

Mantıksal Sistem Tasarımı – BLM 201

Hafta 5: Doğrulama Yaklaşımları



Fenerbahçe Üniversitesi

5. Hafta İçeriği

- Doğrulama Yaklaşımları
 - Verilog Testbench Oluşturma
 - ISIM Simulasyon Aracı

Doğrulama Yaklaşımları

- Verilog/VHDL ile RTL tasarımınızı gerçeklediniz.
- Syntax'ta hata yok, bitstream üretildi, ancak ...
- Tasarımın doğru çalıştığından emin olmak için ne yapmak gerekir?
 - FPGA'e konfigüre edip denemek.
 - Deneme yanılma yapılır.
 - Sorunları gözlemlemek kolay olmayabilir.
 - Zaman açısından çok maliyetlidir.

Doğrulama Yaklaşımları

- Modern dijital sistem doğrulama yaklaşımlarında, fonksiyonel doğrulama yapılmaktadır.
- Bir tasarımın tamamlanmasında harcanan eforun toplamının
 - %30'u tasarım
 - %70'i doğrulama süreçlerioluşturmaktadır.

Doğrulama Yaklaşımları

- Tasarım'ı FPGA'e konfigüre etmeden önce bilgisayar üzerinde simülasyon araçları ile doğrularız.
- Simülasyonun, FPGA'e konfigüre edip denemeye göre avantajları
 - Tasarımda değişiklik yapıldığında, çok hızlı yeniden deneme imkanı
 - Tasarımdaki tüm sinyallerin cycle hassasiyetinde görüntüleyebilme

Doğrulama Yaklaşımları

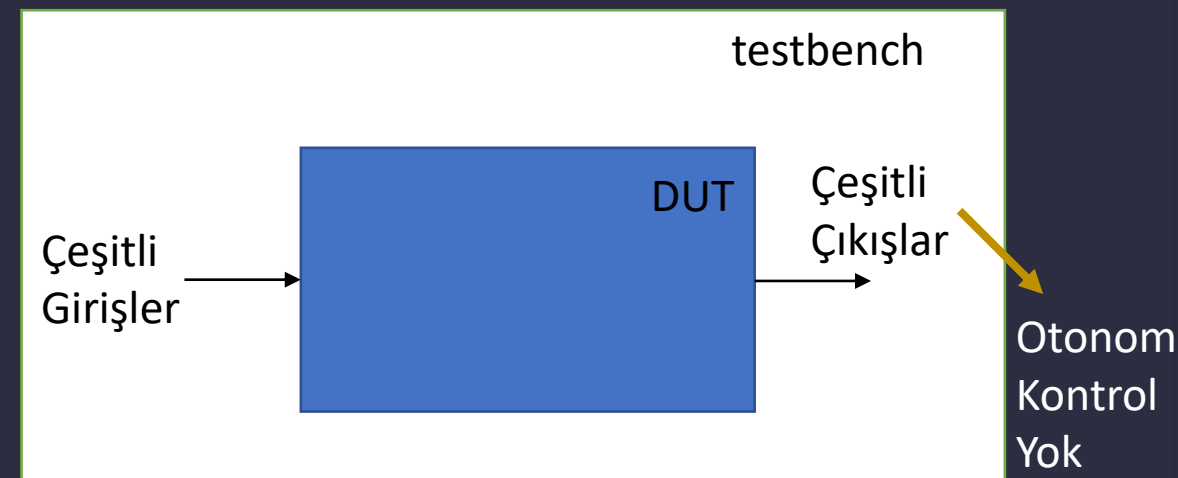
- Simulasyon yapmak için, tasarımı tamamlanmış olan modül alınır.
- Yine Verilog/VHDL kodu yazılarak, test edilmek istenen modül için giriş üreten ve çıkışlarını kontrol eden bir modül yazılır.
- Ancak test modülleri sentezlenebilir olmak zorunda değildir. Yani for, while.. gibi verilog'da sentezlenemeyen bazı yapılar testbench kodlarında kullanılabilir.

Doğrulama Yaklaşımları

- Testbench'ler 3'e ayrılabilirler.

- Basit Testbench: Bu yapıdaki testbench'lerde modülün girişlerine girişler beslenir, modülün ürettiği çıkışlar tasarımcı tarafından incelenerek doğru çalışıp çalışmadığına karar verilir.

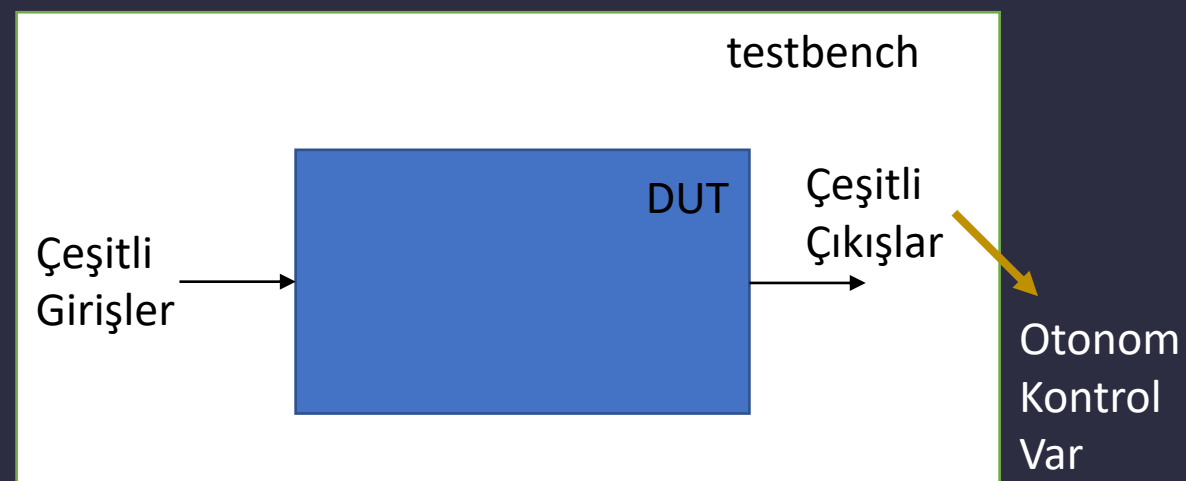
- Test edilen modüllerin instantiate ismi genellikle DUT (Design Under Test) seçilir



Doğrulama Yaklaşımları

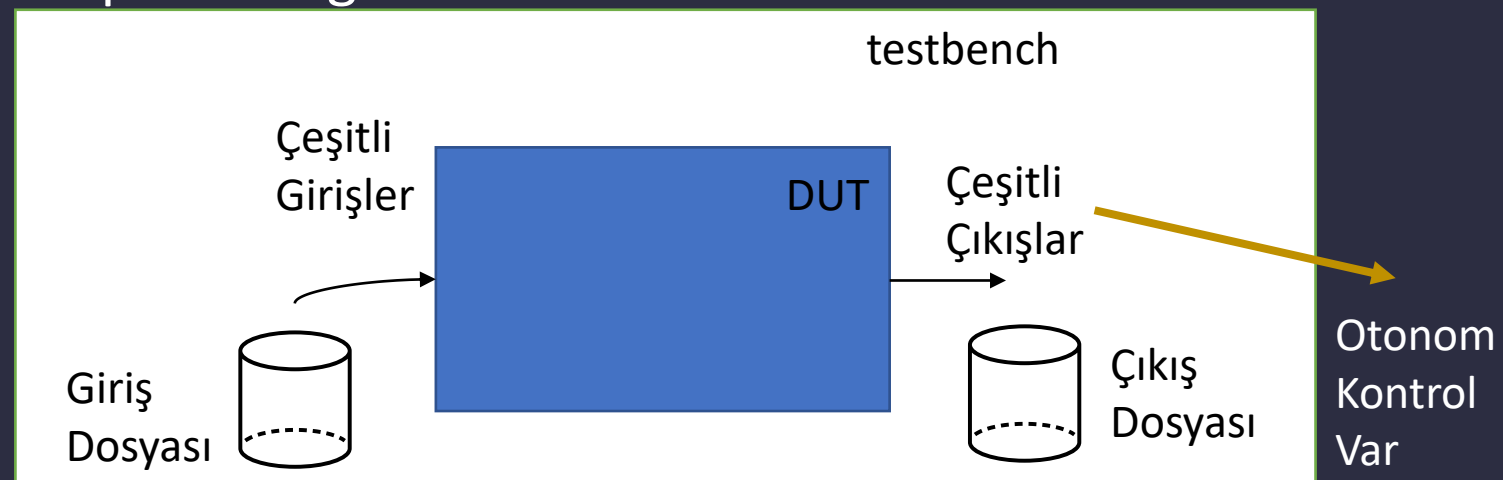
- Testbench'ler 3'e ayrılabilirler.

