Kişisel Bilgisayar Tabanlı Evrensel Hareket Denetleyici Sistemlerinin Geliştirilmesi

Proje No: 108E048

Y. Doç. Dr. Melik DÖLEN
Y. Doç. Dr. A. Buğra KOKU

AĞUSTOS 2010
ANKARA

ÖNSÖZ

Bu projede her türlü endüstriyel sistemin (CNC takım tezgahları, montaj robotları, otomasyon sistemleri, vs.) denetimine imkan sağlayan ve ele alınan uygulamanın gerekleri doğrultusunda kolayca şekillendirilebilen evrensel bir denetim sistemi ortaya konmuştur. Alan programlanabilir kapı dizini (FPGA) ve kişisel bilgisayar içeren bu dizgenin özellikleri ve denetim başarımı çeşitli deneyler ve çevrimiçi donanım benzetimi aracılığıyla gösterilmiştir.

Orta Doğu Teknik Üniversitesi, Makina Mühendisliği Bölümünde yürütülmüş bu çalışma,TÜBİTAK tarafından desteklenmiştir. Proje yürütücüsü ve araştırmacıları, sağladığı finansal destekle bu çalışmanın gerçekleştirilmesine imkan sağlayan TÜBİTAK-EEEAG'ye içtenlikle teşekkürlerini sunarlar.

Y. Doç Dr. Melik DÖLEN
Y. Doç Dr. A. Buğra KOKU
Y. Müh. Ergin KILIÇ
Y. Müh. Barış R. MUTLU
Y. Müh. Serdar ÜŞENMEZ
Y. Müh. Ulaş YAMAN

İÇİNDEKİLER

TABLO LİSTELERİ

ŞEKİL LİSTELERİ

ÖZET

Bu proje kapsamında her türlü endüstriyel sistemin denetimine imkan sağlayan ve uygulamanın gerekleri doğrultusunda (elektronik olarak) biçimlendirilebilir evrensel bir denetim sisteminin kavramı ortaya konmuştur. Denetleyici sisteminin bir prototipi çeşitli FPGA geliştirme kartları (Altera DE1, Xilinx ML 505) ve projeye has tasarlanmış giriş/çıkış kartları vasıtasıyla meydana getirilmiştir. Ayrıca, herhangi bir denetim sisteminin FPGA kırmığı üzerinde devre sentezinin yapılmasına imkan tanıyacak çok sayıda düşünce iyeliği modül (Verilog ve VHDL gibi donanım tanımlama dilleri kullanılarak) oluşturulmuştur. Proje kapsamında yapılan deneyler ve çevrimiçi donanım benzetimi sonucunda sistemin piyasada bulunan ve hemen hemen her tür ticari denetim sistemi (hareket kontrol kartları, endüstriyel PC- veya PLC tabanlı sistemler) ile başa baş bir denetleme başarımı sergileyeceği görülmüştür.

ABSTRACT

In this project, a concept for a universal controller topology that could be electronically tailored to any industrial control application is developed. The prototype of this controller is implemented via various FPGA development boards (Altera DE1, Xilinx ML 505) as well as peripheral device interface cards that are specifically designed (and produced) in this project. Furthermore, a number of intellectual property (IP) modules, which enable the circuit synthesis of the presented controller, is devised using hardware definition languages like Verilog HDL and VHDL. Based on the experiments and hardware-in-the-loop simulations conducted, the performance of the presented controller is found to match to those of any commercial control systems (motion control cards, industrial PC or PLC based systems) available in the market.

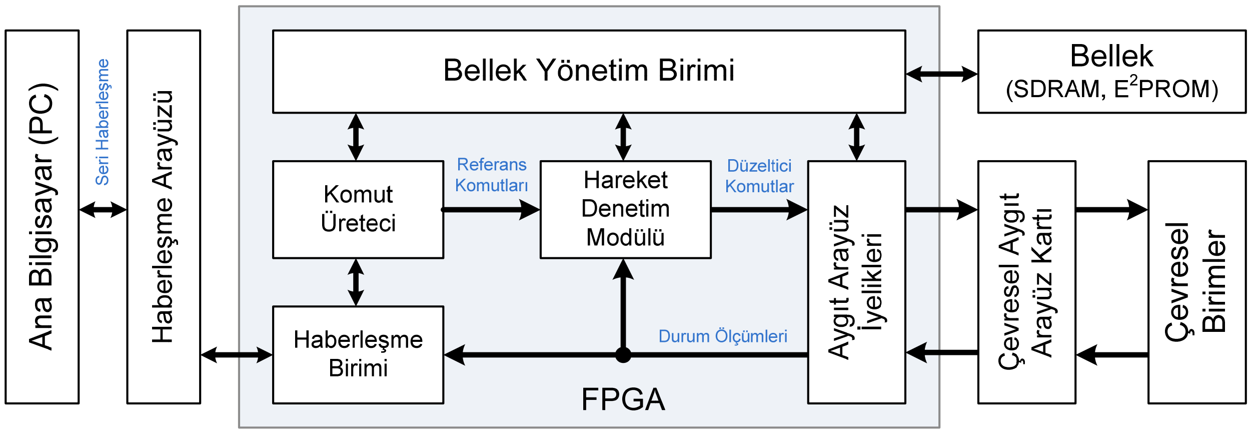
1. GİRİŞ

Hareket denetleyiciler; bilgisayar kontrollü takım tezgahları, işlem denetimi ve takibi, robot ve endüstriyel otomasyon alanında yaygın olarak kullanılan sistemlerdir. Genelde bağımsız olarak çalışan bu kart sistemleri bünyesinde gelişkin mikroişlemcileri ve/veya sayısal işaret işleyiciler (DSP) barındırıp, hareket denetleme sisteminde kullanılan birçok duyucu sistemiyle beraber çalışacak çevresel birimlere sahiptirler. Birçok uygulamaya hitap etmek üzere, özel bir yorumlanan dil vasıtasıyla bu kartları programlamak ve uygulama için tatbik etmek mümkündür. Her ne kadar bu kart sistemleri çok yaygın olarka kullanılsa da bu sistemlerin birçok dezavantajı vardır:

- Bu kart sistemleri sadece belirli bir kontrol kanununu (geliştirilmiş PID) icra etmek üzere tasarlanırlar. Kullanıcılar kendi uygulamaları için mevcut kontrol algoritmasının parametrelerini (PID kazançlarını) ayarlayarak istenilen denetleme performansını sağlamaya çalışırlar. Dolayısıyla çok daha farklı kontrol gereksinimleri olan uygulamalar için mevcut algoritmayı geliştirmek mümkün olmamaktadır.
- Bu kartlar sadece sınırlı sayıda bağımsız eksenin (tipik olarak 8 eksen) kontrol edilmesi için uygundur. Daha çok sayıda eksenin denetlenmesini gerektiren durumlarda birçok kartın kullanımı gerekmekte ve bunların eşgüdümü oldukça zor olmaktadır.
- Bu kartlar sadece belli tiplerdeki duyucuların ve çevresel birimlerin (motor sürücüleri/güçlendiriciler) bağlanmasına imkan tanırlar. Bunlara örnek olarak sınırlı sayıda analog duyucu, hareket sınırlayıcı anahtarlar (“limit switch”), optik konum duyucusu/doğrusal cetvel (“optical position encoder/linear scale”), motor sürücüler için de sayısal analog çeviriciler verilebilir. Gelişkin uygulamalar için gereken özel duyucu sistemlerinin (CCD kameralar, sayısal akustik duyucular) veya farklı çevresel birimlerin bu tip sistemlere “doğrudan” bağlanması mümkün olmamaktadır.
- İşletimi sağlayan çekirdek yazılımı (“firmware”) çok değişik uygulamalara hitap edeceği için, programlamak için birçok özel fonksiyonu barındırmak durumundadır. Uygulamada karşılaşılan birçok kontrol probleminin çözülmesi, bu özel programlar vasıtasıyla sağlanmaktadır. Bu kadar çok fonksiyonu kartta mevcut olan yorumcunun (“interpreter”) barındırması mikroişlemcinin kaynaklarının büyük bir bölümünü harcamaktadır. Kullanıcının geliştirmiş olduğu özel program bu yorumcu vasıtasıyla icra edildiğinden, özel uygulamalar için pek de verimli olmayan bir yapı ortaya çıkmaktadır.

Bu proje kapsamında kavramsal olarak geliştirilen evrensel sayısal denetleyici sistemi, bu tür dezavantajlara sahip olmayacaktır. Önerilen sistem FPGA (Field Programmable Gate Array - Alan Programlanabilir Kapı Dizini) kırmıkları vasıtasıyla çevresel birimlerle ve sisteme bağlanan duyucularla etkileşim halinde olmaktadır. Böylece, FPGA üzerinde sentezlenen mantıksal devreyi (yazılım yoluyla) değiştirerek birçok değişik çevre birimine hitap edecek esnek bir denetleyici mimarisinin oluşturulması sağlanabilmektedir. Denetleyici sisteminin kullanıcısı, uygulamanın gerektirdiği standard veya standard dışı her seviye kontrol algoritmasını (kayan kip, bulanık mantık, uyum sağlayan denetim algoritmaları, değişik durum uzayı denetleyicileri ve benzeri), yazılım ortamında geliştirerek ayrıca kart üzerinde bir yorumcuya ihtiyaç duymaksızın, kart üzerinde doğrudan gerçekleyebilecektir.

FPGA kırmıklarının getirdiği tüm avantajları kullanan evrensel bir sayısal kontrol sistemi (Digital MOtion Control System – DMOX) Şekil 1.1'de gösterilmiştir. Proje çalışmaları çerçevesinde evrimleşerek ortaya çıkan bu sistem FPGA kırmığına sahip bir entegre devre kartından, üzerinde gerçek-zamanlı olmayan bir işletim sistemi koşan kişisel bilgisayardan ve proje kapsamında geliştirilen bir giriş-çıkış kartından oluşmaktadır. Ürünleşme aşamasında, ortaya konan bu denetleme sisteminin birçok sektörün (Üretim, Savunma, Havacılık, Otomotiv) esnek denetleme sistemi ihtiyacını (hızlı ve güvenilir olarak) karşılayacağı düşünülmektedir.



Şekil 1.1: DMOX Mimarisi.

2. PROJE ÇALIŞMALARI

Şekil 1.1'de gösterilen denetleme sistemini gerçeklemek üzere, proje kapsamında çalışmalar altı ayrı alanda (çalışma grubunda) yürütülmüştür. Çalışma alanları ve ilgili etkinlikler şu şekildedir:

- **Grup 1:** Hareket denetiminde yaygın olarak kullanılan duyucu (optik konum enkoderi, doğrusal cetvel, tachogenerator, resolver, vb.), motor (AC/DC servo-motor, adım-motoru) sürücüsü ve çeşitli çevresel donanımların geliştirilmekte olan (yazılımla biçimlendirilebilir) denetim kartına kolayca bağlanabilmesine imkan tanıyacak aygıt giriş/çıkış arayüz kartlarının üretimi/testi.

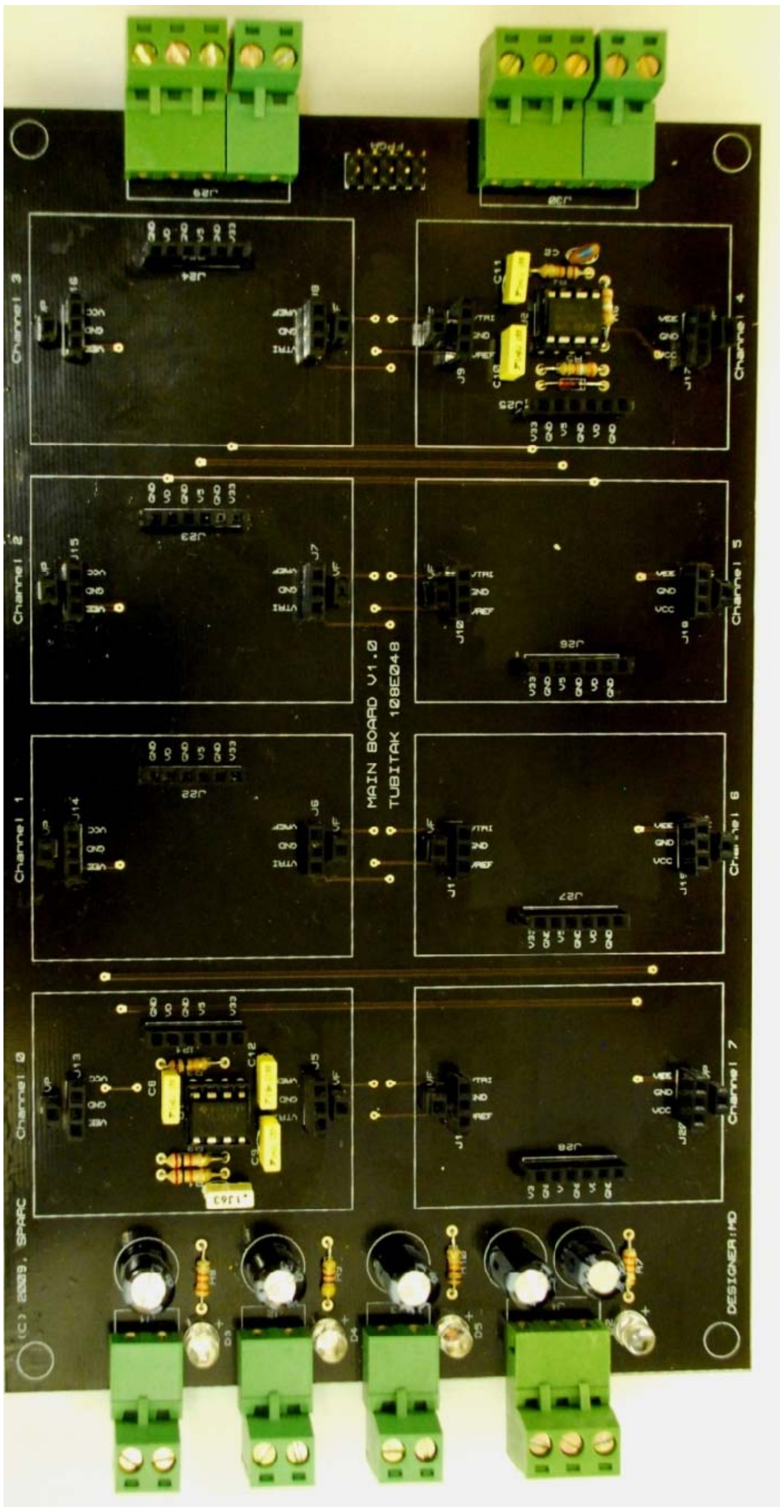
- **Grup 2:** FPGA kırmığını bünyesinde barındıracak olan evrensel denetleme sisteminde kullanılmak üzere; bazı temel işlevleri (duyucu işaretleri işleme, bilgi aktarma/dönüştürme vb.) yerine getirecek düşünce iyeliklerinin geliştirilmesi.
- **Grup 3:** Evrensel denetleme kartında kullanılmak üzere; temel denetim (kontrol) algoritmalarının araştırılması ve onların FPGA kırmığı üzerinde gerçeklemesinin sağlayacak düşünce iyeliklerinin geliştirilmesi.
- **Grup 4:** Evrensel hareket denetleyici sistemin önemli bir bileşeni olarak, komut üretimi ve çeşitli enterpolasyon yöntemlerinin araştırılması; bunların geliştirilen denetleyici prototipinde kullanılmasına yarayan yazılımların ve düşünce iyeliklerinin ortaya konması.
- **Grup 5:** Kişisel bilgisayar tabanlı gerçek-zamanlı olarak çalışan denetim mimarilerinin araştırılması.
- **Grup 6:** **Çevrim-içi donanım benzetimini** (ÇDB) ("Hardware-in-the-loop simulation") gerçekleştirmek üzere; sonlu-fark denklemleri ile adi differansiyel denklemlerle ilgili çözüm/hesaplama tekniklerinin incelenmesi; bunların FPGA kırmığı üzerinde icrasını sağlayacak yazılımların ve düşünce iyeliklerinin tasarlanması.

Bu gruplar içinde yürütülen çalışmalar aşağıda detaylı olarak açıklanmıştır.

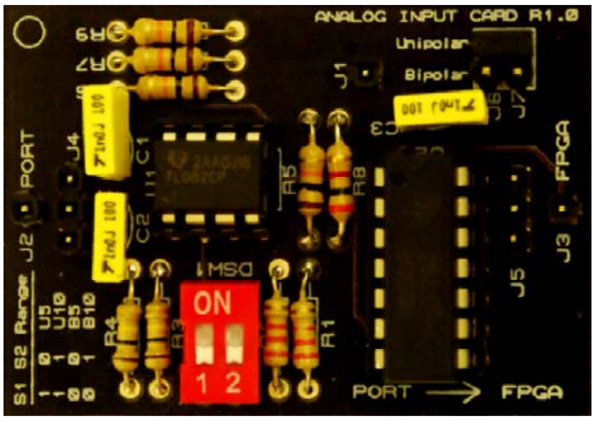
2.1 Aygıt Arayüzünün Geliştirilmesi

Projede ele alınan evrensel denetleme kartının (DMOX) ana bileşenlerinden biri aygıt arayüzüdür. Söz konusu arayüz; yazılımla ilgili port bağlantı biçimlerinin tanımlanabileceği bir kart olup, hareket denetim teknolojisinde yer alan her türlü duyucu ve aygıtın (istenen sırada) denetim sisteme (rastgele) bağlanabilmesine imkan tanıyacaktır.

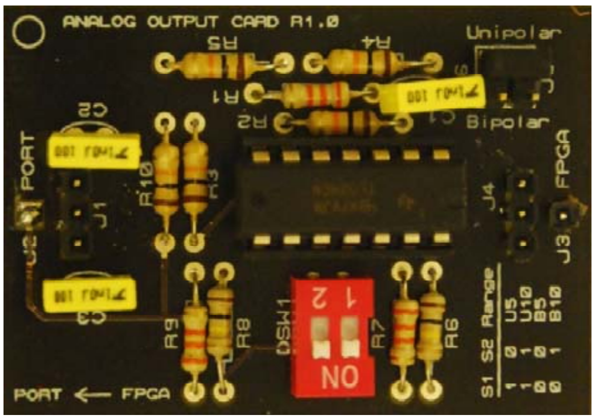
Bu amaçla, projenin ilk altı ayında böylesi çeviricilerle ilgili birtakım ön-tasarımlar yapılarak, önsınamadan geçirilmişti. Elde edilen bilgilerin ışığında, sözü edilen türde bir arayüz kartının tasarımı (ilk sürüm) tamamlanmıştır. Tasarlanan bu kartlarla ilgili detaylı bilgiler önceki gelişme raporlarında sunulmuş olup, projenin üçüncü döneminde bu kartlar üretilerek öntestlerden başarıyla geçirilmiştir. Şekil 2.1 sekiz (giriş/çıkış) kanalından oluşan ana kartı ("main board"); Şekil 2.2 de bu ana kart üzerine takılacak yardımcı kartları ("daughter board") göstermektedir. Şekil 2.3 ilgili kanallara 4 değişik tipteki (analog giris, analog çıkış, sayısal giriş, sayısal çıkış) yardımcı kartların takılmasıyla "biçimlendirilen" bir ana kartı betimlemektedir. Benzer olarak, Şekil 2.4 ise FPGA geliştirme kartına (Altera DE1) bağlanan böylesi arayüz kartı temsil etmektedir. Kullanılan FPGA geliştirme kartına (herbiri 8 aygıt bağlantısı içeren) 4 adet ana kart bağlanabilmektedir.



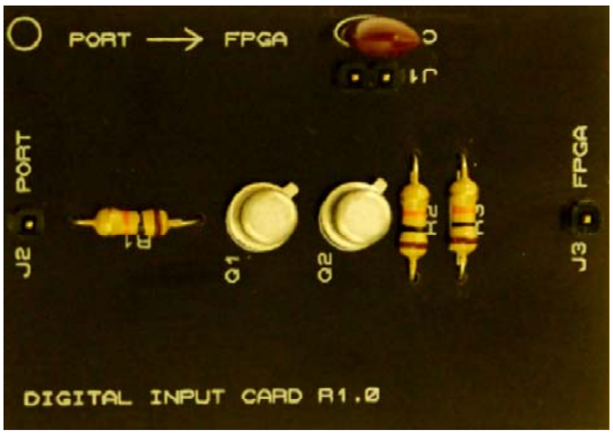
Şekil 2.1: Proje kapsamında tasarlanıp/üretilen ana arayüz kartı (ilk sürüm).



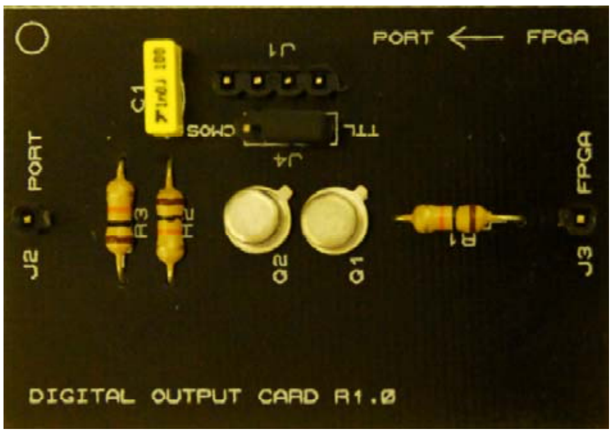
a) Analog giriş kartı



b) Analog çıkış kartı

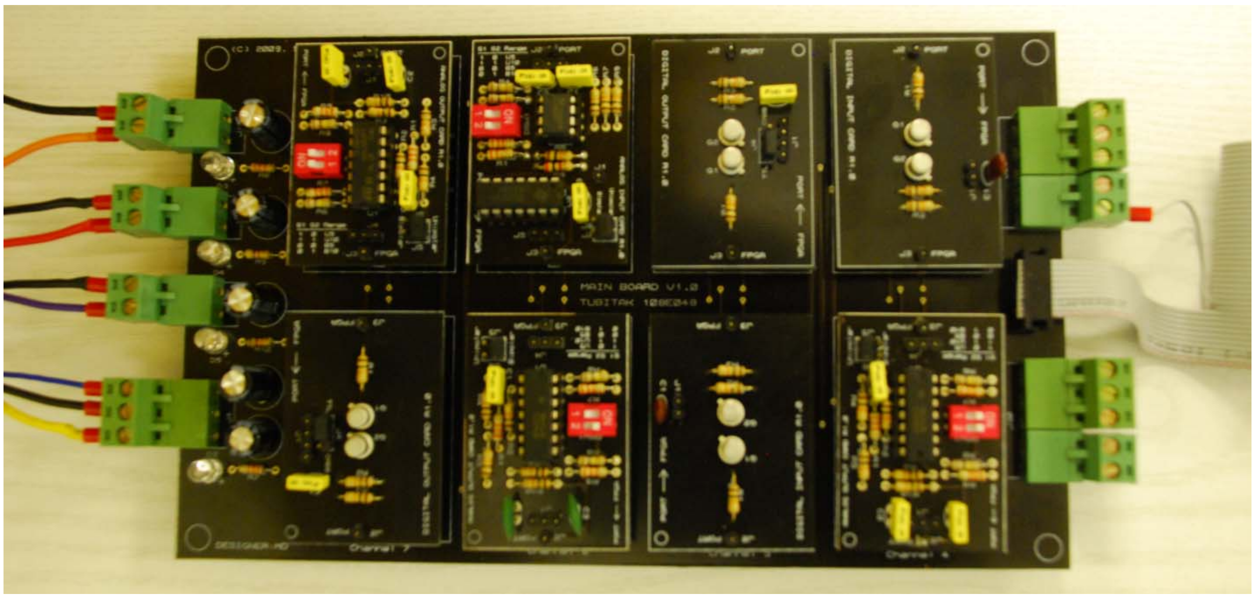


c) Sayısal giriş kartı

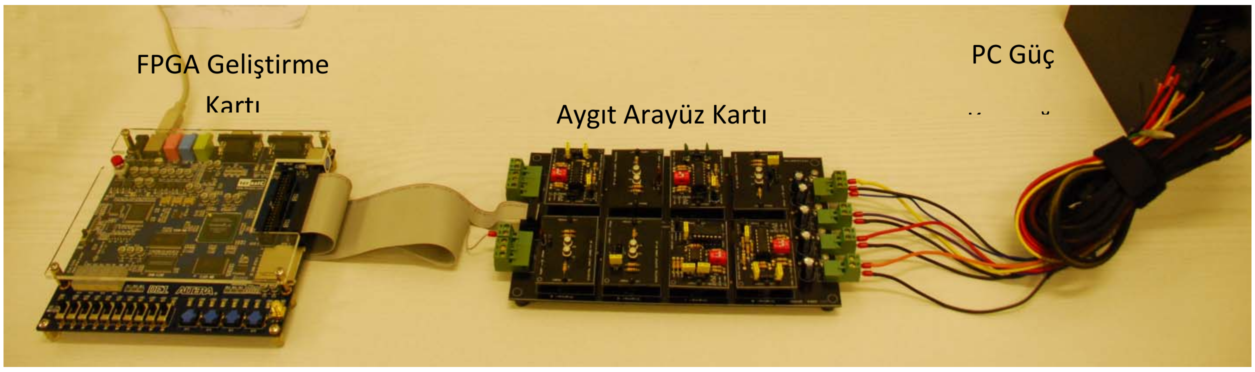


d) Sayısal çıkış kartı

Şekil 2.2: Proje kapsamında tasarlanıp/üretilen yardımcı kartlar (ilk sürüm).



Şekil 2.3: Ana kart üzerine yerleştirilmiş yardımcı kartlar.



Şekil 2.4: FPGA (geliştirme) kartı ve aygıt arayüz kartının genel görünüşü.

Dolayısıyla, tek bir FPGA kartı ile 8 eksen barındıran bir makina sisteminin denetiminin yapılabilmesi imkan dahilindedir. Proje kapsamında geliştirilen tüm kartlarla (ve ilgili testlerle) ilgili detaylı bilgiler [3]'de verilmektedir. Ayrıca [3]'te ilgili kartlarla tümleşik çalışarak analog↔sayısal veri dönüşümünü gerçekleştiren düşünce iyeliklerinin de tanıtımı yapılmaktadır.

Proje kapsamında geliştirilen arayüz kartlarının doğrudan doğruya FPGA geliştirme kartlarına bağlanarak kullanılmakta olup, sunulan aygıt arayüzü elle biçimlendirilmektedir. Yani, yardımcı kartlar üzerindeki anahtarlar ve köprü bağlantıları ("jumper") aracılığıyla; kanalların çalışma kipi belirlenmektedir. Ürünleşme aşamasına gelindiğinde,

- anahtarların (ve köprülerin) yerini çift yönlü akım ileten hızlı analog anahtarların alması,

- kartların seçiminin (kanalların konfigürasyonunun) yine bu tür elemanlar (analog anahtarlar, aktarım kapıları, çoklayıcılar) kullanılarak, tamamen elektronik olarak yapılması,
- SMD eleman kullanımıyla devrelerin minyatürleştirilmesi

olasıdır. Ancak, bu noktada önemle vurgulanması gereken husus, yukarıda betimlenen çerçevede kullanılacak bir evrensel aygıt arayüzü için çok kapsamlı bir (melez) mikro-elektronik devre tasarımı gerekmesidir. Zira, böylesi bir arayüzün ticari olarak kabul görmesi için, bunun çok kanal barındıran (16 ve üzeri) bir kırmık olması zaruridir. Şüphesiz, böylesi bir geliştirme etkinliği için projenin hem zamanı hem de kaynakları yeterli değildir.

2.2 Aygıt Haberleşme Modüllerinin Geliştirilmesi

Herhangi bir denetleyici sistem, ilgili kontrol işlemini yürütürken sürekli olarak çeşitli çevresel birimlerle bilgi alışverişi içinde olmalıdır. Uygulamanın doğasına bağlı olarak seçilen bu birimler en genel anlamıyla duyucu ve eyleyici arayüzlerinden oluşmaktadır. Dolayısıyla, bu proje kapsamındaki evrensel denetleyici sistem geliştirilirken, sisteme bağlanması olası tüm birimler ve onlarla ilgili özellikler (işaretler, protokoller, işaret işleme ve veri dönüşüm işlemleri) dikkate alınmıştır. Bunun sonucu olarak, aşağıda verilen modüllerin bir evrensel denetleyici sistem oluşturulurken temel teşkil edeceği görülmüştür:

- <u>Artımlı optik konum enkoder çözümleyicisi:</u> Birçok hareket denetim uygulamasında temel duyucu enkoder kullanılmaktadır. Bu duyuculardan gelen işaretlerin (darbelerin) işlenerek, sayısal konum ve hız bilgisinin üretilmesi gerekmektedir.
- <u>Dördül işaret üreteci:</u> Bu artımlı (“incremental”) bir sayısal veriyi, iki kanallı dördül (“quadrature”) kodlanmış sayısal işaret dizinine çevirir. Bu modül özellikle enkoder arayüzü olan cihazlarla haberleşmek üzere geliştirilmiş olup, ÇDB’de özellikle tercih edilmektedir.
- <u>Değişken genişlikli darbe üreteci (“Pulse Width Modulator”)</u>: Birçok duyucu ve sürücü darbe genişlik kiplenimiyle bilgiyi (atım çalışma oranı vasıtasıyla) diğer cihazlara aktarmaktadır. Bu üreteç vasıtasıyla (sigma-delta kiplenimi kullanılarak) özellikle sayısal-analog veri dönüşümü işlemini gerçekleştirmek nispeten kolaydır.
- <u>Darbe genişlik çözümleyicisi:</u> Darbe genişlik kiplenimi yoluyla aktarılan bilgilerin sayısal veriye çevrilmesi için gereklidir.
- <u>Frekans üreteci:</u> Sayısal bilgilerin atım frekansı şeklinde kodlanarak (ki atım çalışma oranı sabittir), diğer birimlere aktarılmasında kullanılır. Melez (hibrit) devre teknolojisinde bunun en temel uygulaması frekans-gerilim (“frequency-to-voltage conversion”) dönüşümüdür.

- Frekans çözümleyicisi: Frekans üreteci tarafından oluşturulan darbe dizinlerinin frekansını ölçerek, kodlanmış sayısal bilgiyi ayrıştırır. Bu modül özellikle gerilim-frekans ceviricilerin ("voltage-to-frequency converter") ürettiği bilginin işlenmesinde kullanılır.
- SPI birimi: *Serial Peripheral Interface* (SPI) sayısal cihazlar arasında hızlı bilgi aktarımında kullanılan senkron bir haberleşme protokolüdür. Günümüzde SPI temel haberleşme arayüzü olarak sayısal duyucularda standart yer aldığından, böylesi bir haberleşme biriminin geliştirilmesi gereklidir.

Yukarıda bahsedilen modüller ve bunların Verilog HDL ile FPGA üzerindeki devre sentezi konusunda teknik bilgiler önceki dönem raporlarında kısaca verilmişti. [1]'de bu projeye has geliştirilmiş söz konusu modüller detaylı olarak irdelenmiş olup; projede faydalanılan (ve diğer kaynaklar tarafından geliştirilen) düşünce iyelikleri de (RS-232 birimi, SRAM denetleyicisi, vs.) yine burada tanıtılmaktadır. Böylece sözü edilen temel modülleri bir araya getirerek, herhangi DMOX topolojisini arzu edilen uygulama için şekillendirmek mümkün olmaktadır.

2.3 Denetim Algoritmaları ve İlgili Modüllerin Geliştirilmesi

DMOX mimarisinde kullanılmak üzere, proje kapsamında birçok denetleyici topolojisi geliştirilmiş, (daha önceki gelişme raporlarında açıklandığı üzere) bunlar değişik FPGA platformlarında (Altera Cyclone II, Xilinx Virtex 5) hayata geçirilerek, sınaması başarıyla yapılmıştı. Hatırlanacağı üzere, ticari hareket kontrol kartlarında genellikle ileri besleme içeren PID denetleyiciler standart olarak kullanılmakta olup, uygulamanın gerekleri doğrultusunda başka denetleme algoritmalarının gerçeklemesinin imkanı bulunmamaktadır. Diğer taraftan, bu projede ortaya konan topolojiler henhangi bir kontrol uygulaması için rahatlıkla şekillendirilip, en yüksek denetim sağlamak üzere ilgili parametreleri kolayca ayarlanabilecek durumdadır. Bu projede ele alınan ve FPGA üzerinde kolaylıkla devre sentezi yapılabilen bu denetleyici topolojileri aşağıda irdelenmektedir.

2.3.1 Durum Uzayı Denetleyici Modülü

Durum uzayı denetleyicileri birçok ileri endüstriyel uygulamaya doğrudan doğruya hitap edecek denetleyici sistemleridir. Tipik bir durum uzayı denetleyicisinin sonlu fark denklemi aşağıdaki gibidir:

$$\boldsymbol{u} = -\mathbf{K}(\hat{\mathbf{x}}(k) - \mathbf{x}_{\mathbf{r}}(k) \qquad (2.1)$$

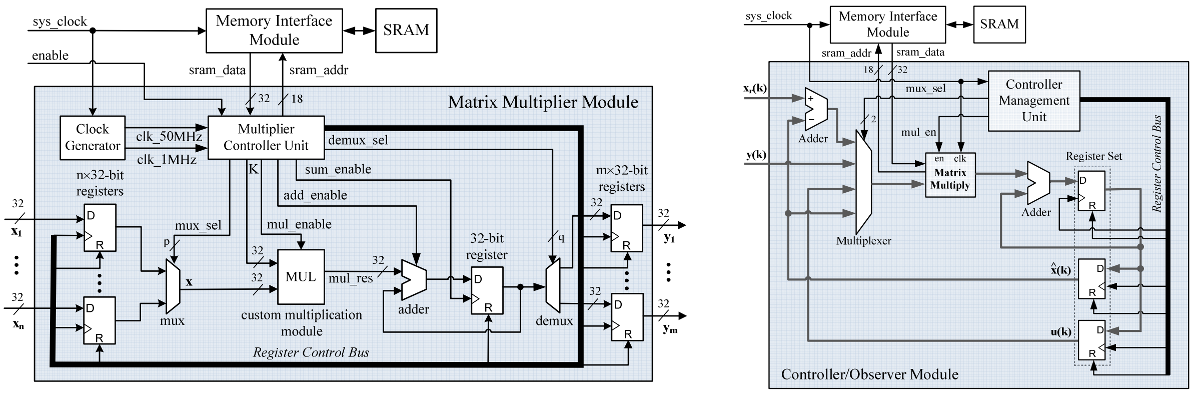
Bu denklemde **u** giriş vektörü, $\hat{\mathbf{x}}$ kestirilen (gözlemlenen) durum vektörü, $\mathbf{x}_{\mathbf{r}}$ referans durum vektörü ve **K** ise kazanç matrisidir. Sistemin durum denklemlerinin edinilmesiyle birlikte, **K** matrisindeki kazançlar modern kontrol teorisi ile istenilen kontrol karakteristiğini gösterecek şekilde ayarlanabilir. Bu noktada, gerçek uygulamalarda sistemin tüm durum değişkenlerinin ölçülmediğinin altı çizilmelidir;

dolayısıyla çoğu zaman bir durum gözlemcisi de, ölçülmeyen durum değişkenlerinin hesaplanması için tasarımda yer almaktadır. Örneğin bir Luenberger tipi durum gözlemcisi, şu şekilde ifade edilebilir:

$$\hat{\mathbf{x}}(k+1) = \mathbf{F}\hat{\mathbf{x}}(k) + \mathbf{G}\boldsymbol{u}(k) + \mathbf{L}[\boldsymbol{y}(k) - \mathbf{H}\hat{\mathbf{x}}(k)] \qquad (2.2)$$

Bu denklemde **F, G, H** sistem matrislerini, **y**(k) denetleyici çıkış vektörünü ve **L** de gözlemcinin kazanç matrisini temsil etmektedir. Endüstriyel hareket denetim sistemlerinde, sistemin durumları çoğunlukla açısal konum ve (anlık) açısal hız olarak seçilmektedir. Dolayısıyla, genel yaklaşım açısal hızın, açısal konum bilgisi kullanılarak kestirilmesidir. Literatürde, enkoder/çözücüye sahip sayısal sistemlerde hız kestirimi yapmak için kullanılabilecek kestirim algoritmaları bulunmaktadır. Öte yandan, dinamik çalışma koşullarının önemli ölçüde değiştiği sistemlerin tamamını kapsayan bir kestirim algoritması bulunmamaktadır. Dolayısıyla, sistem parametreleri için iyi kestirimler mevcutsa, elde olmayan durumlar için yüksek dereceli fark alan kestirim yöntemleri yerine, bu durumların gözlemlenmesi tercih edilmektedir.

Şekil 2.5 geliştirilen durum-uzayı denetleyicisinin genel bir sayısal devre şemasını göstermektedir. Matris çarpımı modülü (Şekil 2.5a) etrafında inşa edilen bu denetleyici klasik *çarp-ve-topla* ("multiply-and-accumulate") ilkesiyle çalışmaktadır. Matris çarpım modülü içinde değişik aritmetik işlem modüllerinin (kayan-noktalı, sabit noktalı, tamsayı) kullanılması ile, uygulamanın gerekleri doğrultusunda en uygun hesap yönteminin seçilmesi imkan dahilindedir. Şekil 2.5b'de temsil edilen genel denetleyici topolojisi SRAM'de depolanmış matrisleri uygun sırada çağırarak, (2.1) ve (2.2)'de belirtilen işlemlerin hepsini ardışık olarak gerçekleştirmektedir. Dikkat edilirse, FPGA üzerinde kullanılan kaynak miktarını azaltmak için, geliştirilen denetleyici de işlemler özellikle paralel olarak yapılmamaktadır. Bu şartlar altında bile FPGA (Cyclone II) üzerindeki hesap süreleri 1 ms'nin altında olup, (seçilen aritmetik işlem tipine bağlı olarak) kontrol çevrim frekansı 1.5 ila 400 kHz arasında değişebilmektedir. Çok yüksek çevrim frekanslarının arzu edildiği uygulamalarda; çoklu matris çarpım ünitesi kullanarak, durum-uzayı denetimleyicisini tamamen koşut hale getirmek mümkündür. Bu topolojiyle ilgili daha detaylı bilgi [1, 5] kaynaklarında bulunabilir.



(a) Matris Çarpım Modülü **(b)** Denetim/İzleme Modülü

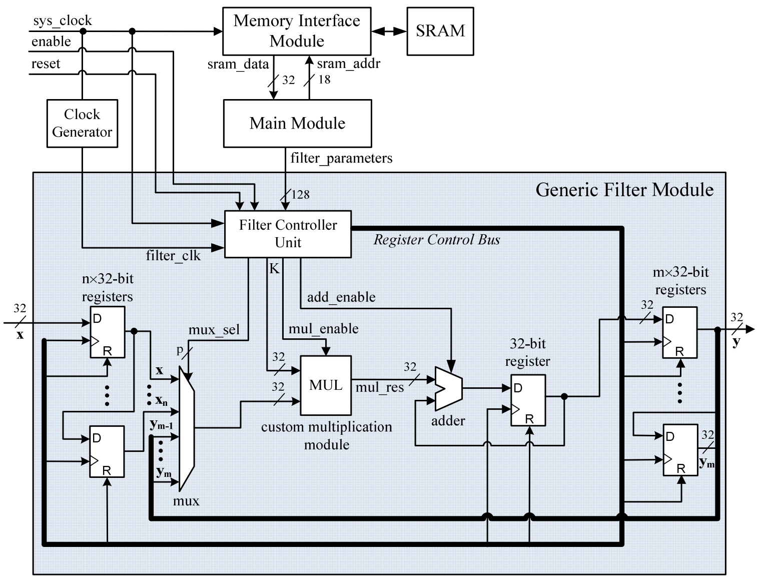
Şekil 2.5: Projede geliştirilen durum-uzayı denetleme topolojisi.

2.3.2 Sayısal Süzgeç Modülü

Sayısal süzgeçler giriş işaretindeki istenmeyen frekans bileşenlerinin ayıklanması için kullanılan (ve sayısal işaret işlemcileri vasıtasıyla gerçeklenen) genel hesaplayıcı sistemler olup, telekomunikasyon dizgelerinde yaygın yer alırlar. En genel şekliyle, bir sayısal (IIR) süzgeci

$$y(k) = \sum_{j=0}^{n} a_i y(k-i) + \sum_{j=0}^{m} b_j x(k-j) \qquad (2.3)$$

şeklinde ifade edilebilir. Burada x(k) ve y(k) sırasıyla giriş ve çıkış işaretlerini; a_i, b_j süzgeç sabitlerini; k is zaman indisini temsil etmektedir. Dikkat edilirse, (2.3)'te verilen sonlu fark denklemini kullanarak, birçok denetleme algoritmasını (P, PI, PD, PID, öncül-artçıl düzenleyici, komut ileri besleme, çentik süzgeci, vs.) hayata geçirmek mümkündür. Dolayısıyla, proje kapsamında Şekil 2.6'da verilen bir sayısal süzgeç modülü geliştirilmiştir. Söz konusu ünite 2.3.1'de anılan matris çarpım modülüne benzemekte olup, *çarp-ve-topla* ilkesine göre ardıl olarak çalışmaktadır. Tıpkı matris çarpımında olduğu gibi, kullanıcı arzu ettiği aritmetik işlem ünitesini (kayan noktalı, sabit noktalı, tam sayı) seçerek, uygulamanın gerektirdiği ölçüde bir hesaplama doğruluğunu elde edebilmektedir. Bu topolojide SRAM'de depolanan (sonlu fark denkleminin) sabitleri ardışık olarak okunup, birikimli çarpma işlemleri yapılmaktadır. Her ne kadar FPGA üzerinde sentezi yapılan devrede işlemler (klasik bir sayısal işaret işlemcide olduğu gibi) ardışık olarak yapılsa da, elde edilen hesaplama süreleriyle 100 kHz'in üzerinde kontrol-çevrim frekanslarını sağlamak mümkündür. Dolayısıyla, işlemlerin koşut olarak yapılmasına pratik olarak gerek duyulmamaktadır. Söz konusu modülün tasarımı, FPGA üzerinde gerçeklenmesi ve sınamasıyla ilgili ayrıntılı bilgiler [1]'de verilmektedir.



Şekil 2.6: Projede geliştirilen sayısal süzgeç (IIR filtre) topolojisi.

2.3.3 Diğer Denetim Topolojileri

Proje kapsamında daha gelişkin denetleyicilerin tasarımına ve FPGA üzerinde uygulanmasına ağırlık verilmiştir. Bu bağlamda, ikinci dönemde geliştirilmeye başlanan denetleyiciler iyileştirilmiş ve proje kapsamında geliştirilen çevrimiçi donanım benzetimi sistemi ile sınanmıştır. Bahsi geçen denetleyiciler şu şekilde sıralanabilir:

- Bulanık mantık denetleyicisi
- Kayan kipli denetleyici
- Ardıl izlem ("hysteresis") denetleyicisi

Bu denetleyiciler mantık seviyesinde tasarım ile geliştirilmiş ve tasarımda (NIOS II gibi) gömülü bir işlemci kullanılmamıştır. Kayan-noktalı işlemlerin gerçekleştirilmesi amacıyla Usselmann tarafından geliştirilen ve ikinci gelişme raporunda da anlatılan bir kayan-noktalı işlem modülü (FPU) kullanılmıştır. Bu denetleyicilerin FPGA üzerindeki başarımı, **çevrimiçi donanım benzetimi** (ÇDB) yöntemiyle "asenkron elektrik motoru ve ona bağlı bir iş-milinden oluşan torna tezgahı" modeli ile sınanmış ve sonuçlar [1, 6]'da sunulmuştur.

2.4 Komut Üretimi ve Enterpolasyon Teknikleri

Hareket denetleme sistemleri için; komut üretimi, denetlenen eksenlerinin hareketinin tanımlayarak, denetim sisteminin periyodik olarak ihtiyaç duyduğu referans işaretlerinin oluşturulması olarak nitelendirilebilir. Birçok hareket denetim kartı ve CNC birimlerinde, makinanın yörüngeleri RS274-D (ISO 6893) tabanlı bir dil ile tanımlanır. Bu dil aracılığıyla doğrusal ve dairesel parçalar içeren karmaşık yörüngeleri programlamak mümkündür. Son on yılda, CNC üreticileri gelişen ihtiyaçlara bağlı olarak, bu dile; parabol, spline ve NURBS gibi gelişkin eğri/yüzey tanımlama komutlarını da eklemişlerdir. Dikkat edilirse, yörüngenin yüksek seviyeli bir dil ile tanımlanmasının ardından, bu dil yorumlanarak yörünge üzerinde (denetim sisteminin gereksinim duyduğu) referans aradeğerlerin gerçek-zamanlı olarak üretilmesi gereklidir. Bu işlem doğrusal ve dairesel parçalar için nispeten kolay olsa da, spline veya NURBS ile betimlenen karmaşık yüzeylerin aradeğerleme işlemi oldukça kapsamlıdır ve işlemciye büyük miktarda bir hesaplama yükü getirmektedir.

