Günümüzde, HDL (Harware Description Language- donanım tanımlama dili) kullanımının birkaç temel sebebi vardır:

* Text ortamında tasarımları gerçekleştirebilmek ve değiştirebilmek.
* Devre tasarımı yapmadan gerçek ortamda benzetimleri yapmak.
* Verilog ortamında özel araçlar ile verilog kodunu otomatik olarak devre gerçekleme boyutuna çevirir.

Verilog ve VHDL olarak iki standart HLD dili vardır. Genellikle verilog yazım formatı C diline oldukça benzediği için çok tercih edilen ve kullanılan bir dildir. Yapısal (structural) ve davranışsal (behavioral) olmak üzere iki farklı şekilde verilog kodları hazırlanabilir. Yapısal tasarımlar tamamen devre elemanları boyutunda olup sentezlenmeme gibi problemlerle karşılaşılmaz. Davranışsal tasarımlar ise devre davranışlarının sözel yapılarda olduğu gibi tanımlanmasından ibarettir. Fakat davranışsal tasarımlarda yazılan kodların bazen sentezlenmeme gibi problemi olabilir.

Verilog kodları bir module olarak adlandırılan module ile başlayıp endmodule ile biten komutlar arasında yazılır. Modüle giriş ve çıkış olan değişkenler belirtilir.

module module\_adı(duyarlılık listesi-giriş çıkış değişkenleri)

…

Bildirimleri

…

endmodule

**Sayılar**

**<boyut><taban\_formatı><sayı>**

**<boyut>** basamak (digit) sayını belirler.

**<taban\_formatı>** Sayının tabanını belirler. ’ ile başlar. H(hexadecimal), b(binary), o(octal) ve d (decimal) sayıları belirler. Hexadecimal sayılarda a-f harfleri büyük küçük olabilir.

**<sayı>** uygun formatta sayının değerini belirler.

659 // onlu sayı

20’h 837FF // hexadecimal sayı

4’b1001 // 4-bit ikili sayı

5 ’D3 // 5-bit onlu sayı

3’b01x // 3-bit ikili sayı en anlamsız bit bilinmiyor

12’hx // 12-bit binmeyen bir sayı

16’hz // 16-bit yüksek-empedans sayı

8 ’d -6 // Geçersiz bir bildirim

-8 ’d 6 // Bu 6’nın 2’e tümleyenidir.

**String Değişkeni**

reg [8\*12:1] stringvar; // Değişken 8 bitlik 12 karakter

initial

begin

stringvar = “Hello world!”;

end

module string\_test;

reg [8\*14:1] stringvar;

initial

begin

stringvar = “Hello world”;

$display(“%s is stored as %h”,stringvar,stringvar);

stringvar = {stringvar,”!!!”}; //Stringe değere ekleme

$display(“%s is stored as %h”,stringvar,stringvar);

end

endmodule

**Özel String Karakterleri**

\n Yeni satıra geç

\t Tab karakteri

\\ Slash

\” Çift tırnak karakteri

**Tanımlamalar, Anahtar kelimeler ve Sistem isimleri**

Tanımlamalarda ilk değerin ir harf ya da alt çizgi (\_) olması gerekmektedir. Rakam ya da $ işareti ile başlanmamalıdır.

shiftreg\_a

busa\_index

error\_condition

merge\_ab

\_bus3

n$657

**Veri Tipleri**

0 Lojik-0 ya da yanlışı temsil eder.

1 Lojik-1 ya da doğruyu temsil eder.

x Belirsiz bir lojik değeri temsil eder.

z Yüksek-empedans durumunu temsil eder.

**İşaretli Nesneler**

wire signed [3:0] signed\_wire; // aralık -8 <-> +7

reg signed [3:0] signed\_reg; // aralık -8 <-> +7

reg signed [3:0] signed\_mem [99:0] //100 kelime aralık -8<->+7

function signed [3:0] signed\_func; // aralık -8 <-> +7

**Kayıtçılar**

reg a; // kayıtçı

reg[3:0] v; // 4-bit vektör kayıtçı en ağırlık bitten en düşüğe

//doğru

tri [15:0] busa; // tri-state 16-bit bus

reg [1:4] b; // a 4-bit vektör register

reg signed [0:3] signed\_reg; // 4-bit işaretli kayıtçı -8 ‘den +7’ye

reg signed [0:3] signed\_mem [99:0] // 100 words with a range of -8 to +7

wire w1, w2; // 2 wire

reg [4:0] x, y, z; // 3 tane 5-bit register

**ALWAYS**

Always bildirimi içindeki değişkenlerin belirlenen değişimlerine göre sürekli olarak arada kalan bildirimlerin gerçekleştirildiği bildirimdir.

always @( )

begin

…

Bildirimleri

…

end

always blocking ve non-blocking olarak iki şekilde tanımlanabilir.

A=10

Blocking non-blocking

A=A+1 (A=11) A<=A+1 (A=11)

B=A (B=11) B<=A (B=10)

**Not:** always bildirimi içinde wire değişkeni kullanılamaz. Sadece reg ve int değişkenleri tanımlanabilir. Always duyarlılık listesi içinde hem level değişimli hem de kenar değişimli işaretler aynı anda değerlendirilmez

**FOR Döngüsü**

module for\_loop\_example(data\_in,data\_out);

/\* bu modüle bitlerin MSB-LSB sıralamasını değiştirir. \*/

input [7:0] data\_in;

output [7:0] data\_out;

reg [7:0] data\_out;

integer i; /\* reg [2:0] olarak da tanımlanabilir fakat reg işaretsiz iken int işaretlidir. \*/

always @ (data\_in)

begin

for (i=0; i<=7; i=i+1)

data\_out[7-i] = data\_in[i];

end

endmodule

**Case Bildirimi**

case duyarlılık listesi (S) içinde hangi durum gerçekleşmiş ise sadece o durum gerçeklenir.

case (S)

durum1:…

durum2:…

endcase

**if bildirimi**

Farklı kullanım şekilleri vardır.

Eğer bildirim tek satır ise aşağıdaki gibi

if (mantıksal işlem) bildirim

Bildirim birden fazla satır ise aşağıdaki gibi olur. Eğer bildirim işlemleri birden fazla satır ise her bir bildirim begin ve end ile etiketlenip aralarına yazılmalıdır.

if (mantıksal işlem)

begin

…

Bildirimler

…

end

if (mantıksal işlem) bildirim

else bildirim

if (mantıksal işlem) bildirim

else if (mantıksal işlem) bildirim

else

**Ayrılmış Değişken İsimleri**

and always assign attribute

begin buf bufif0 bufif1

case cmos deassign default

defparam disable else endattribute

end endcase endfunction endprimitive

endmodule endtable endtask event

for force forever fork

function highz0 highz1 if

initial inout input integer

join large medium module

nand negedge nor not

notif0 notif1 nmos or

output parameter pmos posedge

primitive pulldown pullup pull0

pull1 rcmos reg release

repeat rnmos rpmos rtran

rtranif0 rtranif1 scalared small

specify specparam strong0 strong1

supply0 supply1 table task

tran tranif0 tranif1 time

tri triand trior trireg

tri0 tri1 vectored wait

wand weak0 weak1 while

wire wor

**İlişki operatörleri**

< küçük ise

> büyük ise

<= küçük eşit ise

>= büyük eşit ise

== eşit ise

**Matematiksel operatörler**

\* çarpma (tam sayı)

+ toplama

/ bölme(tam sayı)

- çıkartma

% amodb=a%b a mod(b)

**Mantıksal operatörler**

&& VE

|| VEYA

! DEĞİL

**Eşitlik ve tanımlama operatörleri**

= atama

!= eşit değil

=== bire bir eşitlik

!== bire bir eşitsizlik

**Bit işlemleri**

+ İkili toplama

- İkili çıkarma

& VE

| VEYA

^ XOR

~& NAND

~| NOR

~^ EXOR

~ Bit NOT

**Öteleme operatörleri**

<< Sola öteleme

>> sağa öteleme

?: c=sel? a:b eğer sel doğru ise c=a

değilse c=b

{} {co, sum}=a+b+c a,b ve c toplanır taşma

varsa co’ya atanır, toplam sum’a

{{}} b={3{a}} = {a,a,a}

**Sonlu Durum Makineleri**



**Mealy makinesi örneği**

module seq3\_detect\_mealy(x,clk, y);

input x, clk;

output y;

reg y;

reg [1:0] pstate, nstate;

parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;

always @(x or pstate)

case (pstate)

S0: if (x) begin nstate = S1; y = 0; end

else begin nstate = S0; y = 0; end

S1: if (x) begin nstate = S2; y = 0; end

else begin nstate = S0; y = 0; end

S2: if (x) begin nstate = S3; y = 0; end

else begin nstate = S0; y = 0; end

S3: if (x) begin nstate = S3; y = 1; end

else begin nstate = S0; y = 0; end

endcase

always @(posedge clk)

pstate <= nstate;

endmodule

Moore Makinesi örneği



module seq3\_detect\_moore(x, clk, y);

input x, clk;

output y;

reg [1:0] state;

parameter S0=2'b00, S1=2'b01, S2=2'b10,

S3=2'b11;

always @(posedge clk)

case (state)

S0: if (x) state <= S1;

else state <= S0;

S1: if (x) state <= S2;

else state <= S0;

S2: if (x) state <= S3;

else state <= S0;

S3: if (x) state <= S3;

else state <= S0;

endcase

assign y = (state == S3);

endmodule

**Örnek:** 4 bit genlik karşılaştırıcı

module compare (A, B, AeqB, AgtB, AltB);

input [3:0] A, B;

output AeqB, AgtB, AltB;

reg AeqB, AgtB, AltB;

always @(A or B)

begin

AeqB = 0;

AgtB = 0;

AltB = 0;

if(A == B) AeqB = 1;

else if (A > B) AgtB = 1;

else AltB = 1;

end

endmodule

**Örnek:** F'=(A+B+C)(A'+B'+C) fonksiyonuna ait devrenin Verilog kodu

module devre (F, Fn, A, B, C);

input A, B, C;

output F, Fn;

always @(A or B or C)

begin

Fn=(A~|B~|C)~|(~A~|~B~|C);

F=(Fn~|Fn);

end

endmodule

**Örnek:** Frekans bölme modülü

module frequency(clk,clk\_out);

input clk;

output clk\_out;

reg clk\_out\_hz = 0;

reg [27:0] counter = 28'h0000000;

//50 MHze gore hesaplanmıştır.

