Model Öngörülü Denetimin Bir Sistemde ARM ile Gerçeklenmesi

Ender Kelleci

YÜKSEK LİSANS TEZİ

Elektrik Elektronik Mühendisliği Anabilim Dalı

Ağustos 2009

An Implementation of Model Predictive Control with ARM

Ender Kelleci

MASTER OF SCIENCE THESIS

Department of Electrical & Electronics Engineering

August 2009

Model Öngörülü Denetimin Bir Sistemde ARM ile Gerçeklenmesi

Ender Kelleci

Eskişehir Osmangazi Üniversitesi
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Elektrik Elektronik Mühendisliği Anabilim Dalı
Kontrol ve Kumanda Bilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır

Danışman: Prof. Dr. Abdurrahman Karamancıoğlu

Ağustos 2009

ONAY

Elektrik Elektronik Mühendisliği Anabilim Dalı Yüksek Lisans öğrencisi Ender Kelleci'nin YÜKSEK LİSANS tezi olarak hazırladığı " **Model Öngörülü Denetimin Bir Sistemde Arm ile Gerçeklenmesi**" başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

Danışman **:** Prof. Dr. Abdurrahman Karamancıoğlu

İkinci Danışman **: -**

Yüksek Lisans Tez Savunma Jürisi: **İmza**

Üye : Prof. Dr. Abdurrahman Karamancıoğlu .....

Üye : Prof.Dr. Hasan Hüseyin Erkaya .....

Üye : Doç.Dr. Osman Parlaktuna .....

Üye : Y. Doç. Dr. Bünyamin Tamyürek .....

Üye : Y. Doç. Dr. Ahmet Yazıcı .....

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Prof. Dr. Nimetullah BURNAK

Enstitü Müdürü

ÖZET

Bu tez çalışmasında Model Öngörülü Denetim uygulaması ARM (Acorn RISC Machine) işlemci tabanlı geliştirme kartı kullanılarak gerçekleştirilmiştir.

Öncelikle takometreli bir doğru akım motordan oluşan tek giriş ve tek çıkışlı süreç matematiksel olarak modellenmiş ve bu süreç için bir Model Öngörülü Denetleyici MATLAB, Simulink ve Model Predictive Control Toolbox kullanılarak oluşturulmuştur. Oluşturulan Simulink modeli Real Time Workshop (RTW) ile C koduna dönüştürülerek CygWin ve Linux Cross Compiler Araçları kullanılarak diğer destek dosyalarıyla birlikte derlenmiş ve ftp veya seri haberleşme protokolleri yardımıyla ARM işlemci tabanlı geliştirme kartına yüklenmiştir. Yüklenen program çalıştırılarak Model Öngörülü Denetim uygulaması ARM üzerinde başarılı olarak gerçekleştirilmiştir.

Anahtar Kelimeler: Model Öngörülü Denetim, ARM, MATLAB, Simulink, Real Time Workshop, CygWin, Linux Soft Real Time Target.

SUMMARY

In this thesis, an implementation of Model Predictive Control with ARM (Acorn RISC Machine) processor based development card was studied.

A model predictive controller is implemeneted by using Model Predictive Control Toolbox of MATLAB and relevant design commands are converted to C code with Real Time Workshop using softwares MATLAB and Simulink. Compilation of the code with support files was achieved with CygWin and Linux Cross Compiler Tools and then transfer to ARM processor from serial or file transfer protocol.

The controller above is used in velocity control of a DC motor for validation of its functioning. It is shown that implementation of model predictive control with ARM processor functions successfully.

Keywords: Model Predictive Control, ARM, MATLAB, Simulink, Real Time Workshop, CygWin, Linux Soft Real Time Target.

TEŞEKKÜR

Yüksek lisans çalışmamda elinden gelen desteği veren başta danışmanın Prof. Dr. Abdurrahman Karamancıoğlu'na, derslerimde gösterdiği sabır ve yönlendirmelerden dolayı Doç. Dr. Osman Parlaktuna' ya ve Y. Doç. Dr. Ahmet Yazıcı'ya, bu çalışmanın sonlanmasında beni cesaretlendiren sevgili dostum Serdar Ergen'e ve desteklelerini benden hiçbir zaman esirgemeyen aileme sonsuz saygı ve teşekkürlerimi sunarım.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

ÇİZELGELER DİZİNİ

Çizelge **Sayfa**

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama
ADC	Analog Dijital Dönüştürücü
ARM	(Acorn RISC Machine)
DAC	Dijital Analog Dönüştürücü
MPC	Model Öngörülü Kontrol
RTW	Real-Time Workshop

BÖLÜM 1

GİRİŞ

Bu tez çalışmasında Model Öngörülü Denetim (Model Predictive Control -MPC) uygulamasının ARM işlemci tabanlı geliştirme kartı kullanılarak gerçekleştirilmesi amaçlanmaktadır. Model öngörülü denetim, denetlenmek istenen sürecin dinamik davranışını uzun bir gelecek için optimize eden denetim girişlerini hesaplayarak bunların sadece o an gerek duyulan kısmını sürece uygulayan ve gelecek için yaptığı optimizasyonları sürekli güncelleyen bir denetleyici tasarım yaklaşımı olarak tanımlanabilir. Bu tez çalışması uygulamaya ağırlık vermekte olup; model öngörülü denetim algoritmasının ARM işlemci tabanlı geliştirme kartı ve uygun yazılımlarla birlikte uzman olmayan mühendislerin bile kolaylıkla kullanılabilmesi için bir kaynak oluşturmayı hedeflemektedir.

Yapılan çalışmada model öngörülü denetimin ARM işlemci tabanlı geliştirme kartı kullanılarak doğru akım motor hız kontrolünü gerçekleştirmesi hedeflenmektedir. Bu hedef doğrultusunda; tek giriş ve tek çıkışlı bir süreç olan doğru akım motor hız kontrolü modellenmiş, oluşan model ARM işlemci tabanlı geliştirme kartına özel olarak model tabanlı kod geliştirme araçları kullanılarak kodlanmış ve oluşturulan model öngörülü denetleyici ile doğru akım motor hız kontrolü gerçekleştirilmiştir.

Yapılan tez çalışmasının organizasyonu aşağıdaki gibidir;

Bölüm 2'de Model Öngörülü Denetimin temel felsefesi anlatılmıştır. Konu ile ilgili terminoloji de bu bölümde tanıtılmıştır.

Bölüm 3'de ise Model Öngörülü Denetim uygulamasının ARM işlemci kullanılarak gerçekleştirilmesine yardımcı olan donanımsal ve yazılımsal araçlara değinilmiştir.

Bölüm 4'te tek giriş ve tek çıkışlı bir doğru akım motorun hız kontrolünün modellenmesi ve model öngörülü denetleyicinin oluşturulması anlatılmıştır.

Bölüm 5'te üretilen modelin ARM işlemci tabanlı geliştirme kartı üzerinde gerçekleştirilme basamakları donanımsal ve yazılımsal olarak anlatılmıştır.

Bölüm 6'da ise bu çalışmanın tartışma ve sonuçlarına değinilmiştir.

BÖLÜM 2

MODEL ÖNGÖRÜLÜ DENETİM

Model Öngörülü Denetim yaklaşık kırk yıl önce, petrol rafineleri gibi uzun zaman sabitlerine sahip süreçlerde çeşitli denetim problemlerine alternatif çözüm oluşturmak için geliştirilmiştir.

Model Öngörülü Denetimin (Model Predictive Control -MPC) 1980'lerden itibaren yaygınlaşan kullanımı, hızlı ve büyük bellekli mikroişlemciler o zamanlarda henüz yokken, öncelikle zaman sabiti uzun süreçlerde görülmüştür. Kullandığı donanım ve yazılımlar itibariyle MPC ileri düzey bir denetim tekniği olarak sınıflanabilir. İleri denetim tekniği olması denetim sinyallerini oluştururken optimizasyon algoritması çalıştırarak ilgilenilen süreç çıkış sinyallerini tasarımcının arzu ettiği optimizasyon ölçütüne uygun olarak sağlayan yapıya sahip olmasıdandır (Wikipedia). Daha eski denetim teknikleri donanım olarak analog yükseltici, integrator ve toplayıcıları ağırlıklı olarak kullanırken, MPC yönteminde zorunlu olarak bunları nümerik olarak gerçekleyen karşılıkları kullanılmaktadır.

Lineer olmayan sistem modelleri için seyrek sayıda MPC algoritması varlığına karşın lineer sistem modelleri için olgun ve sistematik algoritmalar bulunmaktadır. Model öngörülü denetim ideal çalışma için denetlenmek istenen sistemin kesin modeline gerek duyar; ancak sistem modelindeki belirsizlikler durumunda dahi uygun geri besleme konfigürasyonları kullanılarak MPC algoritması başarılı olarak çalıştırılabilir.

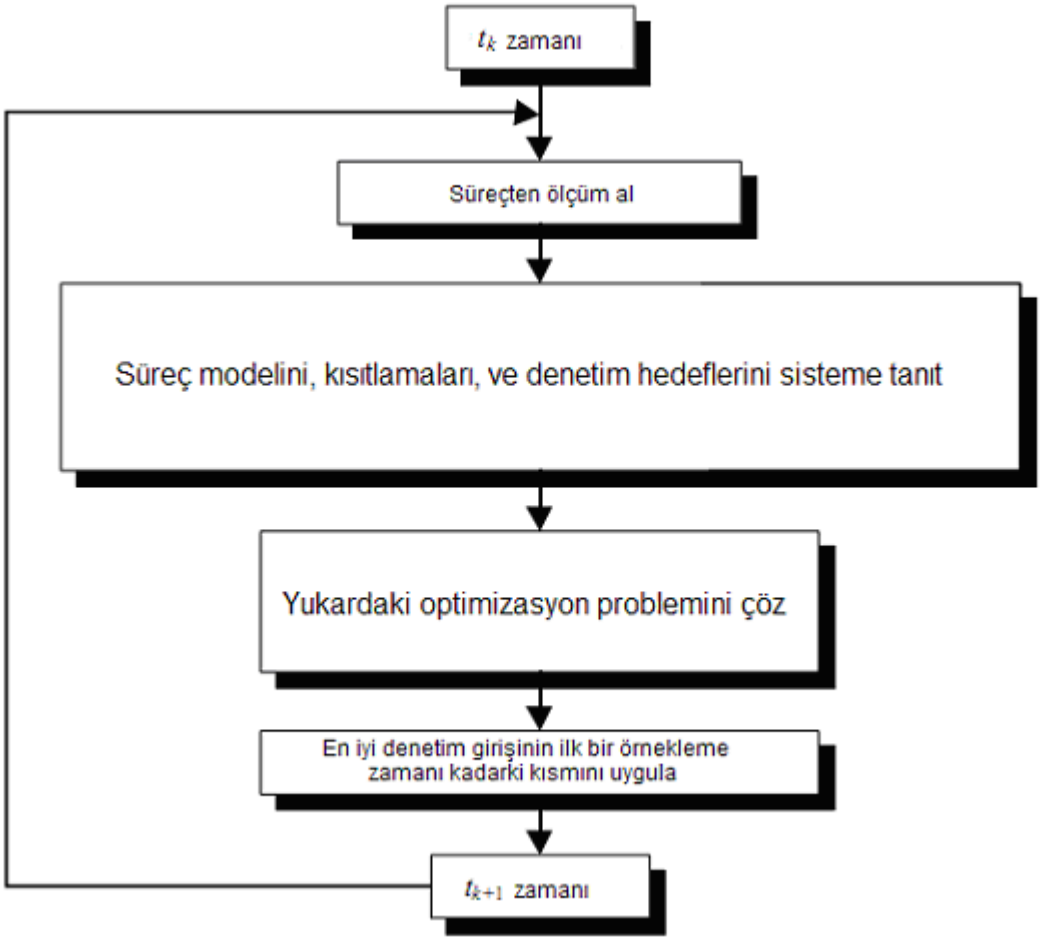
Pratikte, MPC süreç modelinin bir sonlu "ufuk" boyunca optimal performans koşullarını tespit etmesi ve bunları kullanması esasına dayanır. Verilen bir t anından itibaren T saniyelik süre boyunca arzu edilen süreç çıkış yörüngesini daha önceden

belirlenen optimalite ölçütüne göre sağlayan sistem giriş sinyali *u* bulunur. Ancak bulunan *T* saniyelik *u* girişinin tamamı sinyale uygulanmaz. Örnek olarak, *T* saniyenin ilk yüzde beşlik kısmına karşılık gelen *u* sisteme uygulanır. *t+0.05T* kadar sonra aynı işlem yinelenerek *[t+0.05T,t+1.05T]* aralığında arzu edilen süreç çıkış yörüngesi için gerekli *u* fonksiyonu hesaplanır. Bu hesaplama yöntemi yinelemeli olarak istenen süreç çıkış yörüngesi istenen zaman süresi için elde edilene kadar devam eder.

MPC lineer sistem modelleri için kapalı form çözümler üretmektedir. Kapalı form çözümler için lineer sistem modelleri kuadratik optimizasyon ölçütleri ile birlikte kullanılmaktadır. MPC'nin sayısal bilgisayarlar kullanılarak gerçekleştirilmesi için süreç modeli ve optimizasyon ölçütünün kesikli zaman bölgesine taşınması yaygın olarak kullanılmaktadır. Süreç veya optimizasyon ölçütünün herhangi birinin lineer olmadığı durumda çeşitli yaklaşıklıklardan yararlanılmakta veya nümerik yaklaşımlar kullanarak işlem yükü artmaktadır.

MPC'nin çalışma felsefesi Şekil 2.1'deki akış diyagramında ifade edilebilir. (Nikolau, 2009)

Model öngörülü denetimin temel avantajı sistem kısıtlarınında tasarım aşamasında değerlendirilebilir olması ve denetim sinyallerinin tasarımcının belirlediği optimallik ölçütünü sağlar şekilde seçilmesidir. Ana dezavantaj olarak, hesaplama sürelerinin uzunluğu ve hesaplamalar için gerekli işlemlerin ilgili işlemci tarafından desteklenmesinin oluşturacağı donanım ve işletim sistemi kısıtlarıdır.



Şekil 2.1 Model Öngörülü Denetimin Akış Diyagramı

Bu tez çalışmasında model öngörülü denetim algoritması olarak Model Predictive Toolbox'ta kayıtlı algoritma kullanılacaktır. MPC Toolbox'a sistemimiz, kısıtlamalar ve diğer donanım ve yazılım ayarları tanıtılarak sistem denetimi gerçekleştirilecektir. Bu sistem denetiminde kullanılan ARM kartının kontrol mühendisleri için yeni bir alternatif oluşturması amaçlanmaktadır. Tezdeki tanıtımların bu karta dayalı MPC denetiminin uzman olmayan mühendisler tarafından dahi kolaylıkla kullanıma imkan tanıyacağı umulmaktadır.

BÖLÜM 3

KULLANILAN ARAÇLAR

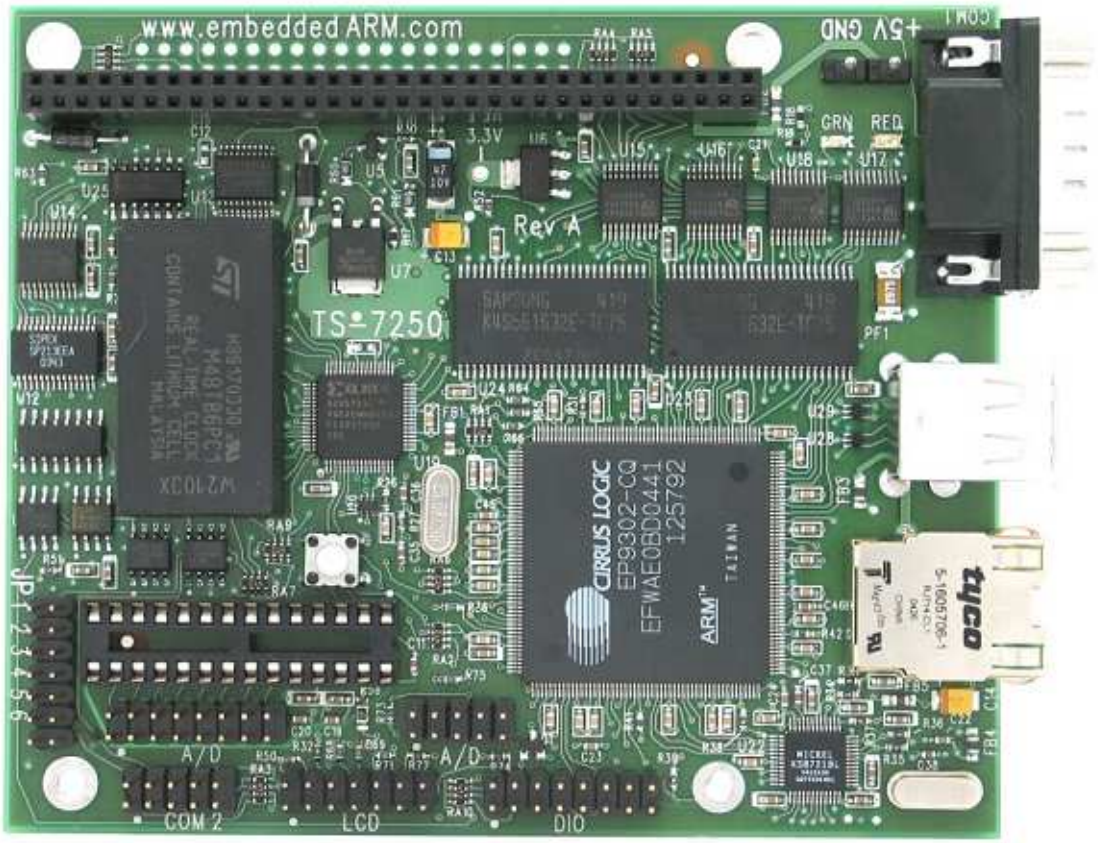
3.1 Donanımsal Araçlar

3.1.1 ARM İşlemci Tabanlı Geliştirme Kartı (ARM)

ARM mimarisi (orijinal adı **Acorn RISC Machine**) pek çok gömülü tasarımda kullanılan 32-bit RISC işlemci mimarisidir. Güç tasarruf özelliklerinden dolayı, ARM işlemciler mobil elektronik gibi düşük güç tüketiminin kritik bir parametre olduğu pazarda en fazla tercih edilen işlemcidir.

Günümüzde ARM işlemci ailesi yeryüzündeki 32-bit gömülü işlemcilerin %75' ini oluşturmaktadır. ARM işlemciler taşınabilir cihazlardan (PDA, cep telefonu, medya oynatıcılar, avuç içi oyun üniteleri ve hesap makineleri) bilgisayar parçalarına kadar (disk sürücüler, masaüstü router' lar) tüketici elektroniğinin her alanında yoğun olarak kullanılmaktadır.

Gömülü tasarım uygulamalarında kullanılan en popüler ARM mimarisi komut kümeleri 32-bit'lik ARM ve 16-bit'lik Thumb komut kümeleridir. Her Thumb komutunun bir ARM komut karşılığı vardır fakat bunun tersi doğru değildir. Bu sorun bu iki komut kümesinin bir arada çalışmasının mümkün olması ile aşılmıştır (interworking). Bu sayede, 16-bit'lik komut kümesinin daha az bellek kullanımı ve 32-bit'lik komut kümesinin üstün işlevsellik özellikleri bir arada kullanılabilmektedir. (Wikipedia)



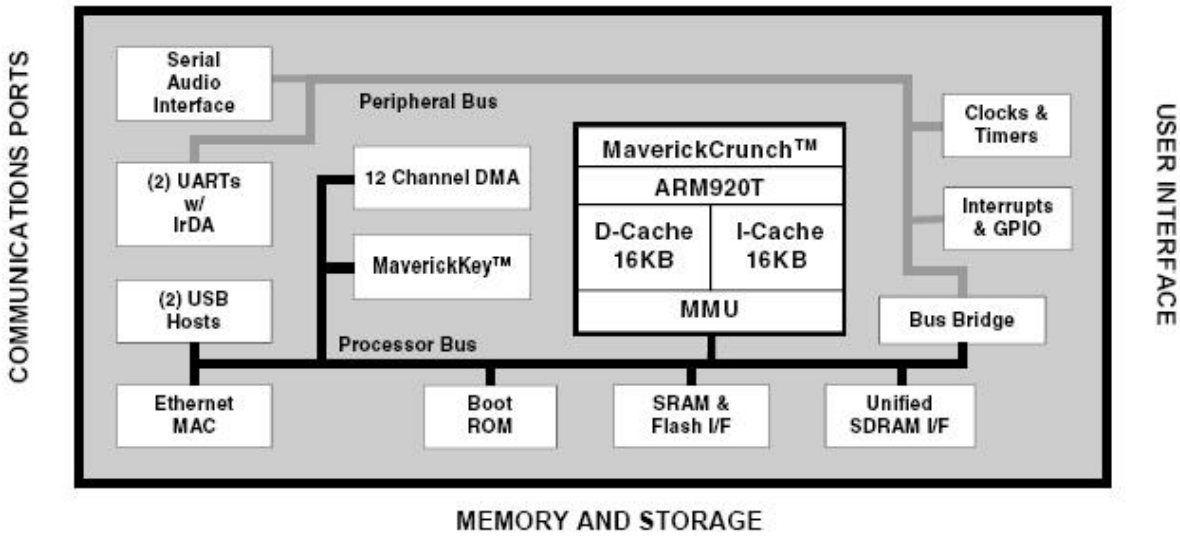
Şekil 3.2 TS-7250 ARM Geliştirme Kartı

Bu çalışmada tercih edilen mikro işlemci geliştirme kartı Technologic Systems tarafından geliştirilen TS-7250 ARM işlemci kartıdır. Şekil 3.1'de Arm tabanlı geliştirme kartı görülmektedir. Bu kartın seçilmesinin sebepleri aşağıda maddeler halinde belirtilmiştir.

