CPLD DENEY SETİ TASARIMI

Salim KIYMAZ

YÜKSEK LİSANS TEZİ
ELEKTRONİK BİLGİSAYAR EĞİTİMİ

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

EYLÜL 2005
ANKARA

CPLD DENEY SETİ TASARIMI

Salim KIYMAZ

YÜKSEK LİSANS TEZİ
ELEKTRONİK BİLGİSAYAR EĞİTİMİ

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

EYLÜL 2005
ANKARA

Salim KIYMAZ tarafından hazırlanan CPLD DENEY SETİ TASARIMI adlı bu tezin yüksek lisans tezi olarak uygun olduğunu onaylarım.

Yrd. Doç. Dr. Rahmi CANAL
Tez Yöneticisi

Bu çalışma, jürimiz tarafından Elektronik Bilgisayar Eğitimi Anabilim Dalında yüksek lisans tezi olarak kabul edilmiştir.

Başkan : : Prof. Dr. İnan GÜLER

Üye : Prof. Dr. Çetin ELMAS

Üye : Yrd. Doç. Dr. M. Rahmi CANAL

Bu tez, Gazi Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygundur.

CPLD DENEY SETİ TASARIMI

(Yüksek Lisans Tezi)

Salim KIYMAZ

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
Eylül 2005

ÖZET

Bu tez çalışmasında XILINX firmasının üretmiş olduğu Karmaşık programlanabilir lojik devreler (CPLD) ile dijital devre tasarımı yapılmıştır. Tasarımlarda aynı firmanın WebPACK ISE programı kullanılmıştır. Tasarımlarda yüksek düzey donanım programlama dili (VHDL) veya şematik çizim kullanarak gerçekleştirilmiştir. Gerçekleştirilen sayısal devrelerin simülasyonları MXE Simulator programı ile test edilmiştir. Tasarlanan uygulamaların direkt olarak CPLD'ye yüklenmesi ve tasarımda yapılacak değişikliklerin kolayca yapılabilmesi en büyük avantajıdır. Bilgisayar ortamında yapılan bu çalışmalar, dijital elektronik uygulamalarında zaman, işgücü ve maliyet yönlerinden avantaj sağlamaktadır.

Bilim Kodu :626.04.01
Anahtar Kelimeler :Xilinx, CPLD, Tasarım
Sayfa Adedi :126
Tez Yöneticisi :Yrd.Doç.Dr. M. Rahmi CANAL

DESIGN OF CPLD EXPERIMENT SET

(M.Sc. Thesis)

Salim KIYMAZ

GAZI UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
September 2005

ABSTRACT

In this thesis, the circuit design of CPLDs produced by XILINX, is studied by using WebPACK ISE program produced by the same company. Designs are done by using VHDL programming language or schematic methods. Simulation of designs are studied by using MXE Simulator Program. Design of applications will be easily transferred to CPLD on the electronic circuit and will be reloaded again after doing the required changes on the design to CPLD. The studies done on the computers provide several advantages such as time, workforce and cost for digital electronic applications.

Science Code :626.04.01
Key Words :Xilinx, CPLD, Design
Page Number :126
Adviser :Asist. Prof. Dr. M. Rahmi CANAL

TEŞEKKÜR

Tezimin hazırlanmasında yardımlarını, destekleyici tavırlarını esirgemeyen ve kıymetli bilgilerinden yararlandığım sevgili hocam Yrd.Doç.Dr. Rahmi CANAL'a sonsuz teşekkürlerimi sunarım. Ayrıca çalışmalarım boyunca yardım ve katkılarını esirgemeyen sevgili arkadaşım Arş.Gör. Fecir DURAN'a ve Uzm.Dr. Tolga ELBİR'e teşekkürlerimi bir borç bilirim.

İÇİNDEKİLER

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil **Sayfa**

Şekil **Sayfa**

Şekil **Sayfa**

KISALTMALAR LİSTESİ

Kısaltmalar	Açıklama
ABEL	Advanced Boolean Equivalent Language, Gelişmiş Boolean Eşitlikleri Dili
ASIC	Application Specific Integrated Circuit, Uygulamaya Özel Entegre Devre
CMOS	Complementary Metal Oxide Semiconductor,
CPLD	Complex Programmable Logic Device, Karmaşık Programlanabilen Mantıksal Devreler
EC	Elde çıkışı
EEPLD	Electrically-Erasable Programmable Logic Device, Elektrikle Silinebilen Programlanabilen Mantıksal Cihazlar
EG	Elde girişi
EPLD	Erasable Programmable Logic Device, Silinebilen Programlanabilen Mantıksal Cihazlar
EPROM	Erasable Programmable Read Only Memory**,**
FPGA	Field Programmable Gate Array, Alansal Programlanabilir Kapı Dizisi
FPIC	Field Programmable InterConnect, Alansal Programlanabilir Bağlantı Dizisi
GAL	Generic Array Logic, Genel Dizi Mantığı
HDL	Hardware Description Language, Donanım Tanımlama Dili
I/O	Input/Output, Giriş/Çıkış
IEEE	Institute of Electrical and Electronics Engineers, Elektrik ve Elektronik Mühendisleri Enstitüsü
ISE	Integrated Sotware Environment, Entegreye Yönelik Yazılım
JTAG	Joint Test Advisory Group, Ortak Grup Test Aracı
LSB	Least Significant Bit, Düşük Değerli Bit
LSI	Large Scale Integration, Geniş Ölçekli Entegreleme
MAX	Multiple Array matriX, Altera

Kısaltmalar	Açıklama
MSB	Most Significant Bit, Yüksek Değerli Bit
MSI	Medium Scale Integration, Orta Ölçekli Entegreleme
MXE	ModelSim XE
PAL	Programmable Logic Array, Programlanabilir Dizi Mantığı
PALASM	PAL Assembler, PAL Derleyici
PALCE	Programmable Array Logic Configurable Erasable, Silinebilir-Düzenlenebilir ve Programlanabilir VE Dizi Mantığı
PLA	Programmable Array Logic, Programlanabilir Mantık Dizisi
PLD	Programmable Logic Device, Programlanabilir Mantık Devreleri
PROM	Programmable Read Only Memory, Programlanabilir Sadece Okunabilir Bellek
PT	Product Term, Ürün Terminali
RAM	Random Access Memory, Rastgele Erişilebilir Memory
ROM	Read Only Memory, Sadece Okunabilir Bellek
RTL	Register Transfer Level, Kaydedici Transfer Seviyesi
SPLD	Simple Programmable Logic Devices, Basit Programlanabilir Mantıksal Devreler
SRAM	Static Random Access Memory, Değişmeyen Rastgele Erişilebilir Memory
SSI	Small Scale Integration, Küçük Ölçekli Entegreleme
TT	Tam Toplayıcı
VHDL	VHISC High Level Description Language, Yüksek Seviyeli Donanım Programlama Dili
VHISC	Very High Speed Integrated Circuit, Çok Yüksek Hızlı Entegre Devre
VLSI	Very Large Scale Integration, Çok Geniş Ölçekli Entegreleme
YT	Yarım Toplayıcı

1. GİRİŞ

Günümüzde büyük bir hızla gelişen teknoloji, kullanılan elektronik devrelerin de daha işlevsel olmalarıyla birlikte daha karmaşık bir hal almasını sağlamışlardır. Artan bu ihtiyaçlar yeni birtakım zorlukları da beraberinde getirmektedir. Örneğin ortalama büyüklükteki bir dijital devrede 30000 ile 100000 arası lojik kapı bulunmaktadır. Bu büyüklükteki devrelerin tamamen insan emeği ile tasarlanması neredeyse imkansız ve çok uzun zaman alacaktı. Ayrıca yapılan tasarımlarda hata yapmaya çok açık olacaktı.

Önceleri, lojik kapılarla yapılan dijital devreler, ihtiyaçların artmasıyla daha sonraları PAL(Programmable Logic Array) ve PLA(Programmable Array Logic) tekniği sayesinde daha kolaylaşmıştır. Günümüzde ise bu teknikler ihtiyaca cevap veremez hale gelmişler, fakat günümüzde kullanılan teknolojilerinin temelini oluşturmuşlardır. Entegre devre teknolojisinin gelişmesiyle bilgisayar için üretilen işlemciler için de gelişim çağı açılmıştır. İşlemcilerin her geçen gün hızla gelişmesine bağlı olarak algoritma tasarımı ve programlama çeşitleri de artmıştır. Programlanabilir mantık, bir mantıksal devre gibi bağımsız çalışan flip-flop linklerinin, programlanabilecek şekilde bir araya gelmeleri ile oluşur. Hafıza hücreleri mantıksal performans fonksiyonlarını kontrol eder ve bunların çeşitli mantıksal fonksiyonlar ile nasıl bağlanacağını tanımlar. Çeşitli aletler farklı mimariler kullansa da hepsi aynı temel düşünceye sahiptir (1).

Gün geçtikçe ASIC(Application Specific Integrated Circuit) pazarındaki ürün çeşidi, devrelerin karmaşıklığı artarken, rekabet koşulları içinde, tasarımların ürüne dönüştürülme süreleri ise kısalmaktadır. ASIC tasarımı, devre tasarımı ve fiziksel gerçekleme olarak iki ana kısma ayrılabilir. Devre tasarımında devrenin, devre blokları veya transistör seviyesinde şeması tasarlanıp simüle edilirken, fiziksel gerçeklemede, entegre devre tasarım mühendisleri; kullanılan teknoloji, malzeme, işlem aşamaları gibi parametreleri de hesaba katarak entegre devreye son şeklini verirler. Bu iki safha da, devrelerin karmaşıklaşması ile otomasyonu gerektirir hale gelmiştir.

Dijital elektronik dünyasında, PLD'ler (Programmable Logic Device) önemli bir yer kaplamaktadır. Çünkü tüm basit mantık kapılarının işlevleri ve milyonlarca kapı ile oluşturulabilecek dijital tasarımlar, tek bir entegre devre olarak üretilen bu programlanabilir mantık elemanları içerisinde gerçekleştirilebilmektedir. Oysa bu tasarımları 74XX serisi entegre devrelerle gerçekleştirmek için geniş bir oda büyüklüğünde alana ihtiyaç duyulacaktı. Örneğin 16*16 Multiplexer için 6000 adet lojik kapı, 30 sayfa şematik çizim, günlerce uğraşı ve hata yapma olasılığı olacaktı. HDL kodlarıyla yazılması halinde ise 1 sayfa ve 8 satırlık kod ve birkaç dakikalık süre gerekecekti.

Bellek, işlemci gibi devrelerin temelinde, mantık kapılarının fonksiyonları yatmaktadır. Buralarda kullanılacak kapı sayısı çok fazla olacaktır. İşte PLD'ler bu soruna çözüm getirmişlerdir. İçlerinde çeşitlerine göre milyonlarca kapı barındırırlar ve bunun yanı sıra kullanıcıya sadece kapılar arasındaki bağlantıları ayarlama işini bırakmaktadır. Tüm bunların tek bir entegre devre içerisinde gerçekleşmesi de PLD'lerin ne kadar yararlı elemanlar olduklarını göstermektedir.

Programlanabilir mantık elemanları geniş bir mantık kapasitesi, çeşitli özellikler, hız ve geniş voltaj karakteristikleri sağlamasının yanında, istenilen başka bir fonksiyonu gerçekleştirmek için defalarca silinip programlanabilirler (2).

74XX ve 40XX serisi gibi entegre devrelerle yapılan sabit mantık devrelerinde tasarımdan ilk örneğe kadar gerekli zaman, elemanın karmaşıklığına göre birkaç aydan birkaç yıla kadar sürebilir. Eleman doğru çalışmaz ise yada değişiklik gerekirse, yeni bir tasarım geliştirilmek zorunda kalınabilir. Ayrıca bu elemanların, istenen tasarımın bir silikon parçasına yüklenmesine kadar geçen zamandaki imalat masrafları da çeşitli kriterlere bağlı olarak birkaç bin dolardan birkaç milyon dolara kadar olabilir. Bu imalat masrafları içerisinde mühendislik giderleri, pahalı bilgisayar gereçleri ve ek elemanlar gibi giderler bulunmaktadır.

PLD, bir mantıksal devre gibi bağımsız çalışan mantık bölümlerinin, programlanabilecek şekilde bir araya gelmeleri ile oluşur. PLD, tasarım geliştirmek, simüle etmek ve test etmek için pahalı olmayan gereçler kullanır. Daha sonra bu tasarım bir eleman içine programlanmadan önce kolayca bilgisayar ortamında simüle edilebilir veya gerçek bir devre içinde de test edilebilir. Hata veya tasarımın değişmesi durumunda PLD, tekrar programlanabileceği için hiç bir şekilde imalat giderleri sabit mantıktaki kadar olmayacak ve tasarım çok daha çabuk tamamlanabilecektir (3).

PLD'lerin son yıllardaki gelişmelerine bakılarak birçok tasarımcı tarafından değişik tasarımlarda kullanılmaktadır. Xilinx, Altera, Lattice, Actel ve Cypress gibi çeşitli firmalar PLD üretmektedirler. Bu hızlı gelişimin sebebi ise Xilinx gibi üreticilerin bu elemanların silikona dökülmesiyle ilgilenmeyip tamamıyla bunların tasarımları ile ilgilenmesidir. Silikona dökme işlemini, ana işlevi Entegre devre üretmek olan IBM, Microelectronics ve UMC gibi şirketlere bırakmaktadır ve bu sayede en son entegre devre üretim teknolojisi kullanılmaktadır. Bu strateji Xilinx gibi firmalara, yeni ürün mimarileri, yazılım gereçleri ve akıllı işlemci merkezleri tasarımlarına daha iyi odaklanma fırsatını vermesinin yanında, PLD'lerin çok daha hızlı olmasına, daha az güç ihtiyaçlarına, fazla özelliği barındırmasına ve çok ucuz olmasına sebep olmaktadır.

PLD temelde bir VE mantık dizisi ve bunu izleyen sabit yada programlanabilir bir VEYA mantık dizisi içermektedir. FPGA (Field Programmable Gate Array) ve CPLD'ler (Complex Programmable Logic Device) gibi karmaşık PLD'ler, bir çok basit PLD'yi ve bunların arasındaki bağlantıları içermektedirler. Bağlantılar basitçe bir sigorta ve diyot çifti ile birbirlerine bağlanabildiği gibi PLD çeşitlerine göre fet-transistörler kullanılarak da bu bağlantı gerçekleştirilebilmektedir. Günümüz teknolojisinde flash teknolojisi kullanılarak entegre devre ebatlarının küçülmesi ve güç tüketimlerinin azalması sağlanmaktadır. PLD içerisinde hangi tür bağlantı kullanılırsa kullanılsın sonuçta işlev olarak aynı görevi yaparlar.

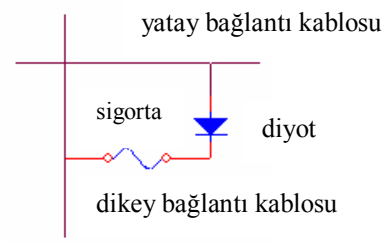
Bir PLD ile tasarım yapılacağı zaman içerisindeki bağlantı noktaların durumları uygun donanım tanımlama dili HDL(Hardware Description Language) ile tasarıma uygun olarak belirlenmektedir. Bu diller kullanıcının isteklerini PLD'nin anlayacağı forma dönüştürür ve aynı mikrodenetleyici elemanlarda olduğu gibi bir yükleme dosyası oluştururlar. Bir donanım arabirim cihazı ile bu dosya PLD'ye yüklendiğinde sonuçta eldeki PLD tamamıyla mantıksal bir devreyi gerçekleyen yeni bir eleman özelliğinde olur.

PLD'ler basit yada karmaşık çeşitli tasarımlar için kullanılmaktadır. Basit bir tasarımda PLA ve PAL gibi basit programlanabilir mantık elemanları kullanılırken karmaşık bir tasarımda tercih edilecek olan PLD'ler mutlaka CPLD'ler yada FPGA'lar olacaktır. Peki bir devre tasarımı için hangisi daha uygun olur? Bunun cevabı aralarındaki farkların bilinmesidir. Örneğin ikisi arasındaki farkı basitçe şu şekilde ifade edebiliriz; CPLD'ler daha az enerjiye ihtiyaç duyarlar ve zaman karakteristikleri gelişmiştir, FPGA'lar ise CPLD'lere göre daha yüksek performans ve daha fazla içerik sunarlar ama daha pahalıdırlar. Bu durumda biz eğer elde taşınabilir bir aygıt tasarımı yapıyorsak az enerji isteyen ve küçük bir pille çalışabilen CPLD'leri tercih edebiliriz. Yapılmak istenen tasarım daha karmaşıksa ve daha yüksek performans bekleniyorsa bu kez FPGA'ları kullanmak daha uygun olacaktır (4).

Tezin bu bölümünde verilen bilgilerle PLD ve CPLD'lerin genel olarak tanınması sağlanmış olacaktır. İkinci bölümde ise Programlanabilir Mantık Elemanları incelenmiştir. Üçüncü bölümde ise VHDL programlama dilinden bahsedilerek CPLD'nin programlanmasından bahsedilmiştir. Tezin dördüncü bölümünde ise WebPack ISE ve ModelSim XE programları kullanılarak tasarım ve simülasyon yapılması gösterilmiştir. Tezin asıl amacı ise XILINX CPLD'leri yakından tanıtmak ve bunların Xilinx Project Navigator programıyla programlanmasını, programın entegre devreye aktarılmasını ve bu CPLD'lerle devre tasarımı yapılmasını göstermektir.

2. PROGRAMLANABİLİR MANTIK ELEMANLARI

Programlanabilir mantık elemanları (PLD) kullanıcılar tarafından programlanabilen ve bir mantık devresi yerine kullanılabilen entegre devrelerdir. PLD içerisinde bu programlamayı mantık kapıları ve flip-floplar arasındaki bağlantı kablolarında yer alan birleştirici elemanların konumlarının belirlenmesi sağlar. Bu birleştirici elemanlar genelde bir diyot ve sigortadır.



Şekil 2.1. PLD içinde bir birleşme noktası

Şekil 2.1 bir bağlantı noktasındaki diyot-sigorta çiftini göstermektedir. Bu çift eleman aslında bir anahtarlama işlemi yapmaktadır. İstenen fonksiyonu gerçekleştirmek için değişik noktalardan bağlanabilirler. Bu konumlandırma PLD programlayıcılar ile yapılmaktadır. Buradaki programlama ise tamamen donanım programlamasıdır. Çünkü bir PLD programlanırken, sadece içindeki elemanları birbirine bağlayan anahtarların konumu değiştirilmektedir. Programlanmamış bir PLD içerisinde tüm anahtarlar kapalı konumdadır ve istenen tasarıma göre bağlantı olmaması gereken yerlerdeki anahtarlar açık duruma getirilir. Anahtarlar çoğu kaynakta sadece sigorta şeklinde gösterilmektedir. Öyle ki anahtarın kapalı konumu sigortanın bozulmamış haline, anahtarın açık konumu ise sigortanın yanmış haline karşılık gelmektedir.

PLD'ler genelde adres çözümleme (Adress Decoding) için kullanılırlar ve yerine kullanıldıkları 74XX serisi entegre devrelere göre birkaç avantaj sağlarlar.

- Birden fazla entegre devre yerine bir entegre devre kullanılır,
- Tek entegre devre kullanıldığından daha az yer kaplarlar,

- Daha az bağlantı kablosu kullanırlar,
- Düşük gerilimle çalışırlar,
- Daha az güç harcarlar,
- Bir diğer avantajı ise PLD içindeki tasarımın esnek olmasıdır.

PLD'lerdeki tasarım esnekliği yapılması istenen bir değişiklik olduğunda, bu sadece PLD'li tasarımının çeşitli programlarla değiştirilerek yada yalnızca PLD'nin, istenen yeni tasarıma uyarlanmış bir başka PLD ile değiştirilmesiyle sağlanabilecektir. Bağlantı kablolarında hiçbir değişiklik olmayacak aynı kalacaktır. Aynı değişiklik entegre devre grubuyla kurulmuş tasarımda yapılmak istenseydi bu kez kullanılan entegre devre sıraları değişebilecek bu da kabloların yerlerinin değiştirilmesini gerektirecekti. Bir mantık tasarımında defalarca değişiklik yapılmak istenebilir ve her defasında kabloların yerleriyle oynamak mutlaka hatalara ve zaman kaybına neden olacaktır. PLD'leri dijital sistemlerin tasarımında kullanmanın avantajı ise karmaşık mantık fonksiyonlarını tek bir entegre devre içerisinde gerçekleştirebilmektir.

Entegre devreleri içerisindeki eleman yoğunluğuna göre dört gruba ayırabiliriz;

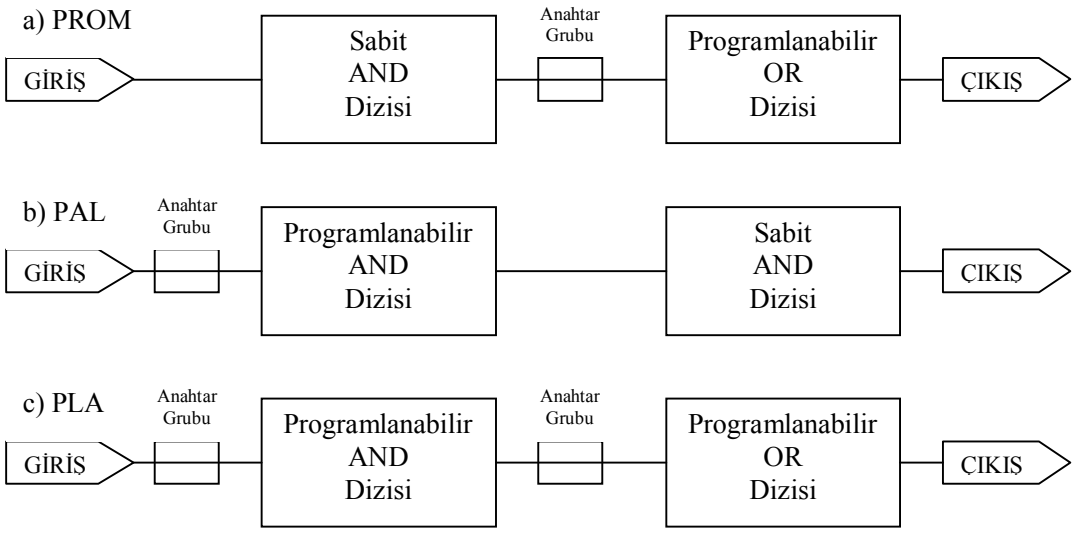
- SSI (Küçük Ölçekli Entegreleme, Small Scale Integration),
- MSI (Orta Ölçekli Entegreleme, Medium Scale Integration),
- LSI (Geniş Ölçekli Entegreleme, Large Scale Integration),
- VLSI (Çok Geniş Ölçekli Entegreleme, Very Large Scale Integration).

Tüm bu büyüklük tanımları aslında entegre devrelerin içinde kullanılan eleman yoğunluğunu göstermektedir. Örneğin SSI entegre devresinde 1000 kadar transistör kullanılıyorsa VLSI entegresinde milyonlarca transistör kullanılmaktadır. SSI ve MSI entegre devreleriyle herhangi bir mantık fonksiyonunu gerçekleştiren devre oluşturulabilir. Ancak birçok küçük entegre devre kullanmak yerine birkaç büyük entegre devre kullanılarak da aynı sonuç elde edilebilir. Bu nedenle çok geniş içerikli uygulamalar için istenen mantık fonksiyonunu gerçekleyecek devre bir VLSI entegre

devresine yerleştirilmektedir. Fakat bu yaklaşımın iki dezavantajı bulunmaktadır; çok uzun fabrikasyon süreci gerektirmesi ve çok pahalı olması nedeniyle programlanabilir LSI ve VLSI entegreleri geliştirilmiştir. Programlanabilir mantık elemanı adı verilen bu elemanlar fabrikada, daha sonra kullanıcı tarafından programlanma olanağını sağlayacak şekilde üretilirler (5).

PLD'ler için çeşitli teknolojiler kullanılsa da temel programlama prensibi, çeşitli ROM'ların (Sadece Okunabilir Bellek, Read Only Memory) içinde bilgi depolamak için kullanılan yöntemlere benzerdir. Ayrıca PROM'lar (Programlanabilir Sadece Okunabilir Bellek, Programmable Read Only Memory) genel bileşik mantık fonksiyonları yerine getirmek için bir PLD olarak kullanılabilir.

Şekil 2.2'de üç çeşit PLD'nin temel yapısı verilmiştir. PROM, PLA ve PAL her üçü de VE ve VEYA mantık dizilerine sahip olmalarına rağmen bu dizilerin programlanabilir yada sabit olmasına göre isimlendirilmişlerdir (6).



Şekil 2.2. PROM, PAL ve PLA temel düzenlemeleri

2.1. Programlanabilir Mantık Elemanlarının Avantajları

Programlanabilir mantık elemanları, kullanıcılara bir çok avantaj sağlamaktadır. Programlanabilir mantık elemanlarının başlıca avantajları şunlardır:

1. PLD kullanıcılara tasarım süresinde çok daha fazla esneklik sağlar çünkü tasarımın ana teması programlama dosyasının değişikliğidir. Program dosyasının değişmesiyle oluşan yeni tasarım hiçbir ek elemana gerek duymadan anında devre üzerinde görülebilir,
2. PLD'ler istenen bir anda silinebilir ve tekrar programlanabilir,
3. Tek bir PLD ile çok çeşitli mantık devreleri tasarlanabilir,
4. Çok geniş ölçekli mantık uygulamaları tek bir PLD elemanı içerisinde gerçekleştirilebilir.

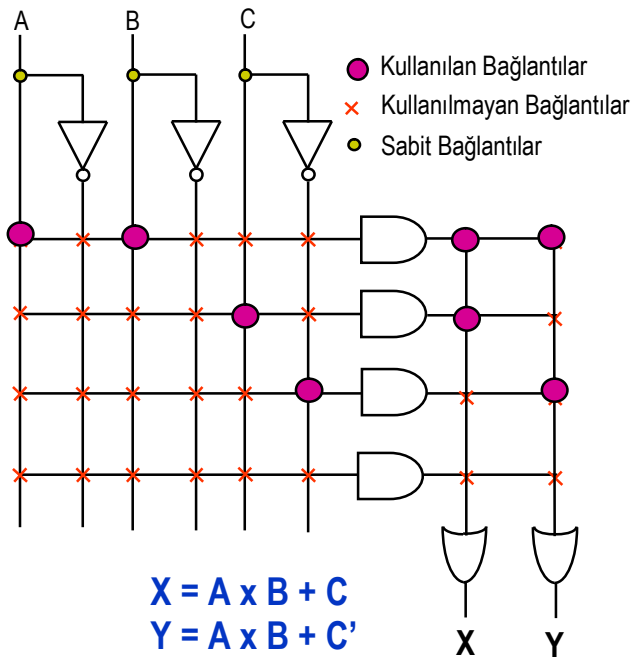
2.2. Temel PLD Yapısı

Bir PLD tipik olarak bir mantık bloğu matrisi ve bunu saran bağlantı kablolarından oluşmaktadır. Bu mantık blokları temel mantık kapılarından ve flip-flop gibi elemanlardan yada bu tür kapı ve elemanların kombinasyonundan oluşmaktadır. PLD içindeki kapılar VE ve VEYA mantık dizileri olarak ayrılmakta ve bu diziler uygun şekillerde birleştirilerek çarpımların toplamı şeklinde mantıksal fonksiyonlar gerçekleştirilmektedir.

Her bir PLD'nin içinde birbirlerine tamamen bağlı makrohücre grupları bulunmaktadır. Bu hücreler içinde de tipik olarak bir grup birleştirilebilir mantık kapıları ve bir flip-flop bulunur. Bir başka deyişle her bir hücre içerisinde küçük bir 'boolean' mantık eşitliği kurulabilir. Bu eşitlik bazı ikilik girişleri, ikilik çıkışlara bağlayacak ve gerekirse diğer saat sinyaline kadar flip-flop içerisinde saklanacaktır. Hücreler içindeki mantık kapılarının ve flip-flopların öznitelikleri her bir imalatçıya ve ürün ailesine göre farklı olabilecektir ama genel düşünce aynı kalacaktır. PLD'ler en küçük ve en basit programlanabilir mantık elemanları olmasına rağmen diğer

büyük ve karmaşık programlanabilir mantık elemanlarının temel yapısını ve düşüncesini oluştururlar.

Temel olarak bir PLD birbirlerine anahtarlarla bağlanmış henüz işlenmemiş mantık hücrelerinden oluşmaktadır. PLD'lerin çoğu bir 'VEYA' mantık dizisini süren bir 'VE' mantık dizisi içerir. Bu VE/VEYA dizileri kullanılarak, çarpımların toplamı şeklinde ifade edilen mantıksal bir fonksiyon gerçekleştirilebilir. PLD'nin programlanması sırasında istenen tasarımı gerçekleyecek bağlantı için gerekli anahtarlar, mantık kapıları ve flip-flopları birbirlerine bağlarken diğerleri açık konumda kalacaklardır.



Şekil 2.3. Basit PLD ile yapılan 3 giriş 2 çıkış örnek tasarım

2.3. Programlanabilir Mantık Elemanı Çeşitleri

Programlanabilir mantık, bir mantıksal devre gibi bağımsız çalışan flip-flop linklerinin, programlanabilecek şekilde bir araya gelmeleri ile oluşur. Hafıza hücreleri mantıksal performans fonksiyonlarını kontrol eder ve bunların çeşitli mantıksal fonksiyonlar ile nasıl bağlanacağını tanımlar. PLD'lerin zamanla farklı

çeşitleri de ortaya çıkmıştır. Bu çeşitliliği; küçük bir mantıksal ifadeyi gerçekleyen küçük elemanlardan, bütün bir işlemci merkezini ve arabirimlerini oluşturabilen, kontrol edebilen son teknoloji FPGA' lara kadar sıralayabiliriz. Çeşitli cihazlar farklı mimariler kullansa da hepsi aynı temel düşünceye sahiptir.

Programlanabilir mantık dizileri(PLA) 70'li yılların başlarında geliştirilmiştir. PLA'lar, içlerindeki 'VE' ve 'VEYA' dizilerinin programlanması ile doğrudan çarpımların toplamı şeklindeki mantıksal ifadeleri gerçekleştirebilirler. Zamanla 70'li yılların sonuna doğru PLA içindeki birçok çarpım yada toplam terimi hiç kullanılmadığından daha ekonomik olan PAL elemanları geliştirildi. Bu elemanlarda toplam terimleri sabitlendi ve kullanıcının sadece çarpım terimlerini programlamasına izin verildi. Daha sonra PAL'ler gibi düzenlenebilen, birçok kez silinip programlanabilen ve diğer bazı fonksiyonları içeren genel dizi mantığı GAL elemanları geliştirildi. Bunun akabinde PALCE elemanları geliştirildi. Bu elemanlar GAL fonksiyonlarına sahiptir ve birçok kez silinip programlanabilmenin tüm önemli özelliklerini taşımaktadır. EPLD, FPGA ve CPLD çeşitli üreticiler tarafından geliştirilen gelişmiş PLD elemanlarıdır. Günümüzde en fazla kullanılan mantıksal devre programlama tipleri aşağıda gösterilmiştir.

1. Basit Programlanabilir Mantıksal Devreler, (SPLD)
2. Kompleks Programlanabilir Mantıksal Devreler, (CPLD)
3. Alansal Programlanabilir Kapı Dizisi, (FPGA)
4. Alansal Programlanabilir Bağlantı Dizisi, (FPIC)

2.3.1. SPLD (Simple Programmable Logic Device)

Aşağıda başlıca SPLD çeşitleri gösterilmiştir.

- PAL (Programlanabilir Dizi Mantığı)
- PLA (Programlanabilir Mantık Dizisi)
- PLD (Programlanabilir Mantık Devreleri)

SPLD'ler programlanabilir mantık formları içinde en ucuza mal olan ve en az kapasiteye sahip olanlarıdır. Bir SPLD tipik olarak 22x4 anahücre ve 7400 TTL dizisinden oluşur. Anahücrelerden her biri devre içinde diğerleri ile bağlıdır. Çoğu SPLD'ler EPROM, EEPROM ve FLASH devreleri gibi fonksiyonelliği tanımlamak için hem yazılabilir (fuses) hem de değişken olmayan (yazılamayan) hafıza hücreleri kullanırlar.

2.3.2. CPLD (Complex Programmable Logic Device)

Entegre devre yoğunlukları yükseldikçe PLD üreticileri için ürünlerini CPLD adı verilen daha büyük (mantık ve işlemsel büyüklük, fiziksel değil) elemanlara doğru yavaş yavaş geliştirmeleri kaçınılmaz oldu. CPLD'ler birçok pratik uygulama için, tek bir entegre devre içerisinde birçok PLD (genelde PAL) gibi düşünülebilir. Daha büyük CPLD' ler daha fazla mantıksal devrenin yada daha karmaşık mantıksal tasarımların uygulanmasına izin vermektedirler. Aslında bu entegre devreler düzinelerce 74XX serisi entegre devrenin yerini alabilecek kadar yeterli büyüklüktedirler.

CPLD'ler PLD'lere göre daha büyük ve kapsamlı tasarımlarda kullanılabildiğinden, potansiyel kullanımları çok çeşitlidir. Adres çözümlemesi gibi küçük uygulamalarda kullanılmaktadır fakat çoğunlukla 'yüksek performans kontrol mantığı' ve 'karmaşık sonsuz durum makinaları' uygulamalarında kullanılırlar. Kullanım alanlarına güncel bir örnek verecek olursak Mars'a araştırma amaçlı gönderilen "*pathfinder"* robotunun tasarımında da CPLD kullanılmıştır (7-13).

CPLD'nin birçok kullanım alanı FPGA'nın (Alansal Programlanabilir Kapı Dizisi, Field Programmable Gate Array) potansiyel kullanımları ile çakışmaktadır. Ticari olarak CPLD'ler yüksek performans istendiğinde FPGA'lara göre tercih edilirler. Öte yandan CPLD'nin daha az esnek iç mimarisinden dolayı, üzerindeki gecikme daha kısa ve önceden tahmin edilebilirdir.

CPLD'ler genellikle kontrol ve yönlendirme tasarımları için pinler arası performansa bağlı olarak en iyi tasarım seçimidir. Makro hücrelerindeki iyi soğutma özellikleri karmaşık ve yüksek performanslı makinelere karşı onları iyi tasarlanmış şekle getirmiştir.

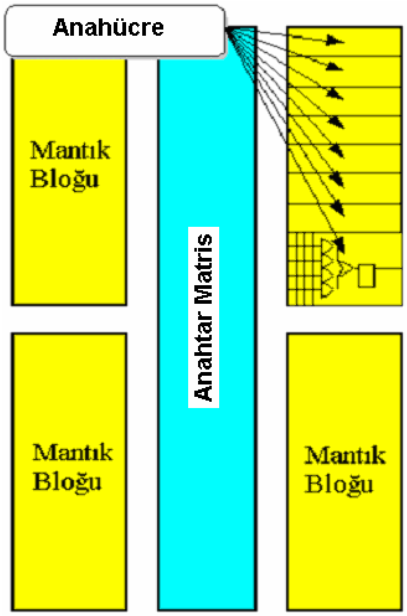
Alt çeşitleri aşağıda gösterilmiştir.

- EPLD (Erasable Programmable Logic Device)
- EEPLD (Electrically-Erasable Programmable Logic Device)
- MAX (Multiple Array matriX, Altera)

CLPD ile SPLD arasında bir fark vardır. CPLD'nin daha fazla kapasiteye sahip olmasıdır. Tipik bir CPLD, iki tane 64 SPLD devresine eşittir. Bir CPLD tipik olarak birkaç yüz anahücre bloğundan oluşur. 16x8 grup anahücre daha büyük fonksiyon blokları içinde birlikte gruplandırılmıştır. Fonksiyon bloğu içindeki anahücreler tamamen birbiri ile bağlantılıdır. Eğer bir devre çoklu fonksiyon bloğu içeriyorsa, o zaman fonksiyon blokları daha gelişmiş bir şekilde birbirine bağlanmış demektir. CPLD'lerin hepsi fonksiyon bloklarının içinde birbirine bağlantılı değildir. Bu satıcı firma ile aile özelliklerine bağlıdır. Fonksiyon bloklarının arasında %100' den daha az bağlantı olması demek, tekrar gözden geçirilmiş tasarımlar arasındaki aynı pin-out çıkışlarının arasında problemler olabileceği veya devrenin yönlendirme yapabilme şansının azalacağı demektir.

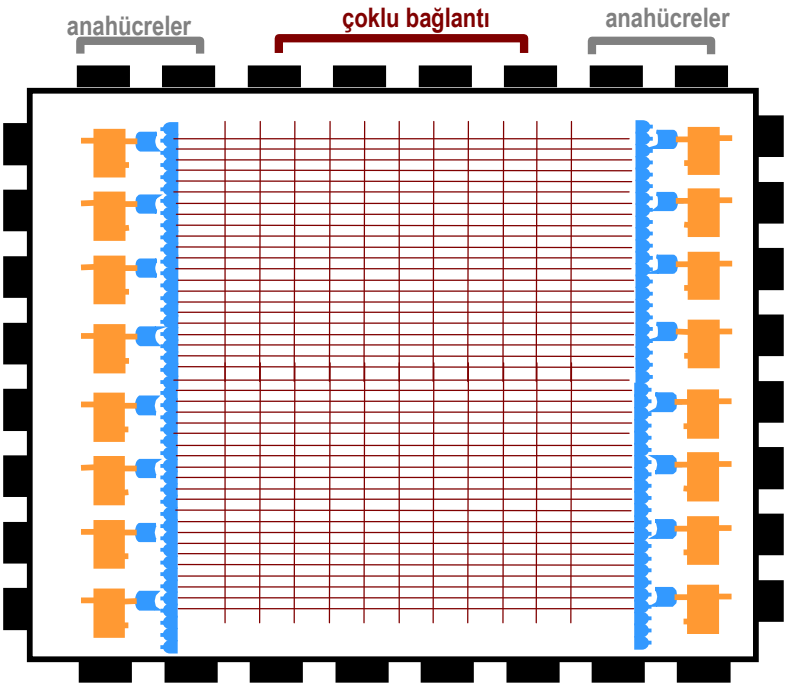
CPLD; programlanabilir anahtar matrisleri yoluyla birbirine bağlanmış mantık blokları olan çoklu PAL devrelerinden oluşur. Tipik olarak her mantık bloğu mimariye bağlı olarak 16x4 anahücreden oluşur (14).

CPLD'ler yüksek yoğunluk arayan SPLD tasarımları için doğal bir geçiş yoludur. CPLD'ler PAL mimarisine sahiptir ve genellikle dört veya daha fazla PAL, bir CPLD içine rahatlıkla sığabilir. Çoğu CPLD'ler ABEL, CuPL, PALASM, VHDL gibi SPLD geliştirme dillerini destekler.



Şekil 2.4. CPLD mimari Blok yapısı

CPLD'ler genellikle kontrol ve yönlendirme tasarımları için pinler arası performansa bağlı olarak en iyi tasarım şeklidir. Anahücrelerdeki iyi soğutma özellikleri karmaşık ve yüksek performanslı makinelere karşı onları iyi tasarlanmış duruma getirmiştir.



