T.C

FIRAT ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

SAHADA PROGRAMLANABİLİR KAPI DİZİLERİ KULLANILARAK DARBE GENİŞLİK MODÜLASYONLU SİNYALLERİN ÜRETİMİ VE DC MOTOR UYGULAMASI

Meral AKARÇAY TÜRK

Tez Yöneticisi

Yrd. Doç. Dr. Servet TUNCER

YÜKSEK LİSANS TEZİ

ELEKTRONİK-BİLGİSAYAR EĞİTİMİ ANABİLİM DALI

ELAZIĞ, 2009

TEŞEKKÜR

Tez çalışmaları boyunca desteğini ve bilgilerini esirgemeyen Danışman Hocam Yrd. Doç. Dr. Servet TUNCER 'e teşekkür ederim.

Meral AKARÇAY TÜRK

Elazığ - 2009

İÇİNDEKİLER

Sayfa No:

ŞEKİLLER LİSTESİ

Sayfa No:

TABLOLAR LİSTESİ

Sayfa No:

EKLER

EK- SPKD'ye yüklenen VHDL programı

KISALTMALAR

SPDK **:** Sahada Programlanabilir Kapı Dizisi

DGM **:** Darbe Genişlik Modülasyonu

LUT **:** Başvuru Çizelgesi

ASIC **:** Uygulamaya Özel Tüm Devre

SRAM **:** Statik Rasgele Erişimli Bellek

EEPROM **:** Elektriksel Olarak Silinebilir Salt Okunur Bellek

HDL **:** Donanım Tanımlama Dili

VHDL **:** Çok Yüksek Hızlı Tümleşik Devreler İçin Donanım Tanımlama Dili

ALU **:** Aritmetik Mantık Ünitesi

ÖZET

Yüksek Lisans Tezi

SAHADA PROGRAMLANABİLİR KAPI DİZİLERİ KULLANILARAK DARBE GENİŞLİK MODÜLASYONLU SİNYALLERİN ÜRETİMİ VE DC MOTOR UYGULAMASI

Meral AKARÇAY TÜRK

Fırat Üniversitesi

Fen Bilimleri Enstitüsü

Elektronik-Bilgisayar Eğitimi Anabilim Dalı

2009, Sayfa: 54

Bu tez çalışmasında, Xilinx üretici firmasına ait XC3S500E-4FG320 Sahada Programlanabilir Kapı Dizisi (SPKD) elemanı kullanılarak 1-16 kHz arasında, değişken frekans ve değişken görev periyotlu Darbe Genişlik Modülasyon (DGM) sinyalleri üretilmiştir. SPKD'nin programlanması için donanım tanımlama dili VHDL (Çok Yüksek Hızlı Tüm Devre Donanım Tanımlama Dili) kullanılmıştır. Üretilen DGM sinyalleri bir doğru akım motorunun sürücü devresine uygulanarak belirli hızlarda çalışması sağlanmıştır. Motor hız bilgisi motor miline bağlı bir disk ve optik algılayıcı devreler yardımıyla alınıp değerler bir sayısal ekranda (LCD) gösterilmiştir. Üretilen DGM sinyalleri, kullanılan Xilinx programlama ara birimi ile hem benzetimleri yapılmış hem de osiloskop ile gözlenerek doğrulukları sınanmıştır.

Anahtar Kelimeler: Sahada programlanabilir kapı dizisi, Darbe genişlik modülasyonu, VHDL.

ABSTRACT

Master Thesis

PRODUCING OF PULSE WIDTH MODULATED SIGNALS USING FIELD PROGRAMMABLE GATE ARRAYS AND DC MOTOR APPLICATION

Meral AKARÇAY TÜRK

Fırat University

Graduate School of Natural and Applied Sciences

Electronic-Computer Education

2009, Pages: 54

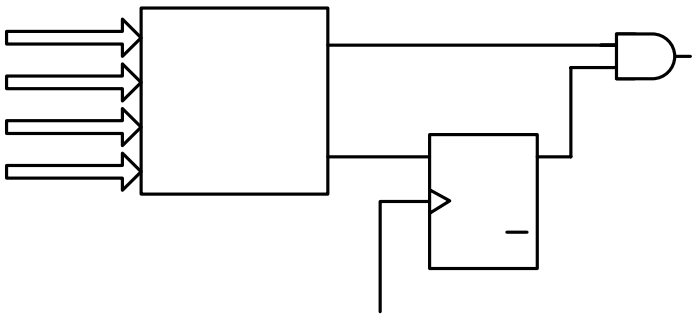
In this thesis, Pulse Width Modulation (PWM) signals of 1-16 kHz variable frequencies and variable duty cycles are produced by using Xilinx XC3S500E-4FG320 Field Programmable Gate Arrays (FPGA). VHDL (Very High Speed Integrated Circuit Hardware Description Language VhsicHDL), hardware description language, is used for programming FPGA. The produced PWM signals are applied to the drive circuit of a direct current motor and thus, the defined operation speeds are achieved. The motor speed is obtained by a disc attached motor shaft and an optic transducer, and is displayed on digital display (LCD). The produced PWM signals are both simulated using Xilinx programming interface unit and validated by observing the oscilloscope screen.

Keywords: Field programmable gate arrays, Pulse width modulation, VHDL.

1. GİRİŞ

Sahada Programlanabilir Kapı Dizileri (SPKD), gerekli fonksiyonların kullanılması ile mantık devrelerinin programlama yardımıyla gerçekleştirilmesini sağlayan sayısal devrelerdir [1-2]. Programlama ile SPKD'de matrissel şekilde bulunan mantık modülleri ve ara bağlantı elemanları istenen amaç doğrultusunda düzenlenebilmektedir. SPKD'ler Tasarımcının ihtiyaç duyduğu mantık işlevlerini gerçekleştirme amacına yönelik olarak üretilmişlerdir [3-7]. Dolayısıyla her bir mantık bloğunun işlevi kullanıcı tarafından düzenlenebilmektedir. SPKD ile temel mantık kapılarının ve yapısı daha karmaşık olan devre elemanlarının işlevselliği artırılmaktadır. Sahada programlanabilir ismi verilmesinin nedeni, mantık bloklarının ve ara bağlantıların imalat sürecinden sonra programlanabilmesidir.

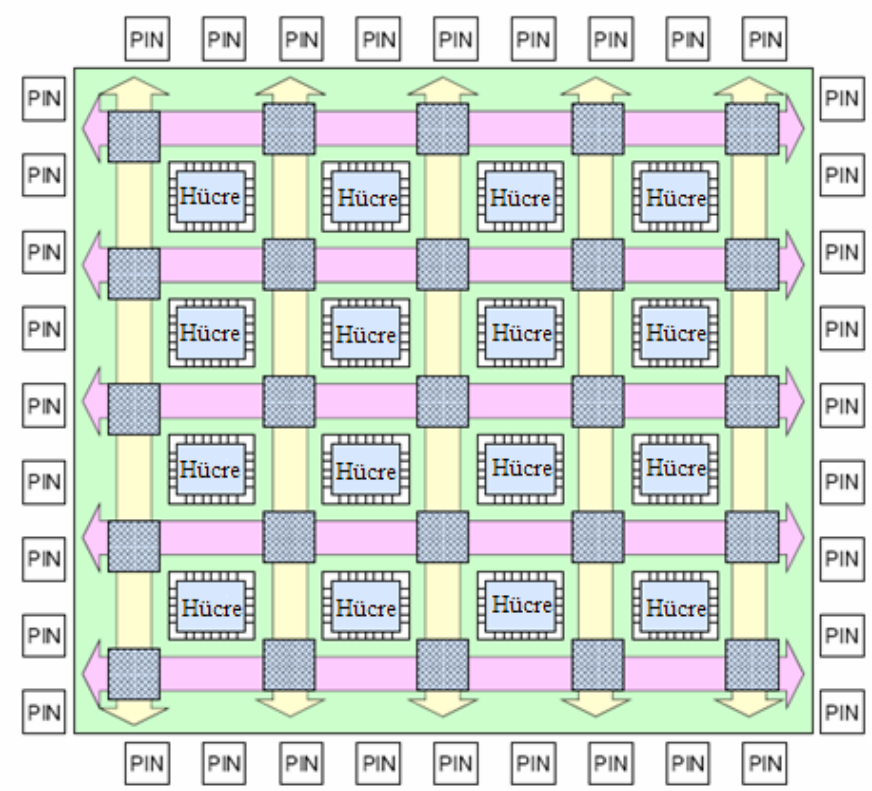
Son yıllarda, programlanabilir mantık devreleri özellikle SPKD'ler hızlı bir biçimde gelişme göstermişlerdir [8-12]. Düşük güç tüketimleri, esnek programlanabilirlikleri, kısa zaman dilimi içerisinde geliştirilebilir olmaları ve oldukça kolay nakledilebilir olmaları en büyük avantajları olarak sıralanabilir. Savunma, sayısal işaret işleme, uzay, tıbbi görüntüleme, haberleşme ve otomotiv gibi oldukça geniş ve yaygın bir kullanım alanı vardır [12-19].



Şekil 1.1. Bir SPKD mantık bloğu.

SPKD, programlanabilir mantık blokları, bu blok dizisini çevreleyen giriş-çıkış blokları ve ara bağlantılar olmak üzere üç ana bölümden oluşur [20]. Programlanabilir mantık blokları, ara bağlantılar içerisine gömülü şekilde bulunur. Programlanabilir mantık bloklarının yapılandırılması ve bu bloklar arasındaki iletişim ara bağlantılar sayesinde gerçekleşir. Giriş-çıkış blokları, ara bağlantılar ile bütünleşmiş devrenin paket bacakları arasındaki ilişkiyi sağlar. Klasik bir SPKD

mantık bloğu Şekil 1.1'de gösterilen dört girişli başvuru çizelgesi (LUT) ve bir flip-flop'tan oluşur. Son yıllardaki çalışmalar ile üreticiler performansın arttırılması için altı girişli başvuru çizelgeleri üretmeye başlamışlardır.



Şekil 1.2. SPKD'nin genel yapısı.

Mantık blokları basit mantık kapılarının (AND, OR) işlemlerini yerine getirmek için programlanabildikleri gibi daha karmaşık fonksiyonların da (kod çözücüler, basit matematiksel fonksiyonlar v.b.) işlevlerini yerine getirebilmektedirler. Her mantık hücresi kendi sınırları içinde belirlenen işleri tek başına yapabilme yeteneğine sahiptir. Bu hücreler bir birlerine programlanabilir anahtarlarla bir matris şeklinde bağlıdırlar. Bu mantık hücrelerinin yapısı farklı cihaz ailelerine göre değişiklikler gösterebilir. Genel olarak, kullanıcı programındaki Boolean mantık fonksiyonuna göre her bir mantık hücresi bir kaç ikili girişten (tipik olarak 3 veya 10) ve bir ya da iki çıkıştan oluşurlar. Bir çok ailede, saatli mantık devreleri rahatlıkla gerçekleştirilebilmesi için kullanıcı, hücrenin kombinezonal çıkışlarını kaydetme opsiyonuna sahiptir. Hücre kombinezonal mantık işlemi başvuru çizelgeleri oluşturularak ya da çoğullayıcı (multiplexer) ve kapı kümeleri ile fiziksel olarak rahatlıkla gerçekleştirilebilir. SPKD'ler genellikle benzerleri uygulamaya özel tüm devreler

(ASIC-Application-Specific Integrated Circuit) göre daha yavaştırlar. Ancak uygulamaya özel tüm devreler karmaşık tasarımların üstesinden gelemezler. SPKD, standart bileşenler olarak kabul edilebilecek ve bir birleriyle aynı özelliklere sahip Şekil 1.2'de gösterilen yapıda olan çok sayıda (64'den 10000'e kadar) mantık hücresi içeren bir tümleşik devredir [21-27].

Üretim teknikleri yönünden SPKD'ler birkaç önemli başlık altında incelenebilir. SRAM temelli olarak üretilen hücreler kullanan SPKD'larda tasarımın hızlı bir şekilde geliştirilebilir ve sınanabilir olması olumlu bir etken olurken sistemin her açılışında aygıtın yapılandırılma zorunluluğu ise olumsuz yan olarak göze çarpar. Antifuse temelli üretimlerde ise olumsuz olarak göz çarpan etken cihazın bir kez programlanır olmasıdır. Bu özelliğinden dolayı uygulama geliştirme için tercih edilmez. EEPROM temelli üretimler, silinebilme ve yeniden programlanabilme özelliğine sahiptir.

Üretici firmalar olarak önde gelen firmalar ise Xilinx, Altera, Actel, Atmel atomic, Lattice Semiconductor, QuickLogic Achronix Semiconductor ve MathStar olarak sıralanabilir. Xilinx ve Altera market liderleridir. Xilinx bedava Linux tasarım yazılımı dağıtmaktadır. Lattice Semiconductor SRAM ve uçucu olmayan, flash-tabanlı SPKD'ler sağlamaktadır. Actel antifuse, tekrar programlanabilir flash-tabanlı SPKD'ler ve karışık sinyal flash-tabanlı SPKD ürünlerine sahiptir. Atmel atomik(ince tanecikli) cihazlar sağlamaktadır, SPKD ile aynı ömre sahip Atmel AVR mikro denetleyici geliştirmek üzere odaklanmışlardır. QuickLogic, antifuse (programmable-only-once) ürünlerine sahip olup genel ilgi alanları askeri uygulamalardır. Achronix Semiconductor geliştirmede olan oldukça hızlı SPKD'leri vardır, hızları 2Ghz'e yaklaşmaktadır. MathStar, Sahada Programlanabilir Nesne Dizisi isimli SPKD benzeri bir cihaz sunmuştur [20].

SPKD'ler, sahada programlanabilirliliği sağlayabilmek için çok sayıda transistor kullandıklarından dolayı uygulamaya özel tüm devrelerden (ASIC) daha az güç verimliliğine sahiptirler. Bir başka değişle, eşdeğer bir tüm devre yapısına göre daha fazla güç harcarlar [24-25].

SPKD'nin davranışını tasarlamak için kullanıcı donanım tanımlama dili (HDL-Hardware Design Language) ya da şematik dizayn tekniklerinden birini kullanmalıdır. Yaygın HDL'ler VHDL ve Verilog'tur [1-2]. Bu tez çalışmasında da SPKD'ye yüklenen programlar için VHDL dili kullanılmıştır.

Tez çalışması genel olarak şu bölümlerden oluşmaktadır. İkinci bölümde, tercih edilen donanım tanımlama dili ile ilgili genel bilgiler verildi. Üçüncü bölümde, yazılım geliştirme ortamında gerçekleştirilmesi gereken adımlar verildi. Dördüncü bölümde, Xilinx üretici firmasına ait XC3S500E-4FG320 SPKD elemanı kullanılarak 1-16 kHz arasında Darbe Genişlik Modülasyon

(DGM) sinyalleri üretimi ve H-köprü kullanan bir dc motorun açık çevrim hız denetiminin gerçekleştirilmesi anlatılmıştır. Sonuç bölümünde ise elde edilen sonuçlar irdelenmiştir.

2. DONANIM TANIMLAMA DİLİ (VHDL)

VHDL, fiziksel bir sistemin ya da devrenin davranışlarını tanımlayan donanım tanımlama dilidir. VHDL, VHSIC HDL'in (Very High Speed Integrated Circuits Hardware Description Language- Çok Yüksek Hızlı Tümleşik Devreler İçin Donanım Tanımlama Dili) kısaltmasıdır. Bu bölümde, VHDL kütüphaneleri ve temel bildirimlerin nasıl yapılabileceği hakkında bilgiler yer almaktadır.

2.1. Temel VHDL Birimleri

VHDL kodları, LIBRARY, ENTITY ve ARCHITECTURE gibi en az üç tane temel kısımdan oluşur [1-4].

LIBRARY bildirimi, tasarımda kullanılacak olan kütüphanelerin bildiriminin yapıldığı kısımdır. Aşağıda görüldüğü gibi iki satır koddan oluşur.

```
LIBRARY kütüphane_adı;
USE kütüphane_adı.paket_adı.paket_bölümleri;
```

Üç tane farklı kütüphane ve paket mevcuttur. Bunlar, ieee.std_logic_1164 (ieee kütüphanesinden), standart (std kütüphanesinden) ve work (work kütüphanesinden). Bildirimleri ise aşağıdaki gibidir.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY std;
USE std.standard.all;

LIBRARY work;
USE work.all;
```

Üç farklı kütüphane/paket bildirimi kullanılmasının amacı şu şekilde açıklanabilir: ieee kütüphanesinin std_logic_1164 paketi, çok seviyeli lojik sistemleri tanımlar, std kütüphanesi veri tiplerini belirtir ve work kütüphanesi ise tasarımları, .vhd dosyalarını ve derleyicinin oluşturduğu dosyaların saklandığı kütüphanedir. Aslında ieee kütüphanesi birkaç paketten oluşur. Bunlar, std_logic_1164 (çok seviyeli lojik devreler), std_logic_arith (işaretli ve işaretsiz veri tipleri, matematiksel ve karşılaştırma işlemleri ve veri dönüşümleri), std_logic_signed ve std_logic_unsigned paketleri ise datanın sanki işaretli ya da işaretsizmiş gibi düşünülerek işlem gördüğü paketlerdir.

ENTITY, devreye ait giriş çıkış uçlarının belirlendiği kısımdır. Bildirimi aşağıdaki gibi yapılmaktadır.

```
ENTITY mevcut_isim IS
        PORT (
                port_adı : sinyal_modu sinyal tipi;
                port_adı : sinyal_modu sinyal tipi;
                …);
END mevcut  isim;
```

Sinyal modu, IN(giriş), OUT(çıkış), INOUT(Giriş-Çıkış) ya da BUFFER olarak tanımlanabilir. IN ve OUT tek yönlü, INOUT çift yönlü olup eğer devrenin çıkışı tekrar devreye giriş olacaksa çıkış BUFFER olarak adlandırılır. Sinyal tipi ise daha sonraki bölümlerde ayrıntılı olarak belirtileceği ama kısaca BIT, STD_LOGIC veya INTEGER olabilir. Mevcut isim kısmına ise VHDL tarafından kullanıma ayrılmamış bir isim seçilmelidir.

ARCHITECTURE, devrenin nasıl çalışması gerektiğine ait VHDL kodlarını içeren kısımdır. Genel yapısı aşağıda görüldüğü gibidir.

```
ARCHITECTURE mimari_adı OF mevcut_isim IS
        [Bildirimler]
BEGIN
        (kodlar);
END mimari  adı;
```

2.2. Veri Tipleri

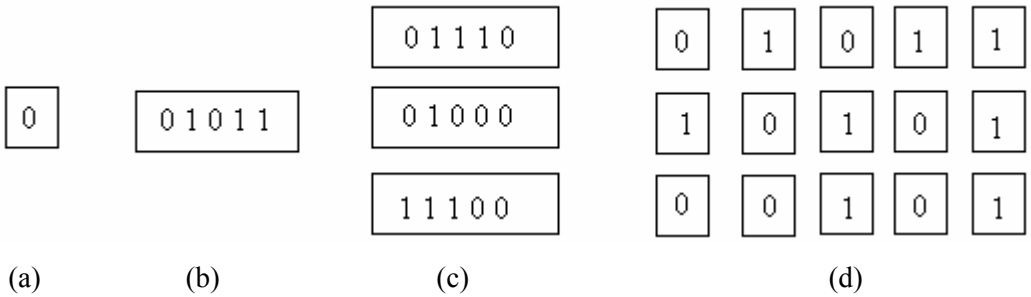
SIGNAL x: BIT; -- x, tek BIT tipinde tek rakamlık bir sinyali belirtir.
SIGNAL y: BIT_VECTOR (3 DOWNTO 0) --y, en anlamlı biti solda 4 bitlik bir vektörü temsil eder.
SIGNAL w: BIT_VECTOR (0 TO 7) -- w, en anlamlı biti en sağda olan 8 bitlik bir vektörü temsil eder.

Eğer sinyal bir atama yapılacaksa "<=" işareti ile yapılır. Örneğin;
x<='1';
y<="1010";
w<="10101010"; gibi.

BOOLEAN, doğru ya da yanlış.
INTEGER, 32-bitlik tam sayı (-2.147.483.647'den +2.147.483.647'ye kadar)
NATURAL, Pozitif tamsayılar (0'dan +2.147.483.647'ye kadar)
REAL, -1.10^{38}'den $+1.10^{38}$'e kadar olan reel sayılar.

2.2.1. Diziler

Diziler, tek boyutlu, iki boyutlu 1x1 boyutlu olabilir. Daha fazla boyut da tanımlanabilir fakat bu kodlar sentezlenemez. Dizi tipleri Şekil 2.1'de gösterildiği gibidir.



Şekil 2.1. Skalar sayı (a), Bir boyutlu dizi (b), 1x1 Boyutlu dizi (c) ve iki boyulu dizi (d).

Herhangi bir değişkenin nasıl dizi şeklinde tanımlanabileceği aşağıda gösterildiği gibidir.

```
TYPE tip_adı IS ARRAY (tanımlamalar) OF veri_tipi;
```

Aşağıdaki örneklerde bazı dizi tanımlamaları verilmiştir.

```
TYPE satir IS ARRAY (7 DOWNTO 0) OF STD_LOGIC; -- Bir boyutlu dizi
TYPE matris IS ARRAY (0 TO 3) OF satir; --1x1 Boyutlu dizi
SIGNAL x: matris; --1x1 Boyutlu bir sinyal
```

İkinci örnekteki tanımlama daha değişik bir şekilde de gösterilebilir.

```
TYPE matris IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
TYPE matris2D IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC; --İki boyutlu dizi
```

Bir örnek üzerinde diziler ile ilgili geçerli ve geçersiz bildirimler aşağıdaki gibi verilmiştir.

