FPGA DENETİMLİ DÜŞÜRÜCÜ DA-DA DÖNÜŞTÜRÜCÜNÜN TASARIMI VE GERÇEKLEŞTİRİLMESİ

Aziz MAMUR

Yüksek Lisans Tezi

Elektronik ve Bilgisayar Eğitimi Anabilim Dalı

Danışman: Doç. Dr. Servet TUNCER

AĞUSTOS-2012

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

FPGA DENETİMLİ DÜŞÜRÜCÜ DA-DA DÖNÜŞTÜRÜCÜNÜN TASARIMI VE GERÇEKLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Aziz MAMUR

092131104

Tezin Enstitüye Verildiği Tarih: 02 Ağustos 2012
Tezin Savunulduğu Tarih: 23 Ağustos 2012

Tez Danışmanı : Doç. Dr. Servet TUNCER (F.Ü)
Diğer Jüri Üyeleri : Prof. Dr. Hanifi GÜLDEMİR (F.Ü)
Doç. Dr. Mustafa TÜRK (F.Ü)

AĞUSTOS-2012

ÖNSÖZ

Bu tez çalışmasında katkılarını esirgemeyen başta danışmanım Doç. Dr. Servet TUNCER'e, Yrd. Doç. Dr. Mehmet GEDİKPINAR'a, Yrd. Doç. Dr. Ayhan ALTINÖRS'e, Arş. Gör. Ümit BUDAK'a, Arş. Gör. Ömer Faruk ALÇİN'e, Aykut DİKER'e ve Fırat Üniversitesi Teknik Eğitim Fakültesi Elektronik ve Bilgisayar Eğitimi Bölümü öğretim elemanları ve çalışanlarına teşekkür ederim. Ayrıca bu çalışma sırasında bana sonsuz sabırlar gösteren sevgili ailemi, özellikle oğlum Berat Yiğit ve kızım Öykü Melek'i kocaman kucaklıyorum.

Aziz MAMUR

ELAZIĞ - 2012

İÇİNDEKİLER

Sayfa No

ÖZET

Bu tez çalışmasında, kararlı ve yük değişimlerinden etkilenmeyen bir çıkış gerilimi üreten düşürücü DA-DA dönüştürücü devresi tasarlanmış ve gerçekleştirilmiştir. Bu amaçla, paralel çalışabilmesi ve programlama üstünlüklerinden dolayı, denetleyici ve PWM sinyal üreteci olarak Alan Programlanabilir Kapı Dizileri (FPGA) tercih edilmiştir. FPGA olarak, Xilinx üretici firmasına ait XC3S500E-4FG320 FPGA elemanı kullanılmıştır. Bu FPGA'la 24.4kHz frekansa sahip ve değişken görev periyotlu PWM sinyali üretilmiş ve IGBT anahtarın sürücü devresine uygulanmıştır. FPGA'nın programlanması için donanım tanımlama dili olarak, Çok Yüksek Hızlı Tüm Devre Donanım Tanımlama Dili (VHDL) kullanılmıştır. VHDL dili kullanılarak hem Spartan-3E Starter Kiti üzerindeki ADC modülü kontrol edilmiş hem de çıkış akım ve voltaj bilgisi alınarak sayısal denetim sağlanmıştır.

Anahtar Kelimeler: Düşürücü DA-DA Dönüştürücü, PWM, FPGA, VHDL

SUMMARY

DESIGN AND IMPLEMENTATION OF FPGA CONTROLLED DC-DC BUCK CONVERTER

In this thesis, a DC-DC buck converter is designed and implemented that its output voltage is stable and not affected by load variations. With this aim, Field Programmable Gate Arrays (FPGA) is chosen for controller and PWM signal generator due to the programming advantages and parallel operation. The XC3S500E-4FG320 is used as FPGA. With this FPGA, PWM signal which has 24.4kHz and adjustable duty cycle is generated and applied to IGBT drive circuit. A Very High Speed Integrated Circuit Hardware Description Language (VHDL) is used for programming the FPGA. In this way, both ADC module in Spartan-3E Starter kit is controlled and digital control is provided by taking current and voltage information.

Key Words: DC-DC buck converter, PWM, FPGA, VHDL

ŞEKİLLER LİSTESİ

Sayfa No

TABLOLAR LİSTESİ

Sayfa No

KISALTMALAR

ADC : Analog Dijital Çevirici
CCM : Sürekli Akım Modu
CLB : Düzenlenebilir Lojik Blok
CPLD : Karmaşık Programlanabilir Lojik Aygıt
DAC : Dijital Analog Çevirici
DSP : Dijital Sinyal İşlemcisi
DCM : Süreksiz Akım Modu
EEPROM : Elekriksel EPROM
EMI : Elektro Magnetik Girişim
EPROM : Silinebilir PROM
ESL : Eşdeğer Seri Endüktans
ESR : Eşdeğer Seri Direnç
FF : Flip-Flop
FPGA : Alan Programlanabilir Kapı Dizileri
HDL : Donanım Tanımlama Dili
IGBT : Kapıdan İzoleli Bipolar Transistor
IOB : Giriş Çıkış Blokları
LCD : Sıvı Kristal Ekran
LRM : Dil Referans Kitabı
LUT : Başvuru Çizelgesi
MUX : Çoklayıcı
PI : Oransal İntegral
PLD : Programlanabilir Mantık Aygıtı
PWM : Darbe Genişlik Modülasyonu
RAM : Rastgele Erişebilir Hafıza
SRAM : Statik RAM
VHDL : Çok Yüksek Hızlı Tüm Devre Donanım Tanımlama Dili

SEMBOLLER LİSTESİ

D : Darbe görev oranı

e : Hata

fc : Filtre köşe frekansı

fs : Anahtarlama frekansı

Ic : Kondansatör akımı

Iin : Giriş akımı

I_L : Ortalama bobin akımı

Io : Çıkış akımı

$I_{o.min}$: Çıkış akımının minimum değeri

Ki : İntegral katsayı

Kp : Oransal katsayı

t : Zaman

t_{on} : İletim süresi

t_{off} : Kesim süresi

Ts : Toplam periyot

Vin : Giriş gerilimi

V_L : Endüktans gerilimi

Vo : Çıkış gerilimi

ΔI_L : Çıkıştaki bozulmalara sebep olan bobin akımı

ΔIo : Çıkış akımının en alt ve üst seviyesindeki farkı

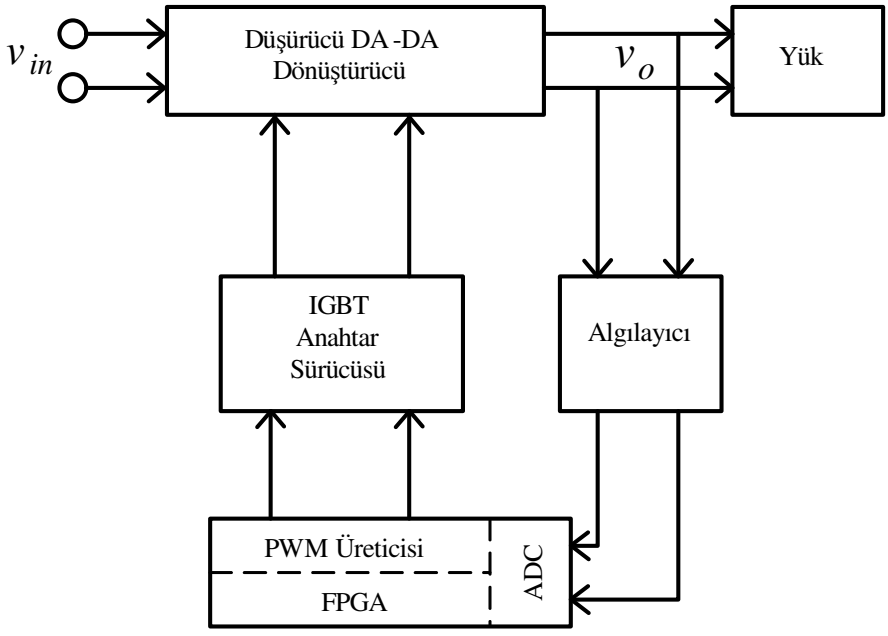
ΔQ : Bobin akımının kondansatörden dolayı oluşturduğu yük

ΔVo : Çıkış geriliminin en alt ve üst seviyesindeki farkı

1. GİRİŞ

Güç elektroniği terimi çok geniş bir alanda elektronik devreleri içine almaktadır. 1950'lerden bu yana yarıiletken güç elemanlarının icat edilmesiyle güç elektroniği; haberleşme, ulaşım, aydınlatma, otomotiv, ısıtma, uzay sistemleri, kesintisiz güç kaynağı, motor kontrolü gibi birçok alanda uygulama bulmuştur.

Günümüz askeri uygulamalarında yüksek enerji verimliliğine sahip, küçük boyut ve hacimli, gerilim regülasyonu oldukça düzgün ve dinamik cevabı iyi olan, esnek denetim yapısı ile haberleşmeye uyumlu güç kaynaklarının kullanımı önem kazanmaktadır. Bu önem özellikle saha uygulamaları ve havacılık sektöründe, sınırlı güç kapasitesi ile yüksek teknolojili cihazların oldukça dar besleme limitlerinde sorunsuz çalışmalarını sağlamak gereksiniminden kaynaklanmaktadır [1,2]. Bu nedenle bu çalışmada gelişen yarıiletken ve dijital kontrol teknolojisi ile yüksek verimlilikte ve esnek yapıda güç kaynakları tasarlamak amacıyla dijital denetimli düşürücü bir dönüştürücü tasarımı amaçlanmıştır (şekil 1.1).



Şekil 1.1. Sistemin blok şeması

Alan Programlanabilir Kapı Dizileri (FPGA), mantık devrelerinin programlama yardımıyla gerçekleştirilmesini sağlayan sayısal devrelerdir [3,4]. Programlama ile FPGA matrissel şekilde bulunan mantık modülleri ve ara bağlantı elemanları istenen amaç doğrultusunda düzenlenebilmektedir. FPGA tasarımcının ihtiyaç duyduğu mantık işlevlerini gerçekleştirme amacına yönelik olarak üretilmiştir [5]. Dolayısıyla her bir mantık bloğunun işlevi kullanıcı tarafından düzenlenebilmektedir. FPGA ile temel mantık kapılarının ve yapısı daha karmaşık olan devre elemanlarının işlevselliği artırılmaktadır. FPGA'nın üstünlüğü, mantık bloklarının ve ara bağlantıların imalat sürecinden sonra programlanabilmesidir. Bu üstünlük FPGA'nın günümüzde çok tercih edilmesinin nedenidir. Paralel çalışabilme ve programlama üstünlüklerinden dolayı bu çalışmada tercih edilmiştir.

FPGA davranışını tasarlamak için kullanıcı, donanım tanımlama dili (HDL-Hardware Design Language) ya da şematik tasarım tekniklerinden birini kullanmalıdır. Yaygın HDL'ler VHDL ve Verilog'tur [3,4,6]. Bu çalışmada FPGA yüklenen programlar için VHDL dili kullanılmıştır.

M. Milanovic ve arkadaşları (2005) tarafından gerçekleştirilen çalışmada, PI (oransal integral) denetleyicisi kullanılarak düşürücü DA-DA dönüştürücünün FPGA ile gerçekleştirilmesi ve PI denetleyicinin çıkış akım ve gerilimini yeterli seviyede denetleyebildiğini deneysel sonuçlar ile gösterilmiştir [7].

J. Alvarez ve arkadaşları (2006) sundukları çalışmada, bir bulanık (Fuzzy) denetleyici kullanarak bir otomobil için gerekli olan DA-DA dönüştürücüyü FPGA kullanarak gerçekleştirmiştir [8].

C. C. Yen vd. (2008) tarafından yapılan çalışmada, bir entegre içerisinde çeşitli gerilim değerlerinde ve güç değerlerinde çeşitli kısımlar tanıtılarak, her birim için ayrı DA-DA dönüştürücü kullanılmış ve denetimi bir FPGA üzerinden yapılmıştır [9].

M. Gamal vd. (2009) tarafından yapılan çalışmada, FPGA tabanlı senkron anahtarlamalı DA-DA dönüştürücü gerçekleştirilmiştir [10].

T. Iida (2009) tarafından yapılan çalışmada, FPGA kullanarak akım modlu kontrol sistemi, DA-DA dönüştürücü için gerçekleştirilmiştir [11].

Günümüz teknolojisinde FPGA kullanarak çok büyük miktarlardaki mantık devreleri yapılarak daha düşük maliyetli ve birleşik tasarımlı arabirimler üretilebilir. Bu çalışmada bu teknolojiden yararlanarak düşürücü DA-DA dönüştürücünün denetiminde

kullanılmıştır. FPGA'lar özellikle eşzamanlı yapılması gereken işlemler için çok uygun olmaktadır.

Bu tez çalışmasının amacı, yaygın olarak kullanılan düşürücü DA-DA dönüştürücü tasarlamak ve uygulama devresini gerçekleştirmektir. FPGA kullanarak, anahtarlama elemanı için gerekli olan PWM sinyal üretilmiş ve dönüştürücünün denetimi sağlanmıştır.

Bu tez çalışması genel olarak şu bölümlerden oluşmaktadır. İkinci bölümde, anahtarlamalı DA-DA dönüştürücü mimarileri incelenmiştir. Üçüncü bölümde, FPGA yapısı, çalışması ve programlanması hakkında genel bilgiler verilmiştir. Dördüncü bölümde, tercih edilen donanım tanımlama dili ile ilgili bilgiler verilmiştir. Beşinci bölümde, yazılım geliştirme ortamı (Xilinx ISE 9.2i) gerçekleştirilmesi için gereken adımlar verilmiştir. Altıncı bölümde, düşürücü DA-DA dönüştürücü tasarlanmış ve gerçekleştirilmiştir. Yedinci bölümde, Xilinx üretici firmasına ait XC3S500E-4FG320 FPGA elemanı kullanılarak, 24.4kHz frekanslı PWM sinyali üretilmiştir. Düşürücü DA-DA dönüştürücü uygulaması ve denetimi gerçekleştirilmiştir. Sonuç bölümünde ise elde edilen veriler değerlendirilmiştir.

2. ANAHTARLAMALI DA–DA DÖNÜŞTÜRÜCÜLER

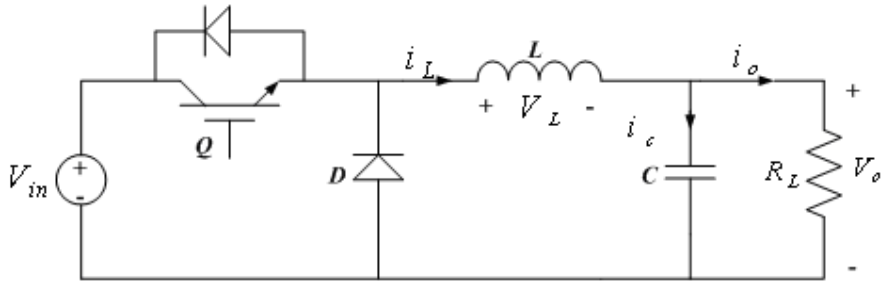
Anahtarlamalı Doğru Akım-Doğru Akım (DA-DA) dönüştürücüleri, anahtarlama tekniği ile bir seviyedeki gerilimi başka bir seviyeye çevirmek için kullanılan güç elektroniği sistemleridir [1,12-16,]. Yüksek verimlilikleri ve küçük boyutları sayesinde günümüzde oldukça popülerdirler. Anahtarlamalı DA-DA dönüştürücüleri bilgisayarlarda ve her türlü cihaz adaptörlerinde sıklıkla kullanılırlar.

Anahtarlamalı DA–DA dönüştürücüler, kontrollü bir yarı iletken güç elemanı, bir diyot ve bobinden oluşan üç temel elemanın farklı şekillerde bağlanmasıyla elde edilmiştir. Devrede bulunan yarı iletken güç elemanı bir anahtar gibi ya doyum bölgesinde ya da kesim bölgesinde çalıştırılır.

Anahtarlamalı DA–DA dönüştürücülerin çalışma prensibi, anahtarlanan endüktansın enerji aktarımına dayalıdır. Girişten çıkışa enerji ayrık paketler halinde enerji depolayan endüktans ve kapasitans elemanları kullanılarak aktarılır. Bu dönüştürücülerde, bir anahtarlama periyodu boyunca ya anahtarlama elemanı ya da diyot iletimdedir. Anahtar iletimde iken endüktansa aktarılan enerji, diyot iletimde iken çıkışa aktarılır [1,17].

2.1. Düşürücü DA-DA Dönüştürücü Mimarisi

Düşürücü DA-DA dönüştürücü devre mimarisi şekil 2.1.'de gösterilmiştir. Burada; *Vin* giriş gerilimini, *L* endüktansı, V_L endüktans gerilimini, *Q* anahtarlama elemanı (IGBT), *D* diyotu, *C* kapasitörü, R_L yük direncini ve *Vo* çıkış gerilimini ifade etmektedir. Burada *L* endüktansı ve *C* kapasitörü birlikte alçak geçiren bir filtre işlevi görür.



Şekil 2.1. Düşürücü DA-DA dönüştürücü devre şeması

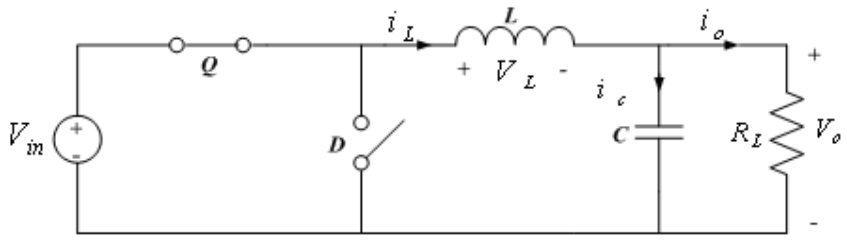
Düşürücü DA-DA dönüştürücü, çıkış gerilimi olarak giriş geriliminden daha düşük seviyede gerilim elde etmek için kullanılan, izole olmayan ve giriş-çıkış gerilim polariteleri aynı olan bir anahtarlamalı DA-DA dönüştürücüdür. En yaygın olarak kullanıldığı yerler, regüle edilmiş DA güç kaynakları, akü şarj sistemleri ve DA motor hız kontrol devreleridir [1]. Bu çalışmada düşürücü DA-DA dönüştürücü ideal bileşenlerden oluştuğu varsayılmıştır.

Şekil 2.1'de Q anahtarlama elemanı çeşitli kontrol teknikleriyle doyum ve kesim bölgesinde çalıştırılır. Doyumda olduğu süre boyunca girişteki enerji çıkışa aktarılır, kesimde olduğu sürede ise endüktansa aktarılarak depolanan enerji çıkışı beslemeye devam eder. Q anahtarlama elemanının doyumda kalma süresi çıkışa aktarılan enerjinin miktarını belirlediği göz önüne alındığında, Q anahtarlama elemanının doyumda tutulduğu süre çıkış geriliminin seviyesini belirler. Denklem 2.1 anahtarlama elemanının doyumda tutulduğu sürenin (ton), tüm periyot süresine (Ts) ve D çalışma oranını verir [1].

$$D = \frac{t_{on}}{T_s} \qquad 0 \le D \le 1 \tag{2.1}$$

2.2. Dönüştürücünün Çalışma İlkeleri ve Analizi

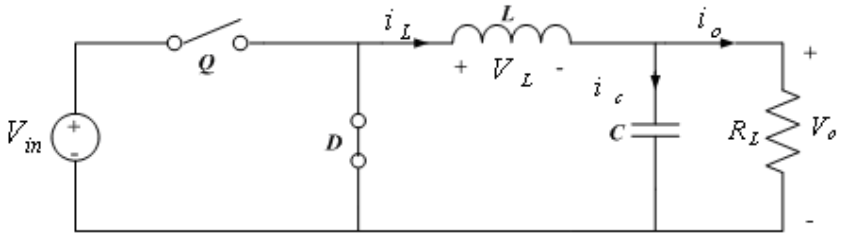
Q anahtarı doyum bölgesinde, diyot kesimde iken oluşan eşdeğer devre şekil 2.2'de verilmiştir. Bu durumda, endüktans üzerindeki gerilim, denklem 2.2'de gösterildiği gibi giriş gerilimi ile çıkışta oluşan gerilimin farkına eşit olur. Giriş gerilimi ve çıkış geriliminin sabit olduğu farz edildiği ideal durumda, endüktansın üzerinden akan akım doğrusal olarak artarak şekil 2.4'te gösterildiği gibi 0-DTs süresi sonunda maksimum değere ulaşır. Ayrıca, akan akımdan dolayı endüktans üzerinde enerji yüklenmesi gerçekleşir.



Şekil 2.2. Q anahtarı doyum bölgesinde çalışıyorken düşürücü dönüştürücü eşdeğer devresi

$$v_L = v_{in} - v_o \qquad 0 \le t \le DT_s \tag{2.2}$$

Q anahtarı kesimde, diyot iletimde iken oluşan eşdeğer devre şekil 2.3'te verilmiştir. Bu durumda, denklem 2.3'te gösterildiği gibi endüktans üzerindeki gerilim, çıkış geriliminin tersine eşittir. Anahtar kesime geçtiğinde bobinde birikmiş enerji nedeniyle endüktans akımı akmaya devam eder. Ancak endüktans üzerindeki gerilimin negatif polarite de olması nedeniyle akım azalarak akar. Şekil 2.4'te gösterildiği gibi DTs-Ts aralığında akım minimum değere ulaşır.

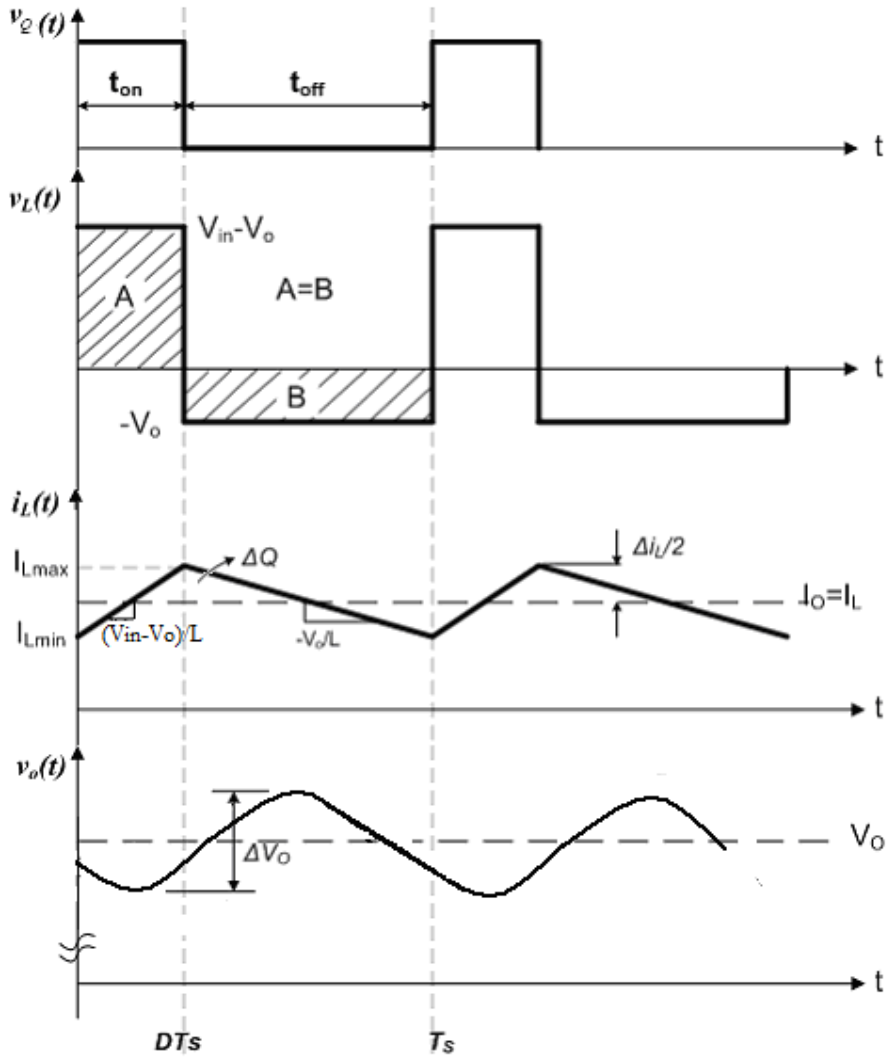


Şekil 2.3. Q anahtarı kesim bölgesinde çalışıyorken düşürücü dönüştürücü eşdeğer devresi

$$v_L = -v_o \qquad DT_s \le t \le T_s \tag{2.3}$$

Anahtarlama elemanının açık ve kapalı olması durumlarında oluşan eşdeğer devrelere göre, devre elemanları üzerinde oluşan akım - gerilim dalga şekilleri şekil 2.4'te verilmiştir.

Sürekli çalışma durumunda dalga şekli bir peryottan diğerine tekrar etmek zorunda olduğundan, endüktansın gerilimin bir peryottaki integrali sıfır olmak zorundadır. Bu durumunda endüktansın üzerindeki ortalama gerilim sıfıra eşit olacağından şekil 2.4'te endüktans geriliminin dalga şeklinde gösterilen A ve B alanlarının eşit olması gerekir (denklem 2.4).