always @(posedge clk)

begin

if (counter==28'h4C4B40)//50 MHze gore hesaplanmıştır. 1 sn 50.000.000 'nun hex karsiligi

begin

counter = 28'h0000000;

clk\_out\_hz =~ clk\_out\_hz;

end

else counter = counter + 1;

end

assign clk\_out = clk\_out\_hz;

endmodule

|  |
| --- |
| **Örnek:** seven segment displaye yazdırma yazdırılacak değer fark adlı 4 bitlik değişkende |
| localparam N = 18; |
|  |
| reg[N-1:0]count1; |
|  |
| always@ (posedge clk) |
| begin |
| count1= count1 + 1; |
|  |
| end |
|  |
| reg[3:0]sseg; |
| reg[3:0]an\_temp; |
|  |
| always @ ( \* ) |
| begin |
| case(count1[N-1:N-2]) |
|  |
| 2'b00: |
| begin |
| sseg = fark[3:0]; |
| an\_temp = 4'b1110; |
| end |
|  |
| 2'b01: |
| begin |
| if (neg==0 & fark[3:0]!=0) sseg = 7'b0111111; //dash; |
| else sseg=7'b1000000; |
| an\_temp = 4'b1101; |
| end |
|  |
| 2'b10: |
| begin |
| sseg = 7'b0111111; //dash //unknown sent to produce '-' |
| an\_temp = 4'b1111; |
| end |
|  |
| 2'b11: |
| begin |
| sseg = 7'b0111111; //unknown sent toproduce '-' |
| an\_temp = 4'b1111; |
| end |
| endcase |
| end |
|  |
| assign an = an\_temp; |
|  |
| reg[6:0] sseg\_temp; |
| always @ ( \* ) |
| begin |
| case(sseg) |
| 4'd0: sseg\_temp = 7'b1000000; //0 |
| 4'd1: sseg\_temp = 7'b1111001; //1 |
| 4'd2: sseg\_temp = 7'b0100100; //2 |
| 4'd3: sseg\_temp = 7'b0110000; //3 |
| 4'd4: sseg\_temp = 7'b0011001; //4 |
| 4'd5: sseg\_temp = 7'b0010010; //5 |
| 4'd6: sseg\_temp = 7'b0000010; //6 |
| 4'd7: sseg\_temp = 7'b1111000; //7 |
| 4'd8: sseg\_temp = 7'b0000000; //8 |
| 4'd9: sseg\_temp = 7'b0010000; //9 |
| default: sseg\_temp = 7'b0111111; //dash |
| endcase |
| end |
| assign leds = sseg\_temp; |
| assign dp = 1'b1; |
| endmodule |

**Örnek:** 0-9999 sayıcı (4 digit display modülü eklenecek)

module bcdcounter(clk,leds,an,reset,dp);

input clk;

input reset;

output [6:0] leds ;

output dp;

output [3:0] an;

reg [15:0] count=16'h0000;

frequency frequency(

.clk (clk),

.clk\_out (clk\_out)

);

always @(posedge clk\_out)

begin

if (reset) count=0;

else count=count+1;

if (count[3:0]==4'b1010) count=count+6;

if (count[7:4]==4'b1010) count=count+96;

if (count[11:8]==4'b1010) count=count+1536;

if (count[15:12]==4'b1010) count=0;

end

// display modülü eklenecek

endmodule

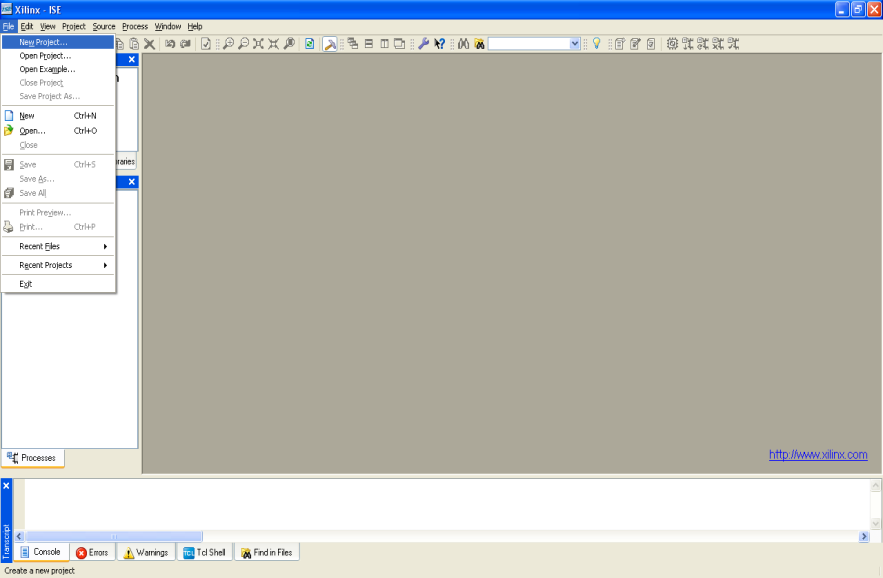
**ISE (Entegre Yazılım Ortamı-Integrated Software Environment)**

Xilinx ISE (Entegre Yazılım Ortamı-Integrated Software Environment), Xilinx FPGA’lerin üzerinde çalışılmasını sağlayan bir yazılım programıdır. Verilog veya VHDL gibi donanımsal diller kullanılarak yazılan kodlar bu program aracılığı ile sentezlenerek FPGA’e ortamına aktarılabilirler. ISE yazılımını çalıştırabilmek için program kurulduktan sonra masaüstünde Şekil 1’deki ikona tıklanarak ya da başlat menüsünden Başla → Programlar → Xilinx ISE 9.1i → Project Navigator yolu ile çalıştırılır.

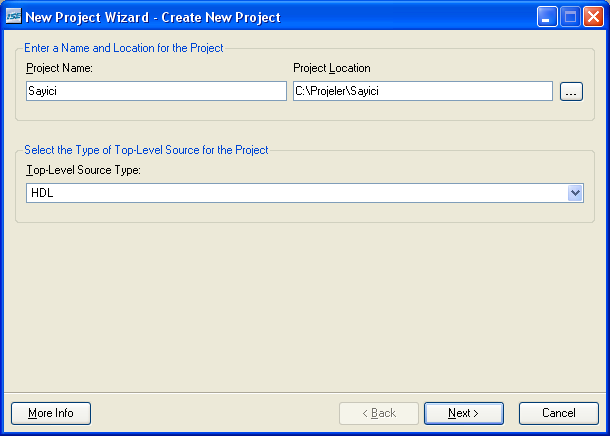


**Şekil 1.** Masaüstündeki kısayol ikonu.

Yeni bir proje oluşturmak için Şekil 2’de görüldüğü gibi File menüsünden New Project kısmı seçilir ve seçildikten sonra Şekil 3’de görülen yeni proje sihirbazı ekranı çıkar.

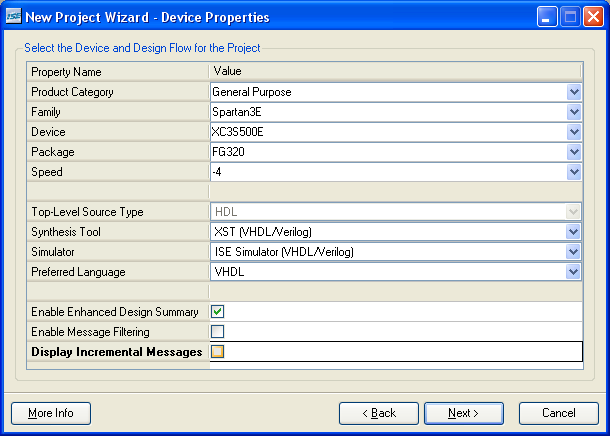


**Şekil 2.** Yeni proje dosyası açma.



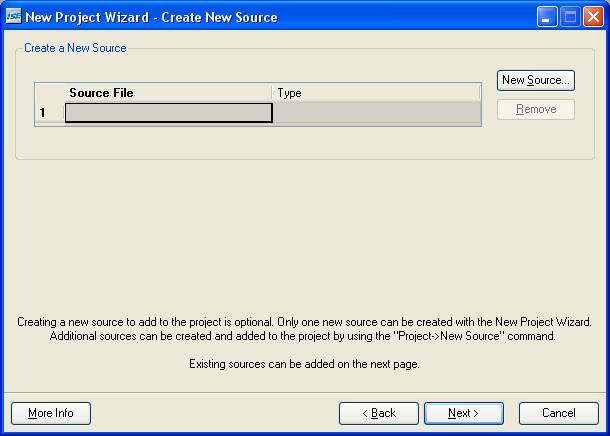
**Şekil 3.** Yeni proje dosyası hazırlama sihirbazı karşılama ekranı.

Project Name kısmına projenin adı, Project Location kısmına projenin kaydedileceği yer ve Top-Level Source Type kısmına da kullanılan kodlama dilini yazdıktan sonra next butonuna basarak sihirbazda bir sonraki aşamaya geçilir. Şekil 4’de görülen kullanılacak olan FPGA’in özelliklerinin yazıldığı aşamaya geçilir. Burada ilk kısım ürünün kategorisi olup starter kit kullanıldığı için genel amaçlı kısım seçilir. Starter kit spartan 3E olup FPGA ise XC3S500E ve paket tipi FG320’dir. Hız kısmı ise -4 değerine getirilir. Kaynak tipi HDL olup bir önceki aşamada belirtiliği için bu aşama da müdahale edilememektedir. Sentez için XST(VHDL/Verilog) seçilir simulatör olarak ISE simulatör kullanılacak ve tercih edilen dil ise verilog olarak seçilecektir. Bir sonraki adım olan yeni kaynak oluşturma adımına geçilir.

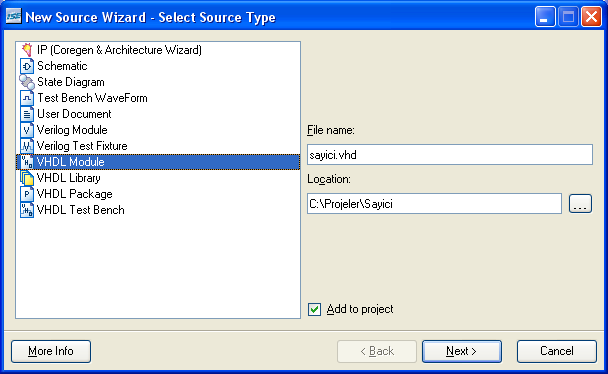


**Şekil 4.** Kullanılan cihazın özellikleri.

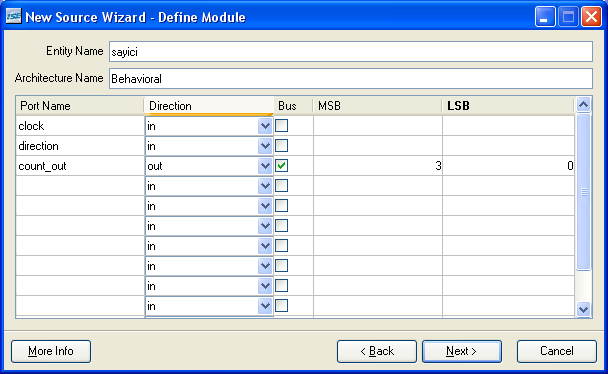
Burada tasarlanan devrenin giriş ve çıkış uçları belirlenerek isimlendirilir. Bu adım daha sonra da yapılabileceği gibi bu adımda da gerçekleştirilebilir. Şekil 5’de açılmış olan penceredeki New Source’a tıklandığında öncelikle kullanılacak olan kaynak tipinin belirlendiği Şekil 6’daki pencere açılır ve burada kaynak adı yerine de bir isim verilir. Bir sonraki adıma geçildiğinde ise giriş-çıkış pinlerinin belirlenmesi ve isimlendirilmesi yapılmaktadır. Buradaki örnekte tek giriş ve tek çıkışlı bir sistem göz önüne alınmıştır.