Şekil 2.5. CPLD'nin mimari görünümü

Bazı mimarilerde anahücre içinde gerekli olan, fakat yeterli sayıda bulunmayan PT(Product Term) çıkış sayısını tutturmak için anahücreler eklenebilir ve bunlar kullanılarak PT ihtiyacı karşılanabilir. Buda CPLD devrelerini daha geniş uygulama çeşitleri için yararlı hale getirir. Yalnız, anahücrelerden ödünç PT'ler yüzünden sistemin verimi düşer. Ödünç alınan PT'ler sistem içinde gecikmelere yol açar.

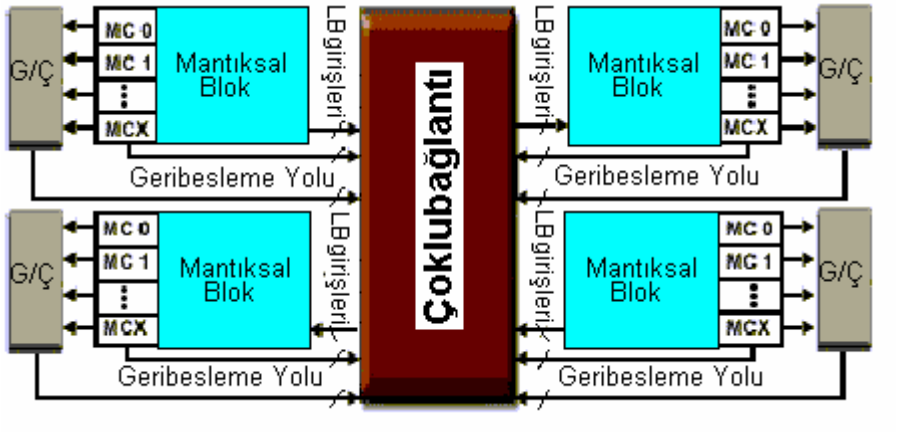
Mimariler arasındaki bir diğer fark ise anahtar matris arası bağlantıların sayısıdır. Bütün olası bağlantı durumlarını destekleyen bir anahtar matris mimariler arasında en iyi olandır. Kısmen popüler olan bir anahtar matris çoğu bağlantıyı destekleyebilir. Anahtar matris içindeki bağlantı sayısı, bir tasarımın verilen devre içine ne kadar kolay uyum sağlayabileceğini belirler. Tam çoğullamalı anahtar matrisi ile bir tasarım, ana devre kaynaklarında kullanılan giriş-çıkış birimlerinin yönlendirilmesi sağlanır. Genellikle tam çoğullamalı anahtar matrisindeki gecikmeler, belirlenmiştir ve önceden tahmin edilebilir.

Yarım çoğullamalı anahtar matris ile oluşturulmuş devre kompleks tasarımlar için yönlendirme problemlerine yol açar. Yinede, farklı bir pin çıkışı kullanılmadan bu devreler içinde yapılacak değişiklikler zor olabilir. Belirlenmiş bir pin çıkışına yönlendirme yağılması önemlidir. Programlanabilir mantıksal devrelerin içeriğini değiştirmek, yani bir devre planı çizmekten daha kolaydır. Yarım çoğullamalı anahtar matristeki gecikmeler, çoğu FPGA devrelerine benzer olarak önceden belirlenemez ve tahmini daha zordur. Yarım çoğullamalı anahtar matris bazı kısıtlamalara sahipse de bunu yapmak daha az masraflıdır.

CPLD'ler üç işlem teknolojisinden biri kullanılarak üretilirler. (EPROM, EEPROM yada FLASH) eğer ışıkla silinebilir değillerse EPROM tabanlı CPLD'ler bir defa programlanabilir. Birçok programcı, üretici yada distribütörün tercihi, EPROM tabanlı CPLD'lerdir.

Genellikle CPLD'ler fonksiyonelliği tanımlamak için CMOS, EPROM, EEPROM yada FLASH bellekler gibi sabit hafıza hücreleri kullanırlar.

CPLD, tümüyle programlanabilir olan VE/VEYA dizilerinin ve anahücre yığınlarından oluşan bir bloktur. VE/VEYA mantıksal dizileri tekrar programlanabilirdir ve mantık fonksiyonlarının çoğunu yapabilecek kapasitededir. Anahücreler ardışık mantık birimlerini oluşturan fonksiyonel bloklardır ve birkaç farklı geri besleme yoluyla doğruluk ve tamlık için fazladan esnekliğe sahiptir (14).



Şekil 2.6. Bir CPLD bloğu

Geleneksel olarak CPLD'lerin mimarileri performansını yükseltmek için analog yükselteçleri kullanırlar. Xilinx tarafından geliştirilen CR-II CPLD'ler çok düşük güç gerekliliklerinden aynı performansı gösterecek yeni bir buluş olan tümü dijital bir yapı kullanırlar. Bu da tasarımcıların CPLD mimarisini hem yüksek performanslı, hem de düşük güçlü tasarımlar için kullanılabilmesini sağlar.

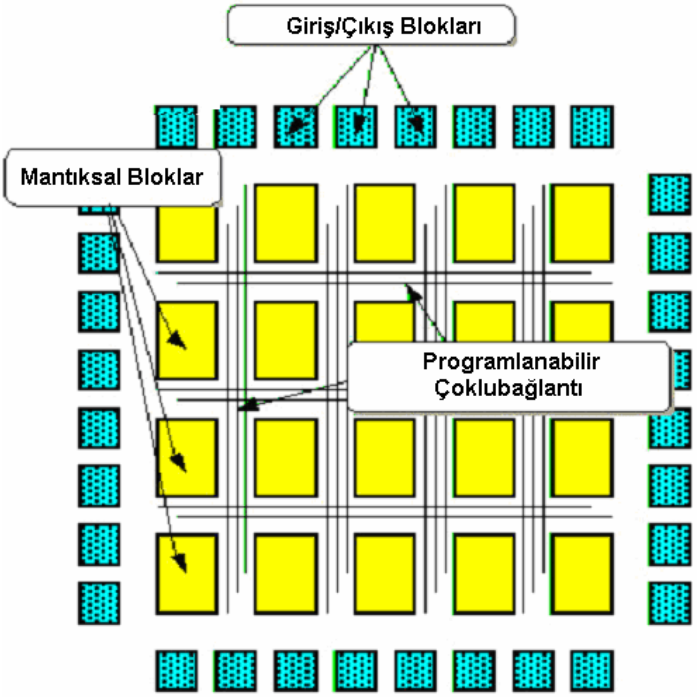
2.3.3. FPGA (Field Programmable Gate Array)

FPGA'lar kullanıcı tarafından herhangi bir dijital devre için basit programlama metotlarıyla programlanabilir en karmaşık entegre devre türüdür. Bir kapı dizisi içerisinde özel entegre devre tasarımları için gerekeli olan eleman ve bağlantıları içeren bir VLSI entegre devresidir.

FPGA'lar CPLD ve SPLD'den çok farklıdırlar ve daha yüksek mantık kapasitesi sunarlar. Mantık blokları dizisinden oluşan bir FPGA programlanabilir giriş-çıkış

blokları ile çevrelenmiştir ve programlanabilir interconnect (çoklubağlantı) ile bağlantılıdır. Tipik bir FPGA on binlerce mantıksal bloktan ve daha büyük sayıda flip-flop' tan oluşur.

FPGA'lar CPLD'lerden daha fazla mantık yoğunluğu sağlamasına karşın, statik RAM (SRAM) yapısında bellek kullanmaları bir dezavantajdır. Çünkü SRAM bellekler içerdikleri bilgileri elektrik kesildiğinde kaybederler. Bu nedenle FPGA'larda, mantıksal program entegre devreye her enerji uygulandığında tekrar yüklenecektir.



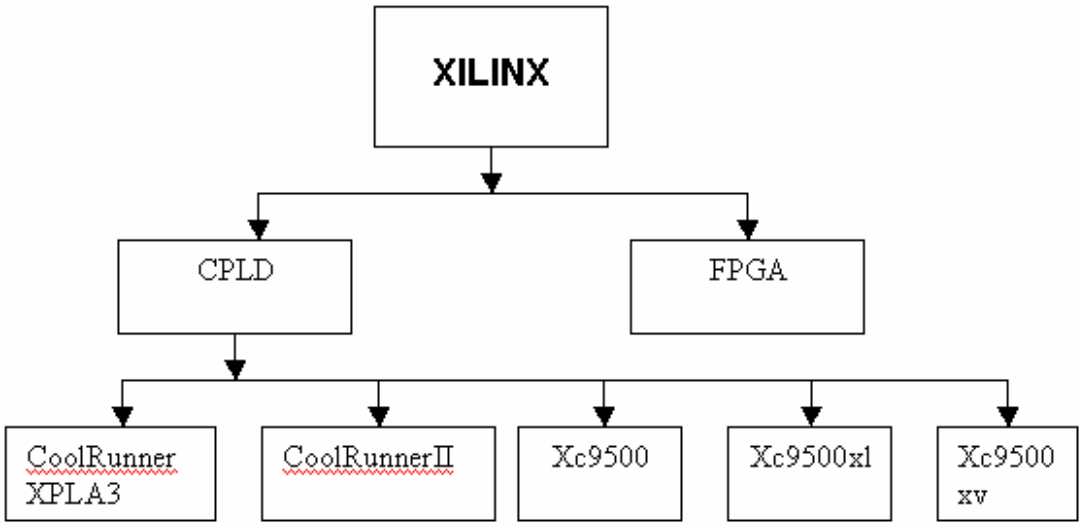
Şekil 2.7. FPGA iç yapısı

2.3.4. FPIC(Field Programmable Interconnect Device)

FPIC aslen bir mantık devresi değildir, programlanabilir elektrik tertibatının devresidir. FPIC; devre içindeki bir pini diğerine, programlanabilir bağlantı şeklini kullanarak bağlar. FPIC'ler hem SRAM hem de anti-fuse programlama teknolojisini kullanır.

2.4. Xilinx Hakkında Bilgi

Xilinx, ileri düzeyde entegre devreler, yazılım ve tasarım gibi komple programlanabilir mantıksal çözümlerde uzman olan bir firmadır. 1984 yılında kurulan ve merkezi California'da bulunan firma, FPGA ve CPLD cihazlarını icat etmiş bulunmaktadır. Günümüzde bu cihazlara duyulan talebin yarısından fazlasını karşılamaktadır. Xilinx CPLD ailesi üyeleri aşağıda bir tablo ile gösterilmiştir.



Şekil 2.8. Xilinx CPLD ailesi

2.4.1. Xilinx CPLD avantajları

CPLD'ler yüksek kapasiteleri sayesinde sistem tasarımında pek çok farklı ve yararlı fonksiyona sahiptir. Bunları sayacak olursak;

Yeniden Programlanabilme: Tasarımları fazla masraf gerektirmeden kolayca değiştirebilme, biçimsel sistemler inşa etme, sistem fonksiyonlarını istenilen zaman ve yerde güncelleme olanaklarına sahiptir

Basitlik ve Kullanım Kolaylığı: CPLD kullanılarak tasarım yapma basit ve kolaydır. Mevcut tasarım akışına kolaylıkla ekleme yapılır ve uyum sağlanabilir.

Düşük Maliyet: CPLD'nin fiyatları çok düşük olduğundan imalat maliyetlerini de düşürmektedir.

Değişken olmama: CPLD tasarımında düşük güç kullanılarak yapılan programlar bile saklanabilir, böylece depolanmış bilgileri çalmak imkansız hale gelir. Güvenlik çok yüksektir.

*Yüksek Hız***: Y**üksek işlem hızına sahiptirler ve hız değişikliğini program ile gerçekleştirmek mümkündür.

Düşük Güç ve Voltaj: Düşük enerji tüketimi sayesinde içerisindeki bilgiler yıllarca saklanabildiği gibi; özellikle yüksek performanslı ve küçük boyutlu taşınabilir sistemlerde (palm, cep telefonu v.b) tercih edilirler.

Değişik Tipte Paketleme: Gelişmiş silikon teknolojisi ile küçültülmüş entegre devre tasarımı, yüksek performans özellikleri göstermektedir. Aynı entegre devre farklı kılıflarda üretilmektedirler,

Yüksek I/O Pin sayısı: Xilinx CPLD' lerde bulunan yüksek pin sayısı tasarımda kolaylık sağlamakla birlikte aynı pin giriş-çıkış gibi farklı amaçlar içinde kullanılabilmektedir.

Ucuz ve Güçlü Tasarım: Benzeri entegre devrelere göre oldukça ucuz olması pazardaki payını güçlendirmektedir

3. CPLD PROGRAMLAMA VE VHDL

Bir PLD'yi programlamak için öncelikle anahtar haritasının bir yükleme dosyası aracılığıyla programlayıcı içine girilmesi gerekmektedir. Bu anahtar haritası programlayıcıya hangilerinin açık hangilerinin kapalı olacağını gösterir. PLD üreticilerinin çoğu kullanıcılara, anahtar haritalarını herhangi bir bilgisayarda girebilmek için serbest programlama yazılımları sunmaktadırlar. Harita bir kez girildikten sonra programlayıcıya bu haritayı PLD'nin anlayacağı forma dönüştürme komutu verilir. Daha sonra programlayıcı soketine yeni bir PLD yerleştirilir ve programlayıcıya programlamayı başlatması için komut verilir. Programlayıcı, bu komutla tüm anahtarları sırasıyla adresleyerek bunları açık konuma alır yada kapalı konumda bırakır. Programlama tamamlanınca, programlayıcı yeni programlanmış PLD'yi test ederek, programlamanın doğru olup olmadığını belirler.

Bir tasarımı PLD içerisine yüklemek için standart bir tasarım metin giriş yada şematik gösterim formatı kullanılmalıdır. PROM gibi basit PLD'lerde, tasarımın metin şeklinde girilmesi, PLD ye yüklenecek dosyanın standart bir formda oluşturulmasını gerektirir. Daha karmaşık PLD'ler için, metin giriş formatı bir programlama diline benzer ve bu nedenle "Donanım Açıklama Dili (HDL)" şeklinde tanımlanır. Tasarım şematik yada metin yoluyla bir kez bilgisayara girildikten sonra, yazılım araçları kullanılarak PLD için yükleme dosyalarına çevrilebilir (15).

Donanım açıklama dillerini genel ve özel amaçlı olmak üzere ikiye ayırabiliriz. VHDL (Yüksek Seviyeli Donanım Açıklama Dili, Very High Speed Hardware Description Language) genel amaçlı bir donanım açıklama dilidir. PALASM (PAL Derleyici, PAL Assembler) ve ABEL (Gelişmiş Boolean Eşitlikleri Dili, Advanced Boolean Equivalent Language) ise özel amaçlı HDL türleri olup bu diller ile herhangi bir PLD içine, programlanacak olan tasarım için gerekli açıklamalar (yükleme dosyası) girilebilir.VHDL dili aynı zamanda ABEL ve PALASM gibi PLD'ler içine tasarım programlarını yüklemek için kullanılabilir. Ancak ABEL ve PALASM daha basit ve daha özlü programlardır. PALASM ve ABEL ile yazılan

tasarım açıklamaları, tipik olarak tasarım ismi, tarihi gibi temel tasarım başlık bilgilerini içerirler.

Bir PLD dijital bir mantık devresi olarak kullanılmak istendiğinde, içindeki mantık blokları bu dijital devrenin elemanlarına çevrilir. Bu çevirme işlemi, yatay ve dikey bağlantı kablolar arasında özel bağlantı noktaları (bkz. Şekil 2.1) sağlanarak gerçekleştirilir. Bu bağlantı noktaları anahtar konumlarının belirlenmesiyle gerçekleştirilir.

Programlama yapılırken bu özel noktalara erişim ve bağlantının sağlanması için, PLD'nin bazı girişleri, satır yada sütun adres çoklayıcısı olarak kullanılabilir. Satır yada sütun çoklayıcıları, özel bir dikey yada yatay bağlantı kablosunu seçmek için kullanılır. Böylece istenen birleşme noktası belirlenir. Birleşme noktası bir kez belirlenince özel programlama gerilimleri yada diğer yöntemler iki kablo arasındaki bağlantıyı kesmek yada sağlamak için kullanılabilir. Yatay ve dikey tüm birleşme noktalarını belirlemek ve uygun gerilimleri uygulamak için tipik olarak PLD programlayıcı, PROM programlayıcı gibi özel programlayıcı elemanlar kullanılır. PLD'nin istenen dijital devreyi algılaması için gerekli birleşme noktaları bilgisi, seri yada paralel portlarla bir bilgisayardan bir PLD programlayıcıya aktarılabilmektedir.

Programlanmamış bir PLD'de tüm anahtarların kapalı konumdadır. Eğer birleşme noktasındaki bağlantı (bkz. Şekil 2.1) tasarım için gerekli değil ise, programlayıcı bu noktaya yüksek gerilim (10…30V) uygulayarak sigortanın yanmasını sağlar ve sonuçta iki hat arasındaki bağlantı kesilmiş olur. Programlama daha sonra gerekli olmayan tüm bağlantıların bu şekilde iptal edilmesi ile devam eder. Birleşme noktalarındaki bağlantının bu şekilde iptal edilmesi veya aynı bırakılması *sigortalanabilir hat* metodudur.

Sigortalanabilir metod ile programlanabilen PLD'ler, yanan sigortalar tekrar eski konumuna alınamayacağı için yalnızca bir kez programlanabileceklerdir. Tekrar programlanabilen ve silinebilen PLD'ler için diyot-sigorta hattı yerine mosfet ya da transistör kullanılmaktadır. Bir kez programlanabilen PLD'ler, diğerlerine nazaran

daha ucuz ve daha yoğundur ayrıca daha yüksek performansa sahiptir. Ancak sadece bir kez programlanabilmeleri büyük dezavantajdır.

3.1. Donanım Tanımlama Dilleri (HDL)

Donanım tanımlama dilleri, sayısal sistemlerin simülasyonu, modellenmesi test edilmesi ve tasarımı amacıyla kullanılırlar. Bazı donanım tanımlama dilleri, sayısal devrelerin şematik gösteriminin yerine geçen sembol ve rakamlar kullanırlar. Mevcut HDL yazılımları, Simulator ve donanım sentezleme programları içerir. HDL'ler, gün geçtikçe karmaşıklaşan tasarımların gerçekleştirilmesini mümkün kılmak amacıyla ortaya atılmışlardır. HDL tabanlı sistem tasarımı, çok sayıda tasarımcının yada tasarımcılar grubunun bir arada çalıştığı büyük projelerin gerçekleştirilmesinde önemli faydalar sağlamaktadır. Çünkü HDL yapısal geliştirmeyi mümkün kılmaktadır. Temel mimari konusundaki kararlar verildikten, temel bloklar belirlendikten ve bunların birbirleri ile olan bağlantıları ortaya konulduktan sonra, tasarım projesi, birbirinden bağımsız alt projelere bölünebilir.

3.1.1. Donanım tanımlama dili kullanmanın avantajları

Donanım tanımlama dillerine dayanan tasarım yönteminin, geleneksel kapı düzeyi tasarım yöntemine göre bir çok avantajı vardır:

1. Tasarım fonksiyonlarının doğruluğu, tasarım sürecinin en başında test edilebilir ve HDL dilinde tanımlanmış bir tasarım üzerinde anında simülasyon yapılabilir.
2. Bir HDL sentezleme aracı kullanılarak, hazırlanan HDL tanımlaması otomatik olarak verilen bir teknolojide gerçeklemeye dönüştürülebilir. Bu adım eskiden mevcut olan kapı düzeyi tasarım problemlerini, devre tasarımına harcanan uzun zamanı ve elle yapılan tasarımda karşılaşılan diğer sorunları ortadan kaldırmaktadır.
3. Sentezleme aracının mantıksal uyumlaştırma özelliği kullanılarak, sentezlenmiş tasarım daha hızlı bir hale dönüştürülebilir. Sentezlenmiş ve uyumlaştırılmış devrelerden kazanılan tecrübeler VHDL tanımlamasına yansıtılabilir.

4. HDL tanımlamaları, bir tasarımın ve fonksiyonun teknolojiden bağımsız dokümantasyonunu mümkün kılmaktadır. Bir HDL tanımlaması, bir netlist yada bir şemaya göre daha rahat okunur ve anlaşılır. Başlangıçtaki HDL tanımlaması, teknolojiden bağımsız olduğundan, aynı tanımlama daha sonra farklı bir teknoloji için kullanılabilir.
5. VHDL, pek çok yüksek seviyeli bilgisayar dillerinde olduğu gibi, oldukça sıkı bir tip kontrolü yapmaktadır. 4 bit genişliğinde işaret bekleyen bir bloğa 3 yada 5 bitlik işaret uygulanamaz. Bu, HDL tanımlaması derlendiğinde bir hataya neden olur. Tip kontrolü, bu hataların sentezleme işleminden önce ortaya çıkarılmasını sağlamaktadır.

3.1.2. Simülasyon

HDL ile yazılan bir tasarım tanımı, bir HDL simülatörü aracılığıyla, modellenen sistemin çalışmasını gözlemek amacıyla kullanılabilir. Tasarımın simülasyonunu yapabilmek için simülasyon programına bir takım test girişleri verilir. Program da daha önce belirlenen aralıklarla, bu test girişlerini, tasarımın modeline uygular ve çıkışları üretir. Bu sonuçlar tasarımcı tarafından gözlenerek modelin, istenilen şekilde çalışıp çalışmadığı belirlenir.

Simulator, tasarımın her aşamasında kullanılabilir. Tasarımın yüksek seviyelerinde yapılan simülasyonda tasarımın sadece fonksiyonel davranışı hakkında bilgi elde edilir. Bu aşamada simülasyon çok hızlıdır, fakat simülasyon sonuçları, tasarımın çalışması ve zamanlaması konusunda çok fazla bilgi içermez. Daha ileri tasarım aşamalarına gidildikçe simülasyon daha fazla zaman alacaktır, fakat simülasyon sonuçları, tasarımın çalışması ve zamanlaması konusunda daha fazla bilgi verir.

3.1.3. Sentezleme

Sentezleme tasarım tanımlamasının herhangi bir seviyeden, daha düşük diğer bir seviyeye çevrilmesi olarak tanımlanabilir. Bu çevirme işlemi, C programlama dilinde, yüksek seviyeli dilin makina diline çevrilmesine benzetilebilir. Sentezleme

sırasında kullanılan araçlar, HDL tanımlama, zamanlama ve alan istekleridir. Sentezleme işlemi tasarımcı tarafından belirtilen teknoloji kütüphanesindeki elemanlar kullanılarak yapılır. Sentezleme sonuçları, optimal devre şeması, beklenen performans ve sentezlenmiş devrenin kapladığı alandır. Aşağıda kısaca davranışsal sentezleme, saklayıcı iletişim seviyesinde (RTL, Register Transfer Level) sentezleme ve mantık sentezlemeden bahsedilmiştir.

Davranışsal sentezleme, C benzeri algoritmik yazılmış tasarımın RTL tanımlamaya çevrilmesidir. RTL seviyesindeki tasarım veri yolları , bellek elemanları ve kontrol birimleri içerir. RTL sentezleme, saklayıcı iletişim fonksiyonlarından, bir ardışıl devre için devre şemasının oluşturulmasıdır. Mantık sentezleme, boolen fonksiyonlarının birleşimsel devre elemanlarıyla gerçeklenmesidir.

3.2. VHDL Donanım Tasarım Dili

VHDL, çok kullanılan donanım tanımlama dillerinden bir tanesidir. 1980'lerin ortalarında, Amerika Savunma Bakanlığı ve IEEE **VHDL** olarak adlandırılan yüksek yetenekli donanım tanımlama dilini geliştirdiler. 1987'de IEEE tarafından standardı oluşturularak (VHDL-87) 1993'de geliştirildi.(VHDL-93) Bu dil aşağıdaki özelliklere sahiptir:

1. Tasarı hiyerarşik bir şekilde basit olabilir.
2. Her bir tasarım elementi, iyi tanımlanmış bir ara yüze (diğer elementlerle bağlantı kurmak için) ve hassas bir davranış (simülasyon için) özelliğine sahiptir.
3. Davranışsal özellikler bir algoritma yada elementin çalışmasını tanımlayan gerçek bir donanım yapısını kullanabilir. Örneğin, element bir algoritma ile başlangıçta tanımlanabilir, gerçeklenen yüksek seviyeli elementlerin tasarımına izin verir. Sonra, algoritmik tanımlama bir donanım yapısına yerleştirilebilir.
4. Uyuşma(concurrency), zamanlama (timing) ve clocking işlemlerinin tamamı modellenebilir. VHDL senkronize ardışık devre yapısını ardışık olmayana göre kontrol altında tutabilir.
5. Mantıksal işlem ve tasarımın zamanlama davranışının simülasyonu yapılabilir.

6. Böylece, VHDL hassas bir şekilde tanımlanan ve simülasyonun dijital sistem davranışlarına izin vermek için dosyalama ve modelleme dili olarak ortaya çıkmıştır.

VHDL sentez alt programları sayesinde lojik devre yapısından VHDL davranışsal tanımlamaları doğrudan oluşturulabilir. VHDL'yi kullanarak, herhangi bir basit kombinasyonel devre veya tam bir mikroişlemci sistemini bir entegre devre üzerinde tasarımını, simülasyonunu ve sentezini yapabiliriz.

3.2.1 Tasarım akışı

VHDL tabanlı tasarım işleminde birkaç basamak vardır, bunlar tasarım akışı olarak adlandırılır. Bu adımlar herhangi bir HDL tabanlı tasarım işlemine uygulanabilir. Büyük lojik tasarımlarda, yazılım programlarındaki gibi, genelde hiyerarşiktir ve VHDL modelleri tanımlamada ve bunların bağlaşımında, iyi bir çalışma çerçevesi (framework) verir.

Son adım modeller için VHDL kodlarının yazılması, bunların bağlaşımı ve bunların dahili detaylarının açıklanması aşamasıdır. VHDL text tabanlı bir dil olduğu yıllarda, prensip olarak bu görevi yerine getirmek için herhangi bir text editörünü kullanmak zorunda idi. Bununla birlikte bir çok tasarım çevreleri bu görevi daha kolay yerine getirmek için özel VHDL text editörlerini kullanır. Böylece editörler, VHDL anahtar kelimesini otomatik tanıyan, sıklıkla kullanılan program yapılarını, şablonlarını kurma, yazım kurallarını otomatik kontrol etme ve derleyiciye tek tuşla ulaşım gibi özellikleri içerir (16).
VHDL derleyicisi yazım kuralı hataları açısından kodların analizini yapar ve diğer modüllerin uygunluğu açısından kodları kontrol eder. Diğer program denemelerinde olduğu gibi, diğer kodları derlemek için kodlama işleminin sonuna kadar beklemek zorunda değiliz.

VHDL simülatörü, fiziksel devreyi inşa etmeksizin çıkışları gözlemenize, tasarım girişlerine uygulamanıza ve tanımlamanıza izin verir. Küçük projelerde, dijital

tasarımlarınızı ev ödevi gibi yapabiliriz; girişleri üretebilir ve çıkışları manuel olarak elde edebilirsiniz. Ama büyük projelerde, VHDL otomatik olarak girişleri uygulayan ve bunları beklenen çıkış değerleri karşılaştıran *Test Benches* oluşturmanıza izin verir.

Simülasyon ile tasarımın kontrol edilmesi büyük bir adımdır. Simülasyonlu çıkışları üreten simüle edilmiş devreyi incelememizi sağlar. Beklenen devre çalışmalarını tanımlar. Bu kategoride tasarım hataları bulmak yüksek öneme sahiptir. Eğer hatalar daha sonra bulunacak olursa, *back-end* diye adlandırılan bütün adımlar tekrarlanmalıdır.

En az iki boyutlu kontrol vardır, fonksiyonel tanımlamada, zamanlama özelliklerine bağlı devrelerin mantıksal operasyonlarıyla çalışılır. Kapı gecikmeleri ve diğer zamanlama parametreleri sıfır olacak şekilde arzu edilir. Zamanlama kontrollerinde göz ardı edilmiş gecikmeleri içeren devrelerle çalışarak, flip-flop gibi ardışık elemanlar için kurma(setup), tutma(hold) ve diğer zamanlama gereksinimlerini tanımlanabilir.

Bu kategorileri tanımlamada üç adımdan bahsedebiliriz; İlki, analiz yapmaktır(synthesis), hedef teknolojiyi düzenlemeye yarayan, VHDL tanımlamalarını elemanlara dönüştürmeyi sağlar. Örneğin, PLD'ler veya CPLD'ler ile analiz yapmaya yarayan aletler iki seviyeli toplamların çarpımı denklemine dönüştürülebilir. ASIC'ler ile kapı listelerine dönüştürülebilir ve bir netlistin nasıl bağlandıklarını tanımlayabilir. Tasarımcı, maksimum lojik düzey numaraları veya lojik bufferların uzunluklarını kullanmak gibi kesin belirli bir teknolojileri tanımlayarak analiz yapan elemana yardımcı olabilir.

Uygun(fitting) basamakta, bir uygun elemanı veya uygunlaştırma haritaları sentezi yapılmış ilkel veya karmaşık eleman kaynaklarından bulunabilir. PLD veya CPLD için mümkün olan AND-OR elementlerinin görev denklemlerinde anlamlı olabilir. ASIC için tasarımdaki kapılar ve fiziksel zorlamaları için bağlantı yollarını kullanmak anlamlı olabilir. Bu *place-and-route* işlemi olarak adlandırılır. Tasarımcı

genellikle bu bölümdeki ek sınırlamaları tanımlayabilir, entegre devre ile modellerin yerleşimi veya harici giriş çıkış pinlerinin tanımlamaları gibi. Son adım, uygunlaştırılmış devredeki tamamlama kontrolleridir. Sadece bu bölümde gerçek devre gecikmelerine bağlı olarak tel uzunlukları, elektriksel yüklemeler ve sonuçsal hassasiyet ile diğer faktörler hesaplanacaktır.

Diğer oluşturma işleminde olduğu gibi, iki basamak ileri bir basamak geri işlemini elde edebilirsiniz. Kodlama işlemi süresince çıkan problemler tasarımcıyı geri gitmeye zorlayacak ve hiyerarşiyi tekrar kontrol etmeyi sağlayacaktır. Çıkan hata kodları, kesinlikle derleme yapmaya ve parça kodlarını tekrar yazmaya zorlayacaktır. En acı verici problemlerden birisi tasarı akışı sonunda beklenmeyen durum ile karşılaşmaktır. Eğer sentezi yapılmış tasarım uygun bir CPLD veya FPGA'yı desteklemiyorsa veya zamanlama gereksinimleri ile karşılaşmıyorsa bütün tasarım yaklaşımını hızlı bir şekilde düşünerek geri dönmek zorundasınız (17).

3.2.2. VHDL'de kullanılan temel yapılar

Bu bölümde, VHDL'de kullanılan temel yapıları ele alacak olursak bunlar:

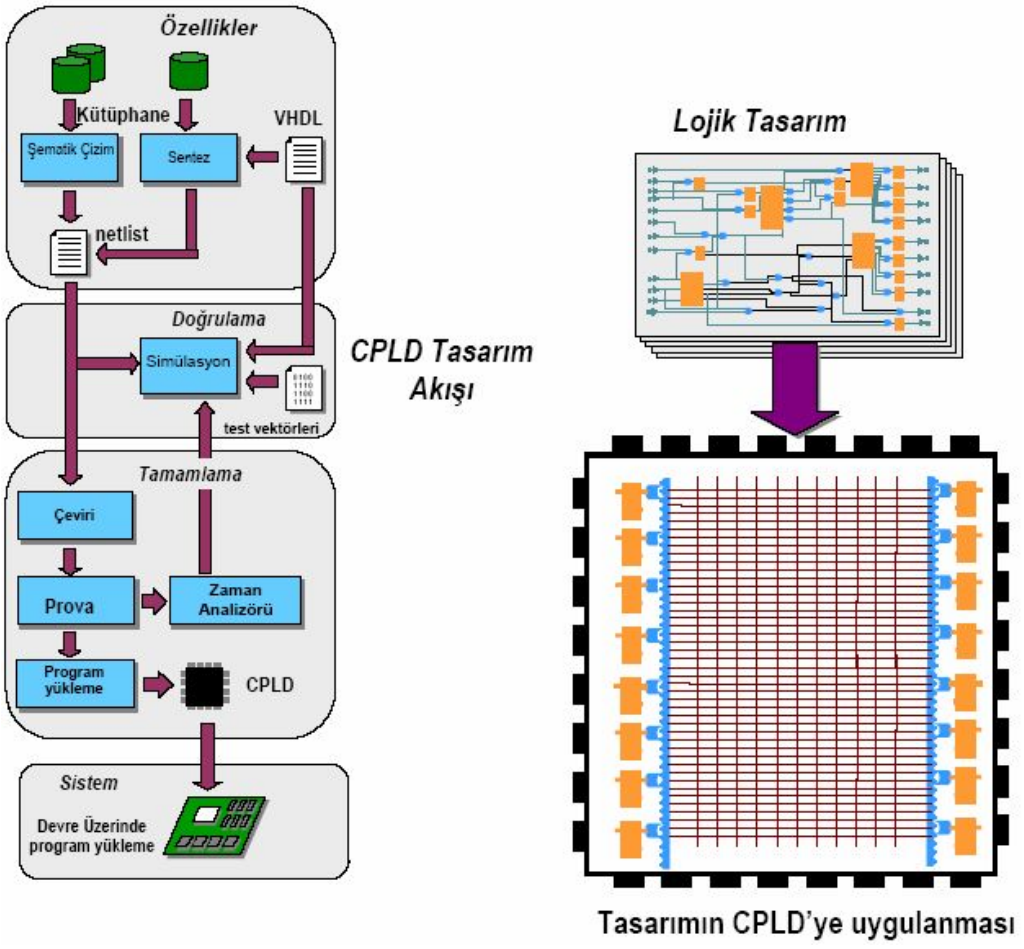
- Modül Bildirimi (Entity Declaration)
- Mimari (Architecture)
- Paket (Package)
- Konfigürasyon (Düzenleme, Configuration)
- Tasarım Kütüphaneleri (Design Libraries)

Sayısal bir sistem, modüllerin bir hiyerarşi içinde birleştirilmesiyle oluşur. Her modül, VHDL'de bağımsız ve tektir. Bir tasarım modülü, giriş çıkışları ve fonksiyonu tamamen belirlenmiş bir donanım yapısını temsil eder. Her modülün iki bölümü vardır; modül bildirimi ve mimari. Modül bildirimi, tasarımın dış bağlantılarını, mimari ise içeride yapılacak işlemleri gösterir. Paket, pek çok modül tarafından kullanılacak genel bilgileri içerir. Düzenleme, yapısal tanımlamada

kullanılacak alt sistemlerin modül ve mimarilerini bir araya getirir. Bir VHDL tasarımı, her biri tasarım kütüphanelerinde bulunan ve önceden tanımlanmış bir çok üniteyi içerir.

4. XILINX CPLD İLE TASARIM VE SİMÜLASYON

CPLD'lerin hızlı çalışmaları, düşük fiyatları ve çok düşük enerji gereksinimleri gibi üstünlükleri nedeniyle tercih sebebi oldukları bilinmektedir. Xilinx CPLD'lerde tasarım aşamaları görüldükten sonra Xilinx firmasının geliştirdiği tasarım programı WebPACK ISE kullanılarak örnek tasarım yapılacaktır. ModelSim XE Simulator programı ise daha sonra kurulan ve WebPACK ISE programı içerisine dahil olarak yapılan tasarımların simülasyonunu gerçekleştirmektedir. Şekil 4.1.de tasarım ve simülasyon için izlenecek yol görülmektedir.



Şekil 4.1. CPLD ile yapılan tasarımda izlenecek yol

WebPACK ISE programı ile iki türlü tasarım yapılabilmektedir. İsteyen kullanıcı programın kendi kütüphanesinden veya kendisi VHDL kodlarını yazmak kaydıyla tasarım yapabildiği gibi isteyen kullanıcılar içinde şematik tasarım yapmak

mümkündür. İki tasarım yolunda da son doğruluk kontrolü yapıldıktan sonra entegre devrenin giriş ve çıkış pinlerinin tanımlanması gerekmektedir. Bu tanımlamadan sonra oluşturulan JED uzantılı dosya CPLD'ye JTAG kablo yardımıyla yüklenebilmektedir.

4.1. Sayısal Sistem Tasarımına Genel Bakış

Basit yapıdaki mantık devrelerin tasarımı genellikle aşağıdaki adımlardan oluşur:

1. Problemin sözle tanımlanması,
2. Durum diyagramının çizimi, (Asenkron ardışıl devreler için ilkel akış tablosunun oluşturulması)
3. Durum indirgeme,
4. Durum kodlama,
5. Gerçekleme.

Bu adımlardan durum indirgeme (ardışıl devredeki flip-flop sayısını etkiler) ve durum kodlama (devrenin birleşimsel kısmının karmaşıklığını etkiler) minimal devrenin elde edilmesinde önemli rol oynarlar. Durum indirgeme ve kodlama için çeşitli yöntemler vardır; fakat bu yöntemlerin hangisi kullanılırsa kullanılsın, tasarlanacak devrenin fonksiyonları ve karmaşıklığı arttıkça tasarım adımlarını insan eliyle gerçekleştirmek zorlaşacaktır.

Şekil 4.2 sayısal sistem tasarımındaki tipik işlemleri gösterir. Tasarım fikri olarak adlandırılan ve gerçekleştirilecek sistemin özelliklerinin ortaya konduğu aşama, sistemin donanımsal gösteriliminin elde edildiği aşamaya gelinceye kadar bir çok dönüşüm geçirir. Her dönüşümün ardından tasarımcı, sonuçları kontrol eder ve istenilenler doğrultusunda gerekli iyileştirmeleri yaparak bir sonraki aşamaya geçer.
Tasarımcı ilk olarak bir tasarım fikri oluşturur. Daha sonra, tasarlanması düşünülen sistemi tamamıyla tanımlayabilen bir yapıya ihtiyaç duyulur. Bu amaçla davranışsal tanımlama yapılır. Bu aşamanın sonunda akış çizelgesi elde edilebilir.

Şekil 4.2. Sayısal sistem tasarımının aşamaları

Tasarımın diğer aşaması veri yolu tasarımıdır. Bu aşamada tasarımcı gerekli saklayıcıları ve mantık birimlerini belirler. Bunlar çift yönlü veya tek yönlü veri yolları ile birbirlerine bağlanırlar. Bir kontrol birimi veri akışını kontrol eder.

Mantıksal tasarım daha sonra gelen aşamadır. Saklayıcıların, veri yollarının, mantık birimlerin ve bunların kontrolünü sağlayan donanımın içerdiği kapı ve flip-floplar arasındaki bağlantılar gösterilir.

Fiziksel Tasarım kısmında ise bir önceki aşamada elde edilen flip-flop ve kapı bağlantıları transistör seviyesine dönüştürülür. En son aşamada ise bu transistör bağlantıları kullanılarak entegre devre üretimi için maskeler oluşturulur ve alansal programlanabilir kapı dizileri ile gerçekleme yapılır.

4.1.1. Tanımlama seviyeleri

Genel olarak problem belirlendikten sonra sistem, birbirleriyle tamamıyla bağlantılı olan belirli alt parçalara ayrılmalıdır. Bu yapılırken, parçaların her biri bir birim olarak düşünülmeli ve birimlerin arasındaki bağlantılar tamamen belirlenmelidir. Bu parçalama işlemine basit mantık kapılara ulaşıncaya kadar devam edilir.