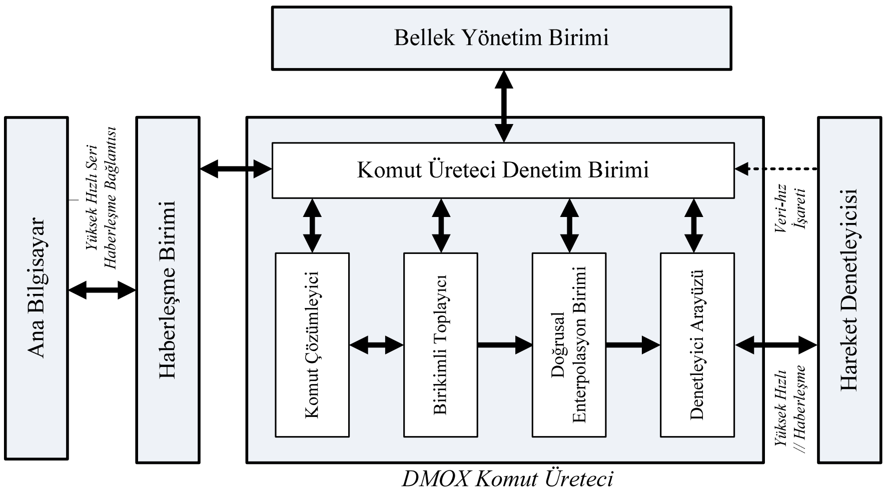
Bu projede benimsenen yaklaşım denetleyiciler için gerekli komutların "off-line" olarak oluşturulmasına ve bilginin doğrudan kullanılmasına dayalıdır. Bu teknikte büyük boyutlu verilerin ortaya çıkacağı şüphesizdir. Diğer taraftan, günümüz teknolojisinde, büyük kapasiteye (1GB++) sahip bellekler (SDRAM, SRAM, SD kartları, flash bellekler vs.) nispeten düşük fiyatlarla temin edilebilmektedir. Dolayısıyla, bu tip cihazların ve FPGA teknolojisinin tüm avantajlarını kullanarak, son derece verimli komut üreteçleri tasarlanma potansiyeli bulunmaktadır. Şekil 1.1 bu projede geliştirilen evrensel denetleme sisteminin mimarisini ve geliştirilmekte olan komut üretecinin bu mimari içindeki yerini göstermektedir. Bu projede üç değişik komut üretim yöntemi ele alınmıştır:

- Polinoma dayalı fonksiyonel yaklaşıklama ("approximation") teknikleriyle (Chebyshev, Legendre, Bernstein/Bezier, vb.) komut dizinlerin temsili,
- Komut dizinlerinin bölümlenmesi, bölümlerin polinoma dayalı tekniklerle temsili, (sıkıştırılmış olarak saklanan) yaklaşıklama hatalarının üretilen seriye eklenerek hatasız olarak komut üretimi,
- Yüksek dereceden sonlu farkı alınmış konum bilgisinin kayıpsız olarak sıkıştırılması.

Bu tekniklerin özellikleri, FPGA üzerinde gerçeklenmesi, karşılaştırmalı başarımları daha önceki gelişme raporlarında kısaca irdelenmişti. Proje kapsamı içinde yürütülen araştırmalarımızın sonucunda, üçüncü tekniğin diğerlerine göre önemli avantajlar sunduğu belirlenmiş, bu yönteme dayalı bir komut üreteci modül geliştirilmiştir. Aşağıda bu modül detaylı olarak ele alınmıştır.

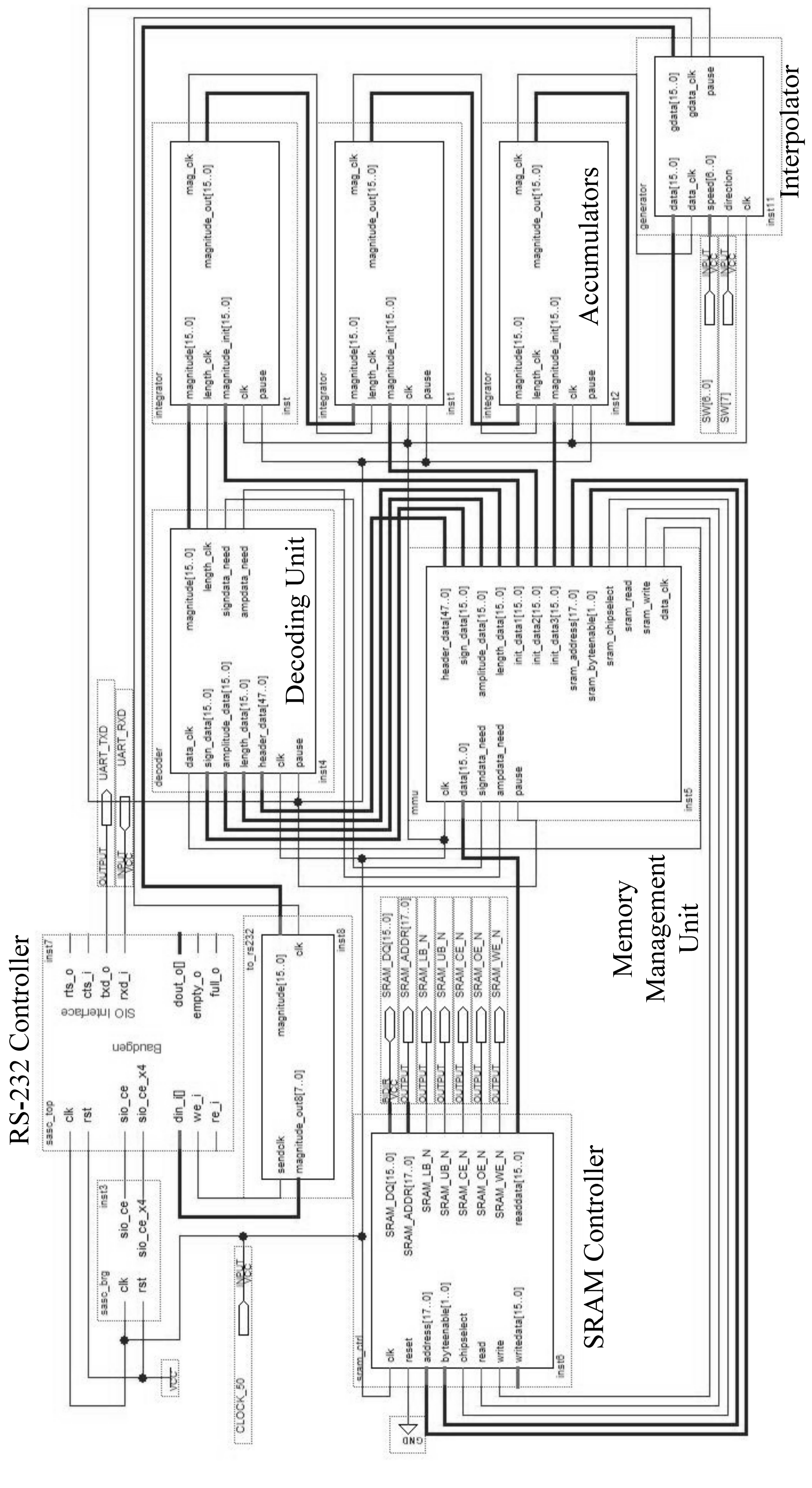
2.4.1 Komut Üreteci Modül

Şekil 2.7 bu proje kapsamında geliştirilen komut üreteci modülün blok diyagramını göstermektedir. Bu sistemde ana bilgisayar tarafından (yüksek dereceden) sonlu farkları alınarak ΔY tekniğiyle sıkıştırılan komut dizinleri, yüksek hızlı seri bir haberleşme bağlantısı üzerinden komut üretecine aktarılır. DMOX'un belleğinde sıkıştırılmış olarak tutulan veriler ΔY çözümleyicisi tarafından sonlu farklara dönüştürülür [4]. Birimdeki birikimli toplama birimi ("accumulator") bu sonlu farkları tümleyerek orijinal yörünge bilgisini elde eder. Dikkat edilirse, burada yer alan doğrusal enterpolasyon birimi yörünge üzeride farklı hızlarda (ve yönlerde) ilerlerken komut verisi üretmeye imkan tanımaktadır. Özellikle, CNC takım tezgahında programlanmış hareket sırasında takımın hızı operatör tarafından keyfi olarak ayarlanabilmektedir. Benzer olarak, CNC elektro-erozyon (EDM) tezgahlarında elektrodun hızı ve yönü programlanmış yörünge boyunca sürekli değişmektedir. Dolayısıyla, ortaya konan bu mimari bu tür gereksinimleri doğal olarak karşılayabilmektedir.



Şekil 2.7: Projede geliştirilen komut üreteci birimin blok diyagramı.

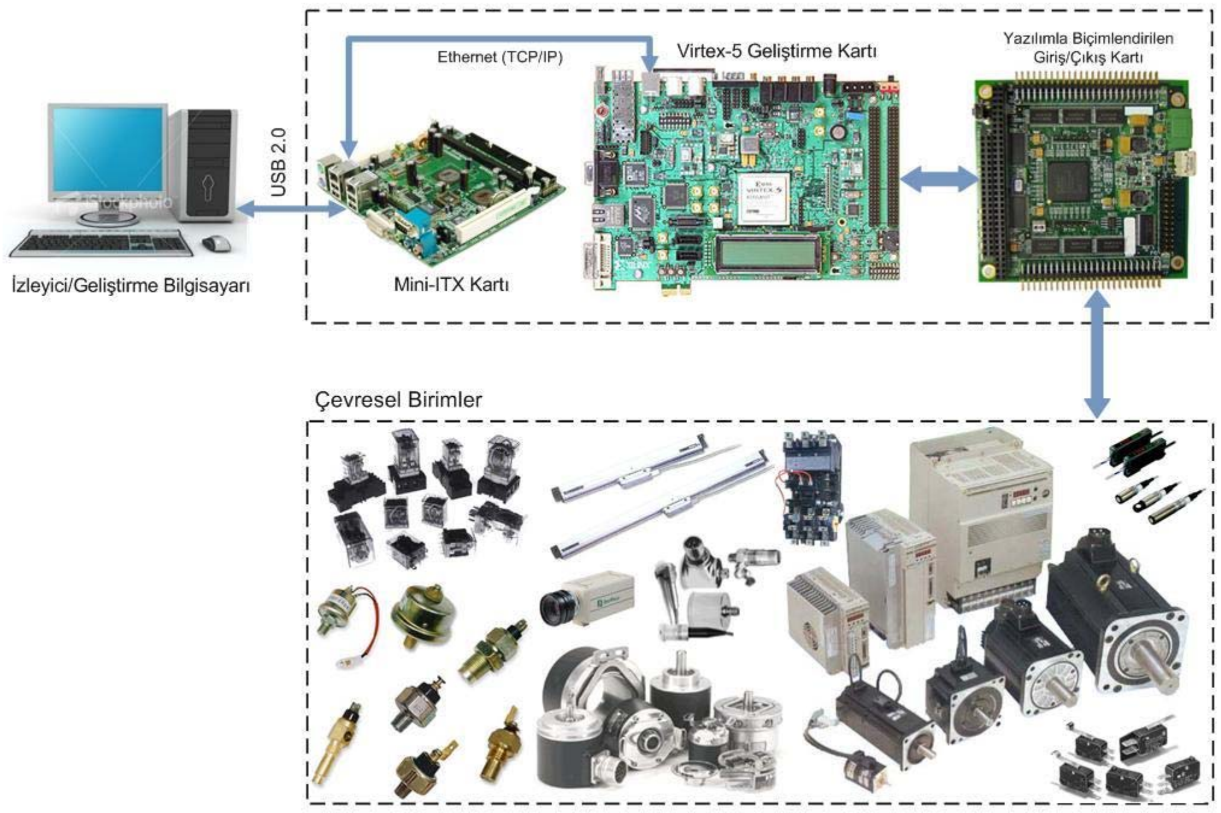
Dikkat edilirse, önerilen modül ilgili literatürde yer alan herhangi bir veri sıkıştırma tekniğini (Huffman, Aritmetik kodlama, LZW, vs) kullanabilmesine imkan tanımaktadır. Diğer taraftan, ΔY sıkıştırma tekniği bu proje kapsamında geliştirilmiş yeni bir yöntem olup, veri sıkıştırma literatüründe bağıl kodlama (“relative encoding”) olarak anılan bir sınıf içinde yer almaktadır. Söz konusu teknik, uygulama kolaylığı ve veri sıkıştırma oranları göz önüne alındığında; özellikle Huffman ve Aritmetik kodlama tekniğine göre ön plana çıktığından, özellikle tercih edilmiştir. Bu teknikle ilgili ayrıntılı bilgi [3, 4]’te bulunabilir. Şekil 2.8’de FPGA üzerinde devre sentezi yapılmış komut üreteci (sonlu durum makinası) gösterilmektedir. Söz konusu üreteç (50 MHz’de çalışan) Cyclone II üzerindeki toplam mantıksal devre elemanların yaklaşık %9’unu kullanmakta olup, herbir komutun üretimi sadece 0.5 μs almaktadır.



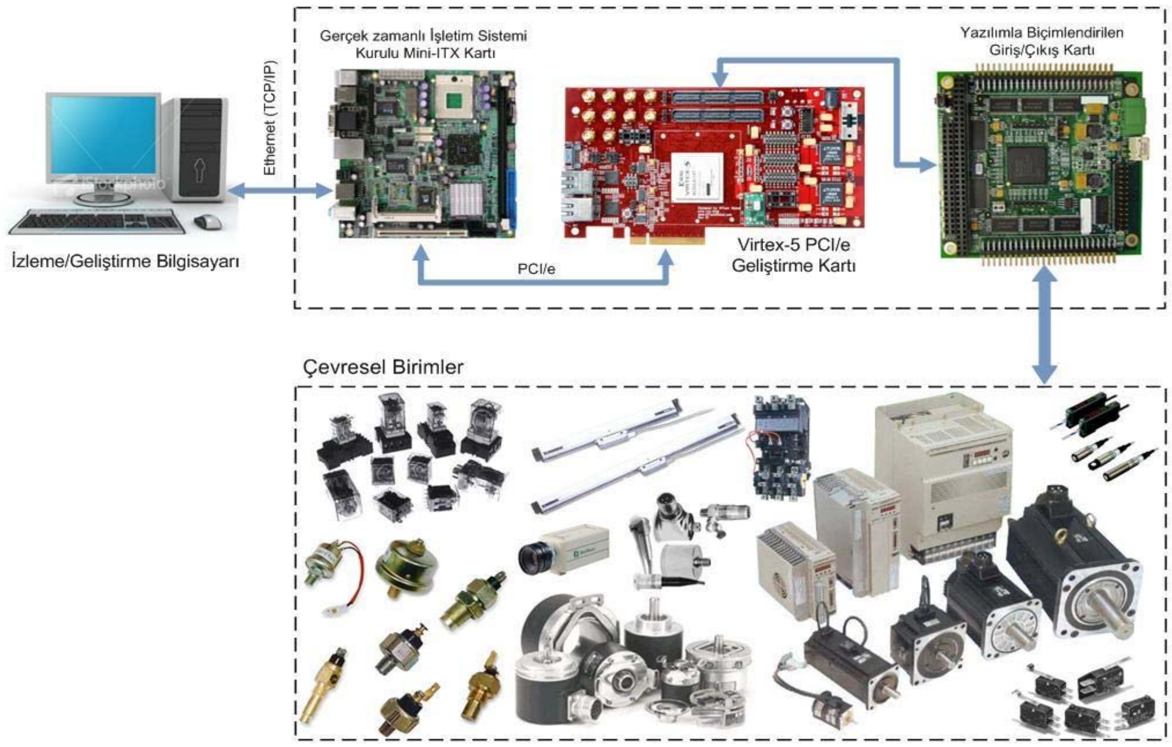
Şekil 2.8: Quartus II ortamında tasarlanan komut üretecinin (ve onu destekleyen birimleriyle beraber) şeması.

2.5 PC Tabanlı Denetim Mimarileri

Hatırlanacağı üzere, bu proje kapsamında iki değişik PC tabanlı denetim mimarisinin (Şekil 2.9 ve 2.10) araştırılması öngörülmekteydi. Proje kapsamında ele alınan ilk sistemde (Şekil 2.10); RT-Linux yada Windows CE gibi gerçek-zamanlı bir işletim sistemi kurulu Mini-ITX (veya Nano-ITX) denetim bilgisayarı, hem hareket komutlarını üretececek hem de gerçek zamanda denetim işlemini gerçekleştirecekti. Hızlı PC iletişim protokolü PCI üzerinden; bilgisayar, FPGA tabanlı arayüz kartıyla çok hızlı (senkron) iletişim kurarak, çevresel birimlerle (kayda değer veri aktarım gecikmeleri olmadan) haberleşmeyi/koordinasyonu sağlamış olacaktı. Bu biçimde, FPGA yalnızca çevresel birimlerden gelen işaretleri işleyip / dönüşüm işlemini icra edeceği düşünülmekteydi. Ayrıca, alt seviyeli (olaya tepki tabanlı) bir takım denetim işlemlerinin bu sistem tarafından yürütülmesi de öngörülmekteydi.


Ethernet (TCP/IP)
Virtex-5 Geliştirme Kartı
Yazılımla Biçimlendirilen Giriş/Çıkış Kartı
USB 2.0
İzleyici/Geliştirme Bilgisayarı
Mini-ITX Kartı
Çevresel Birimler

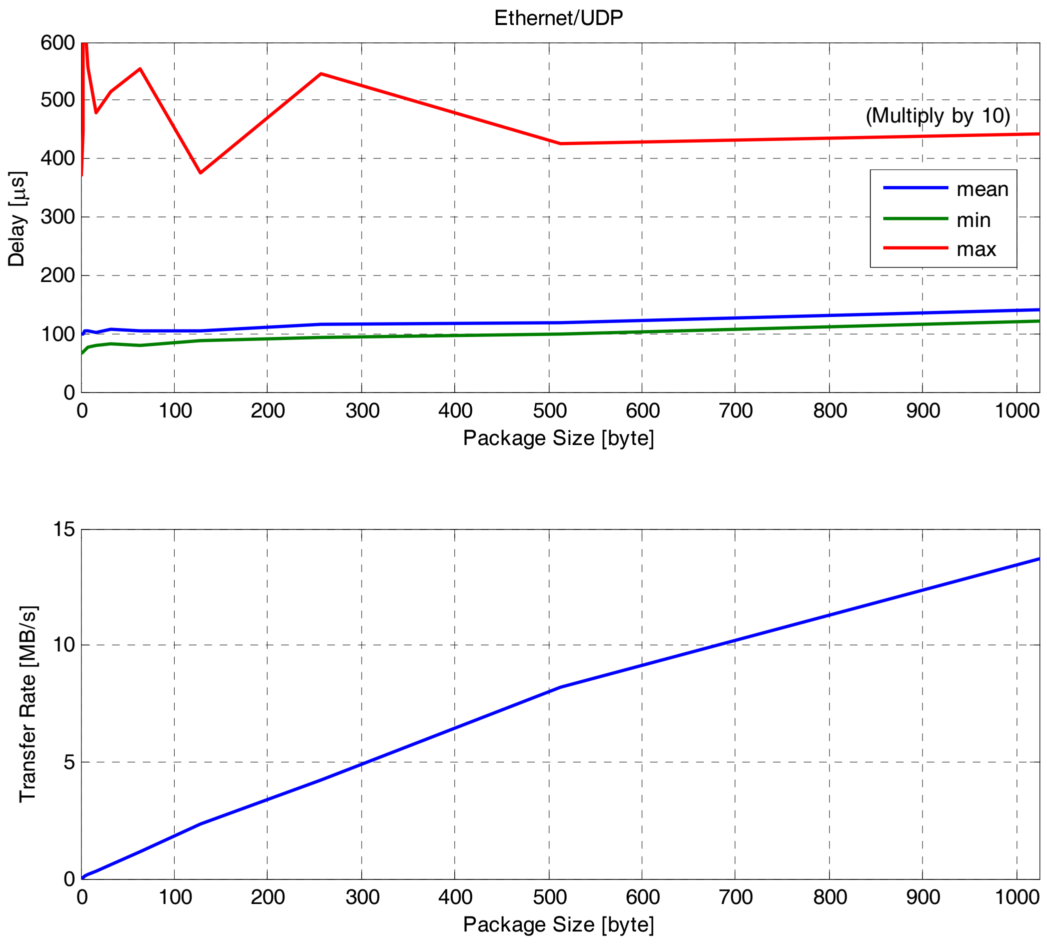
Şekil 2.9: DMOX Hareket Denetleme Kartı Prototipi (A)


Gerçek zamanlı İşletim Sistemi Kurulu Mini-ITX Kartı
Yazılımla Biçimlendirilen Giriş/Çıkış Kartı
Ethernet (TCP/IP)
İzleme/Geliştirme Bilgisayarı
PCI/e
Virtex-5 PCI/e Geliştirme Kartı
Çevresel Birimler

Şekil 2.10: DMOX Hareket Denetleme Kartı Prototipi (B)

Proje kapsamında, üzerinde Windows Embedded CE 6.0 R2 kurulu bir Mini-ITX ile ön-sınamalar yapılmış, sistemin zamanlayıcı kesme isteklerine önemli bir gecikme olmadan yanıt verebildiği görülmüştür. Dolayısıyla, çok yüksek örnekleme frekanslara çıkmadan (< 10 kHz) Windows CE ile gerçek-zamanlı denetim yapılmasının mümkün olduğu anlaşılmıştır. Diğer taraftan, projede kapsamında (Virtex-5 LXT FPGA ML505 geliştirme kartı için) satın alınan Xilinx ISE Design Suite 11 paketinde PCI ile ilgili düşünce iyeliklerinin olmaması nedeniyle, böylesi bir iyeliğin proje kapsamında geliştirilmesi gereği ortaya çıkmıştır. PCI (veya PCI/e) protokolünün çok geniş kapsamı ve ISE Design Suite içinde rastlanan bariz yazılım hataları ("bug"lar) nedeniyle yaşanmakta olan sorunlar dikkate alındığı zaman, böylesi bir çabanın projenin zamanı ve sınırlı kaynaklarını tüketeceği anlaşılmış, bu gibi bir çalışmadan vazgeçilmiştir.

Alternatif olarak, Ethernet (TCP/IP veya UDP) gibi yüksek hızlı bir seri-haberleşme kanalı ile söz konusu iletişimin yapılması gündeme gelmiştir (ki Ethernet için açık kaynak düşünce iyeliklerini bulmak mümkündür). Her ne kadar TCP/IP asenkron bir seri iletişim protokolü olsa da, iletişim sırasında gecikmeler örnekleme periyodunun beşte birinden az olduğu durumlarda bu gecikmenin kontrol sisteminin dinamiğine önemli bir etkisinin olmadığı varsayılabilir. Bu amaçla, Ethernet/UDP protokolünün iletişim hızını ölçebilmek için basit bir ön-test gerçekleştirilmiştir. İki adet PC Ethernet üzerinden birbirine bağlanmış, değişik büyüklükteki veri paketleri (1 ila 1024 bayt) bir bilgisayardan diğerine gönderilip, orijinal veri paketi alıcı bilgisayar tarafından doğrudan doğruya gönderen tarafa aktarılmıştır. Veri paketinin gönderimi ve alımı arasında geçen zaman kaydedilmiştir. Testler bir dakika boyunca sürekli olarak tekrar edilmiş olup, yaklaşık 400 ila 550 bin veri (süre) toplanmıştır. Şekil 2.11'deki üstteki grafik; asgari, ortalama ve azami zaman değerlerini veri paketinin büyüklüğünün fonksiyonu olarak göstermekte olup, aşağıdaki grafik ise görünür veri aktarım hızını temsil etmektedir. Görüleceği üzere, bu iletişim protokolüyle oldukça hızlı bir veri aktarımı yapmak mümkündür. Diğer taraftan, ham verinin işlenerek belli bir formattaki veri paketleri haline getirilmesi gerektiğinden; Ethernet denetleyicisi (veri paketi başına) yaklaşık 100 μs'lik ortalama bir zamana ihtiyaç duymaktadır. Sonuç olarak, Ethernet/UDP kullanıldığında, en iyi şartlar altında bile (PC ile) kontrol çevrim frekansının 2 kHz üzerine çıkması oldukça zordur. Ticari hareket denetleme kartlarında servo-çevrim frekanslarının 30 kHz ... 160 kHz arasında olduğu dikkate alınırsa, böylesi bir örnekleme (veya benzetim) frekansı birçok ticari uygulama için çok yetersiz kalmaktadır.



Şekil 2.11: Ethernet/UDP'nin haberleşme özelliği

Bu proje kapsamında ele alınan ikinci sistemde (Şekil 2.9) Windows Vista gibi gerçek-zamanlı olmayan bir işletim sistemi kurulu bilgisayar; temel olarak hareket komutlarını üretip (kodlayıp), bunları Ethernet (hızlı asenkron seri iletişim protokolü) üzerinden seri olarak, FPGA tabanlı arayüz kartına göndermektedir. Bu mimaride; FPGA tabanlı kart, hem gerçek-zamanlı olarak denetleme işlemini gerçekleştirecek hem de çevresel birimlerden gelen işaretleri işleyerek, yararlı bilgilere dönüştürmektedir. Proje kapsamında özel olarak tasarlanan giriş/çıkış kartı çeşitli çevresel birimlerden gelen işaretleri koşullandırarak, FPGA kırmığının ilgili ayaklarına (portlarına) yönlendirmektedir. Teknik olabilirliği yüksek bulunan bu mimari, projede ilerleyen aşamalarında ön plana çıkmıştır.

2.6 Çevrimiçi Donanım Benzetimi

Geliştirilen evrensel denetleyicisinden elde edilecek başarımın nesnel bir şekilde değerlendirilebilmesi için, çeşitli uygulama senaryoları üzerinde sınanması gerekmektedir. Her ne kadar ilk etapta geliştirilen hareket denetleyiciler basit deney düzenekleri üzerinde sınanmış olsa da; bu proje çerçevesinde ele alınan sistemin uygulama alanınının genişliği göz önünde bulundurulduğunda, sağlıklı sonuçlara ulaşmak için gerekli sınamaların çok değişik sistemler üzerinde yapılması gerektiği açıktır. Ancak, sağlıklı sonuçlar elde edilecek çoklukta sistem üzerinde çalışmak bütçe, zaman ve işgücü açısından projenin kapsamını aşacaktır. Buna karşılık bu sınamaların gerçekleştirilmesi, projenin amacına ulaşıp ulaşmadığını değerlendirmede önemli bir ölçüt olarak görülmektedir. Bu nedenle, geliştirilen kartın sınanması için ÇDB yönteminin kullanılması gündeme gelmiştir. Bu yöntemde, geliştirlen denetleyici donanım, gerçek sisteme bağlamak yerine; sistemin dinamiğini ileri derecede taklit edebilecek bir bilgisayar sistemine bir arayüz üzerinden bağlanır. Denetleyici tarafından bakıldığında, gerçek sisteme bağlanmış olmak ile benzetim bilgisayarının arayüzüne bağlanmak arasında temelde bir fark bulunmamaktadır. Yapılan benzetim esas itibariyle sistemin davranışını betimleyen diferansiyel denklemlerin uygun sayısal yöntemler aracılığıyla çözülmesiyle gerçekleşir. Bu benzetim üzerinde gerçek uygulamalarda ortaya çıkabilecek birçok koşul ve etken taklit edilebilir ve çok sayıda sınama kolaylıkla gerçekleştirilebilir. Bu imkanların yanı sıra, denetleyicinin başka bir sistemde kullanılması istendiğinde bu sistemi betimleyen denklem ve parametrelerin benzetim yazılımına tanıtılması yeterlidir.

2.6.1 FPGA tabanlı gerçek-zamanlı ÇDB

2.6.1.1. Koşut Çözüm

Benzetim işlemlerinde kullanılmak üzere, bir modül (cebirsel düzenlemelerden geçirilerek elde edilen) sabit katsayılı fark denklemlerinin yüksek hızda çözülmesi için geliştirilmiştir. En genel haliyle, bir doğrusal sistemin sonlu fark denklemi (2.3) şeklinde ifade edilebilir. Dikkat edilirse, denklemin y(k) (çıkış) değerini hesaplamak için önceki zaman adımlarındaki değerlerin sabit birer katsayı ile çarpılması gerekmektedir. Daha sonra eldeki bütün çarpımların birbiriyle toplanarak, sonucun hesaplanması mümkündür.

Geliştirilen çözücü modül, bu işlemleri yüksek hızda gerçekleştirebilmek için koşut (paralel) işlem gücünden faydalanır. Modül içerisinde gerçekleştirilecek her işlem için özelleştirilmiş bir kayan-noktalı (FP) hesaplama alt modülü kademeler halinde yer almaktadır. Bunların yanında tüm elemanların aynı anda okunabilir olduğu, FIFO (*"First In, First Out" – 'İlk Giren İlk Çıkar'*) yapısında tampon bellek modülleri bulunmaktadır. Bu belleklerde denklem girdileri ve sonucunun geçmiş değerleri, denklemin ihtiyaç duyduğu zaman adımına kadar tutulur. Bütün bu alt modüller zamanlamayı yöneten bir denetleyici alt modül tarafından yönetilirler. Denklemin çözülmesi için ilk olarak *Kademe 1*'de bulunan FP çarpma alt modülleri eş zamanlı işletilerek, çarpım işlemlerinin sonuçları elde edilir. Bu sonuçların toplanması için önce *Kademe 2*'de bulunan FP toplama alt modülleri yine eş zamanlı yürütülürler. Ardından geriye kalan kademelerdeki alt modüller de kademe sırasıyla icra edilerek sonuca ulaşılır. Son olarak girdiler ve elde edilen sonuç, ilgili tamponlara belleklere gönderilerek bir sonraki zaman adımında yapılacak hesaplama için ihtiyaç duyulan değerler kaydedilmiş olur.

Projede yapılan çalışmalar sonucunda, benzetimi yapılan sisteme dair durum denklemlerinin doğrudan donanım üzerinde çözülmesi için gereken kaynak miktarının, denklem mertebesine bağlı olarak hızla arttığı ortaya çıkmıştır. Özellikle karmaşık sistemlerin benzetiminin gerçekleştirilmesi gerektiğinde eldeki donanım kaynaklarının hızla tükeneceği görülmüştür. Bir örnek vermek gerekirse, beşinci mertebeden bir sonlu fark denkleminin (sözü edilen modülle) paralel olarak hesabı sadece 340 ns (17 saat çevrimi) içinde yapılabilse de, böylesi basit bir işlem için FPGA kırmığı (Altera Cyclone II 2C20) üzerindeki kaynakların (mantık elemanlar, yazmaçlar, gömülü çarpma birimleri vs.) yaklaşık %40'ı kullanılmaktadır. Bu nedenle benzetimin gerçeklenmesi aşamasında bir mikroişlemciden faydalanılması gereği ortaya çıkmış ve bazı hesapların seri (ardışık) olarak yapılmasının daha makul bir seçenek olacağı anlaşılmıştır.

2.6.1.2. Gömülü İşlemciyle Çözüm

Ortaya çıkmış bulunan mikroişlemci kullanımı ihtiyacını karşılamak amacıyla, FPGA kırmığı içerisinde kullanılabilmesi için Altera tarafından geliştirilmiş 32 bitlik NIOS II gömülü işlemcisinden faydalanılmıştır. Bu işlemci tasarımı dahilinde kayar-noktalı aritmetik birimi bulunup, kart üzerinde bulunan çevresel birimlere (SRAM, SDRAM, Flash bellek, USB Blaster kırmık seti) modüler arayüzler sayesinde bağlanması mümkündür. NIOS II işlemcisi ile benzetimde ihtiyaç duyulan modül ve çevre birimler Altera tarafından sağlanan geliştirme gereçleri aracılığıyla birleştirilerek bir SOPC ("system on programmable chip"/programlanabilir kırmık üzerinde sistem) oluşturmak imkan dahilindedir. Bu şekilde, benzetimi yapılacak sistemin hesaplamalarını gerçekleştirmek üzere bir C programı yazılarak NIOS II işlemci için derlenebilmektedir. Yani, sistem değişkenlerini yöneten durum denklemleri sabit katsayılı fark denklemlerine dönüştürülerek çözücü yordamlar oluşturulabilir. Bu C programı; iletişim ve hesaplama görevlerini sürekli olarak yürüten bir döngüye sahiptir. Bu döngü sürekli olarak çalıştırılmak yerine, her örnekleme zamanı için denetleyiciden bir düzeltme komutunun gelmesi beklenir. Komut elde edildiğinde işlemcide bir kesme isteği oluşur ve buna cevaben gelen komut giriş/çıkış arayüzünden okunarak benzetim hesaplamaları bir örnekleme periyodu için gerçekleştirilir. Hesaplamalar sonucu elde edilen sonuçlar, duyuculara dinamik davranışını modelleyen modüllere aktarılır. Böylece denetleyici tarafından sonraki periyotta örneklenmesi beklenen ölçüm değerleri hazır edilmiş olur.

Sözü edilen bu yöntem bir tork denetimli bir asenkron motorun benzetimi yoluyla sınanmıştır ve elde edilen sonuçlar [7]'de sunulmaktadır. Bu sonuçlar ışığında, ikinci dereceden bir sonlu fark denklemi için ortalama hesap süresi yaklaşık 112 μs olmakta, FPGA kaynaklarının yaklaşık yarısı (ki bunun önemli bölümü NIOS II işlemcisine tahsis edilmektedir) bu uygulamada kullanılmaktadır. Bu durumda denetleme-çevrim frekansı 10 kHz gibi nispeten düşük bir değerin altında kalmak durumundadır.

Dikkat edilirse, (2.3) formunda sonlu fark denklemleri kullanılarak sadece doğrusal dinamik sistemlerin (ayrık zamanlı) benzetimi yapılabilmektedir. Diğer taraftan, uygulamadaki birçok sistemin dinamik davranışı doğrusal olmayan adi diferansiyel denklem takımlarıyla betimlenebilmektedir. Dolayısıyla, proje kapsamında böylesi diferansiyel denklemlerin çözümü için 4. mertebeden Runge-Kutta (RK4) yöntemine dayalı bir çözücü ("solver") NIOS II üzerinde gerçekleştirilmiştir. [2, 7] böylesi bir çözücünün başarımını irdelemiş, RK4 gibi çözücülerle ÇDB'nin Cyclone II üzerinde 2kHz'e varan örnekleme frekanslarında başarıyla gerçekleştirilebileceği ortaya konmuştur. Her ne kadar böylesi bir düşük örnekleme frekansları birçok gerçek-zamanlı ÇDB için yetersiz olsa da, daha gelişkin FPGA kırmıkları (Cyclone III, Virtex 6) ve ileri gömülü RISC işlemci (örneğin ARM Cortex) iyelikleri kullanımıyla benzetim hızını 10 ila 100 kata kadar arttırmak olası görülmektedir.

2.6.2 AVR32 İşlemci Platformu

FPGA kırmığı üzerinde gerçekleştirilen benzetim esnasında, bir mikroişlemci kullanımının kaynak kullanımı ve çevre birimlere erişim açısından oldukça avantajlı olduğu görülmüştür. Diğer yandan, enkoder üreteci gibi çeşitli birimlerin doğrudan donanım üzerinde gerçeklenmesi; ihtiyaç duyulan yüksek başarımı (yazılıma kıyasla) daha kolay sağlamaktadır. Bütün bu bulgular ışığında, çevre birim öykünmesi FPGA üzerinde gerçekleştirilmeye devam edilirken, benzetim hesaplarının gerçek bir mikroişlemci üzerine aktarılması tercih edilebilir bir çözüm haline gelmiştir. Bu düzenlemede FPGA kırmığını bulunduran donanım yalnızca çevre birimlerin benzetimi ve denetleyiciyle arayüz oluşturma görevleriyle yükümlüyken, mikroişlemci barındıran donanım ise esas benzetim hesaplamalarını gerçekleştirmekte ve kullanıcı ile PC üzerinden iletişim kurmaktadır.

Söz konusu çözümde kullanılmak üzere Atmel tarafından tasarlanmış AVR32 mimarisine sahip AT32AP7000 sayısal işaret işlemcisi ve bunu barındıran bir uygulama kartı olan NGW100 kiti seçilmiştir. Bu platform üzerinde 100MHz saat hızında çalışan işlemcinin yanı sıra 32MB SDRAM bellek, 16MB Flash bellek, SD ve MMC bellek kartı girişi, iki Ethernet bağlantısı ile RS-232 ve USB seri bağlantı girişleri bulunmaktadır. Bunların yanında 63 adet genel amaçlı giriş/çıkış bacağı kullanım amacına göre sayısal giriş/çıkış olarak kullanılabilmekle beraber işlemci ve kart üzerinde bulunan zamanlama, PWM işareti üretme, LCD ekran sürme, SPI iletişimi gibi işleri gerçekleştirebilen çevre birimlere yönlendirilmesi de mümkündür.

Platform üzerinde üretici tarafından özelleştirilmiş bir Linux 2.6 çekirdeği koşmaktadır. Kit üzerindeki tüm kaynaklara erişim, bu çekirdeğe dahil edilmiş sürücüler aracılığıyla mümkündür. Kitin incelenmesi ve gerekli yazılım gereçlerinin elde edilmesi aşamasından sonra kart üzerinde bir çevrimiçi donanım benzetiminin gerçeklenmesine ilişkin ön çalışmalar yapılmıştır. Ethernet ve RS-232 protokolleri üzerinden kit ile iletişim sorunsuz olarak gerçekleştirilmiştir. Kit üzerindeki girişe takılı 256MB kapasiteli bir SD bellek kartı üzerinde okuma ve yazma işlemleri yapılmıştır. Çekirdek üzerinde çalışan bir sunucu aracılığıyla kit üzerindeki depolama elemanlarına dosya aktarımında alışılmış bir yöntem olan FTP (“file transfer protocol”/dosya aktarım protokolü) ile halihazırda erişim sağlanabilmektedir.

Bir benzetim işinin kitle gerçekleştirilmesi konusunda ön inceleme olarak, daha önceden FPGA üzerinde devre sentezi yapılan NIOS II mikroişlemcisiyle yapılmış olan sabit katsayılı fark denklemi çözümü NGW100 kiti üzerinde tekrarlanmış ve AT32AP7000 işlemcisinin başarımı ölçülmüştür. Bu incelemelerin sonucunda söz konusu çözümlerin 10. mertebeden bir fark denklemi için ortalama 26µs içerisinde tamamlanabildiği görülmüştür. Ortalama değerlerin hayli üstünde (10. mertebe için 0.7ms) çıkan azami hesaplama süreleri dolayısıyla daha detaylı inceleme yapıldığında bu sürelerin benzetim başarımını etkilemeyecek kadar seyrek (on binde 5’ten daha az) oluştuğuna kanaat getirilmiş ve işlemci başarımının benzetim işlemleri için yeterli olduğu kabul edilmiştir. Bu sistemin özellikleri ve kapsamlı testleriyle ilgili bilgiler [2]’de bulunabilir.

2.6.3. PC tabanlı gerçek-zamanlı olmayan ÇDB

Bölüm 2.5’te yapılan çalışmalar sonucunda, gerçek-zamanlı olmayan benzetim çalışmalarının kişisel bilgisayarla yapılabileceği görülmüştür. Her ne kadar gerçek zamanlı olmayan bir ÇDB endüstriyel uygulamalarda pek tercih edilmese de, bu tür bir çözümün özellikle eğitim sektörüne önemli katkılarda bulunacağı anlaşılmıştır. Bu kapsamda, “CADMUS” isimli bir yazılım geliştirilmiştir. Denetleyici donanımla RS-232 seri port üzerinden belirli bir protokol kullanarak iletişim kurabilen bu yazılım, benzetimi yapılan sistemi üç boyutlu katı modeller aracılığıyla görselleştirebilmektedir. Ayrıca, sisteme ait birçok parametre kullanıcı arayüzünden izlenebilmekte ve değiştirilebilmektedir. Yazılımın bir diğer özelliği, benzetimi bağımsız olarak gerçek zamanlı yapmak yerine denetleyici donanımdan gelen sinyalleri saat kaynağı olarak kabul etmesi ve denetleyicinin gerçek zamanlı çalışıp çalışmamasından bağımsız olarak benzetim zamanını buna göre ölçeklendirebilmesidir. Söz konusu yazılımın bir uygulaması ME 534 (Bilgisayar Denetimli Makinalar) dersi kapsamında ODTÜ Makina Mühendisliği bölümünde yapılmış, oldukça başarılı sonuçlar elde edilmiştir. Bu yazılım ve elde edilen sonuçlarla ilgili bilgiler [2, 12, 13]'de verilmektedir.

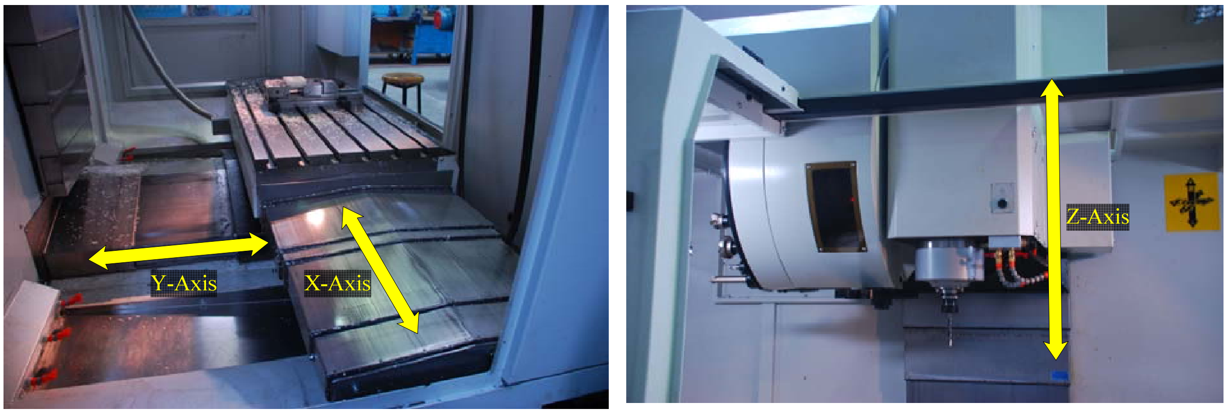
3. GELİŞTİRİLEN DENETİM SİSTEMİNİN SINANMASI

Ortaya konan evrensel denetleyici sisteminin bir bütün olarak başarımının incelenmesi için bu proje kapsamında üç ayrı test gerçekleştirilmiştir. Bu testler ve elde edilen sonuçlar aşağıda irdelenmektedir.

3.1 ÇDB yoluyla Dikey İşleme Merkezinin Denetimi

İlk olarak bu projenin en öncelikli uygulama alanlarından biri olan CNC takım tezgahlarının denetimi incelenmiştir. Ancak, gerçek bir takım tezgahı üzerinde deneysel bir çalışma yapmak Laboratuarımızın imkanları dahilinde olmadığından; bu proje kapsamında geliştirilen DMOX'un ÇDB ile sınanması yoluna gidilmiştir. Bu amaçla, Orta Doğu Teknik Üniversitesi Makina Mühendisliği Atelyesinde yer alan bir CNC işleme merkezi referans olarak ele alınmıştır. Long Chang Makina Ltd. Şti. (Tayvan) tarafından üretilmiş bu üç eksenli CNC dikey işleme merkezi (First MCV-1100) Şekil 3.1'de gösterilmektedir. Söz konusu takım tezgahı üzerinde otomatik takım değiştirici, dönel tabla, soğutma sıvısı dolaşım sistemi, kızak yağlama düzeneği ve talaş tahliye sistemi bulunmakta olup; Şekil 3.2 makinanın temel hareket eksenlerini (X, Y, Z) temsil etmektedir.