**Şekil 5.** Yeni kaynak oluşturma.

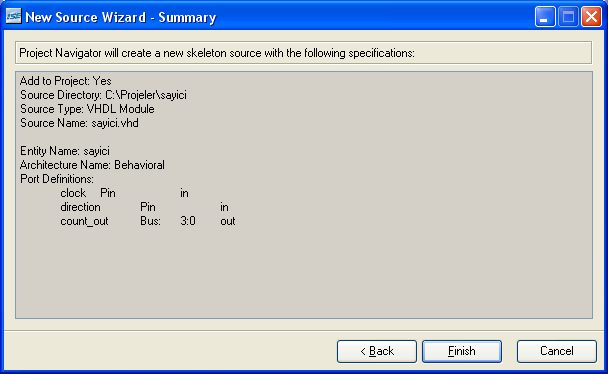


**Şekil 6.** Kaynak tipi seçme.



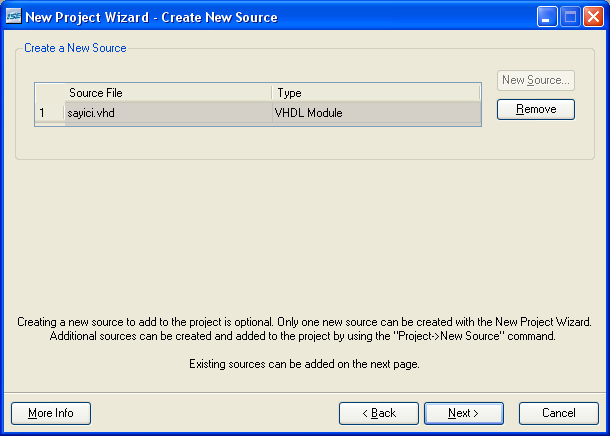
**Şekil 7.** Giriş-çıkış pinlerinin belirlenmesi.

Giriş ve çıkış pinleri belirlendikten sonra belirlenen tanımlamalar ekrana gelir ve bitir butonu ile işlem sonuçlandırılır.

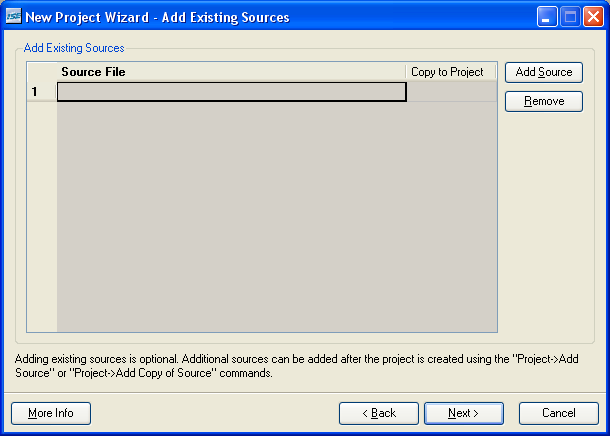


**Şekil 8.** Özet pin tablosu.

Pin tablosu görüldükten ve doğrulandıktan sonra bir sonraki adıma geçildiğinde Şekil 9’daki pencere ile karşılaşılacaktır. Burada, kaynak dosyanın adı ve uzantısı ile tipi yazmaktadır. Next butonuna basarak bir sonraki adıma geçilir. Proje hazırlama sihirbazı ile en fazla bir tane kaynak dosyası oluşturulabilir. Daha fazla oluşturulmak isteniyorsa Project → New Source ile eklenebilir.

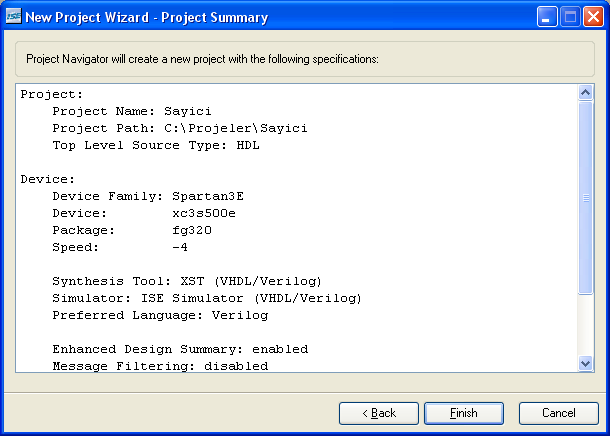


**Şekil 9.** Oluşturulmuş kaynak dosya ve tipi.



**Şekil 10.** Daha önce yazılmış kaynak dosyanın yüklenmesi.

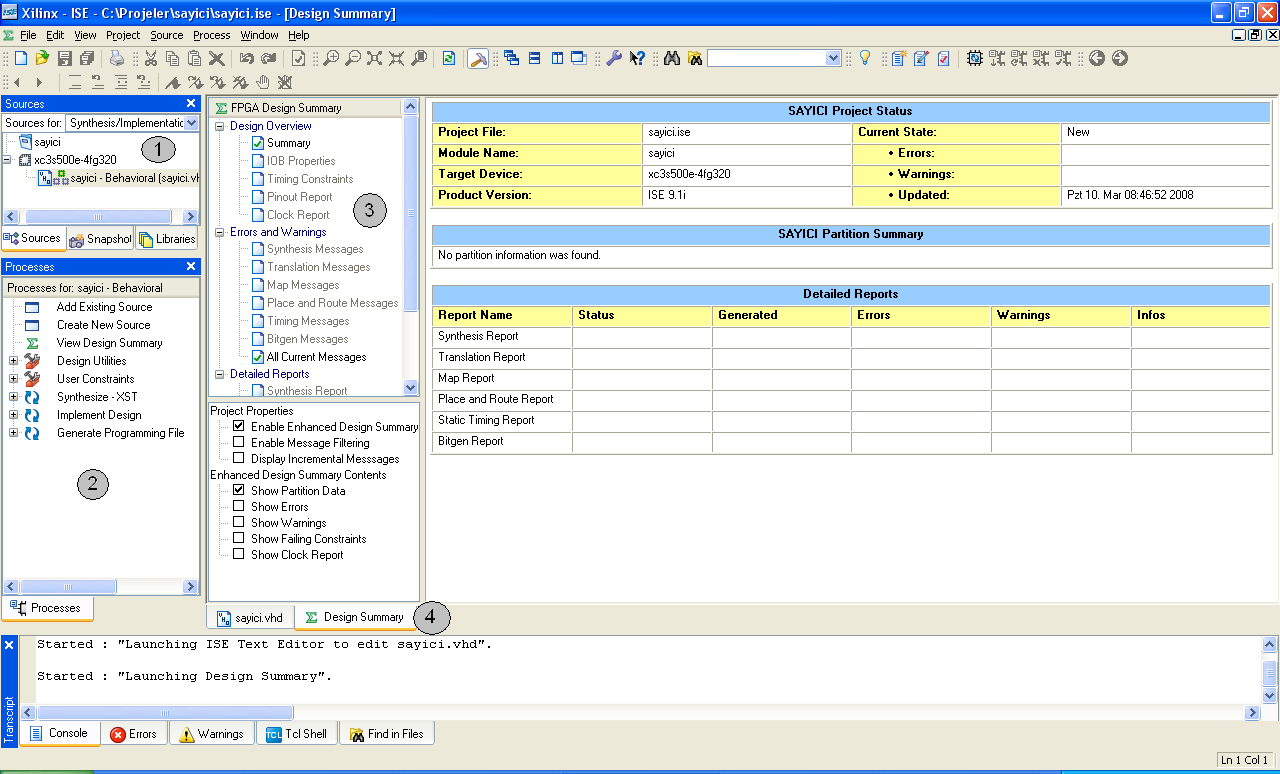
Önceden oluşturulmuş bir kaynak dosyası mevcut ise Şekil 10’da görülen karşılama ekranındaki Add Source butonuna basılarak kaynak dosya eklenebilir. Daha sonra oluşturulacaksa bu adım Next ile geçilir.



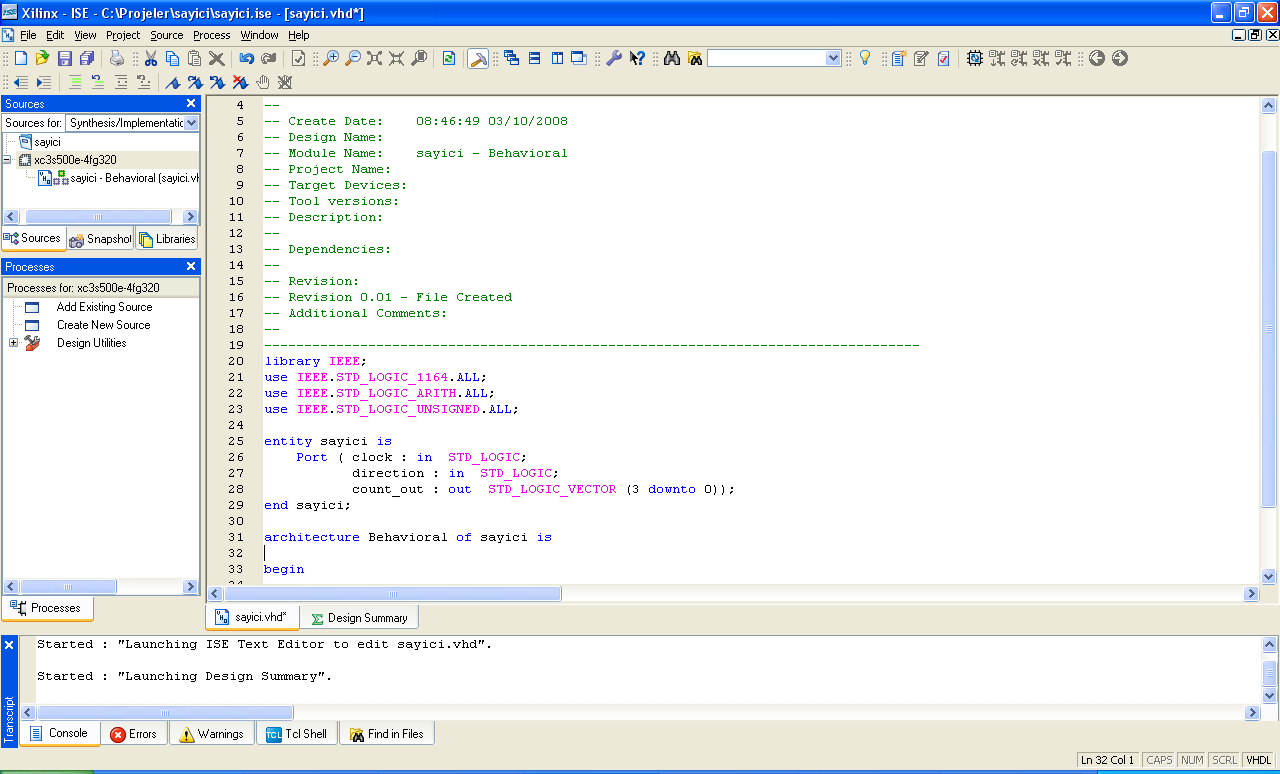
**Şekil 11.** Proje özet penceresi.

En son adımda ise Şekil 11’de görüldüğü gibi projenin bir özeti karşımıza çıkmaktadır. Eğer şimdiye kadar yapılanlardan emin değilsek Back butonu ile geriye dönüp adımları tekrar kontrol edebiliriz. Yapılanlardan emin isek, Finish butonuna basarak sihirbazı sonlandırabiliriz.

Sihirbaz sonlandıktan sonra Şekil 12’de görülen tasarım özet penceresi karşımıza çıkar. Burada  ile işaretlenmiş kısımdan, proje dosyaları ve dosyalar arasındaki bağlantılar görülebilir. xc3s500e-4fg320 seçeneğine farenin sağ tuşu ile tıklanıp özelliklerine bakıldığında ayarlar değiştirilebilir.  ile işaretlenmiş kısımda ise dosya ile ilgili işlemler yapılabilir.  işaretlenmiş kısımda ise dosya içeriği görüntülenir.  tasarım penceresi ve HDL kullanılarak yazılacak dosyalar arasında geçiş yapılabilecek kısmı gösterir. Burada, sayici.v yazan kısıma tıklandığında Şekil 13’deki pencere karşımıza çıkar.