- **Kart üzerinde bulunan kayan nokta ünitesine (floating point unit) sahip 200Mhz hızında ARM9 işlemci**

Kartta kullanılan işlemci, Cirrus Logic tarafından geliştirilen bütünleşik matematik motoruna (math engine) sahip ARM920T tabanlı EP9302 kod adlı işlemcidir. Bu matematik motoru donanımsal kayan nokta işlemlerini, kayan nokta ünitelerini yazılımsal olarak taklit eden işlemcilerden çok daha verimli yapmasını

sağlar. Bu işlemci ayrıca Linux tarafından desteklenen hafıza yönetim ünitesine de sahiptir. EB9302 kodlu işlemcinin fonksiyonel blok şeması Şekil 3.2 ‘de verilmiştir.



Şekil 3.2 EB9302 İşlemcinin Fonksiyonel Blok Şeması (EP9302 Datasheet, 2004)

- **Kart üzerinde bulunan 1/10/100 Mbit LAN**

Kartın üzerinden bulunan standart LAN konnektörü kartla kolay iletişim kurulmasını sağlar.

- **İki adet USB 2.0 destekli konnektör**

Bu donanım standart USB donanımlarıyla haberleşmeyi sağlar. Bu bağlantı noktalarından biri USB üzerinden Debian tabanlı Linux işletim sistemin çalıştırılmasını sağlar.

- **Beş kanal 12-bit A/D dönüştürücülerin kartın üzerinde bulunması**

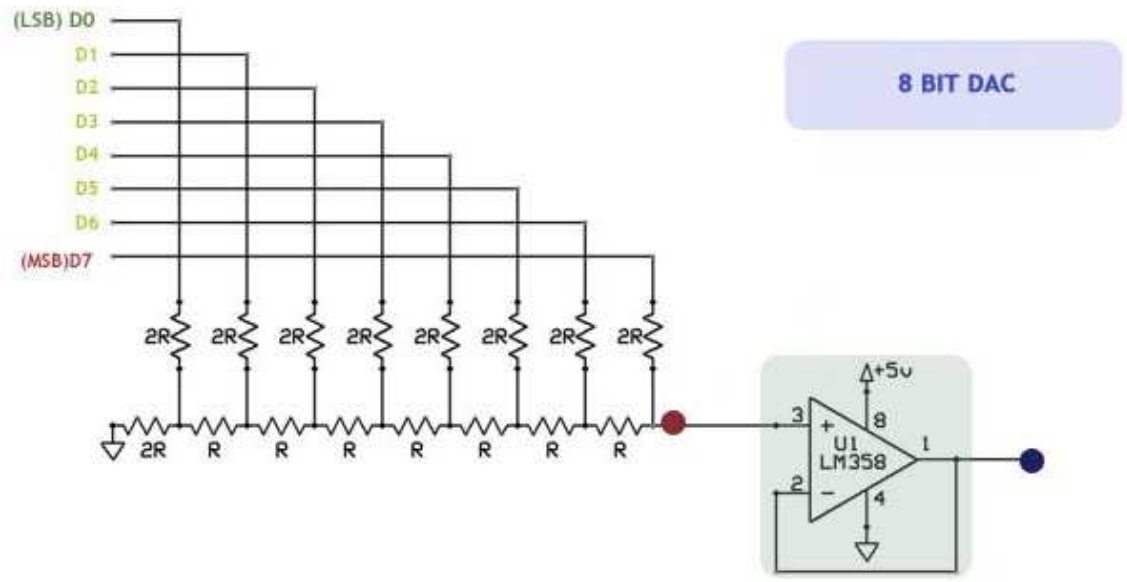
Bu dönüştürücüler dışardan alınan analog ölçümlerin dijital değerlere dönüştürülmesini sağlar. Kart üzerinde bulunan dönüştürücülere kolaylıkla kullanıcı koduyla ulaşılabilir.

- **Kartın üzerinde çalışan Linux işletim sistemi**

Ufak bir Linux dağıtımı kartla beraber Flash RAM üzerinde yüklü olarak gelmektedir. Bu ufak dağıtım kartla LAN üzerinden telnet bağlantısının veya RS232 üzerinden seri haberleşme yapılmasını sağlar. TS-Linux adında ki bu Linux dağıtımı kart üzerinde bulunan bütün donanımları destekler.

3.1.2 D/A Çevirici Kart

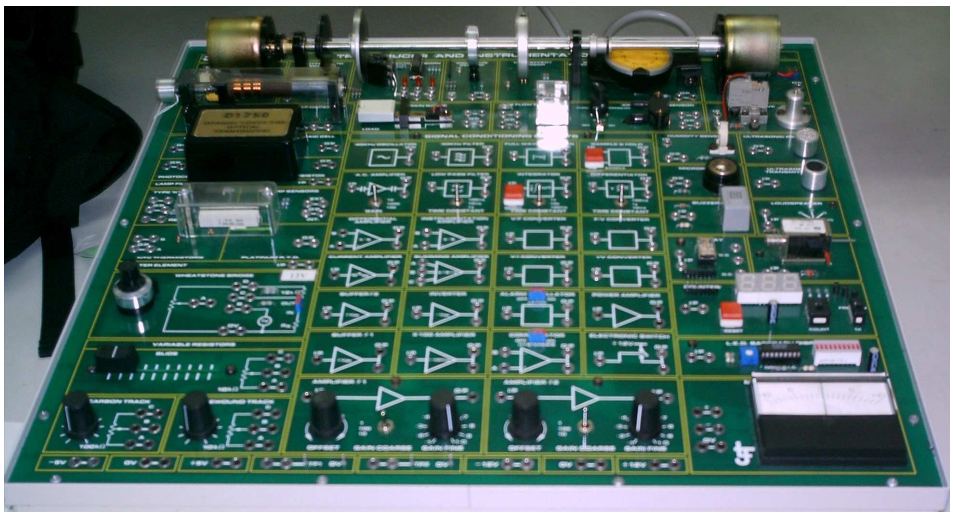
TS-7250 Arm işlemci kartının üzerinde DAC (D/A Çevirici) veya PWM çıkışı bulunmamasından dolayı DAC işlemini görecek kardeş bir kart Şekil3.3' de verildiği şekilde tasarlanmıştır (ikalogic).



Şekil 3.3 D/A Kart Şeması

3.1.3 Digiac 1750

Klasik kontrol sistemleri eğitiminde kullanılan deney setidir. Şekil 3.3'te Digiac1750'nin genel görünümü gösterilmiştir. MPC algoritmasının ARM işlemci geliştirme kartıyla gerçekleştirimi bu deney setindeki doğru akım motorunu ve ilgili donanımlarını kullanarak test edilmiştir.



Şekil 3.4 Digiac 1750

3.2 Yazılımsal Araçlar

3.2.1 MATLAB

MATLAB yüksek seviyeli bir teknik programlama dili olmasının yanında algoritma geliştirme, verilerin görselleştirilmesi, veri analizi ve sayısal hesaplamalar için etkileşimli bir yazılım paketidir. MATLAB ile teknik hesaplama problemlerini, C, C++ ve Fortran gibi geleneksel programlama dillerinden daha hızlı bir şekilde çözülebilir. MATLAB yazılımının birçok alanda uygulamaları vardır. İçerdiği "toolbox" adı verilen paketler aracılığıyla sayısal işaret işleme, kontrol sistemleri tasarımı-simülasyonu, test ve ölçüm, finansal modelleme ve analiz, haberleşme gibi birçok alanda kullanılabilir.

Ana Özellikleri:

- Teknik hesaplamalar için yüksek seviyeli bir dil
- Kodların ,dosyaların ve verilerin düzenlenmesi için bir geliştirme ortamı
- İteratif tasarım ve problem çözme yöntemleri için interaktif araçlar
- Lineer cebir, istatistik, Fourier analizi, filtreleme, optimizasyon ve sayısal integrasyon için matematik fonksiyonlar
- Verilerin görselleştirilmesi için 2 ve 3 boyutlu grafik araçları
- Grafik arayüzler tasarlamak için araçlar

MATLAB'in kullanımı olmayan bir mühendislik alanı yok gibidir. MATLAB bilim ve mühendislik alanlarında kullanılan ortak matematiksel işlemler için birçok fonksiyonu içinde bulundurmaktadır. Bu fonksiyonlar MATLAB dilinin temelini oluşturmaktadır. MATLAB ile double, single ve integer gibi genel veri tipleri üzerinde işlemler gerçekleştirilebilir.

Toolbox adı verilen modüller sayesinde işaret işleme, optimizasyon, istatistik ve sembolik matematik gibi birçok özelleştirilmiş fonksiyonları kullanılabilir. Bu modüller MATLAB'e ayrıyetten dahil edilebilmektedirler. Bu modüllere her geçen gün bir yenisi eklenmektedir. Böylece MATLAB'in kullanım alanları da her geçen gün genişlemektedir.

MATLAB birçok klasik algoritmayı tek bir komutta sunmaktadır. Böylece matematiksel hesaplamaların bilgisayarda yapılması diğer programlama dillerinde (C, C++, Fortran gibi) olduğundan daha kısa sürede ve daha az kod yazılarak gerçekleştirilebilmektedir. Böylece problemin bilgisayara uyarlanmasından çok problemin kendisine yoğunlaşmak daha da kolaylaşmaktadır (Figes).

3.2.2 Model Predictive Control Toolbox

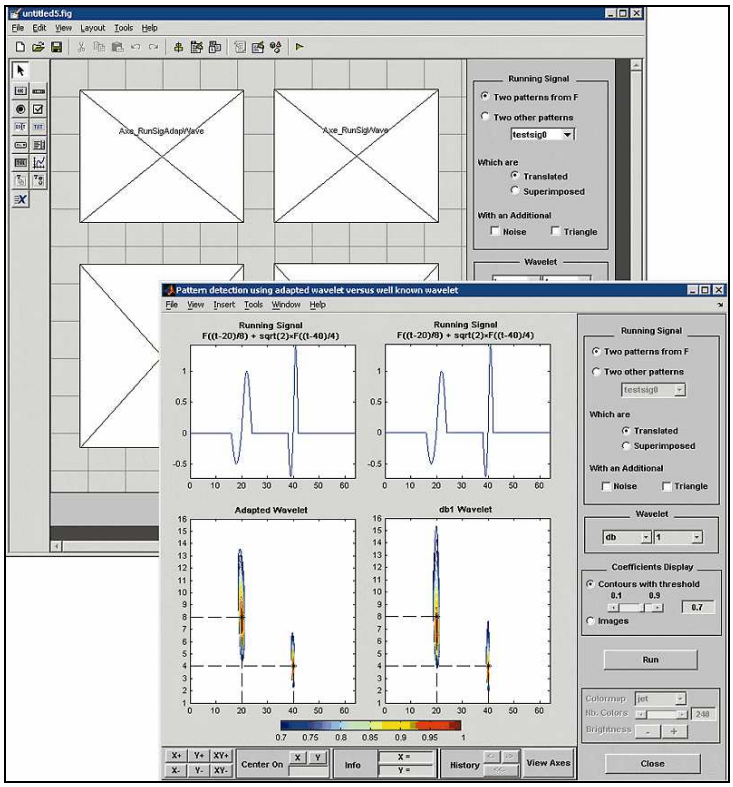
Model Predictive Control Toollbox, Model öngörülü denetleyicilerin MATLAB ve Simulink kullanılarak tasarım ve simülasyonu gerçekleştirilmesi için MATLAB fonksiyonları, kullanıcı arayüzü ve Simulink bloklarını sağlar. Bu denetleyiciler giriş ve çıkış kısıtlamaları altında çok giriş/çok çıkış sistemlerin performanslarını optimize eder (Mathworks).

3.2.3 Simulink

Simulink; MATLAB ailesinin kullanıcılara, zaman domeni ile dinamik ve arayüzleri görsel sistem modellerinin kurulması, benzetimi ve çözümü konusunda hizmet sunan bir ürünüdür. Simulink ile hazırlanan uygulamalar bir test ortamındaymışçasına sürekli veya ayrık zamanlı analiz edilebilir, analize bağlı tasarım ve geliştirme işlemleri gerçekleştirilebilir. Simulink birçok matematiksel ifadeden başlayarak; havacılık, haberleşme, elektrik, elektronik ve kontrol sistemleri, finansal, mekanik, kablosuz sistemler ile sinyal işleme, sanal gerçeklik, yüksek seviyeli gerçek zamanlı donanımsal çalışma, gömülü kontrol ve otomatik kod üretimi ile birim programlama gibi birçok özel alana yönelik blok kütüphaneleri içermektedir. Simulink ile modelleme, bir sistemin kağıt üstüne taslak çizimi yapılması kadar kolaydır. Simulink grafiksel kullanıcı arayüzü; hali hazırda barındırdığı çeşitli blok setleri ile "Sürükle ve Bırak" işlemine dayanan basit bir mantıkla, sistem elemanlarını ve sistemleri oluşturur. Sistem elemanlarının parametrelerini değiştirmek temel çift tıklama ile yapılabilmektedir. Sistem özellikleri de hazır kullanıcı arayüzleri ile belirlenmektedir. İstenildiği takdirde, kullanıcılar kendi bloklarını oluşturabilmekte ve kendi kütüphanelerini hazırlayabilmektedirler (Figes).

3.2.4 Guide

GUIDE, kullanıcı ara yüzü tasarımının yapıldığı MATLAB alt programıdır. Şekil 3.4'te örnek bir ara yüz gösterilmiştir.



Şekil 3.3 Guide

3.2.5 Real-Time Workshop ve Embedded Coder

Real-Time Workshop ile, Simulink ve Stateflow modellerinden C/C++ kodu üretilebilir. Gömülü sistem uygulamaları için özel ek ürünler sayesinde bu kodlar, doğrudan mikroişlemcilere aktarılabilir.

Kod, tamamen otomatik olarak, Real Time Workshop ürünü tarafından üretilir. Yaratılan kod, ANSI/ISO standartlarına uygun olup, bu sayede herhangi bir mikroişlemci veya gerçek zamanlı işletim sistemi üzerinde kullanılabilir.

Real Time Workshop, kodu yaratırken, kodun koşturulacağı hedef platforma göre eniyilemeler ve ayarlamalar yapar. Hedef olarak tek görevli/çok görevli çalışan

gerçek zamanlı işletim sistemleri seçilebileceği gibi, PC tabanlı donanımlar veya çeşitli endüstriyel bilgisayarlar da seçilebilir.

Real Time Workshop'ın temel özellikleri şu şekilde özetlenebilir:

- Ayrık ve/veya sürekli modellerden ANSI/ISO C uyumlu kod oluşturma
- Simulink'in Veri Sözlüğünü kullanarak, değişkenleri belirleme; tamsayı, kayan noktalı veya sabit noktalı veri desteği
- Tek görevli, çok görevli ve asenkron modellerden kod yaratabilme
- Çeşitli kod eniyileme seçenekleri
- Simulink modeline eklenen, önceden yazılmış C/C++ kodlarını da otomatik koda ekleyebilme
- Simulink içinde veya haricinde, kod üzerinde parametre değişimi ve işaret izleme yapabilme

Real Time Workshop'a eklenen Embedded Coder ürünü, gömülü işlemcilere özel eniyilenmiş kod yaratır. Eniyileme seçenekleri, işlemcinin özelliğine göre; bellek kullanımı, hesaplama hızı, veri türleri ve okunabilirlik gibi özellikleri içerir. Böylece yaratılan kod, elde yazılmış profesyonel kod ile kıyaslanabilir düzeyde verimli, eniyileyebilir ve okunabilir şekle dönüşmüş olur. Simulink'in kod üretme araçları, yaratılan kodu doğrudan doğruya gömülü işlemcinin belleğine yükleyebilecek araçlar içerir. Bu süreçte, Real Time Workshop ve Embedded Coder, C kodunu seçilen gömülü işlemciye özel üretirler (Figes).

3.2.6 Linux Soft Real-Time Linux Target

Simulink modellerini Real-Time Workshop kullanarak Linux işletim sistemi tabanlı donanımlara aktarılması için yazılmış hedef kütüphanesidir.

3.2.7 CygWin

CygWin, Microsoft Windows işletim sistemi üzerinde çalışan, açık kaynak kodlu, bir UNIX simülatörüdür. CygWin'in asıl amacı Unix türevi sistemlerde yer alan yazılımların Windows işletim sisteminde çalışmasını sağlamaktır.

3.2.8 Linux Cross Compiler Araçları

Linux Cross Compiler Araçları, ARM işlemci tabanlı Linux işletim sistemi üzerinden çalıştırılacak kodların başka bir bilgisayar üzerinde kod geliştirilmesine ve derlenmesine imkan sağlamaktadır.

BÖLÜM 4

MODELLEME

4.1 Genel Sistem Tanımı

Model Öngörülü Denetim Algoritması kullanılarak kontrol edilecek sistem bir takometreli doğru akım motordan meydana gelmektedir. Bölümümüzün Kontrol ve Kumanda Sistemleri Laboratuvarında bulunan Digiac 1750 kontrol deney kiti bir çok denetim algoritmasının test edilebilmesinde kullanılmaktadır. MPC için farklı donanım kullanan bir tez (Ergen, S., 2009) ile modelleme için grup çalışması yapılarak bu ögenin transfer fonksiyonu belirlenmiştir. Bu öğenin transfer fonksiyonunun belirlenmesi tezin katkı anlamında dışında olup zaten birçok durumda üreticinin katalogunda transfer fonksiyonu bilgisi yer almaktadır.

4.2 Motor Zaman Sabitinin Belirlenmesi

Zaman sabiti bir ölçü kare darbe üretilip buna karşılık motorun verdiği tepkinin ölçülmesi ile elde edilecektir.

Ölçüm Prosedürü:

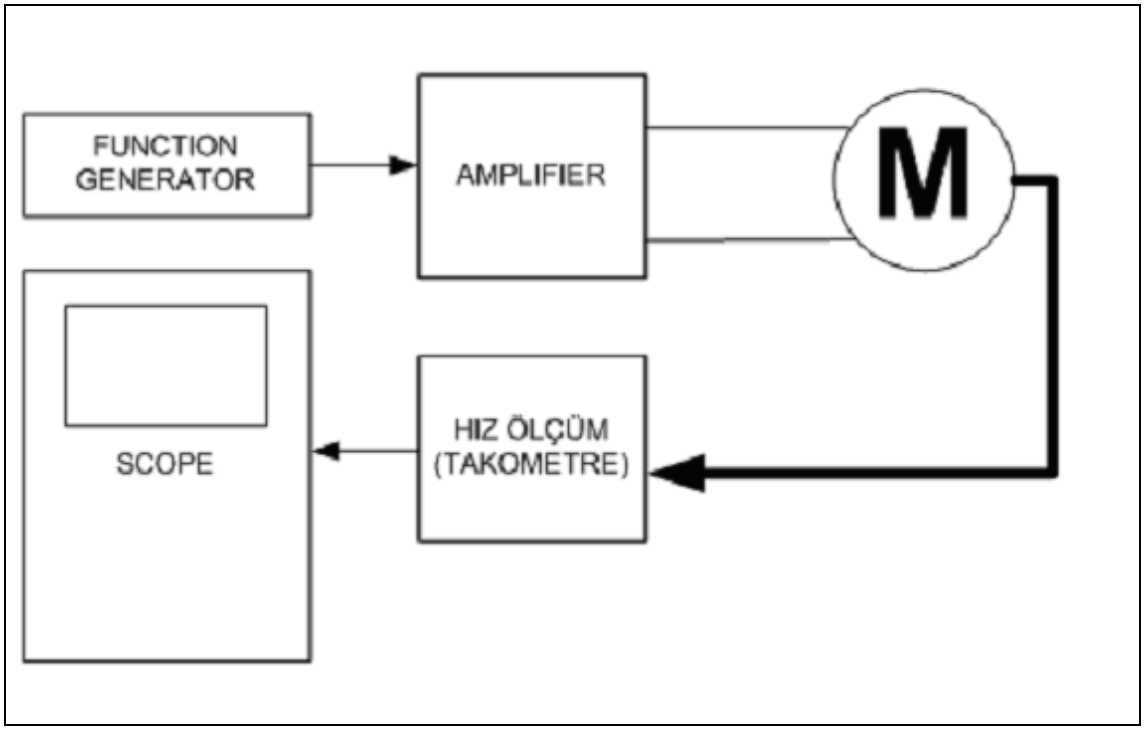
- Motor Sisteminin bağlantılarını Şekil 4.1'deki gibi yapıldı.
- Function Generator ile kare dalga üretip motor hızlanma ve yavaşlama sürelerini ölçüldü.