- Kendini kontrol eden (Self checking): Bu testbench'lerde verilen girişlere göre test edilen modülün ürettiği sinyaller tasarımcı tarafından manuel olarak gözlemlenmez. Testbench, test edilen modülün çıkarttığı sinyalleri otomatik olarak doğru veya yanlış olduğunu kontrol edecek şekilde tasarlanır.



Doğrulama Yaklaşımları

- Testbench'ler 3'e ayrılabilirler.
- Test vektörleri ile kendini kontrol eden (Self checking with vectors): Bu testbench'te, test edilecek modülen verilecek girişler ve modülün üretmesi beklenen çıkışlar bir dosyada önceden hazırlanır. Bu dosyadan testbench istenilen zamanlarda okuma yaparak modüle giriş besler ve modülün ürettiği sonucu, beklenen sonuç ile aynı olup olmadığını kontrol eder



Doğrulama Yaklaşımları

Örnek:

- $y = (b \cdot c) + (a \cdot b)$ devresinin RTL tasarımını yapıp, testbench ortamında doğrulayacak bir tasarım gerçekleştiriniz.

Doğrulama Yaklaşımları

- $y = (b \cdot c) + (a \cdot b)$

```
module ornekRTL(input a, b, c, output reg y);  
    always@(*)  
        y = ~b & ~c | a & ~b;  
endmodule
```

Doğrulama Yaklaşımları

ornekRTL modülüne çeşitli girişler besleyen bir test modülü verilmektedir.

```
`timescale 1ns / 1ps

module testbench();
  reg a, b, c;
  wire y;

  ornekRTL DUT (.a(a), .b(b), .c(c), .y(y) );

  initial begin
    a = 0;
    b = 0;
    c = 0;
    #10;
    c = 1;
    #10;
    b = 1;
    c = 0;
    #10;
    c = 1;
    #10;
  end
endmodule
```

Doğrulama Yaklaşımları

initial bloğu sadece simulasyon başladığında tek seferlik çalıştırılacaktır.

Basit testbench yaklaşımı ile tasarlanmıştır. Girişler otomatik olarak verilmekte ancak çıkışların doğru veya yanlış olduğu otomatik olarak kontrol edilmemektedir.

```
`timescale 1ns / 1ps

module testbench();
  reg a, b, c;
  wire y;

  ornekRTL DUT (.a(a), .b(b), .c(c), .y(y) );

  initial begin
    a = 0;
    b = 0;
    c = 0;
    #10;
    c = 1;
    #10;
    b = 1;
    c = 0;
    #10;
    c = 1;
    #10;
  end
endmodule
```

Doğrulama Yaklaşımları

Kendini kontrol eden testbench

- Kontrol mekanizmaları barındırır, hata varsa kullanıcıyı uyarmak için hata mesajı yazdırabilir
- \$display komutu simulasyon aracında ekrana bilgi mesajı yazdırmak için kullanılmaktadır.

```
`timescale 1ns / 1ps

module testbench2();
  reg a, b, c;
  wire y;

  ornekRTL dut(.a(a), .b(b), .c(c), .y(y));

  initial begin
    a = 0;
    b = 0;
    c = 0;
    #10;
    if (y !== 1)
      $display("1. cikis hatali.");
    c = 1;
    #10;
    if (y !== 0)
      $display("2. cikis hatali.");
    b = 1;
    c = 0;
    #10;
    if (y !== 0)
      $display("3. cikis hatali.");
  end
endmodule
```

Doğrulama Yaklaşımları

Kendini kontrol eden testbench

- Bu yaklaşımda, çok fazla test edilmesi gereken giriş varsa bunları manuel olarak vermek çok zor olabilir.
- Bunun için bir dosyadan okuyup giriş olarak veren bir testbench hazırlanabilir.

```
`timescale 1ns / 1ps

module testbench2();
  reg a, b, c;
  wire y;

  ornekRTL dut(.a(a), .b(b), .c(c), .y(y));

  initial begin
    a = 0;
    b = 0;
    c = 0;
    #10;
    if (y !== 1)
      $display("1. cikis hatali.");
    c = 1;
    #10;
    if (y !== 0)
      $display("2. cikis hatali.");
    b = 1;
    c = 0;
    #10;
    if (y !== 0)
      $display("3. cikis hatali.");
  end
endmodule
```

Doğrulama Yaklaşımları

Clock barındıran bir RTL örneği

```
`timescale 1ns / 1ps

module counter (clk, reset, enable, count);
  input clk, reset, enable;

  output reg [3:0] count = 0;
  reg [3:0] countNext = 0;

  always @ (posedge clk) begin
    count <= #1 countNext;
  end

  always@(*) begin
    countNext = count;
    if (reset == 1'b1) begin
      countNext = 0;
    end else if ( enable == 1'b1) begin
      countNext = count + 1;
    end
  end

endmodule
```


Doğrulama Yaklaşımları

Clock barındıran bir RTL testbench örneği

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

  counter U0 (
    .clk (clk),
    .reset (reset),
    .enable (enable),
    .count (count)
  );

  initial
  begin
    clk = 0;
    reset = 0;
    enable = 0;
    #10;
    enable = 1;
  end

  always #5 clk = !clk;