Şekil 3.1: Benzetimi yapılan CNC dikey işleme merkezi

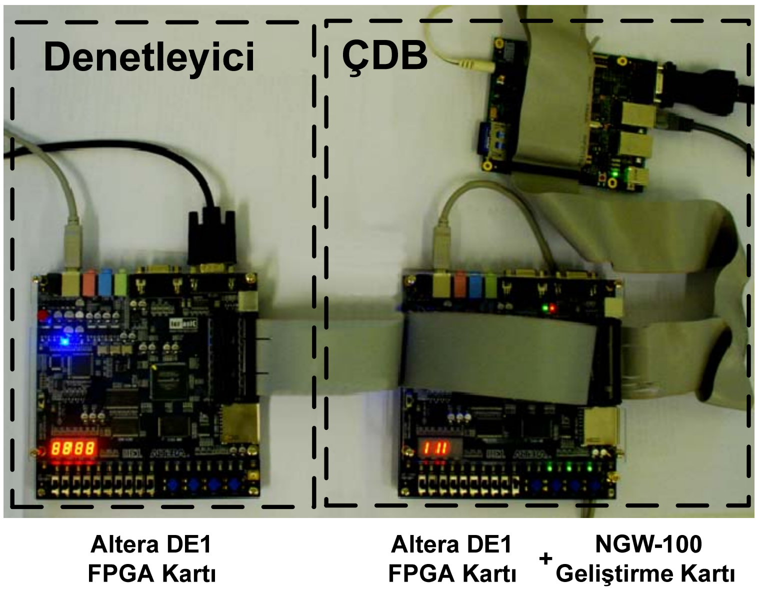


Şekil 3.2: Dikey işleme merkezinin hareket denetimi yapılan eksenleri

Öncelikle bu tezgah sistemini detaylı bir dinamik modeli (yataklar, bilyalı sonsuz vida içeren hareket aktarım sistemi, tork-denetim kipinde sürülen motorlar, vs.) geliştirilerek, Bölüm 2.5'te ortaya yöntemler yardımıyla, kontrol edilecek makina sisteminin ÇDB modeli oluşturulmuştur [2]. Ardından,

- makinanın üzerinde yer alan DC servo-motorların tork-denetim kipinde sürüldüğü,
- hareket denetleyicisinin ürettiği motor tork komutlarının sayısal olarak (darbe genişlik kiplenimi ile) motor sürücüsüne aktarıldığı,
- motora doğrudan doğruya bağlanan optik konum enkoderlerine erişim olduğu,

varsayılarak, tezgahın sanal (elektronik) arayüzü ortaya konmuştur. Şekil 3.3 çalışmanın bu bölümünde kullanılan düzeneği göstermektedir.

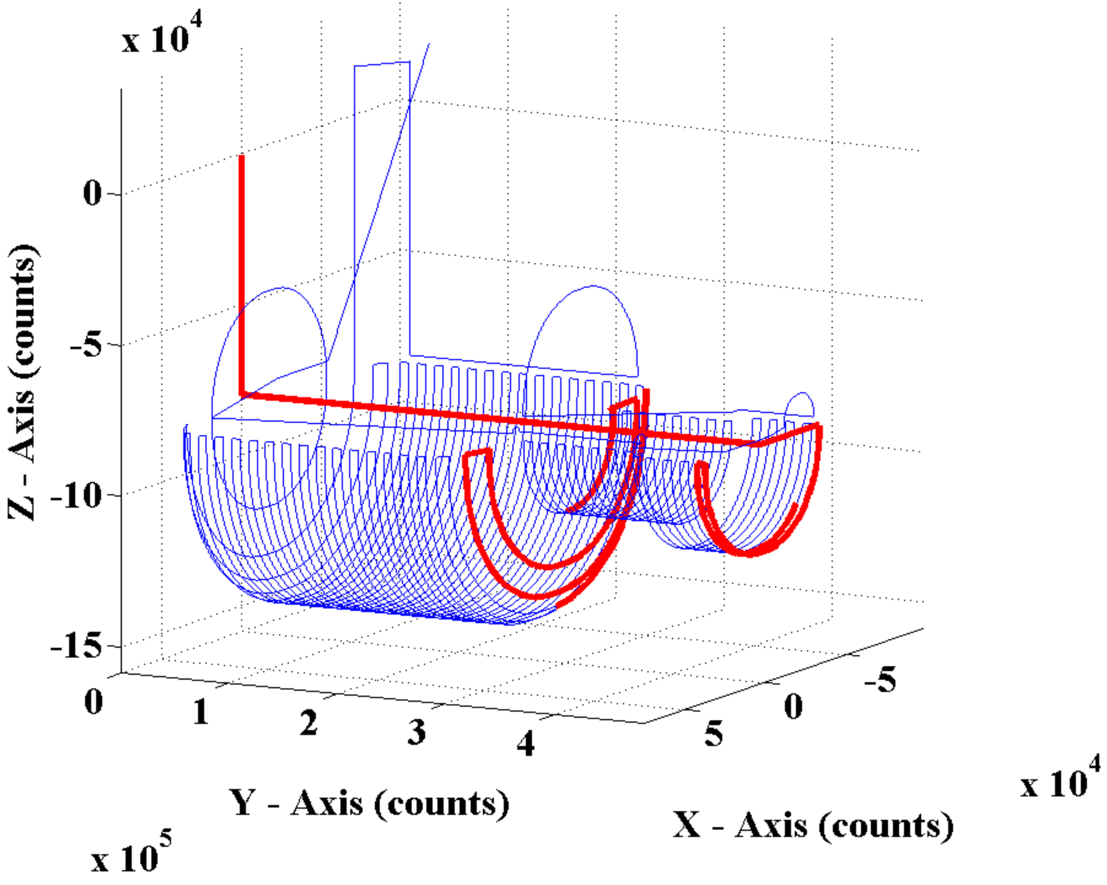


Şekil 3.3: Çevrimiçi donanım benzetimi yapılan test düzeneği

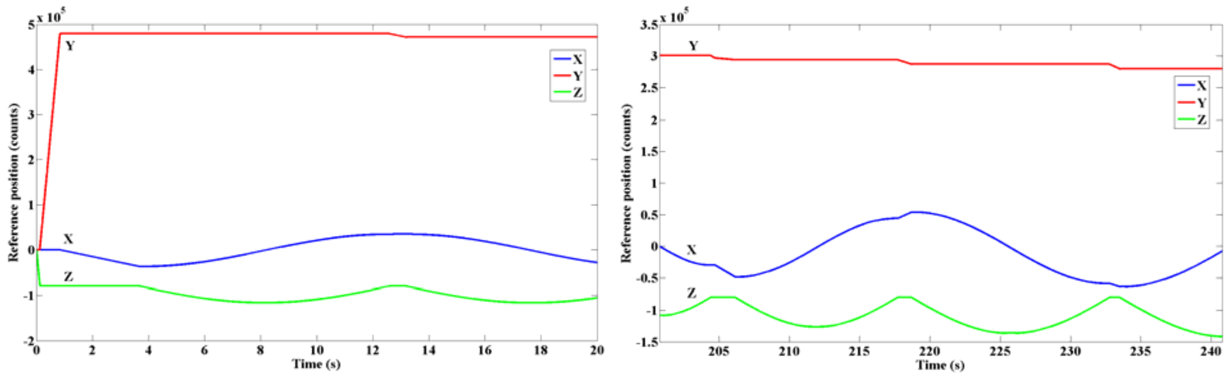
Dikkat edilirse, bu sistemde (Cyclone II FPGA içeren) iki adet Altera DE1 geliştirme kartı ve bir tane de (AVR32-AP7000 sayısal işaret işlemci barındıran) NGW-100 kartı kullanılmıştır. Burada DE1 geliştirme kartlarından biri DMOX olarak üç eksenli hareket denetimini gerçekleştirirken, diğer DE1 kartı makinanın aygıt arayüzüne öykünmektedir. Benzer olarak, (verilen doğrusal olmayan differensiyel denklem takımlarını çözen) NGW-100 takım tezgahının eksen hareketlerini hesaplayarak aygıt arayüzüne aktarmanın yanı sıra; elde edilen sonuçları Ethernet (TCP/IP) üzerinden ana bilgisayara göndermektedir. NGW-100 üzerinde (gerçek zamanlı olmayan) Linux işletim sistemi kurulduğundan; sayısal işaret işlemcinin kaynaklarına ancak Linux işletim sistemi çekirdeği üzerinden erişmek mümkün olmaktadır. Bu da doğal olarak, kesme isteklerinin değerlendirilmesinde ve sayısal port/bellek erişimlerinde önemli gecikmelere yol açmaktadır. Dolayısıyla, söz konusu şartlar altında benzetimin gerçek-olmayan zamanda icra edilmesi mecburiyeti ortaya çıkmaktadır. Bu durumda, denetleyici kart DMOX zamanlama kaynağı olarak görev yapmakta olup, daha düşük örnekleme frekanslarında çalışmaktadır. Örneğin, denetim sisteminin örnekleme frekansı 1 kHz olsa da, üretilen komutlar 100 Hz'lik bir frekans ile simulatöre gönderilmektedir. Böylece, simülatör, denetleyici (tork) komutlarının alımının ardından, (10 ms gibi geniş bir zaman dilimi içinde) makinanın sonraki örnekleme periyodundakini yanıtını hesaplayarak; aygıt arayüzüne rahatça göndermektedir.

Söz konusu takım tezgahı için geliştirilen kontrol sisteminden söz etmek gerekirse, herbir eksen için öncül-artçıl düzenleyici ("lead-lag compensator") tasarlanmış, tezgah modelinin anma değerleri kullanılarak düzenleyicinin katsayıları ayarlanmıştır. Tasarım işlemi ve FPGA üzerinde bu düzenleyicinin gerçeklenmesi [1]'de detaylandırılmıştır. Kısaca özetlemek gerekirse; aygıt arayüzleri (enkoder çözümleyicisi, darbe genişlik modülatörü, SRAM denetleyici, RS-232 haberleşme birimi) ve üç adet tamsayı işlem yapan öncül-ardıl düzenleyici modül içeren DMOX topolojisinin Cyclone II üzerinde devre sentezi yapıldığında, kırmık üzerindeki kaynakların ortalama %60'ı kullanılmaktadır. Her ne kadar komut üreteci modül için de kırmık üzerinde kaynak olsa da, kaynakların sonuna kadar kullanılması Bu da çok eksenli uygulamaların (8 ve üzeri) rahatlıkla daha gelişkin FPGA kırmıklarıyla mümkün olacağını işaret etmektedir. Dikkat edilirse, FPGA üzerinde eksen denetleyicilerinin yanısıra, aygıt arayüzü modülleri de (darbe genişlik modülatörü, enkoder çözümleyici) yer almaktadır. Diğer taraftan, FPGA kırmığında yeterli kaynak olmadığı için, DMOX mimarisinin önemli parçalarından biri olan komut üreteci modül kırmığa yerleştirilmemiştir. Bunun yerine üretilmesi gereken komutlarını tümü DE1 kartındaki SRAM (512 kB) üzerinde depolanması daha uygun görülmüştür.

Denetleyici başarımının sınanması için, bir şişe için plastik enjeksiyon kalıbının talaşlı imalat işlemi dikkate alınmıştır. Şekil 3.4 bu kalıbın işlenmesi sırasında kesici takımın izlediği yörüngeyi göstermektedir. Şekil 3.5 bu yörünge üzerinde şecilmiş özgün özellikler barındırdığı düşünülen iki parçayı temsil etmektedir. Bu iki yörünge parçası üzerinde ilerlerken elde edilen sonuçlar, Şekil 3.6'da gösterilmektedir.

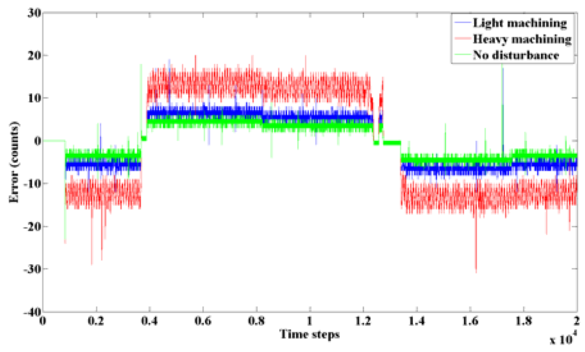


Şekil 3.4: Şise enjeksiyon kalıbı üretilirken kesici takımın izlemesi gereken yörünge

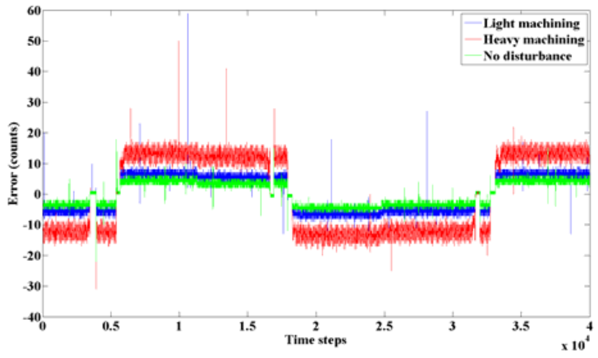


(a) Yörünge A (0 ... 20 saniye arası) **(b)** Yörünge B (201 ... 241saniye arası)

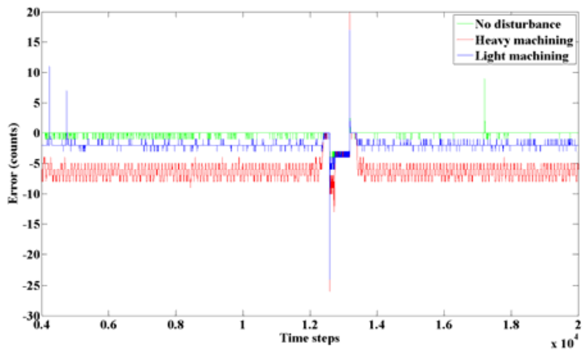
Şekil 3.5: Ana hareket eksenlerinde izlenmesi gereken yörünge parçaları



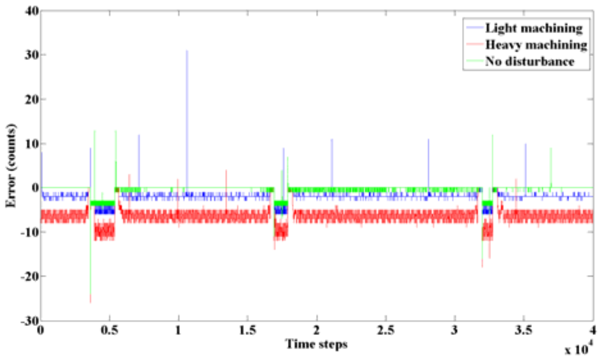
(a) X-ekseni (Yörünge A)



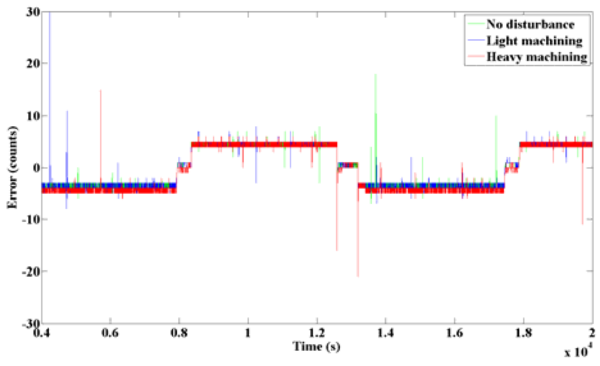
(b) X-ekseni (Yörünge B)



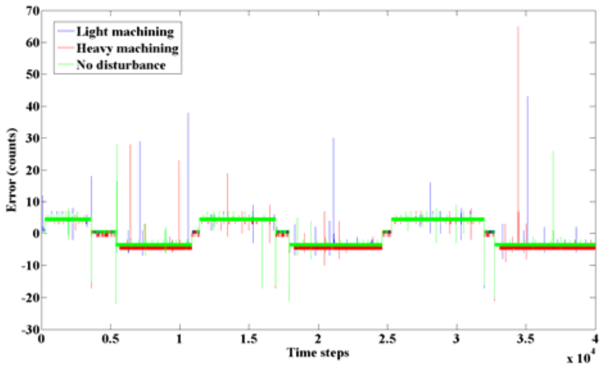
(c) Y-ekseni (Yörünge A)



(d) Y-ekseni (Yörünge B)



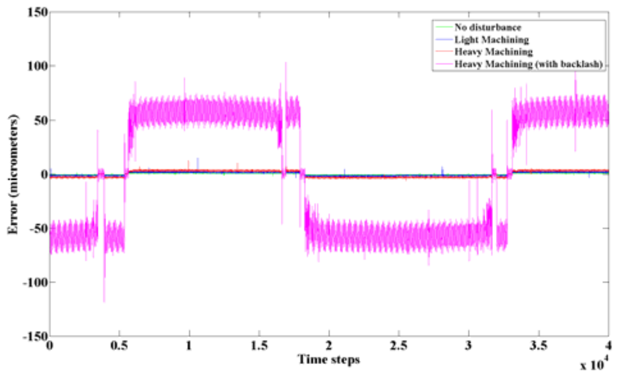
(e) Z-ekseni (Yörünge A)



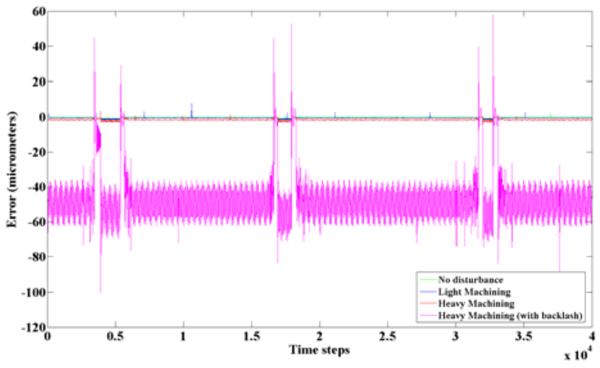
(f) Z-ekseni (Yörünge B)

Şekil 3.6: Ana eksenler üzerindeki konumlama hataları

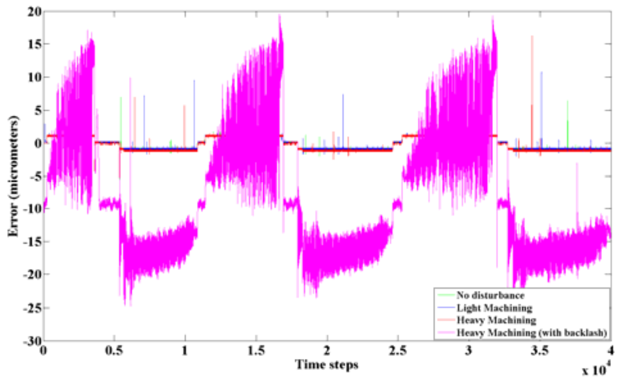
Gerçekçi bir benzetim yapmak amacıyla, uygun bir bozucu modeli de ÇDB'ye eklenmiştir. Benzetim işlemini yapan sistem freze işlemi sırasında ortaya çıkan kuvvetlerin asal eksenler üzerindeki bileşenlerini hesaplayarak, ilgili eksenlere uygulamaktadır. Şekil 3.6'da görülen "light machining," "heavy machining" etiketleri sırasıyla ince ve kalın pasolu frezelemeyi temsil etmekte olup, ince pasoda kesme kuvvetinin genliği 500 N'a, kalın pasoda ise bu genlik yaklaşık 5 kN'na ulaşmaktadır. Son olarak, Şekil 3.7 hareket aktarım sisteminde bir boşluk ("backlash") varken ortaya çıkan durumu göstermektedir. Tablo 3.1 ve 3.2 tüm sonuçların bir özetini sunmaktadır.



(a) X-ekseni



(b) Y-ekseni



(c) Z-ekseni

Şekil 3.7: Hareket mekanizmasındaki boşluk dikkate alındığında ana eksenler üzerinde ortaya çıkan konumlama hataları (Yörünge B)

Tablo 3.1: Yörünge A üzerinde (mikron cinsinden) elde edilen sapma değerleri

Eksen	**X**		**Y**		**Z**	
Bozucu	**RMS***	**STD***	**RMS**	**STD**	**RMS**	**STD**
Yok	0.95	0.95	0.22	0.21	1.02	1.02
İnce paso	1.50	1.49	0.53	0.19	1.05	1.05
Kalın paso	3.15	3.13	1.60	0.39	1.03	1.03

[*] RMS: kareleri alınmış değerlerin ortalamasının kare kökü; STD: Standart sapma.

Tablo 3.2: Yörünge B üzerinde (mikron cinsinden) elde edilen sapma değerleri

Eksen	**X**		**Y**		**Z**	
Koşul	**RMS**	**STD**	**RMS**	**STD**	**RMS**	**STD**
Bozucu yok	0.948	0.95	0.29	0.28	1.01	1.01
İnce paso	1.506	1.50	0.61	0.28	1.04	1.04
Kalın paso	3.14	3.14	1.66	0.49	1.03	1.03
Kalın paso + boşluk	55.94	55.94	49.17	13.56	12.49	9.38

Bu tabloda RMS değeri

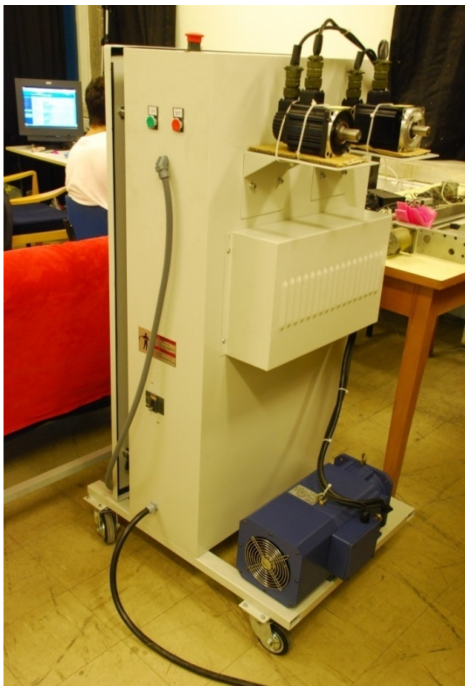
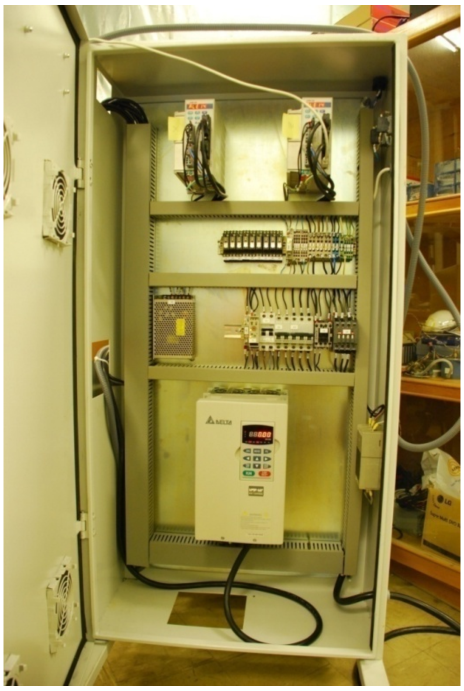
$$\text{RMS} = \sqrt{\frac{1}{K}\sum_{k=0}^{K}[x(k) - x^{*}(k)]^{2}} \qquad (3.1)$$

şeklinde tanımlanmakta olup; K toplanan verinin uzunluğunu, x (k anında) bir eksenin ölçülmüş konumunu, x* ise (o anda) arzu edilen konumu ifade etmektedir.

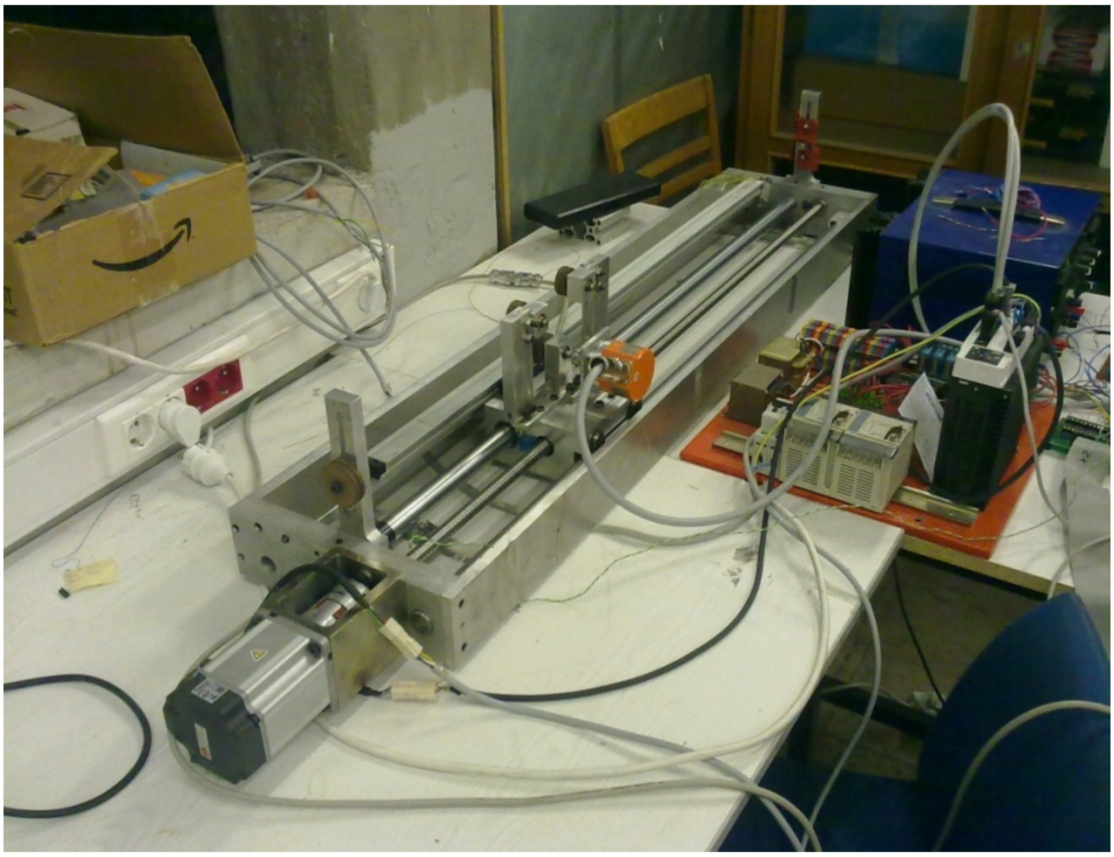
Elde edilen gerçekçi benzetim sonuçlarından da görüleceği üzere, denetleyici sistemi oldukça iyi bir komut takip ve bozucuya karşı mukavemet performansı göstermiştir. 10 mikronun altında bir konum denetim hatası amaçlandığı düşünülürse, sistemin en ağır işletim koşullarında bile tatmin edici bir performans sergileyeceği anlaşılmaktadır. Elde edilen sonuçlar geliştirilen denetleyicinin rahatlıkla ticari denetleyici sistemlerle başa baş bir performans göstereceğini teyit etmektedir. Dikkat edilirse, takım tezgahı eksenlerinde boşluk olduğunda, sistemin konum kontrolu yeterli olmadığı görülmektedir. Zira, ele alınan sistemde sadece motorun açısal konumu ölçüldüğü için, denetim sistemi boşluk nedeniyle kızakta ortaya çıkan salımınları algılayamamakta ve gerekli düzeltmeyi doğal olarak yapamamaktadır.

3.2 Konum Kontrol Deney Düzeneği

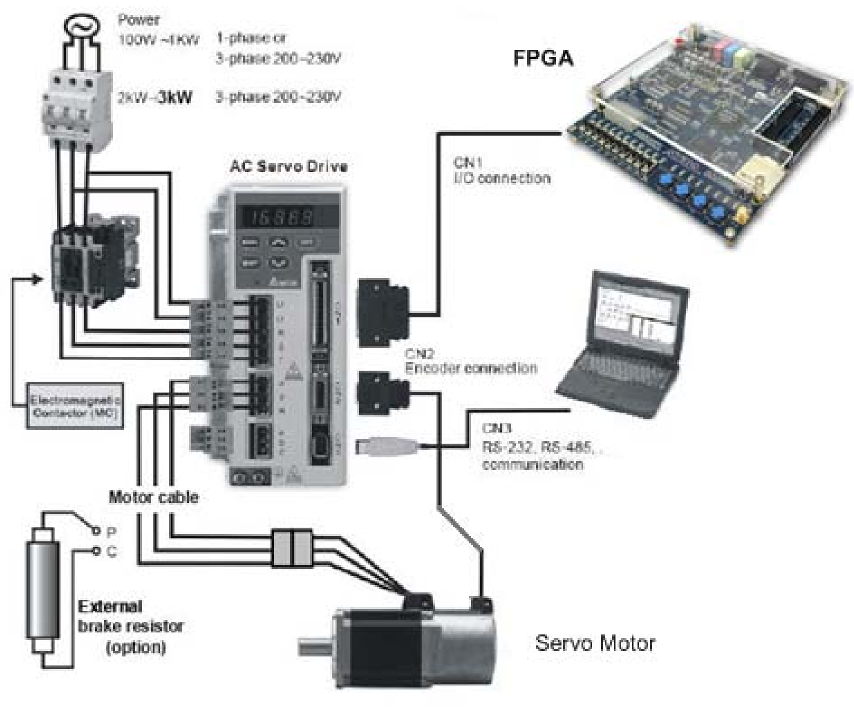
Proje kapsamında geliştirilmekte olan evrensel denetleyici prototipinin gerçekçi bir biçimde sınanabilmesi için, OSTİM'de faaliyet gösteren bir firmadan üzerinde iki servo-motor ve bir AC asenkron motor bulunan ve Şekil 3.8'de gösterilen bir test düzeneğinin ödünç alındığı bir önceki raporda belirtilmişti. Bu düzenek kullanılarak, FPGA tabanlı açık-çevrim denetim çalışması başarıyla gerçekleştirilmiştir.



Şekil 3.8: Geliştirilen denetleyici prototiplerin ön sınamalarının yapıldığı düzenek.

Firmaya test düzeneğinin iade edilmesiyle, Laboratuarımızda mevcut olan tek eksenli doğrusal kızak sistemine (Şekil 3.9) geri dönülmüştür. Bu düzenekte taşıyıcı araba, iki kılavuz mil üzerinde ilerlemekte olup; araba, millerden biri ne doğrusal rulmanlarla diğerine ise hareketi sadece dikey yönde kısıtlayacak makaralı rulmanlarla yataklanmıştır. Bu deney düzeneğinde hem arabanın hem de motor milinin konumu iki ayrı duyucu (doğrusal cetvel / dönel enkoder) aracılığıyla ölçülebilmektedir. Bu kızak sisteminde yer alan fırçalı DC motor değiştirilerek, yerine 750 W Panasonic AC servo motor monte edilmiştir. Sistemin genel bağlantı şeması Şekil 3.10'da gösterilmektedir.



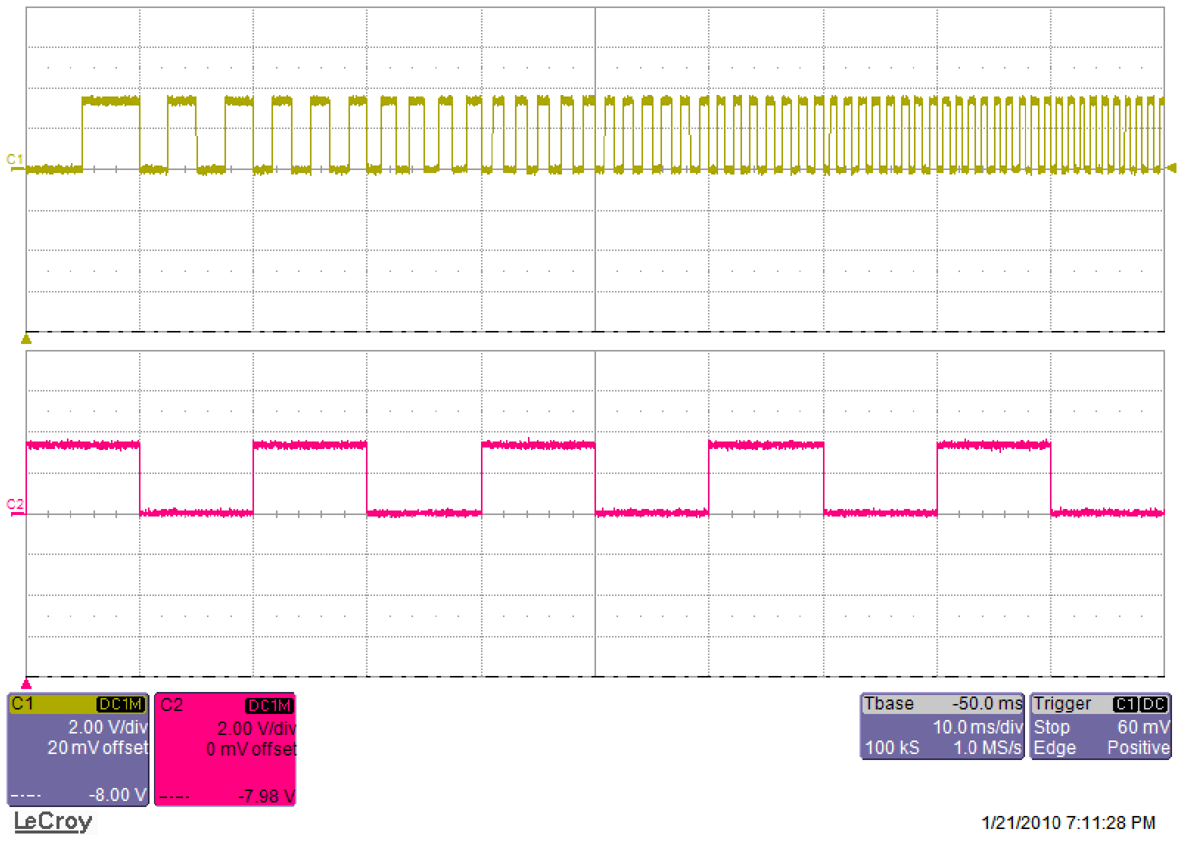
Şekil 3.9: Geliştirilen denetleyici prototiplerinin sınandığı düzenek



Şekil 3.10: AC servo-motor düzeneğinin şeması

Projede geliştirilen hareket denetleyiciler ilk etapta bu deney düzeneği üzerinde (bozucu olmadan/tek eksende) sınanmıştır. Dikkat edilirse, modern servo-motor sürücüleri (bünyelerinde yer alan gelişkin mikrodenetleyiciler sayesinde) çok değişik denetleme (tork, hız, konum) kipinde çalışma imkanı sağlamaktadır. Her ne kadar hassas hareket denetim uygulamalarında motor sürücüsü ayarlanabilir tork kipinde kullanılsa da, günümüzün endüstriyel uygulamalarında konum kipi özellikle tercih edilmeye başlanmıştır. Bu kipte (konum geri-beslemesine sahip) motor sürücüsü konum ve hız kontrol çevrimi gerçekleştirmekte olup, ana denetleme sistemi ise artımlı konum komutlarını sürücüye (tıpkı bir adım motoru sürer gibi) göndermektedir. Yani, ana denetleyici ünitesinin motor sürücüye konumlama komutlarını bir darbe dizini (PULSE) ve bir yön işareti (SIGN) şeklinde göndermesi yeterlidir.

Bunun bir uygulaması önerilen DMOX sistemiyle yapılmıştır. Bu uygulamada ana denetim ünitesindeki FPGA kırmığına (Cyclone II) düşen görev her örnekleme periyodu içerisinde değişik frekanslarda (0 – 200,000 darbe/saniye) ve istenilen sayıda darbeyi üretebilmesidir. Bunun için de motorun her örnekleme periyodu için izleyeceği yörünge (konum) ve yönünün FPGA kırmığında hesaplanması (veya dışarıda hesaplanıp FPGA'ye aktarılması) gerekmektedir. Bu çalışmada ikinci seçenek tercih edilmiş olup; geliştirilen modülde ilk başta PC eşit zaman aralıkları için motor konumlarını (darbe sayısı cinsinden) hesaplayıp, RS-232 üzerinden FPGA geliştirme kartı üzerinde bulunan SRAM'e aktarmaktadır. Komutlar alındıktan sonra kullanıcının geliştirme kartı üzerindeki önceden tanımlanmış bir anahtarı kapatması ile birlikte SRAM'deki komutlar sırası ile icra edilmesi sağlanmaktadır. Burada önemli olan hususlardan biri, her örnekleme periyodu için komutların PC tarafında hassas biçimde hesaplanıp SRAM'e yazdırılmasıdır. Bu komutların gerçek zamanda ("real-time") icra edilmesinden ise FPGA kırmığı sorumlu olmaktadır. Dikkat edilirse, FPGA kırmığının konum darbelerini örnekleme periyoduna yayması gerekmektedir zira bu kontrol stratejisinde hız denetimi dolaylı olarak yapılmaktadır. Bir başka deyişle, konum darbelerin geliş sıklığı motorun aynı zamanda ortalama hızını belirlemektedir. Darbelerin periyoda yayılı şekilde çıkılmasını sağlayan modülün (`pulse_producer`) Verilog HDL kodu Liste 1'de sunulmuştur. Şekil 3.11'den de görüldüğü üzere, FPGA SRAM'deki komutlara uygun olarak söz konusu denetleme işaretlerini üretmiş, konum kipindeki AC servomotorun istenen konum yörüngesini başarıyla takip etmesi sağlanmıştır.

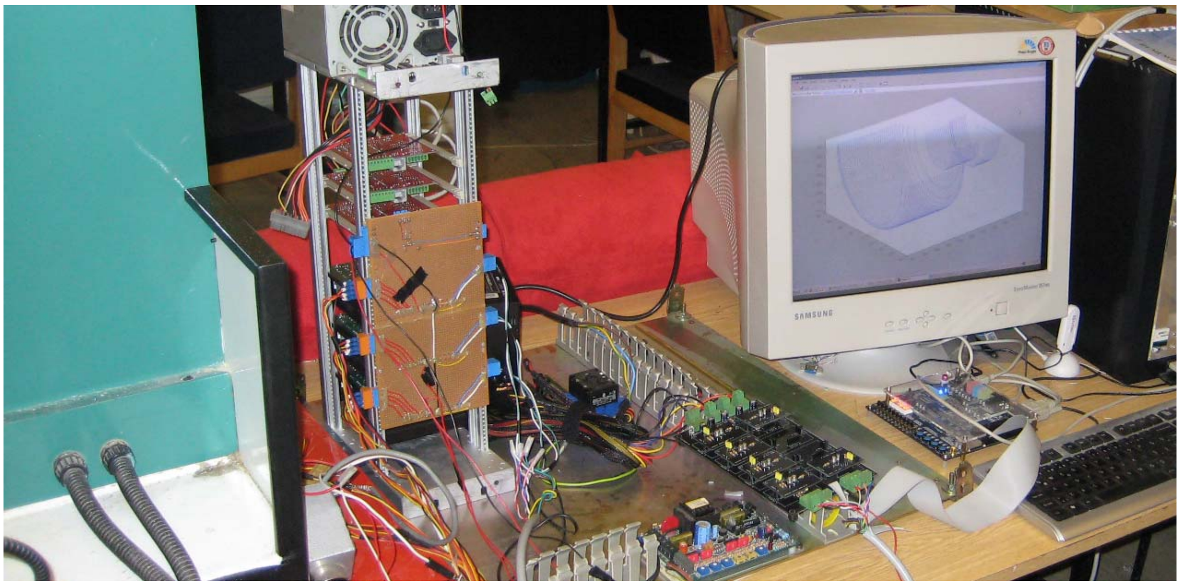


Şekil 3.11: Darbe ve yön sinyallerinin periyoda yayılı şekilde üretimi

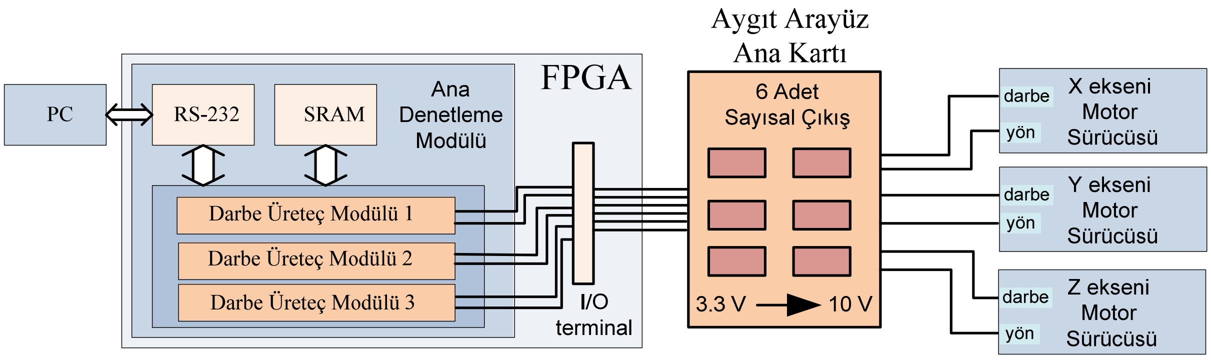
3.3 CNC Masaüstü Freze Tezgahı

Bu çalışma kapsamında geliştirilen denetleyici sisteminin nihai testleri ise Şekil 3.12'de gösterilen üç eksenli bir eğitim tezgahı (DENFORD STARMILL-ATC) üzerinde gerçekleştirilmiştir. Tezgahın halihazırdaki merkezi denetleyicisi çıkartılıp, Şekil 3.13'de görüldüğü üzere, bu proje kapsamında geliştirilen DMOX denetim sistemi (Altera DE1 FPGA geliştirme kartı ve aygıt giriş/çıkış arayüz kartları) *açık çevrim kipinde* devreye alınmıştır. Tezgahın üzerinde yer alan adım motorları ve sürücüler (Parker-Hannifin SD2) olduğu gibi kullanılmış olup, her eksendeki hareket çözünürlüğü 10 mikrondur. Dikkat edilirse, herbir motor sürücüsü kendisine bağlanan adım motorunu sürmek için, CMOS 10V mantık seviyesinde (CMOS ACT ile uyumlu) darbe ve yön (giriş) işaretlerine ihtiyaç duymaktadır. Bu yüzden toplamda 6 adet *sayısal çıkış* kartı (bkz. Bölüm 2.1), geliştirilen ana kart üzerine eklenerek, FPGA tarafından üretilen CMOS 3.3V mantık seviyesindeki denetim işaretleri CMOS 10V mantık seviyesine yükseltilmektedir. Bu sistemde, hareket komutları RS-232 seri iletişim protokolü ile FPGA geliştirme kartı üzerindeki SRAM'e (512 KB statik bellek) aktarılmaktadır. Toplamda üç ana eksen eşgüdümlü olarak denetleneceğinden, bir darbe üreteci modül Verilog HDL kullanılarak geliştirilmiş, bunlardan üç tanesinin devre sentezi FPGA üzerinde yapılmıştır. Ekler kısmındaki Liste 1 bu modül için geliştirilen Verilog HDL kodunu göstermektedir. Son olarak, SRAM'den hareket komutlarını okuyarak, darbe üreteçlerine artımlı konum komutları gönderen bir ana modül ortaya konmuş, Liste 2'de bu modülün Verilog HDL kodu verilmiştir. Şekil 3.14 tüm düzeneği ana hatlarıyla göstermektedir.

Şekil 3.12: DENFORD STARMILL-ATC Eğitim Frezesi

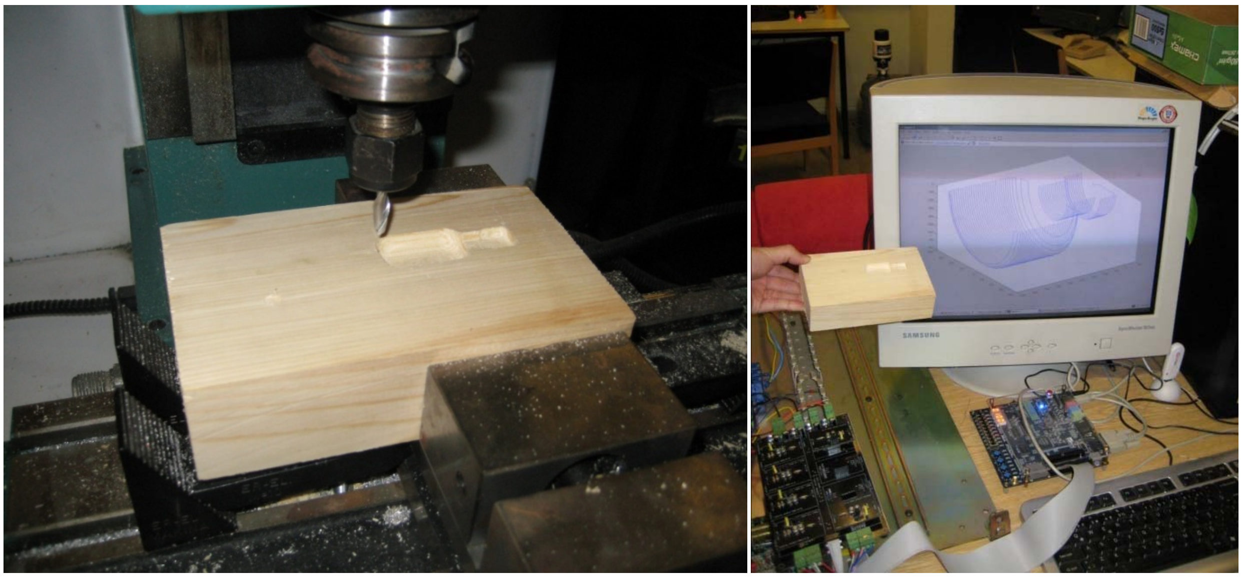


Şekil 3.13: Tüm sistemin görünüşü



Şekil 3.14: Üç eksen denetleme düzeneğinin blok diyagramı

Dikkat edilirse, bu eğitim tezgahının işmili (“spindle”) gücü 250W olup; ancak plastik, ahşap, aluminyum gibi sert olmayan malzemeler (küçük pasoyla ve düşük hızlarda) işlenebilmektedir. Sistemin testi için, Bölüm 3.1’de bahsi geçen plastik enjeksiyon şise kalıbı göz önüne alınmış; kalıbın ahşap bir modeli bu sistem vasıtasıyla başarılı bir şekilde üretilmiştir. Şekil 3.15 elde edilen bu sonuçları göstermekte olup, nihai testler sırasında çekilen video görüntüleri de bu raporun ekindeki CD’de yer almaktadır.