```
TYPE satir IS ARRAY (7 DOWNTO 0) OF STD_LOGIC; -- Bir boyutlu dizi
TYPE dizi1 IS ARRAY (0 TO 3) OF satir; -- 1x1 boyutlu dizi
TYPE dizi2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
-- 1x1 boyutlu dizi
TYPE dizi3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC; -- iki boyutlu dizi
SIGNAL x: satir;
SIGNAL y: dizi1;
SIGNAL v: dizi2;
SIGNAL w: dizi3;
--------- geçerli bildirimler: ----------
x(0) <= y(1)(2); -- (y, 1x1 boyutlu)
x(1) <= v(2)(3); -- (v, 1x1 boyutlu)
x(2) <= w(2,1); -- (w iki boyutlu)
y(1)(1) <= x(6);
y(2)(0) <= v(0)(0);
y(0)(0) <= w(3,3);
w(1,1) <= x(7);
w(3,0) <= v(0)(3);
y(1)(7 DOWNTO 3) <= x(4 DOWNTO 0); --  aynı tip, aynı boyut
v(1)(7 DOWNTO 3) <= v(2)(4 DOWNTO 0); -- aynı tip, aynı boyut
--------- geçersiz bildirimler: ----------
x <= v(1); -- tip uyumsuzluğu, v 1x1 boyutlu bir dizi
x <= w(2); -- w indeksi iki boyutlu olmalı
x <= w(2, 2 DOWNTO 0); -- tip uyumsuzluğu
v(0) <= w(2, 2 DOWNTO 0); -- tip uyumsuzluğu
```

v(0) <= w(2); -- w indeksi iki boyutlu olmalı

y(1) <= v(3); -- tip uyumsuzluğu

w(1, 5 DOWNTO 1) <= v(2)(4 DOWNTO 0); -- tip uyumsuzluğu

2.2.2. Veri Dönüşümleri

VHDL, farklı tipteki verilerin doğrudan işleme tabi tutulmasına izin vermez. Bu nedenle, veri tipini iki farklı veriden herhangi birisine dönüştürmek gereklidir. Aşağıdaki örnekte geçerli ve geçersiz veri dönüşümleri bir örnekle anlatılmaya çalışılmıştır.

TYPE long IS INTEGER RANGE -100 TO 100;

TYPE short IS INTEGER RANGE -10 TO 10;

SIGNAL x : short;

SIGNAL y : long;

y <= 2*x + 5; -- farklı tipteki y değişkenine x atanmak istendiği için hatlı bir söz dizimidir.

y <= long(2*x + 5); -- bu şekilde bir dönüşüm ile yapılırsa doğru olur.

Aşağıda *ieee* kütüphanesinin *std_logic_arith* paketinde bulunan dönüşüm fonksiyonları görülmektedir.

conv_integer(p) : INTEGER, UNSIGNED, SIGNED ya da STD_ULOGIC tipindeki değişkeni INTEGER tipine dönüştürür.(STD_LOGIC_VECTOR değişkenini içermez.)

conv_unsigned(p, b): b bitlik boyuttaki INTEGER, UNSIGNED, SIGNED ya da STD_ULOGIC tipindeki p değişkenini UNSIGNED tip değişkene çevirir.

conv_signed(p, b): b bitlik boyuttaki INTEGER, UNSIGNED, SIGNED ya da STD_ULOGIC tipindeki p değişkenini SIGNED tip değişkene çevirir.

conv_std_logic_vector(p, b): b bitlik boyuttaki INTEGER, UNSIGNED, SIGNED ya da STD_ULOGIC tipindeki p değişkenini STD_LOGIC_VECTOR tip değişkene çevirir.

Buraya kadar anlatılan değişkenler ile ilgili geçerli ve geçersiz bildirimler aşağıdaki örnekte genel olarak verilmiştir.

TYPE byte IS ARRAY (7 DOWNTO 0) OF STD_LOGIC; -- Bir boyutlu dizi

TYPE mem1 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC; -- iki boyutlu dizi

TYPE mem2 IS ARRAY (0 TO 3) OF byte; -- 1x1 boyutlu dizi

TYPE mem3 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(0 TO 7); -- -- 1x1 boyutlu dizi

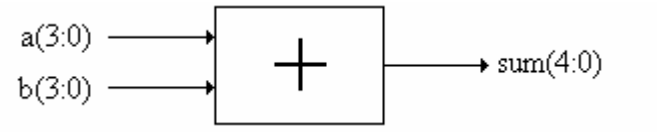
SIGNAL a: STD_LOGIC; -- skalar sinyal

SIGNAL b: BIT; -- skalar sinyal

```
SIGNAL x: byte; -- Bir boyutlu sinyal
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0); -- Bir boyutlu sinyal
SIGNAL v: BIT_VECTOR (3 DOWNTO 0); -- Bir boyutlu sinyal
SIGNAL z: STD_LOGIC_VECTOR (x'HIGH DOWNTO 0); -- Bir boyutlu sinyal
SIGNAL w1: mem1; -- iki boyutlu sinyal
SIGNAL w2: mem2; -- 1x1 boyutlu sinyal
SIGNAL w3: mem3; -- 1x1 sinyal
--------Geçerli skalar bildirimler ----------------------
x(2) <= a; -- aynı tip, doğru indeksleme
y(0) <= x(0); -- aynı tip, doğru indeksleme
z(7) <= x(5); -- aynı tip, doğru indeksleme
b <= v(3); -- aynı tip, doğru indeksleme
w1(0,0) <= x(3); -- aynı tip, doğru indeksleme
w1(2,5) <= y(7); -- aynı tip, doğru indeksleme
w2(0)(0) <= x(2); -- aynı tip, doğru indeksleme
w2(2)(5) <= y(7); -- aynı tip, doğru indeksleme
w1(2,5) <= w2(3)(7); -- aynı tip, doğru indeksleme
------- Geçersiz skalar bildirimler: ----------------------
b <= a; -- Tip uyumsuzluğu (BIT x STD_LOGIC)
w1(0)(2) <= x(2); -- w1'in indeksi iki boyutlu olmalı
w2(2,0) <= a; -- w2'nin indeksi 1x1 boyutlu olmalı
------- Geçerli vektör bildirimleri ----------------------
x <= "11111110";
y <= ('1','1','1','1','1','1','1','0','Z');
z <= "11111" & "000";
x <= (OTHERS => '1');
y <= (7 =>'0', 1 =>'0', OTHERS => '1');
z <= y;
y(2 DOWNTO 0) <= z(6 DOWNTO 4);
w2(0)(7 DOWNTO 0) <= "11110000";
w3(2) <= y;
z <= w3(1);
z(5 DOWNTO 0) <= w3(1)(2 TO 7);
```

```
w3(1) <= "00000000";
w3(1) <= (OTHERS => '0');
w2 <= ((OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'));

------Geçersiz dizi bildirimleri: ----------------------
x <= y; -- Tip uyumsuzluğu
y(5 TO 7) <= z(6 DOWNTO 0); -- yanlış y yönü
w1 <= (OTHERS => '1'); -- w1 iki boyutlu bir dizidir
w1(0, 7 DOWNTO 0) <="11111111"; -- w1 iki boyutlu bir dizidir
w2 <= (OTHERS => 'Z'); -- w2 1x1 boyutlu bir dizi
w2(0, 7 DOWNTO 0) <= "11110000"; -- indeks 1x1 boyutlu olmalı
```

Şekil 2.2'de gösterilen 4-bitlik bir toplayıcının VHDL kullanılarak tasarımı aşağıdaki örnekte verilmiştir. Girişler a ve b olmak üzere 3-bitliktir. Çıkış, sum ile gösterilmiştir ve 4-bitlik olması gerekmektedir.



Şekil 2.2. 4-bitlik toplayıcı.

```
1 ----- Çözüm----------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE ieee.std_logic_arith.all;
5 ----------------------------------------
6 ENTITY adder IS
7 PORT ( a, b : IN SIGNED (3 DOWNTO 0);
8 sum : OUT SIGNED (4 DOWNTO 0));
9 END adder;
10 ----------------------------------------
11 ARCHITECTURE adder1 OF adder IS
12 BEGIN
```

```
13 sum <= a + b;
14 END adder1;
15 ----------------------------------------
```

Görüldüğü üzere birkaç satırda 4-bitlik bir toplayıcının gerçekleştirmesi yapılmıştır. Eğer giriş bit sayısı arttırılmak istendiğinde kodda sadece giriş çıkış portları için ayrılmış olan dizi boyutları değiştirilecektir.

2.3. Operatörler ve Özellikleri

VHDL'de atama operatörleri, lojik operatörler, aritmetik operatörler, karşılaştırma operatörleri ve kaydırma operatörleri ve olmak üzere beş tip operatör kullanılmaktadır.

Atama operatörleri, sinyallere, değişkenlere ve sabitlere değer atamak için kullanılırlar. "<=" operatörü sinyallere, ":=" operatörü ise değişken ve sabitlere başlangıç değeri atamak için kullanılır. ">=" operatörü ise vektör elemanlarına değer atamak için kullanılır. Aşağıdaki örnekte bu operatörler kullanılarak yapılan atamalar gösterilmiştir.

```
SIGNAL x : STD_LOGIC;
VARIABLE y : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL w: STD_LOGIC_VECTOR(0 TO 7);

x <= '1';
y := "0000";
w <= "10000000";
w <= (0 =>'1', OTHERS =>'0'); -- LSB '1', diğerleri '0'
```

Lojik operatörler, lojik işlemlerin gerçekleştirilmesinde kullanılırlar. Veri tipi, BIT, STD_LOGIC, STD_ULOGIC, BIT_VECTOR, STD_LOGIC_VECTOR ya da STD_ULOGIC_VECTOR olmalıdır. Kullanılan operatörler ise NOT, AND, OR, NAND, NOR, XOR ve XNOR'dir. Kullanımları ise aşağıdaki örneklerde görüldüğü gibidir.

```
y <= NOT a AND b; -- (a'.b)
y <= NOT (a AND b); -- (a.b)'
y <= a NAND b; -- (a.b)'
```

Aritmetik oeratörler, matematiksel işlemlerin yapılmasında kullanılırlar. Veri tipi INTEGER, SIGNED, UNSIGNED ya da REAL (doğrudan sentezlenemez) olmalıdır.

+ Toplama

- Çıkarma

/ Bölme

** Üstel

MOD Mod işlemi

REM Kalan

ABS Mutlak değer

Toplama ve çıkarma işleminde her hangi bir sınırlama yoktur. Bölmede ise sadece ikini kuvveti şeklinde (sağa ya da sola öteleme) bir bölme işlemi gerçekleştirilir. y MOD x işleminde x sinyali ile y/x bölümden kalan, y REM x ise y sinyali ile y/x bölümünden kalan geri döndürülür. ABS, mutlak değer alma operatörüdür.

Karşılaştırma operatörleri, karşılaştırma işlemleri için kullanılır. Aşağıda gösterilen operatörlerdir.

= Eşittir

/= Eşit değildir

< Küçüktür

> Büyüktür

<= Küçük eşittir

>= Büyük eşittir

Öteleme operatörleri, verilerin sağa ya da sola ötelenmesinde kullanılır. SLL, veriyi sola doğru ötelerken sağdan sürekli olarak veriye ‘0’ ekler. SRL, veriyi sağa doğru ötelerken veriye soldan sürekli olarak ‘0’ ekler.

Veri operatörleri ve özelliklerine bakılacak olursa aşağıdaki komutlar verilerin bazı özellikleri hakkında fikir verebilir.

d’LOW: Dizinin en küçük indeksini verir

d’HIGH: Dizinin en büyük indeksini verir

d’LEFT: En soldaki indeksi verir

d’RIGHT: En soldaki indeksi verir

d’LENGTH: Vektörün boyutunu verir

d’RANGE: Vektörün aralığını verir

d’REVERSE_RANGE: Vektörün aralığını tersten verir

Aşağıdaki örnekte komutların nasıl çalışacağı gösterilmiştir.

```
SIGNAL d : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

O halde

d'LOW=0, d'HIGH=7, d'LEFT=7, d'RIGHT=0, d'LENGTH=8,

d'RANGE=(7 downto 0), d'REVERSE_RANGE=(0 to 7).

Eğer sinyal sıralı tip bir sinyal ise aşağıdaki komutlar ile sinyal belirli pozisyonlarında ve değerlerinde gezilebilir.

d'VAL(pos): Belirli bir pozisyondaki değeri verir

d'POS(deger): Belirli bir değerin pozisyonunu verir

d'LEFTOF(deger): Belirli bir değerin solundaki pozisyonu verir

d'VAL(satir, sutun): Belirli bir pozisyondaki değeri verir

Sinyallerin ve özellikleri de aşağıdaki komutlar ile elde edilebilir.

s'EVENT: s üzerinde bir olay gerçekleştiğinde doğru değeri geri döndürür

s'STABLE: s üzerinde bir olay gerçekleşmediğinde doğru değeri geri döndürür

s'ACTIVE: Eğer s='1' ise doğru değeri döndürür

s'QUIET <zaman>: Belirlenen zamanda bir olay olmamışsa doğru değeri döndürür

s'LAST_EVENT: Son olaydan sonra geçen zamanı gösterir

s'LAST_ACTIVE: s='1' olduğundan beri geçen zamanı gösterir

s'LAST_VALUE: Son olaydan önceki s'in değerini dönderir

Özellikle s'EVENT oldukça fazla kullanılan bir komuttur. Aşağıda bir örnek ile komutun kullanımı gösterilmiştir.

```
IF (clk'EVENT AND clk='1')... -- Eğer saat işaretinde bir değişim olmuş ve saat 1 ise
IF (NOT clk'STABLE AND clk='1')...
WAIT UNTIL (clk'EVENT AND clk='1'); -- Saat işaretinde bir değişim olacak ve s=1 oluncaya kadar bekle
```

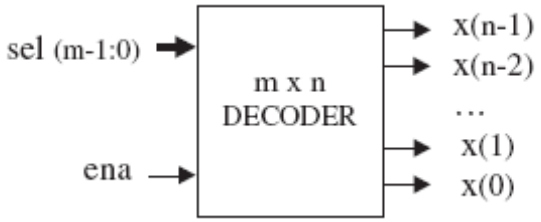
2.4. Genel (Generic) Bildirimi

Eğer her hangi bir değişkene program içerisinde ENTITY kısmında değer atanırken bu değişken generic olarak tanımlanırsa ve değişken değer atama sırasında bir de isim atanırsa bundan sonra bu değişkenin adının programın herhangi bir yerinde geçmesi durumunda değeri daha önce atanan değer olarak alınır.

```
GENERIC (Parametre_adı : parametre_tipi := parametre_değeri);
```

Aşağıdaki örnekte n olarak tanımlanan INTEGER değişkene 8 değerinin atanması gösterilmiştir. Atama ENTITY kısmında yapılmıştır ama ARCHITECTURE kısmında da değişken kullanılabilir.

```
ENTITY my_entity IS
GENERIC (n : INTEGER := 8);
PORT (...);
END my_entity;
ARCHITECTURE my_architecture OF my_entity IS
...
END my_architecture:
```



Şekil 2.3. mxn'lik bir decoder

Daha açık bir örnek üzerinden bu değişken tipi Şekil 2.3'de gösterilen kod çözücü devresinin gerçekleştirilmesi örneğinde verilmektedir. mxn'lik bir decoder m bitlik seçim ucu, enable girişi ve n bitlik çıkış ile aşağıdaki örnekte gösterildiği gibi gerçekleştirilebilir. Bu örnekte 3x8'lik bir decoder gerçekleştirilmiştir.

```
1 ----------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ----------------------------------------------
5 ENTITY decoder IS
6 PORT ( ena : IN STD_LOGIC;
7 sel : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8 x : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9 END decoder;
10 ----------------------------------------------
```

```
11 ARCHITECTURE generic_decoder OF decoder IS
12 BEGIN
13 PROCESS (ena, sel)
14 VARIABLE temp1 : STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);
15 VARIABLE temp2 : INTEGER RANGE 0 TO x'HIGH;
16 BEGIN
17 temp1 := (OTHERS => '1');
18 temp2 := 0;
19 IF (ena='1') THEN
20 FOR i IN sel'RANGE LOOP
21 IF (sel(i)='1') THEN
22 temp2:=2*temp2+1;
23 ELSE
24 temp2 := 2*temp2;
25 END IF;
26 END LOOP;
27 temp1(temp2):='0';
28 END IF;
29 x <= temp1;
30 END PROCESS;
31 END generic_decoder;
32 ---------------------------------------------
```

2.5. Paralel Kod

VHDL kodları paralel (concurrent) ya da ardışıl (sequential) olabilir. Sayısal devreler o an ki çıkışın daha önceki çıkışlara bağlı olmadığı kombinezonal devreler ve hafıza birimleri içeren ardışıl devreler olmak üzere iki kısma ayrılabilirler. Sadece kombinezonal devrelerin gerçekleştirilmesi için paralel kod kullanılabilir. Hem ardışıl devrelerin hem de kombinezonal devrelerin gerçekleştirilmesi için ardışıl kod kullanılır.

2.5.1. WHEN Bildirimi

WHEN bildirimi, paralel koddaki temel ifadelerden birisidir. İki formda kullanılır: WHEN/ELSE (basit WHEN) ve WITH/SELECT/WHEN (seçilmiş WHEN). Kullanımları aşağıda görüldüğü gibidir.

```
İş WHEN Şart ELSE
```

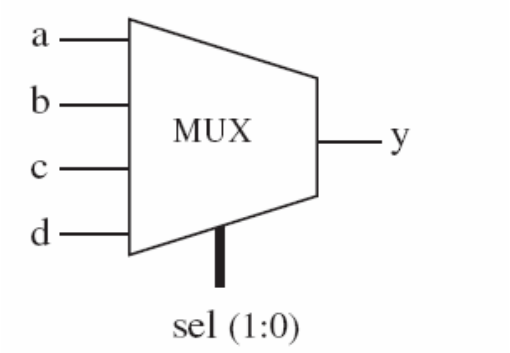
Bir örnek verilecek olursa;

```
------ WHEN/ELSE -------------------------
outp <= "000" WHEN (inp='0' OR reset='1') ELSE
        "001" WHEN ctl='1' ELSE   "010";
```

```
WITH Tanımlayıcı SELECT
İş WHEN Değer,
İş WHEN Değer,
….;
```

```
---- WITH/SELECT/WHEN --------------------
WITH control SELECT
output <= "000" WHEN reset,
          "111" WHEN set,
UNAFFECTED WHEN OTHERS;
```

WITH/SELECT/WHEN yapısında etkili durumlar ayrıntılı bir şekilde analiz edilmelidir. Eğer durum dışında bir değişkenin değerinin değişmesi istenmiyorsa belirli UNAFFECTED oldukça faydalı bir komut olarak ortaya çıkar.



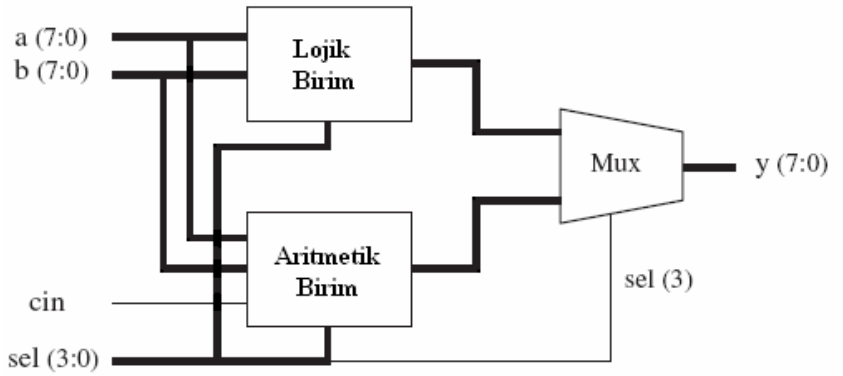
Şekil 2.4. 4x1'lik bir veri seçici (Multiplexer).

Şekil 2.4'deki veri seçici bu yapı kullanılarak VHDL kodları ile üretilmiştir.

```
1 ------- Çözüm 1:  WHEN/ELSE --------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY mux IS
6 PORT ( a, b, c, d: IN STD_LOGIC;
7 sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
8 y: OUT STD_LOGIC);
9 END mux;
10 ---------------------------------------
11 ARCHITECTURE mux1 OF mux IS
12 BEGIN
13                    y <= a WHEN sel="00" ELSE
14                          b WHEN sel="01" ELSE
15                          c WHEN sel="10" ELSE
16                          d;
17 END mux1;
18 ---------------------------------------
1 --- Çözüm 2: WITH/SELECT/WHEN -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY mux IS
6 PORT ( a, b, c, d: IN STD_LOGIC;
7           sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
8           y: OUT STD_LOGIC);
9 END mux;
10 ---------------------------------------
11 ARCHITECTURE mux2 OF mux IS
12 BEGIN
13 WITH sel SELECT
14                    y <= a WHEN "00", -- ";" yerine ","
```

```
15                      b WHEN "01",
16                      c WHEN "10",
17                      d WHEN OTHERS;
18 END mux2;
19 -------------------------------------------
```

Şekil 2.5'de görülen ALU hem lojik işlemleri hem de aritmetik işlemleri yapma kapasitesine sahip bir devredir. Tablo 2.1'deki doğruluk tablosuna göre devrenin VDHL kodu elde edilebilir.



Şekil 2.5. Bir ALU birimi.

