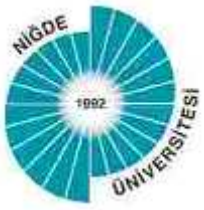
R. İNAN, 2011

YÜKSEK LİSANS TEZİ

NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ



T.C
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

ASENKRON MOTORLARIN HIZ-ALGILAYICISIZ KONTROLÜ İÇİN GENİŞLETİLMİŞ KALMAN FİLTRESİ TABANLI FPGA UYGULAMASI

REMZİ İNAN

Haziran 2011

T.C
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDSİLİĞİ ANABİLİM DALI

ASENKRON MOTORLARIN HIZ-ALGILAYICISIZ KONTROLÜ İÇİN GENİŞLETİLMİŞ KALMAN FİLTRESİ TABANLI FPGA UYGULAMASI

REMZİ İNAN

Yüksek Lisans Tezi

Danışman
Yrd. Doç. Dr. Murat BARUT

Haziran 2011

Remzi İNAN tarafından **Yrd. Doç. Dr. Murat BARUT** danışmanlığında hazırlanan **"Asenkron Motorların Hız-Algılayıcısız Kontrolü İçin Genişletilmiş Kalman Filtresi Tabanlı FPGA Uygulaması"** adlı bu çalışma jürimiz tarafından Niğde Üniversitesi Fen Bilimleri Enstitüsü **Elektrik Elektronik Mühendisliği** Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan : Prof. Dr. Recai KILIÇ (Erciyes Üniversitesi)

Üye : Yrd. Doç. Dr. Murat BARUT (Niğde Üniversitesi)

Üye : Yrd. Doç. Dr. Fuat KARAKAYA (Niğde Üniversitesi)

ONAY:

Bu tez, Fen Bilimleri Enstitüsü Yönetim Kurulunca belirlenmiş olan yukarıdaki jüri üyeleri tarafından …./…./20… tarihinde uygun görülmüş ve Enstitü Yönetim Kurulu'nun …./…./20… tarih ve ……sayılı kararıyla kabul edilmiştir.

…./…./20..

Doç. Dr. Nurettin ACIR

Enstitü Müdürü

ÖZET

ASENKRON MOTORLARIN HIZ-ALGILAYICISIZ KONTROLÜ İÇİN GENİŞLETİLMİŞ KALMAN FİLTRESİ TABANLI FPGA UYGULAMASI

İNAN, Remzi

Niğde Üniversitesi

Fen Bilimleri Enstitüsü

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Yrd. Doç. Dr. Murat BARUT

Haziran 2011, 83 Sayfa

Bu tez çalışmasında mevcut literatüre göre ilk kez, sincap kafesli asenkron motorların (SKASM'lerin) hız-algılayıcısız doğrudan vektör kontrolü için ölçülen stator akım ve gerilimlerini kullanarak stator akımı, rotor akısı, rotor açısal hızı ve yük momenti durumlarını eş zamanlı olarak kestiren bir genişletilmiş Kalman filtresi (GKF) algoritması, 100 MHz'lik saat frekansına sahip bir FPGA'de gerçekleştirilmiştir.

FPGA'nın yapısında bulunan paralel işlem yapma özelliği kullanılarak bu çalışma kapsamında gerçeklenen GKF algoritması ile 250 MHz'lik saat frekansına sahip DS1104 gibi diğer uygulama geliştirme donanımlarında gerçeklenen GKF algoritmasına kıyasla, yaklaşık 11 kat daha hızlı olarak gerçek-zamanlı kestirimler yapılabilmektedir. Yani, GKF gibi hesap yükü fazla ama matematiksel işlemleri paralelleştirilebilen algoritmaların, FPGA ortamında daha küçük örnekleme zamanı ile koşturulabileceği gerçek-zamanlı uygulamalarla gösterilmiş olmaktadır.

Diğer taraftan, FPGA'da geliştirilen GKF algoritmasının sonuçları, MATLAB m-file ortamında gerçeklenen aynı GKF algoritmasının sonuçları ile karşılaştırılarak doğrulanmıştır. Ayrıca, FPGA ile kişisel bilgisayar arasında çift yönlü bilgi alış verişi için seri haberleşme ve ethernet haberleşme protokolleri de gerçekleştirilerek, önerilen GKF algoritmasının sonuçları grafiksel olarak izlenebilir hale getirilmiştir. Bu yönleriyle literatürde bilinen ilk çalışmadır.

Anahtar sözcükler: Asenkron motor, Genişletilmiş Kalman filtresi, Hız-algılayıcısız kontrol, FPGA

SUMMARY

EXTENDED KALMAN FILTER BASED FPGA IMPLEMENTATION FOR SPEED-SENSORLESS CONTROL OF INDUCTION MOTORS

İNAN, Remzi

Nigde University

Graduate School of Natural and Applied Science

Department of Electrical-Electronics Engineering

Supervisor: Assistant Prof. Murat BARUT

June 2011, 83 Pages

In this thesis, according to the existing literature, an extended Kalman filter algorithm including the simultaneous estimations of stator currents, rotor flux, rotor angular velocity, and load torque by using the measured stator currents and voltages is the firstly implemented for the speed-sensorless direct vector control of squirrel cage induction motors (IMs) on a FPGA, with a clock frequency of 100MHz.

The EKF algorithm implemented in the scope of this thesis by using inherently capable of parallel processing FPGA's performs real-time estimations approximately 11 times faster than the EKF algorithm realized in the other hardware platforms, such as DS1104 with the clock frequency of 250 MHz. In other words, the algorithms including computational complexity but parallel mathematical operations, such as EKF, are shown to be run with the less sampling time in a FPGA in real-time applications.

From the other side, the results of the EKF algorithm implemented in FPGA are compared and verified with the results of the same EKF algorithm realized in MATLAB m-file. Moreover, serial and ethernet communication protocols used for two way data transferring between FPGA and PC are realized to see graphically the proposed EKF algorithm results. From these points of view, it is first known research in the literature.

Key words: Induction motor, Extended Kalman filter, Speed sensorless control, FPGA.

TEŞEKKÜR

Bu tezin hazırlanması sürecinde beni yönlendiren, teşvik eden ve gerek teknik bilgi gerekse de döküman konusunda desteğini esirgemeyen değerli danışmanım Sayın Yrd. Doç. Dr. Murat BARUT'a sonsuz teşekkürlerimi sunarım.

Yaptığım çalışmalar boyunca her zaman bana destek çıkan ve bilgi paylaşımını esirgemeyen değerli hocam Sayın Yrd. Doç. Dr. Fuat KARAKAYA'ya teşekkürlerimi sunarım.

Yüksek lisans eğitimim boyunca birçok konuda bana destek çıkan Rıdvan DEMİR, Emrah ZERDALİ ve M. Muzaffer KÖSTEN arkadaşlarıma da çok teşekkür ederim.

Ayrıca, bu tez çalışmasının gerçek-zamanlı uygulamasında, Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) tarafından desteklenen EEEAG-108E187 nolu proje kapsamında oluşturulan deney düzeneği kullanıldığından ve bu proje kapsamında bursiyer olarak desteklendiğimden TUBİTAK'a da teşekkürlerimi sunarım.

Son olarak, hayatım boyunca maddi ve manevi her zaman yanımda olan aileme sonsuz teşekkür ederim.

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

FOTOĞRAFLAR DİZİNİ

KISALTMALAR VE SİMGELER

VK	: Vektör kontrol
DMK	: Doğrudan moment kontrol
D.A	: Doğru akım
DGM	: Darbe genişlik modülasyonu
KF	: Kalman filtresi
GKF	: Genişletilmiş Kalman filtresi
ASM	: Asenkron motor
DSP	: Digital signal processing/Dijital işaret işleme
FPGA	: Field programmable gate array/Alan programlamalı kapı dizisi
VHDL	:Very high hardware description language/Çok yüksek donanım tanımlama dili
ASIC	: Application specific integrated circuits/Özel tüm devreler
SRAM	: Static random access memory/Statik rastgele erişimli bellek
PROM	:Programmable read only memory/Programlanabilir salt okunabilir bellek
PLD	: Programmable logic device/Programlanabilir lojik eleman
SPLD	: Simple programmable logic device/Basit programlanabilir lojik eleman
CPLD	:Complex programmable logic device/Karmaşık programlanabilir lojik eleman
MPGA	: Mask programmable gate array/Maske programlanabilir kapı dizisi
PLA	: Programmable logic array/Programlanabilir lojik dizisi

PAL	: Programmable array logic/Programlanabilir dizi lojiği
VLSI	: Very large scale integration/Çok büyük ölçekli birleşme
FPL	: Field programmable logic/Alan programlamalı lojik
EPROM	:Erasable programmable read only memory/ Silinip programlanabilir salt okunur bellek
E^2PROM	:Electronically erasable programmable read-only memory/ Elektriksel olarak silinip programlanabilir salt okunur bellek
CMOS	:Complementary metal oxide semiconductor/ Bütünleyici metal yarıiletken oksit
LUT	: Look up table
CAD	: Computer-aided design/ Bilgisayar Destekli Tasarım
RAM	: Random access memory/Rastgele erişimli bellek
HDL	: Hardware description language/Donanım tanımlama dili
SPI	: Serial peripheral interface/Seri çevresel bağdaştırıcı
PCI	: Personal computer interface/Kişisel bilgisayar arayüzü
SFP	: Small form-factor pluggable/Küçük birim-katsayısı bağlantısı
LSB	: Least significant bit/En düşük değerlikli bit
MSB	: Most significant bit/En yüksek değerlikli bit
LAN	: Local area network/Ethernet yerel ağları
IEEE	:International Electrical and Electronics Engineering/Uluslararası Elektrik ve Elektronik Mühendisliği Enstitüsü
ECMA	:European Computers Manufacturers Association/Avrupa Bilgisayar Üreticileri Birliği
IEC	:International Electrotechnical Commision/Uluslararası Elektroteknik Komisyonu

ISO	: International Organization for Standardization/Uluslararası Standartlaştırma Komisyonu
MAC	: Media access kontrol/Ortam erişim yönetimi
NIC	: Network interface cards/Ağ arayüz kartları
FIFO	: First in first out/İlk giriş ilk çıkış
sof_n	: Start of frame/Çerçeve başlangıcı
eof_n	: End of frame/Çerçeve sonlandırıcısı
src_rdy_n	: Source ready/Kaynak hazır
dst_rdy_n	: Destination ready/İletim tamamlandı
VHSIC	: Very high speed integrated circuits/Yüksek hızlı tümleşik devreler
AMS	: Analogue mixed-signal/Analog karışık-işaret
ADC	: Analogue to digital converter/Analog dijital çevirici
R_s	: Stator direnci
R_r	: Rotor direnci
t_L	: Yük momenti
$\underline{\boldsymbol{x}}_k, \underline{\boldsymbol{x}}_e(k), \underline{\boldsymbol{x}}_e(t)$	: Dinamik sistemin durum vektörü
$\underline{\boldsymbol{A}}_k, \underline{\boldsymbol{A}}_e$	: Sistem matrisi
$\underline{\boldsymbol{B}}_k, \underline{\boldsymbol{B}}_e$	: Giriş matrisi
$\underline{\boldsymbol{u}}_k, \underline{\boldsymbol{u}}_e, \underline{\boldsymbol{u}}_e(k), \underline{\boldsymbol{u}}_e(t)$	: Giriş (kontrol) vektörü
k, T	: Ayrık zamanlı sistemin örnekleme (ölçme) zamanı
$\underline{\boldsymbol{y}}_k, \underline{\boldsymbol{y}}(t)$	: k. zamanda gözlemlenen çıkış vektörü

t : Zaman

$\underline{\boldsymbol{H}}_k, \underline{\boldsymbol{H}}_e$: Ölçme matrisi

$\underline{\boldsymbol{w}}_k, \underline{\boldsymbol{w}}_1, \underline{\boldsymbol{w}}_1(t), \underline{\boldsymbol{w}}_1(k)$: Sıfır ortalamalı sistem gürültüsü

$\underline{\boldsymbol{v}}_k, \underline{\boldsymbol{w}}_2(t), \underline{\boldsymbol{w}}_2(k)$: Sıfır ortalamalı ölçme gürültüsü

$E[.]$: Beklenen (ortalama) değer

$\underline{\boldsymbol{Q}}_k$: Sistem gürültüsü kovaryans matrisi

$\underline{\boldsymbol{R}}_k$: Ölçme gürültüsü kovaryans matrisi

$\hat{\underline{\boldsymbol{x}}}_k$: k. zamandan sonraki kestirilen durum

$\hat{\underline{\boldsymbol{x}}}_k^-$: k. zamandan önceki kestirilen durum

$\underline{\boldsymbol{G}}_k$: Kalman kazancı

$\underline{\boldsymbol{P}}_k, \underline{\boldsymbol{P}}(k)$: k anından sonraki durum kestirim hata kovaryans matrisi

$\underline{\boldsymbol{e}}_k$: k. zamandan sonraki durum kestirim hata vektörü

$\boldsymbol{I}$: Birim matris

$\underline{\boldsymbol{e}}_k^-$: k. zamandan önceki durum kestirim hata vektörü

$\underline{\boldsymbol{P}}_k^-, \underline{\boldsymbol{P}}^-(k)$: k. zamandan önceki durum kestirim hatasının kovaryans matrisi

$\underline{\boldsymbol{H}}_k^T, \underline{\boldsymbol{H}}_e^T$: Ölçme matrisinin transpozesi

$[.]^{-1}$: Matrisin tersi

$\underline{\boldsymbol{F}}_{k,k-1}, \underline{\boldsymbol{F}}_e(k)$: (k-1). zamandan k. zamana $\underline{\boldsymbol{x}}_k$ durumunu taşıyan geçiş matrisi

$f\left(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k\right)$: Doğrusal olmayan geçiş matrisi

$h\left(k, \underline{\boldsymbol{x}}_k\right)$: Doğrusal olmayan ölçme matrisi

t_e : Motorda endüklenen elektriksel (elektomanyetik) moment

T_L : Ölçülen, motora uygulanan yük momenti

P_p : Kutup çifti sayısı

L_m : Mıknatıslanma endüktansı (Karşılıklı endüktans)

$L_r^{'}$: Statora indirgenmiş rotor öz endüktansı

$\hat{\cdot}$: .'nın kestirildiğini ya da gözlemlendiğini gösterir

$\cdot^*$: .'nın ASM modelinden elde edilen sonuç olduğunu gösterir

$i_{s\alpha,mat}, i_{s\beta,mat}$: Matlab'te elde edilen stator akımlarının α- *ve* β- bileşenleri

$\varphi_{r\alpha,mat}, \varphi_{r\beta,mat}$: Matlab'te elde edilen halkalanan rotor akısının α- *ve* β- bileşenleri

$n_{m,mat}$: Matlab'te elde edilen rotor milinin mekanik hızı

$\hat{t}_{L,mat}$: Matlab'te kestirilen yük momenti

$v_{s\alpha}, v_{s\beta}$: Stator geriliminin α- *ve* β- bileşenleri

$i_{s\alpha}, i_{s\beta}$: Stator akımlarının α- *ve* β- bileşenleri

$\varphi_{r\alpha}, \varphi_{r\beta}$: Halkalanan rotor akısının α- *ve* β- bileşenleri

ω_m : Rotor milinin açısal mekanik hızı

n_m : Rotor milinin mekanik hızı

$e_{\hat{i}_{s\alpha}}, e_{\hat{i}_{s\beta}}$:ASM modelinden elde edilen ile kestirilen stator akım bileşenleri arasındaki fark olarak tanımlanan α- *ve* β- bileşenleri kestirim hataları

$e_{\hat{\varphi}_{r\alpha}}, e_{\hat{\varphi}_{r\beta}}$: ASM modelinden elde edilen ile kestirilen rotor akısı bileşenleri arasındaki fark olarak tanımlanan *α- ve β-* bileşenleri kestirim hataları

$e_{\hat{n}_m}$: ASM modelinden elde edilen ile kestirilen rotor açısal hızları arasındaki fark olarak tanımlanan kestirim hatası

$e_{\hat{t}_L}$: ASM modelinden elde edilen ile kestirilen yük momentleri arasındaki fark olarak tanımlanan kestirim hatası

$e_{\hat{i}_{s\alpha,mat}}, e_{\hat{i}_{s\beta,mat}}$: Matlab ve FPGA'da kestirilen stator akımı bileşenleri arasındaki fark olarak tanımlanan *α- ve β-* bileşenleri kestirim farkları

$e_{\hat{\varphi}_{r\alpha,mat}}, e_{\hat{\varphi}_{r\beta,mat}}$: Matlab ve FPGA'da kestirilen rotor akısı bileşenleri arasındaki fark olarak tanımlanan *α- ve β-* bileşenleri kestirim farkları

$e_{\hat{n}_{m,mat}}$: Matlab ve FPGA'da kestirilen rotor açısal hızları arasındaki fark olarak tanımlanan kestirim farkı

$e_{\hat{t}_{L,mat}}$: Matlab ve FPGA'da kestirilen yük momentleri arasındaki fark olarak tanımlanan kestirim farkı

$e_{i_{s\alpha,mat}}, e_{i_{s\beta,mat}}$: Matlab'te ile FPGA'da gerçeklenen ASM modelinden elde edilen stator akımı bileşenleri arasındaki fark olarak tanımlanan *α- ve β-*bileşenleri farkları

$e_{\varphi_{r\alpha,mat}}, e_{\varphi_{r\beta,mat}}$: Matlab'te ve FPGA'da gerçeklenen ASM modelinden elde edilen rotor akısı bileşenleri arasındaki fark olarak tanımlanan *α- ve β-*bileşenleri farkları

$e_{n_{m,mat}}$: Matlab'te ve FPGA'da gerçeklenen ASM modelinden elde edilen rotor açısal hızları arasındaki fark

$e_{t_{L,mat}}$: Matlab ve FPGA'da gerçeklenen ASM modelinden elde edilen yük momentleri arasındaki fark

L_σ : Stator geçici endüktansı

L_s : Stator öz endüktansı

J_T : Motor ve yükün toplam eylemsizliği

B_L : Motor ve yükün toplam viskoz sürtünme katsayısı

BÖLÜM I

GİRİŞ

Ucuz ve daha az bakım gerektirmeleri, yüksek hız uygulamalarına daha uygun olmaları, yük altında devir sayılarının çok değişmemesi sincap kafesli asenkron motorların (ASM) endüstride en çok kullanılan motorlar olmalarına sebep olmuştur. Gelişen teknoloji ile birlikte ASM'ler geniş aralıklarda değişken hız kontrol uygulamalarında kullanılmaya başlanmıştır. Fakat ASM'lerin 5. dereceden doğrusal (linear) olmayan diferansiyel eşitliklerle tanımlanan bir modele sahip olması, model parametrelerinin sıcaklık ve frekansa bağlı olarak değişmesi ASM'nin sürücü devre tasarımını zorlaştırmaktadır. [1]

ASM'lerin yüksek başarımlı kontrolüne yönelik olarak 1969'da K.Hasse dolaylı vektör kontrol yöntemini ve 1971'de Blaschke doğrudan vektör kontrol yöntemini ortaya atmıştır [2]. Daha sonraki yıllarda bu yöntemin geliştirilmesine yönelik yapılan çalışmalar sonucunda Takahashi ve Noguchi [3] ve Depenbrock [4] tarafından Doğrudan Moment Kontrol (DMK) yöntemi önerilmiştir.

Doğrudan Vektör Kontrol yönteminde, alan yönlendirme koşulu ile rotor akısı dönen eksen takımının *d*- ekseni üzerinde sabit tutularak, rotor akısı ve stator akımı arasındaki kuplaj etkisi ortadan kaldırılır. Böylece akı ve momentin ayrı ayrı kontrol edilmesi mümkün olur. Yani akı bileşeni sabit tutularak momenti oluşturan akım bileşeni ile moment doğrusal olarak değiştirilebilir [5]. DMK, akı ve moment hatalarının önceden belirlenmiş histerisiz bant sınırları içerisinde tutulması için uygun gerilim ara-devreli eviricilerin doğrudan anahtarlama durumlarının seçilmesi ilkesine dayanır [1]. Doğrudan akı ve moment kontrolünün gerçekleştirildiği bu yöntemde, vektör kontrolünde kullanılan akı ve moment kontrolörleri yerine histerisiz karşılaştırıcılar kullanılır. VK'daki koordinat dönüşümü ve Darbe Genişlik Modülasyonu (DGM) yerine ise evirici durumlarını kontrol etmek için bir anahtarlama tablosu kullanılır [5].

ASM'nin akı, hız, moment, rotor ve stator dirençlerinin ölçülmesi, ASM'ye sonradan yapılan fiziksel müdahalelerle mümkündür ve bu bazı uygulama zorlukları meydana getirir. Bu ölçüm zorluklarının önüne geçilmesi için tüm bu durum ve parametrelerin kestirilmesi yoluna gidilir. Bu noktada, literatürdeki belirgin model tabanlı gözlemleyici/kestirici çalışmalarından [6, 7, 8] farklı olarak, bu tez çalışmasında olasıl

durum/parametre kestiricisi olan Genişletilmiş Kalman Filtresi (GKF) kullanılmıştır. ASM modelindeki belirsizlikler ve modelin doğrusal olmama özelliği GKF'lerin olasıl doğası ile örtüşmektedir. Ayrıca bu yöntem ile durum ile birlikte parametrelerin kestirimi, diğer yöntemlere [6, 7, 8] kıyasla çok daha hızlı gerçekleştirilebilmektedir [9]. Bunlara ilave olarak GKF'lerde kullanılan genişletilmiş modelin doğal olarak içerdiği model (ya da sistem) ve ölçme gürültülerinden dolayı sistem sürekli olarak uyarılmakta, böylece kestirim doğruluğu ve kararlılığı özellikle düşük hızlarda iyileşmektedir. Bu özellikleri nedeniyle hesap karmaşıklığına rağmen GKF'ler ASM'lerin hız-algılayıcısız kontrolünde bu kadar yaygın olarak kullanılmaktadır. Ayrıca, yüksek başarımlı işlemci teknolojisinin gelişimi ile birlikte, GKF'lerin gerektirdiği hesaplama yükü ve hızı, problem olmaktan çıkmıştır [8, 10].

Kalman Filtresi (KF) beyaz gürültü ile bozulmuş işaretlerin kestirilmesini sağlayan en iyi (optimal) doğrusal kestiricidir. İlk kez 1960 yılında R.E Kalman tarafından KF algoritmasının önerilmesinden bu yana KF, modern kontrol, işaret işleme, haberleşme, hedef takip, araç takip-yol izleme, güdümlü sistemler gibi birçok alanda kullanılmıştır. İkinci dereceden en iyi ardışıl (recursive), ortalama karesel hatayı minimum yapma temeline dayanan KF, ardışıl olarak her ölçüm için bilinmeyen durum uzay vektörünü kestiren bir gerçek-zamanlı kestirim yapısı sağlar. Diğer bir deyişle, doğrusal dinamik bir sistemin beyaz gürültüyle bozulmuş ani durumlarının kestirilmesi sürecini, içerir. GKF ise KF'nin doğrusal olmayan sistemlere uygulanmış biçimidir [11]. Buna karşın KF yoğun bir hesaplama karmaşasına sahiptir. Çünkü birçok matris çarpımı ve tersi mevcuttur. Bu hesap karmaşıklığından dolayı KF birçok gerçek-zamanlı uygulamalarda sınırlı kalmaktadır [12].

Günümüzde GKF uygulamalarında genellikle yazılım tabanlı Digital Signal Processing/Sayısal İşaret İşleme (DSP) gibi dijital kontrolörler kullanılmaktadır. DSP üniteleri, düşük maliyetli oldukları ve yazılımsal esneklik sağladıkları için tercih sebebi olmuştur. Fakat işlemsel süreçler, DSP'nin yapısı itibariyle serileştirilmek zorunda olduğu için uzun sürmektedir. Bu problemin aşılması için donanımsal çözümlere gidilerek Field Programmable Gate Array/Alan Programlamalı Kapı Dizisi (FPGA) kullanılmaya başlanmıştır. FPGA tabanlı çalışmalarda, tasarım sürecinde mimaride paralelleştirme yapılarak algoritmaların koşturulma süreleri büyük oranda azaltılmakta ve örnekleme zamanı küçültülmektedir. FPGA yeniden programlanabilme özelliği ile de düşük maliyetli algoritma geliştirme ortamı sunmaktadır [13].

Literatürde, filtre [14, 15], işaret işleme [16], mobil robot [17, 18], araç takip-yol izleme , güvenlik sistemleri [19, 20], mikrocerrahi [21], ayrıklaştırılmış dalgacık dönüşümü [22], korelasyon temelli stereo görüntü işleme [23] ve senkron motor [13, 24, 25, 26, 27, 28] gibi bir çok alanda KF tabanlı FPGA uygulamaları gerçekleştirilmiştir. Bu çalışmaların yanı sıra ASM'lerin hız-algılayıcısız kontrolü için GKF tabanlı kestirim yönteminin FPGA üzerinde gerçeklendiği halen iki adet çalışma literatürde mevcuttur. Bu çalışmalarda stator akımları ve gerilimleri ölçülerek [29]'da indirgenmiş-dereceli KF ile stator akısı ve sabit parametre olarak tanımlanmış hız kestirimi, [30]'de ise tam-dereceli GKF ile rotor akısı ve sabit parametre gibi tanımlanmış rotor hızı kestirimleri FPGA kullanılarak gerçekleştirilmiştir. Yani, GKF'nin FPGA üzerinde gerçeklenmesi ile ilgili mevcut literatür göz önüne alındığında, hız-algılayıcısız kontrolde kullanılan akı ve hız durumlarına ilave olarak parametre kestirimleri gerçekleştirilmemiştir. Ancak, bütün hız aralığında hem geçici hem de sürekli halde hız kestirim doğruluğunu arttırmak için hız kestiriminde hareket eşitliğinin kullanılması gerekmektedir [9]. Hareket eşitliğinin kullanılması özellikle yük momenti kestirimini gerekli kılar. Bu yüzden hem geçici hal hem de sürekli hal düşük hızda hız kestirim başarımını iyileştirmek için yük momentinin de kestirilmesi gerekir [31].

Bu tez çalışmasının temel katkısı ASM'lerin hız-algılayıcısız kontrolü için [9]'da geliştirilmiş rotor akılarını, stator akımlarını, rotor hızını ve yük momentini kestiren GKF algoritması Xilinx Virtex 5 XC5VSX50T FPGA modülü üzerinde gerçekleştirmektir. FPGA'da gerçekleştirilen algoritmanın doğruluğu, MATLAB *m-file*'da yazılan aynı GKF algoritmasının kestirim sonuçları ile karşılaştırılarak gösterilmiştir. Öncelikle geliştirilen algoritmalar MATLAB'ta koşturulmuş, aynı algoritmalar tekrar Çok Yüksek Hızlı Entegre Devreler Donanım Tanımlama Dili/Very High Speed Integrated Circuits Hardware Description Language (VHDL) kullanılarak FPGA üzerinde gömülmüştür.

Tez kapsamında sırasıyla;

- Gerçek-zamanlı olarak a.a sürücüsünden elde edilen faz gerilimleri ve moment ölçerden alınan yük momenti bilgileri FPGA ve MATLAB *m*-file'da oluşturulan ASM modeline uygulanarak, ASM modelinin doğrulaması yapılmıştır. Ayrıca [34]'de belirtildiği gibi açık-çevrimli yük momenti kestiricisi tasarlanarak ASM

modelinin doğrulanması, hem FPGA hem de MATLAB *m-file*'da kestirilen yük momenti sonuçları kullanılarak da yapılmıştır.

- FPGA'da oluşturulan GKF algoritmasını doğrulamak için hem ISE12.2 ISim simülatöründe hem de FPGA'da ASM modeli ve GKF birlikte oluşturularak GKF algoritmasına ASM modeline uygulanan faz gerilimleri ve ASM modelinden elde edilen stator akımları uygulanarak, kestirimler gerçeklenmiştir.
- FPGA'da tasarlanan GKF algoritmasını gerçek-zamanlı olarak doğrulamak için ölçülen faz gerilimleri ve akımları hem FPGA hem de MATLAB ortamında oluşturulmuş GKF algoritmalarına uygulanarak stator akımı α ve β bileşenleri, rotor akısı α ve β bileşenleri, rotor açısal hızı ve yük momenti kestirimleri gerçekleştirilmiştir. Her iki algoritmadan elde edilen sonuçlar karşılaştırılarak FPGA'daki GKF algoritması doğrulanmıştır. Bu yönleriyle literatürde bilinen ilk çalışmadır.

Yapılan bu çalışmalar sonucunda 100 MHz'lik saat frekansına sahip bir FPGA ile 250 MHz'lik saat darbesi frekansına sahip DS1104 kontrolör kartına kıyasla yaklaşık 11 kat daha hızlı gerçek-zamanlı kestirimlerin yapılabileceği anlaşılmıştır.

Bu tez çalışması aşağıdaki gibi düzenlenmiştir:

Bölüm I'de literatür özeti verilerek bu tez çalışmasının daha önceki çalışmalara olan katkıları ve farklılıkları belirtilmiştir. KF ile GKF hakkında teorik bilgiler ve GKF tabanlı kestirici tasarımı Bölüm II'de verilmiştir. Bölüm III'de programlanabilir mantık elemanlarının çeşitliliği ve yapısı anlatılmıştır. Bölüm IV'de benzetim ve gerçek-zamanlı deney çalışmaları sunulmuştur. Son olarak da Bölüm V'de, yapılan tez çalışmasının sonuçları sunulmuş ve elde edilen başarımlar tartışılmıştır.

BÖLÜM II

GENİŞLETİLMİŞ KALMAN FİLTRESİ

2.1 Kalman Filtresi

Teorik olarak Kalman Filtresi (KF), beyaz gürültü ile bozulmuş doğrusal dinamik bir sistemin anlık durum kestiriminde doğrusal ikinci dereceden problem için, durum ile doğrusal fakat beyaz gürültü ile bozulmuş olan ölçümleri kullanan bir kestiricidir [33].

KF, gerçek ile kestirilen durumlar arasındaki fark olarak tanımlanan ortalama karesel hatayı minimum yapacak şekilde durumları kestirdiği için en iyi kestiricidir [34].

Pratik olarak KF istatistiksel kestirim teorisi tarihinde büyük bir buluştur ve 20.yy'ın en büyük buluşudur denilebilir.

Tıpkı silisyum gibi, birçok elektronik sistem tasarımında vazgeçilmez unsur olmuştur. KF sürekli üretim sistemlerinde, gemi ve uçak gibi karmaşık dinamik sistemlerde uygulama alanı bulmuştur [33].

Bir dinamik sistemi kontrol etmek için, öncelikle o sistemin ne yaptığını bilmek gerekir. Bu tür uygulamalarda, her zaman kontrol etmek istediğimiz her değişkeni ölçmek uygun veya tercih sebebi olmayabilir ve KF karmaşıklıktan ve gürültünün oluşturduğu bozunmalardan dolayı elde edilemeyen niceliklerin çıkarılmasında bir araç görevi görür. KF aynı zamanda dinamik sistemlerin muhtemel akışlarının, akış süreçlerinin tahmin edilmesinde kullanılır ve bunların insanlar tarafından kontrolü muhtemel değildir.

Sistem analizinde, belirgin (deterministic) sistem ve kontrol teorileri, sistem hakkındaki analiz ve tasarım sonuçlarını tam olarak karşılamazlar. Bunun 3 temel nedeni vardır [35].

Öncelikle, hiçbir matematik model mükemmel değildir. Model, mühendisin amacı doğrultusunda direkt olarak ilgilendiği karakteristikleri yansıtır. Örneğin, bir aracın eğilmesini tam olarak ifade etmek için sonsuz sayıda eğilme modu olmasına rağmen, kullanışlı bir model sonlu sayıda mod içerir. Burada amaçlanan sistem cevabında kritik ya da baskın modların gösterilmesidir.

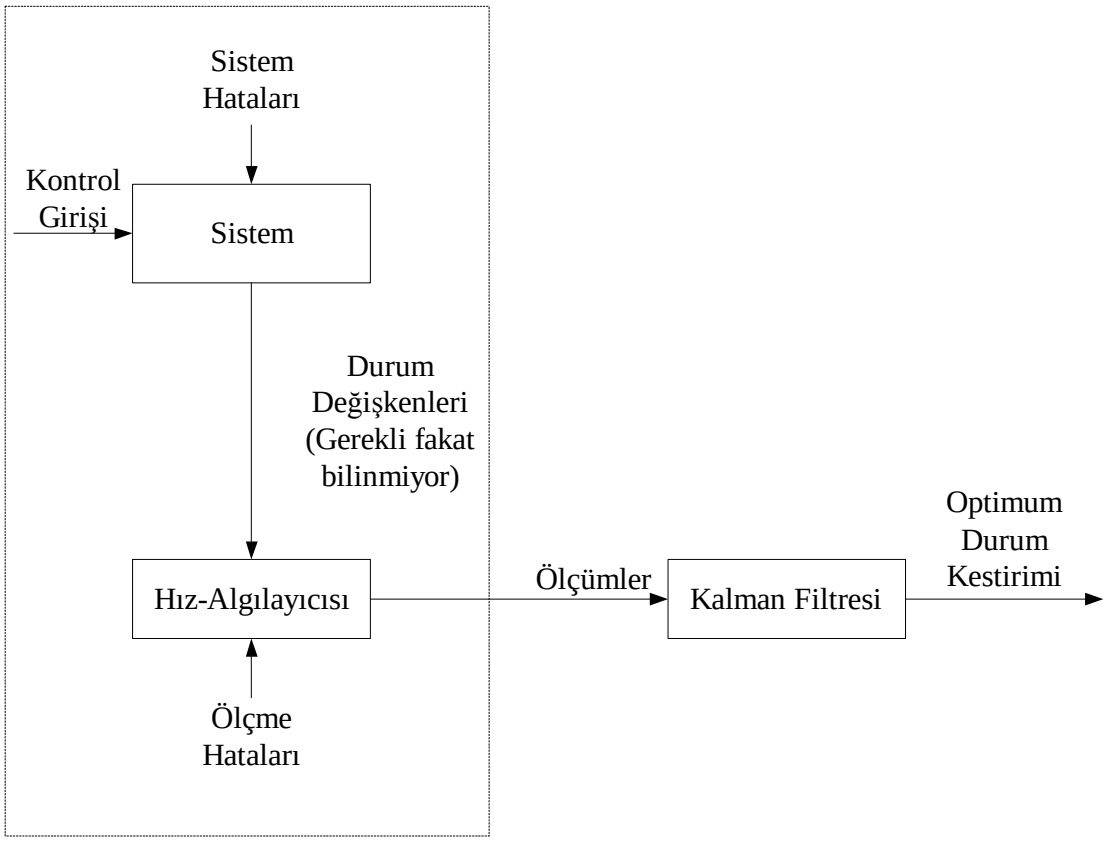
Dinamik sistemler sadece bizim kontrol girişlerimizin etkisi altında değillerdir. Aynı zamanda sisteme ne belirgin olarak modelleyebildiğimiz ne de kontrol edebildiğimiz

bozucu etkenler etkimektedir. Belirgin modelin ikinci eksik tarafı ise bozucu etkenlerin göz ardı edilmesidir.

Üçüncü ve son eksik taraf ise, algılayıcılar (sensörler) sistem hakkında en iyi ve tam veriyi sağlamazlar. Ya algılayıcı isteğimize göre tasarlanmamıştır, ya da ağırlık, hacim, fiyat vb. nedenlerle istediğimiz algılayıcıyı kullanamayız. En önemlisi algılayıcıların kendi dinamikleri vardır, daima gürültüden etkilenirler ve veriyi bozarlar.

Tüm bu saydığımız etkenler bizi olasıl modellemeye yönlendirmektedir. Algılayıcı gürültülerini hesaba katarak olasıl tabanlı kestirim yapan, iyi bilinen ve çok kullanılan yöntemlerden birisi KF'dir [36].

KF, doğrusal dinamik sistemlerde, sistemdeki gürültüleri ve ölçüm hatalarını göz önünde bulundurarak olasıl yaklaşımla, en iyi, yinelemeli (recursive) durum ve parametre kestirimi yapabilen etkili bir algoritmaya sahiptir. Şekil 2.1' de KF'nin tipik bir uygulaması görülmektedir [37].



Şekil 2.1 Tipik Kalman filtresi uygulaması [35].