endmodule
```

Doğrulama Yaklaşımları

Endüstride sıklıkla kullanılan simulasyon araçları

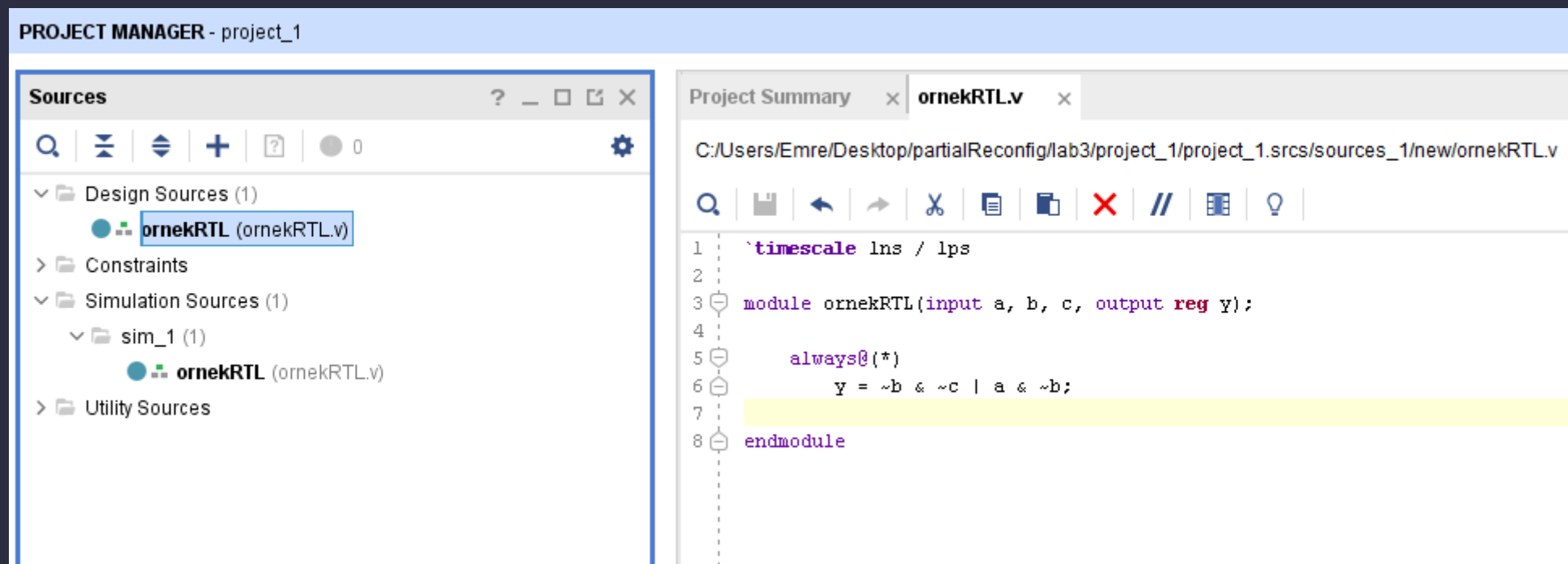
- Vivado ISIM
- Modelsim / Questa (Mentor)
- VCS (Synopsys)
- Icarus Verilog (Açık kaynak)
- Verilator (Açık kaynak)

Doğrulama Yaklaşımları

Vivado ISIM Simulator'u ile Simulasyon

Doğrulama Yaklaşımları

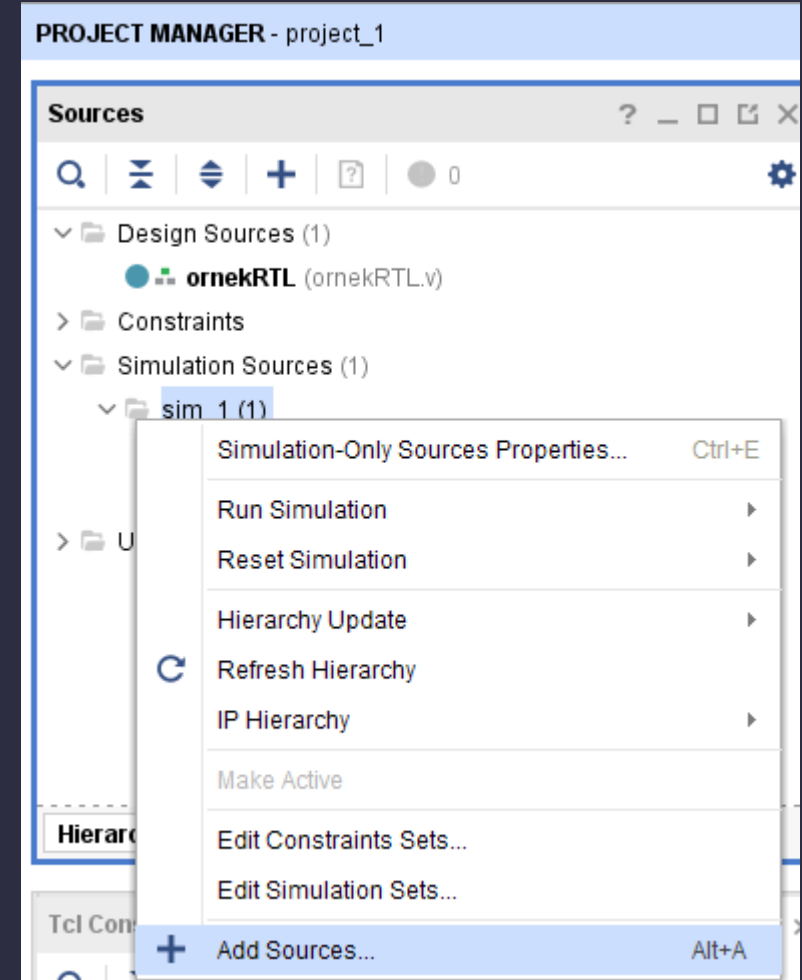
- Vivado ISIM Simulator'u ile Simulasyon



- İlk durumda, test edilecek olan modül(ler)'in tasarımı tamamlanır.

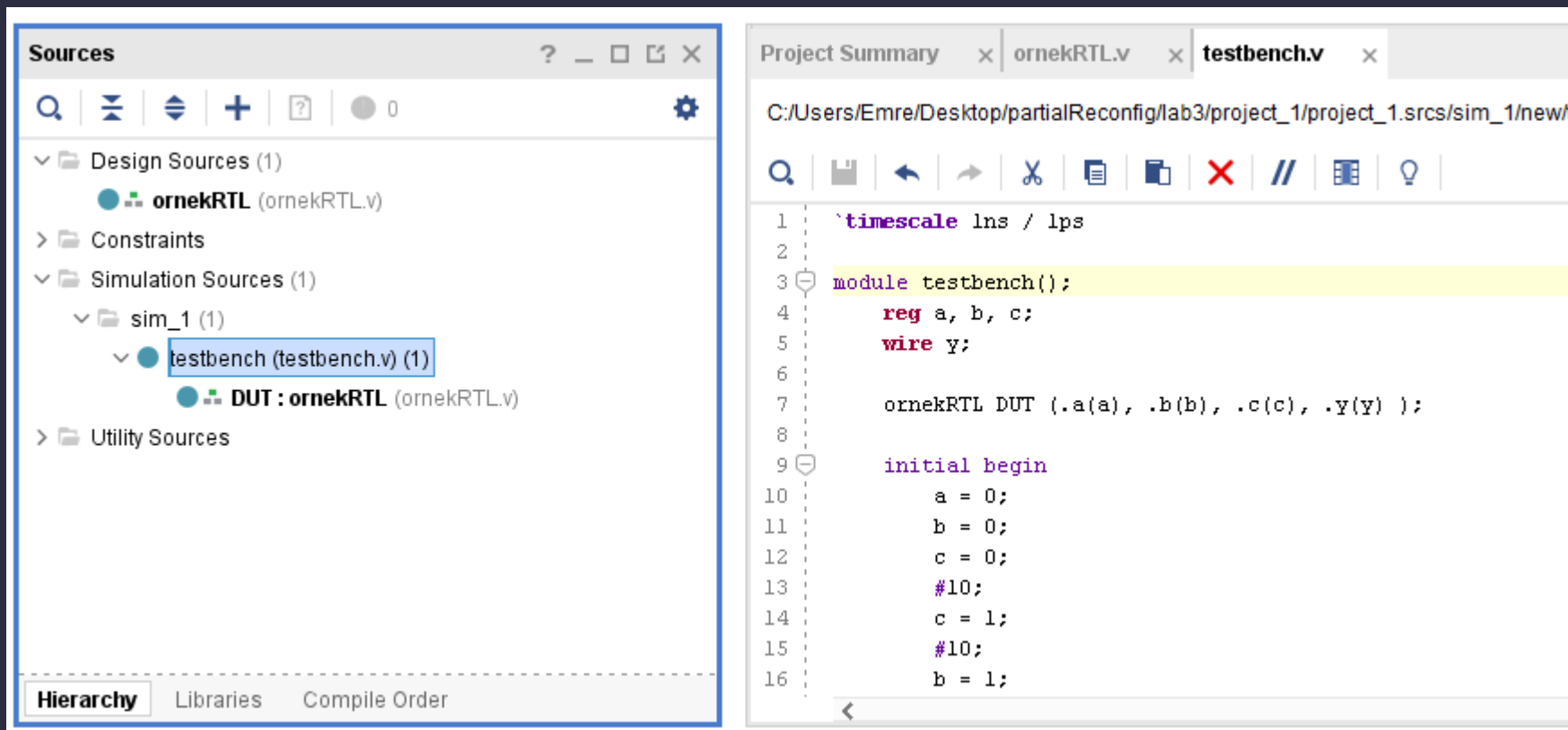
Doğrulama Yaklaşımları

- Tasarıma testbench dosyası eklemek için project manager'daki Sources bölümünde, simulation source'ın altında sim_1'e sağ tıklanır. Add Sources... sekmesine tıklanır. Açılan pencerede aynı bir tasarım dosyası ekler gibi ilerlenir ve boş bir testbench dosyası elde edilir. Bu dosya, sim_1'in içerisine gelecektir.



Doğrulama Yaklaşımları

- Eklenen yeni dosyanın içerisine testbench kodları yazılır.



The screenshot displays the Xilinx Vivado IDE interface. On the left, the 'Sources' window shows a project hierarchy with 'Design Sources (1)' containing 'ornekRTL (ornekRTL.v)', 'Simulation Sources (1)' containing 'sim_1 (1)' which includes 'testbench (testbench.v) (1)', and 'Utility Sources'. The 'testbench (testbench.v) (1)' file is selected. On the right, the 'testbench.v' code editor shows the following Verilog code:

```
1 `timescale 1ns / 1ps
2
3 module testbench();
4     reg a, b, c;
5     wire y;
6
7     ornekRTL DUT (.a(a), .b(b), .c(c), .y(y) );
8
9     initial begin
10         a = 0;
11         b = 0;
12         c = 0;
13         #10;
14         c = 1;
15         #10;
16         b = 1;
```

Doğrulama Yaklaşımları

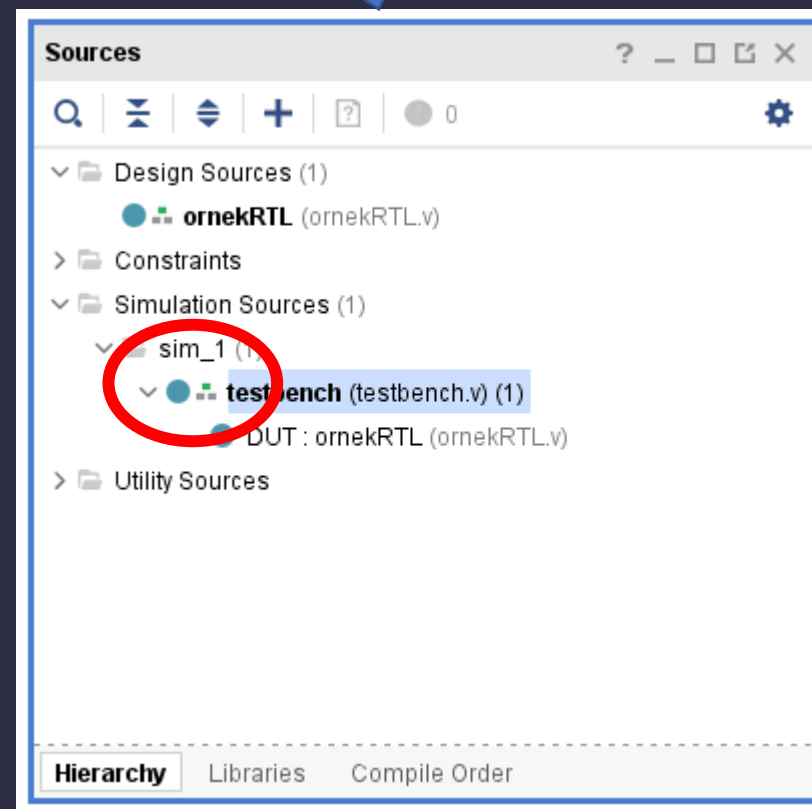
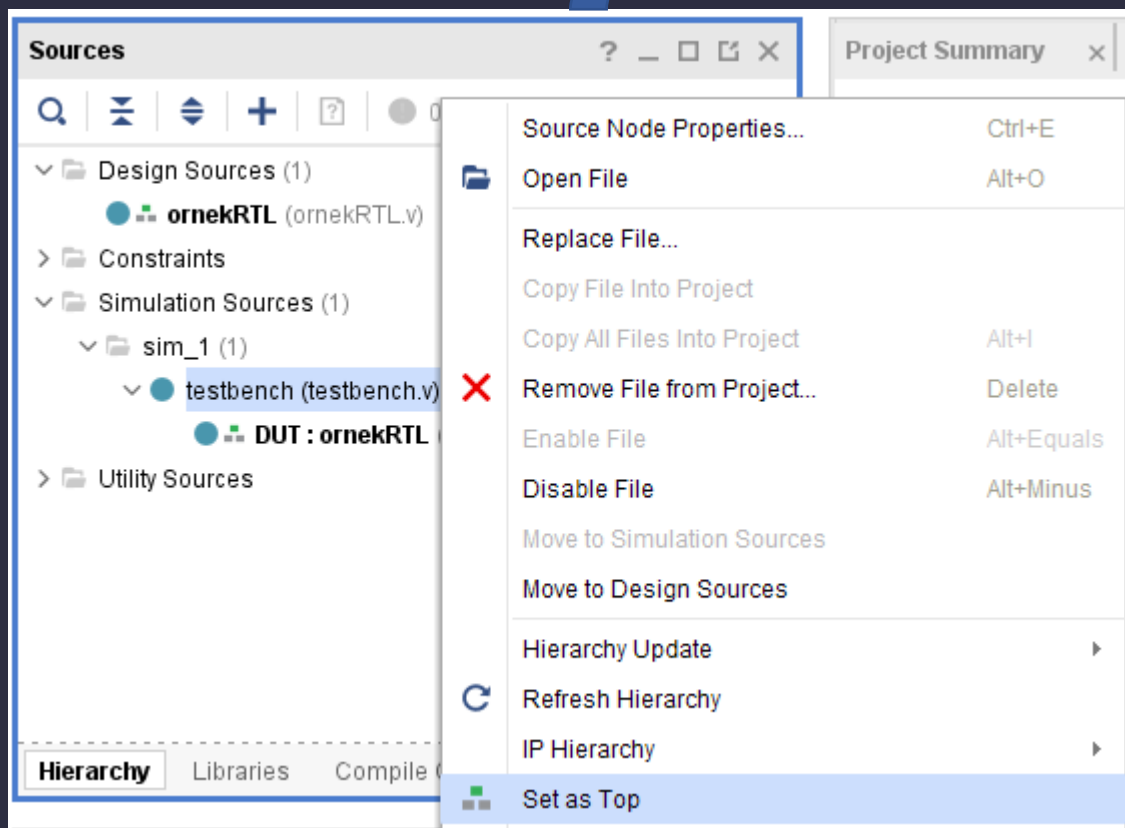
- Bu aşamada dikkat edilmesi gereken bir nokta vardır. Aynı tasarım dosyalarında olduğu gibi simulasyon dosyalarında da tepe modül kavramı vardır. Bir tasarımı simule etmek için kullanılacak simulasyon kodlarının tepe modülü vivado'da belirtilmelidir.

Doğrulama Yaklaşımları

- Vivado'da önce tasarım dosyası eklendiyse, ilk başta bu dosya hem tasarım hem de simulasyon klasörüne eklenmekte ve her iki durum için tepe modül otomatik olarak belirlenmektedir.
- Yani bu durumda simulasyon tepe modülü olarak yeni tasarlanan test modülü seçilmelidir.

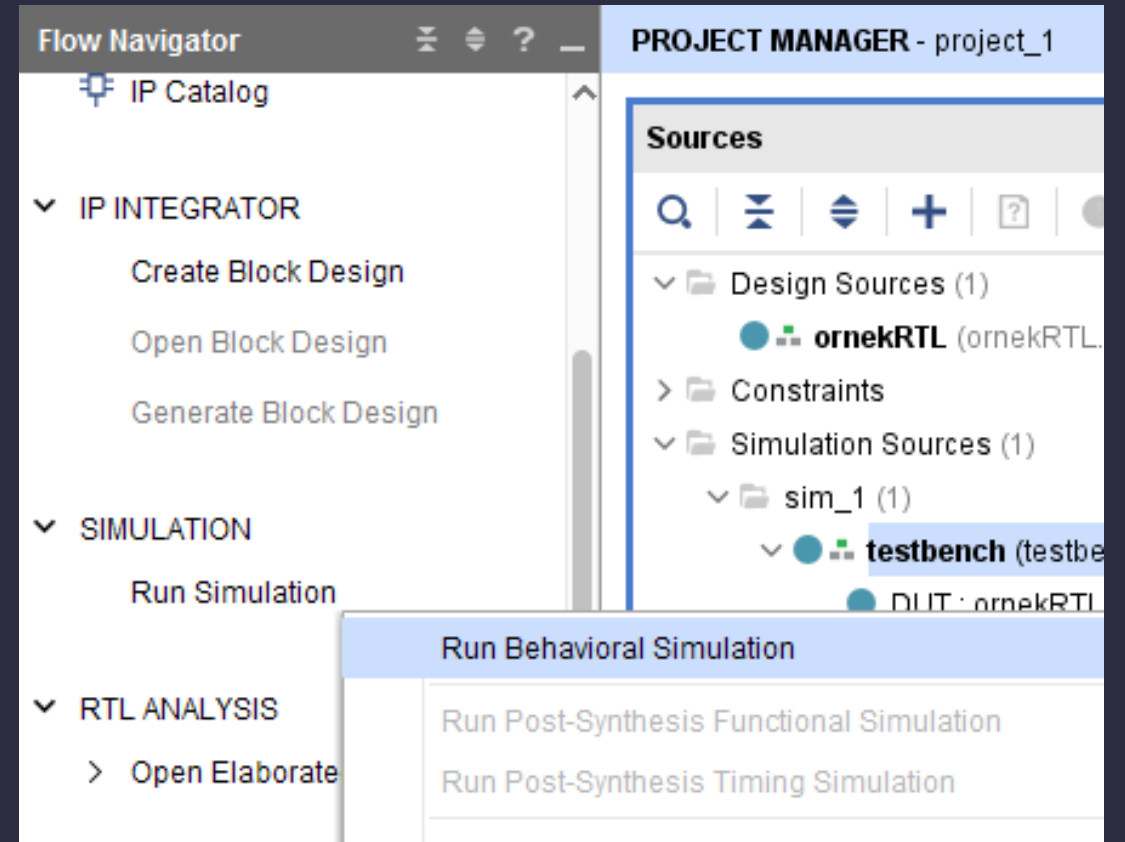
Doğrulama Yaklaşımları

- Simulasyon tepe modülünün değiştirilmesi



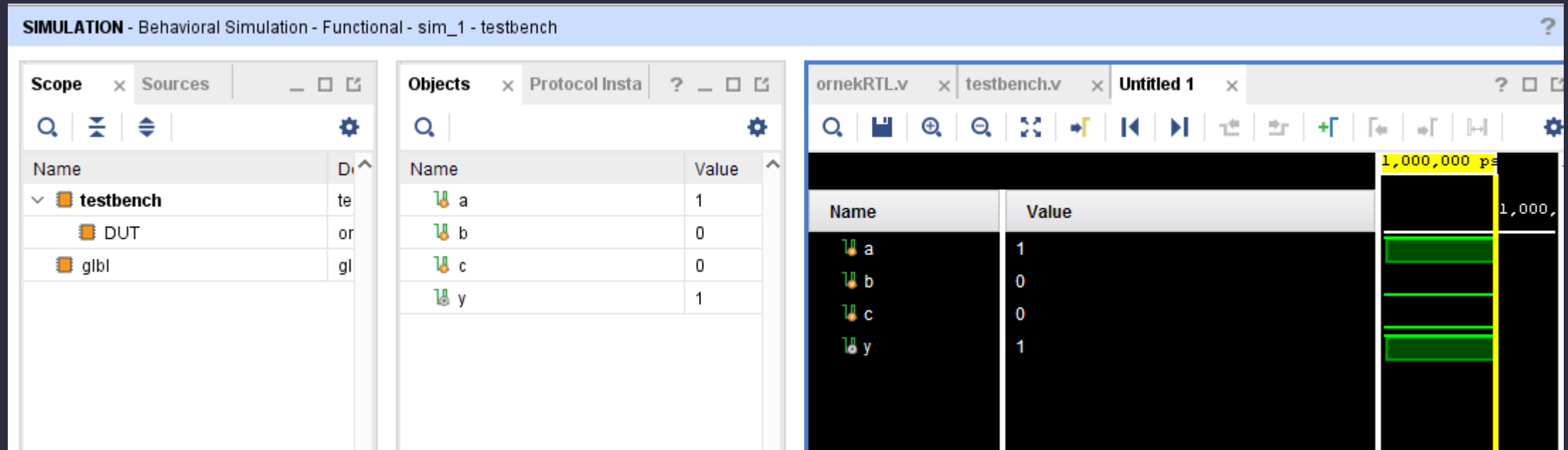
Doğrulama Yaklaşımları

- Bu aşamadan sonra simulasyon başlatılabilir. Bunun için simulasyon penceresinden "Run Behavioral Simulation" 'a basılır.



Doğrulama Yaklaşımları