Tablo 2.1. ALU birimine ait doğruluk tablosu.

sel	Yapılacak işlem	Fonksiyon	Çalışan birim
0000	y<=a	a'yı çıkışa transfer et	Lojik Birim
0001	y<=a+1	a'yı arttır	
0010	y<=a-1	a'yı azalt	
0011	y<=b	b'yi çıkışa transfer et	
0100	y<=b+1	b'yi arttır	
0101	y<=b-1	b'yi azalt	
0110	y<=a+b	a ve b'yi topla	
0111	y<=a+b+cin	a ve b'yi elde biti ile topla	
1000	y<=NOT a	a'nın değili	Aritmetik Birim
1001	y<= NOT b	b'nin değili	
1010	y<=a AND b	a AND b	
1011	y<=a OR b	a OR b	
1100	y<=a NAND b	a NAND b	
1101	y<=a NOR b	a NOR b	
1110	y<=a XOR b	a XOR b	
1111	y<=a XNOR b	a XNOR b	

```
1 ------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE ieee.std_logic_unsigned.all;
5 ------------------------------------------------
6 ENTITY ALU IS
7         PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8                   sel: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
9                   cin: IN STD_LOGIC;
10                  y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END ALU;
12 ------------------------------------------------
13 ARCHITECTURE dataflow OF ALU IS
14        SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15         BEGIN
16 ----- Arithmetic unit: ------
17        WITH sel(2 DOWNTO 0) SELECT
18                  arith <= a WHEN "000",
19                  a+1 WHEN "001",
20                  a-1 WHEN "010",
21                  b WHEN "011",
22                  b+1 WHEN "100",
23                  b-1 WHEN "101",
24                  a+b WHEN "110",
25                  a+b+cin WHEN OTHERS;
26 ----- Logic unit: -----------
27        WITH sel(2 DOWNTO 0) SELECT
28                  logic <= NOT a WHEN "000",
29                  NOT b WHEN "001",
30                  a AND b WHEN "010",
31                  a OR b WHEN "011",
32                  a NAND b WHEN "100",
33                  a NOR b WHEN "101",
```

```
34                  a XOR b WHEN "110",
35                  NOT (a XOR b) WHEN OTHERS;
36 -------- Mux: ---------------
37        WITH sel(3) SELECT
38         y <= arith WHEN '0',
39         logic WHEN OTHERS;
40 END dataflow;
41 ---------------------------------------------
```

2.5.2. GENERATE Bildirimi

GENERATE diğer bir paralel kod döngü elamanıdır. Ardışıl koddaki LOOP komutuna karşılık gelir. Düzenli formu FOR / GENERATE şeklindedir. Düzensiz formu ise IF / GENERATE şeklinde olup kullanım şekilleri aşağıdaki gibidir. Bir etiket ile beraber kullanılmalıdır.

```
Etiket: FOR Tanımlayıcı IN Aralık GENERATE
          (Paralel ifade)
END GENERATE;
```

```
Etiket1: FOR Tanımlayıcı IN Aralık GENERATE
...
Etiket2: IF Şart GENERATE
          (Paralel ifade)
END GENERATE;
...
END GENERATE;
```

Aşağıda kullanımına ilişkin bir örnek verilmiştir.

```
SIGNAL x: BIT_VECTOR (7 DOWNTO 0);
SIGNAL y: BIT_VECTOR (15 DOWNTO 0);
SIGNAL z: BIT_VECTOR (7 DOWNTO 0);
...
```

```
G1: FOR i IN x'RANGE GENERATE
        z(i) <= x(i) AND y(i+8);
END GENERATE;
```

2.5.3. BLOCK Bildirimi

Basit (SIMPLE) ve bekçili (GUARDED) olmak üzere iki çeşidi vardır. Özellikler uzun kodların bölünmesinde kullanılırlar. Böylelikle kod daha rahat okunabilir ve yönetilebilir hale gelir. Genel kullanımı aşağıda görüldüğü gibidir.

```
Etiket: BLOCK
        [Tanımlamalar]
BEGIN
        (Paralel İfadeler)
END BLOCK Etiket;
```

Bekçili BLOCK ise BLOCK bildiriminin ek ifadeler içeren özel bir şeklidir. Eğer ifadedeki bekçi ifade doğru ise bu blok işlem görür. İfadenin şekli aşağıdaki gibidir.

```
Etiket: BLOCK (Bekçi ifade)
        [Tanımlayıcı kısım]
BEGIN
        (Paralel bekçili ya da bekçisiz ifadeler)
END BLOCK Etiket;
```

Aşağıdaki örnekte bir kapının bekçili BLOCK ifade ile gerçekleştirilmesi görülmektedir. ancak saat işareti “1” ise q<=d olur. Burada bekçi ifade clk=‘1’ ifadesidir.

```
1 --------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 --------------------------------
5 ENTITY latch IS
6 PORT (d, clk: IN STD_LOGIC;
7 q: OUT STD_LOGIC);
```

```
8 END latch;
9 -------------------------------
10 ARCHITECTURE latch OF latch IS
11 BEGIN
12 b1: BLOCK (clk='1')
13 BEGIN
14 q <= GUARDED d;
15 END BLOCK b1;
16 END latch;
```

2.6. Ardışıl Kod

PROCESS, FUNCTION ve PROCEDURE ardışıl olarak işlev gören kodlardır. Ardışıl kodda önemli olan noktalardan birisi sadece ardışıl lojik devrelerin analizinde kullanılması ile sınırlı değildir. Ardışıl devreleri modelleyebileceğimiz gibi kombinezonal devreleri de modelleyebiliriz. Bu nedenle bu kod davranışsal kod (behavioral code) olarak da bilinir. Tüm ardışıl ifadeler PROCESS, FUNCTION ve PROCEDURE blokları içersinde yer almalıdırlar. Bu ifadeler, IF; WAIT; CASE ve LOOP'tur. VARIABLE da sadece yukarıda adı geçen blokların içerisinde tanımlanmalıdır.

2.6.1. PROCESS Bildirimi

PROCESS, VHDL kodun ardışıl kısmını oluşturur. PROCESSES ifadesi ana kodun içersinde yer almalıdır. Bildirim ifadesinin yapısı aşağıda görüldüğü gibidir.

```
[Etiket:] PROCESS (Duyarlılık listesi)
[VARIABLE ad tip [aralık] [:= başlangıç değeri;]]
BEGIN
(Ardışıl kod)
END PROCESS [Etiket];
```

VARIABLE kısmı opsiyoneldir. Eğer yapılması gerekiyorsa PROCESS bildiriminden sonra yapılmalıdır. Başlangıç şartı sentezlenebilir değildir sadece simülasyonlarda sentezlenir. Etiket kullanımı da opsiyonel olup kullanımındaki amaç kodun okunabilirliliğini arttırmaktır.

2.6.2. IF Bildirimi

IF bir ardışıl kod elemanı olduğu için sadece PROCESS, FUNCTION ve PROCEDURE blokları arasında kullanılabilir. IF bildiriminin yapısı aşağıda görüldüğü gibidir.

```
IF şart THEN ifade;
ELSIF şart THEN ifade;
...
ELSE ifade;
END IF;
```

Aşağıda kullanımı ile ilgili bir örnek verilmiştir.

```
IF (x<y) THEN temp:="11111111";
ELSIF (x=y AND w='0') THEN temp:="11110000";
ELSE temp:=(OTHERS =>'0');
END IF
```

2.6.3. WAIT Bildirimi

WAIT komutunun çalışması bazen IF'in çalışmasına benzer. Bununla birlikte WAIT komutunun kullanılmasının birkaç formu mevcuttur. Bu formlar aşağıda gösterildiği gibidir. PROCESS bildirimi WAIT ile beraber kullanıldığında duyarlılık listesi diye bir listesi olmaz.

```
WAIT ON sinyal1 [, siynal2, ... ];
```

```
WAIT FOR zaman;
```

```
WAIT UNTIL sinyal şartı;
```

WAIT UNTIL bildirimi sadece sinyaller üzerinde kullanılır. PROCESS bloğunun herhangi bir duyarlılık listesi olmadığından dolayı WAIT UNTIL bildirimi PROCESS bloğu içersindeki ilk ifade olmalıdır. Kullanımı ile ilgili bir örnek aşağıda verilmiştir.

```
PROCESS
BEGIN
WAIT UNTIL (clk'EVENT AND clk='1');
IF (rst='1') THEN
output <= "00000000";
ELSIF (clk'EVENT AND clk='1') THEN
output <= input;
END IF;
END PROCESS;
```

2.6.4. CASE Bildirimi

CASE bildirimini yapısı aşağıda görüldüğü gibidir.

```
CASE tanımlayıcı IS
WHEN değer => ifade;
WHEN değer => ifade;
...
END CASE;
```

Kullanımını bir örnek ile görecek olursak;

```
CASE kontrol IS
WHEN "00" => x<=a; y<=b;
WHEN "01" => x<=b; y<=c;
WHEN OTHERS => x<="0000"; y<="ZZZZ";
END CASE;
```

Ardışıl koddaki CASE bildirimi kombinezonal ifadede kullanılan WHEN bildirimine çok benzerdir. Burada da tüm olası durumlar çok iyi değerlendirilmelidir. Dolayısıyla burada OTHERS bildirimi oldukça faydalı bir komut haline gelir.

2.6.5. LOOP Bildirimi

Eğer kod birkaç kez çalıştırılacaksa LOOP komutunun kullanılması oldukça faydalı olur. LOOP komutunun yapısı aşağıdaki gibidir. LOOP komutunun birkaç farklı kullanımı mevcuttur. Bu formlar aşağıda gösterilmiştir.

FOR / LOOP bildirimi belirli bir sayıda işlemi tekrar eder.

```
[Etiket:] FOR Tanımlayıcı IN Aralık LOOP
(Ardışıl ifadeler)
END LOOP [Etiket];
```

WHILE / LOOP: Döngü şart sağlanmayıncaya kadar devam eder.

```
[Etiket:] WHILE şart LOOP
(Ardışıl ifadeleri)
END LOOP [Etiket];
```

EXIT: Döngüyü sonlandırmak için kullanılır.

```
[Etiket:] EXIT [Etiket] [WHEN şart];
```

NEXT: Döngü adımlarını atlamak için kullanılır.

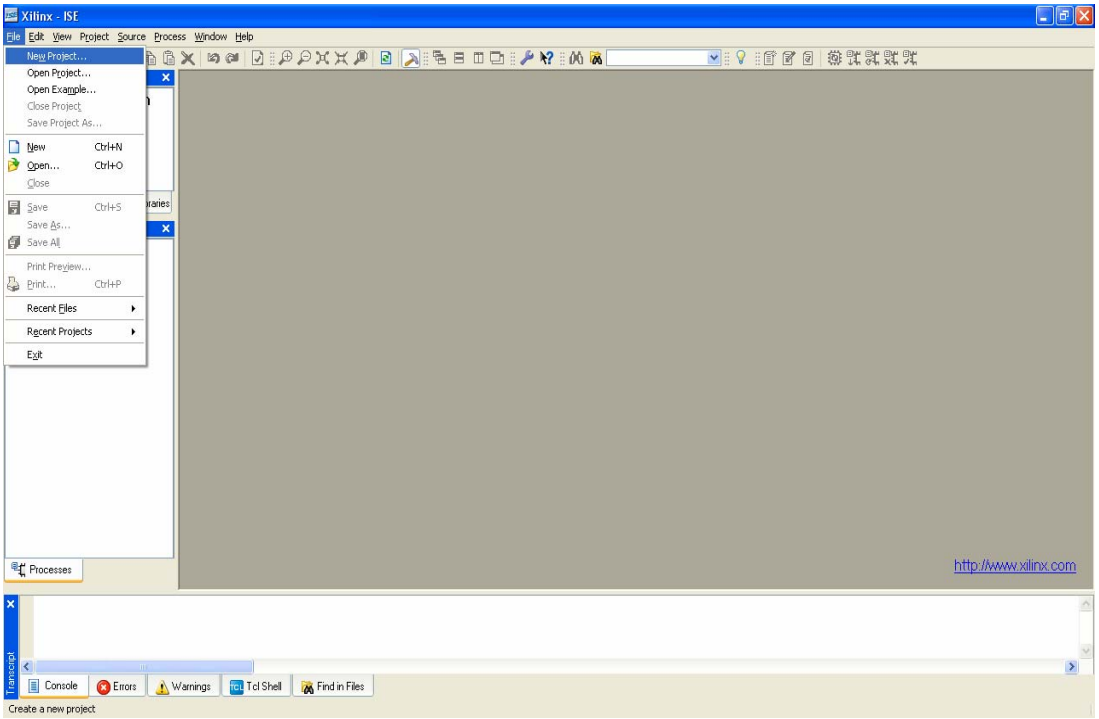
```
[Etiket:] NEXT [Döngü Etiketi] [WHENşart];
```

3. ENTEGRE YAZILIM ORTAMI

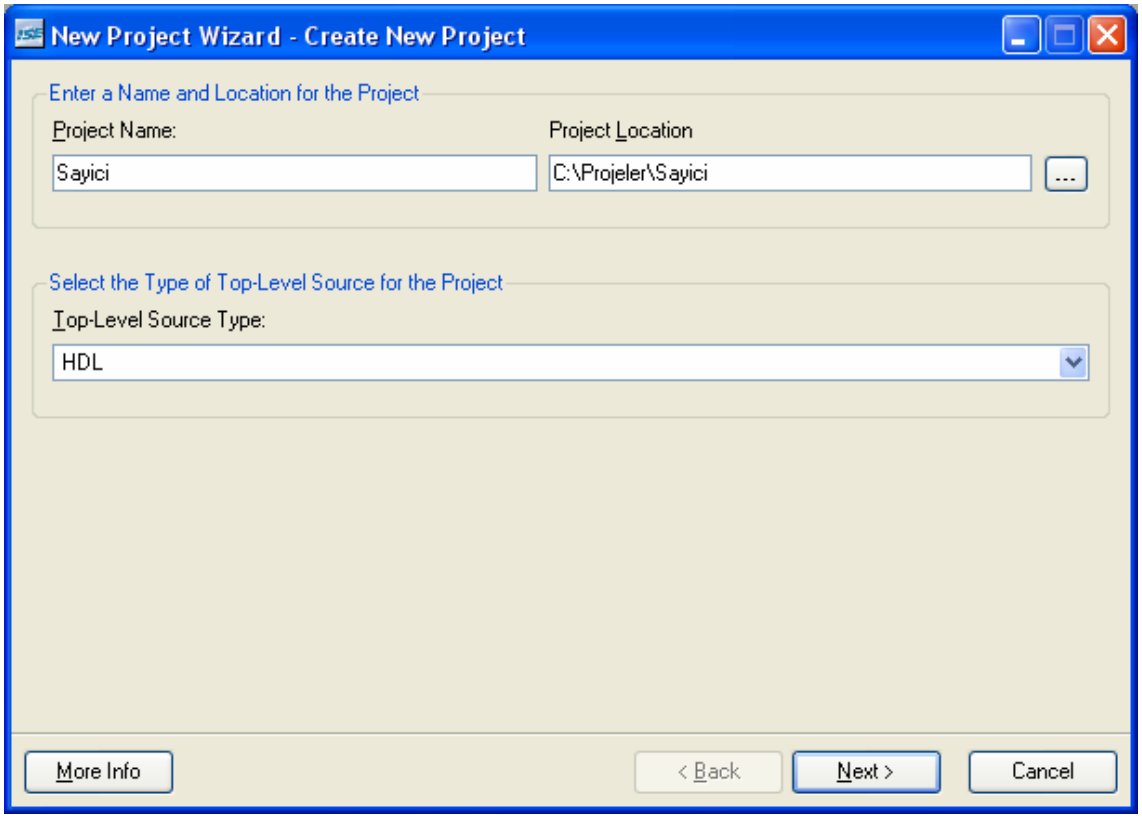
Xilinx ISE (Entegre Yazılım Ortamı-Integrated Software Environment), Xilinx SPKD'lerin üzerinde çalışılmasını sağlayan bir yazılım programıdır. Verilog veya VHDL gibi donanımsal diller kullanılarak yazılan kodlar bu program aracılığı ile sentezlenerek SPKD ortamına aktarılabilirler. ISE yazılımını çalıştırabilmek için program kurulduktan sonra masaüstünde Şekil 3.1'deki ikona tıklanarak ya da başlat menüsünden Başla → Programlar → Xilinx ISE 9.1i → Project Navigator yolu ile çalıştırılır.

Şekil 3.1. Masaüstündeki kısayol ikonu.

Yeni bir proje oluşturmak için Şekil 3.2'de görüldüğü gibi File menüsünden New Project kısmı seçilir ve seçildikten sonra Şekil 3.3'de görülen yeni proje sihirbazı ekranı çıkar.

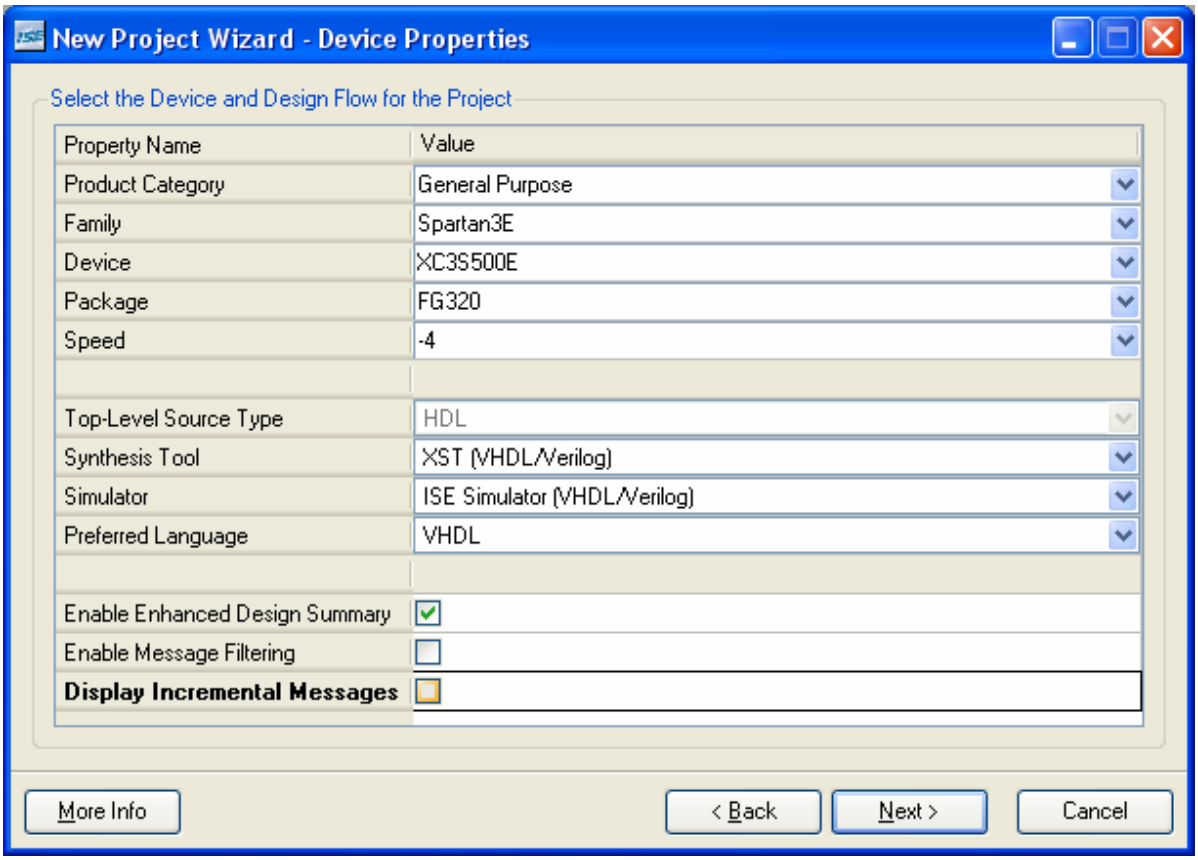


Şekil 3.2. Yeni proje dosyası açma.



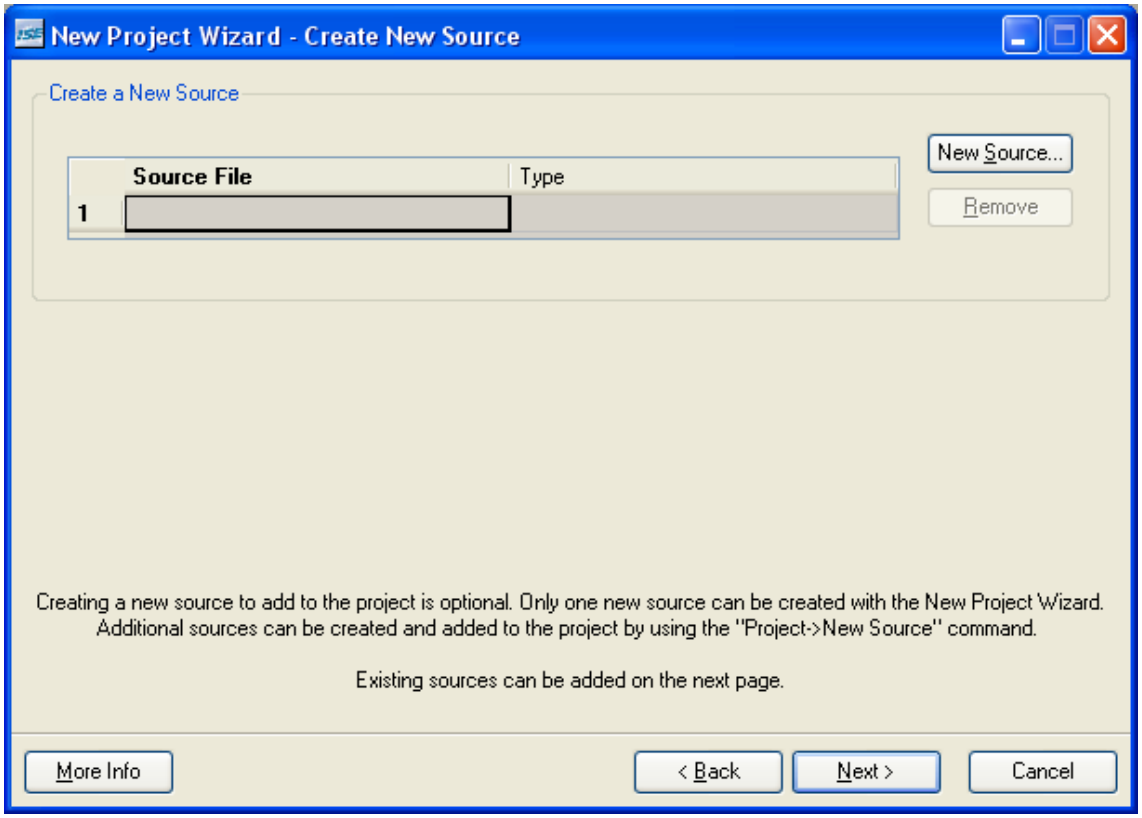
Şekil 3.3. Yeni proje dosyası hazırlama sihirbazı karşılama ekranı.