Sistem ve ölçme gürültüleri de göz önünde bulundurularak doğrusal dinamik sistemin ayrık zamanda, olasıl durum ve çıkışı, eşitlik 2.1 ve 2.2'deki gibi ifade edilebilir.

Durum denklemi:

$$\underline{\boldsymbol{x}}_{k+1} = \underline{\boldsymbol{A}}_k \underline{\boldsymbol{x}}_k + \underline{\boldsymbol{B}}_k \underline{\boldsymbol{u}}_k + \underline{\boldsymbol{w}}_k \tag{2.1}$$

Çıkış denklemi:

$$\underline{\boldsymbol{y}}_k = \underline{\boldsymbol{H}}_k \underline{\boldsymbol{x}}_k + \underline{\boldsymbol{v}}_k \tag{2.2}$$

Burada, $\underline{\boldsymbol{x}}_k$, dinamik sistemin durum vektörü, sistemin bir sonraki davranışının kestirilebilmesi için, bu sistemin bir önceki davranışına ait en az veri kümesi, $\underline{\boldsymbol{A}}_k$, sistem matrisidir, $\underline{\boldsymbol{B}}_k$, giriş matrisi, $\underline{\boldsymbol{u}}_k$, giriş (kontrol) vektörü, k ayrık zamanlı sistemin örnekleme (ölçme) zamanı, $\underline{\boldsymbol{y}}_k$, k. zamanda gözlemlenen çıkış vektörü, $\boldsymbol{H}_k$, ölçme matrisi, $\underline{\boldsymbol{w}}_k$, sıfır ortalamalı sistem gürültüsünü temsil eder, beyaz gürültülü (white noise) ve Gaussyen biçiminde rastgele değişken, $\underline{\boldsymbol{v}}_k$, sıfır ortalamalı ölçme gürültüsünü temsil eder, beyaz gürültülü (white noise) ve Gaussyen biçiminde rastgele değişkendir.

Sistem ve ölçme gürültüsü kovaryans matrisleri sırasıyla aşağıdaki gibi tanımlanmıştır.

$$E\left[\underline{\boldsymbol{w}}_n \underline{\boldsymbol{w}}_k^T\right] = \begin{cases} \underline{\boldsymbol{Q}}_k & n = k \\ 0 & n \neq k \end{cases} \tag{2.3}$$

$$E\left[\underline{\boldsymbol{v}}_n \underline{\boldsymbol{v}}_k^T\right] = \begin{cases} \underline{\boldsymbol{R}}_k & n = k \\ 0 & n \neq k \end{cases} \tag{2.4}$$

Burada, $E[.]$, beklenen (ortalama) değer, $\underline{\boldsymbol{Q}}_k$ sistem gürültüsü kovaryans matrisi, $\underline{\boldsymbol{R}}_k$ ölçme gürültüsü kovaryans matrisidir. $\underline{\boldsymbol{w}}_k$ ve, $\underline{\boldsymbol{v}}_k$ birbirleriyle ilişkisi olmayan (uncorrelated) matrislerdir.

Buna göre,

$$E\left[\underline{\boldsymbol{w}}_n \underline{\boldsymbol{v}}_k^T\right] = \begin{cases} 0 & n = k \\ 0 & n \neq k \end{cases} \tag{2.5}$$

olur.

Kalman filtreleme problemi (sistemin bilinmeyen durumu için sistem ve ölçme eşitliklerini birlikte en iyi şekilde çözme problemi) aşağıdaki gibi açıklanabilir [34]:

$k \geq 1$'in her değeri için $\boldsymbol{x}_i$ durumunun minimum ortalama-karesel hatasının tespit edilmesi için gözlemlenen tüm $\underline{\boldsymbol{y}}_1, \underline{\boldsymbol{y}}_2, \ldots, \underline{\boldsymbol{y}}_k$ çıkış vektörleri kullanılır.

Burada eğer $i=k$ ise filtreleme, $i>k$ ise öngörü ve $i<k$ ise yumuşatma olarak adlandırılır.

Buna göre, başlangıç koşulları bilinmek kaydıyla, önceki (priori) kestirilen durum ile k anındaki ölçme vektörü kullanılarak sonraki (posteriori) durum kestirimi aşağıdaki gibi hesaplanır.

$$\underline{\hat{\boldsymbol{x}}}_k = \underline{\hat{\boldsymbol{x}}}_k^- + \underline{\boldsymbol{G}}_k(\underline{\boldsymbol{y}}_k - \underline{\boldsymbol{H}}_k\underline{\hat{\boldsymbol{x}}}_k^-) \tag{2.6}$$

Burada, $\underline{\boldsymbol{G}}_k$, k anından sonraki hata kovaryans matrisi $\underline{\boldsymbol{P}}_k$'yı minimize eden kazanç faktörü ya da Kalman kazancı olarak adlandırılır.

Durum hata vektörü,

$$\underline{\boldsymbol{e}}_k = \underline{\boldsymbol{x}}_k - \underline{\hat{\boldsymbol{x}}}_k \tag{2.7}$$

olur.

Eşitlik 2.2 ve 2.6, Eşitlik 2.7 ile ilişkilendirilerek durum hata vektörü aşağıdaki gibi elde edilir.

$$\underline{\boldsymbol{e}}_k = \underline{\boldsymbol{x}}_k - \left(\underline{\hat{\boldsymbol{x}}}_k^- + \underline{\boldsymbol{G}}_k\left(\underline{\boldsymbol{y}}_k - \underline{\boldsymbol{H}}_k\underline{\hat{\boldsymbol{x}}}_k^-\right)\right)$$

$$\underline{\boldsymbol{e}}_k = \underline{\boldsymbol{x}}_k - \left(\underline{\hat{\boldsymbol{x}}}_k^- + \underline{\boldsymbol{G}}_k\left(\underline{\boldsymbol{H}}_k\underline{\boldsymbol{x}}_k + \underline{\boldsymbol{v}}_k - \underline{\boldsymbol{H}}_k\underline{\hat{\boldsymbol{x}}}_k^-\right)\right)$$

$$\underline{\boldsymbol{e}}_k = \left(\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k\right)\left(\underline{\boldsymbol{x}}_k - \underline{\hat{\boldsymbol{x}}}_k^-\right) - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{v}}_k$$

$$\underline{\boldsymbol{e}}_k = \left(\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k\right)\underline{\boldsymbol{e}}_k^- - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{v}}_k \tag{2.8}$$

Buradan durum kestirim hata kovaryans matrisi $\underline{\boldsymbol{P}}_k$, aşağıdaki gibi elde edilir.

$$\underline{\boldsymbol{P}}_k = E[\underline{\boldsymbol{e}}_k\underline{\boldsymbol{e}}_k^T]$$

$$= E[\left(\left(\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k\right)\underline{\boldsymbol{e}}_k^- - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{v}}_k\right)\left(\left(\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k\right)\underline{\boldsymbol{e}}_k^- - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{v}}_k\right)^T] \tag{2.9}$$

Ölçme gürültüsü $\underline{\boldsymbol{v}}_k$, $\underline{\boldsymbol{e}}_k^-$'dan bağımsız ve sıfır ortalama değerli olduğu için $E\left[\underline{\boldsymbol{v}}_k\right] = 0$'dır. Bu durumda Eşitlik 2.9 aşağıdaki gibi düzenlenir.

$$\underline{\boldsymbol{P}}_k = E\left[\left((\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k)\underline{\boldsymbol{e}}_k^-\right)\left((\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k)\underline{\boldsymbol{e}}_k^-\right)^T\right] + E[(\underline{\boldsymbol{G}}_k\underline{\boldsymbol{v}}_k)(\underline{\boldsymbol{G}}_k\underline{\boldsymbol{v}}_k)^T]$$

$$= (\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k)E\left[\underline{\boldsymbol{e}}_k^-(\underline{\boldsymbol{e}}_k^-)^T\right](\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k)^T + \underline{\boldsymbol{G}}_k E[\underline{\boldsymbol{v}}_k\underline{\boldsymbol{v}}_k^T]\underline{\boldsymbol{G}}_k^T$$

$$= (\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k)\underline{\boldsymbol{P}}_k^-(\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k)^T + \underline{\boldsymbol{G}}_k\underline{\boldsymbol{R}}_k\underline{\boldsymbol{G}}_k^T \tag{2.10}$$

Burada, $\underline{\boldsymbol{e}}_k^-$, k anından önceki durum kestirim hata vektörü, $\underline{\boldsymbol{P}}_k^-$, k anından önceki durum kestirim hatasının kovaryans matrisi, ***A, B*** ve ***C*** matrisel ifadeler olmak üzere bir matrisin izi (trace) ve türev alma ile ilgili Eşitlik 2.11' de verilen özellikler kullanılarak,

$$Tr(\boldsymbol{A}) = Tr(\boldsymbol{A}^T)$$

$$\frac{dTr(\boldsymbol{AC})}{d\boldsymbol{A}} = \boldsymbol{C}^T$$

$$\frac{dTr(\boldsymbol{ABA}^T)}{d\boldsymbol{A}} = 2\boldsymbol{AB} \tag{2.11}$$

Eşitlik 2.10'daki hata kovaryans matrisi $\underline{\boldsymbol{P}}_k$'nın izinin türevi alınıp sıfıra eşitlendiğinde Kalman kazancı $\underline{\boldsymbol{G}}_k$, aşağıdaki gibi elde edilir [36].

$$\frac{dTr(\underline{\boldsymbol{P}}_k)}{d\underline{\boldsymbol{G}}_k} = -2\underline{\boldsymbol{P}}_k^-\underline{\boldsymbol{H}}_k^T + 2\underline{\boldsymbol{G}}_k\left(\underline{\boldsymbol{H}}_k\underline{\boldsymbol{P}}_k^-\underline{\boldsymbol{H}}_k^T + \underline{\boldsymbol{R}}_k\right) = 0 \tag{2.12}$$

$$\underline{\boldsymbol{G}}_k = \underline{\boldsymbol{P}}_k^-\underline{\boldsymbol{H}}_k^T[\underline{\boldsymbol{H}}_k\underline{\boldsymbol{P}}_k^-\underline{\boldsymbol{H}}_k^T + \underline{\boldsymbol{R}}_k]^{-1} \tag{2.13}$$

Ayrıca Eşitlik 2.10'un açık ifadesi çıkarılır ve Eşitlik 2.13 ile ilişkilendirilirse $\underline{\boldsymbol{P}}_k$, aşağıdaki gibi basit bir biçime indirgenebilir [34].

$$\underline{\boldsymbol{P}}_k = (\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k)\underline{\boldsymbol{P}}_k^- \tag{2.14}$$

KF'nin algoritma akış diyagramı aşağıdaki gibi düzenlenebilir.

1. Durum uzay modeli:

$$\underline{\boldsymbol{x}}_{k+1} = \underline{\boldsymbol{A}}_k\underline{\boldsymbol{x}}_k + \underline{\boldsymbol{B}}_k\underline{\boldsymbol{u}}_k + \underline{\boldsymbol{w}}_k \tag{2.15}$$

$$\underline{\boldsymbol{y}}_k = \underline{\boldsymbol{H}}_k\underline{\boldsymbol{x}}_k + \underline{\boldsymbol{v}}_k \tag{2.16}$$

2. Başlangıç değerlerinin atanması:

$k=0$ anında gözlemlenen herhangi bir veri olmadığından, kestirilen durumun başlangıç değeri aşağıdaki gibi seçilebilir.

$$\hat{\underline{\boldsymbol{x}}}_0 = E[\underline{\boldsymbol{x}}_0] \tag{2.17}$$

$$\underline{\boldsymbol{P}}_0 = E[(\underline{\boldsymbol{x}}_0 - E[\underline{\boldsymbol{x}}_0])(\underline{\boldsymbol{x}}_0 - E[\underline{\boldsymbol{x}}_0])^T] \tag{2.18}$$

3. Önceki durum vektörü $\hat{\underline{\boldsymbol{x}}}_k^-$ kestiriminin öngörümü

$$\hat{\underline{\boldsymbol{x}}}_k^- = \underline{\boldsymbol{A}}_k \hat{\underline{\boldsymbol{x}}}_{k-1} + \underline{\boldsymbol{B}}_k \underline{\boldsymbol{u}}_{k-1} \tag{2.19}$$

4. Önceki hata kovaryansı $\underline{\boldsymbol{P}}_k^-$'nın öngörümü:

$$\underline{\boldsymbol{P}}_k^- = \underline{\boldsymbol{A}}_k \underline{\boldsymbol{P}}_{k-1} \underline{\boldsymbol{A}}_k^T + \underline{\boldsymbol{Q}}_{k-1} \tag{2.20}$$

5. Hata kovaryansı $\underline{\boldsymbol{P}}_k$'nın güncellenmesi:

$$\underline{\boldsymbol{P}}_k = (\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k \underline{\boldsymbol{H}}_k) \underline{\boldsymbol{P}}_k^- \tag{2.21}$$

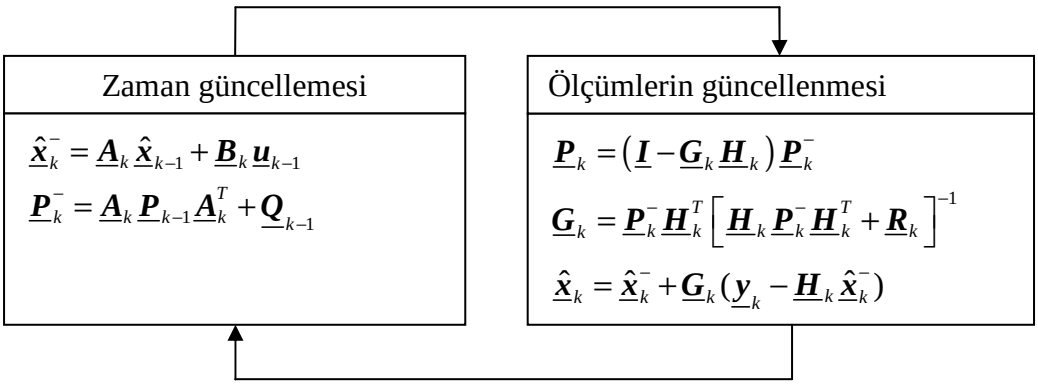
6. Kalman kazancı hesabı:

$$\underline{\boldsymbol{G}}_k = \underline{\boldsymbol{P}}_k^- \underline{\boldsymbol{H}}_k^T [\underline{\boldsymbol{H}}_k \underline{\boldsymbol{P}}_k^- \underline{\boldsymbol{H}}_k^T + \underline{\boldsymbol{R}}_k]^{-1} \tag{2.22}$$

7. Durum vektörü kestirimi $\hat{\underline{\boldsymbol{x}}}_k$'nin güncellenmesi:

$$\hat{\underline{\boldsymbol{x}}}_k = \hat{\underline{\boldsymbol{x}}}_k^- + \underline{\boldsymbol{G}}_k (\underline{\boldsymbol{y}}_k - \underline{\boldsymbol{H}}_k \hat{\underline{\boldsymbol{x}}}_k^-) \tag{2.23}$$

8. Önceki durum vektörü $\hat{\underline{\boldsymbol{x}}}_k^-$ kestiriminin öngörümü için tekrar 3. adıma dönülür.



Şekil 2.2 Ayrık zamanlı Kalman filtresi algoritması

2.2 Genişletilmiş Kalman Filtresi

KF doğrusal dinamik sistemlerde durum ve parametre kestirimi için uygulanmaktadır. Doğrusal olmayan (nonlineer) dinamik sistemler için ise Genişletilmiş Kalman Filtresi (GKF) kullanılır.

GKF'de, durum ve çıkış denklemlerinin o anki kestirilen durum değişkenleri ($\underline{\hat{\boldsymbol{x}}}_k$ ve $\underline{\hat{\boldsymbol{x}}}_k^-$) civarında doğrusallaştırılarak elde edilen doğrusal modele KF algoritması uygulanır [34].

Doğrusal olmayan olasıl durum ve çıkış denklemleri aşağıdaki gibidir.

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}\left(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k\right) + \underline{\boldsymbol{w}}_k \tag{2.24}$$

$$\boldsymbol{y}_k = \boldsymbol{h}\left(k, \underline{\boldsymbol{x}}_k\right) + \underline{\boldsymbol{v}}_k \tag{2.25}$$

Burada, $\boldsymbol{f}(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k)$ doğrusal olmayan geçiş matrisi ve $\boldsymbol{h}(k, \underline{\boldsymbol{x}}_k)$ doğrusal olmayan ölçme matrisidir.

Doğrusal olmayan geçiş matrisi $\boldsymbol{f}(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k)$, $\underline{\hat{\boldsymbol{x}}}_k$ etrafında, ölçme matrisi $\boldsymbol{h}(k, \underline{\boldsymbol{x}}_k)$, $\underline{\hat{\boldsymbol{x}}}_k^-$ etrafında birinci dereceden Taylor serisine aşağıdaki gibi açılır.

$$\underline{\boldsymbol{F}}_{k+1} = \frac{\partial \boldsymbol{f}(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k)}{\partial x}\Big|_{\underline{\boldsymbol{x}}_k = \underline{\hat{\boldsymbol{x}}}_k} \tag{2.26}$$

$$\underline{\boldsymbol{H}}_k = \frac{\partial \boldsymbol{h}(k, \underline{\boldsymbol{x}}_k)}{\partial x}\Big|_{\underline{\boldsymbol{x}}_k = \underline{\hat{\boldsymbol{x}}}_k^-} \tag{2.27}$$

$\boldsymbol{f}(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k)$ ve $\boldsymbol{h}(k, \underline{\boldsymbol{x}}_k)$ ifadelerinin yaklaşık eşitlikleri aşağıdaki gibi alınabilir.

$$\boldsymbol{f}\left(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k\right) \approx \boldsymbol{f}\left(k, \underline{\hat{\boldsymbol{x}}}_k, \underline{\boldsymbol{u}}_k\right) + \underline{\boldsymbol{F}}_{k+1}\left(\underline{\boldsymbol{x}}_k - \underline{\hat{\boldsymbol{x}}}_k\right) \tag{2.28}$$

$$\boldsymbol{h}\left(k, \underline{\boldsymbol{x}}_k\right) \approx \boldsymbol{h}\left(k, \underline{\hat{\boldsymbol{x}}}_k^-\right) + \underline{\boldsymbol{H}}_k\left(\underline{\boldsymbol{x}}_k - \underline{\hat{\boldsymbol{x}}}_k^-\right) \tag{2.29}$$

Böylelikle Eşitlik 2.24 ve Eşitlik 2.25' de verilen doğrusal olmayan modeller doğrusallaştırılmış olur.

$$\underline{\boldsymbol{x}}_{k+1} = \underline{\boldsymbol{F}}_{k+1}\underline{\boldsymbol{x}}_k + \underline{\boldsymbol{w}}_k + \underline{\boldsymbol{a}}_k \tag{2.30}$$

$$\underline{\boldsymbol{y}}_k = \underline{\boldsymbol{H}}_k\underline{\boldsymbol{x}}_k + \underline{\boldsymbol{v}}_k + \underline{\boldsymbol{\beta}}_k \tag{2.31}$$

Burada,

$\underline{\boldsymbol{F}}_{k+1}$, $\underline{\boldsymbol{x}}_k$ durumunu *k* anından *k+1* anına taşıyan geçiş matrisidir.

$$\underline{\boldsymbol{a}}_k = \boldsymbol{f}\left(k, \underline{\hat{\boldsymbol{x}}}_k, \underline{\boldsymbol{u}}_k\right) - \underline{\boldsymbol{F}}_{k+1}\underline{\hat{\boldsymbol{x}}}_k \tag{2.32}$$

$$\underline{\boldsymbol{\beta}}_k = \boldsymbol{h}\left(k, \underline{\hat{\boldsymbol{x}}}_k^-\right) - \underline{\boldsymbol{H}}_k\underline{\hat{\boldsymbol{x}}}_k^- \tag{2.33}$$

Sonuç olarak GKF akış diyagramı (algoritması) aşağıda belirtildiği gibidir.

1. Durum uzay modeli:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}\left(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k\right) + \underline{\boldsymbol{w}}_k \tag{2.34}$$

$$\boldsymbol{y}_k = \boldsymbol{h}\left(k, \underline{\boldsymbol{x}}_k\right) + \underline{\boldsymbol{v}}_k \tag{2.35}$$

2. Durum uzay modelinin doğrusallaştırılması:

$$\underline{\boldsymbol{F}}_{k+1} = \frac{\partial \boldsymbol{f}(k, \underline{\boldsymbol{x}}_k, \underline{\boldsymbol{u}}_k)}{\partial x}\Big|_{\underline{\boldsymbol{x}}_k = \underline{\hat{\boldsymbol{x}}}_k}, \ \underline{\boldsymbol{H}}_k = \frac{\partial \boldsymbol{h}(k, \underline{\boldsymbol{x}}_k)}{\partial x}\Big|_{\underline{\boldsymbol{x}}_k = \underline{\hat{\boldsymbol{x}}}_k^-} \tag{2.36}$$

3. Başlangıç değerlerinin atanması:

$$\underline{\hat{\boldsymbol{x}}}_0 = E[\underline{\boldsymbol{x}}_0], \ \underline{\boldsymbol{P}}_0 = E[(\underline{\boldsymbol{x}}_0 - E[\underline{\boldsymbol{x}}_0])(\underline{\boldsymbol{x}}_0 - E[\underline{\boldsymbol{x}}_0])^T] \tag{2.37}$$

4. Önceki durum vektörü $\underline{\hat{\boldsymbol{x}}}_k$ ve hata kovaryans vektörü $\underline{\boldsymbol{P}}_k^-$ tahmini:

$$\underline{\hat{\boldsymbol{x}}}_k^- = \underline{\boldsymbol{F}}_{k,k-1}\underline{\hat{\boldsymbol{x}}}_{k-1}, \ \underline{\boldsymbol{P}}_k^- = \underline{\boldsymbol{F}}_{k,k-1}\underline{\boldsymbol{P}}_{k-1}\underline{\boldsymbol{F}}_{k,k-1}^T + \underline{\boldsymbol{Q}}_{k-1} \tag{2.38}$$

5. Kovaryans matrisi $\underline{\boldsymbol{P}}_k$'nın güncellenmesi:

$$\underline{\boldsymbol{P}}_k = (\underline{\boldsymbol{I}} - \underline{\boldsymbol{G}}_k\underline{\boldsymbol{H}}_k)\underline{\boldsymbol{P}}_k^- \tag{2.39}$$

6. Kalman kazancının hesaplanması:

$$\underline{\boldsymbol{G}}_k = \underline{\boldsymbol{P}}_k^-\underline{\boldsymbol{H}}_k^T[\underline{\boldsymbol{H}}_k\underline{\boldsymbol{P}}_k^-\underline{\boldsymbol{H}}_k^T + \underline{\boldsymbol{R}}_k]^{-1} \tag{2.40}$$

7. Durum kestirimi vektörü $\underline{\hat{\boldsymbol{x}}}_k$'nın güncellenmesi:

$$\underline{\hat{\boldsymbol{x}}}_k = \underline{\hat{\boldsymbol{x}}}_k^- + \underline{\boldsymbol{G}}_k(\underline{\boldsymbol{y}}_k - \underline{\boldsymbol{H}}_k\underline{\hat{\boldsymbol{x}}}_k^-) \tag{2.41}$$

8. Durum uzay modeli oluşturulup doğrusallaştırıldığı için ve başlangıç değerleri tekrar atanmayacağı için tekrar 4.adıma gidilir.

2.3 Tez Kapsamında Gerçeklenmiş GKF Tabanlı Kestirici

ASM'nin stator duran eksenindeki genişletilmiş olasıl durum uzay modeli, GKF tabanlı kestirici tasarımı için aşağıdaki gibi genel bir biçimde verilebilir [9].

$$\underline{\dot{\boldsymbol{x}}}_{\boldsymbol{e}}(t) = \underline{\boldsymbol{f}}_e\left(\underline{\boldsymbol{x}}_e(t), \underline{\boldsymbol{u}}_e(t)\right) + \underline{\boldsymbol{w}}_1(t)$$

$$= \underline{\boldsymbol{A}}_e(\underline{\boldsymbol{x}}_e)\underline{\boldsymbol{x}}_e + \underline{\boldsymbol{B}}_e\underline{\boldsymbol{u}}_e + \underline{\boldsymbol{w}}_1 \tag{2.42}$$

$$\underline{\boldsymbol{Z}}(t) = \underline{\boldsymbol{H}}_e\underline{\boldsymbol{x}}_e(t) + \underline{\boldsymbol{w}}_2(t) \tag{2.43}$$

Burada $\underline{\boldsymbol{f}}_e$, durumların doğrusal olmayan fonksiyonudur. $\underline{\boldsymbol{x}}_e$, genişletilmiş durum vektörüdür. $\underline{\boldsymbol{A}}_e$, sistem matrisidir. $\underline{\boldsymbol{u}}_e$, kontrol giriş vektörüdür. $\underline{\boldsymbol{B}}_e$, giriş matrisidir. $\underline{\boldsymbol{H}}_e$, ölçme matrisidir. $\underline{\boldsymbol{w}}_1$ ve $\underline{\boldsymbol{w}}_2$, sırasıyla sistem ve ölçme gürültüsüdür.

Eşitlik 2.42 ve 2.43'deki matrisler ayrıntılı bir şekilde aşağıdaki gibi verilebilir:

$$\underbrace{\begin{bmatrix} \dot{i}_{s\alpha} \\ \dot{i}_{s\beta} \\ \dot{\varphi}_{r\alpha} \\ \dot{\varphi}_{r\beta} \\ \dot{\omega}_m \\ \dot{t}_L \end{bmatrix}}_{\underline{\dot{\boldsymbol{x}}}_e} = \underbrace{\begin{bmatrix} -\left(\frac{R_s}{L_\sigma} + \frac{L_m^2 R_r'}{L_\sigma L_r'^2}\right) & 0 & \frac{L_m R_r'}{L_\sigma L_r'^2} & \frac{L_m}{L_\sigma L_r'} p_p\omega_m & 0 & 0 \\ 0 & -\left(\frac{R_s}{L_\sigma} + \frac{L_m^2 R_r'}{L_\sigma L_r'^2}\right) & -\frac{L_m}{L_\sigma L_r'} p_p\omega_m & \frac{L_m R_r'}{L_\sigma L_r'^2} & 0 & 0 \\ \frac{R_r'}{L_r'} L_m & 0 & -\frac{R_r'}{L_r'} & -p_p\omega_m & 0 & 0 \\ 0 & \frac{R_r'}{L_r'} L_m & p_p\omega_m & -\frac{R_r'}{L_r'} & 0 & 0 \\ -\frac{1}{J_L}\frac{3}{2} p_p \frac{L_m}{L_r'}\varphi_{r\beta} & \frac{1}{J_L}\frac{3}{2} p_p \frac{L_m}{L_r'}\varphi_{r\alpha} & 0 & 0 & 0 & -\frac{1}{J_T} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\underline{\boldsymbol{A}}_e} \underbrace{\begin{bmatrix} i_{s\alpha} \\ i_{s\beta} \\ \varphi_{r\alpha} \\ \varphi_{r\beta} \\ \omega_m \\ t_L \end{bmatrix}}_{\underline{\boldsymbol{x}}_e} +$$

$$+ \underbrace{\begin{bmatrix} \frac{1}{L_\sigma} & 0 \\ 0 & \frac{1}{L_\sigma} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}_{\underline{\boldsymbol{B}}_e} \underbrace{\begin{bmatrix} v_{s\alpha} \\ v_{s\beta} \end{bmatrix}}_{\underline{\boldsymbol{u}}_e} + \boldsymbol{w_1}(t) \tag{2.44}$$

$$\underbrace{\begin{bmatrix} i_{s\alpha} \\ i_{s\beta} \end{bmatrix}}_{\underline{\boldsymbol{Z}}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\underline{\boldsymbol{H}}_e} \underbrace{\begin{bmatrix} i_{s\alpha} \\ i_{s\beta} \\ \varphi_{r\alpha} \\ \varphi_{r\beta} \\ \omega_m \\ t_L \end{bmatrix}}_{\underline{\boldsymbol{x}}_e} + \boldsymbol{w_2}(t) \tag{2.45}$$

Burada L_σ stator geçici endüktansıdır. p_p, kutup çifti sayısıdır. L_s ve R_s, sırasıyla stator endüktansı ve direncidir. L_r' ve R_r', sırasıyla stator tarafına indirgenmiş rotor endüktansı ve direncidir. $\varphi_{r\alpha}$ ve $\varphi_{r\beta}$, rotor akısının stator duran eksen bileşenleridir. $v_{s\alpha}$ ve $v_{s\beta}$, stator gerilimlerinin stator duran eksen bileşenleridir. $i_{s\alpha}$ ve $i_{s\beta}$, stator akımının stator

duran eksen bileşenleridir., ω_m rotor açısal mekanik hızıdır. t_L, yük momenti. T, örnekleme zamanıdır.

GKF ile kestirim süreci için elde edilen bu model geriye doğru fark yöntemi ile Eşitlik 2.46 ve 2.47'deki gibi ayrıklaştırılır.

$$\underline{\boldsymbol{x}}_e(k+1) = \underline{\boldsymbol{f}}_e\left(\underline{\boldsymbol{x}}_e(k), \underline{\boldsymbol{u}}_e(k)\right) + \underline{\boldsymbol{w}}_1(k)$$

$$= \underline{\boldsymbol{A}}_e\left(\underline{\boldsymbol{x}}_e(k)\right)\underline{\boldsymbol{x}}_e(k) + \underline{\boldsymbol{B}}_e\underline{\boldsymbol{u}}_e(k) + \underline{\boldsymbol{w}}_1(k) \tag{2.46}$$

$$\underline{\boldsymbol{Z}}(k) = \underline{\boldsymbol{H}}_e\underline{\boldsymbol{x}}_e(k) + \underline{\boldsymbol{w}}_2(k) \tag{2.47}$$

Daha soran doğrusal olmayan bu modelin (Eşitlik 2.46 ve 2.47) KF algoritmasında kullanılması için bir önceki adımda kestirilen durumlar ($\underline{\hat{\boldsymbol{x}}}_e(k)$) ve girişler ($\underline{\boldsymbol{u}}_e(k)$) civarında Eşitlik 2.48 kullanılarak doğrusallaştırılır.

$$\underline{\boldsymbol{F}}_e(k) = \left.\frac{\partial \underline{\boldsymbol{f}}_e\left(\underline{\boldsymbol{x}}_e(k), \underline{\boldsymbol{u}}_e(k)\right)}{\partial \underline{\boldsymbol{x}}_e(k)}\right|_{\underline{\hat{\boldsymbol{x}}}_e(k), \underline{\boldsymbol{u}}_e(k)} \tag{2.48}$$

Sonuç olarak GKF algoritması aşağıdaki ardışıl ilişkileri elde edilir.

$$\underline{\boldsymbol{P}}^-(k) = \underline{\boldsymbol{F}}_e(k)\underline{\boldsymbol{P}}(k)\underline{\boldsymbol{F}}_e^T(k) + \boldsymbol{Q} \tag{2.49a}$$

$$\underline{\boldsymbol{P}}(k) = \underline{\boldsymbol{P}}^-(k) - \underline{\boldsymbol{P}}^-(k)\underline{\boldsymbol{H}}_e^T(\boldsymbol{R} + \underline{\boldsymbol{H}}_e\underline{\boldsymbol{P}}^-(k)\underline{\boldsymbol{H}}_e^T)^{-1}\underline{\boldsymbol{H}}_e\underline{\boldsymbol{P}}^-(k) \tag{2.49b}$$

$$\underline{\hat{\boldsymbol{x}}}_e(k+1) = \underline{\hat{\boldsymbol{f}}}_e\left(\underline{\hat{\boldsymbol{x}}}_e(k), \underline{\boldsymbol{u}}_e(k)\right) + \underline{\boldsymbol{P}}(k)\underline{\boldsymbol{H}}_e^T\boldsymbol{R}^{-1}(\underline{\boldsymbol{Z}}(k) - \underline{\boldsymbol{H}}_e\underline{\hat{\boldsymbol{x}}}_e(k)) \tag{2.49c}$$

Burada $\boldsymbol{Q}$, sistem gürültüsünün kovaryans matrisidir. $\boldsymbol{R}$, çıkış/ölçme gürültüsünün kovaryans matrisidir. $\underline{\boldsymbol{P}}$ ve $\underline{\boldsymbol{P}}^-$, sırasıyla, durum kestirim hatasının kovaryans matrisi ve onun yayınım (extrapolation) matrisidir.

BÖLÜM III

PROGRAMLANABİLİR MANTIK (LOJİK) ELEMANLAR

3.1 Giriş

Sayısal elektronik sistem dünyasında, üç tür eleman mevcuttur. Bunlar; hafıza, mikroişlemci ve mantık (lojik) elemanlarıdır. Hafıza elemanları tablo veya veritabanlarının rastgele bilgilerini depo eden elemanlardır. Mikrokontrolörler yazılımsal kodların kelime işlemcisi ya da video oyunları gibi çok çeşitli amaçlarla çalıştırılmasını sağlar. Mantık elemanları, cihazdan cihaza arayüzleri, veri iletişimini, işaret işlemeyi, veri gösterimini, zamanlama, kontrol operasyonlarını ve diğer sistemlerin fonksiyonlarını kapsayan belirli fonksiyonların uygulanabilirliğini sağlar.

Donanımsal olarak sayısal işaretlerin işlenebilmesi için Özel Tüm Devreler/Application Specific Integrated Circuit (ASIC) kullanılır. Bu tip devreler yalnızca bir fonksiyonu gerçeklemek için üretildikleri için bu fonksiyonu hızlı ve verimli bir şekilde yerine getirirler fakat üzerlerine farklı bir fonksiyon yüklenemezler. Bu sebeple kapasiteleri ve işlevsellikleri sınırlıdır yalnızca ilgili oldukları uygulamaya yönelik üretilmişlerdir.

Mantık elemanları sabit ve programlanabilir olmak üzere iki ana başlık altında toplanabilir. Adından da anlaşılabileceği gibi sabit mantık elemanları kalıcıdır, yalnızca bir fonksiyon veya fonksiyon dizisi için tasarlanarak üretilirler ve değiştirilemezler. Diğer yandan, Programmable Logic Devices/Programlanabilir Mantık Elemanları (PLDs) diğer bir adıyla Field-Programmable Devices/Alan-Programlamalı Elemanlar (FPDs) standarttırlar, kullanıcının talebine göre üretilen ve kullanıcıya yüksek oranda mantık kapasitesi, özellik, hız ve gerilim karakteristiği sunan her hangi bir fonksiyonun yerine getirilmesi için yapısı değiştirilebilen elemanlardır.

Sabit mantık elemanlar, yapısal karmaşıklıklarından dolayı, istenilen tasarımın elde edilmesi, prototiplerinin yapılması ve son üretiminin gerçekleştirilmesi zaman alan elemanlardır hatta bu zaman bir yılı aşkın bir süreyi bile bulabilmektedir. Bu denli zahmetli çalışmanın bir ürünü olmasına rağmen eğer istenen fonksiyonları tam anlamıyla yerine getiremezse ve ya herhangi bir yapısal değişikliğe ihtiyaç duyulursa yeni bir sabit mantık elemanının tasarlanması gerekir. Bu sebeple maliyet artışı hat safhadadır.

PLD’ler, kapıların ve flip-flopların birbirlerine bağlanmasıyla oluşturulan devrelerdir. Bellek hücreleri, mantık kapılarının gerçeklediği fonksiyonları tanımlanmasını, bunların kontrolünü ve birbirleriyle olan bağlantı bilgilerinin (Giriş/Çıkış) kayıt altında tutulmasını sağlar.

PLD’lerde, tasarımcı yüksek maliyetli olmayan yazılım araçlarını kullanarak hızlı bir şekilde tasarımını gerçekleyebilir, benzetimini yapabilir ve kendi tasarımını test edebilir. Oluşturulan tasarım bir eleman üzerine hızlı ve kolay bir şekilde programlanabilir ve anında test edilebilir. PLD’ler, ağ dağıtıcısı, Digital Subscriber Line/Sayısal Kullanıcı Hattı (DSL) modemi, DVD oynatıcısı ya da bir otomobil takip sisteminde son gereçlerin bir parçası olarak son üretimlerdir.