Şekil 3.15: Kalıp modelinin işlenmesi ve sonuçların karşılaştırılması

4. SONUÇLAR VE TARTIŞMA

Bu proje kapsamında her türlü endüstriyel sistemin denetimine imkan sağlayan ve uygulamanın gerekleri doğrultusunda esnek bir şekilde biçimlendirilebilir bir evrensel denetim sistemi ortaya konmuştur. Denetleyici sisteminin bir prototipi çeşitli FPGA geliştirme kartları ve projeye has tasarlanmış giriş/çıkış kartları vasıtasıyla meydana getirilmiş olup, herhangi bir denetim sisteminin FPGA kırmığı üzerinde sentezlenmesine imkan tanıyacak çok sayıda düşünce iyeliği kapsamındaki modül (Verilog ve VHDL gibi donanım tanımlama dilleri kullanılarak) oluşturulmuştur. Proje kapsamında tüm bir denetleyici topolojisinin sentezi (bu modüllerden faydalanılarak) FPGA geliştirme ortamında (Quartus II, ISE Design Studio) manuel olarak yapılsa da; ürünleşme aşamasına gelindiğinde söz konusu modüllerin bir kullanıcı dostu grafik arayüzü aracılığıyla bir araya getirerek, arzu edilen bir denetim mimarisinin devre sentezini otomatik olarak yapan bir yazılımın geliştirilmesi olasıdır. Proje kapsamında yapılan deneyler ve ÇDB sonucunda sistemin piyasada bulunan ve ticari olarak pazarlanan hemen hemen her tür denetim sistemi (hareket kontrol kartları, PC104 veya PLC tabanlı sistemler) ile başa baş bir denetleme başarımı sergileyeceği anlaşılmıştır.

Sonuç olarak, projede FPGA tabanlı evrensel bir denetleyici sisteminin teknik olabilirliği gösterilmiş, bu sistemin denetim özellikleri ve başarımı net bir şekilde ortaya konmuştur. Ancak, proje çıktısının ürünleşebilmesi için, yapılması gereken daha çok iş olduğu açıktır. Bunları kısaca özetlemek gerekirse, **i)** üzerinde gelişkin bir FPGA kırmığı, bilgisayar çevre birimleri, programlanabilir araç arayüzünü barındıran çok katlı devre basımı/üretimi/testi; **ii)** Denetleyicinin muhafazası ve paketleme çalışmaları; **iii)**

Denetleyici sentezinin yapılmasına imkan tanıyan kullanıcı dostu yazılım araçlarının geliştirilmesi; **iv)** Sistemin saha ve kullanıcı testleri; **v)** Dokümantasyon ve belgeleme çalışmalarıdır. Bunlar da doğal olarak, geniş bir mühendis grubu tarafından nispeten uzun bir süre zarfında yürütülmesi gereken (ve büyük finansal kaynağa ihtiyaç duyan) etkinliklerdir ve doğal olarak projemizin kapsamını dışında tutulmuştur.

Bu proje sonunda elde edilen (akademik ve endüstriyel) kazanımları şu şekilde özetlemek mümkündür:

- (Paketleme ve saha test çalışmaları yapıldığı taktirde) ürünleşebilecek bir evrensel denetleyici sistemi: DMOX-A,
- FPGA tabanlı (Xilinx Virtex 5, Altera Cyclone II) gömülü sistem tasarımı konusunda elde edilen kapsamlı bilgi birikimi ve deneyim,
- Evrensel denetleme sisteminde kullanılmak üzere geliştirilmiş kırkın üzerinde özgün düşünce iyeliği modülü,
- Yeni bir veri sıkıştırma algoritması (ΔY),
- Eğitim uygulamalarına hitap eden CADMVS isimli yeni bir ÇDB yazılımı,
- 3 adet yüksek lisans tezi [1-3],
- 5 adet uluslararası konferans makalesi [4-8]
    - [4] numaralı çalışmamız WISES 2010'da en iyi makale ödülünü kazanmıştır.
- 4 adet ulusal konferans makalesi [9-12],
- 4 adet (gönderim aşamasında olan) uluslarası dergi makalesi [13-16],
- 2 adet buluş ("invention"):
    - *Alan Programlanabilir Kapı Dizini Kırmıkları için Programlanabilir bir Cihaz Arayüzü* isimli çalışma için TPE'ye başvuru yapılması planlanmaktadır.
    - *Hareket Kontrol Uygulamaları için Komut Üreteci* isimli çalışma için de başvuru öncesi patent araştırmaları devam etmektedir.

Bu proje kapsamında alımı yapılan ekipmanlar sayesinde, FPGA tabanlı ileri elektronik sistemler geliştirilebilecek bir araştırma laboratuarının ("Endüstriyel Elektronik") altyapısı da (nüvesi) ODTÜ Makina Mühendisliği Bölümünde oluşturulmuştur. Proje yürütücüsü ve araştırmacıları, bu projeye finansal destek sağlayarak büyük katkılarda bulunan TÜBİTAK-EEEAG'ye içtenlikle teşekkür eder.

REFERANSLAR

1. Mutlu, B. R., *Real-time Motion Control using Field Programmable Gate Arrays*, Yüksek Lisans Tezi, ODTÜ Fen Bilimleri Enstitüsü, Haziran 2010.

2. Üşenmez S., *Design of an Integrated Hardware-in-the-Loop Simulation System*, Yüksek Lisans Tezi, ODTÜ Fen Bilimleri Enstitüsü, Haziran 2010.

3. Yaman, U., *Design of Advanced Motion-Command Generators utilizing Field Programmable Gate Arrays*, Yüksek Lisans Tezi, ODTÜ Fen Bilimleri Enstitüsü, Haziran 2010.

4. Yaman, U., Dölen, M., and Koku, A. B., "A Novel Command Generation Method with Variable Feedrate utilizing FGPA for Motor Drives," *Proc. of the 8th IEEE Workshop on Intelligent Solutions in Embedded Systems (WISES)*, Heraklion, Crete (Yunanistan), 8-9 Temmuz 2010.

5. Mutlu, B. R., Dölen, M., "Novel Implementation of State-Space Controllers using Field Programmable Gate Arrays," *Proc. of the Int'l Symposium on Power Electronics, Electrical Drives, Automation, and Motion (SPEEDAM)*, Pisa (İtalya), 14-16 Haziran 2010.

6. Mutlu, B. R., Yaman, U., Dölen, M., and Koku, A. B., "Performance Evaluation of Different Real-Time Motion Controller Topologies Implemented on a FPGA," *Proc. of the 12th International Conference on Electrical Machines and Systems (ICEMS)*, Tokyo (Japonya), 15-18, Kasım 2009.

7. Üşenmez, S., Dilan, R. A., Dölen, M., Koku, A. B., "Real-time Hardware in the Loop Simulation of Electrical Machine Systems using FPGAs," *Proc. of the 12th International Conference on Electrical Machines and Systems (ICEMS)*, Tokyo (Japonya), 15-18 Kasım 2009.

8. Yaman, U., Mutlu, B. R., Dölen, M., Koku, A. B., "Direct Command Generation Methods for Servo-Motor Drives," *Proc. of the 12th International Conference on Electrical Machines and Systems (ICEMS)*, Tokyo (Japonya), Nov. 15-18, 2009.

9. Mutlu, B. R., Yaman, U., Dölen, M., Koku, A. B., "Kayan Kipli DC Motor Konum Denetiminin FPGA ile Gerçekleştirilmesi," *Türkiye Otomatik Kontrol Ulusal Toplantısı 2009 (TOK 2009) Bildiri Kitabı*, Istanbul,13-16 Ekim 2009.

10. Üşenmez, S., Dilan, R. A., Dölen, M., Koku, A. B., "Bir Doğru Akım Motorunun FPGA Üzerinde Gerçek Zamanlı Benzetiminin Gerçekleştirilmesi," *Türkiye Otomatik Kontrol Ulusal Toplantısı 2009 (TOK 2009) Bildiri Kitabı*, Istanbul,13-16 Ekim 2009.

11. Yaman, U., Dolen, M., Koku, A. B., "Endüstriyel Kontrol Uygulamaları için Komut Üretim Yöntemleri," *Türkiye Otomatik Kontrol Ulusal Toplantısı 2009 (TOK 2009) Bildiri Kitabı*, Istanbul,13-16 Ekim 2009.

12. Üşenmez S., Dilan R. A., Yaman, U., Mutlu, B. R., Dölen, M., Koku B. A., "Çevrimiçi Donanım Benzetimi için Yeni bir Yazılım Paketi: Cadmus," *Türkiye Otomatik Kontrol Ulusal Toplantısı 2008 (TOK 2008)*, cilt 2, sf. 685-691, 2008.

13. Üşenmez, S., Koku, A. B., Dölen, M., "A New Hardware-in-the-Loop Simulator for Control Engineering Education," *IEEE Transactions on Education* isimli dergiye gönderilecektir.

14. Yaman, U., Dölen, M., Koku, A. B., "FPGA Implementation of a Novel Motion-Command Generator for Motor Drive Systems," *IEEE Transactions on Industrial Electronics* isimli dergiye gönderilecektir.

15. Mutlu, B. R., Dölen, M. "Evaluation of Digital Motion Controllers Implemented on Field Programmable Gate Arrays," *Control Engineering Practice* isimli dergiye gönderilecektir.

16. Üşenmez, S., Mutlu, B. R., Dölen, M., Koku, A. B., "Hardware-in-the-Loop Simulation of a Three-axis Vertical Machining Center utilizing Field Programmable Gate Arrays," *Control & Automation* isimli dergiye gönderilecektir.

Performance Evaluation of Different Real-Time Motion Controller Topologies Implemented on a FPGA

B. R. Mutlu[1,2], U. Yaman[1], M. Dolen[1], A. B. Koku[1]

[1] Department of Mechanical Engineering, Middle East Technical University, Ankara, Turkey
[2] Department of Mechanical Engineering, Hacettepe University, Ankara, Turkey

***Abstract* — This paper presents a comprehensive comparison of several real-time motion controller topologies implemented on a field programmable gate array (FPGA). Controller topologies are selected as proportional-integral-derivative controller with command feedforward, sliding mode controller, fuzzy controller, and a hysteresis controller. Controllers and other necessary modules are developed using Verilog HDL and they are implemented on a ML505 development board with a Xilinx Virtex-5 FPGA chip. In order to take full advantage of FPGA and to provide a more accurate comparison, an (soft-core) embedded processor is not employed in the design. The developed modules, which include PWM generator, quadrature encoder decoder, velocity estimator, reference profile generator etc, are fully tailored for the application. To perform the necessary calculations for certain controller topologies, an open-core floating point unit (FPU) is also adopted to the design. The performances of the aforementioned controllers are rigorously evaluated via a hardware-in-the-loop simulation of a field-oriented induction motor system.**

***Index Terms* — Controller Design, FPGA, Hardware-in-the-loop Simulation, Motion Control**

I. Introduction

Motion control plays a key role in most industrial applications including CNC machine tools, factory automation, robotics, semi-conductors manufacturing, etc. In such applications, motion controllers employ single or multiple processors to perform complex tasks required by the control algorithm with accuracy and speed.

In some cases, a low-end FPGA might be incorporated to the motion controllers to create custom logic circuits that interface with various peripheral devices [1]. On the other hand, FPGAs have become more resourceful with recent advances in VLSI technology. Fig. 1 shows the increase in resources of FPGAs released by two leading FPGA manufacturers (Altera and Xilinx) over years. Note that in the FPGAs designed by Altera and Xilinx, the basic building block is a 4-input look-up table (LUT), a flip-flop, and some additional circuitry called logic element (LE) or logic cell (LC). The logic cell to logic element ratio is 1.125:1, despite generally similar functionality [2]. In fact, FPGAs exhibit some superior qualities over microcontrollers and DSPs such as parallel processing capability, high sampling rates, flexibility and reliability. Therefore, in recent studies, FPGAs find their way as the primary control processor.

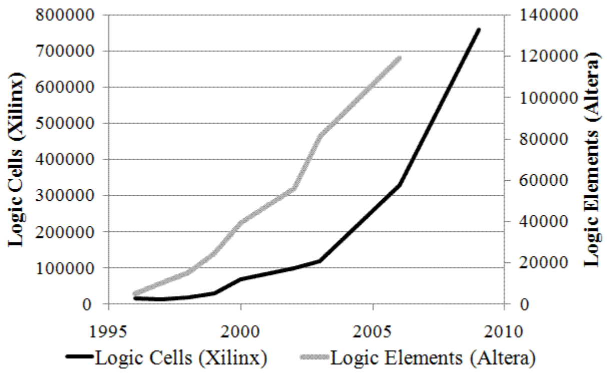


Fig.1. Increase in number of logic resources of Xilinx and Altera FPGA chips over years

Besides the favorable characteristics of FPGAs, the design with FPGA involves the use of low-level hardware description languages (HDL) which may in turn increase product design periods. To overcome this difficulty, FPGA manufacturers offer many intellectual properties (IPs) including soft-core embedded processor IPs. However, as algorithmic state machines, such processors consume most resources of FPGAs while reducing the flexibility of the controller. Consequently, in order to explore the capabilities of an FPGA to a full extend, all the IPs included in the design should be customized to perform a certain task.

In this study, motion controller topologies commonly encountered in literature are implemented on a Xilinx Virtex-5 FPGA without the use of a soft-core processor and the necessary modules for a motion control task are developed with the utilization of Verilog HDL. The organization of this paper is as follows: In section 2, background work is presented on the subject from architectural and controller topology points of view. In section 3, the implementation details of the controller topologies are presented. In section 4, test setup of the system is presented and hardware-in-the-loop simulation setup is introduced. In section 5, results are presented where performances of the controllers are evaluated with criteria such as resource cost, attainable sampling rate and

This work was supported by the Scientific and Technological Research Council of Turkey under the project contract 108E048.

the success of the controller.

II. Background Work

A. Control system architectures employing an FPGA

In many studies conducted in the last decade, FPGAs are generally considered as an interface between a DSP and its peripheral units. In this configuration, FPGAs do not usually share the computational burden of the main processor. For instance, in the design proposed by Dong *et al.* [3] FPGA serves as an external circuitry processing position encoder signals while managing some I/O functions where the computations are handled by a DSP. A similar scheme appears in the study of Birou and Imecs [4] where the FPGA performs pulse width modulation and encoder interfacing. In the design proposed by S. Jung and S.S. Kim [5], a neural network controller is implemented on a DSP and FPGA is only used to perform rather simple calculations.

However, latter research studies tend to shift the computational load from DSP to FPGA. A single FPGA is generally employed to implement the entire motion control system. Many designs facilitates a single FPGA that makes good use of an embedded processor IP developed by FPGA manufacturers. While some designs use them only to implement controllers, the others utilize the embedded processor for complex calculations along with the "hardwired" custom modules for simpler tasks. For instance, Ni *et al* [6] design a PID controller on an Altera FPGA with a Nios II processor to realize a highly integrated joint servo system. In another study, Li *et al* [7] develops an FPGA-based servo controller for PMSM drives while a sliding mode controller is implemented on a Nios II processor. Jung *et al* [8] adopt a hybrid fuzzy-PI controller to realize a motion control IC and they also used a Nios II embedded processor for the realization of the fuzzy controller. Using an embedded processor IP within the FPGA is proven to be successful in some aspects, but it reduces the flexibility of the controller and full performance of a configurable controller cannot be attained.

While the current tendency is to use an embedded processor for complex calculations required by controllers, it is also possible to eliminate the embedded processor and develop a total hardware solution. In a very recent study, Cho *et al* [9] have proposed an FPGA-based multiple axis motion control chip with no embedded processor employed. The chip has all the essential features such as velocity profile generation, interpolation, inverse kinematics calculation and a PID controller which are required to control a multiple axis motion control system such as a robotic manipulator. They managed to avoid floating point calculations by multiplying certain values by constant integers. Using no embedded processor; they attained lower resource costs and power consumption rates.

B. Control topologies utilized on FPGA controllers

In part A, different controller architectures employing an FPGA are presented. In this part, focus will be on the controllers rather than the architecture.

Controller implementation on FPGA is often a trade-off between resource and execution time of the controllers. The reason for that is; while it is possible to benefit from the parallel processing capability of the FPGA chip by calling many instances of a certain module which would certainly require more resources, it is also possible to use certain modules repeatedly in a sequential manner to increase the time, rather than the resource cost. Chan *et al* [10] have conducted a study on PID controller implementation on an FPGA and they managed to decrease the resources required by a multiplier-based design significantly. What they propose is to replace the multipliers by a distributed arithmetic based design utilizing look up tables and they managed to decrease the resource requirement down to 4 to 13% of the former design. However they increased the computation time from 1 cycle to 13 to 26 cycles. A very similar study by Tao *et al* [11] managed to decrease the logic element requirement from 51.7% to 0.8-1.5%, increasing the computation cycle from 1 to 64-33 cycles. While these studies offer good improvements for a PID controller, there exist many controller algorithms including hybrid topologies and there is no single way to implement each of them more efficiently on an FPGA.

Fuzzy controller designed by Lanping *et al* [12] requires no embedded processor or a floating point unit to perform fuzzy control. On the other hand, the proposed method is similar to a rule based control topology and the method is not generally applicable. An Elman neural network implementation is proposed by Lin *et al* [13] for a linear ultrasonic motor, where a fixed point arithmetic unit is implemented to perform the calculations. A hybrid sliding mode controller is proposed by Li *et al*, where the sliding mode controller is implemented using an embedded processor, and PI regulator with hardware logic.

These studies prove that it is possible to implement different controller topologies on FPGAs utilizing embedded processors, arithmetic logic units or custom hardware solutions. It is also seen that PID controllers are easier to implement by custom hardware logic and in practice usually realized by hardware modules. On the other hand more complex topologies generally require fixed/floating number calculations and in practice they are either realized by an embedded processor or a fixed/floating point unit.

III. Controller Topologies

In this work, a hardware solution is proposed for a number of popular real-time motion controllers. No embedded processor is implemented within the design of any controller. Furthermore, controller topologies are evaluated by certain criteria including resource cost (total memory space and logic unit requirement), attainable sampling rate and the success of the controller. For certain controllers in this study, floating point (FP) arithmetic calculations were unavoidable and consequently a floating point calculation unit (FPU) developed by Ussellman [14] is implemented in the design. However, the use of this FPU is avoided as much as possible and in certain cases where the required module is too costly in resource manner, some design methodologies are revealed for certain algorithms to balance resource-time cost of the controller.

A. *PID controller with Command Feedforward*

Proportional-integral-derivative (PID) controller is the most common type of controller used in industrial applications [15]. Furthermore, command feedforward is a possible improvement in systems where future reference values are available [16]. In many industrial systems such as CNC machines and robotics; reference trajectory is mostly pre-defined and therefore command feedforward is applicable. Hence, PID controller with command feedforward is a common choice for industrial motion controller designs. The implementation of the controller on an FPGA chip is presented in Fig. 2. In Fig. 2, controller parameters b_0, b_1, b_2 are related with the PID parameters as shown in (1-3).

$$b_0 = K_P + \frac{K_d}{T} + K_i T \quad (1)$$

$$b_1 = -K_P - \frac{2K_d}{T} \quad (2)$$

$$b_2 = \frac{K_d}{T} \quad (3)$$

Controller parameters are selected as $K_p = 50$, $K_i = 2000$, $K_d = 0.02$. Initial parameters are selected by root locus technique and the closed-loop bandwidth frequency is set around 100 Hz. Fine tuning is performed during simulations to improve the performance.

B. *Fuzzy Controller*

Fuzzy control is an intelligent control topology based on fuzzy set theory. It has been applied to many control applications including the control of drives [17]-[18]. Typical method for fuzzy control application in discrete-time control is to calculate error and change in error in each sampling time, then to define a linguistic representation of the error and change in error based on membership functions. As a final step, these linguistic representations having fuzzy memberships go through a defuzzification process to generate a manipulation signal based on a fuzzy rule base. A schematic for the implementation of the fuzzy controller on the FPGA is presented in Fig. 3. Membership functions are presented in Fig. 4. Here, NB, NS, Z, PB, and PS correspond to "*Negative big,*" "*Negative small,*" "*Zero,*" "*Positive big,*" and "*Positive small*" respectively. Notice that the functions are formed by the pulse error per sampling time period. Therefore, the data from the encoder can be directly used without further calculations while the fuzzy memberships can be represented as integers.

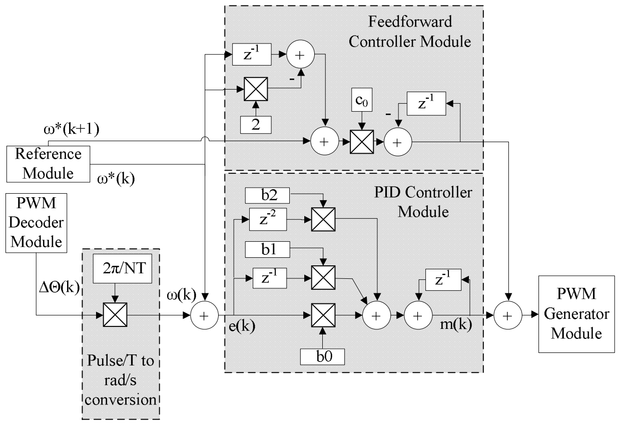


Fig. 2. PID controller implementation with command feedforward

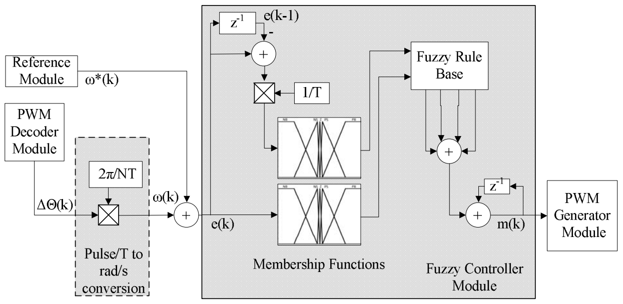


Fig. 3. Fuzzy controller implementation

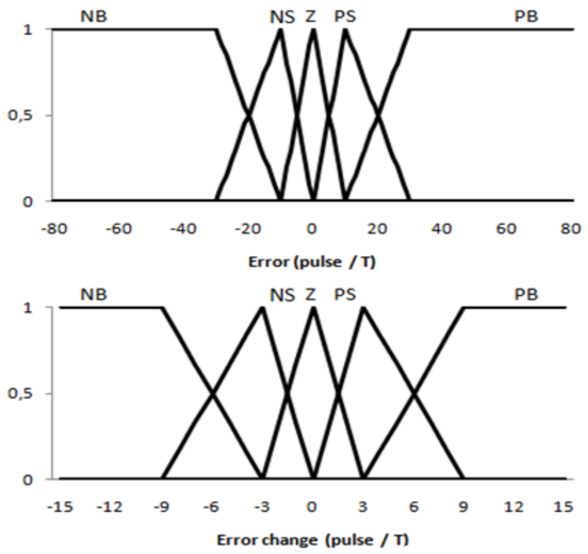


Fig. 4. Membership functions of the fuzzy controller

The defuzzification process depends on the fuzzy rule base presented on Table I. It can be observed that the membership functions are formed based on pulse error per sampling time period. Therefore, the data from the encoder can be directly used without further calculations and the fuzzy memberships can be represented in integer forms. The defuzzification process is based on the fuzzy rule base presented on Table I. Fuzzy membership functions and the rule base are selected based on the common experience in the literature and after a trial and error process.

TABLE I
FUZZY RULE BASE OF THE FUZZY CONTROLLER

error / Δ error	NB	NS	Z	PS	PB
NB	NB	NB	NS	NS	Z
NS	NB	NS	NS	Z	PS
Z	NS	NS	Z	PS	PS
PS	NS	Z	PS	PS	PB
PB	Z	PS	PS	PB	PB

C. Sliding Mode Controller

Sliding mode control is a robust control method developed to deal with model uncertainties and unknown parameters in the expense of high computational cost [19]. Its applications include position/speed control of servo/induction motor drives [20]-[21].

Sliding mode controller is based on a control law with varying control structures. The basic idea is to force the trajectory of the system state to a sliding surface through switching of the control structures. The most general form of the sliding surface is

$$s = \dot{e} + \lambda e \tag{4}$$

where s is the sliding surface; e is the error of the controlled state and λ is a controller parameter. After calculating the sliding surface, an equivalent control term needs to be calculated such that applying equivalent control would make $\dot{s} = 0$. Hence, the sliding mode control law takes the form:

$$u = u_{eq} + K sgn(s) \tag{5}$$

where u_{eq} is the equivalent control and u is control output. The implementation of the sliding mode controller is presented in Fig. 5. The sliding mode controller parameters are selected as $\lambda = 10000$ and $K = 15$. These parameters are selected based on the disturbance rejection characteristics of the controller and the maximum admissible torque based on the rated torque of the motor.

D. Hysteresis Controller

Hysteresis control (also known as bang-bang control) is the most elementary (nonlinear) control method in which the controller selects two output states (high or low) according to its input and defined hysteresis band. The controller switches back and forth between these

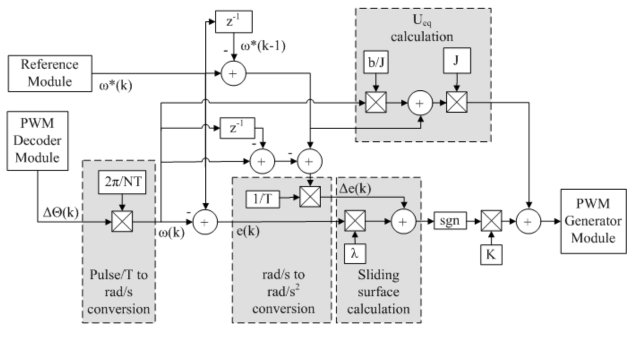


Fig. 5. Sliding mode controller implementation

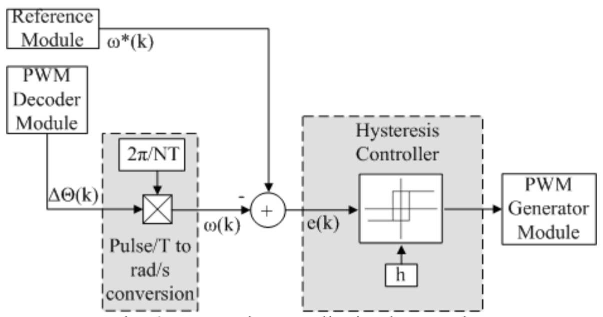


Fig. 6. Hysteresis controller implementation

states as the error exceeds the hysteresis band. Implementation of the hysteresis controller on the FPGA is shown in Fig. 6. Note that in the figure, h represents half the hysteresis band and is selected as 2 rad/s.

IV. TEST SETUP

As an illustration, the control of CNC turning centers is considered for performance evaluation of the above mentioned controller topologies. In CNC turning centers, the spindle housing the workpiece is the key component of the machine where a constant speed is required during most machining operations. Therefore, to sustain constant speed, the controller must effectively reject the disturbance torque observed in turning operations. Fig. 7 illustrates the simplified model of a typical turning center. In this system, a field-oriented induction motor, which is further elaborated in [23], is employed. Induction motor parameters are as follows: motor power: 5.5kW; rated motor torque (T_r): 35Nm; rated speed (ω_r): 1500rpm; maximum speed (ω_p): 8000 rpm. It is critical to note that the presented system is realized via a hardware-in-the-loop simulation (HILS) performed on an FPGA development board [23]. For the sake of implementation, certain simplifications (ideal DTC motor drive, ideal timing belt) are to be made on the system. Fig. 8 shows the block diagram of the resulting system.

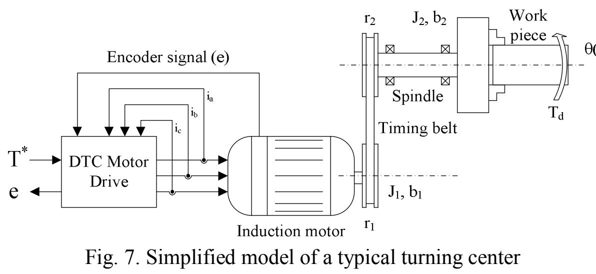


Fig. 7. Simplified model of a typical turning center

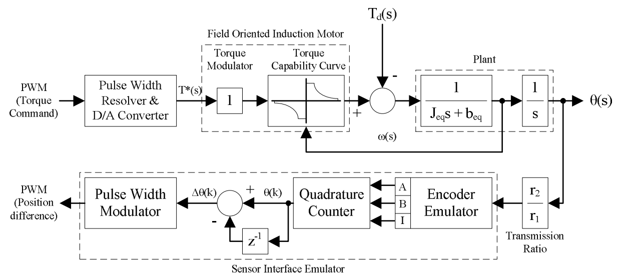


Fig. 8. Simplified model of the system

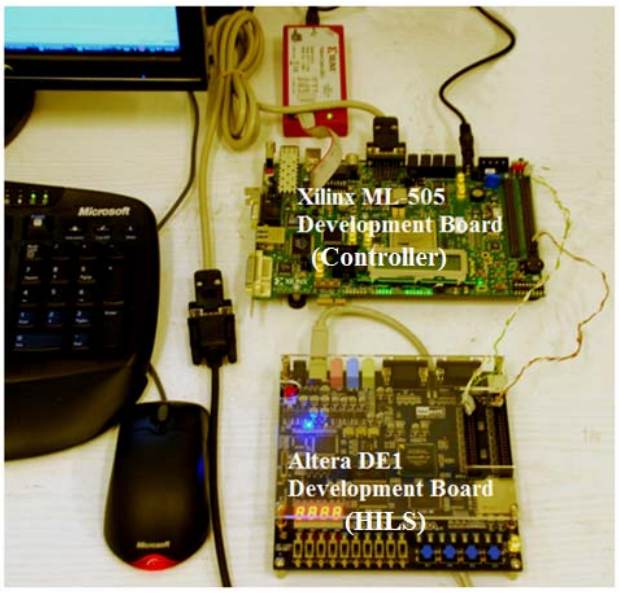


Fig. 9. Test setup

As can be seen in Fig. 8, the communication with the simulator is accomplished via PWM signals. A 10-bit resolution is selected for the PWM signal to represent torque command and position feedback. In order to prevent problems that could arise from insufficient resolution of the PWM signal, the position difference is transmitted from the HILS setup. Note that the real setup, which is shown in Fig. 9, consists of two FPGA development boards. One of them is the Xilinx ML-505 development board on which the controller modules are implemented. Likewise, an Altera DE1 development board performs the HILS. Boards are connected through I/O pins, facilitating the PWM connection.

V. RESULTS

In this study, 4 different controller topologies are designed and implemented on an FPGA chip. In this section, the results of the HILS are presented and controller topologies are evaluated by their success (tracking performance, disturbance rejection) and their resource cost on an FPGA chip.

The reference input to the system is given as position difference between time samples and a constant 10 Nm torque is applied at t = 1s to simulate the interrupted machining operation. Responses of the system with different controllers are presented in Fig. 10. The inset in Fig. 10 highlights the disturbance rejection characteristics of each controller. As can be observed, all the controllers achieve similar performances until the disturbance kicks in at 1s. From this point on, the best performances are achieved by the PID+CFF and sliding mode controllers. While the system under fuzzy control is affected slightly, the controller acts fast enough to compensate the disturbance. Similarly, the hysteresis controller also rejects the disturbance while maintaining an (tracking) error within hysteresis band. Note that from the standpoint of performance, all of these controllers are comparable.

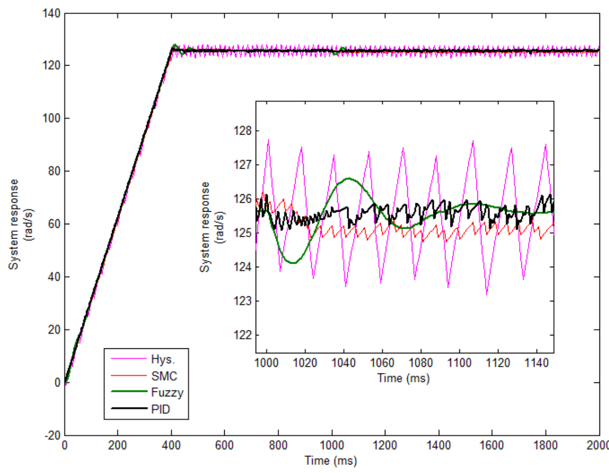


Fig. 10. Controller performances under disturbance input

TABLE II
RESOURCE COSTS OF DIFFERENT CONTROLLER TOPOLOGIES ON THE XILINX VIRTEX-5 FPGA CHIP

	Slice LUTs	Slice Registers
PID+CFF	5015 (17%)	1878 (6%)
Fuzzy	12863 (44%)	2448 (8%)
SMC	4192 (14%)	1675 (5%)
Hysteresis	1163 (4%)	490 (1%)

On the other hand, resource costs of the controllers vary in a wide range. These requirements are given in Table II in terms of slice look up tables (LUTs) and slice registers followed by the percentage consumed out of the available amount on the FPGA chip.

As can be seen from Table II, the PID+CFF and sliding mode controllers are consuming around 5% of the registers and 15% of the LUTs available in the FPGA chip. If a multiple-axis solution is sought on the chip, these numbers are sufficient enough for driving 5-axis simultaneously. On the other hand, fuzzy controller requires high amount of resources as Slice LUTs and therefore does not seem applicable for multi-axis solutions. Hysteresis controller, being an extremely simple control methodology, requires minimal amount of resources.

The complexity in the design affects not only the resources but also the attainable sampling period of the controller. In Table III, maximum attainable sampling rates by controllers are presented. As seen in Table III, all the controllers (except hysteresis) can perform the required calculations around 85 cycles, which corresponds to 1.7 µs on an FPGA with a 50 MHz clock.

TABLE III
MINIMUM ATTAINABLE SAMPLING PERIODS OF CONTROLLERS

	Cycles to complete loop	Minimum period
PID+CFF	80 cycles	1.6 µs
Fuzzy	87 cycles	1.74 µs
SMC	84 cycles	1.68 µs
Hysteresis	6 cycles	120 ns

On the implementation end, hysteresis controller is certainly the fastest and easiest to implement. PID and sliding mode controllers may also be implemented with a sufficiently higher effort. However, fuzzy controller is complicated to implement on an FPGA chip, especially when an embedded processor is not employed.

VI. Conclusion

In this paper, different controller topologies are implemented on a Virtex 5 FPGA for the purpose of performance evaluation. Soft-core processors (like MicroBlaze or PowerPC) are deliberately not employed in the designs so as to take full advantage of the parallel processing capability of FPGA. All necessary modules (IPs) are developed by Verilog HDL. Key results of this study may be summarized as follows:

- Several standard modules (encoder decoder, PWM generator, reference profile generator, etc.) for industrial motion control have been developed and implemented with a controller on a single FPGA chip.
- Control algorithms are implemented on an FPGA chip without using an embedded processor or any other complex IPs other than a single floating point arithmetic calculation unit.
- Control algorithms are compared by their success and their resource cost on an FPGA chip. Results are discussed and important features are highlighted.

References

[1] Xiaoyin Shao, Dong Sun, "An FPGA Based Motion Control IC and Its Application to Robotic Manipulators," *9th International Conference on Control, Automation, Robotics and Vision, 2006.*, pp.1-6, 5-8 Dec. 2006.

[2] http://www.altera.com/cgi-bin/device_compare.pl

[3] Xu Dong; Wang Tianmiao; Wei Hongxing; Liu Jingmeng, "A new dual-core Permanent Magnet Synchronous Motor Servo System," *Industrial Electronics and Applications, 2009. ICIEA 2009. 4th IEEE Conference on* , vol., no., pp.715-720, 25-27 May 2009.

[4] Birou, I.; Imecs, M., "Real-time robot drive control with PM-synchronous motors using a DSP-based computer system," *Power Electronics and Motion Control Conference, 2000. Proceedings. IPEMC 2000. The Third International* , vol.3, no., pp.1290-1295 vol.3, 2000.

[5] Seul Jung; Sung su Kim, "Hardware Implementation of a Real-Time Neural Network Controller With a DSP and an FPGA for Nonlinear Systems," *Industrial Electronics, IEEE Transactions on* , vol.54, no.1, pp.265-271, Feb. 2007.

[6] Ni, F.L.; Jin, M.H.; Xie, Z.W.; Shi, Sh.C.; Liu, Y.Ch.; Liu, H.; Hirzinger, G., "A Highly Integrated Joint Servo System Based on FPGA with Nios II Processor," *Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on* , vol., no., pp.973-978, 25-28 June 2006.

[7] Li, Yan; Zhuang, Shengxian; Zhang, Luan, "Development of an FPGA-Based Servo Controller for PMSM Drives," *Automation and Logistics, 2007 IEEE International Conference on* , vol., no., pp.1398-1403, 18-21 Aug. 2007.

[8] Ying-Shieh Kung; Rong-Fong Fung; Ting-Yu Tai, "Realization of a Motion Control IC for X-Y Table Based on Novel FPGA Technology," *Industrial Electronics, IEEE Transactions on* , vol.56, no.1, pp.43-53, Jan. 2009.

[9] Jung Uk Cho; Quy Ngoc Le; Jae Wook Jeon, "An FPGA-Based Multiple-Axis Motion Control Chip," *Industrial Electronics, IEEE Transactions on* , vol.56, no.3, pp.856-870, March 2009.

[10] Chan, Y.F.; Moallem, M.; Wang, W., "Efficient implementation of PID control algorithm using FPGA technology," *Decision and Control, 2004. CDC. 43rd IEEE Conference on* , vol.5, no., pp. 4885-4890 Vol.5, 14-17 Dec. 2004.

[11] Yao dong Tao; Hu Lin; Yi Hu; Xiaohui Zhang; Zhicheng Wang, "Efficient implementation of CNC Position Controller using FPGA," *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on* , vol., no., pp.1177-1182, 13-16 July 2008.

[12] Jia Lanping; Zhou Runjing; Liang Zhian, "Realization of position tracking system based on FPGA," *Signal Processing, 2004. Proceedings. ICSP '04. 2004 7th International Conference on* , vol.3, no., pp. 2588-2591 vol.3, 31 Aug.-4 Sept. 2004.

[13] Faa-Jeng Lin; Ying-Chih Hung, "FPGA-based elman neural network control system for linear ultrasonic motor," *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on* , vol.56, no.1, pp.101-113, January 2009.

[14] R. Usselmann, *Open Floating Unit Manual*, www.opencores.org, 2000.

[15] K.J. Astrom and B. Wittenmark, *Computer Controlled Systems*, Prentice Hall, New Jersery, USA, 1997.

[16] Tsai, M.-C.; Chiu, I.-F.; Cheng, M.-Y., "Design and implementation of command and friction feedforward control for CNC motion controllers," *Control Theory and Applications, IEE Proceedings* - , vol.151, no.1, pp. 13-20, 17 Jan. 2004.

[17] Liaw, C.-M.; Wang, J.-B., "Design and implementation of a fuzzy controller for a high performance induction motor drive," *Systems, Man and Cybernetics, IEEE Transactions on* , vol.21, no.4, pp.921-929, Jul/Aug 1991.

[18] Alexei, Z.; Sandor, H., "Robust speed fuzzy logic controller for DC drive," *Intelligent Engineering Systems, 1997. INES '97. Proceedings., 1997 IEEE International Conference on* , vol., no., pp.385-389, 15-17 Sep 1997.

[19] Sio, K.C.; Lee, C.K., "Identification of a nonlinear motor system with neural networks," *Advanced Motion Control, 1996. AMC '96-MIE. Proceedings., 1996 4th International Workshop on* , vol.1, no., pp.287-292 vol.1, 18-21 Mar 1996.

[20] Slotine, J.J.E., and Li, W., *Applied Nonlinear Control,* Prentice-Hall, 1991.

[21] Brock, S.; Deskur, J.; Zawirski, K., "Modified sliding-mode speed controller for servo drives," *Industrial Electronics, 1999. ISIE '99. Proceedings of the IEEE International Symposium on* , vol.2, no., pp.635-640 vol.2, 1999.

[22] Zhang, D.Q.; Panda, S.K., "Chattering-free and fast-response sliding mode controller," *Control Theory and Applications, IEE Proceedings* - , vol.146, no.2, pp.171-177, Mar 1999.

[23] Usenmez, S.; Dilan R.A; Dolen M.; Koku A.B., “Real-Time Hardware-in-the-Loop Simulation of Electrical Machine Systems Using FPGAs”, to appear in the Proc. of the *International Conference on Electrical Machines and Systems (ICEMS), 2009.*

Real-Time Hardware-in-the-Loop Simulation of Electrical Machine Systems Using FPGAs

S. Usenmez*, R.A. Dilan*, M. Dolen* and A.B. Koku*
*Department of Mechanical Engineering, Middle East Technical University, Turkey

***Abstract*-- This study focuses on the development an integrated software and hardware platform that is capable of performing real-time simulation of dynamic systems, including electrical machinery, for the purpose of hardware-in-the-loop simulation (HILS). The system to be controlled is first defined using a block diagram editor. The defined model is then compiled and downloaded onto an FPGA ("Field Programmable Gate Array") based hardware platform, which is to interface with the controller under test and carry out the simulation in real-time. The full paper will elaborate the characteristics of this novel HILS platform.**

***Index Terms*-- Electrical Machine Systems, Field Programmable Gate Array, Hardware in the Loop Simulation**

I. INTRODUCTION

During the development of control systems, it is necessary to investigate the controller performance and fine tune its parameters. This ensures maximum possible utilization of the product on which the controller is to be used. It is evident that the best method of performing such work is to connect the controller being developed to the system of interest itself. However, this might not always be desirable due to reasons such as system cost and risk of damage on malfunction. Furthermore, fitting the system of interest with the various sensors that are necessary to monitor the testing processes is a complex task requiring great care.

Hardware-in-the-Loop Simulation is an invaluable tool that circumvents such difficulties. It provides the designer with the ability to perform numerous experiments, while monitoring all desired system states with ease.

This study aims to develop an integrated and flexible HILS solution that consists of a simulator hardware platform supported by a user-friendly software package. The organization of the paper is as follows: Next section presents a brief background on HILS applications. Section III gives the details of the solution proposed by the study, while section IV explains a sample application. Finally the last section draws a conclusion.

II. BACKGROUND

HILS is an important technique involving the testing of developed hardware and/or software on a virtual system which is run by a computer or various other chips. In control engineering, this technique is frequently used during the solution of design problems in order to test systems that are difficult or costly to test otherwise [1-2]. During the last decade, HILS applications using mostly personal computers and DSP ("digital signal processor") chips [3-4] have been employed.

Use of FPGA chips, which have found widespread use thanks to their exceptional capability of parallel processing, has increased in recent HILS studies. FPGAs have become the choice of researchers that desire to perform real-time applications such as HILS. Researchers have shown that use of FPGA chips in HILS applications results in high performance in speed when compared to other simulation applications such as MATLAB/Simulink and UPPAAL [5]. Similarly, researchers have succeeded in minimizing the time between fault occurrence and detection in three-phase parallel circuit active filters to 10μs by using FPGA-based HILS [6]. Another HILS application was developed for simulation of permanent-magnet synchronous motors. In this study, the motor model is obtained via finite element analysis and realized on an FPGA chip. The researchers were able to reduce the simulation time step to as low as 1.3μs [7-8].

FPGA chips are used for simulation of electrical systems in various researches. In performing simulation of an electrical system, the multiple sampling approach was used by utilizing the FPGA's parallel processing capabilities. This has made it possible to reduce the time step to as low as 2μs, allowing real-time behavior [9]. Real-time simulation of DC machines has been the subject of another research which showed that simulations using FPGAs have outperformed commercial simulation software [10].

It has been shown that use of new architectures on FPGAs in real-time simulation of power systems greatly improve simulation performance while the cost of such a solution is approximately the same as older approaches [11]. FPGA chips have also been used in real-time simulation of power electronic converters. The performance of the application has been verified with a MATLAB application in parallel and was showed to have better performance [12]. Real-time simulation of aerospace power systems using FPGAs has been performed and the simulation time step was reduced to 0.4μs [13].

In an attempt to develop more flexible solutions, researchers have developed a floating point block library and a multi-chip FPGA platform for use with MATLAB in order to perform HILS [14]. Aside from researches, a number of commercial products for HILS also exist in the

This work was supported by the Scientific and Technological Research Council of Turkey under the project contract 108E048.

market. RT-LAB and ADvantage Framework provide tools for designing HILS processes while Modular Real-time Target Machine is an expandable hardware platform for realizing simulations developed using MATLAB [17-19]. On the other hand, dSPACE Simulator and LabVIEW FPGA are products that attempt to provide both hardware and software solutions for HILS purposes [20-21].

III. Solution Being Developed

The solution developed in this study consists of three main elements: a *software package* which is used to describe the system to be simulated, a *hardware platform* on which the simulation computations are performed and *interface emulators* that enable the simulation to properly communicate with other devices. The following three subsections give the details of each of these elements.