Şekil 2.4. Düşürücü DA-DA dönüştürücü temel dalga şekilleri

Şekil 2.4'te; V_Q**;** Q anahtarı kapı sinyalini, V_L**;** endüktans gerilimini, i_L**;** endüktans akımını, Vo**;** çıkış gerilimini göstermektedir.

$$(V_{in} - V_0)DT_s = V_0(T_s - DT_s) \tag{2.4}$$

$$D = \frac{V_o}{V_{in}} \tag{2.5}$$

Denklem 2.4 üzerinde gerekli sadeleştirmeler yapıldığında, düşürücü DA-DA dönüştürücünün ortalama giriş ve çıkış gerilimleri arasındaki bağıntı bulunur (denklem 2.5). Görüldüğü üzere D çalışma oranı çıkış geriliminin giriş gerilimine oranıdır. D çalışma oranı 0 ile 1 arasında bir değer olduğundan denklem 2.1, çıkış gerilimi giriş geriliminden her zaman küçüktür, düşürücü DA-DA dönüştürücünün genel özelliğidir.

Endüktans akımındaki dalgalanma istenen bir durum değildir. Bu sebeple dalgalanma miktarının sınırlandırılması gerekmektedir ve bu dalgalanma miktarı, dönüştürücü devresindeki endüktansın değerinde önemli rol oynar. Endüktans değeri ile endüktans akımındaki dalgalanma arasındaki bağıntı denklem 2.6'da verilmiştir [1].

$$i_L(t) = \frac{1}{L}\int v_L(t)dt \tag{2.6}$$

Buradan endüktans akımındaki dalgalanmanın şekil 2.4'teki endüktans gerilim (V_L) eğrisi altındaki alanın endüktans değerine oranı ile belirlendiği görülmektedir. Buna göre endüktans akımındaki dalgalanma denklem 2.7 elde edilir.

$$\Delta i_L = \frac{1}{L}(V_{in} - V_o)DT_s = \frac{1}{L}V_o(1-D)T_s \tag{2.7}$$

Benzer şekilde, çıkış gerilimi üzerindeki dalgalanma da hiçbir dönüştürücü için istenen bir durum değildir. Çıkış gerilimindeki dalgalanma, kondansatör yardımıyla sınırlandırılır. Kondansatörün kapasitans değeri ile çıkış gerilimi üzerindeki dalgalanma miktarı arasındaki bağıntı yardımıyla (denklem 2.8) elde edilmiştir [1].

$$v_c(t) = \frac{1}{C}\int i_c(t)dt \tag{2.8}$$

Çıkış gerilimdeki dalgalanmalar uygulanabilir bir kapasite değeri için şekil 2.4'teki dalga şekilleri göz önüne alınarak sürekli akım iletim durumu için hesaplanabilir. Endüktans akımı i_L üzerindeki tüm dalgalanmanın kapasite üzerinden aktığı ve akımın ortalama bileşeninin de direnç üzerinden yüke aktığını varsayarsak, ΔQ alanı ek yüklemeyi göstermektedir. Bu durumda iki tepe değer arası gerilim dalgalanması denklem 2.9'da verilmiştir [1].

$$\Delta v_o = \frac{\Delta Q}{C} = \frac{1}{C}\frac{1}{2}\frac{\Delta i_L}{2}\frac{T_s}{2} \tag{2.9}$$

2.3. Sürekli ve Süreksiz Akım Çalışma Modu

Düşürücü DA-DA dönüştürücü sürekli ve süreksiz akım modları olmak üzere iki modda çalışabilir. Anahtarlama elemanın kesimde olduğu sürede endüktans akımı sıfır seviyesine kadar düşer ise süreksiz akım modu (discontinuous current mode-DCM), düşmez ise sürekli akım modu (continuous current mode-CCM) oluşur. Bu iki durumun gösterdiği frekans domeni cevapları çok farklı olduğundan, çeviricinin sürekli olarak bu modlardan sadece birinde çalışması istenir. Denklem 2.10'da sürekli akım modu (CCM) ve denklem 2.11'de süreksiz akım modu (DCM) şartları verilmiştir [1,2].

$$\frac{\Delta i_L}{2} < I_{o\min} \tag{2.10}$$

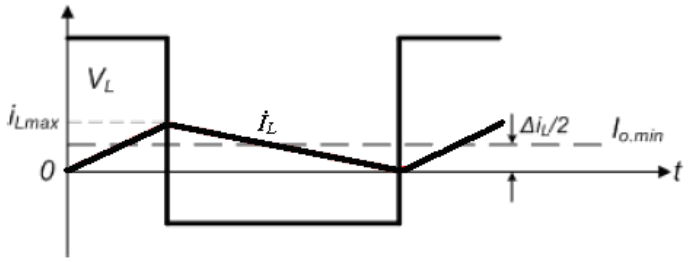
$$\frac{\Delta i_L}{2} > I_{o\min} \tag{2.11}$$

Şekil 2.4'te gösterildiği gibi, Δi_L endüktans akımı üzerindeki tepeden-tepeye dalgalanma miktarını ifade eder. Denklem 2.12'de endüktans akımı yükselirken, denklem 2.13'te endüktans akımı azalırkenki Δi_L değerlerini gösterir.

$$\Delta i_L = \frac{V_{in} - V_o}{L} DT_s \tag{2.12}$$

$$\Delta i_L = \frac{V_o}{L}(1-D)T_s \tag{2.13}$$

Sürekli iletim durumunun uç değerindeki V_L ve i_L şekil 2.5'te gösterilmiştir. Sınırda olmasından dolayı periyodun sonunda endüktans akımı sıfır olur. Şekil 2.5'te de görüldüğü gibi çıkış akımı *Io.min*'den daha küçük olursa sistem süreksiz akım durumuna geçer. Uygulamamızda sürekli akım modunda (CCM) çalışma yapılmıştır.



Şekil 2.5. Sürekli ve süreksiz akım durumu sınır değerleri

Dönüştürücünün sürekli akım modunda (CCM) çalışması için gerekli olan en düşük endüktans değeri denklem 2.14'te verilmiştir. Buna göre endüktans değerinin düşürülmesi için anahtarlama frekansının arttırılması gerektiği ortaya çıkar (denklem 2.15).

$$I_{o.\min} = \frac{\Delta i_L}{2} \Rightarrow I_{o.\min} = \frac{V_o(1-D)}{2L} T_s \tag{2.14}$$

$$L > \frac{V_o(1-D)}{2I_{o.\min} f_s} \tag{2.15}$$

2.4. Çıkış Gerilimindeki Dalgalanma

Doğrusal ve ideal devre elemanlarının kullanıldığı bir devrede, kondansatör içerisinden akan akım, her periyot boyunca kondansatör üzerinde *Q*'luk bir yük değişimine neden olur. Kondansatör üzerindeki gerilimin değişimi (dolayısı ile çıkış gerilimi üzerindeki gerilimin değişimi) yük değişiminin kapasitans değerine oranına eşittir. Burada kondansatör içinden akan akımdaki değişimin, endüktans içinden akan akım üzerindeki salınım ile aynı olduğu varsayıldığından çıkış voltajındaki salınım oranı denklem 2.16'da belirtildiği gibi şekil 2.4'teki ΔQ alanından hesaplanabilir[1].

$$\Delta v_o = \frac{1}{C}\left[\frac{1}{2}\frac{\Delta i_L}{2}\frac{T_s}{2}\right] = \frac{\Delta i_L}{8C}T_s \tag{2.16}$$

Buradan, denklem 2.13'deki Δi_L denklem 2.16'da yerine koyulursa denklem 2.17 elde edilir. Alçak geçiren filtrenin frekansı denklem 2.18'de ifade edilmiştir. Denklem 2.18 denklem 2.17'de yerine konulup gerekli sadeleştirmeler yapıldığında denklem 2.19 elde edilir.

$$\Delta v_o = \frac{T_s}{8C}\frac{V_o}{L}(1-D)T_s \tag{2.17}$$

$$f_c = \frac{1}{2\Pi\sqrt{LC}} \tag{2.18}$$

$$\frac{\Delta v_o}{V_o} = \frac{1}{8}\frac{(1-D)T_s^2}{LC} = \frac{\Pi^2}{2}(1-D)\left[\frac{f_c}{f_s}\right]^2 \tag{2.19}$$

Denklem 2.19'da görülebileceği gibi çıkış gerilimindeki dalgalanmalar alçak geçiren filtrenin (*fc)* köşe frekansı, anahtarlama frekansından (*fs)* çok küçük (*fc<<fs*) seçilerek azaltılabilir. Ayrıca dalgalanma, dönüştürücü sürekli akım durumunda çalıştığı sürece çıkış gücünden bağımsızdır [1].

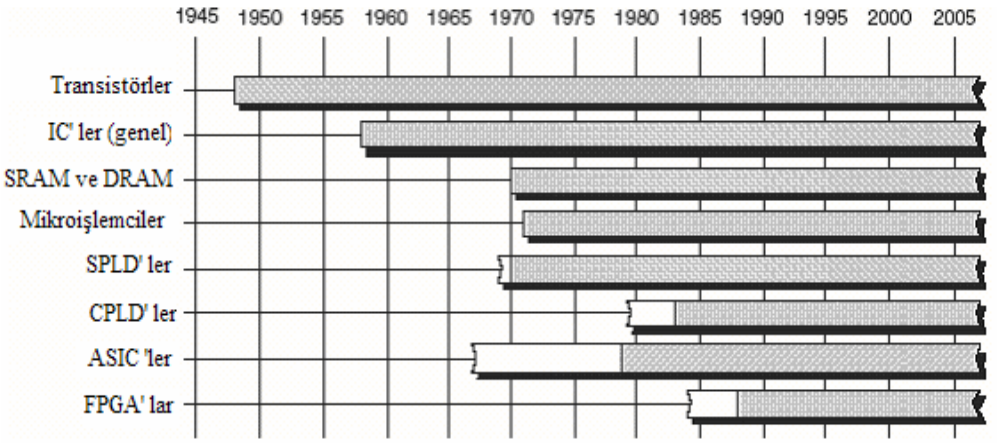
3. ALAN PROGRAMLANABİLİR KAPI DİZİSİ

Alan programlanabilir kapı dizisi (Field Programmable Gate Arrays), 1984 yılında Xilinx'in kurucusu Ross Freeman tarafından icat edilmiştir. FPGA mimari yapısı, tamamı kullanıcı tarafından belirlenebilen lojik bloklar ve ara bağlantılarından oluşmuştur. FPGA günümüzde 10 milyon eşdeğer kapı sayısına (iki girişli AND olarak) ulaşmıştır [18-22].

FPGA'lar programlanabilir mantık blokları ve bu bloklar arasındaki ara bağlantılardan oluşan ve geniş uygulama alanlarına sahip olan sayısal tümleşik devrelerdir. Tasarımcının ihtiyaç duyduğu mantık fonksiyonlarını gerçekleştirme amacına yönelik olarak üretilmiştir. Dolayısıyla her bir mantık bloğunun fonksiyonu kullanıcı tarafından düzenlenebilmektedir. FPGA ile temel mantık kapılarının ve yapısı daha karmaşık olan devre elemanlarının işlevselliği arttırılmaktadır. Alan programlanabilir kapı dizisi ismi verilmesinin nedeni, mantık bloklarının ve ara bağlantıların imalat sürecinden sonra programlanabilmesidir.

3.1. FPGA Teknolojisinin Gelişimi

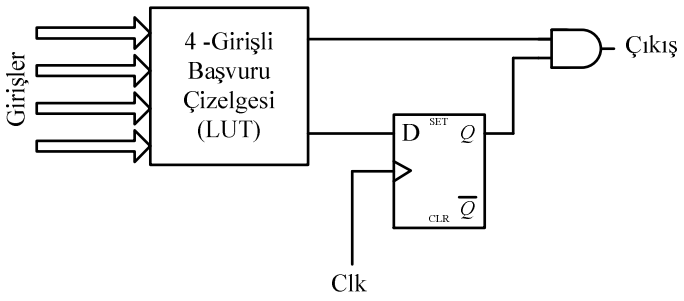
İlgili teknolojilerin kronolojik gelişimi şekil 3.1'de verilmiştir. Transistörler 1950'li yıllarda, mikroişlemciler 1970'li yıllarda hayatımıza girmiştir. Alan Programlanabilir Kapı Dizileri (FPGA) ise 1980'li yıllarda hayatımıza girerek teknolojinin her sahasında önemli yer edinmiştir [20].



Şekil 3.1. FPGA teknolojisinin gelişimi [20].

3.2. FPGA'nın İç Yapısı

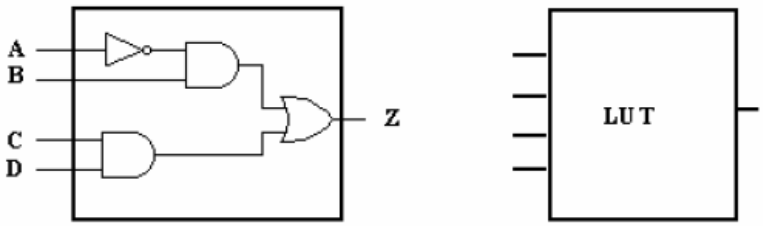
Tipik bir FPGA lojik bloğu şekil 3.2'de görüldüğü gibi 4 girişli Başvuru Tablosu (LUT, Look Up Table) ve bir flip-flop'tan oluşur [21].



Şekil 3.2. Bir FPGA mantık bloğu [21].

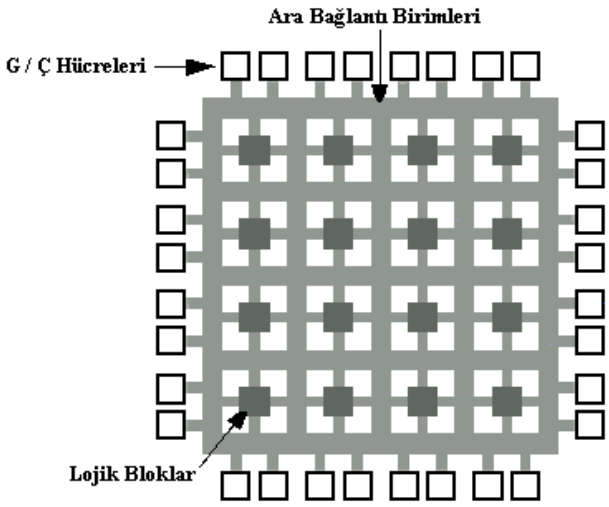
3.2.1. LUT Tabanlı Hücre

Bu yapıda giriş işaretleri başvuru tablosundan doğru çıkışı bulmak için işaretçi olarak kullanılır. Girişlerin alabileceği her değer için tabloda bir çıkış değeri bulunur. $Z = (\overline{A} \,\&\, B) \mid (C \,\&\, D)$ fonksiyonunu LUT tabanlı mimaride gerçekleştirilmesi için oluşturulan başvuru tablosu aşağıdaki gibi olmalıdır. Bu fonksiyon 4 girişli bir LUT ile gerçeklenebilmektedir şekil 3.3 [22].



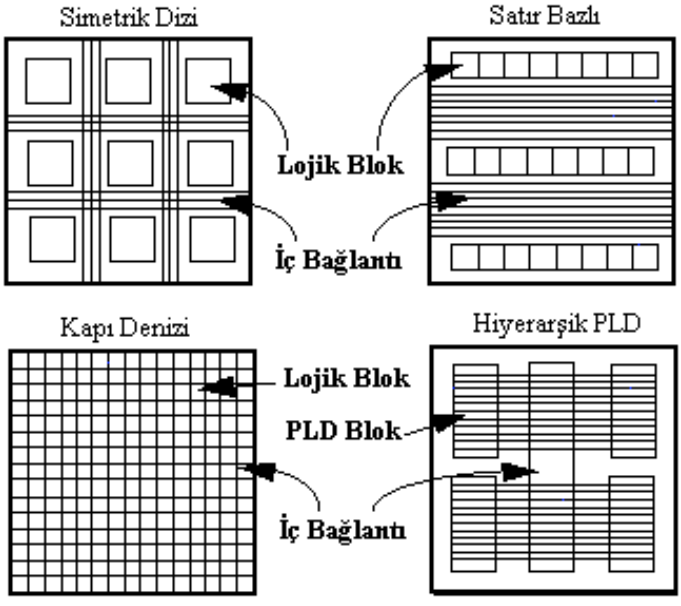
Şekil 3.3. LUT tabanlı hücre [22].

FPGA, düzenlenebilir lojik bloklar (configurable logic blocks, CLB), giriş / çıkış blokları (Input / Output Blocks, IOB) ve ara bağlantılar olmak üzere üç tane önemli düzenlenebilir elemana sahiptir. FPGA programlanması, ara bağlantıların nasıl olacağını belirleme ve FPGA bellek hücrelerinin programlanması işlemidir. Şekil 3.4'te FPGA'da bulunan lojik bloklar ve ara bağlantı birimleri görülmektedir [22-24].



Şekil 3.4. FPGA genel yapısı [22].

FPGA bağlantı çeşitlerine göre; simetrik dizi, sıra tabanlı, hiyerarşik PLD (Programmable Logic Device) ve kapı denizi olmak üzere dörde ayrılır şekil 3.5 [25].



Şekil 3.5. Bağlantı şekillerine göre FPGA [25].

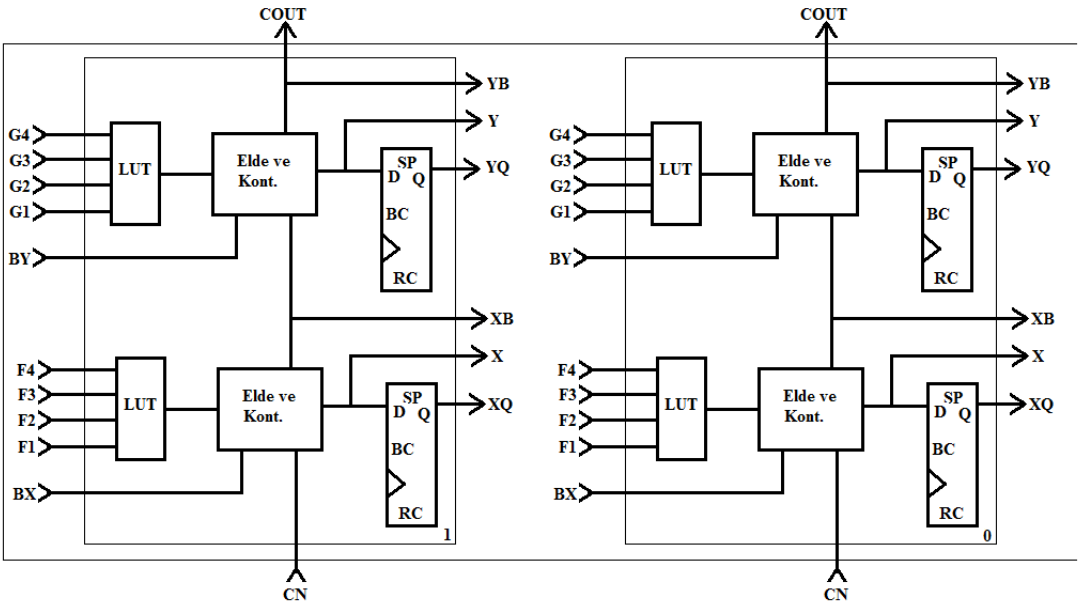
3.2.2. FPGA Mimarisi

FPGA mimarisi şekil 3.4'te gösterildiği gibi üç bileşenden oluşur [26].

- Düzenlenebilir lojik blok (Configurable Logic Blocks, CLB)
- Giriş/Çıkış blokları (Input/Output Blocks, IOB)
- Ara bağlantılar (Anahtarlama matrisi)

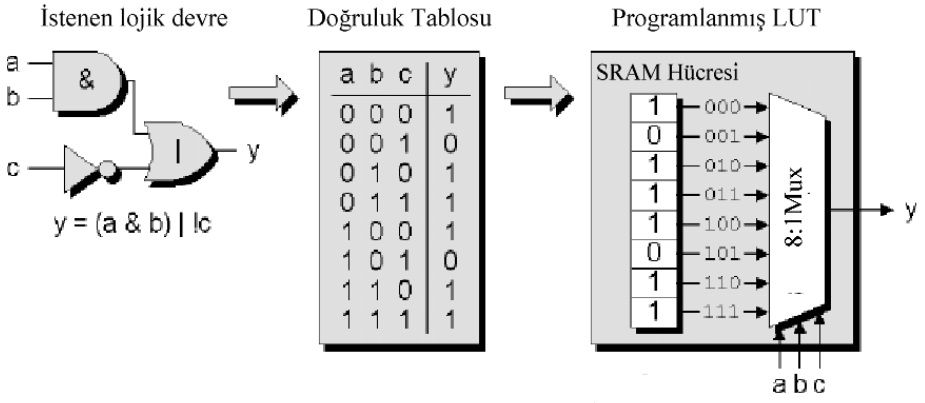
3.2.2.1. Düzenlenebilir Lojik Blok

CLB, tasarımcının oluşturmak istediği lojik devre için programlanabilen fonksiyonel aygıttır. FPGA'lar çok sayıda bu CLB'ler den oluşmaktadır. Yongadaki her lojik blok farklı bir fonksiyonu gerçekleştirmek için uygun SRAM (farklı bir teknolojide olabilir) programlama hücreleri vasıtasıyla yapılandırılabilir [19,25]. Şekil 3.6'da bir CLB bloğunun iç yapısı gösterilmiştir. Bir CLB bloğu bir çift Flip-Flop (FF) ve iki adet dört girişli fonksiyon üreticisi (üretici firmaya göre farklılık gösterebilir) içerir. Bu fonksiyon üreticileri dört girişten az olan fonksiyonları üretilebilme esnekliğine sahiptir. Bu bloklar FPGA yongaya uygulanacak olan lojik devrenin büyük bir kısmını oluşturur. CLB mimarisinin sahip olduğu esneklik ve simetri özellikleri uygulamaların kolaylıkla yerleştirilmesine ve yönlendirmesine olanak sağlar [25].



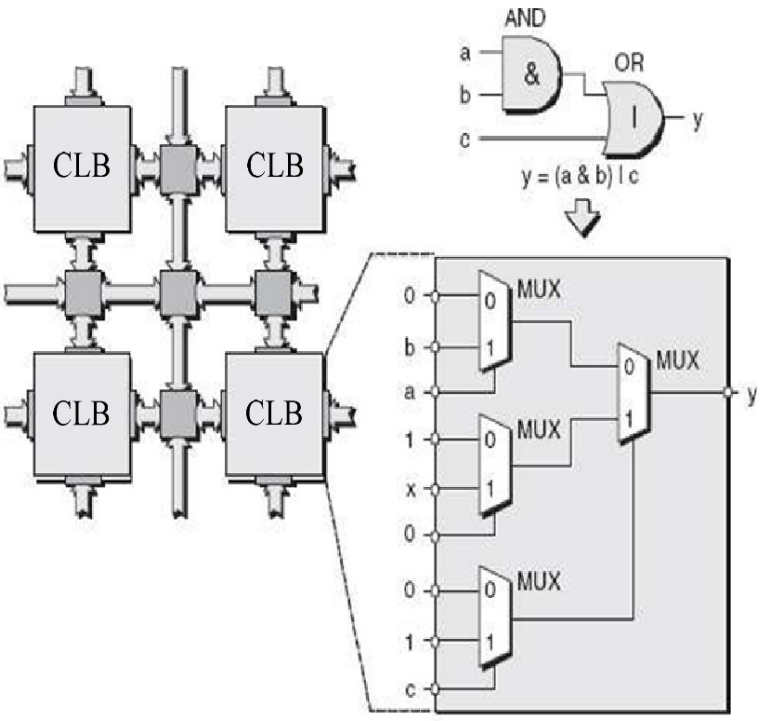
Şekil 3.6. CLB'nin iç yapısı [25].

CLB mimarisi; doğruluk tablosu (Look-Up Table, LUT) tabanlı ya da çoklayıcı (Mux) tabanlı yapıdan oluşurlar. Bazılarında ise ardışık devrelerin de gerçeklenebilmesi amacıyla FF kullanılmıştır [19,25]. LUT tabanlı yapının temel bloğu (3-6 arasında sabit sayı) değişkenli her boole fonksiyonunu gerçekleyebilen devredir. Şekil 3.7'de bir lojik fonksiyonun gerçeklenmesi için LUT' un yapılandırılması gösterilmiştir [19,25].



Şekil 3.7. LUT'un yapılandırılması [19].

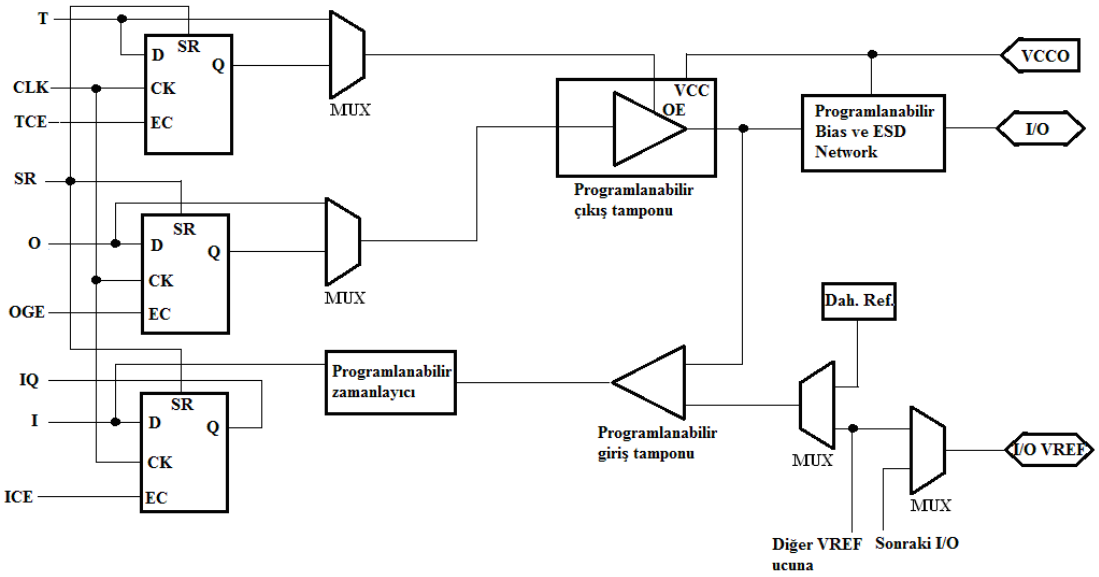
Çoklayıcı tabanlı yapının temel bloğu çeşitli yapılandırmalardan ve olabildiğince az lojik kapılardan (VE, VEYA) oluşur. Bu yapıdaki FPGA'ların içinde tutucu (latch) ve Flip-Flop (FF) bulunmadığından çoklayıcı kullanılarak CLB gerçeklenir. Şekil 3.8'de bir lojik fonksiyonun Mux tabanlı yapı ile gerçeklenmesi gösterilmiştir [19,27,28].