PLD’lerin sağladığı diğer bir yarar ise kullanıcının isteğini karşılayabilecek, anlık olarak yapısal değişiklikleri sağlayabilen bir esnekliğe sahip olmasıdır. Bunun sebebi ise PLD’lerin yeniden yazılabilir hafıza elemanları temeline dayandırılmasıdır. Bu da tasarım değişikliğinde esneklik ve yeniden programlamada basitlik ve hız sağlar. Ayrıca PLD’lerin programlanabilmesi için gerekli olan yazılım dosyaları ve araçları üretici firmalar tarafından kullanıcıya sunulmaktadır.

Yapısal olarak yeniden düzenlenebilir sayısal veri işlemcilerinin ihtiyaç duyduğu esnek donanımlar, FPGA kullanılarak karşılanabilmektedir. Özellikle Static Random Access Memory/Statik Rastgele Erişimli Bellek (SRAM) tabanlı FPGA’lar tekrar düzenlenebilirlik kabiliyetleri ve yüksek performanslı uygulamalardaki yeterlilikleri sayesinde, genel amaçlı sayısal donanımların tasarımlarında önemli rol oynar. Günümüz teknolojisinde FPGA ve VHDL gittikçe önem kazanmaya başlamıştır. VHDL teknolojisinin kullanılmasıyla tanımlanan ve sentezlenen sistemler FPGA ile gerçeklenerek daha güçlü sistemler ortaya konulmaktadır.

3.2 Programlanabilir Mantık Elemanların Gelişimi

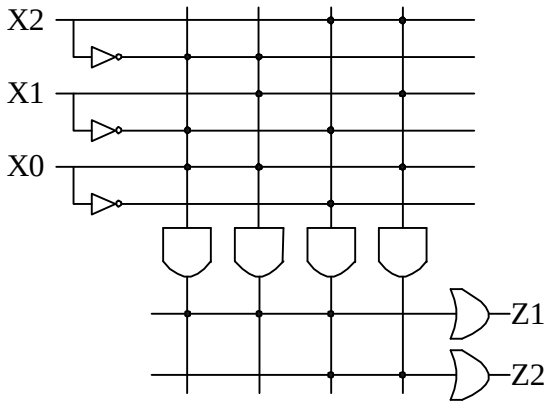
Sayısal tasarımlarda kullanılan ilk programlanabilir mantık devre elemanı Programmable Read Only Memory/Programlanabilir Salt Okunabilir Bellek’tir (PROM). Bu elemanda, belleğin adres girişleri, mantık devresinin girişlerine, adreslenmiş gözdeki bilgiler de, mantık fonksiyonun çıkışlarına karşılık düşer. Genellikle karmaşık mantık fonksiyonlar için verimsizdirler [38].

Programlanabilir yapıların sonraki türleri PLD'lerdir. PLD'ler; Simple Programmable Logic Device/Basit Programlanabilir Mantık Eleman (SPLD), Complex Programmable Logic Device/Karmaşık Programlanabilir Mantık Eleman (CPLD), Mask Programmable Gate Arrays/Maske Programlanabilir Kapı Dizileri (MPGA) ve FPGA olmak üzere dört bölümde incelenebilir.

SPLD'ler, Programmable Logic Arrays/Programlanabilir Kapı Dizileri (PLA) ve Programmable Array Logic/Programlanabilir Dizi Mantığı (PAL) olmak üzere iki kısma ayrılır.

3.2.1 PLA

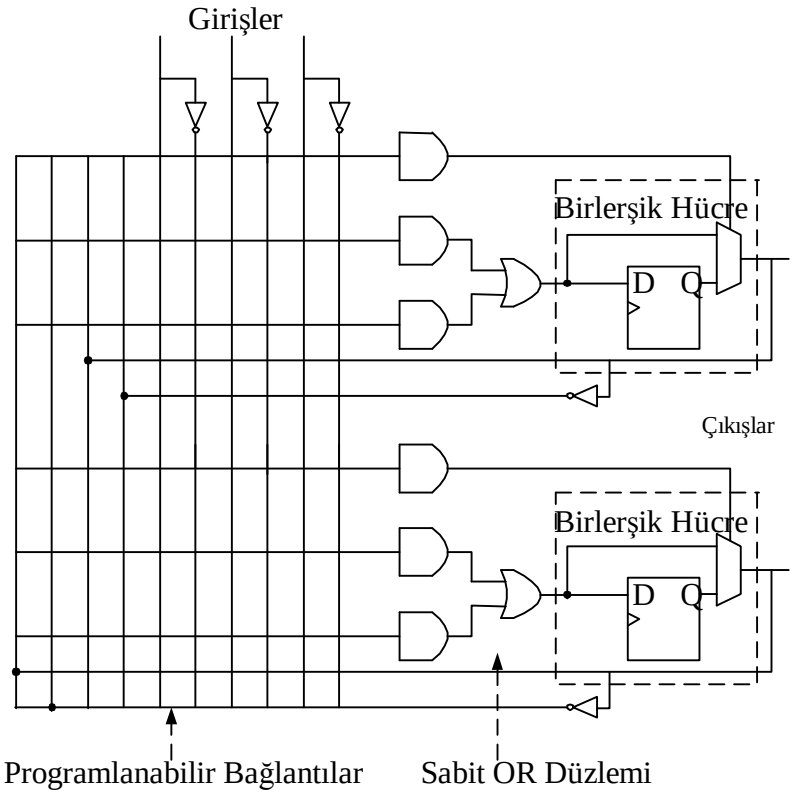
PLA'ya nispeten küçük PLD de denebilir. İki adet programlanabilir düzleme sahiptir. Bunlardan biri AND kapılarından oluşan düzlem diğeri de OR kapılarından oluşan düzlemdir. PLA bu AND ve OR kapılarından oluşan iki düzlemin birbiriyle ilişkilendirilmesi esasına dayanır. Şekil 3.1'den de görüldüğü üzere AND işlemini bir çok OR kapısı üzerinde paylaştırma esasına dayanmaktadır. Bu programlanabilir mantık elemanı mimarisi yönünden oldukça esnektir fakat iki tane programlanabilir düzleme sahip olması üretim ve yollanmada (mapping) zaman kaybına neden olmaktadır.



Şekil 3.1 PLA mantık devre elemanının iç mimarisi [39]

3.2.2 PAL

PAL PLA'lar gibi küçük PLD olarak da adlandırılabilir. PAL mimarisinde OR kapılarından oluşan sabit düzlemi takiben AND kapılarından oluşan bir adet programlanabilir düzlem bulunmaktadır. PAL'lar bu özellikleri ile ucuz ve yüksek hızlı performans sağlamaktadır.



Şekil 3.2 PAL mantık devre elemanının iç mimarisi [39]

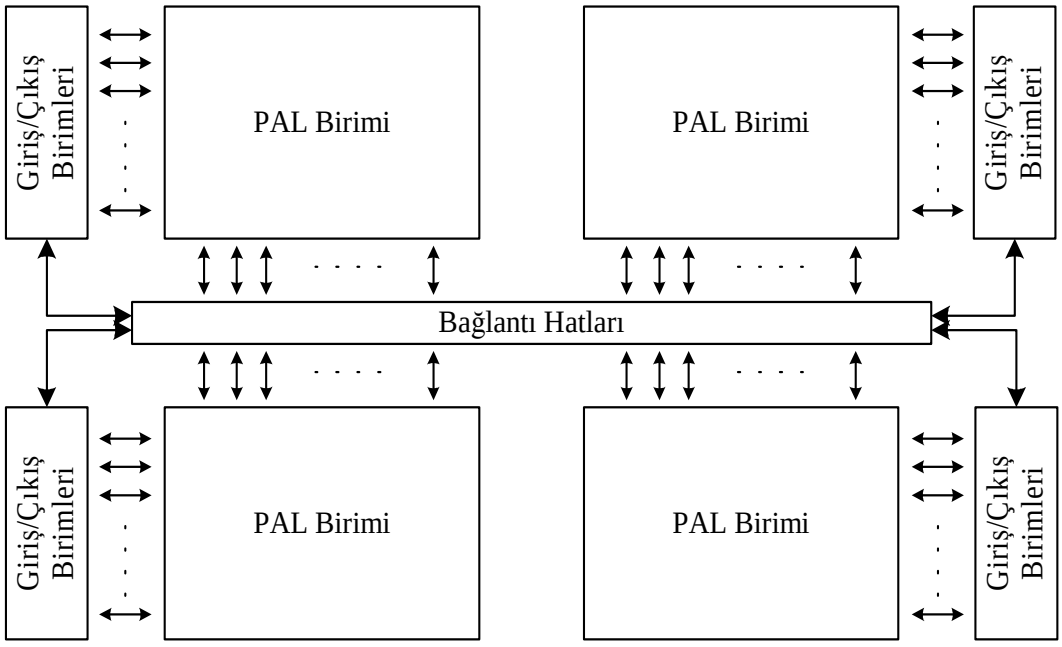
3.2.3 CPLD

CPLD SPLD'lere göre daha fazla kapasiteye sahip aygıtlardır. Gelişmiş programlanabilen elemanların teknoloji dünyasında, karmaşık programlanabilir mantık elemanları CPLD'leri oluşturmak için bir tek tümleşik devre içine birkaç adet PLD'nin bir araya getirilmesiyle yeni mantık elemanları geliştirilmiştir. CPLD, SPLD elemanlarının uzatılmasıyla 1970'lerin sonlarında tanıtılmıştır. Bu kavram tek bir eleman üzerinde bazı PLD bloklarıyla veya birleşik (makro) hücrelerle birlikte birbirleri arasındaki genel amaçlı bağlantıları da içerir. Temel olarak bir CPLD birbirine bağlanmış birkaç PLD bloğundan oluşur. Her bir PLD bloğunun giriş/çıkış birimleri küresel bağlantı dizisiyle bağlanmıştır. Her bir mantık bloğu üreticiye ve mimariye bağlı olarak 4 ile 16 arasında birleşik hücreye sahiptir. CPLD üzerindeki bir birleşik hücre tümleşik fonksiyon ve isteğe bağlı bir flip-flopu kapsayan yapıları içerir. Tümleşik mantık fonksiyonu tipik bir şekilde geniş bir yelpazede 4 ile 16 arasında çarpım terimleri içerir. Diğer bir deyişle bir birleşik hücre fonksiyonu birden çok girişe sahip olabilir, fakat mantık fonksiyonunun karmaşıklığı sınırlıdır. Yapısında bulunan pinden pine hızlı performansı nedeniyle CPLD'ler genellikle kontrol-yönlendirmeli

tasarımlarda çok iyidir. CPLD'de bulunan birleşik hücrelerin geniş bir yelpazede olması bu mantık elemanlarının karmaşık ve yüksek performanslı durum makineleri oluşturulurken kullanılmasının nedenidir.

CPLD'ler arasında, bir bireşik hücre içerisinde yer alan çarpım terimleri sayısı, bir birleşik hücrenin diğer hücreden çarpım terimi alıp almaması ve anahtarlama matrisinin tamamen veya kısmen kullanılması gibi mimari farklılıklar vardır. Anahtarlama matrisindeki bağlantı sayısı da mimariler arasındaki diğer önemli bir farktır. Tekrar programlanabilir bir cihazın tasarım değişikliklerinde devrenin iç yapısını değiştirmek kolay olduğu için aynı devre şemasını kullanmak daha verimlidir. Kısmi kullanılabilir matrislerde bekleme süresi, sabit değildir ve önceden hesaplamak mümkün değildir. Tam kullanılabilir CPLD'ler esneklik bakımından biraz kısıtlı olan bir iç mimariye sahiptir ve gecikmeler hem kısa (nanosaniye) hem de tahmin edilebilir boyutlardadır. Bunlara ilaveten bireysel SPLD bloklarının birbirleri ile olan bağlantıları programlanabilir bağlantı matrisleri vasıtasıyla programlanabilir. Bu da kullanıcıya daha çok esneklik sağlar ve tasarımcılar tarafından tercih edilir.

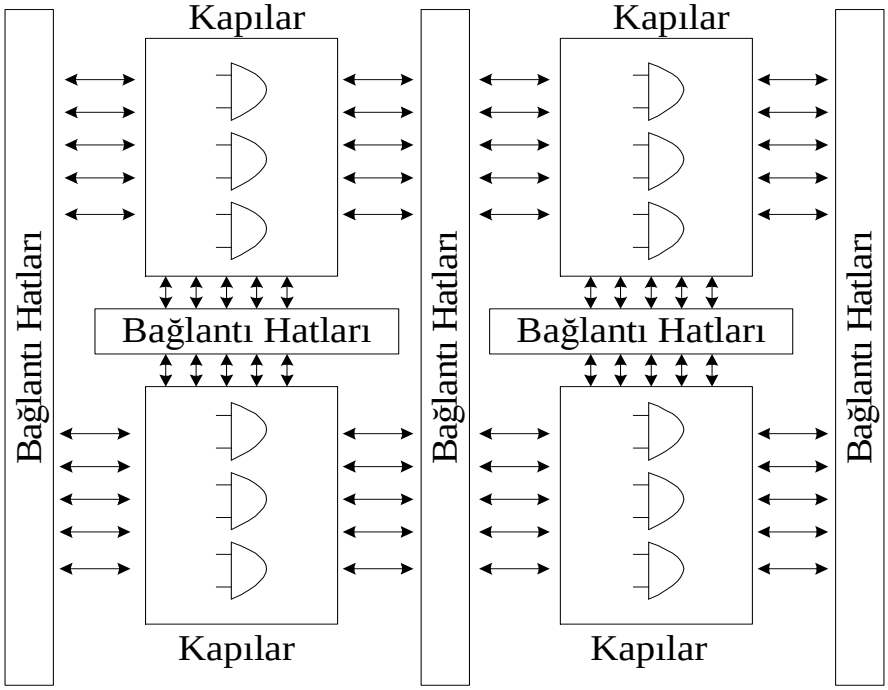
Bu elemanlar üreticilerin kullandığı teknolojiye bağlı olarak EPROM hücreleri, E2PROM hücreleri ya da Flash EPROM hücreleri vasıtasıyla programlanabilir. Şekil 3.3'te CPLD iç mimarisi görülmektedir.



Şekil 3.3 CPLD mantık devre elemanının iç mimarisi [38]

3.2.4 MPGA

Büyük mantıksal devreleri oluşturabilmek için MPGA çipler üretilmiştir. Genel anlamda bir MPGA yapısı içerisinde istenen mantıksal devreyi tasarlayabilmek için birbirleri ile ilişkilendirilmiş transistör satırları mevcuttur. Tasarımcı veya kullanıcı tarafından tanımlanan bağlantılar hem satır içerisinde hem de satırlar arasında bulunur. Böylelikle hem mantıksal kapılar oluşturulabilir hem de bu oluşturulan kapıları birbirine bağlayabilme imkânı sağlanmış olur. MPGA iç mimarisinde kullanılan metal tabakalar üretici tarafından tanımlandığı için üretim sayısı arttıkça hem zamandan hem de fiyat açısından kazanç sağlanmış olur. Şekil 3.4'te MPGA iç yapısı görülmektedir.



Şekil 3.4 MPGA mantık devre elemanı iç mimarisi [40]

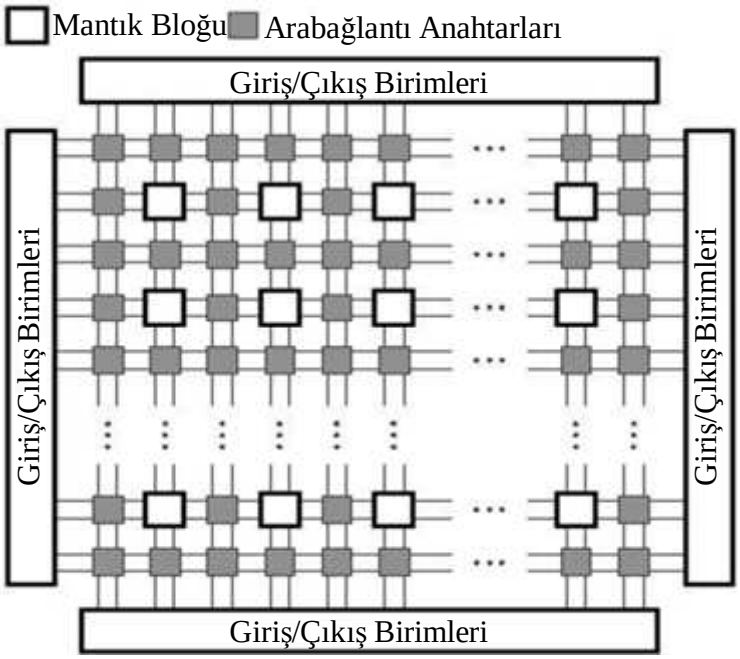
3.2.5 FPGA

FPGA, istenilen fonksiyonu gerçekleştirmek için programlanabilen mantık kapılar bütünüdür. Türkçesi "Alan Programlamalı Kapı Dizisi" olarak bilinir. 1980'lerin başında sayısal gömülü devrelerin gelişiminde bir boşluk olduğu bariz bir şekilde anlaşılmaya başlanmıştır. SPLD ve CPLD gibi programlanabilen elemanlar mevcut olup bunlar çok çeşitli şekillerde düzenlenebilme, hızlı tasarım ve değişiklik yapabilme özellikleri olmasına rağmen büyük ve karmaşık devreleri desteklememekteydiler. Diğer bir gelişim süreci de ASIC devreleri kapsamaktadır. Bu tür devreler oldukça büyük ve

karmaşık fonksiyonları desteklemekte fakat tasarımı oldukça pahalı ve zaman alıcıdır. Bu iki gelişim arasındaki boşluğu doldurmak ve her iki yapının da avantajlarını birleştirmek için 1984 yılında Xilinx firması tarafından FPGA geliştirilmiştir.

FPGA'lar programlanabilen mantık blokları ve ara bağlantılardan oluşur. Kullanıcı, tasarım sürecinde mantık bloklarını ve ara bağlantıları üretici firma tarafından sağlanan arayüz yazılımları yarımıyla gerçekleştirebilmektedir. FPGA'lar, tasarım esnekliği, düşük maliyet ve hızlı örnekleme zamanı gibi özellikleri nedeniyle sayısal tasarım ortamlarında vazgeçilmez bir hale gelmiştir.

FPGA'lar büyük ve karmaşık devrelerin tasarlanması için son derece uygundur. FPGA'da bulunan mantık elemanların her biri istenilen mantıksal fonksiyonu yapmak üzere ve bağlantı noktalarının her biri bağlantı oluşturacak ya da oluşturmayacak şekilde programlanabilir. Şekil 3.5'te FPGA'nın genel bir görünümü verilmiştir.



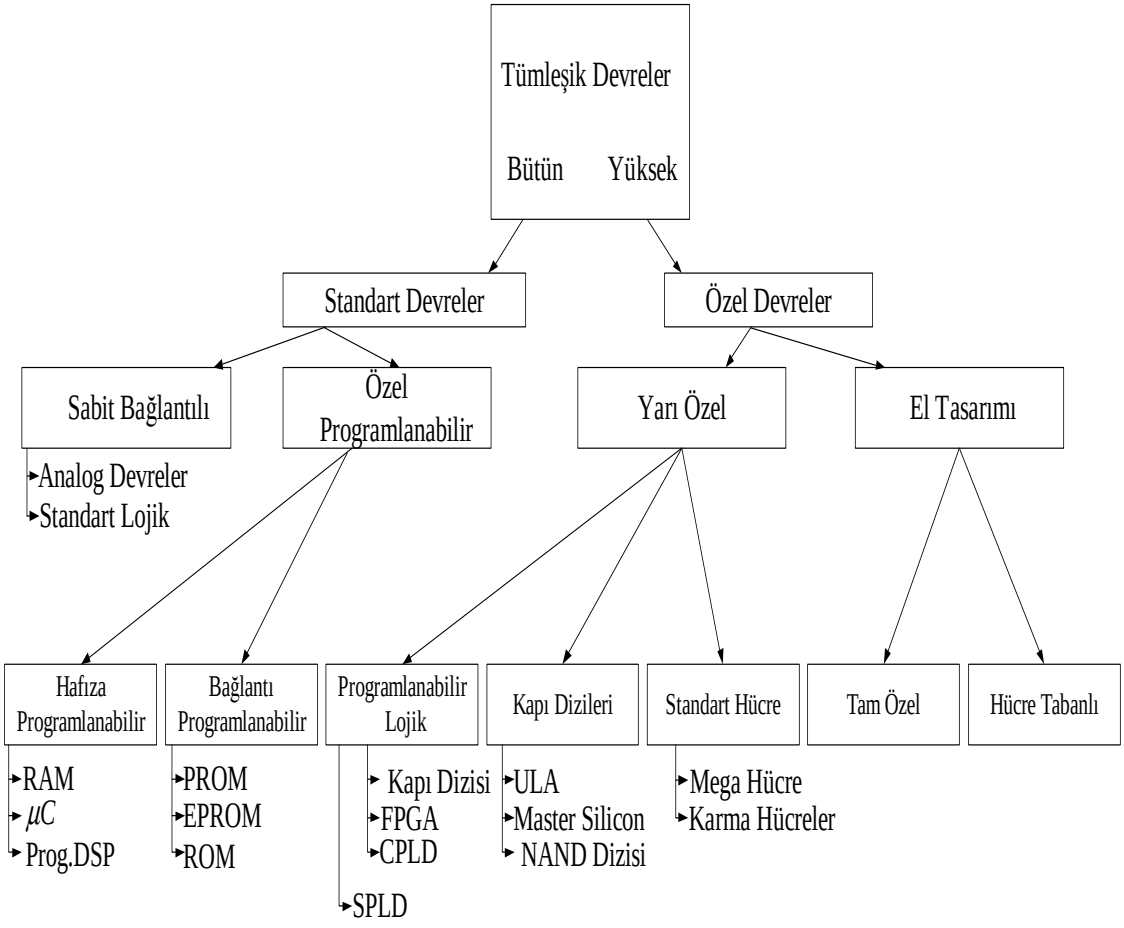
Şekil 3.5 FPGA blok diyagramı [40]

FPGA ile CPLD arasında mimari açıdan farklar mevcuttur. CPLD, üzerinde saat üreteci ile beslenen az sayıda programlanabilen mantık blokları, devreler ve bu sebeple sınırlı bağlantı içerir. Bu da çok fazla tasarım esnekliği sağlayamamaktadır. FPGA da ise tüm bağlantılar egemendir. Böylelikle daha esnek ve daha karmaşık yapılı tasarımlar gerçeklenebilir. Bu iki mimari arasında hafıza ve yüksek düzey fonksiyonlar (çarpıcı ve toplayıcı gibi) açısından da farklar bulunmaktadır.

Günümüzde simetrik dizi, satır bazlı, hiyerarşik PLD ve kapı denizi olmak üzere 4 farklı tipte FPGA mevcuttur. FPGA üretiminin %80'i Altera ve Xilinx firmaları tarafından gerçekleştirilmektedir.

3.3 VLSI Sisteminde FGPA'nın Yeri

Very Large Scale Integration/ Çok Büyük Ölçekli Birleşme (VLSI) devreleri sınıflandırılması Şekil 3.6'da gösterilmektedir [41]. FPGA'lar bu sınıflandırmada alan programlanabilir mantık (FPL) arasında yer almaktadır. FPGA'lar ASIC ya da özel uygulama gerçekleştiren devreler içindedir.



Şekil 3.6 VLSI'ya göre programlanabilir mantık sınıflandırması [41]

3.3.1 FPGA'ların programlama teknolojileri

FPGA'lar, iç yapılarında bulundurdukları programlanabilir anahtarlar ve mantık bloklarının programlanabilirlik özelliklerinin gerçeklenmesine göre sınıflara ayrılırlar.

3.3.1.1 Statik RAM programlama teknolojisi

SRAM programlama teknolojisinde programlanabilir bağlantıları kontrol eden geçiş transistörü iletim kapısında veya çoklayıcı yapısında ileri sürülmektedir. Tümleşik devreler üzerinde SRAM hücrelerine yer ayrılmamıştır ve bunlar kontrol ettikleri mantık elemanlar arasında dağıtılmışlardır.

SRAM teknolojisi standart tümleşik devre teknolojisinin sağladığı hızlı yeniden programlanabilme, çok karmaşık devrelerde dahi küçük güç tüketimi ve programlama tamamlandığı anda tasarımı test edebilme imkanı sağlaması gibi avantajlarının yanı sıra FPGA'nın enerjisi kesildiği anda tüm programlanmış anahtarların özelliklerini yitirmesi ve bu teknolojisinin büyük yer kaplaması gibi dezavantajları mevcuttur.

3.3.1.2 Anti sigorta tabanlı programlama teknolojisi

Bu teknolojide yollanma kanalları arasındaki bağlantı FPGA programlanmadan önce yapılmamıştır, bu bağlantılar programlama esnasında gerilim uygulanmasıyla oluşturulur. Böylelikle yalnızca gerekli olan bağlantılar yapılarak kullanılmayan bağlantı sayısı en aza indirgenerek programlama süresi kısaltılmış olur.

Anti sigorta teknolojisi, programlama için gerekli gerilimin entegre içine dağıtılmasını sağlayan transistörler yarıiletken üzerinde geniş alan kaplamasına rağmen diğer teknolojilerle kıyaslandığında daha küçük bir alana ihtiyaç duyar. Bu teknolojiyle programlanan FPGA'lar bir kez programlanabildiği için ilk üretimleri pahalı olmaktadır [42].

3.3.1.3 EPROM ve EEPROM programlama teknolojisi

Bu teknolojide kayan kapı programlaması teknolojisi kullanılır ve bu teknoloji transistör üzerinden akan gerilimin bir kapıya depolanması temeline dayanır. Yüksüz durumda bulunan transistör programlanmak istenildiği takdirde kaynak ve akaç arasında bir akım akıtılması yeterlidir ve böylelikle kapı ile dielektrik arasında yük depolanması sağlanarak transistör iletime geçer. Bu yaklaşım flash ya da EEPROM hafıza hücreleri üzerinde kullanılır. Bu hücreler uçucu değillerdir ve enerjileri kesilse dahi üzerlerindeki bilgiyi kaybetmezler [39].

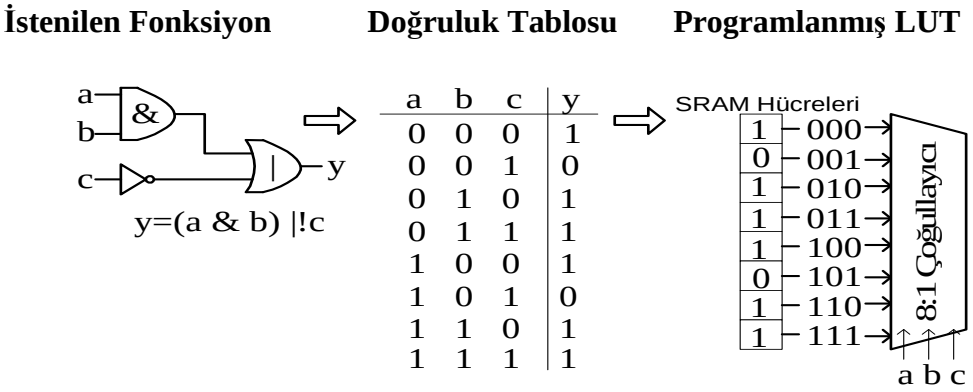
3.3.2 FPGA’ların mantık hücre yapısı

Programlanabilir mantık bloklar doğruluk tablosu tabanlı ve çoklayıcı tabanlı yapı olmak üzere iki temel hücre yapısına sahiptir.

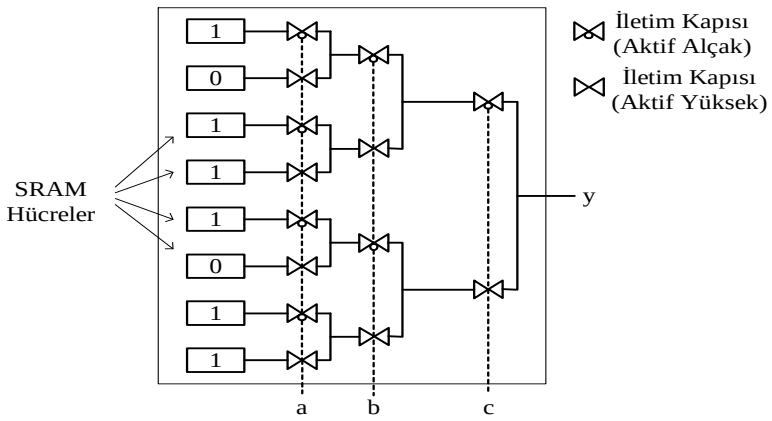
3.3.2.1 Doğruluk tablosu tabanlı yapı

Doğruluk tablosu tabanlı yapıdaki temel blok LUT (Look Up Table) adı verilen, bir veya birden fazla değişkenli fonksiyonları gerçekleyen yapıdır. Giriş sinyalleri bir grup halinde LUT içinde indeks olarak kullanılır. Bu tablonun içindekiler her bir giriş kombinasyonu istenen değeri içerecek şekilde düzenlenir.

Doğruluk tablosu tabanlı yapı SRAM hücreleri ile gerçeklenmektedir. Bu yapıda genel olarak kullanılan teknik, girişlerin iletim kapılarından oluşan kaskat yapıları kullanarak istenen SRAM hücresini seçtirmesi işlemidir. SRAM hücreleri yapılandırma amaçlı olarak birbirleriyle bir zincir yapı şekli üzerinde bağlanabilirler [43]. Şekil 3.7’de 3 girişli bir doğruluk tablosu tabanlı yapı örneği ve Şekil 3.8’de de aynı fonksiyonun iletim kapıları ve SRAM hücreleriyle oluşturulan yapısı görülmektedir.



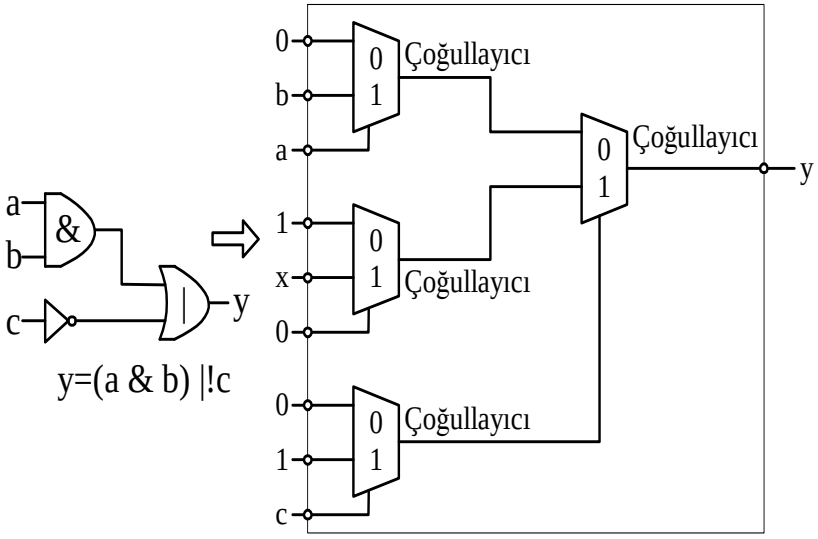
Şekil 3.7 Bir fonksiyona göre düzenlenmiş LUT'un basit gösterimi [43]



Şekil 3.8 Programlanmış LUT'un daha gerçekçi gösterimi [43]

3.3.2.2 Çoklayıcı tabanlı yapı

Şekil 3.9'da da görüldüğü gibi 3 girişli ve bir çıkışlı bir fonksiyon yalnızca çoğullayıcılar kullanılarak gerçeklenmiştir. FPGA ya da mantık elemanları, giriş blokları mantıksal olarak 1 veya 0 seviyesinde, elemanının kendisi şeklinde veya elemanın tersi şeklinde diğer bloklardan gelecek şekilde veya başlangıç girişi şeklinde alınarak programlanabilir. Bu da her bloğa olasıl fonksiyonların bir yığınının gerçeklenebilmesi için düzenlenebilmesine birçok imkan tanımaktadır. Bu yapının temel bloğu çeşitli konfigürasyonlardan ve olabildiğince az mantık kapılarından oluşur.



Şekil 3.9 Bir mantık fonksiyonun çoklayıcı tabanlı yapı kullanılarak gerçeklenmesi [41]

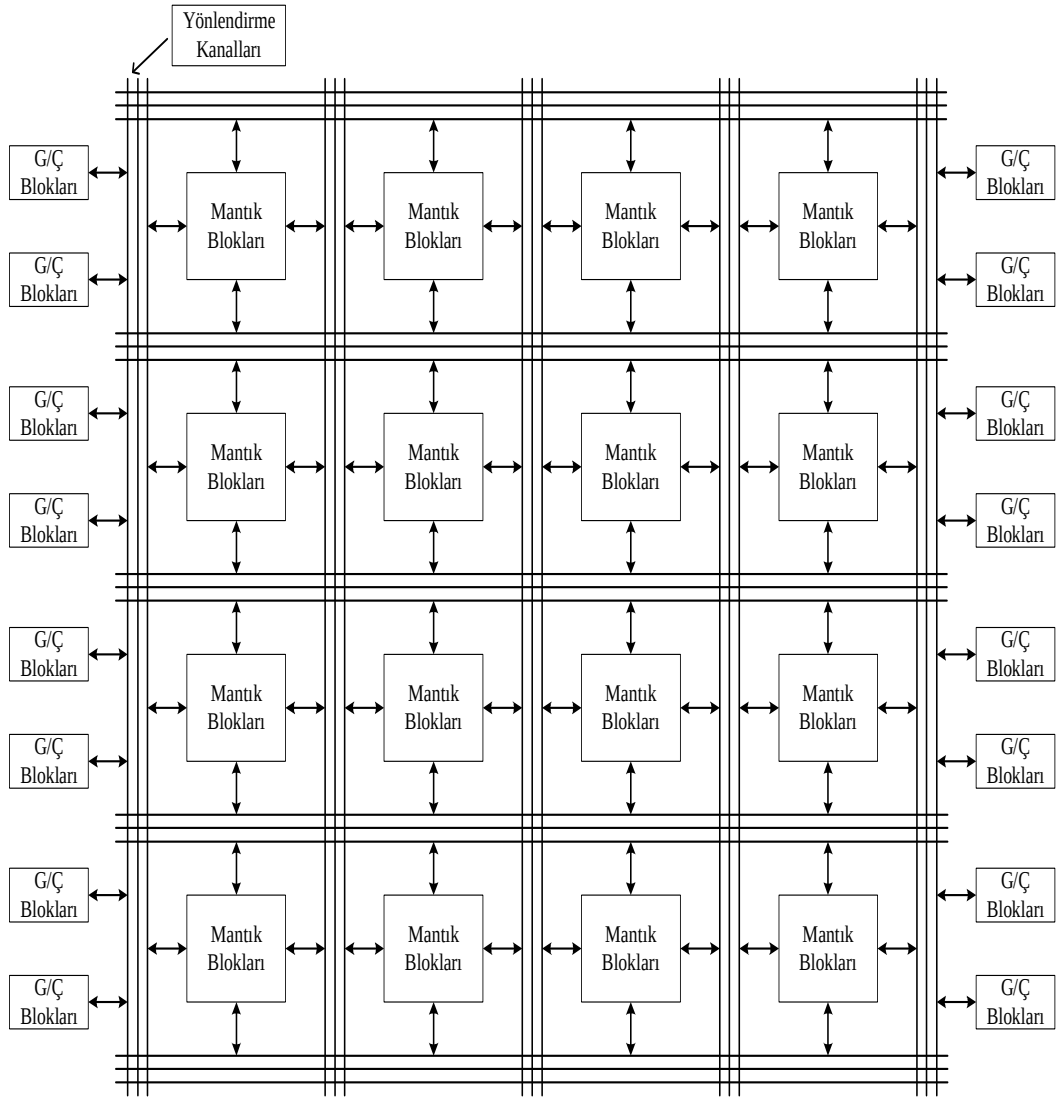
3.3.3 FPGA mimarisi

FPGA üç temel bloktan oluşmaktadır. Bunlar mantık blokları, giriş/çıkış blokları ve bağlantı bloklarıdır. FPGA'lar, bu temel blokların haricinde aritmetik işlem blokları, bellek blokları gibi özel bloklar da içermektedirler.