Project Name kısmına projenin adı, Project Location kısmına projenin kaydedileceği yer ve Top-Level Source Type kısmına da kullanılan kodlama dilini yazdıktan sonra next butonuna basarak sihirbazda bir sonraki aşamaya geçilir. Şekil 3.4'de görülen kullanılacak olan SPKD'nin özelliklerinin yazıldığı aşamaya geçilir. Burada ilk kısım ürünün kategorisi olup starter kit kullanıldığı için genel amaçlı kısım seçilir. Starter kit spartan 3E olup SPKD ise XC3S500E ve paket tipi FG320'dir. Hız kısmı ise -4 değerine getirilir. Kaynak tipi HDL olup bir önceki aşamada belirtiliği için bu aşama da müdahale edilememektedir. Sentez için XST(VHDL/Verilog) seçilir, simulatör olarak ISE simulatör kullanılacak ve tercih edilen dil ise VHDL olarak seçilmiştir. Bir sonraki adım olan yeni kaynak oluşturma adımına geçilir.

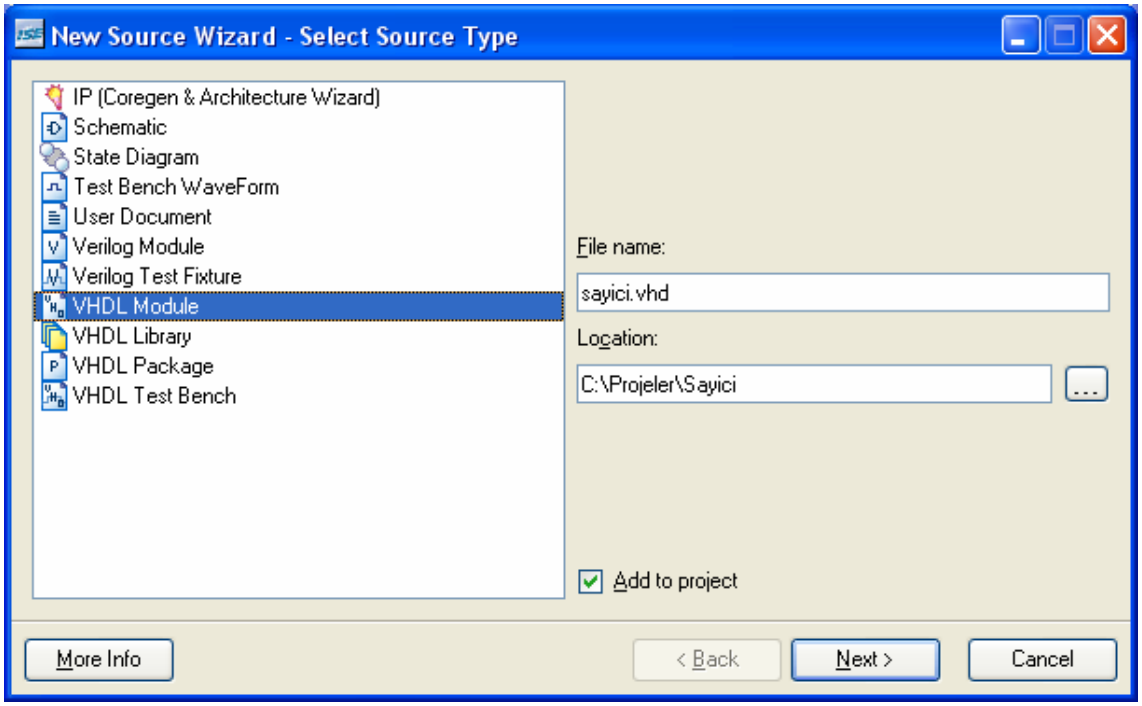


Şekil 3.4. Kullanılan cihazın özellikleri.

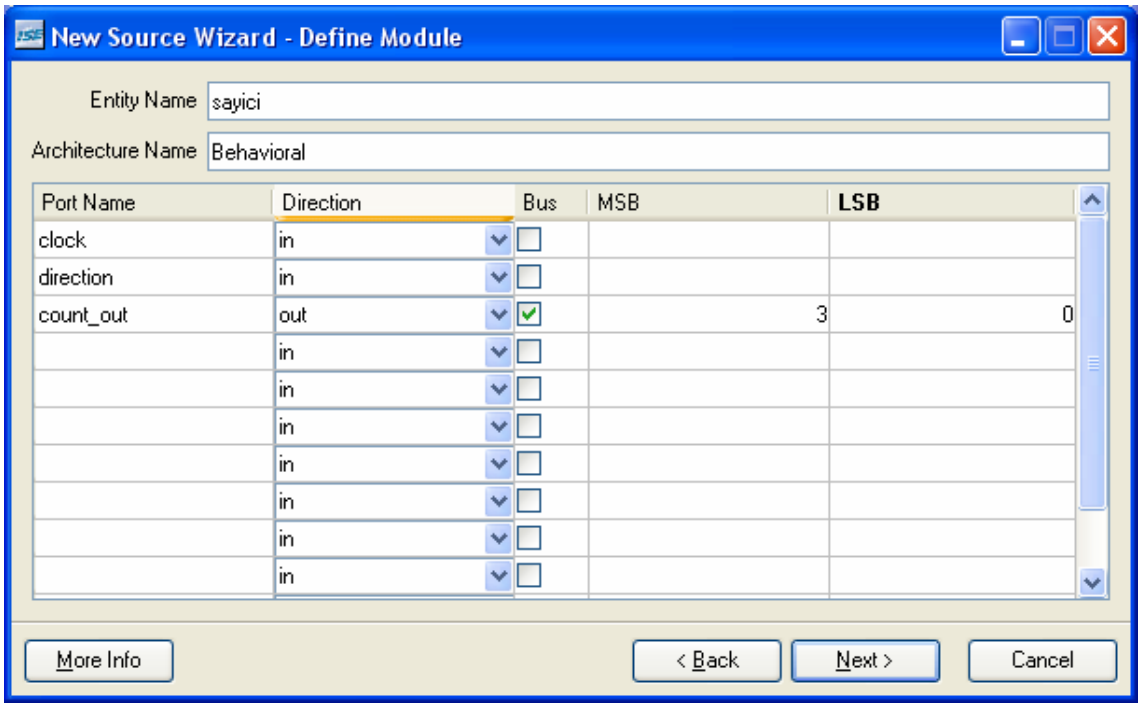
Burada tasarlanan devrenin giriş ve çıkış uçları belirlenerek isimlendirilir. Bu adım daha sonra da yapılabileceği gibi bu adımda da gerçekleştirilebilir. Şekil 3.5'de açılmış olan penceredeki New Source'a tıklandığında öncelikle kullanılacak olan kaynak tipinin belirlendiği Şekil 3.6'daki pencere açılır ve burada kaynak adı yerine de bir isim verilir. Bir sonraki adıma geçildiğinde ise giriş-çıkış pinlerinin belirlenmesi ve isimlendirilmesi yapılmaktadır. Buradaki örnekte tek giriş ve tek çıkışlı bir sistem göz önüne alınmıştır.



Şekil 3.5. Yeni kaynak oluşturma.

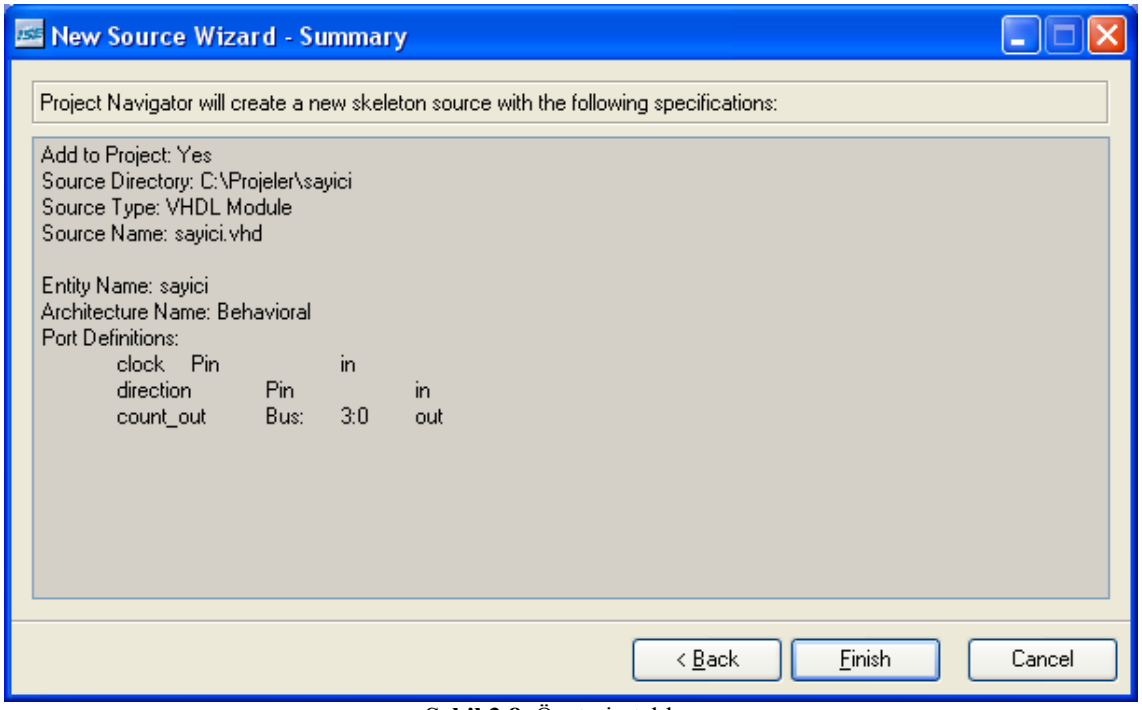


Şekil 3.6. Kaynak tipi seçme.



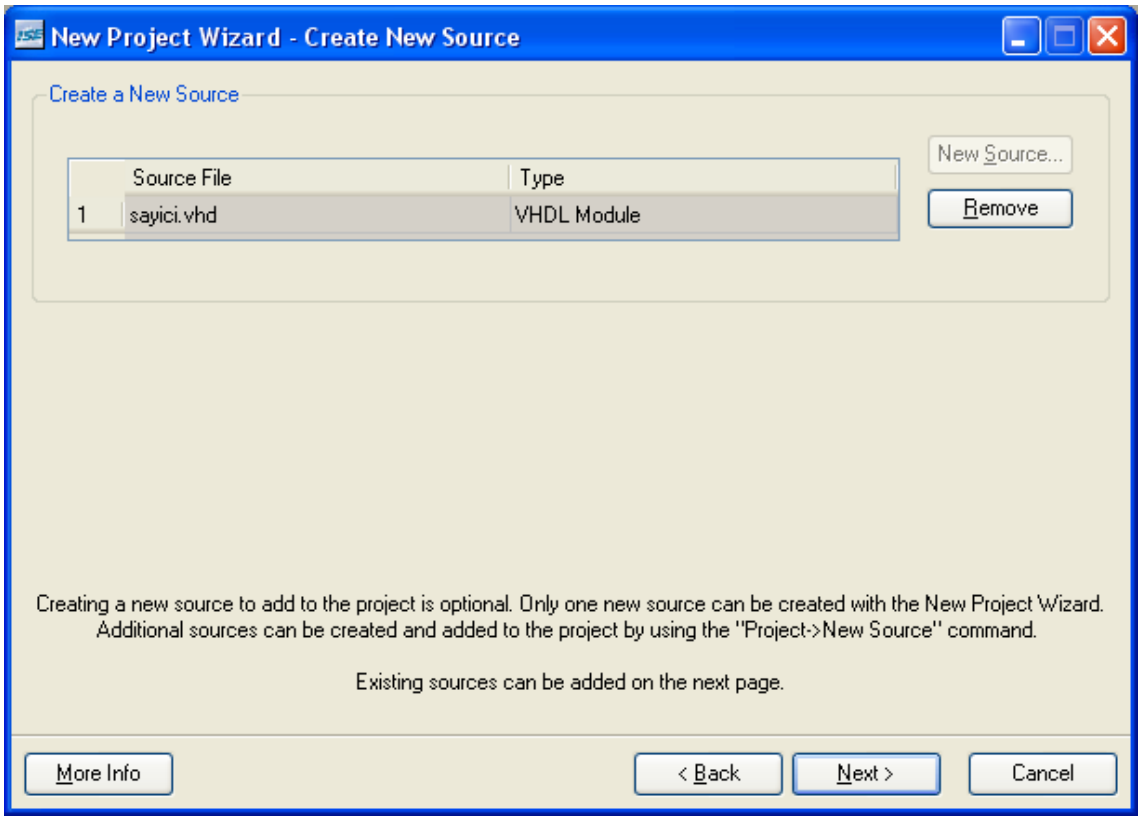
Şekil 3.7. Giriş-çıkış pinlerinin belirlenmesi.

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



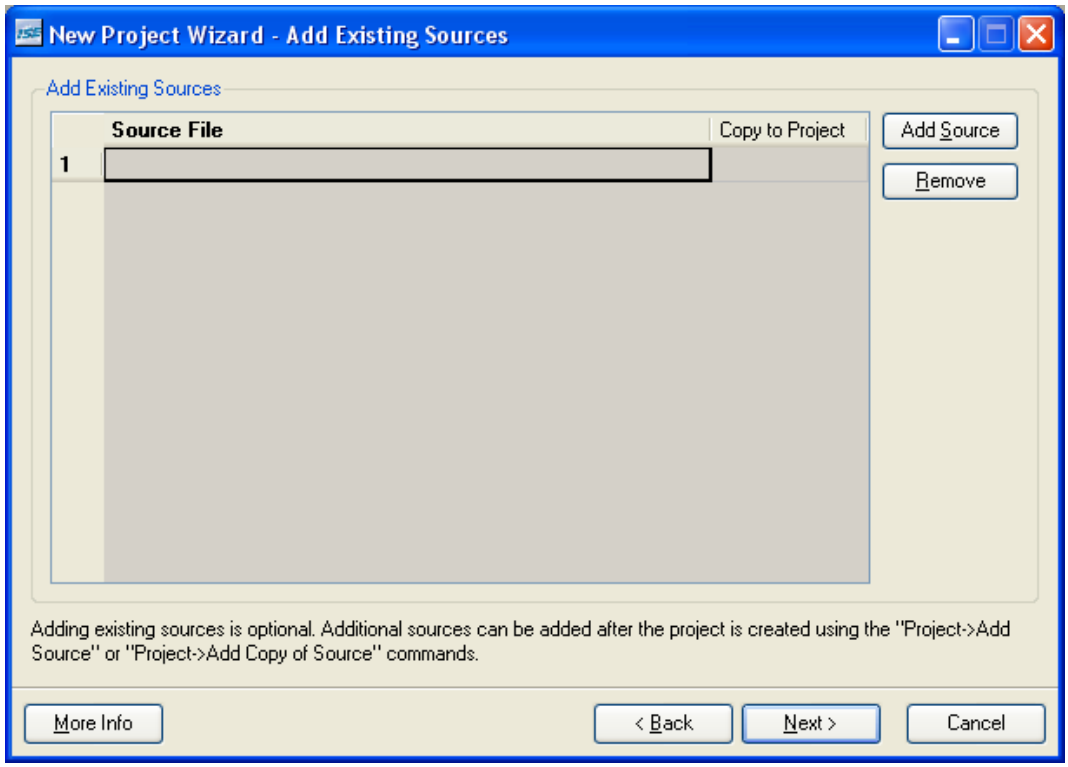
Şekil 3.8. Özet pin tablosu.

Pin tablosu görüldükten ve doğrulandıktan sonra bir sonraki adıma geçildiğinde Şekil 3.9'daki pencere ile karşılaşılacaktır. Burada, kaynak dosyanın adı ve uzantısı ile tipi yazmaktadır. Next butonuna basarak bir sonraki adıma geçilir. Proje hazırlama sihirbazı ile en fazla bir tane kaynak dosyası oluşturulabilir. Daha fazla oluşturulmak isteniyorsa Project → New Source ile eklenebilir.

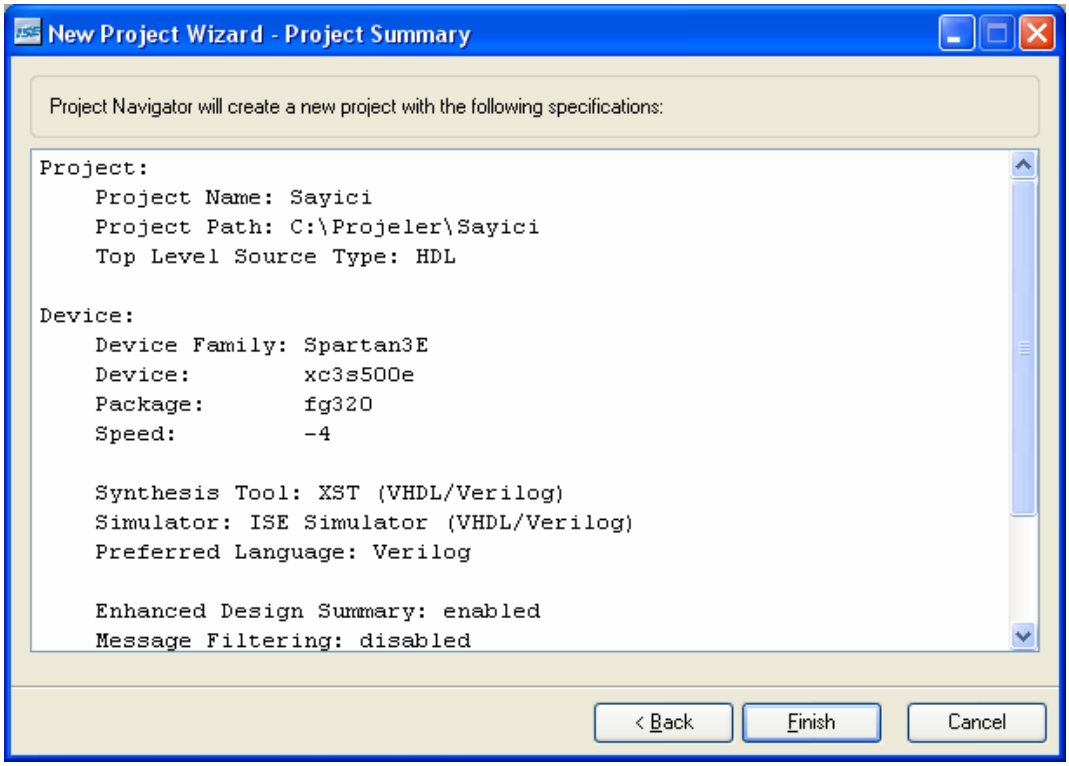


Şekil 3.9. Oluşturulmuş kaynak dosya ve tipi.

Önceden oluşturulmuş bir kaynak dosyası mevcut ise Şekil 3.10'da görülen karşılama ekranındaki Add Source butonuna basılarak kaynak dosya eklenebilir. Daha sonra oluşturulacaksa bu adım Next ile geçilir.



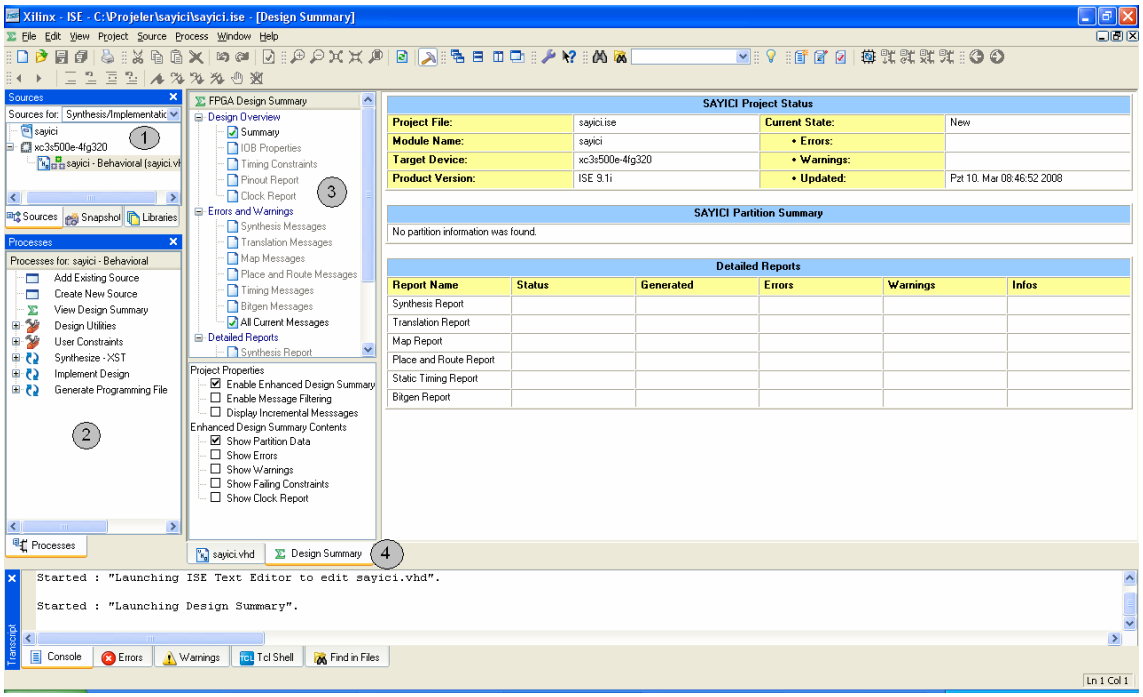
Şekil 3.10. Daha önce yazılmış kaynak dosyanın yüklenmesi.