Tanımlama genel olarak şu seviyelere ayrılır; sistem seviyesi, saklayıcı iletişim seviyesinde, mantık seviyede ve devre seviyesi. Bu seviyeler tanımlanırken üç farklı tanımlama yöntemi kullanılır; davranışsal, yapısal ve fiziksel. Davranışsal tanımlamada, sistemin nasıl gerçekleneceği göz önüne alınmaksızın sistem fonksiyonu tanımlanır. Yapısal tanımlamada, daha önce tanımlanmış elemanların birleşimi tanımlanır.

Tasarım şekli ve devre karmaşıklığına göre tasarımcı bu tanımlama seviyelerinden herhangi birini veya hepsini kullanabilir. Çizelge 4.1 her tanımlama seviyesi için temel elemanları göstermektedir.

4.1.2. Tasarım otomasyonu

Tasarım sırasında işlemlerin önemli bir bölümünü bir aşamadan diğerine geçiş oluşturur. Bu, yorucu ve sürekli yinelenen bir işlemdir. Bu çevirme işlemlerinde ve her bir tasarım aşamasının çıktılarının eldesinde bilgisayar desteğinden faydalanılır, buna tasarım otomasyonu denir.

Çizelge 4.1. Farklı tanımlama seviyelerinin elemanları

Tanımlama Seviyeleri	*Davranışsal Tanımlama*	*Yapısal Tanımlama*	*Fiziksel Tanımlama*
Sistem Seviyesi	Algoritmalar Akış grafikleri	İşlemciler Bellekler	Baskı devre kartları Entegre Devreler
Saklayıcı İletişim Seviyesi	Saklayıcı iletişimleri	Saklayıcılar Fonksiyon birimleri Veri toplayıcılar	Entegre Devreler Modüller
Mantık Seviye	Boolean fonksiyonları	Kapılar Flip-floplar	Entegre Devreler Hücreler
Devre Seviyesi	Transfer fonksiyonları	Transistörler Bağlantılar	Hücreler Yol parçaları

Tasarım otomasyonunda kullanılan araçlar kullanıcıya, tasarım başlangıcında donanım üretiminde, kayıtları tutmada, gerçeklemede ve tasarım kontrolünde

kolaylık sağlarlar. Bu araçlar, ilgili tasarım aşamasında, üstlendikleri işlevleri yerine getirirler. Örneğin veri yolu tasarımı aşamasının çıkışları bir simülasyon programı yardımıyla test edilebilir. Daha sonra sentezleyici içeren araçlar yardımıyla flip-flop ve kapıların bağlantıları elde edilebilir. Yani bir sonraki tasarım aşamasına geçilebilir.

Donanım tanımlama dilleri, değişik tasarım aşamalarındaki çıkışların gösterilimi ile ilgili standart sağlarlar. HDL tabanlı bir tasarım aracı ise girişinde bu standart gösterilimi kullanır. Mesela bir sentez aracı uygulanan HDL girişini daha fazla donanım bilgisi içeren diğer bir dile çevirebilir.

4.2. CPLD Tasarım Akışı

CPLD tasarımında takip edilecek olan ana aşamalar aşağıda verilmiştir.

1. Tasarım girişi
2. Derleme
3. Simülasyon
4. Elemanın programlanması

Tasarım girişi iki şekilde yapılabilmektedir; şematik ve metin girişi. Şematik giriş tasarlanacak olan devrenin açık diyagramının şematik editör içerisine mantık elemanları şeklinde çizimi, aralarındaki bağlantıların gösterilmesi ve giriş-çıkış uçlarının şema üzerinde belirtilmesidir. Metin girişi donanım açıklama dilleriyle yapılmaktadır. Önceki bölümde anlatılan VHDL en yaygın olarak kullanılan donanım açıklama dilidir. Bir tasarım hem şematik hem de metin şeklinde girilebilir. Ancak şematik girişler genelde üretici firmalara bağımlıdır. Yani A firmasının CPLD'leri için kullanılan şematik editör B firmasının CPLD'leri için kullanılmayabilir. HDL dilleri ile yazılan bir metin girişi ise tüm CPLD'ler için kullanılabilir.

Derleme şematik yada metin yoluyla hazırlanan devrenin program içerisinde netlist dosyasını oluşturur. Netlist dosyası devre içerisindeki kapıların neler olduğunu, arasındaki bağlantıları ve giriş ve çıkışları belirten metin dosyasıdır. Dosya içerisinde kullanılacak olan CPLD'nin üreticisi ve ailesi belirtilir. Dosya devredeki elemanların sentez edilmesiyle oluşturulur.

Simülasyon aşaması, derlenen tasarımın çeşitli simülasyon programları ile bilgisayar desteği ile denenmesidir. Genel olarak çoğu CPLD üreticileri kendi simülasyon programlarını kullanıcılara sunarlar. Bunların yanında Proteus, Orcad ve Multisim gibi elektronik programlarda da kısmi olarak CPLD simülasyonu yapılabilmektedir(18).

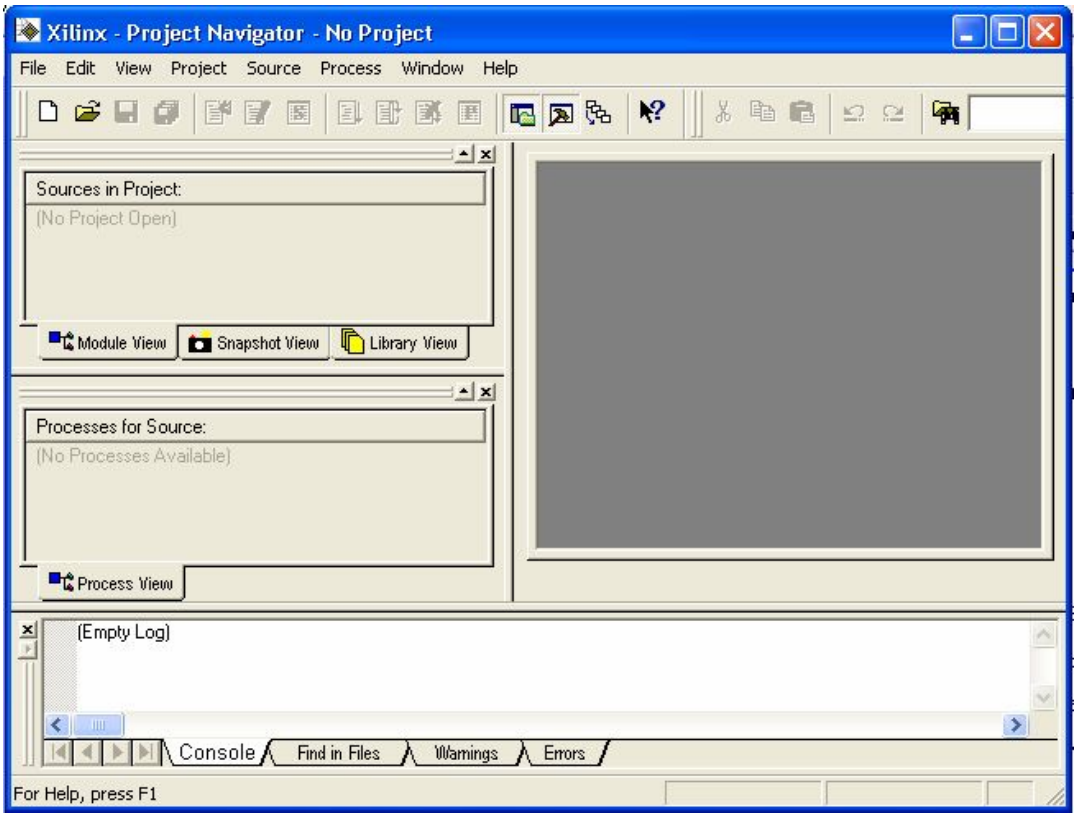
Eleman denendikten sonra, bir programlayıcı vasıtasıyla netlist dosyasının makine koduna çevrilmesiyle elde edilen yükleme dosyası CPLD elemanı içerisine yüklenir ve CPLD istenen tasarım için hazır duruma gelir (19).

4.3. WebPack ISE ve ModelSim XE Programları ile Tasarım

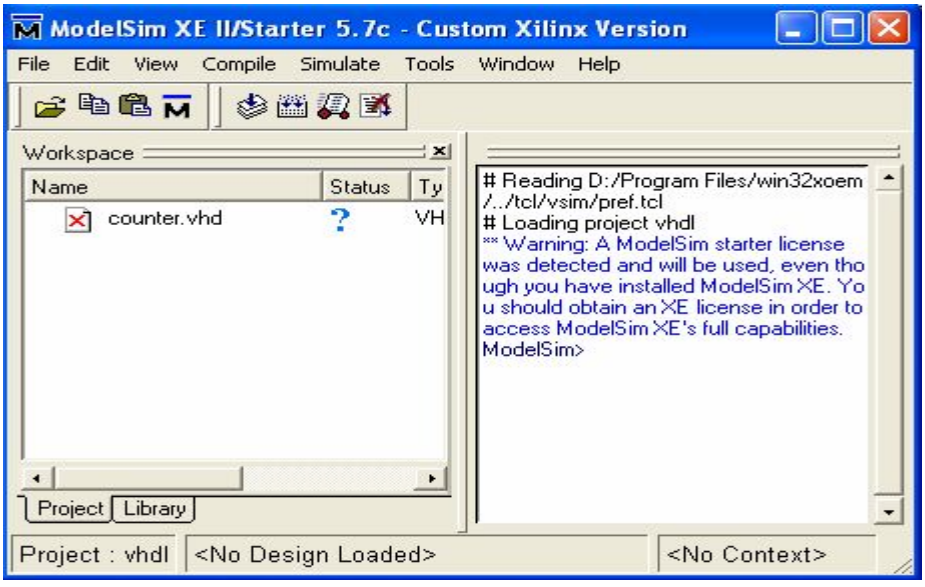
Xilinx firmasının geliştirmiş olduğu WebPack ISE programı PLD'lerle yapılmak istenen bir tasarımın metin ve şema yoluyla hazırlanmasını sağlayan kapsamlı bir programdır. Metin yoluyla (VHDL yada Verilog) girilen tasarımları bir netlist dosyası oluşturmak için sentezler. Şematik şekilde girilen tasarımları ise önce metin formatına dönüştürür ve netlist dosyası oluşturur. ModelSim XE programı ise WebPack ISE programının simülasyon kısmıdır. Kurulumları ayrı şekilde yapılır ve ModelSim XE kurulduktan sonra ISE içerisine entegre olur. ISE içerinde hazırlanan tasarımlar ModelSim programında simüle edilir.

Şekil 4.3. WebPack ISE ve ModelSim XE masaüstü kısayolları

Her iki programda Xilinx firması tarafından kendi üretimleri için geliştirilmiştir. Şekil 4.4'te WebPack ISE programının ve Şekil 4.5'de ModelSim XE programlarının anayüzleri görülmektedir. Programların ayrıntıları tasarım ve simülasyonları gerçekleştirirken adım adım nasıl olduğu açıklanacaktır.



Şekil 4.4. WebPack ISE programı anayüzü

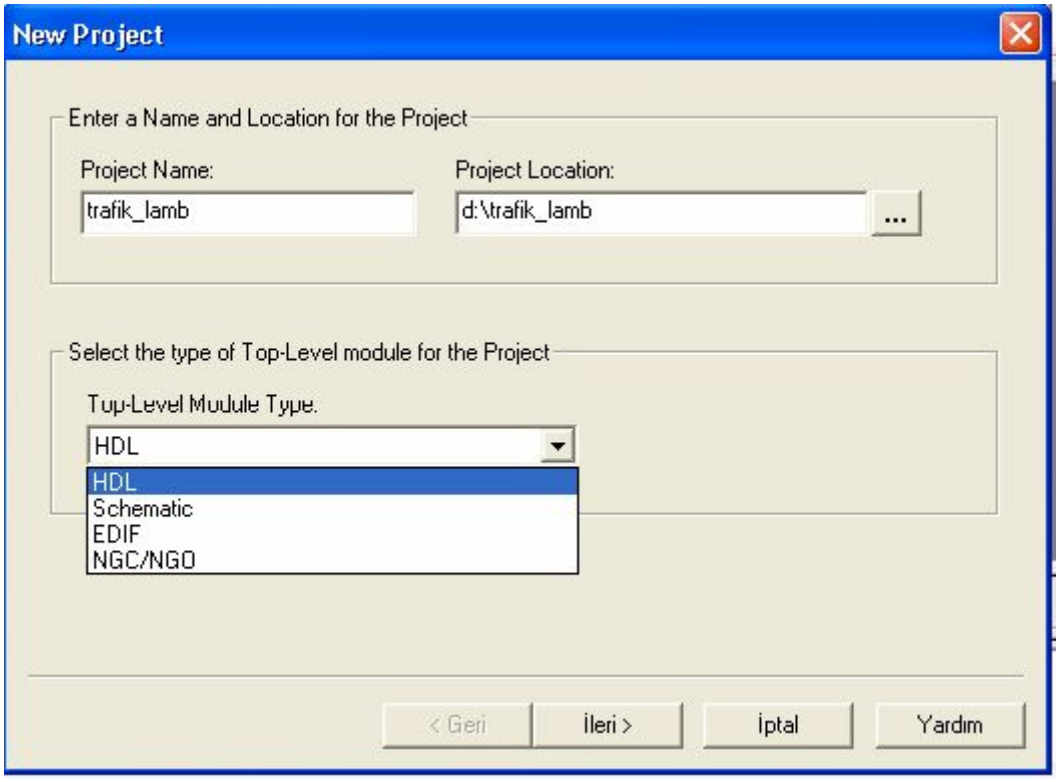


Şekil 4.5. ModelSim XE programı anayüzü

4.3.1. Örnek VHDL tasarım yapılması

Yapacağımız örnek tasarım ile programın kullanılması pekiştirilecek menülerin kullanımı anlatılacaktır. Yapacağımız örnek tasarımda trafik lambası kontrolü uygulaması yapılacaktır. Tasarım WebPack ISE programında gerçekleştirilecektir. Tasarımda öncelikle trafik ışıkları olarak kullanılan ledlerin yanma sürelerini ayarlamak için bir sayıcıya ihtiyacımız vardır ve önce bu sayıcıyı gerçekleştireceğiz.

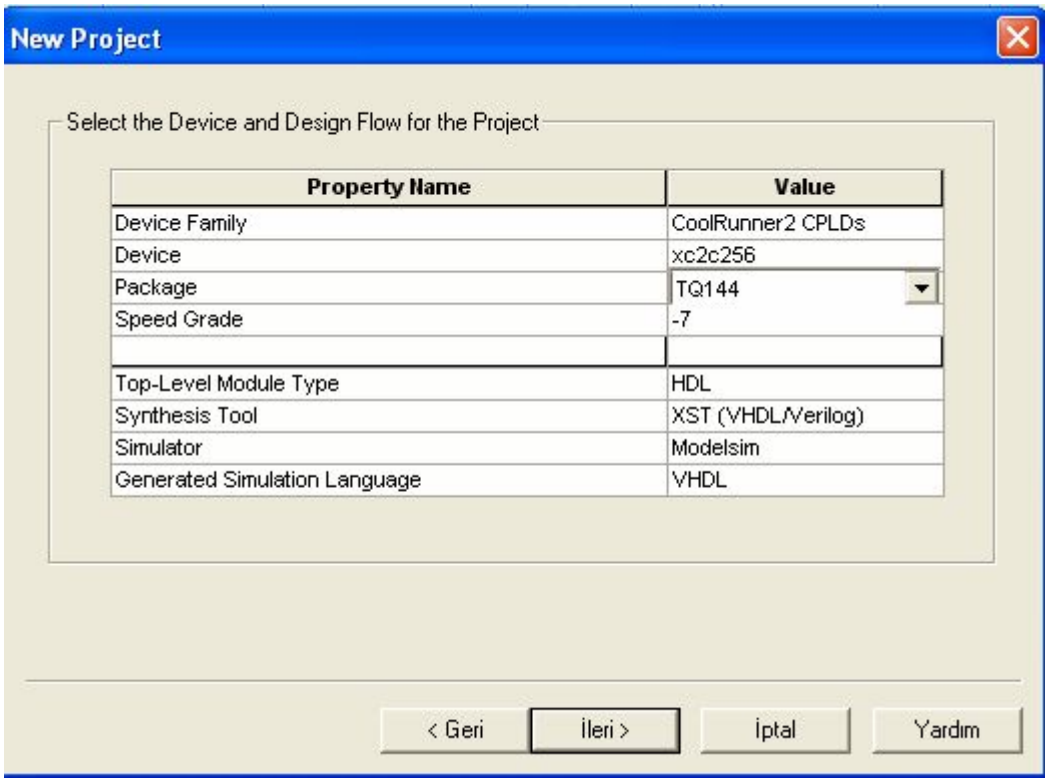
Önce ISE ana yüzünden *File* > *New Project* seçilir, ekrana şekil 4.6'daki pencere gelecektir.



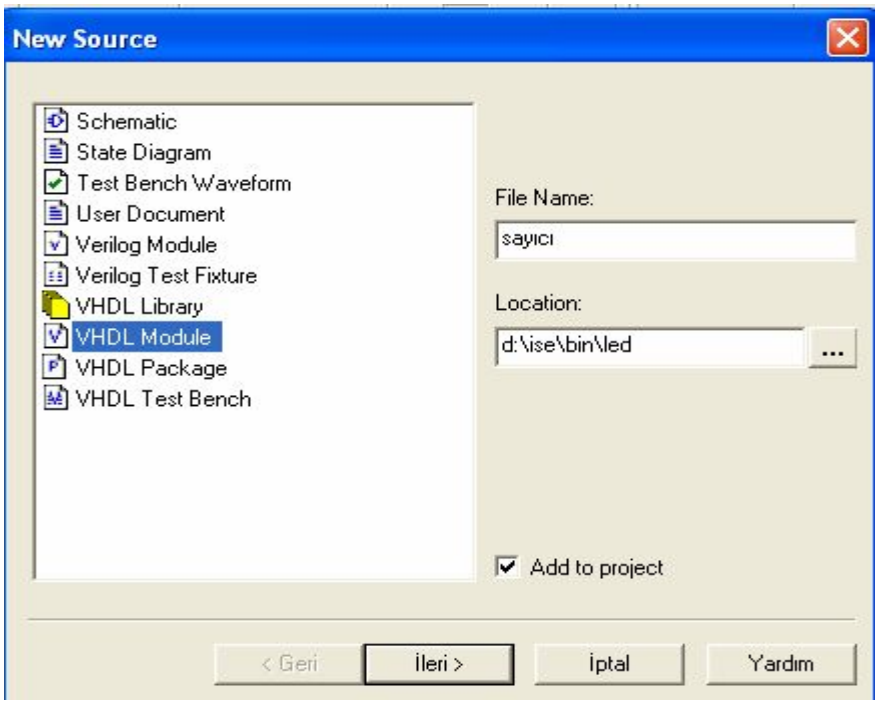
Şekil 4.6. WebPack ISE'de yeni proje açılması

Tasarımın ismi, konumu ve kullanılacak olan modül bu pencere içerisinde belirtilir. VHDL ile tasarım girişi yapılacağı için modül HDL olarak seçilmiştir. Eğer şematik tasarım yapacak olsaydık *Top-Level Module Type* penceresinden *Schematic*

seçilecekti. Bilgiler girildikten sonra *ileri* butonu tıklanır ve ekrana şekil 4.7'deki pencere gelir.



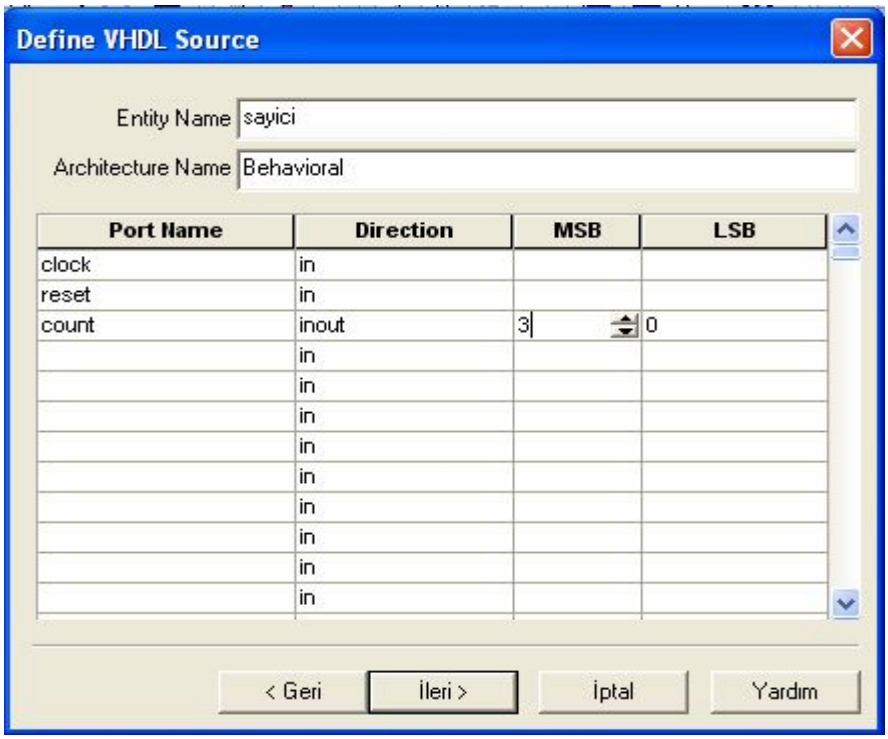
Şekil 4.7. WebPack ISE'de proje için eleman ve ayrıntıların seçimi



Şekil 4.8. WebPack ISE'de kaynak seçimi

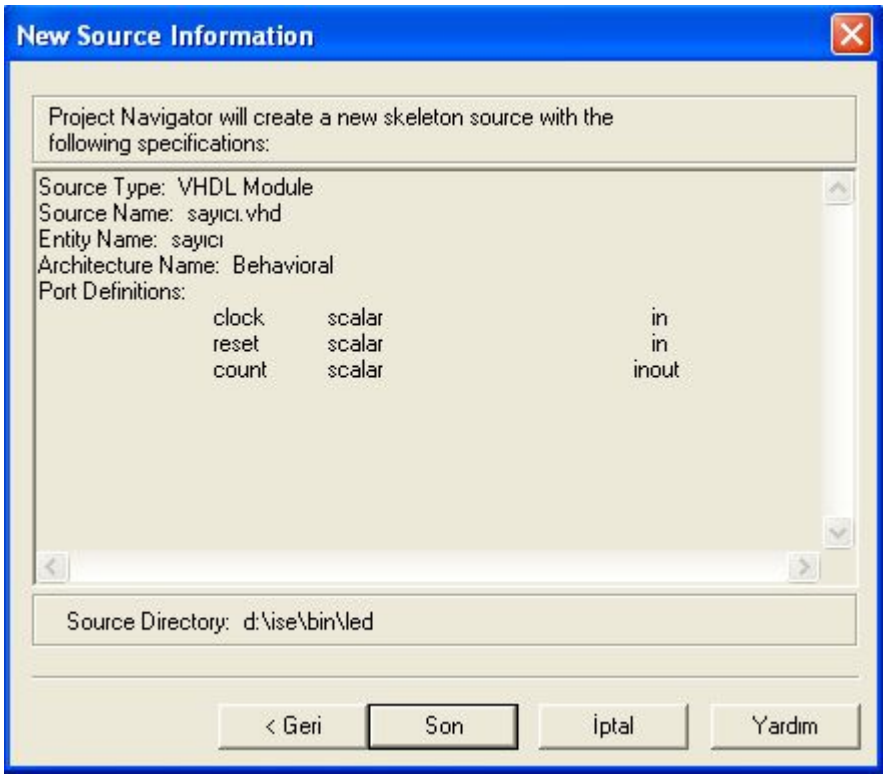
Tasarımda kullanılacak eleman(XC2C256), elemanın kılıfı(TQ144), simülatör programı vb. bu pencereden seçilir. Eleman olarak CoolRunner CPLD'ler seçilmiştir. Simülatör programı ModelSim olarak görülmektedir. İleri butonuna basılarak kaynak ekleme penceresine geçilir (Şekil 4.8).

Bu pencereden bir dosya ismi verilerek bir kaynak seçilir. Tasarımda VHDL kullanılacağı için VHDL modül sayıcı dosya ismi verilerek seçilmiştir. Seçilen dosya ismi bizim VHDL içerisinde kullanacağımız modülün ismidir. Kaynak seçiminden sonra kaynağı tanımlamak için ileri butonuna basılır ve ekrana tanımlama penceresi gelir.(Şekil 4.9)



Şekil 4.9. WebPack ISE'de kaynak tanımlama

Bu pencerede sayıcı olarak tanımladığımız modülün uçları belirlenir. *Clock* ve *reset* uçları giriş olarak ayarlanırken sayma ucu hem giriş hem çıkış olarak ayarlanmıştır. Buradaki 'architecture name' 3. bölümde açıklanan VHDL mimarilerinden 'davranışsal mimaridir'. Count bölümündeki MSB ve LSB hanesindeki 3 ve 0 sayıcının 4 bitlik olduğunu gösterir. Ayarlamalardan sonra kontrol için ileri butonuna basılır ve ekrana Şekil 4.10'daki pencere gelir.



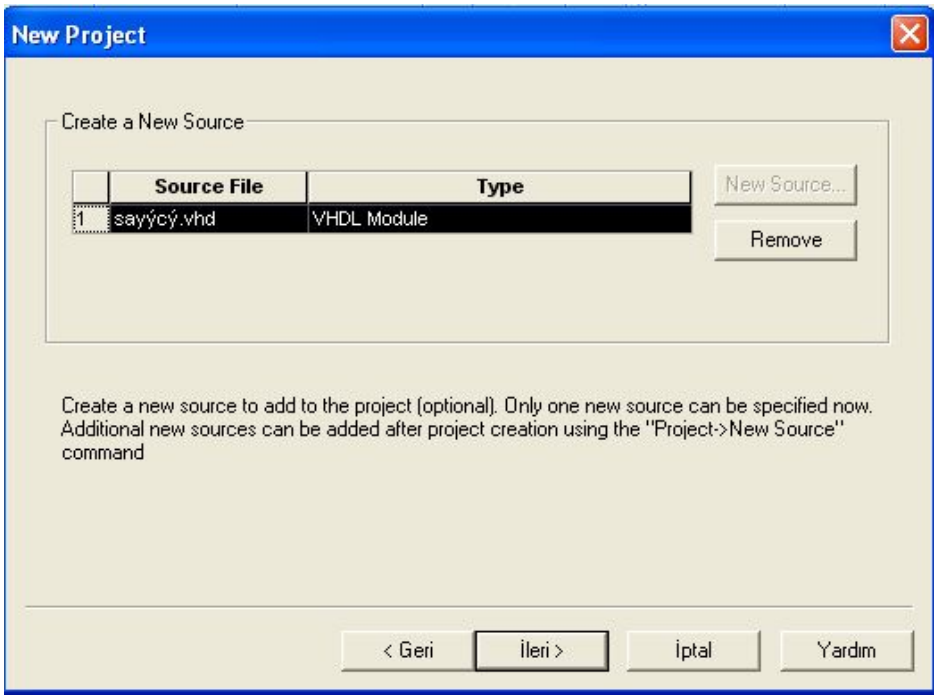
Şekil 4.10. WebPack ISE'de giriş kontrolü

Bu pencere içerisinde şu ana kadar yapılan ayarlamalar ve seçimler bir liste halinde kullanıcının kontrol edebilmesi için sunulmuştur. Son butonuna basıldığında VHDL modülü içerisinde bir sayıcı modülü meydana getirilmiş olur. Ekrana şekil 4.11'deki pencere gelir. Görüldüğü gibi 4 bitlik bir *sayıcı.vhd* kaynak dosyası proje içerisine eklenmiştir.

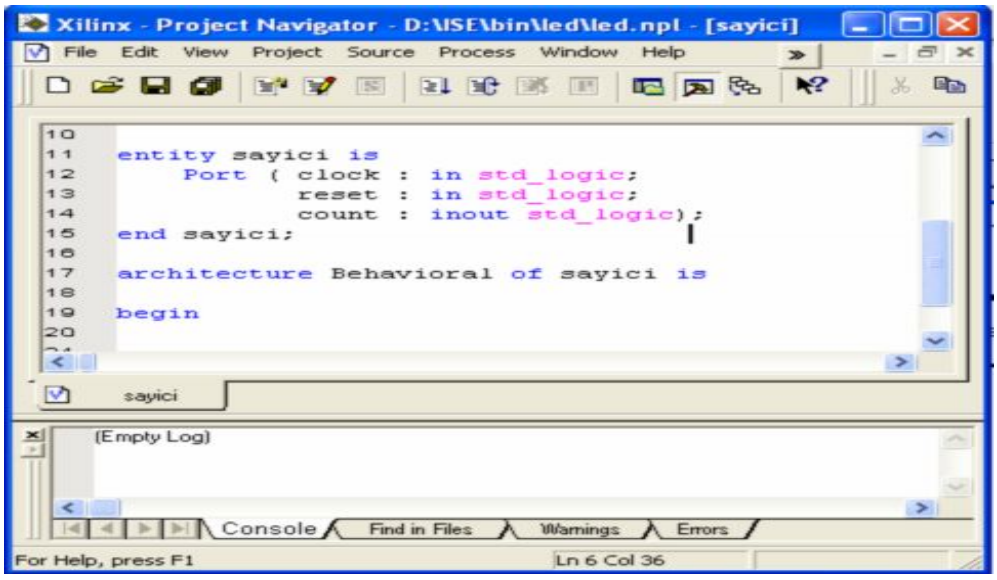
İleri butonu tıklandığında yeni bir kaynak ekleme penceresi gelir. Eğer yeni bir kaynak eklenmeyecekse ileri butonuna basılarak bu aşama geçilir ve ekrana *sayıcı.vhd* modülü için WebPack ISE metin editörü gelir.(Şekil 4.12)

Şimdi elimizde boş bir VHDL modülü bulunmakta ve henüz modülün ne iş yapacağı açıklanmamıştır. Bunun için program içerisinde *Language Template* adı verilen çok kullanışlı bir bölüm bulunmaktadır. Bu bölüm içerisinde sayıcı, multiplexer, dekoder gibi uygulamaların hazır kodları bulunmaktadır. *Edit* menüsünden *Language Templates* seçilirse ekrana Şekil 4.13'teki pencere gelecektir. Burada templates

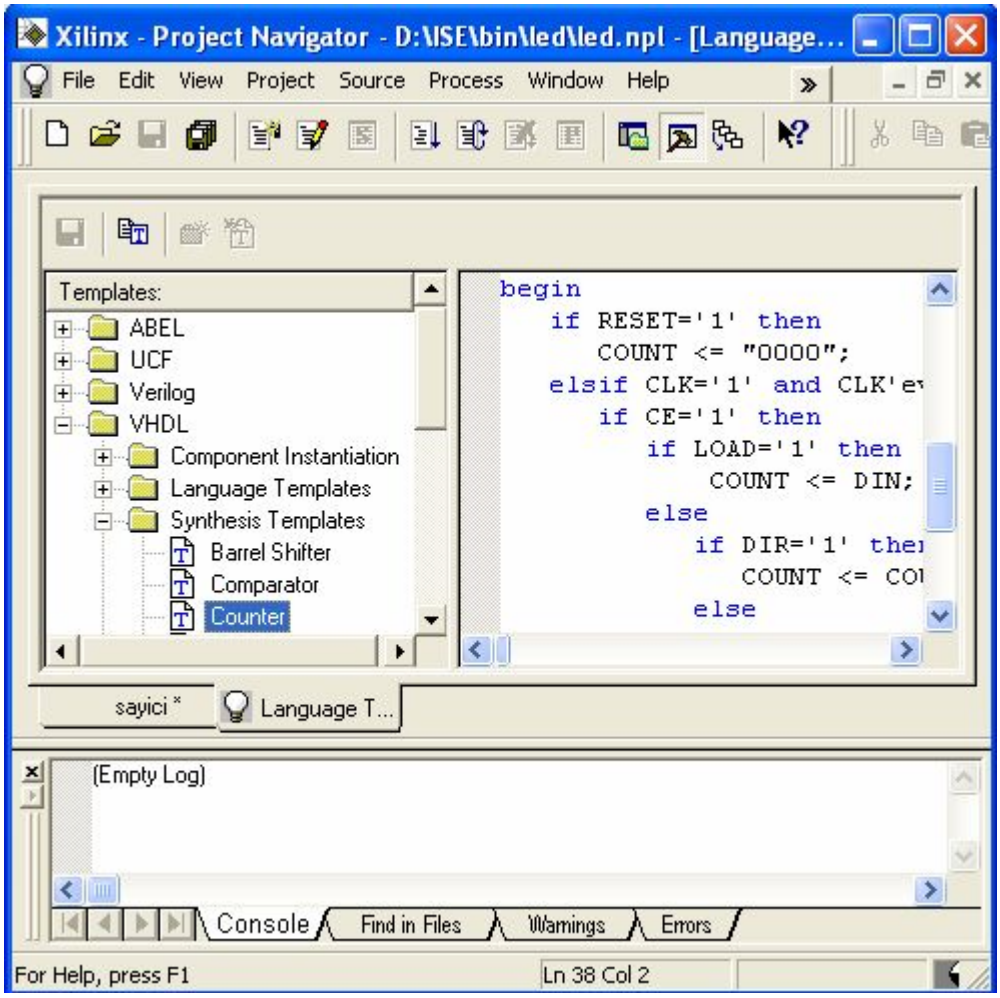
bölümünden *VHDL > Synthesis Templates* seçilir. Bir çok hazır VHDL modül görülecektir. Bu hazır kodlardan *counter* seçildiğinde pencerenin sağ bölümünde sayıcı VHDL kodları gösterilecektir. Bu bölüm kopyalanarak *sayıcı.vhd* modülü içerisinde *architecture* bölümünde *begin-end* arasına yapıştırılır.



Şekil 4.11. WebPack ISE'de oluşturulan modül

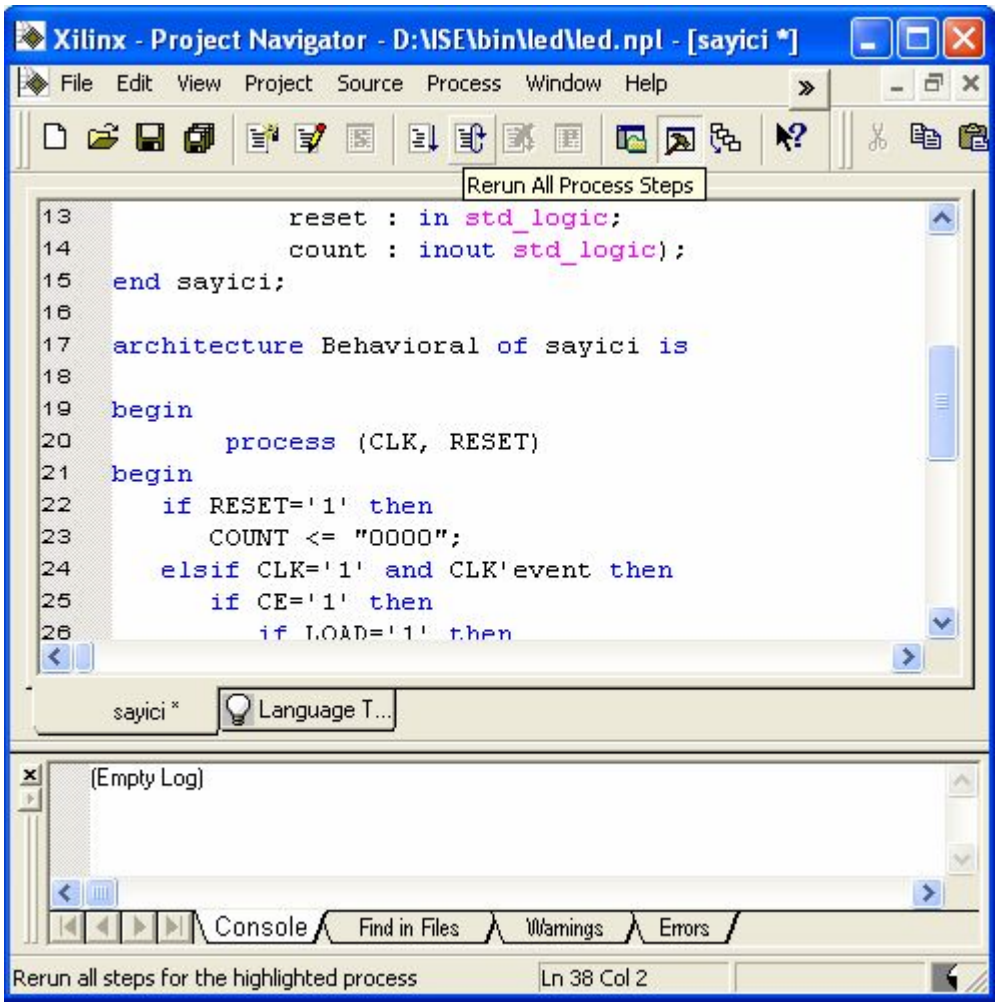


Şekil 4.12. WebPack ISE'de metin editörü

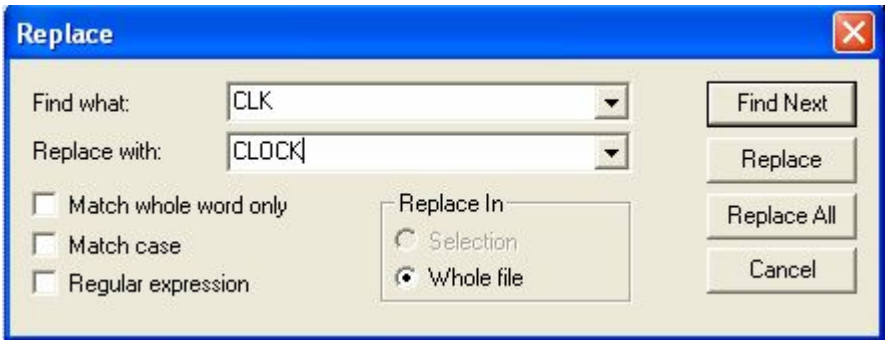


Şekil 4.13. WebPack ISE'de Language Templates menüsü

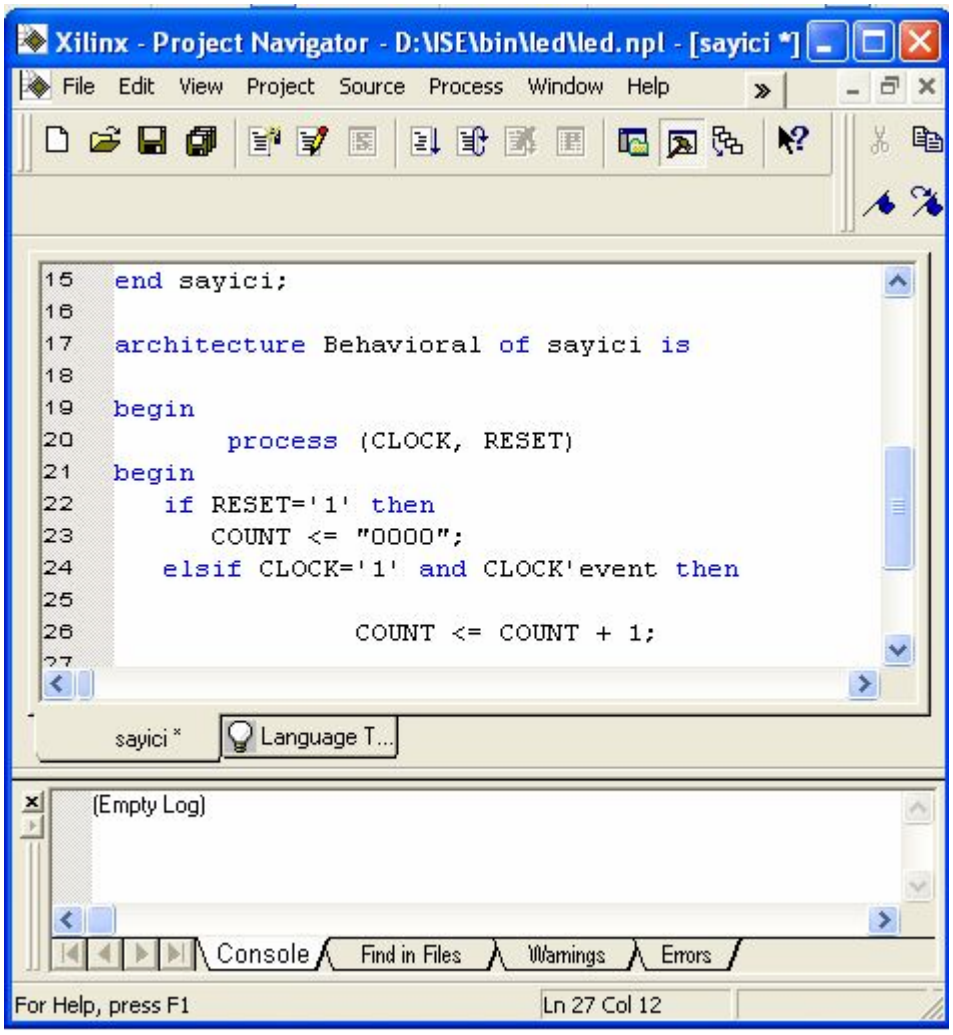
Şekil 4.14 sayıcı kodları eklenmiş *sayıcı.vhd* dosyasını göstermektedir. Yalnız burada dikkat edilmesi gereken bir nokta *clock* gösterimlerinin *clk* şeklinde yapıldığıdır. Tanımlamalarda ise doğrudan *clock* olarak isimlendirilmiştir. Bu nedenle *clk* ifadelerinin *clock* ile değiştirilmesi gerekir. Bunun için *edit* menüsünden '*replace*' seçilir. *Replace* penceresi Şekil 4.15'de görülmektedir. Burada değiştirilmek istenen kelime ile yerine yazılacak kelimeler girilir ve *replace* butonuna basılır. Metin editörü Şekil 4.16'daki gibi olur.