- Simulasyon ilk açılışta aşağıdaki görselde verilen pencereler ekrana gelecektir.

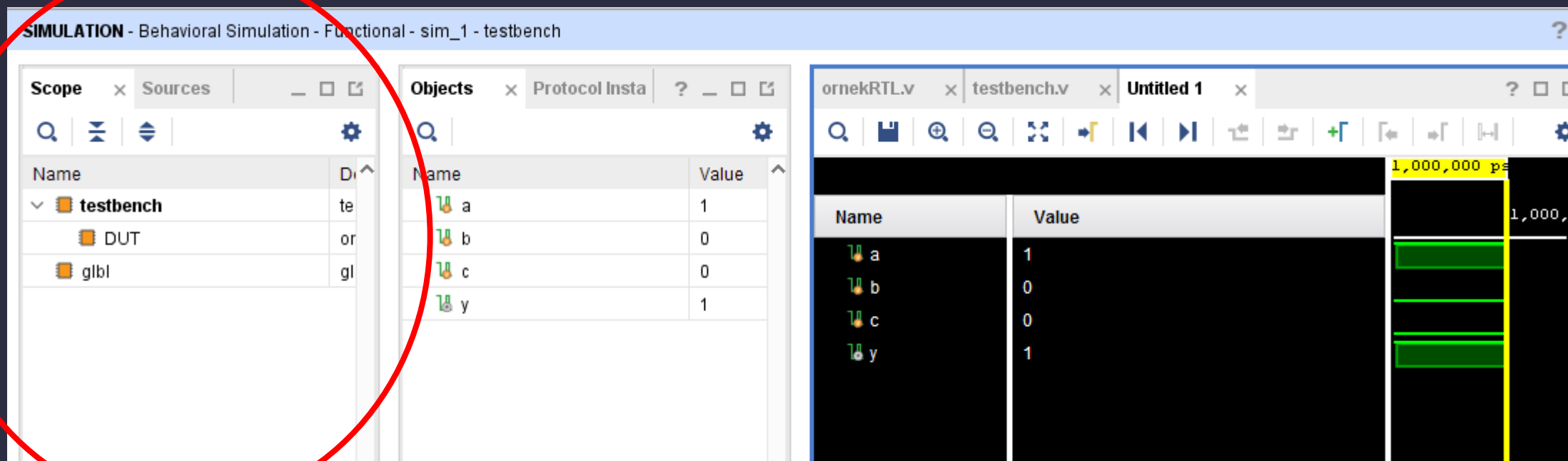


The screenshot displays a behavioral simulation environment with three main panels:

- Scope Panel:** Shows a hierarchical tree of simulation objects. The root is 'testbench', which contains 'DUT' and 'glbl'.
- Objects Panel:** Displays a table of current object values.
- Waveform Viewer:** Shows a signal trace for variables 'a', 'b', 'c', and 'y' over time. A vertical yellow line indicates the current simulation time at 1,000,000 ps.

Name	Value
a	1
b	0
c	0
y	1

Doğrulama Yaklaşımları



The screenshot displays a behavioral simulation window titled "SIMULATION - Behavioral Simulation - Functional - sim_1 - testbench". It features two main panels: "Scope" and "Objects".

Scope Panel: This panel is circled in red. It shows a tree view of the simulation hierarchy. The root is "testbench", which contains "DUT" and "gbl".

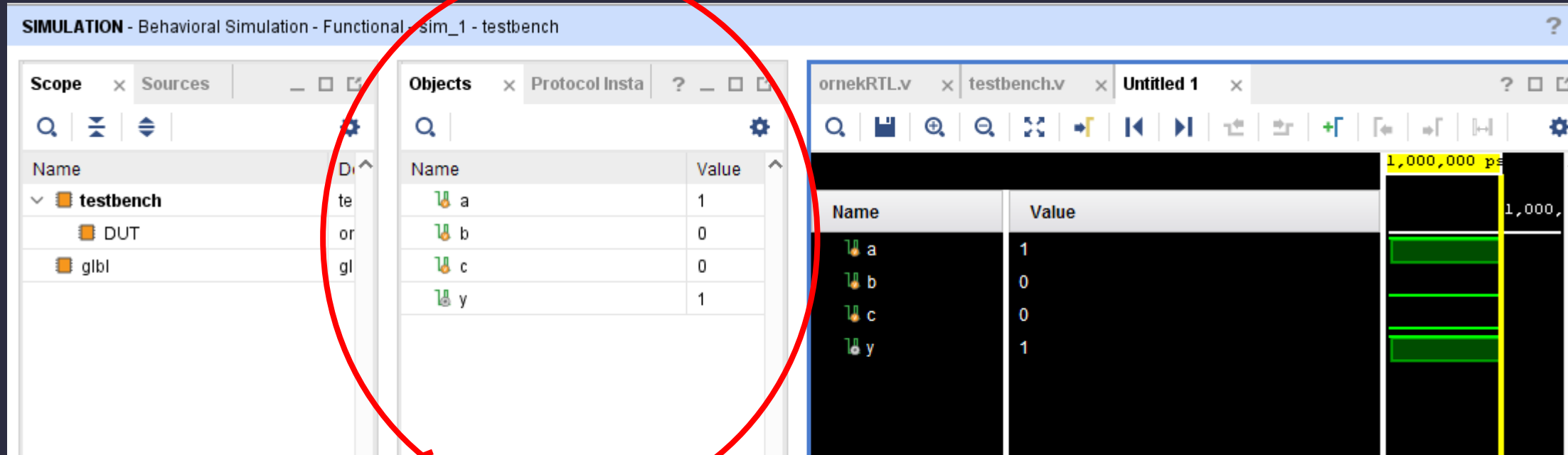
Objects Panel: This panel shows the internal signals of the selected module. It contains a table with the following data:

Name	Value
a	1
b	0
c	0
y	1