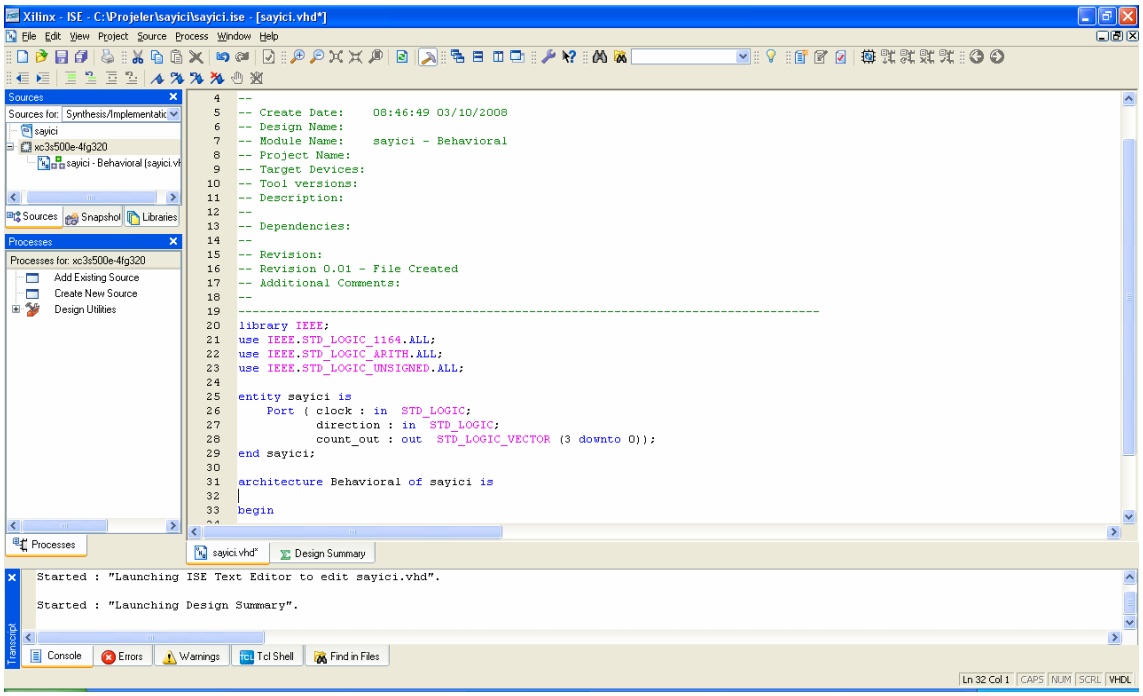
Şekil 3.11. Proje özet penceresi.

En son adımda ise Şekil 3.11'de görüldüğü gibi projenin genel bir özeti verilmektedir. Eğer şimdiye kadar yapılanlardan emin değilsek Back butonu ile geriye dönüp adımları tekrar kontrol edebiliriz. Yapılanlardan emin isek, Finish butonuna basarak sihirbazı sonlandırabiliriz.

Sihirbaz sonlandıktan sonra Şekil 3.12'de görülen tasarım özet penceresine geçilir. Burada ① ile işaretlenmiş kısımdan, proje dosyaları ve dosyalar arasındaki bağlantılar görülebilir. xc3s500e-4fg320 seçeneğine farenin sağ tuşu ile tıklanıp özelliklerine bakıldığında ayarlar değiştirilebilir. ② ile işaretlenmiş kısımda ise dosya ile ilgili işlemler yapılabilir. ③ işaretlenmiş kısımda ise dosya içeriği görüntülenir. ④ tasarım penceresi ve HDL kullanılarak yazılacak dosyalar arasında geçiş yapılabilecek kısmı gösterir. Burada, sayici.vhd yazan kısma tıklandığında Şekil 3.13'de görüldüğü gibi .vhd dosyasının bulunduğu pencere açılır.



Şekil 3.12. ISE tasarım özeti.



Şekil 3.13. Sayici.vhd dosyası.

Şekil 3.13'den görüldüğü üzere gerekli port tanımlamaları sihirbazın tamamlanması aşamasında girildiği için otomatik olarak yapılmıştır.

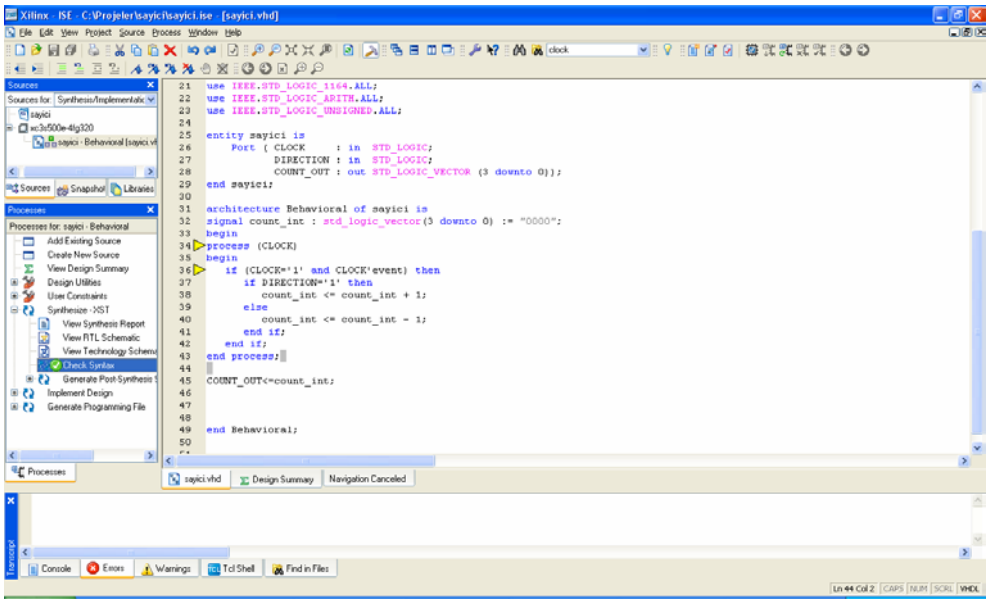
Architecture ve End Behavioral satırları arasına yukarıda görülen programdaki gibi sayıcının yapması gereken işlemler yazılır.

```
-- Module Name:    sayici - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
-- Dependencies:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sayici is
   Port ( CLOCK     : in  STD_LOGIC;
```

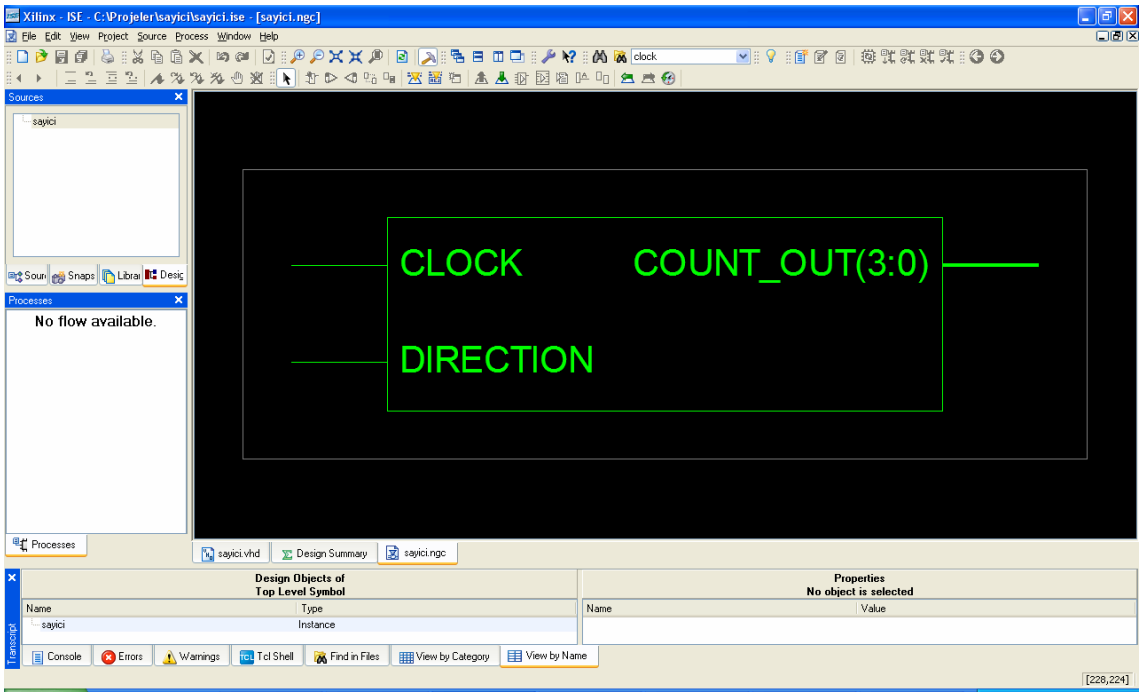
```
        DIRECTION : in  STD_LOGIC;
        COUNT_OUT : out STD_LOGIC_VECTOR (3 downto 0));
end sayici;

architecture Behavioral of sayici is
signal count_int : std_logic_vector(3 downto 0) := "0000";
begin
process (CLOCK)
begin
  if (CLOCK='1' and CLOCK'event) then
    if DIRECTION='1' then
      count_int <= count_int + 1;
    else
      count_int <= count_int - 1;
    end if;
  end if;
end process;
 COUNT_OUT<=count_int;
end Behavioral;
```

Program yazılımı bittikten sonra kaydedilerek Şekil 3.14'de görülen Process kısmında bulunan Check Syntax ile hata denetimi yapılır.

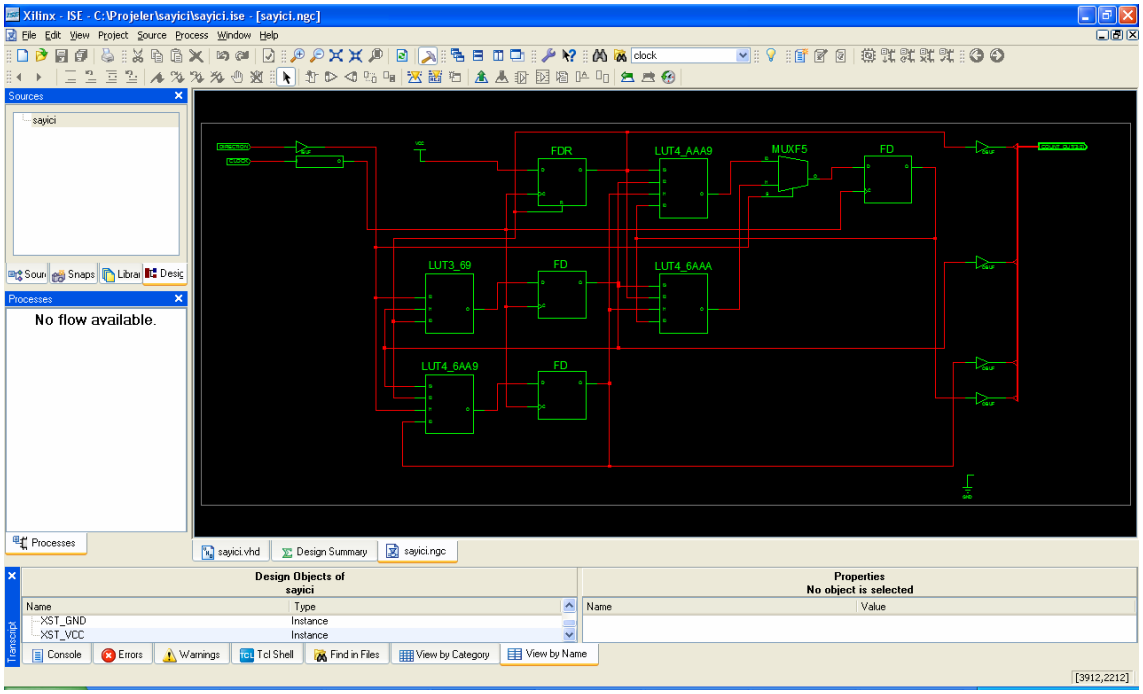
View Technology Shematic kısmına çift tıklandığında Şekil 3.15'deki gibi devrenin şematik gösterimi elde edilir. Bu gösterime de çift tıklandığında Şekil 3.16'da görüldüğü üzere devrenin tam tasarımı görülebilir. Eğer yazılan kodda hatta varsa bu çizimleri elde etmek mümkün olamaz.



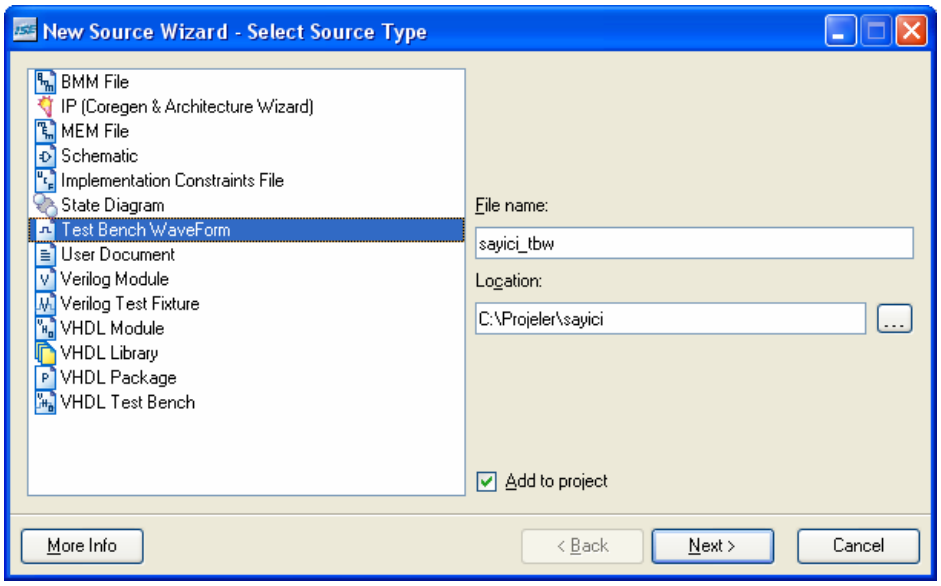
Şekil 3.14. Hata denetiminin gerçekleştirilmesi.



Şekil 3.15. Devrenin şematik gösterimi.

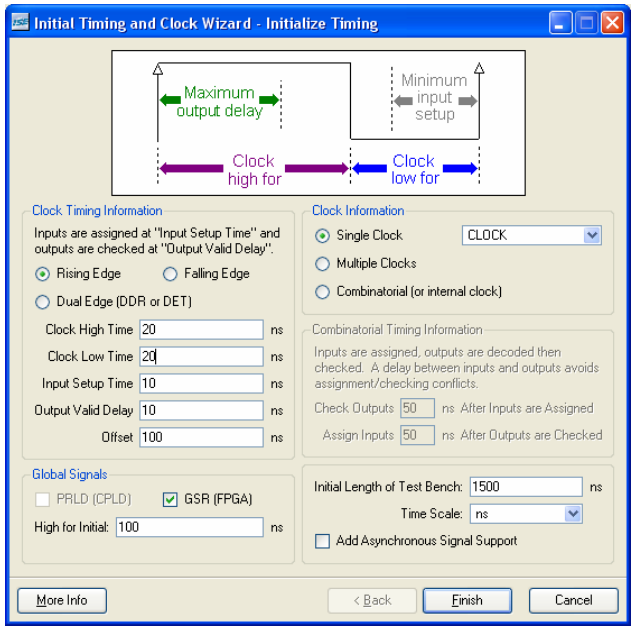


Şekil 3.16. Devrenin tam yapısı.



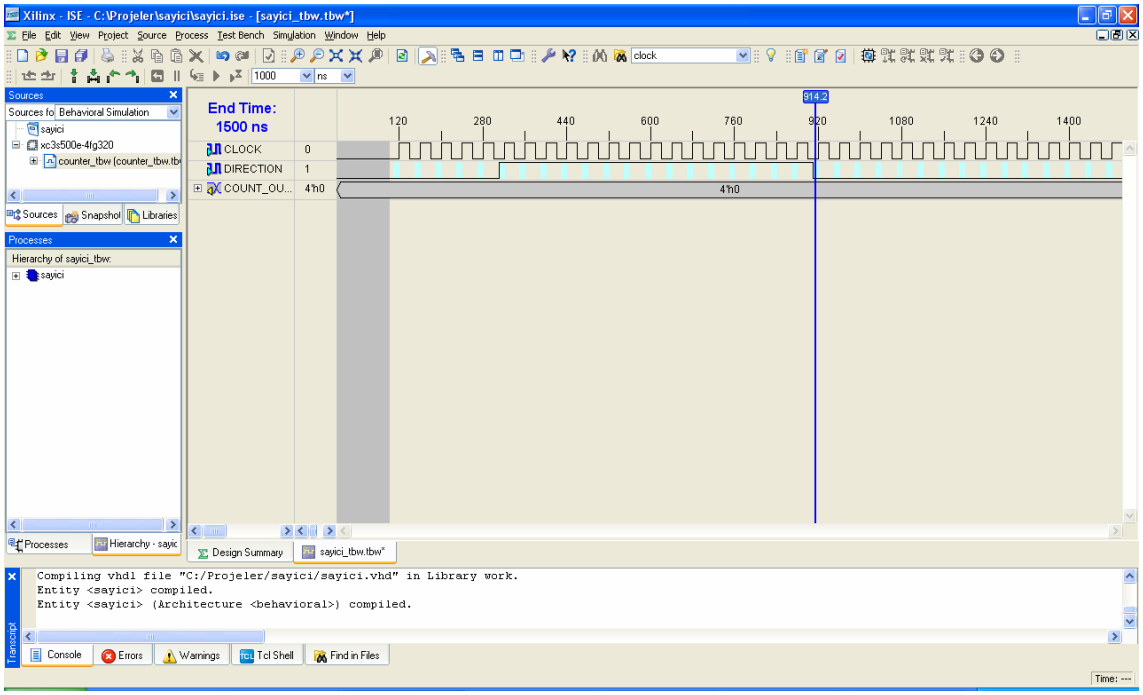
Şekil 3.17. Test sinyallerinin oluşturulması.

Bu işlemler bittikten sonra Implement Design seçeneğine çift tıklanarak modül gerçekleştirme işlemine başlanır. Yapılan tasarımın çalışıp çalışmadığını anlamak için ISE içerisinde kullanıcı tarafından oluşturulan sinyaller ile simülasyonu gerçekleştirilebilir. Bunun için, Source kısmından New source seçilir ve Şekil 3.17'deki gibi bir pencere açılır. Buradan Test Bench Waveform seçilir. Dosya adına da sayici_tbw verilmiş olsun. Next butonu ile gelen pencerelerde bir değişiklik yapılmadıktan sonra Şekil 3.18'de gösterilen uygulanacak olan kare dalgaların periyot değerleri ve özellikleri girilir.



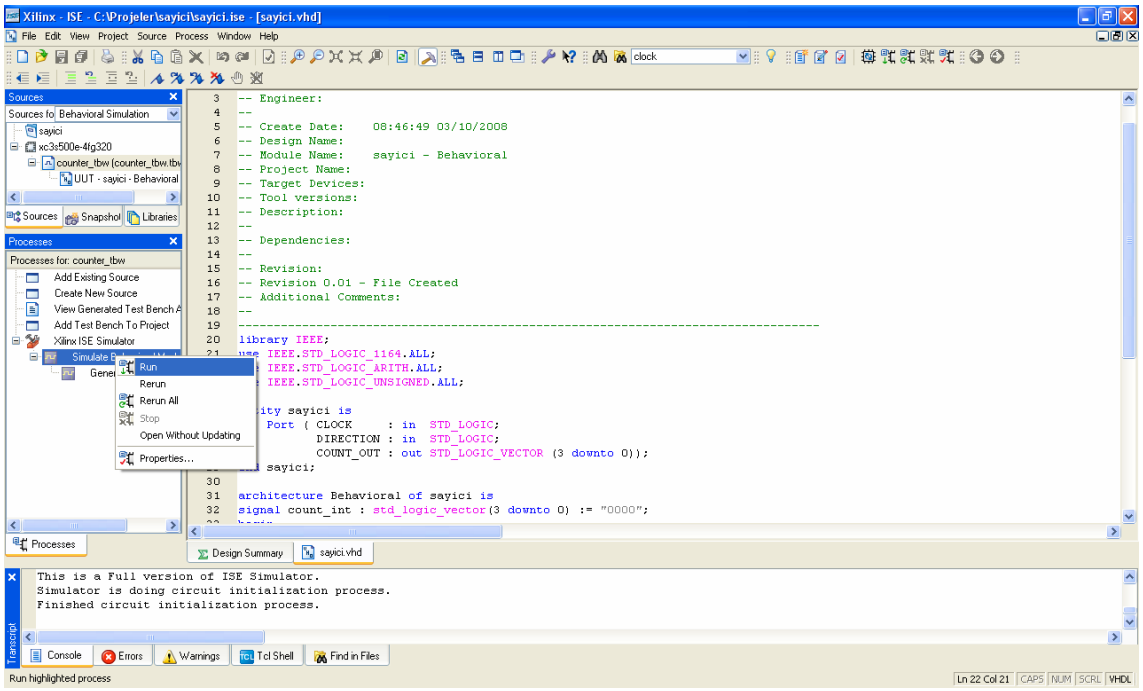
Şekil 3.18. Test sinyallerinin özellikleri ve test süresi.