Mantık blokları, iki boyutlu dizi oluşturacak şekilde konumlandırılmışlardır ve bağlantı blokları da mantık bloklarının satır ve sütunları arasındaki yatay ve düşey bağlantıyı sağlamakla görevlendirilmişlerdir. Bağlantı blokları, mantık blokları ile bağlantı hatlarının farklı şekillerde bağlanabilmeleri ve bu sebeple mimaride esneklik kazandırılabilmesi için programlanabilir anahtarlardan oluşurlar.

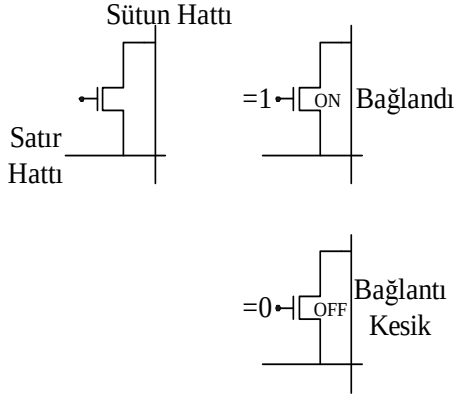
Bir bağlantı noktasının bağlı olup olmaması o noktadaki transistörün açık ya da kapalı olması ile ayarlanır [44]. Bağlantı noktalarında kullanılan programlanabilir anahtarların

satır ve sütun hatlarında bağlantı oluşturması ve yapıları Şekil 3.11'de görülmektedir.



Şekil 3.10 FPGA iç yapısı ve blokları [39]

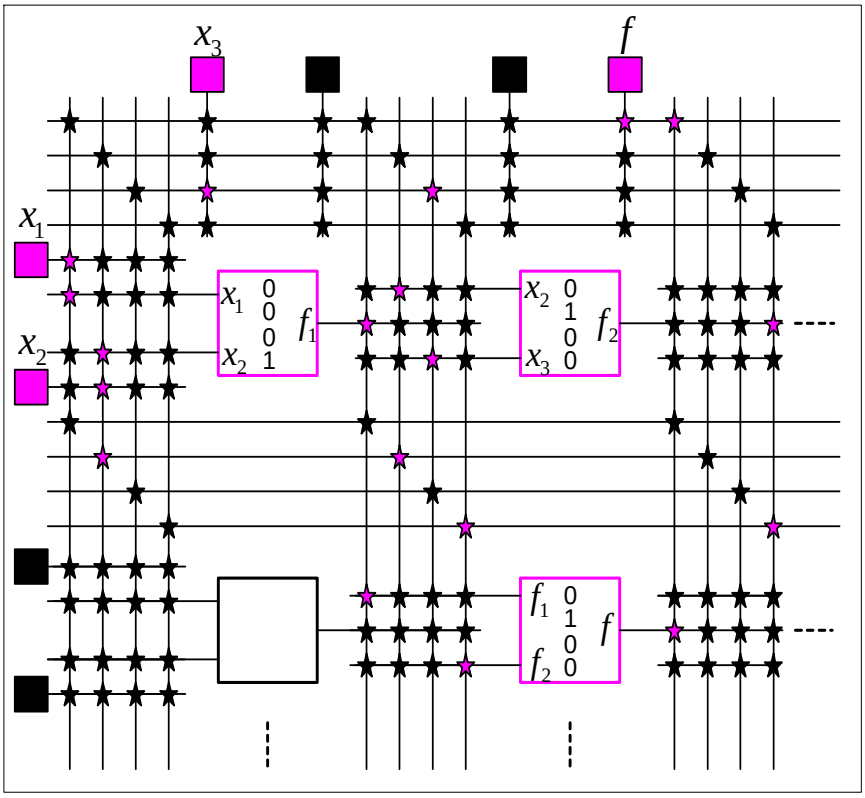
FPGA'larda bir adet yapılandırma (Configuration) biti bulunur. Bu bitler teker teker kaydırılarak yerleştirilir [44].



Şekil 3.11 FPGA ara bağlantıları [40]

FPGA'nın sahip olduğu alan yapısı dikkate alınarak, FPGA üzerinde mantık devrelerinin gerçeklenebilmesi için tasarlanacak olan devreyi gerçekleyecek olan fonksiyonların her birinin küçük mantık blokları yapısına indirgenmesi gerekmektedir. Bu da Computer Aided Design/Bilgisayar Destekli Tasarım (CAD) yazılımları ile gerçekleştirilebilmektedir.

Aşağıda bir devreyi gerçekleştirebilmek için programlanmış bir FPGA görülmektedir.



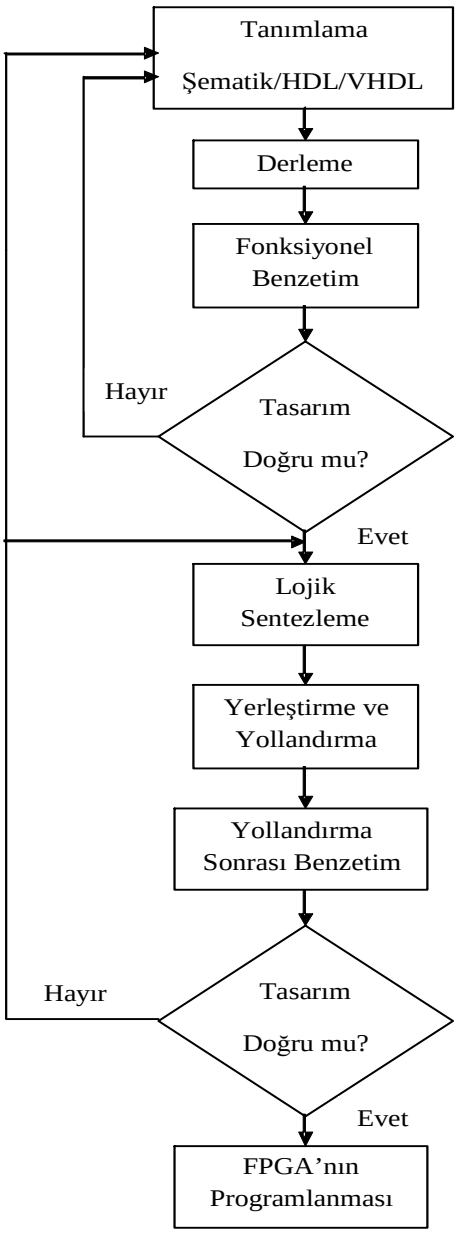
Şekil 3.12 Programlanmış FPGA'nın bir bölümü [40]

ASIC devrelerden aldığı paralelleştirilebilirlik özelliği ve tekrar programlanabilme özelliği ile FPGA'lar GKF uygulaması için ve GKF algoritmalarının işleyiş sürelerinin kısaltılması yani örnekleme zamanının düşürülmesi için çok uygundurlar.

3.3.4 FPGA kullanarak gerçekleştirilen devrelerin tasarım aşamaları

Tasarım başlangıcı, devrede gerçeklenecek olan fonksiyonların yüksek seviyeli donanım tasarımı ile tanımlanmasıyla başlar. Tanımlanan fonksiyonlar şematik düzenleme programları yardımı ile şematik fonksiyon bloklarına çevrilir. Şematik biçime çevrilen fonksiyonlar derlenerek bağlantı listesi (netlist) formuna çevrilir. Oluşturulan fonksiyonların üzerlerine yüklenen görevleri yerine getirip getirmediği ise

benzetim programları ile test edilir. Yapılan test sonucunda eğer ki gerekiyorsa değişiklikler yapılarak tekrar tasarım başlangıcına gider. Eğer değişiklik yapılmasına gerek yoksa sentezleme, yollama ve yerleştirme ile yollandırma sonrası benzetim işlemleri yapılarak gerçeklenen tasarım FPGA'ya gömülür.

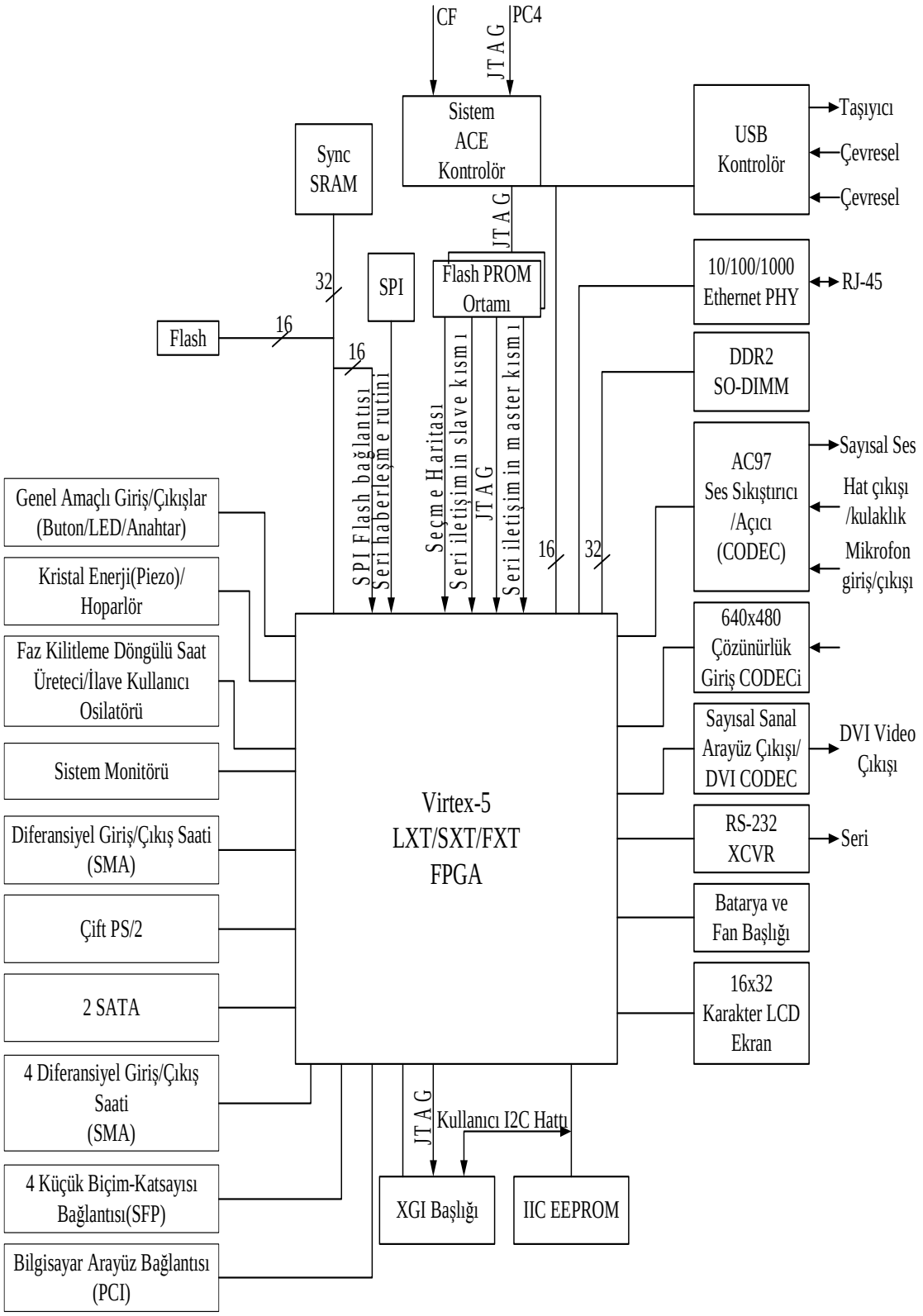


Şekil 3.13 FPGA kullanarak gerçekleştirilen tasarımların akış şeması [40]

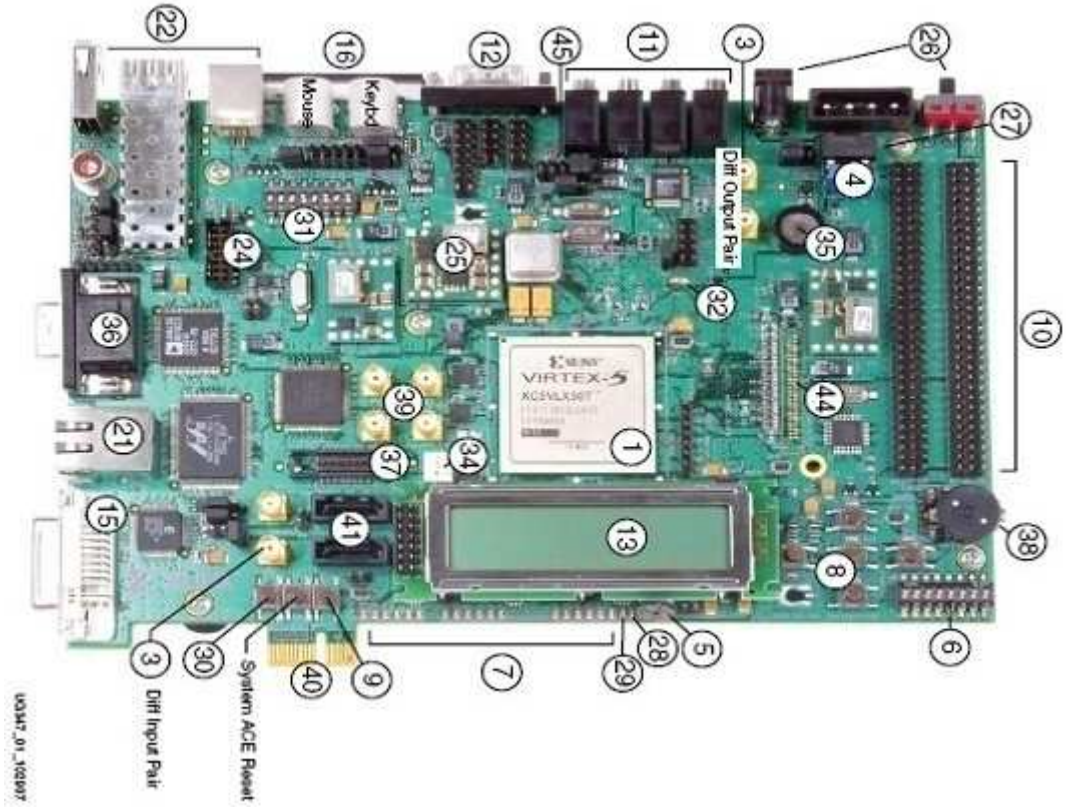
3.4 Xilinx Virtex 5 XC5VSX50T ve ML506 Geliştirme Kartı

Çalışmamızda Xilinx Virtex5 serisi FPGA ve bu FPGA'yi üzerinde işlemci olarak barındıran ML506 geliştirme kartı kullanılmıştır. Aşağıda Xilinx Virtex 5'in blok

şeması ve bulundurduğu birimler ile ML506 geliştirme kartının üstten ve alttan görünüşüyle üzerinde barındırdığı birimler gösterilmiştir.



Şekil 3.14 Xilinx Virtex-5 blok şeması [45]



Şekil 3.15 ML506 geliştirme kartının üstten görünüşü [45]

Şekil 3.16 ML506 geliştirme kartının alttan görünüşü [45]

3.4.1 ML506 geliştirme kartının üzerinde bulunan birimler

Şekil 3.15 ve Şekil 3.16'da numaralandırılmış birimler aşağıda verilmiştir.

1. Xilinx Virtex 5
2. DDR2 SODIMM Ram hafıza modülü
3. Diferansiyel saat giriş ve çıkışı
4. Osilatörler
5. LCD parlaklık kontrast ayarı
6. Genel amaçlı anahtarlar
7. Kullanıcı ve hata LED'leri
8. Kullanıcı butonları
9. CPU reset butonu
10. Genişleme başlığı
11. Stereo AC97 ses kodlayıcı-kod çözücü
12. RS-232 seri haberleşme kanalı
13. 16 karakter-2 satırlı LCD ekran
14. 8 kb EEPROM'lu IIC veri yolu
15. DVI (Harici video ekranı)bağlantısı
16. Fare ve klavye bağlantısı
17. Compact (Sıkıştırılmış) Flash kart bağlantısı
18. SRAM
19. 32 Mb flash hafıza
20. Xilinx XC95144XL CPLD
21. 3 hızlı Ethernet girişi
22. USB denetleyicileri
23. Xilinx XCF32P PROM düzenleme hafızaları
24. JTAG düzenleme giriş (FPGA'nın programlanabilmesini ve derlenmesini sağlayan giriş)
25. Güç kaynağı
26. AC adaptörü ve güç girişi
27. Güç belirtme LED'i
28. FPGA'in başarıyla düzenlendiğini belirten DONE LED'i
29. FPGA'e gücün başarıyla geldiğini gösteren INIT LED'i
30. Programlama anahtarı
31. Adres ve durum düzenleme anahtarları
32. Şifreleme anahtar pili

33. SPI (Serial Peripheral Interface/Seri Çevresel Haberleşme Arayüzü) Flash hafızası
34. Fan denetleyicisi ve sıcaklık/gerilim ekranı
35. Basit ses tonları çıkartan piezo ses dönüştürücüsü
36. VGA görüntü kodlayıcı-kod çözücü girişi
37. JTAG izleyici
38. Döngü kodlayıcı
39. Diferansiyel GTP/GTX (Genel Bağlantı) giriş ve çıkışları
40. PCI arayüzü
41. SATA bağlantısı
42. SFP (Small Form-Factor Pluggable/Küçük Biçim-Katsayısı Bağlantısı)
43. GTP/GTX saat devresi
44. Mantıksal analiz dokunma noktaları
45. Sistem ekranı

3.5 Çalışmada Kullanılan Haberleşme Protokolleri ve Arayüzleri

Bu tez çalışmasında seri haberleşme ve ethernet haberleşmesi protokolleri kullanılmıştır.

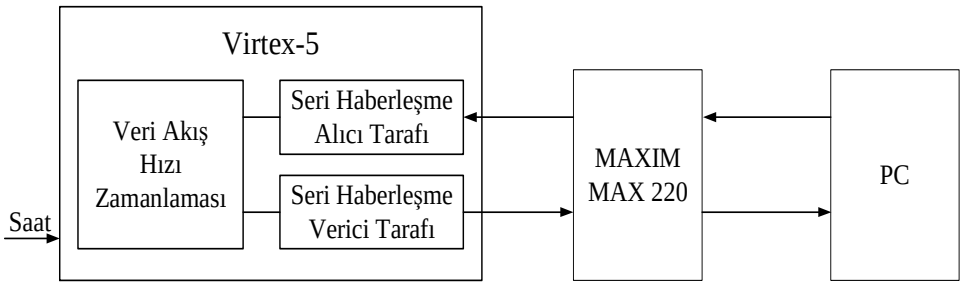
3.5.1 RS-232 seri haberleşme protokolü

ML506 geliştirme kartı FPGA'nın diğer birimlerle seri veri iletişimi için bir adet erkek DB-9 RS-232 seri portu barındırmaktadır. Seri port FPGA ve diğer araçlar arasında bir taşıyıcı hat görevi görmektedir ve bunun için herhangi bir bilgisayarla bağlantı sırasında ara kablo gerekmektedir. ML506 üzerindeki RS-232 seri portu saniyede 115200 bit gönderme kapasitesine sahiptir (Baud=115200 bit/s). FPGA ve RS-232 sinyalleri arasındaki gerilim uyumlaması için bir arayüz çipi kullanılmaktadır.

FPGA sadece TX ve RX pinleri ile bağlantılıdır. Diğer RS-232 sinyalleri, donanım akış kontrol sinyalleri kullanılmazlar. Bir bilgisayar ile haberleşirken akış kontrolü kesilmelidir.

3.5.1.1 Standart seri haberleşme protokolü

UART_TX ve UART_RX makroları, Xilinx aygıtını bilgisayar ya da mikrokontrolörle haberleştirmek için seri haberleşme protokol standartlarını referans alarak kullanılabilirler. Eğer RS-232 sinyallerinin tümü aktifse MAXIM MAX220 (multi-channel RS232 drivers/receivers) gibi ara araçlar kullanılmalıdır [41].



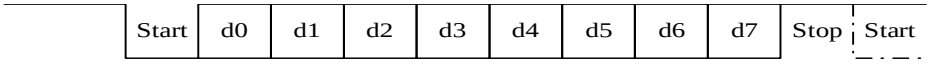
Şekil 3.17 Xilinx ile standart seri haberleşme protokolünün uygulanışı [46]

3.5.1.2 Seri haberleşme ile FPGA'dan bilgisayara veri aktarılması

Çalışmalarımızda uyguladığımız algoritmaların sonuçlarını bilgisayara seri haberleşme aracılığıyla taşıdığımız anda sadece FPGA'nın iletim özelliği yani seri haberleşme protokolünün iletici (Transmitter) kısmı kullanılmıştır. Burada FPGA bilgisayara bilgiyi gönderen araç olmuştur. Bilgisayardan FPGA'ya veri gönderme işlemi de algoritmalarının çevrim sürelerinin uzatılması esnasında uygulanmıştır.

Seri haberleşmede UART_TX pini her 8 bitlik verinin iletilmesi için bir zamanlama referansı kullanır. Bu zamanlama referansına baud yani saniyede gönderilebilecek bit sayısı denir.

Veri seri şekilde belirlenen zamanlama referansına göre önce LSB (En düşük değerlikli bit)'den başlayarak MSB (En yüksek değerlikli bit)'e kadar iletilir. Alıcıya verinin transfer edilmesi için öncelikle verici tarafından bir adet '0' biti gönderilir. Buna start (başlama) biti denir. Verinin iletilmesi için gönderilen başlama bitinden sonra 8 bitlik veri alıcı tarafına transfer edilir ve transfer işlemi bittikten sonra da verici tarafından bir bitlik '1' bilgisi gönderilir. Buna ise stop (durdurma) biti denir. Şekil 3.18'de seri haberleşme veri akışı verilmiştir [41].



Şekil 3.18 Seri haberleşme ile veri gönderme akışı [46]

3.5.2 Ethernet haberleşmesi

Ethernet, Local Area Network/Yerel Ağlar (LAN) için kullanılan veri çerçevesi (Data Frame) tabanlı bir bilgisayar ağı teknolojileri ailesidir. Ethernet 1973-1975 yılları arasında Xerox PARC tarafından geliştirildi.

Ethernet ilk olarak ortak bir eşeksenli kablo üzerinden birbirine bağlanan bilgisayarların yayın iletimi yöntemiyle haberleşmesi fikrine dayanmaktadır. Kullanılan yöntemler kısmen radyo sistemlerine benzemekte, ancak, kablolu bir yayın iletimi sistemindeki çakışmaları saptamanın radyo yayınına kıyasla çok daha kolay olması gibi temel farklılıkları da mevcuttur. "Ethernet" adı iletişim kanalını oluşturan ortak kablonun ethere benzetilmesinden gelmektedir.

Ethernet bu öncel ve göreceli olarak basit kavramdan, günümüzdeki pek çok LAN altyapısını oluşturan karmaşık ağ teknolojisi yapısına evrimleşmiştir. Eşmerkezli kablolamanın yerini düşük kurulum masrafı, yüksek güvenilirlik, noktadan-noktaya ağ yönetimi ve arıza bulma kolaylıkları gibi avantajlar sebebiyle ethernet hublar ile birleştirilmiş noktadan-noktaya bağlantılar ve/veya ağ anahtarları almıştır. StarLAN Ethernet'in eşmerkezli kablolama yapısından hub ile yönlendirilen bükülü tel çifti ağ yapısına evrimleşmesindeki ilk adımdır. Bükülü tel çifti kablolamanın gelişi kurulum masraflarını eski ethernet teknolojileri de dahil olmak üzere benzer teknolojilere kıyasla önemli ölçüde düşürmüştür.

Teknik kabiliyetlerine rağmen ethernetin başarısı hızlı standartlaştırılmasına bağlanmıştır. Bunun için Uluslararası Elektrik ve Elektronik Mühendisleri Enstitüsü (International Electrical and Electronics Engineering (IEEE)), Avrupa Bilgisayar Üreticileri Birliği (European Computer Manufacturers Association (ECMA)), Uluslararası Elektroteknik Komisyonu (International Electrotechnical Commission (IEC)) ve Uluslararası Standartlaştırma Kurumu (International Organization for Standardization (ISO)) içinde koordineli çalışmalar yürütülmesi gerekmiştir.

Şubat 1980'de IEEE Yerel Ağların (LAN) standartlaştırılması için IEEE 802 adında bir proje başlatmıştır. 1982 yılında IEEE 802.3 CSMA/CD onaylanmıştır. Ethernet'in uluslararası standart olarak kabulü de ISO/IEEE 802/3 Uluslararası Standardı olarak 1984 yılında olmuştur.

Ethernet istasyonları birbirlerine donanım katmanı üzerinden veri bloklarından oluşan ve ayrı ayrı gönderilip alınan veri paketleri göndererek haberleşmektedir. Diğer IEEE 802 LAN'larda olduğu gibi her ethernet istasyonunun paket gönderme ve alma adreslerini belirleyen 48-bitlik kendine özgü MAC adresleri vardır. Ağ bağdaştırıcı kartları (Network Interface Card (NIC)) ya da çipleri normalde diğer ethernet istasyonlarına gönderilen paketleri kabul etmezler. Bağdaştırıcılar genellikle kendine özgü tek bir genel (global) adrese sahip olarak gelir ancak kart değiştirildiğinde adres çakışması olmaması ya da yerel yönetim ağları içinde kullanıldıklarında bu adres değiştirilebilir.

10 Mbit/s hızındaki eşmerkezli kablodan 1 Gbit/s hızındaki noktadan-noktaya bağlantıya kadar tüm Ethernet türevleri aynı veri çerçevesi formatını (dolayısıyla üst katmanlarda aynı arayüzü) kullandıklarından kolaylıkla birbirlerine bağlanabilirler.

Ethernetin çok yaygın olması, donanım maliyetinin giderek düşmesi ve bükülü tel çifti ethernet arayüzünün fazla yer kaplamaması nedeniyle pek çok üretici PC anakartlarına ve FPGA kartlarına Ethernet arayüzü koymakta, böylelikle ayrı bir ağ bağdaştırıcı kartına gerek kalmamaktadır.

Hali hazırda, kullanılan fiziksel ortam ve hız yönünden farklılıklar gösteren pek çok Ethernet türü vardır. En yaygın olarak kullanılan türler 10BASE-T, 100BASE-TX, ve 1000BASE-T 'dir. Her üçünde de 8P8C modüler bağlayıcı (connector) kullanılır. Sırasıyla 10 Mbit/s, 100 Mbit/s, ve 1 Gbit/s veri hızlarında çalışmaktadırlar. Ancak her birinin çalışabilmesi için farklı kablolama gerektiğinden kurulumcular sunuculara yapılan kısa bağlantılar haricinde 1000BASE-T'yi kullanmaktan uzak durmaktadırlar.

Çizelge 3.1 1500 baytlık maksimum iletim birimi için gönderildiği haliyle, bütün bir Ethernet çerçevesini göstermektedir. Daha yüksek hızlı bazı gigabit Ethernet uygulamaları "jumbo frame" denilen daha büyük çerçeve boyutlarını desteklemektedir. Dikkat edilmesi gereken nokta Giriş ve Çerçeve Başlangıç Sınırlayıcı alanlarındaki bit düzenlerinin bayt olarak değil karakter dizisi olarak yazılmış olmasıdır. Bu gösterim IEEE 802.3 standardında kullanılan ile uyuşmaktadır. Bir Oktet modern bilgisayarlarda "bayt" olarak adlandırılan sekiz bitlik veri anlamındadır.

Çizelge 3.1 802.3 MAC Çerçevesi

Giriş	**Çerçeve Başlangıç Sınırlayıcı**	**Hedef MAC Adresi**	**Kaynak MAC Adresi**	**Ether Type**	**Yararlı Yük**	**CRC 32**	**Çerçeveler arası Boşluk**
7 oktet 10101010	1 oktet 10101011	6 oktet	6 oktet	2 oktet	46–1500 oktet	4 oktet	12 oktet
		64–1518 oktet					
72–1526 oktet							

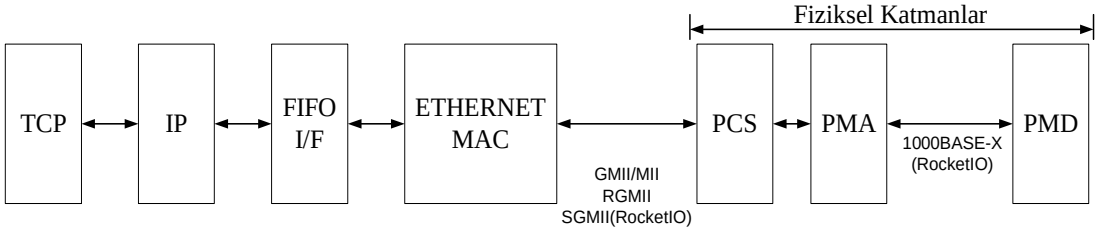
Bu çalışmada FPGA ile PC arasında Ethernet haberleşmesinin sağlanması için Virtex 5'in yapısında modül olarak gömülü olan Ethernet bileşeni (Component) ve bu bileşenin örnek VHDL projesi kullanılmıştır. Kullanılan bileşenin ismi ise Üç-Hızlı Ethernet MAC Bandı'dır (Tri-Mode Ethernet MAC Wrapper v1.7).

3.5.2.1 Virtex 5 gömülü üç-hızlı ethernet MAC bandı (Virtex 5 fpga embedded tri-mode ethernet MAC wrapper v1.7)

LogiCORE™ IP Virtex®-5 FPGA Gömülü Üç-Hızlı Ethernet MAC Bandı, Virtex 5 LXT, SXT, FXT ve TXT FPGA'larının yapısında bulunan Gömülü Üç-Hızlı Ethernet MAC Bandı için Xilinx firmasının CORE GENERATOR (Çekirdek Üreteci) yazılımı sayesinde HDL band dosyalarını otomatik olarak üretmektedir.

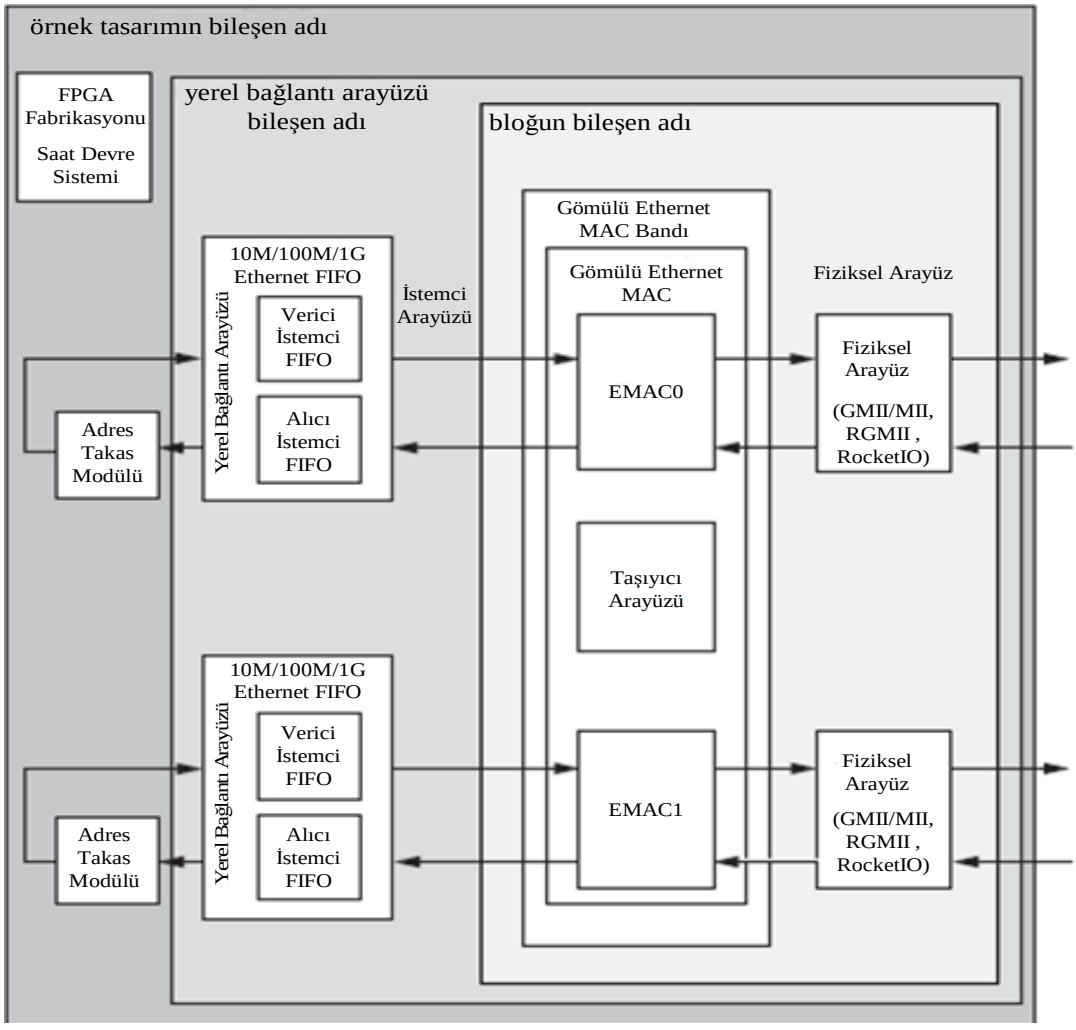
Bu yazılım sayesinde kullanıcı tek ya da tüm Ethernet Mac bileşenlerini (EMAC0/EMAC1) aktif duruma getirebilir. Bu bileşenlerin tüm niteliklerini kontrol edebilir. Ethernet haberleşmesinin tüm arayüzlerini ve haberleşme hızlarını belirleyebilir. Bu belirlemelerden ve bileşenin oluşturulmasından sonra CORE GENERATOR kullanıcıya tüm kolaylıkları sağlayarak bir adet Verilog ya da VHDL dili ile örnek bir proje oluşturmakta ve kullanıcı bu örnek vasıtasıyla haberleşme protokolünü oluşturabilmektedir. Bu yazılım, haberleşme protokolünün akışında kullanılan File Input File Output/Veri Saklama Ünitesi (FIFO)'nin geri çevrimli bir şekilde kullanılabilmesini sağlar. Böylelikle FPGA'dan gönderilecek sekizer bitlik verilerin biriktirilip daha hızlı bir şekilde bilgisayara aktarılması sağlanır [45]. Çalışmamızda VHDL dili kullanarak haberleşme protokolü sağlanmıştır ve oluşturulan Ethernet modülünün yalnızca Yerel Bağlantı (LocalLink) arayüzüne müdahale edilmiştir. Diğer bir deyişle gönderilecek veriler, alıcının (bilgisayar) MAC adresi,

göndericinin (FPGA) MAC adresi ve verinin uzunluğu kullanıcı tarafından belirlenmiştir. Aşağıda Ethernet blok şeması ve akışı belirtilmiştir.



Şekil 3.19 Ethernet blok şeması [47]

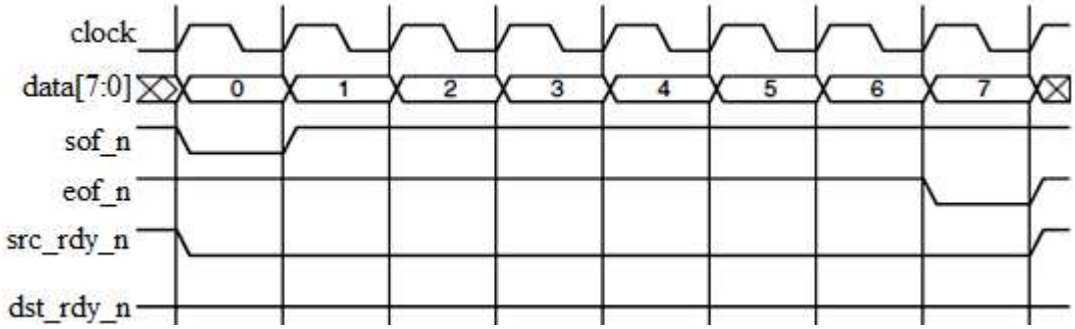
Burada Ethernet MAC, Ethernet protokolünün belirlenmesinde ve hata belirleme işleminde görev üstlenir. MAC her hangi bir alt katmana bağlantı kurmakta serbesttir. Şekil 3.20'de Hardware Description Language/Donanım Tanımlama Dili (HDL) ile oluşturulan bir Ethernet haberleşme sisteminin blok diyagramı gösterilmiştir.