Şekil 4.14. WebPack ISE'de sayıcı kodları



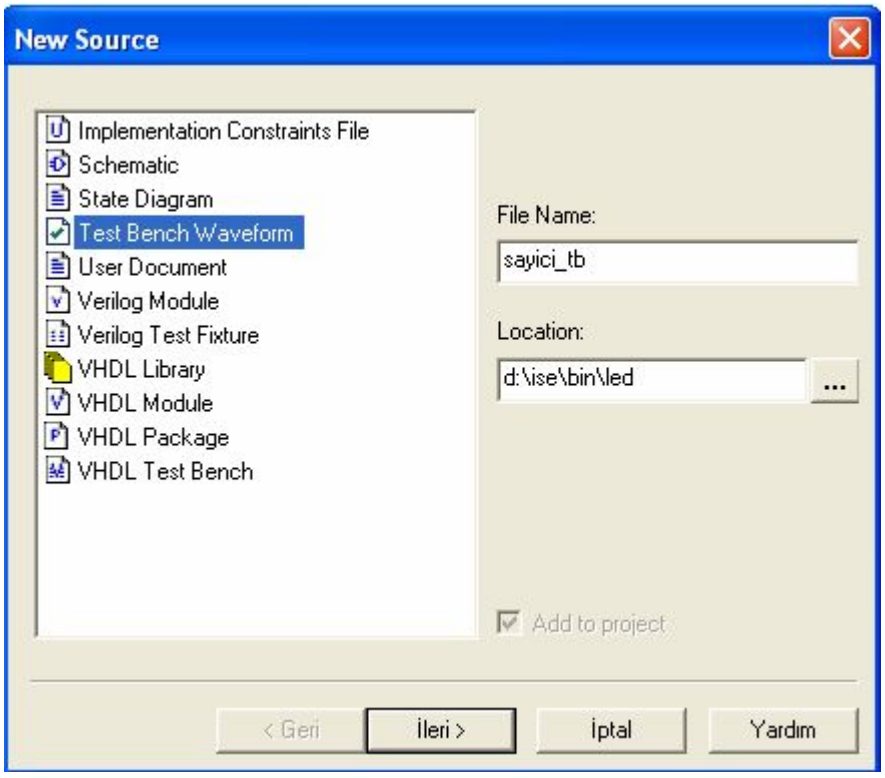
Şekil 4.15. WebPack ISE'de replace penceresi



Şekil 4.16. WebPack ISE'de replace sonrası sayıcı.vhd kodları

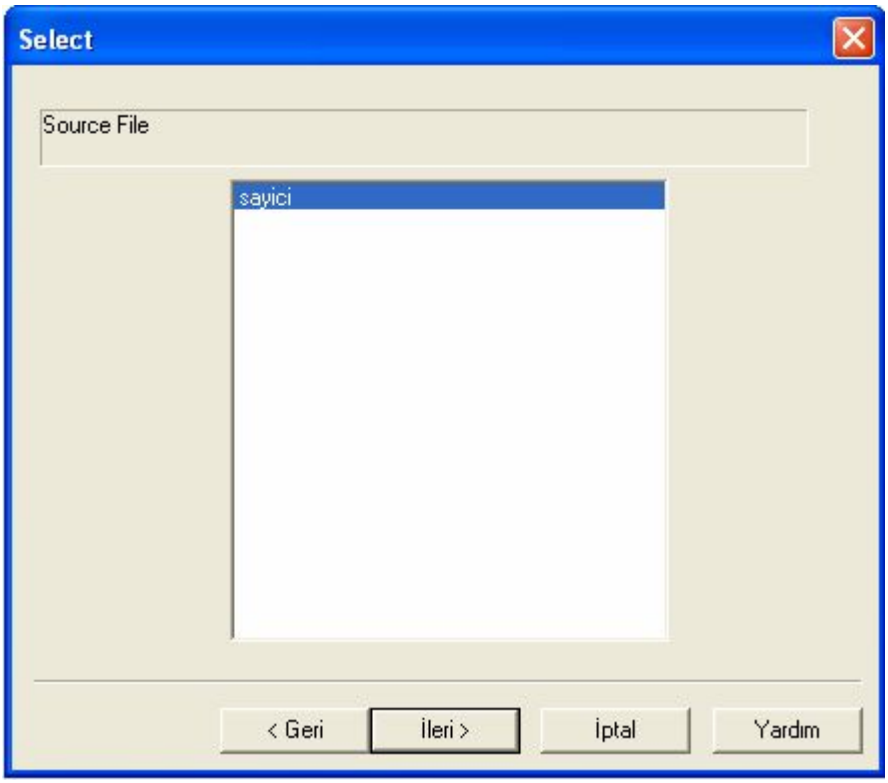
Tasarımımız şimdi iki giriş, clock, reset ve bir çıkış içeren bir devredir. Devrenin nasıl çalışacağı VHDL kodları içerisinde *architecture* bölümünde kodlanmıştır. Tanımladığımız *count* sinyali *clock* sinyali "1" olduğunda ve üzerinde bir değişme olduğunda yani pozitif kenarda artacaktır.

Şu anda sayıcı modülü simülasyon için hazırdır. Fakat simülasyon için dijital analizör dosyası(testbench waveform) oluşturulmalıdır. Bunun için *Projec*t menüsünden 'new source' seçilir ve ekrana Şekil 4.17'deki pencere gelir.

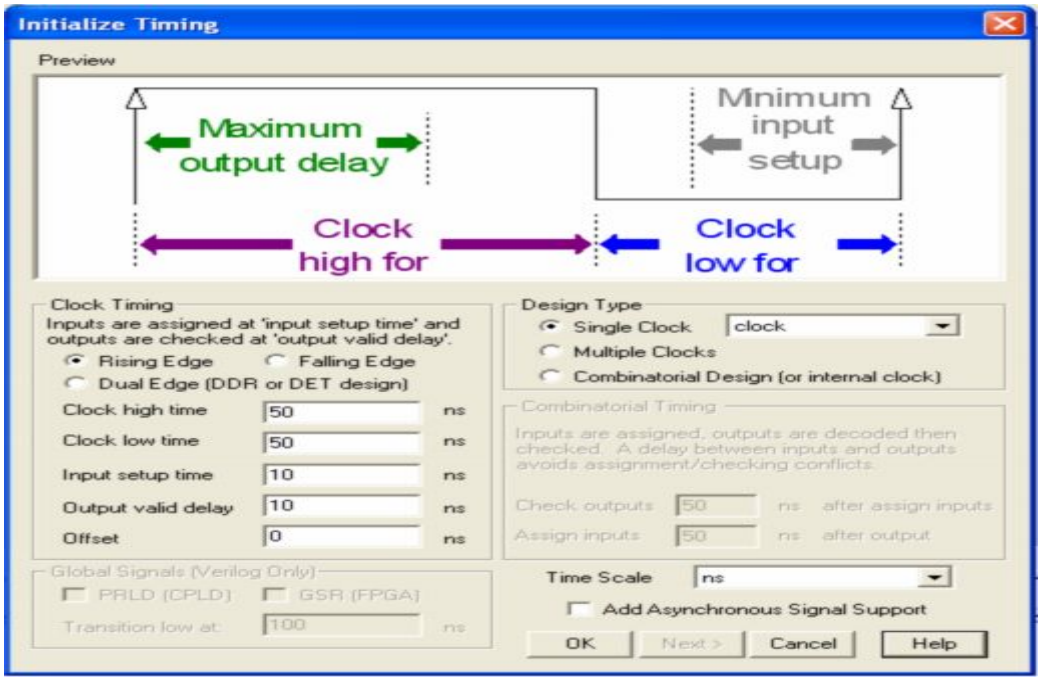


Şekil 4.17. WebPack ISE'de test sinyalinin oluşturulması

Burada test bench waveform seçilir ve bir isim verilir (sayici_tb). Analizörde simüle edilecek bir modül seçmek için ileri tuşuna basılır ve ekrana Şekil 4.18'deki pencere gelir. Burada yaratılan modül (sayıcı) seçilir ve *ileri* butonuna basılır. Modül hakkında bilgiler ekrana gelir ve son butonuna basılırsa analizörün clock frekansı ve çıkış gecikmeleri gibi özelliklerini ayarlamak için ekrana Şekil 4.19'daki pencere gelir. Burada gerekli ayarlamalar yapılır ve OK butonuna basılır.

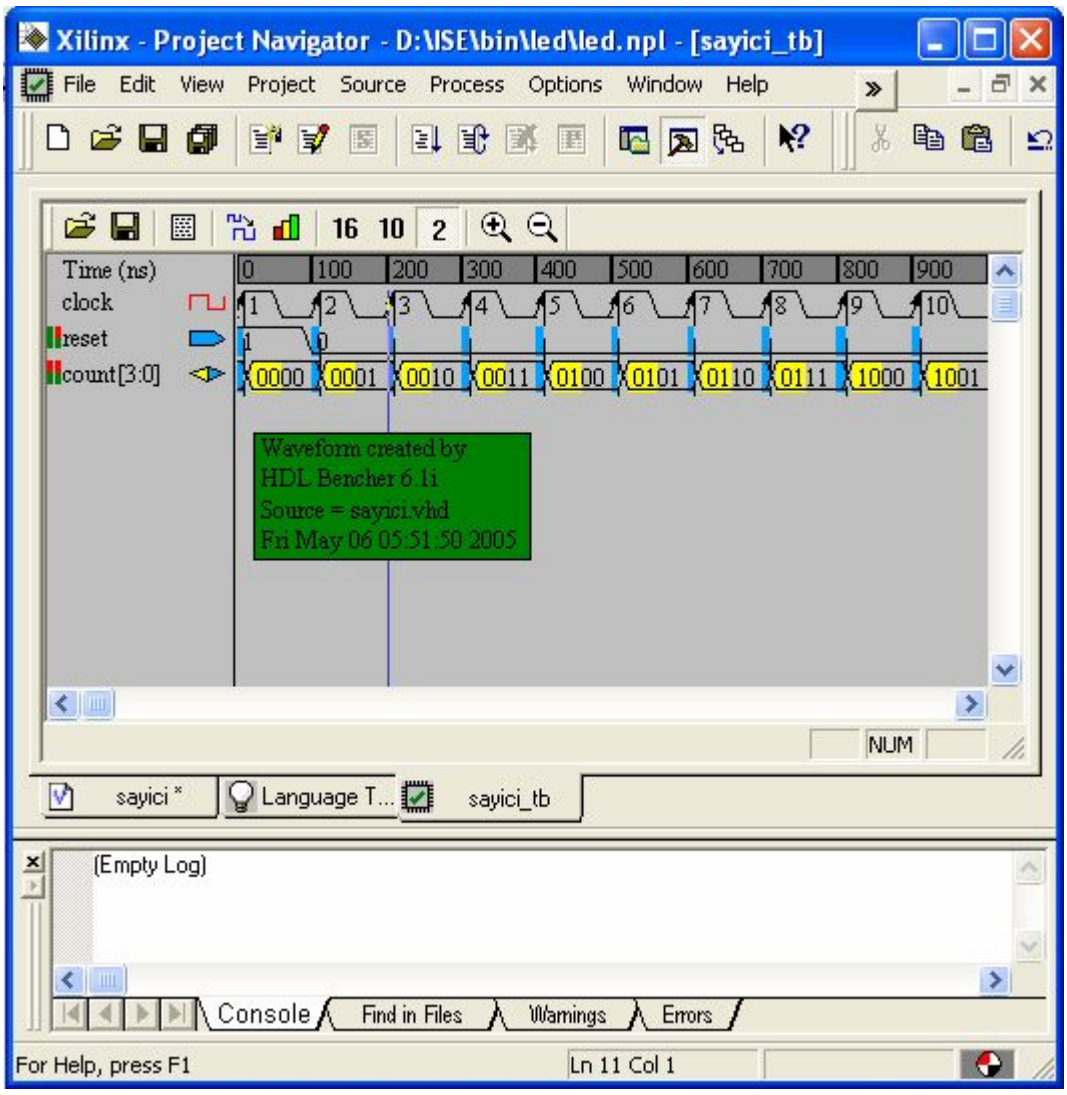


Şekil 4.18. WebPack ISE'de analizör için modül seçimi



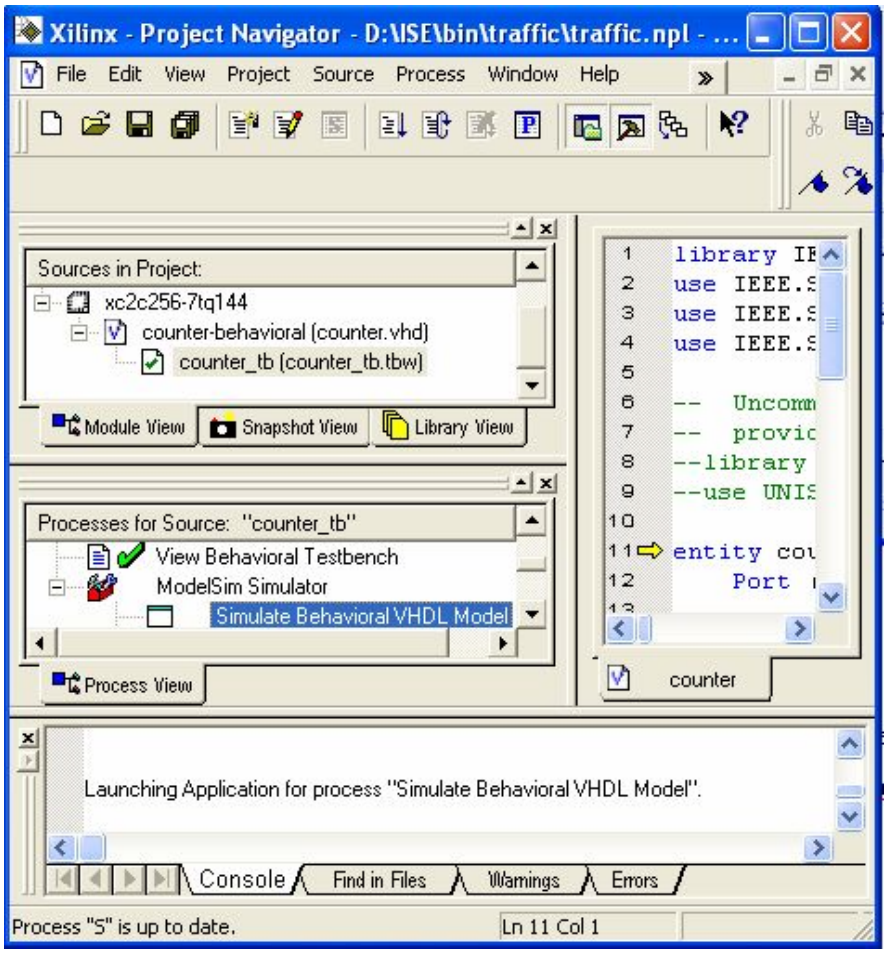
Şekil 4.19. WebPack ISE'de analizör için clock sinyalinin ayarlanması

Ekrana sayma işlemini gösteren dijital analizör gelir (Şekil 4.20). Count [3:0] sinyali incelenirse sayıcının 0-15 arası saydığı görülebilir. Burada sayma sınırları ilk clock sinyali altındaki count sinyaline çift tıklanarak ayarlanabilir. File/save menüsünden testbench kaydedilirek kapatılır.



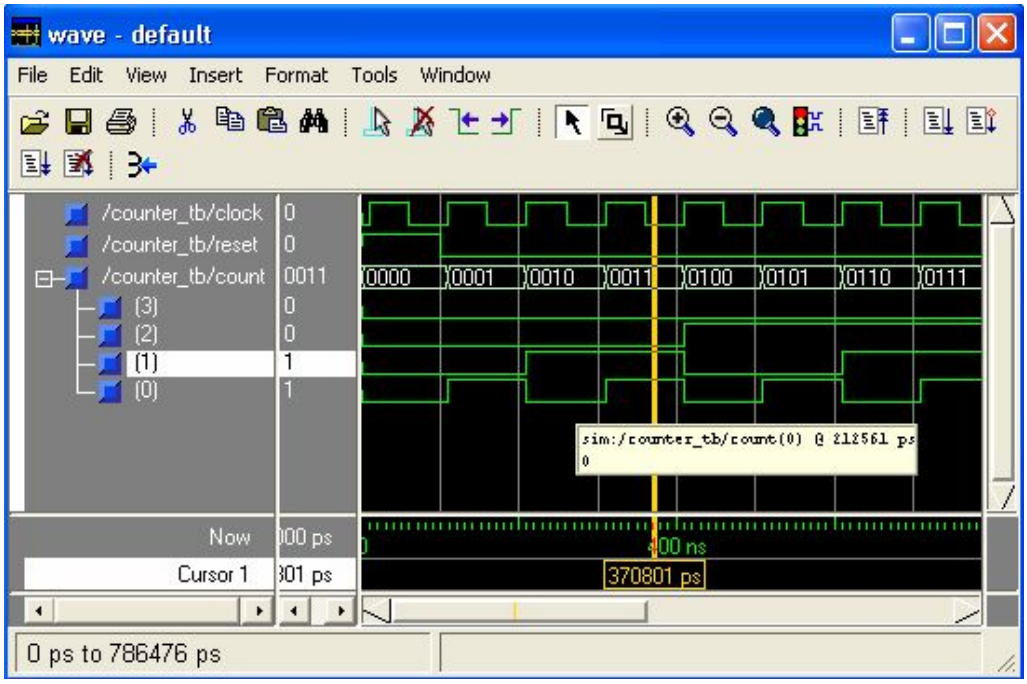
Şekil 4.20. WebPack ISE'de analizörde sayıcı sinyalleri

Sayıcı için testbench oluşturulduğuna göre ModelSim'de sayıcının simülasyonu yapılabilir. Bunun için view menüsünden 'sources' ve 'processes' seçilir. Source penceresinde counter _tb.tbw seçilir ve processes penceresinden 'simulate behavioral VHDL model' çift tıklanır.(Şekil 4.21)



Şekil 4.21. WebPack ISE'de ModelSim'in açılması

Çift tıklamadan sonra ekrana MXE simülatörünün dalga analizörü gelir.(Şekil 4.22)

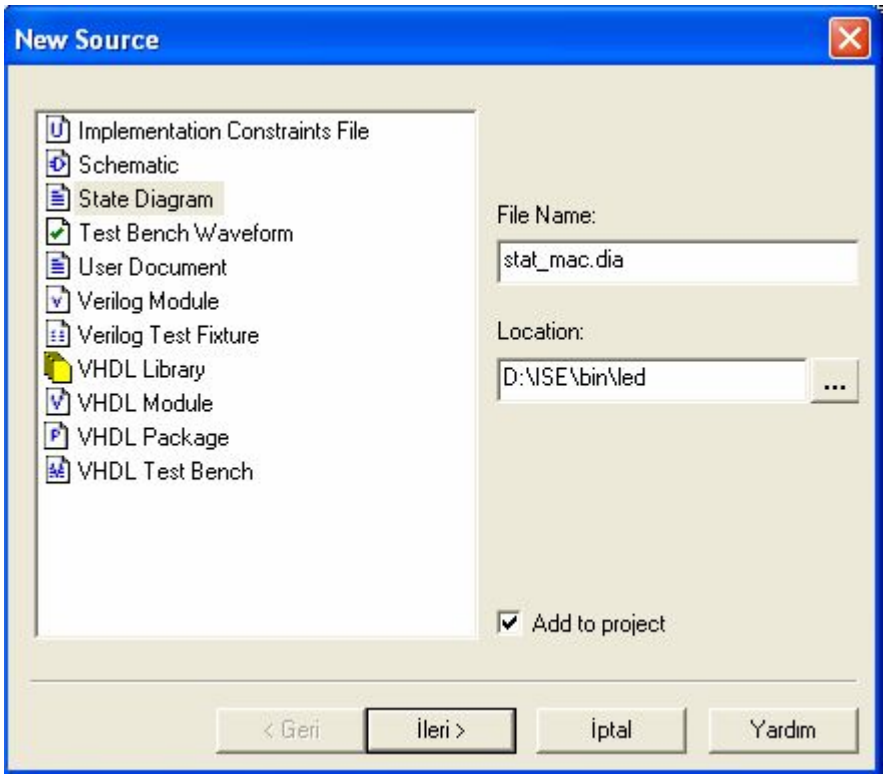


Şekil 4.22. MXE Simülasyon görünümü

Led devresi için sayıcı bölümü daha öncede belirtildiği gibi zamanlayıcı olarak çalışacaktır ve ledlerin geçişlerini ayarlayacaktır. Devrede 3 adet led kullanacağız; kırmızı, yeşil ve sarı. Her bir durumda hangi ledlerin aktif olacağı aşağıda verilmiştir.

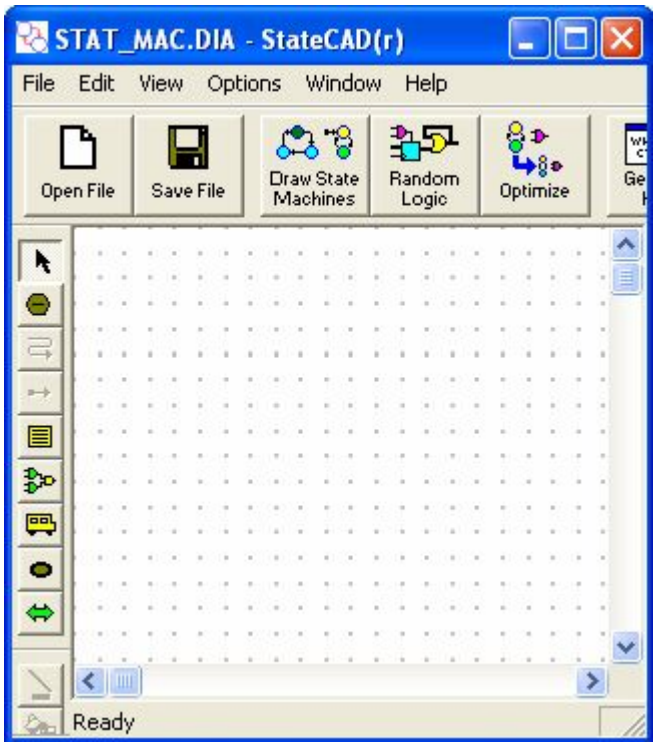
1. Durumda kırmızı led,
2. Durumda kırmızı ve sarı ledler,
3. Durumda yeşil led,
4. Durumda sarı led yandıktan sonra tekrar birinci duruma dönülecektir.

Bu dört ledin durumlarını oluşturmak için WebPack program içerisinde ‘state diagram (durum diyagramı)’ adı verilen diyagramı kullanacağız. Bu diyagramı açmak için Project menüsünden ‘new source’ seçilir ve ‘state diagram’ Şekil 4.23’deki gibi işaretlenir.

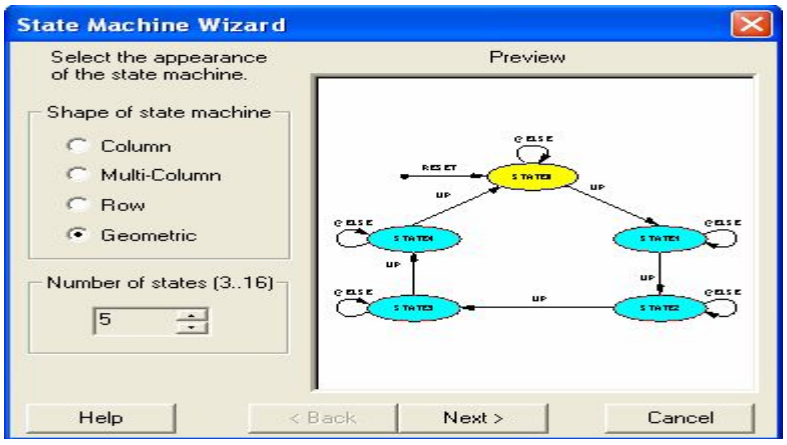


Şekil 4.23. WebPack ISE'de durum diyagramı açılması

İleri butonuyla kontrol penceresine gelinir ve son butonuna basıldığında durum diagram penceresi ekranda görülür (Şekil 4.24). Buradan 'draw state machine' buttonu tıklanır. Ekrana gelen pencereden (Şekil 4.25) 'number of states (durum sayısı)' seçeneği devremizde 4 durum olacağı için 4 yapılır ve ileri butonuna basılır.

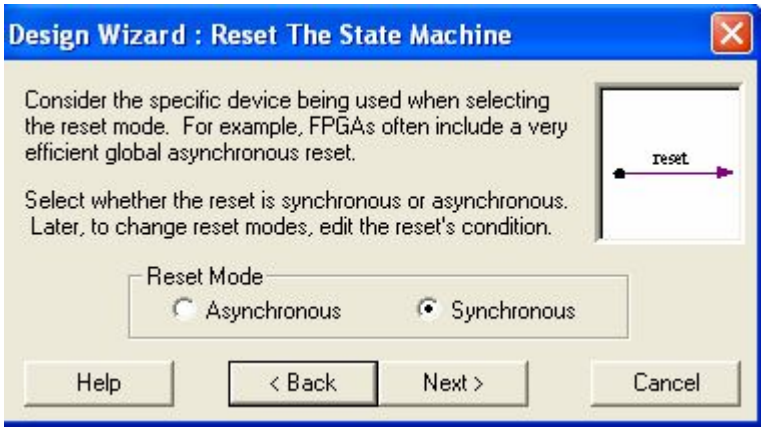


Şekil 4.24. WebPack ISE'de durum diyagramı



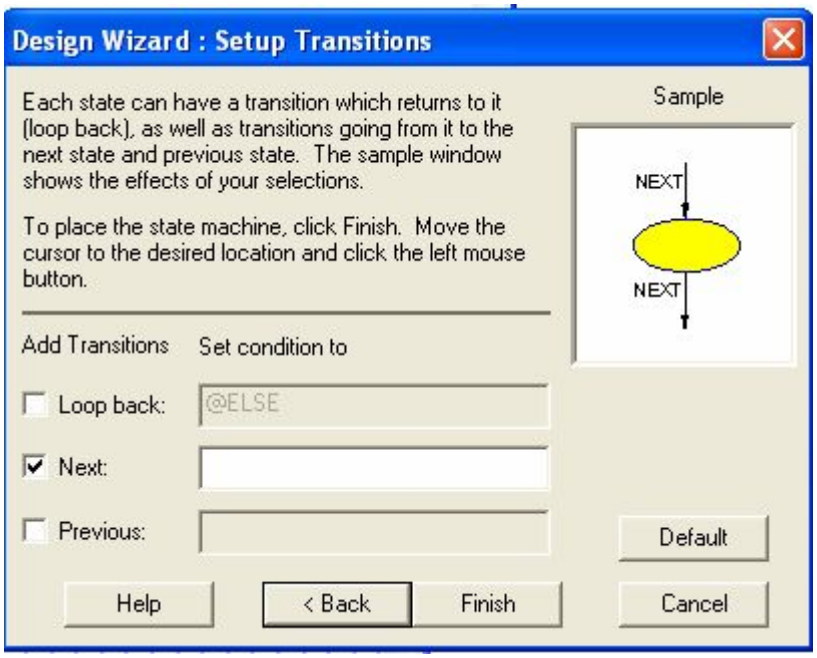
Şekil 4.25. WebPack ISE'de durum diyagram ayarları

Ekrana Şekil 4.26'daki pencere gelir. Buradan senkron bir durum diyagramı oluşturmak için 'syncronhous' işaretlenir ve ileri butonuna basılır.



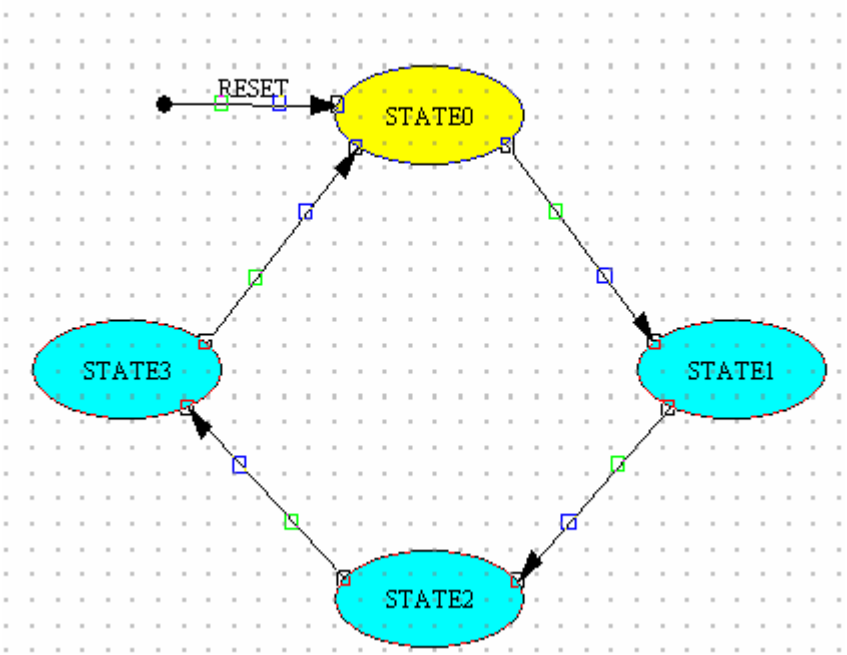
Şekil 4.26. WebPack ISE'de mod seçimi

Ekrana Şekil 4.27'deki pencere gelecektir. Burada 'next' alanına timer yazılır ve son butonuna basılır.

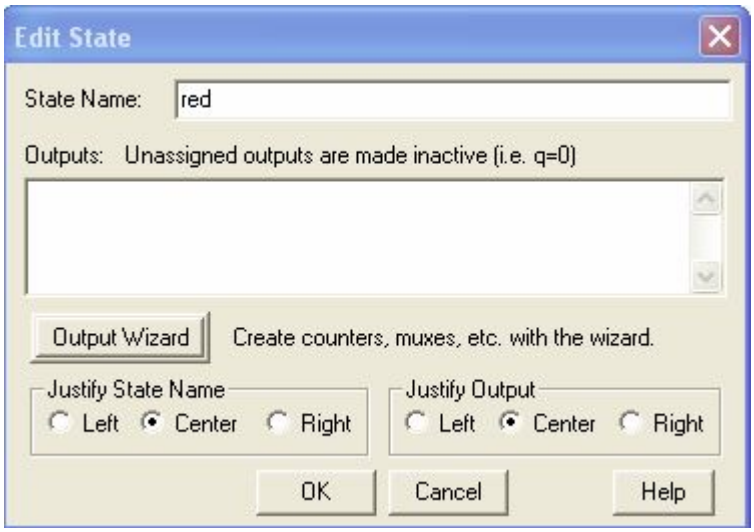


Şekil 4.27. WebPack ISE'de durum diyagram geçiş kurulumu

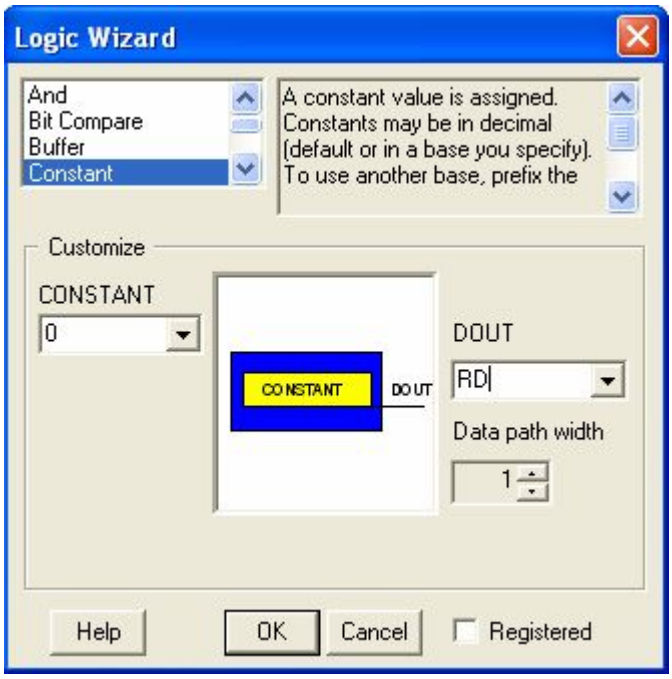
Ekrana dört durumlu diyagram gelir (Şekil 4.28). Burada 'state 0' baloncuğu çift tıklanarak ismi 'red' yapılır (Şekil 4.29) ve 'output wizard' butonuna basılır. Ekrana Şekil 4.30'daki pencere gelir.



Şekil 4.28. WebPack ISE'de dört durumlu diyagram

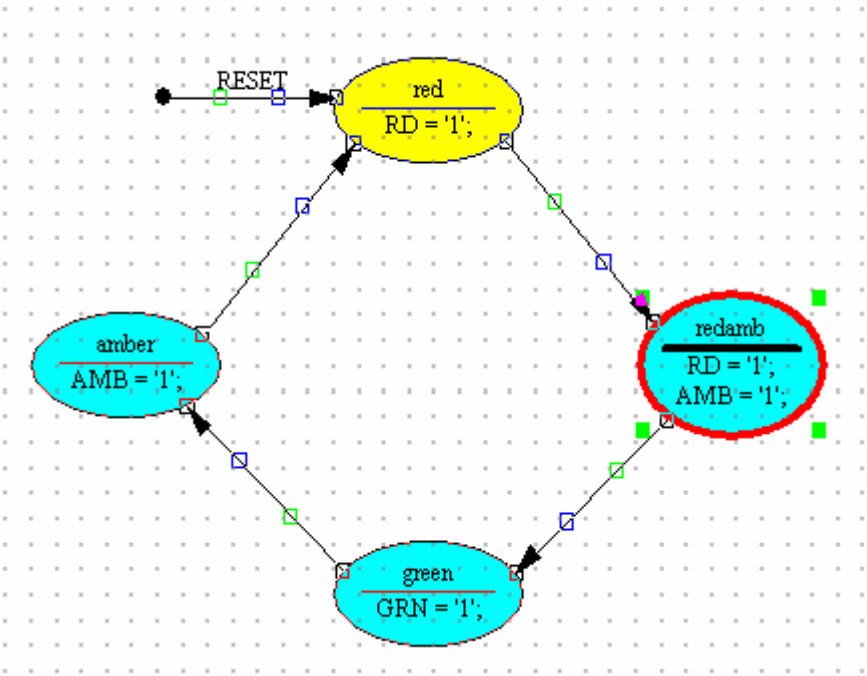


Şekil 4.29. WebPack ISE'de durum adı değiştirme



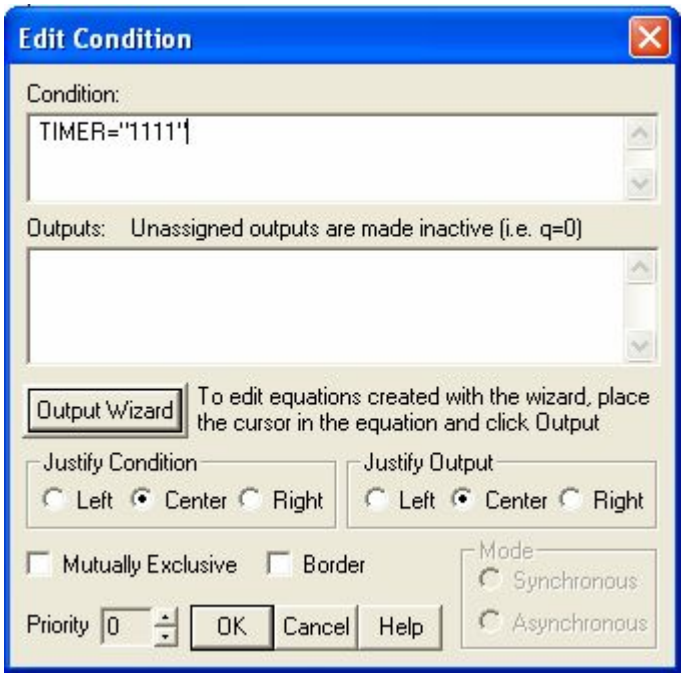
Şekil 4.30. WebPack ISE'de durum adı değiştirme

Devrede üç çıkış olacaktır; RD, AMB, GRN. DOUT bölümü RD yapılır. Ok butonuyla geri dönülür ve diğer tüm diyagramlar için aynı işlemler tekrarlanır. Diyagramın son hali Şekil 4.31'deki gibidir.



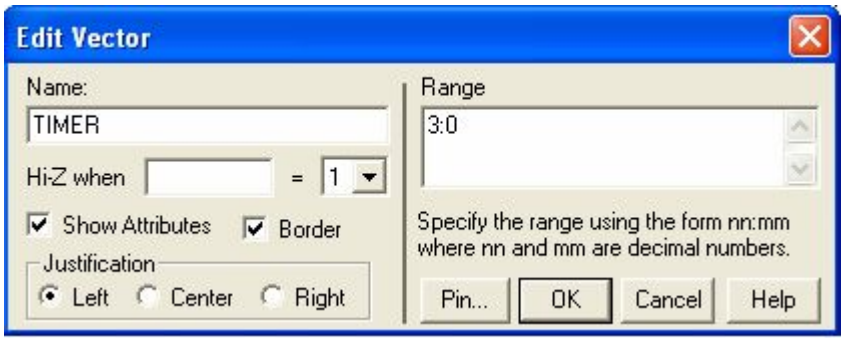
Şekil 4.31. Durum diyagramlarının son hali

‘Red’ ve ‘redamb’ kutuları arasındaki iletim hattı çift tıklanır ve ekrana gelen şart penceresinde (Şekil 4.32) *condition* bölümüne *timer*= “1111” yazılır.