Doğru akım motor sistemini doğrusal olarak modellemiş olduk. Ancak gerçekte sürtünme nedeniyle motor sistemi doğrusal değildir. Bu yüzden ölçülen değerler:

τ_1: 0,2 s

τ_2: 0.0196 s

şeklindedir.



Şekil 4.1 Motor Sisteminin Bağlantı Şeması

4.3 Motor Dinamiği

Genel DC motor transfer fonksiyonu

$$G(s) = \frac{w(s)}{Va(s)} = \frac{\frac{1}{K_b}}{(s\tau_1 + 1)(s\tau_2 + 1)}$$

şeklindedir.

Tablo 1 Motor Değerleri

Paremetreler		
Sembol	Değer (SI Units)	Tanım
Kb	10/2400 v/rpm	Back emf Sabiti
$\tau 1$	0.2 s	Mekanik Zaman Sabiti
$\tau 2$	19.6 ms	Elektriksel Zaman Sabiti

Tabloda belirtilen değerleri girdikten sonra sistemin transfer fonksiyonu;

$$G(s) = \frac{0.24}{(0.0196s + 1)(0.2s + 1)} = \frac{51.02}{(s + 51.02)(s + 5)}$$

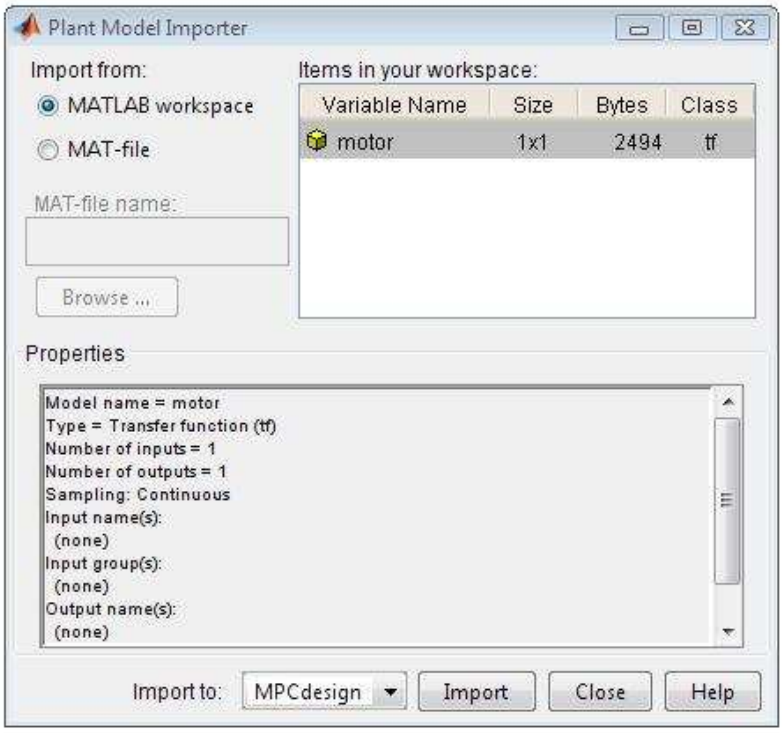
şeklinde olur. Motor transfer fonksiyonunu kullanarak MATLAB'ta oluşturulan süreç modelinin transfer fonksiyonunu tezle birlikte verilen CD' de yer almaktadır.

4.4 Model Öngörülü Deneteleyici Oluşturulması

Model Öngörülü Denetleyici MATLAB/Model Predictive Toolbox'ta aşağıdaki adımlarla gerçekleştirilir.

- Model Predictive Control Toolbox tasarım aracının çağrılması

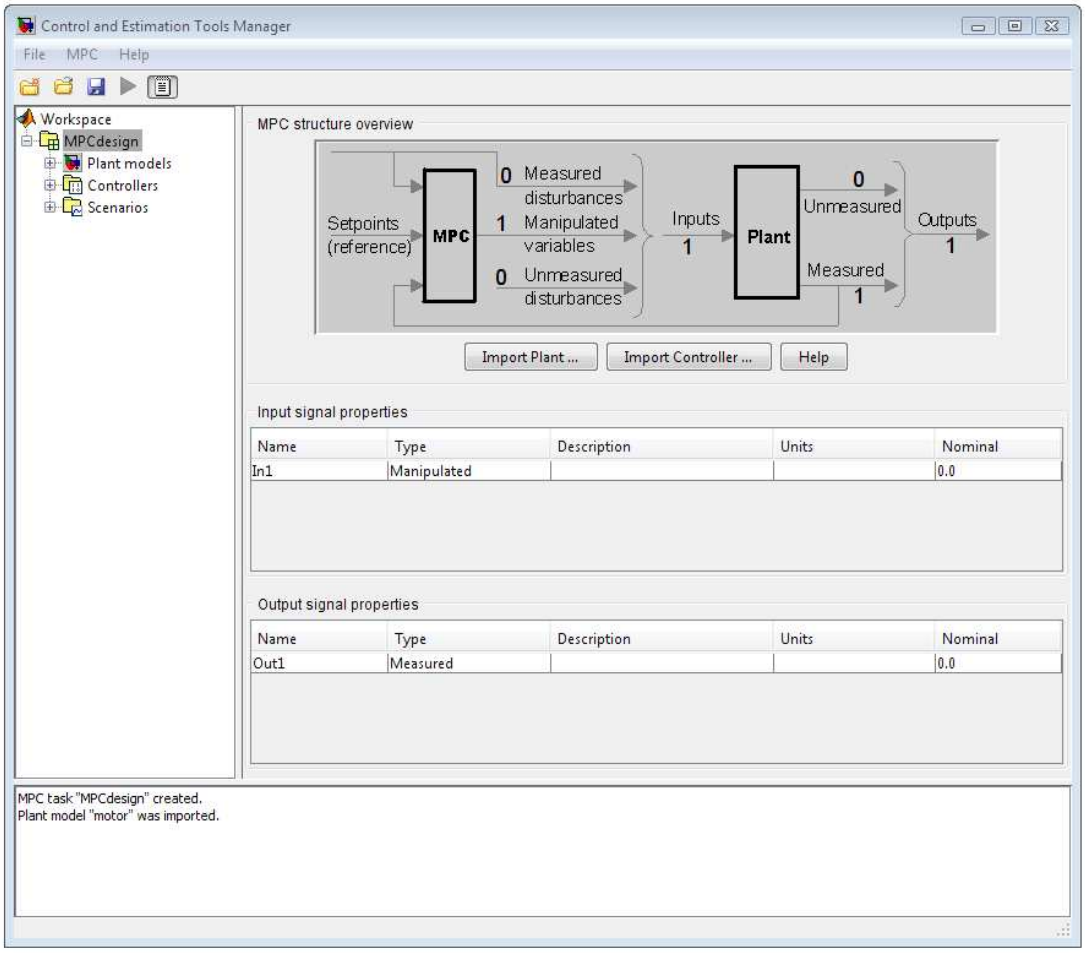
MATLAB komut isteminde "mpctool" yazılarak tasarım aracını çağrılır.



Şekil 4.2 Süreç Modeli Yükleme

- Süreç modelinin yüklenmesi

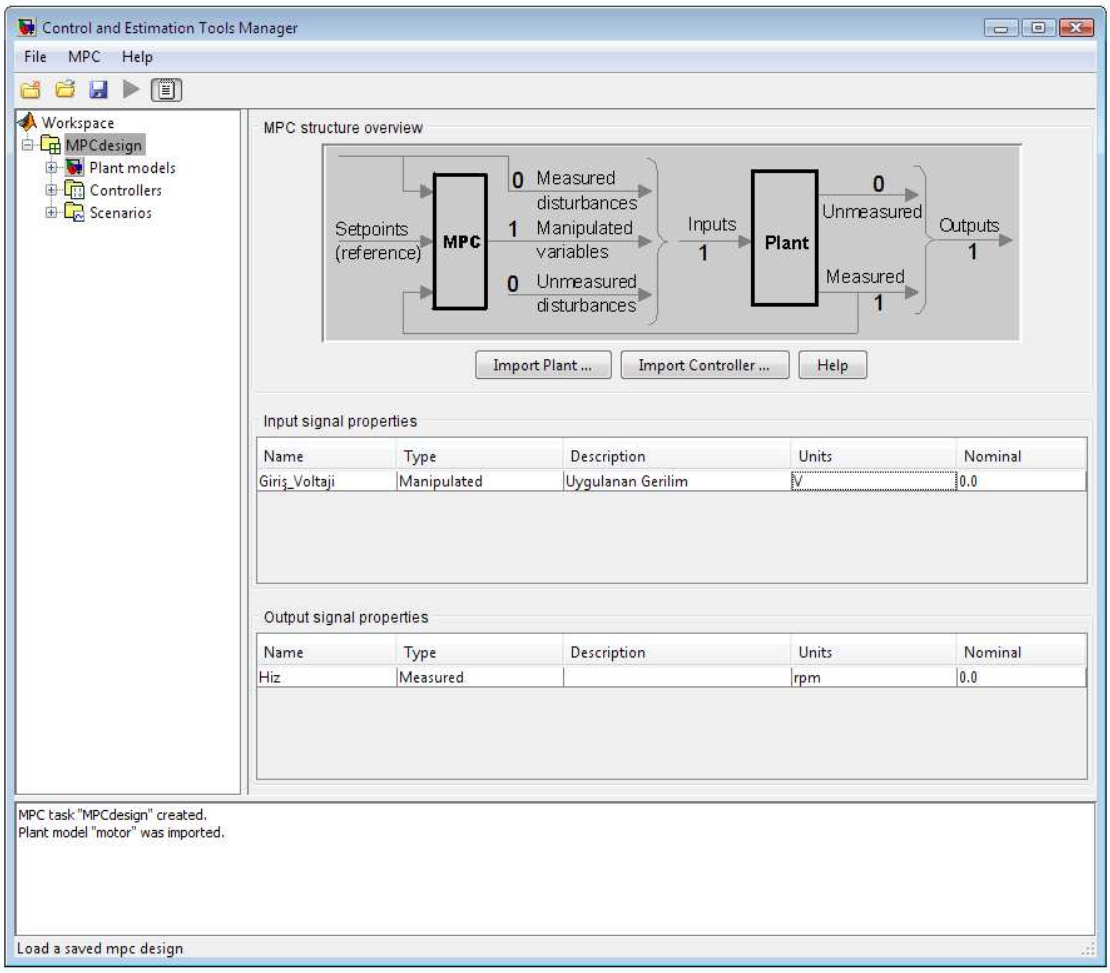
Model Predictive Control Toolbox' a çeşitli formatlarda süreç modelini tanıtmak mümkündür. Tezde söz konusu olan tek giriş tek çıkış doğru akım motor için en doğal modelleme transfer fonksiyonu olup süreç Model Predictive Toolbox'a bu formatta tanıtılmıştır. Oluşturduğumuz süreç modelinin transfer fonksiyonunu Şekil 4.2 te görüldüğü gibi Model Predictive Control Toolbox'a yüklenir. Model yüklendiği zaman Model Predictive Toolbox tasarım aracının ana penceresi Şekil 4.3'da göründüğü gibi ortaya çıkar.



Şekil 4.3 MPC Tasarım Aracı

- Sinyal Özelliklerinin Tanımlanması

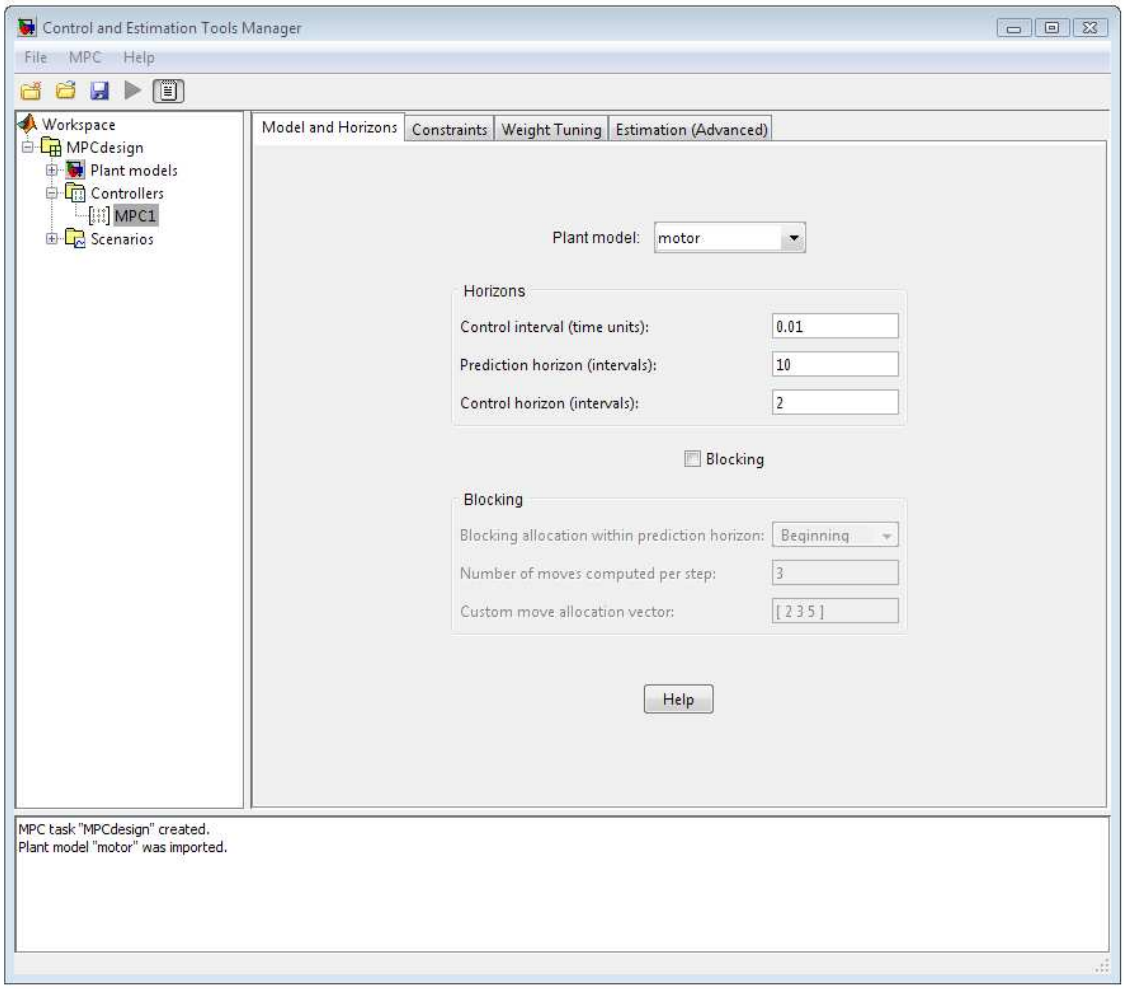
Model yüklendikten sonra tasarım aracı giriş çıkış sinyallerini varsayılan şekilde In1, Out1 isimleriyle yapılandırır. Bu giriş ve çıkış sinyalleri anlamlı şekilde tekrardan tanımlanır. Giriş ve çıkış sinyalleri "Giris_Voltaji" ,"Hiz" olarak adlandırdık. Description (Açıklama) ve Units (Birim) sütunları varsayılan olarak sıfır olacaktır. Bütün değişiklikleri yaptıktan sonra sistem aşağıdaki Şekil 4.4'deki gibi olacaktır.



Şekil 4.4 Sinyal Özelliklerinin Tanımlanması

- Denetleyici Özelliklerinin Belirlenmesi

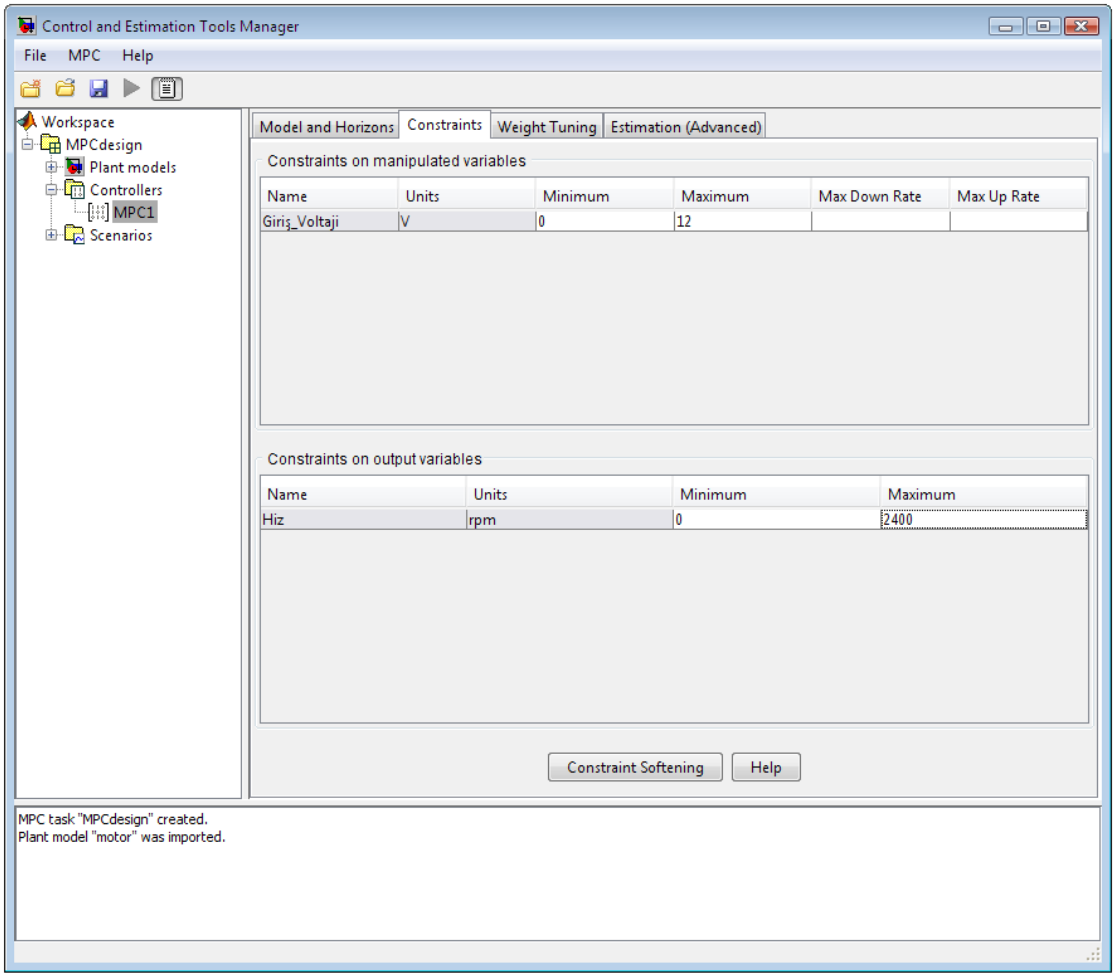
Sinyal özellikleri tanımlandıktan sonra Şekil 4.5'deki gibi Model Öngörülü Denetleyicinin özellikler penceresini açılıp "Control Interval" (Kontrol Aralığı) değeri 0,01 saniye, Prediction Horizon (Tahmin Ufku) değeri "10" ve Control Horizon (Kontrol Ufku) değeri "2" olarak değiştirilir.



Şekil 4.5 Model Öngörülü Denetleyicinin Özellikler Penceresi

- Kısıtlama Özelliklerinin Belirlenmesi

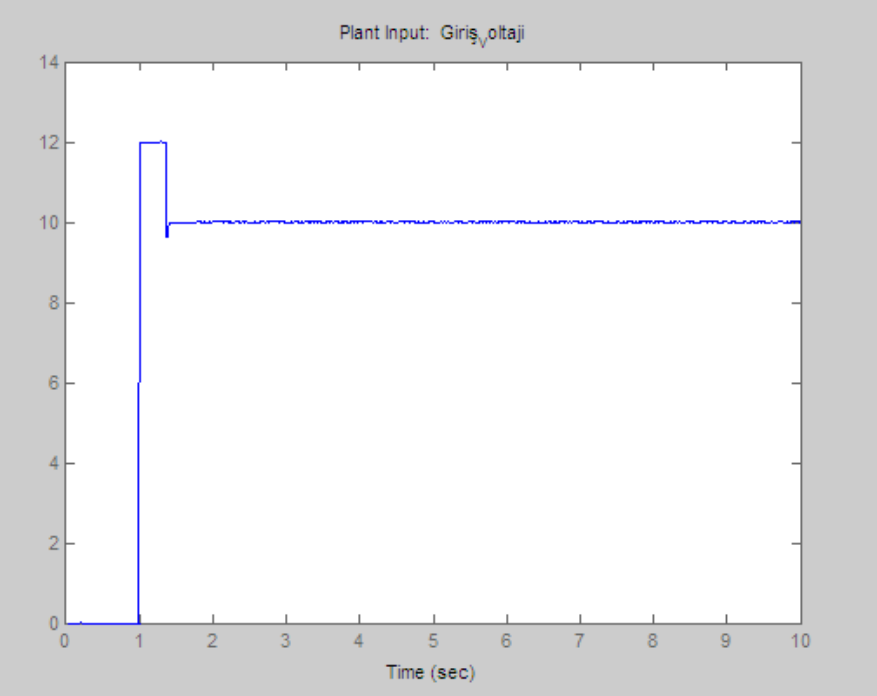
Model Öngörülü Denetleyicinin özelliklerini belirledikten sonra Constraints (Kısıtlamalar) özelliklerini değiştirmek için Constraints sekmesinde giriş voltajı için 0-12 volt ve çıkış hızını 0-2400 rpm olarak Şekil 4.6'deki gibi sınırlandırılır.