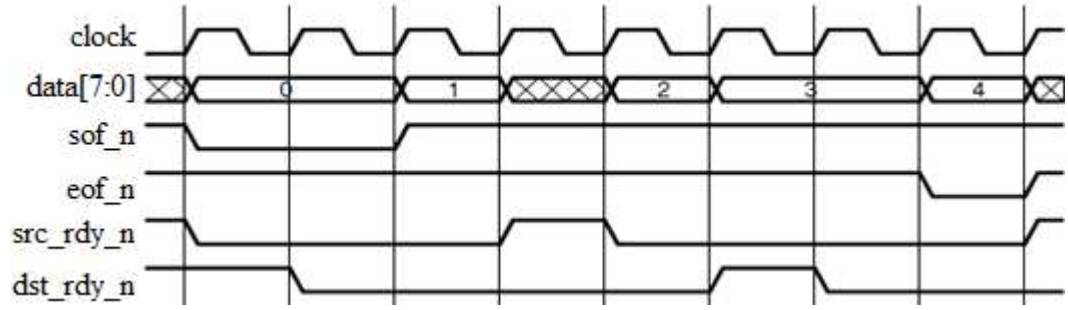
Şekil 3.20 Ethernet haberleşmesinin HDL örnek blok diyagramı [48]

3.5.2.2 Yerel bağlantı arayüzü (Locallink interface)

LocalLink Interface'de dört adet düşen kenarda aktif olan sof_n (start of frame/çerçevenin başlangıcı), eof_n (çerçevenin sonlandırıcısı), src_rdy_n (source ready/kaynak hazır) ve dst_rdy_n (destination ready/iletim tamamlandı) sayısal işaretleriyle kaynaktan alıcıya bilgi akışı kontrol edilir. Yalnızca bu işaretlerin sürülmesiyle veri akışı alıcı tarafa doğru gerçekleştirilebilir. Bireysel veri paket demetleri sof-n ve eof-n sinyalleri ile belirlenir. Şekil 3.21 ve şekil 3.22'de LocalLink Interface ile veri akışının bu dört adet işaret yardımı ile nasıl sağlandığı belirtilmiştir.



Şekil 3.21 8 baytlık verinin LocalLink Interface ile iletimi [48]



Şekil 3.22 5 baytlık verinin src_rdy_n ve dst_rdy_n sinyalleri ile LocalLink Interface üzerinde alıcı tarafına iletimi [48]

3.6 VHDL (Donanım Tasarlama Dili)

VHDL bir donanım tanımlama dilidir. Bir elektronik devrenin ya da sistemin davranışını, hangi fiziksel devrenin ya da sistemin sonradan elde edilebileceğini açıklar.

3.6.1 VHDL tarihçesi

VHDL Very High Speed Integrated Circuits Hardware Description Language/Çok Yüksek Hızlı Entegre Devreler Donanım Tanımlama Dili (VHSIC HDL)'nden gelmektedir. 1980'lerde United States Department of Defense tarafından VHDL'in

ortaya çıkması için finanse edilmiş bir girişimdir. İlk versiyonu VHDL 87 bir sonraki versiyonu ise VHDL 93 adını almıştır. VHDL Instutute of Electrical and Electronics Engineers tarafından IEEE 1076 standartlarına uygunlaştırılmış orjinal ve ilk donanım tanımlama dilidir. IEEE 1164 standardı da çok-değerli mantık sistemleri için çıkarılmıştır.

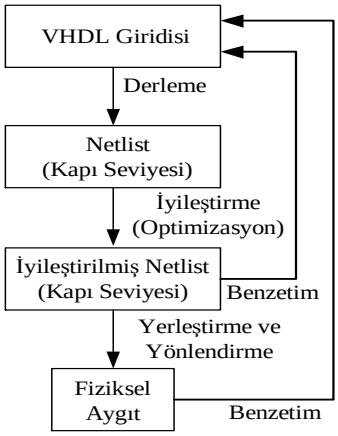
VHDL, devre benzetimine olduğu kadar devre sentezine de yönelik bir dildir. Ancak, VHDL tam anlamıyla benzetim yapılabilirliği olmasına rağmen tüm yapıların sentezlenebilirliği olanaklı değildir.

VHDL kullanmanın temelinde VHDL'in bir standart, teknoloji/satıcı ilişkisinden bağımsız, değişken ve yeniden kullanılabilir olduğu yatmaktadır. VHDL'in iki adet uygulama alanı vardır ve bunlar FPGA ve ASIC'dir. Öncelikle VHDL kodu yazılır ve yazılan VHDL kodu programlanabilen elemanlarda bir devre oluşturmak için ya da ASIC yongaların oluşturulması ve üretimi esnasında kullanılabilir. Günümüzde, çoğu karmaşık yongaların tasarımında bu tür bir yaklaşım kullanılmaktadır.

VHDL durumları diğer bilgisayar programlarının aksine paralel bir yapıya sahiptir. VHDL'de yalnızca işlem (PROCESS), fonksiyon (FUNCTION) ve yöntem (PROCEDURE) yapılarının içindeki kodlar seridirler.

3.6.2 Tasarım akış diyagramı

VHDL'in en büyük yararı, programlanabilir bir aygıt veya ASIC içinde bir devrenin ya da sistemin sentezlenmesine olanak sağlamasıdır. Bir proje oluşturulurken izlenen adımlar Şekil 3.23'de gösterilmektedir. Tasarıma ilk olarak VHDL komutları yazılarak başlanır ve bu komutlar .vhd uzantılı ve giriş (ENTITY) ismi ile aynı olacak şekilde bir dosya içine kaydedilir ve böylelikle ilk adım tamamlanır. Tamamlanan bu adım, yazılımsal liste içerisinde kapı seviyesinde kaydedici transfer seviyesinde devreyi tanımlayan, yüksek-seviyeli VHDL dilinin dönüşümüdür. İkinci adım ise hız ve alan için kapı seviyesinde düğüm listesinde çalışan iyileştirmedir. Bu adımda tasarımın benzetimi yapılabilir. Son olarak yerleştirme ve yönlendirme (place and route), PLD/FPGA yongaları için fiziksel katman, ASIC'ler için maskeler üretir.



Şekil 3.23 VHDL ile oluşturulan bir çalışmanın işlem basamakları [48]

3.6.3 Temel VHDL birimleri

VHDL kodu üç temel bölümden oluşur. Bunlar;

- LIBRARY (Kütüphane) Tanımlaması: Tasarımda kullanılacak olan tüm kütüphaneleri içeren birim. Örneğin: ieee, std, work gibi.
- ENTITY (Varlık): Devrenin giriş/çıkışlarının belirtildiği birim.
- ARCHITECTURE (Mimari): Devrenin nasıl davranması gerektiğini tanımlayan VHDL yazılım yapılarını ya da kodlarını içeren birim.

Kütüphane ortak kullanılan kodlar bütünüdür ve kütüphane içinde mevcut olan kısımlar paylaşılabilir ve yeniden kullanılabilir bir yapıya sahiptir.

3.6.3.1 Kütüphane (Library) tanımlaması

Bir kütüphanenin genel yapısı Şekil 3.24'de belirtilmiştir. Kütüphane oluşturulurken uygulanan yazılımsal tasarım kodları paketler (PACKAGES) içinde, fonksiyon (FUNCTIONS), yöntem (PROCEDURE) ve bileşen (COMPONENTS) yapılarında hedef kütüphane içinde derlenir.

Kütüphane tanımlaması oluşturulan proje içinde iki satırdan oluşan çağırma işlemi ile gerçeklenir. Bunlardan ilki kütüphanenin adı ve diğeri ise bu kütüphanenin kullanıma açılmasına yönelik tanımlama işlemidir.

Tasarımlarda genellikle farklı kütüphanelerden aşağıda belirtilen 3 adet pakete ihtiyaç duyulur.[49]

- ieee.std_logic_1164 (ieee kütüphanesinden)

- standard (std kütüphanesinden)
- work (work (tasarımcının kendi oluşturduğu kütüphane) kütüphanesinden)

En çok kullanılan ieee kütüphanesidir ve bu kütüphane atimetik ve mantık işlemlerinde kullanılan özel paketler içermektedir. Bunlar,

- std_logic_1164: 8 seviyeli mantık (STD_LOGIC) ve 9 seviyeli mantık (STD_ULOGIC) uygulamalarında kullanılır.
- std_logic_arith: İşaretli (SIGNED) ve işaretsiz (UNSIGNED) sayı sistemlerinde aritmetik ve karşılaştırma işlemlerinde kullanılır.
- std_logic_signed: Standart mantık vektörleri (STD_LOGIC_VECTOR) ile çeşitli işlemler yapılabilmesine olanak sağlayan fonksiyonları içeren pakettir. Tüm veriler işaretli (SIGNED) olmak zorundadır.
- std_logic_unsigned: Standart mantık vektörleri (STD_LOGIC_VECTOR) ile çeşitli işlemler yapılabilmesine olanak sağlayan fonksiyonları içeren pakettir. Tüm veriler işaretsiz (UNSIGNED) olmak zorundadır.

Şekil 3.24 VHDL Kütüphane yapısı [49]

3.6.3.2 Varlık (Entity)

Bu birimde tasarlanan devrenin tüm giriş ve çıkışları belirtilir ve listelenir. Diğer bir deyişle devrenin yazılımsal olarak giriş/çıkış kanalları açılır.

Bu birimde tanımlanan sinyaller giriş (IN), çıkış (OUT), giriş-çıkış (INOUT) veya tampon (BUFFER) görevini üstlenirler ve bu sinyaller bit (BIT), standart mantık (STD_LOGIC), tamsayı (INTEGER) vb. bir tür olabilir.

3.6.3.3 Mimari (Architecture)

Bu birim tasarlanan devrenin nasıl davranması gerektiğinin tanımlandığı birimdir. Bu birim iki kısımdan oluşur. Birinci kısım sinyallerin ve değişkenlerin tanımlanabildiği kısım, ikinci kısım ise mimarinin başlangıcından sonra sistem davranışının tanımlandığı kısımdır.

Bir tasarımda gerektiği durumlarda birden fazla mimari (ARCHITECTURE) oluşturulabilir.

3.6.4 VHDL veri nesneleri

Aşağıda VHDL tasarımı yapılırken kullanılan ve farklı işlem bölgelerinde çalışma özelliğine sahip veri nesneleri tanımlanmıştır.

3.6.4.1 Sabit (Constant)

Sabitler değerlerini sürekli koruyan veri nesneleridir. Sabitler paket, varlık ve mimari içerisinde tanımlanabilirler ve tanımlandıkları birimlere göre genel yapı içerisinde görev üstlenirler. Hem genel hem de yerel nesnedir.

3.6.4.2 Sinyal (Signal)

Sinyaller devrenin giriş ve çıkış birimlerinin değerlerini taşıyan ve ileten veri nesneleridir ve bu nesneler aynı zamanda kanallar ve varlıklar arasındaki ara bağlantıları da gerçeklerler. Sabitlerin tanımlanabildiği yapıların içerisinde tanımlanabilirler. Değerleri anlık değişmeyen nesnelerdir ve çoklu atamalar olduğu anda beklenmeyen ve hatalı sonuçlar verebilirler. Aynı zamanda böyle bir durumda sentezleme esnasında sıkıntılar yaşanabilir. Genel bir nesnedir.

3.6.4.3 Değişken (Variable)

Sabit ve sinyallerin aksine yalnızca yerel bir veri nesnesi olarak kullanılabilir. Yalnızca süreç (PROCESS), fonksiyon ve yöntem içerisinde seri işleyen komutlarda kullanılabilir ve kullanıldıkları bu üç yapı dışından müdahale edilemezler.

3.6.5 Veri türleri

VHDL donanım tanımlama dilinde kullanılan veri türleri aşağıda belirtilmiştir.

3.6.5.1 Ön tanımlamalı veri türleri

IEEE 1076 ve IEEE 1064 standartlarına uygun olarak geliştirilmiştir. Bu veri türlerinin tanımlamaları bazı paketler ve kütüphaneler içinde bulunabilir. Bu paketlerden ve kütüphanelerden daha önce bahsedilmiştir. Bütün ön tanımlamalı veri türleri aşağıda belirtilmiştir.

- BIT: Mantık 0 ya da mantık 1 olarak iki seviyeli bir veri türüdür.
- STD_LOGIC ve STD_LOGIC_VECTOR: Standart mantık ve standart mantık vektörü 8 seviyeli mantık sisteminden oluşan bir veri türüdür.
  ‘X’: Bilinmeyen,
  ‘0’: Düşük seviye,
  ‘1’: Yüksek seviye,
  ‘Z’: Yüksek empedans,
  ‘W’: Zayıf bilinmeyen,
  ‘L’: Zayıf düşük,
  ‘H’: Zayıf yüksek,
  ‘-’: Önemsiz.
  Bunlardan yalnızca ‘0’, ‘1’ ve ‘Z’ sentezlenebilir seviyelerdir.
- STD_ULOGIC ve STD_ULOGIC_VECTOR: 9 seviyeli mantık sisteminden oluşan bir veri türüdür. Standart mantık ile standart mantık vektörlerine ilave olarak ‘U’ (Çözümlenemeyen) mantık değerini de içermektedir.
- BOOLEAN: Bu veri türü yalnızca doğru ve yanlış değerlerini kapsayan bir veri türüdür.
- INTEGER: 32 bitlik tamsayılardır.
- NATURAL: 32 bitlik doğal sayılardır.
- REAL: -1E38 ile +1E38 arasındaki değerlerdir ve sentezlenemezler.
- Physical Literals (Fiziksel Türler): Zaman gibi fiziksel nicelikleri belirtmek için kullanılır.
- Character Literals (Karakter Türleri): ASCII kod sistemine göre tekli ya da dizi şeklindeki karakterleri belirleyen türdür.

- SIGNED ve UNSIGNED: İşaretli ve işaretsiz tamsayılardır ve aritmetik işlemlerin gerçeklenmesi sırasında kullanılır.

3.6.5.2 Tasarımcı tarafından tanımlanan veri türleri

Tasarımcı tarafından iki tip veri türü tanımlanabilir. Bunlar tamsayı şeklinde bir aralık olabilir ya da liste (enumerated) şeklinde çekip kullanılabilen bir dizi şeklinde olabilir.

3.6.5.3 Alt tipler (Subtypes)

Alt tipler veri türlerinin bir alt türüdür ve belli bir aralıkta sınırlandırılmış şeklidir.

3.6.5.4 Diziler (Arrays)

Aynı veri türlerini kullanarak oluşturulan veri bütünüdür. Bir boyutlu ve iki boyutlu olabilirler. Ön tanımlamalı veri türleri skaler ya da vektörel diziler şeklinde belirtilirler. Skaler dizi şeklinde tanımlanan veri türleri, BIT, STD_LOGIC, STD_ULOGIC VE BOOLEANdir. Vektörel şeklinde belirtilen veri türleri ise, BIT_VECTOR, STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, INTEGER, SIGNED VE UNSIGNED'dir.

3.6.6 Sistem tasarımı

Bir sistem tasarlanırken kütüphaneleri oluşturmak için paketler (packages), bileşenler (components), fonksiyonlar (functions) ve yöntemler (procedures) kullanılır. Bu tür bir kullanım tasarımda karmaşıklığı önler, kodların tekrar kullanılabilmesini imkan tanıyarak esneklik sağlar.

3.6.6.1 Paket (Package)

Paketler, bileşenler, fonksiyonlar, yöntemler, veri tipleri ve sabitlerden oluşurlar. Paketler iki kısımdan oluşur. Birinci kısım kullanılan paketin kendisidir ve kullanılan sabitler, değişkenler, tipler tanımlanır. İkinci kısımda, eğer fonksiyon, bileşen veya yöntem kullanıldı ise kullanılan fonksiyon, bileşen ve yöntemin tanımlandığı kısımdır.

3.6.6.2 Bileşen (Component)

Bileşenler, kütüphanesi, varlık birimi ve mimarisi tanımlanarak oluşturulan tasarım yapılarıdır. Aşamalı (hierarchical) bir tasarım oluşturmak için kullanılır ve yeniden

çağrılabilme özelliklerinden dolayı tasarımda esneklik ve sadelik sağlarlar. Bileşenler kullanılmadan önce tanımlanması gerekir.

3.6.6.3 Fonksiyon (Function)

Karşılaşılan bazı problemlerin çözümü ve bu problemlere cevap veren bir görev oluşturmak için fonksiyon yapıları kullanılır. Fonksiyon yapıları seri şekilde işlem yapma kabiliyetine sahiptir ve seri işlem yapmaya elverişli komutlar kullanılabilir. Fonksiyon yapıları iki kısımdan oluşur. Bunlardan ilki fonksiyonun kendisinin tanımlandığı ve seri komutların işlendiği kısımdır. Diğeri ise fonksiyonun ana program veya paket içinde çağrılmasıdır. Fonksiyonun giriş değerleri sabit veya sinyal olabilir. Fonksiyonlar tek bir değeri sonuç olarak döndürebilirler.

3.6.6.4 Yöntem (Procedure)

Yöntemler fonksiyonlar ile aynı yapıya ve aynı kısımlara sahiptirler. Fonksiyonlardan tek farkı, birden fazla değeri sonuç olarak döndürebilirler. Bu demektir ki sistemin cevabı birden fazla niceliği kontrol edebilir kapasitedir. Bu yapıda giriş/çıkış değerleri sinyal, değişken ya da sabit olabilir.

BÖLÜM IV

BENZETİM VE GERÇEK–ZAMANLI DENEY ÇALIŞMALARI

Hız-algılayıcısız doğrudan moment kontrollü ASM sürücü sistemleri akı ve rotor açısal hızı bilgilerine ihtiyaç duyar. Ancak bu bilgilerin ölçülerek belirlenmesindeki fiziksel güçlükler (hava aralığında akı vektörünün ölçülmesini sağlayan Hall algılayıcılar, rotor hızı bilgisinin elde edilmesinde kullanılan tako jeneratör, ya da artımsal kodlayıcılar gibi algılayıcılar özel bir fiziksel müdahale gerektirmektedir), ekonomik nedenler (algılayıcıların hava aralığına yerleştirilmesi için yapılan özel imalatın getirmiş olduğu yüksek maliyet, algılayıcıların kendi maliyetleri vb) ve bu algılayıcılardan alınan ölçme işaretin, ASM'nin çalışma ortamından (sıcaklık, nem vb. nedenler) etkilenmesi sonucu hatalı sonuç verebilmesi gibi nedenler gözlemleyici ya da kestirici tasarımını zorunlu hale getirmektedir.

Bu tez çalışmasında, gerçeklenmiş olan GKF algoritması hem benzetimlerle hem de gerçek-zamanlı deneysel çalışmalar ile doğrulanmıştır. Aşağıda sırasıyla benzetim çalışması ve gerçek-zamanlı deneysel çalışmalar açıklanmıştır:

4.1 ASM Modelinin Gerçek-Zamanlı Olarak Doğrulanması

Bu kısımda Eşitlik 2.44'te belirtilen 6.dereceden ASM modeli üzerinde, yük momenti durumu çıkarılıp 5.dereceye indirgenen [$i_{s\alpha}$, $i_{s\beta}$, $\varphi_{r\alpha}$, $\varphi_{r\beta}$, ω_m] değişkenli ASM modeli öncelikle FPGA'da gerçeklenmiştir. Ayrıca, ölçülen yük momenti üzerinden ASM modelinin doğrulanması için daha önce [32]'de tasarlanan açık-çevrimli yük momenti kestiricisi de kullanılmıştır (Eşitlik 4.1).

$$t_{L_{k+1}} = \frac{3}{2} P_p \frac{L_m}{L_r'} (\varphi_{r\alpha_{k+1}} i_{s\beta_{k+1}} - \varphi_{r\beta_{k+1}} i_{s\alpha_{k+1}}) - J_T (\frac{\omega_{m_{k+1}} - \omega_{m_k}}{T}) - B_L \omega_{m_{k+1}} \quad (4.1)$$

Burada, J_T, toplam eylemsizlik momentidir. B_L,viskoz sürtünme katsayısıdır.

Bu uygulamaya ilişkin kullanılan gerçek-zamanlı deney düzeneği ve tasarlanan ASM modelinin VHDL akış diyagramı Şekil 4.1'de gösterilmiştir. Bu tez kapsamında gerçek-zamanlı deney çalışmaları için kullanılan deney düzeneği Fotoğraf 4.1'de ve deney düzeneğine ilişkin bağlantı şeması Şekil 4.1'de görülmektedir.

Bu deney düzeneğinde;

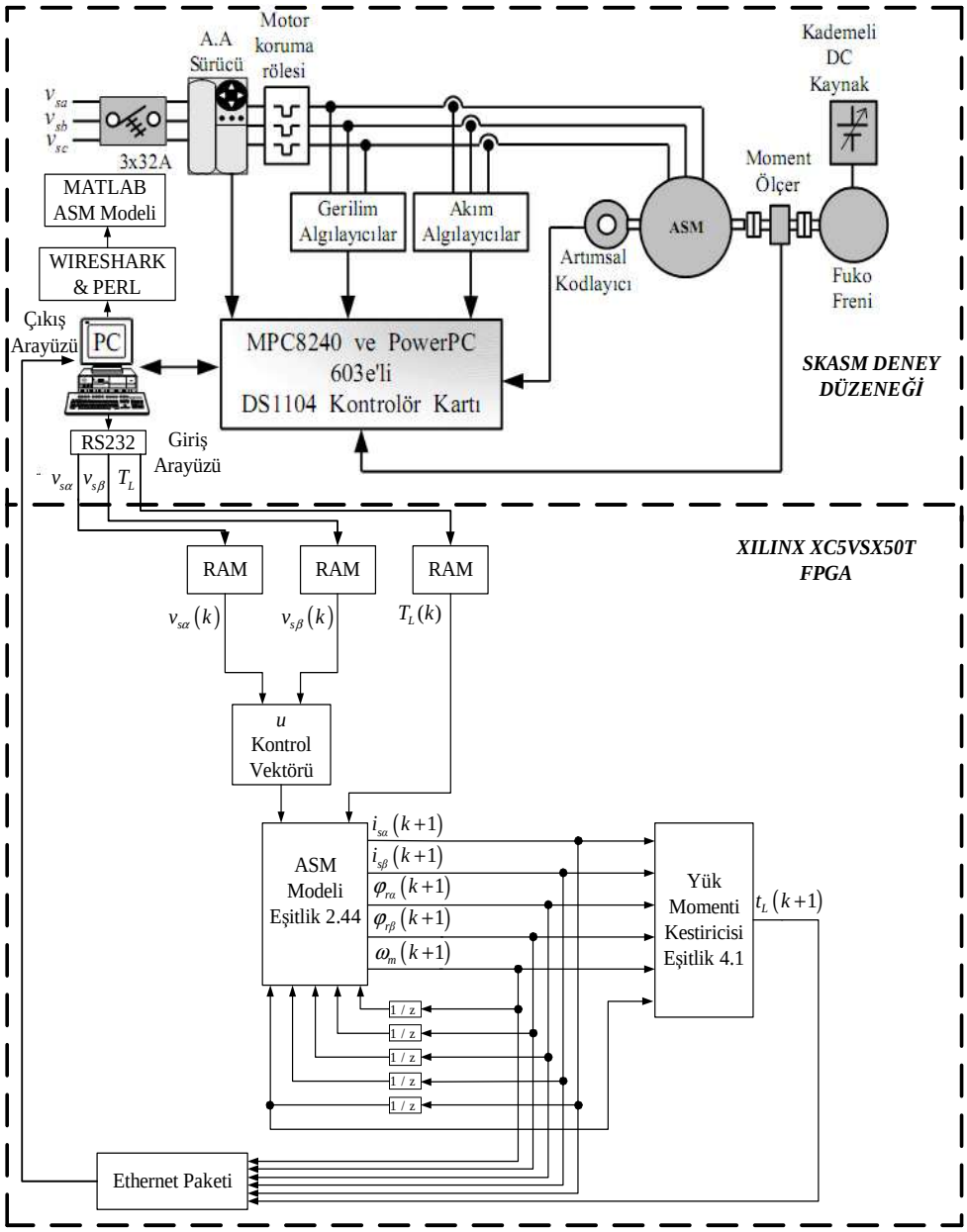
- Parametreleri Çizelge 4.1'de verilen ABANA'nın sincap kafesli ASM'si,
- Rotor açısal hızı ve yük momentine ilişkin gerçek değerleri elde etmek için ETH'ın DBRK-50 serisi 50 [N.m]'lik moment ölçeri ve Heidenhain'nın ERN120 serisi 5000 dilimlik artımsal kodlayıcısı,
- Sincak kafesli ASM'ye sinüzoidal yerine vektör kontrolü altında elde edilmiş darbe genişlik modülasyonlu (DGM'li) besleme gerilimi uygulamak için Control Techniques'in 7.5 [kW]'lık Unidrive'ı,
- Üç faz gerilim ve akımlarını ölçmek için sırasıyla LEM'in LV100-400 ve LA55-P/SP1 algılayıcıları,
- Farklı yük koşullarını oluşturmak için, FEMSAN'nın 30 [N.m]'lik fuko freni,
- Verileri alabilmek için ControlDesk arayüz yazılımlı DS1104 kontrolör kartı ve kişisel bilgisayar,
- Xilinx Virtex 5 ML506 FPGA geliştirme kiti,

kullanılmıştır.

Fotoğraf 4.1 Gerçek-zamanlı çalışmada ASM verilerinin elde edildiği deney düzeneği

Çizelge 4.1 Gerçek-zamanlı çalışmalarda kullanılan ASM parametreleri

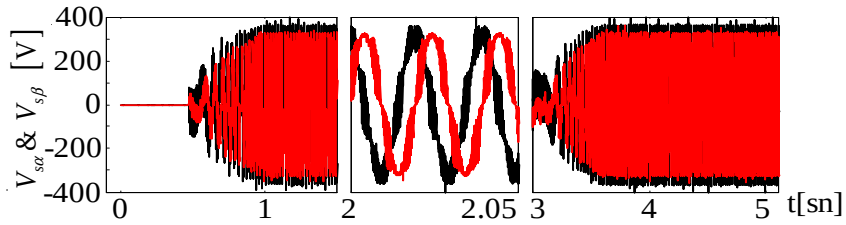
$P\ [kW]$	$f\ [Hz]$	$J_T[kg.m^2]$	$B_L[Nm/(rad/sn)]$	P_p	$V\ [V]$	$I[A]$
2.2	50	0.055	0.0019	3	380	5.5
$R_s[\Omega]$	$R_r'[\Omega]$	$L_s[H]$	$L_r'[H]$	$L_m[H]$	$n_m[rpm]$	$t_L[N.m]$
3	2.53	0.1466	0.524	0.135	1000	20



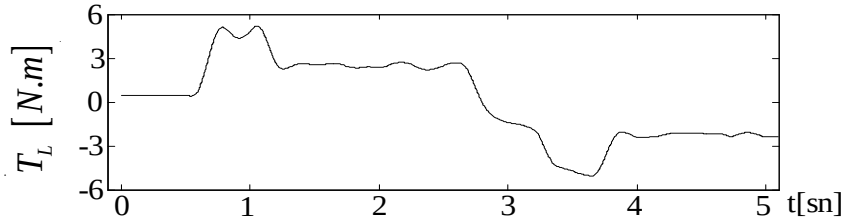
Şekil 4.1 ASM modelinin yapısı ve VHDL akış diyagramı

Şekil 4.1'de görüldüğü gibi, deney düzeneğinden elde edilen gerçek-zamanlı stator gerilimleri ($v_{s\alpha}(k)$ ve $v_{s\beta}(k)$) ve yük momenti bilgisi ($T_L(k)$) sırası ile Şekil 4.2 ve 4.3'te sunulmuştur. Bu bilgiler ASM modeline giriş olarak uygulanmaktadır (Şekil 4.1). Gerçek-zamanlı ASM deney düzeneğinden elde edilen faz gerilimleri ve yük momenti bilgisi öncelikle FPGA'nın yapısında bulunan dahili hafıza birimlerine (RAM) kaydedilmiştir. FPGA hafıza birimleri deney düzeneğinden elde edilen tüm verilerin kaydedilmesinde yetersiz kalmakta ve bu sebeple tüm bu veriler FPGA'nın sahip olduğu hafıza kapasitesine bağlı olarak bölünüp kısım kısım hafıza birimlerine kaydedilir. Böylelikle FPGA'nın hafıza yetersizliğinden kaynaklanan algoritmanın

ardışıl olarak tekrarlanma sınırlılığı ortadan kaldırılarak daha uzun çevrimli çalışmalar sağlanabilmiştir. Bu amaçla da gerçek-zamanlı verilerin FPGA hafıza birimlerine kaydedilebilmesi için bilgisayar ile FPGA arasındaki seri haberleşmeyi sağlayan ve C# (C SHARP) yazılımı ile gerçekleştirilen bir veri gönderme giriş arayüzü (Şekil 4.4) tasarlanmıştır. FPGA üzerinde gerçeklenen ASM modeline ilişkin stator akımı, rotor akısı, rotor hızı ve yük momenti bilgileri, FPGA'da saf Ethernet paketleri haline çevrilerek ethernet haberleşmesi protokolü aracılığıyla anlık olarak bilgisayara aktarılmıştır. FPGA tarafından gönderilen bu bilgiler, Microsoft firması tarafından geliştirilen WIRESHARK (Şekil 4.5) ethernet paketi toplama çıkış arayüzü aracılığıyla *.txt* formatında kaydedilmiştir. Kaydedilen ASM modeli sonuçlarına ait bilgiler PERL yazılım dili kullanılarak dizi şeklinde stator akımı, rotor akısı, rotor hızı ve yük momenti olarak ayrıştırılıp tekrar *.txt* formatında kaydedilerek MATLAB *m-file*'da yazılan program aracılıyla grafiksel olarak gözlemlenebilmiştir.

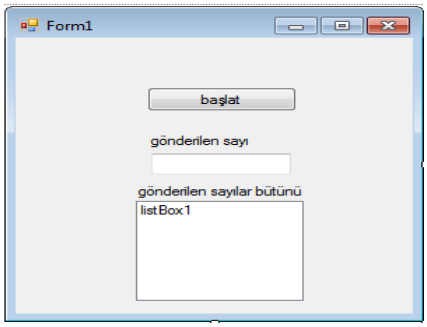


Şekil 4.2 Gerçek-zamanlı olarak ölçülen stator faz gerilimleri

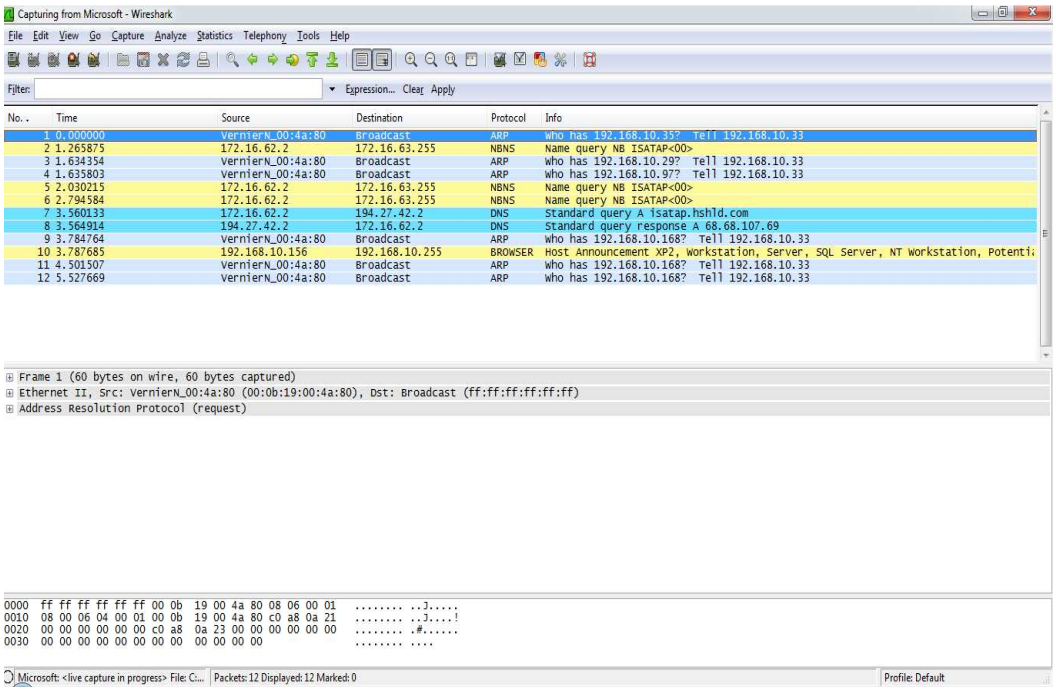


Şekil 4.3 Gerçek-zamanlı olarak ölçülen yük momenti

Tez çalışmasının bu kısmında, FPGA'da gerçeklenen ASM modelini doğrulamak için aynı ASM modeli MATLAB *m-file*'da da gerçeklenmiştir. FPGA'da gerçeklenen ASM modelinde olduğu gibi MATLAB ortamından da ASM modeline gerçek-zamanlı olarak deney düzeneğinden minimum 50 [μs] örnekleme zamanıyla ölçülebilen stator gerilimleri ve yük momenti bilgisi giriş olarak verilmiştir. Modele ilişkin giriş bilgileri ile modelin uyumlu çalışabilmesi için modelin örnekleme zamanı 50 [μs] alınmıştır.

