b1 = 1;
end endmodule
```

Verilog code for Rising Edge D Flip Flop

```
module d_flip_flop(input d, input clk, output reg q);
always @(posedge clk)
begin
 q <= d;
end
endmodule
```



Rising-Edge D Flip-Flop

ISE Project Navigator (P.58f) - D:\Users\USER\Desktop\Test1\Test1.xise - [d_flip_flop.v*]

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

Hierarchy

- Test1
- xc3s700an-5fgg484
  - d_flip_flop (d_flip_flop.v)

No Processes Running

Processes: d_flip_flop

- Design Summary/Reports
- Design Utilities
- User Constraints
- Synthesize - XST
  - View RTL Schematic
  - View Technology Schematic
  - Check Syntax
  - Generate Post-Synthesis Simulation Model
- Implement Design
- Generate Programming File
- Configure Target Device
- Analyze Design Using ChipScope

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Additional Comments:
4  //
5  //////////////////////////////////////////////////////////////////////////////////
6  module d_flip_flop(input d, input clk, output reg q);
7  always @(posedge clk)
8  begin
9   q <= d;
10 end
11 endmodule
12
13
```

XILINX ISE DESIGN SUITE

AMD XILINX

Start Design Files Libraries

d_flip_flop.v*

Design Summary (Synthesized)

ISE Project Navigator (P.58f) - D:\Users\USER\Desktop\Test1\Test1.xise - [d_flip_flop (RTL1)]

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

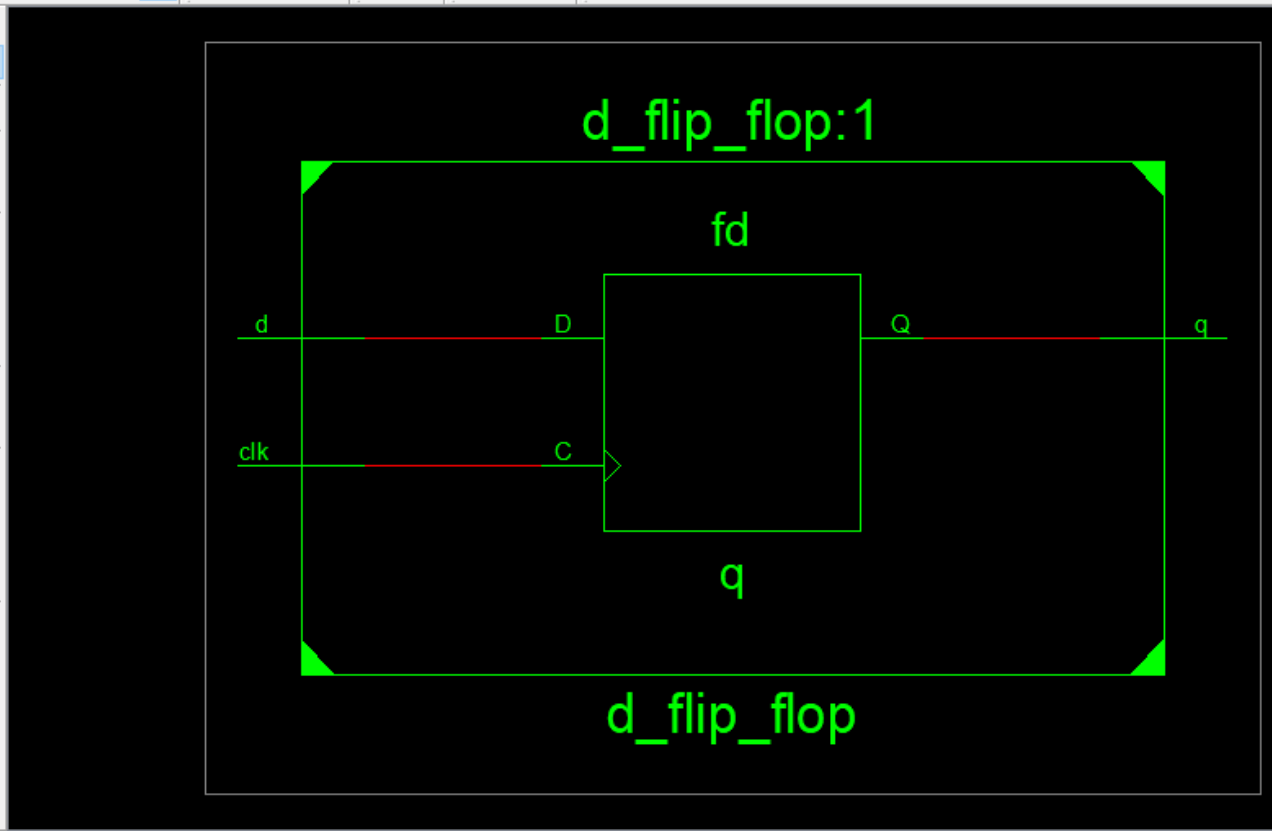
Hierarchy

- Test1
- xc3s700an-5fgg484
  - d_flip_flop (d_flip_flop.v)