A. Software Package

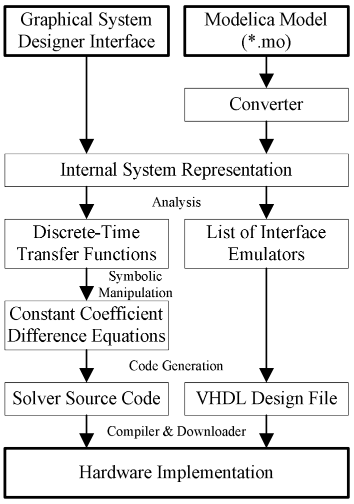
The software package developed provides the user with the tools for representing the simulated system. A graphical user interface allows the user to describe the system using a library of commonly used blocks such as transfer functions, arithmetic and logic operators etc. These blocks can be arranged to represent virtually any system. The user also specifies the states to be monitored throughout the HILS session. In addition, the package can import system models defined by other means such as MATLAB/Simulink files or Modelica language [23].

After the definition process, the package manipulates the differential equations governing the system states to obtain a set of constant coefficient difference equations ("CCDEs"), which can be used to solve for these states. Then, a C source code which solves these equations is generated and compiled for the processor used in the hardware platform (see next sections). Additionally, a Very-high-speed Hardware Description Language (VHDL) code that instantiates the necessary simulation-controller interface emulators is generated and synthesized. Finally, these are downloaded onto the hardware platform. Fig. 1 illustrates the basic steps taken by the software.

The software package also serves as a terminal for monitoring and recording the desired system states. The user can watch the states during runtime and export these to other software packages for further analysis.

B. Hardware Platform

FPGA is a programmable chip designed to allow digital circuit designs to be easily implemented. The hardware platform accommodates an FPGA chip as the main workhorse. This chip is used to instantiate a microprocessor that will solve for the system states, as well as the necessary interface emulators. A number of peripheral units are connected to the FPGA: serial communication adapters, memory elements and storage devices. The hardware platform also accommodates a flexible connectivity interface, which can be reconfigured to communicate with a variety of controller hardware using almost any connection type (analog and digital inputs/outputs, encoders etc. and various connector geometries).


Graphical System Designer Interface
Modelica Model (*.mo)
Converter
Internal System Representation
Analysis
Discrete-Time Transfer Functions
List of Interface Emulators
Symbolic Manipulation
Constant Coefficient Difference Equations
Code Generation
Solver Source Code
VHDL Design File
Compiler & Downloader
Hardware Implementation
Fig. 1. Steps in converting system model to simulation.

After the simulation is set up and downloaded, the hardware platform performs the simulation computations while communicating with the controller hardware. The system variables requested by the user are also sent to the user's PC during run time. If, however, the number of states requested is too many and the simulation step size is very small, this task requires a very large bandwidth. In such a case, the states are sent to the PC only at certain time steps. The rest of the states are saved on the storage elements on the platform, only to be dumped to the PC after the simulation is completed.

If the simulation requires so, the platform is capable of working in conjunction with the user's PC during simulation. When processor-intensive, slow operations are required; the PC performs these and passes the results to the hardware platform, while the platform itself handles high-speed computations.

C. Interface Emulators

A number of external interface cores for the FPGA chip are developed in order to enable the simulation to communicate with external devices (e.g. the controller under test) or imitate peripherals (e.g. encoders or other sensors).

Encoder pulse generator core imitates a quadrature encoder. The input to this core is a floating point number representing the angular position of the encoder shaft in radians. This value is sampled and converted to an integer based on the desired pulse-per-revolution number. The integer is then used to obtain the states of the encoder's three output signals (channels A and B, index). The signal generation requires 7 clock cycles, meaning a ~7MHz sampling rate when a 50MHz clock generator is used.

10-bit PWM value transmitter and receiver cores allow for 10-bit digital data communication between the simulator and external components.

IV. Sample Application

In order to obtain a proof of concept for the proposed solution, a simple HILS application was designed. An induction motor, which is used as spindle drive in a turning center, is simulated in this application. The following sections elaborate on the details of this system and its implementation.

A. System Model

The induction motor used in the system is assumed to be connected to a direct torque controller (DTC), which receives the desired torque command and drives the motor [15]. A timing belt connects the rotor shaft to the spindle shaft. A disturbance torque due to the cutting forces occurring during the machining process acts on the work piece. In Fig. 2 showing the spindle drive system; T^* represents the torque command, T_d is the disturbance torque, r_1 and r_2 are the pulley radii, J_1 and J_2 are the moments of inertia, b_1 and b_2 are the viscous friction coefficients, and θ is the angular position of the spindle, i_a, i_b and i_c are the motor winding currents and e is the encoder signal.

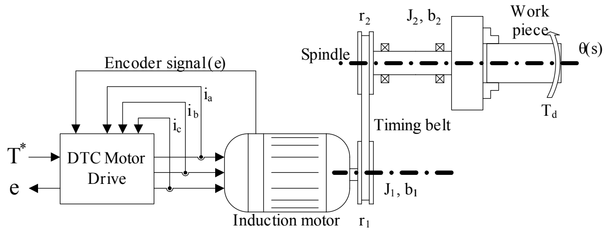


Fig. 2. Spindle drive system.

When the DTC is assumed to be ideal, the motor can be taken as a component which applies the desired torque directly on the rotor shaft and other connected bodies. However, electric motors have a certain torque generation limit, or maximum torque curve, which is a function of the motor velocity (ω_m). This torque curve consists of three distinct regions: *constant torque* region, *constant power* region and *natural mode* region (Fig. 3). Up to the rated speed (ω_r), the motor is able to deliver a maximum torque equal to its rated torque (T_r). Beyond this speed, the motor enters the constant power region, where the maximum delivered torque is inversely proportional to the speed, maintaining constant power output. At much higher speeds, the natural region begins and the torque becomes inversely proportional to the square of the speed. In most applications, however, the motor is not pushed into this region, and ω_p is considered to be the maximum speed the motor can attain [16].

Using the explained model, the envelope of the torque capability of the motor is

$$T_{\max} = \begin{cases} T_r, & |\omega_m| \le \omega_r \\ \dfrac{T_r \omega_r}{|\omega|}, & \omega_r < |\omega| \le \omega_p \\ 0, & else \end{cases} \tag{1}$$

and the torque applied by the motor within this limit is

$$T_m = \begin{cases} \text{sgn}(T^*) T_{\max}, & T^* \ge T_{\max} \\ T^*, & else \end{cases} \tag{2}$$

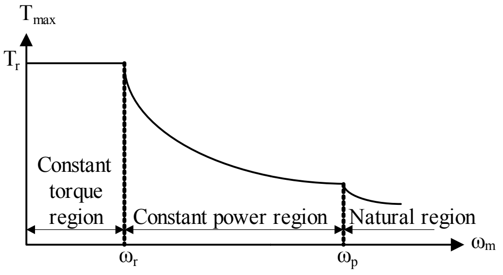


Fig. 3. Torque-speed characteristic of electric motors.

The transmission between the motor shaft and the spindle is assumed to be ideal, i.e. the timing belt perfectly transfers torque between the rotor and spindle shafts. It is then possible to model the combined rotor and spindle (and mounted work piece) loads as a single rotating mass. Then, the equivalent moment of inertia can be obtained by summing the rotor inertia with the spindle inertia, multiplied by the square of the transmission ratio. Likewise, the equivalent viscous friction is the sum of rotor friction with the spindle friction, multiplied by the transmission ratio.

The direct torque controller in the simulated system is assumed to be receiving the torque command via a PWM resolver connected to a D/A converter. Since the simulation is performed digitally, instead of simulating a D/A converter, the inputs are fed into the PWM receiver and a 10-bit integer representing the torque command is obtained. The encoder emulator generates the encoder signals from the angular position of the spindle. These signals are counted by a quadrature counter and their difference is transmitted to the controller via another 10-bit PWM signal. The resulting simulated system model is given in Fig. 5.

The disturbance torque acting on the work piece (and spindle shaft) is taken as a repeating series of discrete pulses, imitating the effect of intermittent contact between the cutter and work piece during the machining process. The maximum magnitude of this disturbance is selected as 10Nm (Fig. 4). In order to allow the motor to reach a high enough speed, the disturbance is introduced only after the simulation time reaches 8 seconds.

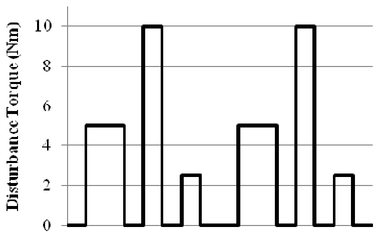


Fig. 4. Form of the disturbance torque applied on the spindle shaft.

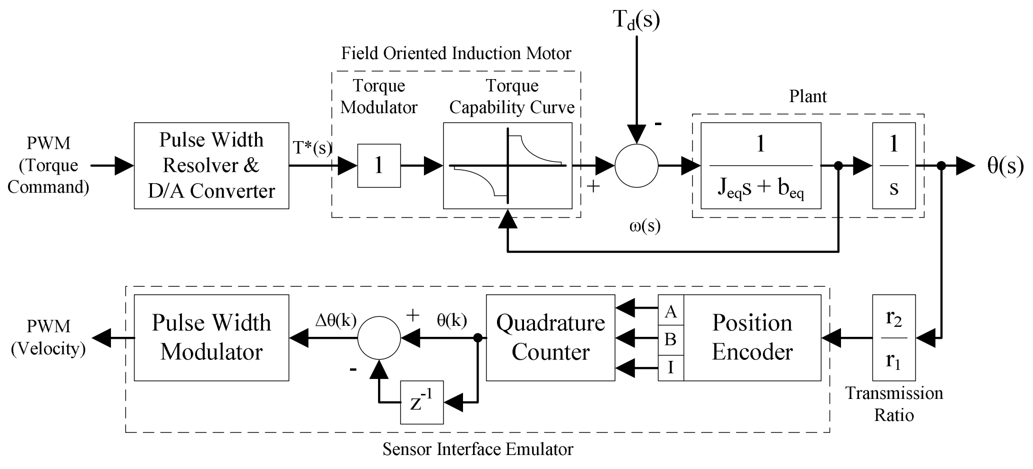


Fig. 5. Block diagram of sample application system.

For the purposes of the sample application, the rated motor torque is selected as 35Nm. The rated speed is 1500rpm while the maximum speed is 8000rpm, resulting in an approximate motor power of 5.5kW. The equivalent moment of inertia is taken as 0.07Nm^2, the equivalent viscous friction coefficient is taken 0.008Nms and the ratio of pulley radii is unity. Finally, using a sampling time of 1ms, the discrete-time transfer functions governing this system states as a function of the net torque (T_{net} = T_m - T_d) on the spindle shaft can be obtained as

$$\frac{\dot{\theta}(z)}{T_{net}(z)} = \frac{0.01428}{z - 0.9999} \tag{3}$$

$$\frac{\theta(z)}{T_{net}(z)} = \frac{7.143\times10^{-6}z + 7.142\times10^{-6}}{z^2 - 2z + 0.9999} \tag{4}$$

B. Implementation

The sample application is implemented on the Altera DE1 Development and Education board. This board accommodates the Altera Cyclone II 2C20 FPGA along with an 8MB SDRAM chip, USB Blaster serial communication adapter, a 50MHz oscillator and many other useful peripherals [24]. On the FPGA chip, an instance of the Nios II 32-bit Embedded Processor, which is designed by Altera solely for FPGA implementation, is implemented. This processor design includes a floating point arithmetic unit ("FPU") implementation, and can be connected to modular interfaces to a variety of peripheral units which include those on the DE1 board [25]. The Nios II processor and the interfaces needed to utilize the aforementioned peripherals are instantiated and connected together using the development tools provided by Altera, and a "system on programmable chip" (SOPC) is obtained [26]. This system is coupled with the encoder generator and PWM transmitter and receiver components, and downloaded onto the FPGA chip. The resulting configuration is illustrated in Fig. 6.

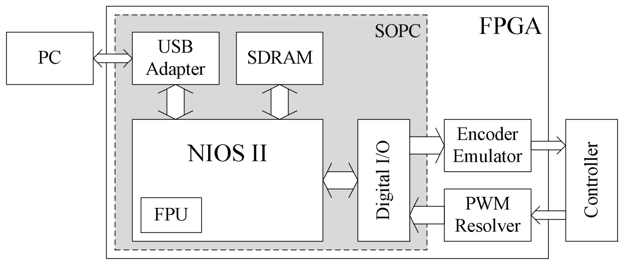


Fig. 6. Hardware configuration for the sample application.

A C program that will perform the simulation calculations of the explained system is written and compiled for the Nios II processor. The program consists of a main loop that repeatedly performs the communication and computation tasks. Instead of a continuously running loop, the program waits for a torque command to arrive from the controller. When a command is received, an interrupt is raised in the processor. The incoming command is read from the digital I/O interface, and the simulation computations for a single sampling period are performed. The shaft position, which is used to generate the encoder signal, is output from the processor via the digital I/O interface to the relevant register. All system states, as well as the torque command and net torque, are recorded in arrays stored inside the SDRAM chip. When the simulation is completed, these records are dumped to the user's PC via the USB connection. The source code is arranged in a modular fashion and can be easily decomposed to obtain templates for automated generation in the future.

C. Application Results

The offline testing of the application is performed by applying a constant torque command of 30Nm on the system. The results obtained are compared against solutions performed on PC using the MATLAB/Simulink software package. Two different implementations are tested, using single- and double-precision floating point representations. The performance of the simulator is also evaluated in terms of computation time by utilizing the performance counter component of the Nios II processor.

Table I presents the test results while the resource usage of the application is given in Table II.

TABLE I
OFFLINE SIMULATION TEST RESULTS

		Single-precision implementation	Double-precision implementation
Difference from Simulink in ω	RMS	410.1μrad/s	409.2μrad/s
	Mean	51.5mrad/s	51.3mrad/s
	Maximum	240.2mrad/s	240.2mrad/s
Difference from Simulink in θ *	RMS	3.7μrad	3.7μrad
	Mean	466.9mrad	470.3mrad
	Maximum	869.2mrad	881.4mrad
Computation Time	RMS	13.3μs	111.7μs
	Mean	13.2μs	111.5μs
	Maximum	26.8μs	140.4μs

* After 20 seconds of simulation.

TABLE II
SAMPLE APPLICATION RESOURCE USAGE

Total logic elements	10,737 (57%)
Total combinational functions	9,121 (49%)
Dedicated logic registers	6,447 (34%)
Total memory bits	66,040 (28%)
Embedded multiplier 9-bit elements	11 (21%)
Program size in SDRAM memory**	52kB (0.006%)

* Percentages are based on resources on the Altera DE1 board.
** Excluding stack and heap memories.

The numerical results of the simulation appear to have an acceptable amount difference from those obtained using Simulink, due to the difference between the employed solution techniques. It should be noted here that the error in angular position is the integral of the error in speed and will vary with the simulation duration. Again in terms of numerical results, there appears to be no significant difference between single- and double-precision floating point representation techniques. The single-precision solution times, however, are much lower than double-precision solution times. The single-precision implementation is therefore an adequate solution as far as the sample application is concerned.

In order to further explore the performance of the sample application, the simulation program is modified to solve for CCDEs of varying degrees and the computation times are measured. Results are given in Fig. 7.

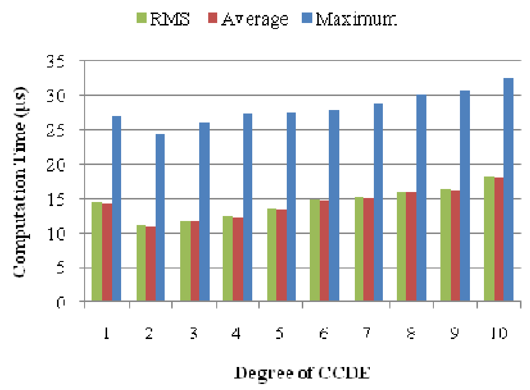


Fig. 7. Computation time by CCDE degree, one equation per loop.

For the sake of comparison, the solution of the system CCDEs is also performed using direct hardware instantiations of floating point arithmetic units, instead of employing the Nios II processor. The FPU design used in this test is capable of performing a floating point operation every 4 clock cycles [27]. The units are arranged in a binary-tree fashion, first performing all multiplication operations in parallel and then adding up all the results together with maximum possible parallelism. In such an arrangement, the solution time can be calculated as

$$n_c = 4 \times (\lceil \log_2(n_t) \rceil + 1) \tag{5}$$

where n_t is the number of terms added together in the CCDE and n_c is the number clock cycles required to obtain the solution. For this approach, the solution times using a 50MHz clock signal are given in Fig. 8, and Fig. 9 shows the approximate resource usage.

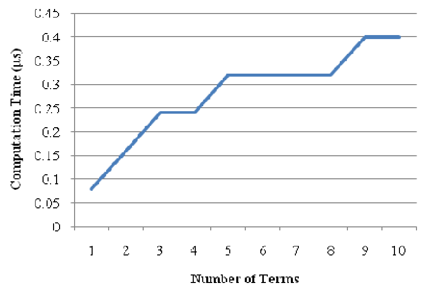


Fig. 8. Hardware implementation computation time by CCDE degree.

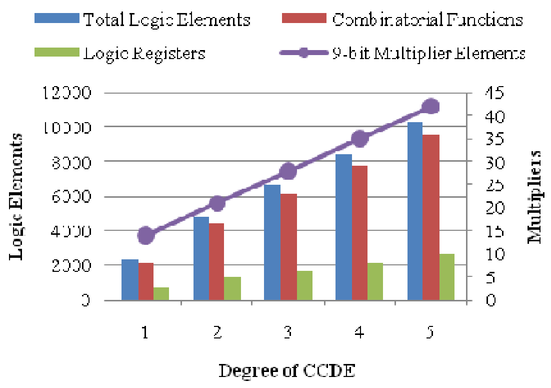


Fig. 9. Hardware implementation resource usage by CCDE degree.

A final comparison is made with a numerical method approach, the Runge-Kutta algorithm of order 4 (commonly abbreviated "RK4") [17]. The method is implemented on the Nios II processor, only modifying the solution part of the simulation program. The state space equations governing the system are obtained and also coded into the solver, and numerical integration is performed. Two tests are made, one with a single step per loop (1ms step size) and one with 10 steps per loop (0.1ms step size). The computation times are presented in Table III, while the resource usage is identical to the originally proposed solution.

TABLE III
4TH ORDER RUNGE-KUTTA METHOD COMPUTATION TIMES

	1 step per loop	10 steps per loop
RMS	19.1μs	63.5μs
Mean	19.0μs	63.4μs
Maximum	36.0μs	70.3μs

V. DISCUSSION AND CONCLUSION

This paper proposes an integrated, complete and user-friendly solution for preparing and executing accurate hardware-in-the-loop simulations in real-time. The block diagram style system definition tools, combined with the ability to import existing models, enable the user to easily describe the simulated system. Automated generation of all necessary machine code saves the user from having to learn a variety of programming languages and tools, and makes it possible to focus on the simulation itself. The powerful hardware platform, coupled with a flexible connectivity interface, further reduces the difficulty of the preparation process, enabling the user to spend less time preparing and more time simulating.

The application sample is a good demonstration of the computational capabilities of the proposed solution. The CCDE solution times show that it is possible to successfully simulate systems of order up to 10 using sampling times as high as 5 kHz.

Although implementing a direct hardware solution method makes it possible to perform the computations at much higher speeds, the rapidly increasing resource usage limits the capability of this approach. Moreover, introduction of non-arithmetical operations, such as conditional statements, requires redundant resource usage in order to maintain the desired computational speed, further limiting the method's capabilities.

While solving CCDEs to obtain system states is quite fast, it has the limitation of only being able to represent linear systems. When non-linear elements exist in the system to be simulated, linearization techniques become necessary to obtain a CCDE, but the accuracy of the model is greatly reduced. In such cases, it is possible to resort to numerical methods such as the RK4. It is also shown that the proposed solution can be easily modified to use this approach, still being able to perform simulations with sampling times up to 2 kHz successfully.

Majority of the remaining work on the proposed system is the completion of the software package as well as the development of a library of system components and interface emulators. The connectivity interface of the hardware platform is also being developed. When these are completed, a standalone, easy to use hardware-in-the-loop simulation solution will be achieved.

REFERENCES

[1] W. Grega, "Hardware-in-the-loop simulation and its application in control education," *29th ASEE/IEEE Frontiers in Education Conference*, 1999.

[2] M. Bacic, "On hardware-in-the-loop simulation," *Proceedings of the 44th IEEE Control Conference*, 2005.

[3] R. Crosbie, J. Zenor, R. Bednar, N. Hingorani and T. Ericsen, "High-Speed, Scalable, Real-Time Simulation Using DSP Arrays," *IEEE Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, Kufstein, Austria, vol.2, p.52-59, 2004.

[4] P. Baracos, G. Murere, C.A. Rabbath and W. Jin, "Enabling PC-Based HIL Simulation for Automotive Applications," *IEEE International Electric Machines and Drives Conference (IEMDC 2001)*, Boston, U.S.A., vol.1, p.721-729, 2001.

[5] J. Krakora and Z. Hanzalek, "FGPA Based Tester Tool for Hybrid Real-time Systems," *Microprocessors and Microsystems - Embedded Hardware Design*, vol.32, p.447-459, 2008.

[6] S. Karimi, P. Poure and S. Saadate, "FPGA-based Hardware in the Loop Validation for Fault Tolerant Three-Phase Active Filter," *IEEE International Symposium on Industrial Electronics (ISIE 2008)*, Cambridge, UK, vol.1, p.2189-2194, 2008.

[7] C. Dufour, J. Belanger, V. Lapointe and S. Abourida, "Real-Time Simulation on FPGA of a Permanent Magnet Synchronous Machine Drive Using a Finite-Element Based Model," *9th International Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM 2008)*, Ischia, Italy, vol.6, p.19-26, 2008

[8] S. Abourida, J. Belanger and C. Dufour, "Real-Time HIL Simulation of a Complete PMSM Drive at 10μs Time Step," *2005 European Conference on Power Electronics and Applications*, Dresden, Germany, vol.1, p.9-P.9, 2005.

[9] I. Bahri, M-W. Naouar, E. Monmassın, I. Slama-Belkhodja and L. Charaabi, "Design of an FPGA-Based Real-Time Simulator for Electrical System," *13th IEEE Power Electronics and Motion Control Conference (EPE-PEMC 2008)*, Poznan, Poland, vol.4, p.1365-1370, 2008

[10] K. Jayalakshmi and V. Ramanarayanan, "Real-Time Simulation of Electrical Machines on FPGA Platform," *IEEE Proceedings of India International Conference on Power Electronics*, Chennai, India, v.3, p.259-263, 2006

[11] J.C.G. Pimentel, "Implementation of Simulation Algorithms in FPGA for Real Time Simulation of Electrical Networks with Power Electronics Devices," *IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006)*, San Luis Potosi, Mexico, vol.1, p.1-8, 2006.

[12] P. Le-Huy, S. Guerette, L.A. Dessaint and H. Le-Huy, "Dual-Step Real-Time Simulation of Power Electronic Converters Using an FPGA," *IEEE International Symposium on Industrial Electronics*, Montreal, Quebec, Canada, vol.7, p.1548-1553, 2006.

[13] J.C.G. Pimentel and Y.G. Tirat-Gefen, "Real-Time Hardware in the Loop Simulation of Aerospace Power Systems," *5th International Energy Conversion Engineering Conference and Exhibit (IECEC 2007)*, St. Louis, Missouri, U.S.A., vol.2, p.8-17, 2007.

[14] D. Castells-Rufas and J. Carrabina, "Jumble: A Hardware-in-the-Loop Simulation System for JHDL," *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*, Napa Valley, California, U.S.A., vol.1, p.345-348, 2007.

[15] E. Ozkop and H.I. Okumus, "Direct Torque Control of Induction Motor Using Space Vector Modulation (SVM-DTC)", *12th International Middle-East Power System Conference (MEPCON 2008)*, Aswan, Egypt, p.368-372, 2008.

[16] S.W. Moore, K.M. Rahman and M. Ehsani, "Effect on Vehicle Performance of Extending the Constant Power Region of Electric Drive Motors", *SAE 1999-01-1152*, 1999.

[17] W. H. Press, S.A. Teukolsky, W.T. Vettering and B.P. Flannery, *Numerical Recipes in C*. Cambridge University Press, 1992.

[18] dSPACE GmbH, http://www.dspace.de

[19] Opal-RT Technologies Inc., http://www.opal-rt.com

[20] Applied Dynamics International Inc., http://www.adi.com

[21] Speedgoat GmbH, http://www.speedgoat.ch

[22] National Instruments Inc., http://www.ni.com

[23] Modelica Language, http://www.modelica.org/

[24] Altera DE1 Development and Education Board User Manual Version 1.1, http://www.terasic.com.tw/cgi-bin/page/archive.pl?No=83, 2006.

[25] Nios II Processor Reference Handbook Version 9.0, http://www.altera.com/literature/lit-nio2.jsp, March 2009.

[26] Quartus II Handbook Version 9.0, http://www.altera.com/literature/lit-qts.jsp, March 2009.

[27] R. Usselmann, "Open Floating Point Unit Manual", http://www.opencores.org/, 2009.

Direct Command Generation Methods for Servo-Motor Drives

U. Yaman[1], B. R. Mutlu[1,2], M. Dolen[1], and A. B. Koku[1]
[1] Department of Mechanical Engineering, Middle East Technical University, Ankara, Turkey
[2] Department of Mechanical Engineering, Hacettepe University, Ankara, Turkey

Abstract--**This study focuses on advanced direct command generation paradigms that can be easily incorporated to modern servo-motor drive systems. As an alternative to conventional methods, two new command generation techniques are proposed in this paper. In the first method, higher-order differences of a given trajectory (i.e. position) are computed and the resulting data are compressed via lossless data compression techniques. Besides conventional approaches, a novel compression algorithm is also introduced in the paper. With respect to the second method, the command trajectory is first divided into segments according to the inflection points. The segments are then approximated using various polynomial techniques. The sequence originating from modeling error is included to the generated series. These two command generation techniques are then implemented using Field Programmable Gate Arrays (FPGAs) and their performances are evaluated according to the resources used in FPGAs and the speed of computation.**

Index Terms--**Command Generation, Data Compression, FPGA, Polynomial Approximation**

I. Introduction

Modern servo drive systems employ digital motion controllers (DSPs, micro-controllers) to regulate precisely not only motor currents (electromagnetic torque) but also motor's angular velocity along with the position. If the drive system is configured for (digital) motion control, the relevant reference signals (velocity or position) must be generated by a central controller unit (host) depending on the trajectory to be followed. These signals are eventually transferred to each motor driver via a serial communication protocol (SERCOS, CAN, Profibus, TCP/IP, RS-232, RS-485, etc.). This approach frequently pushes the communication interface to its limits for high-end applications. Consequently, the objective of this study is to develop a direct command generator system for servo motor drives where the commands can be produced directly in the drive system without the need for intermittent data transfer from a host controller. This FPGA based system, which could be directly embedded into a motor drive system, is to generate the relevant commands by utilizing structured and compressed data being produced in advance to represent trajectory to the desired accuracy.

Industrial motion controller units frequently utilize vector data tables to represent the trajectory in terms of linear patches. These cards can then perform a linear interpolation between the two consecutive entries in real-time to produce the relevant reference signals for the position servo-control loop. For complex trajectories the size of the vector table may exceed the available resources on the system.

In today's technology, memory devices with large capacity as well as FPGAs with great parallel processing capabilities are becoming widely available in the market at relatively low cost. Consequently, there is a potential for devising simple yet very effective command generators for electrical drive systems that benefit fully from the properties of these advanced devices.

This paper focuses on two different techniques: the first technique, which is commonly referred as differencing in literature [1], computes the higher order differences of the trajectory data (e.g. position) and then uses universal data compression techniques (Huffman [2] and Arithmetic Coding [3]) and the novel compression method to reduce the size of the resultant data; the second technique utilizes segmentation with respect to the inflection points of the trajectory and then employs function approximation paradigms [4] such as, Chebyshev, Legendre, Bernstein – Bezier, etc. in order to represent the continuous trajectory efficiently. In the latter technique, errors resulting from approximation are also stored after applying the proposed data compression algorithm in the first technique to extract the original trajectory.

The paper is organized as follows: Section II represents a review of the relevant research fields. Section III describes the first direct command generation method using differencing and compression along with the novel compression method and its hardware realization. In Section IV, the second method, which is based on segmentation and polynomial approximation, is elaborated and FPGA implementation of these methods is discussed. Section V compares the techniques described in the previous sections in terms of compression ratio utilizing MATLAB. Finally, Section VI concludes this paper.

This work was supported by the Scientific and Technological Research Council of Turkey under the project contract 108E048.

II. Background

In this section, the technical literature related to the proposed techniques is summarized.

A. *Implementations of Data Compression Techniques*

Lossless data compression applications, where the original data is extracted without any loss of generality after decompression, have increased over the past years due to the need to improve the storage capacity and data transfer rate [5]. There are many examples of hardware implementations of conventional data compression techniques in the literature. Among these techniques Huffman [6]-[7], Lempel-Ziv (LZ) [8]-[9], and Golomb [10] compression algorithms are the most popular ones for FPGA implementations. Rigler *et al* [6] implemented Huffman and LZ encoders on an FPGA and concluded that modified Huffman algorithm uses less hardware resources than the LZ algorithm. El ghany *et al.* [8] also implemented the LZ encoding and decoding algorithm on FPGA. In order to increase the efficiency they used systolic array approach which resulted in a 40% decrease for the compression rate.

Among conventional data compression techniques, hardware implementations of different algorithms for compressing specific data structures are also present in the literature. For instance, Yongming *et al.* [11] implemented the Linear Approximation Distance Threshold algorithm on FPGA in order to compress the Electrocardiograph signals. On the other hand, Valencia and Plaza [12] developed an FPGA-based data compression technique based on the concept of spectral unmixing to compress hyperspectral data. The novel compression method described in the paper can be regarded as specific since it is developed to compress the integer encoder pulses.

B. *Function Approximation Paradigms*

In the second method, function approximation algorithms are applied to model the segmented trajectory in a piecewise fashion. The aim of this segmentation is to decrease the error resulting from approximation. In the literature, there exist various segmentation approaches. The most common one is uniform segmentation, in which the widths of the segments are equal and the number of segments is limited to powers of two. Since the segment widths cannot be customized according to local function characteristics, a huge amount of segments are necessary to fulfill the error requirement [13]. To overcome this problem, dynamic segmentation depending on the inflection points of the trajectory is proposed in the paper.

Selecting an appropriate function approximation technique is very important especially in hardware implementations since errors resulting from the approximation should be stored for lossless reconstruction [14]. Another important aspect in approximation is the degree of polynomials used. When memory resources are limited, higher degree polynomials are commonly applied at the expense of increased computational complexity [15].

Hardware implementations of function approximation techniques are frequently used in developing Direct Digital Frequency Synthesizers (DDFS) in the literature. Ashrafi *et al.* [16] proposed an FPGA-based method that utilizes Chebyshev polynomial series interpolation. The developed technique unifies the results of ROM-less polynomial approximation methods for sinusoidal DDFS implemented on FPGAs [17]-[18]. Among various approximation techniques, Chebyshev polynomials are usually preferred for hardware implementation. This is due to the fact that Chebyshev polynomials give better results for non-periodic signals that are limited in range. In the paper it is also turned out that the performance of Chebyshev polynomials are superior to Legendre, and Bernstein – Bezier polynomials.

III. Direct Command Generation via Differencing and Compression

The first method suggested for command generation first calculates higher order differences of the command trajectory. Then, data compression algorithms (Huffman and Arithmetic Coding) are used in conjunction with differencing.

Methods involving higher order differences of sequences are usually applied to decrease the memory required for storage [1]. Rather than storing directly the command trajectory data, it is enough to store the differenced data and the necessary initial values. Fundamental differencing formula is as follows:

$$\nabla q = q(k) - q(k-1) \tag{1a}$$

$$\nabla^2 q = \nabla q(k) - \nabla q(k-1) \tag{1b}$$

$$\nabla^n q = \nabla^{n-1} q(k) - \nabla^{n-1} q(k-1) \tag{1c}$$

where $\nabla^n q$ is the nth order difference. As the order difference increases, the memory needed for the storage of the sequence decreases. In order to represent the relationship between the memory requirement and the order of difference, several trajectories are formed and their differences are taken up to 7th order. This correlation is shown in Fig. 1. As can be seen from the figure, after the third-order difference, there is an increase in the memory usage since the sign of each data point frequently changes in an alternating fashion after the fourth order difference. Hence, the range of data broadens considerably. Best solution for memory requirement is achieved when the order of difference is 3 or 4 for most of the motion control applications.

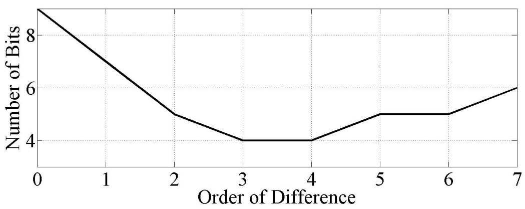


Fig. 1. Effect of Order of Difference on Memory

Note that the original data can be extracted using the differenced data along with the initial values. For first order differences, this integration operation can be expressed as

$$q(k) = q(k-1) + \nabla q(k) \tag{2}$$

When k = 1, an initial value q(0) (along with ∇q(1)) is needed to calculate q(1). Note that if the order of difference is greater than one, the number of necessary initial values increases correspondingly.

Data compression is regarded as the crucial component of efficient data transfer and storage. Two types of compression exist in the literature, lossless and lossy. In lossless data compression, original data is encoded to a smaller body of data which can later be decoded back to the original data, whereas in lossy compression, original data can only be approximated after decompression. Details of the proposed compression method, which is a lossless data compression technique, and its hardware implementation are presented in the following sub-sections.

A. Proposed Method

Most of the compression techniques in the literature are generic methods which can be applied to compress any sort of data including command sequences, whereas the proposed method is only applicable to integer command sequences that have distinctive acceleration / deceleration characteristics. Encoding and decoding algorithms of the first proposed method are explained in the following sub-sections.

1) Encoding Process: During encoding, the difference of the command sequence is calculated based on the specified order. The resulting data is compressed utilizing suggested new (range based) compression algorithm. A sample encoding process for third order difference is given in Fig. 2. Note that the compressed code basically consists of three fields (Sign, Amplitude, and Length Fields) and initial values. The length of the sign field is equal to the number of data values in the differenced set and it consists of bits that show if the number in the differenced data sequence is positive or negative. 0 represents positive and zero values and 1 stands for negative values. In the amplitude field, absolute values (in binary) of the numbers in the differentiated set are stored successively. In order to recover the original data, another field that gives the length of each value in the amplitude field is formed, namely the length field. It contains sequences of 1's and 0's in an alternating manner as can be seen from Fig. 2. Lastly, the initial values should also be stored to extract the original command sequence.

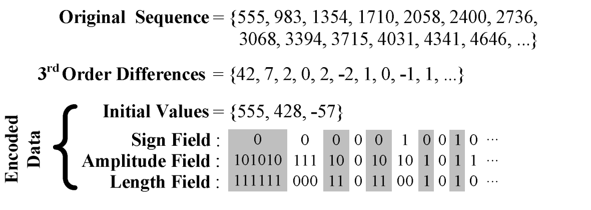

Original Sequence = {555, 983, 1354, 1710, 2058, 2400, 2736, 3068, 3394, 3715, 4031, 4341, 4646, ...}
3rd Order Differences = {42, 7, 2, 0, 2, -2, 1, 0, -1, 1, ...}
Encoded Data
Initial Values = {555, 428, -57}
Sign Field : 0 0 0 0 0 1 0 0 1 0 ···
Amplitude Field : 101010 111 10 0 10 10 1 0 1 1 ···
Length Field : 111111 000 11 0 11 00 1 0 1 0 ···

Fig. 2. Encoding Process of the Proposed Method

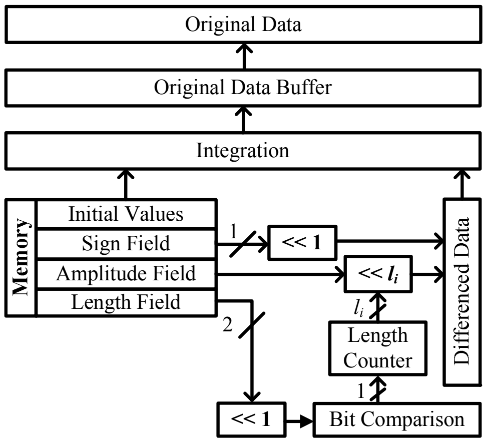


Fig. 3. Decompressor Architecture

2) Decoding Process: Architecture illustrating the decoding process is shown in Fig. 3. Decoding starts out with the comparison of the consecutive two bits of the length field. After the binary length of a data point is determined by the bit comparison module and the counter, the value is passed onto the left shift register. Then l_i many bits from the amplitude field are shifted into the differenced data module. After taking the sign value utilizing the left shift register from the sign field, differenced data module completes its task and sends the data to the integration module. Thus, the initial values are transferred to the integration module in the proper order and original data is generated from the buffer.

B. FPGA Implementation of the Proposed Method

Decompressor architecture for the proposed method, shown in Fig. 3, is realized utilizing Altera DE1 Development Board (with Cyclone II FPGA) [19]. Rather than writing directly in hardware description language, architecture is implemented in NIOS II Embedded Development Environment [20], where a soft-core processor is developed on FPGA is used as a microcontroller. Table I represents the allocated resources in FPGA for the implementation of NIOS II processor kernel.

TABLE I
FPGA RESOURCES USED FOR THE SOFT-CORE PROCESSOR

Total Logic Elements	3095 (17%)
Total Combinatorial Functions	2722 (15%)
Dedicated Logic Registers	1687 (9%)
Total Memory Bits	28992 (12%)
Program Size in SDRAM Memory (8 MB)	56 kB (0.006%)

With the help of the performance counter module of the soft-core processor, the time needed for decoding a sequence of 600 data points is roughly 25 ms.

IV. COMMAND GENERATION VIA POLYNOMIAL APPROXIMATION AND ERROR COMPRESSION

In the second method, the segmented command trajectory is to be modeled via polynomial based techniques. Since the considered techniques only approximate a given segment (to the desired accuracy), the exact representation of the original sequence is not possible by definition. To circumvent this limitation, the modeling errors can be stored and added to the generated sequence.

A. Proposed Method

Encoding and decoding algorithms of the second proposed command generation method are explained in the following sub-sections.

1) Encoding Process: During encoding, firstly the corresponding command trajectory is divided into segments from its inflection points. Then a polynomial based approximation method (like Chebyshev) is applied on all the segments. Since an approximation is carried out, errors are inevitable. The resulting errors could be compressed using the proposed compression algorithm described in the previous section for the purpose of generating the original command trajectory.

2) Decoding Process: Architecture illustrating the decoding process is provided in Fig. 4.

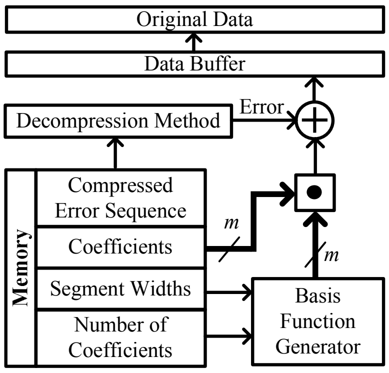


Fig. 4. Decompressor Architecture

Decoding starts out with the generation of necessary Chebyshev polynomials using the width of the corresponding segment and the order of Chebyshev polynomials. The dot (inner) product of coefficient vector and basis functions (polynomials) is performed in synch with the decoding of the compressed error data. In the figure, *m* represents the number of polynomials used in the corresponding segment. After the summation of decompressed error data and approximated data, original data sequence is generated.

B. FPGA Implementation

The described decoding algorithm is again implemented on FPGA using the NIOS II Embedded Development Environment. The only difference of this implementation from the one described in the previous section is the size of SDRAM Memory, which is 61 kB (0.007%). The time necessary for generating a command sequence of 600 points is around 247 ms, which is much higher than the first method. It is critical to note that the basis functions (i.e. Chebyshev polynomials) can be generated in advance and it takes roughly 213 ms.

V. Evaluation of Proposed Methods

In this section, the proposed methods are evaluated using MATLAB, which includes special functions that are not directly implemented on the FPGA used in this study. It is critical to note that whenever applicable, there is no observable difference between the command sequences generated by MATLAB and FPGA.

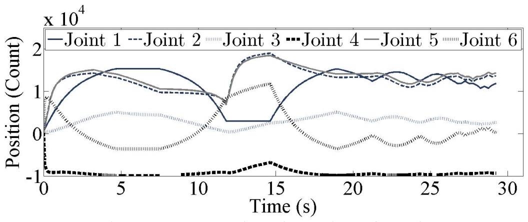


Fig. 5. Command Trajectories of 6 Joints

These proposed command generation methods in the previous two sections are applied on the command sequences generated for all six joints of a PUMA 560 manipulator. Fig. 5 shows these trajectories in encoder counts. It is assumed that the quadrature encoder at each joint has the ability to generate 40000 (= 4 × 10000 pulse/rev) pulses in one revolution. The results of the implemented methods are discussed in the following sub-sections.

A. Differencing and Compression

Huffman, Arithmetic Coding, and the proposed compression algorithms are applied on to the trajectories of the PUMA 560 manipulator. After the finite differences of the command trajectory of the first joint are taken for various orders, data compression algorithms are employed to compress the differenced trajectory data. Resulting compression ratios in percentage are presented in Table II. In the table, *n* represents the order of finite difference. Note that while calculating the compression ratios for the methods utilizing differencing, the memory required for the initial values is also taken into consideration.

It is inferred from Table II that if differencing of order higher than one is applied, compression algorithms give much better results. This is due to the fact that the encoder pulses in the command sequence are very different from each other when higher order differences are not taken. Another conclusion drawn from the table is that the performance of the proposed compression algorithm is always better than the other compression methods. With the third order difference, command sequence can be compressed to one-third of its memory size. Results given in Table II are also in accordance with Fig. 1. Till the third order difference compression ratio decreases and after that it has an increasing trend. This situation may be explained by the decline in the frequencies of the numbers in the sequence and the expansion in the range of the data values.

TABLE II
Compression Ratios for Different Orders

	Compression Ratio (%)		
n	**Huffman**	**Arithmetic Coding**	**Proposed Method**
0	197.1	181.4	177.0
1	131.0	118.6	78.9
2	34.1	32.7	34.0
3	34.8	31.9	27.8
4	46.4	42.3	34.5
5	60.6	54.1	42.5
6	81.7	70.9	52.1

In calculation of the compression ratios of the algorithms, all necessary values including compressed code, initial values and dictionary tables are taken into account. The following expression is used to calculate the compression ratio of the proposed technique:

$$r=\frac{ceil\left(\frac{1}{8}\left(\sum_{i=1}^{N-n}2l_i+N-n\right)\right)+ceil\left(\frac{n}{8}fix\left(\frac{\log(d_{\max}-d_{\min})}{\log(2)}+1\right)\right)}{ceil\left(\frac{N}{8}fix\left(\frac{\log(d_{\max}-d_{\min})}{\log(2)}+1\right)\right)} \quad (3)$$

In this equation, N is the length of the original data sequence, n is the order of finite difference, d_{max} and d_{min} represents the maximum and the minimum value of the original data sequence respectively.

To be able to determine which compression algorithm is suitable for command sequences, discussed data compression methods are applied on all trajectories of PUMA 560 manipulator after taking third order finite differences of the trajectories. The results are shown in Table III.

TABLE III
COMPRESSION RATIOS FOR THIRD ORDER DIFFERENCES (%)

		Joint Number					
		1	2	3	4	5	6
Method	Huffman	34.8	43.9	36.0	37.0	46.0	41.3
	Arithmetic	31.9	40.2	33.8	34.8	42.0	37.9
	Proposed	29.7	29.8	30.5	28.2	30.3	31.3

It is clearly inferred from the table that the proposed compression method is always better than Huffman and Arithmetic Coding.

B. Segmentation, Approximation, and Compression

Chebyshev, Legendre, and Bernstein – Bezier polynomials based approximation methods are applied on the command trajectory of the first joint of PUMA 560 manipulator. Plot showing the effect of error root mean square (RMS) value on the compression ratio is provided in Fig. 6. These methods cannot approximate the trajectory below certain error values, since singularity problems occur in the Pseudo-inverse technique. Even though when there is no singularity problem, the performance of the approximation methods cannot be regarded as good when the compression ratios are considered. Since the performance of the approximations is not acceptable, trajectories are divided into sections from their inflection points and then approximation paradigms are applied according to the proposed method.