Şekil 3.8. Bir lojik fonksiyonun MUX tabanlı yapı ile gerçeklenmesi [19].

3.2.2.2. Giriş/Çıkış Birimi

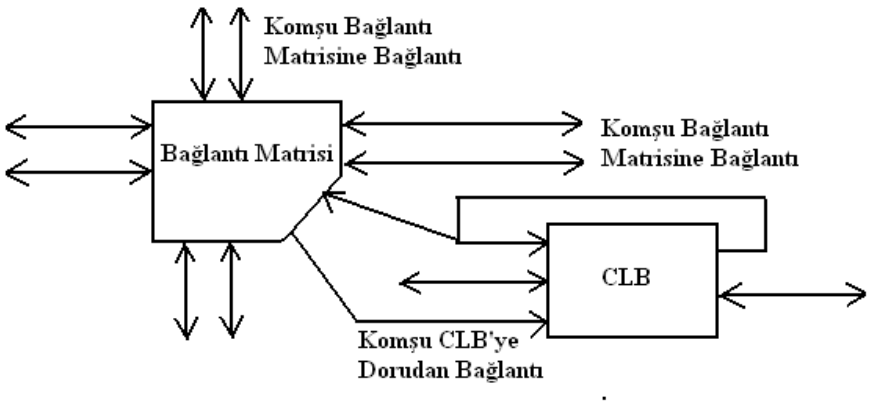
IOB'lar kılıf bacaklarıyla tasarım için kullanılan birimler (CLB, Blok RAM) arasında bağlantı kurar. FPGA'ların giriş-çıkış blokları; giriş, çıkış veya giriş-çıkış olarak 3 farklı şekilde tanımlanır [19,29]. Şekil 3.9'da Xilinx Firmasının ürettiği FPGA'ya ait giriş-çıkış bloklarının yerleşimi ve bir giriş-çıkış bloğunun ayrıntılı şeması gösterilmiştir [25,30].



Şekil 3.9. Xilinx FPGA IOB yapısı [25].

3.2.2.3. Ara Bağlantılar

IOB'lar ile istenen lojik devre için hazırlanan CLB'ler arasındaki bağlantılar, ara bağlantı aygıtları ile sağlanır. FPGA içerisindeki ara bağlantı aygıtları, CLB'ler arasına satırlar ve sütunlar halinde yerleştirilmiş bağlantı hatları ve bu hatların kesişim noktalarına yerleştirilmiş bağlantı matrislerinden oluşur. Şekil 3.10'da bir FPGA yongası içerisindeki bağlantı elemanları gösterilmiştir [19,29].



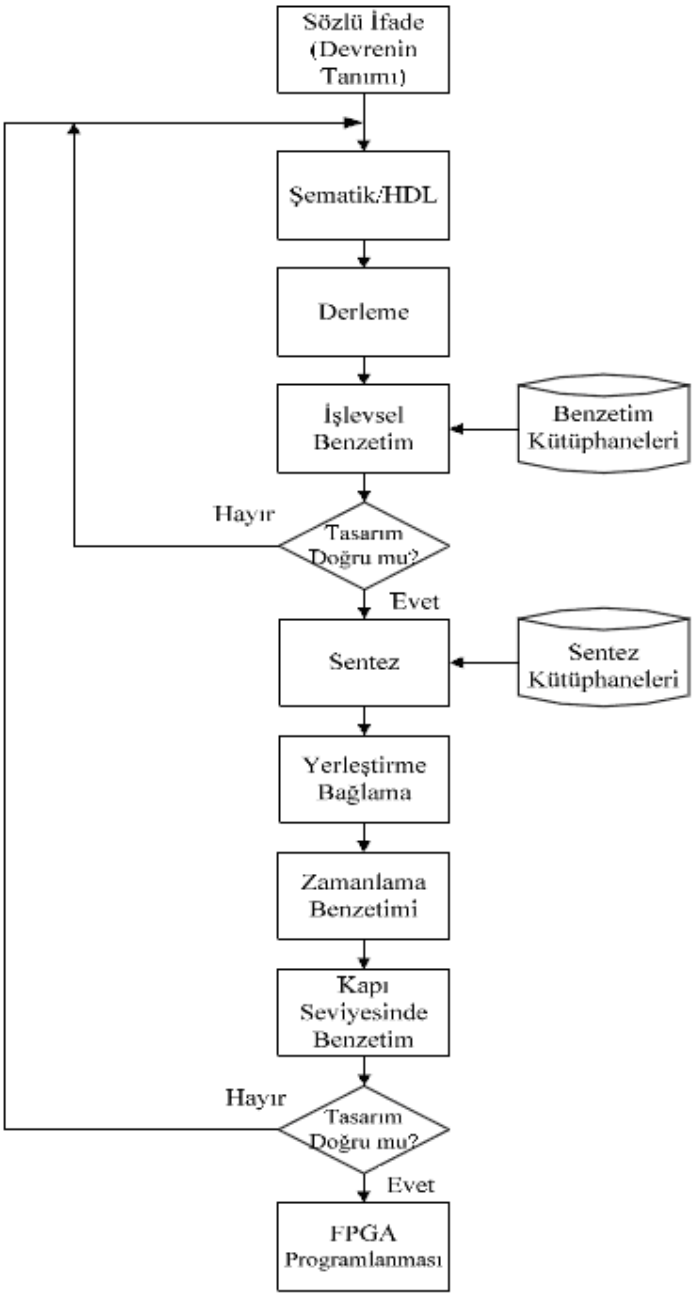
Şekil 3.10. FPGA ara bağlantı birimi [19].

3.3. FPGA Programlanması

Bir FPGA programlanmasında aşağıdaki adımlar izlenir [19].

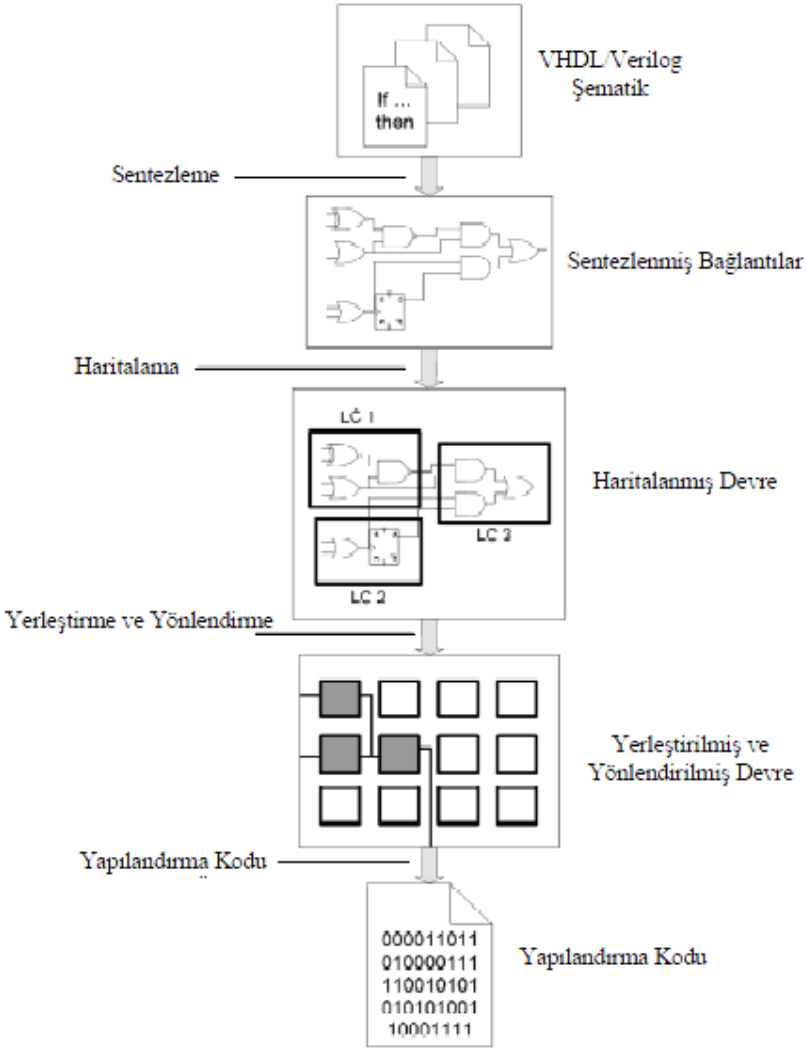
- Devrenin sözle tanımı
- Şematik veya HDL (VHDL, Verilog, vb.) kullanılarak tasarımı hazırlama
- Devreye ait standart bağlantı listesi (Netlist)
- Fonksiyonel benzetim (Functional Simulation)
- Lojik sentezleme
- FPGA seçimi
- Sentezleme işleminde varsa, devreye ait lojik kısıtlamalar (I/O, zamanlama, yerleştirme, saat frekansı, kritik yollar,..) kullanıcı kısıtlama dosyası (Xilinx FPGA için User constraints file)
- Lojik sentezleyici ile istenilen fonksiyonların gerekli lojik indirgemeleri (logic optimization)
- Elde edilen lojik fonksiyonlar FPGA içerisindeki lojik bloklarla eşleştirilmesi işlemi (technology mapping) kapı seviyesinde bir bağlantı listesi
- Teknoloji haritalaması sırasında, kullanıcı kısıtlama dosyası da kullanılarak zamanlama gereksinimi karşılanmak amacıyla gerekirse daha fazla lojik eleman kullanımı.
- Sentezleme sonrasında, yerleştirme ve yönlendirme (Placement and Routing)
- Bu asamadan sonra kapı seviyesinde benzetimin gerçekleştirilmesi uygun olacaktır.

Yerleştirme ve yönlendirme sonrası benzetimlerde istenilen sonuçlar elde edildikten sonra FPGA'nın programlanması aşamasında kullanılacak bit dizisi, üreticinin sağladığı yazılımla elde edilir. Bu süreç izlenerek oluşturulan yapılandırma veri dosyası (bit stream kodu) FPGA üreticisi firmanın geliştirdiği uygun donanım kullanılarak programlanması ile tasarım tamamlanır. Bu programlama adımlarına ilişkin tasarım akış algoritması şekil 3.11'de verilmiştir.



Şekil 3.11. FPGA tasarım algoritması [29].

Şekil 3.11'deki tasarım algoritmasının akış seması şekil 3.12'de verilmiştir.



Şekil 3.12. FPGA tasarım akışı [29].

3.4. FPGA Kullanım Alanları

1990'ların başında haberleşme ve ağ ortamlarında, 1990'ların sonlarına doğru tüketiciye yönelik otomotiv ve endüstriyel kullanım alanlarında. 2000'li yıllarda, milyonlarca kapı içeren yüksek performanslı modeller, ek olarak gömülü mikroişlemci çekirdekleri, yüksek hızlı I/O ara yüzleri, gömülü RAM ve DSP öbekleri, kriptoloji uygulamaları ve tıbbi görüntüleme sistemlerinde sıkça kullanılmakta kullanım alanları hızla çoğalmaktadır [25-28].

3.5. FPGA Üreticileri

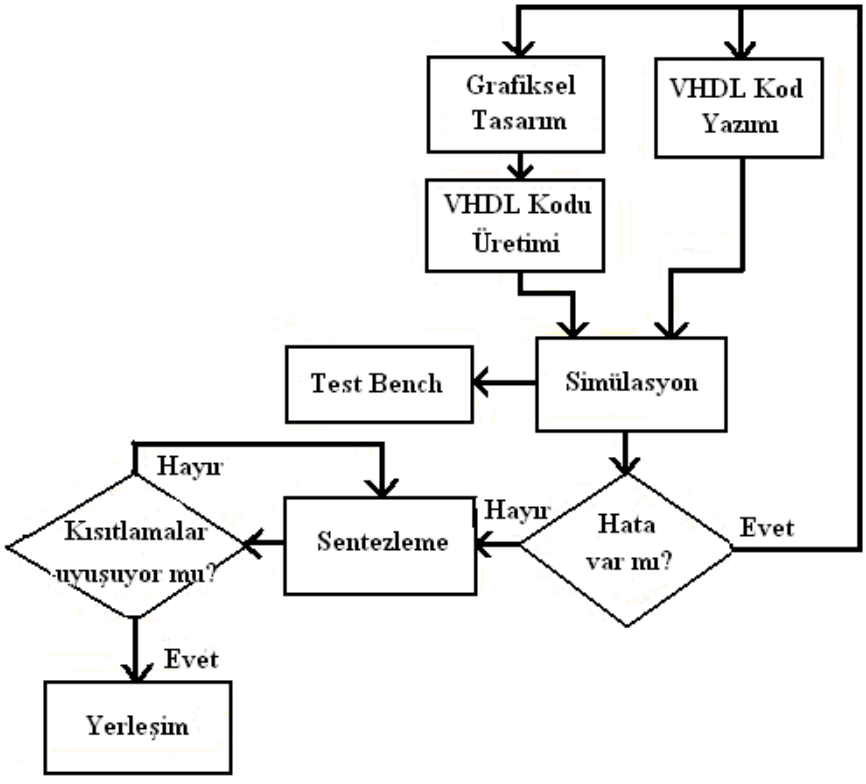
Üretici firmalar olarak önde gelen firmalar; Xilinx, Altera, Actel, Atmel Atomic, Lattice Semiconductor, QuickLogic Achronix Semiconductor ve MathStar olarak sıralanabilir. Xilinx ve Altera market liderleridir. Lattice Semiconductor SRAM ve uçucu olmayan, flash tabanlı FPGA'lar sağlamaktadır. Actel antifuse, tekrar programlanabilir flash tabanlı FPGA'lar ve karışık sinyal flash tabanlı FPGA ürünlerine sahiptir. Atmel atomik (ince tanecikli) cihazlar sağlamaktadır, FPGA ile aynı ömre sahip Atmel AVR mikrodenetleyici geliştirmek üzere odaklanmışlardır. QuickLogic, antifuse (programmable-only-once) ürünlerine sahip olup genel ilgi alanları askeri uygulamalardır. Achronix Semiconductor geliştirmede olan oldukça hızlı FPGA'lar vardır, hızları 2Ghz'e yaklaşmaktadır. Math Star, Alan Programlanabilir Nesne Dizisi isimli FPGA benzeri bir cihaz sunmuştur [31,32].

3.6. Üretim Teknikleri

1. **SRAM:** SRAM temelli yapılandırma hücreleri kullanılır. SRAM tekniğinin olumlu yanı, tasarım fikirlerinin hızlı bir şekilde geliştirilebilir ve sınanabilir olmasıdır. Olumsuz yanı ise sistemin her açılışında aygıtın yapılandırılma zorunluluğudur.
2. **Antifuse:** Olumsuz yanı, bir kez programlanır olmasıdır. Bu özelliğinden dolayı uygulama geliştirme için tercih edilmez.
3. **EPROM:** Silinebilme ve yeniden programlanabilme özelliğine sahiptir. EPROM yongaları üzerine açılan pencere vasıtasıyla program belli bir zaman güneş ışığına ve morötesi ışınlara tutularak silinmektedir.
4. **EEPROM:** Elektrikle silinip programlanabilme özelliğine sahiptir.
5. **Flash:** Bazı çeşitleri devre içinde programlanabilirdir. EEPROM'a benzer yapıdadır.
6. **Fuse**: Bir kez programlanır. Bu özelliğinden dolayı uygulama geliştirme için tercih edilmez.

4. DONANIM TANIMLAMA DİLİ

Çok Yüksek Hızlı Tüm Devre Donanım Tanımlama Dili (Very High Speed Integrated Circuit Hardware Description Language, VHDL), geliştirilmesi ilk olarak Amerikan Savunma Departmanı tarafından başlatılmıştır. Donanımı tanımlamak için, bilgisayar ve insanlar tarafından aynı anda okunabilir olacak ve geliştiricileri yapısal ve anlaşılır kod yazmaya zorlayacak, yani kaynak kodun kendisi bir tür belirtim dokümanı (specification document) olarak sunulabilinecek bir dil istemekteydiler. Ayrıca bu ürün kompakt yapıdaki kompleks fonksiyonları modelleyecek ardışık deyimleri desteklemeliydi. 1987'de, VHDL Amerikan Elektrik ve Elektronik Mühendisleri Enstitüsü (IEEE) tarafından ilk resmi güncelleştirilmesi gerçekleştirilmiş, 1993 yılında ilk kez standartlaştırılmıştır. Dosya işleme prosedürü dışında bu iki standart birbiriyle uyumludur. Dilin standardı (Language Reference Manual (LRM)) Dil Referans Elkitabı ile tanımlanmıştır. Şekil 4.1'de FPGA'ya bir programın yüklenme aşamalarının blok şeması verilmiştir [32-35].



Şekil 4.1. FPGA'ya bir programın yüklenme aşamaları blok şeması [33].

4.1. Ardışık ve Paralel Kod Yazımı

VHDL kodları paralel ya da ardışık olabilir. Sayısal devreler çıkışın daha önceki çıkışlara bağlı olmadığı kombinezonal devreler ve hafıza birimleri içeren ardışık devreler olmak üzere iki kısma ayrılabilirler. Sadece kombinezonal devrelerin gerçekleştirilmesi için paralel kod kullanılabilir. Hem ardışık devrelerin hem de kombinezonal devrelerin gerçekleştirilmesi için ardışık kod kullanılır [33-36].

4.2. VHDL Temel Bildirimleri

VHDL temel bileşenleri aşağıda verilmiştir [36];

- Varlık (Entity)
- Mimari (Architecture)
- Paket (Package)
- Bileşen (Component)
- İşlem (Process)

4.2.1. Varlık

Bir tasarımın en temel bloğudur. Verilen bir mantık fonksiyon için bütün giriş ve çıkışları yani mantık fonksiyonun dış dünya ile bağlantısını tanımlar. Her VHDL tasarım mutlaka en az bir varlık içerir. Port tanımlamaları farklı biçimlerde olabilir. Burada port tanımlaması ile işaretin giriş / çıkış olduğu ve veri tipi belirtilir. Örnek bir varlık tanımlaması aşağıda verilmiştir [33].

ENTITY entity-adı **IS**
PORT(giriş: in STD_LOGIC;
Çıkış: out STD_LOGIC_VECTOR(3 downto 0));
END entity-adı;

4.2.2. Mimari

Mimari varlığın davranışını tanımlar. Bir varlık birden fazla mimariye sahip olabilir. Bir mimari davranışsal tanımlama, yapısal tanımlama, veri akışı olmak üzere üç farklı biçimde kullanılabilir. Sırasıyla genel tanımlama biçimleri aşağıda verilmiştir [33,34].

4.2.2.1. Davranışsal Tanımlama

ARCHITECTURE architecture-adı **OF** entity-adı **IS**
Sinyal tanımlamaları;
Fonksiyon tanımlamaları;
Procedure tanımlamaları;
BEGIN
PROCESS bloğu;
Eş zamanlı işlemler;
END architecture-adı;

4.2.2.2. Yapısal Tanımlama

ARCHITECTURE architecture-adı **OF** entity-adı **IS**
Component tanımlamaları;
Sinyal tanımlamaları;
BEGIN
Anlık-adı: PORT MAP ifadeleri;
Eş zamanlı işlemler ifadeleri;
END architecture-adı;

4.2.2.3. Veri Akışı Tanımlama

ARCHITECTURE architecture-adı **OF** entity-adı **IS**
Sinyal tanımlamaları;
BEGIN
Eş zamanlı işlemler;
END architecture-adı;

4.2.3. Paket

Paket; varlık üniteleri tarafından kullanılan tanımlamaları bir grup haline getirir ve paylaştırır. Kullanımı aşağıda verilmiştir [33-36].

PACKAGE paket-adı **IS**
Tip tanımlaması;
Alt tip tanımlaması;
Sinyal tanımlamaları;
Değişken tanımlamaları;
Sabit tanımlamaları;
Component tanımlamaları;
Fonksiyon tanımlamaları;
Procedure tanımlamaları;
END paket-adı;

4.2.4. Bileşen

Bileşen yapısı; devre tanımlamasında bir alt devre gibi kullanılan bileşenin adını ve ara yüzünü tanımlar. Aşağıdaki gibi kullanılır [34].

COMPONENT bileşen–adı **IS**
PORT(port-adı listeleri ve tipleri);
END COMPONENT;

4.2.5. İşlem

İşlem bloğu sıralı şekilde gerçekleştirilecek durumları içerir. Bir mimaride birden fazla işlem bloğu anlık gerçekleştirilir. Yani işlem blokları aynı anda başlar ve her bir işlem bloğu kendi içinde satır satır sıralı olarak gerçekleştirilir. Kullanımı aşağıdaki gibidir [34].

işlem-adı: **PROCESS**(Hassasiyet-listesi)
Değişken tanımlamaları;
BEGIN
Sıralı deyimler;
END PROCESS işlem-adı;

4.3. Veri Tipleri

VHDL nesnel özelliklere sahip olan bir dildir. Nesnelerin davranış ve işlevlerine göre fonksiyonellik kazanır [36].

4.3.1. Sinyal

Sinyaller varlık, mimari ve paket içinde kullanılabilir. Sinyallerin paket içinde kullanımı çok önemlidir. Çünkü paket genel bir ifade olduğundan dolayı işaretler burada kullanıldığı zaman diğer varlık veya mimariler tarafından çağrılarak kullanılabilir. Sinyallere başlangıç değeri atanabilir ve işlem gerçekleşirken bu ifade yenilenebilir [34]. Kullanımı aşağıdaki gibidir.

SIGNAL sinyalin-adı: sinyal_tipi [:= başlangıç değeri];

4.3.2. Değişken

Genel olarak geçici değerleri kullanmak için tercih edilir. Böylece programın daha hızlı çalışması sağlanabilir. Fakat gerçek zamanlı işlemlerde özellikle tasarımın FPGA'da gerçekleşmesinde değişken kullanımının sakıncaları bulunmaktadır [33].

VARIABLE değişkenin-adı: değişkenin tipi [:= başlangıç değeri];

4.3.3. Sabit

Eğer tasarımda değişmez verilere ihtiyaç duyulursa sabitler kullanılır. Bu durum, tasarımın daha iyi gözlemlenebilmesi ve anlaşılabilmesi için önemli bir faktördür. Özellikle tasarım içinde sık sık kullanılan değeri aynı olan ifadeler için vazgeçilmez bir veri tipidir. Kullanımı aşağıdaki gibidir [35].

CONSTANT sabitin-adı: sabitin tipi [:= başlangıç değeri];

4.3.4. Ön Tanımlamalı Veri Tipleri

Bu veri tipleri aşağıda verilmiştir. STD_LOGIC ve STD_LOGIC_VECTOR tiplerinin kullanımı için, VHDL tasarım dosyası "std_logic_1164" paketini içermelidir [33].