Waveform Panel: On the right, a waveform viewer shows the signal values over time. The signals a, b, c, and y are visible, with a yellow vertical line indicating the current time point at 1,000,000 ps.

Scope penceresinde, tasarımdaki modüller görülmektedir. Bu modüllerden hangisine tıklanırsa, yan taraftaki objects kısmında o modülün iç sinyalleri listelenecektir.

Doğrulama Yaklaşımları



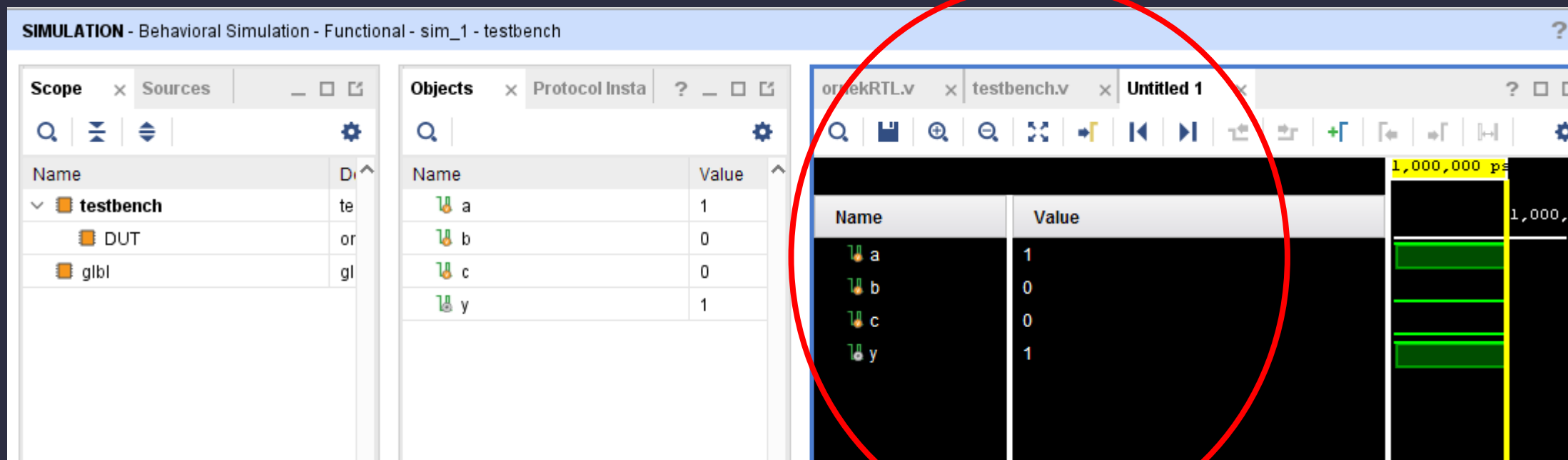
The screenshot shows a simulation environment with two main windows. The 'Objects' window on the left lists signals and their values:

Name	Value
a	1
b	0
c	0
y	1

The 'Waveform' window on the right shows the signal values over time. The signals are a, b, c, and y. The values are 1, 0, 0, and 1 respectively. The waveform is displayed on a black background with green horizontal lines representing the signal levels. A yellow vertical line is positioned at 1,000,000 ps.

Objects penceresindeki sinyallerden hangileri testbench ortamında değişimleri gözlemlenmek isteniyorsa, o sinyallere sağ tıklanarak add to wave window denir.

Doğrulama Yaklaşımları



The screenshot shows the ISIM simulation tool interface. The 'Scope' window displays a tree view of the simulation hierarchy, including 'testbench', 'DUT', and 'glbl'. The 'Objects' window shows a table of objects with their values:

Name	Value
a	1
b	0
c	0
y	1