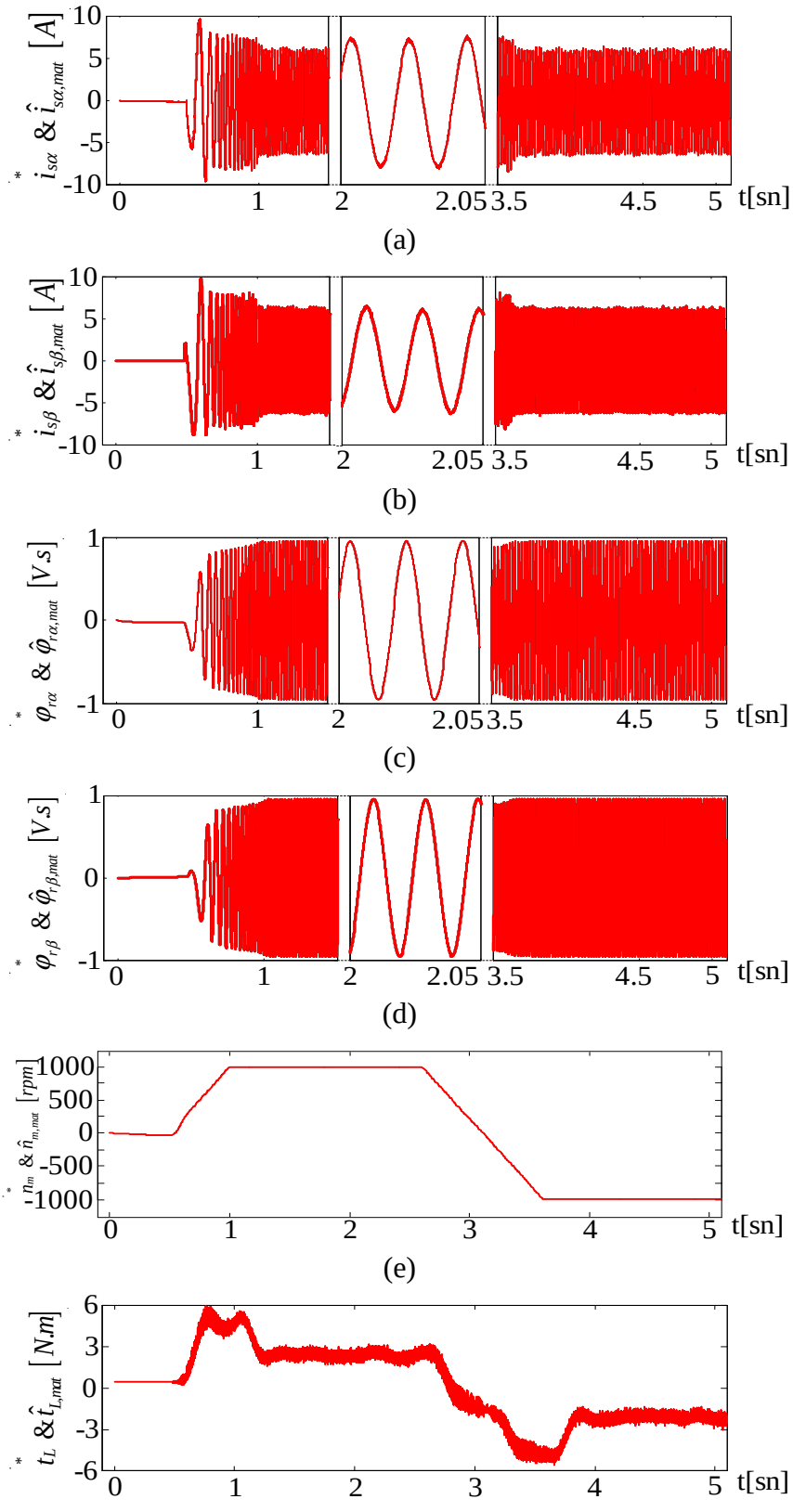


Şekil 4.4 C# ile tasarlanan seri port ile veri gönderme arayüzü

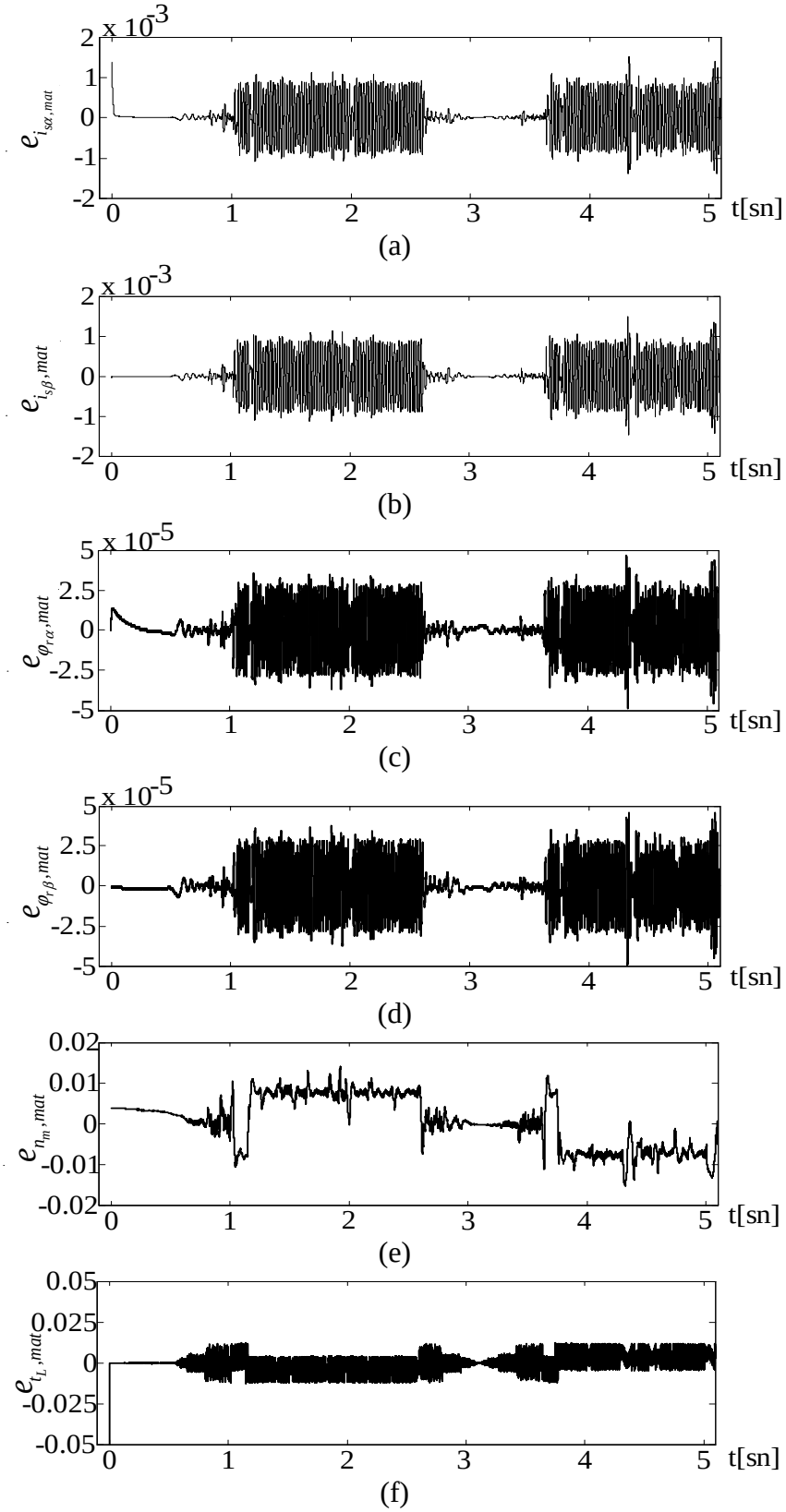


Şekil 4.5 Microsoft Wireshark ethernet paketi toplama arayüz programı

Bu kısımda hem FPGA hem de MATLAB *m-file* ortamlarında elde edilen sonuçlar aynı grafik üzerinde birlikte çizdirilerek Şekil 4.6 ve FPGA'da oluşturulan ASM modeline ve MATLAB *m*-file'da oluşturulan ASM modeline ilişkin sonuçlar arasındaki farklar Şekil 4.7'de sunulmuştur. ASM modelinin doğrulanması amacıyla 0.6 [s]'den itibaren ASM sıfır hızdan anma hızına doğrusal olarak hızlandırılmış ve motorun hızlanması ile üstel olarak değişen yaklaşık 2.5 [$N.m$] değerindeki yük ile yüklenmiştir. ASM anma hızında iken yaklaşık 2.7. [s]'de terslendirilmiştir. Bu şekillerde "*" modele ilişkin FPGA sonuçlarını, "*mat*" modele ilişkin MATLAB *m-file* sonuçlarını ve $e_{[.]mat}$, MATLAB ile FPGA sonuçları arasındaki farkı göstermektedir.

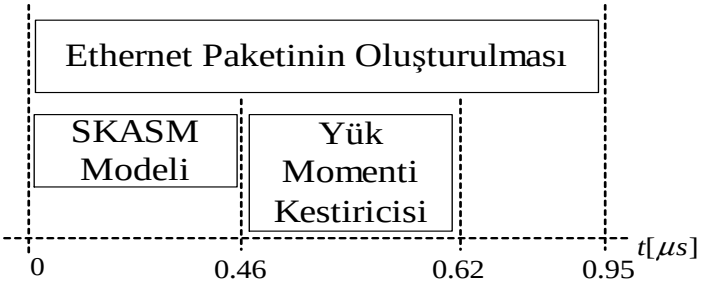


Şekil 4.6 FPGA'da ve Matlab *m-file*'de gerçeklenen ASM modelinin a) Stator akımı α bileşenleri ($i^*_{s\alpha}$ ve $\hat{i}_{s\alpha,mat}$). b) Stator akımı β bileşenleri ($i^*_{s\beta}$ve $\hat{i}_{s\beta,mat}$). c) Rotor akısı α bileşenleri ($\varphi^*_{r\alpha}$ve $\hat{\varphi}_{r\alpha,mat}$). d) Rotor akısı β bileşenleri ($\varphi^*_{r\beta}$ve $\hat{\varphi}_{r\beta,mat}$). e) Mekanik hızlar ($n^*_m$ ve $\hat{n}_{m,mat}$). f) Yük momentleri (t^*_L ve $\hat{t}_{L,mat}$).



Şekil 4.7 FPGA ile Matlab sonuçlarının karşılaştırması. a) Stator akımı α bileşenleri arasındaki farklar ($e_{i_{s\alpha,mat}}$). b) Stator akımı β bileşenleri arasındaki farklar ($e_{i_{s\beta,mat}}$). c) Rotor akısı α bileşenleri arasındaki farklar ($e_{\varphi_{r\alpha,mat}}$). d) Rotor akısı β bileşenleri arasındaki farklar ($e_{\varphi_{r\beta,mat}}$). e) Mekanik hızlar arasındaki farklar ($e_{n_{m,mat}}$). f) Yük momentleri arasındaki farklar ($e_{t_{L,mat}}$).

ASM modelinin matematiksel işlemlerinde, MATLAB doğası gereği 64 bit kayan noktalı sayı (double precision floating-point) kullanırken, FPGA'da ise 32 bit kayan noktalı sayı (single precision floating-point) kullanılmıştır. Bu sebeple Şekil 4.7'de gözlemlenen farklar meydana gelmiştir. Şekil 4.8'de oluşturulan algoritmaların FPGA üzerinde gerçeklenme sıraları ve her bloğun ne kadar sürede gerçeklendiği, Çizelge 4.2'de algoritmada kullanılan mantık kaynakların sayıları, Çizelge 4.3'te aritmetik işlem bloklarının kullandığı mantık kaynaklarının sayısı verilmiştir.



Şekil 4.8 FPGA'da gerçeklenen ASM modelinin zaman diyagramı

Çizelge 4.2 ASM modeli oluşturulurken kullanılan kaynak sayıları

Slice Register	5683
Flip Flops	5669
LUTs	15572
DSP48Es	14

Çizelge 4.3 Aritmetik işlem bloklarının kullandığı kaynak sayıları

Operator	**LUTs**	**Flip-Flops**
Çarpma	242	53
Bölme	450	751
Toplama	388	76
Çıkarma	388	76

FPGA'da gerçeklenen ASM modeli, kullanılan Ethernet haberleşme protokolü ile birlikte bir çevrimini 0.62 [μs]'de tamamlamaktadır. Buradan da anlaşılıyor ki eğer ASM'den örnekleme zamanı daha düşük olacak şekilde veri eldesi sağlanabilirse ASM modeli daha küçük örnekleme zamanı ile kullanılarak gerçeklenebilir.

4.2 GKF Tabanlı Kestirici İçin Benzetim Çalışmaları

Tez çalışmasının bu kısmında,

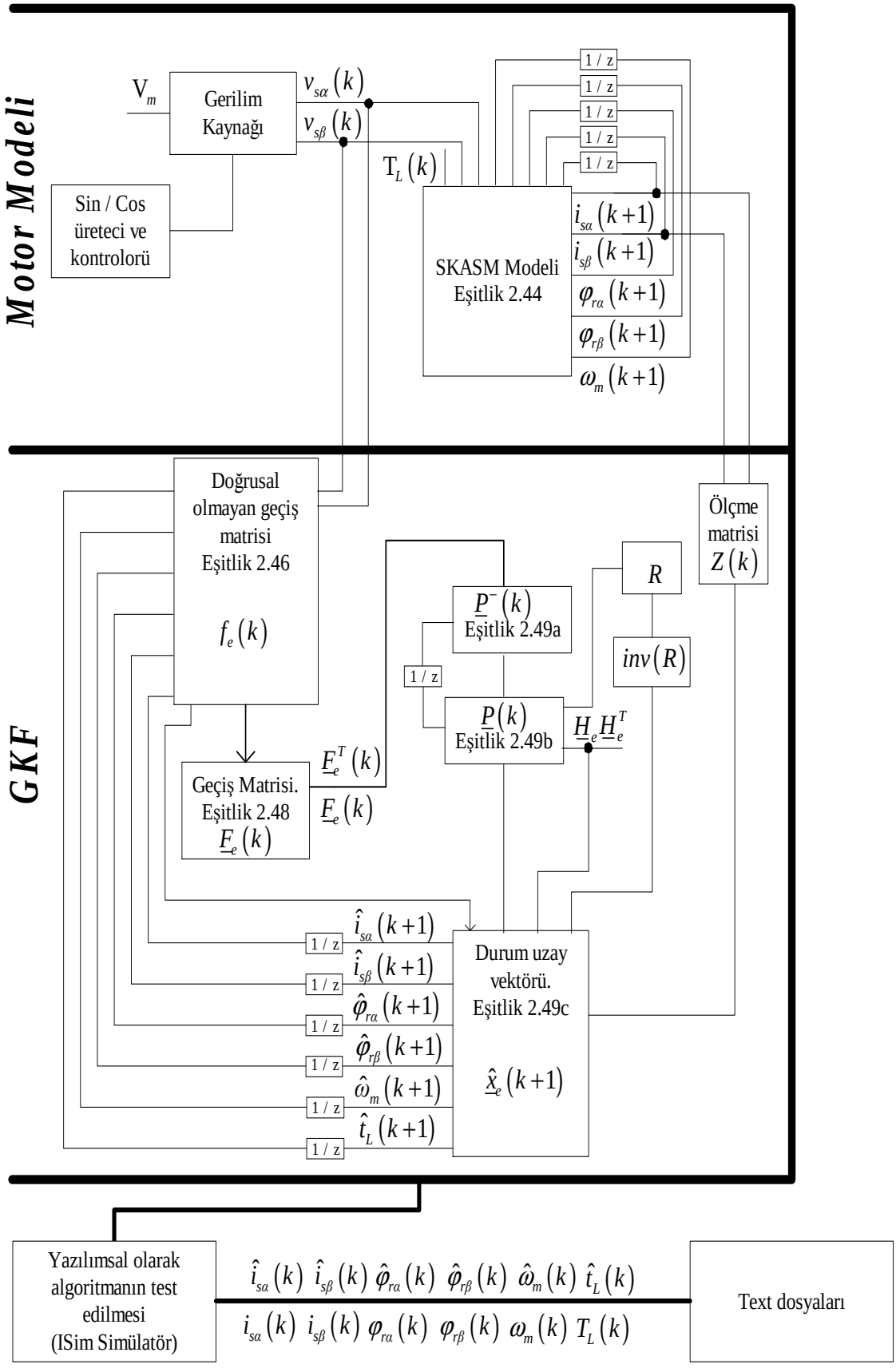
- FPGA simülatörü üzerinde ASM modeli ve GKF birlikte oluşturulmuştur. Hem ASM'ye hem de GKF algoritmasına stator gerilimleri olarak simülatörde oluşturulan gerilim kaynağı ve yük momenti bilgileri girilmiştir. Böylelikle simülatörde oluşturulmuş GKF algoritmasının kestirim başarımı yine simülatörde oluşturulmuş ASM modeli çıkışları kullanılarak doğrulanmıştır.
- FPGA üzerinde ASM modeli ve GKF birlikte oluşturulmuştur. Hem ASM'ye hem de GKF algoritmasına stator gerilimleri olarak FPGA'da oluşturulan gerilim kaynağı ve yük momenti bilgileri girilmiştir. Böylelikle FPGA'da oluşturulmuş GKF algoritmasının kestirim başarımı yine FPGA'da oluşturulmuş ASM modeli çıkışları kullanılarak doğrulanmıştır.

4.2.1 Çalışma I: GKF tabanlı kestirici için yazılımsal benzetim çalışması

Bu kısımda benzetim çalışması için Şekil 4.9'daki GKF ve ASM modeli Xilinx Virtex 5 XC5VSX50T FPGA modülü için VHDL kodu ile oluşturulup FPGA'ya gömülmeden önce Xilinx firmasının geliştirdiği ISE 12.2 VHDL derleyici programının ISim (VHDL/Verilog) adlı simülatöründen yararlanılarak benzetimi yapılmıştır. Diğer bir deyişle bu kısımda Test-Ortamı/Bankası (Test-Bench) uygulaması yapılmıştır. FPGA'ya gömülmek üzere tasarlanan ASM modeli, bir önceki kısımda da gerçeklenen ve Eşitlik 2.44'te belirtilen 6.dereceden ASM modeli üzerinde, yük momenti durumu çıkarılıp 5.dereceye indirgenen [$i_{s\alpha}$, $i_{s\beta}$, $\varphi_{r\alpha}$, $\varphi_{r\beta}$, ω_m] durum değişkenli modeldir. GKF olarak da Eşitlik 2.49 kullanılmıştır.

Çizelge 4.4 Benzetim çalışmasında kullanılan ASM parametreleri

P [kW]	f [Hz]	J_T[$kg.m^2$]	B_L[$Nm/(rad/sn)$]	P_p	V [V]	I[A]
3	50	0.005	0.01	2	380	6.9
R_s[Ω]	R_r'[Ω]	L_s[H]	L_r'[H]	L_m[H]	n_m[rpm]	t_L[$N.m$]
2.283	2.133	0.231	0.231	0.22	1430	20



Şekil 4.9 ASM modeli ve GKF'nin VHDL akış blokları

Şekil 4.9'da öncelikle parametreleri Çizelge 4.4'te verilen ASM için ASM modeli gerçeklenmiştir. ASM modelinde üretilen stator akımları ve simülatörde üretilen gerilim kaynağı stator gerilimleri olarak GKF'ye uygulanmıştır. Gerilim kaynağından önce, VHDL ve FPGA'nın matematiksel işlem kabiliyetinde var olmayan sinüs ve kosinüs fonksiyonları sinüs ve kosinüs üreteci ile her çevrim için hesaplanmıştır. Çalışma zamanına bağlı sinüs ve kosinüs fonksiyonları hesaplanırken periyodikliği tam olarak sağlayabilmek için örnekleme zamanı 20 [μs] olarak alınmıştır. Örnekleme zamanı ve döngü sayısına bağlı olarak elde edilen çalışma süresi referans alınarak, bu süre Taylor serisine açılıp buradan sinüs ve kosinüs matematiksel fonksiyonları elde edilmiştir. Hesaplanan kosinüs ve sinüs değerleri ile stator gerilimleri oluşturularak hem ASM modeline hem de GKF'ye giriş olarak verilmiştir. ASM modeline ilişkin sonuçlar ile GKF kestirim sonuçları *.txt* formatında kaydedilmiştir. Kaydedilen bu sonuçlar MATLAB *m-file*'da yazılan bir program ile grafiksel olarak gözlemlenmiş ve GKF kestirim başarımı ASM modeli çıkışları ile doğrulanmıştır.

GKF'nin gerçeklenmesi için sistem gürültüsü kovaryans matrisi ***Q***, durum hata kovaryans matrisi ***P*** ve ölçme gürültüsü kovaryans matrisi ***R***'nin, ilgili işaretlerin istatistiksel değerleri göz önüne alınarak hesaplanması gerekir. Ancak, uygulamada ***R*** hesaplanarak belirlenebilirken; ***P*** ve ***Q***, kesin pozitifliği sağlamak ve hesap yükünü azaltmak üzere köşegen matris şeklinde tanımlanarak, değerleri istenilen kestirim başarımını elde etmek için ayar-parametreleri gibi düşünülerek deneme-yanılma yöntemiyle belirlenir. Bu düşüncelerle, benzetimlerde kullanılan ***Q***, ***R*** ve ***P*** aşağıdaki gibi belirlenmiştir [50]:

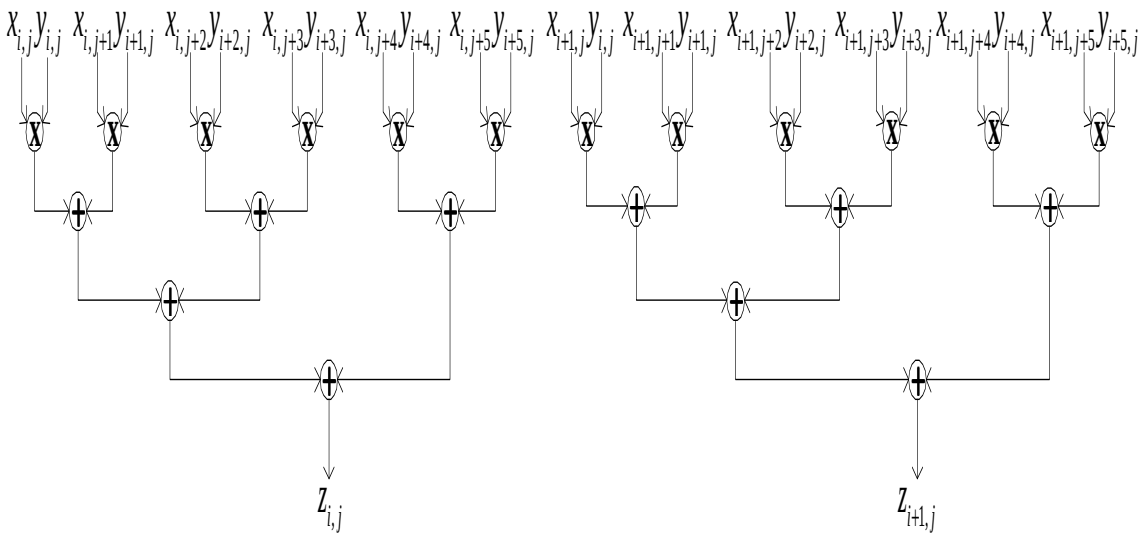
$\boldsymbol{Q}$=diag[10^{-6} (A^2) 10^{-6} (A^2) 10^{-8} (Wb^2) 10^{-8} (Wb^2) 10^{-6} ($(rad/sn)^2$) 10^{-6} ($(N.\mathrm{m})^2$)]

$\boldsymbol{R}$=diag[10^{-6} 10^{-6}]

$\boldsymbol{P}$=diag[10 (A^2) 10 (A^2) 10 (Wb^2) 10 (Wb^2) 10 ($(rad/sn)^2$) 10 ($(N.\mathrm{m})^2$)]

Algoritmanın gerçeklenmesi esnasında karşılaşılan ve FPGA'nın "zaman/alan" kavramını zorlayan bir konu da büyük boyutlu matrislerin çarpımı olmuştur. Bu problemin çözümünde FPGA'nın sahip olduğu mantık kapı kapasitesi sınırını zorlamadan ve minimum sürede bu işlemi gerçekleştirebilen bir durum makinesi (State Machine) önerilmiştir. Yapılan bu işlem, durum içinde paralellik, durumlar arasında ise seri işlem yapma yeteneği sağlamaktadır. Şekil 4.10'da 6x6 matris çarpımının nasıl

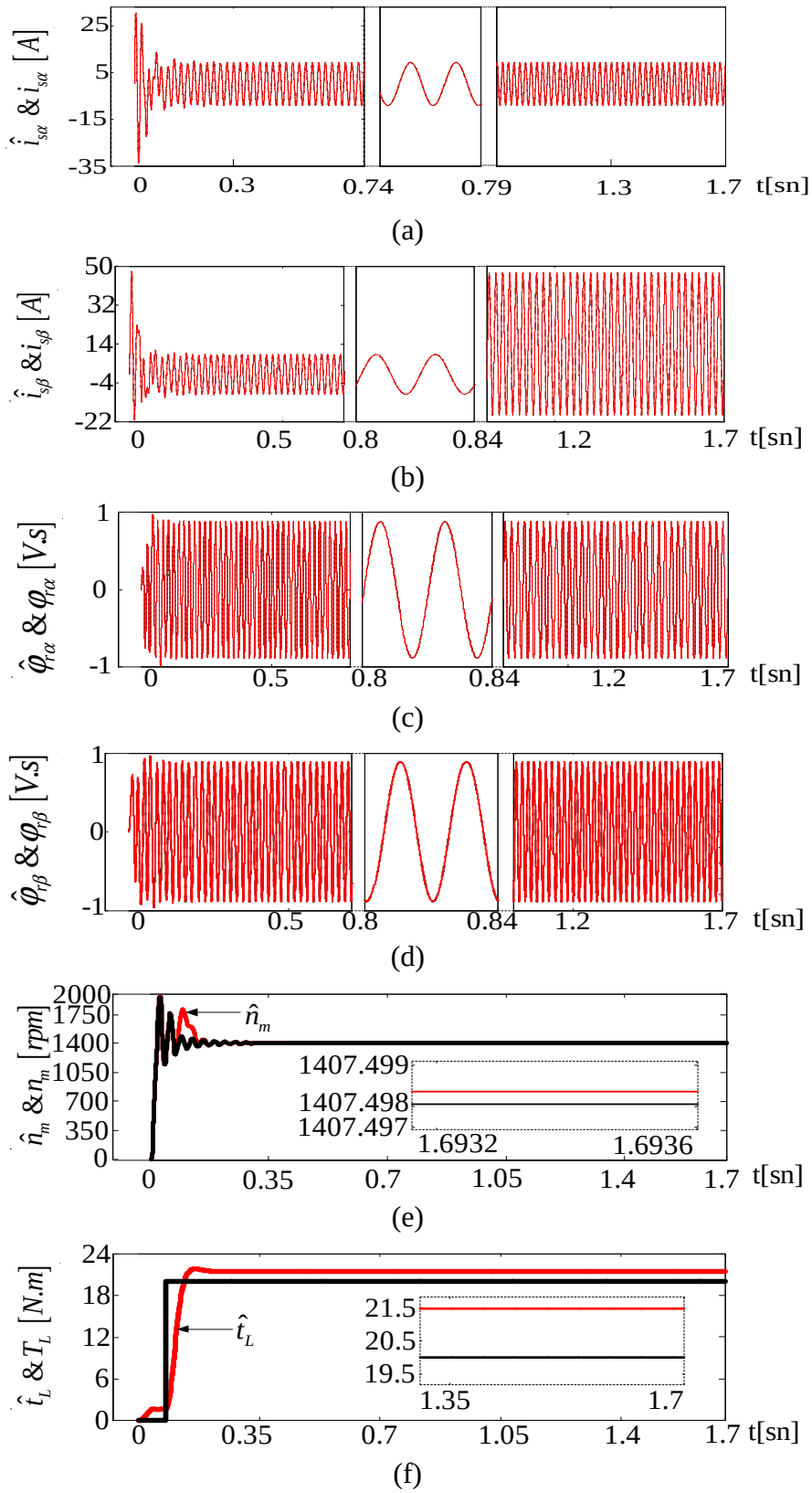
gerçekleştirildiği gösterilmiştir. Bu şekilde *x*, ilk matristir. *y*, ikinci matristir. *z*, sonuç matrisidir. *i*, satır numarasıdır ve *j*, sütun numarasını belirtmektedir. Daha küçük boyutlu matris çarpımları da bu mantıkla gerçekleştirilmiştir.



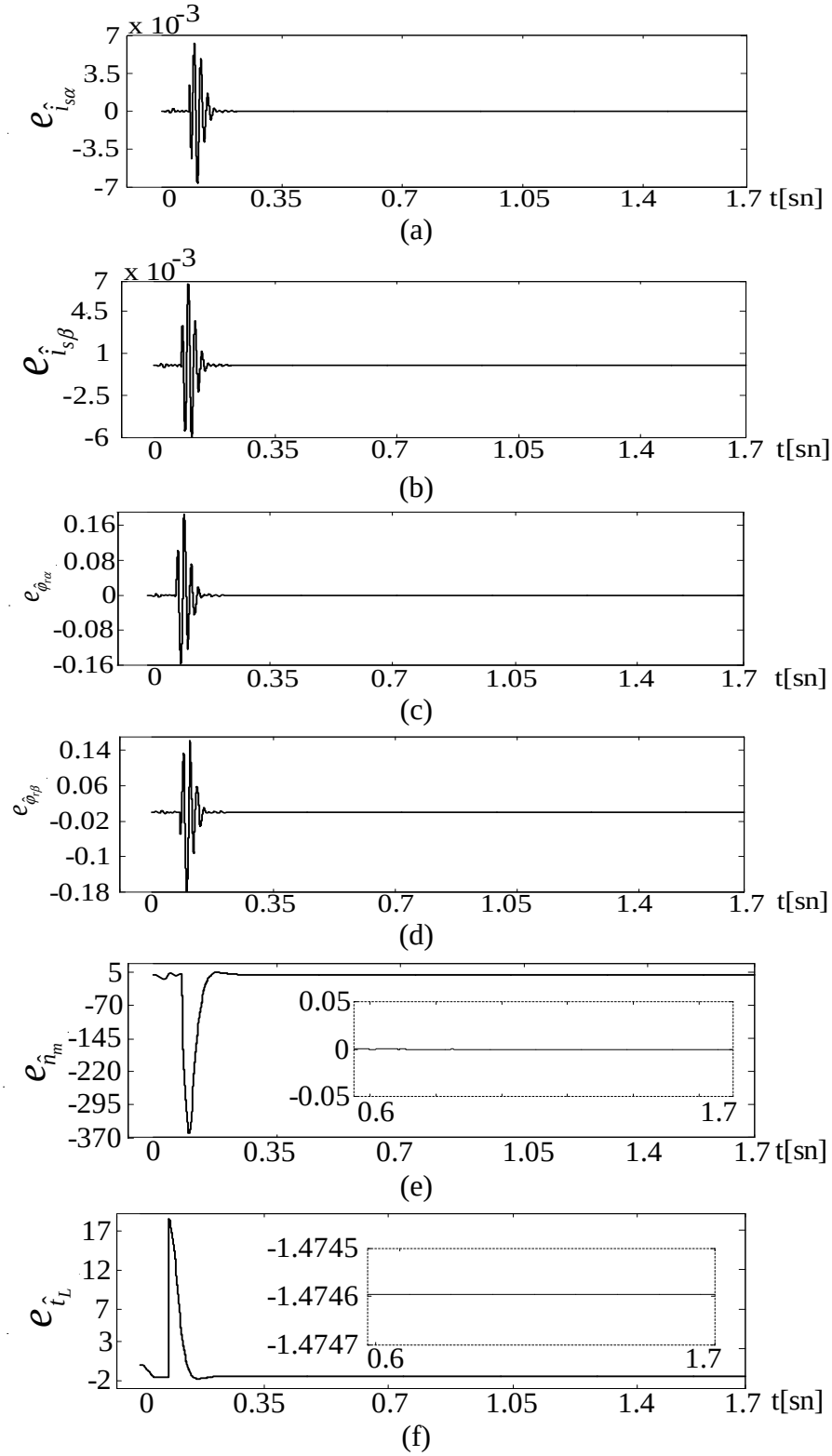
Şekil 4.10 Matris çarpımının veri akış diyagramı

Bu kısımda simülatörde oluşturulan ASM modeline ve GKF algoritmasına ilişkin sonuçlar birlikte Şekil 4.11 ve bu sonuçlar arasındaki farklar Şekil 4.12'de sunulmuştur. GKF'nin kestirim başarımının ASM modeli ile doğrulanması amacıyla motor $0 \leq t < 0.08$ [*s*] aralığında yüksüz, $0.08 \leq t < 1.7$ [*s*] aralığında 20 [*N.m*] yük momenti ile yüklenmiş olup hız ise sıfır hızdan anma hızına ulaşılıncaya kadar artırılmıştır. Şekillerde "^" GKF'nin kestirim sonuçlarını ve $e_{[.]}$, ASM modeline ilişkin sonuçlar ile GKF'nin kestirim sonuçları arasındaki farkı göstermektedir.

Şekil 4.12'den anlaşılacağı üzere GKF'nin kestirim başarımı oldukça yüksektir. Yalnızca momentin yüklendiği durumda hızda bir sapma meydana gelmiş fakat çok kısa bir sürede kestirilen gerçek rotor hızı ASM modeline ilişkin rotor hızını yakalamıştır.



Şekil 4.11 Xilinx ISE12.2 ISim simülatöründe gerçeklenen GKF algoritması ve ASM modelinden elde edilen sonuçlar. a) Stator akımı α bileşenleri ($\hat{i}_{s\alpha}$ ve $i_{s\alpha}$). b) Stator akımı β bileşenleri ($\hat{i}_{s\beta}$ ve $i_{s\beta}$). c) Rotor akısı α bileşenleri ($\hat{\varphi}_{r\alpha}$ ve $\varphi_{r\alpha}$). d) Rotor akısı β bileşenleri ($\hat{\varphi}_{r\beta}$ ve $\varphi_{r\beta}$). e) Mekanik hızlar ($\hat{n}_m$ ve n_m). f) Yük momentleri ($\hat{t}_L$ve T_L).



Şekil 4.12 Xilinx ISE12.2 ISim simülatöründe gerçeklenen ASM modeli ile GKF algoritması sonuçları arasındaki farklar. a) Stator akımı α bileşeni hatası($e_{\hat{i}_{s\alpha}}$). b) Stator akımı β bileşeni hatası ($e_{\hat{i}_{s\beta}}$). c) Rotor akısı α bileşeni hatası ($e_{\hat{\varphi}_{r\alpha}}$). d) Rotor akısı β bileşeni hatası ($e_{\hat{\varphi}_{r\beta}}$). e) Mekanik hız hatası ($e_{\hat{n}_m}$). f) Yük momenti hatası ($e_{\hat{t}_L}$).

Şekil 4.12'de görülen -1.4746 kestirim hatası değildir. Diğer bir ifade ile önerilen kestirim algoritması kestirilen yük momenti $\hat{t}_L$ içerisinde viskoz sürtünme terimini de hesaba katma yeteneğine sahiptir. Bu gerçek aşağıdaki gibi hesaplarla kolay bir şekilde görülebilir.

$$\omega_m(\infty) = \hat{\omega}_m(\infty) + e_{\omega_{m(\infty)}}$$

$$= \frac{2\pi(1407.4275 - 1.5279x10^{-4})}{60} = 147.3854$$

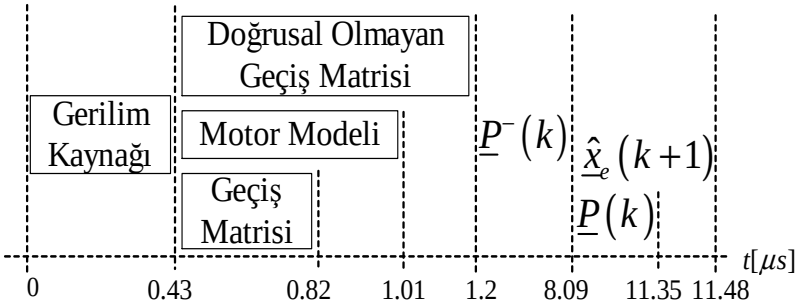
$$e_{t_L} = -\beta_T \omega_m(\infty)$$

$$-1.4746 = -0.01x147.3854$$

$$-1.4746 \cong -1.473854$$

Burada β_T, toplam viskoz sürtünme katsayısıdır.

Şekil 4.13'de oluşturulan algoritmaların FPGA simülatöründe gerçeklenme sıraları ve süreleri, Çizelge 4.5'de oluşturulan algoritmadaki blokların her birinin gerçeklenmesi için ne kadar saat darbesi gerektiği verilmiştir.