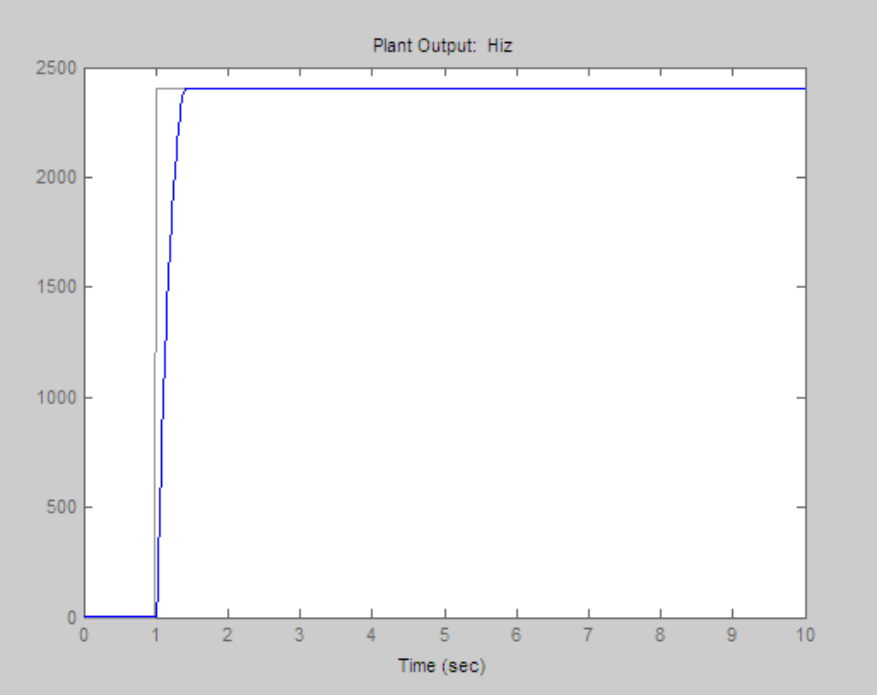
Şekil 4.6 Constraints (Kısıtlamalar) Özelliklerini Tanımlanması

- Model Öngörülü Denetleyecinin Senaryolarla Test Edilmesi

Scenarious (Seneryo) sekmesinde oluşturulan süreç step, ramp, continuous vb. şekilde simüle edilebilir. Şekil 4.7 ve 4.8'de gösterilen örnek giriş ve çıkış grafiklerinde mavi çizgiler çıkış sinyallerini gri çizgiler ise setpoint değerlerini ifade etmektedir.



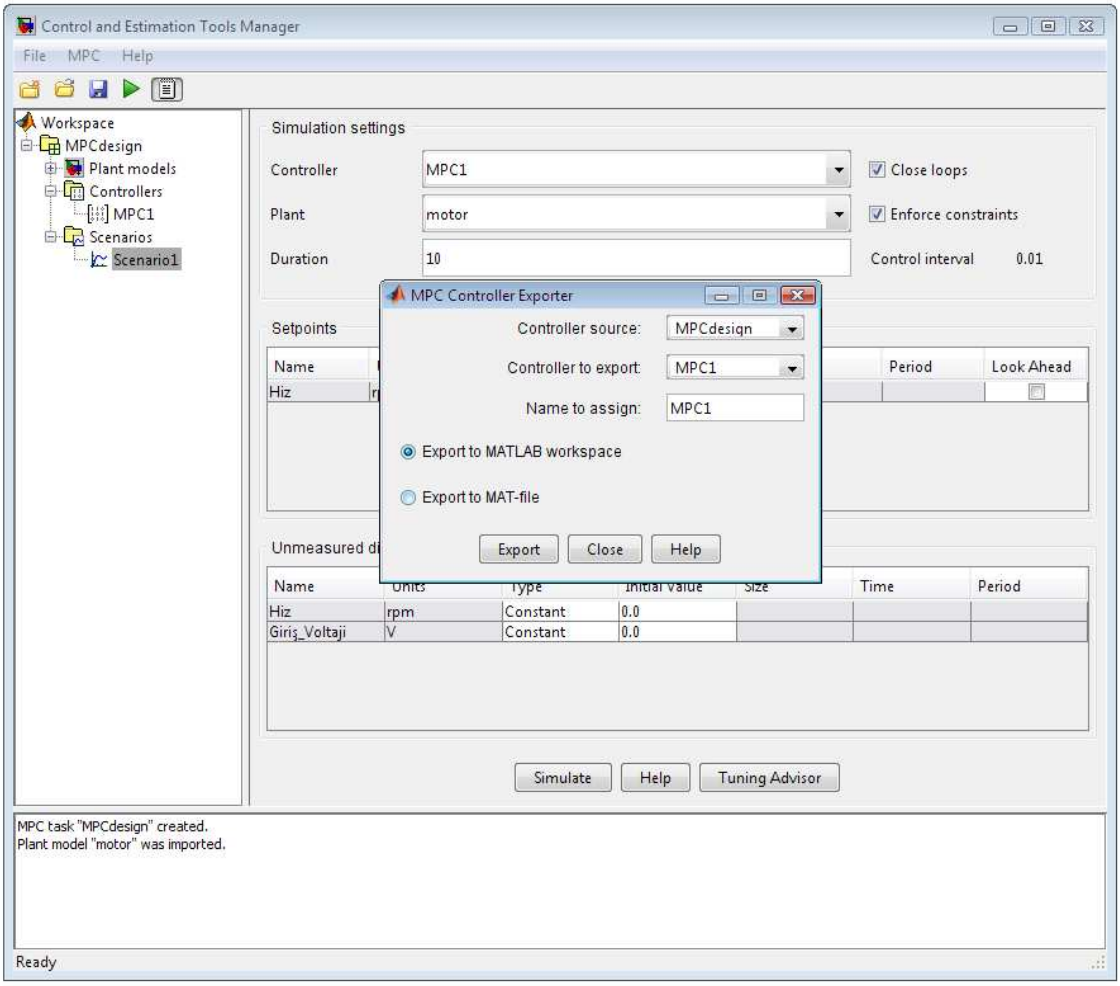
Şekil 4.7 Süreç Giriş Grafiği



Şekil 4.8 Süreç Çıkış Grafiği

- Model Öngörülü Denetleyicinin Çalışma Alanına Aktarılması

Sonuç olarak oluşturmuş olan Model Öngörülü Denetleyici Şekil 4.9'da görüldüğü gibi çalışma alanına aktarılır.



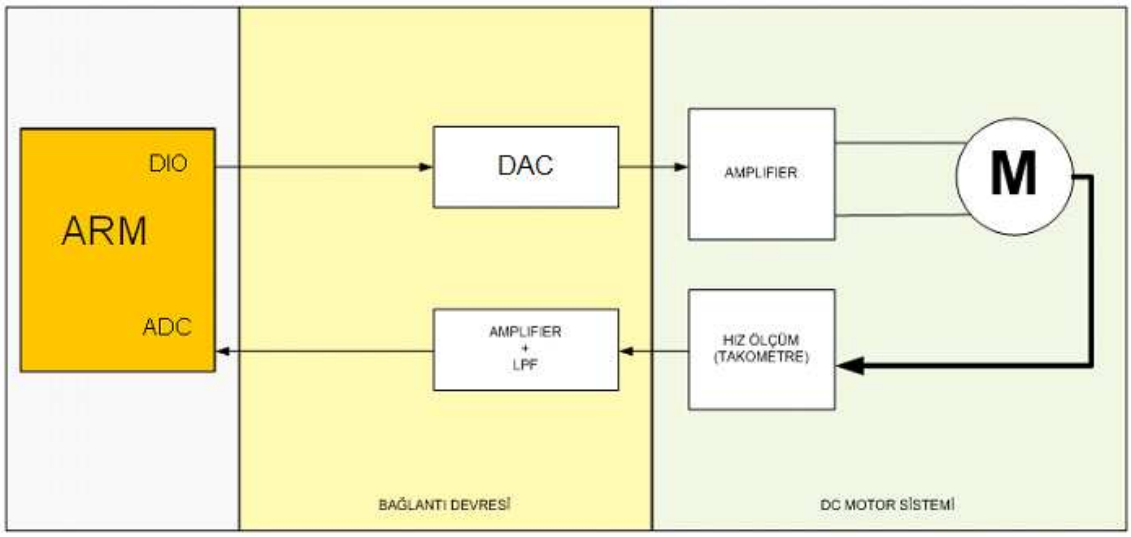
Şekil 4.9 MPC Dışa Aktarımı

BÖLÜM 5

UYGULAMA

5.1 Elektriksel Bağlantı

Uygulama Şekil 5.1'deki gibi TS-7250 ARM kartı, Bağlantı Devresi ve DC Motor olmak üzere üç bölümden oluşmaktadır.



Şekil 5.4 Elektriksel Bağlantı Şeması

DAC karttan çıkan gerilim yükseltici yardımıyla yükseltilerek motor için gerekli olan 0-12 V arasında gerilimi üretmektedir. Motor sisteminde takometreden gelen hız değeri analog 0-12 V arasında olacaktır. Takometreden ölçülen gerilim yükseltici ve Alçak Geçirgen Süzgeç (low pass filter) yardımı ile 0-3 V değerlerine düşürülerek ARM kartın ADC girişine bağlantısı yapılmaktadır.

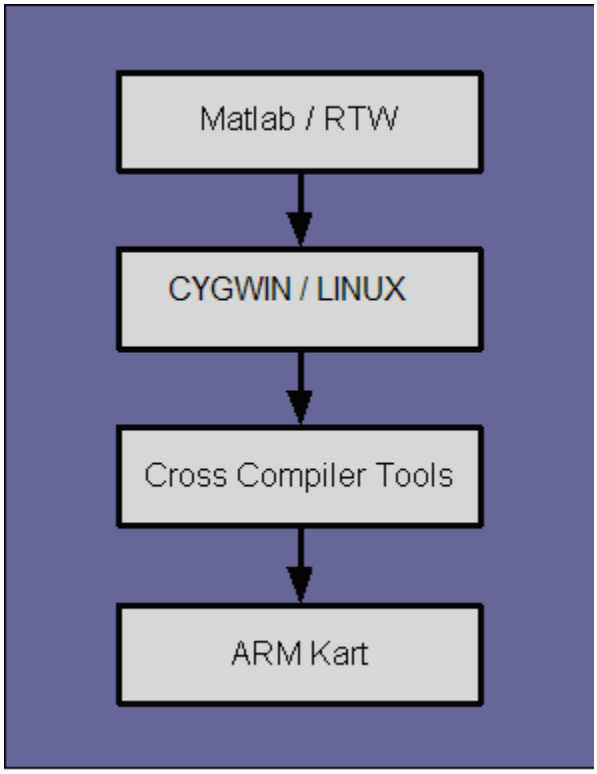
5.2 ARM Kartın Bağlantı Ayarlarının Yapılması

TS-7250 Arm işlemci tabanlı geliştime kartı flash'a önceden yüklenmiş bir Linux dağıtımıyla birlikte gelmektedir. TSLinux olarak adlandırılan bu dağıtıma yerel alan ağ kablosu yardımıyla ftp veya telnet prokolleri üzerinden veya seri bağlantı kablosuyla hyperterminal veya benzeri seri haberleşme yazılımları yardımıyla erişilebilmektedir.

5.3 Uygulamanın Genel Yapısı

Yeniden kullanılabilir ve bakım aşamasını kolaylaştıran yazılımların geliştirilmesi ve yazılım geliştirmede üretkenliğin artırılması, yazılım mühendisliği araştırmacılarının öncelikli hedeflerini oluşturmaktadır. Model tabanlı yazılım geliştirme, bu hedeflere ulaşılabilmesi için, yazılım geliştirmenin model geliştirme ve model dönüşümleri ile gerçekleştirilmesi fikrine dayanmaktadır.

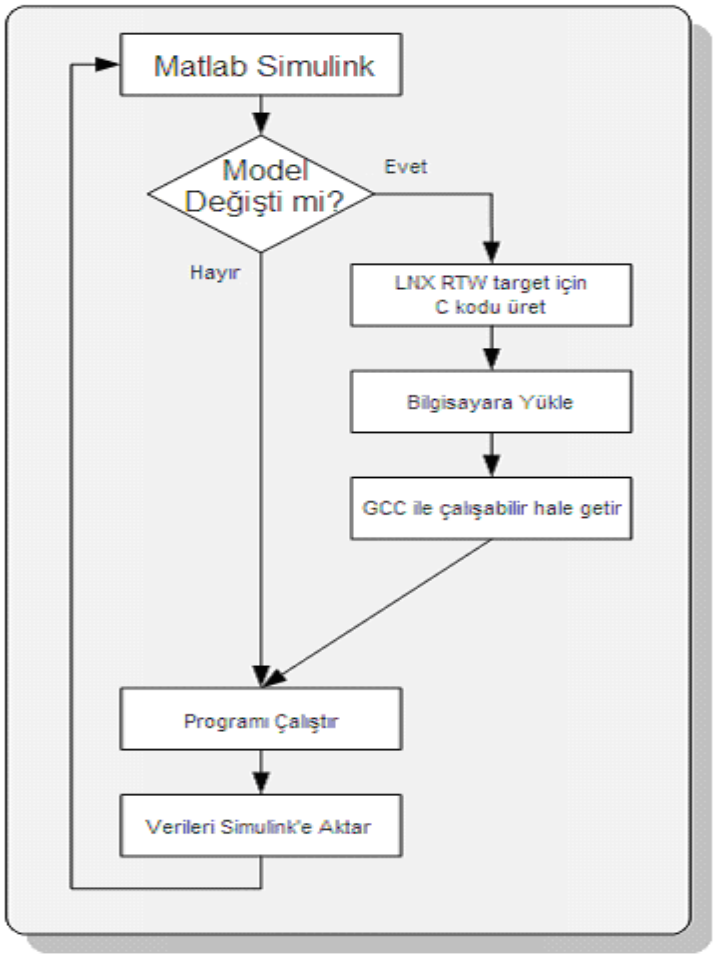
MATLAB/Simulink, denetim algoritmalarının blok diyagramlarla gösterilebilen benzetimi için model tabanlı bir geliştirme ortamıdır. RTW (Real Time Workshop), Simulink ortamında oluşturulan grafiksel blok diyagramından C kodlarını otomatik olarak üretmektedir. Bu kodlar CygWin'in ulaşabildiği bir klasörün altına kopyalanarak ARM tabanlı Linux dağıtımında çalışacak şekilde Cross Compiler araçları yardımıyla derlenir ve ftp protokolü kullanılarak oluşan çalışabilir dosya karta aktarılır. Aktarılan çalışabilir dosyaya çalışma yetkileri konsol veya telnet üzerinden verilerek kart üzerinde çalışabilir hale getirilir. Şekil 5.2 de yazılım geliştirme basamakları görülmektedir.



Şekil 5.2 Yazılım Geliştirme Basamakları

ARM için geliştirilmiş özel kütüphaneler olmadığı için işlemci kartı için gerekli fonksiyonları yerine getirecek bloklar Real Time Workshop'a bütünleşik olarak üretilmiştir. Üretilen bloklar ARM üzerinde yüklü bulunan Linux işletim sistemiyle birlikte çalışacak şekilde elle tekrardan düzenlenmektedir.

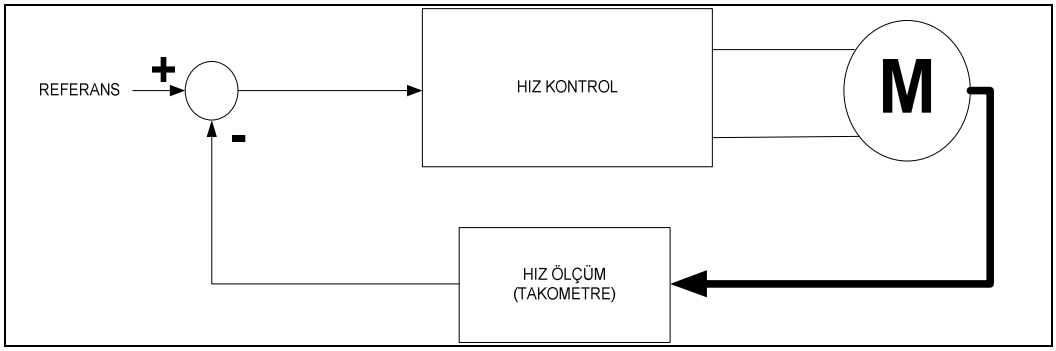
Blok diyagramlarla verilen ARM algoritmaları, Real Time Workshop (RTW), Linux Real Time Workshop Embedded Target, Cross Compile Tool (GCC v.b.) alt yazılımları ile Şekil 5.3'de görüldüğü gibi hedef Linux işletim sistemi yüklü ARM işlemci kartı için çalışır hale çevrilmekte, gerçek zamanda uygulanması ve tasarım doğrulaması çok hızlı bir şekilde yapılabilmektedir.



Şekil 5.3 Uygulamanın Genel Yapısı

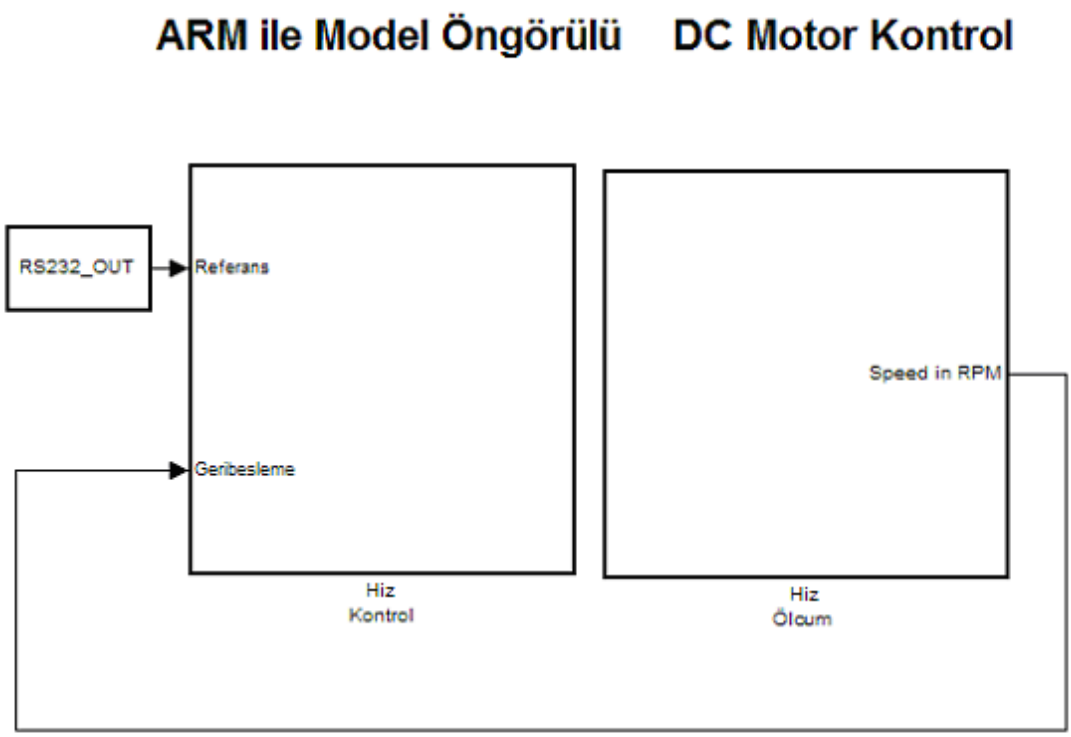
5.4 Sistemin Modellenmesi

Sistem modelinde, motor hedef hızı kullanıcı arayüzü aracılığı ile kullanıcı tarafından belirlenir ve sistemden alınan veriler grafiksel olarak gösterilir. Denetleyici takometreden alınan ve motora verilen gerilim değerlerini denetleyerek istenilen hızı korur. Denetim döngüsü Şekil 5.4'de gösterilmiştir.