İşlemci frekansı 50 MHZ olacak şekilde kare dalgaların yüksek ve düşük zaman süreleri seçilir. Global Signal kısmından GSR(FPGA) seçilir simülasyon süresi ise 1500ns olarak belirlendikten sonra Finish butonuna basılarak işlem bitirilir. Şekil 3.19'da da görüldüğü gibi sayıcının ileri geri çalışma durumunu belirleyen DIRECTION değeri yaklaşık 310 ns ve 910 ns değerleri arasında lojik-1'de olacak şekilde belirlenir. Mavi ile görülen kursor istenilen zaman bölgesine getirilip DIRECTION değişkeninin zaman eksenine tıklandığında bu değişkenin değer değiştirdiği görülecektir. Bu süreler arasında sayıcı ileri yönde bu değerler dışında ise sayıcı geriye doğru sayacaktır. İşlemler bittikten sonra test sinyalleri kaydedilir ve simülasyon dosyası içerisine gömülerek pencere kapatılır.

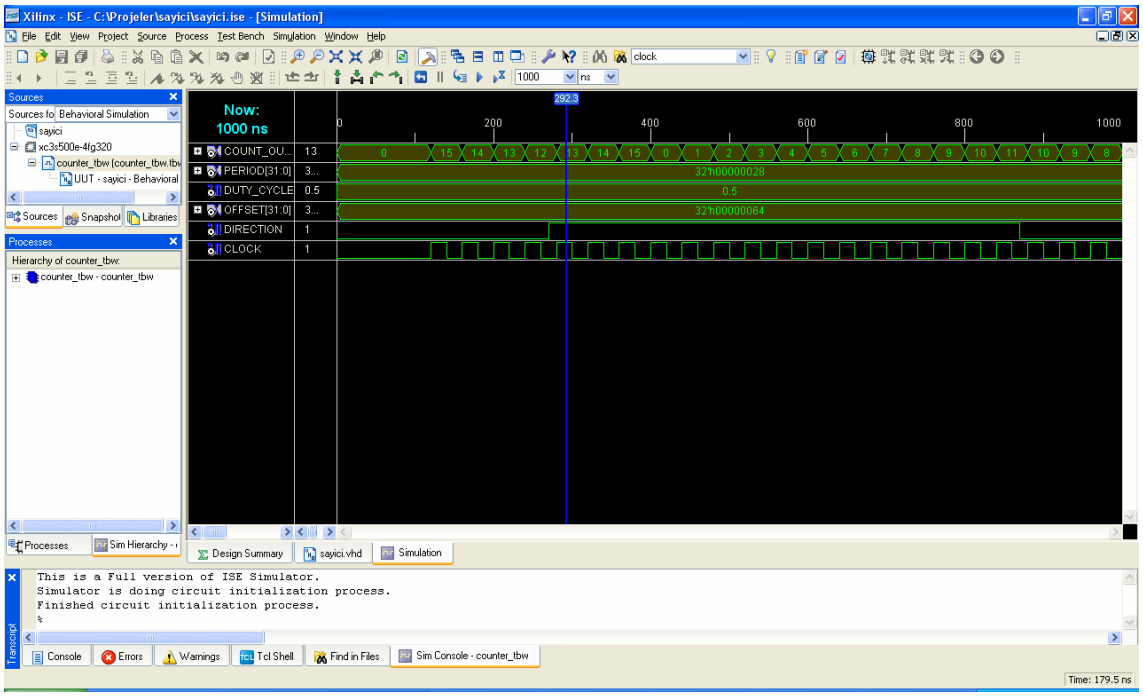


Şekil 3.19. Test değişkenlerinden olan DIRECTION değişkenine değer atanması.

Simülasyonu çalıştırmak için, Process kısmından Şekil 3.20'de görüldüğü gibi Simulate Behavioral Model farenin sağ tuşu ile seçilip Run seçeneğine tıklanıp simülasyon başlatılır.

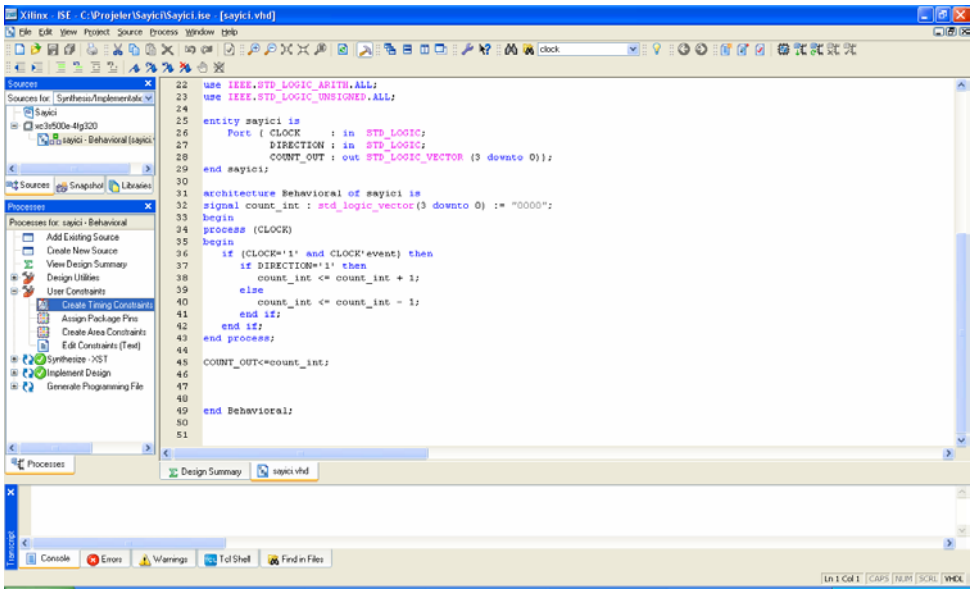


Şekil 3.20. Simülasyonun başlatılması.

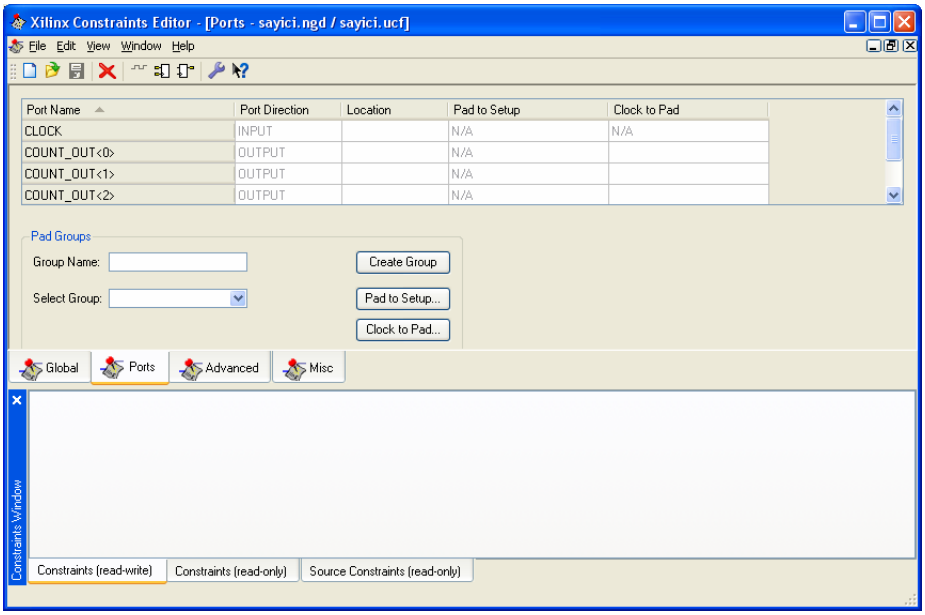


Şekil 3.21. Simülasyonun gerçekleştirilmesi.

Simülasyon gerçekleştirildikten sonra zaman kursörü istenilen zaman değerine getirilerek sayıcının bitlerinin nasıl değiştiğine bakılır. Periyot değerlerinin girilmesi için Şekil 3.22'de görülen Source kısmından Syntesis/Implementation seçilir ve Create Timing Constraints seçeneğine çift tıklanarak açılan pencereden de Evet seçeneği seçilerek Şekil 3.23'deki Editöre ulaşılır. Evet ile oluşturmak istenen UCF dosyasının projeye eklenmesinin istenildiği söylenmektedir.

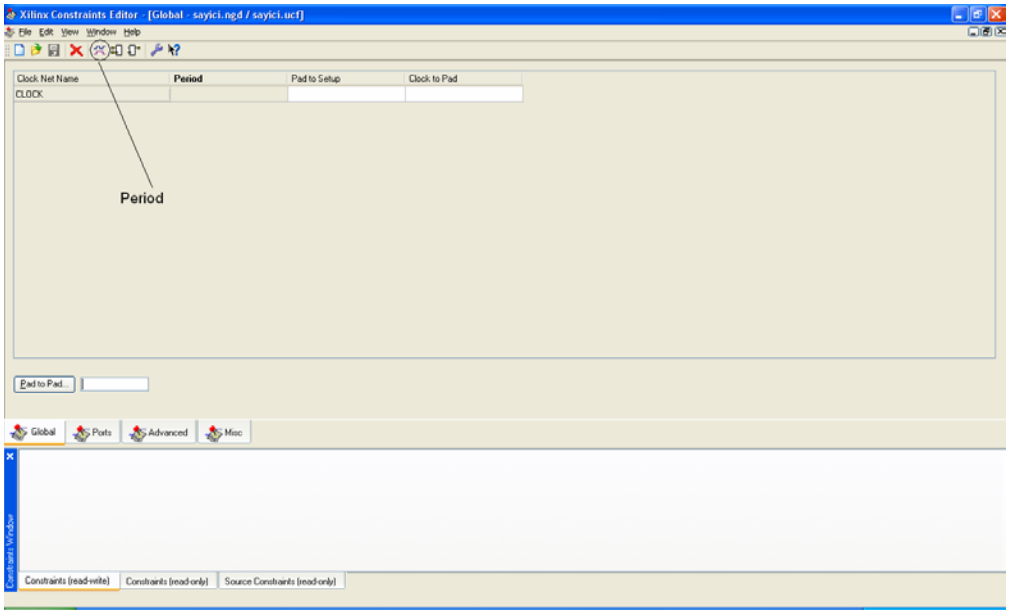


Şekil 3.22. SPKD uçlarının seçilmesi için gerekli ilk adımın gerçekleştirilmesi.



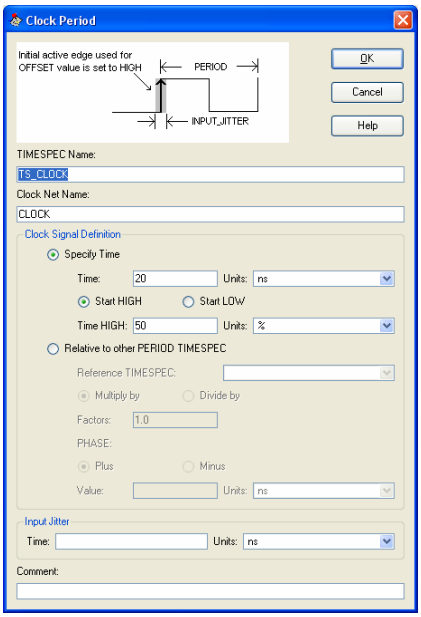
Şekil 3.23. SPKD uçlarının seçiminin gerçekleştirildiği editör.

Global kısmı seçildiğinde Şekil 3.24'deki gibi bir pencere açılacak ve bu pencereden Period kısmına tıklanarak periyot değerinin girilmesi işlemi gerçekleştirilecektir.



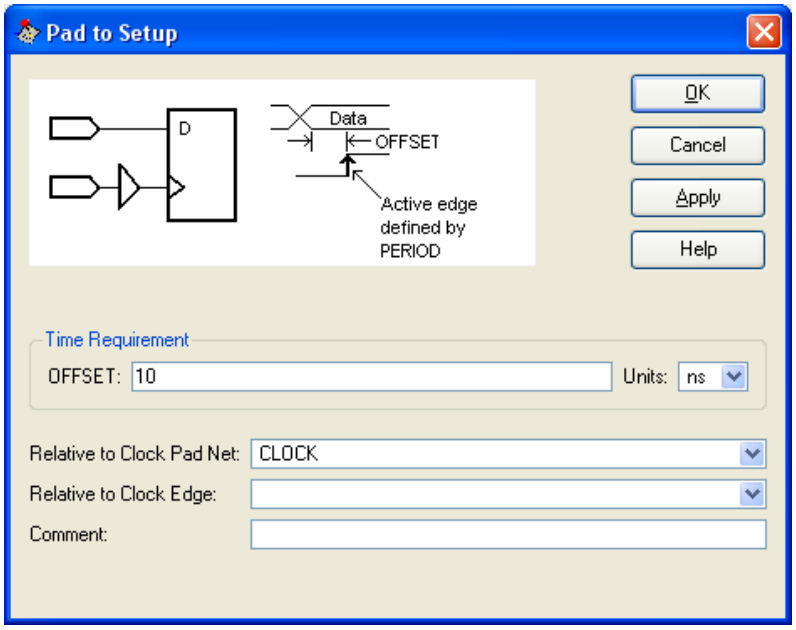
Şekil 3.24. Periyot değerinin girilmesi.

Bu seçim yapıldığında ekrana Şekil 3.25'de görülen pencere açılacaktır. Bu pencerede Time değerine 20 ns değeri girilir.



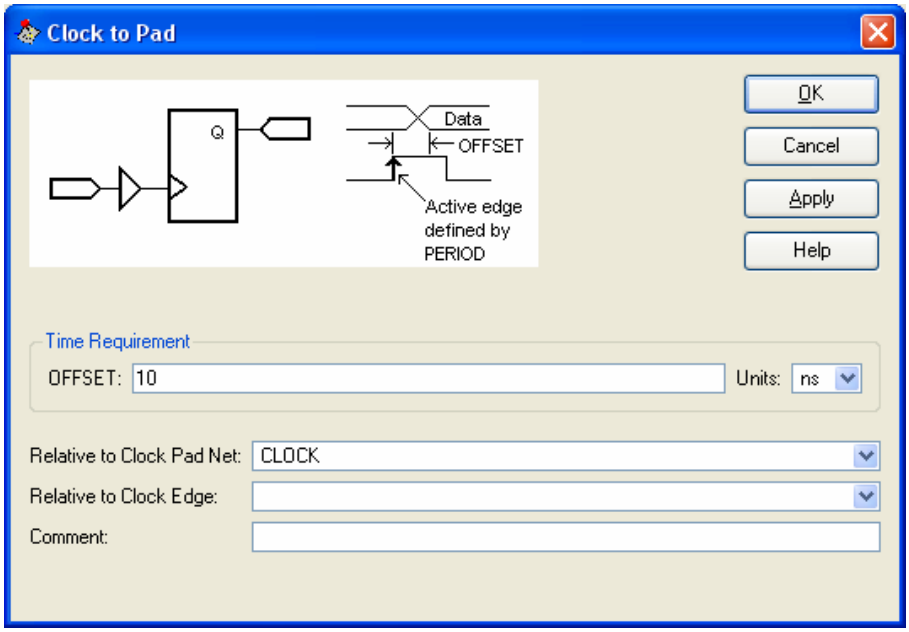
Şekil 3.25. Periyot değerinin girilmesi.

Daha sonra Pad to Setup ikonuna tıklanır ve Şekil 3.26'daki gibi offset değeri olarak 10 ns girilir.



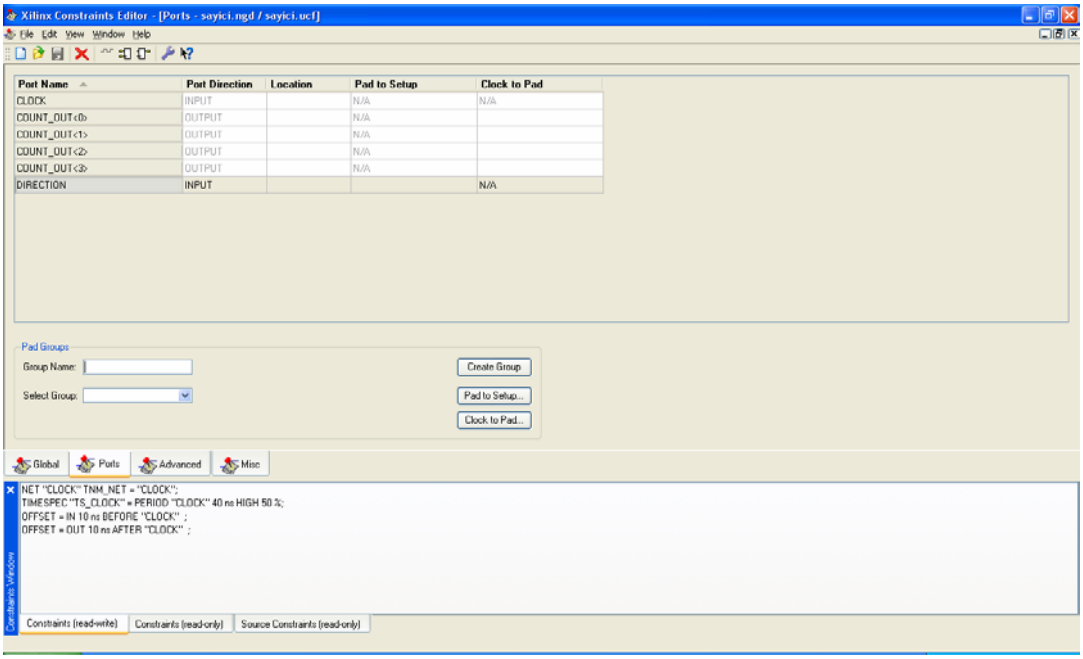
Şekil 3.26. Ofset değerinin girilmesi.

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



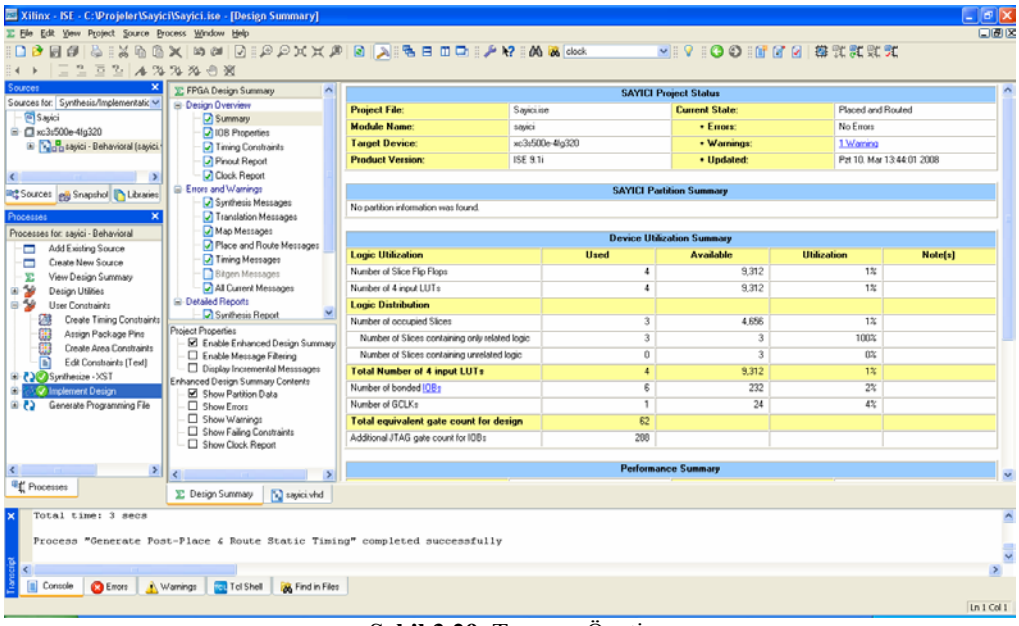
Şekil 3.27. Clock to Pad değerinin girilmesi.

Değerlerin girilmesinden sonra Şekil 3.28'de görülen dosya menüsünden bu değerler kaydedilir. Daha sonra Process kısmında bulunan Implement Design seçeneğine çift tıklanarak tasarım gerçeklenir ve oluşan tasarım özeti ekranda Şekil 3.29'daki gibi elde edilir.