Standart Mantık Tip: **STD_LOGIC, STD_LOGIC_VECTOR**
Bit Tip: **BIT, BIT_VECTOR**
Tam Sayı Tipi: **INTEGER**
Kayan noktalı Tip: **REAL**
Fiziksel Tip: **TIME**
Liste Tip: **BOOLEAN, CHARACTER**

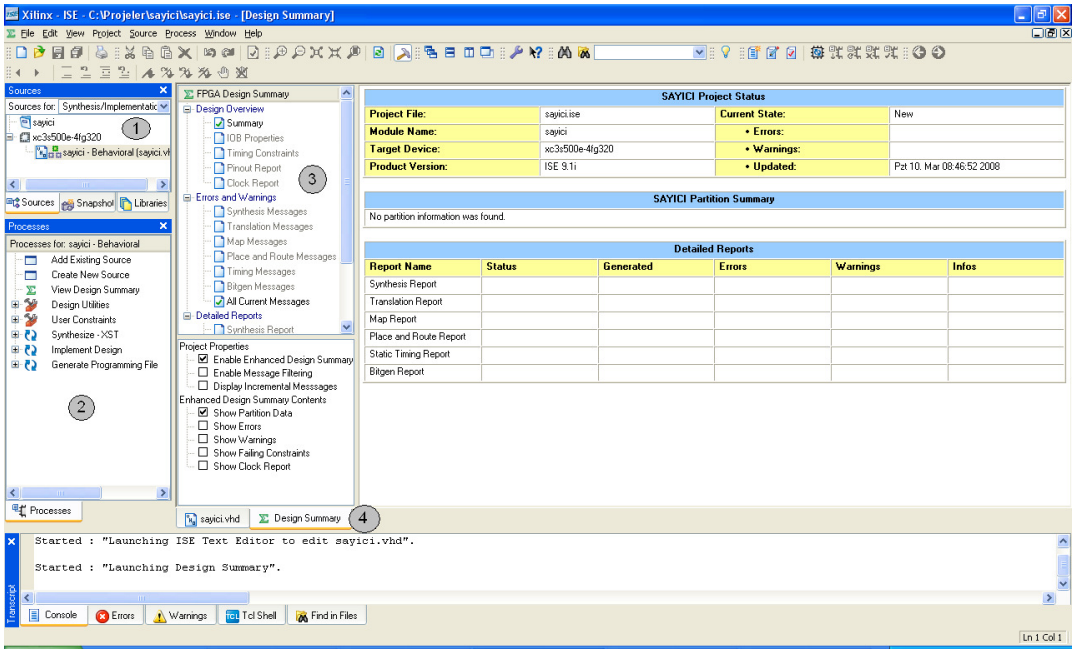
5. XILINX ISE 9.2i DERLEYİCİSİ

5.1. Genel Yapısı

Entegre Yazılım Ortamı (Integrated Software Environment, ISE), Xilinx FPGA'ların üzerinde çalışılmasını sağlayan bir yazılım programıdır. Verilog veya VHDL gibi donanımsal diller kullanılarak yazılan kodlar bu program aracılığı ile sentezlenerek FPGA ortamına aktarılırlar [36]. ISE yazılımını çalıştırabilmek için program kurulduktan sonra masa üstünde şekil 5.1'deki ikona tıklanarak ya da başlat menüsünden Başla → Programlar → Xilinx ISE 9.2i → Project Navigator yolu ile çalıştırılır.

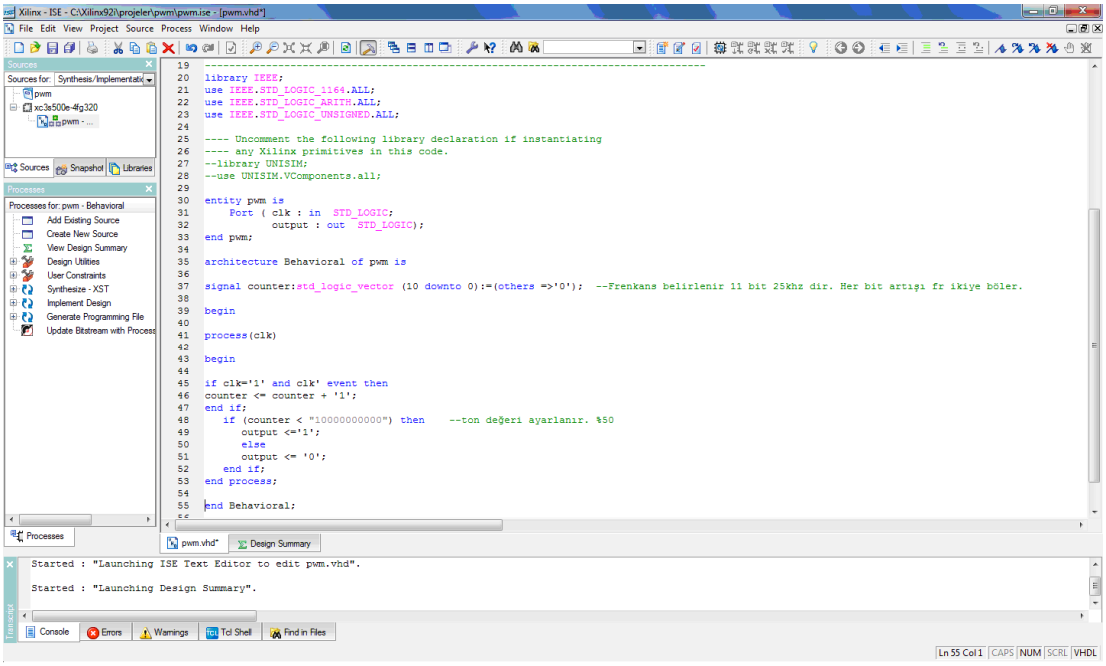
Şekil 5.1. Masa üstündeki kısa yol ikonu

Şekil 5.2'de bir tasarım ekranı gösterilmiştir. 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 yapılmaktadır. ③ işaretlenmiş kısımda ise dosya içeriği görüntülenir. ④ tasarım penceresi ve HDL kullanılarak yazılacak dosyalar arasında geçiş yapılabilecek kısmı gösterir [36].



Şekil 5.2. ISE tasarım ekranı

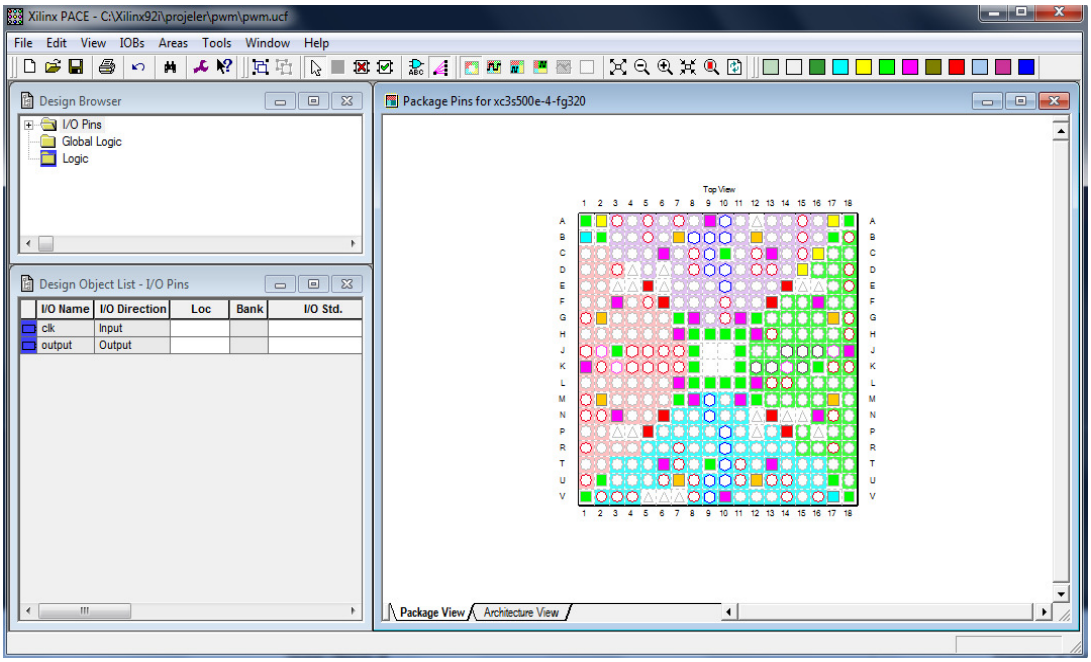
New Project simgesi tıklanarak boş sayfası açılır ve VHDL kodu bu sayfaya yazılır (şekil 5.3). Yazım hataları check syntax simgesine tıklanarak düzeltilir.



Şekil 5.3. VHDL dosyası

VHDL kodu derleme işleminden sonra devrenin RTL (Register Transfer Level) şematik tam yapısı incelenir. Bu yapıda mantık kapıları giriş çıkış uçlarının gösterildiği devre yapısıdır. VHDL kodu yazılan devrenin çalışmasının benzetimi yapılarak program içerisinde eksiklikler giderilir. Yazılan VHDL kodu FPGA içerisine yerleştirilir.

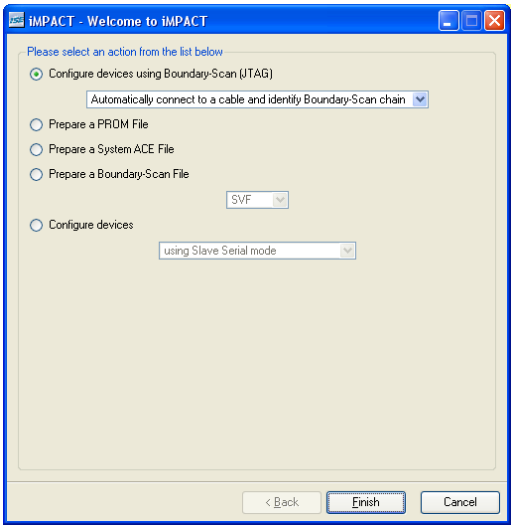
Şekil 5.4 'de VHDL kodu yazılan programın giriş çıkış pinleri seçilerek FPGA' ya yüklenmesi gösterilmektedir. FPGA seçilen giriş çıkış pinlerini kullanılarak devre bağlantılarını gerçekleştirir.



Şekil 5.4. FPGA'nın fiziksel pinlerinin seçimi

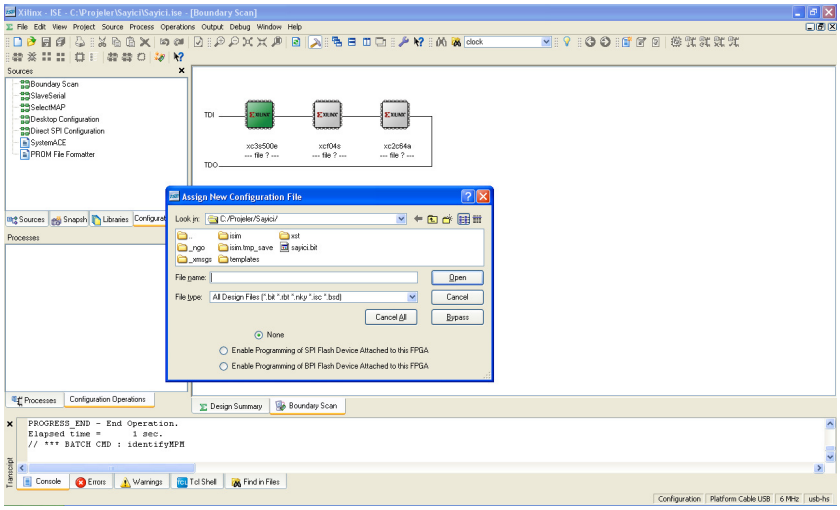
5.2. VHDL ile Hazırlanmış Bit Dosyasının FPGA' ya Yüklenmesi

İmpact simgesi tıklanarak bilgisayar ile FPGA arasında bağlantı sağlanır (şekil 5.5). Finish butonuna tıklanarak bu işlem bitirilir.



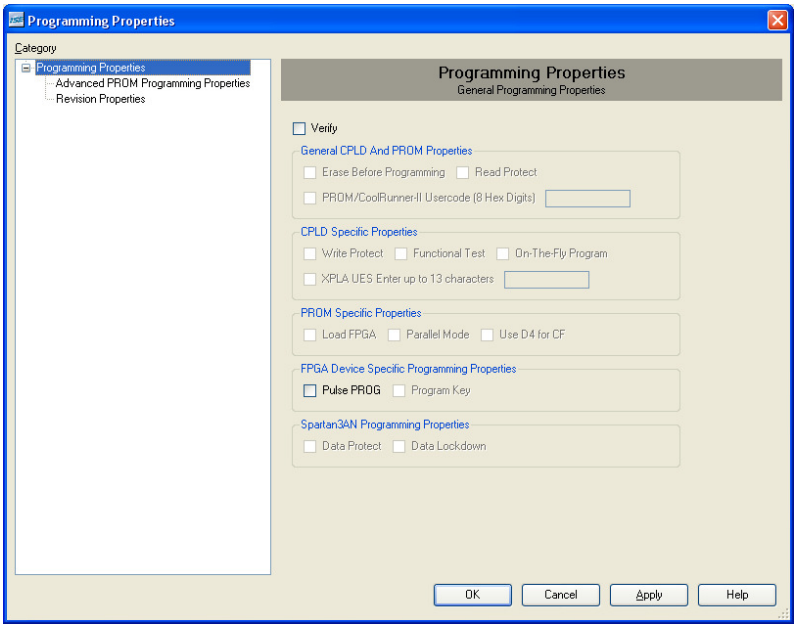
Şekil 5.5. Bilgisayarın FPGA'ya bağlanması

VHDL programının FPGA'ya yüklenmesi için dosya yerinin sorulduğu ekran görüntüsüne (şekil 5.6). Burada yazılmış kodu içeren dosya seçilmeli. Daha sonra gelen iki ekranda ise bypass simgesine tıklanmalı.



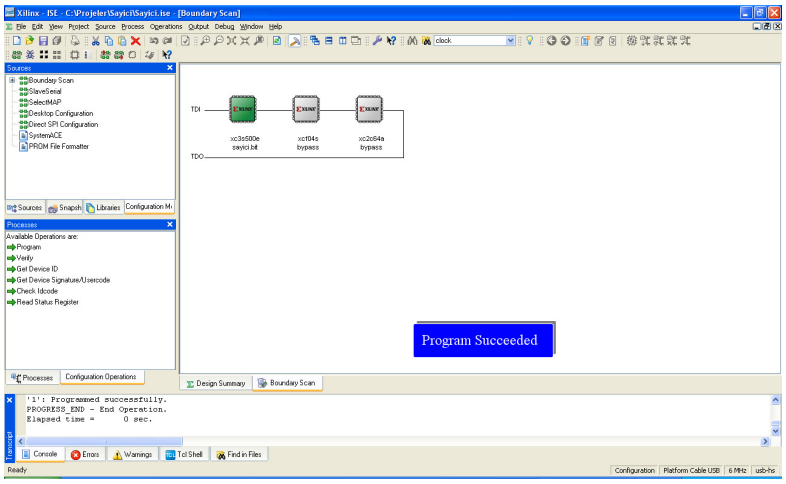
Şekil 5.6. VHDL kodunun FPGA'ya yüklenmesi

Üzerinde xc3s500e yazılı işlemciye farenin sağ tuşu ile tıklanarak açılan kısımdan program seçilir, şekil 5.7'de görülen pencereden OK butonuna basılarak FPGA'ya yazma işlemi gerçekleşir.



Şekil 5.7. Programın yüklenmesi

Program Succeeded uyarısıyla başarılı bir yazma işleminin gerçekleşmiş olur (şekil 5.8). Program artık FPGA starter kitine yüklenmiştir.



Şekil 5.8. Programın başarılı bir şekilde yüklendiğinde verdiği mesaj

6. GÜÇ DEVRESİNİN TASARIMI VE GERÇEKLEŞTİRİLMESİ

Düşürücü DA-DA dönüştürücü devresini gerçekleştirebilmek için, anahtarlama frekansının, devrede kullanılan eleman değerlerinin, giriş ve çıkış voltaj değerlerinin, akım ve gerilim salınım değerlerinin önceden belirlenmesi gerekmektedir.

6.1. Devre Elemanlarının Değerlerinin Belirlenmesi

Tablo 6.1. Güç devresi tasarım parametreleri

PARAMETRE	AÇIKLAMA	DEĞER
V_{in}	Giriş voltajı	24V
V_o	Çıkış voltajı	12V
ΔI_o	Çıkış akım salınımı	30mA
ΔV_o	Çıkış voltaj salınımı	30mV
f_s	Anahtarlama frekansı	24.4kHz

Bobinin endüktans değeri, devreyi sürekli akım modunda (Continuous Conduction Mode-CCM) tutacak olan minimum ortalama çıkış akım değerini belirlemektedir. Bu çalışmada, güç devresi mümkün olduğunca CCM modunda çalışacak şekilde tasarlanmıştır, çünkü bu modda akımın tepe değerleri daha düşüktür, gürültü daha azdır ve çalışma oranı ile çıkış gerilimi direkt orantılı olduğu için hassas gerilim ayarı yapmak daha kolaydır [1,13].

Bu çalışmada, sürekli ve süreksiz akım modları arasındaki geçiş akım değeri 15 mA olarak seçilmiştir, çünkü bu değer devrenin çoğu durumda CCM'de çalışmasını sağlamakta ve piyasada toroid şeklinde hazır olarak bulunabilen bir bobinin kullanılmasıyla elde edilebilmektedir. Bobinin endüktans değerinin ve akım dalgalılığının hesaplanışı denklem 2.15'den yararlanılarak aşağıda hesaplanmıştır. D oranı için denklem 2.5'ten yararlanılmıştır.

$$D = \frac{V_o}{V_{in}} \qquad D = \frac{12}{24} \Rightarrow D = 0{,}5$$

$$L > \frac{V_o(1-D)}{2I_{o.\min} f_s} \qquad L > \frac{12.(1-0{,}5)}{2.15.10^{-3}.24{,}4.10^3} \qquad L > 8{,}2mH$$

Bu hesaplamalar dikkate alınarak 8,8mH değerinde bobin kullanılmıştır. Bobin değerinin daha da yükseltilmesi sistemin dinamik performansını düşüreceği için tercih edilmemiştir.

Çıkış geriliminin dalgalılığı, filtre kondansatörünün eşdeğer seri direnci (ESR), eşdeğer seri endüktansı (ESL) ve kapasitesi (C) tarafından belirlenmektedir. Tasarlanan güç devresinin çalışma frekansı düşük olduğu için ESL'nin etkisi ihmal edilebilmektedir. CCM'de çalışma durumu için, çıkış gerilimindeki dalgalılığın sadece kapasiteye bağlı olduğu varsayılırsa, 30mV'luk bir dalgalılık için gereken kapasite değeri denklem 2.17'den yararlanılarak aşağıdaki şekilde hesaplanmıştır.

$$\Delta V_o = \frac{T_s}{8C}\frac{V_o}{L}(1-D)T_s$$

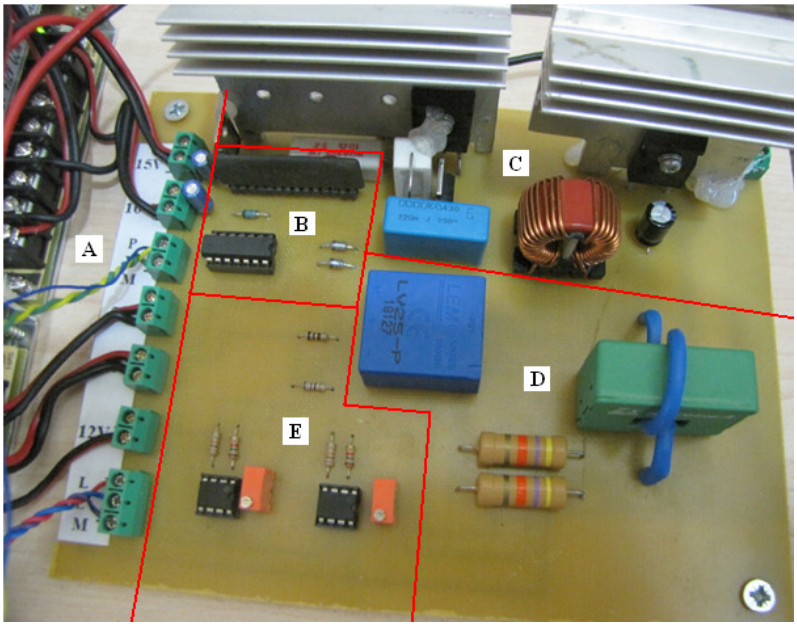
$$C = \frac{V_o(1-D)}{8\Delta V_o L f_s^2} \qquad C = \frac{12(1-0{,}5)}{8.30.10^{-3}.8.10^{-3}\left(24{,}4.10^3\right)^2} \qquad C = 5{,}25\mu F$$

Elde edilen sonuçtan görülebileceği gibi, ESR'nin (Rs) ihmal edilmesi durumunda, dalgalılık gereksinimini sağlayamayacağı açıkça belli olan çok düşük bir kapasite değeri elde edilmektedir. Gerilim dalgalılığının düşük kaliteli, yüksek ESR'li bir kondansatör kullanılması durumunda bile 30mV'dan daha düşük olacağından emin olmak için 220µF değerinde bir kondansatör kullanılmasına karar verilmiştir.

Anahtarlama elemanları maruz kalacağı akım ve gerilim değerlerine dayanabilecek şekilde seçilmelidir. Burada dikkat edilmesi gereken diğer bir nokta, çeviricinin çalışma şartlarına göre verimliliğini arttırmak için güç kaybını en aza indirecek şekilde bir seçim yapabilmektir. Çalışmada (Q) anahtarlama kayıplarının minimum seviyede tutulması ve (D) diyotunun iletim durumundaki kayıplarının minimum seviyede tutulması gerekmektedir. Bunun için IGBT anahtarlama elemanı olarak APT25GP90BDQ1 seçilmiş

ve diyot olarak da ters iyileşme zamanı (trr) düşük olan DSEI 60-06A diyot seçilmiştir. IGBT'ler MOSFET ve BJT elemanlarının avantajlarını bir arada bulunduran gerilim kontrollü bir yarıiletken anahtardır. IGBT'ler BJT ye göre hızlı bir elemandır, ancak MOSFET kadar hızlı değildir. IGBT'ler yüksek akım, yüksek gerilim ve yüksek frekanslar için uygun bir elemandır. Anahtar sürücü entegresi olarak da VLA 531-01R seçilmiştir. Anahtar ve diyot için yüksek akım ve gerilim değerlerinin seçilmesinin nedeni devrenin yapısını güçlendirerek uygulama alanlarını arttırmaktır.

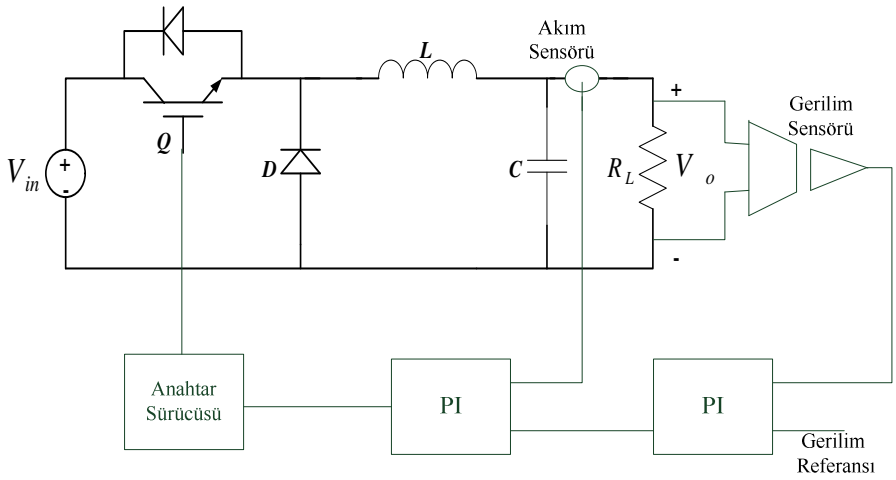
Şekil 6.1'de tamamlanmış devrenin bölümleri gösterilmiştir. A'da, devre için gerekli besleme kaynağı bağlantı uçları, PWM girişi ve sensör çıkışları, B'de, anahtar sürücü entegresi ve değil kapısı, C'de, düşürücü DA-DA dönüştürücüsü devre elemanları ve koruma (Snubber) elemanları, D'de, akım ve gerilim bilgilerini okuyan sensörler, E'de, işlemsel yükselteçler bulunmaktadır.



Şekil 6.1. Devrenin bölümleri

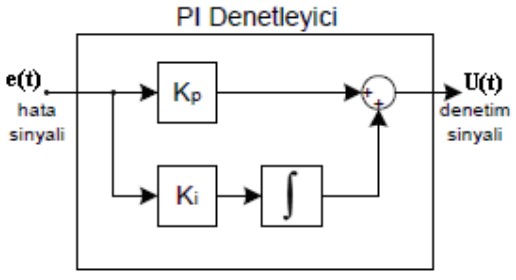
6.2. Akım ve Gerilim Denetimi

Şekil 6.2'de verilen devrenin blok diyagramında akım ve gerilim denetimi için PI denetleyici kullanılmıştır. Akım ve gerilim bilgileri sensörler aracılığı ile alınıp, şekil 6.2'de gösterildiği gibi bağlantı gerçekleştirilmiş ve yük için hem akım hem de gerilim denetimi sağlanmıştır. Bu denetim yazılımsal olarak Ek B'de verilmiştir.



Şekil 6.2. Devrenin blok diyagramı

Kullanılan PI denetleyici, DA-DA dönüştürücünün çıkış gerilim ve akım bilgisine ait geri besleme denetiminde verimi arttırmak için, hata değerlerini belirli bir oran ve zaman aralığında girişe uygulamak gerekir. Bunun sağlanması için yazılımla kontrol edilen bir PI denetim algoritması geliştirilmiştir. Şekil 6.3'de genel bir PI denetleyicinin blok diyagramı verilmektedir [37-39].



Şekil 6.3. PI denetleyici genel blok diyagramı [37].

$$U = K_P e + K_i \int e dt \tag{6.1}$$

Denklem 6.1'de K_P oransal katsayıyı, K_i integral katsayısını göstermektedir. e, gerçek ile referans değeri arasındaki hata değerini göstermektedir. U çıkış görev saykıl değişim değerini göstermektedir. Denklem 6.1 göz önüne alınarak gerekli işlemler yapılırsa denklem 6.2 elde edilir [39].

$$U_i(k) = K_i(e(k) + e(k-1))$$
$$U_{pi}(k) = K_P.e(k) + U_i(k-1) + U_i(k) \tag{6.2}$$

6.3. Ölçüm Devresinin Tasarımı ve Gerçekleştirilmesi

Düşürücü DA-DA dönüştürücünün çıkış gerilimini yükün çektiği akıma göre kontrol edebilmek için, öncelikle bu iki büyüklüğün ölçülmesi gerekir. Tasarımı bu bölümde anlatılmış olan devrenin amacı, bu iki büyüklüğü ölçmek ve FPGA kiti ile güç devresinin gerilim seviyeleri arasındaki çevirme işlemlerini yaparak ölçüm sonuçlarını seri olarak FPGA kitine göndermektir.