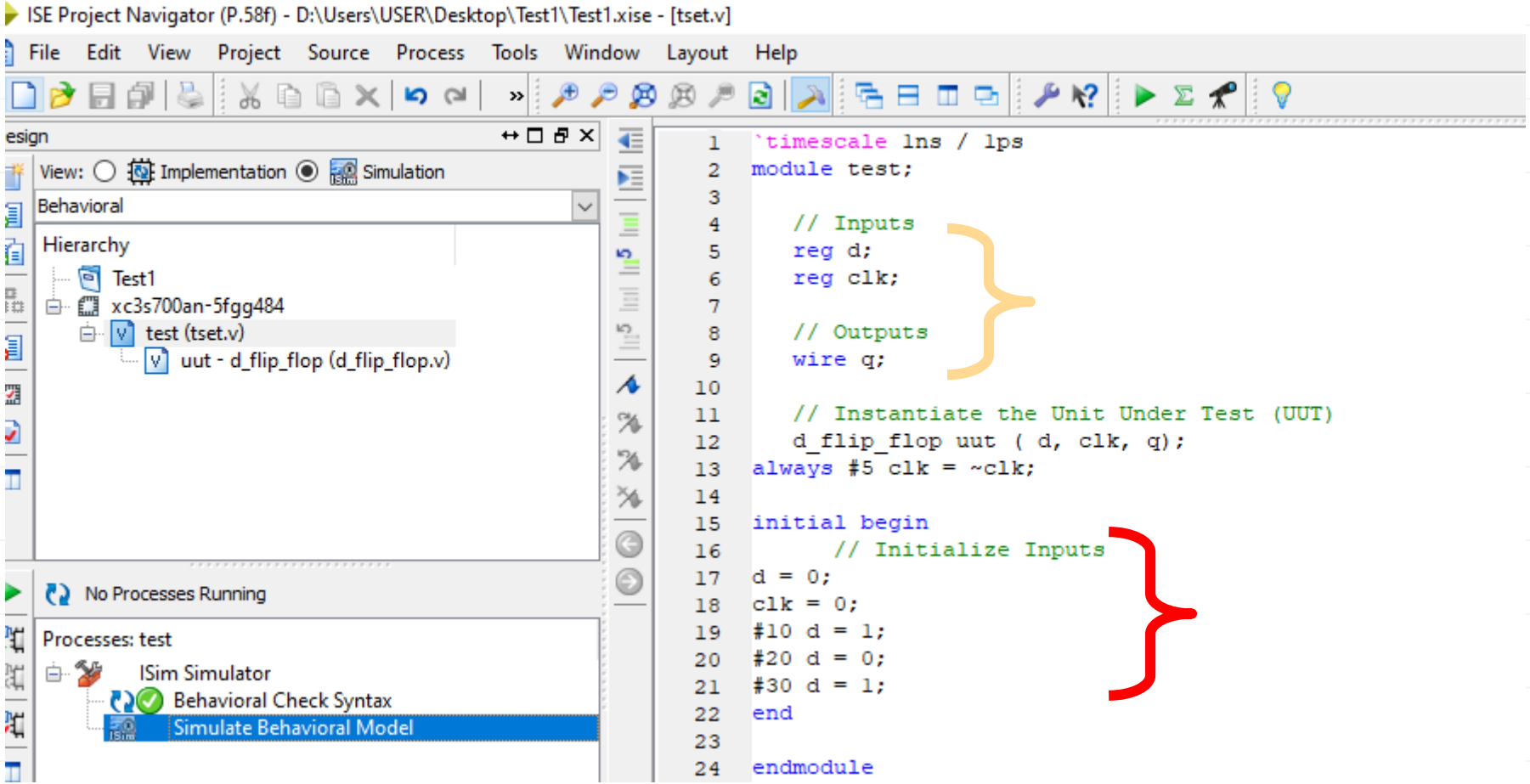
No Processes Running

Processes: d_flip_flop

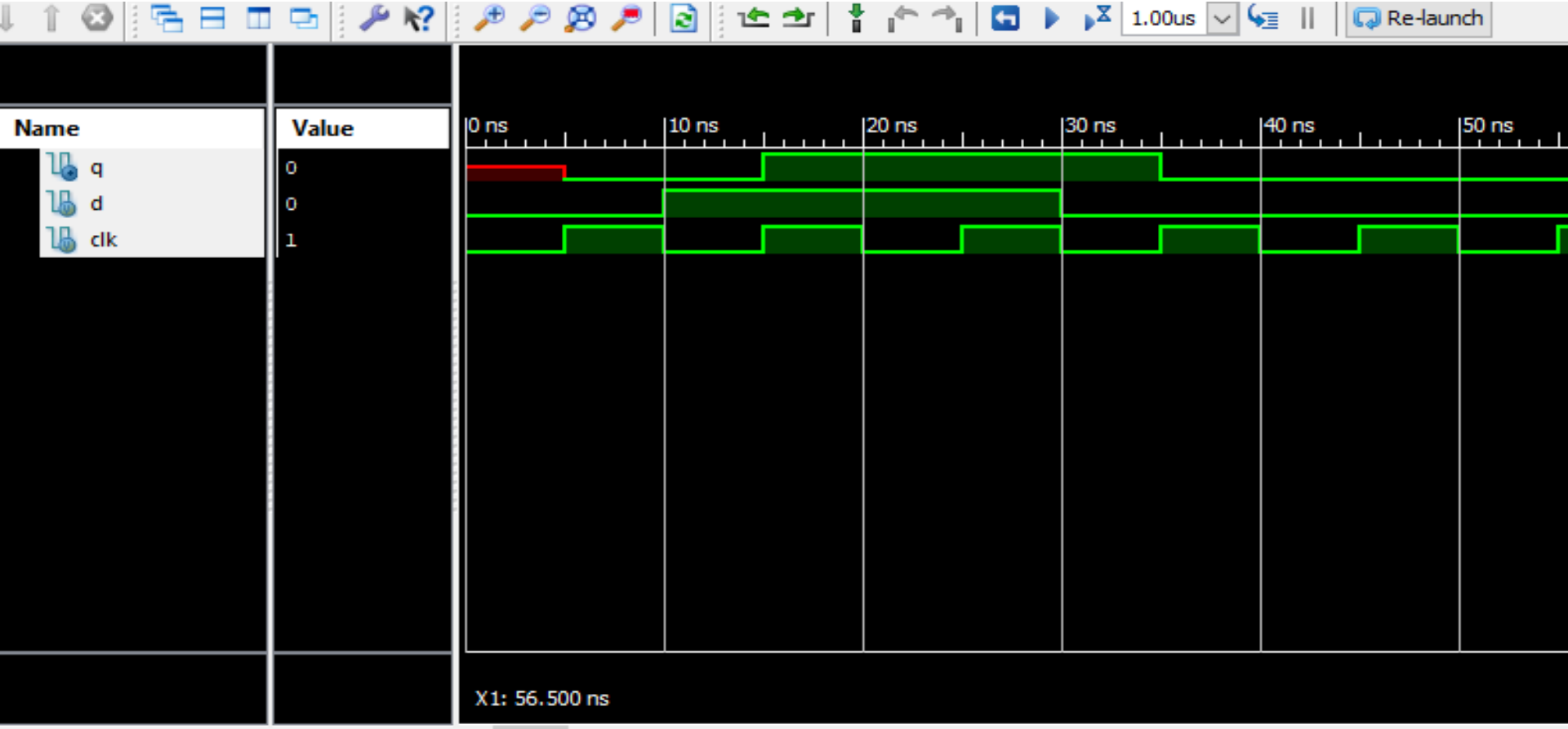
- Design Summary/Reports
- Design Utilities
- User Constraints
- Synthesize - XST
  - View RTL Schematic
  - View Technology Schematic
  - Check Syntax
  - Generate Post-Synthesis Simulation Model
- Implement Design
- Generate Programming File
- Configure Target Device
- Analyze Design Using ChipScope



Verilog Testbench code to simulate D Flip-Flop:



Verilog Testbench code to simulate D Flip-Flop:



Verilog code for up counter

```
module up_counter(input clk, reset, output[3:0] counter);
reg [3:0] counter_up;
// up counter
always @(posedge clk or posedge reset)
begin
if(reset)
 counter_up <= 4'b0;
else
 counter_up <= counter_up + 4'b0001;
end
assign counter = counter_up;
endmodule
```