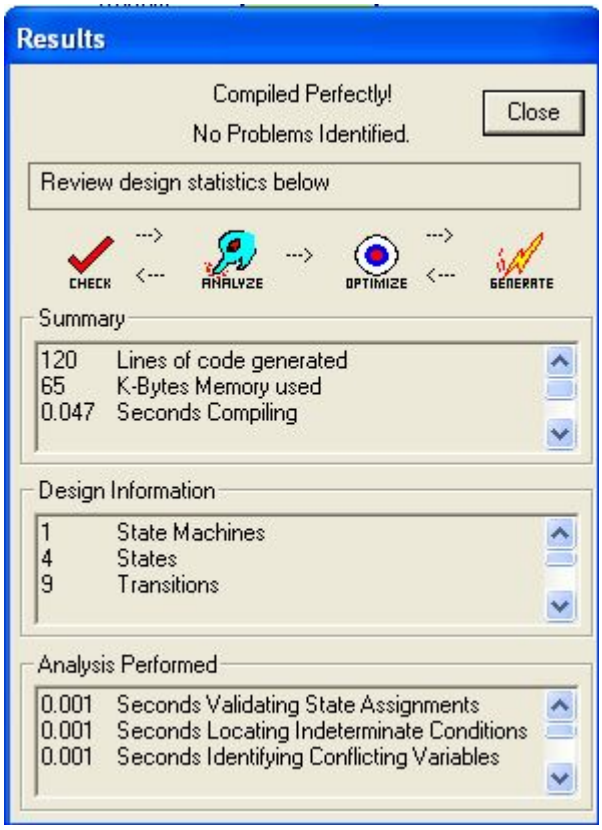
Şekil 4.32. Durum diyagramı şart penceresi

Benzer şekilde ‘redamb’ ‘green’ kutuları arasında ‘timer= “0100”’, ‘green’ ‘amber’ kutuları arasında ‘timer= “0011”’ ve ‘amber’ ‘red’ kutuları arasında ‘timer= “0000” yazılır. Böylece sayıcının her üç saykılında durumlar birer kez sağlanmış olur. Daha sonra zamanlayıcı vektörü seçilir (Şekil 4.33), ismi *timer* yapılır ve aralığı 3:0 olarak seçilirek ve OK butonuna basılır.



Şekil 4.33. Durum diyagramı zamanlayıcı vektörü

Böylece durum diyagramı tamamlanmış olmaktadır. ‘Generate HDL’ butonuyla HDL kodu için derleme yapılır ve ekrana gelen diyalog kutusunda (Şekil 4.34) ‘Compiled Perfectly’ mesajı görülür. Bu mesaj tasarımımızda hata olmadığını göstermektedir.

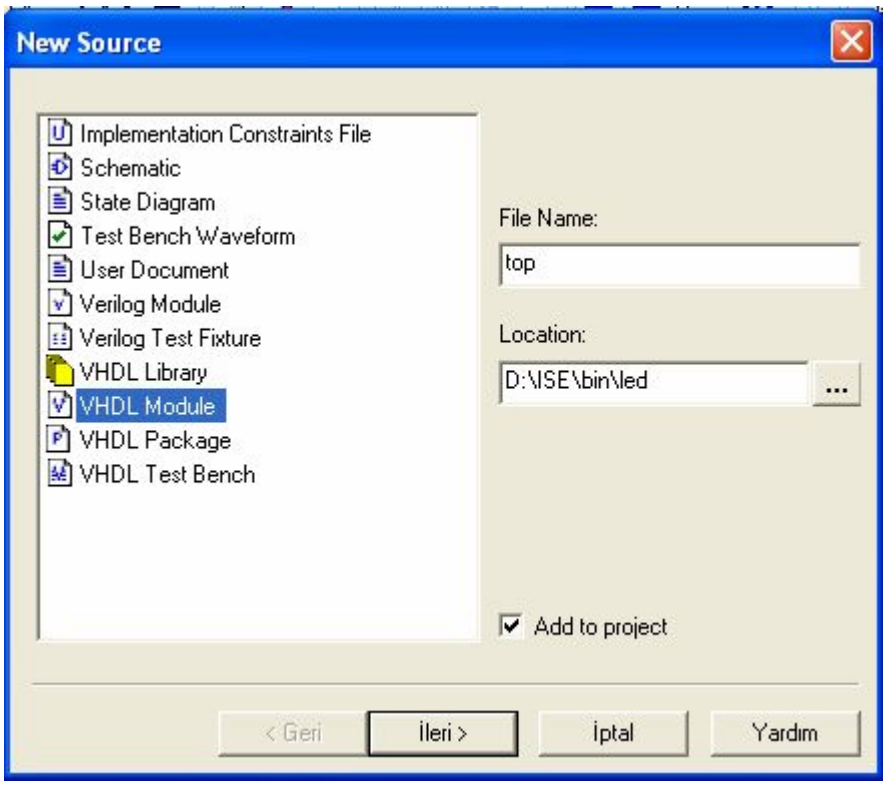


Şekil 4.34. Durum diyagramının HDL koduna dönüşümü

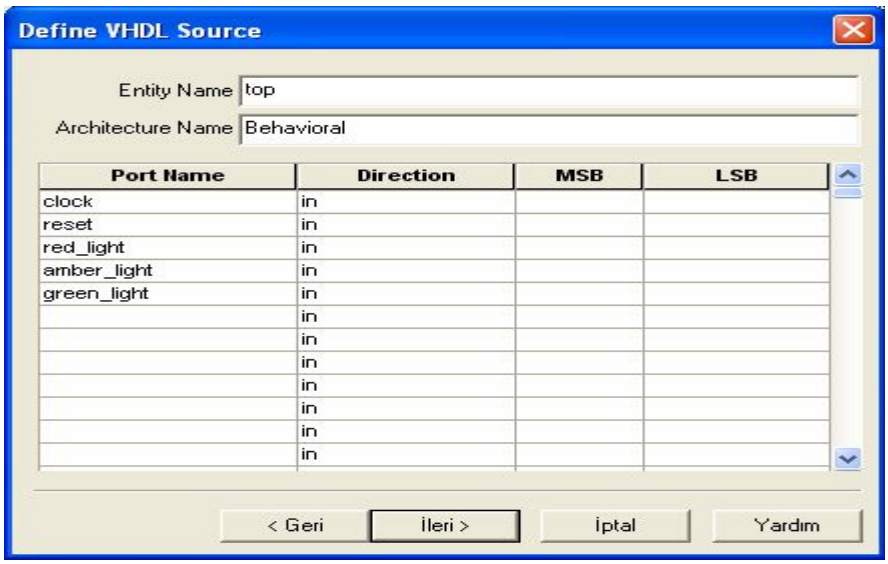
Durum diyagramı kaydedilir ve kapatılır. Diyagram şu anda WebPack ISE’ye eklenebilir durumdadır. ISE penceresinde Project menüsünden ‘add source’ seçilir. Buradan ‘*stat_mac.vhd’* açılır ve vhd design file olarak tanımlanır. Böylece diyagram kaynakların arasına eklenmiş olur.

Şimdi elimizde iki adet modül var, biri sayıcı modülü ve diğeri trafik lambası tasarımımızın durumlarını ayarlayan durum diyagramı. Bu iki modül bir ‘*top_level*’ dosya ile birleştirilecektir. Birleştirme hem şematik yolla hemde metin girişi şeklinde yapılabilir.

Project menüsünden *new source* seçilir ve top adında bir VHDL modülü oluşturularak (Şekil 4.35) ileri butonuna basılır ve modül tanımlaması yapılır. (Şekil 4.36)

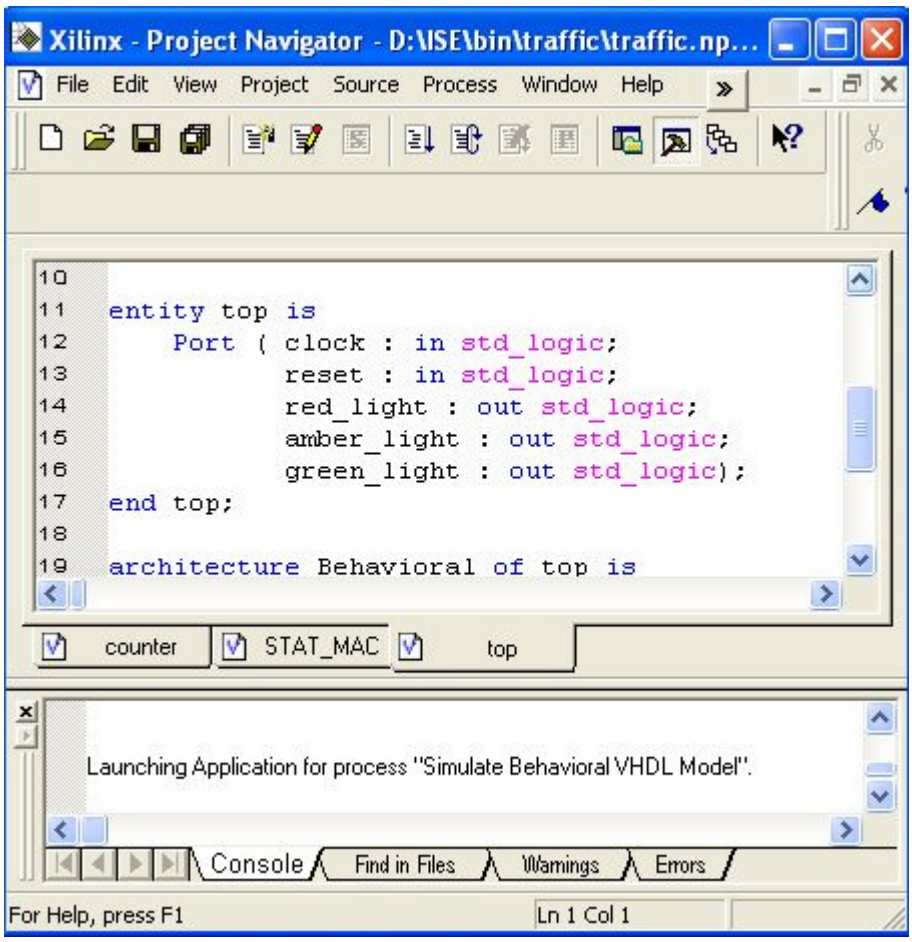


Şekil 4.35. Yeni modülün oluşturulması



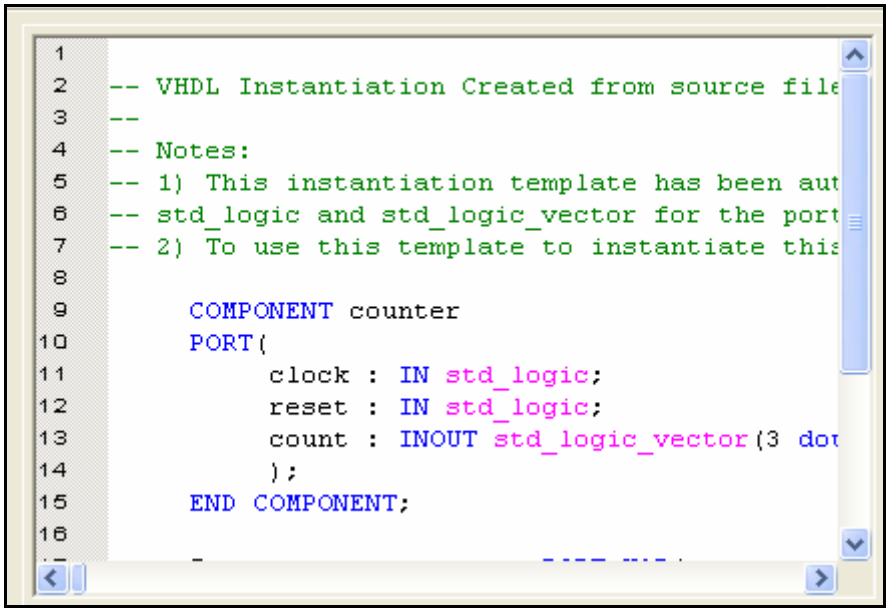
Şekil 4.36. Top modülü tanımlamaları

Daha sonra *ileri* butonuna basılıp modül kontrol edilir ve *son* butonu tıklanır. Aynen sayıcı için yaptığımız gibi hazır VHDL kod bloğu ekrana gelir.(Şekil 4.37) Böylece *top.vhd* kaynaklara eklenmiş olacaktır.



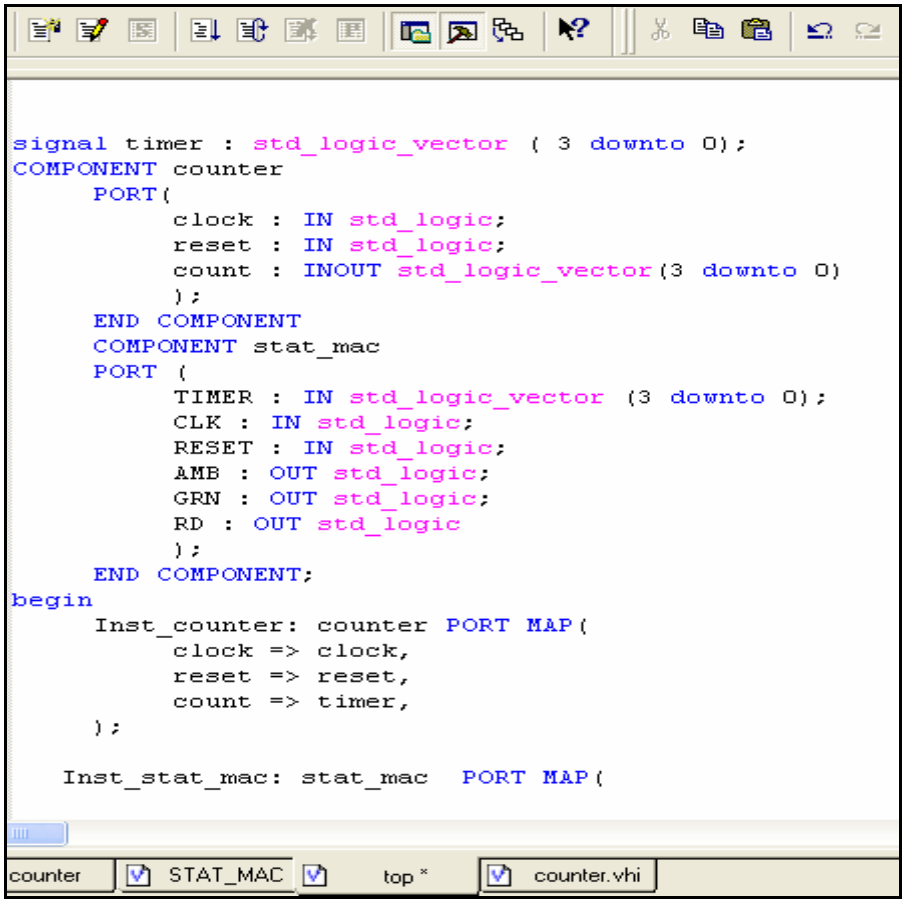
Şekil 4.37. Top modülü VHDL kodları

Kaynak penceresinde *counter.vhd* seçilir ve sonra *process* penceresinden '*view VHDL instantion template'* çift tıklanır. Ekrana port tanımlama kodları gelecektir (Şekil 4.38). Bu kod kopyalanıp *top.vhd* dosyası içerisine yapıştırılır ve Şekil 4.39'da olduğu gibi düzenlenir.



```
1
2   -- VHDL Instantiation Created from source file
3   --
4   -- Notes:
5   -- 1) This instantiation template has been aut
6   -- std_logic and std_logic_vector for the port
7   -- 2) To use this template to instantiate this
8
9       COMPONENT counter
10      PORT(
11           clock : IN std_logic;
12           reset : IN std_logic;
13           count : INOUT std_logic_vector(3 dow
14           );
15      END COMPONENT;
16
```

Şekil 4.38. Port tanımlama kodları

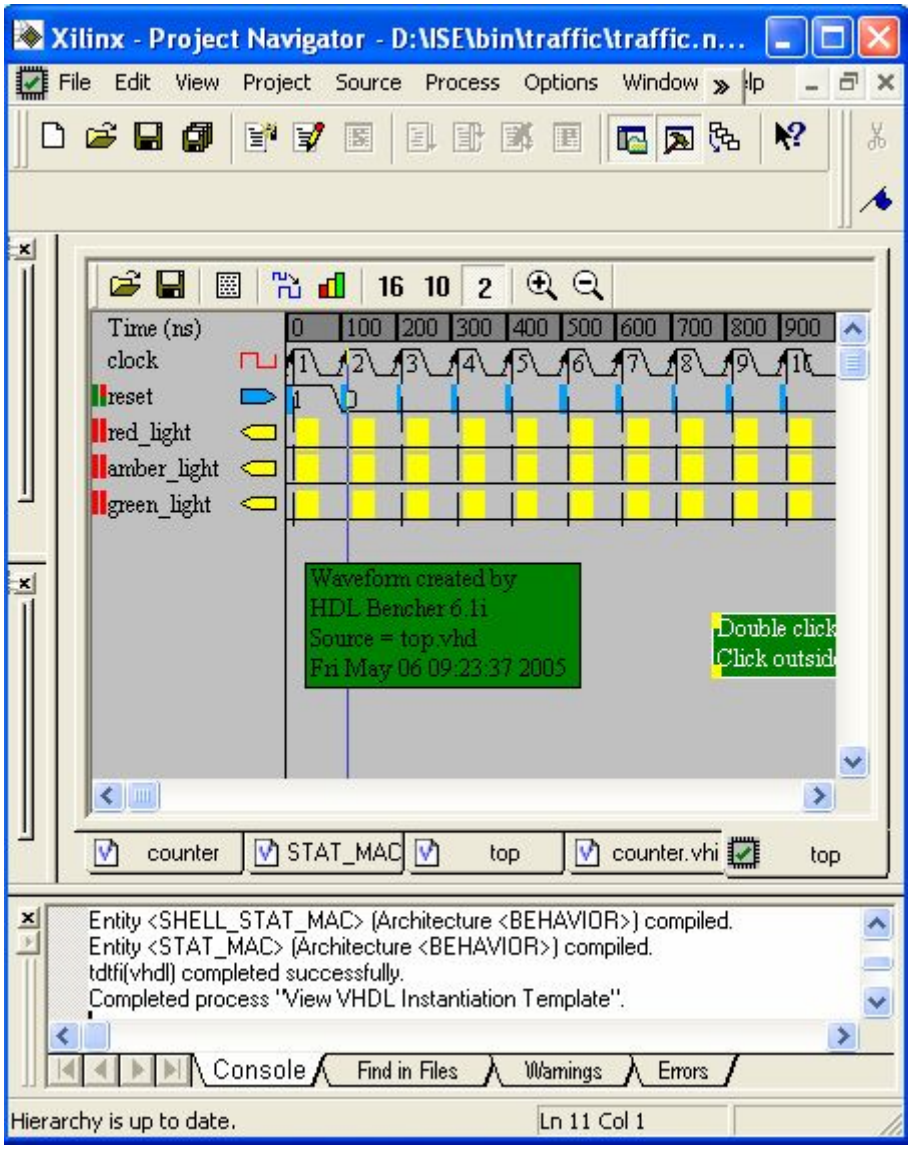


```
signal timer : std_logic_vector ( 3 downto 0);
COMPONENT counter
     PORT(
          clock : IN std_logic;
          reset : IN std_logic;
          count : INOUT std_logic_vector(3 downto 0)
          );
     END COMPONENT
     COMPONENT stat_mac
     PORT (
          TIMER : IN std_logic_vector (3 downto 0);
          CLK : IN std_logic;
          RESET : IN std_logic;
          AMB : OUT std_logic;
          GRN : OUT std_logic;
          RD : OUT std_logic
          );
     END COMPONENT;
begin
     Inst_counter: counter PORT MAP(
          clock => clock,
          reset => reset,
          count => timer,
     );

   Inst_stat_mac: stat_mac  PORT MAP(
```

counter | STAT_MAC | top * | counter.vhi

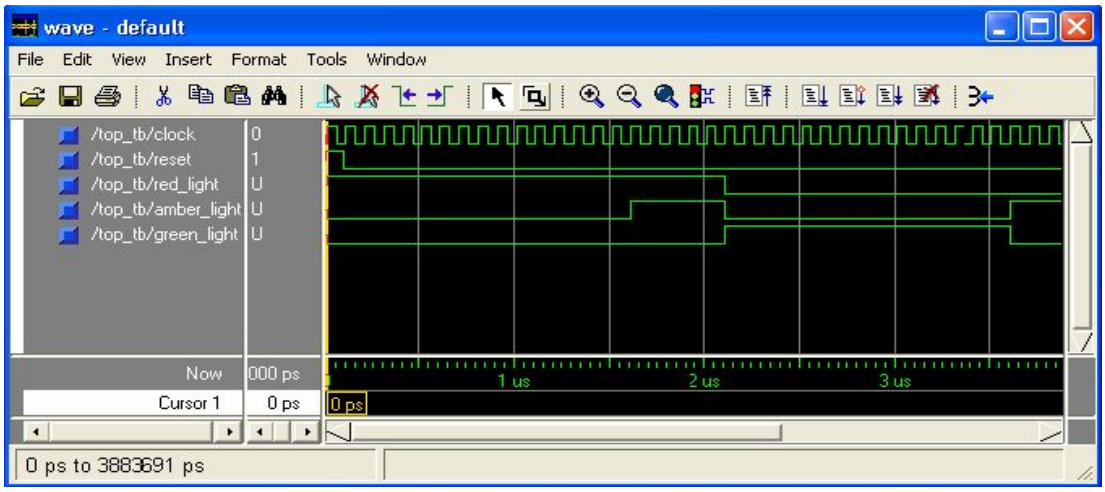
Şekil 4.39. Top modülü düzenlenmesi

Top modülü bu haliyle kaydedilir, kaynak penceresinde *sayıcı.vhd* ve *stat_mac.vhd* dosyaları *top.vhd* dosyasının alt dosyaları olmaktadır. Şu anda devre simüle edilebilecek konuma gelmiştir. Bunun için öncelikle sayıcı için yaptığımız gibi bir *testbench* oluşturulur (Şekil 4.40). Fakat bu kez ismi 'top' olacaktır.



Şekil 4.40. Top modülü için testbench

Top modülü için *testbench* oluşturulduğuna göre ModelSim XE ile simülasyonu yapılabilir. Bunun için sayıcı tasarımında yaptığımız gibi ModelSim analizörü açılır (Şekil 4.41).



Şekil 4.41. Yapılan tasarımın ModelSim'de görülmesi

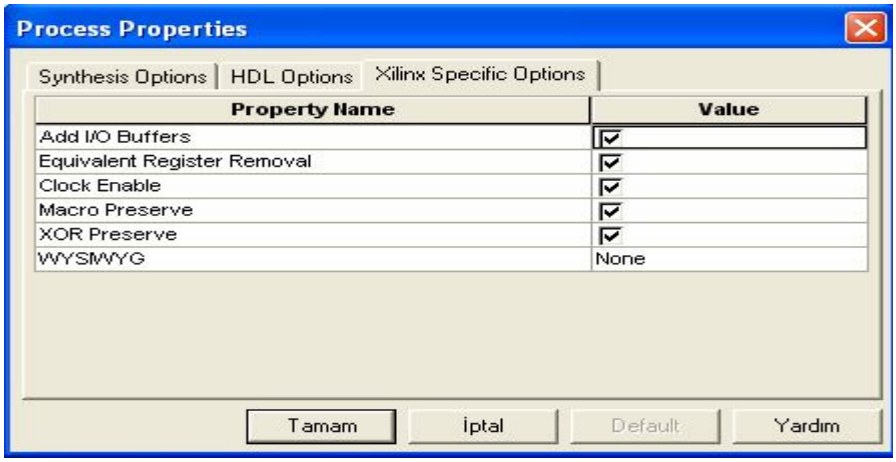
Analizördeki led sinyalleri takip edilirse devre reset aldıktan sonra ilk önce kırmızı led bir süre yanmaktadır. Kırmızı led sönmesine yakın sarı (amber) ışıkla beraber yanmakta ve daha sonra her ikisi sönüp sadece yeşil ışık yanmaktadır. Yeşil ışık söndüğünde ise önce sarı ışık bir süre yanmakta ve daha sonra sarı ışık sönüp tekrar kırmızı ışık yanmaktadır. Sonuç olarak bir trafik lambası gibi çalışmaktadır. Bu işlemler *clock* sonuna kadar periyodik olarak tekrarlanmaktadır.

Tasarım başarıyla simüle edildiğine göre şimdi bir CPLD içerisine yerleştirilebilir. Bunun için ilk aşama ISE sentez bölümünün metin tabanlı tasarımı bir netlist dosyasına dönüştürmesidir. Daha sonra netlist dosyasının seçilen CPLD mimarisi ile uyumlu olup olmadığı kontrol edilir. Kısaca sentezlenmiş tasarım CPLD içerisinde kullanılacak duruma getirilir. Diğer bir aşama tasarımın kullanılacak olan CPLD'ye uydurulmasıdır. Örneğin D tipi bir flip flop değilde T tipi flip flop kullanılabilir, bu değişikliği uyumlaştırıcı yapar. Uyumlaştırmada tamamlandığında ModelSim XE simülasyon için bilgileri kullanabilir. Uyumlaştırıcı aynı zamanda CPLD'nin programlanmasını sağlayacak olan JED uzantılı JEDEC dosyasını üretir. Daha sonra giriş ve çıkışlar CPLD pinleri için ayarlanır.

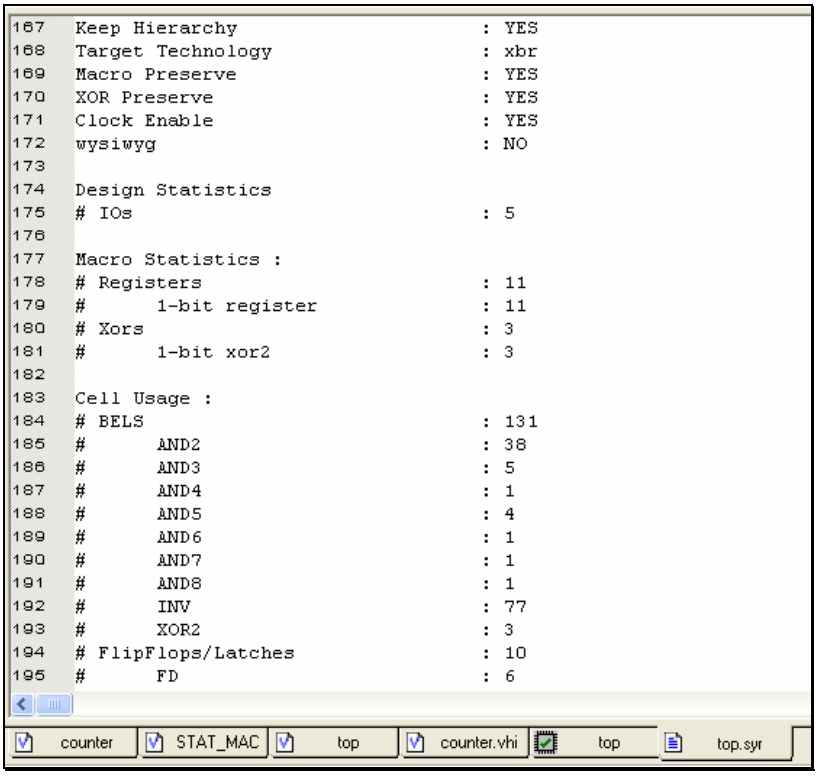
Kaynak penceresinden *top.vhd* seçilip, *processes* pencersinden *syhnthesis* tıklanarak içerik doğruluk testi yapılır. Eğer hata varsa hangi satırda olduğu belirtilir. Hatasız durumda yeşil bir tik işareti görülür. Daha sonra *snythesis* üzerinde sağ tıklanarak,

özellikler seçilir. Buradan giriş çıkış tamponları seçeneği aktif yapılır.(Şekil 4.42) Sentez tasarımı hiçbir şekilde değiştirmeyecektir ancak tasarımın eleman içerisinde nasıl oluşacağını büyük ölçüde belirler. Sentez özellik penceresi kapatılır ve sentez üzeri çift tıklanır. Bu kez sentez doğrulanır ve bir sentez raporu görüntülenir (Şekil 4.43).

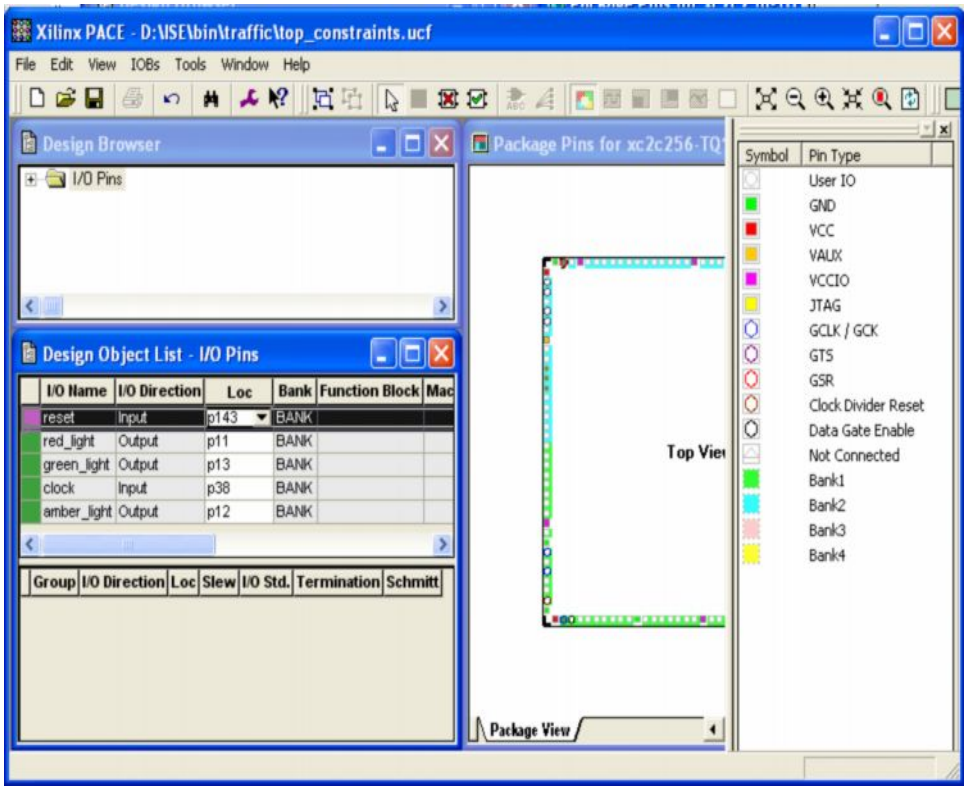
CPLD'den yüksek performans almak için bir takım sınırlılıkların programda sunulması gerekmektedir. Örneğin CPLD pin sınırlaması gibi. Bunun için önce *implementation constraints file* tipinde *top_constraints* adında yeni bir kaynak oluşturulur ve *top* modülüyle ilişkilendirilir. Bu dosya UCF (kullanıcı sınırlama dosyası, user constraint file) uzantılı olacaktır. *Processes* penceresinde *user constraints* bölümü genişletilerek *assign package pin* sekmesi çift tıklanır. Önce çevirme işlemi otomatik olarak gerçekleşir ve uygulama katı netlist dosyasını okur. Ekrana Xilinx PACE (pin ayarları ve sınırlama editörü, pinout area and constraint editor) penceresi gelir.(Şekil 4.44) Burada başlangıçta belirlenen giriş ve çıkış isimleri otomatik olarak belirlenir. Bunların karşılarına CPLD'nin hangi pinlerine karşılık geleceği yazılır. Her bir pin yazılırken sağdaki bölümde CPLD kılıfı üzerinde ilgili pin parlar. Pinler girildikten sonra PACE penceresi kaydedilir ve kapatılır.



Şekil 4.42. Sentez özellik penceresi

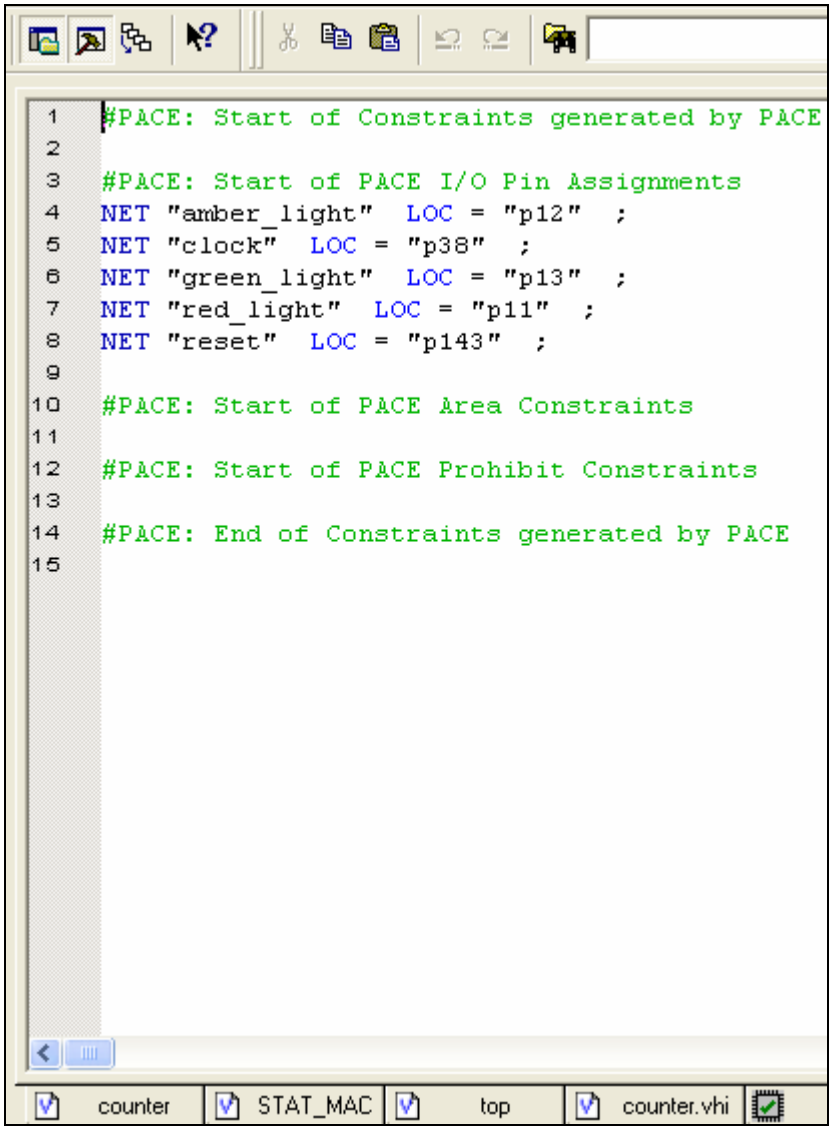


Şekil 4.43. Sentez raporu



Şekil 4.44. PACE penceresi

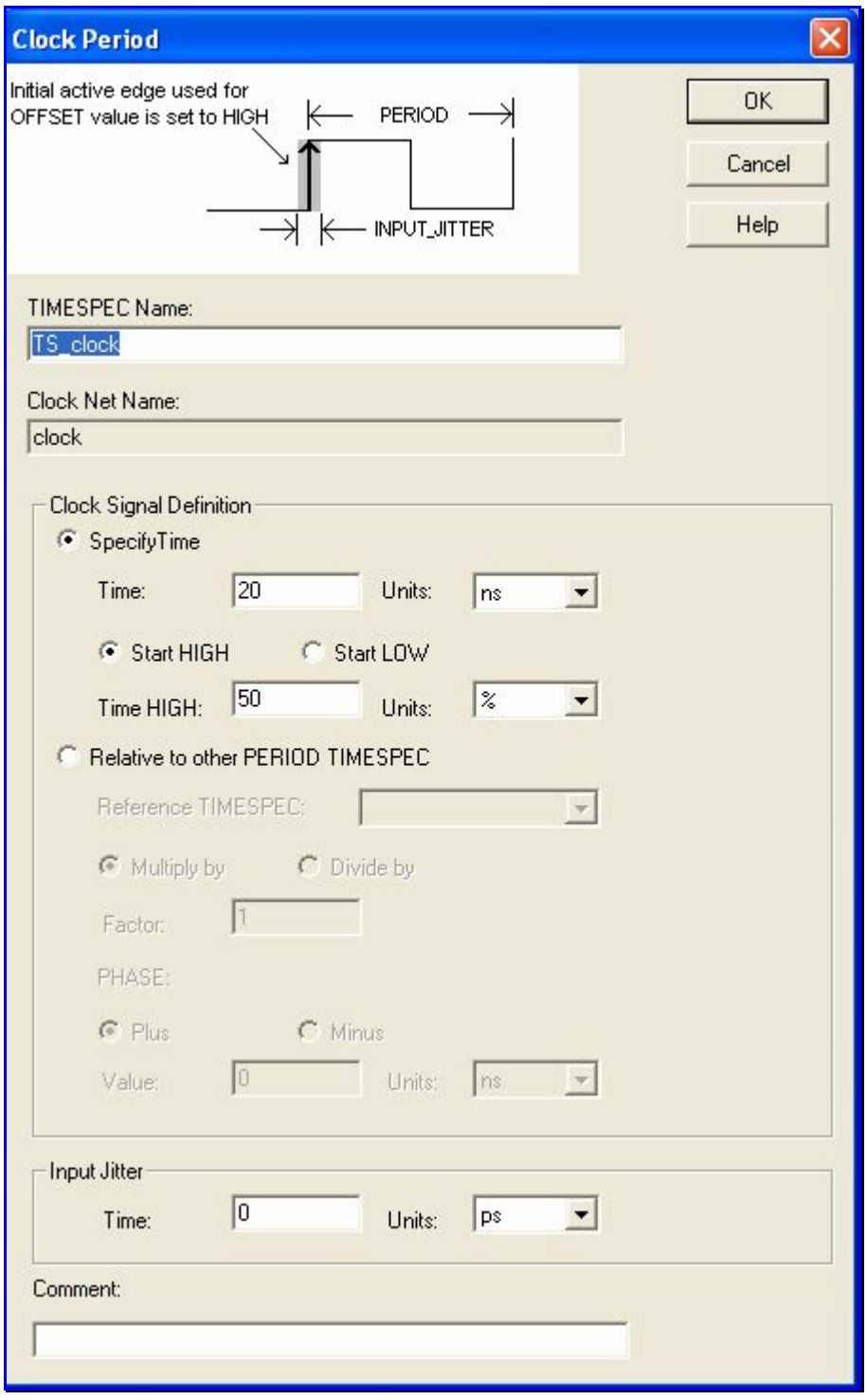
Şimdi bu pin numaraları UCF dosyasında görülebilecektir. Kaynak penceresinde UCF seçiliyken *processes* penceresinde 'edit constraints' çift tıklanırsa ekrana Şekil 4.45'deki pencere gelir. Bu dosyada PACE penceresinde girilen bilgiler görülür.