**Şekil 12.** ISE tasarım özeti.



**Şekil 13.** Sayici.v dosyası.

Şekil 13’den görüldüğü üzere gerekli port tanımlamaları sihirbazın tamamlanması aşamasında girildiği için otomatik olarak yapılmıştır.

Architecture ve end Behavioral satırları arasına yukarıda görülen programdaki gibi sayıcının yapması gereken işlemler yazılır.

module sayici(clk, direction, count\_out);

input clk, direction;

output [3:0] countout;

reg [3:0] countout04’b0000;

always @(posedge clk)

begin

if (direction==1) countout=countout+1

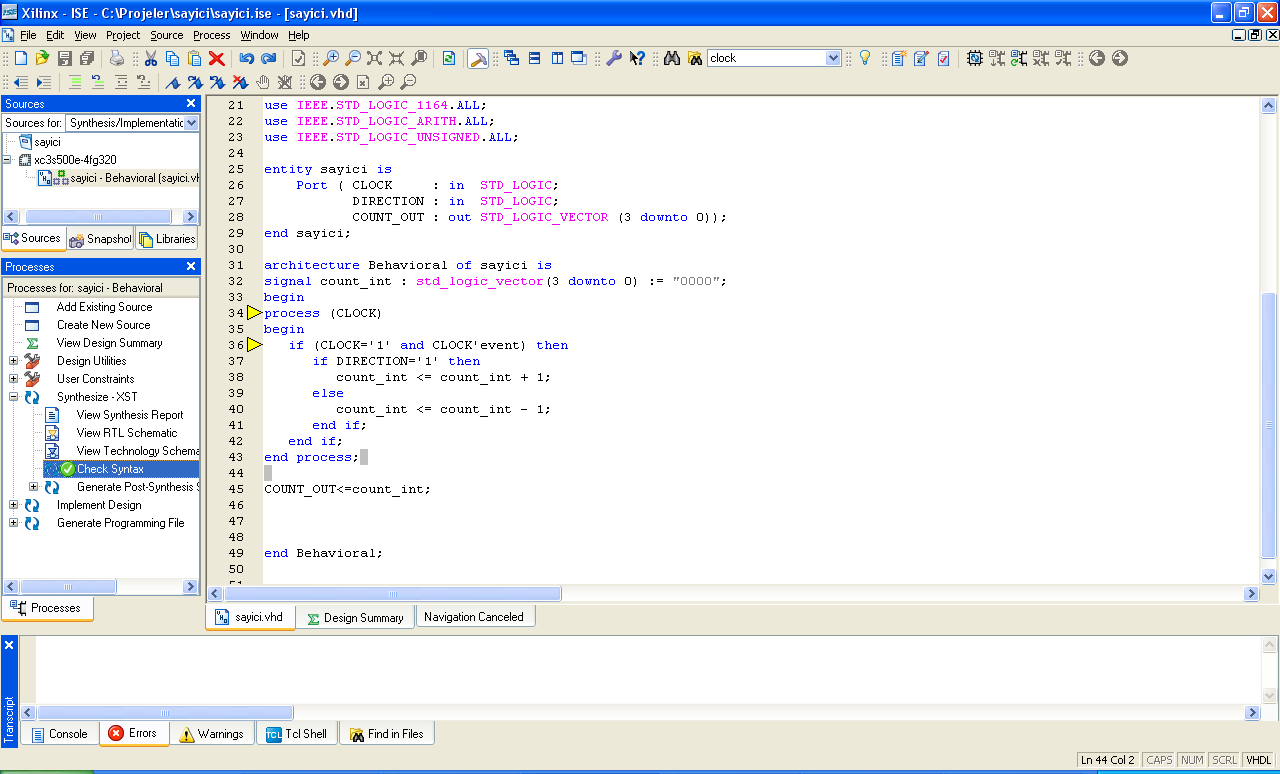
else countout=countout-1;

end

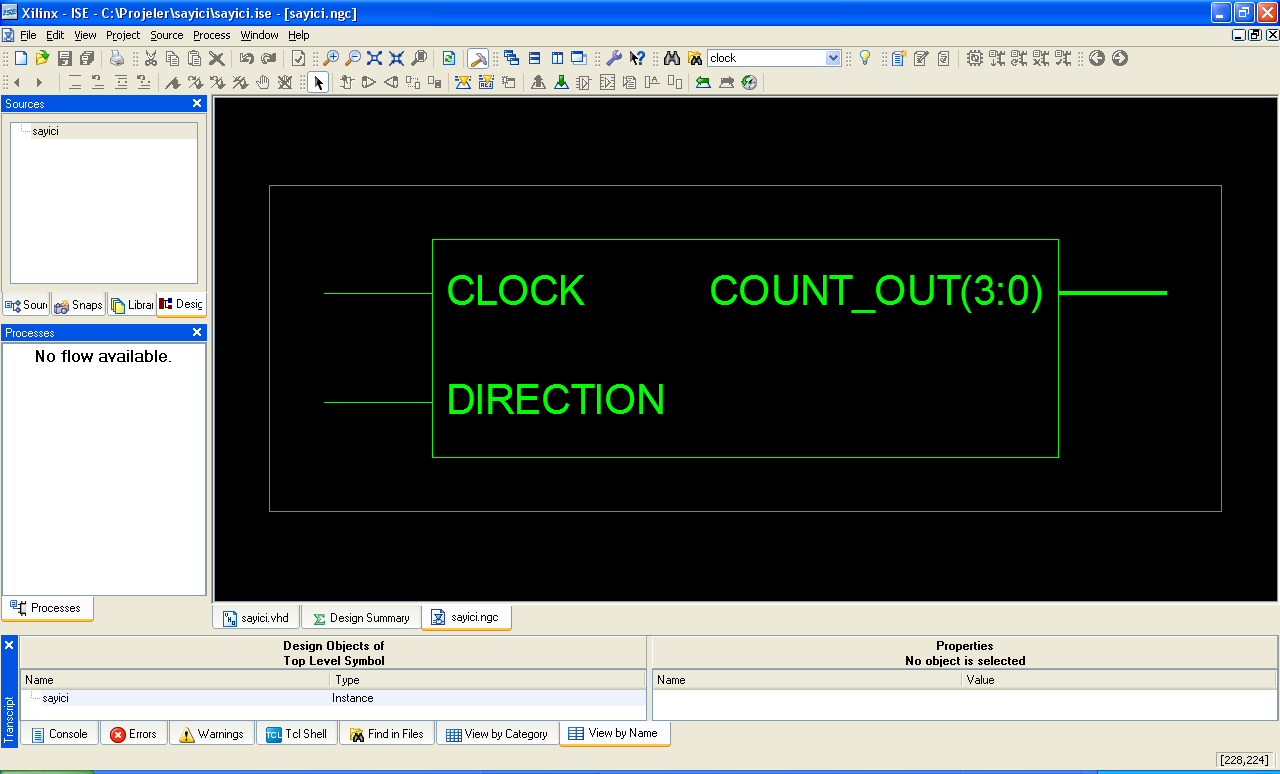
endmodule

Program yazılımı bittikten sonra kaydedilerek Şekil 14’de görülen Process kısmında bulunan Check Syntax ile hata denetimi yapılır.

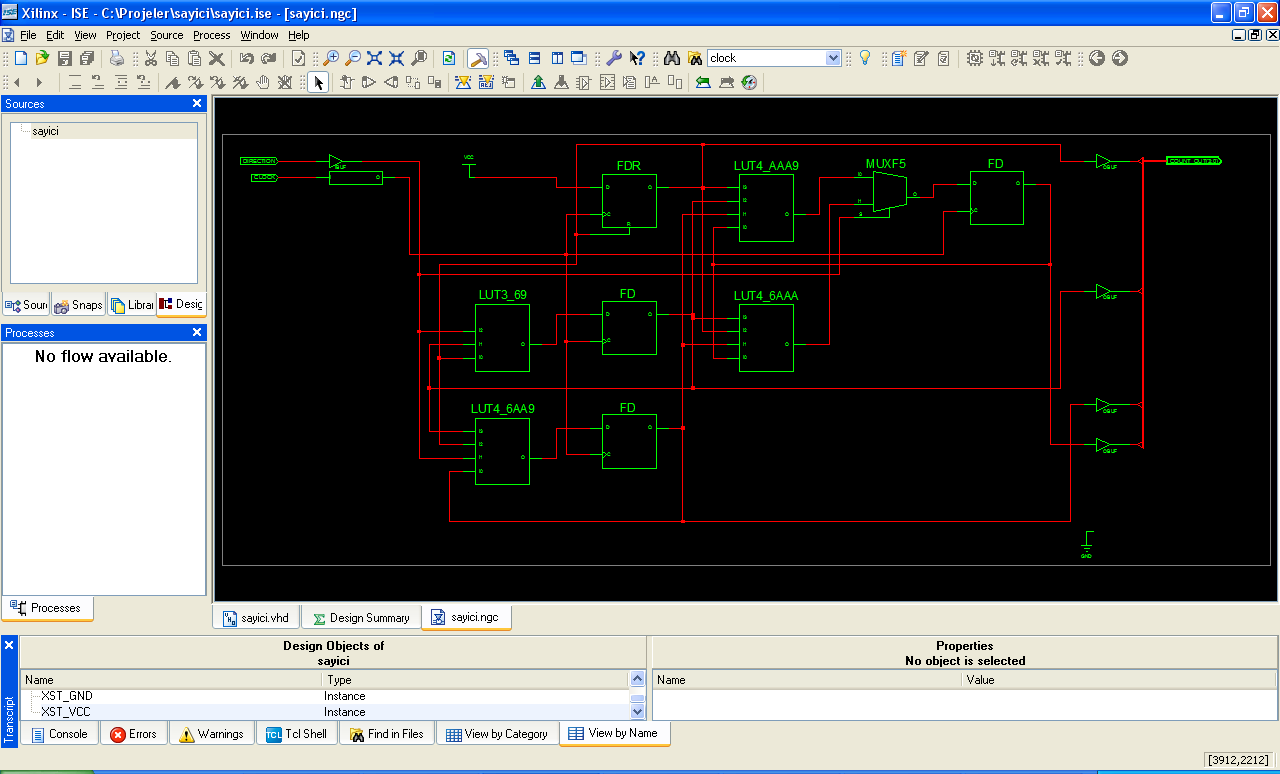
View Technology Shematic kısmına çift tıklandığında Şekil 15’deki gibi devrenin şematik gösterimi elde edilir. Bu gösterime de çift tıklandığında Şekil 16’da görüldüğü üzere devrenin tam tasarımı görülebilir. Eğer yazılan kodda hatta varsa bu çizimleri elde etmek mümkün olamaz.



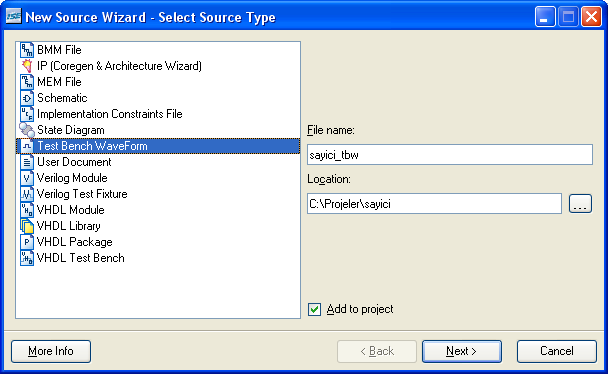
**Şekil 14.** Hata denetiminin gerçekleştirilmesi.