Şekil 4.13 FPGA simülatöründe gerçeklenen algoritmalardaki fonksiyon bloklarının gerçeklenme sıraları ve süreleri

Çizelge 4.5 FPGA simülatöründe oluşturulan ASM modeli ve GKF algoritmasındaki fonksiyon bloklarının her birinin gerçeklenmesi için gerekli saat darbesi sayıları

Gerilim Kaynağı	43 clock
Geçiş Matrisi	39 clock
Motor Modeli	58 clock
Doğrusal Olmayan Geçiş Matrisi	58 clock
$\underline{P_k^-}(k)$	689
$\underline{P}(k)$	326
$\hat{\underline{x}}_e(k+1)$	35 clock

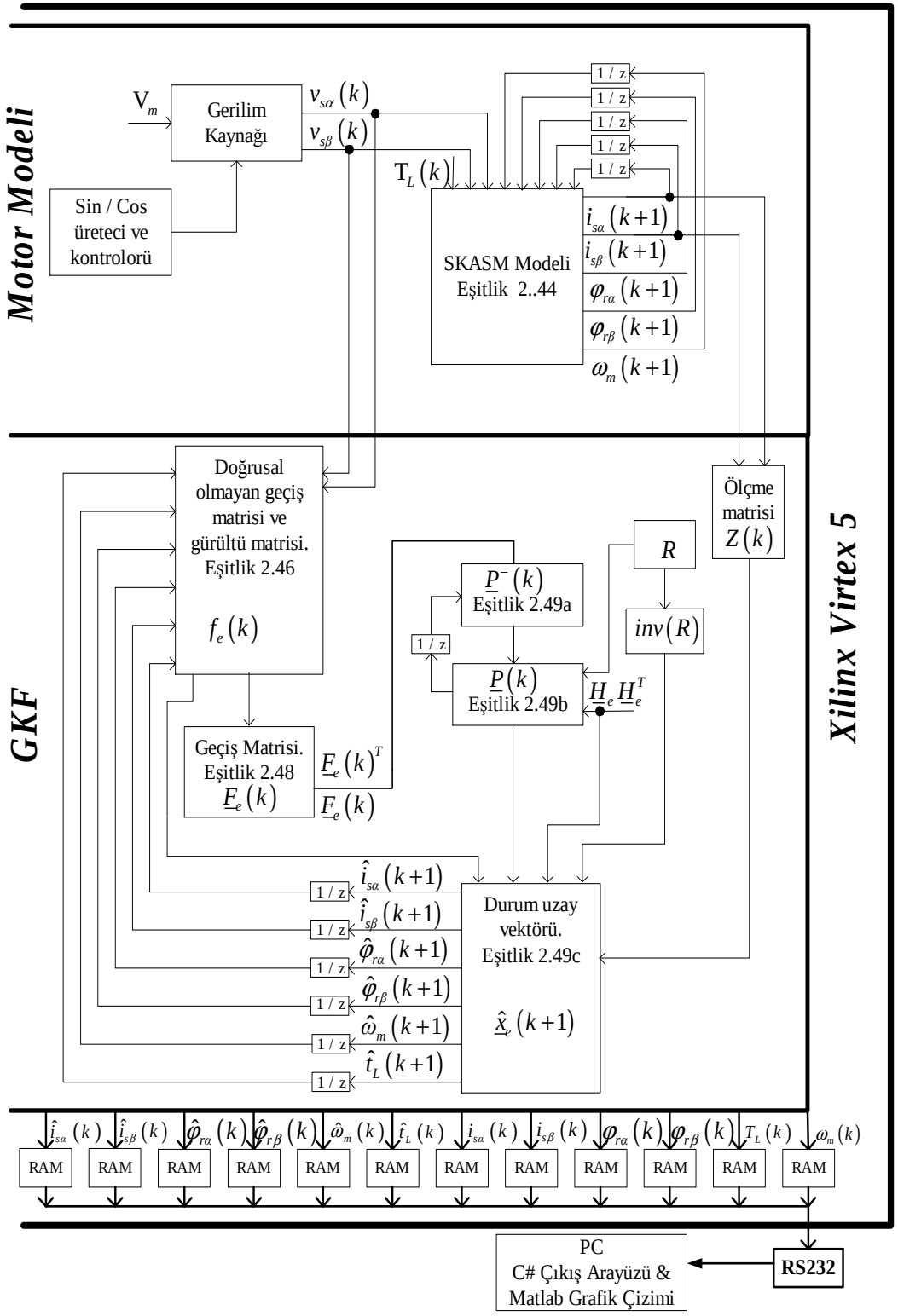
4.2.2 Çalışma II: GKF tabanlı kestirici için donanımsal benzetim çalışması

Tez çalışmasının bu kısmında bir önceki kısımda ISE12.2 ISim Simülatörü kullanılarak yapılan ve GKF'nin ASM modeli ile birlikte doğruluğunun gerçeklendiği benzetim çalışmasından farklı olarak, tasarlanan ASM modeli ve GKF FPGA'ya gömülerek donanımsal (gerçek) benzetim çalışması yapılmıştır.

Şekil 4.14'de bu kısımda gerçeklenen benzetim çalışmasının VHDL akış şeması sunulmuştur. Şekil 4.14 incelendiğinde bir önceki çalışmada belirtilen akış diyagramından tek farkının ASM modeline ilişkin [$i_{s\alpha}$, $i_{s\beta}$, $\varphi_{r\alpha}$, $\varphi_{r\beta}$, ω_m] sonuçlarının, yük momentinin ve GKF'nin kestirdiği [$\hat{i}_{s\alpha}$, $\hat{i}_{s\beta}$, $\hat{\varphi}_{r\alpha}$, $\hat{\varphi}_{r\beta}$, $\hat{\omega}_m$, $\hat{t}_L$] sonuçlarının FPGA'nın dahili hafızalarına kaydedilmesi ve bu hafıza birimlerinden seri haberleşme ile bilgisayara gönderilmesi olduğu anlaşılmaktadır.

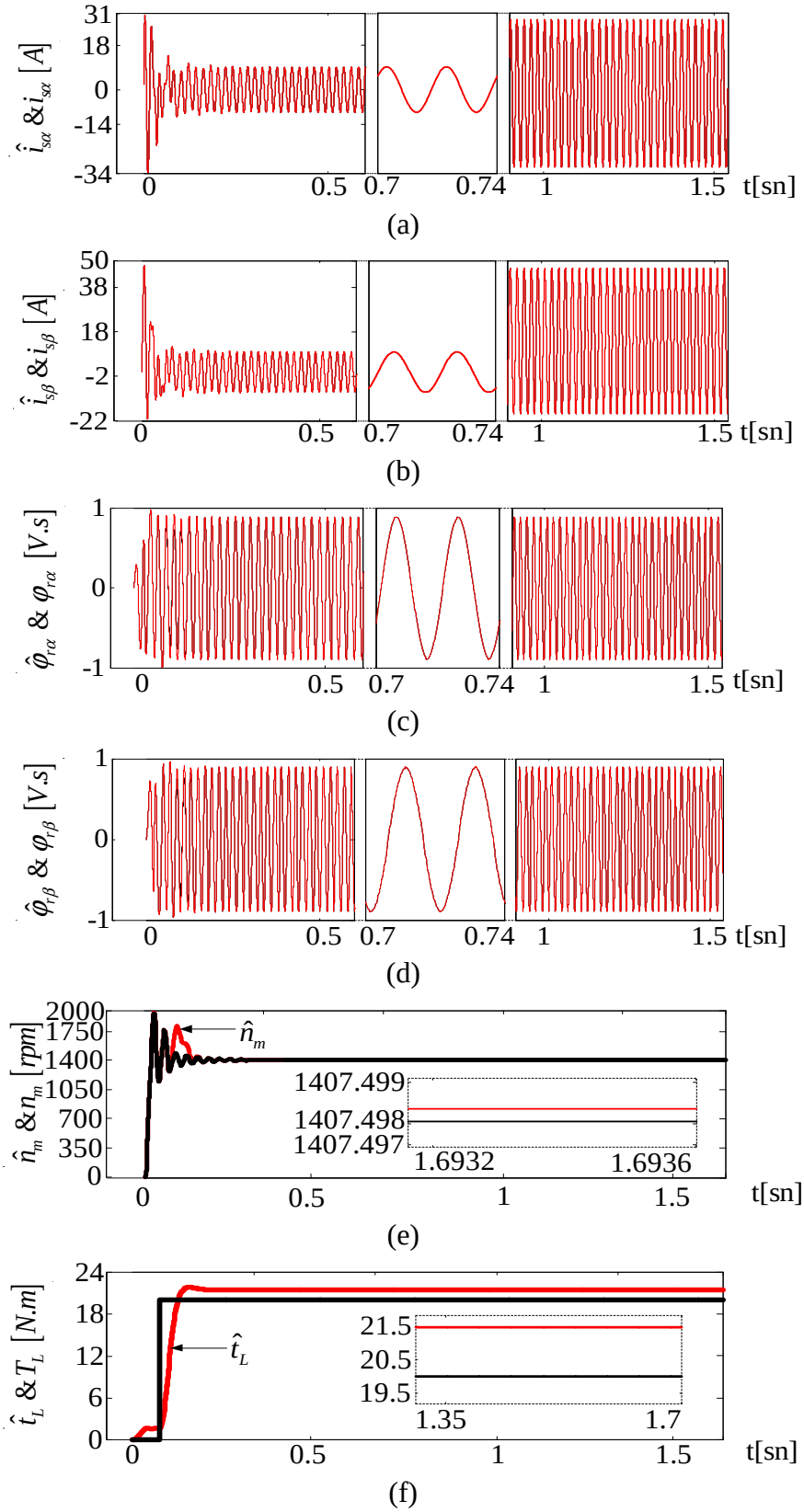
Seri haberleşme ile bilgisayara gönderilen hem modele ilişkin sonuçlar hem de kestirim sonuçları, çalışmanın bu kısmı için C# ile yazılan bir arayüz programı kullanılarak *.txt* formatında kaydedilmiştir. Kaydedilen sonuçlar MATLAB *m-file* ile yazılan program aracılığıyla grafiksel olarak gözlemlenmiştir.

Bu kısımda FPGA'da oluşturulan ASM modeline ve GKF algoritmasına ilişkin sonuçlar Şekil 4.15 ve bu sonuçlar arasındaki farklar Şekil 4.16'da sunulmuştur. GKF'nin kestirim başarımının ASM modeli ile doğrulanması amacıyla motor $0 \leq t < 0.08$ [s] aralığında yüksüz, $0.08 \leq t < 1.7$ [s] aralığında 20 [$N.m$] yük momenti ile yüklenmiş olup hız ise sıfır hızdan anma hızına ulaşılıncaya kadar artırılmıştır. Şekillerde "ˆ" GKF'nin kestirim sonuçlarını ve $e_{[.]}$, ASM modeline ilişkin sonuçlar ile GKF'nin kestirim sonuçları arasındaki farkı göstermektedir.

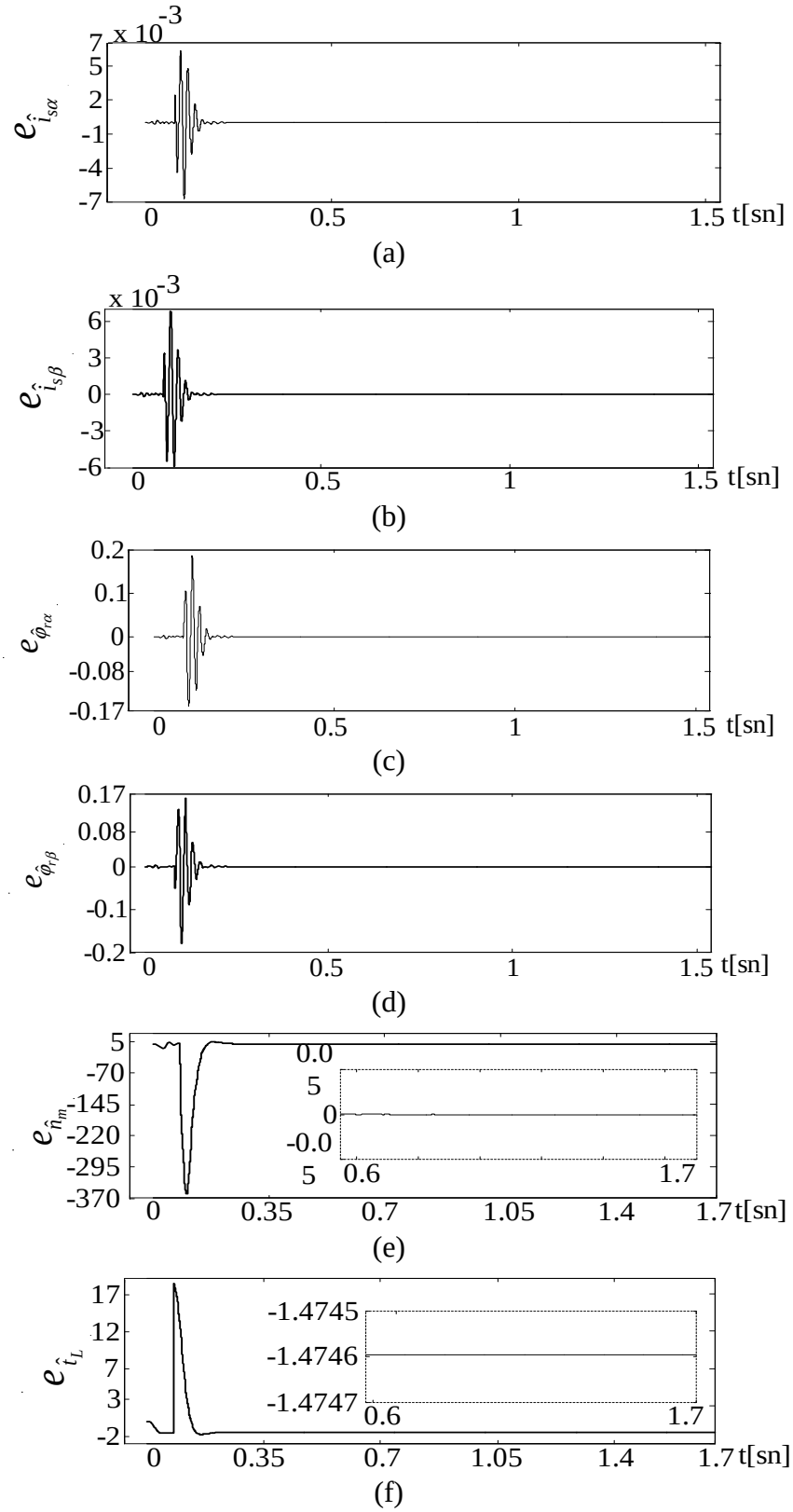


Şekil 4.14 ASM modeli ve GKF'nin donanım (FPGA) üzerinde gerçeklenen VHDL akış blokları

Şekil 4.15 ve 4.16'dan anlaşılacağı üzere GKF'nin kestirim başarımı oldukça yüksektir. Yalnızca momentin yüklendiği durumda hızda bir hata meydana gelmiş fakat çok kısa bir sürede kestirilen rotor hızı ASM modeline ilişkin gerçek rotor hızını yakalamıştır.

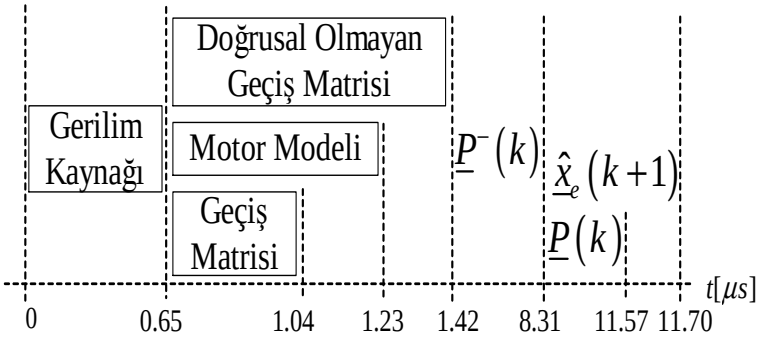


Şekil 4.15 FPGA'da gerçeklenen GKF algoritması ve ASM modeli sonuçları a) Stator akımının α bileşenleri ($\hat{i}_{s\alpha}$ ve $i_{s\alpha}$). b) Stator akımının β bileşenleri ($\hat{i}_{s\beta}$ ve $i_{s\beta}$). c) Rotor akısının α bileşenleri ($\hat{\varphi}_{r\alpha}$ ve $\varphi_{r\alpha}$). d) Rotor akısının β bileşenleri ($\hat{\varphi}_{r\beta}$ ve $\varphi_{r\beta}$). e) Mekanik hızlar ($\hat{n}_m$ ve n_m). f) Yük momentleri ($\hat{t}_L$ve T_L).



Şekil 4.16 FPGA'da gerçeklenen ASM modeli ile GKF algoritması arasındaki farklar. a) Stator akımı α bileşeni hatası ($e_{\hat{i}_{s\alpha}}$). b) Stator akımı β bileşeni hatası ($e_{\hat{i}_{s\beta}}$). c) Rotor akısı α bileşeni hatası ($e_{\hat{\varphi}_{r\alpha}}$). d) Rotor akısı β bileşeni hatası ($e_{\hat{\varphi}_{r\beta}}$). e) Mekanik hız hatası ($e_{\hat{n}_m}$). f) Yük momenti hatası ($e_{\hat{t}_L}$).

Şekil 4.17'de oluşturulan algoritmaların FPGA'da gerçeklenme sıraları ve süreleri, Çizelge 4.6'da algoritmada kullanılan mantık kaynakların sayıları, Çizelge 4.7'de ise algoritmadaki her bloğun gerçeklenmesi için ne kadar saat darbesi gerektiği belirtilmiştir.



Şekil 4.17 FPGA'da gerçeklenen algoritmalardaki fonksiyon bloklarının gerçeklenme sıraları ve süreleri

Çizelge 4.6 ASM modeli ve GKF algoritması için FPGA'nın kullandığı kaynak sayısı

Slice Register	18662
Flip Flops	18644
LUTs	30844
DSP48Es	18

Çizelge 4.7 FPGA'da oluşturulan ASM modeli ve GKF algoritmasındaki fonksiyon bloklarının her birinin gerçeklenmesi için gerekli saat darbesi sayıları

Gerilim Kaynağı	65 clock
Geçiş Matrisi	39 clock
Motor Modeli	58 clock
Doğrusal Olmayan Geçiş Matrisi	58 clock
$\underline{P}_k^-(k)$	689 clock
$\underline{P}(k)$	326 clock
$\underline{\hat{x}}_e(k+1)$	35 clock

4.2.3 Çalışma III: GKF tabanlı gerçek-zamanlı deneysel çalışmalar

Tez çalışmasının bu kısmında bir önceki kısımdaki GKF tabanlı gözlemleyici için benzetim çalışmalarından farklı olarak yalnızca Eşitlik 2.49'da belirtilen [$i_{s\alpha}$, $i_{s\beta}$, $\varphi_{r\alpha}$, $\varphi_{r\beta}$, ω_m, t_L] değişkenli GKF algoritması FPGA'ya gömülüp gerçek-zamanlı olarak farklı senaryolarla GKF'nin kestirim doğruluğu gösterilmiştir.

Bu uygulamaya ilişkin kullanılan gerçek-zamanlı deney düzeneği ve tasarlanan GKF VHDL akış diyagramı Şekil 4.18'de gösterilmiştir. Şekil 4.18'de görüldüğü gibi, motor parametreleri Çizelge 4.1'de verilen ASM'nin kullanıldığı deney düzeneği aracılığıyla öncelikle elde edilen stator gerilimleri, stator akımları, rotor hızı ve yük momenti bilgileri DS1104 kontrolör kartı ve bu kartın bilgisayar arayüzü olan Control Desk programı yardımıyla 130 [μs] örnekleme zamanında bilgisayara kaydedilmiştir. Elde edilen stator gerilimleri ($v_{s\alpha}(k)$ ve $v_{s\beta}(k)$) ve stator akımları ($i_{s\alpha}(k)$ ve $i_{s\beta}(k)$) bilgileri GKF'ye giriş olarak uygulanmaktadır. Giriş değerlerinin elde edilmesi esnasında DS1104 kontolör kartının örnekleme zamanını en az 130 [μs]'ye mertebelerine indirebilmesinden dolayı GKF'nin matematiksel işlemleri esnasında bir değişken olarak kullanılan örnekleme zamanı da giriş verileriyle paralellik sağlanabilmesi için 130 [μs] olarak alınmıştır. Fakat FPGA'da oluşturulan GKF algoritması yalnızca 11.75 [μs]'de gerçeklenmiştir. Elde edilen stator gerilim ve akım bilgileri FPGA'nın dahili hafıza birimlerine kaydedilmiştir. Kaydedilen stator gerilimleri giriş matrisi olarak, stator akımları ise ölçüm matrisi olarak GKF'nin gerçeklenmesi esnasında kullanılmıştır.

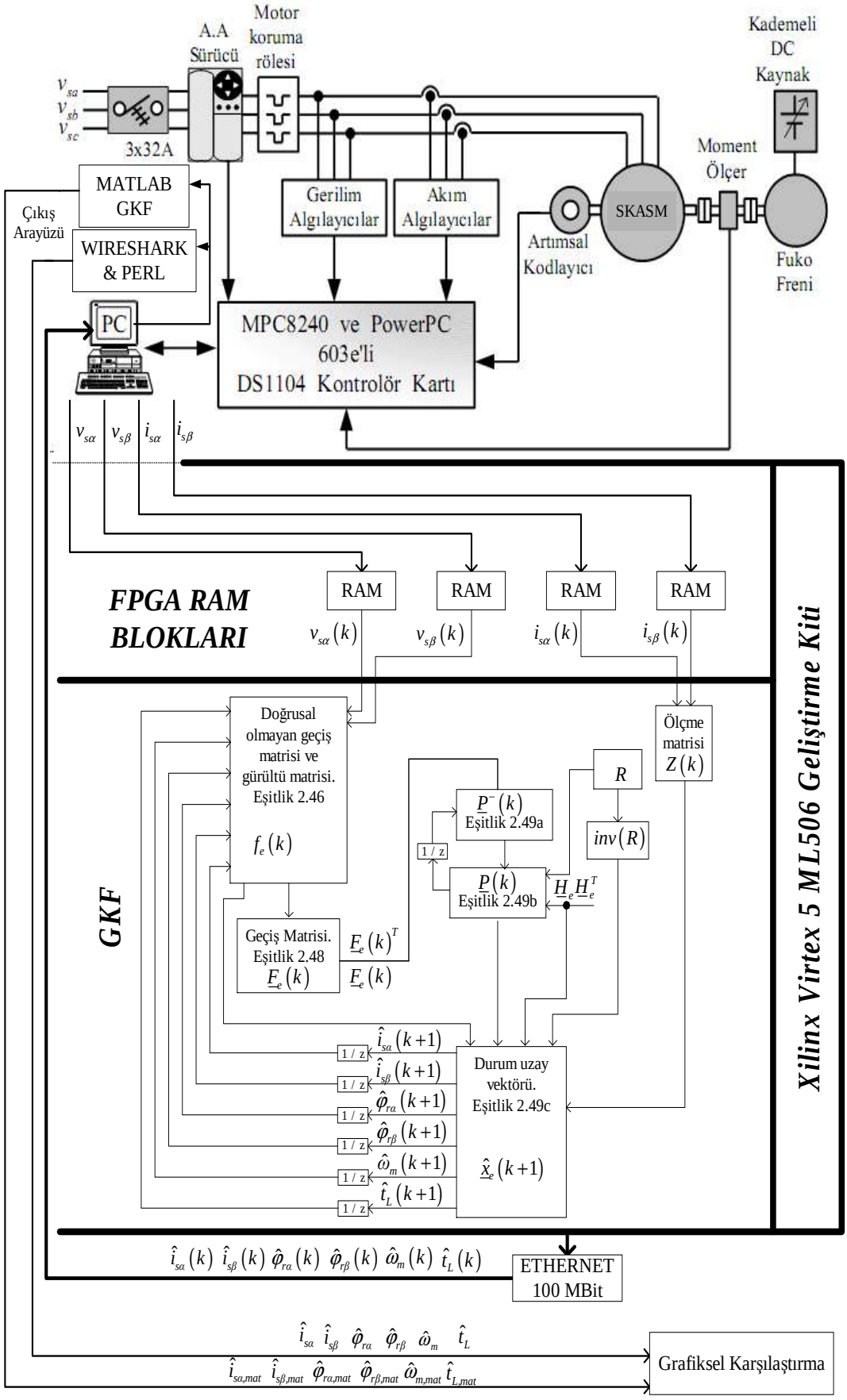
GKF'nin gerçeklenmesi için [51]'de belirlenen sistem gürültüsü kovaryans matrisi ***Q***, durum hata kovaryans matrisi ***P*** ve ölçme gürültüsü kovaryans matrisi ***R*** değerleri kullanılmıştır.

Q=diag[2.27917210637658 (A^2) 0.851449912250745 (A^2) 0.001 (Wb^2) 0.001 (Wb^2)

0.0960327675863374 ((rad/sn)^2) 0.0192600342824213((N.m)^2)]

R=diag[4.10989368154280 2.40406662188350]

P=diag[10 (A^2) 10 (A^2) 10 (Wb^2) 10 (Wb^2) 10 ((rad/sn)^2) 10 ((N.m)^2)]

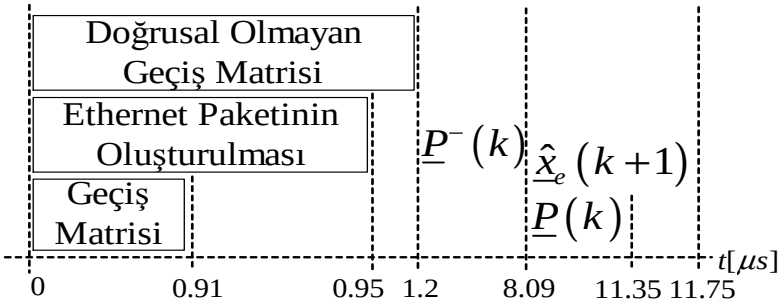


Şekil 4.18 Gerçek-zamanlı deney düzeneği ve GKF'nin VHDL akış diyagramı

FPGA üzerinde gerçeklenen GKF'nin kestirim sonuçlarına ilişkin stator akımı, rotor akısı, rotor hızı ve yük momenti bilgileri, FPGA'da saf Ethernet paketleri haline çevrilerek ethernet haberleşmesi protokolü aracılığıyla anlık olarak bilgisayara aktarılmıştır. FPGA tarafından gönderilen bu bilgiler, Microsoft firması tarafından geliştirilen WIRESHARK (Şekil 4.5) ethernet paketi toplama arayüzü aracılığıyla *.txt* formatında kaydedilmiştir. Kaydedilen ASM modeli sonuçlarına ait bilgiler PERL yazılım dili kullanılarak dizi şeklinde stator akımı, rotor akısı, rotor hızı ve yük momenti olarak ayrıştırılıp tekrar *.txt* formatında kaydedilerek MATLAB *m-file*'da yazılan program aracılıyla grafiksel olarak gözlemlenebilmiştir.

Tez çalışmasının bu kısmında, FPGA'da gerçeklenen GKF'yi doğrulamak için aynı GKF algoritması MATLAB *m-file*'da gerçeklenmiştir. FPGA'da gerçeklenen GKF'de olduğu gibi MATLAB ortamından da GKF'ye gerçek-zamanlı olarak deney düzeneğinden elde edilen stator gerilimleri ve stator akımları bilgisi giriş olarak verilmiştir. MATLAB *m-file*'da uygulanan GKF'den elde edilen sonuçlar FPGA'da uygulanan GKF sonuçları ile karşılaştırılmıştır.

Şekil 4.19'da gerçek-zamanlı uygulama için oluşturulan GKF algoritmasının fonksiyon bloklarının FPGA'da gerçeklenme sıraları ve süreleri, Çizelge 4.8'de algoritmada kullanılan mantık kaynakların sayıları, Çizelge 4.9'da ise algoritmadaki her bloğun gerçeklenmesi için ne kadar saat darbesi gerektiği belirtilmiştir.



Şekil 4.19 Gerçek-zamanlı uygulama için oluşturulan GKF algoritmasının fonksiyon bloklarının gerçeklenme sıraları ve süreleri

Çizelge 4.8 Gerçek-zamanlı uygulamada GKF algoritması için FPGA'nın kullandığı kaynak sayısı

Slice Register	13962
Flip Flops	13944
LUTs	24802
DSP48Es	18

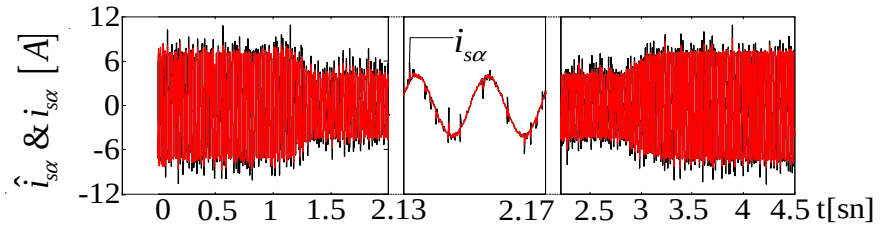
Çizelge 4.9 Gerçek-zamanlı uygulama için FPGA'da oluşturulan GKF algoritması fonksiyon bloklarının her birinin gerçeklenmesi için gerekli saat darbesi sayıları

Geçiş Matrisi	91 clock
Doğrusal Olmayan Geçiş Matrisi	120 clock
$\underline{P_k^-}(k)$	689 clock
$\underline{P}(k)$	326 clock
$\underline{\hat{x}_e}(k+1)$	35 clock

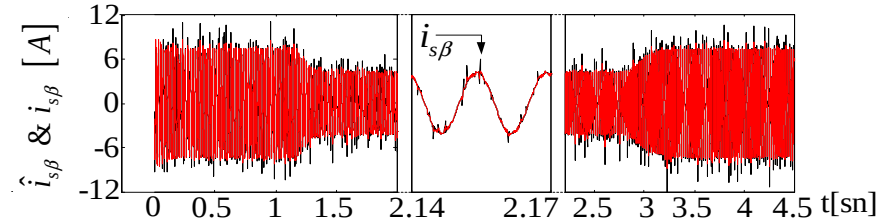
4.2.3.1 Senaryo I

GKF algoritmasının doğrulanması amacıyla izlenen ikinci senaryoda ASM anma hızında dönerken, yük momenti ilk olarak yaklaşık 1.3 [s]'de 20 [$N.m$]'den 0 [$N.m$]'ye daha sonra yaklaşık 3 [s]'de tekrar 20 [$N.m$]'ye yükseltilmiştir. Yük momentindeki bu değişime bağlı olarak hem FPGA hem de MATLAB *m-file* ortamlarında doğrulanan GKF algoritmasının sonuçları Şekil 4.20, 4.21 ve 4.22'de sunulmuştur.

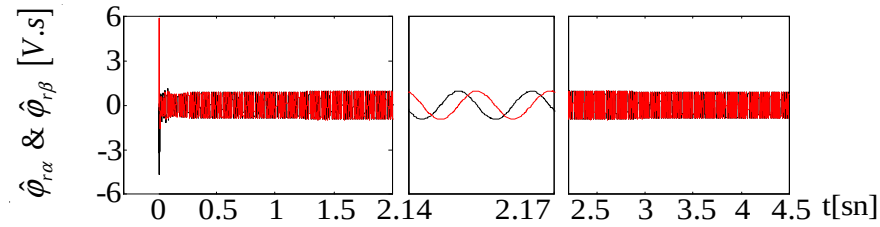
Şekil 4.20'da GKF algoritmasının, başlangıç koşullarının sıfır alınması sebebiyle kestirim sürecinin ilk başladığı anlarda rotor hızını ve yük momentini izlemelerinde hata oluştuğu, fakat 0.3 [s] gibi kısa bir süre sonra rotor hızı ve yük momentinin gerçek değerlerine yakınsadığı görülmektedir. Şekil 4.22'de de FPGA'da oluşturulan GKF algoritmasının sonuçları ile MATLAB *m-file*'da oluşturulan GKF algoritmasının sonuçları arasındaki fark olarak tanımlanan hatalara bakıldığında FPGA'da oluşturulan GKF algoritmasının kestirim başarımının doğrulandığı anlaşılmaktadır.



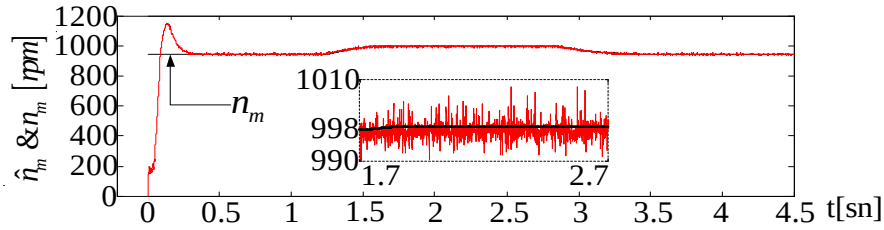
(a)



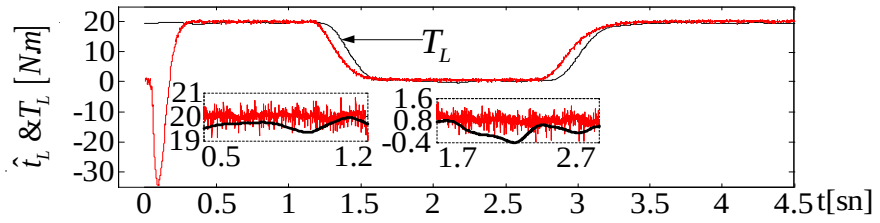
(b)



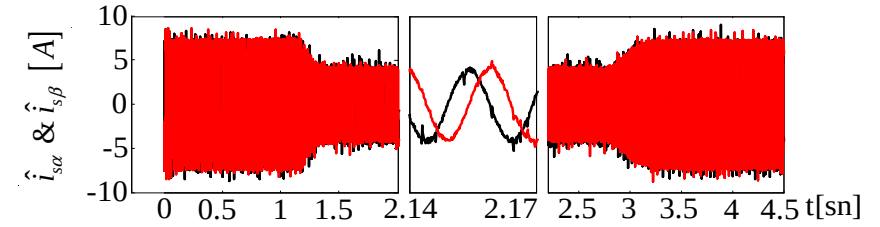
(c)



(d)

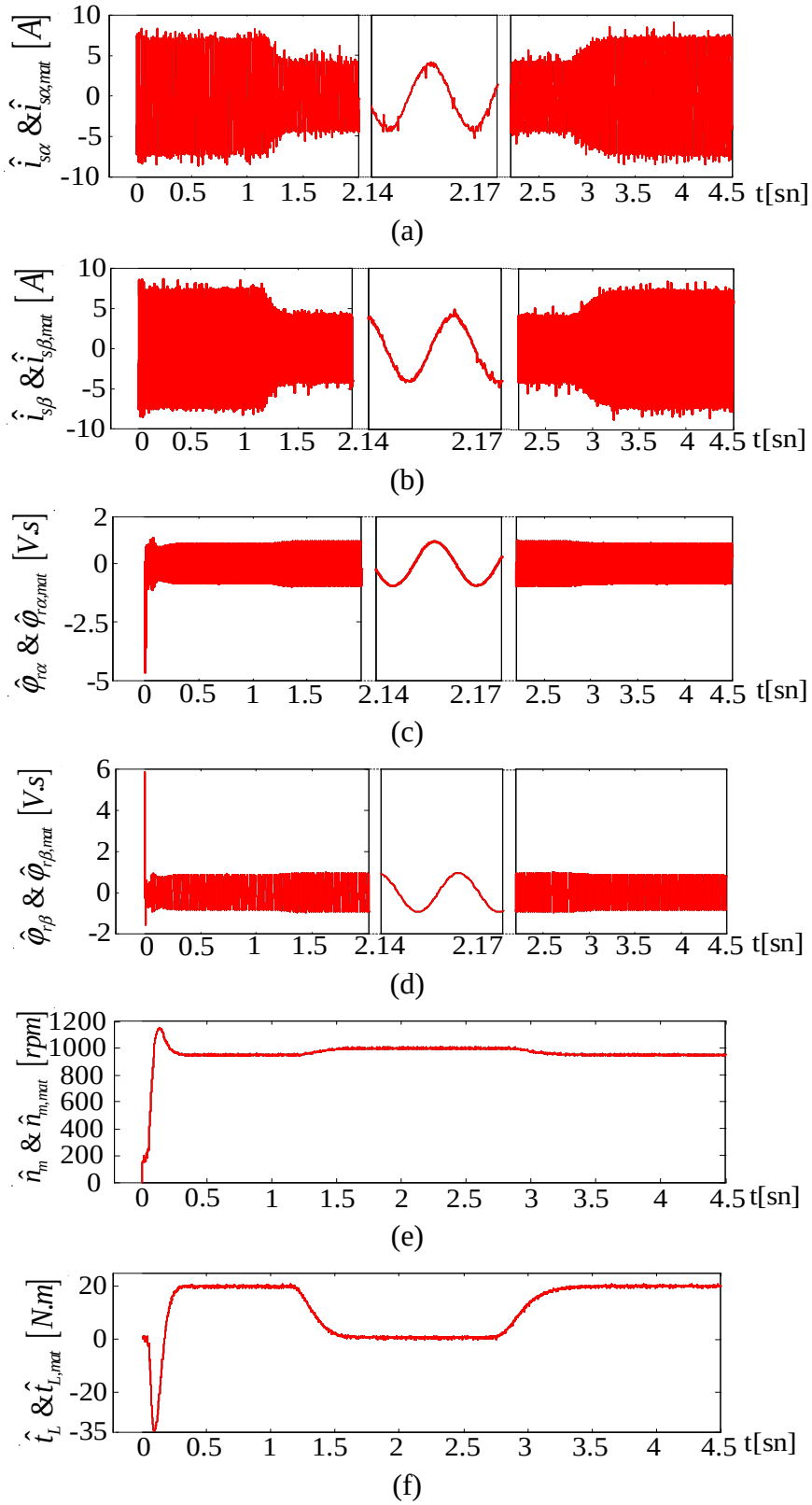


(e)