Şekil 4.45. UCF dosyası

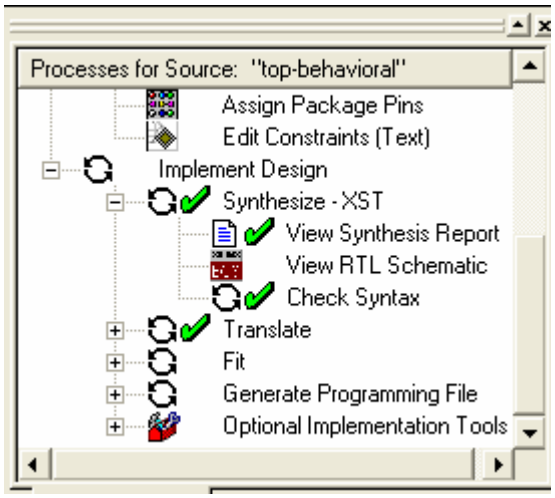
Şimdi clock zaman ayarlarını yapmak için processes penceresindeki 'create time constraints' seçeneği çift tıklanır. Ekrana gelen pencereden period çift tıklanırsa ekrana Şekil 4.46'daki pencere gelir. Buradan clock periyodu ayarlanır. Zaman

değeri 10 ns yapılır. OK butonuyla önceki pencereye dönülür. Burada zamanlama ile ilgili diğer ayarlar yapılır. Sonra kaydedilerek kapanır.

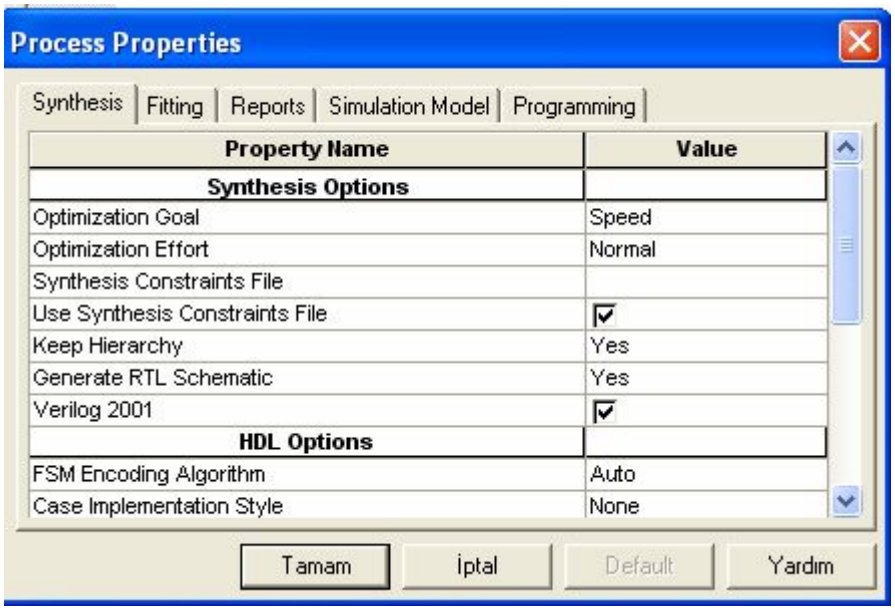


Şekil 4.46. Clock periyodu

Şu anda işlem penceresinde uygulama adımları (implement design) görülebilmektedir.(Şekil 4.47) Bu seçenek üzerinde çift tıklanırsa tüm uygulama adımları için nelerin seçilip seçilmediği görülebilecektir.(Şekil 4.48)



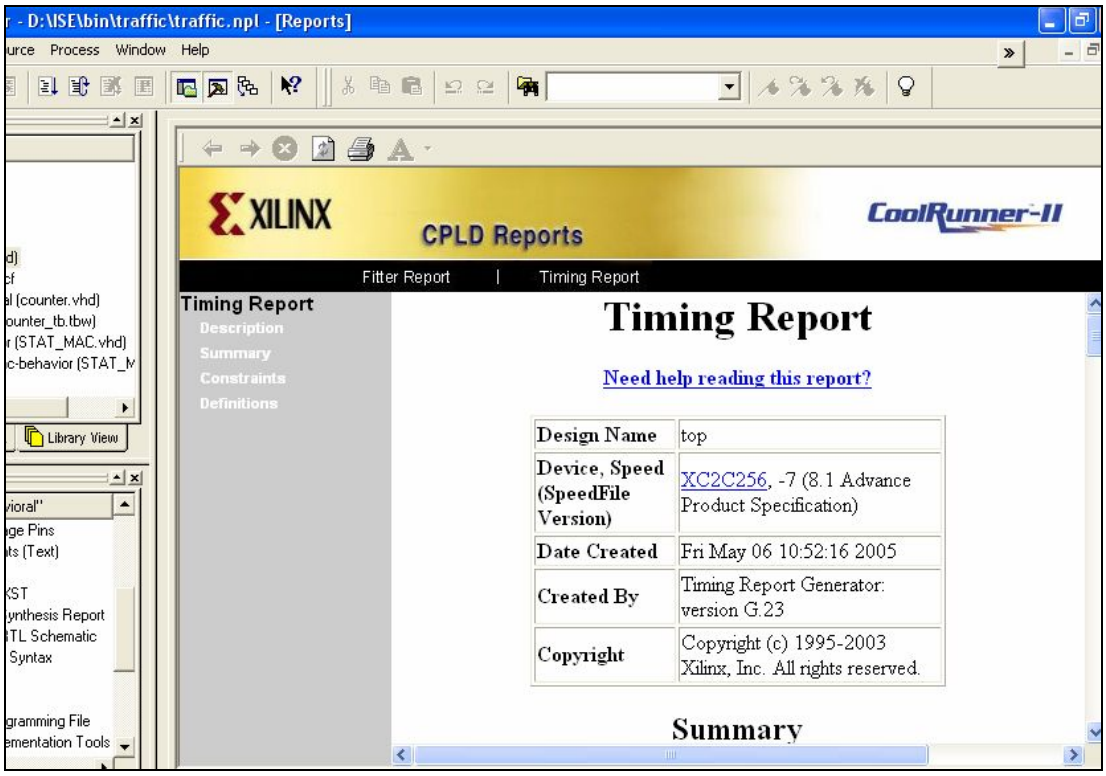
Şekil 4.47. Uygulama adımları



Şekil 4.48. Uygulama adımları açıklamaları

Şu anda tasarım işlem pencersindeki *implementation* seçeneği çift tıklanarak uygulatılabilir. Sonucunda iki tane CPLD raporu görülebilir, zaman ve uyum raporu. Zaman raporu tasarım ile yapılan tüm ayarları, seçilen CPLD özelliklerini gösterir. Zaman raporunu görmek için önce işlem penceresindeki *optional implementation*

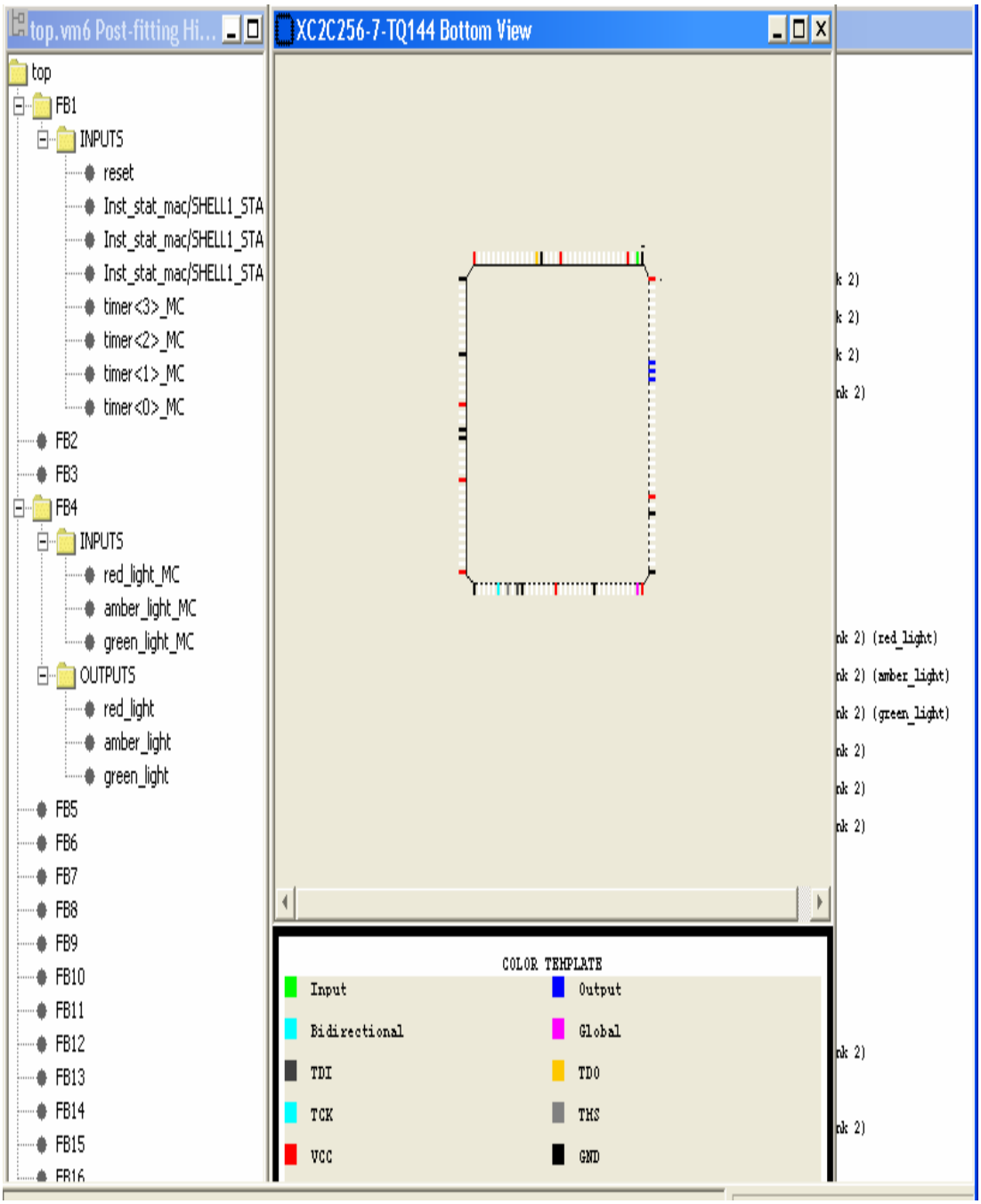
tools sekmesi sonrada bu sekme altındaki *generate timing* sekmesi genişletilir. Buradan *timing report* çift tıklanırsa rapor ekranda belirir. (Şekil 4.49)



Şekil 4.49. Zaman raporu

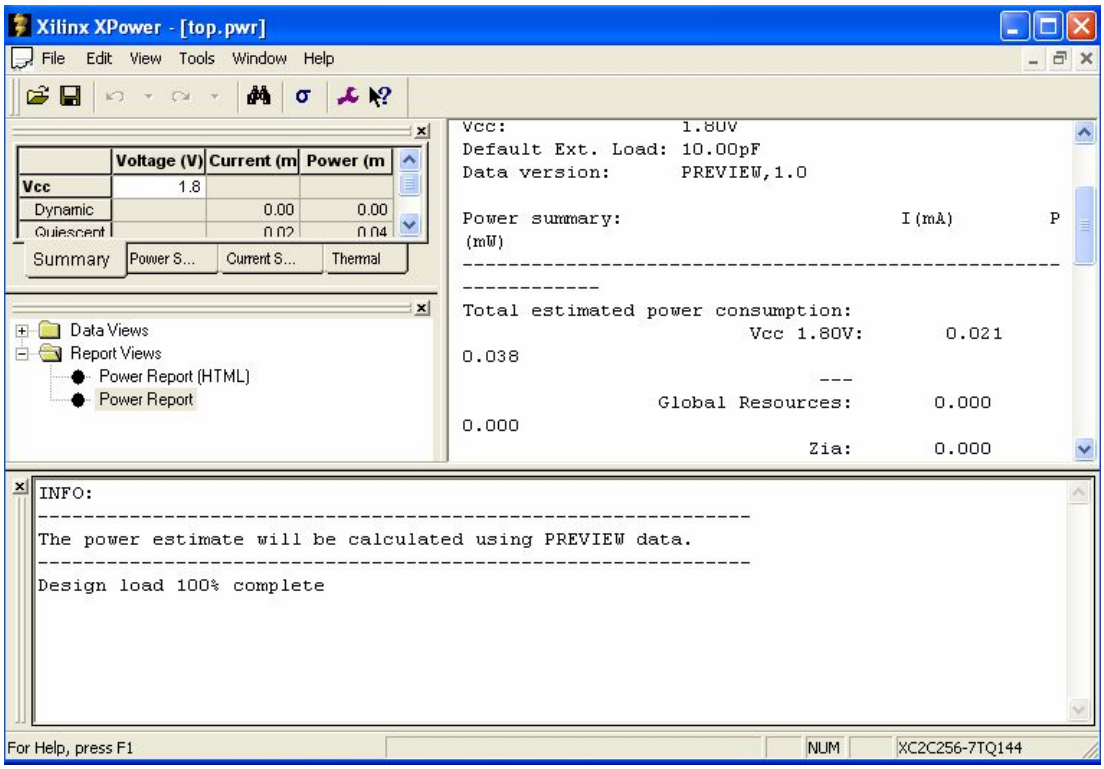
Uyum raporu için işlem penceresinde fit sekmesi genişletilir ve 'fitter report' çift tıklanır. Uyum raporu kullanılan CPLD içerisindeki elemanlar ve tasarımda ne kadarının kullanıldığı özet bölümünde belirtilir. Hata ve uyarılar bölümünde uyumlaştırma sırasındaki olumsuzluklar görülebilir. 'Mapped inputs' ve 'mapped logic' bölümleri uyumlaştırılan tasarım içerisindeki sinyaller, makro hücreler ve pinler hakkında bilgi verir. 'Function Block' bölümünde her bir makro hücrenin içi görülebilir. Kullandığımız CPLD'de 16 fonksiyon bloğu bulunmaktadır.

Şekil 4.50 tasarlamış olduğumuz devrenin CPLD'ye aktarımından sonra, CPLD kılıfını ve giriş çıkış uçlarının hangileri olduğunu göstermektedir.



Şekil 4.50. Tasarıyla programlanmış CPLD pinlerinin görünümü

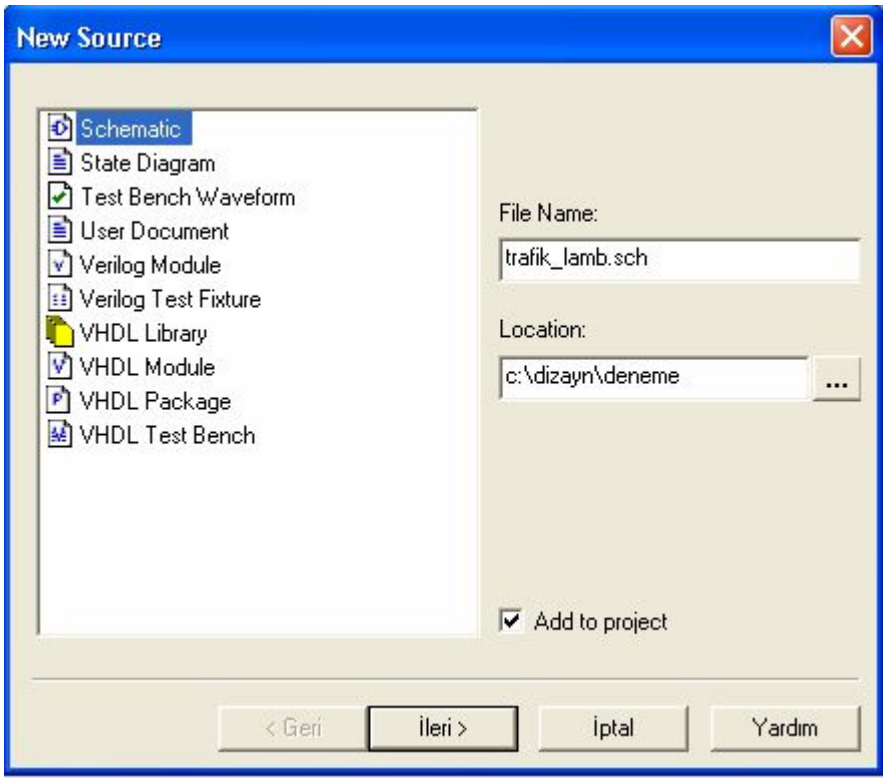
İşlem penceresinden ‘analysis power’ seçeneği çift tıklanırsa tasarımda CPLD’de harcanan güç değerleri ekrana gelir. (Şekil 4.51)



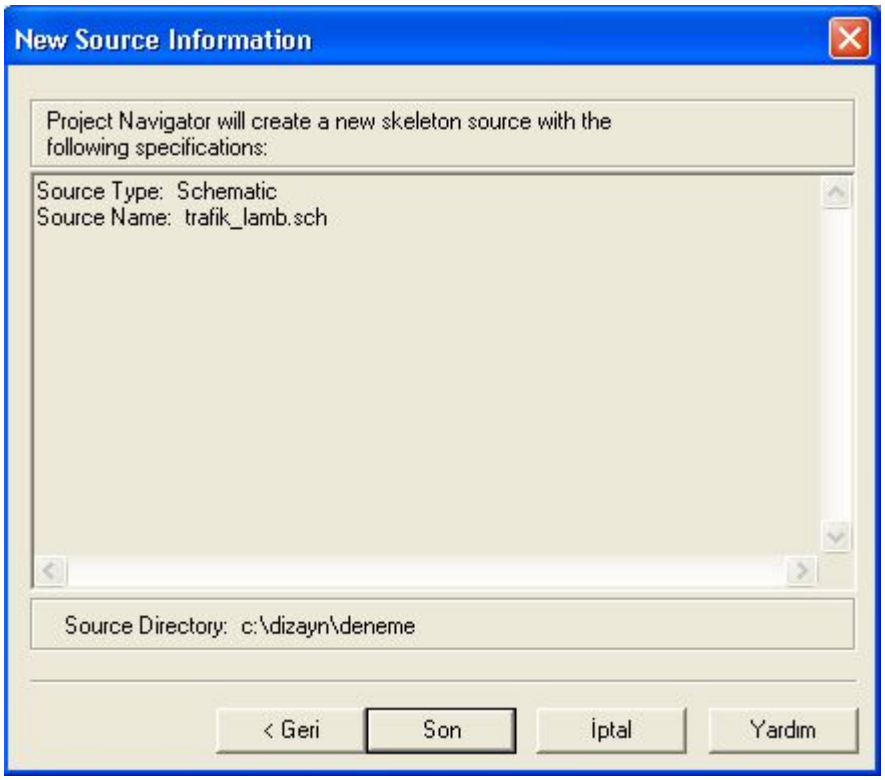
Şekil 4.51. CPLD güç analizi

4.3.2. Şematik tasarım yapılması

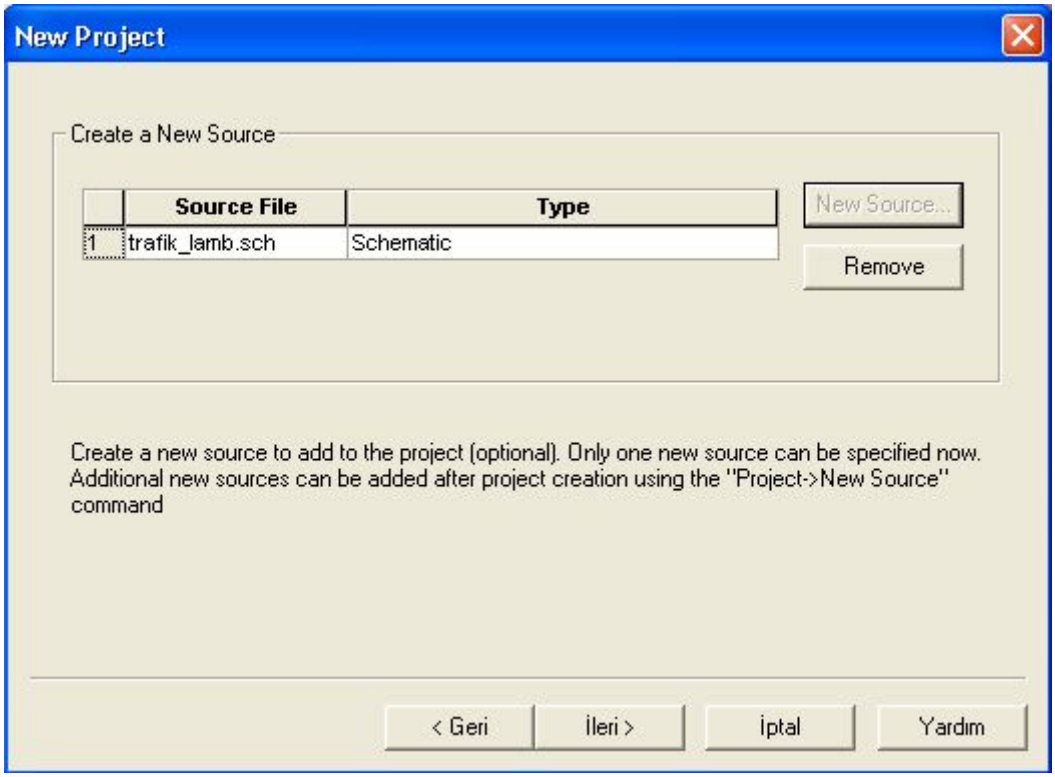
Önceki sayfalarda da bahsedildiği bibi WebPack ISE programı kullanılarak şematik tasarım yapmak ve bunu CPLD'ye aktarmak mümkündür. Yapılan şematik tasarım aynı şekilde ModelSim ile simüle edilebilmektedir. Diğer tasarımda olduğu gibi ISE ana yüzünden *File>New Project* seçilir, ekrana şekil 4.6'daki pencere gelecektir. Bu penceredeki *Top-Level Module Type* kısmından *Schematic* seçilecek ve ileri butonu tıklanır. Ekrana yine Şekil 4.7. gelecek. Bu pencerede de ilgili yerler seçildikten sonra aşağıdaki Şekil 4.52 gelecektir. Bu pencerede ileri tuşuna basıldığında Şekil 4.53 gelecek ve burada da son tuşuna basıldığında Şekil 4.54 ekrana gelecektir. Burada daha önce New Source olarak yazdığımız trafik_lamb.sch isimli dosya görülecektir. Yine ileri tuşuna bastığımızda Şekil 4.55 gelecektir. Burada dikkat etmemiz gereken önemli bir nokta eğer kendi kütüphanesinden elemanlarla tasarımımızı oluşturacaksak ileri tuşuna basmak gerekir.



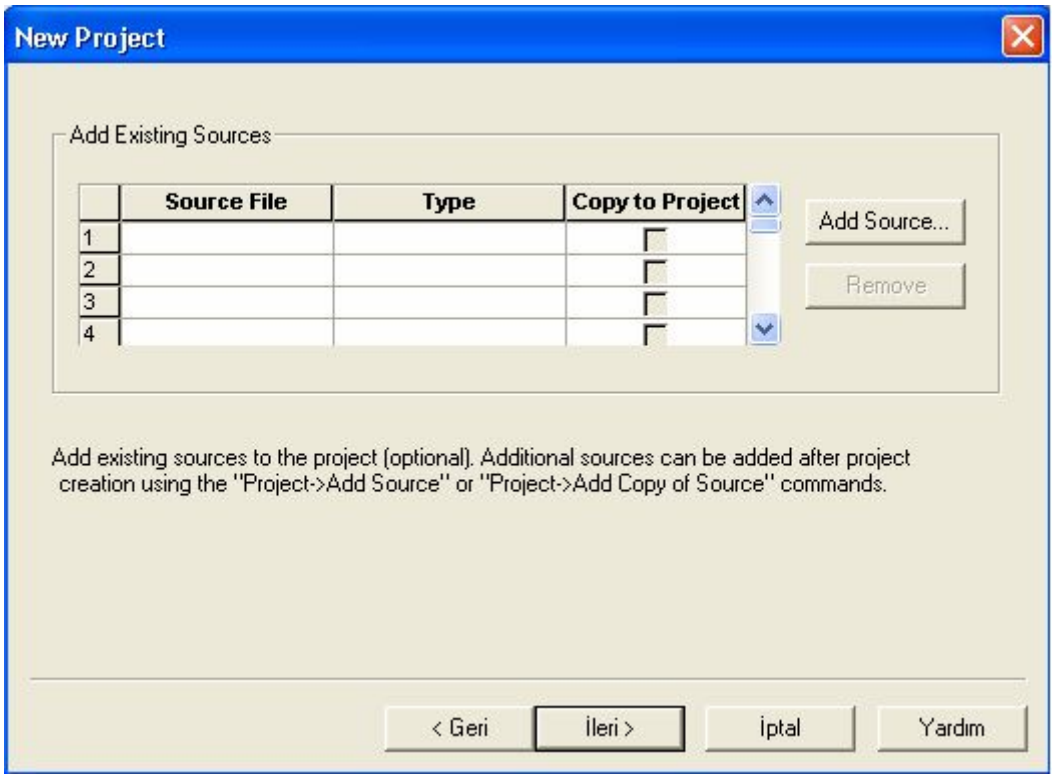
Şekil 4.52. Şematik tasarım penceresi



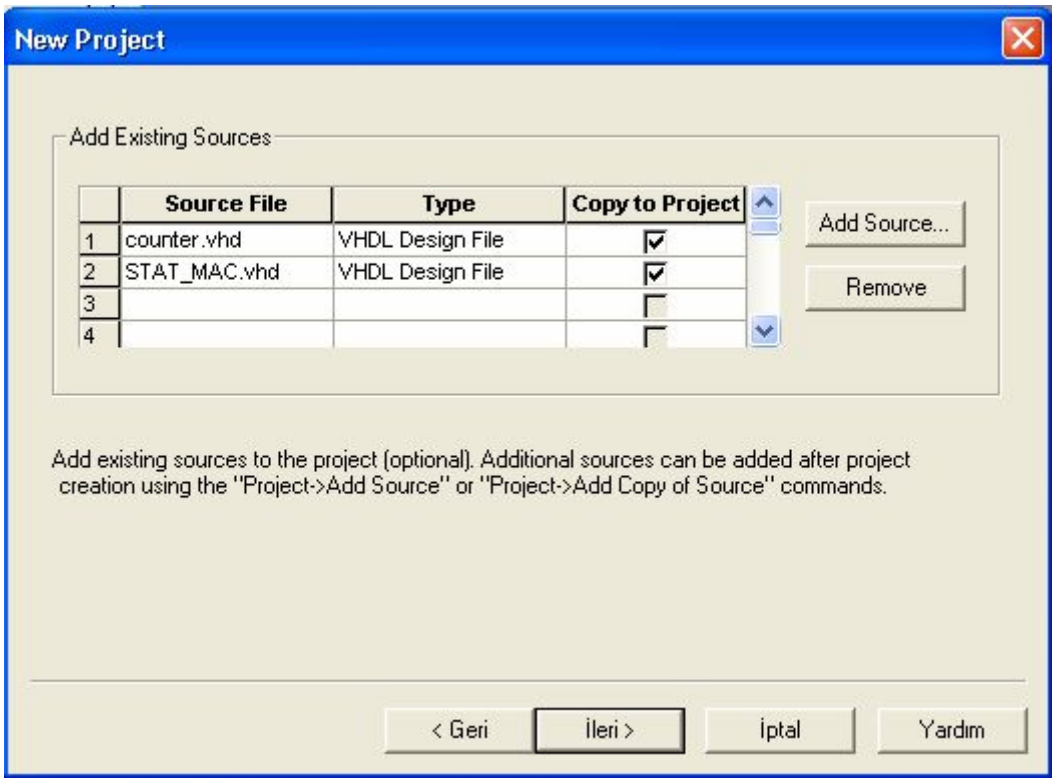
Şekil 4.53. Şematik tasarım dosyasının oluşturulması



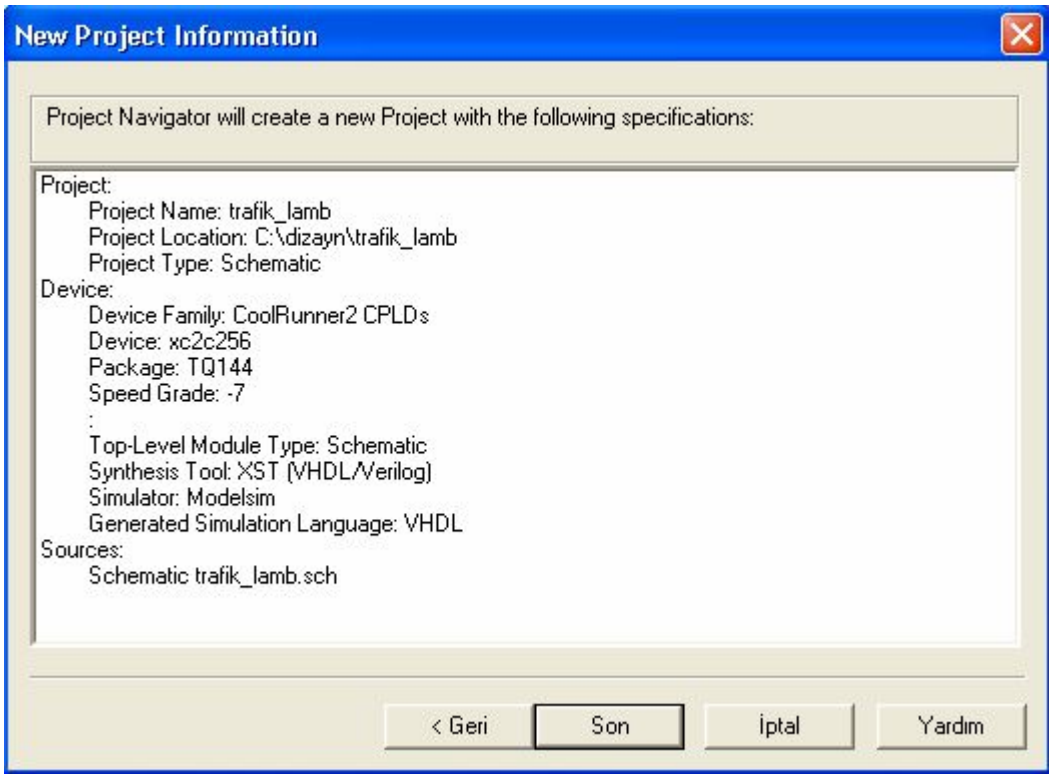
Şekil 4.54. "sch" uzantılı dosyanın eklenmesi



Şekil 4.55. Tasarıma şematik dosyanın eklenmesi



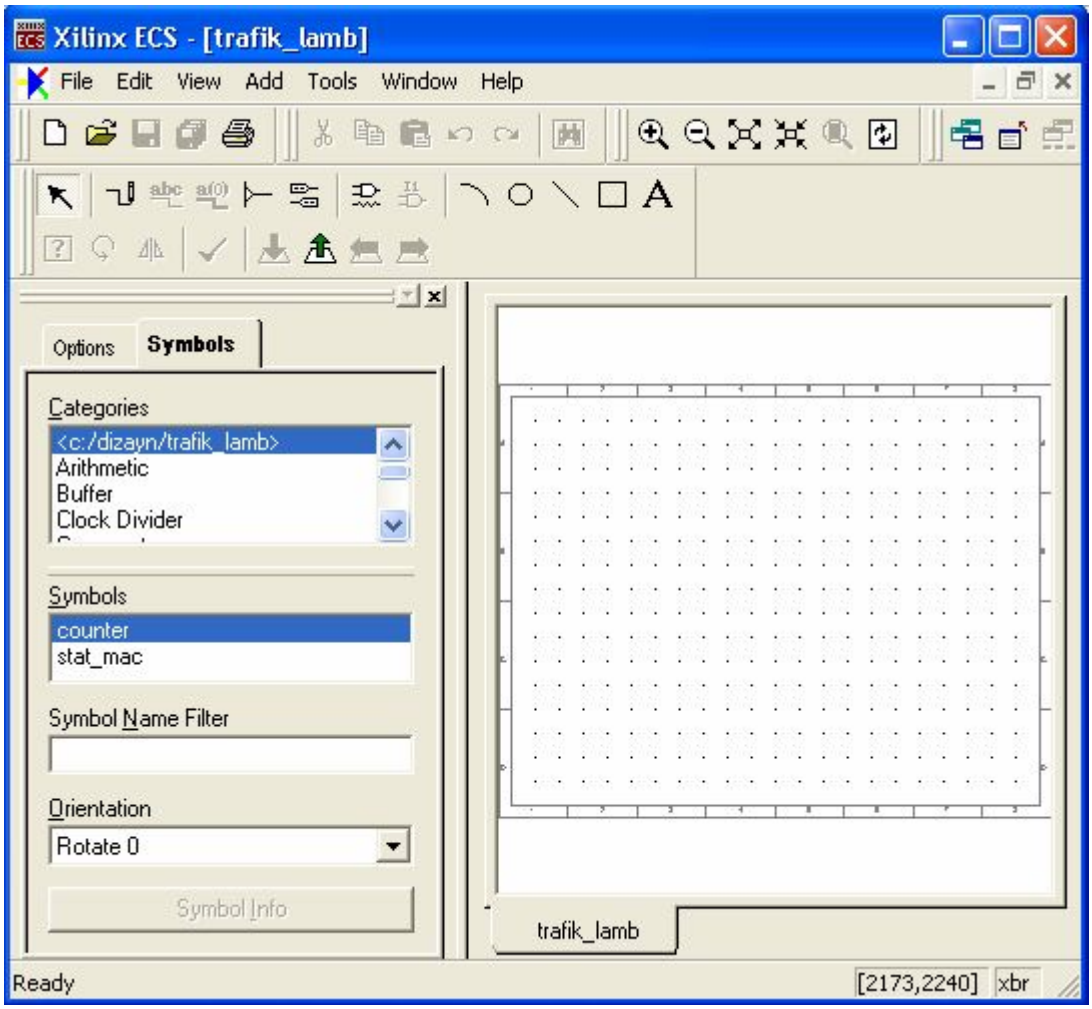
Şekil 4.56. Eklenecek dosyaların seçilmesi



Şekil 4.57. Tasarımla ilgili bilgilerin görülmesi

Eğer kendimiz VHDL kodlarını yazarak oluşturduğumuz elemanın gelmesini istiyorsak *Add Source* yaparak kaynak VHDL dosyasını gösteririz.(Şekil 4.56) İleri butonuna basıldıktan sonraki pencere Şekil 4.57. gelir ve buradaki son tuşuna bastığımızda şematik çizim editörü penceresi (Şekil 4.58) gelir. Burada tasarımda kullanacağımız sembolleri sembol alt penceresinde bulmak mümkündür. Bu pencerede bizim oluşturduğumuz elemanları ve programın kendi kütüphanesinde bulunan çeşitli sayısal devre elemanlarını bulmak mümkündür.

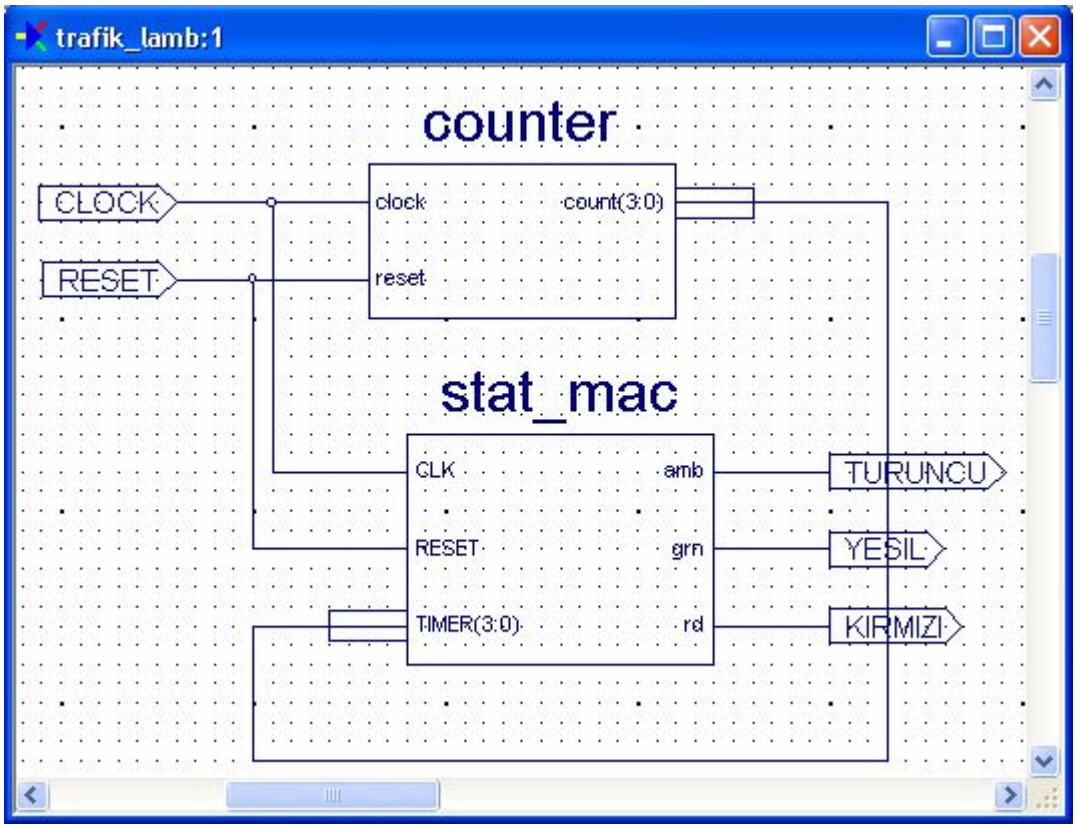
Sembol penceresinden VHDL kodlarıyla kendimizin oluşturduğu c:/dizayn/trafik_lamb kategorisinin içindeki “counter” ve “stat_mac” isimli elemanları şematik tasarım alanına sırayla yerleştirme yaparız.



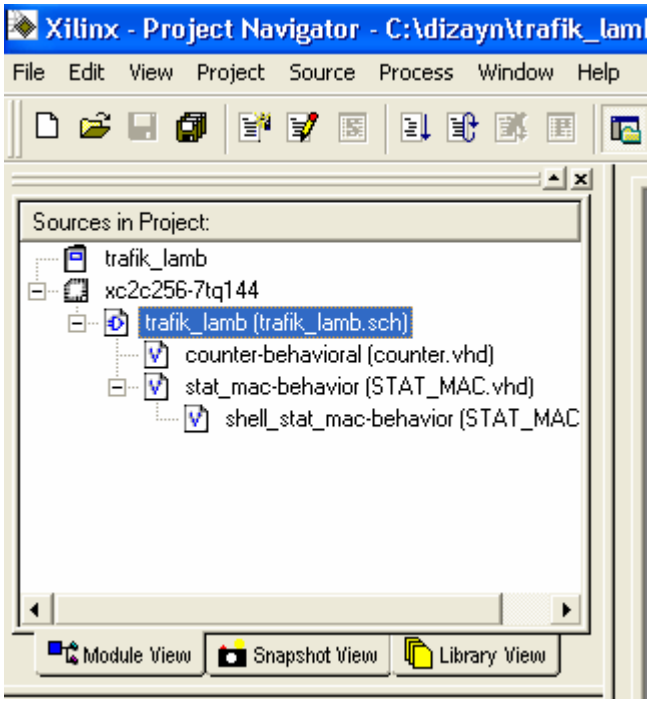
Şekil 4.58. Şematik çizim penceresi

Şekil 4.58'deki gibi *symbol* kısmında görülen devre elemanlarını yerleştiririz. Eğer başka sayısal elemanlarda eklenecekse "Add Symbol" ile devremize ekleme yapabiliriz. Eleman yerleştirme tamamlandıktan sonra menüdeki Add kısmından veya ekrandaki kısayol tuşlarından "Add Wire" ile elemanların bacak bağlantılarını yapabiliriz, elemanlardaki bağlantılarda grup bağlantıları tek bir hatla göstermek mümkün olacaktır "Add I/O Marker" ile giriş ve çıkışları seçebiliriz.