6.3.1. Gereksinimlerin Belirlenmesi

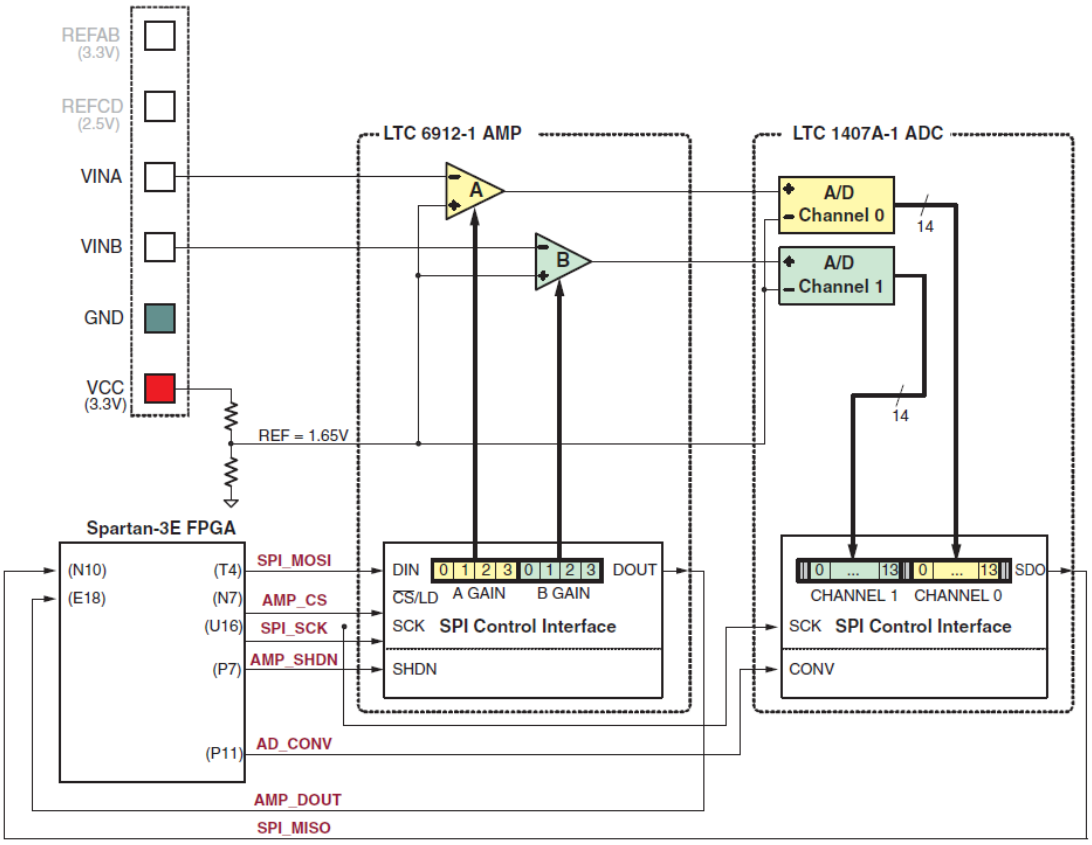
Akım ve gerilim değerlerinin FPGA'ya gönderilebilmesi için sayısal bilgiye çevrilmesi gerekmektedir. Bu işlem için kit içerisindeki ADC kullanılmıştır. Spartan-3E Starter Kiti içerisinde 14 bit çözünürlüğe sahip LTC 1407A-1 entegresi kullanılmıştır.

6.3.2. Devrede Kullanılan Entegreler ve LEM Modülleri

Bu bölümde, ölçüm devresinde kullanılmış olan entegre ve LEM modüller ile bunların devredeki işlevleri hakkında kısa bilgi verilmiştir.

6.3.2.1. Spartan-3E Starter Kitinin Kazanç ve ADC Kontrolü

Spartan-3E Starter Kit inin içerisinde Linear Technology firmasının üretmiş olduğu LTC 6912-1 kazanç değeri ayarlana bilen entegresi ve LTC 1407A-1 ADC entegresi bulunmaktadır. Spartan-3E Starter Kiti 2 analog girişi (VINA, VINB) bulunmaktadır. Spartan-3E Starter Kiti ADC blok diyagramı şekil 6.4'te gösterilmiştir. Kazanç entegresi ve ADC entegresi FPGA tarafından kontrol edilmektedir [40].

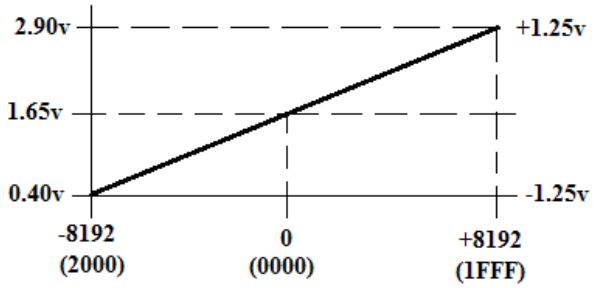


Şekil 6.4. Spartan-3E Starter Kiti ADC blok diyagramı [40].

Analog girişlerden gelen bilgi 14 bitlik dijital bilgiye çevrilir (denklem 6.3).

$$D[13:0] = Kazanç \frac{(V_{in} - 1.65)}{1.25V} 8192 \tag{6.3}$$

LTC 6912-1 kazanç entegresi ve LTC 1407A-1 ADC entegresi, 1.65V referans değere sahip ve 1.25V'luk bir aralıkta giriş alabilmektedir (şekil 6.5). Kazanç ayar değerleri tablo 6.2'de belirtilmektedir. Her bir yükseltecin ayarı -1 den -100 e kadar kazanç değeri alır. Bu çalışmada kazanç ayarlarını -1 olarak ve 6 bitlik sayısal bilgi kullanılmıştır. Ek B'de ADC modül komutları verilmiştir.



Şekil 6.5. VINA ve VINB giriş gerilim aralığı

Tablo 6.2. LTC 6912-1 kazanç entegresi giriş voltaj aralığı değerleri

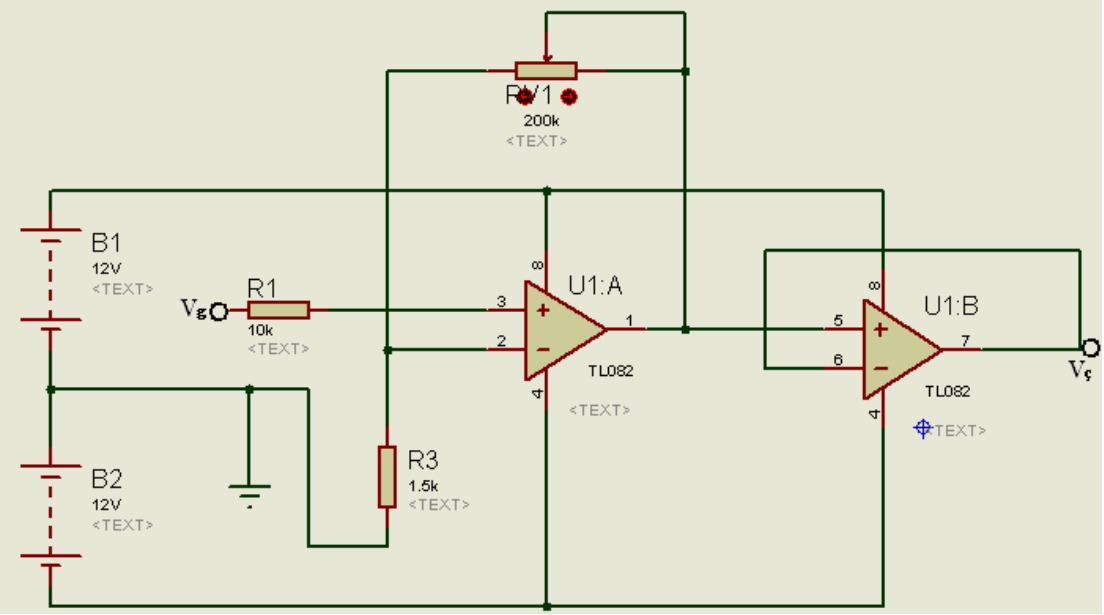
KAZANÇ	A3	A2	A1	A0	Giriş Voltaj Aralığı	
	B3	B2	B1	B0	Minimum	Maksimum
0	0	0	0	0		
-1	0	0	0	1	0.4	2.9
-2	0	0	1	0	1.025	2.275
-5	0	0	1	1	1.4	1.9
-10	0	1	0	0	1.525	1.775
-20	0	1	0	1	1.5875	1.7125
-50	0	1	1	0	1.625	1.675
-100	0	1	1	1	1.6375	1.6625

6.3.2.2. Spartan-3E Starter Kitinin LCD Kontrolü

Bu çalışmada Spartan-3E Starter Kit'inin LCD modülü kullanılmıştır. Modülün çalışması incelenmiş ve modül içerisine devrenin ismi yazılmıştır. Ek B'de LCD modül komutları verilmiştir.

6.3.2.3. İşlemsel Yükselteç (OPAMP) Entegresi

Bu çalışmada TL082 işlemsel yükselteç entegresi kullanılmıştır. Bu entegre içerisinde 2 adet işlemsel yükselteç bulunur. Bu işlemsel yükselteçlerden biri LEM modüllerden alınan değeri çevirmeden yükselterek çıkışa aktarır. İkinci işlemsel yükselteç de yükseltilerek alınan çıkış değeri ile bu değerin gönderileceği ADC arasında tampon görevi üstlenir (şekil 6.6).



Şekil 6.6. İşlemsel yükselteç (OPAMP) devre şeması

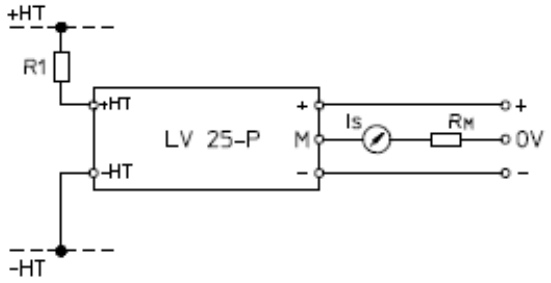
6.3.2.4. Akım ve Gerilim Bilgilerinin Okunması

Bu çalışmada yük üzerine düşen gerilim ve akım değerlerinin sabit bir değerde tutulması hedeflenmektedir. Bunun içinde çeşitli sensörlerin kullanılması gerekmektedir. Yük üzerine düşen gerilim değerini ölçebilmek için LV 25-P, yük üzerinden geçen akımı ölçebilmek için LA 55-P LEM modülleri kullanılmıştır.

6.3.2.4.1. LV 25-P Gerilim Sensörü

Gerilim bilgisini okumak için devreye bir adet LEM firmasının üretmiş olduğu LV 25-P gerilim sensörü bağlanmıştır. LV 25-P gerilim sensörü 10V ile 500V arası gerilimleri ölçebilmektedir. R1 hat direnci çıkış akım değerini sınırlandırmak için kullanılmıştır. Elde edilen gerilim bilgisi çıkışa bağlanan R_M direnç üzerinden alınır.

Şekil 6.7'de LV 25-P'nin devreye nasıl bağlandığı görülmektedir. Bu çalışmada 12 volt besleme gerilimi kullanılmıştır. R1 direnci olarak 23.5kohm'luk direnç seçilmiştir. Sensör çıkışı 100ohm'luk dirence uygulanmış ve direnç üzerinden ölçülen gerilim bilgisi işlemsel yükseltecin girişine uygulanmıştır.

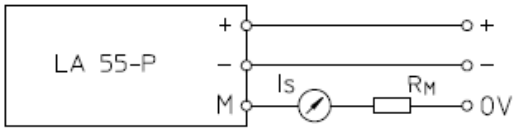


Şekil 6.7. LV 25-P gerilim sensörü devre bağlantı şeması

6.3.2.4.2. LA 55-P Akım Sensörü

Akım bilgisini okumak için devreye bir adet LEM firmasının üretmiş olduğu LA 55-P akım sensörü bağlanmıştır. 12 volt beslemede akım sensörünün kullanılabileceği nominal akım 50A, 15 volt beslemede 70 amperdir. Sekonder nominal akımı 50mA'dir. Sekonderden elde edilen akım bilgisi çıkışa bağlanan R_M direnç üzerinden alınır. Sensörün dönüştürme oranı 1:1000'dir.

Şekil 6.8'de LA 55-P'nin devreye nasıl bağlandığı görülmektedir. Bu çalışmada 12 volt besleme gerilimi kullanılmıştır. Sensör çıkışı 56ohm'luk dirence uygulanmış ve direnç üzerinden ölçülen gerilim akım bilgisi olarak kullanılmıştır. Elde edilen bu gerilim işlemsel yükseltecin girişine uygulanmıştır.



Şekil 6.8. LA 55-P akım sensörü devre bağlantı şeması

6.3.2.5. Değil Kapısı

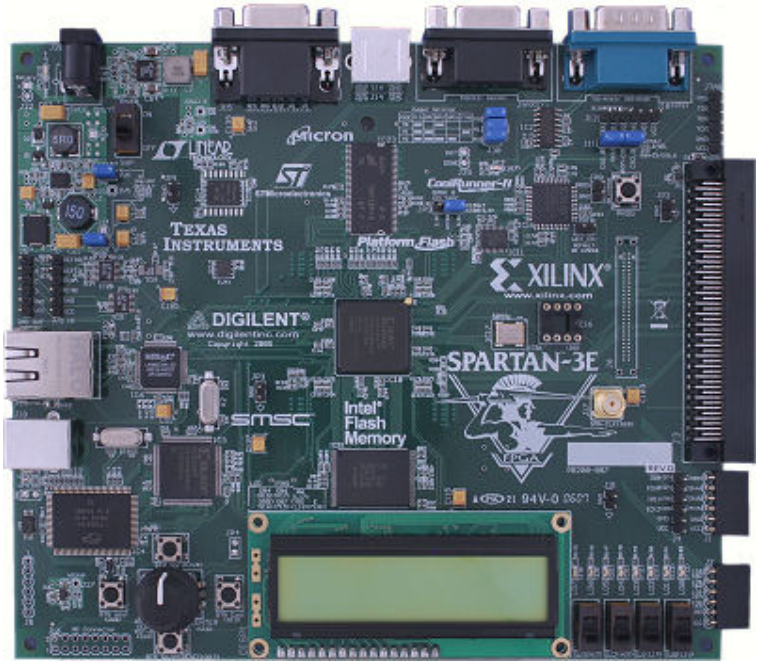
Spartan-3E Starter Kit'inden alınan PWM sinyali anahtar sürücü entegresine direk bağlamak yerine hem izalasyonu sağlamak hem de 3.3V olan çıkış sinyalini sürücü entegresine gerekli olan 5V seviyesine çıkarmak için 74HC04 entegresi (değil kapısı) kullanılmıştır.

6.3.3. Koruma Devresi

Yüksek frekanslı uygulamalarda anahtar üzerindeki gerilim kısa bir süre için kaynak geriliminden büyük olabilmektedir. Bu darbe şeklinde bir gerilim anlamına gelmektedir. Darbe gerilimi anahtarın gerilim bloklama değerinin üzerinde olabilir ve bu aşırı gerilim nedeniyle devre zarar görebilir. Bu nedenle, anahtarın karşısına bir koruma devresi (snubber) yerleştirilir. Koruma devresi anahtarlama geçişini düzgünleştirir ve anahtar geriliminin yükselmesini daha yavaş yapar. Bu şekildeki yüksek gerilim darbesi sönümlendirilir. İletim anı; eleman iletime girerken büyük aşırı akımları minimum yapmak için (di/dt koruması), seri bağlı bobin, direnç ve diyot'tan oluşur. Kesim anı; eleman kesime girerken eleman üzerindeki büyük aşırı gerilimleri minimum yapmak için (dv/dt koruması), paralel bağlı kondansatör, direnç ve diyot tan oluşur [1,2,12,13]. Bu çalışmada iletim anı korumasına gerek duyulmamış, kesim anı koruması anahtarın karşısına yerleştirilmiştir.

6.4. Denetleyici Olarak Kullanılan Donanım

Spartan-3E Starter Kit, Digilent firmasının Xilinx Spartan-3E 500 model (**Xilinx XC3S500E**) FPGA'sı kullanarak ürettiği bir geliştirme platformudur şekil 6.9 [40].



Şekil 6.9. Spartan-3E Starter Kit

6.4.1. Spartan-3E Starter Kitinin Özellikleri

- Xilinx XC3S500E FPGA
- Xilinx XCF04 CPLD
- 32MB Micron DDR SDRAM
- 16MB Numonyx StrataFlash
- 2MB ST Microelectronics Serial Flash
- Linear Technologies Power Supplies
- Texas Instruments TPS75003 Triple-Supply Power Management IC
- SMSC LAN83C185 Ethernet PHY

6.4.2. Kart Üzerindeki Konektörler ve Donanımlar

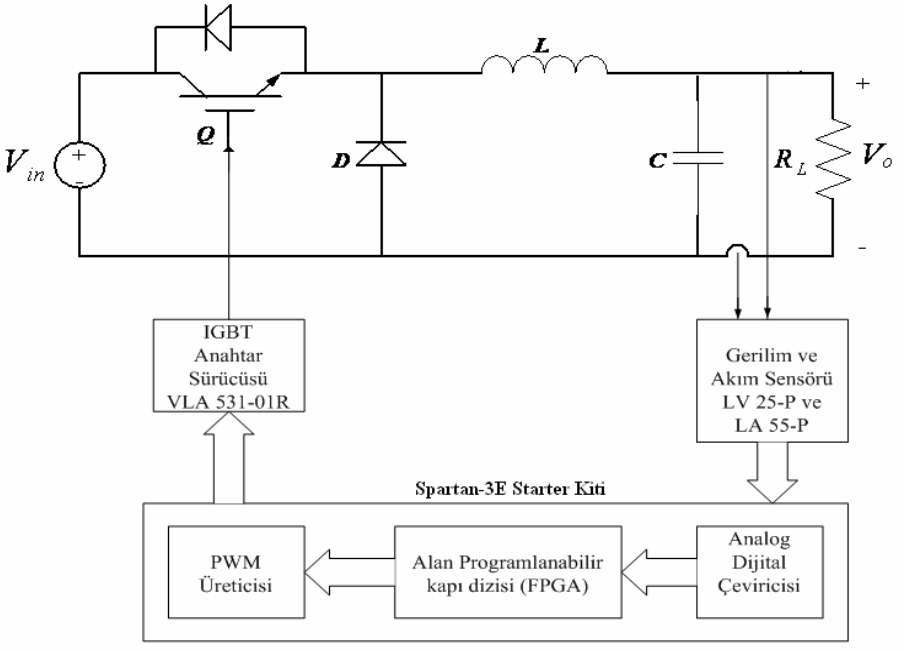
- 100-pinli FX2 Hirose konektörü
- 3 adet 6-pin Pmod konektörü
- 1 adet VGA konektörü
- PS/2 keyboard konektörü
- 2 adet DB9 RS-232 konektörü
- RJ-45 Ethernet konektörü
- Opsiyonel LCD modulü için 16-pin header
- Yüksek hızlı saat girişi için SMA konektörü
- 16x2 LCD ekran
- 4 çıkışlı DAC
- 2 girişli ADC
- 128 Mbit Flash bellek
- 64 Mbyte SDRAM, EEPROM
- Toplam 48 pinlik genişleme portları
- 4 Buton
- 8 Led

7. DENEYSEL ÇALIŞMALAR

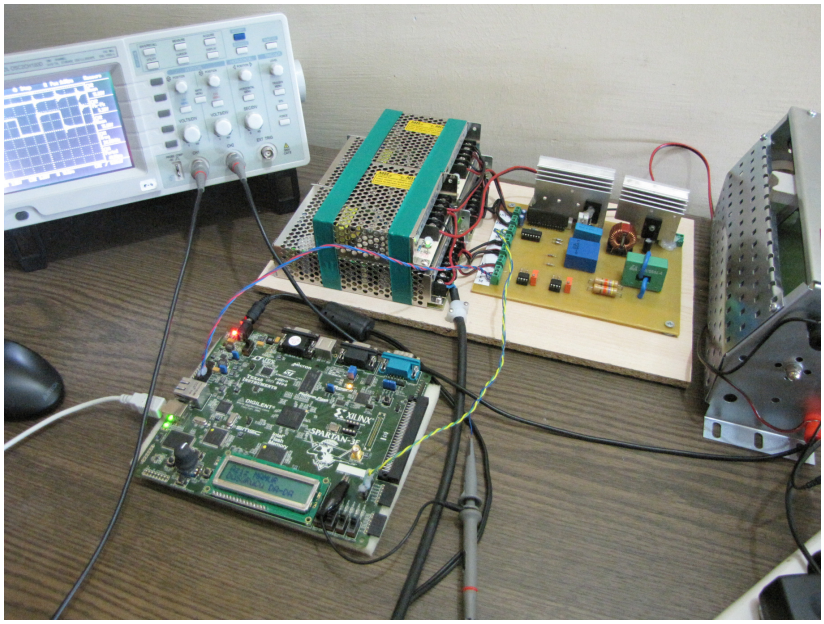
Tasarımı gerçekleştirilen düşürücü DA-DA dönüştürücünün blok şeması şekil 7.1'de verilmiştir. Düşürücü DA-DA dönüştürücü deney sonuçları şekil 7.2'de gösterilen deney düzeneği kullanılarak elde edilmiştir. Düşürücü DA-DA dönüştürücü devresinin çalışma basamaklarından elde edilen sonuçlar, DL OSC2CH100D model osilaskop ile gözlenmiş, elde edilen sonuçlar osilaskop'un DSO (Digital Storage Osillascope) ara yüz programı ile bilgisayar ortamında görüntülenmiştir.

Spartan-3E starter kiti içerisindeki 50MHz'lik dahili osilatöründen yararlanılarak, FPGA içerisinde 24.4kHz'lik bir PWM sinyali üretilmiştir. Ek B'de PWM sinyal komutları verilmiştir.

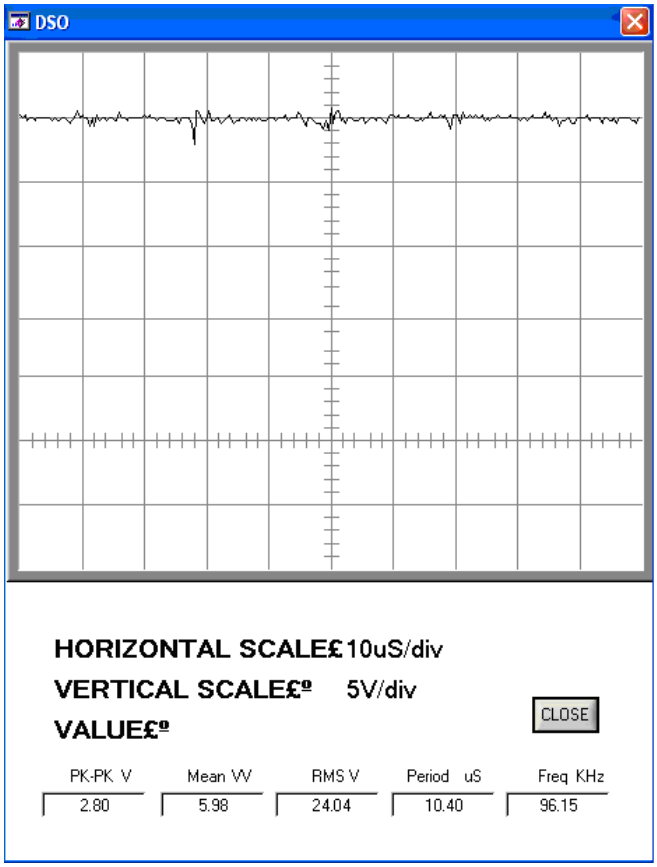
Deneysel sonuçlar, 30ohm ve 50ohm'luk yük değerleri için alınmıştır. İlk olarak denetimsiz ve sabit görev oranlı (D) PWM sinyal değerleri için sonuçlar incelenmiştir. Daha sonra, PI denetimi gerçekleştirilerek çıkış akım ve gerilim değerini sabit tutacak şekilde görev oran değeri sürekli olarak değiştirilmiş ve bu şekilde çıkışın yük değişiminden etkilenmemesi sağlanmıştır. Burada, D oranı olarak 0.25, 0.5 ve 0.75 değerleri kullanılmıştır. Şekil 7.3'de devreye uygulanan 24 voltluk giriş gerilimi görülmektedir.