**Şekil 15.** Devrenin şematik gösterimi.

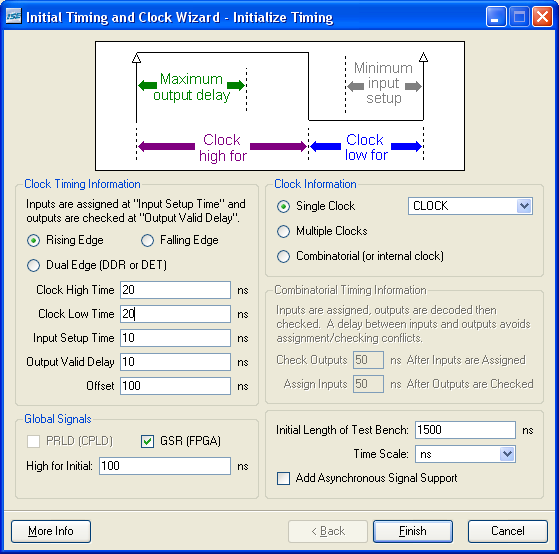


**Şekil 16.** Devrenin tam yapısı.

****

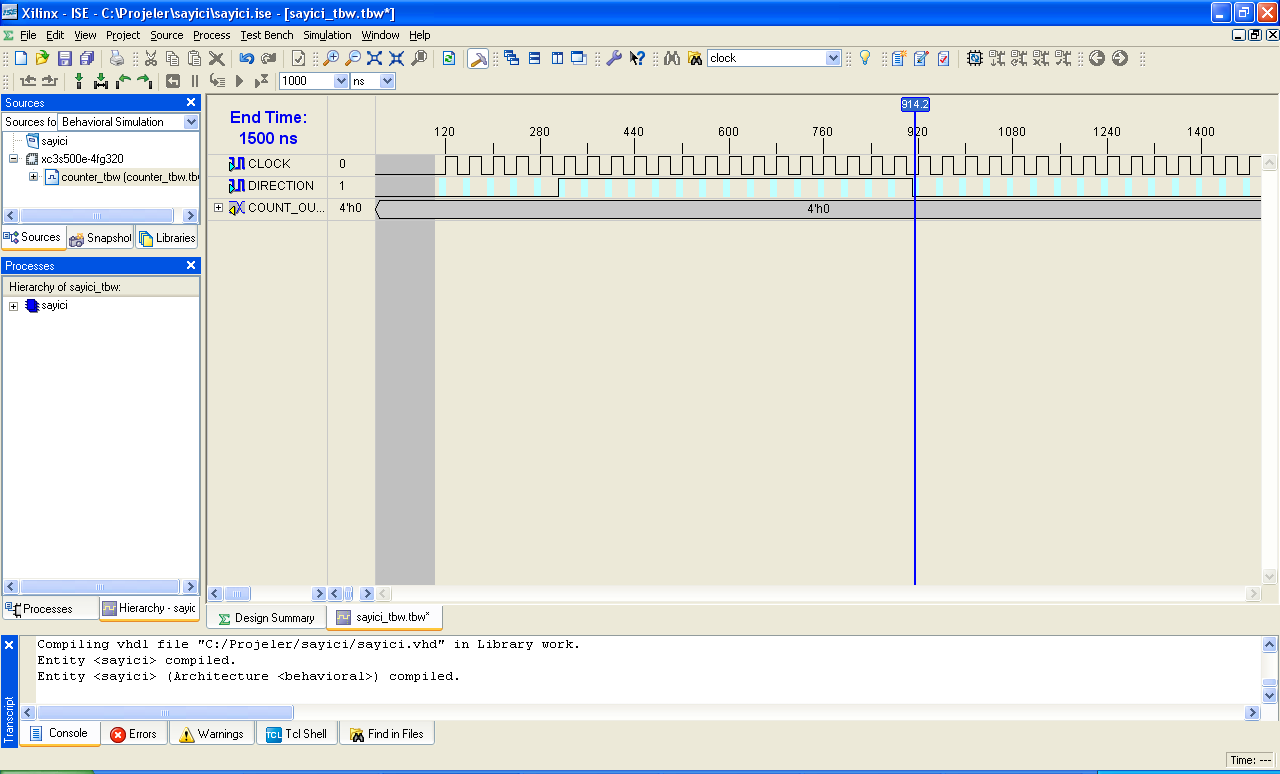
**Şekil 17.** Test sinyallerinin oluşturulması.

Bu işlemler bittikten sonra Implement Design seçeneğine çift tıklanarak modül gerçekleştirme işlemine başlanır. Yapılan tasarımın çalışıp çalışmadığını anlamak için ISE içerisinde kullanıcı tarafından oluşturulan sinyaller ile simülasyonu gerçekleştirilebilir. Bunun için, Source kısmından New source seçilir ve karşımıza Şekil 17’deki gibi bir pencere çıkar. Buradan Test Bench Waveform seçilir. Dosya adına da sayici\_tbw verilmiş olsun. Next butonu ile gelen pencerelerde bir değişiklik yapılmadıktan sonra Şekil 18’de gösterilen uygulanacak olan kare dalgaların periyot değerleri ve özellikleri girilir.

****

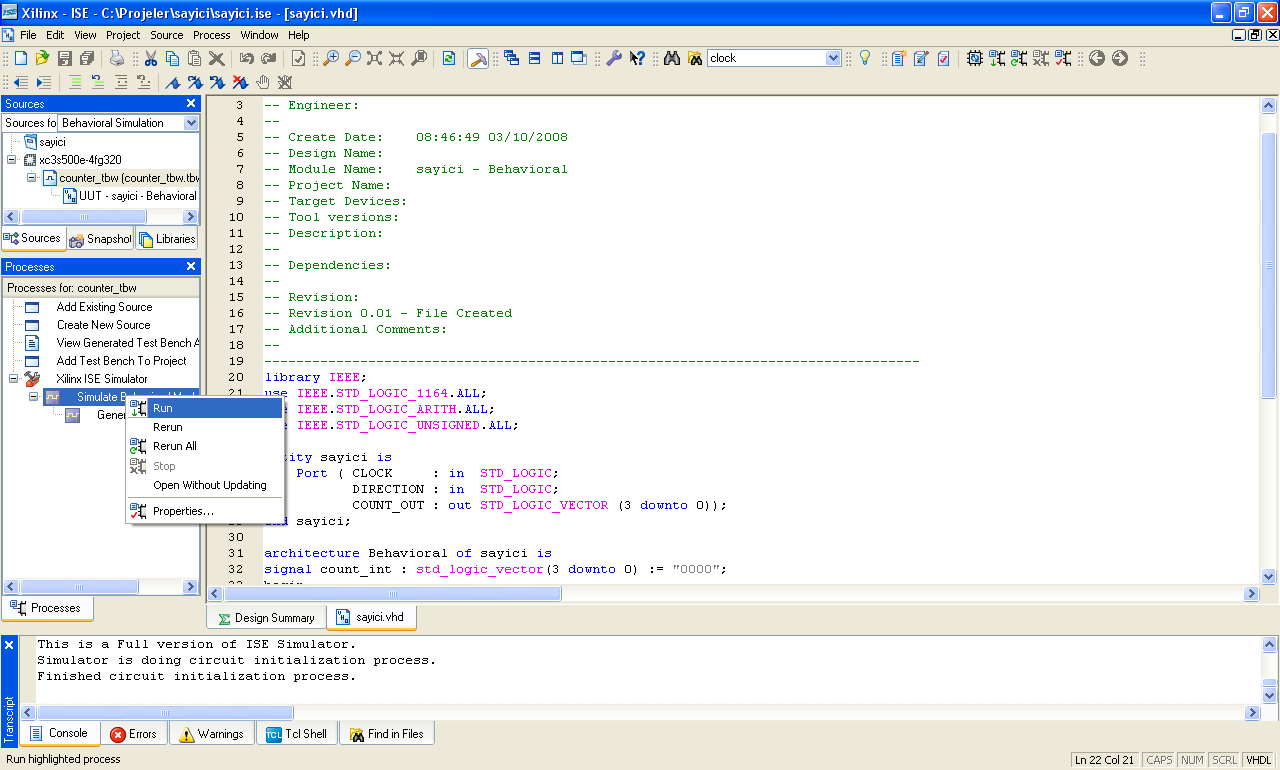
**Şekil 18.** Test sinyallerinin özellikleri ve test süresi.

İşlemci frekansı 50 MHZ olacak şekilde kare dalgaların yüksek ve düşük zaman süreleri seçilir. Global Signal kısmından GSR(FPGA) seçilir simülasyon süresi ise 1500ns olarak belirlendikten sonra Finish butonuna basılarak işlem bitirilir. Şekil 19’da da görüldüğü gibi sayıcının ileri geri çalışma durumunu belirleyen DIRECTION değeri yaklaşık 310 ns ve 910 ns değerleri arasında lojik-1’de olacak şekilde belirlenir. Mavi ile görülen kursor istenilen zaman bölgesine getirilip DIRECTION değişkeninin zaman eksenine tıklandığında bu değişkenin değer değiştirdiği görülecektir. Bu süreler arasında sayıcı ileri yönde bu değerler dışında ise sayıcı geriye doğru sayacaktır. İşlemler bittikten sonra test sinyalleri kaydedilir ve simülasyon dosyası içerisine gömülerek pencere kapatılır.

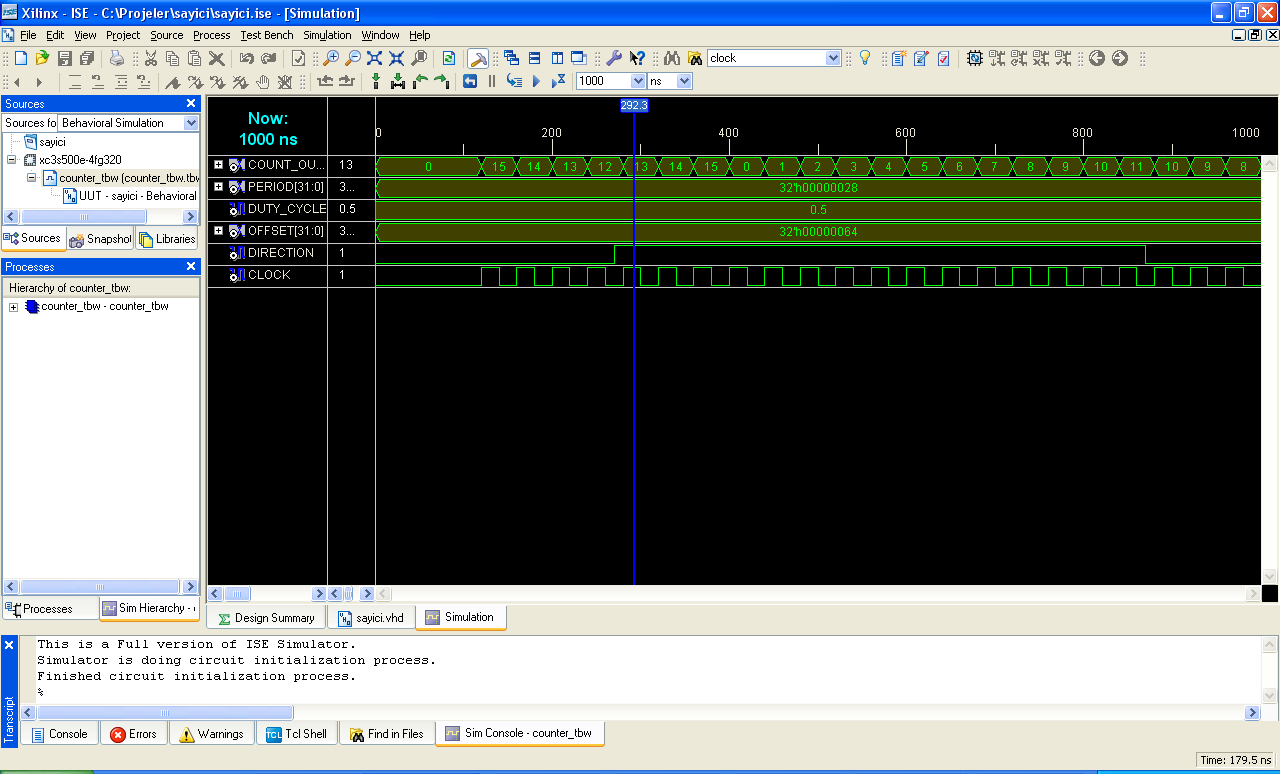


**Şekil 19.** Test değişkenlerinden olan DIRECTION değişkenine değer atanması.

Simülasyonu çalıştırmak için, Process kısmından Şekil 20’de görüldüğü gibi Simulate Behavioral Model farenin sağ tuşu ile seçilip Run seçeneğine tıklanıp simülasyon başlatılır.