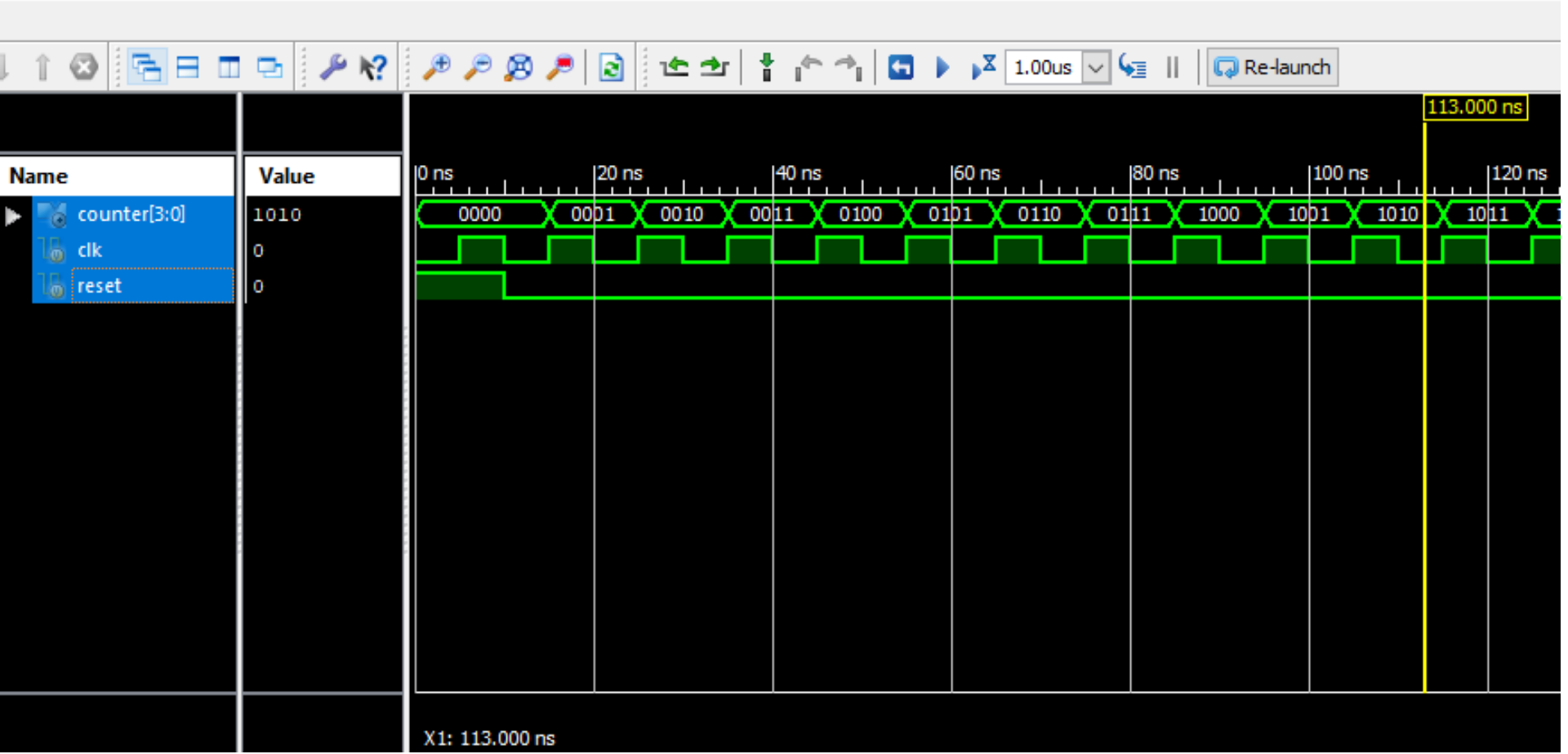
Verilog code for up counter test

```
module upcounter_testbench;
    // Inputs
    reg clk;
    reg reset;
    // Outputs
    wire [3:0] counter;
    // Instantiate the Unit Under Test (UUT)
    up_counter uut (
        .clk(clk),
        .reset(reset),
        .counter(counter) );
always #5 clk=~clk;
    initial begin
        // Initialize Inputs
clk = 0;
reset=1;
#10;
reset=0;
#10;
end
endmodule
```

Or

up_counter uut (clk, reset, counter)

Verilog code for up counter test



الأسئلة والمناقشة