Şekil 7.1. Düşürücü DA-DA dönüştürücü blok şeması



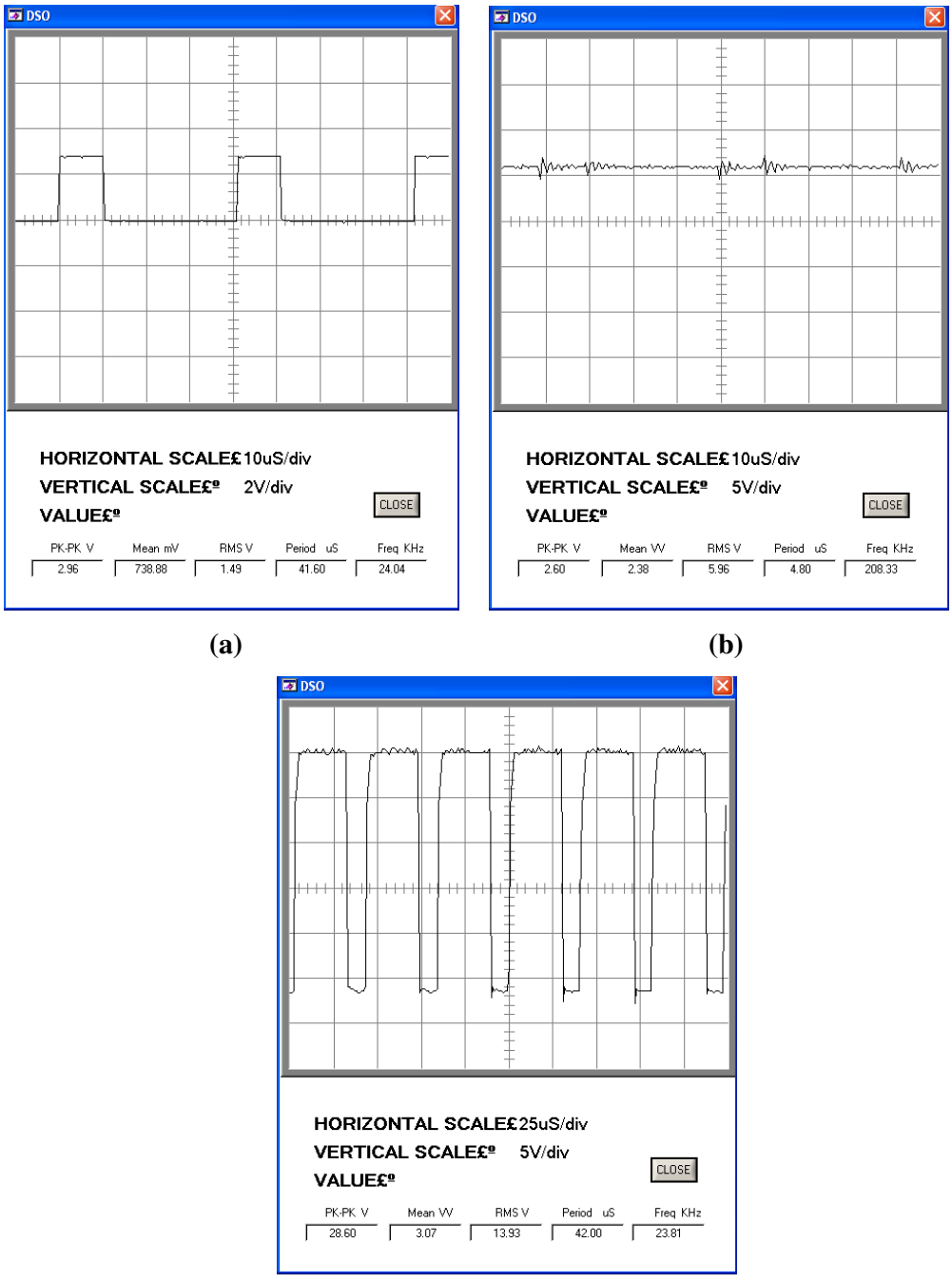
Şekil 7.2. Düşürücü DA-DA dönüştürücünün deney düzeneği



Şekil 7.3. Giriş gerilimi (Vin=24V)

7.1. RL=30ohm Yük Değeri İçin Deneysel Sonuçlar

Şekil 7.4'de 24.4kHz'lik anahtarlama frekansı ve D = 0.25 oranı için PWM sinyali, Vo çıkış gerilimi ve VCE gerilimi görülmektedir. Anahtar üzerine düşen maksimum gerilimin tepeden-tepeye değeri 28.60 volt ve çıkış geriliminin 5.96 volt olduğu görülmektedir.

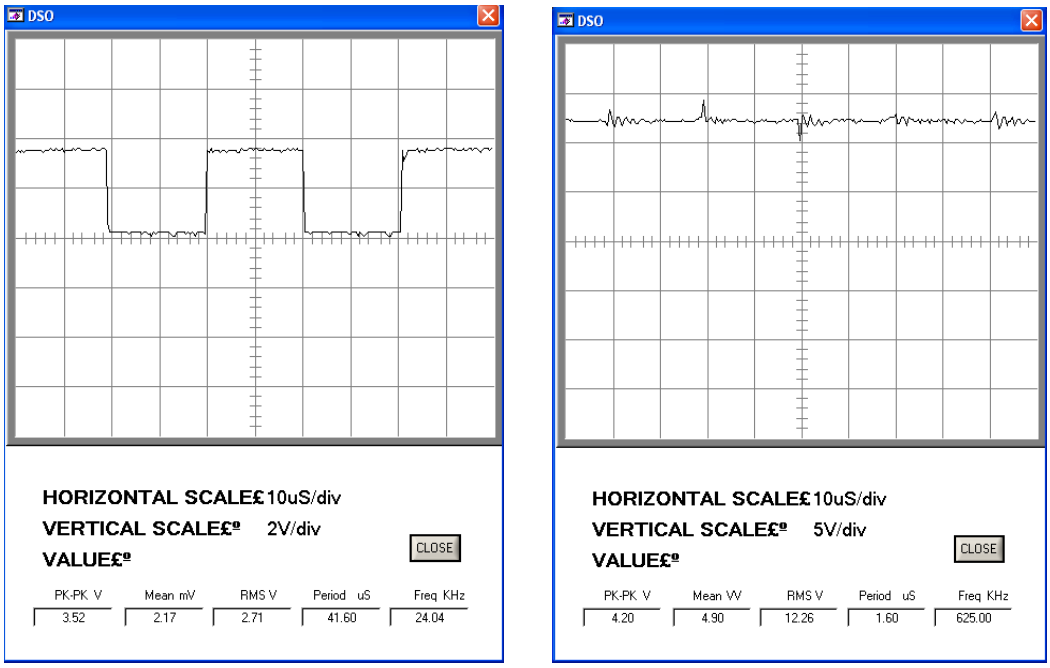


(a) **(b)**

(c)

Şekil 7.4. RL=30ohm, D=0.25 (a) PWM sinyali, (b) Vo çıkış gerilimi, (c) VCE gerilimi

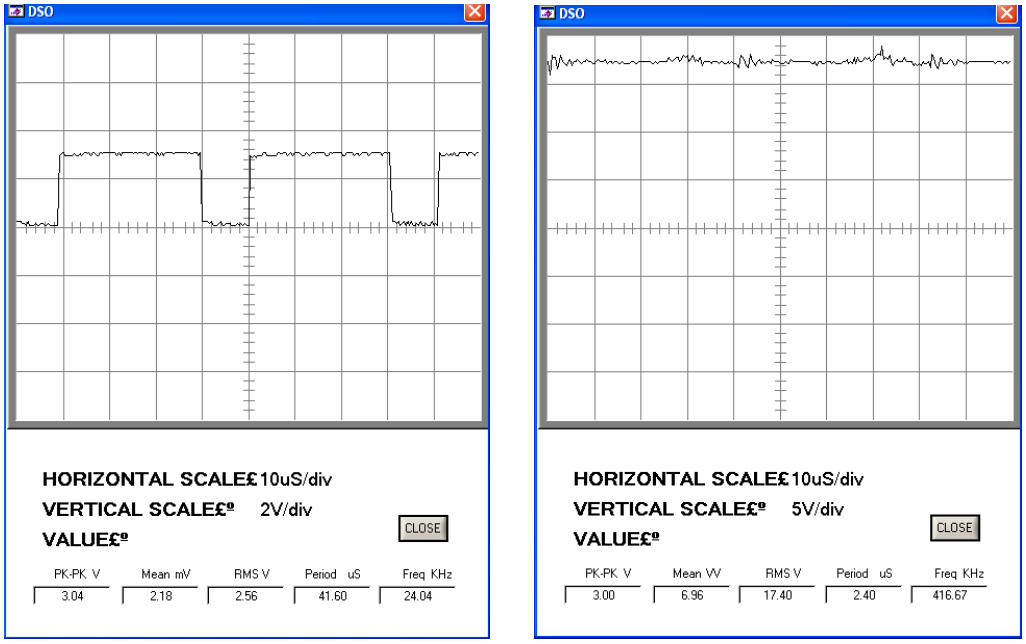
Şekil 7.5'de 24.4kHz'lik anahtarlama frekansı ve D = 0.5 oranı için PWM sinyali ve Vo çıkış gerilimi görülmektedir. Yük üzerine düşen gerilim değeri 12.26 volttur.



(a) **(b)**

Şekil 7.5. R_L=30ohm, D=0.5 (a) PWM sinyali, (b) Vo çıkış gerilimi

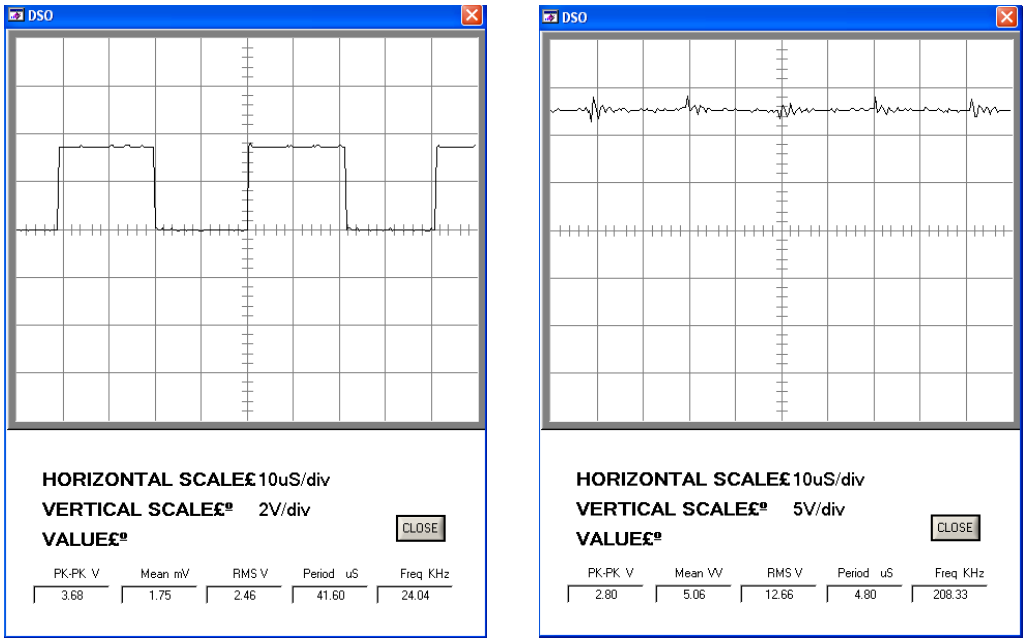
Şekil 7.6'da 24.4kHz'lik anahtarlama frekansı ve D = 0.75 oranı için PWM sinyali ve Vo çıkış gerilimi görülmektedir. Çıkış gerilim değeri 17.40 volttur.



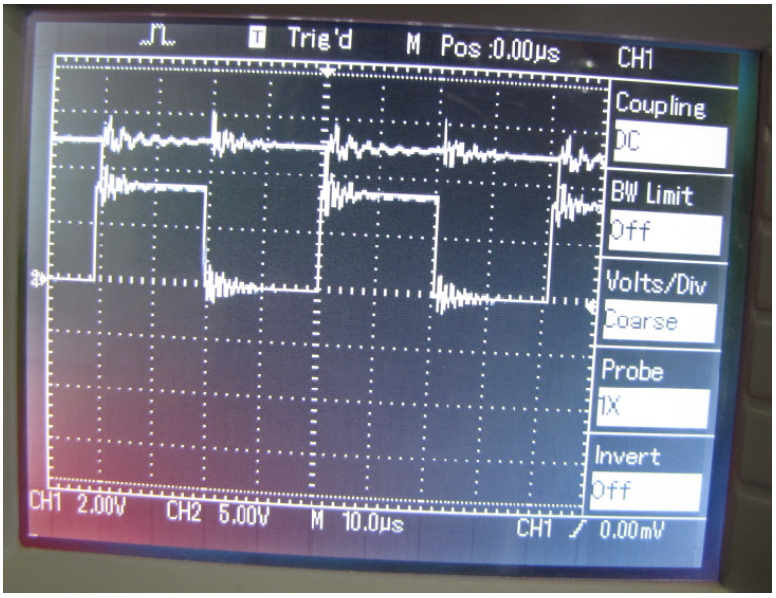
(a) **(b)**

Şekil 7.6. R_L=30ohm, D=0.75 (a) PWM sinyali, (b) Vo çıkış gerilimi

Şekil 7.7'de 24.4kHz'lik anahtarlama frekansı ve PI denetimi uygulanan devre için PWM sinyali ve Vo çıkış gerilimi ve osilaskop görüntüsü verilmiştir. PI denetleyicisi, PWM görev oranını 0.5 seviyelerinde tutarak yük üzerine düşen gerilim değerinin 12 volt seviyelerinde tutmaktadır.



(a) **(b)**

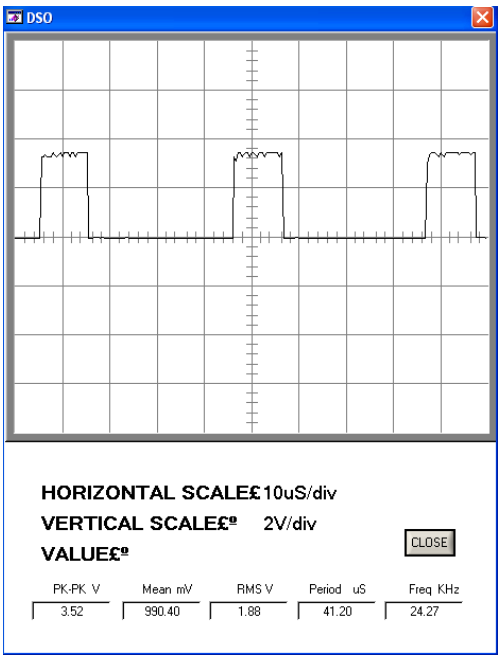


(c)

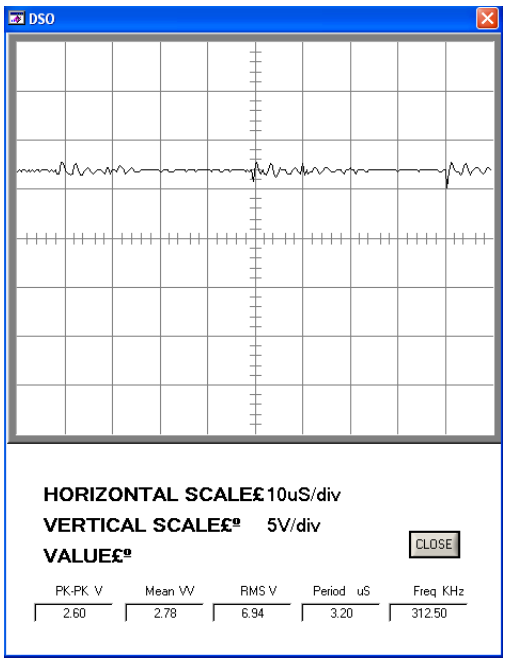
Şekil 7.7 R_L=30ohm (a) PWM sinyali, (b) Vo çıkış gerilimi, (c) Osilaskop görüntüsü

7.2. R_L=50ohm Yük Değeri İçin Deneysel Sonuçlar

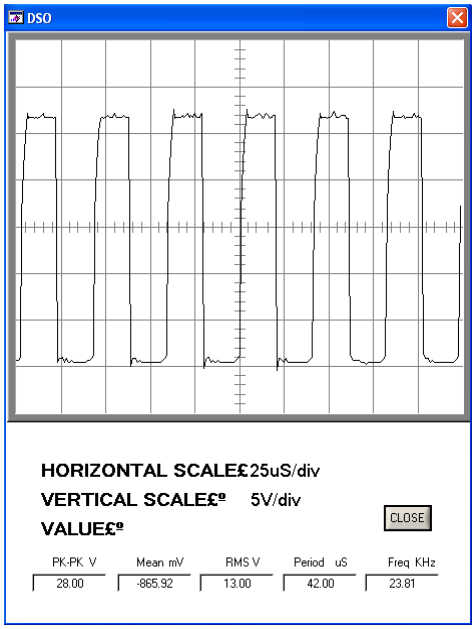
Şekil 7.8'de 24.4kHz'lik anahtarlama frekansı ve D = 0.25 oranı için PWM sinyali, Vo çıkış gerilimi ve V_{CE} gerilimi görülmektedir. Çıkış geriliminin 6.94 volt olduğu görülmektedir.



(a)



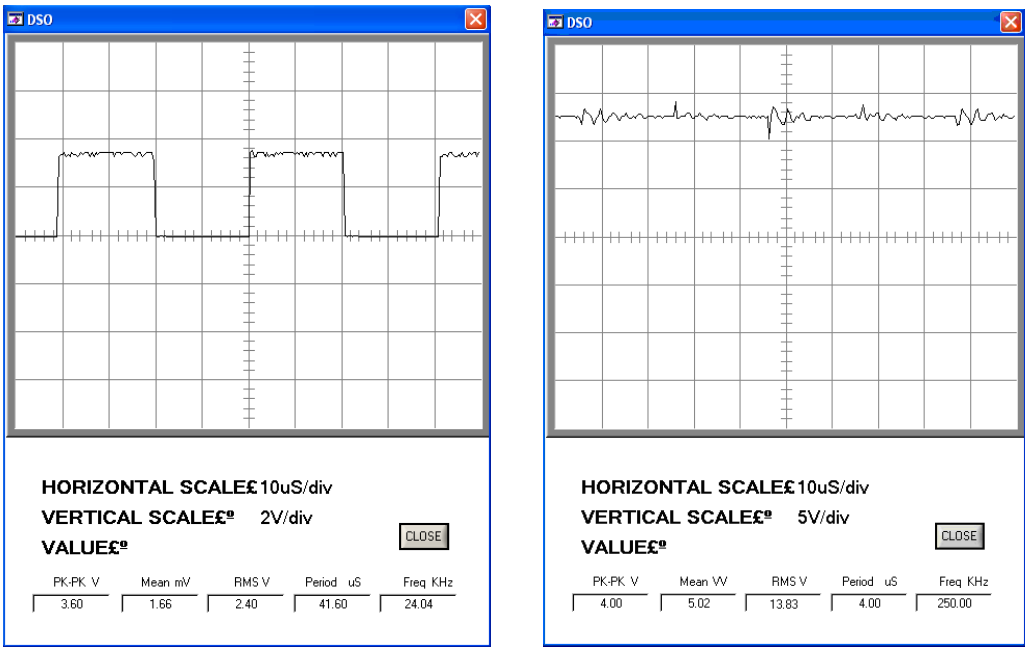
(b)



(c)

Şekil 7.8. R_L=50ohm, D=0.25 (a) PWM sinyali, (b) Vo çıkış gerilimi, (c) V_{CE} gerilimi

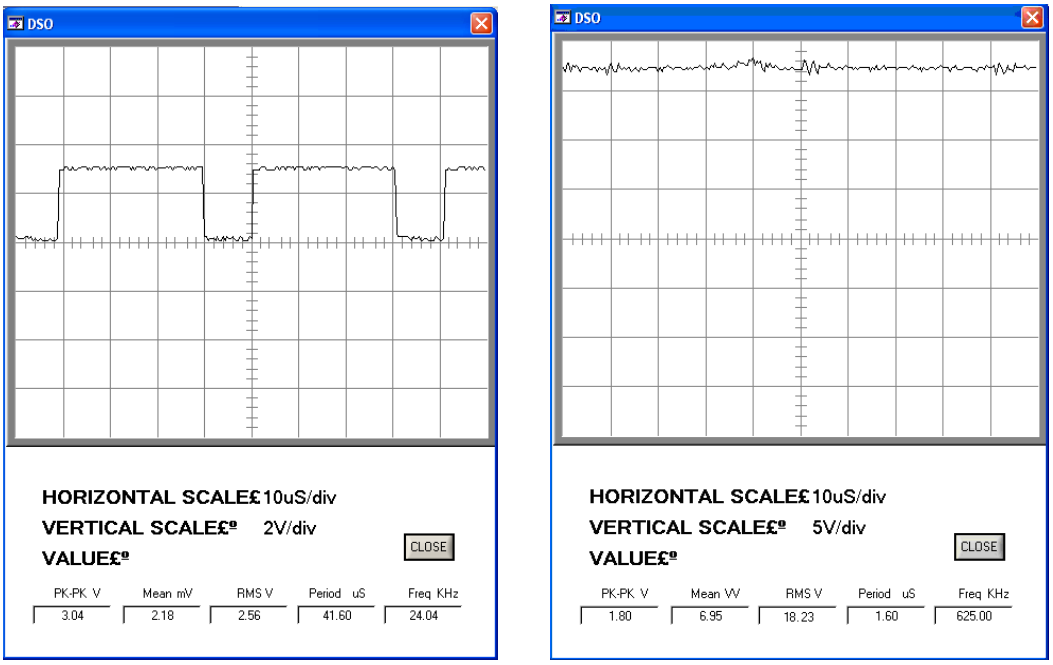
Şekil 7.9'da 24.4kHz'lik anahtarlama frekansı ve D = 0.5 oranı için PWM sinyali ve Vo çıkış gerilimi görülmektedir. Yük üzerine düşen gerilim değeri 13.83 volttur.



(a) **(b)**

Şekil 7.9. R_L=50ohm, D=0.5 (a) PWM sinyali, (b) Vo çıkış gerilimi

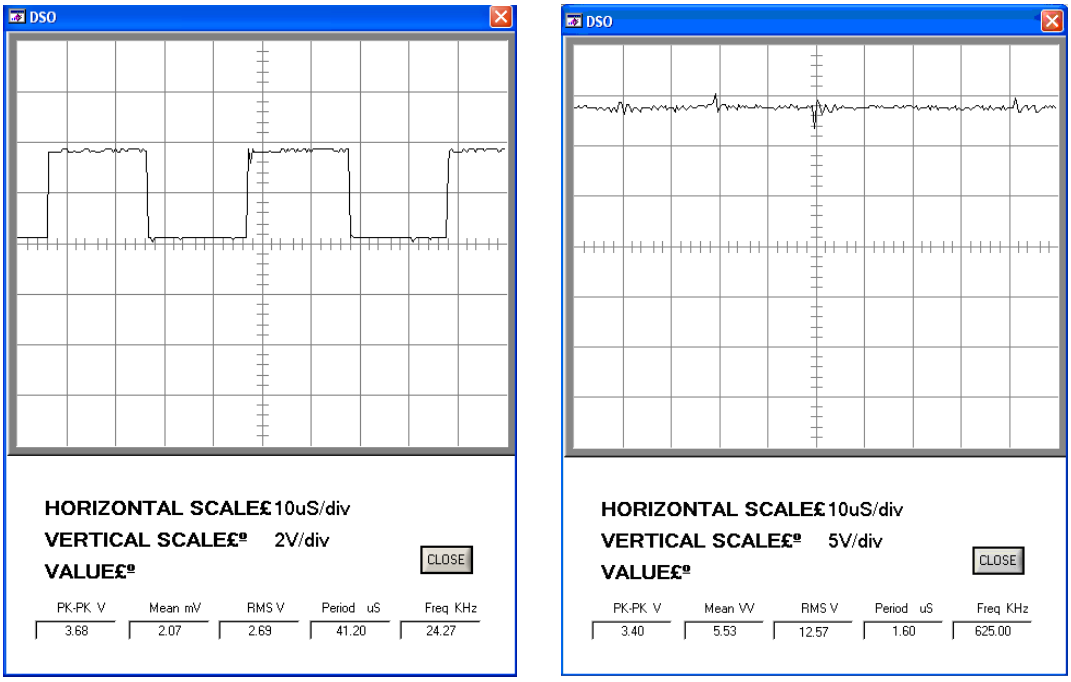
Şekil 7.10'da 24.4kHz'lik anahtarlama frekansı ve D = 0.75 oranı için PWM sinyali ve Vo çıkış gerilimi görülmektedir. Çıkış gerilim değerinin 18.23 volt olduğu görülmektedir.



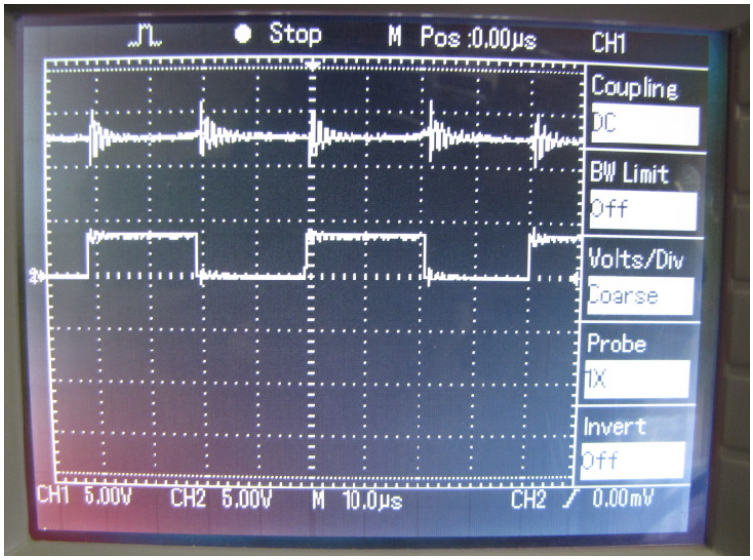
(a) **(b)**

Şekil 7.10. R_L=50ohm, D=0.75 (a) PWM sinyali, (b) Vo çıkış gerilimi

Şekil 7.11'de 24.4kHz'lik anahtarlama frekansı ve PI denetimi uygulanan devre için PWM sinyali ve Vo çıkış gerilimi ve osilaskop görüntüsü görülmektedir. PI denetleyicisi, PWM görev oranını 0.5 seviyelerinde tutmaktadır. Yük değişse bile yük üzerine düşen gerilim değeri 12 volt seviyelerinde olmaktadır.