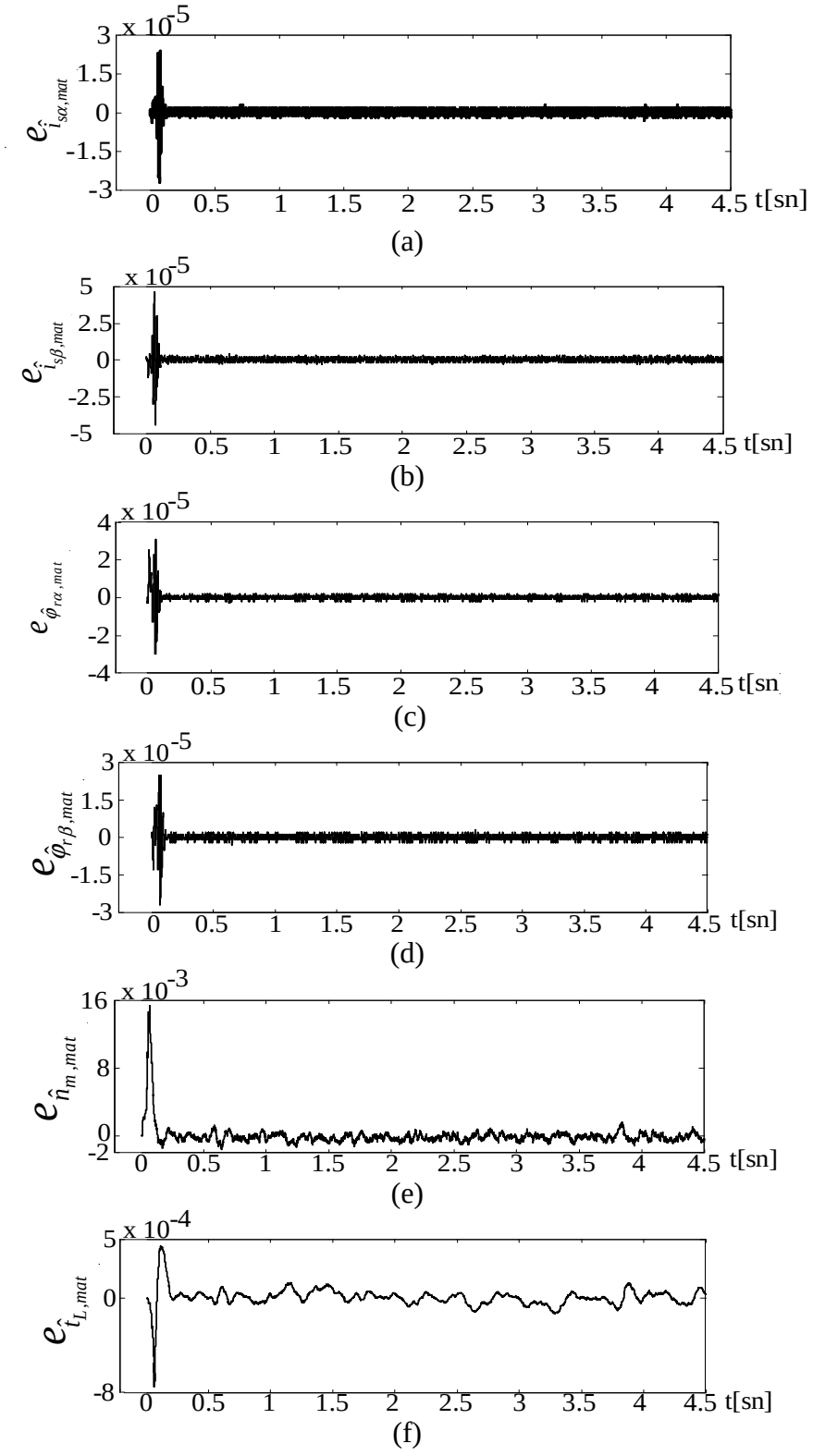


(f)

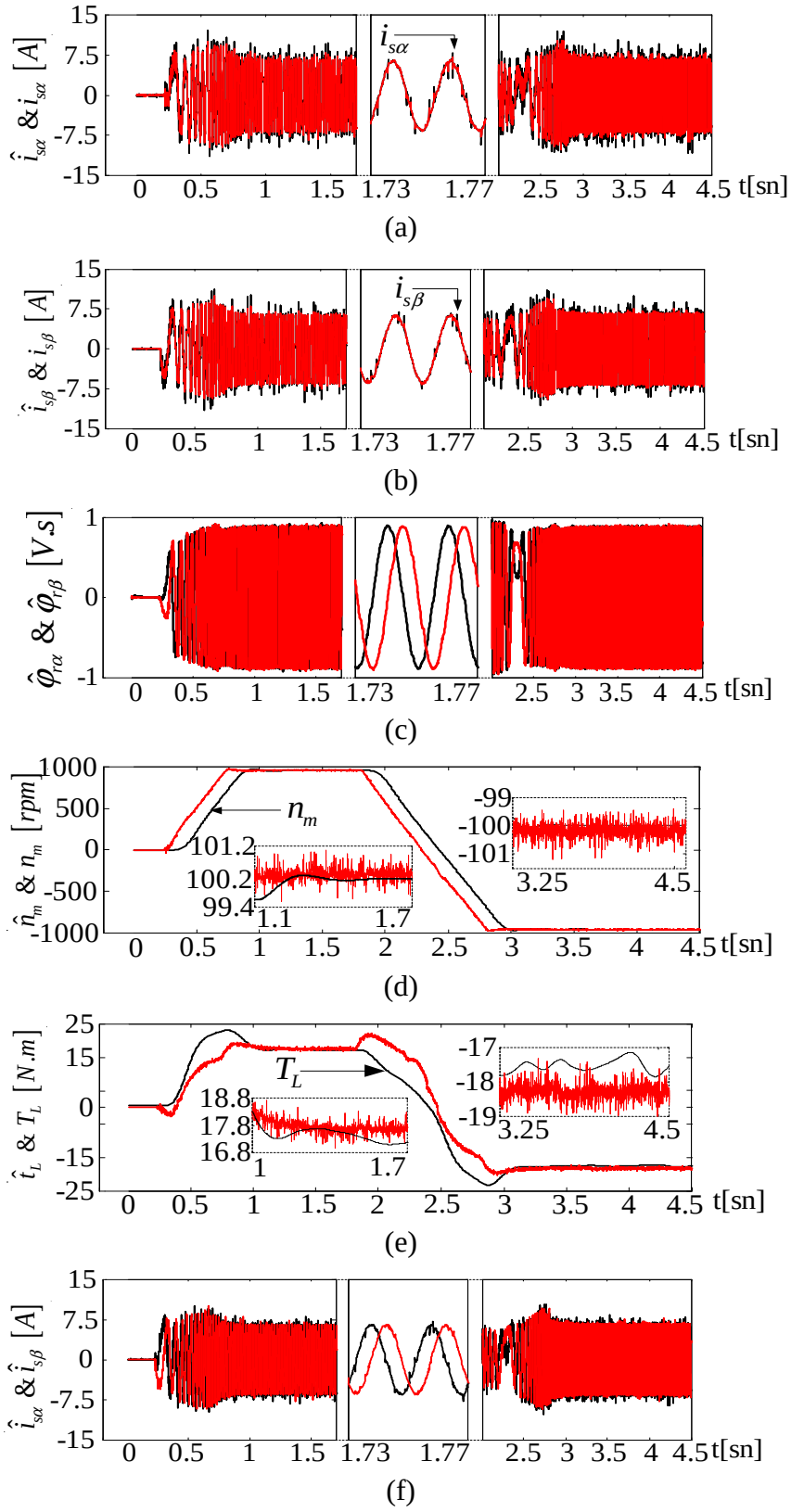
Şekil 4.20 Senaryo I: Gerçek-zamanlı yük momenti değişimleri için GKF algoritmasının kestirim sonuçları ve ölçülen değerler. a) Stator akımının α bileşenleri ($\hat{i}_{s\alpha}$ ve $i_{s\alpha}$). b) Stator akımının β bileşenleri ($\hat{i}_{s\beta}$ ve $i_{s\beta}$). c) Rotor akıları ($\hat{\varphi}_{r\alpha}$ve$\hat{\varphi}_{r\beta}$). d) Mekanik hızlar ($\hat{n}_m$ ve n_m). e) Yük momentleri ($\hat{t}_L$ve T_L). f) Stator akımı α ve β bileşeni ($\hat{i}_{s\alpha}$ ve $\hat{i}_{s\beta}$).



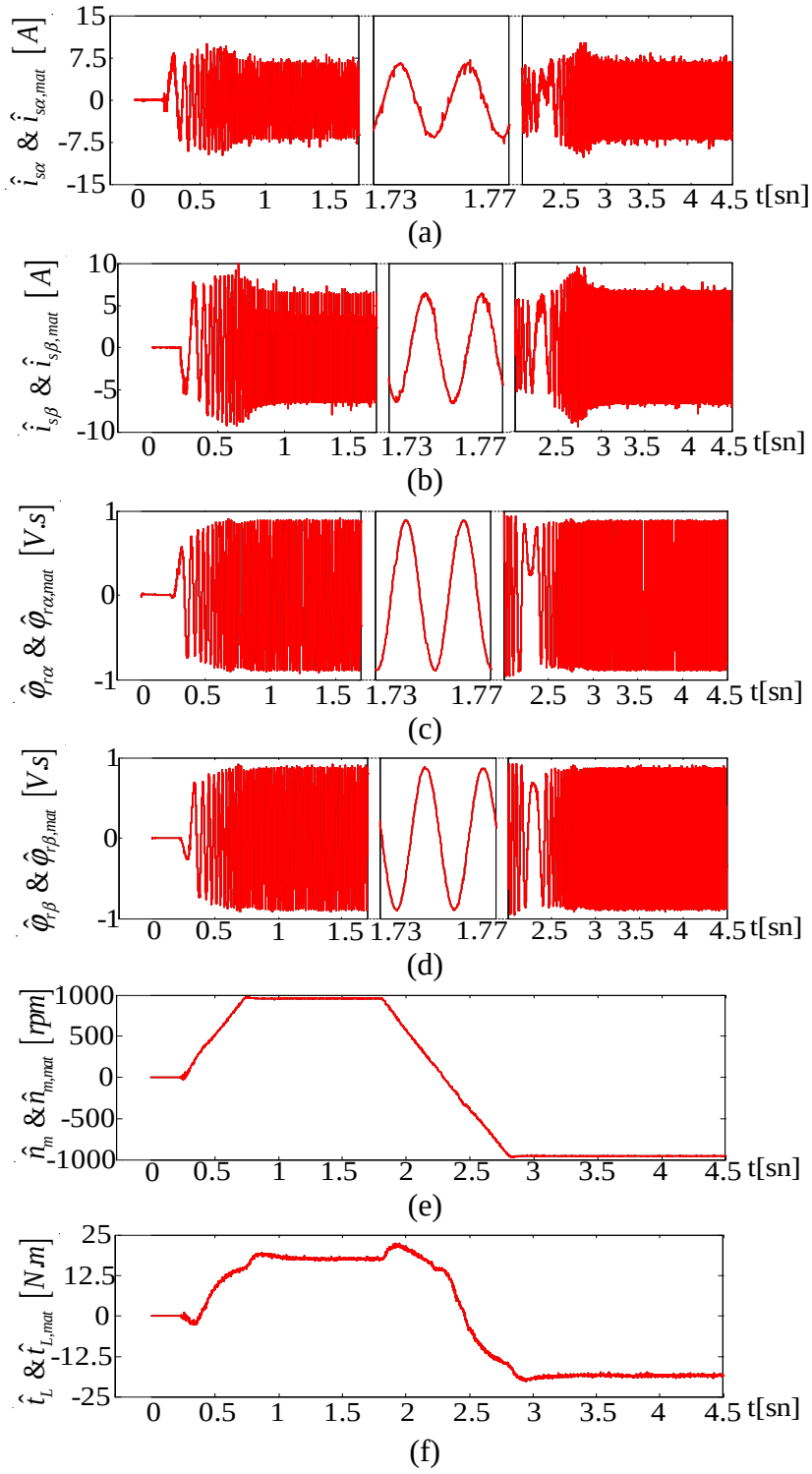
Şekil 4.21 Senaryo I: Gerçek-zamanlı yük momenti değişimleri için FPGA ve Matlab'te gerçekleştirilen GKF algoritmalarının sonuçları. a) Stator akımının α bileşenleri ($\hat{i}_{s\alpha}$ ve $\hat{i}_{s\alpha,mat}$). b) Stator akımının β bileşenleri ($\hat{i}_{s\beta}$ ve $\hat{i}_{s\beta,mat}$). c) Rotor akısının α bileşenleri ($\hat{\varphi}_{r\alpha}$ ve $\hat{\varphi}_{r\alpha,mat}$). d) Rotor akısının β bileşenleri ($\hat{\varphi}_{r\beta}$ ve $\hat{\varphi}_{r\beta,mat}$). e) Mekanik hızlar ($\hat{n}_m$ ve $\hat{n}_{m,mat}$). f) Yük momentleri ($\hat{t}_L$ ve $\hat{t}_{L,mat}$).



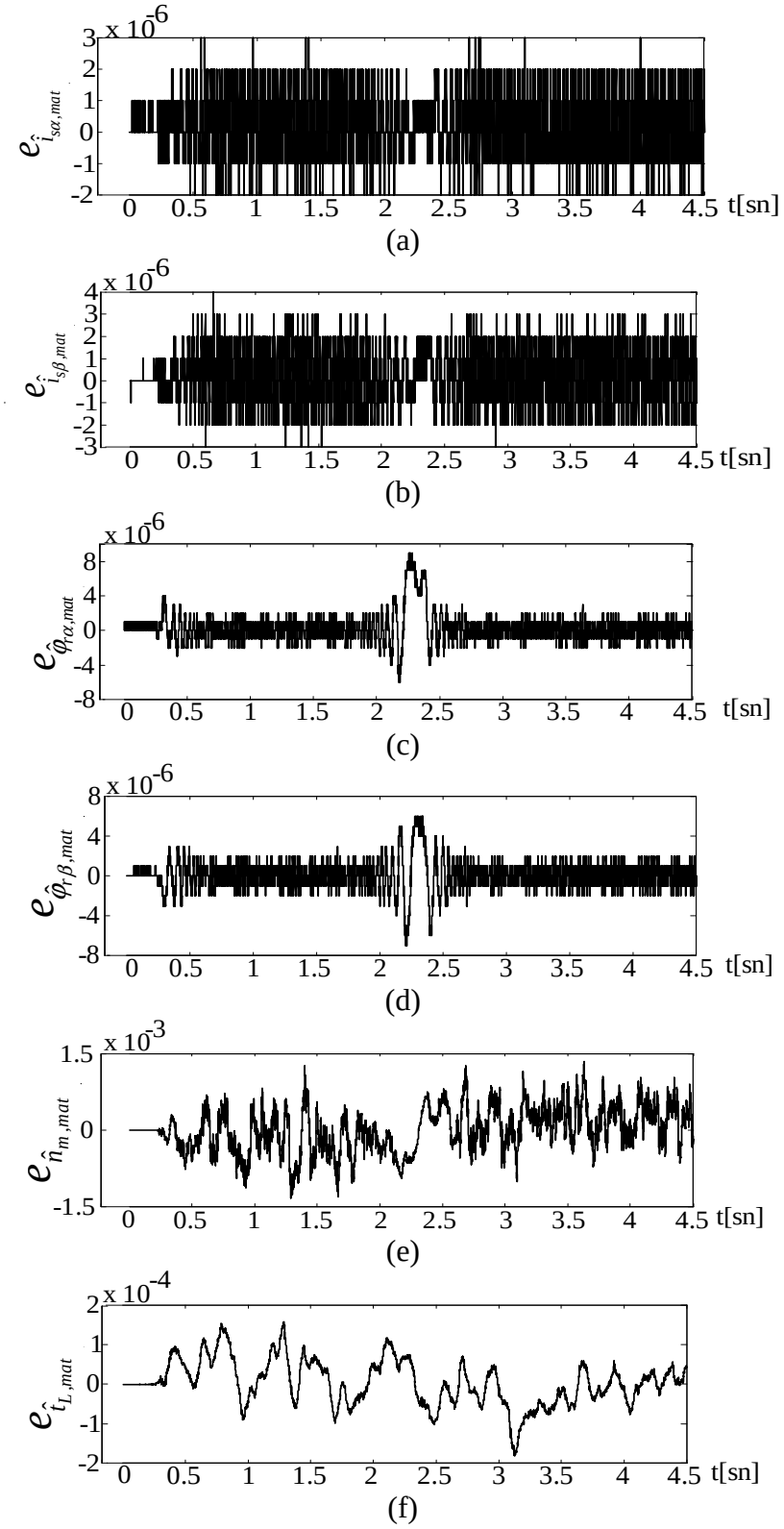
Şekil 4.22 Senaryo I: Gerçek-zamanlı yük momenti değişimleri için FPGA ve Matlab'te gerçekleştirilen GKF algoritmalarının sonuçları arasındaki farklar. a) Stator akımı α bileşenleri arasındaki farklar ($e_{\hat{i}_{s\alpha,mat}}$) b) Stator akımı β bileşenleri arasındaki farklar ($e_{\hat{i}_{s\beta,mat}}$), c) Rotor akısı α bileşenleri arasındaki farklar ($e_{\hat{\varphi}_{r\alpha,mat}}$), d) Rotor akısı β bileşenleri arasındaki farklar ($e_{\hat{\varphi}_{r\beta,mat}}$), e) Mekanik hızlar arasındaki farklar ($e_{\hat{n}_{m,mat}}$), f) Yük momentleri arasındaki farklar ($e_{\hat{t}_{L,mat}}$).



Şekil 4.23 Senaryo II: Gerçek-zamanlı rotor hızı terslendirmesi için GKF algoritmasının kestirim sonuçları ve ölçülen değerler. a) Stator akımının α bileşenleri ($\hat{i}_{s\alpha}$ ve $i_{s\alpha}$). b) Stator akımının β bileşenleri ($\hat{i}_{s\beta}$ ve $i_{s\beta}$). c) Rotor akıları ($\hat{\varphi}_{r\alpha}$ve$\hat{\varphi}_{r\beta}$). d) Mekanik hızlar ($\hat{n}_m$ ve n_m). e) Yük momentleri ($\hat{t}_L$ve T_L). f) Stator akımı α ve β bileşeni ($\hat{i}_{s\alpha}$ ve $\hat{i}_{s\beta}$).



Şekil 4.24 Senaryo II: Gerçek-zamanlı rotor hızı terslendirmesi için FPGA ve Matlab'te gerçekleştirilen GKF algoritmalarının sonuçları. a) Stator akımının α bileşenleri ($\hat{i}_{s\alpha}$ ve $\hat{i}_{s\alpha,mat}$). b) Stator akımının β bileşenleri ($\hat{i}_{s\beta}$ ve $\hat{i}_{s\beta,mat}$). c) Rotor akısının α bileşenleri ($\hat{\varphi}_{r\alpha}$ ve $\hat{\varphi}_{r\alpha,mat}$). d) Rotor akısının β bileşenleri ($\hat{\varphi}_{r\beta}$ ve $\hat{\varphi}_{r\beta,mat}$). e) Mekanik hızlar ($\hat{n}_m$ ve $\hat{n}_{m,mat}$). f) Yük momentleri ($\hat{t}_L$ ve $\hat{t}_{L,mat}$).bileşenleri ($\hat{\varphi}_{r\alpha}$ ve $\hat{\varphi}_{r\alpha,mat}$) d) rotor akısı β bileşenleri ($\hat{\varphi}_{r\beta}$ ve $\hat{\varphi}_{r\beta,mat}$), e) mekanik hız ($\hat{n}_m$ ve $\hat{n}_{m,mat}$), f) yük momentleri ($\hat{t}_L$ ve $\hat{t}_{L,mat}$)



Şekil 4.25 Senaryo II: Gerçek-zamanlı rotor hızı terslendirmesi için FPGA ve Matlab'te gerçekleştirilen GKF algoritmalarının sonuçları arasındaki farklar. a) Stator akımı α bileşenleri arasındaki farklar ($e_{\hat{i}_{s\alpha,mat}}$) b) Stator akımı β bileşenleri arasındaki farklar ($e_{\hat{i}_{s\beta,mat}}$), c) Rotor akısı α bileşenleri arasındaki farklar ($e_{\hat{\varphi}_{r\alpha,mat}}$), d) Rotor akısı β bileşenleri arasındaki farklar ($e_{\hat{\varphi}_{r\beta,mat}}$), e) Mekanik hızlar arasındaki farklar ($e_{\hat{n}_{m,mat}}$), f) Yük momentleri arasındaki farklar ($e_{\hat{t}_{L,mat}}$).

4.2.3.2 Senaryo II

Bu senaryoda, ASM yüklü durumda sıfır hızdan anma hızına kadar hızlandırılmış, daha sonra ASM anma hızı ve yükünde çalışırken yaklaşık 2. [*s*]'de terslendirilmiştir. FPGA'da gerçeklenen GKF algoritmasının sonuçları Şekil 4.23'de sunulmuştur.Karşılaştırma yapmak üzere hem FPGA hem de MATLAB *m-file* ortamlarında gerçeklenen GKF algoritmalarının sonuçları Şekil 4.24 ve 4.25'de sunulmuştur.

Şekil 4.23'den FPGA'da gerçeklenen GKF algoritmasının oldukça iyi bir izleme ve yakınsama başarımına sahip olduğu görülmektedir. Diğer taraftan, Matlab'taki eşdeğer algoritma sonuçlarının da Şekil 4.24 ve 4.25'den görüldüğü gibi hemen hemen FPGA'da oluşturulan GKF algoritması sonuçlarıyla bire bir olduğu söylenebilir. Bu sonuçlar (Şekil 4.23, 4.24 ve 4.25) FPGA'da tasarlanan GKF algoritmasının çok iyi bir doğrulukta olduğunu göstermektedir.

BÖLÜM V

SONUÇ

Bu çalışmada, ASM'nin rotor akısı yönlendirmeli hız-algılayıcısız kontrolü için daha önce [9]'da geliştirilmiş GKF algoritması VHDL ile FPGA üzerinde gerçeklenmiştir. Ref. [9]'daki gibi DS1104 donanımının kullanıldığı önceki çalışmalara göre aynı GKF algoritmasının gerçek-zamanda yaklaşık 11 (130 μs/11.75 μs) kat daha hızlı kestirimler yapılabileceği anlaşılmıştır. Böylece, GKF gibi hesap yükü fazla ama paralel işlem içeren algoritmaların, FPGA'da daha hızlı (daha küçük örnekleme zamanı ile) işletilebileceği gerçek-zamanlı olarak doğrulanmıştır. FPGA'nın diğer işlemcilere kıyasla oldukça düşük frekanstaki saat işaretine rağmen GKF algoritmasını daha hızlı işletebilmesinin en önemli nedeni, FPGA'nın mimari yapısında var olan paralel işlem yapabilme özelliğidir. Bu özellik kullanılarak, özellikle matris çarpım yoğunluğu fazla olan GKF algoritmasının işlem yoğunluğunun getirdiği dezavantajlar ortadan kaldırılmaya çalışılmış ve tatmin edici sonuçlar elde edilmiştir.

Çalışmada ilk olarak, ASM modeli ile GKF algoritması VHDL ile oluşturularak Xilinx firmasının geliştirmiş olduğu ISE12.2 ISim (VHDL/Verilog) simülatöründe benzetim çalışması yapılmıştır. İkinci olarak, VHDL ile oluşturulmuş olan ASM modeli ile GKF algoritması FPGA'ya gömülerek FPGA üzerinde donanımsal olarak benzetim çalışması yapılmıştır ve sonuçları RS232 seri haberleşmesi ile bilgisayara aktarılmış ve burada grafiksel olarak gözlemlenmiştir. Üçüncü olarak, ASM modeli devreden çıkarılarak yalnızca GKF algoritması FPGA'ya gömülmüştür. Gerçek-zamanlı deney düzeneğinden GKF'ye girilen akım ve gerilimler çekilerek FPGA'nın ROM belleklerine kaydedilmiştir. Kaydedilen akım ve gerilimler algoritmanın işleyişi sırasında sırasıyla ROM belleklerden okunarak GKF algoritmasında kullanılmıştır. GKF ile elde edilen kestirim sonuçları ethernet haberleşmesi ile bilgisayara aktarılmıştır ve grafiksek olarak gözlemlenmiştir. Bu çalışmada 3 farklı senaryo izlenmiştir. Bunların ikisinde yük momenti değişimine karşı, birinde ise rotor hızı ve yük momenti değişimlerine karşı GKF'nin stator akımlarını, rotor akılarını, rotor hızını ve yük momentini iyi bir başarımla kestirdiği gözlemlenmiştir. FPGA üzerinde uygulanmış olan üçüncü çalışma Matlab benzetim ve hesaplama programında da gerçekleştirilmiştir. Matlab ile gerçekleştirilen bu algoritmanın sonuçları FPGA üzerinde gerçekleştirilen algoritmanın

sonuçları ile karşılaştırılmış ve böylelikle FPGA'nın, GKF'deki işlem yoğunluğunu başarılı bir şekilde gerçekleştirebildiği anlaşılmıştır.

GKF algoritmasının sonuçlarının grafiksel olarak gözlemlenebilmesi için verilerin FPGA'dan bilgisayara aktarılabilmesi gerekmektedir. Bu amaç için öncelikle RS232 seri haberleşme protokolü kullanılmış ve başarım sağlanmıştır. Daha sonra ise Ethernet haberleşme protokolüne geçilerek veri aktarımı sırasında meydana gelen gecikmelerin önüne geçilmiş ve 27 Mbit/s mertebelerinde FPGA'dan bilgisayara veri iletişimi sağlanmıştır. Böylelikle FPGA üzerinde çeşitli haberleşme protokolleri de gerçekleştirilmiştir. Diğer taraftan ölçülen stator gerilimi ve akımlarını kullanarak stator duran eksenindeki stator akım bileşenlerini, rotor akısı bileşenlerini, rotor açısal hızını ve yük momentini kestiren GKF algoritması mevcut literatüre göre ilk kez bu tez kapsamında FPGA'da gerçeklenmiştir.

Ancak GKF algoritması rotor ve stator dirençlerinin değişimlerine de duyarlıdır. Daha iyi bir kestirim yapılması için stator ve rotor dirençlerinin algoritmaya güncellenmesi gerekmektedir. Bu sebeple gelecek çalışmalar bilinmeyen yük koşulu altında rotor ve stator direnci değişimlerine dayanıklı GKF algoritmasını gerçekleştirmeye ve uygun, örnekleme zamanı daha yüksek bir analog dijital çevirici kullanarak gerçek-zamanda sürekli (on-line) çalışan GKF tabanlı çalışmalar yapmaya yönelik olacaktır.

KAYNAKLAR

[1] Barut M., Sincap kafesli asenkron motorların algılayıcısız yüksek başarımlı kontrolüne yönelik genişletilmiş Kalman filtresi tasarım ve uygulaması, Doktora Tezi, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, İstanbul, 2005.

[2] Bose B. K., High performance control and estimation in AC drives, Proceedings of IEEE-the 23th International Conference on Control and Instrumentation (IECON'01), 1, 564-569, 1997.

[3] Takahashi, I. and Noguchi, T., A new quick-response and high-efficiency control strategy of an induction motor, Proceedings of IEEE-the Transactions on Industry Applications, 22 (5), 820–827, 1986.

[4] Depenbrock, M., Direct self-control (DSC) of inverter-fed induction machine, Proceedings of IEEE-Transactions on Power Electronics, 3 (4), 420-429, 1988.

[5] Yumuşak S., Asenkron motorda genişletilmiş Kalman filtresi ile sensörsüz doğrudan moment kontrolü, Yüksek Lisans Tezi, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, İstanbul, 2008.

[6] Holtz J., Sensorless control of induction machines-with or without signal injection", IEEE Transactions on Industrial Electronics, 53 (1), 7–30, 2006.

[7] Holtz J., Sensorless control of induction motor drives, Proceedings of The IEEE, 90 (8), 1359–1394, 2002.

[8] Finch J. W. and Giaouris D., Controlled AC electrical drives, IEEE Transactions on Industrial Electronics, 55 (2), 481–491, 2008.

[9] Barut M., Bogosyan S. and Gokasan M., Speed-sensorless estimation for induction motors using extended Kalman filters, IEEE Transactions on Industrial Electronics, 54 (1), 272-280, 2008.

[10] Bogosyan S., Barut M. and Gokasan M., Braided extended Kalman filters for sensorless estimation in induction motors at high-low/zero speed, IET Control Theory & Applications, formerly Proceedings of IEE–Control Theory & Applications, 1 (4), 987-998, 2007.

[11] Grewal M. S. and Andrews A. P., Kalman Filtering Theory and Practice Using Matlab (Third Edition), John Wiley & Sons Inc., 7, 3-4, 2008.

[12] Lee C. R. and Salcic Z., High-performance FPGA-based implementation of Kalman filter, Proceedings of Science Direct-Microprocessors and Microsystems, 21 (4), 257-265, 1997.

[13] Idkhajine L., Monmasson E. and Maalouf A., Fully FPGA-based sensorless control for AC drive using an extended Kalman filter, Proceeding of IEEE-the 35th Annual Conference (IECON ’09), 2925-2930, 2009.

[14] Charoensak C. and Abeysekera S. S., FPGA Implementation of Kalman low-pass filter for applications in sigma-delta (LA) demodulation, Proceedings of IEEE-Signal Processing Systems, (SIPS ’03), 219-223, 2003.

[15] Charoensak C. and Abeysekera S. S., FPGA implementation of efficient Kalman band-pass sigma-delta filter for application in FM demodulation, Proceedings of IEEE–the SOC Conference, 137-138, 2004.

[16] Turney R. D., Reza A. M. and Delva J. G. R., FPGA implementation of adaptive temporal Kalman filter for real time video filtering, Proceedings of IEEE-the International Conference-Acoustics, Speech, and Signal Processing (ICASSP '99), 4, 2231-2234, 1999.

[17] Bonato V., Peron R., de Holanda D. F. W. J. A. M., E., Cardoso M. and J. M. P., An FPGA implementation for a Kalman filter with application to mobile robotics, Proceedings of Industrial Embedded Systems-the International Symposium (SIES '07), 148-155, 2007.

[18] Bonato V., Marques E. and Constantinides G. A., A floating-point extended Kalman filter implementation for autonomous mobile robots, Proceedings of Field Programmable Logic and Applications-the International Conference (FPL ’07), 576-579, 2007.

[19] Islam A., Langlois J. M. P. and Noureldin A., A design methodology for the implementation of embedded vehicle navigation systems, Proceedings of IEEE-the International Conference-Electro/Information Technology (EIT '09), 297-300, 2009.

[20] Chappell S., MacArthurA., Sullivan C., Preston D. and Olmstead D., Exploiting real-time FPGA based adaptive systems technology for real-time sensor fusion in next generation automotive safety systems, Proceedings of Digital Object Identifier-the Design, Automation and Test, 3, 180-185, 2005.

[21] Robinson B. W., Hernandez-Garduno D. and Saquib M., Fixed and floating-point implementations of linear adaptive techniques for predicting physiological hand tremorin microsurgery, Proceedings of IEEE-the Selected Topics in Signal Processing, 4 (3), 659-667, 2010.

[22] Sudarsanam A., Barnes R., Carver J., Kallam R. and Dasu A., Dynamically reconfigurable systolic array accelerators: a case study with extended Kalman filter and discrete wavelet.transform algorithms, Proceedings of IEEE- the Computers & Digital Techniques (IET), 4 (2), 126-142, 2010.

[23] Pfeiffer D. and Franke U. Efficient representation of traffic scenes by means of dynamic stixels, Proceedings of IEEE- the Intelligent Vehicles Symposium (IV), 217–224, 2010.

[24] Idkhajine L., Monmasson E. and Maalouf A., FPGA-based sensorless controller for synchronous machine using an extended Kalman filter, Proceedings of Everyday Practical Electronics-the 13th European Conference-Power Electronics and Applications (EPE ’09), 1-10, 2009.

[25] Vyncke T. J., Boel R. K. and Melkebeek J. A. A., A comparison of stator flux linkage estimators for a direct torque controlled PMSM drive, Proceedings of IEEE-the 35th Annual Conference (IECON '09), 971-978, 2009.

[26] Idkhajine L. and Monmasson E., Design methodology for complex FPGA-based controllers–application to an EKF sensorless AC drive, Proceedings of IEEE-the Power Electronics Electrical Drives Automation and Motion International Symposium (SPEEDAM), 1012-1017, 2010.

[27] Idkhajine L., Monmasson E. and Maalouf A., Extended Kalman filter for AC drive sensorless speed controller-FPGA-based solution or DSP-based solution, Proceedings of IEEE-the Industrial Electronics International Symposium (ISIE), 2759-2764, 2010.

[28] Vyncke T.J., Boeland R.K. and Melkebeek J.A.A., On extended Kalman filters with augmented state vectors for the stator flux estimation in SPMSMs, Proceedings of IEEE-the 25th Annual Applied Power Electronics Conference and Exposition (APEC), 1711-1718, 2010.

[29] Ouhrouche M. Speed sensorless stator flux oriented control of an induction motor drive, Proceedings of IEEE-the International Conference (ICIT ’09), 1-6, 2009.

[30] Beguenane R., Mailloux J.-G., Simard S. and Tisserand A., Towards the system-on-chip realization of a sensorless vector controller with microsecond-order computation time, Proceedings of IEEE-the Canadian Conference on Electrical and Computer Engineering (CCECE ’06), 1073-1077, 2006.

[31] Barut M., Bogosyan S. and Gokasan M., Experimental evaluation of braided EKF for sensorless control of induction motors," Proceedings of IEEE-Transactions on Industrial Electronics, 55 (2), 620-632, 2008.

[32] Barut M. ve Demir R., DS1104 kontrolör kartı ile gerçek-zamanlı olarak asenkron motor modelinin doğrulanması, Türkiye Otomatik Kontrol Komitesi Ulusal Toplantısı (TOK 2010), 21-23 Eylül 2010.

[33] Grewal M. S. and Andrews A. P., Kalman Filtering Theory and Practice Using Matlab, John Wiley & Scons Inc., Canada, 2008.

[34] Haykin S., Kalman filtering and neural networks, John Wiley & Scons Inc., New York, 2001.

[35] Maybeck P.S., Stochastic models, estimation and control, Academic Press, New York, 1979.

[36] Welch G. and Bishop, G., An introduction to the Kalman filter, SIGGRAPH, Los Angeles, CA, 2001.

[37] Vas P., Sensorless vector and direct torque control, Oxford University Press, Oxford, 1998.

[38] Brown S. and Rose J., Architecture of FPGAs and CPLDs: A tutorial, Department of Electrical and Computer Engineering University of Toronto.

[39] Kuan I., Tessier R. and Rose J., FPGA architecture: survey and challenges, Foundations and Trends in Electronic Design Automation, 2 (2), 135-253, 2007.

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

[41] Yılmaz N., Alan programlamalı kapı dizileri (FPGA) üzerinde bir YSA'nın tasarlanması ve donanım olarak gerçekleştirilmesi, Yüksek Lisans Tezi, Selçuk Üniversitesi Fen Bilimleri Enstitüsü, Konya, 2008.

[42] BURR J., Digital neurochip design in parallel digital implementations of neural networks, Prentice Hall Inc., 223-1214.

[43] Maxfield C., The designer warrior's guide to FPGAs devices, tools and flows, Mentor Graphics Corporation and Xilinx Inc., 2004.

[44] Karen P. and Mehta N., Programmable logic design quick start hand book (Second Edition), Xilinx.

[45] Product Specification of Xilinx, ML505/ML506/ML507 evaluation platform user guide, UG347, 2009.

[46] Chapman K., 200 MHz UART with internal 16-byte buffer, Xilinx XAPP223, 2001.

[47] Product Specification of Xilinx, Virtex-5 FPGA embedded tri-mode Ethernet MAC wrapper v1.7, DS550, 2010.

[48] Product Specification of Xilinx, Virtex-5 FPGA embedded tri-mode Ethernet MAC wrapper v1.7 getting started guide, UG340, 2010.

[49] Pedroni V. A., Circuit design with VHDL, MIT Press, Cambridge, 2004.

[50] Barut M., Bogosyan S. ve Gökaşan M "Genişletilmiş Kalman filtresi tabanlı akı, hız ve yük momenti kestiricisi", Türkiye Otomatik Kontrol Komitesi Ulusal Toplantısı (TOK 2002), 2002.

[51] Zerdali E., Hız-algılayıcısız genişletilmiş Kalman filtresi tabanlı kestirici tasarımının genetik algoritma ile optimizasyonu, Yüksek Lisans Tezi, Niğde Üniversitesi Fen Bilimleri Enstitüsü, Niğde, 2011.