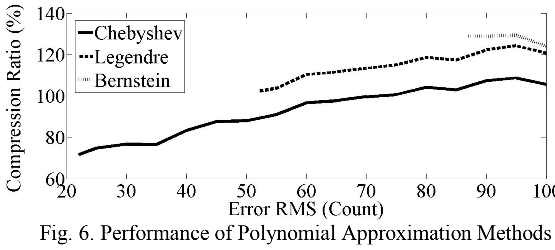


Fig. 6. Performance of Polynomial Approximation Methods

Errors resulting from the approximations are also compressed in a similar way explained in the section where the first method is described. Hence, with this approach command sequences are to be formed without any error. The method is applied on the first trajectory of the manipulator and the plot demonstrating the results is given in Fig. 7. In the method, the most important parameter is the error RMS value. If it is selected to be small, the error in the approximation decreases but the number of polynomials and coefficients increase tremendously. Therefore, a proper RMS value should be selected minimum compression ratio.

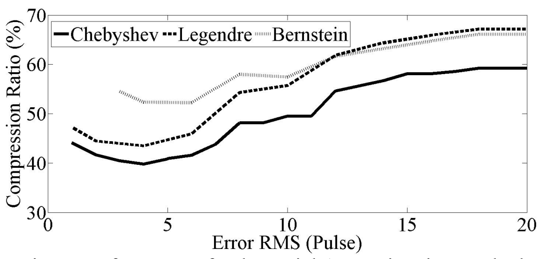


Fig. 7. Performance of Polynomial Approximation Methods

As it is inferred from Fig. 7 that the best performance is achieved for all approximation methods when the error RMS value is 4. Relatively high compression ratios at the beginning of the plot are due to the number of polynomials used to achieve corresponding RMS value. After the value of 4, compression ratio tends to rise for all polynomial types since the memory required for storing the errors increases. Lastly, when the three approximation methods are considered, Chebyshev polynomials outperform the Legendre and Berstein – Bezier polynomials for approximation.

In the previous two sections, different methods for generating command sequences from compressed data were proposed. Among data compression methods, the proposed technique was apparently better than Huffman and Arithmetic Coding methods. On the other hand, for the second method; Chebyshev polynomials based approximation turned out to be the best among other polynomials. These two superior methods are applied on all joint trajectories of the PUMA 560 manipulator shown in Fig. 5 and the results are given in Table IV. The proposed compression method is employed after taking third order difference of the trajectories and Chebyshev approximation technique is applied after segmentation with respect to the inflection points. In approximation, error RMS value is taken as 4 in order to obtain minimum compression ratios.

TABLE IV
COMPARISON OF PROPOSED TECHNIQUES IN TERMS OF COMPRESSION RATIO (%)

		Proposed Compression Method	Chebyshev Approximation Method
Joint Number	1	29.7	39.8
	2	29.8	49.5
	3	30.5	39.4
	4	28.2	39.7
	5	30.3	49.7
	6	31.3	42.4

Considering the results in Table IV, the performance of the two methods generating the command sequences without any error are relatively comparable. For all the joint trajectories, proposed compression method gives better results than Chebyshev approximation method. As a conclusion, in order to generate any trajectory with less memory and computational effort, the proposed compression method is more advantageous than the other techniques. Another advantage of the proposed data compression method is that while generating position sequences, velocity and acceleration profiles can also be generated without any extra computational effort as a result of previous differencing.

VI. Conclusions

In the paper, methods that are able to generate command sequences using less memory resources are presented. Basically, data compression and function approximation techniques are used to represent the command trajectories. With these proposed methods, original data sequence can be compressed to approximately one-third of its initial memory size.

When the methods utilizing data compression techniques are considered, taking higher order differences of the trajectory before compression is necessary. After differencing, the frequency of numbers in the sequence increases, (entropy based) data compression makes sense. Note that the novel compression method suggested in paper is not a universal technique. Its advantages reveal when the command sequence consists of integers showing acceleration and deceleration characteristics. As a future work, the hardware implementation of the novel method is planned to be implemented using a hardware description language rather than NIOS II.

In the methods where function approximation paradigms are used, it is impossible to generate the trajectory without any error. Thus, errors resulting from approximation should be stored for perfect recovery. Before storing, the proposed compression method is employed on the error sequence to decrease the memory requirement. In consequence of segmenting the trajectory before approximation, errors decrease in a manner that the novel compression method is more applicable.

Even though the compression ratios of approximation methods are greater than data compression methods, in the future approximation techniques are to be used to generate command sequences with variable speed.

References

[1] H. K. Reghbati, “Special Feature An Overview of Data Compression Techniques,” *Computer*, vol. 14 no.4 pp. 71-75, 1981.

[2] D. A. Huffman, “A method for the construction of minimum redundancy codes,” *Proceedings of IRE*, vol. 40, no. 10, pp. 1098-1101, 1952.

[3] I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmetic coding for data compression,” *Communications of the ACM*, vol. 30, pp. 520-540, 1987.

[4] W. J. Cody, “A survey of practical rational and polynomial approximation of functions,” *SIAM Rev.,* vol. 12, no. 3, pp. 400–423, 1970.

[5] K. Dickson, Cisco IOS Data Compression, Cisco Syst., San Jose, CA, 2000.

[6] S. Rigler, W. Bishop and A. Kennings, "FPGA-Based Lossless Data Compression using Huffman and LZ77 Algorithms," *Canadian Conference on Electrical and Computer Engineering*, pp.1235-1238, 2007.

[7] T. M. U. De Araujo, E. R. Pinto, J. A. G. De Lima, and L. V. Batista, “An FPGA Implementation of a Microprogrammable Controller to Perform Lossless Data Compression Based on the Huffman Algorithm,” *13th IBERCHIP Workshop,* 2007.

[8] M. A. Abd El ghany, A. E. Salama and A. H. Khalil, “Design and Implementation of FPGA-based Systolic Array for LZ Data Compression,” *IEEE International Symposium on Circuits and Systems*, pp.3691-3695, 2007.

[9] W. Cui, “New LZW Data Compression Algorithm and Its FPGA Implementation,” *Picture Coding Symposium 2007,* Lisbon (Portugal), Nov. 2007.

[10] G. H. H’ng, M. F. M. Salleh and Z. A. Halim, “Golomb Coding Implementation in FPGA,” *Elektrika Journal of Electrical Engineering*, pp. 36-40, 2008.

[11] Y. Yongming, L. Jungang, and W. Jianmin, “LADT Arithmetic Improved and Hardware Implemented for FPGA - Based ECG Data Compression,” *2nd IEEE Conference on Industrial Electronics and Applications, ICIEA 2007*, May 2007, pp.2230-2234.

[12] D. Valencia and A. Plaza, “FPGA-Based Hyperspectral Data Compression Using Spectral Unmixing and the Pixel Purity Index Algorithm,” *Computational Science,* pp.881-891, 2006.

[13] D. Lee, W. Luk, J. Villasenor, and P. Cheung, “Hierarchical Segmentation Schemes for Function Evaluation,” *Proc. IEEE Int. Conf. on Field-Programmable Technology*, pp. 92-99, 2003.

[14] D. Lee, R. C. C. Cheung, W. Luk, J. D. Villasenor, “Hardware Implementation Trade-Offs of Polynomial Approximations and Interpolations,” *IEEE Transactions on Computers*, vol.57, no.5, pp.686-701, May 2008.

[15] R. Michard, A. Tisserand, and N. Veyrat-Charvillon, “Small FPGA Polynomial Approximations with 3-bit Coefficients and Low-Precision Estimations of the Powers of X,” *Proc. 16t h IEEE Int. Conf. On Application-Specific Systems,Architecture and Processors,* pp.334-339, 2005.

[16] A. Ashrafi, R. Adhami, L. Joiner, and P. Kaveh, “Arbitrary Waveform DDFS utilizing Chebyshev Polynomials Interpolation,” *IEEE Transactions on Circuits and Systems I: Regular Paper,* vol. 51, no. 8, pp. 1468-1475, 2004.

[17] A. M. Sodagar, and G. R. Lahiji, “A Pipeline ROM-Less Architecture for Sine-Output Direct Digital Frequency Synthesizers Using the Second-Order Parabolic Approximation,” *IEEE Transactions on Circuits and Systems. II,* vol. 48, no. 9, pp. 850-857, 2001.

[18] A. Ashrafi, Z. Pan, R. Adhami, and B. E. Wells, “A Novel ROM-less Direct Digital Frequency Synthesizer Based on Chebyshev Polynomial Interpolation,” *Proc. of the 36th Symposium on System Theory,* pp. 393-397, 2004.

[19] http://www.terasic.com.tw/attachment/archive/83/DE1_UserManual_v1018.pdf

[20] http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

REAL-TIME MOTION CONTROL USING FIELD PROGRAMMABLE GATE ARRAYS

BARIŞ RAGIP MUTLU

JUNE 2010

REAL-TIME MOTION CONTROL USING FIELD PROGRAMMABLE GATE ARRAYS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BARIŞ RAGIP MUTLU

IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

JUNE 2010

Approval of the thesis:

REAL-TIME MOTION CONTROL USING FIELD PROGRAMMABLE GATE ARRAYS

submitted by **BARIŞ RAGIP MUTLU** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen ________________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Suha Oral ________________
Head of Department, **Mechanical Engineering**

Assist. Prof. Dr. Melik Dölen ________________
Supervisor, **Mechanical Engineering Dept., METU**

Assist. Prof. Dr. A. Buğra Koku ________________
Co-Supervisor, **Mechanical Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Mehmet Çalışkan ________________
Mechanical Engineering Dept., METU

Assist. Prof. Dr. Melik Dölen ________________
Mechanical Engineering Dept., METU

Assist. Prof. Dr. A. Buğra Koku ________________
Mechanical Engineering Dept., METU

Assist. Prof. Dr. E. İlhan Konukseven ________________
Mechanical Engineering Dept., METU

Assoc. Prof. Dr. Veysel Gazi ________________
Electrical and Electronics Engineering Dept., TOBB-ETU

Date: 22.06.2010

ABSTRACT

REAL-TIME MOTION CONTROL USING FIELD PROGRAMMABLE GATE ARRAYS

Mutlu, Barış Ragıp
M.Sc., Department of Mechanical Engineering
Supervisor: Assist. Prof. Dr. Melik Dölen
Co-Supervisor: Assist. Prof. Dr. A. Buğra Koku

June 2010, 160 pages

In this thesis, novel implementation methods for FPGA based real-time motion control systems are investigated. These methods are examined for conventional and modern controller topologies as well as peripheral device interfaces which are mutually essential pieces of a motion controller. The developed methods are initially tested one by one to assess the performance of the individual design; and finally an assembled solution is developed to test the overall design. Tests of the overall design are realized via hardware-in-the-loop simulation of a real-world control problem, selected as a CNC machining center. The developed methods are discussed in terms of their success, resource consumptions and attainable sampling rates.

Keywords: FPGA, Motion control

ÖZ

ALAN PROGRAMLANABİLİR KAPI DİZİNİ KULLANILARAK
GERÇEK ZAMANLI HAREKET DENETİMİ

Mutlu, Barış Ragıp

Yüksek Lisans. Makina Mühendisliği Bölümü

Tez Yöneticisi: Yard. Doç. Dr. Melik Dölen

Ortak Tez Yöneticisi: Yard. Doç. Dr. A. Buğra Koku

Haziran 2010, 160 sayfa

Bu çalışmada, alan programlanabilir mantık kapısı dizini (Field programmable gate array - FPGA) tabanlı gerçek zamanlı hareket denetim sistemleri için yeni uygulama yöntemleri araştırılmıştır. Bu yöntemler, bütün bir hareket denetim sisteminin alt parçalarını oluşturan, geleneksel ve modern denetim topolojileri ile çevresel birim arayüzleri için incelenmiştir. Geliştirilen yöntemler başlangıçta tek tek sınanarak bireysel başarımları ölçülmüş, sonunda ise parçalar biraraya getirilerek oluşturulan tasarım sınanmıştır. Bütün tasarımın sınanımları, bir CNC işleme merkezinin çevrimiçi donanım benzetimi kullanılarak gerçekleştirilmiştir. Geliştirilen yöntemler, başarımları, kaynak tüketimler ve erişilebilir örnekleme hızları bazında tartışılmıştır.

Anahtar kelimeler: Alan programlanabilir kapı dizini, Hareket denetimi

To My Parents

DON’T PANIC

ACKNOWLEDGEMENTS

I would like to thank to my supervisor Assist. Prof. Dr. Melik Dölen and co-supervisor Assist. Prof. Dr. A. Buğra Koku for giving me this research opportunity. I have been privileged to have mentors with a great deal of knowledge and I sincerely appreciate their guidance and support during this study.

I would also like to thank to The Scientific & Technological Research Council of Turkey, for financial support under project contract 108E048.

I would like to thank to my parents, Selma and Mehmet Mutlu, for always supporting me with their love and care. It is a blessing to know that regardless of the path I choose to take, they will be there for me.

I would like to thank to my colleagues, Ulaş Yaman, Serdar Üşenmez and Rasim Aşkın Dilan, for all the support and fun they have provided during prolonged hours of study in the laboratory.

I would like to thank to my bıdık Müge, for her love, understanding and patience, especially during the final period of this thesis. I have been lucky to have her by my side.

Finally, I would like to thank to Tuna Soncul, Can Özer, Sinan Değer, Can Özgiresun and all the others who cared enough for me to make this path significantly harder. If it wasn't for them, I would probably have ended up in the same place, but for all the wrong reasons.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

In general terms, motion control can be described as the position and/or velocity control of a machine part such as the end effector of a robotic arm, cutting tool of a CNC machine, tip of an automated welding machine and etc. Evidently, motion control plays a key role in industrial automation systems and success of the motion control system directly affects the production speed, quality and quantity. Considering that industrial automation systems are used globally in production and assembly lines of many industrial plants including automotive, aviation, packaging, printing, textile and semiconductor production industries, better, faster and more robust motion control systems are always desirable. Schematic of a typical motion control system is shown in Figure 1.1.

Motion control systems are being developed by both academia and private entities, in correlation with the technological advances in semiconductor technology. As new semiconductor devices with faster processing speeds and higher resources are available in the market, new motion controllers are developed by these entities that fully utilize the features of these novel chips.

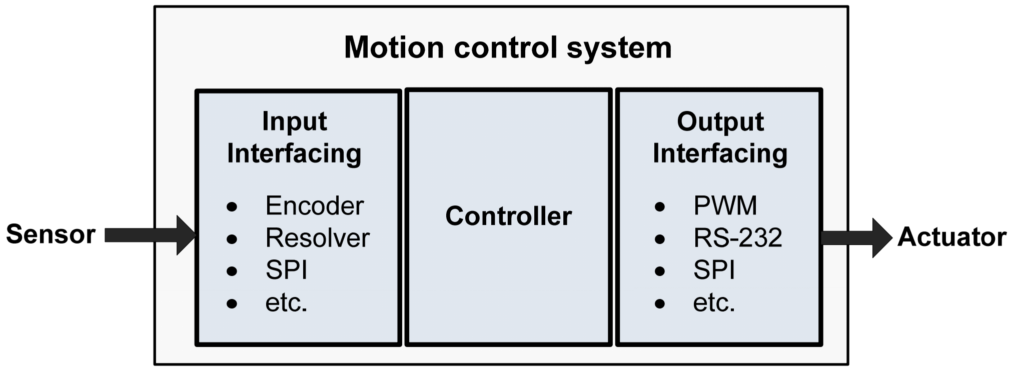


Figure 1.1 – Schematic of a typical motion controller

In the currently available motion control systems, the most commonly employed chip is the digital signal processors (DSPs) which appear in almost all of the recent products developed by both academia and private manufacturers, including the most recent products of the two major motion controller manufacturers, including the DMC-18x6 by Galil and PMAC2 by Delta-Tau. The situation is similar in the academic literature and many studies published in the last decade employ DSPs in motion control systems, especially if the emphasis of the study is on the controller's theory rather than the implementation.

On the other hand, FPGAs are also suitable options for real-time motion control applications, exhibiting some superior qualities over traditional processors (micro-processors, micro-controllers and DSPs) such as parallel processing capability, high sampling rates, flexibility in design, and reliability. Therefore, in the academic end, field programmable gate arrays (FPGAs) also start to take significant parts in motion controller designs. As further discussed in the literature survey chapter, there are many recent studies that employ FPGAs in motion controller designs and obtain very successful results. Especially after the FPGAs' resources have increased significantly in the first half of this decade, there has been a considerable increase in these studies that utilize an FPGA in the motion control system. Figure 1.2 shows the increase in resources (in terms of logic cells/elements) of FPGAs that are produced by two major FPGA manufacturers: Xilinx and Altera in the last 15 years.

As can be seen in Figure 1.2, resources of the FPGAs have increased approximately 700% in the last decade, which resulted in the academia's increased level of interest in FPGAs for motion control applications.

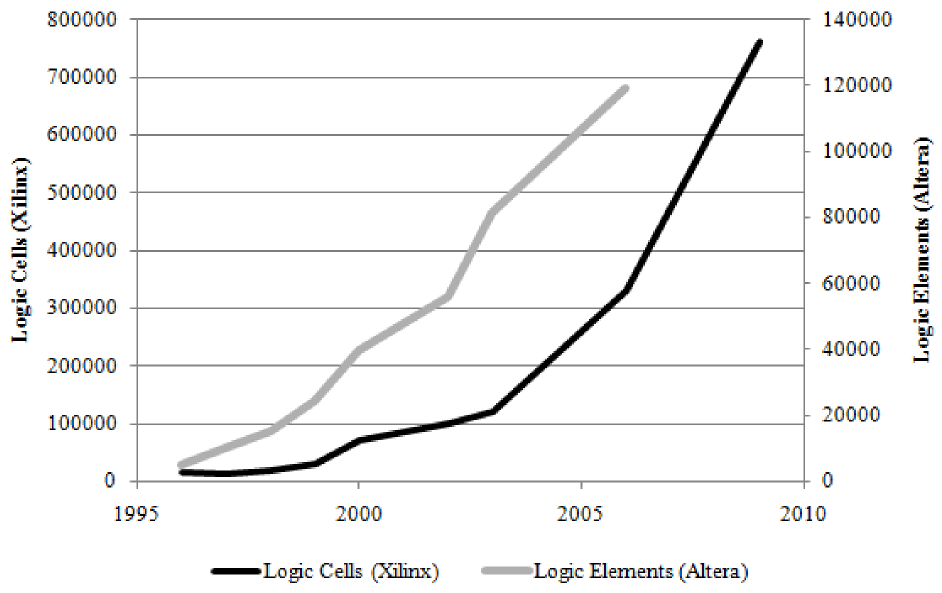


Figure 1.2 – Increase in number of logic cells of Xilinx and Altera FPGA chips over years

It can be estimated that this interest will increase or at least stay at this level, as FPGA manufacturers continue to increase the capabilities of their FPGA chips and develop new and more advanced intellectual properties (softcore processors, SoPC tools, etc.) for embedded designers. Therefore, developing methods for an FPGA based motion control system is desirable, as the FPGAs’ role in motion control applications increases.

1.2 Scope of the thesis

As presented in the previous section, the main objective of this study is to develop an FPGA based motion controller and by FPGA based motion controller, a “single chip” solution that handles all the necessary tasks of a typical motion control system is implied. Requirements of a typical motion control system are well-known and the developed system must be able to:

1. Provide a convenient interface for obtaining sensor data from common sensors that are employed in motion control applications and interpret this data as required.
2. Obtain reference inputs from an outside source via a suitable interface and/or a communication protocol or generate the reference commands for the controller itself.
3. Execute a controller algorithm that is able to provide the desired performance that is comparable to the modern motion controllers in the literature.
4. Provide means for transmitting the manipulated input computed by the controller via a suitable interface or a communication protocol to a driver, servo-amplifier or the actuator itself.

Evidently, there are many device/design choices to be made for each item and it is not possible to cover every aspect of FPGA based motion controller design in a single study. However, it is possible to develop methods for the most common choices in the academic studies and industrial applications and if possible extrapolate the results to others. For instance, in Chapter 3, a design method for an incremental encoder interface is provided, leaving out the absolute encoders. However, even though the interface required by an absolute encoder would be different, the obtained result that is hardwired implementation by HDL design is more efficient for the encoder interface is valid for both of the sensor devices. Therefore, most of the results are applicable to similar devices or design choices.

1.3 Organization

This thesis is divided into 8 chapters. The second chapter provides a review on the studies relevant to the motion control applications, in a limited context of FPGA employing designs. In the third chapter, peripheral interfaces that are developed and implemented on the motion controller design are introduced. In the fourth chapter, design and implementation of a state-space controller with a Luenberger-type observer on an FPGA is presented. In the fifth chapter, digital filters are introduced and their implementation methods on FPGAs are discussed. In the sixth chapter, a test case is provided where the developed FPGA modules are tested on a hardware-in-the-loop simulation platform. In the seventh chapter, a preliminary study on advanced controllers is presented. In the last chapter, conclusions are drawn and future work is discussed.

CHAPTER 2

LITERATURE SURVEY

Since the last decade, FPGAs are becoming gradually dominant in control applications. In numerous studies, implementation methods for embedded controller design and application results of different topologies on FPGAs are presented. In this chapter, a general overview of the relevant technical literature is provided.

Due to their flexible hardware, uses of FPGAs in control literature differ significantly among different design topologies. For instance, while there are implementations of an FPGA as an interface between the controller chip and peripheral devices in some designs, there are also architectures in which a complex motion controller is realized by only utilizing a single FPGA chip. Therefore, it is necessary to classify FPGA employing controller designs into two categories as: "Hybrid control systems" and "FPGA-based control systems".

In hybrid control systems, FPGA may be employed in two different schemes. In the first scheme, FPGA is used for peripheral device interfacing and it has minimal amount of computational tasks. On the other hand, in the second scheme, FPGA is utilized as a secondary processor, and a noteworthy computational load is handled by the FPGA, in addition to its interfacing tasks. Nevertheless, it is obvious that in these hybrid designs there needs to be a central processor employed in the design, which is usually selected as a digital signal processor (DSP) or a microprocessor.

In a generic scheme of a hybrid design, peripheral device interfacing (sensor interfacing, pulse-width modulation generation, SPI and etc.) is a task assigned to the FPGA. This is due to the fact that FPGA implementation of those modules is fast, relatively simple and requires a small amount of resources. On the other hand, complex controller algorithms such as intelligent (fuzzy, neural-network) or nonlinear (sliding mode, variable structure and etc.) controllers are generally implemented on the processor, as well as the memory management units (SRAM, EEPROM and etc.) and communication controllers (UART, JTAG and etc.). It can be seen that, as the complexity of the task increases and the speed requirement decreases, the load shifts from the hardwired FPGA implementation to the processor, as shown in Figure 2.1.

As can be seen from Figure 2.1, the conventional controllers find their place in between this speed/complexity distinction, which means that there are designs that both include or exclude a conventional controller unit on the FPGA and both options are equally feasible. Hence, these hybrid control systems are also classified according to the existence of a controller load on the FPGA as it is further discussed in Section 2.1.

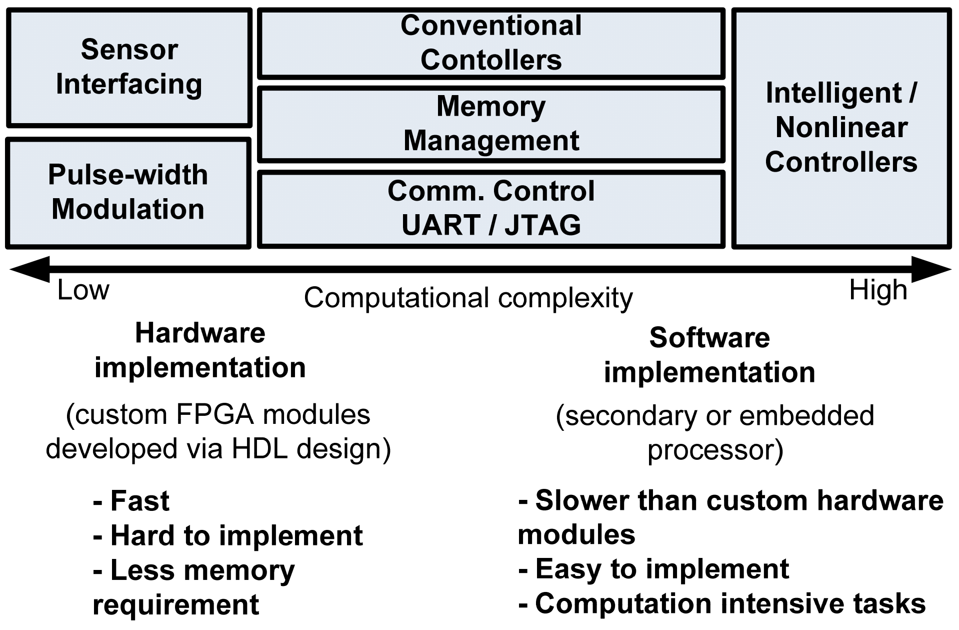

Sensor Interfacing
Pulse-width Modulation
Conventional Contollers
Memory Management
Comm. Control UART / JTAG
Intelligent / Nonlinear Controllers
Low
Computational complexity
High
Hardware implementation
(custom FPGA modules developed via HDL design)
- Fast
- Hard to implement
- Less memory requirement
Software implementation
(secondary or embedded processor)
- Slower than custom hardware modules
- Easy to implement
- Computation intensive tasks

Figure 2.1 – Generic schematic for FPGA implementation of a motion controller

As a final remark for hybrid designs, the presented scheme is not a universal approach and while it is a common approach to use FPGA for interfacing and implementation of simpler controllers; complex controller algorithms, memory management units and communication controllers are also applicable by FPGA implementation and in some designs there are topologies that do not correspond to the generic scheme provided in Figure 2.1.

In the FPGA-based designs, there exist a single FPGA chip that is responsible for the main computational task as well as the interfacing, memory management and communication functions. However though, a single chip implementation does not mean that every task is handled by a custom hardwired FPGA module. Since there are many embedded processor IPs developed by FPGA manufacturers, it is possible to employ an embedded processor in an FPGA based design. If an embedded processor is employed on an FPGA-based design, it can be seen that the embedded processor implementation along with the custom hardware modules is very similar to the hybrid design approach. The difference would be that the connecting wires between the custom modules and the processor are external on the hybrid design and processor's resources don't consume FPGA's resources. Therefore, the generic scheme provided in Figure 2.1 applies with a slight difference on the employed processor.

Nevertheless, it is also possible that the overall motion control system is implemented on the FPGA chip with custom hardwired modules. It is a significantly different approach than the processor-based designs; therefore it is convenient to classify the FPGA-based designs into two by the utilization of an embedded processor.

In FPGAs, there are different ways that computations can be handled. In hardwired implementation via HDL design, it is possible to include or develop custom arithmetic module IPs for fixed-point arithmetic. Furthermore, it is also feasible to insert a floating point unit (FPU) in the design to perform floating point calculations, such as the one devised by R. Usselmann [1] [1] as an open-core IP. Nevertheless, it should be noted that the development platform of hardware description languages such as Verilog HDL or VHDL does not support fixed/floating point number representations (and accompanying calculations) directly and therefore the design paradigm is long and tedious. On the other hand, in the embedded processor development environment, it is possible to use different number types conveniently by coding through a high-level language like C.

A similar problem occurs when implementing a memory management unit or a communication controller on the FPGA as a hardwired module. In this hardware solution, it is the designer's job to arrange and synchronize the clock signals, read/clear buffers etc. On the other hand, if a softcore processor is employed, it provides a system on programmable chip (SoPC) solution for these tasks and it is significantly easier to implement an SRAM controller or a UART controller when an embedded processor is employed. As a matter of fact, the designer does not interact at all with the memory management unit or the UART controller, after including those units to the SoPC design interface and connecting them to the selected embedded processor.

2.1 Hybrid control systems

In this section, designs are discussed in which FPGA is either employed as a sole interface between a main processor and peripheral devices or utilized as a secondary processor in addition to its interfacing task.

2.1.1 Control systems utilizing FPGA as an interface

In many previous studies, FPGAs are generally considered as an interface between a processor unit that handles the controller (such as a DSP or a microprocessor) and its peripheral units. This is due to the fact that FPGA implementations of peripheral interfaces are particularly fast and efficient by design with hardware description languages (such as Verilog HDL and VHDL) on FPGAs. For instance, a PWM generator is a very common and necessary interface in motion controller designs and in a study conducted by Arbit et al. [2] it has been shown that different shapes of waveforms of PWM signals can be generated precisely on an Altera Spartan II FPGA and even simpler FPGAs could handle the task quite efficiently. Therefore, implementing an FPGA based input/output interface is a good option for motion controllers, as further discussed in Chapter 3. In this section, designs in which FPGA is employed as a peripheral device interface are presented.

In such configurations, FPGAs do not share the computational burden of the main processor. For instance, in the design proposed by Dong et al. [3], the control system consists of a TMS320LF2407 DSP of Texas Instruments accompanied by an EP1K30-144 FPGA of Altera. While this system is presented as a dual-core system, there is a significant difference between the computational loads of the cores. In this design, FPGA core deals with the external circuitry (that are FPGA modules) to process position encoder signals, some keyboard inputs and displays while also managing some I/O functions as well as a data bus to the DSP core. On the other hand, the DSP core handles all the computations of the PID controller. Therefore, although the system is presented as a dual core system, the primary computation load is on the DSP core.

A similar scheme appears in the study of Birou and Imecs [4] where the FPGA is coupled to a TMS32OC50 DSP of Texas Instruments. In this configuration, an FPGA module generates pulse width modulation

(PWM) signals for the power converters and another FPGA module decodes incremental encoder signals. Controller is again implemented on the digital signal processor (DSP) and FPGA has no computational load.

In another study conducted by Lee et al [5], a DSP 2812 controller board employing a TMS320F2812A DSP is utilized with a Altera Cyclone FPGA chip in order to realize a RBF neural network controller for nonlinear systems. Analogous to the previous studies, the scheme is topologically the same as the previous studies and the FPGA is used for the sole purpose of encoder interfacing/counting and PWM generation.

Another recently published study by Caporal and Pacas [6], presents implementation of a direct mean torque controller that is realized on a hybrid system consisting of an ADSP 21062 DSP by Analog Devices and a Xilinx XC4010E FPGA. Nevertheless, FPGAs duty for interfacing remains with no significant computational load.

On the other hand, the situation is analogous in industrial motion controllers developed by private manufacturers. For instance, "DMC-18x6" which is the most recent and highest performance motion controller of Galil [7], utilizes a Xilinx FPGA chip for interpreting the encoder signals, while PID compensation with velocity and acceleration feedforward is realized by a 32-bit Motorola processor. Likewise, another product "PMAC2" which is developed by Delta-Tau [8], employs an Altera FPGA chip as an encoder interface and the central processor is selected as a Freescale DSP563xx.

These recent studies show that even the capabilities of the FPGAs are significantly increased in the last two decades; they are still employed for simply interfacing purposes regardless to their increasing capacity and processing capabilities. Nevertheless, it is also important to note that even if FPGAs do not carry heavy computational loads in the mentioned designs, their role is crucial in demanding control applications where sensor signals need to be decoded in a fast and efficient manner and complex output signals need to be generated. Therefore in these examples, the fact that FPGAs do not have a computational burden does not imply that their role in the controller design is unimportant.

For instance, in the study conducted by Al-Ayasrah et al. [9], an N-Motor speed controller for brushless DC motors is proposed by the implementation of a ADSP-21992 DSP by Analog Devices that is employed with a Xilinx Virtex-E FPGA chip. The key feature of this design is the N-motor speed control, which is realizable by the PWM module implemented on the FPGA chip.

Another study conducted by Toh et al. [10] presents implementation of torque and flux controllers for direct torque control (DTC) of an induction machine. In this study, complex torque/flux estimations are computed and a P flux controller along with a PI torque controller are realized on a DS1102 DSP board from dSPACE, which is based on a TMS320C31 DSP. However the success of the control algorithm depends on the high speed generation of the output signals and therefore it is vital that a faster hardware circuit handles that operation, which is realized on an Altera EPF10K20 FPGA device.

2.1.2 Control systems utilizing FPGA as a secondary processor

As evidently shown by the presented studies in the previous section, FPGAs are widely used for interfacing purposes in controller designs coupled with a digital signal processor, having none or very limited computation loads. However, there are also other hybrid control systems in which FPGAs share more amount of computational load than just an interfacing chip; deserving its role as a secondary processor.

In the design proposed by Jung and Kim [11], Texas Instrument's TMS320C6711 DSP is employed with Altera's EP20K300EQC240 FPGA chip, similar to the schemes in previous section. The controller topology is composed of a neural network controller and a PID controller; however in this case the PID controller is implemented on the FPGA; therefore contrary to the previous designs, the computation load of the FPGA is increased.

A study conducted by Yu et al. [12] proposes a novel multi-redundancy electro-mechanical actuator (EMA) controller, implemented on a TMS320VC33 DSP by Texas Instruments and Cyclone II FPGA by Altera. In this design, DSPs functions include receiving instructions of position, and carrying out calculations of the position loop and velocity loop; whereas the FPGAs functions are to provide interface for position signals, velocity signals and current signals, to carry out the calculation of the current loop, and to generate the PWM signal that is required by the multi-redundancy EMA. As can be seen, in this configuration FPGA handles the current controller as well as its typical interfacing tasks, earning its title as a secondary processor.

It is obvious that in these hybrid designs, FPGAs computation load as a secondary processor vary significantly from design to design. For instance, in a novel AC servo system implementation proposed by Esmaeli et al. [13], a Spartan FPGA of the Xilinx is employed with a TMS230LF2407 DSP of Texas Instruments. In the system, DSP chip implemented a position loop control which includes a PI regulator, velocity and acceleration feedforward and a digital notch filter. On the other hand, FPGA chip implemented the speed loop and current loop control, which also includes a PI regulator as well as a vector transformation module. This design can be considered as a truly hybrid design, because of the fact that the computational loads of both chips are equivalent and demanding.

As demonstrated by the provided examples, FPGAs can be utilized in many different controller topologies along with another processor, to achieve tasks at a range of complexities from relatively simple to utterly complex. However, there are also designs that employ a single FPGA chip for the overall motion controller design, and they are presented in the following section.

2.2 FPGA-based control systems

Recent research studies tend to shift the computational load from the primary processor to the FPGA, as the capabilities of FPGAs significantly increased in time over the last two decades. Many examples of hybrid control systems are provided in the previous section, however in those studies the FPGA has always been accompanied by a primary processor, which has mostly been a digital signal processor or a microprocessor.

Nevertheless, it is also possible to realize a control system on a single FPGA chip, and in this section, designs in which FPGA is solely employed to realize the entire motion control system are presented.

2.2.1 Control systems using softcore processors

FPGAs require low-level (i.e. logic-level) circuit design through hardware description languages (HDL) and such a design effort using HDLs is a relatively long and tedious process. To overcome this difficulty, FPGA manufacturers offer many intellectual properties (IPs) including embedded processor IPs. Therefore, many motion controller designs that facilitate a single FPGA makes good use of an embedded processor IPs.

While some designs use softcore processors only to implement controllers, the others utilize the embedded processor for complex calculations along with some "hardwired" custom modules for simpler tasks. As a matter of fact, utilizing an embedded processor does not necessarily mean that the whole implementation is realized on the processor. Therefore, it should be noted that while the control systems presented in this section are classified as systems with embedded processors, most of the designs in this section also utilize custom hardwired modules developed by HDL design tools. However, modules for complex arithmetic operations are not generally included since the processor easily handles those operations.

In a study conducted by Ni et al [14], an integrated motion control system is realized on an Altera Cyclone FPGA with a Nios II processor. This hardware design scheme realizes all sensor data acquisition, SPWM generation, motor vector control, torque control and trajectory generation in a single FPGA chip. Since a Nios II processor is employed, all the calculations are performed with floating point arithmetic.

In another study, Li et al [15] developed an FPGA-based servo controller for PMSM drives on an Altera Cyclone EP1C20 FPGA. In this design, while a custom HDL designed module performs the current/speed loop control for PMSM drives, which includes vector control strategy, the PI regulator, coordinate transformation and the SVPWM generator; Nios II processor performs the function of position control, based on the discrete-time sliding mode variable structure control. Similar to the previous study, the softcore processor allows floating point arithmetic for the variable structure controller.

In a recent study, Kung et al [16] realized a motion control IC for an X-Y table on a Altera Stratix II EP2S60F672C5ES FPGA chip. In this implementation, two axes' fuzzy position controllers and P speed controllers, as well as a motion trajectory generator are implemented by software using Nios II softcore processor. On the other hand, two axes' current vector controllers, along with the SVPWM generators and the encoder interfaces are realized by custom hardware (HDL design) modules. It is noted that the Nios II processor has less than 1 kHz sampling frequency, while the custom hardwired implementation works at 16 kHz. Therefore this study is a good example for demonstrating the better speed characteristic of the custom hardware.

Das and Banerjee [17] developed a digital PID controller for precision control of a brushed DC servo motor, employing a Spartan 3E FPGA chip and utilizing a PicoBlaze softcore processor by Xilinx. As expected, the PID control algorithm is realized on the softcore processor and the supporting interface modules such as: RS232 controller and the PWM generator is developed as custom hardware modules by VHDL. It is also noted that since the PWM generation is a time critical job, it cannot be handled by the processor along with the PID controller an therefore a custom module is developed for PWM generation task.

Salem et al. [18] proposed a servo drive system in which the peripheral interface and speed control is handled by hardware and position control along with the networking functions are handled by softcore processor, and implemented it on a Xilinx Virtex-II Pro XC2VP30 FPGA chip available on a Xilinx ML310 board. Furthermore, two different RT kernels, that are μC/OS-II and Xilkernel are investigated on the PowerPC 405 processor and their performances are tested on a test case of a PI controlled DC motor emulator.

These examples demonstrate that using an embedded processor IP within the FPGA is proven to be a successful approach. Furthermore, it is also shown that using an embedded processor with custom hardware modules further increases the efficiency and speed of the design, and it is proven that oftentimes the softcore processor needs to be accompanied by custom hardware modules. Unsurprisingly, this remark raises the issue of the performance of an FPGA-based design that is fully hardwired HDL design. The following section presents studies that do not employ an embedded processor and the whole motion controller design is based on custom hardware modules.

2.2.2 Hardware implementation of control systems

While the current tendency is to use an embedded processor for complex calculations required by controllers, it is also possible to eliminate the embedded processor and develop a total hardware solution. This is a desirable feature since the hardware modules are significantly faster than the embedded processors and oftentimes more efficient in terms of resources and power consumption rates. However, it is a long task to develop custom hardwired modules and furthermore, in order to benefit from the efficiency of the flexible hardware, modules need to be optimized for a specific task to fully observe the improvement obtained by the hardware implementation.

It is important to note that controller implementation on FPGA is often a trade-off between resource and execution time of the controllers. The reason for that is; while it is possible to benefit from the parallel processing capability of the FPGA chip by calling many instances of a certain module which would certainly require more resources, it is also possible to use certain modules repeatedly in a sequential manner to increase the time, rather than the resource cost. Therefore, it is possible to modify a design according to the resource/time requirements of the controller.

In a very recent study, Cho et al [19] have proposed an FPGA-based multiple axis motion control chip with no embedded processor employed. The chip has all the essential features such as velocity profile generation, interpolation, inverse kinematics calculation and a PID controller which are required to control a multiple axis motion control system such as a robotic manipulator. As discussed earlier, certain methods need to be developed in order to avoid complex calculations and in this study, they managed to avoid floating point calculations by multiplying coefficients by constant integers. Using no embedded processor; they attained lower resource costs and power consumption rates. The hardware modules are developed by VHDL and implemented on a Xilinx XC2V6000 FPGA.

Chan et al [20] have conducted a study on PID controller implementation on an FPGA and they managed to decrease the resources required by a multiplier-based design significantly on the target platform that is a Xilinx Spartan-II-E FPGA. They have proposed to replace the multipliers by a distributed arithmetic based design utilizing look up tables and they managed to decrease the resource requirement down to 4 to 13% of the former design. However they increased the computation time from 1 cycle to 13 to 26 cycles. The increase in efficiency of this design refers to the resource cost of the controller in terms of slices and power consumption. A very similar study by Tao et al [21] managed to decrease the logic element requirement of a PID-based CNC position controller from 51.7% to 0.8-1.5% in an Altera Cyclone II FPGA, increasing the computation cycle from 1 to 32-64 cycles, utilizing two different methods. The methods provided in these studies are a mere result of trying to reduce the resource requirements of the conventional PID controller and while these studies offer good improvements for a PID controller, there exist many controller algorithms including intelligent, nonlinear and hybrid topologies and there is no single way to implement each of them more efficiently on an FPGA. However, they are valuable to demonstrate that without the embedded processor IPs, it is still possible to implement a conventional controller as hardwired modules successfully via the HDL design tools on an FPGA.

There are also some other studies where more complex controllers are implemented on the FPGA via custom hardwired modules. Fuzzy controller designed by Lanping et al [22] requires no embedded processor or a floating point unit to perform fuzzy control. The selected platform for development is EPF10KlOLC84-15 of FLEXlOK series from Altera. On the other hand, the proposed method is similar to a rule based control topology and the method is not generally applicable for different fuzzy controller topologies. In another study by Kung et al. [23], an adaptive fuzzy controller for AC motor drive is proposed and the speed control IP is fully realized on the hardware platform of an Altera Cyclone EP1C20 FPGA. As a matter of fact, a Nios II processor is also included in this design for SRAM and UART control, however since the controller is fully implemented as hardwired, it is included as a hardware implementation. In this design, the current loop sampling frequency is 16 kHz, while the speed control loop's sampling frequency is 2 kHz.

An Elman neural network implementation is proposed by Lin et al [24] for a linear ultrasonic motor, where a fixed point arithmetic unit is implemented to perform the calculations. The proposed design is implemented on a Xilinx XC2V1000 FPGA chip and 723 Hz sampling frequency is attained for the controller. In the previous section, a study by Kung et al. [16] is provided for realization of a motion control IC for a X-Y table, in a more recent study by the same group [25], a self-tuning PID controller is realized using RBF neural network and is applied to the X-Y table. Different from the previous study, in this case the Nios II processor is only employed for trajectory generation purposes and the rest of the design is implemented as hardwired custom modules, including the neural network. The same chip that is Altera Stratix II EP2S60, is employed in the design and current loop can be closed at 16 kHz while the position loop's sampling frequency is 500 Hz. From these examples, it can be noticed, as the complexity of the controller increases, the attainable sampling frequency decreases.

These studies prove that it is possible to implement different controller topologies on FPGAs utilizing arithmetic logic units or custom hardware solutions. It is also seen that PID controllers are easier to implement by custom hardware logic and in practice usually realized by hardware modules. On the other hand it is also possible to implement more complex topologies by custom hardwired modules.

2.3 Closure

In this chapter, a literature review was provided for motion controller topologies that employ an FPGA chip. A classification was made according to the utilization of the FPGA in the design as a secondary chip or a single chip; and a further classification was performed according to the FPGA's duty in the design for the secondary chip case and utilization of an embedded processor in the FPGA for the single chip case. This classification is non trivial and different categorizations are also possible; nevertheless the studies in this review are provided according to this classification, in order to better demonstrate the differences between the design approaches.

In the provided studies, it has been shown that FPGAs can be successfully used in tasks within a wide complexity range, from primitive interfacing tasks to complex controller algorithms. It is shown that FPGAs can perform tasks utilizing custom hardware IPs and/or embedded processors. However, when an FPGA is accompanied by a secondary processor chip, it is seen that FPGA is mostly utilized by custom hardwired modules.

As a general rule, in almost all of the studies the peripheral interfacing has been implemented by hardwired FPGA modules, however, for implementation of control algorithms, it is shown that both custom FPGA modules and embedded processors are viable options. For the case of memory management and communication control, embedded processor IPs are generally selected for their simplicity in design; however it is shown that, custom modules are also available for these tasks.

In the presented studies, it was shown that the custom modules developed by HDL design tools have less processing time and resource requirement than the embedded processors. This is due to the fact that, custom modules are developed and optimized according to the controllers' needs. On the other hand, while some parameters and configuration of the embedded processor can be modified before implementation, it is still much less flexible than a custom hardwired module. In this study, an FPGA-based solution is proposed and evaluated.

CHAPTER 3

PERIPHERAL DEVICE INTERFACING FOR MOTION CONTROLLERS

A motion controller is always required to receive and transmit signals and data from/to other devices employed within the control system; therefore it is necessary for a motion controller design to include some means for peripheral device interfacing. For instance, motion controllers need to receive some sort of sensor information from a mechanical/optical/magnetic sensor and produce a corresponding manipulation signal that can be interpreted by a driver or can drive the actuator itself. Furthermore, other peripheral units (such as a host PC for setting the design parameters of the controller online) may also be included in the control system, which would require a communication protocol (such as RS-232) or a bus interface (such as SPI) in the controller architecture.

Evidently, there exist various interface topologies employed in sensors, actuators and other devices; and it is not practical to include numerous interfacing modules in a single design. Therefore, it is necessary to limit these modules by considering the most common interfaces used in common practice and if possible taking the currently employed devices into account.

In this chapter, a number of interfacing modules are presented, which are selected conveniently from the commonly used devices in motion control applications.