Şekil 5.4 Denetim Döngüsü

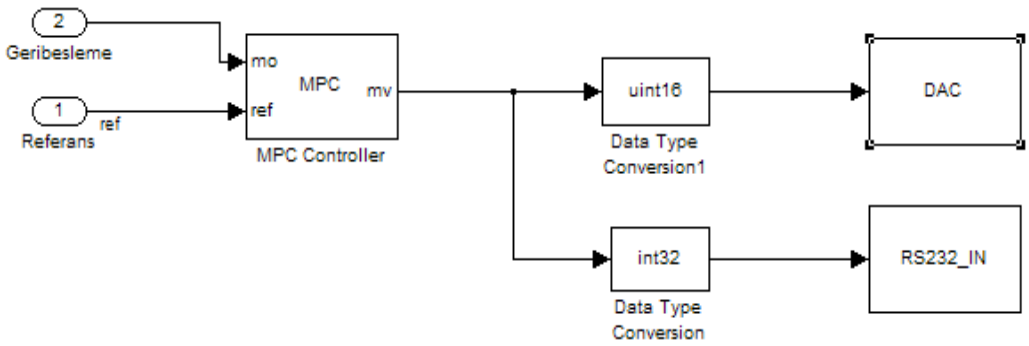
Şekil 5.5'de genel sistem modeli görülmektedir Sistem modeli "Hiz Kontrol" ve "Hiz Ölcum" adında iki alt modelden oluşmaktadır. Şekilde, kullanıcı arayüzü tarafından belirlenen yeni hız değerleri "RS232_OUT" blok aracılığıyla modele gönderilir. Bu blok seri haberleşmeyle ARM karta gönderilen verileri modele aktarmaya yarar. Hız Ölçüm alt modelinin çıkışı ile de Hız Kontrol alt modelinin geri besleme girişi arasında yapılan bağlantıyla sistem modeli oluşturulmuştur.



Şekil 5.5 Sistem Modeli

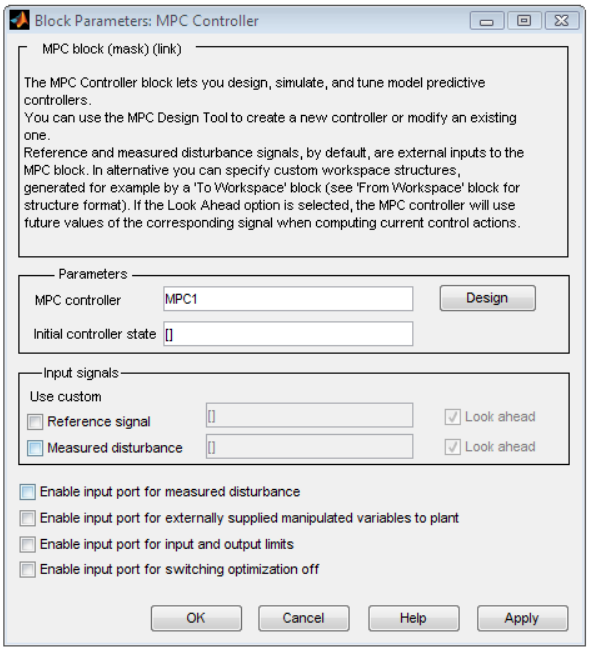
- **Hız Kontrol Alt Modeli:**

  Bu alt model, ölçülen hız ile istenilen hızı karşılaştırır ve motora buna göre gerilim uygulaması için DAC 'ye iletilir. Aynı zamanda iletilen değerlerin seri haberleşme yardımıyla bilgisayarda grafiksel gösterimi için RS232_IN ' ye iletilir. Şekil 5.6 'da hız kontrol alt modeli verilmektedir.



Şekil 5.6 Hız Kontrol Alt Modeli

Bu alt modelde en önemli blok olan MPC' nin yapılandırılması için blok çift tıklanarak Şekil 5.7'deki gibi özellikler penceresi açılıp daha önceden oluşturulmuş ve MATLAB çalışma alanına kaydedilmiş olan MPC denetleyicisinin ismi yazılıp "OK" butonuna basılarak Model Öngörülü Denetleyici eklenir.

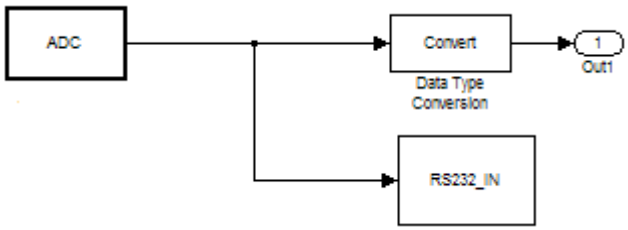


Şekil 5.7 MPC Blok Parametreleri

MPC Denetleyicisi için geri besleme ve referans olmak üzere iki adet giriş bloğu eklenmiştir. Buradaki birinci giriş Hız Ölçüm Alt Modelinden gelen veri olarak diğer giriş ise RS232_OUT bloğundan gelen istenilen hız değeri olarak ayarlanmıştır. MPC Denetleyicisinden çıkan veriyi uint16 ya dönüştürüp DAC ile bağlantısı yapılır. Kullanıcı arayüzünde DAC değerlerini görmek için RS232_IN bloğuyla bağlantı yapılır.

- **Hız Ölçüm Alt Modeli:**

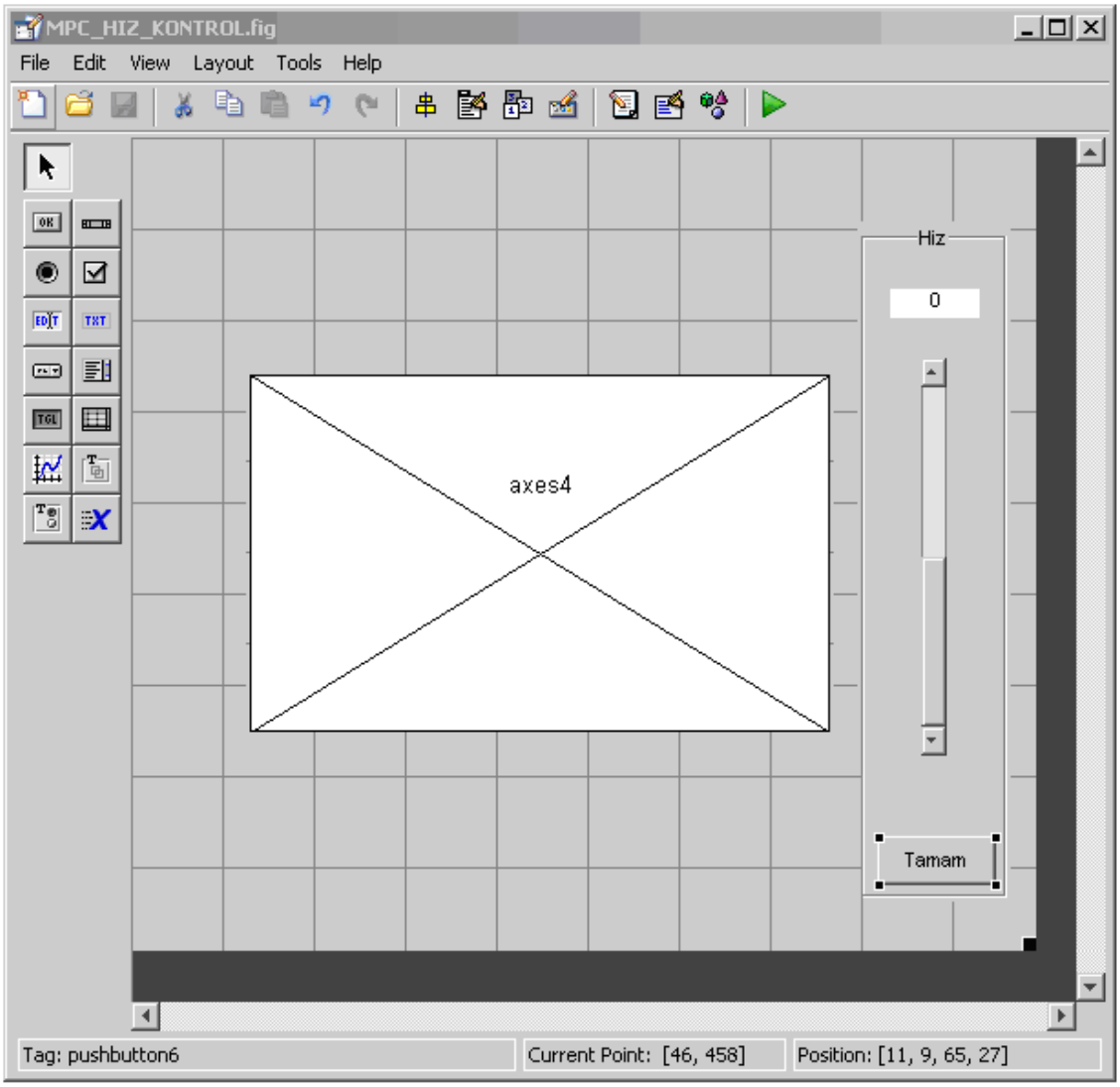
Bu blokta takometreden gelen hız ölçüm değerleri alınarak MPC Denetleyicisi için bir çıkış oluşturulur. Ayrıca RS232_IN bloğuyla bilgisayara ölçülen hız değerleri gönderilir. Şekil 5.8 'da hız ölçüm alt modeli verilmektedir.



Şekil 5.8 Hız Ölçüm Alt Modeli

5.6 Kullanıcı Arayüzü Tasarımı

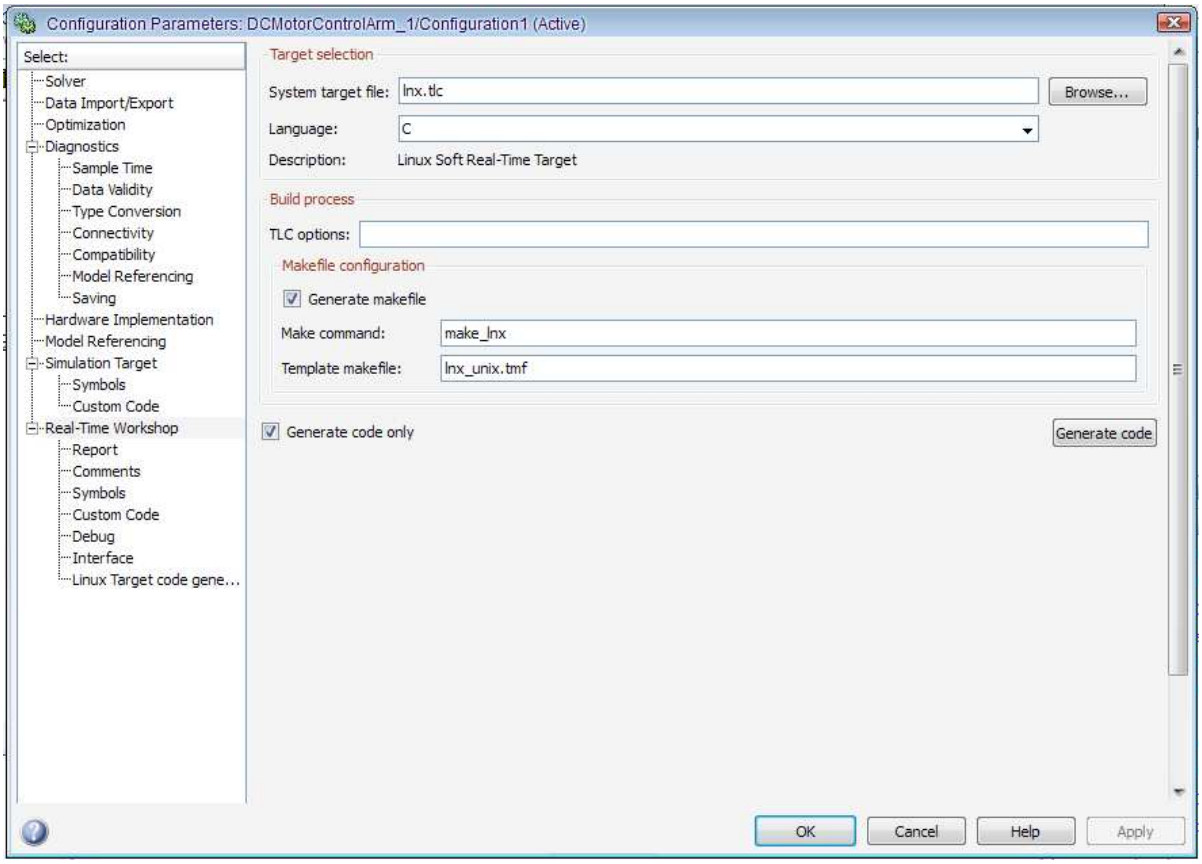
Şekil 5.9'te gösterildiği gibi kullanıcı arayüzü motor hızını gösterecek bir grafikten ve hızı ayarlamak için bir sürgüden oluşmaktadır. ARM kartı ve bilgisayar arasındaki bağlantıları sağlayacak gerekli kodlamalar yapıldıktan sonra kullanıcı arayüzü tasarımı tamamlanır.



Şekil 5.9 Kullanıcı Arayüzü

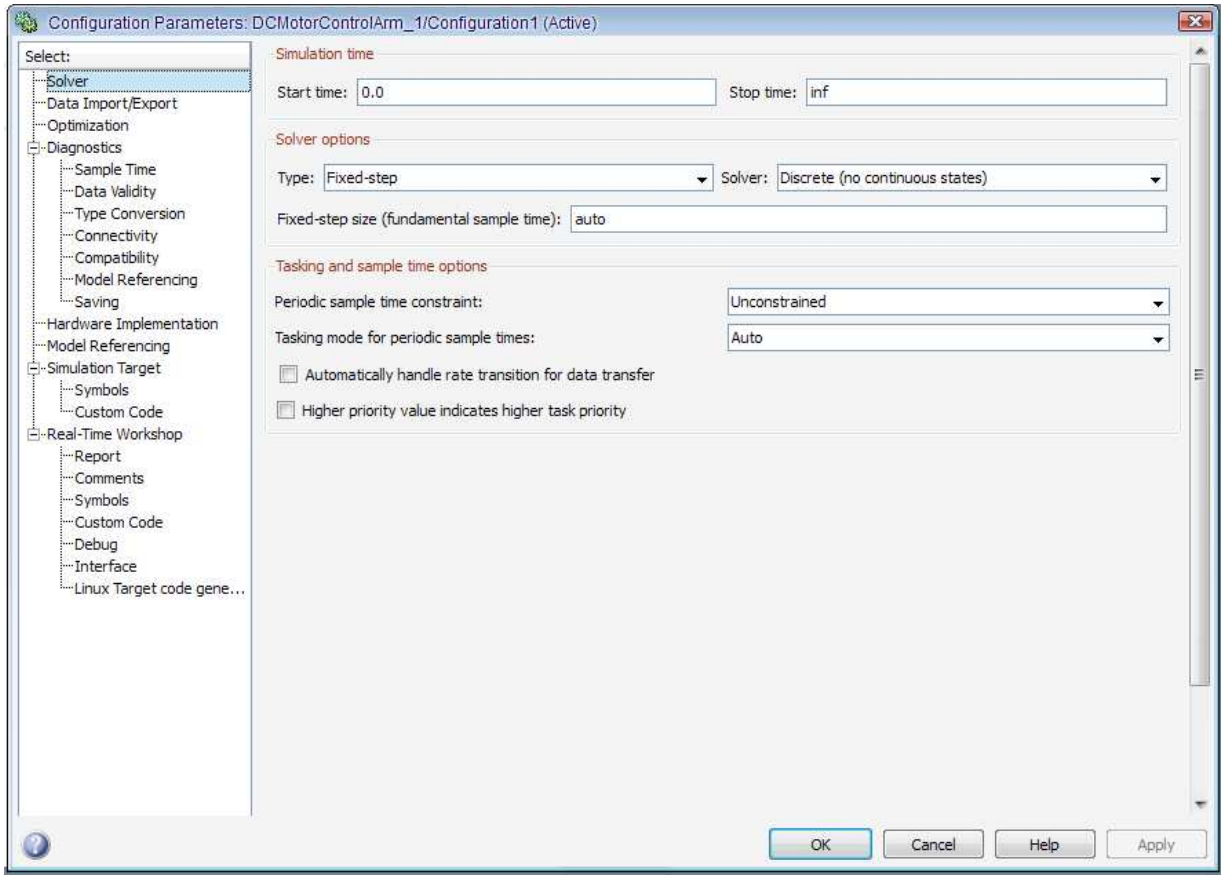
5.7 Gerçek Zamanlı Uygulama

Model oluşturulduktan sonra Simulink'te Tools menüsünden Real-Time Workshop - Options tıklanarak Şekil 5.10'teki özellikler penceresi açılır. Açılan pencerede "Target File" özelliği "Linux Soft Real Time Target" olarak seçilir.



Şekil 5.10 RTW Hedef Kart Seçimi

Sol menüden Solver'ı tıklanarak Şekil 5.25'teki gibi "Type" özelliği "Fixed-step" ve "Solver" özelliği "Discrete" olarak değiştirilir.



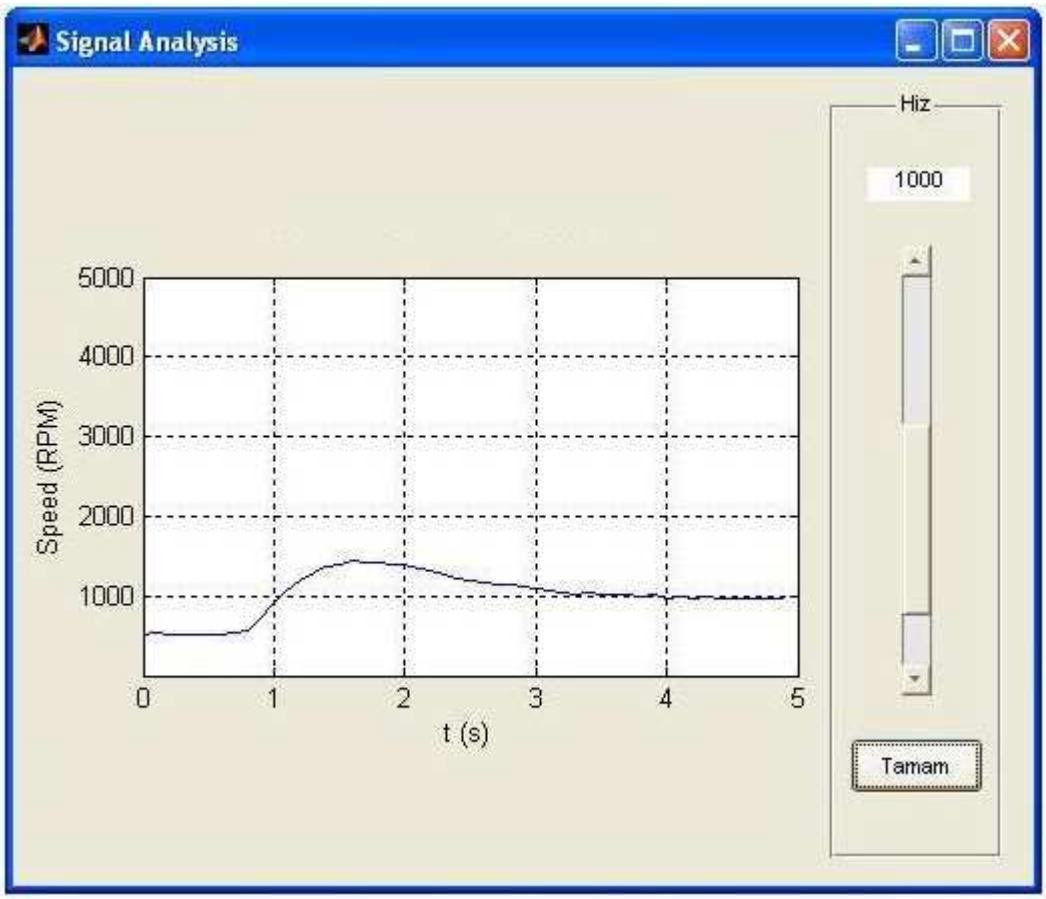
Şekil 5.11 Solver Özellikleri

Real-Time Workshop üzerinde gerekli düzenlemeleri yapıldıktan sonra "Simulink > Tools >Real-Time Workshop > Build Model" tıklanılarak oluşturulmuş olan model Real-Time Workshop, Embeded Coder ve Target Language Coder yardımı ile TS-7250 ARM işlemci geliştirme kartı için koda dönüştürülür. Üretilen bu kod CygWin ve Cross Compiler Araçları yardımıyla gerekli düzenlemeler yapıldıktan sonra derlenir ve ftp protokolü veya seri haberleşmeyle TS-7250 ARM karta yüklenerek Model Öngörülü hız kontrol denetimi ARM işlemci tabanlı geliştirme kartı üzerinde gerçekleştirilmiş olur.