Devremizi Şekil 4.59'daki gibi tamamladıktan sonra "save" tuşuyla kaydederiz ve şematik editörü kapatırız.



Şekil 4.59. Şematik çizim

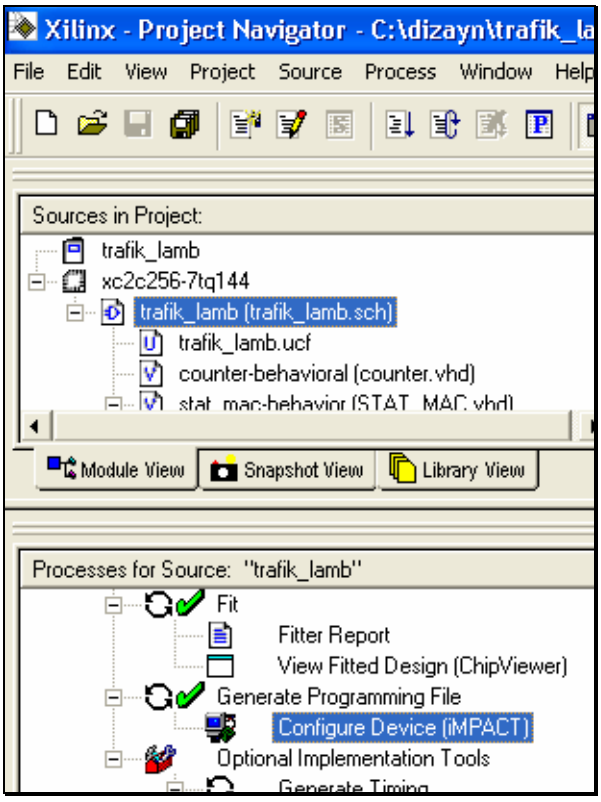


Şekil 4.60. Şematik çizimin proje penceresinde görünümü

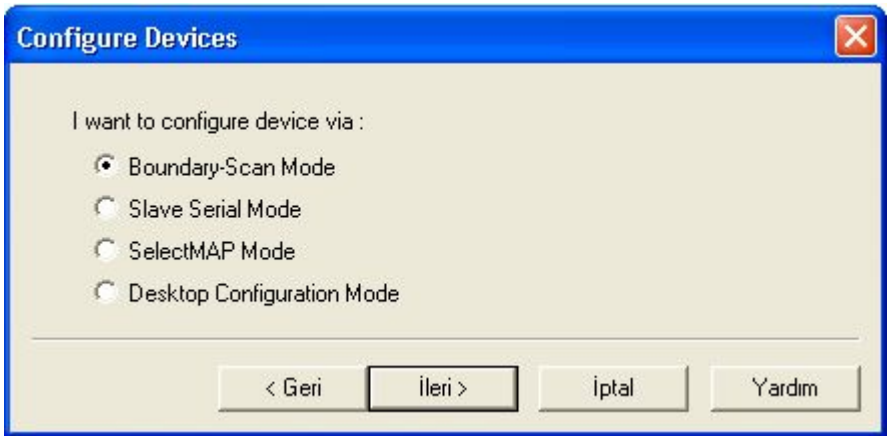
WebPack ISE' programına dönüldüğünde *Sources in Project* penceresinde kaydettiğimiz şematik dosyanın oluştuğunu görürüz.(Şekil 4.60) Bu aşamadan sonra yapılacak kısım simülasyon yapmak istenirse testbench dosyasını oluştururuz ve önceki bölümde anlatıldığı gibi Şekil 4.34'den sonraki basamakları aynen uygulayabiliriz. Tasarımın sentezi yapıldıktan sonra kullanacağımız pin numaralarını belirlemek için "Assign Package Pins" çift tıklanarak çalıştırılır ve pinlerin seçilmesi tamamlanır ve ekran kaydedildikten sonra kapatılarak WebPack ISE programına dönülür.

4.3.3. Tasarımın entegre devreye yüklenmesi

WebPack ISE programında *Sources in Project* penceresindeki *trafik_lamb.sch* seçilerek *Process for Source* penceresindeki *Generate Programing File* dizini altındaki *Configure Device (iMPACT)* çift tıklanarak entegre devrenin programlanması işlemi başlatılır.(Şekil 4.61)



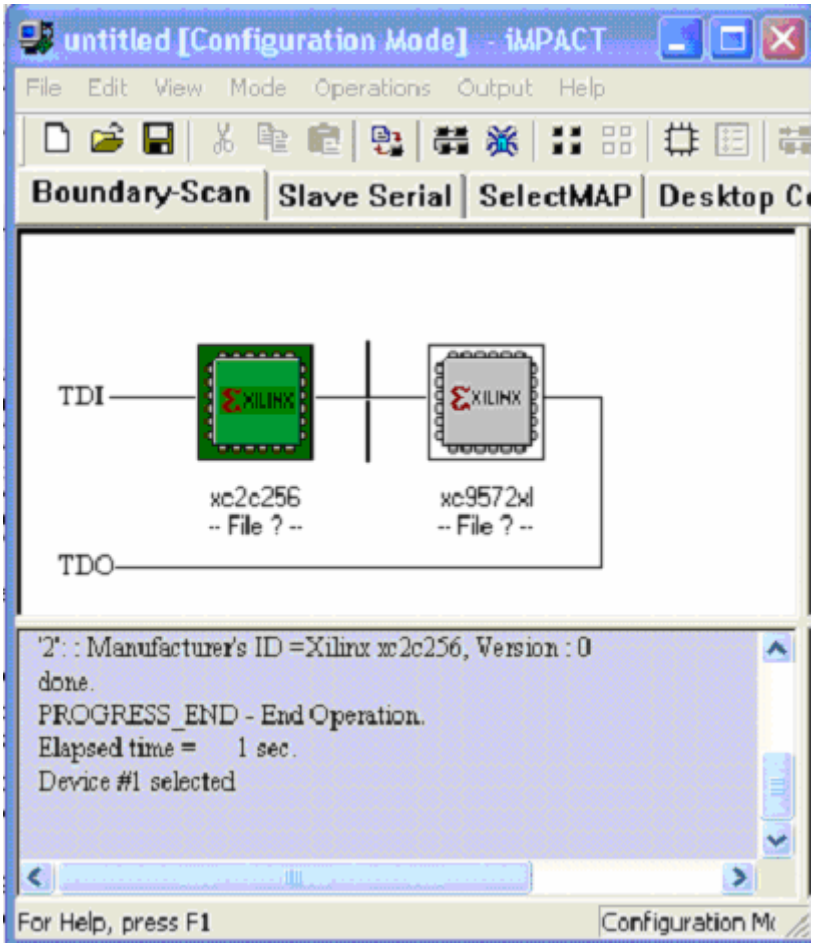
Şekil 4.61. Yüklenecek şematik tasarımın seçilmesi



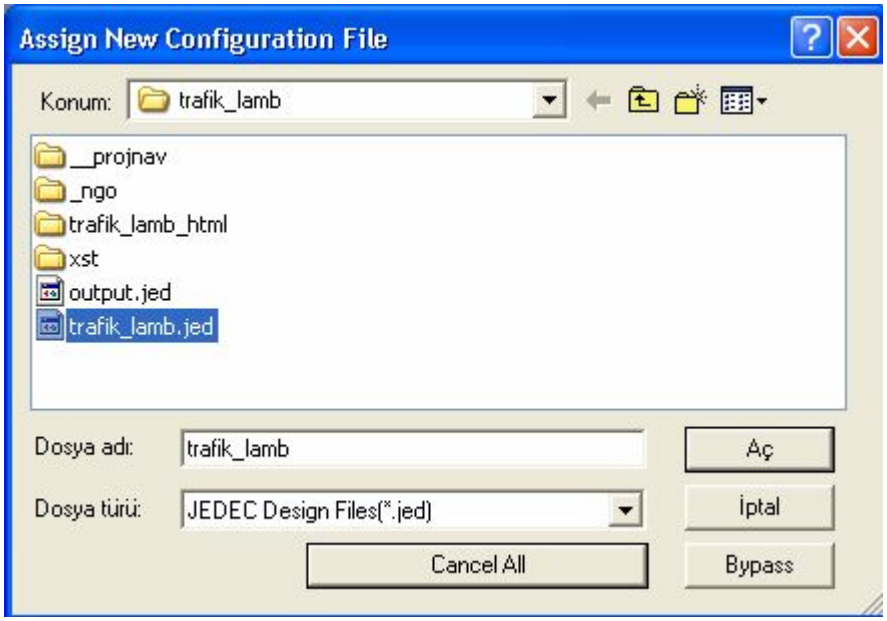
Şekil 4.62. Donanım tarama modu seçimi

Şekil 4.63. Bağlantının seçimi

Şekil 4.62'deki ekranda *ileri* ve Şekil 4.63'deki *son* tuşlarına basarak işleme devam edilir. Bu kısım bilgisayarımızın paralel portuna bağlanan JTAG kablo ile yapılmaktadır. Bu işlemlerden sonra program devremize bağlı olan programlanacak CPLD'leri otomatik olarak tarayacak ve bulacaktır.(Şekil 4.64.) Şekil 4.65'deki gibi konfigürasyon dosyasını soracaktır. *Trafik_lamb.jed* dosyasını seçerek *Aç* butonuna bastığımızda entegre devrenin altındaki file kısmına seçtiğimiz dosyanın adının geldiğini görürüz. Daha sonra 2. entegre devrenin de programlanıp programlanmayacağı sorulacaktır. İkinci entegre devre için gelen pencerede "Bypass" seçilerek entegre devre geçilebilir.

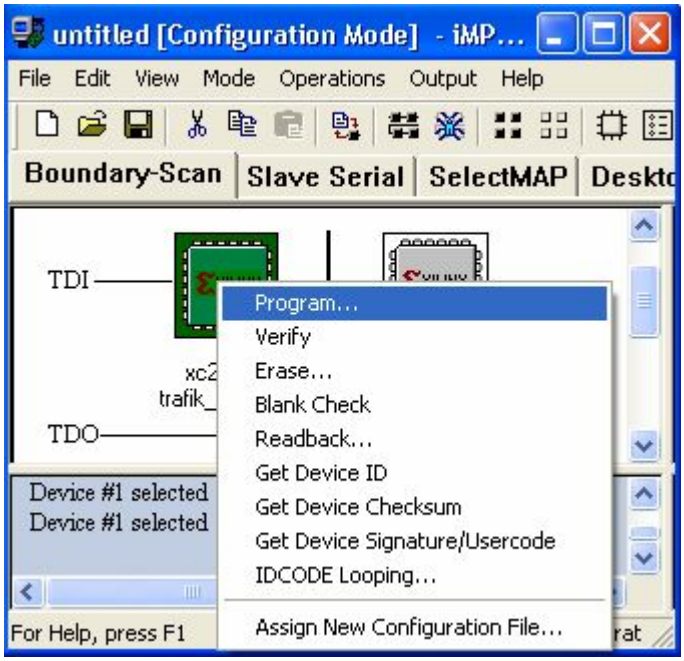


Şekil 4.64. Yükleme yapılabilecek entegre devrelerin görülmesi

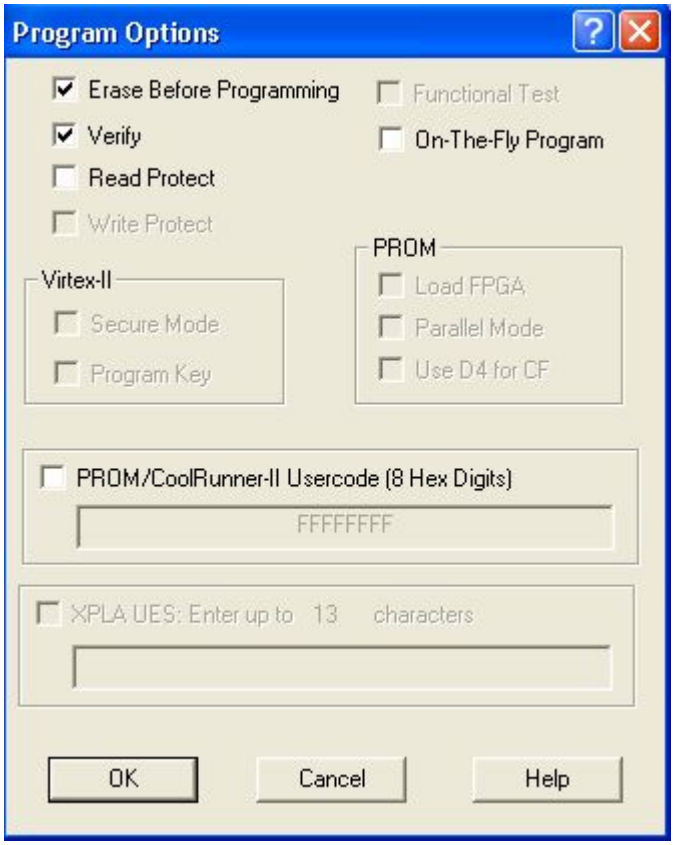


Şekil 4.65. Yüklenecek hedef dosyanın seçimi

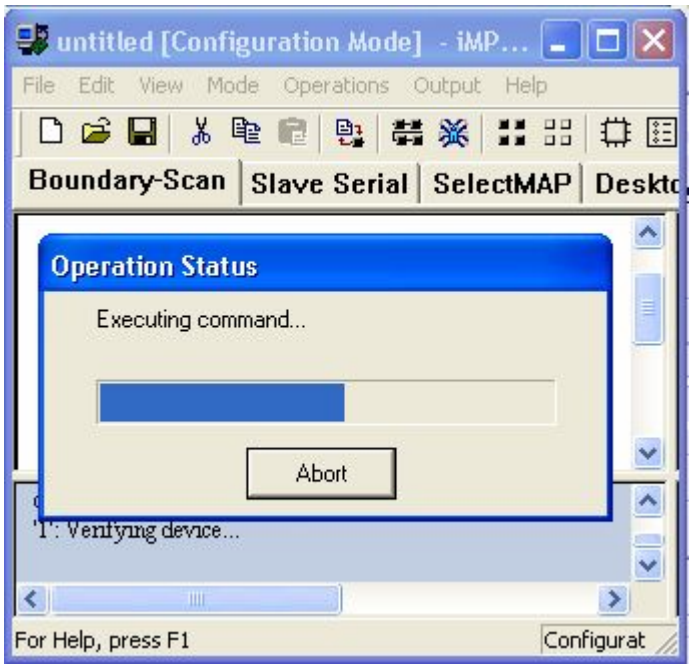
Bu işlemlerden sonra programlayacağımız entegre devrenin üzerine *fare* ile sağ tıklanarak gelen pencereden Şekil 4.66'daki gibi *Program* seçilir. Devamında gelen Şekil 4.67'deki pencerede entegre devre üzerinde silme, yazma, koruma gibi işlemler seçilerek *OK* tuşuna basıldığında entegre devremiz yüklenmeye başlayacaktır. Yükleme işleminin sonuçlanmasıyla ekrana Şekil 4.69'daki mesaj gelecektir. Bu ekran şimdiye kadar yaptığımız basamakların sonuçlandığını gösterir ve işlem başarıyla sonuçlanarak CPLD'miz yaptığımız tasarıma göre devre üzerinde programlanmıştır.



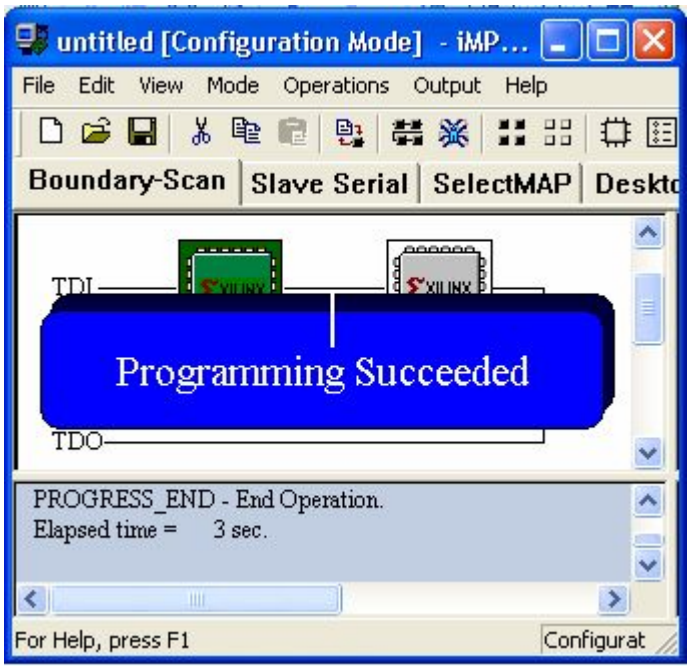
Şekil 4.66. Seçilen entegre devreye program yazılması



Şekil 4.67 Programlama penceresi



Şekil 4.68. Programlamanın başlaması



Şekil 4.69. Yüklemenin tamamlanması

5. SONUÇ

Bu tez çalışmasında, Xilinx firması tarafından üretilen CPLD'lerle, aynı firmaya ait olan WebPack ISE programı ile tasarım yapılması açıklanmıştır. Yapılan tasarımlarım ModelSim XE programı ile simülasyonu gösterilerek; yapılan tasarımların fiziki olarak gerçekleştirilmeden önce kontrollerinin yapılması sağlanmıştır.

Xilinx CPLD'lerin sayısal devre tasarımında tercih edilmeleri birçok üstünlüğünden kaynaklanmaktadır. Son sistem entegre devre üretim teknikleri kullanılarak imal edilmektedirler ve çok yüksek frekanslarda kararlı bir çalışma göstermektedirler. Değişik kılıflarda üretilebilen entegre devreler yüksek pin sayıları ile giriş/çıkış pinlerinin de fazla olmasını sağlamışlardır. Çok düşük güç tüketimleri ve besleme gerilimi ihtiyaçları ile gelişen teknolojide küçülen cihaz boyutlarında vazgeçilmez olma yolundadırlar. Fiyatlarının çok düşük olması ve devre üzerinde kolaylıkla programlanmaları da ayrı bir üstünlükleridir.

Tasarımcıya, tasarım esnasında kullanılan WebPack ISE programıyla birçok kolaylıklar sağlanmaktadır. Tasarımlar kullanıcının tercihine göre şematik çizim veya VHDL kodlarıyla yapılabilmektedir.

Dijital Elektronik dersi uygulamaları için oldukça kullanışlı ve basittir. Öğrenciler açısından yapılan uygulamaların pratiğe geçirilmesini hızlandırmaktadır. Ayrıca uygulamalar için elimizde çok sayıda sayısal elektronik entegre devrelerini bulundurmamıza gerek yoktur. Bir CPLD içerisinde binlerce kapı kullanarak tasarım yapmak mümkündür. Devre kurma süresini kısaltarak sabit kurulmuş bir devre üzerinde giriş/çıkış pinleri sabit kalmak şartıyla tasarımda değişiklikler yapılabilir.

CPLD, PIC ve benzeri mikro denetleyicilere göre ülkemizde yeteri derecede tanınmamaktadır. Oysa uygulama, programlama ve hız bakımından belirgin bir üstünlükleri vardır. Yakın gelecekte bu tür çalışmalarla ülkemizde ve Dijital Elektronik derslerinde gerekli yeri bulacaktır.

KAYNAKLAR

1. Chen S. L., Hwang T. T., Liu C. L., "A Technology Mapping Algorithm for CPLD Architectures", ***Proc. of IEEE International Conference on Field Programmable Technology***, Hong Kong, 204-210, (2002).

2. Zilic Z., Lemieux G., Loveless K., Brown S., Vranesic Z., "Designing for High Speed-Performance in CPLDs and FPGAs", ***The 3rd Canadian Workshop on Field-Programmable Devices (FPD'95),*** Kanada, 108-113, (1995).

3. Vocke N. J, Stroud C. E., Heath J. R., "Computer Aided Routing for Complex Programmable Logic Device Manufacturing Test Development", ***Proceedings of the IEEE***, Southeastcon, 171-176, (2000).

4. Nag S. K., Rutenbar R. A., "Performance-Driven Simultaneous Placement and Routing for FPGA's", ***IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems***, 17(6): 499-518, (1998).

5. Mano, M. M., "Sayısal Tasarım 3. Baskı", Sinan Akbaytürk, Burhanettin Can, ***Milli Eğitim Yayınevi***, İstanbul, 31-32, 202-209, (2000).

6. Yağımlı M., Akar F., "Dijital Elektronik 3. Baskı", ***Beta***, İstanbul, 372-377, (2000).

7. Newman K. E., Hamblen J. O., Hall T. S., "An Introductory Digital Design Course Using a Low-Cost Autonomus Robot", ***IEEE Transactions on Education***, 45(3): 289-296, (2002).

8. Popescu S. "Hardware Implementation of Fast Neural Networks Using CPLD", ***5th Seminar on Neural Network Applications in Electrical Engineering, NEUREL-2000***, Belgrad, 121-124, (2000).

9. Samman F. A., Syamsuddin E. Y., "Programmable Fuzzy Logic Controller Circuit on CPLD", ***Circuits and Systems, APCCAS '02***, Bali, 2:561-564, (2002).

10. Izad A, "Xilinx Circuit Gives Versatility and Size Reduction to Portable Electronic Radio Frequency Tag Reader", ***Euro ASIC'92***, Paris, 390-391, (1992).

11. Chen R. X., Chen M. J.,Chen L. G., Tsai T. H., "The System Implementation of I-Phone Hardware by Using Low Bit Rate Speech Coding", ***Signal Processing Systems, 1997. SIPS 97 - Design and Implementation., 1997 IEEE Workshop on,*** Leicester, 489-499, (1997).

12. El-Bardawil A., "Realization of a Versatile 8-VSB Exciter in a Single CPLD for Multi-Purpose Digital Television Applications", ***Proceedings of the 2000 Third IEEE International Caracas Conference,*** Caracas, 5-13, (2000).

13. Wang F., Zhang W., Yu S., "Implementation of HDTV PES Combiner Based on Horizantal Six-Block Segmentation", ***IEEE Transactions on Broadcasting***, 49(2): 217-220, (2003).

14. Parnell K, Mehta N., "Programmable Logic Design Quick Start Handbook". ***Xilinx***, USA, 1-190, (2003).

15. Cummings C. E., "Verilog Simulation of Xilinx Designs", ***Verilog HDL Conference,*** Kaliforniya, 93-100, (1994).

16. Chang M., **"**Teaching Top-down Design Using VHDL and CPLD", ***Frontiers in Education Conference,. FIE '96***, Utah, 2: 514-517, (1996).

17. Wunnava S. V., "Correlating Software Modelling and Hardware Responses for VHDL and Varilog Based Designs", ***Southeastcon '99. Proceedings. IEEE***, Lexington, 122-125, (1999).

18. Hofer F., Minaříková K., "VHDL Tools", ***CESNET Technical Report***, 1-11, (2002)..

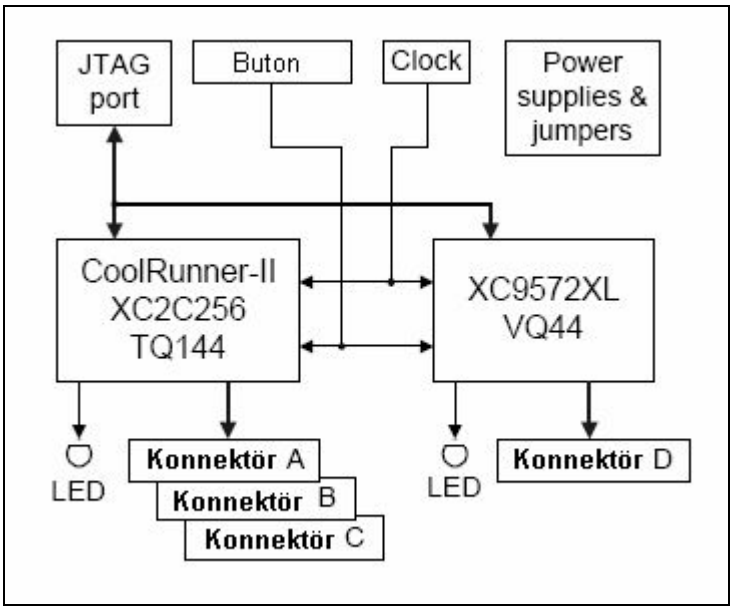
19. Areibi S., "A First Course in Digital Design Using VHDL and Programmable Logic", ***31st ASEE/IEEE Frontiers in Education Conference***, Reno 1-5, (2001).

20. Xilinx Application Notes, ***www.xilinx.com***, (2002).

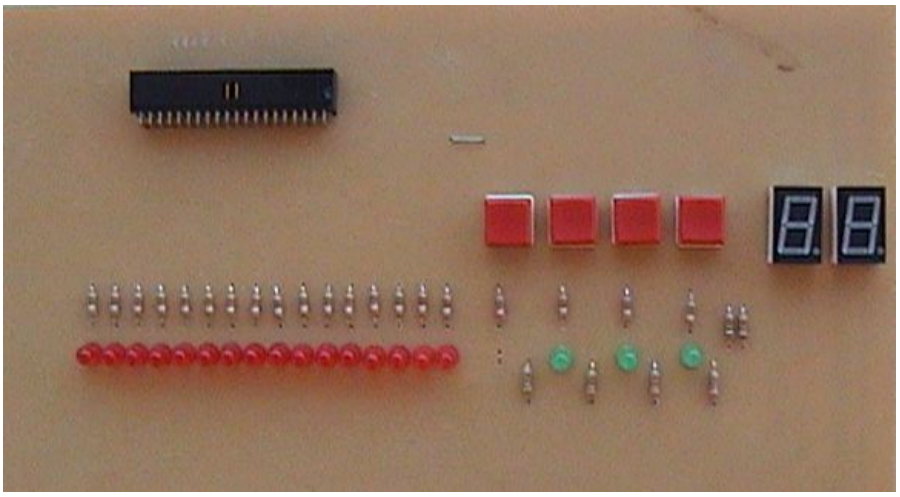
EKLER

EK-1 TASARIM DEVRESİ

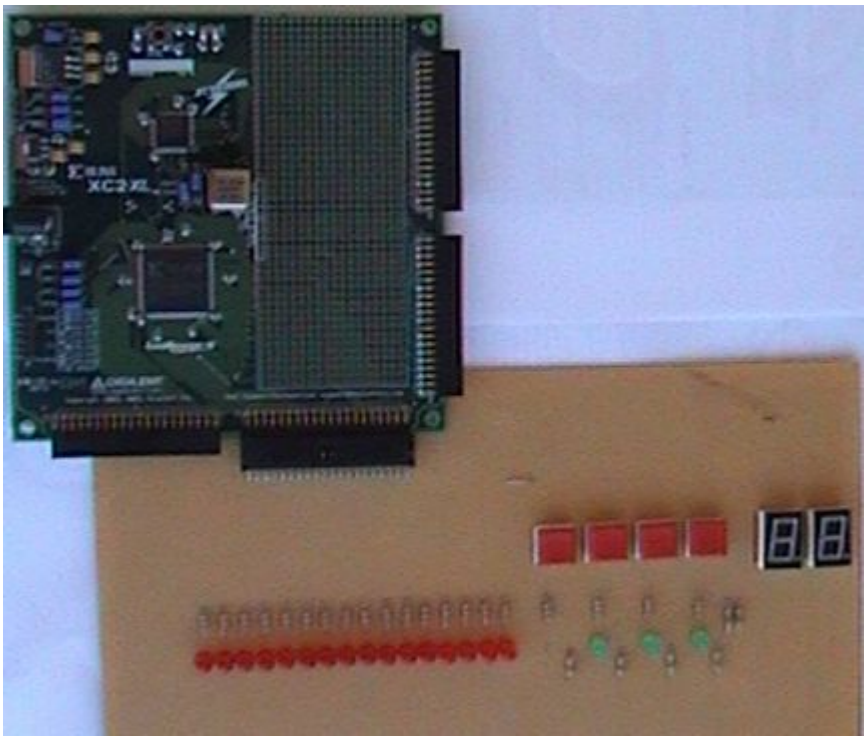
Şekil Ek.1.1. Xilinx CPLD Tasarım Devresi



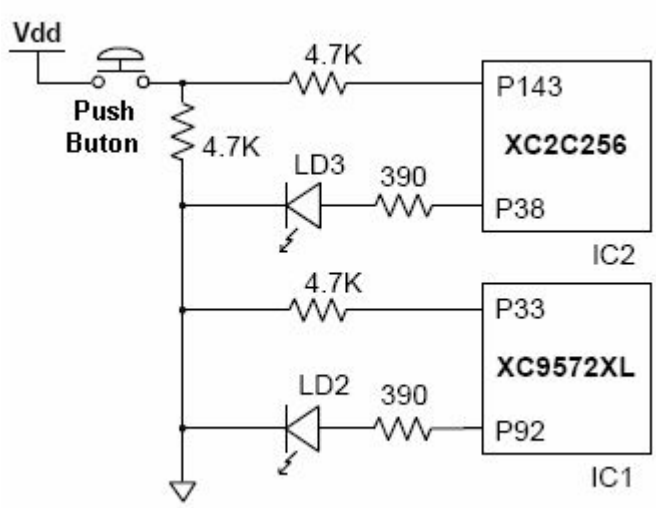
Şekil Ek.1.2. Tasarım Devresi Blok Diyagramı



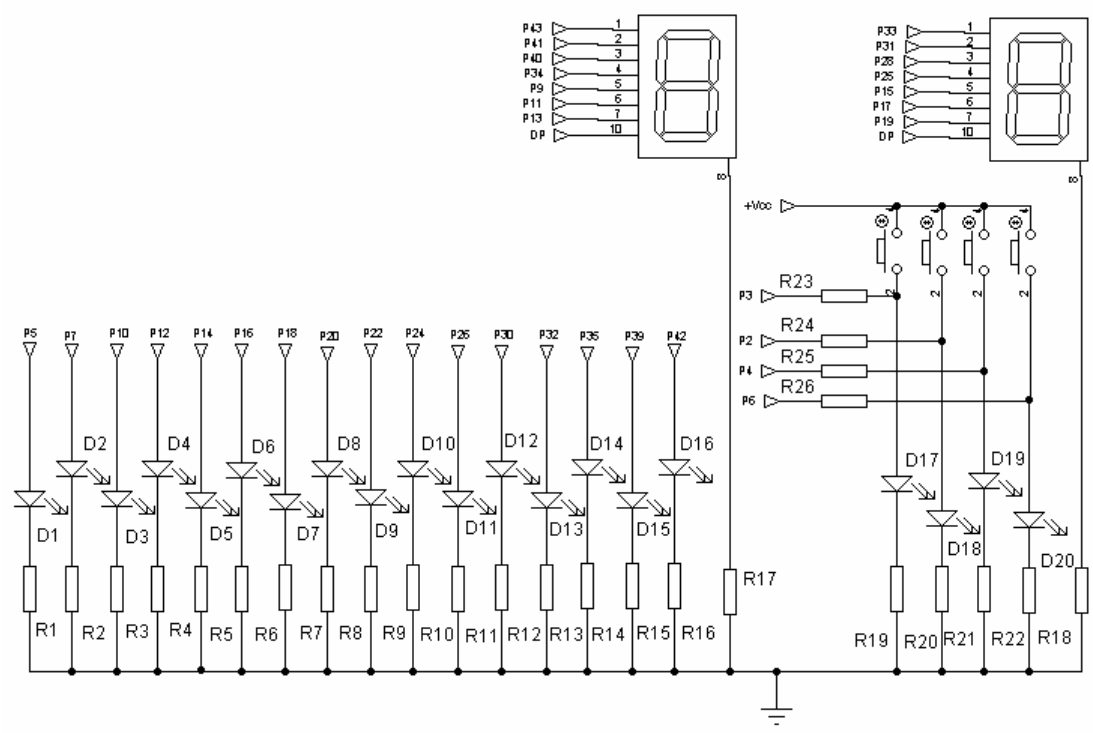
Şekil Ek.1.3. Uygulama Devresi



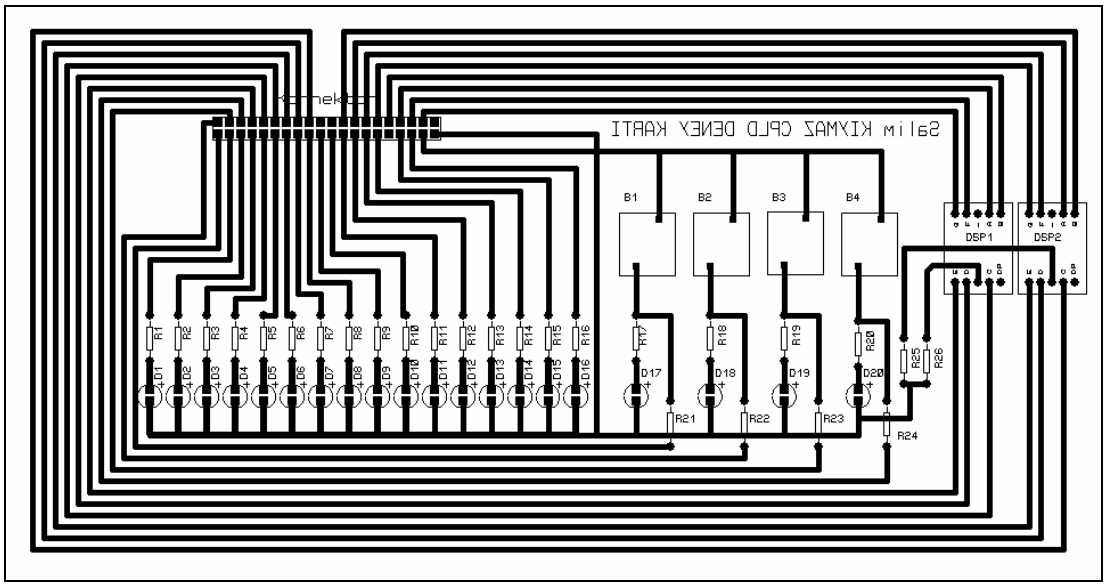
Şekil Ek.1.4. Tasarım devresine Uygulama devresinin eklenmesi



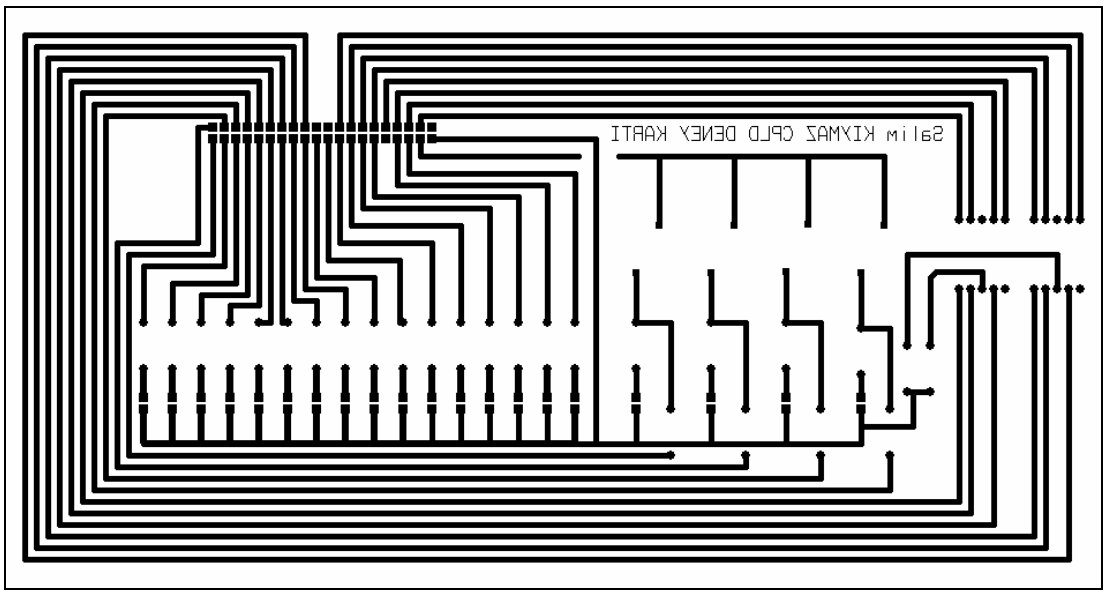
Şekil Ek.1.5. Uygulama Devresi Buton ve Led Bağlantısı



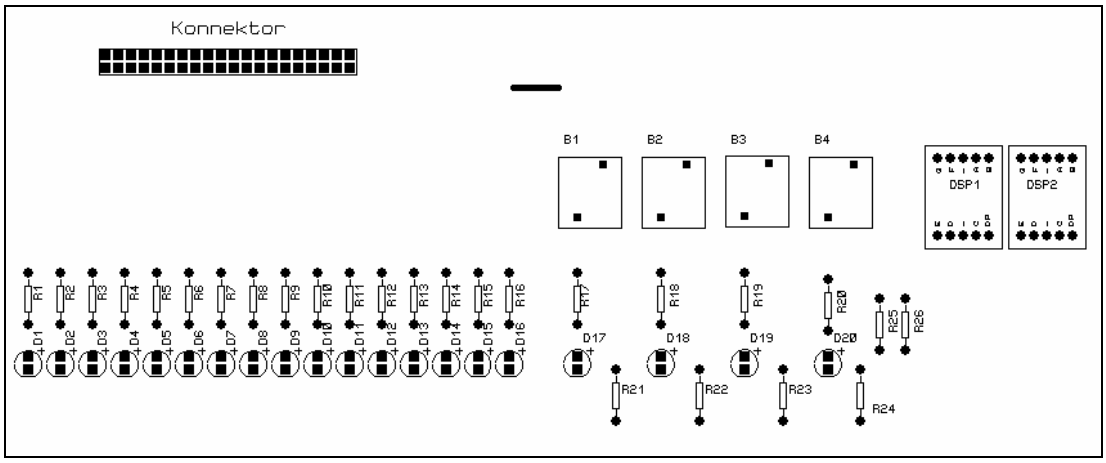
Şekil Ek.1.6. Uygulama Devresi Şematik Gösterilimi