(a) **(b)**



(c)

Şekil 7.11. R_L=50ohm (a) PWM sinyali, (b) Vo çıkış gerilimi, (c) Osilaskop görüntüsü

8. SONUÇLAR

Bu tez çalışmasının amacı, düşürücü DA-DA dönüştürücü tasarlamak ve uygulama devresini gerçekleştirmektir. Anahtarlama elemanı için gerekli olan PWM sinyal ve dönüştürücünün denetimi FPGA elemanı ile sağlanmıştır.

Bu çalışmada, ilk olarak denetimsiz ve sabit görev oranlı PWM sinyal değeri üretilmiş ve 30ohm ve 50ohm'luk yük değerleri için sonuçlar incelenmiştir. Sonuçlar göstermiştir ki çıkış gerilim değeri yük değişiminde etkilenmektedir. Çıkış gerilim değerini sabit tutabilmek için PI denetimi gerçekleştirilmiştir. PI denetimi, çıkış akım ve gerilim değerini sabit tutacak şekilde PWM görev oran değeri sürekli olarak değiştirilmiş ve bu şekilde çıkışın yük değişiminden etkilenmemesi sağlanmıştır.

Bu çalışmada, gerçekleştirilen donanım oldukça esnek bir yapıya sahiptir. Sadece FPGA'ya yüklenen kodlarda yapılacak olan birkaç değişiklik ile gerçekleştirilmek istenen donanım kısa bir zaman içerisinde değiştirilebilmektedir. Günümüzde kullanılan mikrodenetleyicilerin aksine komutları paralel işleme yeteneğine de sahip olduklarından FPGA ile gerçekleştirilen donanımların doğrulukları oldukça yüksek olmaktadır. Ayrıca, oldukça yüksek hızlarda anahtarlama sinyalleri, donanım gerçekleştirme sorunu olmadan birkaç kod dizisi üretimi ile gerçekleştirilebilmektedir.

KAYNAKLAR

[1] **Mohan, N., Undeland, T.M., Robbins, W.P.** 2007. "Güç Elektroniği, Çeviriciler, Uygulamalar ve Tasarım", Literatür Yayıncılık,

[2] **Erdoğan, E.** 2010. Dijital Kontrollü Çok Fazlı Senkronize DC-DC Alçaltıcı Çevirici Tasarımı, *Yüksek Lisans Tezi,* Eskişehir Osmangazi Üniversitesi, Fen Bilimleri Enstitüsü, Elektronik Bilim Dalı, Eskişehir.

[3] **Pedroni, V. A.** 2004. Circuit Design with VHDL, MIT Press, 363s.

[4] **Dueck, R. K.** 2000. Digital Design with CPLD Applications and VHDL, Thomson Delmar Learning, 837s.

[5] **Wang, Q., Zhang, D.** 2008. All Digital DC/DC Converters on FPGA, International Conference on Intelligent Computation Technology and Automation (ICICTA), 11-15.

[6] **Başçıl, M.S.** 2011. FPGA tabanlı uzaktan erişilebilir sayısal sistem laboratuar prototipi tasarımı, *Yüksek Lisans Tezi,* Sakarya Üniversitesi, Fen Bilimleri Enstitüsü, Sakarya.

[7] **Milanovic, M., Truntic, M. ve Slibar, P.** 2005. FPGA Implementation of Digital Controller for DC-DC Converter, IEEE Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS'05)

[8] **Alvarez, J., Lago, A. ve Nogueiras, A.** 2006. FPGA Implementation of a Fuzzy Controller for Automobile DC-DC Converters, IEEE Conference on Industrial Electronics and Applications, 237-240

[9] **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.

[10] **Dousoky, G.M., Shoyama, M., Ninomiya, T.** 2009. A Novel Implementation of an FPGA-Based Controller for Conducted-Noise Reduction in Randomly Switched DC-DC Converters, IEEE Conference, 65-69

[11] **Iida, T.** 2009. Emulated Current Mode Control System for DC-DC Boost Converter using FPGA, IEEE Conference, 432-436

[12] **Bodur, H.** 2004. "Güç Elektroniği Endüstriyel Uygulamaları 1" YTÜ Ders Notları, YTÜ, İstanbul.

[13] **Rashid, M. H.** 2001. Power Electronics Handbook, University of West Florida, Pensacola, Florida, 895 p.

[14] **Agrawal, J. P.** 2001. Power Electronics Systems: Theory and Design, Prentice-Hall, Upper Saddle River, NJ.

[15] **Şahin, N.** 2006. Güç elektroniği devrelerinin bilgisayar destekli analizi, *Yüksek Lisans Tezi,* Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Elazığ.

[16] **Özkan, G.** 2007. Yakıt Pilleri Uygulamaları için Dijital Kontrollü D.A.-D.A. Dönüştürücü Devre Tasarımı, *Yüksek Lisans Tezi,* Eskişehir Osmangazi Üniversitesi, Fen Bilimleri Enstitüsü, Eskişehir.

[17] **Patella, B. J., Prodic, A., Zirger, A. and Maksimovic, D.** 2003. High-Frequency Digital PWM Controller IC for DC–DC Converters, IEEE Transactions On Power Electronics, 18, No. 1, 438-446.

[18] **Gacar, A**., 2009. FPGA tabanlı görüntü isleme arabirimi, *Yüksek Lisans Tezi,* Ege Üniversitesi, Fen Bilimleri Enstitüsü, İzmir.

[19] **Güdenler, İ.** 2010. Saklayıcı bellek mimarisinde, 16 bitlik, gömülü sistem (mikroişlemci) tasarımı ve sentezlenmesi, *Yüksek Lisans Tezi,* Dumlupınar Üniversitesi, Fen Bilimleri Enstitüsü, Kütahya.

[20] **Zeidman, B.** 2002. *Designing with FPGAs and CPLDs*, CMP Books, Kansas.

[21] **Brown, S., Rose, J.** "Architecture of FPGAs and CPLDs: A Tutorial", University of Toronto, IEEE Conference, 110-122

[22] **Smith, G.R.** 2010. FPGAs 101 Everything you need to know to get started, Newnes Press, 230s.

[23] **Cofer, R.C., and Harding, B. F.** 2006. *Rapid System Prototyping with FPGA's*, Elsevier Inc., Burlington.

[24] **Guoa, R.J., Youa, J.R., Zhoua, C., Chua, K., Curana, M., Diaoa, P.F., Godab, J., Krafta, B., Donalda, J.F.** 2005. A 10GHz 4:1 MUX and 1:4 DEMUX implemented by a Gigahertz SiGe FPGA for fast ADC, *Integration, the VLSI Journal*, 38, 525-540.

[25] **Öcal, F.** 2006. Güvenli iletişim için FPGA kullanarak şifreleme sistemi tasarımı ve gerçekleştirilmesi, *Yüksek Lisans Tezi*, Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Ankara.

[26] **Kuon, I., Tessier, R. and Rose, J.** 2008. *FPGA Architecture: Survey and Challenges*, the essence of knowledge, Boston.

[27] **Sırmaçek, B.** 2007. FPGA ile mobil robot için öğrenme algoritması modellenmesi, *Yüksek Lisans Tezi,* Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

[28] **Yılmaz, N.** 2008. Alan programlamalı kapı dizileri (FPGA) üzerinde bir yapay sinirağları (YSA)'nın tasarlanması ve donanım olarak gerçekleştirilmesi, *Yüksek lisans tezi,* Selçuk Üniversitesi, Fen Bilimleri Enstitüsü, Konya.

[29] **Alçin, Ö.F.** 2011. Alan programlanabilir kapı dizisi ile sigmadelta modülatörlerin gerçeklenmesi, *Yüksek Lisans Tezi,* Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Elazığ.

[30] **Maxfield, C.** 2004. "*Design Warriors Guide to FPGA*", Mentor Graphics Corporation and Xilinx, Inc.,

[31] **Dikmese, S.** 2007. Kablosuz haberleşme sistemlerinde FPGA uygulaması, *Yüksek lisans tezi,* Kocaeli Üniversitesi, Fen Bilimleri Enstitüsü, Kocaeli.

[32] **Özbey, R.S.** 2004. Bilgisayar aritmetik ünitelerinin tasarımı için VHDL tabanlı kütüphane geliştirilmesi, *Yüksek Lisans Tezi,* İstanbul Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

[33] **Douglas, L.P.** 1991. *VHDL*, McGraw-Hill Inc., California.

[34] **Enoch, O., Hwang, E.O.** 2006. *Digital logic and microprocessor design with VHDL*, Thomson, California.

[35] **Roth C.H.** 1998. Digital Systems Design Using VHDL, PWS Press, 470s.

[36] **Türk, M., Tuncer, S. ve Türk, M.A.** 2009. Sahada Programlanabilir Kapı Dizileri Kullanılarak İki Kanallı Darbe Genişlik Modülasyonlu Sinyallerin Üretimi: Bir H-Köprü Dönüştürücü Uygulaması. Fırat Üniversitesi, Mühendislik Bilimleri Dergisi

[37] **Elmas, Ç.**, 2007. Yapay Zeka Uygulamaları, Seçkin Yayıncılık. 289-290.

[38] **Uysal, A., Bay, Ö.F.** 2009. Elektroliz yapmak için pi denetimli senkron da-da dönüştürücü tasarımı, *5. Uluslararası İleri Teknolojiler Sempozyumu (IATS'09), 13–15 Mayıs 2009, Karabük, Türkiye.*

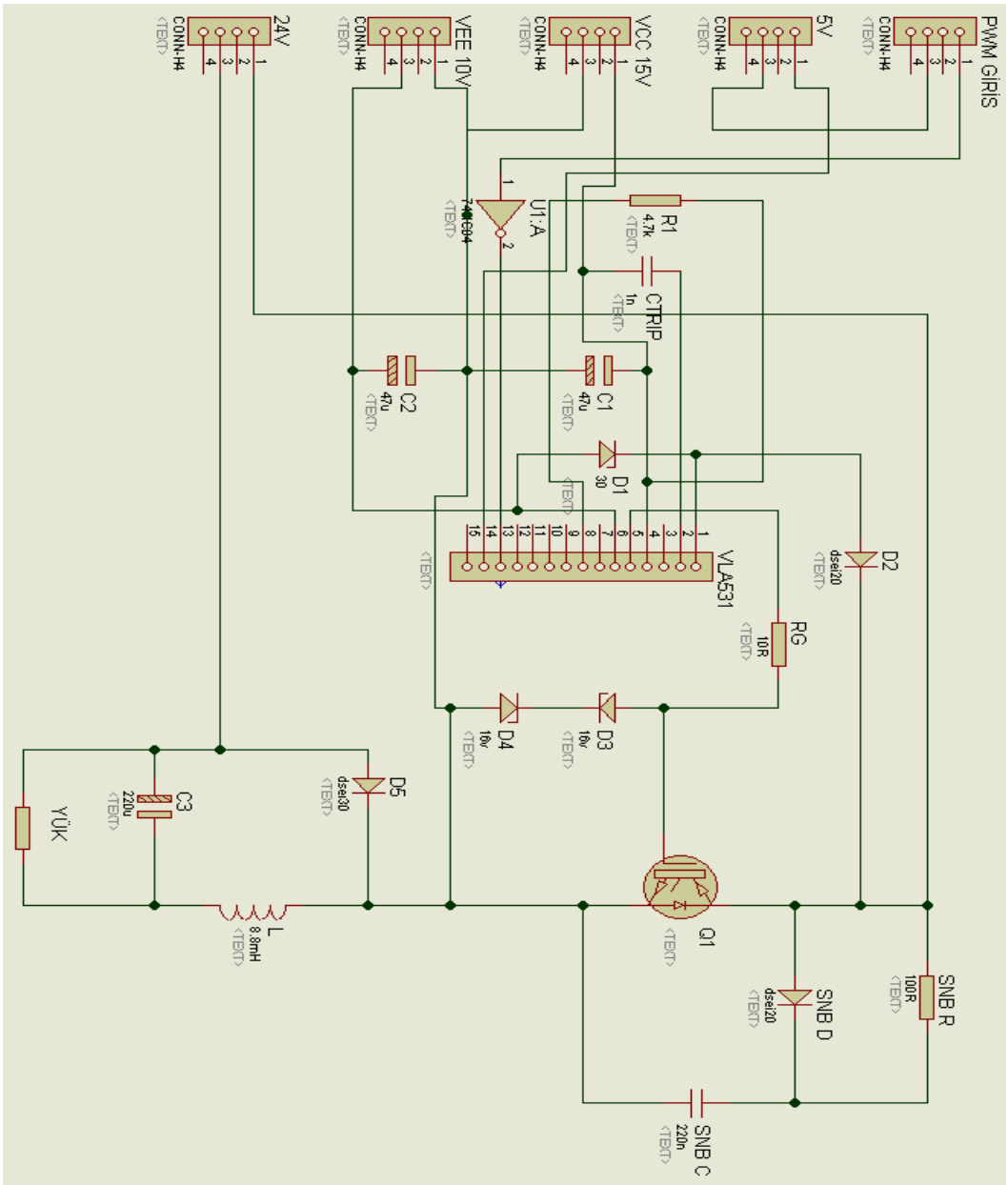
[39] **Fakhrulddin, H.A., Hussein M.M., Ismael S.M.B.** 2010. LabVIEW FPGA Implementation Of a PID Controller For D.C. Motor Speed Control, IEEE Conference, 139-144

[40] "Spartan-3E Starter Kit Board User Guide", UG230 (v1.0) 9 March 2006, Xilinx.

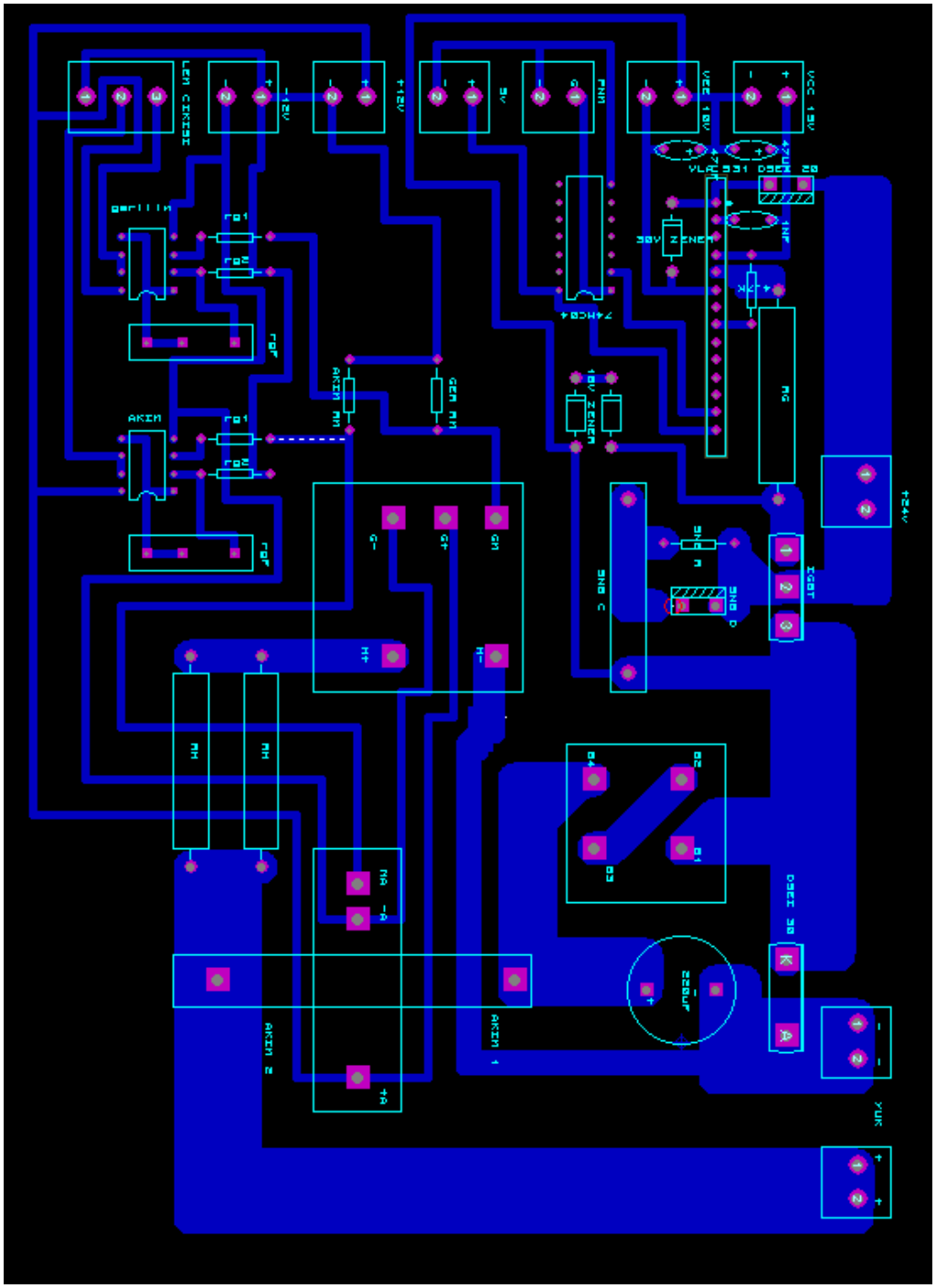
EKLER

EK A

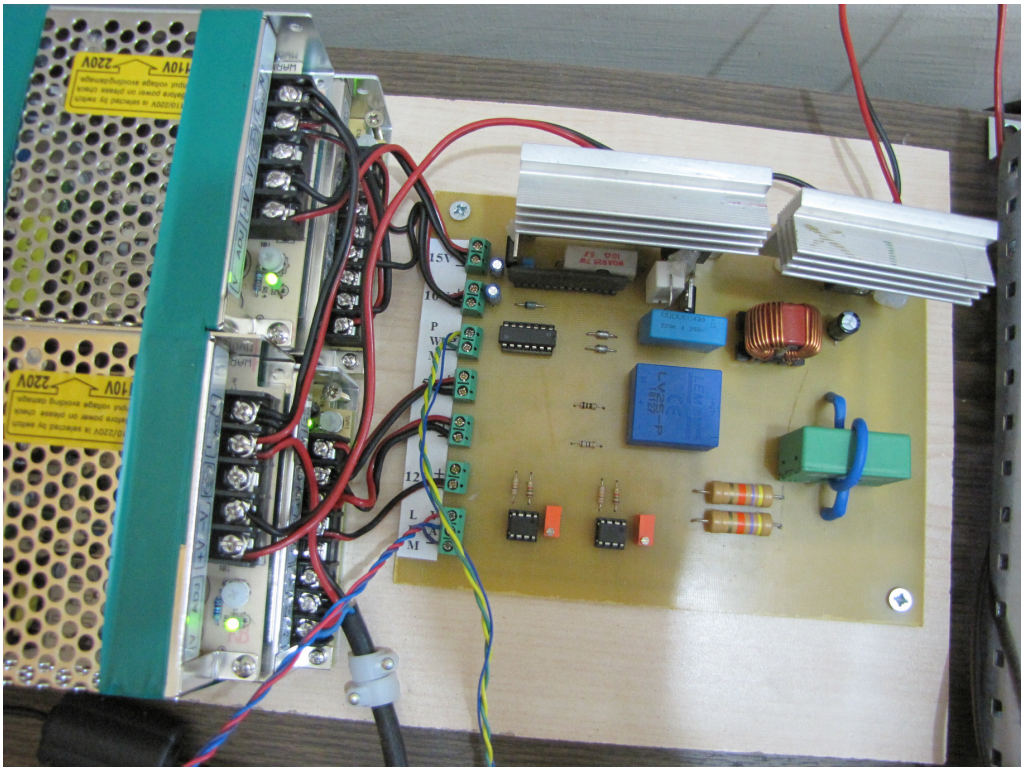
Tasarım tamamlandıktan sonra devrenin Ek A Şekil 1'de verilmiş olan şeması ve Ek A Şekil 2'de verilmiş olan baskı devre şeması Proteus programı kullanılarak çizilmiştir. Devrenin fotoğrafı Ek A Şekil 3'de verilmiştir.



Ek A Şekil 1. Düşürücü DA-DA dönüştürücü devre şeması



Ek A Şekil 2. Düşürücü DA-DA dönüştürücü baskı devre şeması



Ek A Şekil 3. Devre fotoğrafı

EK B

VHDL (Very High Speed Integrated Circuit Hardware Description Language) dili ile yazdığım ve FPGA (Field Programmable Gate Arrays) yüklediğim komutlar.

----------ANA PROGRAM----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity main is

Port (   CLK_50MHZ : in STD_LOGIC;
         SF_CE0          :inout std_logic;
         SPI_MISO : inout std_logic;
          AD_CONV : out  STD_LOGIC;
          AMP_CS : out  STD_LOGIC;
          SPI_MOSI : out  STD_LOGIC;
          SPI_SCK : out  STD_LOGIC;
          output_pwm : out  STD_LOGIC ;
          lcd_kume  :inout std_logic_vector(6 downto 0);
          AMP_SHDN:OUT STD_LOGIC;
          AMP_DOUT:IN STD_LOGIC;
          SPI_SS_B:OUT STD_LOGIC;
          DAC_CS:OUT STD_LOGIC;
          SF_WE: out std_logic;
          SF_OE: out std_logic;
          FPGA_INIT_B:OUT STD_LOGIC );
end main;
```

```
architecture Behavioral of main is
component aziz_adc_control
        Port (    CLK_50MHZ : in STD_LOGIC;
                  AD_CONV : out  STD_LOGIC;
                  SPI_SCK : out  STD_LOGIC;
                  SPI_MISO : inout  STD_LOGIC;
                  SPI_MOSI : out  STD_LOGIC;
                 AMP_CS : out  STD_LOGIC;
                AMP_DOUT : in  STD_LOGIC;
                 AMP_SHDN : out std_logic ;
                SPI_SS_B: out std_logic;
                 DAC_CS: out std_logic;
                 SF_CE0: inout std_logic;
                 FPGA_INIT_B: out std_logic;
                 SF_WE: out std_logic;
                 SF_OE: out std_logic;
                 ADC0_OUT : out  signed (13 downto 0):= (others => '0');
                 ADC1_OUT : out  signed (13 downto 0):= (others => '0'));
end component;

component aziz_lcd_control
        port (   clk50MHz  :in std_logic;
                lcd_kume  :inout std_logic_vector(6 downto 0));
end component;

component aziz_pwm_control
        port(     clk      :in std_logic ;
                 o_pwm :out std_logic ;
                  i_dutycycle : in std_logic_vector(10 downto 0));
end component;

signal counter : std_logic_vector (10 downto 0):=(others =>'0');  --Frenkans belirlenir 11
bit 24.4kHz dir. Her bit artışı fr ikiye böler.
```

```
signal hata, hata2 : integer range -200 to 200 := 0 ;
signal e_hata, e_hata2 : integer range -200 to 200 := 0 ;
signal ei_hata, ei_hata2 : integer range -200 to 200 := 0 ;
signal upi,upi2 : integer range -200 to 200 := 0 ;
constant Kp  : integer := 8 ;
constant Ki  : integer := 3 ;
constant Kp2 : integer := 5 ;
constant Ki2 : integer := 2 ;
constant ref : integer := 0 ;
constant eksi : integer := -1 ;
signal cnt :integer range 0 to 9 := 0 ;
signal ADC0_OUTPUT, ADC1_OUTPUT : signed(13 downto 0) := (others => '0');
signal ADC0_OUTPUT_0, ADC1_OUTPUT_1 : signed(5 downto 0) := (others => '0');
signal duty : std_logic_vector (10 downto 0 ) ;
signal upii :integer range 0 to 2 := 0 ;
begin

adc: aziz_adc_control
        port map (
                CLK_50MHZ => CLK_50MHZ,
                AD_CONV => AD_CONV,
                SPI_SCK => SPI_SCK,
                SPI_MISO => SPI_MISO,
                SPI_MOSI => SPI_MOSI,
                AMP_CS => AMP_CS,
                AMP_DOUT => AMP_DOUT,
                AMP_SHDN => AMP_SHDN,
                SPI_SS_B => SPI_SS_B,
                DAC_CS       => DAC_CS,
                SF_CE0 => SF_CE0,
                SF_WE => SF_WE,
                SF_OE => SF_OE,
                FPGA_INIT_B => FPGA_INIT_B,
```