5.8 Uygulamanın Çalıştırılması

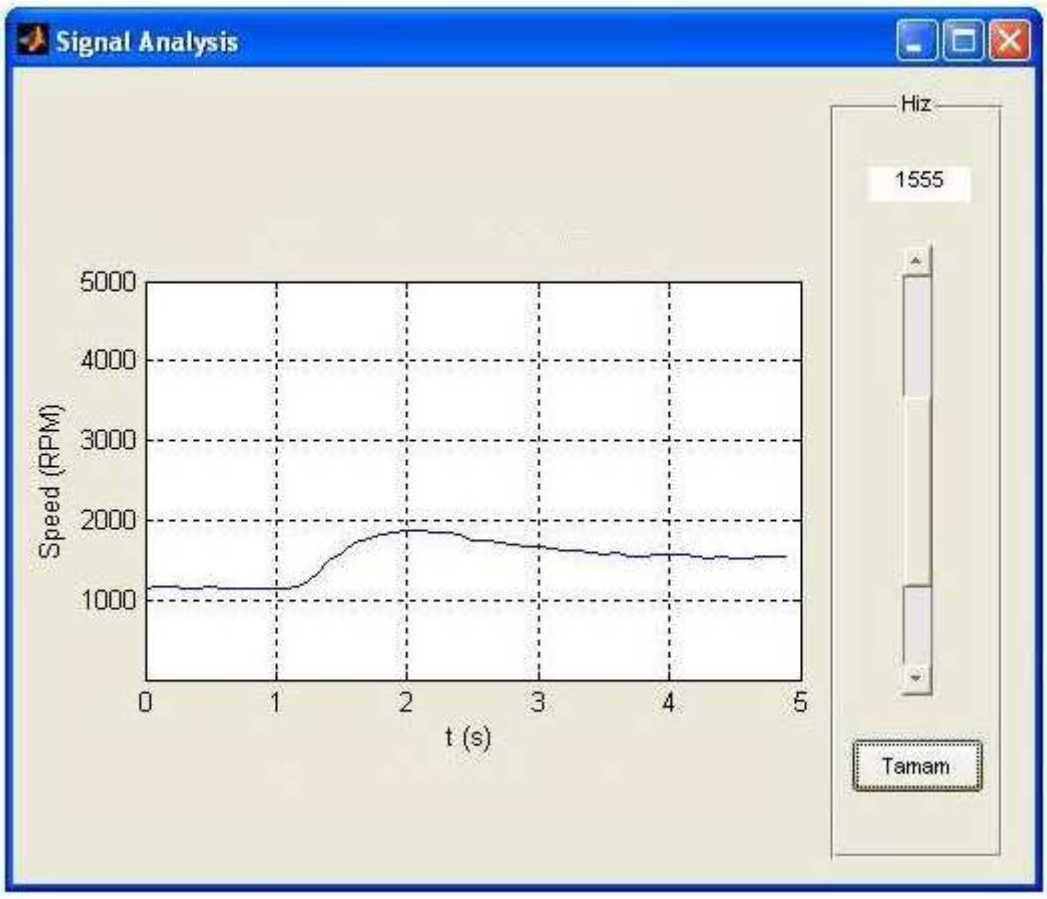
Uygulama çalıştırıldıktan sonra çalıştırılan kullanıcı arayüzü, motor hızı değerleri görülebilir ve sürgü kullanılarak motor hızı ayarlanabilir. Aşağıda çeşitli hız değerleri için örnekler görülmektedir.

Şekil 5.12'de sonuçları görüldüğü gibi motor hızı 520 rpm'den 1000 rpm'e çıkarılmıştır.



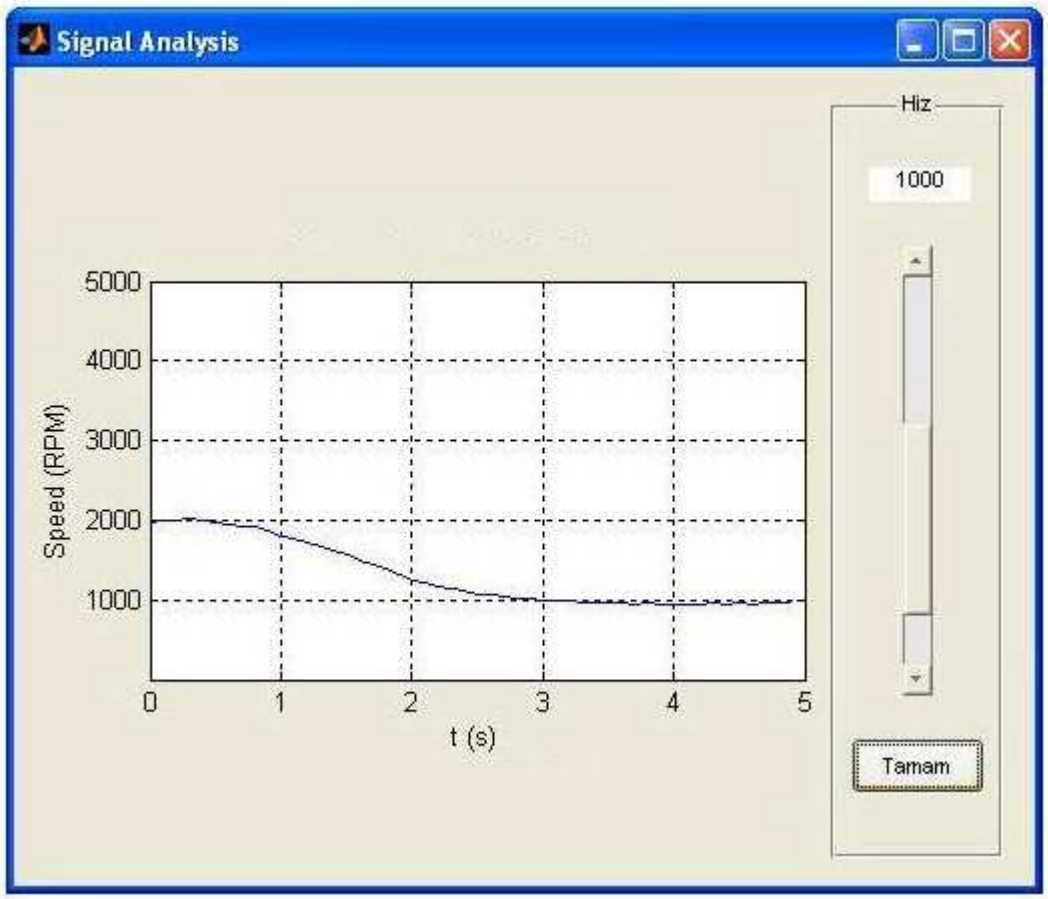
Şekil 5.12 520 rpm den 1000 rpm e çıkış

Şekil 5.13'de sonuçları görüldüğü gibi motor hızı 1155 rpm'den 1555 rpm'e çıkarılmıştır.



Şekil 5.13 1155 rpm den 1555 rpm e çıkış

Şekil 5.14'de sonuçları görüldüğü gibi motor hızı 2000 rpm'den 1000 rpm'e düşürülmüştür.



Şekil 5.14 2000 rpm den 1000e düşüş

Eğer mekanik ve elektriksel zaman sabitini Tablo2'deki gibi yüzde beş daha düşük ölçmüş olsaydık.

$\tau 1$	0.19 s	Mekanik Zaman Sabiti
$\tau 2$	18.6 ms	Elektriksel Zaman Sabiti

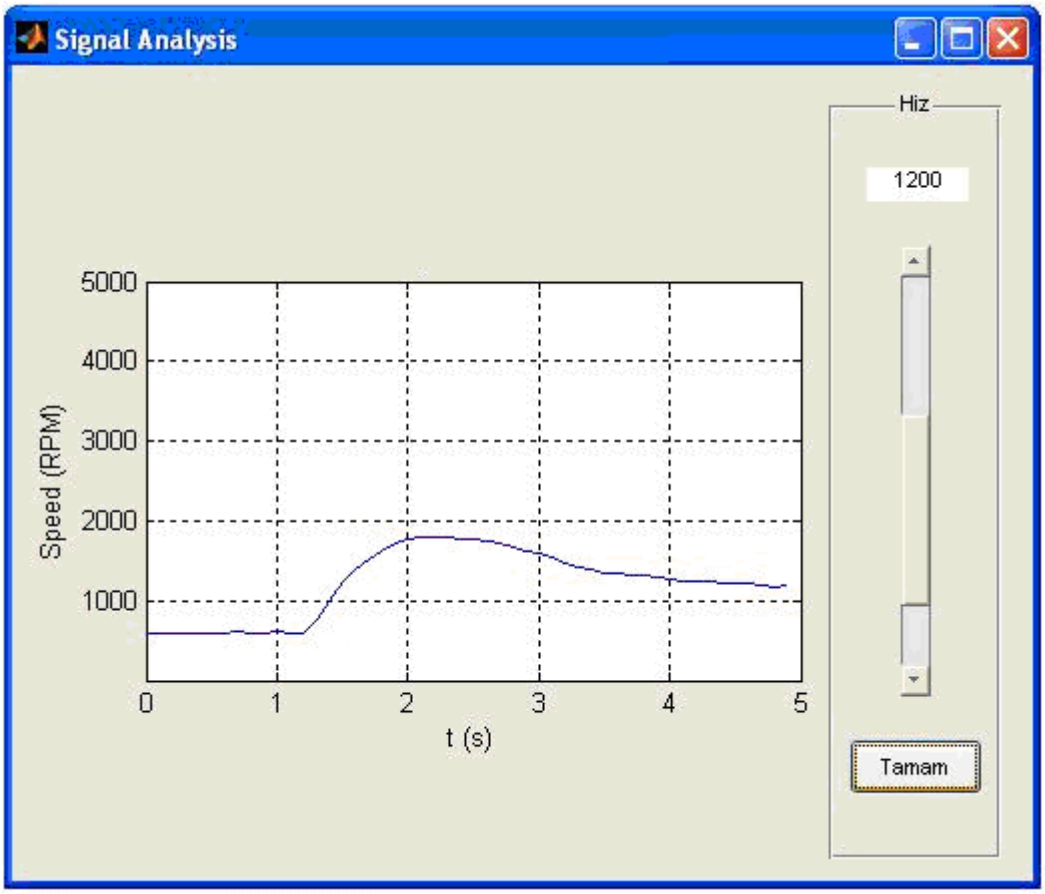
Tablo 2 Değişen Motor Değerleri

Tabloda belirtilen değerleri girdikten sonra sistemin transfer fonksiyonu;

$$G(s) = \frac{0.24}{(0.0186s + 1)(0.19s + 1)} = \frac{67.91}{(s + 53.76)(s + 5.26)}$$

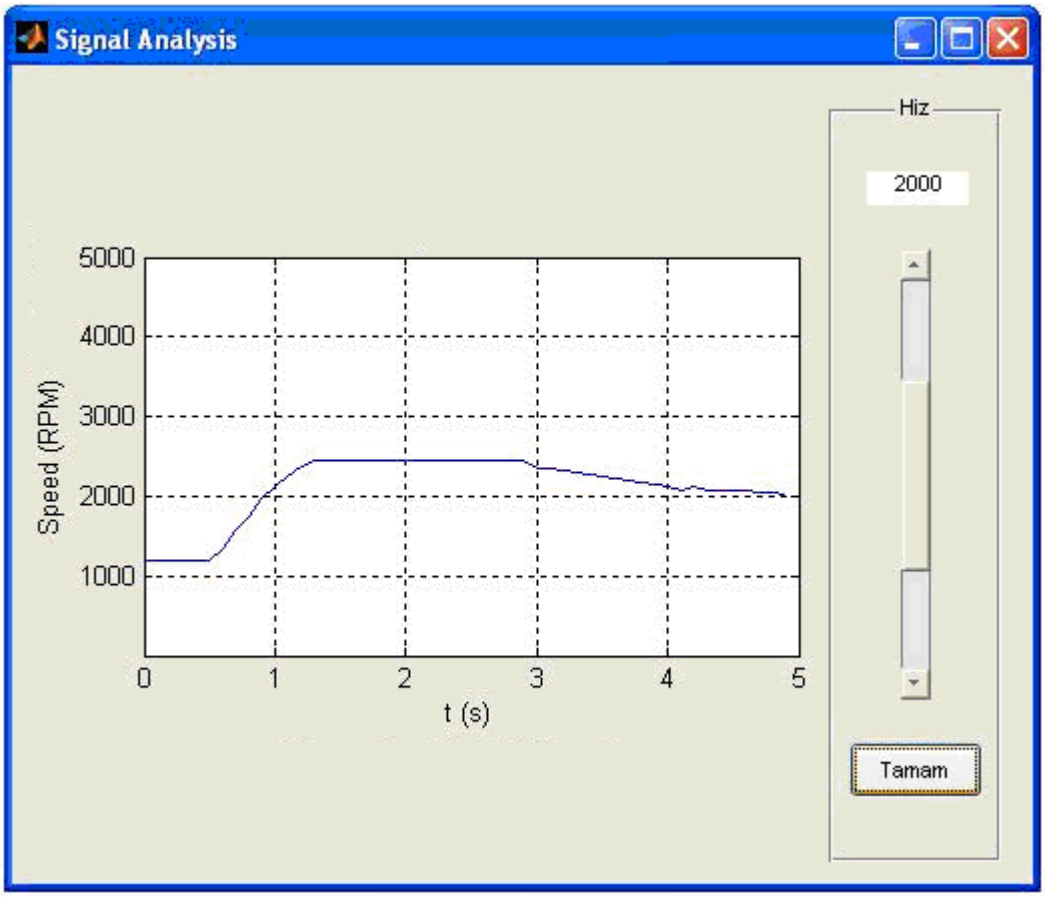
Bu transfer fonksiyon bilgisini MPC toolboxa tekrardan girince ve yukardaki adımları tekrarlayıp uygulamayı çalıştırınca aşağıdaki grafikler elde edilmektedir.

Şekil 5.15'de sonuçları görüldüğü gibi motor hızı 600 rpm'den 1200 rpm'e çıkarılmıştır.



Şekil 5.15 600 rpm den 1200 rpm e çıkış

Şekil 5.16'da sonuçları görüldüğü gibi motor hızı 1200 rpm'den 2000 rpm'e çıkarılmıştır.



Şekil 5.16 1200 rpm den 2000 rpm e çıkış

Görüldüğü gibi sistem modelini yüzde beş kadar hatalı elde edip MPC Toolbox'a bildirmemize rağmen geri beslemeli MPC konfigürasyonumuz hala istenen yeni hıza inme/çıkma görevini yerine getirmektedir.

BÖLÜM 6

SONUÇ VE TARTIŞMA

Yapılan çalışma Model Öngörülü Denetleyicinin ARM işlmeci tabanlı geliştirme kartı kullanılarak gerçeklenmesidir. Bu amaçla bir hız kontrol uygulaması yapılmıştır.

Yapılan çalışmada öncelikle Model öngörülü Kontrol incelenmiş üstünlükleri ve eksik yönleri belirlenmiştir. Takometreli bir doğru akım motordan oluşan tek giriş ve tek çıkışlı süreç matematiksel olarak modellenmiş ve bu süreç için bir Model Öngörülü Denetleyici MATLAB, Simulink ve Model Predictive Control Toolbox kullanılarak oluşturulmuştur. Oluşturulan Simulink modeli Real Time Workshop (RTW) ile C koduna dönüştürülerek CygWin ve Linux Cross Compiler Araçları kullanılarak diğer destek dosyalarıyla birlikte derlenmiş ve ftp veya seri haberleşme protokolleri yardımıyla ARM işlemci tabanlı geliştirme kartına yüklenmiştir. Yüklenen program çalıştırılarak Model Öngörülü Denetim uygulaması ARM üzerinde başarılı olarak gerçekleştirilmiştir. Gerçekleştirilen bu uygulama model tabanlı yazılım geliştirme araçları kullanılarak esnek, tekrar kullanılabilir ve geliştirmeye açık olarak tasarlanmıştır. Tüm işlemleri izleyebilme olanağı sağlayan bu uygulamada kullanıcı arayüzü ile sürecin denetimi gerçek zamanlı olarak sağlanmıştır.

Öngörü için sürecin modeli kullanılmakta olduğu için model MPC denetleyicinin vazgeçilmez bir bileşenidir. Modellerde oluşabilecek hatalar dolayısıyla geri beslemelerden yararlanılarak öngörü daha güvenli hale getirilmektedir.

Uygun yazılımlar kullanılarak model tanıtımı ve hedef belirlenmesinin ardından uzman olmayan bir mühendisin dahi bu yöntemle sistem denetiminin sıradan bir mühendislik işi olduğunu bu çalışma göstermektedir.

KAYNAKLAR DİZİNİ

ARM Mimarisi, 2009, Wikipedia, http://tr.wikipedia.org/wiki/ARM_mimarisi

Camacho, F.E, and Bordons, C., 2004, Model Predictive Control, Springer, New York

Digital Analog Converter (8 bit), 2009, ikalogic, http://ikalogic.com/dac08.php

EP9302 Datasheet, 2004, Cirrus Logic, Texas

Ergen S., 2009, Model Öngörülü Denetimin Bir Sistemde DSP ile Gerçeklenmesi , Yüksek Lisans Tezi, Osmangazi Üniversitesi Fen Bilimleri Enstitüsü

Kod Üretme Araçları, 2009, Figes, http://www.figes.com.tr/urunler/simulink/uygulama2.simulink.php

Liuping Wang, 2009, Model Predictive Control System Design and Implementation Using MATLAB, Springer

M. Nikolau, 2009, Model Predictive Controllers: A Critical Synthesis of Theory and Industrial Needs, Chemical Eng. Dept. , University of Houston, http://www.chee.uh.edu/faculty/nikolaou/MPCtheory/revised.pdf

MATLAB, 2009, Wikipedia, http://tr.wikipedia.org/wiki/MATLAB

Model predictive Control Toolbox, 2009, Mathworks, http://www.mathworks.com/products/mpc/

Model Predictive Control, 2009, Wikipedia, http://en.wikipedia.org/wiki/Model_predictive_control