All the modules are developed using a hardware description language (mostly Verilog HDL), in order to increase the processing rate and decrease the resource cost of the module, benefiting from the flexible hardware of an FPGA.

On the other hand, it should also be noted that while implementing a softcore processor is also a viable option, for primitive designs such as sensor/actuator interfacing or communication protocol controller modules, it is a more efficient design approach to develop customized modules via HDL design. Further discussion on softcore processor implementation on FPGAs is available in the following chapters.

3.1 Encoder Interface

In motion control applications, the most commonly employed sensors to obtain positional feedback are encoders. Therefore an encoder decoding module is the most critical sensor interface of a motion control system. Encoders are electromechanical devices that generate an analog or digital (commonly digital) signal, in order to provide linear or rotational position feedback. Encoders that provide absolute position feedback are called "absolute position encoders" and their output signals require no interpretation; since they provide absolute position information in proportion to their resolution. However, due to their limited resolution, absolute encoders do not succeed in high speeds and in industrial control applications, incremental encoders are generally preferred.

Incremental encoders provide digital signals as square wave forms from two channels, with a phase difference of 90°. Therefore, it is possible to count every rising/falling edge of channels A and B (which is called quadrature decoding method), and increase the resolution of the encoder to 4 times its initial resolution. Ideal output of an incremental encoder is shown in Figure 3.1.

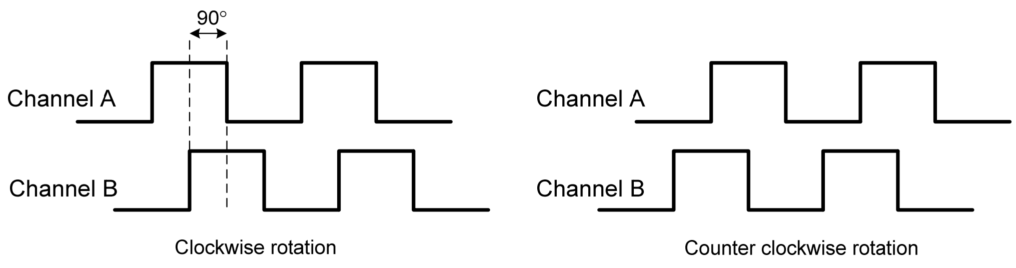


Figure 3.1 – Ideal output of an incremental encoder

The main advantage of incremental encoders is that the resolution is not correlated with the number of output channels, which is 2 for any resolution, unlike absolute encoders, which need to have 2^n output ports to provide n-bit resolution. Therefore, if an incremental encoder is employed, no hardware modification is required in the motion controller interface if a different encoder with higher (or lower) resolution is implemented.

On the other hand, the output of incremental encoders needs to be interpreted by the decoder chip, by means of counting the rising/falling edges of the output signals A and B of the encoder. It is obvious that the decoding process needs to be realized at a high sampling frequency since missing a logic level change of a signal would mean misinterpretation of the feedback.

FPGA is a suitable choice for high frequency decoding purposes, because of its logic level design capability, as stated in the introduction of this chapter. A perfectly optimized module is developed for the sole purpose of encoder decoding; in order to utilize the available clock source with full capacity (i.e. no computational delays). The developed encoder module is presented in the next section.

3.1.1 Incremental encoder decoding module

Incremental encoder decoding module, or as commonly referred in this text as the “encoder module” is developed using Verilog HDL and its symbolic representation as generated by Quartus II (Altera’s programmable logic device design software that is used throughout this chapter) is presented in Figure 3.12.

As can be seen from Figure 3.12, the module takes two channels from the incremental encoder as its input, as well as a clock and reset signal. The output of the module is simply the result of the current pulse count, obtained via quadrature decoding. Note that *clk* signal is the input clock of the module and a sufficiently fast clock is necessary for successful decoding. In this implementation, the fastest clock on the *Altera DE1* board (50 MHz) is connected to this signal. For the implementation, first the inputs from channels A and B are buffered in a 3-bit register in order not to miss any logic level changes in the signal. Verilog HDL code of this implementation is as shown in Table 3.1. Note that signal buffering is realized with this approach throughout this design.

Table 3.1 – Verilog HDL code segment for encoder channel buffering

```
reg[2:0] inA_delayed, inB_delayed;
always @(posedge clk) inA_delayed <=
{inA_delayed[1:0], inA};
always @(posedge clk) inB_delayed <=
{inB_delayed[1:0], inB};
```

As can be observed, at the system clock frequency, input channels A and B are sampled and buffered into registers called inA_delayed and inB_delayed. Using these signals that are delayed 2 clock cycles (corresponding to 40 ns for the 50 MHz clock), it is possible to detect a pulse generated by the encoder, as well as determine its direction. Verilog HDL code of this implementation is shown in Table 3.2.

Table 3.2 – Verilog HDL code segment for pulse and direction detection

```
wire count_enable = inA_delayed[1] ^
inA_delayed[2] ^ inB_delayed[1] ^
inB_delayed[2];
wire count_direction = inA_delayed[1] ^
inB_delayed[2];
```

As can be seen, 3 XOR gates are implemented to detect the level change of the pulse and 1 XOR gate is implemented to determine the direction. After detecting a pulse from an encoder channel and determining its direction, it is trivial that a count register needs to be increased/decreased according to its input, as shown in Table 3.3.

Table 3.3 – Verilog HDL code segment for encoder count calculation

```
always @(posedge clk)
begin
  if(count_enable)
  begin
    if(count_direction) count<=count+1;
    else count<=count-1;
  end
end
```

At this point, comparing this design with a softcore processor design for the same purpose would be helpful to demonstrate the suitability of the customized module approach. Even if minimum specifications are selected for the processor, it would still require a significant processing time, where the design could also be implemented with a simple and efficient HDL code. Note that the success of the module depends on a high sampling clock and a simple design, which are fulfilled by this approach.

The generated module is tested by two means. In the first experiment, an incremental encoder generating 2000 pulses/rev (which corresponds to 8000 pulses/rev with quadrature decoding) is employed and the encoder shaft is driven manually for a certain period of time. In the end, when the encoder shaft is returned to its original position, a multiple of 8000 is read on the PC via RS-232 connection. After this relatively simple experiment, a more complex and convincing experiment is performed.

In the next experiment, an FPGA module to generate an encoder-like signal is developed to test the decoding module. This module is capable of generating encoder signals up to 6.25 MHz, which is limited by the fastest clock available (50 MHz) on the FPGA board. The output of this module is connected to the general purpose I/O ports of the FPGA board; which can be treated as output signals of a real encoder. Therefore, using this module, encoder signals in a wide range of frequencies could be generated for testing purpose. Two signals generated by this module with 6.25 kHz and 625 kHz frequencies are shown in Figure 3.2 and Figure 3.3.

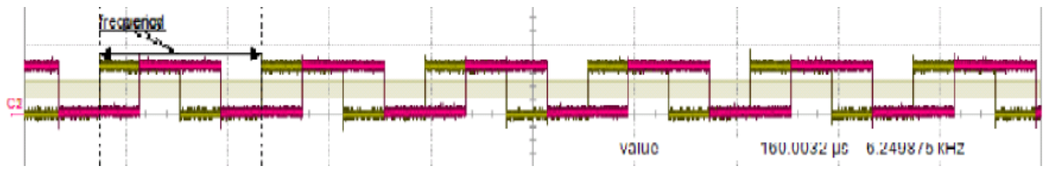


Figure 3.2 – Encoder signal generated by the FPGA (6.25 kHz)

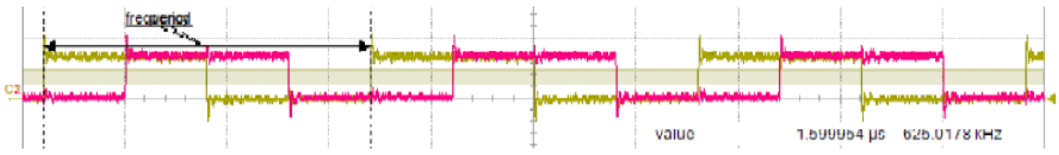


Figure 3.3 – Encoder signal generated by the FPGA (625 kHz)

As seen in Figure 3.2 and Figure 3.3, the signals successfully represent the ideal encoder signal, as shown in Figure 3.1. Using this module, an experiment is performed with different frequencies ranging from 200 Hz to 1 MHz, and the direction of rotation is changed numerously (between 5 to 20 times) via the on-off switches available on the FPGA board. The result of this experiment is shown in Figure 3.4.

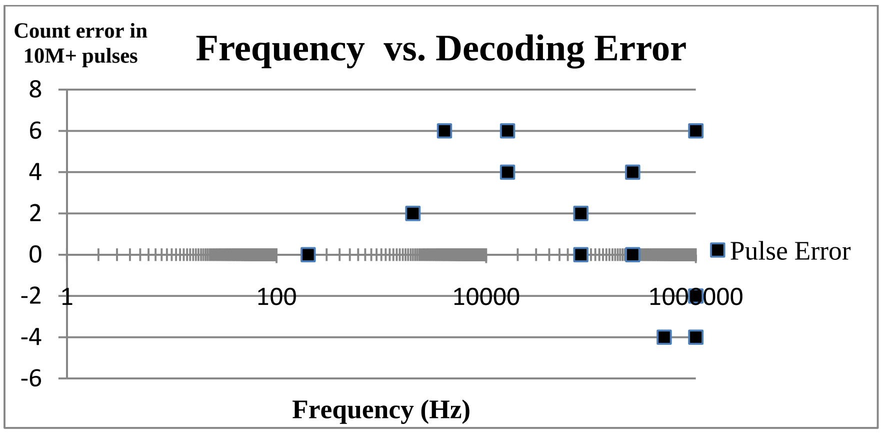


Figure 3.4 – Frequency vs. decoding error

As shown in Figure 3.4, in frequencies greater than 1 kHz (1000 rev/s), count error is equal to only a few pulses when a sum more than 10 Million pulses are generated by the encoder module. Those frequencies are already extremely high for industrial applications and it can be deduced that FPGA implementation of an incremental rotary decoder is sufficiently successful for industrial motion control applications.

3.2 Pulse width modulation (PWM)

Pulse width modulation (PWM) is a technique to provide intermediate electrical power by changing the duty cycle of a high frequency digital signal. Therefore, it is a commonly employed method to represent an equivalent analog signal with a digital signal. While it is possible to use PWM signal to directly drive a motor (via a servoamplifier), it is also possible to feed the PWM signal to a motor driver where PWM represents a manipulated input command (such as torque). Furthermore, PWM can also be used for serial communication, in a 2-wire scheme similar to SPI.

Essentially, frequency of the PWM signal is crucial and there are two factors affecting the attainable frequency of the PWM signal: frequency of the input clock and the desired resolution. The relationship between the system clock, resolution and the attainable PWM frequency is obtained as $f_{pwm} = f_{clk} / 2^R$ where f_{clk} and R correspond to the system clock and the resolution respectively. A calculation based on the fastest available clock on *Altera DE1* board (50 MHz) and a 10-bit resolution (which is a highly sufficient value for industrial applications) can be performed as 50MHz / 2^{10} = 48.8 kHz. This equation shows that an equivalent analog signal can be represented with 10-bit accuracy at 48.8 kHz, when a 50 MHz clock is available.

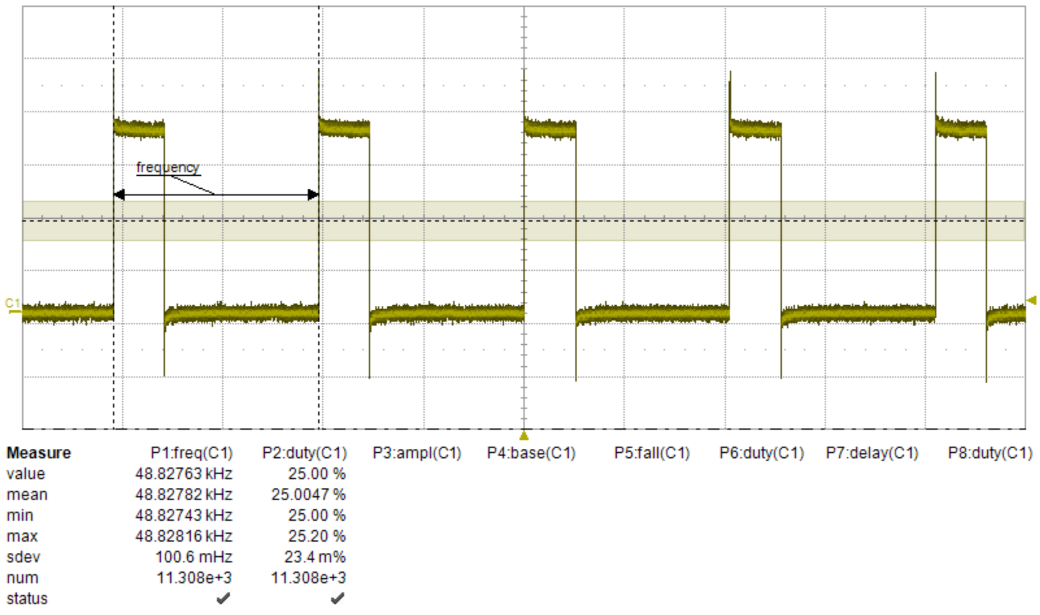
As discussed earlier, PWM can be employed in different schemes. It can be used directly to drive a servo-motor via a servo-amplifier: its output can represent a torque command for another controller or it can be used to transmit serial data to another chip. However, the design differs between the two cases when PWM signal represents an analog signal and is used to transmit data. Therefore, two modules called PWM generator (for producing an equivalent analog signal) and PWM transmitter (for serial communication) are developed and included in the motion controller design.

3.2.1 PWM generator module

PWM generator module (or commonly referred in this text as PWM module) is a simple module that can create a PWM signal that represents its input data. PWM module is developed using Verilog HDL and its symbolic representation as generated by Quartus II is presented in Figure 3.12. As can be seen in Figure 3.12, the module takes a 10-bit `duty_in` signal as its input, as well as a clock and enable signal. The output of the module is simply the generated PWM signal called `pwm_out`. Verilog HDL code of this implementation is shown in Table 3.4.As can be observed, as long as the `clkctr` is smaller than the `duty_in` signal, the output of the PWM is equal to 1. As can be seen, PWM module is simple; yet efficient in creating equivalent analog signals. Output of the PWM module is shown in Figure 3.5.

Table 3.4 – Verilog HDL code segment for encoder count calculation

```
always @(posedge clk)
begin
   clkctr <= clkctr+1;
   if(enable)
   begin
      if(clkctr < duty_in) pwm_out <= 1;
      else pwm_out <= 0;
   end
   else pwm_out <= 0;
end
endmodule
```



Measure	P1:freq(C1)	P2:duty(C1)	P3:ampl(C1)	P4:base(C1)	P5:fall(C1)	P6:duty(C1)	P7:delay(C1)	P8:duty(C1)
value	48.82763 kHz	25.00 %						
mean	48.82782 kHz	25.0047 %						
min	48.82743 kHz	25.00 %						
max	48.82816 kHz	25.20 %						
sdev	100.6 mHz	23.4 m%						
num	11.308e+3	11.308e+3						
status	✓	✓						

Figure 3.5 – Output of the PWM module with 25% duty cycle

Notice that the frequency value is 48.8 kHz, as calculated. Duty cycle is set as 25%, which can be adjusted with a 10-bit resolution.

3.2.2 PWM transmitter module

PWM transmitter module creates a PWM signal; however this signal is not continuous as a regular PWM signal and is generated only when a data needs to be transmitted. PWM transmitter module is developed using Verilog HDL and its symbolic representation as generated by *Quartus II* is presented in Figure 3.12.

As shown in Figure 3.12, PWM transmitter module has an extra input and output different from the PWM module. This is due to the fact that, since the PWM transmit module is used for serial communication, an initiate transmission signal *init_xmit* is necessary for the transmission start. Furthermore, a similar signal is required as the output, in order to inform the receiver module that the transmission has started. Output of this module is shown in Figure 3.6.

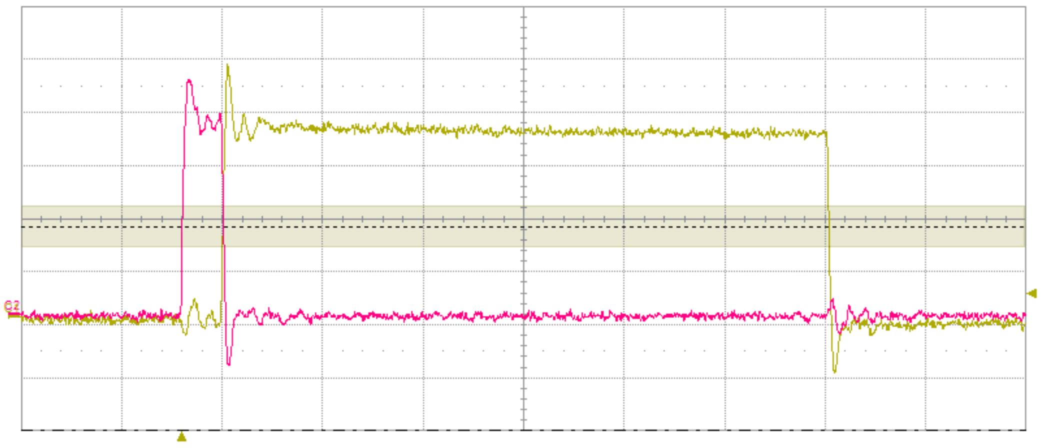


Figure 3.6 – Output of the PWM transmit module

In Figure 3.6, pink line represents the `xmit_clk` and the yellow line represents the `PWM_data_out`. As can be observed, falling edge of the `xmit_clk` initiates the transmission and `PWM_data_out` is set to 1 with the same method employed in the PWM module. Besides the `xmit_clk`, another difference of this module is the frequency of this signal, which is **not** equal to 48.8 kHz, since the module is initiated only when data transmission is necessary. In a controller topology, frequency of this signal would be equal to the sampling frequency of the controller.

3.2.3 PWM receiver module

PWM receiver module interprets the PWM signal created by the PWM transmitter module. PWM receiver module is developed using Verilog HDL and its symbolic representation as generated by *Quartus II* is presented in Figure 3.12.

As can be seen in Figure 3.12, input of the module matches the output of the transmitter module and outputs of the module are the interpreted data that is `PWM_recv_data` and `PWM_OK` signal which generates a rising edge when data transmission is complete.

3.3 Finite pulse generation

In motion control applications, oftentimes the controller needs to drive multiple axes that have different control requirements. For instance, if a 3-axes turning center is considered, position control of the cutting tool requires a much simpler control approach than the speed control of the spindle motor. Therefore, in cases where a simple controller is sufficient, it may be desirable to drive that axe via the "pulse control mode" of the driver.

In order to drive a controller in "pulse control mode", it is necessary to produce a number of square pulses at a certain frequency that corresponds to the speed of the motor shaft. Evidently, this frequency is limited by the specifications of the driver/motor couple.

3.3.1 Pulse generator module

Pulse generator module gets a number of pulses and frequency as its input and produces the specified number of pulses at the desired frequency. Pulse generator module is developed using Verilog HDL and its symbolic representation as generated by *Quartus II* is presented in Figure 3.12. As can be seen in Figure 3.12, finite pulse generation has three inputs (other than the system clock) called `init`, `pulse_no` and `pulse_freq_div`. `Init` signal represents the initialize signal for the generator and rising edge of this signal enables this module. Rising edge of this input is detected by buffering the signal in a 3-bit register and checking the consecutive bits, as shown in Table 3.5. Note that this Verilog HDL code shown in Table 3.5 is commonly employed for rising/falling edge detection of enable/initialize signals throughout this design.

Table 3.5 – Verilog HDL code segment for *init* rising edge detection

```
reg[7:0] init_delayed;

always @(posedge clk) init_delayed <=
{init_delayed[6:0], init};

wire init_risingedge =
(init_delayed[2:1]==2'b01);
```

`Pulse_freq_div` is employed to set the frequency of the generated finite pulse signal by utilizing another module called the *clock divider* module as shown in Figure 3.12. This clock divider module is utilized in various parts of the overall design, in order to generate clocks (infinite square waveforms) at different frequencies. As can be seen, its inputs are the system clock and a divider and its output is a clock with frequency f_{clk_out} that is equal to f_{clk_sys} / divider. This clock divider module is utilized in the pulse generator module as shown in Table 3.6.

Table 3.6 – Verilog HDL code segment for *clock divider* utilization

```
clk_divider clk_div1(.divider((pulse_freq_div >>
1)),
                           .sys_clk(clk),
                           .clk_out(clk_pulse));
```

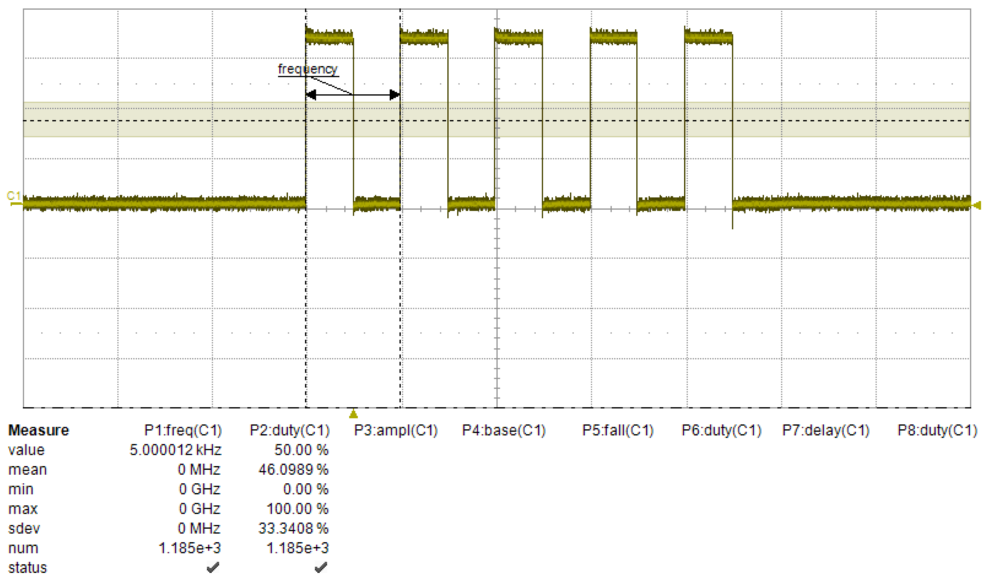
As can be seen in Table 3.6, `pulse_freq_div` signal is directly connected to the input of the divider module, with a bitshift to right, which would double the output frequency of the divider. The reason for this shift is revealed in the pulse generator segment of this code, shown in Table 3.7.

Table 3.7 – Verilog HDL code segment for pulse generation

```
always @(posedge clk)
begin

   if(init_risingedge)
   begin
   PULSE_on <= 1;
   clkctr <= 0;
   end

   if(PULSE_on && clk_pulse_risingedge)
   begin
   clkctr <= clkctr + 1;

   if(clk_pulse_risingedge) pulse_out <= (1-
pulse_out);

   end

   if((clkctr >> 1) == pulse_no)
   begin
   PULSE_on <= 0;
   pulse_out <= 0;
   end

end
```

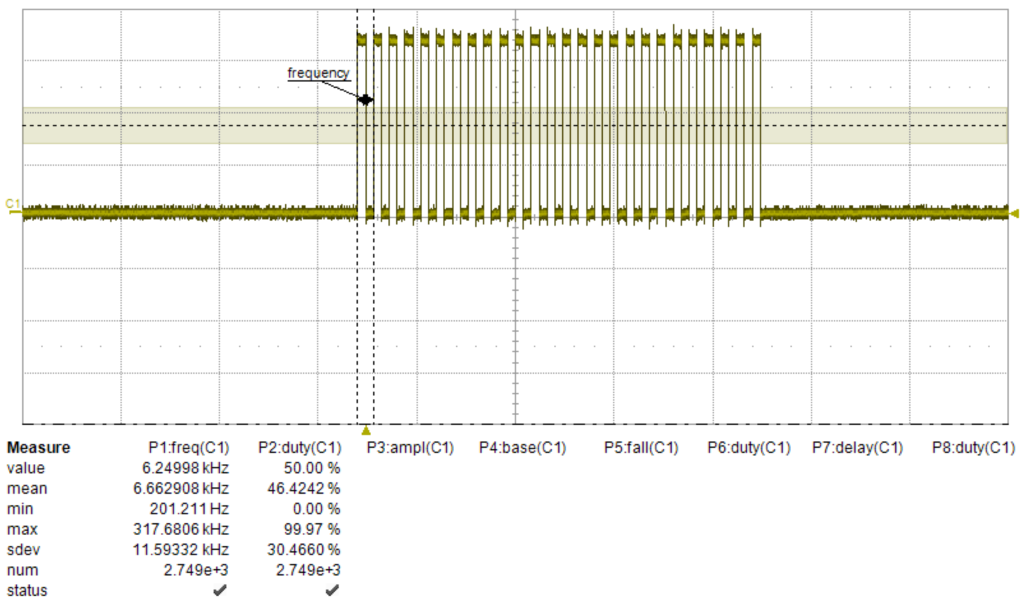
As can be seen in Table 3.7, a register called `pulse_on` is set to 1, when a rising edge of the initialize signal is detected. As long as this registers logical value is true and the clock pulse generated by the clock divider module has a rising edge (which has an analogous code to the one provided in Table 3.5) `pulse_out` signal changes its logical value. Since this process halves the frequency, the output of the clock divider module needs to be twice the desired output frequency, which explains the reason why `pulse_freq_div` signal is shifted 1 bit to right.

Using this module, different numbers of pulse sequences are generated at different frequencies. Two of these sequences are provided in Figure 3.7 and Figure 3.8.



Measure	P1:freq(C1)	P2:duty(C1)	P3:ampl(C1)	P4:base(C1)	P5:fall(C1)	P6:duty(C1)	P7:delay(C1)	P8:duty(C1)
value	5.000012 kHz	50.00 %						
mean	0 MHz	46.0989 %						
min	0 GHz	0.00 %						
max	0 GHz	100.00 %						
sdev	0 MHz	33.3408 %						
num	1.185e+3	1.185e+3						
status	✓	✓						

Figure 3.7 – Sequence of 5 pulses generated at 5 kHz



Measure	P1:freq(C1)	P2:duty(C1)	P3:ampl(C1)	P4:base(C1)	P5:fall(C1)	P6:duty(C1)	P7:delay(C1)	P8:duty(C1)
value	6.24998 kHz	50.00 %						
mean	6.662908 kHz	46.4242 %						
min	201.211 Hz	0.00 %						
max	317.6806 kHz	99.97 %						
sdev	11.59332 kHz	30.4660 %						
num	2.749e+3	2.749e+3						
status	✓	✓						

Figure 3.8 – Sequence of 26 pulses generated at 6.25 kHz

3.4 Serial Peripheral Interface Bus (SPI)

Serial Peripheral Interface Bus (`SPI`) is a serial communication protocol that is commonly used for data transmission between chips. SPI bus operates in full duplex mode and communication is achieved in master/slave mode where the master chip initiates the data transmission by selecting the target chip via the slave select signal (`SSEL`), followed by the serial clock signal (`SCLK`). During operation, data is exchanged from master out slave in (`MOSI`) and master in slave out (`MISO`) ports in full duplex mode. While the most common word size for transmission is 8-bits, other sizes are also commonly implemented. A simple schematic diagram of the SPI bus between a master and two slave chips is presented in Figure 3.9.

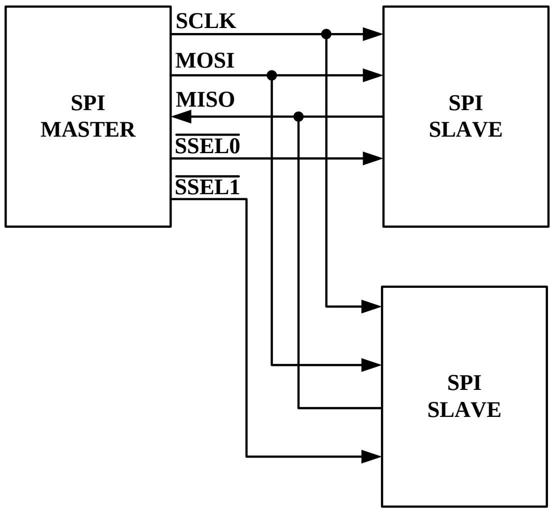


Figure 3.9 – SPI bus between a master and two slave chips

While there are different serial communication protocols such as I^{2}C and One-Wire that can also be implemented on an FPGA; in terms of data transmission speed and multiple chip support, SPI is a better choice for development. For instance, in terms of transmission rates, the closest rival of SPI is I^2C with transmission speeds of 100-400 kbit/s, whereas SPI can achieve 5-10 Mbit/s speed. Therefore, an SPI module is selected for development for the motion controller design and the details are presented in the next section.

3.4.1 SPI module

SPI module is developed using Verilog HDL and its symbolic representation as generated by Quartus II is presented in Figure 3.12. As can be seen in Figure 3.12, SPI module is configured for operating in slave mode; however it is also possible to change the operation mode from slave to master with a few modifications in the HDL code. Note that `clk` signal is the input system clock of the module and not relevant with the SPI bus. Verilog HDL code segment realizing the data transmission is provided in Table 3.8.

As seen in Table 3.8, SPI module is working only when *SSEL* signals logical value is 0. When a falling edge of the *SCK* signal is detected (analogous to the method explained in Table 3.5), `bitcnt` register is increased by 1 and `byte_data_received` is shifted left with the new bit coming from *MOSI*. On the other hand, at the rising edge of the *SCK* signal, *MISO* register is shifted 1 bit to left, in order to set the new bit for transmission.

Table 3.8 – Verilog HDL code segment for *SPI* utilization

```
always @(posedge clk)
begin
  if(SSEL)
    bitcnt <= 0;
  else
  begin
  if(SCK_fallingedge)
  begin
    bitcnt <= bitcnt + 1;
    byte_data_received <=
{byte_data_received[6:0], MOSI};
  end
  if(SCK_risingedge)
    MISO << 1;
  end
end
```

The developed SPI module is implemented on the FPGA board as the slave chip, and the communication is tested via an 8-bit microcontroller (PIC16F877A), operating as the master chip. Two instances obtained during operation of the SPI bus via an oscilloscope are presented in Figure 3.10 and Figure 3.11. In Figure 3.10 and Figure 3.11, D0 represents the data signal (*MOSI*), D1 represents the serial clock signal (*SCLK*) and D2 represents the slave select signal (*SSEL*). As can be observed, data transmission starts with falling edge of the *SSEL* signal, and then *SCLK* produces 8 pulses (since the word size for this application of SPI is selected as 8-bits). During this process, logic level of the *MOSI* signal changes according to the data that is to be transmitted at each rising edge of the *SCLK*; and read by the slave chip at each falling edge of the *SCLK*. As can be seen in Figure 3.11, before the next byte is transmitted, initial states of the *SCK* signal and the *SSEL* signal are identical while *MOSI* is different, since it represents the data.

Transmission tests are performed between the FPGA board and PIC 16F877A microcontroller with this module and the module is proven to be working successfully at speeds up to 250kbit/s, which is a highly sufficient value for the motion controller design problem.

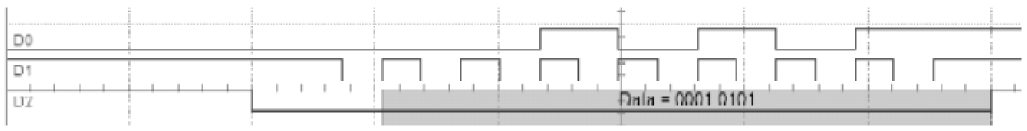


Figure 3.10 – Transmission of an 8-bit data via SPI bus

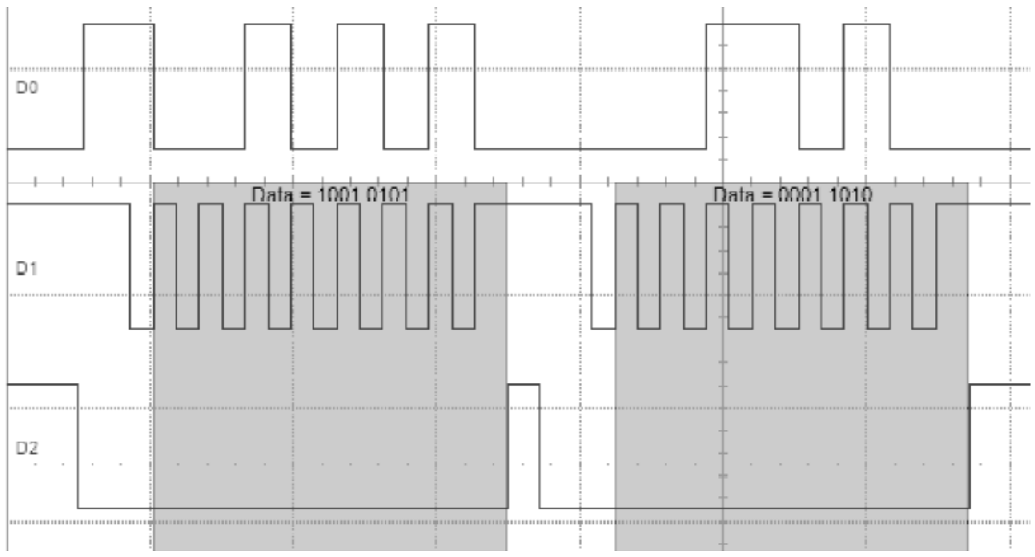


Figure 3.11 – Transmission of an 16-bit data via SPI bus

3.5 Other Interfaces

Other than the aforementioned interfaces and modules, there are a few more modules that are implemented in the design. These interfaces include a ***custom parallel data receive*** module, a custom ***pulse frequency modulation*** module as well as an ***RS-232 controller***. The reason why these modules are not discussed in detail is that, even if these modules are employed in some parts of the design, their applications are custom and their design methodology is very similar to the explained cases; and therefore further discussion would not provide significant information. On the other hand, while RS-232 controller is a relatively important design, it is adapted from an open core source as an intellectual property and therefore, is not comprehensively discussed.

3.5.1 Custom parallel data receive

This interface is developed for fast and simple data transmission between two FPGA chips. The transmission takes place in parallel with a 32-bit width and a clock is generated by the receiver to change the input data at the transmitter end. Symbolic presentation of this receive block is shown in Figure 3.12. As can be seen, *`cmnd_in`* signal is the 32-bit parallel input signal. Every time the *`clk_out`* produces a rising edge, the transmitter block sets the *`cmnd_in`* signal and therefore the new data can be read in parallel. This receive block buffers 9 set of 32-bit data and transmits this to the main module with a *`CMND_recv_ok`* signal indicating that the transmission is over. As can be seen, this is a custom module and is not applicable to industrial peripheral devices without modification in the HDL design.

3.5.2 Pulse frequency modulation

Pulse frequency modulation (PFM) method is a very similar method to pulse width frequency, however in this case the duty cycle of the signal is constant where the frequency is varying. The PFM module is used for transmitting data just as the PWM transmitter and receiver modules that are explained in sections 3.2.2 and 3.2.3. Symbolic presentations are not provided since they are analogous to PWM modules and Verilog HDL codes are not discussed since a more complex design is explained in the finite pulse generation module.

3.5.3 RS-232 controller

RS-232 controller is an important module, and is excessively used in the following chapters for realizing a hardware-in-the-loop simulation on a PC. While it is possible to develop a controller, it is a long and tedious design effort, especially when open core modules are available for utilization. Therefore, an open core IP is utilized in the design and is used throughout this study. The utilized module is "*Simple Asynchronous Serial Comm. Device*" that is developed by *R.Usselmann [1]* and is available at *www.opencores.org.* A symbolic presentation of the "Simple Asynchronous Serial Comm. Device (SASC)" module is presented in Figure 3.12. It should also be noted that this top level module utilizes three other modules that are also developed by the same author.

3.5.4 SRAM controller

SRAM controller is used for utilizing the SRAM available on the FPGA board. An open core IP that is also developed by *R.Usselmann [1]* is utilized in the design and is used throughout this study. A symbolic presentation of the "*SRAM controller*" module is presented in Figure 3.12.

3.6 Closure

In this section, a hardwired design methodology is presented for peripheral device interfaces which are used frequently in motion controller designs. The designs are developed via Verilog HDL, however it should be noted that any other hardware description language (such as VHDL) is evenly applicable for these designs. A number of these presented modules are also employed in Chapter 7 of this thesis.

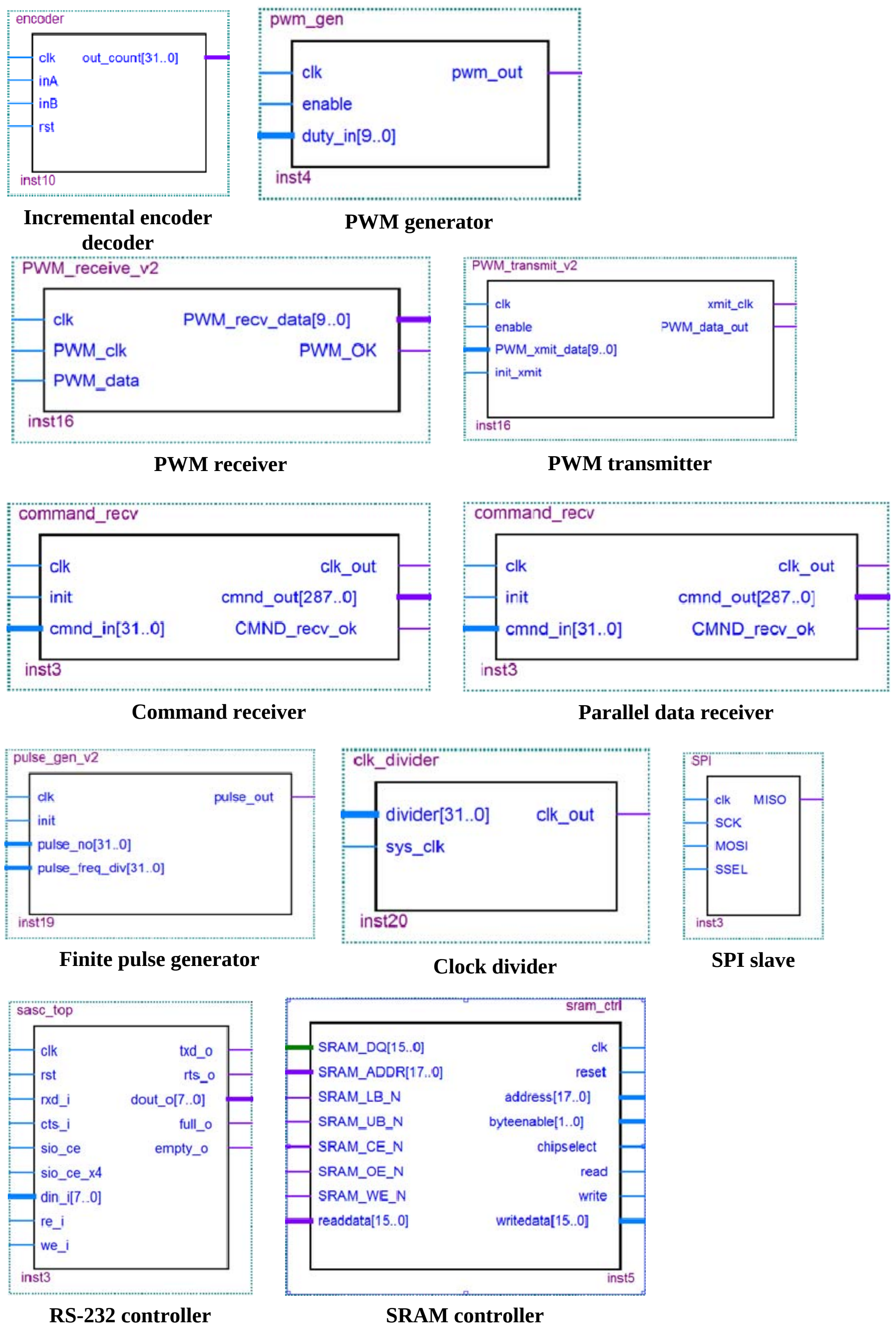


Figure 3.12 – Schematic representations of the developed modules obtained via Quartus II schematic tool

CHAPTER 4

STATE-SPACE CONTROLLER AND OBSERVER DESIGN AND IMPLEMENTATION

State-space controllers are convenient choices for multi-output systems, since they provide means for controlling multiple states of the plant using time-domain based design methodology. However, they exhibit a challenge since all the available states are not generally available in a control system. The general scheme of a motion control system is that the angular/linear position feedbacks are obtained via encoders but the time derivatives of these states are unavailable from sensor feedback; since providing a second feedback (velocity/acceleration) is generally costly or unfeasible. Therefore the controller is usually employed as coupled with a state observer, in order to estimate these unavailable states that are required by the controller.

The overall implementation of this controller and observer scheme can be realized by employing a series of matrix multiplications (as will be discussed in the following section); however the initial problem remains as the realization of the basic multiplication of two elements of these matrices and how these elements could be defined with different data types to reduce the computation load from the multiplication unit.

In this chapter, after a brief introduction of the controller/observer scheme that is to be implemented, two diverse approaches are presented for performing these calculations required by a full-state feedback controller and a Luenberger-type state observer. However, as it will be revealed later, applications of these methods are not limited to state-space controllers and can be used for realization of other algorithms. For instance, in Chapter 5, digital filter implementation on FPGAs is thoroughly discussed and the same methods presented here are applied for the digital filter design problem.

4.1 Full-state feedback controller

State space controllers find their use in almost all industrial motion control applications. The control law of a typical state-space controller can be simply expressed as;

$$\boldsymbol{u} = -\mathbf{K}\big(\hat{\mathbf{x}}(k) - \mathbf{x}_{\mathbf{r}}(k)\big) \tag{4.1}$$

Here $\mathbf{u}$ is the manipulated input vector; $\hat{\mathbf{x}}$ is the estimated state vector; $\mathbf{x}_r$ is the reference state vector and $\mathbf{K}$ refers to the gain matrix. Once the system equations governing the dynamics of the plant are obtained, the gains in (4.1) can be adjusted to yield desired control characteristics using modern control theory. Note that since all states of the plant are not measured for all practical purposes, a full-state observer needs to accompany the design in order to estimate the missing components of the state vector. For instance, a Luenberger-type state observer takes the following form:

$$\hat{\mathbf{x}}(k+1) = (\mathbf{F} - \mathbf{LH})\hat{\mathbf{x}}(k) + \mathbf{G}\boldsymbol{u}(k) + \mathbf{L}\boldsymbol{y}(k) \tag{4.2}$$

Here **F**, **G**, **H** correspond to the system matrices formed by the estimated parameters; y(k) is the controlled output vector while **L** denotes the gain matrix of the observer. Figure 4.1 shows the block diagram of the state-space control system with the Luenberger-type state observer.

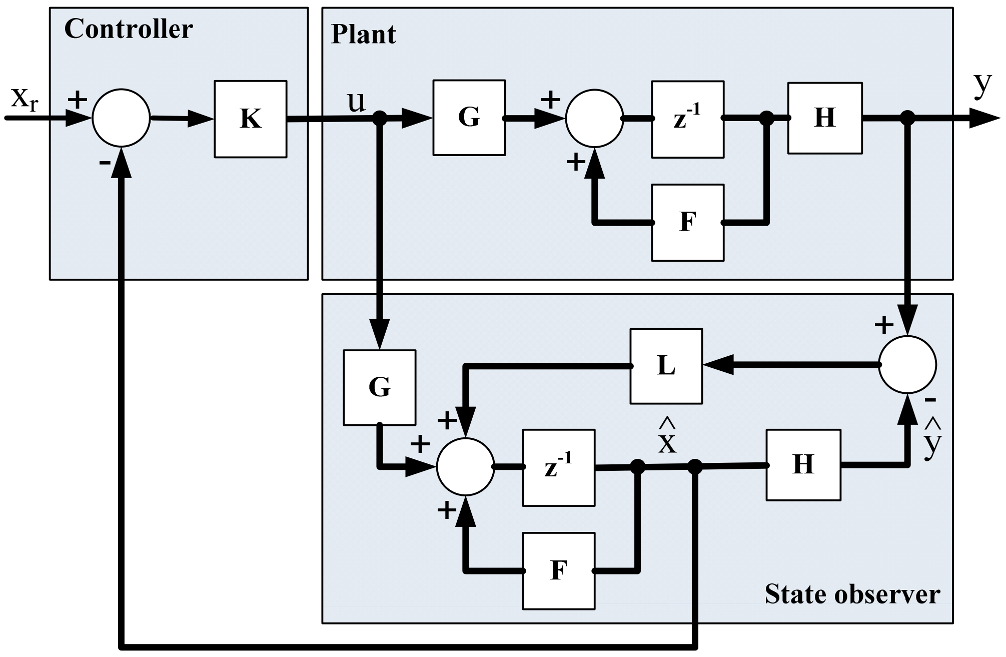


Figure 4.1 – Block diagram of the control system

In industrial motion control, the states of the system are frequently selected as angular position (θ) and (instantaneous) angular velocity (ω). Thus, the general approach is to estimate the velocity using the measured position. There exist versatile estimator algorithms in the literature for velocity estimation in digital systems employing encoders/resolvers.