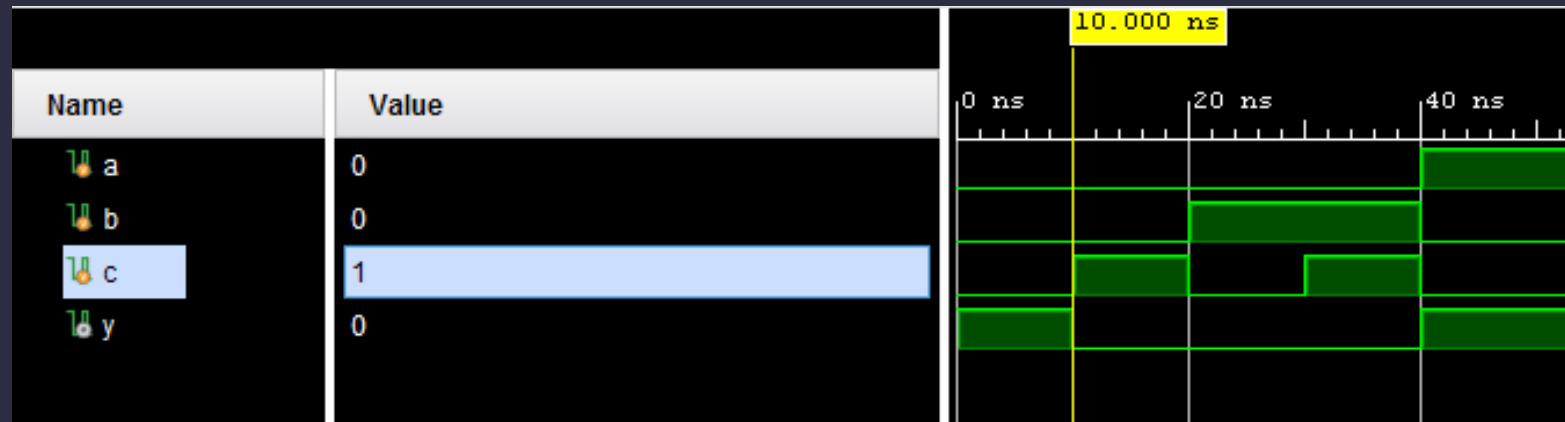
The 'Waveform' window shows a table of signals and their values, with a red circle highlighting the table and a red arrow pointing to the text below:

Name	Value
a	1
b	0
c	0
y	1

Waveform'da eklenmiş olan sinyaller (simulasyon açıldığında ISIM aracı default olarak testbench tepe modülündeki sinyalleri ekler) görülmektedir.

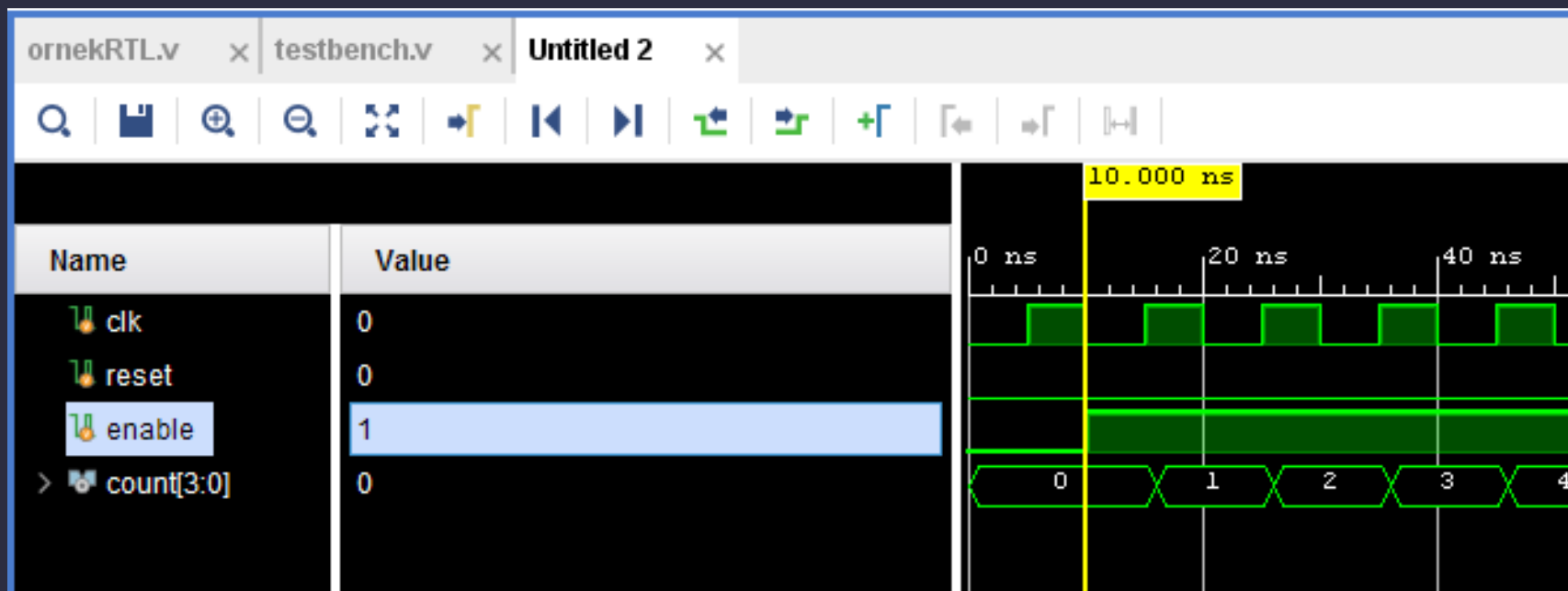
Doğrulama Yaklaşımları

- Simulasyon başladığında waveform çok fazla zoom-in yapılmış durumda başlatılmaktadır. Waveform'a sağ tıklayarak zoom-out'a veya ctrl+ mouse orta teker'i geriye çevirerek zoom-out sağlanabilir.
- Zoom-out yapılır simulasyonun başına gidildiğinde aşağıdaki figür'deki görsel görülebilir.



Doğrulama Yaklaşımları

- Counter tasarımı simülasyon çıktısı

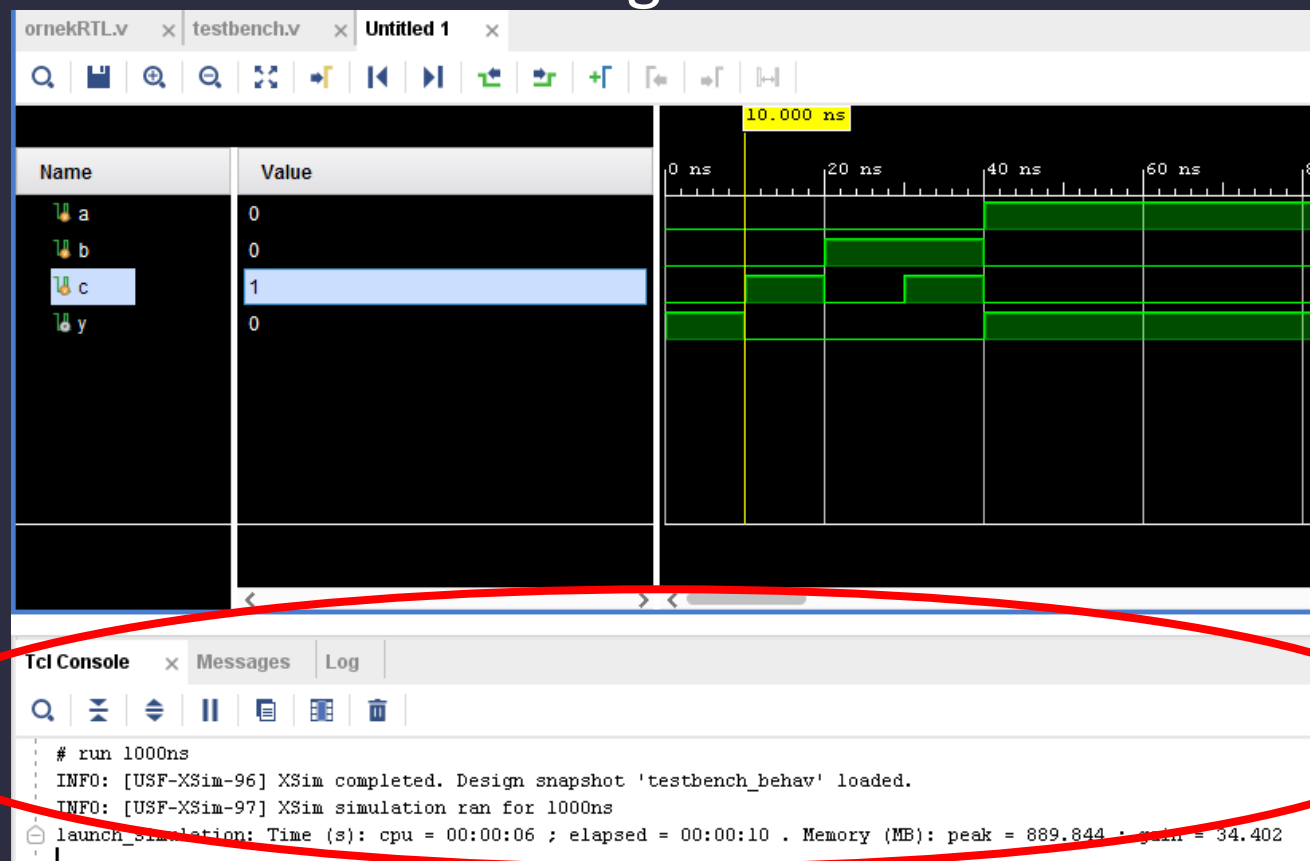


Doğrulama Yaklaşımları

- Simulasyon aracı ile modülün üretmesi beklenen sonuçları ve ara sonuçların (örneğin, dışarı çıkmayan saklayıcı değerleri) doğru üretip üretmediği kontrol gözlemlenebilir.

Doğrulama Yaklaşımları

- Simulasyonun içerisinde çeşitli ekrana bastırma komutları kullanıldı ise, aşağıdaki TCL konsol bölümünden gözlemlenebilir.



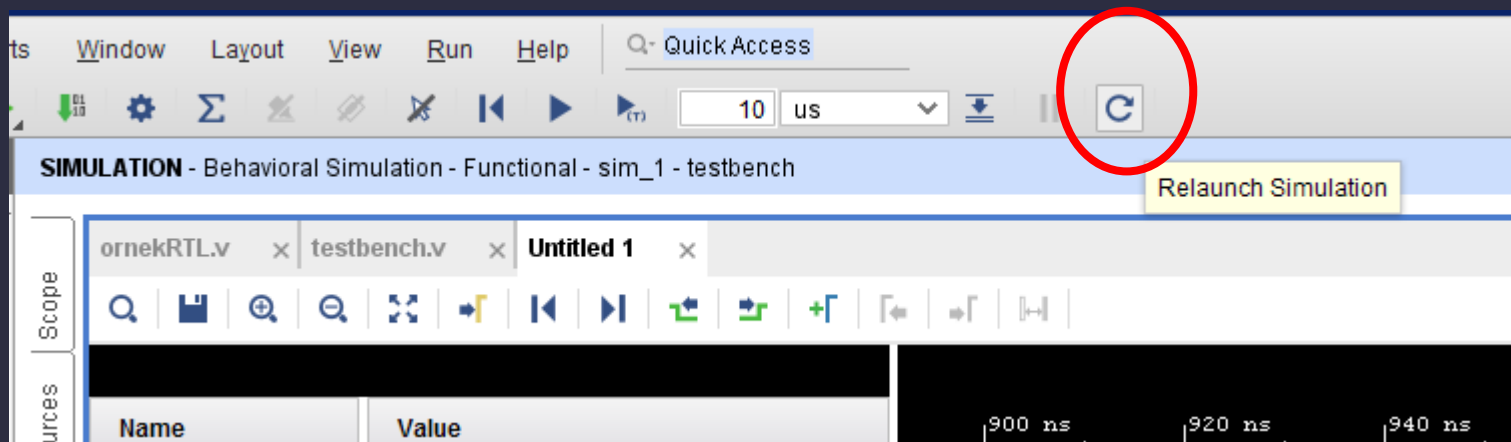
Name	Value
a	0
b	0
c	1
y	0

Tcl Console