```
                ADC0_OUT => ADC0_OUTPUT,
                ADC1_OUT => ADC1_OUTPUT );

lcd : aziz_lcd_control
        port map (      clk50MHz => CLK_50MHZ,
                        lcd_kume => lcd_kume );

pwm : aziz_pwm_control
        port map (      clk => CLK_50MHZ,
                        o_pwm => output_pwm,
                        i_dutycycle => duty );

pi1: process (CLK_50MHZ)
begin
if CLK_50MHZ' event and CLK_50MHZ = '1' then
cnt <= cnt + 1;
case cnt is
        when 0 =>
                                ADC0_OUTPUT_0(0) <= ADC0_OUTPUT (8);
                                ADC0_OUTPUT_0(1) <= ADC0_OUTPUT (9);
                                ADC0_OUTPUT_0(2) <= ADC0_OUTPUT (10);
                                ADC0_OUTPUT_0(3) <= ADC0_OUTPUT (11);
                                ADC0_OUTPUT_0(4) <= ADC0_OUTPUT (12);
                                ADC0_OUTPUT_0(5) <= ADC0_OUTPUT (13);
                                --ADC0_OUTPUT_0(6) <= ADC0_OUTPUT (13);
                                --ADC0_OUTPUT_0(7) <= ADC0_OUTPUT (13);
                                ADC1_OUTPUT_1(0) <= ADC1_OUTPUT (8);
                                ADC1_OUTPUT_1(1) <= ADC1_OUTPUT (9);
                                ADC1_OUTPUT_1(2) <= ADC1_OUTPUT (10);
                                ADC1_OUTPUT_1(3) <= ADC1_OUTPUT (11);
                                ADC1_OUTPUT_1(4) <= ADC1_OUTPUT (12);
                                ADC1_OUTPUT_1(5) <= ADC1_OUTPUT (13);
                                --ADC1_OUTPUT_1(6) <= ADC1_OUTPUT (13);
```

```
                        --ADC1_OUTPUT_1(7) <= ADC1_OUTPUT (13);
        when 1 =>
                        hata <= ref - ( eksi *  (conv_integer (ADC0_OUTPUT_0))) ;
        when 2 =>
                        upi <= (Kp * hata) + (ei_hata + (Ki * (hata + e_hata)));
        when 3 =>
                        ei_hata <= ei_hata + Ki *( hata + e_hata);
        when 4 =>
                        e_hata <= hata ;
        when 5 =>
                    hata2 <= upi - ( eksi * ( conv_integer (ADC1_OUTPUT_1))) ;
        when 6 =>
                   upi2 <= (Kp2 * hata2) + (ei_hata2 + (Ki2 * (hata2 + e_hata2)));
        when 7 =>
                        ei_hata2 <= ei_hata2 + Ki2 *( hata2 + e_hata2);
        when 8  =>
                        e_hata2 <= hata2 ;
        when others =>

                        cnt <= 0;
        end case;
end if;
end process pi1;

aa: process (upi2)
begin
        if upi2 = 0 then
                upii <= 0;
        else
          if ADC0_OUTPUT_0(5) = '0' then
                upii <= 2;
          else
                upii <= 1;
```

```
            end if;
          end if;
    end process aa;

d: process (CLK_50MHZ)
begin
   if CLK_50MHZ = '1' and upii = 0 then
                    duty <= "10000000000" ;
          elsif CLK_50MHZ = '1' and upii = 1 then
                    duty <= duty - "00000000001";
          elsif CLK_50MHZ = '1' and upii = 2 then
                    duty <= duty + "00000000001";
          end if;
      end process d;
end Behavioral;
```

----------ADC PROGRAMI----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity aziz_adc_control is

Port (          CLK_50MHZ : in STD_LOGIC;
                 AD_CONV : out  STD_LOGIC;
                 SPI_SCK : out  STD_LOGIC;
                 SPI_MISO : inout  STD_LOGIC;
                 SPI_MOSI : out  STD_LOGIC;
                AMP_CS : out  STD_LOGIC;
                AMP_DOUT : in  STD_LOGIC;
                AMP_SHDN : out std_logic ;
                SPI_SS_B: out std_logic;
                DAC_CS: out std_logic;
                SF_CE0: inout std_logic;
                FPGA_INIT_B: out std_logic;
                SF_WE: out std_logic;
                SF_OE: out std_logic;
                ADC0_OUT : out signed(13 downto 0) ;
                ADC1_OUT : out signed(13 downto 0) );
end entity ;

architecture Behavioral of aziz_adc_control is
        constant gain : std_logic_vector(7 downto 0) := "00010001";        -- kazanç (-1)
        signal count : integer range 0 to 2326 := 0;
        signal ADC0, ADC1 : signed(13 downto 0) := "00000000000000";
        begin
SPI_SS_B <= '1';
SF_CE0 <='1';                           --strataflash_ce
```

```
FPGA_INIT_B <='1';                  --platformflash_oe <= '0';
DAC_CS <= '1';
AMP_SHDN <= '0';
SF_OE <= '1';                       --strataflash_oe
SF_WE <= '1';                       --strataflash_we

process (CLK_50MHZ)
begin
        if CLK_50MHZ' event and CLK_50MHZ = '1' then
            count  <= count + 1 ;
        case count is
                when 0 to 50=>
                        AD_CONV <= '1';
                        AMP_CS <= '1' ;
                        SPI_SCK <= '0';
                        SPI_MISO <= 'Z';
                        SPI_MOSI <= 'Z';
                when 51 to 99 =>
                        AMP_CS <= '0' ;
                        SPI_SCK <= '0';
                when 100 to 110 =>          --KAZANÇ AYARI 1 MHZ
                        SPI_SCK <= '0';
                        SPI_MOSI <= gain(7);
                when 111 to 136 =>
                        SPI_SCK <= '1';
                        SPI_MOSI <= gain(7);
                when 137 to 152 =>
                        SPI_SCK <= '0';
                        SPI_MOSI <= gain(7);
                when 153 to 163 =>
                        SPI_SCK <= '0';
                        SPI_MOSI <= gain(6);
```

```
when 164 to 189 =>
                SPI_SCK <= '1';
                SPI_MOSI <= gain(6);
when 190 to 205 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(6);
when 206 to 216 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(5);
when 217 to 242 =>
                SPI_SCK <= '1';
                SPI_MOSI <= gain(5);
when 243 to 258 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(5);
when 259 to 269 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(4);
when 270 to 295 =>
                SPI_SCK <= '1';
                SPI_MOSI <= gain(4);
when 296 to 311 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(4);
when 312 to 322 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(3);
when 323 to 348 =>
                SPI_SCK <= '1';
                SPI_MOSI <= gain(3);
when 349 to 364 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(3);
```

```
when 365 to 375 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(2);
when 376 to 401 =>
                SPI_SCK <= '1';
                SPI_MOSI <= gain(2);
when 402 to 417 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(2);
when 418 to 428 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(1);
when 429 to 454 =>
                SPI_SCK <= '1';
                SPI_MOSI <= gain(1);
when 455 to 470 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(1);
when 471 to 481 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(0);
when 482 to 507 =>
                SPI_SCK <= '1';
                SPI_MOSI <= gain(0);
when 508 to 523 =>
                SPI_SCK <= '0';
                SPI_MOSI <= gain(0);
when 524 to 574 =>
                AD_CONV <= '1';
                AMP_CS <= '1' ;
                SPI_SCK <= '0';
```

```
                    SPI_MOSI <= 'Z';
                    SPI_MISO <='Z';

            when 575 to 625 =>                  --adc
                    AD_CONV <= '0';
                    SPI_SCK <= '0';
                    SPI_MISO <='Z';
            when 626 to 650 =>
                    SPI_SCK <= '1';    --25 adet 1 olacak 1 MHZ
                    SPI_MISO <='Z';
            when 651 to 675 =>
                    SPI_SCK <= '0';    --25 adet 0 olacak 1 MHZ
                    SPI_MISO <='Z';
            when 676 to 700 =>
                    SPI_SCK <= '1';
                    SPI_MISO <='Z';
            when 701 to 725 =>
                    SPI_SCK <= '0';
                    SPI_MISO <='Z';
            when 726 to 750 =>
                    SPI_SCK <= '1';
                    ADC0(13) <= SPI_MISO;     --KANAL  0
        when 751 to 775 =>
                    SPI_SCK <= '0';
                    ADC0(13) <= SPI_MISO;
            when 776 to 800 =>
                    SPI_SCK <= '1';
                    ADC0(12) <= SPI_MISO;
            when 801 to 825 =>
                    SPI_SCK <= '0';
                    ADC0(12) <= SPI_MISO;
```

```
when 826 to 850 =>
                SPI_SCK <= '1';
                ADC0(11) <= SPI_MISO;
when 851 to 875 =>
                SPI_SCK <= '0';
                ADC0(11) <= SPI_MISO;
when 876 to 900 =>
                SPI_SCK <= '1';
                ADC0(10) <= SPI_MISO;
when 901 to 925 =>
                SPI_SCK <= '0';
                ADC0(10) <= SPI_MISO;
when 926 to 950 =>
                SPI_SCK <= '1';
                ADC0(9) <= SPI_MISO;
when 951 to 975 =>
                SPI_SCK <= '0';
                ADC0(9) <= SPI_MISO;
when 976 to 1000 =>
                SPI_SCK <= '1';
                ADC0(8) <= SPI_MISO;
when 1001 to 1025 =>
                SPI_SCK <= '0';
                ADC0(8) <= SPI_MISO;
when 1026 to 1050 =>
                SPI_SCK <= '1';
                ADC0(7) <= SPI_MISO;
when 1051 to 1075 =>
                SPI_SCK <= '0';
                ADC0(7) <= SPI_MISO;
when 1076 to 1100 =>
                SPI_SCK <= '1';
                ADC0(6) <= SPI_MISO;
```

```
when 1101 to 1125 =>
                SPI_SCK <= '0';
                ADC0(6) <= SPI_MISO;
when 1126 to 1150 =>
                SPI_SCK <= '1';
                ADC0(5) <= SPI_MISO;
when 1151 to 1175 =>
                SPI_SCK <= '0';
                ADC0(5) <= SPI_MISO;
when 1176 to 1200 =>
                SPI_SCK <= '1';
                ADC0(4) <= SPI_MISO;
when 1201 to 1225 =>
                SPI_SCK <= '0';
                ADC0(4) <= SPI_MISO;
when 1226 to 1250 =>
                SPI_SCK <= '1';
                ADC0(3) <= SPI_MISO;
when 1251 to 1275 =>
                SPI_SCK <= '0';
                ADC0(3) <= SPI_MISO;
when 1276 to 1300 =>
                SPI_SCK <= '1';
                ADC0(2) <= SPI_MISO;
when 1301 to 1325 =>
                SPI_SCK <= '0';
                ADC0(2) <= SPI_MISO;
when 1326 to 1350 =>
                SPI_SCK <= '1';
                ADC0(1) <= SPI_MISO;
```

```
when 1351 to 1375 =>
                SPI_SCK <= '0';
                ADC0(1) <= SPI_MISO;
when 1376 to 1400 =>
                SPI_SCK <= '1';
                ADC0(0) <= SPI_MISO;
when 1401 to 1425 =>
                SPI_SCK <= '0';
                ADC0(0) <= SPI_MISO;
when 1426 to 1450 =>
                SPI_SCK <= '1';   --25 adet 1 olacak 1 MHZ
                SPI_MISO <='Z';
when 1451 to 1475 =>
                SPI_SCK <= '0';   --25 adet 0 olacak 1 MHZ
                SPI_MISO <='Z';
when 1476 to 1500 =>
                SPI_SCK <= '1';
                SPI_MISO <='Z';
when 1501 to 1525 =>
                SPI_SCK <= '0';
                SPI_MISO <='Z';
when 1526 to 1550 =>
                SPI_SCK <= '1';
                ADC1(13) <= SPI_MISO;   --KANAL 1
when 1551 to 1575 =>
                SPI_SCK <= '0';
                ADC1(13) <= SPI_MISO;
when 1576 to 1600 =>
                SPI_SCK <= '1';
                ADC1(12) <= SPI_MISO;
```

```
when 1601 to 1625 =>
                SPI_SCK <= '0';
                ADC1(12) <= SPI_MISO;
when 1626 to 1650 =>
                SPI_SCK <= '1';
                ADC1(11) <= SPI_MISO;
when 1651 to 1675 =>
                SPI_SCK <= '0';
                ADC1(11) <= SPI_MISO;
when 1676 to 1700 =>
                SPI_SCK <= '1';
                ADC1(10) <= SPI_MISO;
when 1701 to 1725 =>
                SPI_SCK <= '0';
                ADC1(10) <= SPI_MISO;
when 1726 to 1750 =>
                SPI_SCK <= '1';
                ADC1(9) <= SPI_MISO;
when 1751 to 1775 =>
                SPI_SCK <= '0';
                ADC1(9) <= SPI_MISO;
when 1776 to 1800 =>
                SPI_SCK <= '1';
                ADC1(8) <= SPI_MISO;
when 1801 to 1825 =>
                SPI_SCK <= '0';
                ADC1(8) <= SPI_MISO;
when 1826 to 1850 =>
                SPI_SCK <= '1';
                ADC1(7) <= SPI_MISO;
when 1851 to 1875 =>
                SPI_SCK <= '0';
                ADC1(7) <= SPI_MISO;
```

```
when 1876 to 1900 =>
                SPI_SCK <= '1';
                ADC1(6) <= SPI_MISO;
when 1901 to 1925 =>
                SPI_SCK <= '0';
                ADC1(6) <= SPI_MISO;
when 1926 to 1950 =>
                SPI_SCK <= '1';
                ADC1(5) <= SPI_MISO;
when 1951 to 1975 =>
                SPI_SCK <= '0';
                ADC1(5) <= SPI_MISO;
when 1976 to 2000 =>
                SPI_SCK <= '1';
                ADC1(4) <= SPI_MISO;
when 2001 to 2025 =>
                SPI_SCK <= '0';
                ADC1(4) <= SPI_MISO;
when 2026 to 2050 =>
                SPI_SCK <= '1';
                ADC1(3) <= SPI_MISO;
when 2051 to 2075 =>
                SPI_SCK <= '0';
                ADC1(3) <= SPI_MISO;
when 2076 to 2100 =>
                SPI_SCK <= '1';
                ADC1(2) <= SPI_MISO;
when 2101 to 2125 =>
                SPI_SCK <= '0';
                ADC1(2) <= SPI_MISO;
```

```
when 2126 to 2150 =>
                SPI_SCK <= '1';
                ADC1(1) <= SPI_MISO;
when 2151 to 2175 =>
                SPI_SCK <= '0';
                ADC1(1) <= SPI_MISO;
when 2176 to 2200 =>
                SPI_SCK <= '1';
                ADC1(0) <= SPI_MISO;
when 2201 to 2225 =>
                SPI_SCK <= '0';
                ADC1(0) <= SPI_MISO;
when 2226 to 2250 =>
                SPI_SCK <= '1';
                SPI_MISO <='Z';
when 2251 to 2275 =>
                SPI_SCK <= '0';
                SPI_MISO <='Z';
when 2276 to 2300 =>
                SPI_SCK <= '1';
                SPI_MISO <='Z';
when 2301 to 2325 =>
                SPI_SCK <= '0';
                SPI_MISO <='Z';
when others   =>
                AD_CONV <= '1';
                SPI_SCK <= '0';
                SPI_MOSI <='Z';
                AMP_CS <= '1' ;
                SPI_MISO <='Z';
                ADC0_OUT <= ADC0 ;
                ADC1_OUT <= ADC1 ;
                count <= 524 ;          --ADC'yi oku
```

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

---------LCD PROGRAMI------------------------

```
library IEEE;                           -- Kütüphane dosyaları
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity aziz_lcd_control is      --entity başlangıcı

Port (      clk50MHz        :in std_logic;              -- 50MHz clock
            lcd_kume        :inout std_logic_vector(6 downto 0));    --LCD control ve data
sinyallerini içerir.
end entity;
architecture Behavioral of aziz_lcd_control is
signal cntr       : std_logic_vector(28 downto 0):="00000000000000000000000000000";
--sayaç için başlangıç değeri
signal lcd_rsrwdl       : std_logic_vector(5 downto 0);
signal lcd_cdf          : std_logic_vector(6 downto 0);
signal lcd_stb          : std_logic;
signal enable           : std_logic := '1';
begin

process (clk50MHz)
begin
   if clk50MHz'event and clk50MHz='1' then
            if enable = '1' then                    -- DDRAM 'e Bir kere yazıldıktan
sonra programdan çıkılmasını sağlıyor
                    cntr  <= cntr + '1';            -- Sayac arttırılıyor
```

```
                --SF_ce0 <= '1';
                case cntr(28 downto 23) is     -- Sayaca göre yazma ve ayarlama
işlemleri yapılıyor
                        when "000000" =>       -- Bu kısma kada açılış beklemesi
yapılmış oluyor.
                                lcd_rsrwdl <= "000011" ;      -- Başlangıçta
yapılması gereken yüklemeler yapılıyor
                        when "000001" =>
                                lcd_rsrwdl <= "000011" ;
                        when "000010" =>
                                lcd_rsrwdl <= "000011" ;
                        when "000011" =>
                                lcd_rsrwdl <= "000010" ;
                        when "000100" =>
                                lcd_rsrwdl <= "000010" ;
                        when "000101" =>
                                lcd_rsrwdl <= "001000" ;
                        when "000110" =>
                                lcd_rsrwdl <= "000000" ;      -- Imlec otomatik
olarak sağa kayarken ekran hareketsiz.
                        when "000111" =>
                                lcd_rsrwdl <= "000110" ;


                        when "001000" =>
                                lcd_rsrwdl <= "000000" ;      -- Display on/
Imleç off/ yanıp sönme (blinking) off
                        when "001001" =>
                                lcd_rsrwdl <= "001100" ;
                        when "001010" =>
                                lcd_rsrwdl <= "000000" ; -- Display i temizle
                        when "001011" =>
                                lcd_rsrwdl <= "000001" ;      -- Bu kısma
kadar başlangıç işlemleri tamamlanıyor
```

```
                when "001100" =>
                        lcd_rsrwdl <= "100100" ;--A   -- Artık
karakterler yazılabilir
                when "001101" =>
                        lcd_rsrwdl <= "100001" ;
                when "001110" =>
                        lcd_rsrwdl <= "100111" ;--z    --(10) write data
(data yaz) 0111 (data msb)
                when "001111" =>
                        lcd_rsrwdl <= "101010" ;       --(10) write data
(data yaz) 1010 (data lsb)
                when "010000" =>
                        lcd_rsrwdl <= "100110" ;--i
                when "010001" =>
                        lcd_rsrwdl <= "101001" ;
                when "010010" =>
                        lcd_rsrwdl <= "100111" ;--z
                when "010011" =>
                        lcd_rsrwdl <= "101010" ;
                when "010100" =>
                        lcd_rsrwdl <= "101111" ;--bosluk
                when "010101" =>
                        lcd_rsrwdl <= "101110" ;
                when "010110" =>
                        lcd_rsrwdl <= "100100" ;--M
                when "010111" =>
                        lcd_rsrwdl <= "101101" ;
                when "011000" =>
                        lcd_rsrwdl <= "100100" ;--A
                when "011001" =>
                        lcd_rsrwdl <= "100001" ;
```

```
when "011010" =>
                lcd_rsrwdl <= "100100" ;--M
when "011011" =>
                lcd_rsrwdl <= "101101" ;
when "011100" =>
                lcd_rsrwdl <= "100101" ;--U
when "011101" =>
                lcd_rsrwdl <= "100101" ;
when "011110" =>
                lcd_rsrwdl <= "100101" ;--R
when "011111" =>
                lcd_rsrwdl <= "100010" ;
when "100000" =>
                lcd_rsrwdl <= "001100";   -- DDRAM adresi
```
değiştiriliyor (Alt satıra ilgili yere geçiliyor)
```
when "100001" =>
                lcd_rsrwdl <= "000000";
when "100010" =>
                lcd_rsrwdl <= "100100" ;--D
when "100011" =>
                lcd_rsrwdl <= "100100" ;
when "100100" =>
                lcd_rsrwdl <= "101111" ;--Ü
when "100101" =>
                lcd_rsrwdl <= "100101" ;
when "100110" =>
                lcd_rsrwdl <= "100101" ;--S
when "100111" =>
                lcd_rsrwdl <= "100011" ;
when "101000" =>
                lcd_rsrwdl <= "101111" ;--Ü
when "101001" =>
                lcd_rsrwdl <= "100101" ;
```

```
when "101010" =>
                lcd_rsrwdl <= "100101" ;--R
when "101011" =>
                lcd_rsrwdl <= "100010" ;
when "101100" =>
                lcd_rsrwdl <= "101111" ;--Ü
when "101101" =>
                lcd_rsrwdl <= "100101" ;
when "101110" =>
                lcd_rsrwdl <= "100100" ;--C
when "101111" =>
                lcd_rsrwdl <= "100011" ;
when "110000" =>
                lcd_rsrwdl <= "101111" ;--Ü
when "110001" =>
                lcd_rsrwdl <= "100101" ;
when "110010" =>
                lcd_rsrwdl <= "101111" ;--
when "110011" =>
                lcd_rsrwdl <= "101110" ;
when "110100" =>
                lcd_rsrwdl <= "100100" ;--D
when "110101" =>
                lcd_rsrwdl <= "100100" ;
when "110110" =>
                lcd_rsrwdl <= "100100" ;--A
when "110111" =>
                lcd_rsrwdl <= "100001" ;
when "111000" =>
                lcd_rsrwdl <= "101011" ;----
when "111001" =>
                lcd_rsrwdl <= "100000" ;
```

```
                        when "111010" =>
                                lcd_rsrwdl <= "100100" ;--D
                        when "111011" =>
                                lcd_rsrwdl <= "100100" ;
                        when "111100" =>
                                lcd_rsrwdl <= "100100" ;--A
                        when "111101" =>
                                lcd_rsrwdl <= "100001" ;
                        when "111111" =>
                                enable <= '0';  -- Case disable edilerek bir daha
aynı işlemleri yapması engelleniyor

                -------- Böylece yazılanlar FPGA ya başka bir program yüklense bile
elektrik kesilmedikçe LCD de kalıyor
                        when others   =>                                        ---
----- Bunun nedeni yazdıklarımızın DDRAM de korunuyor olması.
                                lcd_rsrwdl <= "010000" ;
                end case;

                lcd_stb <= (cntr(22) xor cntr(21)) and  not lcd_kume(4); --LCD
enable kontrol sinyali belirleniyor
                lcd_cdf  <= lcd_stb & lcd_rsrwdl;
        -- Enable kontrol sinyali veriye ekleniyor
                lcd_kume  <= lcd_cdf ;
                 -- Son olarak veri ve kontrol sinyalleri çıkış olarak LCD ye
gönderiliyor.
            end if;
        end if;
  end process;
end Behavioral;
```

-------PWM ÜRETEÇ PROGRAMI----------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity aziz_pwm_control is      -- PWM kontrolcüsü

  port( clk       :in std_logic := '0';   -- 50 MHz Saat Darbesi Girişi
        o_pwm :out std_logic := '0';     -- 1 bit PWM data çıkışı..
        i_dutycycle : in std_logic_vector(10 downto 0));         --ana  pr.  hesaplanan  D  oranı
girişi..
end entity;

architecture Behavioral of aziz_pwm_control is   -- 11 bit pwm sayacı
signal counter_pwm   : std_logic_vector(10 downto 0):= (OTHERS => '0');
signal dutycycle         : std_logic_vector(10 downto 0):= "10000000000" ;
begin
counter_pwm_p:process(clk) -- pwm peryot sayıcı
begin

  if(clk = '1' and clk'event) then
      counter_pwm <= counter_pwm + '1';
  end if;
end process;

dutycycle_p:process(clk)  --dutycyle periyot kontrol..(1 periyot boyunca sabit kalmalı..)
begin
  if(clk = '1' and clk'event) then
    if(counter_pwm = "11111111111") then
       dutycycle <= i_dutycycle;
            end if;
  end if;
```

```
end process;

o_pwm_p:process(clk) -- PWM data çıkışı..
begin
  if(clk = '1' and clk'event) then
     if(counter_pwm < dutycycle) then
        o_pwm <= '1';
      else
        o_pwm <= '0';
      end if;
  end if;
end process;
end architecture;
```

----------PİN TANIMLAMA KISMI (UCF)--------

```
1. NET "SF_CE0" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW =
SLOW ;
2. NET "CLK_50MHZ" PERIOD = 20.0ns HIGH 50%;
3. NET "CLK_50MHZ"  LOC = "C9" | IOSTANDARD = LVCMOS33 ;
4. NET "AD_CONV" LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ;
5. NET "SPI_SCK" LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 8 ;
6. NET "SPI_MISO" LOC = "N10" | IOSTANDARD = LVCMOS33 ;
7. NET "SPI_MOSI" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ;
8. NET "AMP_CS" LOC = "N7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ;
9. NET "AMP_SHDN" LOC = "P7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 6 ;
10. NET "AMP_DOUT" LOC = "E18" | IOSTANDARD = LVCMOS33 ;
```

11. NET "DAC_CS" LOC = "N8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
12. NET "FPGA_INIT_B" LOC = "T3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4 ;
13. NET "SPI_SS_B" LOC = "U3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
14. NET "SF_OE" LOC = "C18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
15. NET "SF_WE" LOC = "D17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
16. NET "lcd_kume<0>" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
17. NET "lcd_kume<1>" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
18. NET "lcd_kume<2>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
19. NET "lcd_kume<3>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
20. NET "lcd_kume<4>" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
21. NET "lcd_kume<5>" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
22. NET "lcd_kume<6>" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
23. NET "output_pwm" LOC = "D7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;

ÖZGEÇMİŞ

1973 yılında Elazığ'da doğdum. İlk, orta ve lise öğrenimini Elazığ'da tamamladım. 1996 yılında Fırat Üniversitesi Teknik Eğitim Fakültesi Elektronik ve Bilgisayar Eğitimi Bölümü'nü kazandım. 2000 yılında bu bölümden mezun oldum. Elazığ Ticaret Meslek Lisesinde Elektronik Öğretmeni olarak görev yapmaktayım. 2009 yılında Fırat Üniversitesi Fen Bilimleri Enstitüsü Elektronik ve Bilgisayar Eğitimi Ana Bilim Dalında yüksek lisans eğitimine başladım.