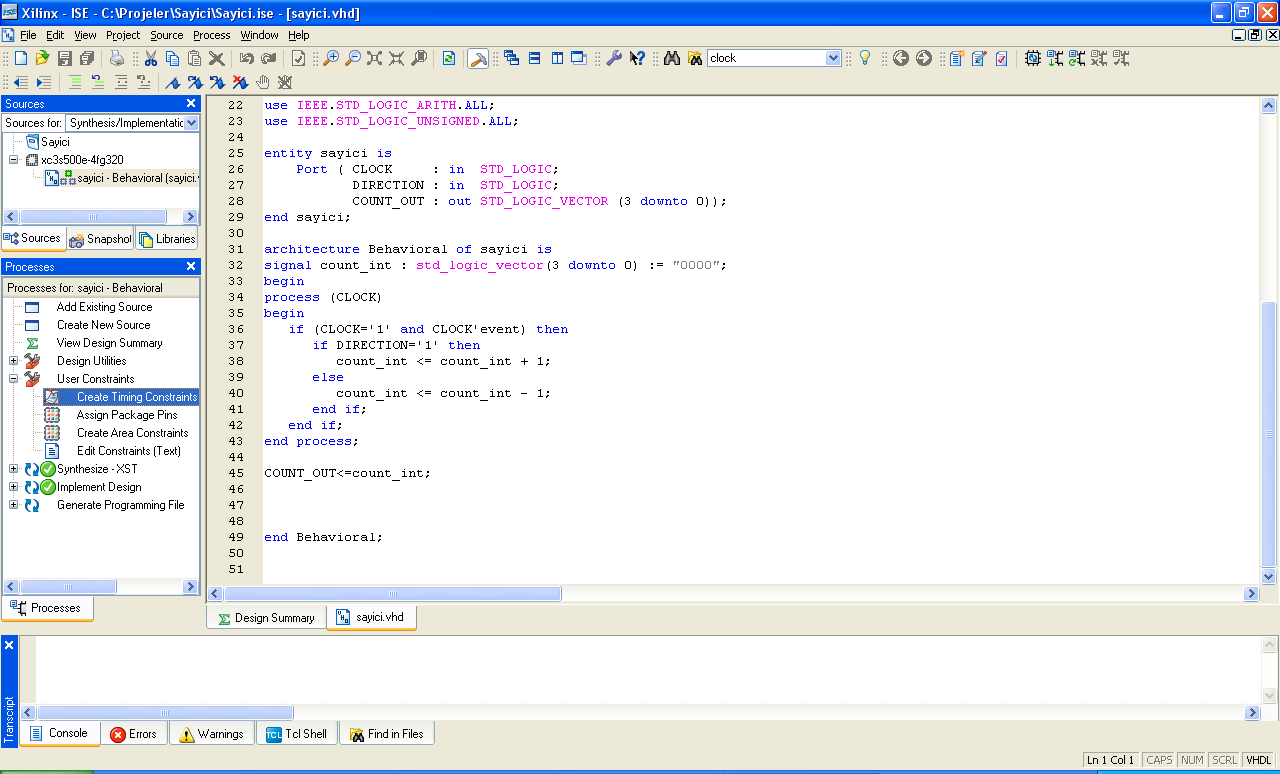


**Şekil 20.** Simülasyonun başlatılması.

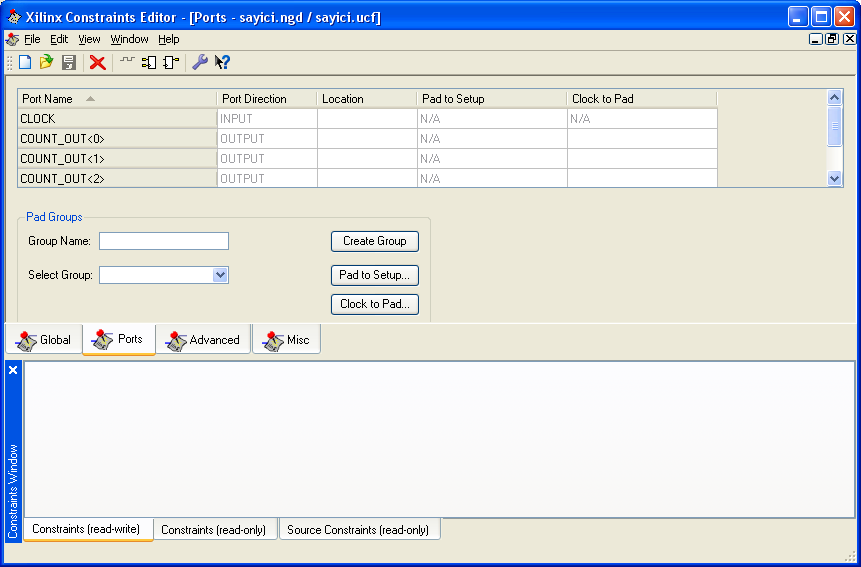


**Şekil 21.** Simülasyonun gerçekleştirilmesi.

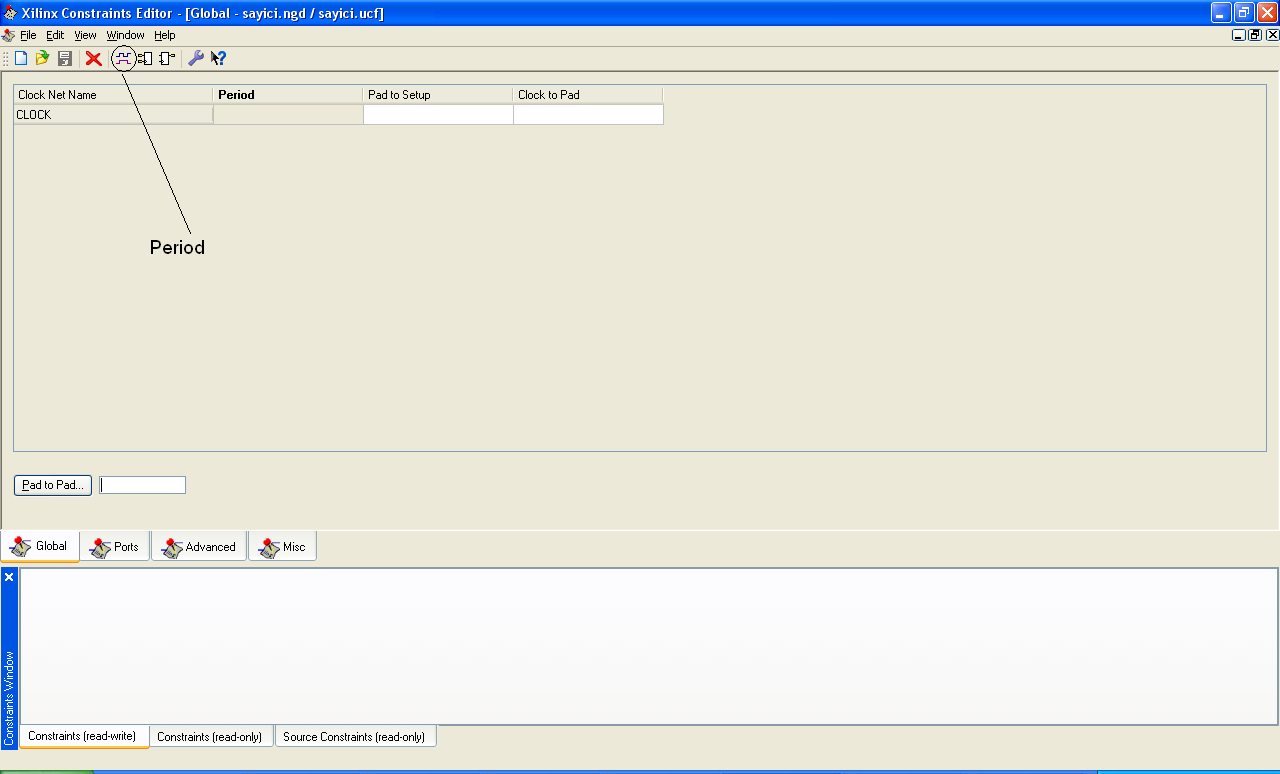
Simülasyon gerçekleştirildikten sonra zaman kursörü istenilen zaman değerine getirilerek sayıcının bitlerinin nasıl değiştiğine bakılır. Periyot değerlerinin girilmesi için Şekil 22’de görülen Source kısmından Syntesis/Implementation seçilir ve Create Timing Constraints seçeneğine çift tıklanarak açılan pencereden de Evet seçeneği seçilerek Şekil 23’deki Editöre ulaşılır. Evet ile oluşturmak istenen UCF dosyasının projeye eklenmesinin istenildiği söylenmektedir.



**Şekil 22.** FPGA uçlarının seçilmesi için gerekli ilk adımın gerçekleştirilmesi.

****

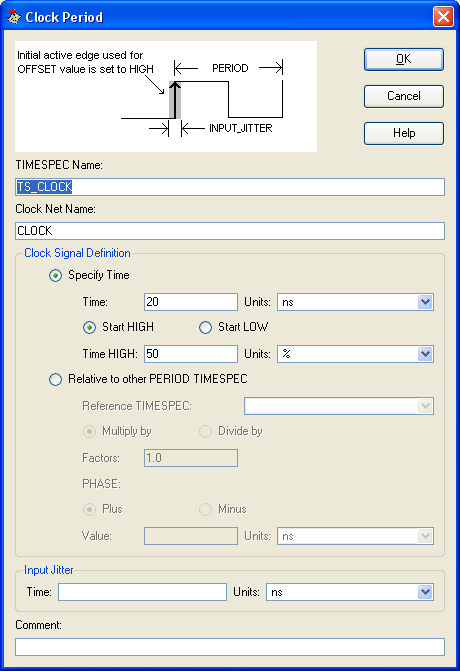
**Şekil 23.** FPGA uçlarının seçiminin gerçekleştirildiği editör.