Simulink, 2009, Figes, http://www.figes.com.tr/urunler/simulink/simulink.php

EKLER

Ek.1 Model Öngörülü Kontrol Yazılımı

Ek.1 Model Öngörülü Kontrol Yazılımı

```
// adc.c – Analog Digital Converter  External

#include "adc.h"

int adc_one()
{
	volatile unsigned short * complete;
	volatile unsigned char * lsb, * msb, * control;
	int res;
	int fd = open("/dev/mem", O_RDWR);
	assert(fd != -1);
	lsb = control = (unsigned char *)mmap(0,
getpagesize(),PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0x10c00000);
	msb = lsb + 1;
	complete = (unsigned short *)mmap(0,
getpagesize(),PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0x10800000);
	// Initiate conversion, channel #1, unipolar, 5V
	*control = 0x41;
	// Wait for completion
	while ((*complete & 0x80) != 0);
	// Print result on a scale from 0 to 2^12 - 1
	res = *lsb;
	res |= *msb << 8;
	//printf("result: %d\n", res);
	close(fd);
	return res;
}
```

```
// adc.h – Analog Digital Converter  External

#include<unistd.h>
#include<sys/types.h>
#include<sys/mman.h>
#include<stdio.h>
#include<fcntl.h>
#include<assert.h>

extern int adc_one();
```

```
//dac.c –  Digital Analog Converter  External

#include "dac.h"

 int todac(int deger)
{
        volatile unsigned int *PBDR, *PBDDR;
        int fd = open("/dev/mem", O_RDWR);
        start = mmap(0, getpagesize(), PROT_READ|PROT_WRITE, MAP_SHARED,
fd, 0x80840000 );
        PBDR = (unsigned int *)(start + 0x04); //port b
        PBDDR = (unsigned int *)(start + 0x14); //port b direction
        *PBDDR = 0xff; //all output
        *PBDR=deger;
        close(fd);
        return 0;
}



//dac.h

#include<unistd.h>
#include<sys/types.h>
#include<sys/mman.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>

extern int todac (int deger);
```

```
// adrport.c - Serial Port Handler
// Copyright MMI, MMII by Sisusypro Incorporated


#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <string.h>
#include <errno.h>
#include "adrport.h"
static int fd = 0;

// opens the serial port
// return code:
//   > 0 = fd for the port
//   -1 = open failed
int OpenAdrPort(char* sPortNumber)
{
   char sPortName[64];
   printf("in OpenAdrPort port#=%s\n", sPortNumber);
   sprintf(sPortName, "/dev/ttyAM%s", sPortNumber);
   printf("sPortName=%s\n", sPortName);

   // make sure port is closed
   CloseAdrPort(fd);

   fd = open(sPortName, O_RDWR | O_NOCTTY | O_NDELAY);
   if (fd < 0)
   {
      printf("open error %d %s\n", errno, strerror(errno));
   }
   else
   {
      struct termios my_termios;
      printf("fd is %d\n", fd);
      tcgetattr(fd, &my_termios);
      // NOTE: you may want to save the port attributes
      //       here so that you can restore them later
      printf("old cflag=%08x\n", my_termios.c_cflag);
      printf("old oflag=%08x\n", my_termios.c_oflag);
      printf("old iflag=%08x\n", my_termios.c_iflag);
      printf("old lflag=%08x\n", my_termios.c_lflag);
      printf("old line=%02x\n", my_termios.c_line);
```

```
        tcflush(fd, TCIFLUSH);

        my_termios.c_cflag = B9600 | CS8 |CREAD | CLOCAL | HUPCL;

        cfsetospeed(&my_termios, B9600);
        tcsetattr(fd, TCSANOW, &my_termios);

        printf("new cflag=%08x\n", my_termios.c_cflag);
        printf("new oflag=%08x\n", my_termios.c_oflag);
        printf("new iflag=%08x\n", my_termios.c_iflag);
        printf("new lflag=%08x\n", my_termios.c_lflag);
        printf("new line=%02x\n", my_termios.c_line);
    } // end if
    return fd;
} // end OpenAdrPort

// writes zero terminated string to the serial port
// return code:
//   >= 0 = number of characters written
//   -1 = write failed
int WriteAdrPort(char* psOutput)
{
    int iOut;
    if (fd < 1)
    {
        printf(" port is not open\n");
        return -1;
    } // end if
    iOut = write(fd, psOutput, strlen(psOutput));
    if (iOut < 0)
    {
        printf("write error %d %s\n", errno, strerror(errno));
    }
    else
    {
          printf("wrote %d chars: %s\n", iOut, psOutput);
    } // end if
    return iOut;
} // end WriteAdrPort

// read string from the serial port
// return code:
//   >= 0 = number of characters read
//   -1 = read failed
int ReadAdrPort(char* psResponse, int iMax)
{
    int iIn;
```

```
   printf("in ReadAdrPort iMax=%d\n", iMax);
   if (fd < 1)
   {
      printf(" port is not open\n");
      return -1;
   } // end if
   strncpy (psResponse, "N/A", iMax<4?iMax:4);
   iIn = read(fd, psResponse, iMax-1);
   if (iIn < 0)
   {
        if (errno == EAGAIN)
        {
                return 0; // assume that command generated no response
        }
        else
        {
                printf("read error %d %s\n", errno, strerror(errno));
        } // end if
   }
   else
   {
        psResponse[iIn<iMax?iIn:iMax] = '\0';
          printf("read %d chars: %s\n", iIn, psResponse);
   } // end if

   return iIn;
} // end ReadAdrPort

// closes the serial port
void CloseAdrPort()
{
        // you may want to restore the saved port attributes
   if (fd > 0)
   {
      close(fd);
   } // end if
} // end CloseAdrPort



// adrport.h
// Copyright MMI, MMII by Sisusypro Incorporated

int OpenAdrPort (char* sPortNumber);
int WriteAdrPort(char* psOutput);
int ReadAdrPort(char* psResponse, int iMax);
void CloseAdrPort();
```

```
/*
 * File: DCMotorControlArm.c
 *
 * Real-Time Workshop code generated for Simulink model DCMotorControlArm.
*/

#include "DCMotorControlArm.h"
#include "DCMotorControlArm_private.h"

/* Block signals (auto storage) */
BlockIO_DCMotorControlArm DCMotorControlArm_B;

/* Block states (auto storage) */
D_Work_DCMotorControlArm DCMotorControlArm_DWork;

/* External inputs (root inport signals with auto storage) */
ExternalInputs_DCMotorControlArm DCMotorControlArm_U;

/* External outputs (root outports fed by signals with auto storage) */
ExternalOutputs_DCMotorControlArm DCMotorControlArm_Y;

/* Real-time model */
RT_MODEL_DCMotorControlArm DCMotorControlArm_M_;
RT_MODEL_DCMotorControlArm *DCMotorControlArm_M =
&DCMotorControlArm_M_;
real_T DCMotorControlArm_RGND = 0.0;   /* real_T ground */

/* Initial conditions for atomic system: '<Root>/Hiz Kontrol' */
void Speed_Correction_Init(void)
{
  /* Level2 S-Function Block: '<S4>/sfunction' (mpc_sfun) */
  {
    SimStruct *rts = DCMotorControlArm_M->childSfunctions[0];
    sfcnInitializeConditions(rts);
    if (ssGetErrorStatus(rts) != (NULL))
      return;
  }
}

/* Output and update for atomic system: '<Root>/Hiz Kontrol' */
void Speed_Correction(void)
{
  {
    real_T tmp;
    uint16_T tmp_0;

    /* Level2 S-Function Block: '<S4>/sfunction' (mpc_sfun) */
```

```
    {
     SimStruct *rts = DCMotorControlArm_M->childSfunctions[0];
     sfcnOutputs(rts, 0);
    }

    /* DataTypeConversion: '<S1>/Data Type Conversion1' */
    tmp = floor(DCMotorControlArm_B.sfunction + 0.5);
    if (tmp < 65536.0) {
     if (tmp >= 0.0) {
       tmp_0 = (uint16_T)tmp;
     } else {
       tmp_0 = 0U;
     }
    } else {
     tmp_0 = MAX_uint16_T;
    }

    DCMotorControlArm_B.DataTypeConversion1 = tmp_0;

    /* Level2 S-Function Block: '<S4>/sfunction' (mpc_sfun) */
    {
     SimStruct *rts = DCMotorControlArm_M->childSfunctions[0];
     sfcnUpdate(rts, 0);
     if (ssGetErrorStatus(rts) != (NULL))
       return;
    }
  }
}

/* Terminate for atomic system: '<Root>/Hiz Kontrol' */
void Speed_Correction_Term(void)
{
  /* Level2 S-Function Block: '<S4>/sfunction' (mpc_sfun) */
  {
    SimStruct *rts = DCMotorControlArm_M->childSfunctions[0];
    sfcnTerminate(rts);
  }
}

/* Output and update for atomic system: '<Root>/Hiz Ölcum ' */
void Measure_Speed(void)
{
  /* DataTypeConversion: '<S2>/Data Type Conversion' incorporates:
   *  Inport: '<Root>/ADC'
   */
  DCMotorControlArm_B.DataTypeConversion =
(real_T)DCMotorControlArm_U.ADC;
```

```
}

/* Model step function */
void DCMotorControlArm_step(void)
{
  /* Outputs for atomic SubSystem: '<Root>/Hiz Ölcum ' */
  Measure_Speed();

  /* end of Outputs for SubSystem: '<Root>/Hiz Ölcum ' */

  /* Outputs for atomic SubSystem: '<Root>/Hiz Kontrol' */
  Speed_Correction();

  /* end of Outputs for SubSystem: '<Root>/Hiz Kontrol' */

  /* Outport: '<Root>/DAC' */
  DCMotorControlArm_Y.DAC = DCMotorControlArm_B.DataTypeConversion1;

  /* Update absolute time for base rate */
  DCMotorControlArm_M->Timing.t[0] =
    (++DCMotorControlArm_M->Timing.clockTick0) *
    DCMotorControlArm_M->Timing.stepSize0;
}

/* Model initialize function */
void DCMotorControlArm_initialize(void)
{
  /* Registration code */

  /* initialize non-finites */
  rt_InitInfAndNaN(sizeof(real_T));
  rtsiSetSolverName(&DCMotorControlArm_M->solverInfo,"FixedStepDiscrete");
  DCMotorControlArm_M->solverInfoPtr = (&DCMotorControlArm_M->solverInfo);

  /* Initialize timing info */
  {
    int_T *mdlTsMap = DCMotorControlArm_M->Timing.sampleTimeTaskIDArray;
    mdlTsMap[0] = 0;
    DCMotorControlArm_M->Timing.sampleTimeTaskIDPtr = (&mdlTsMap[0]);
    DCMotorControlArm_M->Timing.sampleTimes =
      (&DCMotorControlArm_M->Timing.sampleTimesArray[0]);
    DCMotorControlArm_M->Timing.offsetTimes =
      (&DCMotorControlArm_M->Timing.offsetTimesArray[0]);

    /* task periods */
    DCMotorControlArm_M->Timing.sampleTimes[0] = (1.0);
```

```
    /* task offsets */
    DCMotorControlArm_M->Timing.offsetTimes[0] = (0.0);
  }

  rtmSetTPtr(DCMotorControlArm_M, &DCMotorControlArm_M->Timing.tArray[0]);

  {
    int_T *mdlSampleHits = DCMotorControlArm_M->Timing.sampleHitArray;
    mdlSampleHits[0] = 1;
    DCMotorControlArm_M->Timing.sampleHits = (&mdlSampleHits[0]);
  }

  rtmSetTFinal(DCMotorControlArm_M, -1);
  DCMotorControlArm_M->Timing.stepSize0 = 1.0;
  DCMotorControlArm_M->solverInfoPtr = (&DCMotorControlArm_M->solverInfo);
  DCMotorControlArm_M->Timing.stepSize = (1.0);
  rtsiSetFixedStepSize(&DCMotorControlArm_M->solverInfo, 1.0);
  rtsiSetSolverMode(&DCMotorControlArm_M->solverInfo,
SOLVER_MODE_SINGLETASKING);

  /* child S-Function registration */
  {
    RTWSfcnInfo *sfcnInfo = &DCMotorControlArm_M->NonInlinedSFcns.sfcnInfo;
    DCMotorControlArm_M->sfcnInfo = (sfcnInfo);
    rtssSetErrorStatusPtr(sfcnInfo, ((const char_T **)(&rtmGetErrorStatus
      (DCMotorControlArm_M))));
    rtssSetNumRootSampTimesPtr(sfcnInfo,
      &DCMotorControlArm_M->Sizes.numSampTimes);
    rtssSetTPtrPtr(sfcnInfo, &rtmGetTPtr(DCMotorControlArm_M));
    rtssSetTStartPtr(sfcnInfo, &rtmGetTStart(DCMotorControlArm_M));
    rtssSetTFinalPtr(sfcnInfo, &rtmGetTFinal(DCMotorControlArm_M));
    rtssSetTimeOfLastOutputPtr(sfcnInfo, &rtmGetTimeOfLastOutput
      (DCMotorControlArm_M));
    rtssSetStepSizePtr(sfcnInfo, &DCMotorControlArm_M->Timing.stepSize);
    rtssSetStopRequestedPtr(sfcnInfo,
&rtmGetStopRequested(DCMotorControlArm_M));
    rtssSetDerivCacheNeedsResetPtr(sfcnInfo,
      &DCMotorControlArm_M->ModelData.derivCacheNeedsReset);
    rtssSetZCCacheNeedsResetPtr(sfcnInfo,
      &DCMotorControlArm_M->ModelData.zCCacheNeedsReset);
    rtssSetBlkStateChangePtr(sfcnInfo,
      &DCMotorControlArm_M->ModelData.blkStateChange);
    rtssSetSampleHitsPtr(sfcnInfo, &DCMotorControlArm_M->Timing.sampleHits);
    rtssSetPerTaskSampleHitsPtr(sfcnInfo,
      &DCMotorControlArm_M->Timing.perTaskSampleHits);
    rtssSetSimModePtr(sfcnInfo, &DCMotorControlArm_M->simMode);
    rtssSetSolverInfoPtr(sfcnInfo, &DCMotorControlArm_M->solverInfoPtr);
```

```
  }

  DCMotorControlArm_M->Sizes.numSFcns = (1);

  /* register each child */
  {
   (void) memset((void *)&DCMotorControlArm_M-
>NonInlinedSFcns.childSFunctions
                [0],0,
                1*sizeof(SimStruct));
   DCMotorControlArm_M->childSfunctions =
    (&DCMotorControlArm_M->NonInlinedSFcns.childSFunctionPtrs[0]);
   DCMotorControlArm_M->childSfunctions[0] =
    (&DCMotorControlArm_M->NonInlinedSFcns.childSFunctions[0]);

   /* Level2 S-Function Block: DCMotorControlArm/<S4>/sfunction (mpc_sfun) */
   {
    SimStruct *rts = DCMotorControlArm_M->childSfunctions[0];

    /* timing info */
    time_T *sfcnPeriod = DCMotorControlArm_M-
>NonInlinedSFcns.Sfcn0.sfcnPeriod;
    time_T *sfcnOffset = DCMotorControlArm_M-
>NonInlinedSFcns.Sfcn0.sfcnOffset;
    int_T *sfcnTsMap = DCMotorControlArm_M-
>NonInlinedSFcns.Sfcn0.sfcnTsMap;
    (void) memset((void*)sfcnPeriod,0,
                  sizeof(time_T)*1);
    (void) memset((void*)sfcnOffset,0,
                  sizeof(time_T)*1);
    ssSetSampleTimePtr(rts, &sfcnPeriod[0]);
    ssSetOffsetTimePtr(rts, &sfcnOffset[0]);
    ssSetSampleTimeTaskIDPtr(rts, sfcnTsMap);

    /* Set up the mdlInfo pointer */
    {
     ssSetBlkInfo2Ptr(rts, &DCMotorControlArm_M->NonInlinedSFcns.blkInfo2[0]);
     ssSetRTWSfcnInfo(rts, DCMotorControlArm_M->sfcnInfo);
    }

    /* Allocate memory of model methods 2 */
    {
     ssSetModelMethods2(rts, &DCMotorControlArm_M-
>NonInlinedSFcns.methods2[0]);
    }

    /* Allocate memory of model methods 3 */
```

```
    {
     ssSetModelMethods3(rts, &DCMotorControlArm_M-
>NonInlinedSFcns.methods3[0]);
    }

    /* inputs */
    {
     _ssSetNumInputPorts(rts, 9);
     ssSetPortInfoForInputs(rts,
       &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.inputPortInfo[0]);

     /* port 0 */
     {
      real_T const **sfcnUPtrs = (real_T const **)
        &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs0;
      sfcnUPtrs[0] = &DCMotorControlArm_B.DataTypeConversion;
      ssSetInputPortSignalPtrs(rts, 0, (InputPtrsType)&sfcnUPtrs[0]);
      _ssSetInputPortNumDimensions(rts, 0, 1);
      ssSetInputPortWidth(rts, 0, 1);
     }

     /* port 1 */
     {
      real_T const **sfcnUPtrs = (real_T const **)
        &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs1;
      sfcnUPtrs[0] =
&DCMotorControlArm_ConstB.TmpHiddenBufferAtDataTypeCo;
      ssSetInputPortSignalPtrs(rts, 1, (InputPtrsType)&sfcnUPtrs[0]);
      _ssSetInputPortNumDimensions(rts, 1, 1);
      ssSetInputPortWidth(rts, 1, 1);
     }

     /* port 2 */
     {
      real_T const **sfcnUPtrs = (real_T const **)
        &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs2;
      sfcnUPtrs[0] = &DCMotorControlArm_RGND;
      ssSetInputPortSignalPtrs(rts, 2, (InputPtrsType)&sfcnUPtrs[0]);
      _ssSetInputPortNumDimensions(rts, 2, 1);
      ssSetInputPortWidth(rts, 2, 1);
     }

     /* port 3 */
     {
      real_T const **sfcnUPtrs = (real_T const **)
        &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs3;
      sfcnUPtrs[0] = &DCMotorControlArm_RGND;
```

```
  ssSetInputPortSignalPtrs(rts, 3, (InputPtrsType)&sfcnUPtrs[0]);
  _ssSetInputPortNumDimensions(rts, 3, 1);
  ssSetInputPortWidth(rts, 3, 1);
}

/* port 4 */
{
  real_T const **sfcnUPtrs = (real_T const **)
    &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs4;
  sfcnUPtrs[0] = &DCMotorControlArm_RGND;
  ssSetInputPortSignalPtrs(rts, 4, (InputPtrsType)&sfcnUPtrs[0]);
  _ssSetInputPortNumDimensions(rts, 4, 1);
  ssSetInputPortWidth(rts, 4, 1);
}

/* port 5 */
{
  real_T const **sfcnUPtrs = (real_T const **)
    &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs5;
  sfcnUPtrs[0] = &DCMotorControlArm_RGND;
  ssSetInputPortSignalPtrs(rts, 5, (InputPtrsType)&sfcnUPtrs[0]);
  _ssSetInputPortNumDimensions(rts, 5, 1);
  ssSetInputPortWidth(rts, 5, 1);
}

/* port 6 */
{
  real_T const **sfcnUPtrs = (real_T const **)
    &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs6;
  sfcnUPtrs[0] = &DCMotorControlArm_RGND;
  ssSetInputPortSignalPtrs(rts, 6, (InputPtrsType)&sfcnUPtrs[0]);
  _ssSetInputPortNumDimensions(rts, 6, 1);
  ssSetInputPortWidth(rts, 6, 1);
}

/* port 7 */
{
  real_T const **sfcnUPtrs = (real_T const **)
    &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs7;
  sfcnUPtrs[0] = &DCMotorControlArm_RGND;
  ssSetInputPortSignalPtrs(rts, 7, (InputPtrsType)&sfcnUPtrs[0]);
  _ssSetInputPortNumDimensions(rts, 7, 1);
  ssSetInputPortWidth(rts, 7, 1);
}

/* port 8 */
{
```

```
    real_T const **sfcnUPtrs = (real_T const **)
      &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.UPtrs8;
    sfcnUPtrs[0] = &DCMotorControlArm_RGND;
    ssSetInputPortSignalPtrs(rts, 8, (InputPtrsType)&sfcnUPtrs[0]);
    _ssSetInputPortNumDimensions(rts, 8, 1);
    ssSetInputPortWidth(rts, 8, 1);
  }
}

/* outputs */
{
  ssSetPortInfoForOutputs(rts,
    &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.outputPortInfo[0]);
  _ssSetNumOutputPorts(rts, 1);

  /* port 0 */
  {
    _ssSetOutputPortNumDimensions(rts, 0, 1);
    ssSetOutputPortWidth(rts, 0, 1);
    ssSetOutputPortSignal(rts, 0, ((real_T *)
      &DCMotorControlArm_B.sfunction));
  }
}

/* states */
ssSetDiscStates(rts, (real_T *) &DCMotorControlArm_DWork.sfunction_DSTATE
                [0]);

/* path info */
ssSetModelName(rts, "sfunction");
ssSetPath(rts, "DCMotorControlArm/Hiz Kontrol/MPC Controller/sfunction");
ssSetRTModel(rts,DCMotorControlArm_M);
ssSetParentSS(rts, (NULL));
ssSetRootSS(rts, rts);
ssSetVersion(rts, SIMSTRUCT_VERSION_LEVEL2);

/* parameters */
{
  mxArray **sfcnParams = (mxArray **)
    &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.params;
  ssSetSFcnParamsCount(rts, 65);
  ssSetSFcnParamsPtr(rts, &sfcnParams[0]);
  ssSetSFcnParam(rts, 0, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 1, (mxArray*)
                 &DCMotorControlArm_ConstP.sfunction_P2_Size[0]);
  ssSetSFcnParam(rts, 2, (mxArray*)
                 &DCMotorControlArm_ConstP.sfunction_P3_Size[0]);
```

```
ssSetSFcnParam(rts, 3, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
ssSetSFcnParam(rts, 4, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P5_Size[0]);
ssSetSFcnParam(rts, 5, (mxArray*)&DCMotorControlArm_ConstP.pooled7[0]);
ssSetSFcnParam(rts, 6, (mxArray*)&DCMotorControlArm_ConstP.pooled9[0]);
ssSetSFcnParam(rts, 7, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
ssSetSFcnParam(rts, 8, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
ssSetSFcnParam(rts, 9, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
ssSetSFcnParam(rts, 10, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
ssSetSFcnParam(rts, 11, (mxArray*)&DCMotorControlArm_ConstP.pooled9[0]);
ssSetSFcnParam(rts, 12, (mxArray*)&DCMotorControlArm_ConstP.pooled9[0]);
ssSetSFcnParam(rts, 13, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P14_Size[0]);
ssSetSFcnParam(rts, 14, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 15, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P16_Size[0]);
ssSetSFcnParam(rts, 16, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P17_Size[0]);
ssSetSFcnParam(rts, 17, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P18_Size[0]);
ssSetSFcnParam(rts, 18, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P19_Size[0]);
ssSetSFcnParam(rts, 19, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P20_Size[0]);
ssSetSFcnParam(rts, 20, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P21_Size[0]);