```
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:06 ; elapsed = 00:00:10 . Memory (MB): peak = 889.844 ; gain = 34.402
```

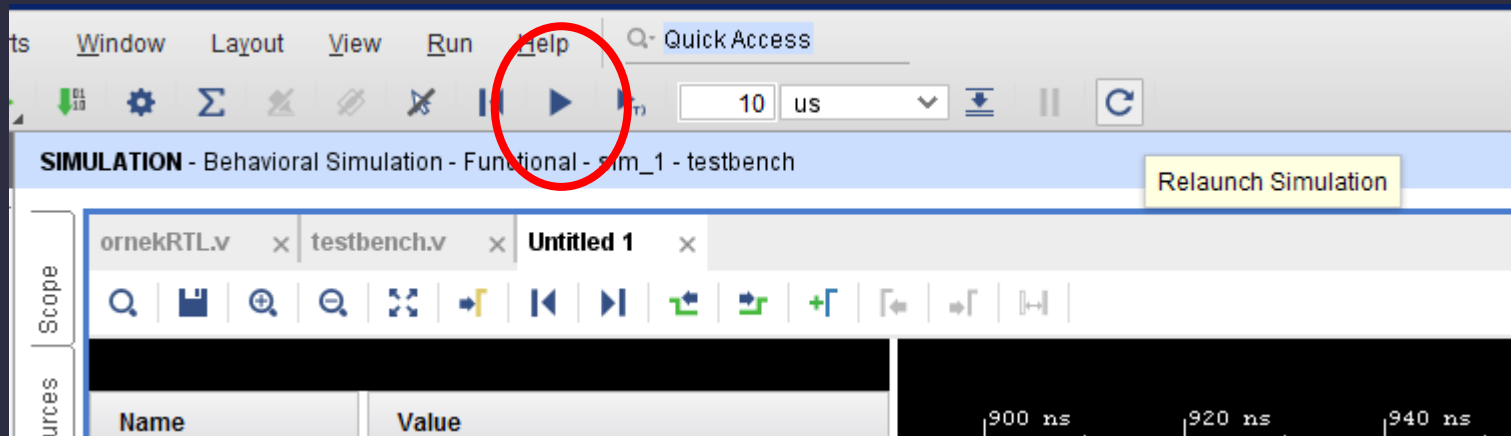
Doğrulama Yaklaşımları

- Tasarımda bir değişiklik yapıldığında tekrar simulasyon yapmak için aşağıda gösterilen tuş kullanılabilir.



Doğrulama Yaklaşımları

- Simulasyon başlatıldığında default olarak 10 mikro saniye çalışıp durmaktadır. Simulasyon devam ettirilmek istenirse, aşağıda gösterilen play tuşuna basılabilir.



Doğrulama Yaklaşımları

- Simulasyon aracında sinyallerin değeri X olarak gözüküyorsa, o sinyalin başlangıç ataması yapılmamış demektir. Ne olduğu belli olmayan bir durumdan başlamıştır.

Doğrulama Yaklaşımları

- Simulasyon üzerinde doğrulamanın temel zorluğu, çok fazla denenebilecek kombinasyonda giriş olabilir.
- Örneğin iki 32 bitlik sayının toplamını yapan devre için, 2^{64} farklı giriş beslenebilir. Tüm olası girişleri denemek yıllar sürebilir.
- Dolayısıyla her durumu denemek yerine, kritik girişler besleyerek test edilmelidir.

Doğrulama Yaklaşımları

- Genellikle bir algoritmanın çip tasarımı karşılığı yapılması istendiğinde, bu algoritmalar öncelikle C, C++, Matlab gibi dillerde kodlanır.
- Bu dillerde kodlanmış koda, girişler beslenir ve kodun ürettiği çıkışlar ve giriş olarak beslenen değerler dosyalara yazdırılır.
- Buna "Golden Model" denir. Bu dosyalar, tasarlanan modülün testbench dosyalarında giriş vermek ve üretilen çıkışları kontrol etmek için kullanılır.

Doğrulama Yaklaşımları

- Tasarımınızın testbench simulasyon üzerinde çalışıyor olması, FPGA'e konfigüre ettiğinizde kesinlikle çalışacağı anlamına gelmez.
- Genellikle latch veya tutturulamayan frekans hataları gibi nedenlerden ötürü simulasyonda yakalanamayan ancak kart üzerinde karşılaşılan sorunlar görülmektedir.