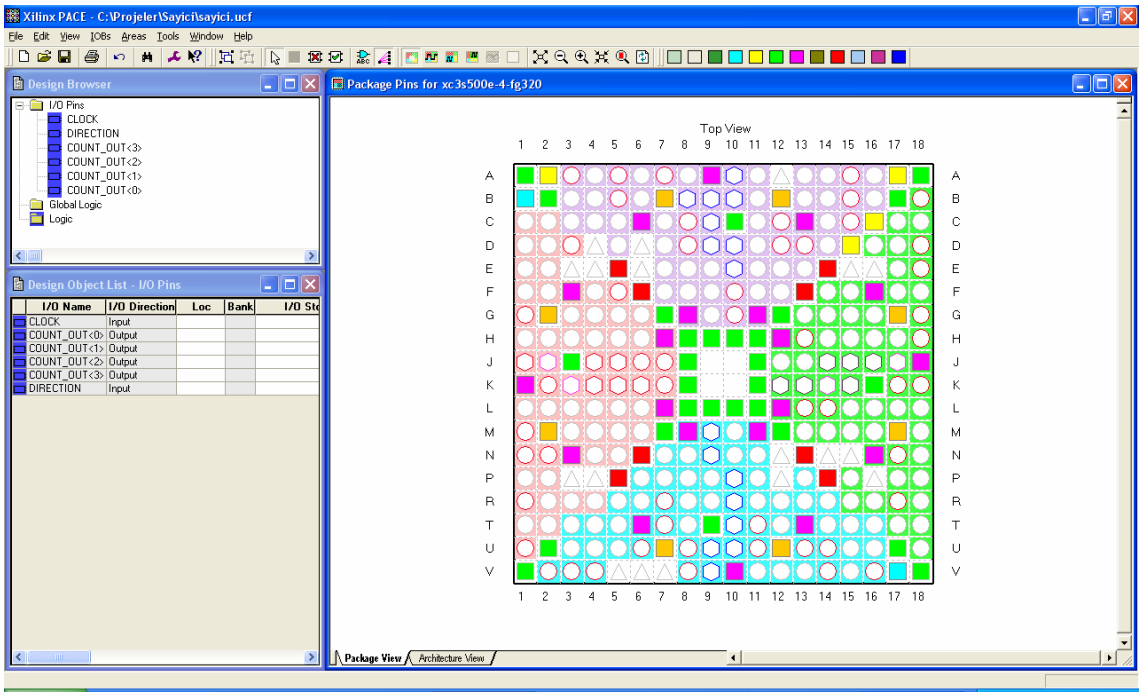


Şekil 3.28. Oluşan UCF dosyasının en son hali.

Fiziksel olarak yazılan programda hangi giriş ve çıkışların SPKD'nin hangi uçlarına karşılık geldiğini belirtmek gerekmektedir. Process menüsünden User Constraints'in altında bulunan Package Pins seçeneğine çift tıklanarak Şekil 3.30'daki SPKD'nin fiziksel pinlerinin seçimi ekrana gelir. Buradan hangi girişin ya da çıkışın hangi pine karşılık geldiği belirlenir.



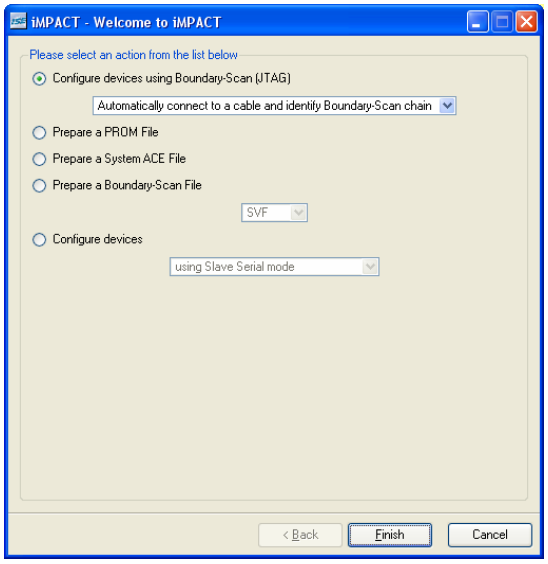
Şekil 3.29. Tasarım Özeti.



Şekil 3.30. SPKD'nin fiziksel pinlerinin seçimi.

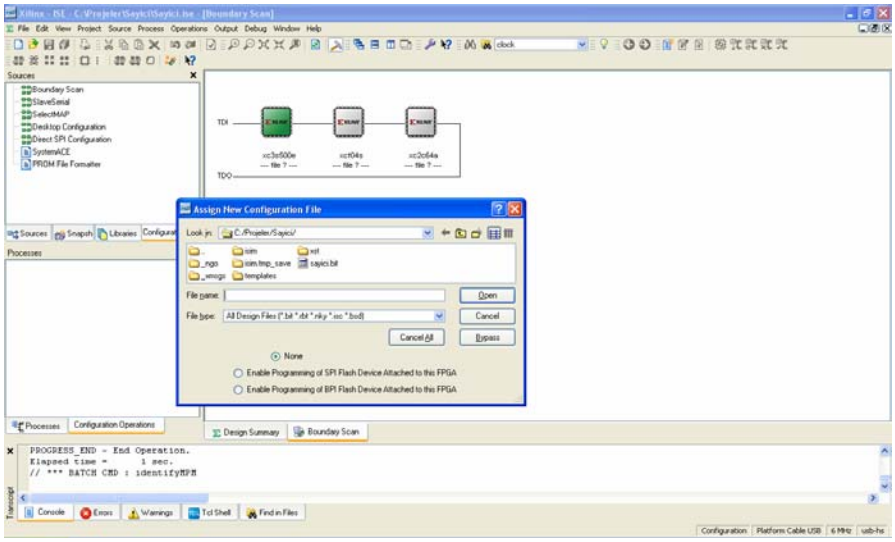
Seçimden sonra tekrar Sythesis Design ve Implement Design seçenekleri çalıştırılarak program güncellenir. Son aşama olarak Generate Programming File kısmından Configure Device (IMPACT) seçeneğine çift tıklanarak çalıştırılır. Şekil 3.31'de görüldüğü üzere gelen pencereden

otomatik olarak PC'nin starter kit'e bağlanmasına izin verilir. Finish butonuna tıklanarak bu işlem geçilir.



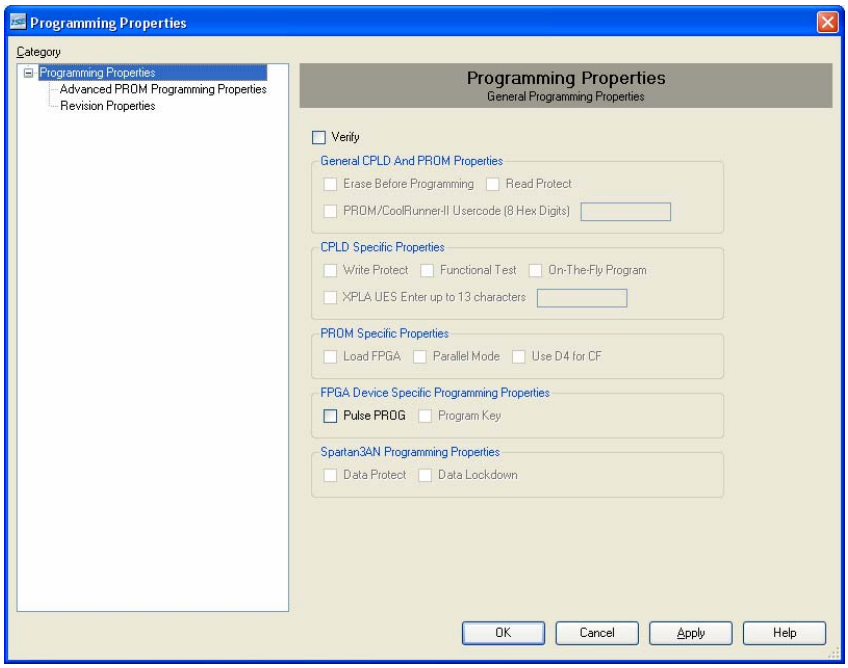
Şekil 3.31. IMPACT kullanıcı arabirimi karşılama ekranı.

Sonra yazılmış olan VHDL programının starter kit'e yüklenmesi için dosya yerinin sorulduğu Şekil 3.32'deki ekran görüntüsüne gelinir. Burada dosya seçilerek tamam butonuna basılır. Daha sonra gelen iki ekran ise bypass ile boş geçilir.

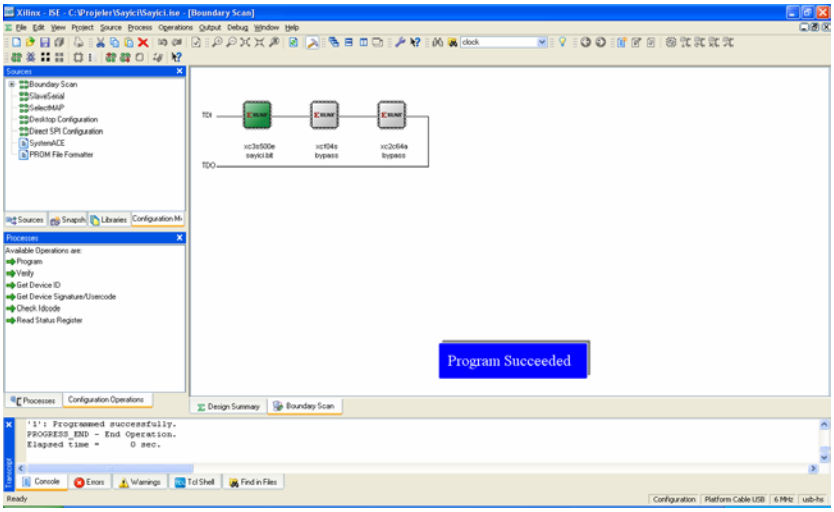


Şekil 3.32. VHDL kodunun seçimi.

Üzerinde xc3s500e yazılı işlemciye farenin sağ tuşu ile tıklanarak açılan kısımdan Program seçeneği seçilir. Şekil 3.33'de görülen pencereden OK butonuna basıldığında eğer işlemciye yazma işi gerçekleşmişse Şekil 3.34'de görüldüğü gibi Program Succeeded şekline başarılı olarak yazma işleminin gerçekleştiği görülür. Program artık SPKD starter kitinde çalışmaktadır. IMPACT kısmı kapatılırken program herhangi bir şekilde kayıt yapmak istediğinde kabul edilmeden çıkılır.



Şekil 3.33. Programın yüklenmesi.



Şekil 3.34. Programın başarılı bir şekilde yüklendiğinde verdiği mesaj.

4. SPKD KULLANILARAK İKİ KANALLI DGM SİNYALİNİN ÜRETİMİ VE DC MOTOR UYGULAMASI

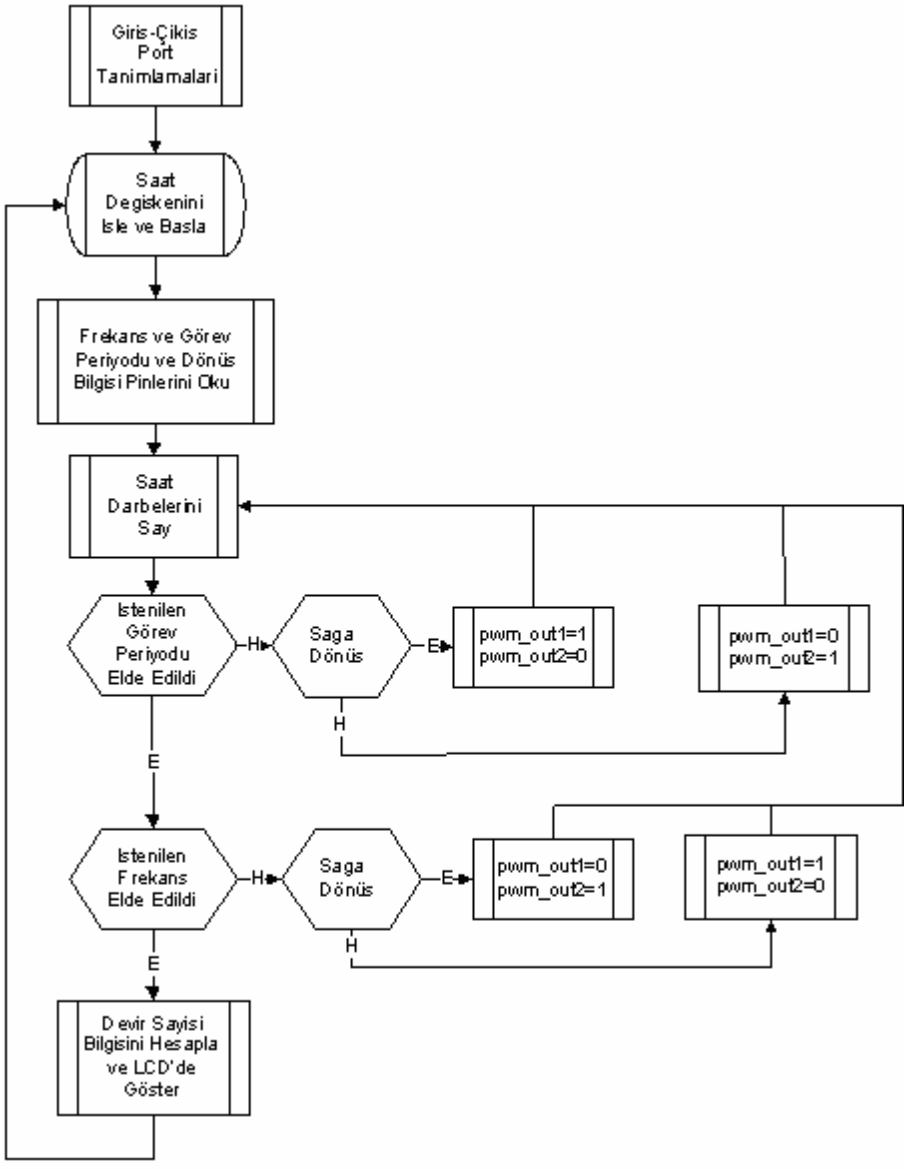
Bu çalışmada, SPKD'yi programlamak için kullanılan donanım tanımlama dili VHDL'e ait giriş-çıkış pinlerinin tanımlandığı *entity* kısmı aşağıda görüldüğü gibi tasarlanmıştır.

```
entity pwm_gen is
Port (
        pwm_out1 : out std_logic;
        pwm_out2 : out std_logic;
        tour :  in std_logic;
        rotary_a: in std_logic;
        rotary_b: in std_logic;
        rotary_c : in std_logic;
        lcd_rs : out   std_logic;
        lcd_rw : out   std_logic;
        lcd_enable : out   std_logic;
        lcd_data : inout std_logic_vector(7 downto 0);
        freq_in  : in std_logic_vector (3 downto 0);
        clk : in std_logic);
  end pwm_gen;
```

Burada, pwm_out1 ve pwm_out2 DGM sinyallerin çıkış olarak atanacağı değişkenleri belirtmektedir. Diğer değişkenler ise frekans (freq_in), görev periyodunun (rotary_a, rotary_b) değişimlerini ve dönüş yönünü (rotary_c) belirleyen değişkenlerdir. Clk değişkeni ise SPKD'de kullanılan 50 MHz'lik osilatöre ait saat değişkenidir. Tour değişkeni, optik alıcı-verici devresinden gelen darbeleri sayarak motorun devir sayısı bilgisinin bulunmasını sağlamaktadır. Görev periyodu ayarı ve dönüş yönü bilgisi bir mutlak değerli döner algılayıcı ile sağlanmakta olup frekans bilgisi ise 4-bitlik sayısal bir veridir. 4-bitlik veri, bir kod çözücü yardımı ile çözüldükten sonra karşılık gelen frekans değerine göre DGM sinyalleri üretilir. DGM sinyallerini üretmek için oluşturulan programın akış diyagramı Şekil 4.1'de verildiği gibidir.

Şekil 4.1'deki akış diyagramına göre tasarlanan ve SPKD'ye yüklenen devre Şekil 4.2'de gösterilmiştir. Gerçekleştirilen devre, yazılımla oluşturulmuş donanımsal sayısal bir devredir. Bu

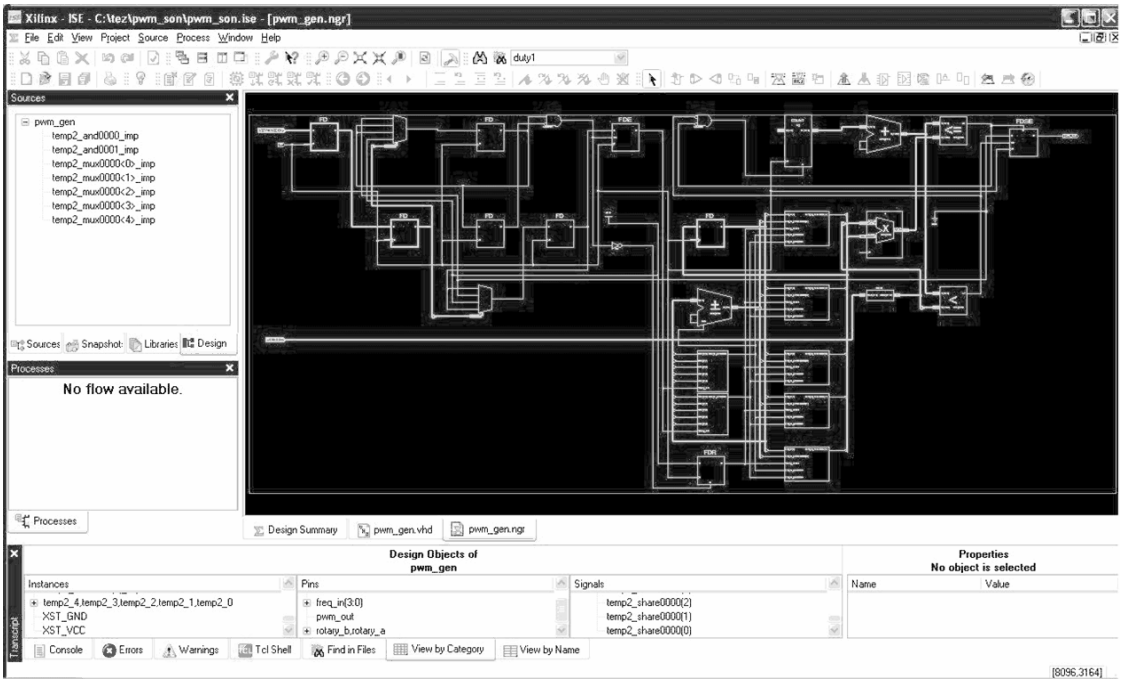
nedenle, sistem donanımsal olarak gerçekleştirildiğinden, saat darbelerinin dikkate alınmaması, mikroişlemcilerdeki gecikmeler, kesme beklenmesi gibi olumsuz etkenler oluşmayacağından devrenin çalışmasında sürekli zamanda bir problem ortaya çıkmayacaktır.



Şekil 4.1. İki kanallı DGM sinyallerini üremek için SPKD'ye yüklenen programın akış şeması.

DGM'li sinyallerin üretimi için devre tasarlanırken SPKD'de bulunan yaz-boz'ların (flip-flop), doğruluk tablolarının, kaydırmalı kayıtçıların özeti Şekil 4.3'de gösterildiği gibidir. Özette, 28

yaz-boz, 72 doğruluk tablosundan oluştuğu ve kullanılan bu değerlerin, kullanılabilecek toplam değerlerin %1'i gibi küçük bir kısmını oluşturduğu görülmektedir.



Şekil 4.2. Kayıtçı transfer seviyesinde tasarlanan devre.

PWM_SON Project Status			
Project File:	pwm_son.ise	**Current State:**	Programming File Generated
Module Name:	pwm_gen	**• Errors:**	No Errors
Target Device:	xc3s500e-4fg320	**• Warnings:**	40 Warnings
Product Version:	ISE 9.2i	**• Updated:**	Tue Jun 3 14:37:26 2008

PWM_SON Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	28	9,312	1%	
Number of 4 input LUTs	72	9,312	1%	
Logic Distribution				
Number of occupied Slices	56	4,656	1%	
Number of Slices containing only related logic	56	56	100%	
Number of Slices containing unrelated logic	0	56	0%	
Total Number of 4 input LUTs	104	9,312	1%	
Number used as logic	72			
Number used as a route-thru	30			
Number used as Shift registers	2			
Number of bonded IOBs	8	232	3%	
IOB Flip Flops	1			

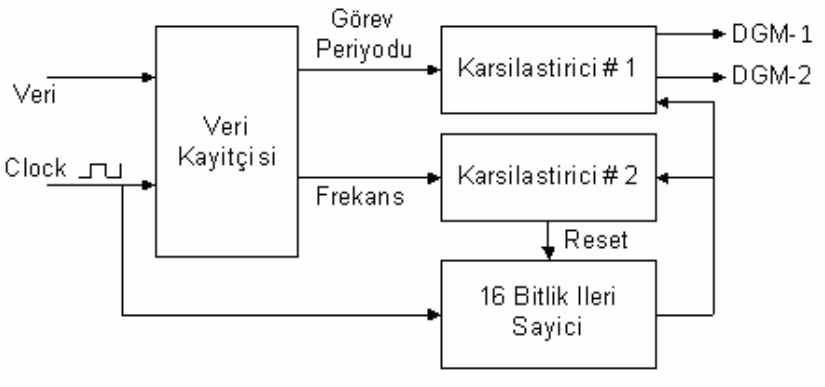
Şekil 4.3. Tasarlanan devreye ilişkin tasarım özeti.

4.1. Deneysel Çalışma

4.1.1. Sistemin SPKD'de gerçekleştirilmesi

Yazılan program ile SPKD'de gerçekleştirilen donanımın blok diyagramı Şekil 4.4'de verilmiştir. Bir veri kayıtçısı ile çevre birimi olarak nitelendirilebilecek mutlak değerli döner algılayıcı ve 4-bitlik frekans giriş bilgisi öncelikle bu kayıtçı tarafından tutularak istenilen frekans ve görev periyodu değerlerine karar verilir.