ssSetSFcnParam(rts, 21, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 22, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 23, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 24, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 25, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 26, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 27, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 28, (mxArray*)
        &DCMotorControlArm_ConstWithInitP.pooled1[0]);
ssSetSFcnParam(rts, 29, (mxArray*)&DCMotorControlArm_ConstP.pooled7[0]);
ssSetSFcnParam(rts, 30, (mxArray*)
        &DCMotorControlArm_ConstP.sfunction_P31_Size[0]);
ssSetSFcnParam(rts, 31, (mxArray*)&DCMotorControlArm_ConstP.pooled7[0]);
```

```
  ssSetSFcnParam(rts, 32, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 33, (mxArray*)
            &DCMotorControlArm_ConstP.sfunction_P34_Size[0]);
  ssSetSFcnParam(rts, 34, (mxArray*)
            &DCMotorControlArm_ConstP.sfunction_P35_Size[0]);
  ssSetSFcnParam(rts, 35, (mxArray*)
            &DCMotorControlArm_ConstP.sfunction_P36_Size[0]);
  ssSetSFcnParam(rts, 36, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 37, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 38, (mxArray*)
            &DCMotorControlArm_ConstWithInitP.pooled1[0]);
  ssSetSFcnParam(rts, 39, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 40, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 41, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 42, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 43, (mxArray*)
            &DCMotorControlArm_ConstWithInitP.pooled1[0]);
  ssSetSFcnParam(rts, 44, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 45, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 46, (mxArray*)
            &DCMotorControlArm_ConstWithInitP.pooled1[0]);
  ssSetSFcnParam(rts, 47, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 48, (mxArray*)
            &DCMotorControlArm_ConstP.sfunction_P49_Size[0]);
  ssSetSFcnParam(rts, 49, (mxArray*)&DCMotorControlArm_ConstP.pooled9[0]);
  ssSetSFcnParam(rts, 50, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 51, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 52, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 53, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 54, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 55, (mxArray*)
            &DCMotorControlArm_ConstWithInitP.pooled1[0]);
  ssSetSFcnParam(rts, 56, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 57, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 58, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 59, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 60, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 61, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 62, (mxArray*)&DCMotorControlArm_ConstP.pooled3[0]);
  ssSetSFcnParam(rts, 63, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
  ssSetSFcnParam(rts, 64, (mxArray*)&DCMotorControlArm_ConstP.pooled5[0]);
}

/* work vectors */
ssSetPWork(rts, (void **) &DCMotorControlArm_DWork.sfunction_PWORK[0]);

{
```

```
  struct _ssDWorkRecord *dWorkRecord = (struct _ssDWorkRecord *)
    &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.dWork;
  struct _ssDWorkAuxRecord *dWorkAuxRecord = (struct _ssDWorkAuxRecord *)
    &DCMotorControlArm_M->NonInlinedSFcns.Sfcn0.dWorkAux;
  ssSetSFcnDWork(rts, dWorkRecord);
  ssSetSFcnDWorkAux(rts, dWorkAuxRecord);
  _ssSetNumDWork(rts, 2);

  /* PWORK */
  ssSetDWorkWidth(rts, 0, 5);
  ssSetDWorkDataType(rts, 0,SS_POINTER);
  ssSetDWorkComplexSignal(rts, 0, 0);
  ssSetDWork(rts, 0, &DCMotorControlArm_DWork.sfunction_PWORK[0]);

  /* DSTATE */
  ssSetDWorkWidth(rts, 1, 3);
  ssSetDWorkDataType(rts, 1,SS_DOUBLE);
  ssSetDWorkComplexSignal(rts, 1, 0);
  ssSetDWorkUsedAsDState(rts, 1, 1);
  ssSetDWork(rts, 1, &DCMotorControlArm_DWork.sfunction_DSTATE[0]);
}

/* registration */
mpc_sfun(rts);
sfcnInitializeSizes(rts);
sfcnInitializeSampleTimes(rts);

/* adjust sample time */
ssSetSampleTime(rts, 0, 1.0);
ssSetOffsetTime(rts, 0, 0.0);
sfcnTsMap[0] = 0;

/* set compiled values of dynamic vector attributes */
ssSetNumNonsampledZCs(rts, 0);

/* Update connectivity flags for each port */
_ssSetInputPortConnected(rts, 0, 1);
_ssSetInputPortConnected(rts, 1, 1);
_ssSetInputPortConnected(rts, 2, 0);
_ssSetInputPortConnected(rts, 3, 0);
_ssSetInputPortConnected(rts, 4, 0);
_ssSetInputPortConnected(rts, 5, 0);
_ssSetInputPortConnected(rts, 6, 0);
_ssSetInputPortConnected(rts, 7, 0);
_ssSetInputPortConnected(rts, 8, 0);
_ssSetOutputPortConnected(rts, 0, 1);
_ssSetOutputPortBeingMerged(rts, 0, 0);
```

```
    /* Update the BufferDstPort flags for each input port */
    ssSetInputPortBufferDstPort(rts, 0, -1);
    ssSetInputPortBufferDstPort(rts, 1, -1);
    ssSetInputPortBufferDstPort(rts, 2, -1);
    ssSetInputPortBufferDstPort(rts, 3, -1);
    ssSetInputPortBufferDstPort(rts, 4, -1);
    ssSetInputPortBufferDstPort(rts, 5, -1);
    ssSetInputPortBufferDstPort(rts, 6, -1);
    ssSetInputPortBufferDstPort(rts, 7, -1);
    ssSetInputPortBufferDstPort(rts, 8, -1);
   }
  }

  /* InitializeConditions for atomic SubSystem: '<Root>/Hiz Kontrol' */
  Speed_Correction_Init();

  /* end of InitializeConditions for SubSystem: '<Root>/Hiz Kontrol' */
}

/* Model terminate function */
void DCMotorControlArm_terminate(void)
{
  /* Terminate for atomic SubSystem: '<Root>/Hiz Kontrol' */
  Speed_Correction_Term();

  /* end of Terminate for SubSystem: '<Root>/Hiz Kontrol' */
}

/* File trailer for Real-Time Workshop generated code.
 *
 * [EOF]
 */
```

```
/*
 * File: DCMotorControlArm.h
 *
 * Real-Time Workshop code generated for Simulink model DCMotorControlArm.
*/

#ifndef RTW_HEADER_DCMotorControlArm_h_
#define RTW_HEADER_DCMotorControlArm_h_
#ifndef DCMotorControlArm_COMMON_INCLUDES_
# define DCMotorControlArm_COMMON_INCLUDES_
#include <stddef.h>
#include <math.h>
#include <string.h>
#include "rtwtypes.h"
#include "simstruc.h"
#include "fixedpoint.h"
#include "rt_nonfinite.h"
#endif                                 /* DCMotorControlArm_COMMON_INCLUDES_ */

#include "DCMotorControlArm_types.h"

/* Macros for accessing real-time model data structure */
#ifndef rtmGetFinalTime
# define rtmGetFinalTime(rtm)          ((rtm)->Timing.tFinal)
#endif

#ifndef rtmGetSampleHitArray
# define rtmGetSampleHitArray(rtm)     ((rtm)->Timing.sampleHitArray)
#endif

#ifndef rtmGetStepSize
# define rtmGetStepSize(rtm)           ((rtm)->Timing.stepSize)
#endif

#ifndef rtmGetErrorStatus
# define rtmGetErrorStatus(rtm)        ((rtm)->errorStatus)
#endif

#ifndef rtmSetErrorStatus
# define rtmSetErrorStatus(rtm, val)   ((rtm)->errorStatus = (val))
#endif

#ifndef rtmGetStopRequested
# define rtmGetStopRequested(rtm)      ((rtm)->Timing.stopRequestedFlag)
#endif

#ifndef rtmSetStopRequested
```

```
# define rtmSetStopRequested(rtm, val) ((rtm)->Timing.stopRequestedFlag = (val))
#endif

#ifndef rtmGetStopRequestedPtr
# define rtmGetStopRequestedPtr(rtm)   (&((rtm)->Timing.stopRequestedFlag))
#endif

#ifndef rtmGetT
# define rtmGetT(rtm)                 (rtmGetTPtr((rtm))[0])
#endif

#ifndef rtmGetTFinal
# define rtmGetTFinal(rtm)            ((rtm)->Timing.tFinal)
#endif

#ifndef rtmGetTStart
# define rtmGetTStart(rtm)            ((rtm)->Timing.tStart)
#endif

#ifndef rtmGetTimeOfLastOutput
# define rtmGetTimeOfLastOutput(rtm)   ((rtm)->Timing.timeOfLastOutput)
#endif

/* Block signals (auto storage) */
typedef struct {
  real_T DataTypeConversion;          /* '<S2>/Data Type Conversion' */
  real_T sfunction;                   /* '<S4>/sfunction' */
  uint16_T DataTypeConversion1;       /* '<S1>/Data Type Conversion1' */
} BlockIO_DCMotorControlArm;

/* Block states (auto storage) for system '<Root>' */
typedef struct {
  real_T sfunction_DSTATE[3];         /* '<S4>/sfunction' */
  void *sfunction_PWORK[5];           /* '<S4>/sfunction' */
} D_Work_DCMotorControlArm;

/* Invariant block signals (auto storage) */
typedef struct {
  const real_T TmpHiddenBufferAtDataTypeCo;/* 'synthesized block' */
} ConstBlockIO_DCMotorControlArm;

/* Constant parameters (auto storage) */
typedef struct {
  /* Computed Parameter: P1Size
   * '<S4>/sfunction'
   */
  real_T pooled3[2];
```

```
/* Expression: ts
 * '<S4>/sfunction'
 */
real_T pooled4;

/* Computed Parameter: P2Size
 * '<S4>/sfunction'
 */
real_T sfunction_P2_Size[2];

/* Expression: A
 * '<S4>/sfunction'
 */
real_T sfunction_P2[4];

/* Computed Parameter: P3Size
 * '<S4>/sfunction'
 */
real_T sfunction_P3_Size[2];

/* Expression: Cm
 * '<S4>/sfunction'
 */
real_T sfunction_P3[2];

/* Computed Parameter: P4Size
 * '<S4>/sfunction'
 */
real_T pooled5[2];

/* Expression: Dvm
 * '<S4>/sfunction'
 */
real_T pooled6;

/* Computed Parameter: P5Size
 * '<S4>/sfunction'
 */
real_T sfunction_P5_Size[2];

/* Expression: Bu
 * '<S4>/sfunction'
 */
real_T sfunction_P5[2];

/* Computed Parameter: P6Size
```

```
 * '<S4>/sfunction'
 */
real_T pooled7[2];

/* Expression: Bv
 * '<S4>/sfunction'
 */
real_T pooled8[2];

/* Computed Parameter: P7Size
 * '<S4>/sfunction'
 */
real_T pooled9[2];

/* Expression: PTYPE
 * '<S4>/sfunction'
 */
real_T pooled10;

/* Computed Parameter: P14Size
 * '<S4>/sfunction'
 */
real_T sfunction_P14_Size[2];

/* Expression: L
 * '<S4>/sfunction'
 */
real_T sfunction_P14[2];

/* Computed Parameter: P16Size
 * '<S4>/sfunction'
 */
real_T sfunction_P16_Size[2];

/* Expression: KduINV
 * '<S4>/sfunction'
 */
real_T sfunction_P16[4];

/* Computed Parameter: P17Size
 * '<S4>/sfunction'
 */
real_T sfunction_P17_Size[2];

/* Expression: Kx
 * '<S4>/sfunction'
 */
```

```
real_T sfunction_P17[4];

/* Computed Parameter: P18Size
 * '<S4>/sfunction'
 */
real_T sfunction_P18_Size[2];

/* Expression: Ku1
 * '<S4>/sfunction'
 */
real_T sfunction_P18[2];

/* Computed Parameter: P19Size
 * '<S4>/sfunction'
 */
real_T sfunction_P19_Size[2];

/* Expression: Kut
 * '<S4>/sfunction'
 */
real_T sfunction_P19[20];

/* Computed Parameter: P20Size
 * '<S4>/sfunction'
 */
real_T sfunction_P20_Size[2];

/* Expression: Kr
 * '<S4>/sfunction'
 */
real_T sfunction_P20[20];

/* Computed Parameter: P21Size
 * '<S4>/sfunction'
 */
real_T sfunction_P21_Size[2];

/* Expression: Kv
 * '<S4>/sfunction'
 */
real_T sfunction_P21[22];

/* Computed Parameter: P31Size
 * '<S4>/sfunction'
 */
real_T sfunction_P31_Size[2];
```

```
  /* Expression: utarget
   * '<S4>/sfunction'
   */
  real_T sfunction_P31[10];

  /* Computed Parameter: P34Size
   * '<S4>/sfunction'
   */
  real_T sfunction_P34_Size[2];

  /* Expression: p
   * '<S4>/sfunction'
   */
  real_T sfunction_P34;

  /* Computed Parameter: P35Size
   * '<S4>/sfunction'
   */
  real_T sfunction_P35_Size[2];

  /* Expression: Jm
   * '<S4>/sfunction'
   */
  real_T sfunction_P35[20];

  /* Computed Parameter: P36Size
   * '<S4>/sfunction'
   */
  real_T sfunction_P36_Size[2];

  /* Expression: DUFree
   * '<S4>/sfunction'
   */
  real_T sfunction_P36[10];

  /* Computed Parameter: P49Size
   * '<S4>/sfunction'
   */
  real_T sfunction_P49_Size[2];

  /* Expression: maxiter
   * '<S4>/sfunction'
   */
  real_T sfunction_P49;
} ConstParam_DCMotorControlArm;

/* Constant parameters with dynamic initialization (auto storage) */
```

```
typedef struct {
  /* Computed Parameter: P15Size
   * '<S4>/sfunction'
   */
  real_T pooled1[2];
} ConstParamWithInit_DCMotorControlArm;

/* External inputs (root inport signals with auto storage) */
typedef struct {
  uint16_T ADC;                        /* '<Root>/ADC' */
} ExternalInputs_DCMotorControlArm;

/* External outputs (root outports fed by signals with auto storage) */
typedef struct {
  uint16_T DAC;                        /* '<Root>/DAC' */
} ExternalOutputs_DCMotorControlArm;

/* Real-time Model Data Structure */
struct RT_MODEL_DCMotorControlArm {
  struct SimStruct_tag * *childSfunctions;
  const char_T * volatile errorStatus;
  SS_SimMode simMode;
  RTWSolverInfo solverInfo;
  RTWSolverInfo *solverInfoPtr;
  void *sfcnInfo;

  /*
   * NonInlinedSFcns:
   * The following substructure contains information regarding
   * non-inlined s-functions used in the model.
   */
  struct {
    RTWSfcnInfo sfcnInfo;
    SimStruct childSFunctions[1];
    SimStruct *childSFunctionPtrs[1];
    struct _ssBlkInfo2 blkInfo2[1];
    struct _ssSFcnModelMethods2 methods2[1];
    struct _ssSFcnModelMethods3 methods3[1];
    struct {
      time_T sfcnPeriod[1];
      time_T sfcnOffset[1];
      int_T sfcnTsMap[1];
      struct _ssPortInputs inputPortInfo[9];
      real_T const *UPtrs0[1];
      real_T const *UPtrs1[1];
      real_T const *UPtrs2[1];
      real_T const *UPtrs3[1];
```

```
    real_T const *UPtrs4[1];
    real_T const *UPtrs5[1];
    real_T const *UPtrs6[1];
    real_T const *UPtrs7[1];
    real_T const *UPtrs8[1];
    struct _ssPortOutputs outputPortInfo[1];
    uint_T attribs[65];
    mxArray *params[65];
    struct _ssDWorkRecord dWork[2];
    struct _ssDWorkAuxRecord dWorkAux[2];
  } Sfcn0;
} NonInlinedSFcns;

/*
 * ModelData:
 * The following substructure contains information regarding
 * the data used in the model.
 */
struct {
  boolean_T zCCacheNeedsReset;
  boolean_T derivCacheNeedsReset;
  boolean_T blkStateChange;
} ModelData;

/*
 * Sizes:
 * The following substructure contains sizes information
 * for many of the model attributes such as inputs, outputs,
 * dwork, sample times, etc.
 */
struct {
  uint32_T options;
  int_T numContStates;
  int_T numU;
  int_T numY;
  int_T numSampTimes;
  int_T numBlocks;
  int_T numBlockIO;
  int_T numBlockPrms;
  int_T numDwork;
  int_T numSFcnPrms;
  int_T numSFcns;
  int_T numIports;
  int_T numOports;
  int_T numNonSampZCs;
  int_T sysDirFeedThru;
  int_T rtwGenSfcn;
```

```
  } Sizes;

  /*
   * Timing:
   * The following substructure contains information regarding
   * the timing information for the model.
   */
  struct {
    time_T stepSize;
    uint32_T clockTick0;
    time_T stepSize0;
    time_T tStart;
    time_T tFinal;
    time_T timeOfLastOutput;
    boolean_T stopRequestedFlag;
    time_T *sampleTimes;
    time_T *offsetTimes;
    int_T *sampleTimeTaskIDPtr;
    int_T *sampleHits;
    int_T *perTaskSampleHits;
    time_T *t;
    time_T sampleTimesArray[1];
    time_T offsetTimesArray[1];
    int_T sampleTimeTaskIDArray[1];
    int_T sampleHitArray[1];
    int_T perTaskSampleHitsArray[1];
    time_T tArray[1];
  } Timing;
};

/* Block signals (auto storage) */
extern BlockIO_DCMotorControlArm DCMotorControlArm_B;

/* Block states (auto storage) */
extern D_Work_DCMotorControlArm DCMotorControlArm_DWork;

/* External inputs (root inport signals with auto storage) */
extern ExternalInputs_DCMotorControlArm DCMotorControlArm_U;

/* External outputs (root outports fed by signals with auto storage) */
extern ExternalOutputs_DCMotorControlArm DCMotorControlArm_Y;
extern ConstBlockIO_DCMotorControlArm DCMotorControlArm_ConstB;/* constant
block i/o */

/* Constant parameters (auto storage) */
extern const ConstParam_DCMotorControlArm DCMotorControlArm_ConstP;
```

```
/* Constant parameters with dynamic initialization (auto storage) */
extern ConstParamWithInit_DCMotorControlArm
DCMotorControlArm_ConstWithInitP;/* constant parameters */

/* External data declarations for dependent source files */
extern real_T DCMotorControlArm_RGND;  /* real_T ground */

/* Model entry point functions */
extern void DCMotorControlArm_initialize(void);
extern void DCMotorControlArm_step(void);
extern void DCMotorControlArm_terminate(void);

/* Real-time Model object */
extern RT_MODEL_DCMotorControlArm *DCMotorControlArm_M;

/*-
 * The generated code includes comments that allow you to trace directly
 * back to the appropriate location in the model.  The basic format
 * is <system>/block_name, where system is the system number (uniquely
 * assigned by Simulink) and block_name is the name of the block.
 *
 * Use the MATLAB hilite_system command to trace the generated code back
 * to the model.  For example,
 *
 * hilite_system('<S3>')    - opens system 3
 * hilite_system('<S3>/Kp') - opens and selects block Kp which resides in S3
 *
 * Here is the system hierarchy for this model
 *
 * '<Root>' : DCMotorControlArm
 * '<S1>'   : DCMotorControlArm/Hiz Kontrol
 * '<S2>'   : DCMotorControlArm/Hiz Ölcum
 * '<S3>'   : DCMotorControlArm/Set Speed
 * '<S4>'   : DCMotorControlArm/Hiz Kontrol/MPC Controller
 */
#endif                                 /* RTW_HEADER_DCMotorControlArm_h_ */

/* File trailer for Real-Time Workshop generated code.
 *
 * [EOF]
 */
```

```
/*
 * File: ert_main.c
 *
 * Real-Time Workshop code generated for Simulink model DCMotorControlArm.
*/

#include <stdio.h>                    /* This ert_main.c example uses printf/fflush */
#include "DCMotorControlArm.h"        /* Model's header file */
#include "rtwtypes.h"                 /* MathWorks types */

static boolean_T OverrunFlag = 0;

/* Associating rt_OneStep with a real-time clock or interrupt service routine
 * is what makes the generated code "real-time".  The function rt_OneStep is
 * always associated with the base rate of the model.  Subrates are managed
 * by the base rate from inside the generated code.  Enabling/disabling
 * interrupts and floating point context switches are target specific.  This
 * example code indicates where these should take place relative to executing
 * the generated code step function.  Overrun behavior should be tailored to
 * your application needs.  This example simply sets an error status in the
 * real-time model and returns from rt_OneStep.
 */
void rt_OneStep(void)
{
  /* Disable interrupts here */

  /* Check for overrun */
  if (OverrunFlag++) {
    rtmSetErrorStatus(DCMotorControlArm_M, "Overrun");
    return;
  }

  /* Save FPU context here (if necessary) */
  /* Re-enable timer or interrupt here */
  /* Set model inputs here */

  /* Step the model */
  DCMotorControlArm_step();

  /* Get model outputs here */

  /* Indicate task complete */
  OverrunFlag--;

  /* Disable interrupts here */
  /* Restore FPU context here (if necessary) */
  /* Enable interrupts here */
```

```
}

/* The example "main" function illustrates what is required by your
 * application code to initialize, execute, and terminate the generated code.
 * Attaching rt_OneStep to a real-time clock is target specific.  This example
 * illustates how you do this relative to initializing the model.
 */
int_T main(int_T argc, const char_T *argv[])
{
  /* Initialize model */
  DCMotorControlArm_initialize();

  /* Simulating the model step behavior (in non real-time) to
   *  simulate model behavior at stop time.
   */
  while ((rtmGetErrorStatus(DCMotorControlArm_M) == (NULL)) &&
        !rtmGetStopRequested(DCMotorControlArm_M)) {
    rt_OneStep();
  }

  /* Disable rt_OneStep() here */

  /* Terminate model */
  DCMotorControlArm_terminate();
  return 0;
}

/* File trailer for Real-Time Workshop generated code.
 *
 * [EOF]
 */
```