Şekil Ek.1.7. Uygulama Devresi Baskı Devre Üstten Görünüş

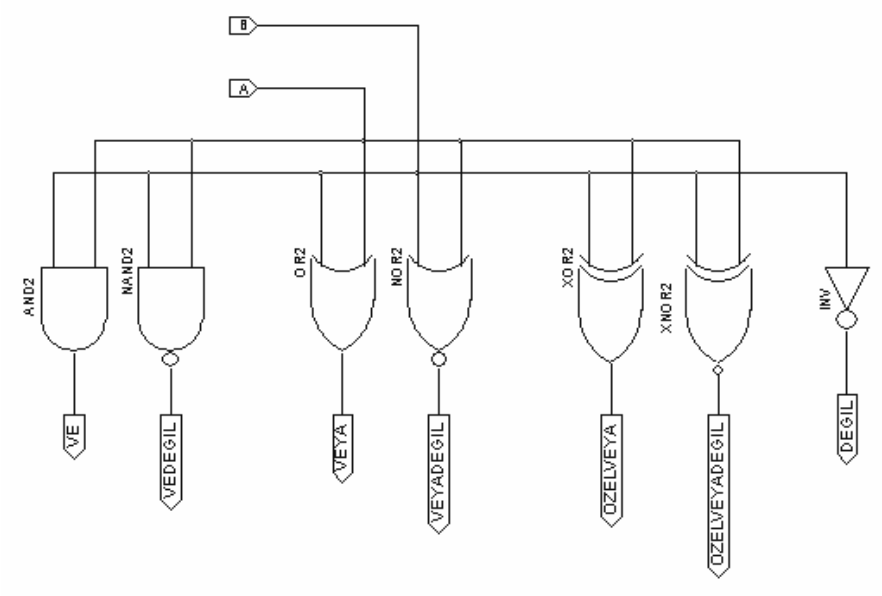


Şekil Ek.1.8. Uygulama Devresi Baskı Devre Görünüşü

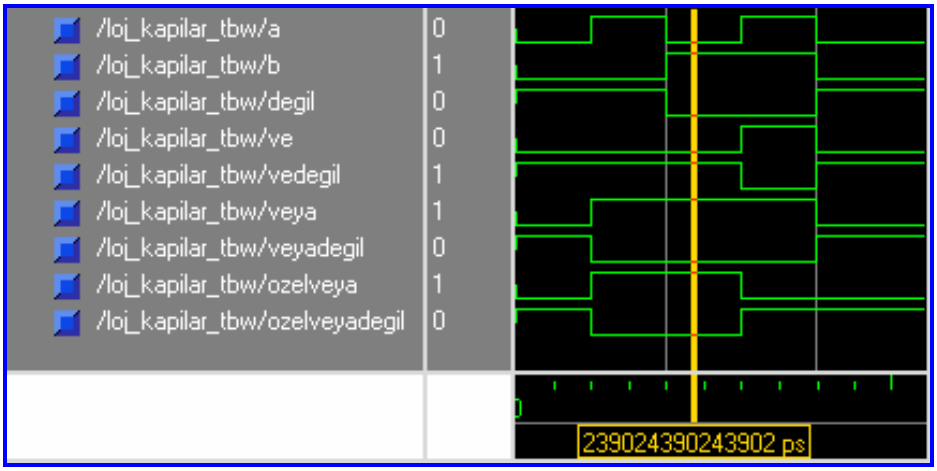


Şekil Ek.1.9. Uygulama Devresi Malzeme Yerleşimi

EK-2 MANTIKSAL KAPI DENEYİ



Şekil Ek.2.1. Mantık Kapıları Şematiği

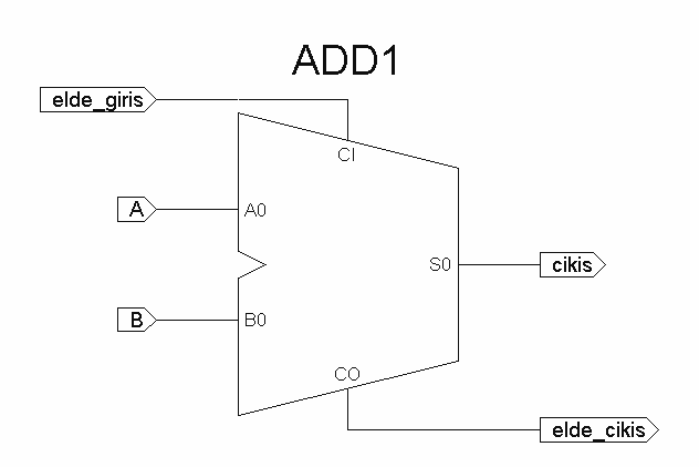


Şekil Ek.2.2. Mantık Kapıları Simülasyonu

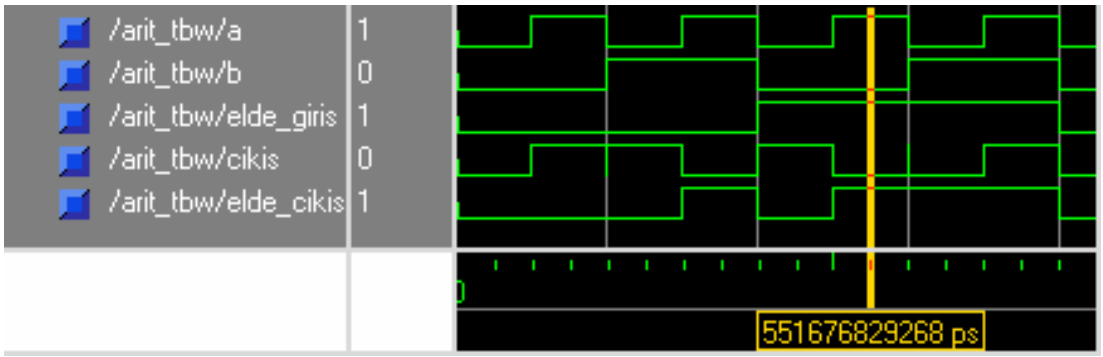
B	A	VE	VE DEĞİL	VEYA	VEYA DEĞİL	ÖZEL VEYA	ÖZELVEYA DEĞİL	DEĞİL (B`)
0	0	0	1	0	1	0	1	1
0	1	0	1	1	0	1	0	1
1	0	0	1	1	0	1	0	0
1	1	1	0	1	0	0	1	0

Şekil Ek.2.3. Mantık Kapıları Doğruluk Tablosu

EK-3 TAM TOPLAYICI DENEYİ



Şekil Ek.3.1. Tam Toplayıcı Şematiği

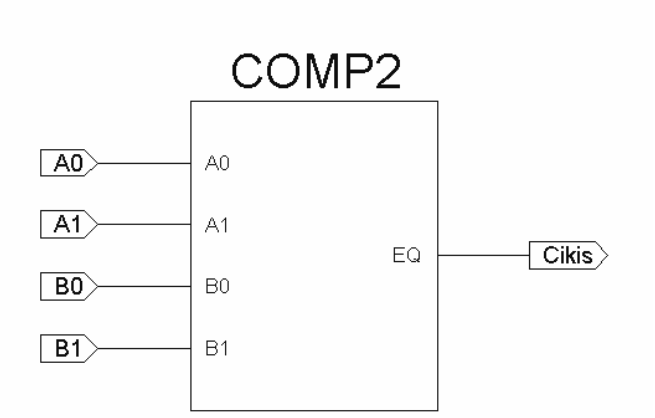


Şekil Ek.3.2. Tam Toplayıcı Simülasyonu

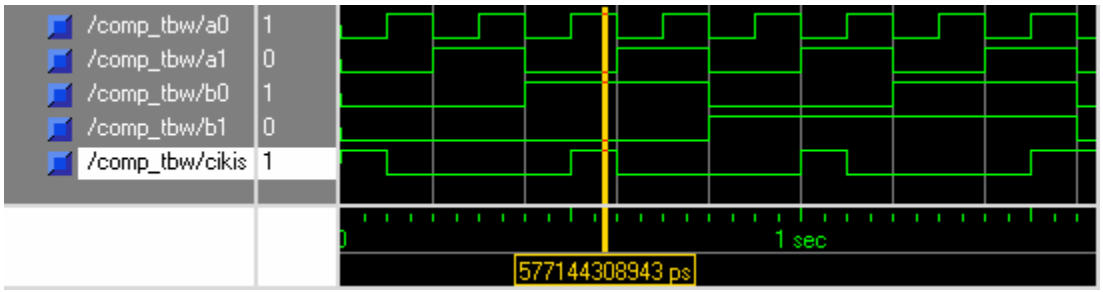
Girişler			Çıkışlar	
B	A	Elde Girişi	Çıkış	Elde Çıkışı
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Şekil Ek.3.3. Doğruluk Tablosu

EK-4 KARŞILAŞTIRICI DENEYİ



Şekil Ek.4.1. İki Bitlik Karşılaştırıcı Şematiği

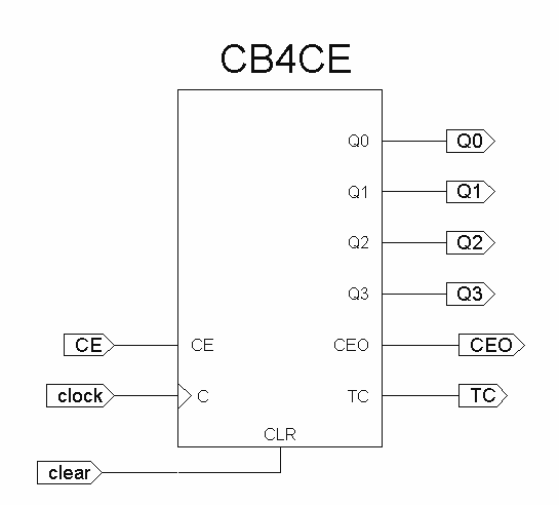


Şekil Ek.4.2. İki Bit Karşılaştırıcı Simülasyonu

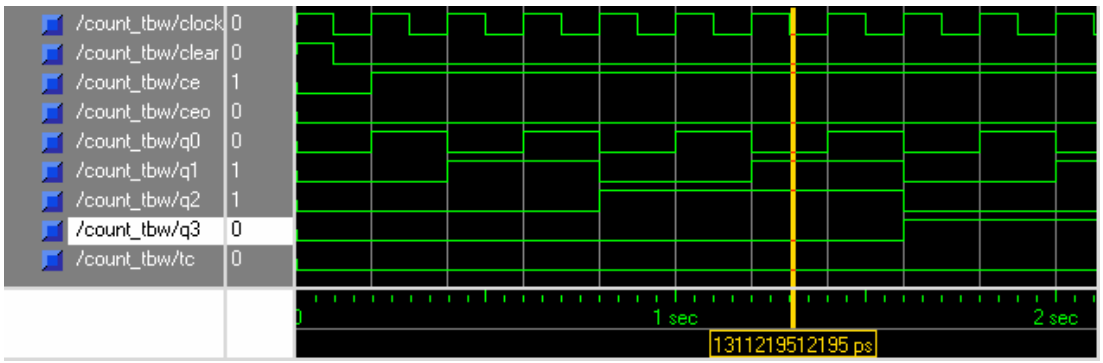
B1	B0	A1	A0	ÇIKIŞ
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Şekil Ek.4.3. Doğruluk Tablosu

EK-5 DÖRT BİT BİNARY SAYICI DENEYİ



Şekil Ek.5.1. Dört Bit Binary Sayıcı Şematiği

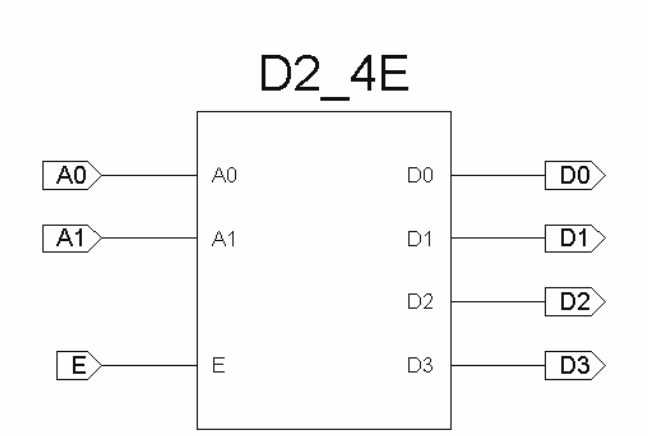


Şekil Ek.5.2. Dört Bit Binary Sayıcı Simülasyonu

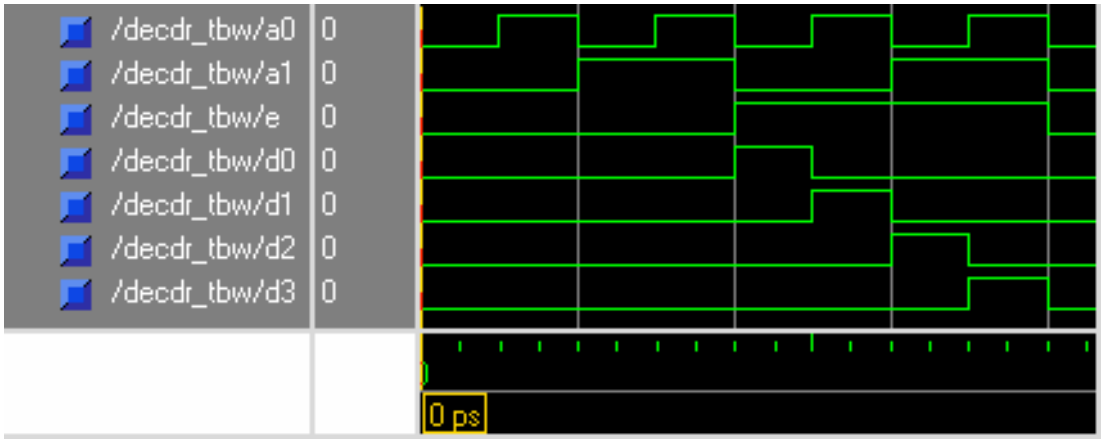
GİRİŞLER			ÇIKIŞLAR		
CLR	CE	C	Qz-Qo	TC	CEO
1	X	X	0	0	0
0	0	X	Değişmez	Değişmez	0

Şekil Ek.5.3. Doğruluk Tablosu

EK-6 2 GİRİŞ 4 ÇIKIŞLI KOD ÇÖZÜCÜ DENEYİ



Şekil Ek.6.1. 2 Girişli 4 Çıkışlı Kod Çözücü Şematiği

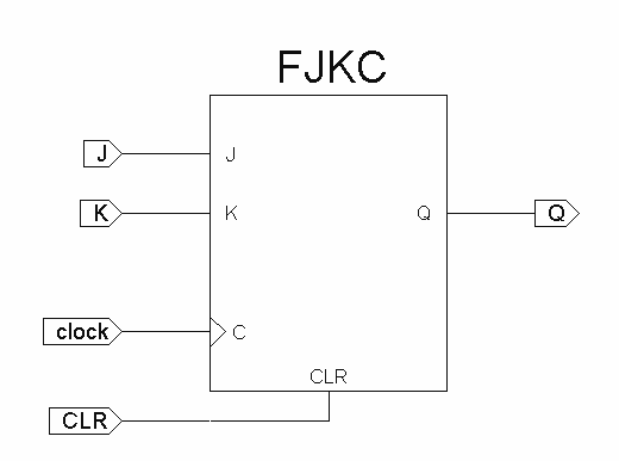


Şekil Ek.6.2. 2 Girişli 4 Çıkışlı Kod Çözücü Simülasyonu

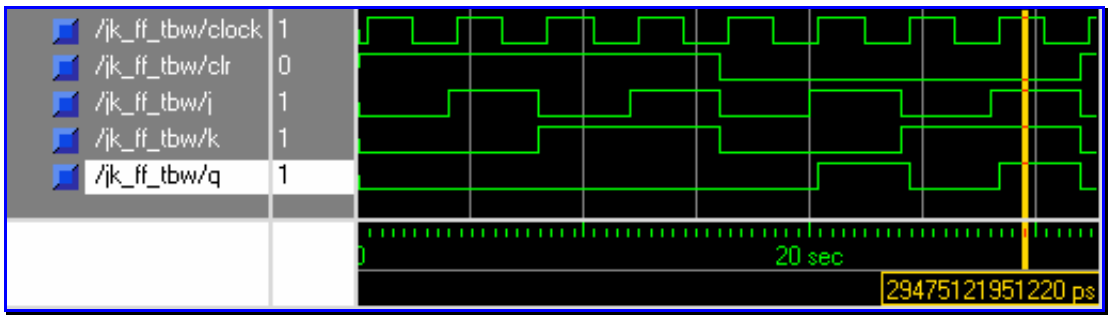
GİRİŞLER			ÇIKIŞLAR			
A1	A0	E	D3	D2	D1	D0
X	X	0	0	0	0	0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0

Şekil Ek.6.3. Doğruluk Tablosu

EK-7 J-K FLİP-FLOP DENEYİ



Şekil Ek.7.1. J-K F/F Şematiği

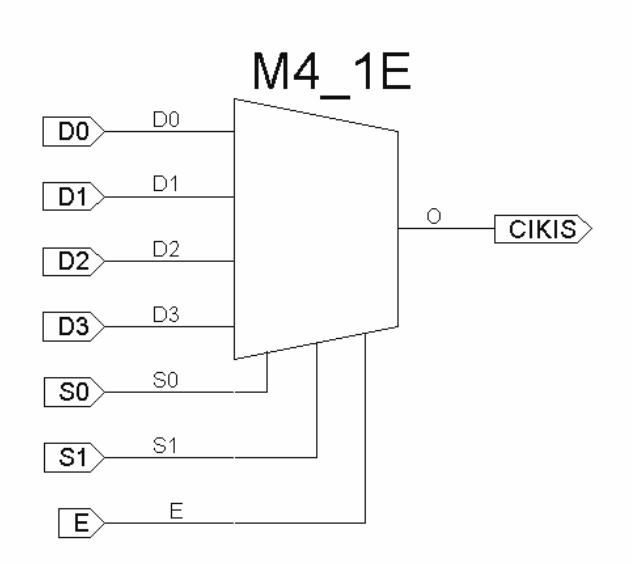


Şekil Ek.7.2. J-K F/F Simülasyonu

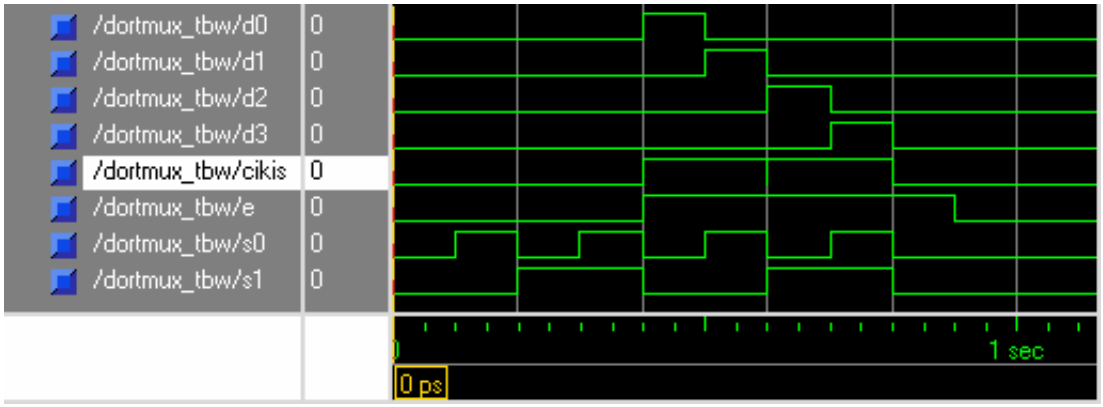
GİRİŞ				ÇIKIŞ
CLR	J	K	CLOCK	Q
1	X	X	X	0
0	0	0	↑	Değişmez
0	0	1	↑	0
0	1	0	↑	1
0	1	1	↑	Toggle

Şekil Ek.7.3. Doğruluk Tablosu

EK-8 4X1 MULTİPLEXER DENEYİ



Şekil Ek.8.1. 4X1 Multiplexer Şematiği

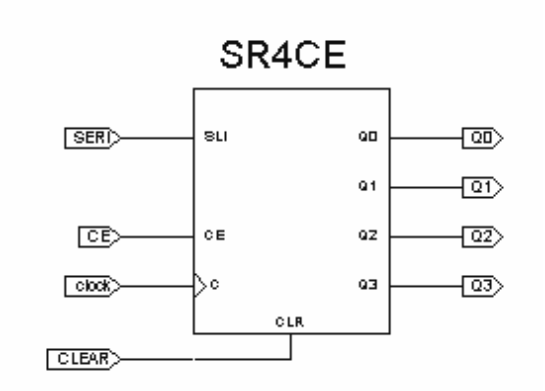


Şekil Ek.8.2. 4X1 Multiplexer Simülasyonu

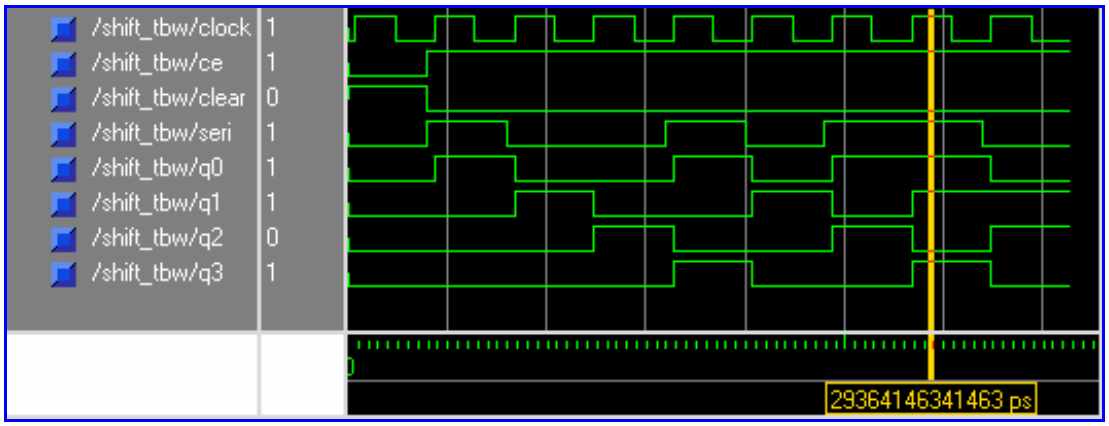
GİRİŞ							ÇIKIŞ
E	S1	S0	D3	D2	D1	D0	Q
0	X	X	X	X	X	X	0
1	0	0	X	X	X	D0	D0
1	0	1	X	X	D1	X	D1
1	1	0	X	D2	X	X	D2
1	1	1	D3	X	X	X	D3

Şekil Ek.8.3. Doğruluk Tablosu

EK-9 SERİ GİRİŞ 4 BİT ÇIKIŞ SHIFT REGISTER DENEYİ



Şekil Ek.9.1. Seri Giriş 4 Bit Çıkış Shift Register Şematiği



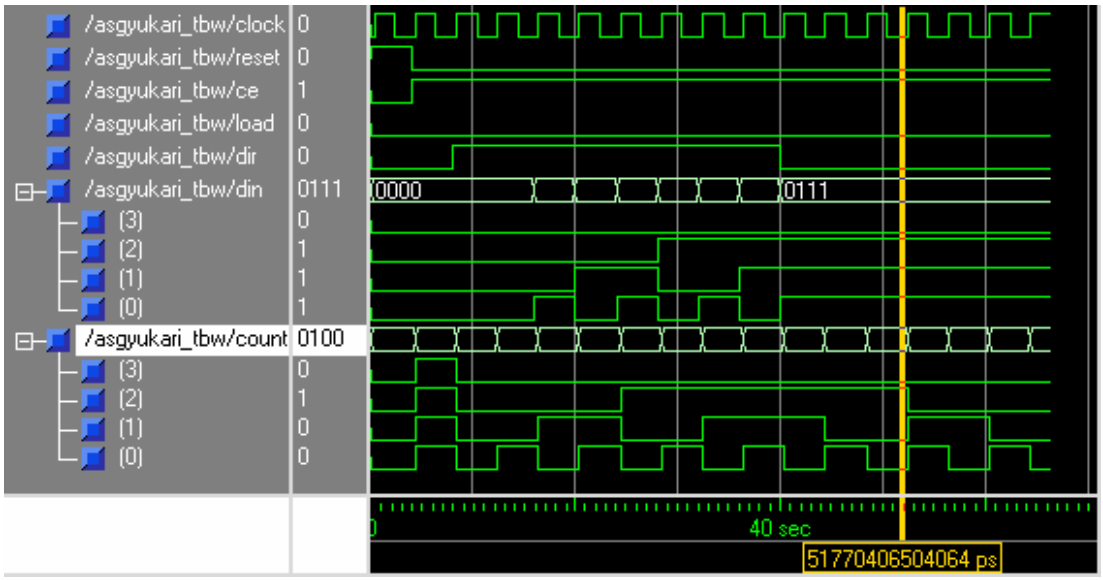
Şekil Ek.9.2. Seri Giriş 4 Bit Çıkış Shift Register Simülasyonu

GİRİŞ				ÇIKIŞ	
CLR	CE	SERİ	C	Qo	Qz–Q1
1	X	X	X	0	0
0	0	X	X	Değişmez	Değişmez
0	1	1	↑	1	Q_{n-1}

Şekil Ek.9.3. Doğruluk Tablosu

EK-10 VHDL KODLARIYLA 4 BİTLİK BİNARY AŞAĞI-YUKARI SAYICI

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity asagi_yukari is
    Port ( CLOCK : in std_logic;
           RESET : in std_logic;
           CE : in std_logic;
           LOAD : in std_logic;
           DIR : in std_logic;
           DIN : in std_logic_vector(3 downto 0);
           COUNT : inout std_logic_vector(3 downto 0));
end asagi_yukari;

architecture Behavioral of asagi_yukari is
begin
-- Required Libraries
--library IEEE;
--use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
 -- 4-bit synchronous counter with count enable,
-- asynchronous reset and synchronous load
--     CLK: in STD_LOGIC;
--     RESET: in STD_LOGIC;
--     CE, LOAD, DIR: in STD_LOGIC;
--     DIN: in STD_LOGIC_VECTOR(3 downto 0);
--     COUNT: inout STD_LOGIC_VECTOR(3 downto 0);

process (CLOCK, RESET)
begin
  if RESET='1' then
     COUNT <= "0000";
  elsif CLOCK='1' and CLOCK'event then
     if CE='1' then
        if LOAD='1' then
         COUNT <= DIN;
        else
          if DIR='1' then
             COUNT <= COUNT + 1;
          else
             COUNT <= COUNT - 1;
          end if;
        end if;
     end if;
  end if;
end process;

end Behavioral;
```

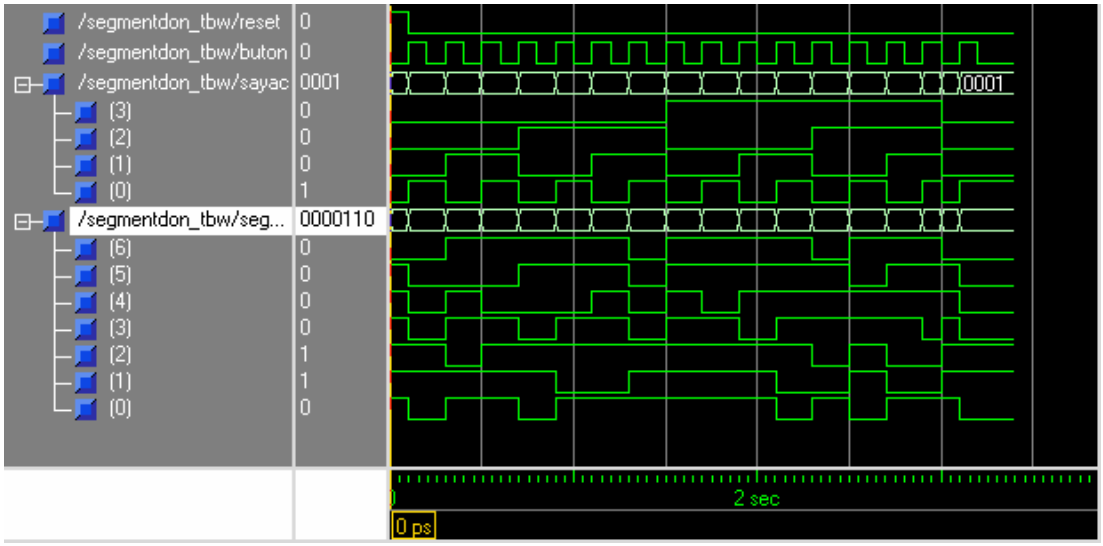


Şekil Ek.10.1. 4 Bitlik Binary Aşağı-Yukarı Sayıcı Simülasyonu

EK-11 HEX SAYICI

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity yedisegment is
    Port ( reset : in std_logic;
           buton : in std_logic;
           sayac : inout std_logic_vector(3 downto 0);
           segment : inout std_logic_vector(6 downto 0));
end yedisegment;

architecture Behavioral of yedisegment is

begin

process (buton, RESET)
begin
   if (RESET='1' or sayac= "1111") then
      sayac <= "0000";

   elsif buton='1' and buton'event then
              sayac <= sayac + 1;

   end if;
end process;

--HEX-to-seven-segment decoder
--   HEX:   in    STD_LOGIC_VECTOR (3 downto 0);
--   LED:   out   STD_LOGIC_VECTOR (6 downto 0);
--
-- segment encoding
--      0
--     ---
--  5 |   | 1
--     ---   <- 6
--  4 |   | 2
--     ---
--      3

    with sayac SELect
   segment <= "0000110" when "0001",   --1
         "1011011" when "0010",   --2
         "1001111" when "0011",   --3
         "1100110" when "0100",   --4
         "1101101" when "0101",   --5
         "1111101" when "0110",   --6
         "0000111" when "0111",   --7
```

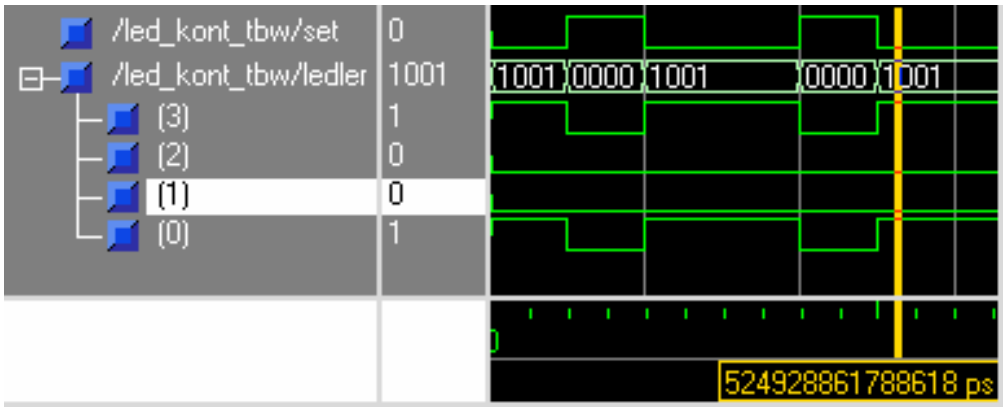
```
        "1111111" when "1000",   --8
        "1101111" when "1001",   --9
        "1110111" when "1010",   --A
        "1111100" when "1011",   --b
        "0111001" when "1100",   --C
        "1011110" when "1101",   --d
        "1111001" when "1110",   --E
        "1110001" when "1111",   --F
        "0111111" when others;   --0
end Behavioral;
```



Şekil Ek.11.1. Hex Sayıcı Simülasyonu

EK-12 BUTONLA LED KONTROL

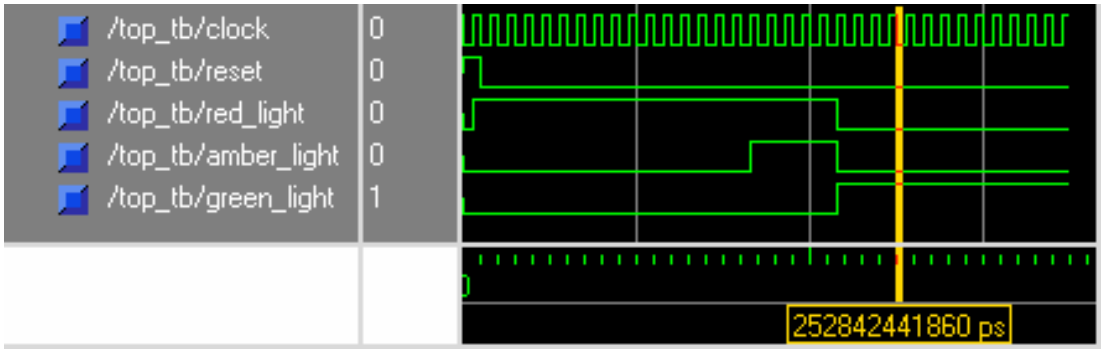
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ledkontrol is
    Port ( set : in std_logic;
           ledler : out std_logic_vector(3 downto 0));
end ledkontrol;

architecture Behavioral of ledkontrol is

begin

-- Required Libraries
--library IEEE;
--use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;

-- 4-bit synchronous counter with count enable,
-- asynchronous reset and synchronous load
--      CLK: in STD_LOGIC;
--      RESET: in STD_LOGIC;
--      CE, LOAD, DIR: in STD_LOGIC;
--      DIN: in STD_LOGIC_VECTOR(3 downto 0);
--      COUNT: inout STD_LOGIC_VECTOR(3 downto 0);

process (SET)

begin
   if set='1'  then
      ledler <= "0000";
  else
       ledler <= "1001";
   end if;

end process;

end Behavioral;
```



Şekil Ek.12.1. Led Kontrol Simülasyonu

EK-13 TRAFİK LAMBASI

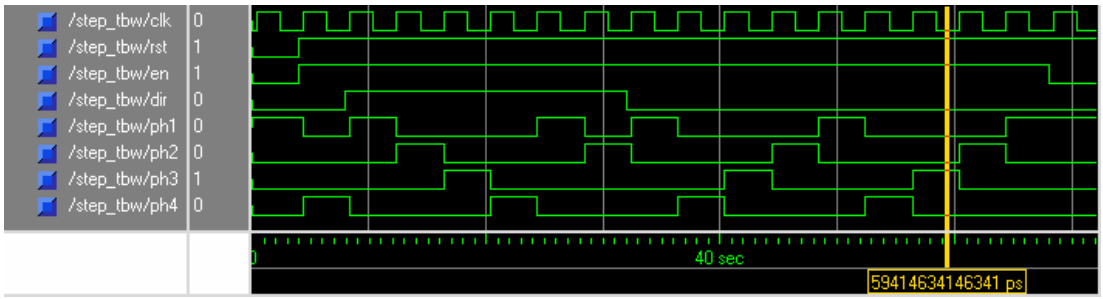
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top is
    Port ( clock : in std_logic;
           reset : in std_logic;
           red_light : out std_logic;
           amber_light : out std_logic;
           green_light : out std_logic);
end top;

architecture behavioral of top is
signal timer : std_logic_vector(3 downto 0);
      COMPONENT counter
      PORT(
            clock : IN std_logic;
            reset : IN std_logic;
            count : INOUT std_logic_vector(3 downto 0)
            );
      END COMPONENT;

      COMPONENT stat_mac
      PORT(
            TIMER : IN std_logic_vector(3 downto 0);
            CLK : IN std_logic;
            RESET : IN std_logic;
            amb : OUT std_logic;
            grn : OUT std_logic;
            rd : OUT std_logic
            );
      END COMPONENT;
begin
      Inst_counter: counter PORT MAP(
            clock => clock ,
            reset => reset,
            count => timer
      );
      Inst_stat_mac: stat_mac PORT MAP(
            TIMER => timer,
            CLK => clock,
            RESET => reset,
            amb => amber_light,
            grn => green_light,
            rd =>  red_light
      );
end behavioral;
```



Şekil Ek.13.1. Trafik Lambası Simülasyonu

EK-14 STEP MOTOR KONTROLÜ

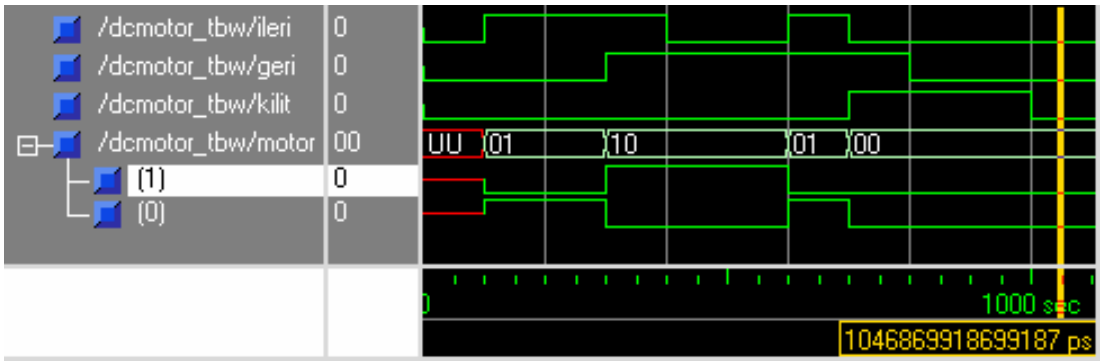
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity step is
    Port ( clk : in std_logic;
           rst : in std_logic;
           en : in std_logic;
           dir : in std_logic;
           ph1 : inout std_logic;
           ph2 : inout std_logic;
           ph3 : inout std_logic;
           ph4 : inout std_logic);
end step;

architecture Behavioral of step is

begin
  Process (rst,clk)
  begin
            if rst = '0' then
            ph1 <= '1';
            ph2 <= '0';
            ph3 <= '0';
            ph4 <= '0';
            else
            if clk'event and clk='1' then
--step motor kontrolu
            ph1 <= (not(dir)and en and not(ph1)and ph2 and not(ph3)and not (ph4))
            or (dir and en and not (ph1) and not (ph2)  and not (ph3) and ph4)
            or (not (en) and ph1);

            ph2 <= (not(dir)and en and not(ph1)and not(ph2) and ph3 and not (ph4))
            or (dir and en and ph1 and not (ph2)  and not (ph3) and not (ph4))
            or (not (en) and ph2);

            ph3 <= (not(dir)and en and not(ph1)and not(ph2) and not(ph3)and ph4)
            or (dir and en and not (ph1) and (ph2)  and not (ph3) and not (ph4))
            or (not (en) and ph3);

            ph4 <= (not(dir)and en and ph1 and not(ph2) and not(ph3)and not (ph4))
            or (dir and en and not (ph1) and not (ph2)  and (ph3) and not (ph4))
            or (not (en) and ph4);

            end if;
            end if;
            end process;
end Behavioral;
```



Şekil Ek.14.1. Step Motor Kontrolü Simülasyonu

EK-15 DC MOTOR KONTROLÜ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity dcmotor is
    Port ( ileri : in std_logic;
           geri : in std_logic;
           kilit : in std_logic;
           motor : inout std_logic_vector(1 downto 0));
end dcmotor;


architecture Behavioral of dcmotor is

begin

       process (ileri, geri, kilit)


begin

       if ileri='1' and ileri'event then
             if motor /= "00" then
                   motor <="00";
                   motor <="01";
             else
                   motor <="01";
             end if;
       end if;

       if geri='1' and geri'event then
             if motor /= "00" then
                   motor <="00";
                   motor <="10";
             else
                   motor <="10";
             end if;
       end if;

    if kilit='1' and kilit'event then
             motor <="00";
       end if;

end process;

end Behavioral;
```



Şekil Ek.15.1. DC Motor Kontrol Simülasyonu

ÖZGEÇMİŞ

1976 yılında Ankara'da doğdu. İlk, orta ve lise öğrenimini Ankara'da tamamladı. Lisans eğitimini 1994-1999 yılları arasında Gazi Üniversitesi Teknik Eğitim Fakültesi Elektronik Bilgisayar Eğitimi, Elektronik Öğretmenliği bölümünde tamamladı. 2000 yılında askerlik hizmetini tamamlayarak İskitler Teknik ve Endüstri Meslek Lisesi'ne Elektronik Öğretmeni olarak atandı. Halen İskitler Teknik ve Endüstri Meslek Lisesi'nde öğretmen olarak çalışmakta olup evli ve bir çocuk babasıdır.