Görev periyodu değeri, %5'lik adımlarla mutlak değerli döner algılayıcı ile artırılabilen ya da azaltılabilen bir değişkene karşılık gelmektedir. Bu kayıtçı, özellikle ileri sayıcı için sayılacak değerleri ve görev periyodunun değerini belirleyen bir bilgiyi her iki karşılaştırıcıya ayrı olarak gönderir. Şekil4.4'deki karşılaştırıcılardan *karşılaştırıcı#1* sayılan değerin görev periyodu ile belirlenen değere eşit oluncaya kadar geçen zamanda çıkışa '1' veya dönüş yönüne göre '0' değerini vermesi için kullanılırken diğer sayıcı da frekans değerine eşit oluncaya kadar geçen zamanı belirleyerek DGM sinyallerinin periyodunu belirler. DGM sinyallerinin frekansı, istenilen değere eşit olduğunda 16-bitlik ileri sayıcı *karşılaştırıcı#2* tarafından resetlemektedir.



Şekil 4.4. SPKD'de tasarlanan DGM üretici.

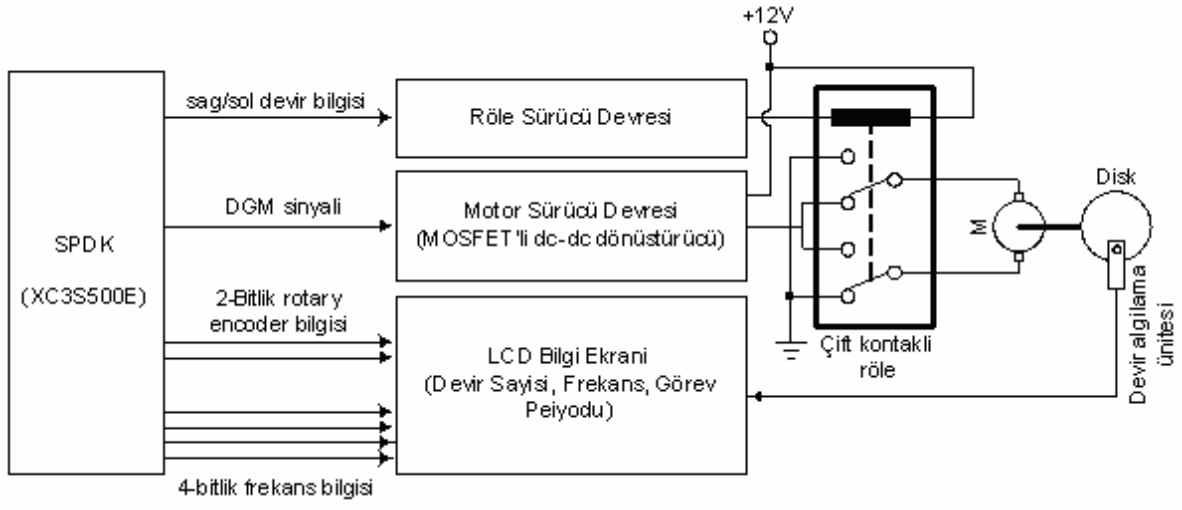
4.1.2. Deneysel ortam ve sonuçlar

Deneysel ortama ait blok diyagramı Şekil 4.5'de görüldüğü gibidir. Sistem, SPKD ile sürülen bir motor sürücü devresi, miline disk bağlı bir doğru akım motoru ve devir sayısı algılama düzeneğinden oluşmaktadır. Mutlak değerli döner algılayıcı ile görev periyodu frekansa bağlı olarak %5'lik adımlarla arttırılıp azaltılabilmekte ve değişim bir PIC16F877 tarafından sürülen bir LCD

ekran yardımıyla görsel olarak izlenebilmektedir. Mutlak değerli döner algılayıcı sağa sola dönebildiği gibi üzerine basıldığında da bir anahtar görevi görmekte olup motorun dönüş yönünü de değiştirilebilmektedir. Frekans değeri 4-bitlik bir sayısal bilginin SPKD'nin ilişkilendirilmiş pinlerine yaklaşık 3.3 V'luk gerilim uygulanması ile bir kod çözücü yardımıyla 1 kHz ve 16 kHz arasında bir tam değer olacak şekilde ayarlanabilmektedir.

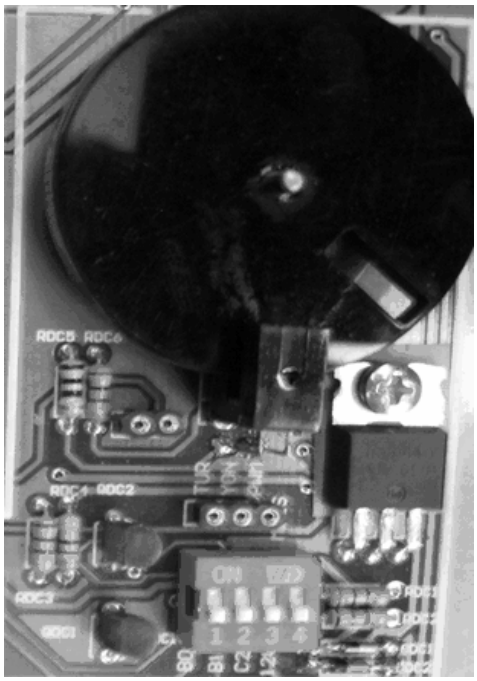
Frekans değişimleriyle birlikte görev periyodu sabit kalmaktadır. Örneğin, frekans 2 kHz ve görev periyodu %45 olduğu durumda frekans 6 kHz yapılsa da görev periyodu yine %45 olarak kalmaktadır.

Donanım bir yazılımla gerçekleştirildiği için, VHDL kodunda yapılacak birkaç küçük değişiklikle istenilen frekans aralığı ve görev periyodu değişim hassasiyeti kullanıcının isteklerine bağlı olarak rahatlıkla ayarlanabilir.



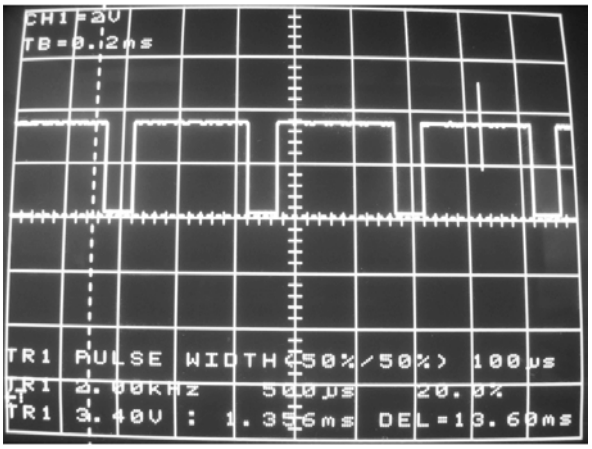
Şekil 4.5. Deneysel ortamın blok diyagramı.

Çalışmada, SPKD'de üretilen iki-kanallı DGM sinyalleri mosfet anahtarlı motor sürücü devresine uygulanmaktadır. Motor sürücü devresinin güç devresi; 11A/200V akım gerilim değerine sahip bir p-kanallı güç mosfetini (IRF9640) içeren dc-dc dönüştürücüden oluşmaktadır (Şekil 4.6). Motorun devir sayısı bilgisi, motor miline bağlı bir delikli disk ve bir kızılötesi (IR) emiter-detektör elemanı yardımıyla alınmış ve LCD ekranda görüntülenmiştir. Daha önceden belirlenen frekans ve görev periyodu değişkenleri de aynı zamanda bu ekran üzerinde bilgi amaçlı verilmiştir.

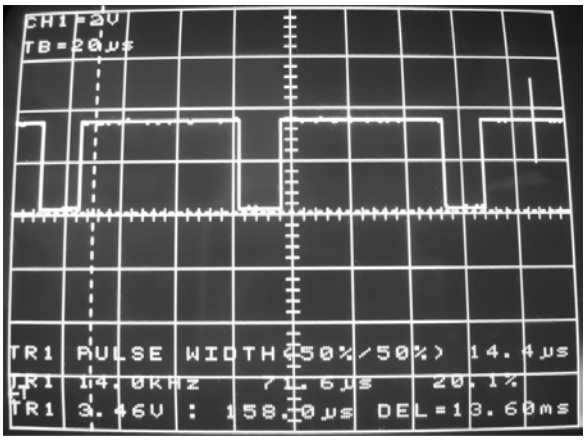


Şekil 4.6. Motor sürücü devresi ve devir sayısı ölçüm düzeneği.

Farklı anahtarlama frekansı ve farklı görev periyotları için SPKD çıkışından elde edilen DGM sinyallerinin dalga şekilleri Şekil 4.7'de verilmiştir. Elde edilen sinyaller oldukça düzgün olup, doğruluklarının da çok yüksek oldukları yapılan ölçümlerle tespit edilmiştir. SPKD çıkış pinleri 3.3 V'luk yüksek seviyeye ayarlanmıştır. Bu nedenle çıkış dalga şekilleri 0 ile 3.3 V arasında değişmektedir.



(a)



(b)

Şekil 4.7. %80 görev periyodu için üretilen DGM sinyalleri (a) 2 kHz (b) 14 kHz.

5. SONUÇLAR

DGM sinyallerinin üretimi SPKD'de sayısal olarak oluşturulması için bir yöntem sunulmuştur. Sayısal olarak üretilen bu sinyallerin, SPKD'de hem testleri hem de H-köprü dönüştürücü devresi ile beslenen bir doğru akım motorunun sürülmesi gerçekleştirilmiştir. Üretilen DGM sinyallerin doğruluklarının oldukça tatmin edici olduğu elde edilen sonuçlardan görülmüştür. Ayrıca SPKD'nin kaynaklarının %1 gibi oldukça küçük bir kısmı kullanılarak gerçekleştirilmek istenen sistem oluşturulmuştur.

SPKD ile yapılan tasarımların yüksek hız, düşük maliyet, tekrar programlanabilme gibi getirileri, sayısal sistemlerin tasarımında son yıllarda tercih edilmelerinin ana sebeplerindendir. Sayısal sistemlerin elektrik ve elektroniğin her alanında kullanımı hız etkeni bakımından oldukça önem kazanmıştır. Hızın yanı sıra gerçekleştirilen donanımların statik bir yapıda olması dışında artık dinamik bir yapıda da olması tercih sebebidir. Bu iki önemli gereksinimi sahada programlanabilir kapı dizileri önemli bir ölçüde karşılamaktadır.

Bu çalışmada da gerçekleştirilmek istenen donanım oldukça esnek bir yapıya sahiptir. Sadece SPKD'ye yüklenen kodlarda yapılacak olan birkaç değişiklik ile gerçekleştirilmek istenen donanım kısa bir zaman içerisinde elde edilebilmektedir. Mikrodenetleyiciler ile de benzer uygulamalar gerçekleştirilebilir fakat bu gibi uygulamalarda oldukça fazla çevre biriminin (LCD, devir sayısı algılayıcı, v.b) eş zamanlı kontrolünü gerektirdiğinden gerçek zamanlı bir uygulamadan uzak ve doğrulukları tartışmaya açık olacaktır.

Günümüzde kullanılan mikrodenetleyicilerin aksine komutları paralel işleme yeteneğine de sahip olduklarından SPKD ile gerçekleştirilen donanımların doğrulukları bu uygulamadan da görüldüğü üzere oldukça yüksek olmaktadır. Ayrıca, oldukça yüksek hızlarda anahtarlama sinyalleri, donanım gerçekleştirme sorunu olmadan birkaç kod dizisi üretimi ile gerçekleştirilebilmektedir.

Gerçekleştirilmek istenen donanım ne kadar karmaşık olursa olsun, donanım belirli kod dizileri ile ifade edildiği için gerçekleştirme, doğruluk ve hız olarak oldukça tatmin edici sonuçlar verecektir.

EK-A SPKD'ye yüklenen VHDL programı

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

entity pwm_gen is
   Port (     pwm_out1 : out std_logic;
              pwm_out2 : out std_logic;
              tour :  in std_logic;
              rotary_a: in std_logic;
              rotary_b: in std_logic;
              rotary_c : in std_logic;
              lcd_rs : out   std_logic;
              lcd_rw : out   std_logic;
              lcd_enable : out   std_logic;
              lcd_data : inout std_logic_vector(7 downto 0);
              freq_in  : in std_logic_vector (3 downto 0);
              clk : in std_logic
);
   end pwm_gen;
architecture mimari of pwm_gen is
signal      rotary_a_in     : std_logic;
signal      rotary_b_in     : std_logic;
signal      rotary_in       : std_logic_vector(1 downto 0);
signal      rotary_q1       : std_logic;
signal      rotary_q2       : std_logic;
signal     delay_rotary_q1 : std_logic;
signal      rotary_event    : std_logic;
signal      rotary_left     : std_logic;

SHARED VARIABLE freq : INTEGER RANGE 0 TO 50000;
```

```
SHARED VARIABLE duty1 : INTEGER RANGE 0 TO 50000;
SHARED VARIABLE duty_coef : INTEGER RANGE 0 TO 2500;
begin
 say: process(clk)
 VARIABLE temp : INTEGER RANGE 0 TO 50000;   --Maksimum 22Khz DGM periyodu
 VARIABLE temp2 : INTEGER RANGE 0 TO 20;   --Maksimum 22Khz DGM periyodu
  begin
        if clk'event and clk='1' then
  CASE freq_in IS
                WHEN "0000" => freq:=50000; --1KHz
                               duty_coef:=2500;
                WHEN "0001" => freq:=25000; --2KHz
                               duty_coef:=1250;
                WHEN "0010" => freq:=16680; --2.99KHz
                               duty_coef:=834;
                WHEN "0011" => freq:=12500; --4KHz
                               duty_coef:=625;
                WHEN "0100" => freq:=10000; --5KHz
                               duty_coef:=500;
                WHEN "0101" => freq:=8340;  --5.99KHz
                               duty_coef:=417;
                WHEN "0110" => freq:=7140;  --6.99KHz
                               duty_coef:=357;
                WHEN "0111" => freq:=6260;  --8KHz
                               duty_coef:=313;
                WHEN "1000" => freq:=5560;  --9.01KHZ
                               duty_coef:=278;
                WHEN "1001" => freq:=5000;  --10KHz
                               duty_coef:=250;
                WHEN "1010" => freq:=4540;  --11KHZ
                               duty_coef:=227;
                WHEN "1011" => freq:=4160;  --12KHZ
                               duty_coef:=208;
```

```
                WHEN "1100" => freq:=3840;  --13KHZ
                               duty_coef:=192;
                WHEN "1101" => freq:=3580;  --14KHZ
                               duty_coef:=179;
                WHEN "1110" => freq:=3340;  --15Khz
                               duty_coef:=167;
                WHEN "1111" => freq:=3120;  --16Khz
                               duty_coef:=156;
                when others => freq :=freq;
                               duty_coef:=duty_coef;
      END CASE;
rotary_a_in <= rotary_a;
rotary_b_in <= rotary_b;
rotary_in <= rotary_b_in & rotary_a_in;

                case rotary_in is
  when "00" => rotary_q1 <= '0';
            rotary_q2 <= rotary_q2;
  when "01" => rotary_q1 <= rotary_q1;
            rotary_q2 <= '0';
  when "10" => rotary_q1 <= rotary_q1;
            rotary_q2 <= '1';
  when "11" => rotary_q1 <= '1';
            rotary_q2 <= rotary_q2;
  when others => rotary_q1 <= rotary_q1;
              rotary_q2 <= rotary_q2;
end case;

delay_rotary_q1 <= rotary_q1;
if rotary_q1='1' and delay_rotary_q1='0' then
  rotary_event <= '1';
  rotary_left <= rotary_q2;
 else
```

```
      rotary_event <= '0';
      rotary_left <= rotary_left;
    end if;

    if rotary_event='1' then
      if rotary_left='1' AND temp2/=1 then
                    temp2:=temp2-1;
                        elsif rotary_left/='1' AND temp2/=20 then
                        temp2:=temp2+1;
                        else
                       end if;
      end if;
          duty1:=duty_coef*temp2;
  temp := temp + 1;
                IF (temp<=duty1) THEN pwm_out<='1';
                        ELSIF (temp>duty1 AND temp<freq) THEN pwm_out<='0';
                        ELSE temp:=0;
                END IF;
          END IF;
 END PROCESS say;
END mimari;
```

Kaynaklar

1. Pedroni, V. A. (2004). Circuit Design with VHDL, MIT Press, 363s.
2. Dueck, R. K. (2000). Digital Design with CPLD Applications and VHDL, Thomson Delmar Learning, 837s.
3. Wang, Q., Zhang, D. (2008). All Digital DC/DC Converters on FPGA, International Conference on Intelligent Computation Technology and Automation (ICICTA), 11-15.
4. Yen, C. C., Wu, C. H. (2008). Implementation of the Programmable Low Power DC-DC Voltage Converter by FPGA, 3rd IEEE Conference on Industrial Electronics and Applications, ICIEA2008, 405-410.
5. Xinyu, X., Khambadkone, A. M., Oruganti, R. (2007). A Soft-Switched Back-to-Back Bi-directional DC/DC Converter with a FPGA based Digital Control for Automotive Applications, 33rd Annual Conference of the IEEE Industrial Electronics Society, IECON'07, 262-267.
6. Zhaojin, W., Weihai, C., Zhiyue, X., Jianhua, W. (2006). Analysis of Two-Phase Stepper Motor Driver Based on FPGA, IEEE International Conference on Industrial Informatics, 821-826.
7. Dufour, C., Belanger, J., Abourida, S., Lapointe, V. (2007). Real-time Simulation of Finite-Element Analysis Permanent Magnet Synchronous Machine Drives on a FPGA Card, European Conference on Power Electronics and Applications, 1-10.
8. Jeich, M., You, R. L., Ti, H. L., Ting, H. W. (2006). Realization of OFDM Modulator and Demodulator for DSRC Vehicular Communication System Using FPGA Chip, International Symposium on Intelligent Signal Processing and Communications, ISPACS'06, 477-480.
9. Birla, M. K. (2006). FPGA Based Reconfigurable Platform for Complex Image Processing, IEEE International Conference on Electro/information Technology, 204-209.
10. Jiang, R. M., Crookes, D. (2007). FPGA Implementation of 3D Discrete Wavelet Transform for Real-time Medical Imaging, 18th European Conference on Circuit Theory and Design ECCTD'07, 519-522.
11. Farouk, H. A., Saeb, M. (2005). An Improved FPGA Implementation of the Modified Hybrid Hiding Encryption Algorithm (MHHEA) for Data Communication Security, Proceedings Design, Automation and Test in Europe, 3, 76-81.

12. Zheng, Y., Sun, H., Jia, Q., Shi G. (2008). Kinematics Control for A 6-DOF Space Manipulator Based on ARM Processor and FPGA Co-processor, 6th IEEE International Conference on Industrial Informatics, INDIN'08, 129-134.
13. Fei, L., Lin, Y., Lei, H., Deming, C., Cong, J. (2005). Power Modeling and Characteristics of Field Programmable Gate Arrays, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(11), 1712-1724.
14. Dabdoub, M. R., Wunnava, S.V. (1996). VHDL: A Powerful Digital Design and Simulation Tool, Proceedings of the IEEE Bringing Together Education, Science and Technology, Southeastcon'96, 537-540.
15. Mekhilef, S., Rahim, N. A. (2002). Xilinx FPGA Based Three-Phase PWM Inverter and Its Application for Utility Connected PV System, Proceedings of IEEE TENCON'02, 2079-2082.
16. Tzou, Y. Y., Hsu, J. H. (1997). FPGA Realization of Space-Vector PWM Control IC for Three-Phase PWM Inverters, IEEE Transactions on Power Electronics, 12(6), 953-963.
17. Mekhilef, S., Masaoud, A. (2006). Xilinx FPGA Based Multilevel PWM Single Phase Inverter, IEEE International Conference on Industrial Technology, ICIT2006, 259-264.
18. Lai, Y. S., Shyu, F.S., Chang, Y. H. (2004). Novel Loss Reduction Pulsewidth Modulation Technique for Brushless DC Motor Drives Fed by MOSFET Inverter, IEEE Transactions On Power Electronics, 19(6), 1646-1652.
19. Pongiannan, R. K., Yadaiah, N. (2006). FPGA based Space Vector PWM Control IC for Three Phase Induction Motor Drive, IEEE International Conference on Industrial Technology, ICIT'06, 2061-2066.
20. Product Selection Guides. (2008), Xilinx Corp.
21. Ayasrah, O.A. Alukaidey, T. and Pissanidis, G., 2006, DSP based n-motor speed control of brushless dc motors using external FPGA design, IEEE International Conference on Industrial Technology, ICIT'2006.
22. Medany, W.M., 2008, FPGA remote laboratory for hardware e-learning courses, IEEE Region 8 International Conference on Computational Technologies in Electrical and Electronics Engineering, SIBIRCON 2008, 106-109.
23. Li, J., and Cheng, C.-K. (1995). Routability improvement using dynamic interconnect architecture, IEEE Symposium on FPGAs for Custom Computing Machines Proceedings, 61-67.

24. Tessier, R. Betz, V. Neto, D. Egier, A. and Gopalsamy, T. (2007). Power-Efficient RAM Mapping Algorithms for FPGA Embedded Memory Blocks, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26(2), 278-290.
25. Krasniewski, A. (1999). Application-dependent testing of FPGA delay faults. Proceedings 25th EUROMICRO Conference, 1, 260-267.
26. Knack, K. (1994). Panel: Design Automation Tools for FPGA Design. 31st Conference on Design Automation, 676 - 676.
27. Pedram, M., Nobandegani, B.S. and Preas, B.T. (2007). Design and analysis of segmented routing channels for row-based FPGA's. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13(12), 1470-1479.

ÖZGEÇMİŞ

Meral AKARÇAY TÜRK, Elazığ'da doğdu. İlk ve orta öğrenimini Elazığ'da tamamladı. Fırat Üniversitesi, Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği bölümünden 2004 yılında mezun oldu. 2005 yılından beri Fırat Elektrik dağıtım A.Ş.'de Elektrik-Elektronik Mühendisi olarak çalışmaktadır. Evli ve bir çocuk annesidir.