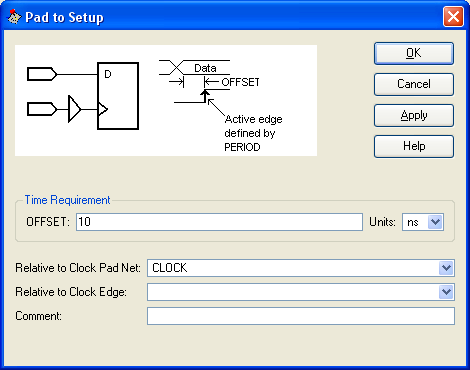
**Şekil 24.** Periyot değerinin girilmesi.

Global kısmı seçildiğinde Şekil 24’deki gibi bir pencere açılacak ve bu pencereden Period kısmına tıklanarak periyot değerinin girilmesi işlemi gerçekleştirilecektir.

Bu seçim yapıldığında ekrana Şekil 25’de görülen pencere açılacaktır. Bu pencerede Time değerine 20 ns değeri girilir.



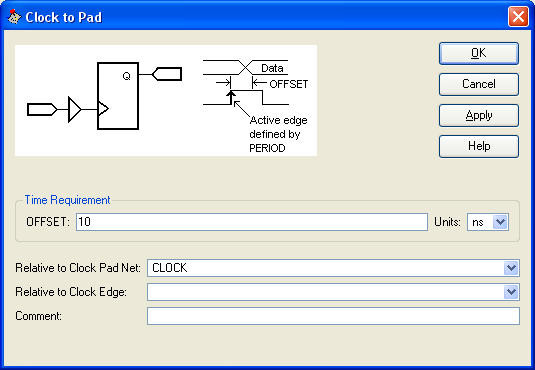
**Şekil 25.** Periyot değerinin girilmesi.



**Şekil 26.** Ofset değerinin girilmesi.

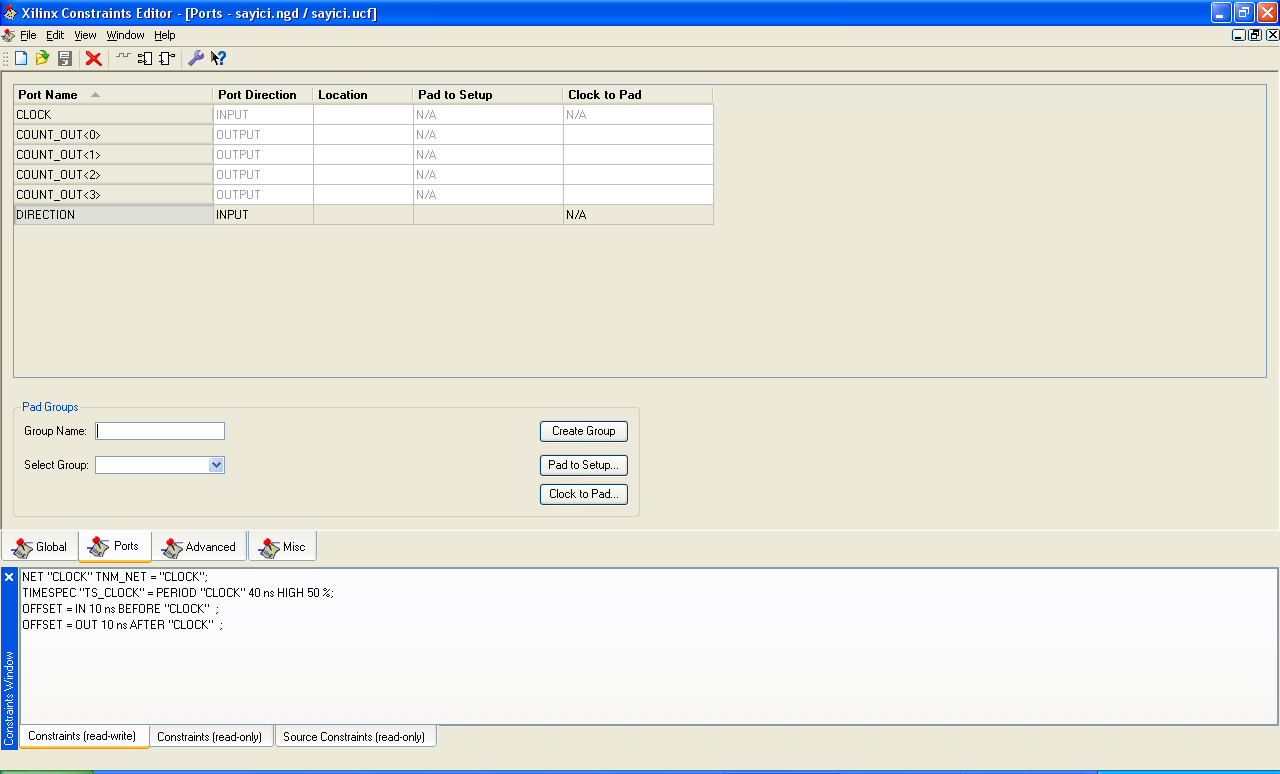
Daha sonra Pad to Setup ikonuna tıklanır ve Şekil 26’daki gibi offset değeri olarak 10 ns girilir.

Daha sonra Clock to Pad ikonuna tıklanarak Şekil 27’deki gibi değeri girilir.



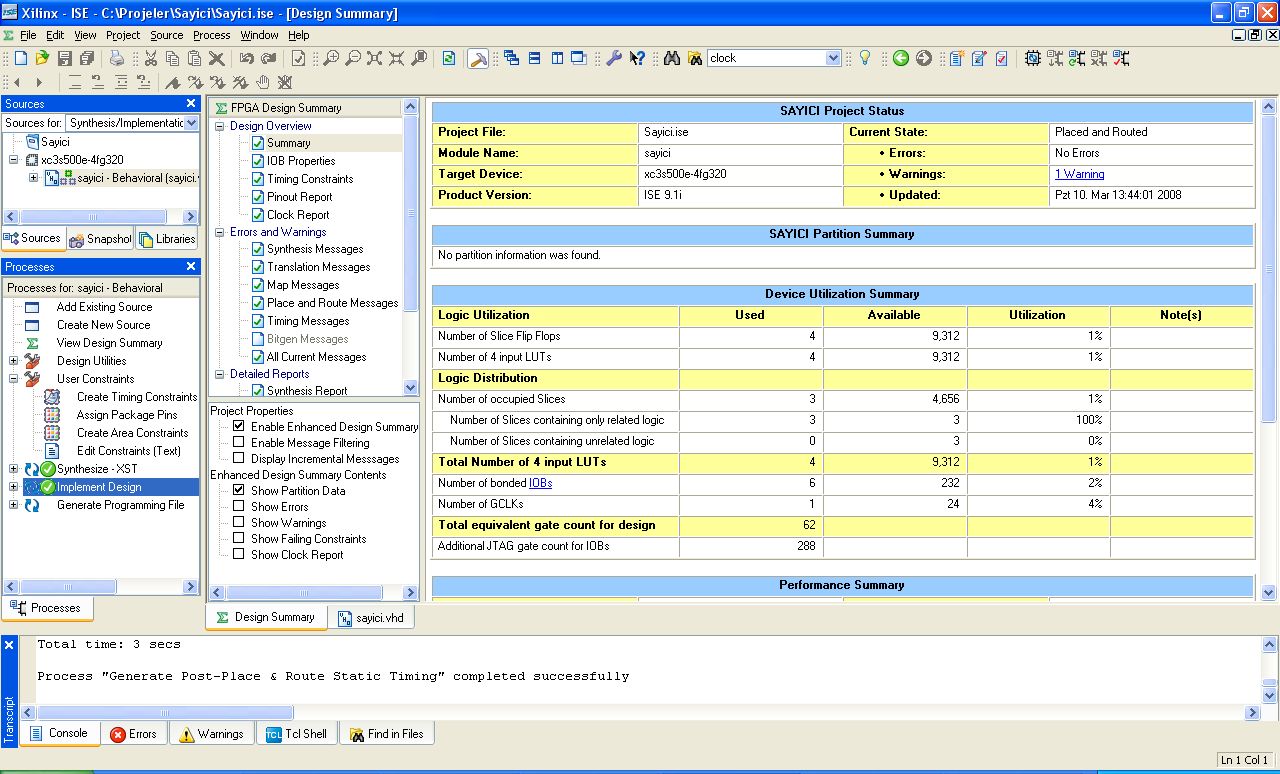
**Şekil 27.** Clock to Pad değerinin girilmesi.

Değerlerin girilmesinden sonra Şekil 28’de görülen dosya menüsünden bu değerler kaydedilir. Daha sonra Process kısmında bulunan Implement Design seçeneğine çift tıklanarak tasarım gerçeklenir ve oluşan tasarım özeti ekranda Şekil 29’daki gibi elde edilir.

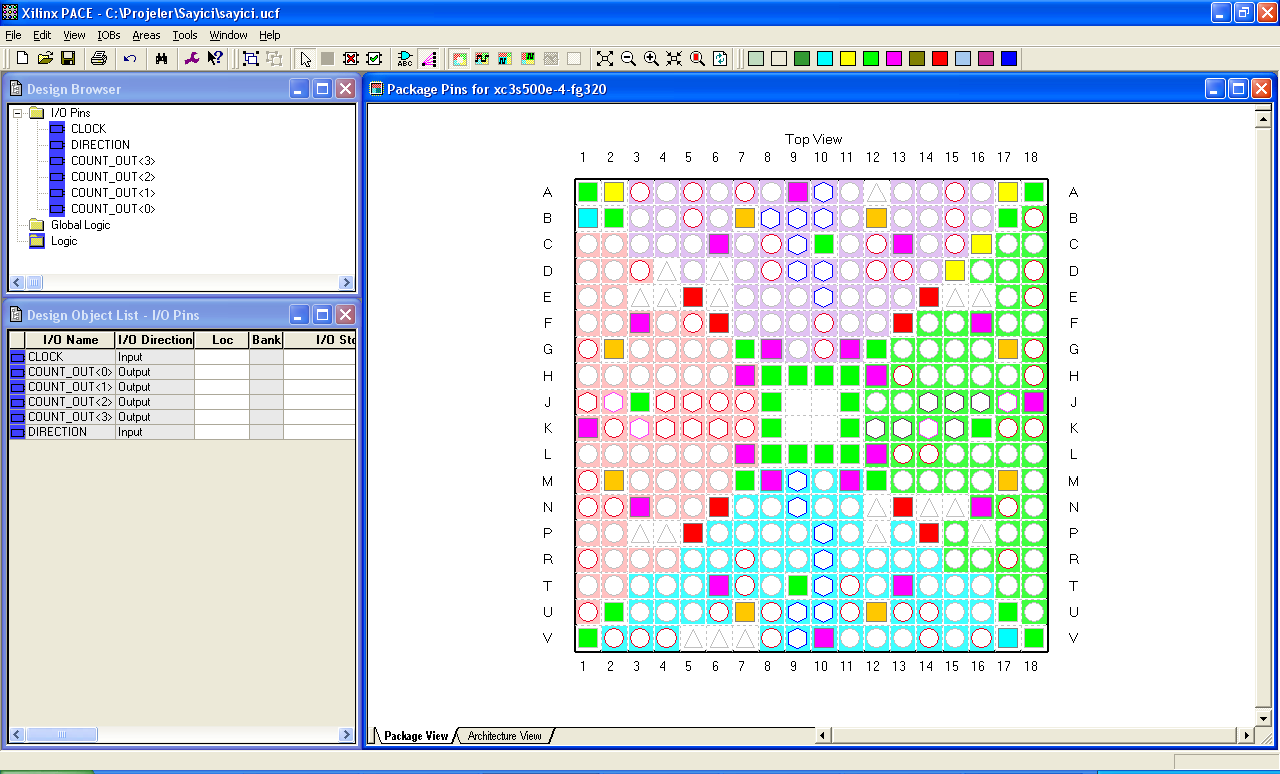
****

**Şekil 28.** Oluşan UCF dosyasının en son hali.

Fiziksel olarak yazılan programda hangi giriş ve çıkışların FPGA’nın hangi uçlarına karşılık geldiğini belirtmek gerekmektedir. Process menüsünden User Constraints’in altında bulunan Package Pins seçeneğine çift tıklanarak Şekil 30’daki FPGA’nın fiziksel pinlerinin seçimi ekrana gelir. Buradan hangi girişin ya da çıkışın hangi pine karşılık geldiği belirlenir.

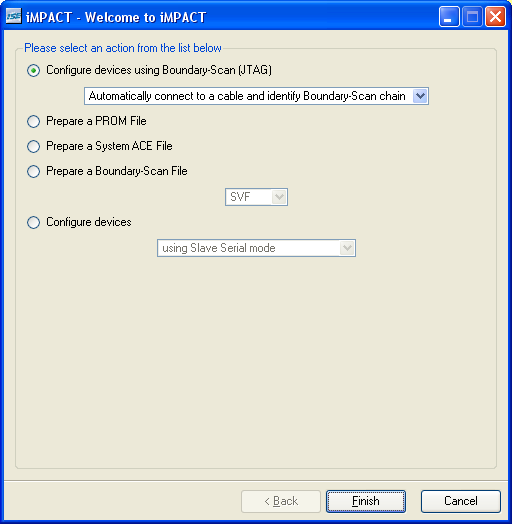


**Şekil 29.** Tasarım Özeti.



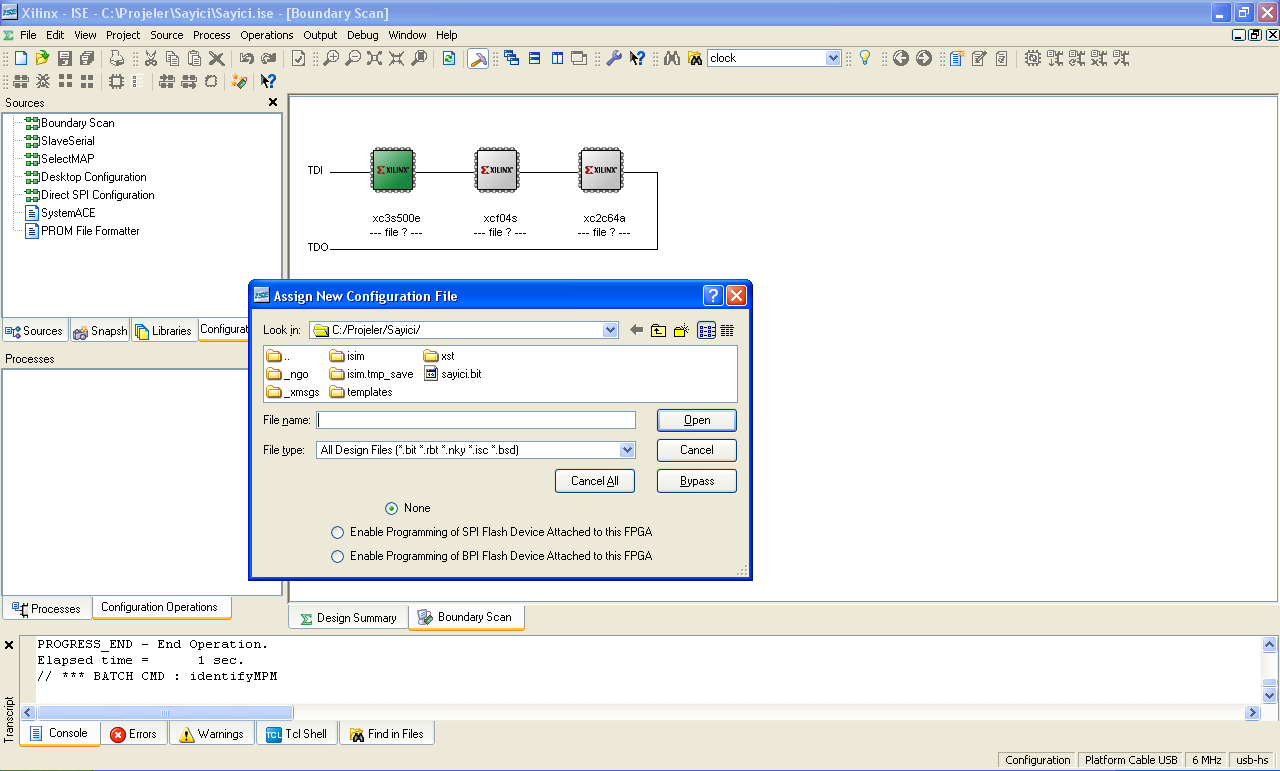
**Şekil 30.** FPGA’nın fiziksel pinlerinin seçimi.

Seçimden sonra tekrar Sythesis Design ve Implement Design seçenekleri çalıştırılarak program güncellenir. Son aşama olarak Generate Programming File kısmından Configure Device (IMPACT) seçeneğine çift tıklanarak çalıştırılır. Şekil 31’de görüldüğü üzere gelen pencereden otomatik olarak PC’nin starter kite bağlanmasına izin verilir. Finish butonuna tıklanarak bu işlem geçilir.



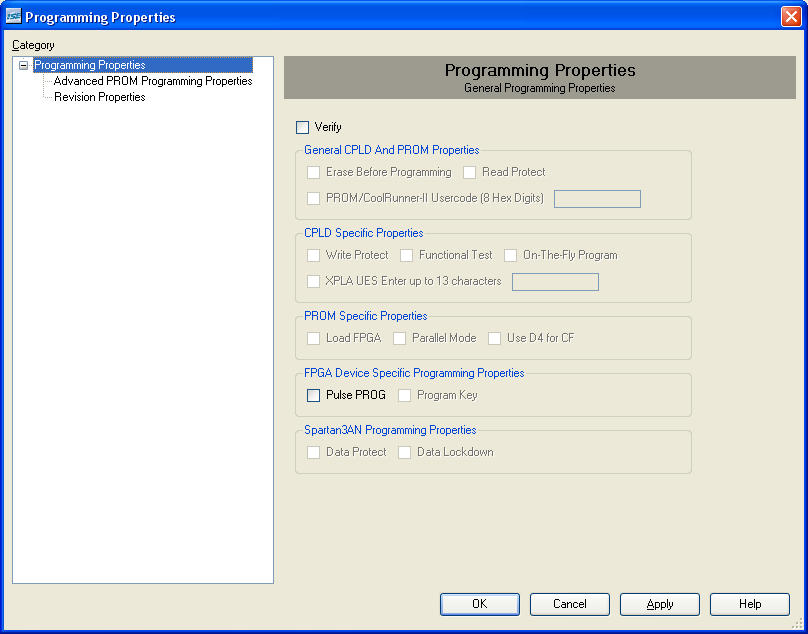
**Şekil 31.** IMPACT kullanıcı arabirimi karşılama ekranı.

Sonra yazılmış olan verilog programının starter kit’e yüklenmesi için dosya yerinin sorulduğu Şekil 32’deki ekran görüntüsüne gelinir. Burada dosya seçilerek tamam butonuna basılır. Daha sonra gelen iki ekran ise bypass ile boş geçilir.

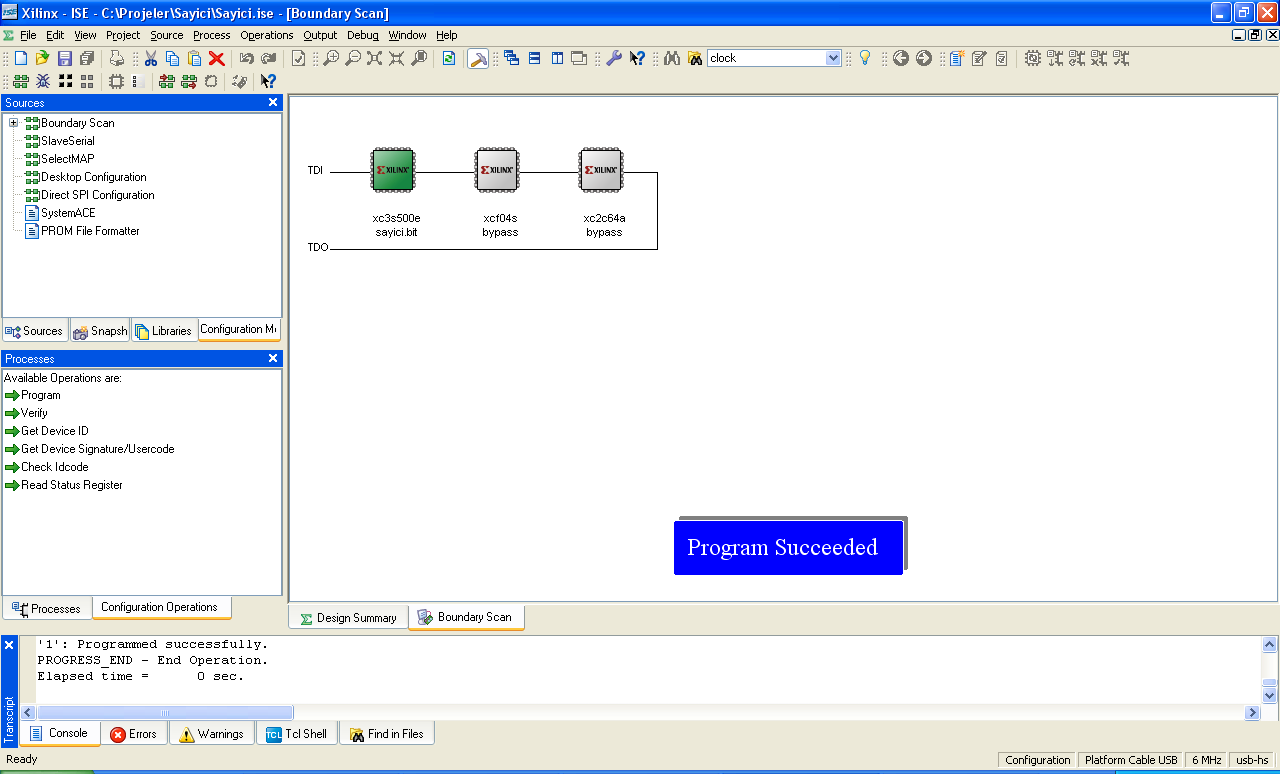


**Şekil 32..**bit dosyasının seçimi.

Üzerinde xc3s500e yazılı işlemciye farenin sağ tuşu ile tıklanarak açılan kısımdan Program seçeneği seçilir. Şekil 33’de görülen pencereden OK butonuna basıldığında eğer işlemciye yazma işi gerçekleşmişse Şekil 34’de görüldüğü gibi Program Succeeded şekline başarılı olarak yazma işleminin gerçekleştiği görülür. Program artık FPGA starter kitinde çalışmaktadır. IMPACT kısmı kapatılırken program herhangi bir şekilde kayıt yapmak istediğinde kabul edilmeden çıkılır.



**Şekil 33.** Programın yüklenmesi.



**Şekil 34.** Programın başarılı bir şekilde yüklendiğinde verdiği mesaj.