However, no single estimation algorithm exists to cover all applications where dynamic operating conditions do vary considerably such as the one considered in this study. Hence, when good estimates on system parameters are available, it is more desirable to observe the unavailable states, rather than to estimate them using higher-order differencing methods. Implementation details of this controller/observer scheme are presented in the proceeding sections.

4.2 Implementation

In Chapter 2, many recent studies employing an FPGA in controller designs are presented and different architectures are discussed where FPGAs are used coupled with a DSP/MCU or used exclusively as the controller. Furthermore, for the FPGA based (non-hybrid) designs, various alternatives are discussed to handle the computations required by the controller algorithm; including embedded softcore processors and hardwired solutions (including embedded multipliers, floating point units, custom arithmetic modules and etc.)

Unlike embedded softcore processors, hardwired solutions require relatively long and tedious design effort with hardware description languages (HDLs) and therefore many studies in recent literature tend to include an embedded processor in their FPGA based designs. However, using an embedded processor may have disadvantages such as consuming a bulk amount of the resources of the chip and decreasing the flexibility and processing rate of the module. Therefore, in order to fully benefit from the flexible architecture of an FPGA, it is necessary to develop custom FPGA modules for computation.

In this section, two different approaches are presented for realization of the controller/observer scheme that is presented in the preceding section. The first approach uses a *matrix multiplier module,* which is developed by HDL design (with Verilog HDL) and incorporates other custom multiplication modules called *IMUL*, *FMUL* and *FPU*. Evidently, custom multiplier modules are the key features of this method and are discussed thoroughly. The second method is a softcore processor solution which utilizes the *Nios II* softcore processor developed by *Altera* and the design approach is drastically different from the modular approach, since a processor is involved in the design.

4.2.1 Method I: Matrix multiplier module

Matrix multiplier module utilizes a custom multiplication module (which can be selected as *IMUL*, *FMUL* or *FPU*) and an adder, in order to realize a matrix multiplication. However, before proceeding with the details of this module, it is necessary to present the custom multiplication modules and explain the methodology behind these modules as well as how these modules are developed and customized for a motion control application.

4.2.1.1 Customized multiplication modules

Digital control systems have unique properties that relax the usage of floating point arithmetic and thus there is a potential to develop inexpensive yet high-performance solutions. In order to explore the capabilities of a flexible hardware, all the IPs included in the design should be customized (and optimized) for a specific task at hand. Consequently, it is necessary to adapt low level open source IPs within the design or develop custom modules via HDL design methods for a specific controller topology (i.e. a state-space controller) combined with a specific control system (i.e. a motion control system with a motor drive and encoder feedback).

Some of the advantageous attributes of digital motion control systems are as follows:

- Outputs of all sensors used in controls technology are essentially amplitude-quantized and can be conveniently represented as (signed/unsigned) integers.
- Reference signals (i.e. the command vector), which are to be compatible with the sensory data, are to be generated as integer (number) sequences.
- Manipulated outputs of almost all control systems need to be amplitude-quantized while sending them out to the output interface.
- With proper scaling, the controller gains might be cast as integers without a significant change in the overall dynamics of the controlled system dynamics.

As explained in Chapter 3, in many motion control applications, incremental optical encoders (either linear or rotational) are exclusively employed to measure position of which is commonly characterized as integer counts of pulses being produced by these devices. Similarly, the manipulated output (torque or velocity command to the motor driver) is represented as a finite-length binary number to be latched onto a digital-to-analog converter. Note that within the context of this study, such systems will be referred to as **"quantized input/output"** control systems. Furthermore, if the control gains could be also cast as integers, the resulting system will be called **"fully quantized system."**

For a fully quantized system, a state-space controller can be a suitable choice since the pole placement techniques offer a margin for manipulation on the controller gains. Note that the casting of these gains as integers is not a straightforward task as the input arguments of the gain matrix must be pre-scaled which may in turn aggravate the quantization noise. Hence, the overall problem requires a fine balance among conflicting objectives.

It is critical to notice that if a system is fully quantized, one may employ integer-arithmetic entirely in all calculations. On the other hand, for a quantized input/output system, the decimal multiplication algorithms of digital signal processing (such as shift-and-add algorithm) can be utilized. These well-known algorithms are easy to implement on FPGAs with the low-level design tools provided by FPGA manufacturers. Since many industrial motion control applications employ the quantized input-output, such methods can reduce resource costs significantly.

The implementation methods for the state-space controller/observer is to be developed for a quantized input-output (hence including fully quantized) system. In this system, representing the input, output, and feedback states as floating-point numbers do not have an advantage in terms of accuracy. Therefore, it is appropriate to cast and store these quantities as signed/unsigned integers. Note that the selection of the word size is up to the designer and is to be chosen by considering both the system properties (such as feedback resolution) as well as the features of the implementation method. The methods elaborated in the following section make good use of the special properties for control systems.

Two techniques are proposed for matrix calculations which are essential in state-space controllers and observers. Using these techniques, two modules called "IMUL" and "FMUL" are designed specifically to take advantage of the aforementioned special characteristics of control systems. Furthermore, a custom floating point unit (FPU) is also included, in order to demonstrate the resource and time-wise pros and cons of the proposed methods.

4.2.1.2 Method I-a: Integer multiplication (IMUL module)

In FPGAs, it is possible to develop efficient integer multiplication/division algorithms with the logic-level (i.e. combinational circuit and/or embedded multipliers of the FPGA chip) design. Therefore, the first method, which employs a special multiplier module called "**IMUL**", focuses on multiplication of controller gains with system states employing integer arithmetic. To be specific, let us consider the following operation: $y = \lfloor ax \rfloor$ where a ($\in \Re$) is the multiplicand; x and y ($\in \mathbb{Z}$) are the multiplier and the result (product) respectively while $\lfloor \ \rfloor$ refers to floor function. It is obvious that one can represent the fractional number (a) in this operation as the ratio of two integers (N_a, D_a): $y \cong \frac{N_a x}{D_a}$. Hence, the overall problem is reduced to multiplication and division by integers. Note that the success of this approximation is directly correlated to the bit-length of the numerator and denominator.

4.2.1.3 Method I-b: Fixed point multiplication (FMUL module)

Similar to the previous one, this method focuses on performing multiplication/division where the multiplier is essentially an integer. In fact, the proposed method facilitates a fixed-point multiplication unit (called "**FMUL**") where bit-shift/add operations are successively employed to obtain the result. In this paradigm, the fractional number (*multiplicand*) is separated into an integer- and a fractional portion. Two instances of the multiplier module are used to multiply these portions in parallel. When both calculations are complete, the partial products are added to obtain the result.

4.2.1.4 Method I-c : Floating point multiplication (FPU module)

In previous sections, custom multiplier modules "IMUL" and "FMUL" are presented. Those modules are customized modules developed to perform a specific sort of multiplication; and their main advantage is their low amount of resource requirement. In turn, they may not be applicable to other controller topologies; since they are developed to perform a certain type of multiplication. Therefore, a floating point unit is a necessity, in order to perform more complex arithmetic operations required by advanced controllers. Furthermore, employing floating point unit in state space controller design will be also helpful in demonstrating the resource improvement provided by the custom multiplication modules "IMUL" and "FMUL". The implemented floating point unit is an open core IP, developed by *R. Usselmann [1]*. A schematic of this unit with its input/output ports is shown in Figure 4.2 and the terminology is explained in Table 4.1.

As can be seen in Table 4.1, the FPU performs single precision (32-bit) floating point operations, as well as integer to float and float to integer operations. The operation is performed in one-clock, however the output is provided after a 4 cycle delay period. Therefore, while 2 unrelated operations can be completed in 5 cycles, 2 consecutive operations need 8 clock cycles to be completed.

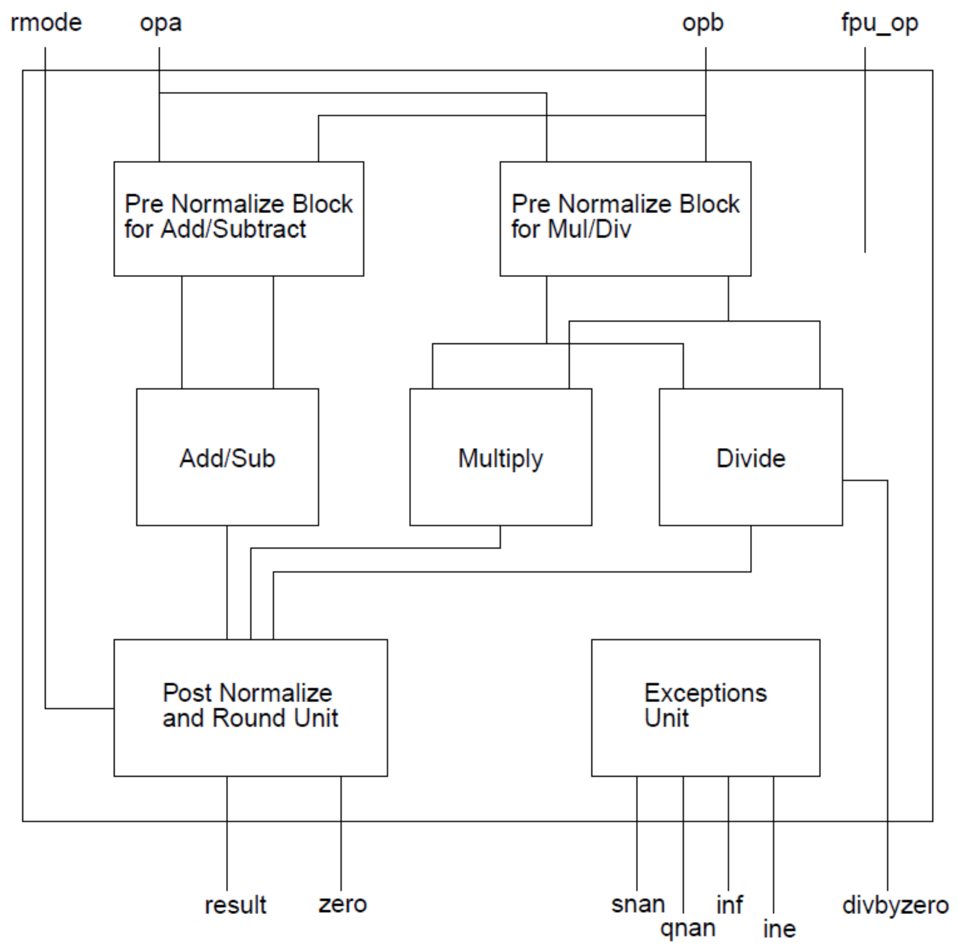

rmode
opa
opb
fpu_op
Pre Normalize Block for Add/Subtract
Pre Normalize Block for Mul/Div
Add/Sub
Multiply
Divide
Post Normalize and Round Unit
Exceptions Unit
result
zero
snan
qnan
inf
ine
divbyzero

Figure 4.2 – Floating point unit

This floating point unit is also adapted as a custom multiplication module, and implemented in the overall architecture as explained in the next section.

Table 4.1 – Terminology used in FPU

Signal name	**Length [Bits]**	**Direction**	**Explanation**
`clk`	1	Input	System clock
`rmode`	2	Input	Rounding mode
`fpu_op`	3	Input	Operation
`opa-opb`	32	Input	Inputs a and b
`out`	32	Output	Output
`snan`	1	Output	Result is not a number
`qnan`	1	Output	Result is not a number
`inf`	1	Output	Result is infinity
`ine`	1	Output	Result is indefinite
`overflow`	1	Output	There is overflow
`underflow`	1	Output	There is underflow
`div_by_zero`	1	Output	Division by zero
`zero`	1	Output	Result is zero

4.2.1.5 Overall Architecture

A matrix multiplier module can be designed, utilizing the afore-mentioned multiplication modules (IMUL, FMUL and FPU). Proposed module, which operates on (classical) *multiply-and-accumulate* principle, is illustrated in Figure 4.3. Note that the matrices are stored in the SRAM and thus the memory interface module sends out the relevant data to the registers of the multiplier controller unit (a finite state machine) on demand. Hence, the architecture provides flexibility in the controller design as the designers can change the data at will.

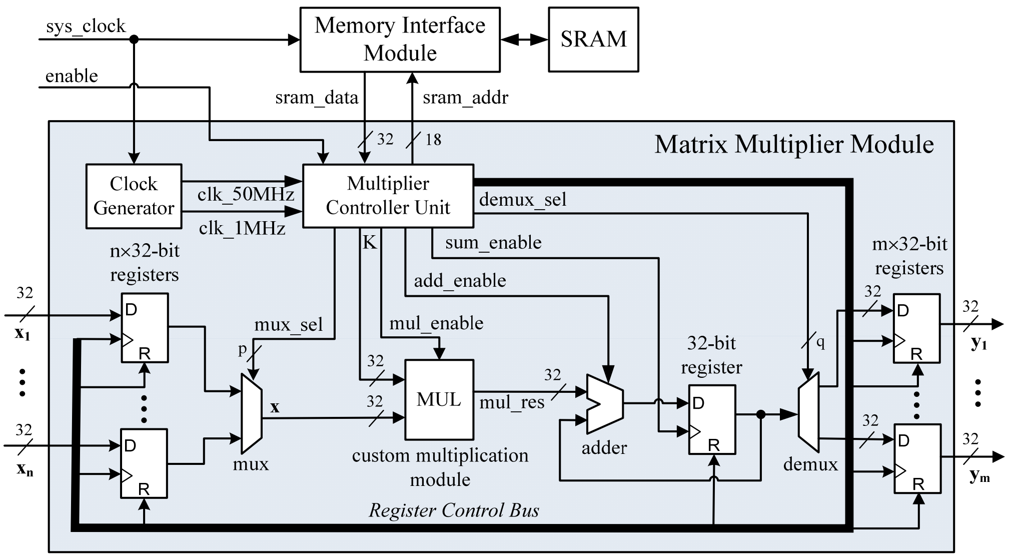


Figure 4.3 – Matrix multiplier module

Similarly, the overall design, which is built on these matrix multiplier units, is presented in Figure 4.4. This unit performs the computations in (4.2) sequentially to obtain $\hat{\mathbf{x}}(k)$ and then proceeds to calculate **u(k)** in (4.1). Note that in the shown architecture all the computations are performed in sequential fashion for the sake of reducing the hardware cost. However, the parallel implementation can be easily realized by eliminating the multiplexer /demultiplexer units in Figure 4.3 while using the instances of custom multiplication modules.

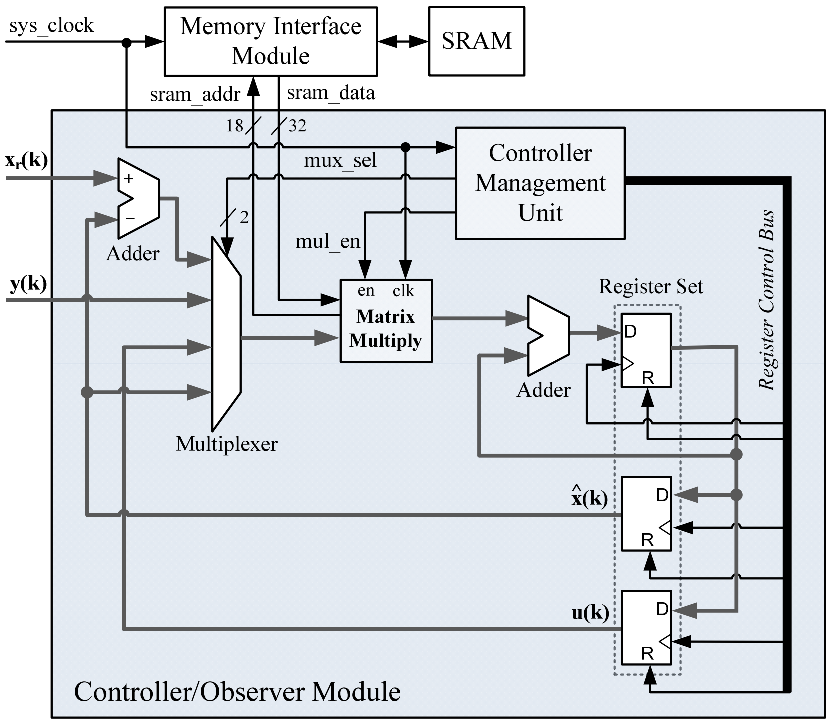


Figure 4.4 – Controller/Observer module

4.2.2 Method II: Softcore processor IP module

Softcore processors are embedded processor IPs developed by FPGA manufacturers, in order to decrease the long and tedious design periods required by low-level (i.e. logic-level) circuit design through hardware description languages (HDL). While shorter and easier design periods are favorable characteristics, the flexibility of the design inevitably decreases due to the bulk implementation of the processor and it may not always be a resource-wise and time-wise beneficial method.

The design approach of a softcore processor is significantly different from the modular approach used in state-space controller implementation and filter implementation via generic filter module. The difference is that, in modular design approach, all the modules are designed with HDLs or schematic design tools and all the connections (data buses, clock signals, enable/reset signals and etc.) between the modules are defined and implemented by the designer. On the other hand, utilizing a softcore processor is via a user-friendly GUI of the design tool provided by the FPGA manufacturer and all the parameters of the processor can be selected easily using this tool. Furthermore, all the peripheral controllers (such as memory management units, serial controllers, GPIOs and etc.) are also provided in this tool, making the overall design process a lot easier and faster.

After implementing the embedded processor along with peripheral controllers to the FPGA, it is possible to develop a C code to realize any algorithm that is supported by the specifications of the implemented processor. In this design method, FPGA implementation of the algorithm is handled by the compiler and therefore the development process is much faster than the modular design approach. Note that, this is one of the most significant differences between the two methods.

4.2.2.1 Method II-a: Softcore processor with integer arithmetic

Implementation details of the second method is as follows; first the specifications of the processor is selected and peripheral controllers such as SRAM and UART controller are added via SOPC (system on programmable chip) builder tool of *Altera Quartus II*. Note that a floating point unit is **not added** to the processor in order to reduce the amount of resources to a minimum. After the embedded processor is implemented to the FPGA, Nios II IDE is utilized to develop and implement the C code into the processor. Similar to methods I-a and I-b, all the parameters and states are stored as 32-bit integers in the C code. A significant advantage of employing a softcore processor is that the processor handles all the SRAM and UART operations, as well as the timing constraints and therefore the design process takes significantly less time than the HDL design case.

Note that the resource requirements of the design does not depend on the algorithm but only the specifications of the processor and peripheral control units. Therefore, only the execution time of the process inevitably depends on the algorithm. As discussed earlier, bulk implementation of the process drastically eases the design in cost of flexibility.

4.2.2.2 Method II-b: Softcore processor with fixed point arithmetic

Implementation details of this module is exactly the same with Method II-a, therefore the resource requirements of this method is also the same. On the other hand, the C code significantly differs from the previous case and multiplications are realized via the fixed point multiplication method presented in Method I-b. The only difference is that in this implementation the "FMUL" module is realized via a C function.

4.2.2.3 Method II-c: Softcore processor with floating point arithmetic

The only difference of this method from the preceding one is the floating point arithmetic unit employed in the design. Therefore, the only change in the methodology is in the specifications of the processor selected. In this method, an optional floating point arithmetic unit (which is not the same unit employed in Method I-c) is added via the SOPC builder tool to the processor and the C code is modified so that the parameters are stored as 32-bit floating point variables.

In the next section, the test case (a nonlinear inverted pendulum system) which is employed via hardware in the loop simulation is presented. In the subsequent section, results of the simulation are provided for all of the presented methods.

4.3 Test case

As a benchmark case, an inverted pendulum system shown in Figure 4.5 is considered.

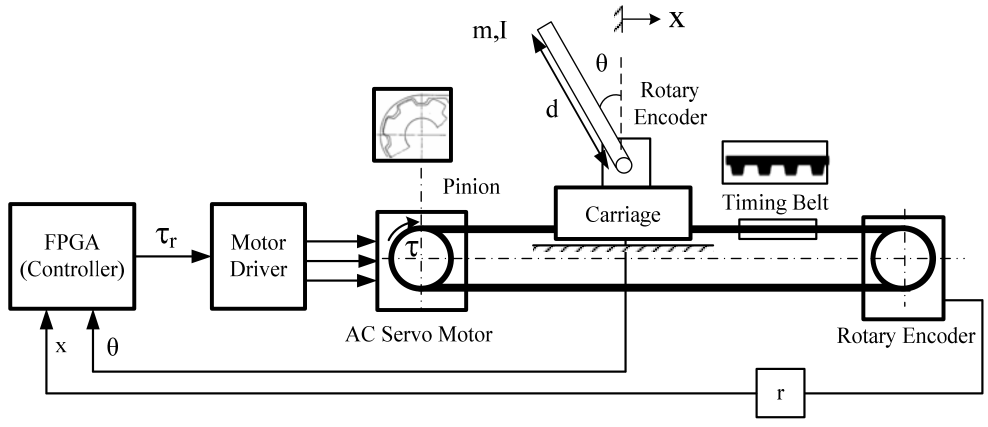


Figure 4.5 – Generic model for pendulum drive system

The system includes an AC servo-motor coupled directly to a timing belt. In this configuration, the motor driver is in the torque regulation mode where the motor along with its driver can be regarded as an ideal torque modulator. Hence, the state-space controller can directly generate the relevant (torque) commands through a digital-to-analog converter. Note that two rotary encoders (which can produce 10000 pulses/rev) are to supply feedback on the position of the carriage as well as the angular position of the pendulum. This choice is also convenient since the system has four states; $(x, \dot{x}, \theta \; and \; \dot{\theta})$ and given that only two states (x, θ) are available via encoder feedback in order to estimate $\dot{x} \; and \; \dot{\theta}$, an observer is required to be implemented in the design that satisfies the testing requirements for the developed methods.

Notice that this can be regarded as a "quantized input/output system" since all the sensor feedback is coming from incremental encoders as pulses and that the output of the FPGA is also an integer representing the motor torque. However, the system cannot be considered as a "fully quantized system" since there exist an observer in the design and while the coefficients of the controller can be cast to close integers via pole placement, it is not possible to implement an observer by using only integers.

The force acting on the carriage is related to the motor torque as $F = \tau\eta/r$; here τ is the motor torque [Nm]; r is the pinion (pitch circle) radius [m] and η is the overall transmission efficiency. Equations of motion for this system become

$$(M+m)\ddot{x} + b\dot{x} + m\frac{d}{2}\ddot{\theta}cos\theta - m\frac{d}{2}\dot{\theta}^2 sin\theta = \frac{\tau}{r}\eta \tag{4.3}$$

$$\left(I + m\frac{d^2}{4}\right)\ddot{\theta} + mg\frac{d}{2}sin\theta = -m\frac{d}{2}\ddot{x}cos\theta \tag{4.4}$$

Here, M is the mass of the carriage [kg]; b is its viscous damping [Nms]; m is the mass of the pendulum, I is its mass moment of inertia [kgm^2]; τ is the manipulated input (i.e. motor torque). Numerical values for the system parameters are provided in Table 4.2.

Table 4.2 – Inverted pendulum parameters

Parameter	M [kg]	m [kg]	I [kgm^2]	b [Nms]	d [m]	r [m]
Value	0.5	0.2	0.006	0.1	0.6	0.01

In order to design a controller and a state observer, the system is linearized around an operating point (θ = 0), since the goal of the control system is to hold the pendulum in upright position. Using (4.3) and (4.4), the state-space representation of the linearized system can be given as:

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-b}{C_2} & -m\frac{d}{2}\frac{C_1}{C_2}g & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-C_1}{C_2}b & \frac{C_1}{C_2}(M+m)g & 0 \end{bmatrix}}_{\boldsymbol{A}} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{\eta}{rC_2} \\ 0 \\ \frac{\eta C_1}{rC_2} \end{bmatrix}}_{\boldsymbol{B}} \tau \tag{4.5a}$$

$$\mathbf{y} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{C}} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{4.5b}$$

$$C_1 = -m\frac{d}{2}\left(I + m\frac{d^2}{4}\right)^{-1}, C_2 = M + m + \frac{C_1 d}{2} \tag{4.6}$$

Having obtained the continuous-time state space representation, the discrete representations of (4.5) can be simply obtained as:

$$\boldsymbol{x}(k+1) = \mathbf{F}\boldsymbol{x}(k) + \mathbf{G}u(k) \tag{4.7a}$$

$$\boldsymbol{y}(k) = \mathbf{H}\boldsymbol{x}(k) \tag{4.7b}$$

where

$$\mathbf{F} = e^{\mathbf{A}T}; \quad \mathbf{G} = \left(\int_{\mathbf{0}}^{\mathrm{T}} e^{\mathbf{A}t}\,\mathrm{d}t\right)\mathbf{B}; \quad \mathbf{H} = \mathbf{C} \tag{4.8}$$

Here, **A**, **B** and **C** denote to the system matrices of (4.5) while T refers to the sampling period of the controller [s]. With the discrete-time model at hand, a state-space controller (along with an observer) can be designed using modern control theory.

4.4 Hardware-in-the-loop simulation results

In the simulation study, the controller topology presented is implemented on the Altera Cyclone II FPGA chip using Verilog hardware definition language. The nonlinear inverted pendulum system (as explained in the previous section) is realized via (non-real-time) hardware-in-the-loop simulation (HILS) in MATLAB environment.

The controller system (controller+observer) is implemented on the FPGA using the presented techniques: Method I-a (integer arithmetic - IMUL), Method I-b (fixed-point arithmetic – FMUL), Method I-c (floating-point arithmetic – FPU), Method II-a (integer arithmetic with softcore processor), Method II-b (fixed-point arithmetic with softcore processor), and Method II-c (floating point arithmetic with softcore processor).

In this scheme, the manipulated torque input ($u = \tau$) generated by the FPGA is transmitted to hardware-in-the-loop simulator running on MATLAB platform. Once the new system states are computed using the dynamic model of the controlled system, they are sent back to the FPGA via RS-232 communication. As explained in preceding sections, the word size allocated to the matrix elements is a major design parameter. In this study, a 2×16-bit word size is selected for each element of the controller/observer matrices: 32-bit for numerator/ denominator pair for IMUL multiplication module (Method I-a) or integer/ fractional portion pair for FMUL module (Method I-b). On the other hand, the system states are stored as 32-bit signed integers. However, this choice is also a design choice and can be modified according to the needs of a particular application.

Results of the HILS are presented in Figs. Figure 4.6 through Figure 4.11. Note that the initial condition of the state vector is selected as $[0\ 0\ -0.05\ 0]^T$ to start out with a pendulum angle to create a challenge for the controller. As can be seen in Figs. Figure 4.6 through Figure 4.11, all controllers react fast and bring the carriage around 0.02m to prevent pendulum arm from falling; and after the initial reaction, the implemented controllers can successfully hold the inverted pendulum around $\theta = 0$ and the carriage's position are bounded around its initial position. Since the system is nonlinear, small differences in calculation of the results between the proposed methods: I-a (II-a), I-b (II-b), and I-c (II-c) cause a relatively different path for the carriage after t = 2s; Nevertheless, all sub-methods of method I yield acceptable performance.

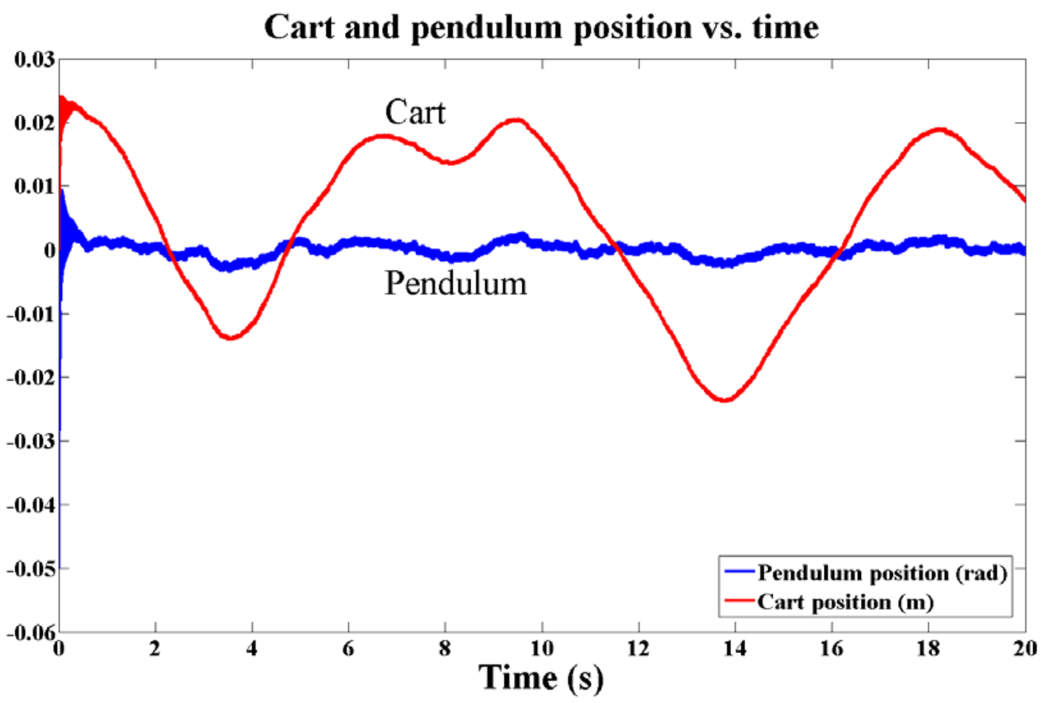


Figure 4.6 – HILS result of method I-a

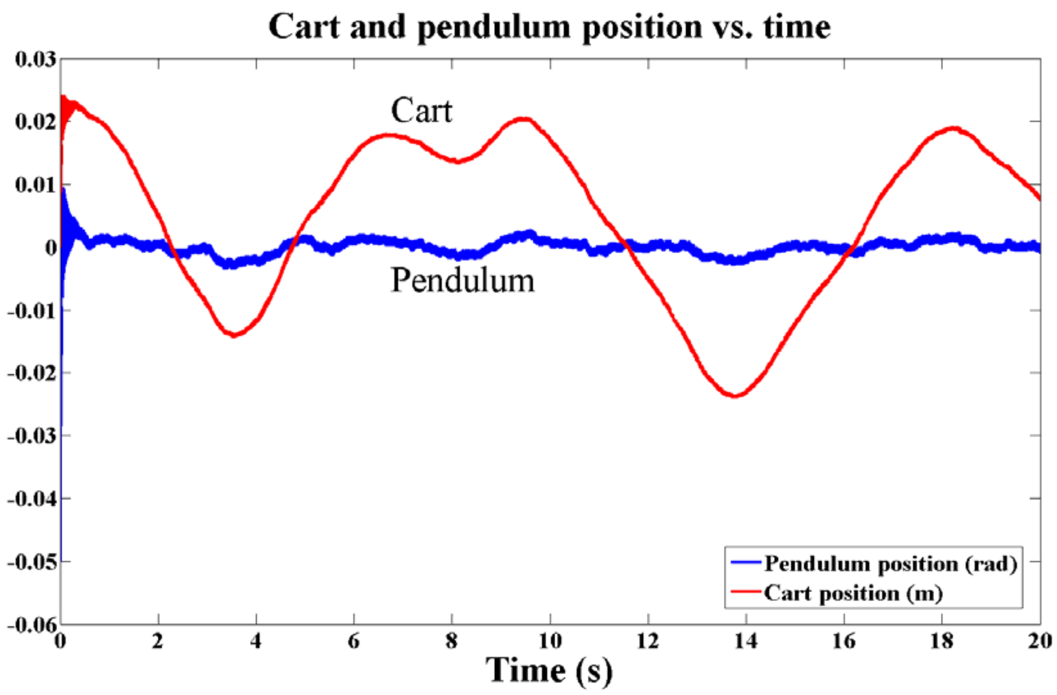


Figure 4.7 – HILS result of method II-a

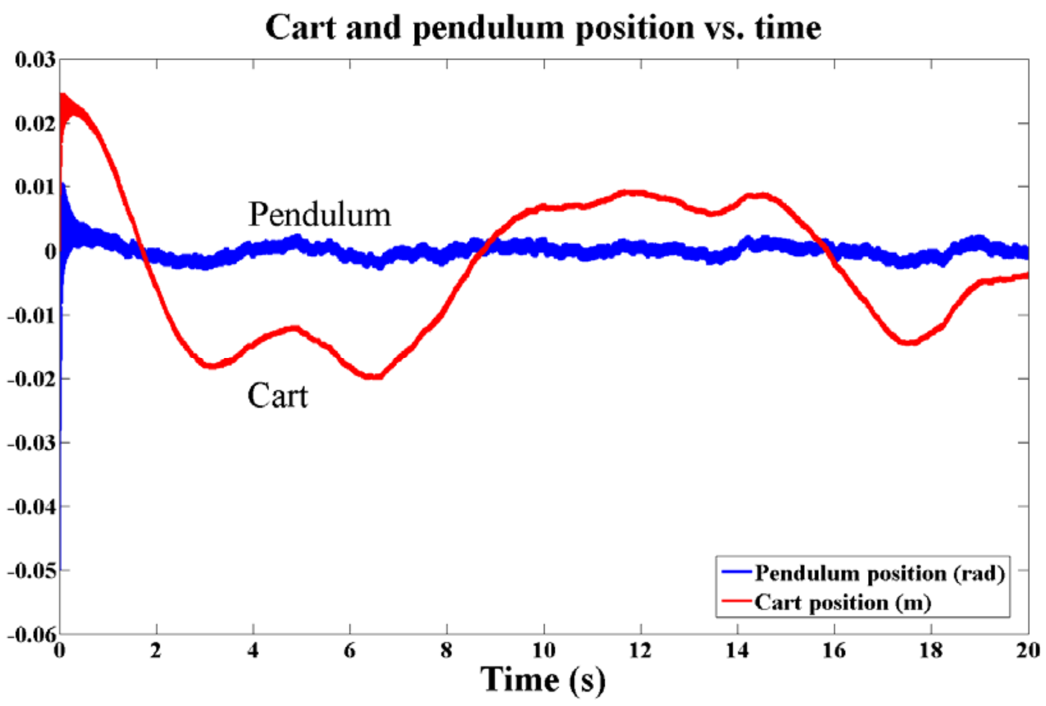


Figure 4.8 – HILS result of method I-b

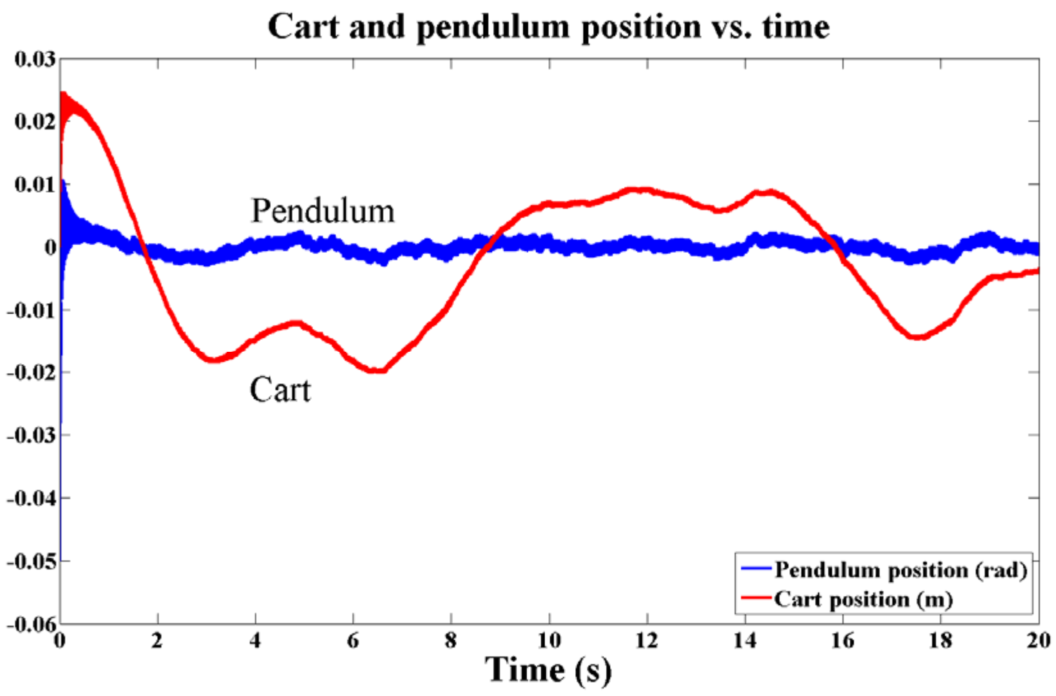


Figure 4.9 – HILS result of method II-b

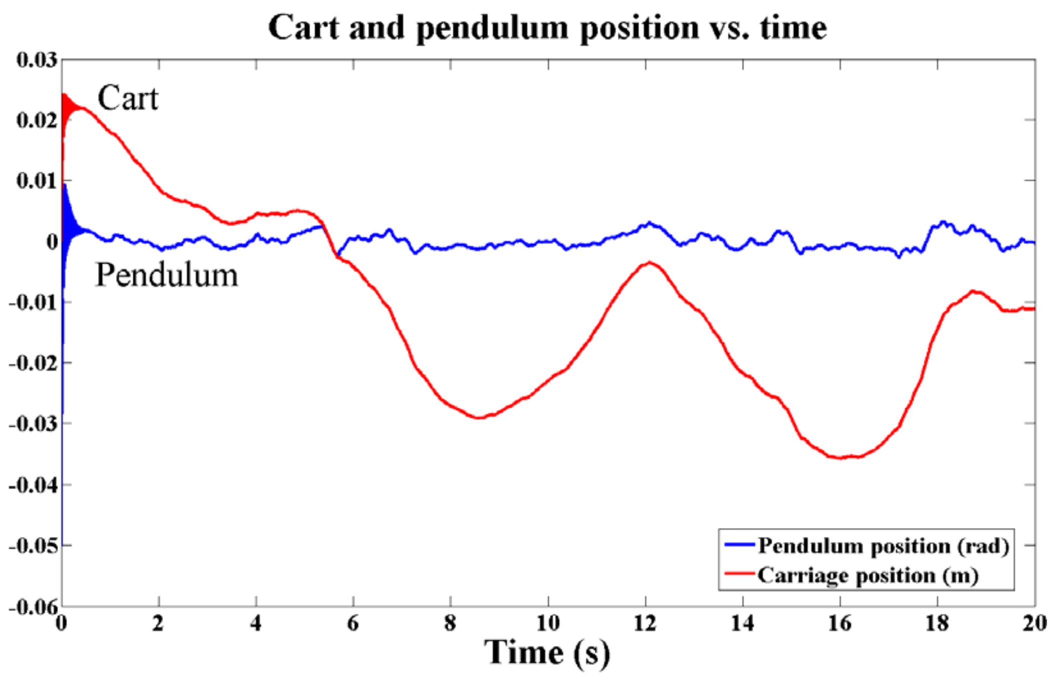


Figure 4.10 – HILS result of method I-c

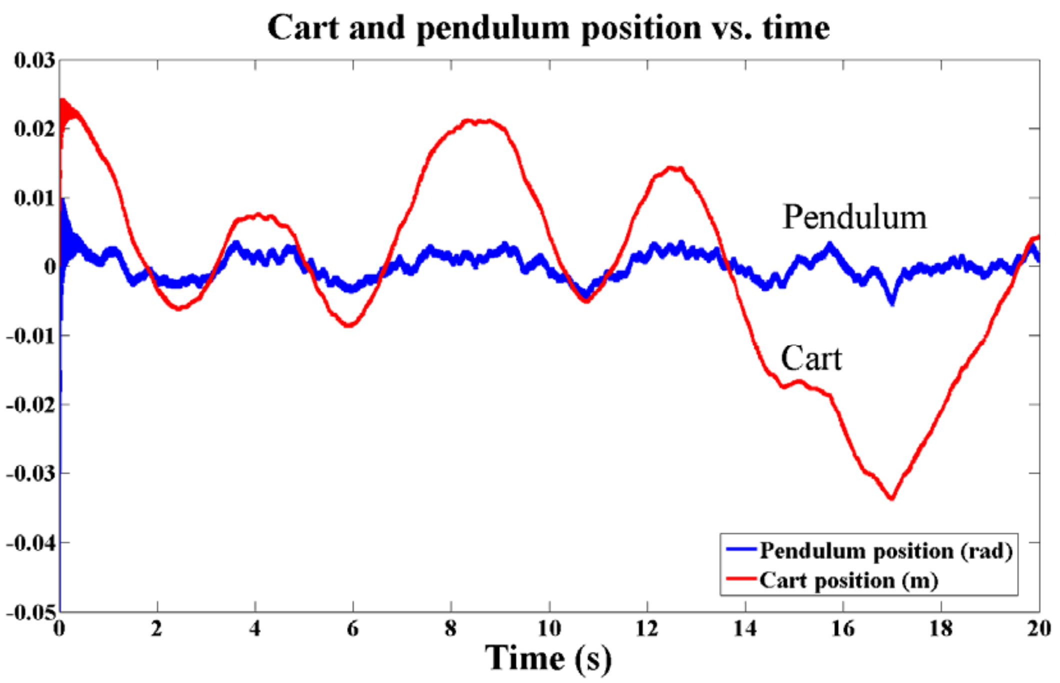


Figure 4.11 – HILS result of method II-c

As expected, result of the method II-a is exactly the same with method I-a, because even if the approach is different, both of the implementations are based on integer arithmetic and therefore this result is predicted. The case is also the same with method II-b and I-b, since the fixed point arithmetic algorithm is the same. On the other hand, as seen in Figure 4.11, result of method II-c is not the same with method I-c, even if they both employ a floating point unit for calculations. However, this is also expected since the floating point units are not the same (method I-c employs an open floating point by *Usselmann [1]* and method II-c employs a floating point developed by *Altera* for *Nios II* processors) and a small computational difference yields a different trajectory.

4.5 Comparison and discussion

In the previous section, it has been proven that both Method I and Method II are feasible options for implementing a state space controller and an observer on an FPGA. However, certain factors such as: total logic elements used, number of clock cycles to complete loop, hardware multipliers employed and etc. should also be taken into consideration, when developing a design for FPGAs. An illustrative presentation of the resource consumptions of these methods on the *Altera Cyclone II* FPGA chip are obtained using the *"Chip planner"* tool of *Quartus II* software and provided in Figs. Figure 4.12 through Figure 4.16.

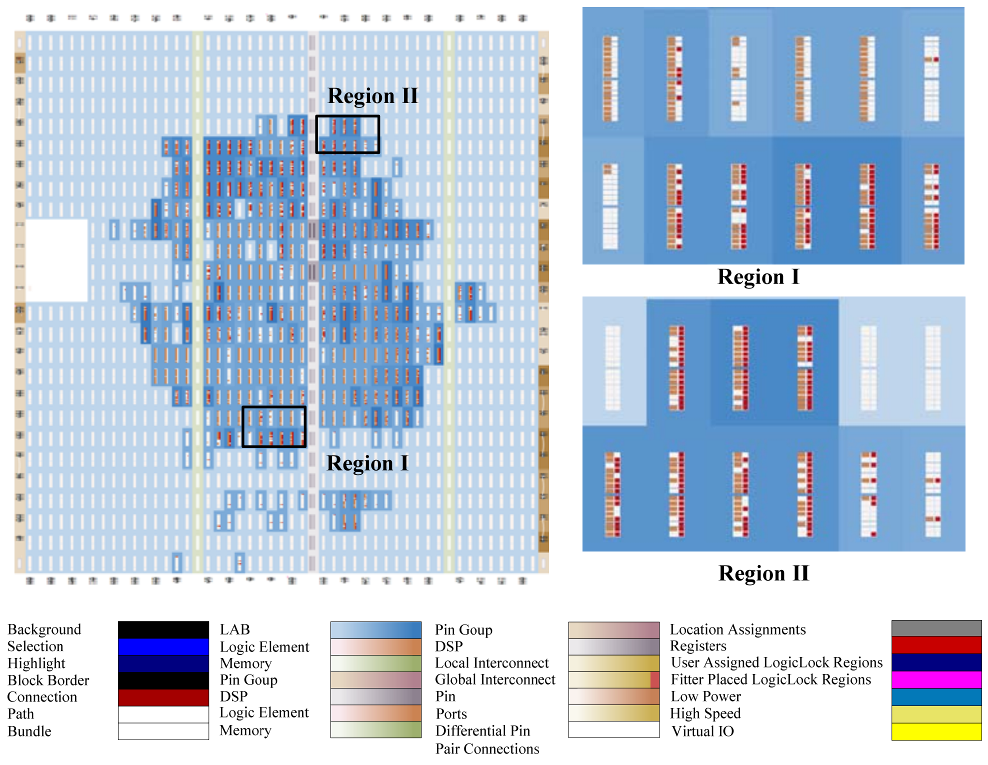


Figure 4.12 – Resource utilization of Method I-a on Cyclone II FPGA

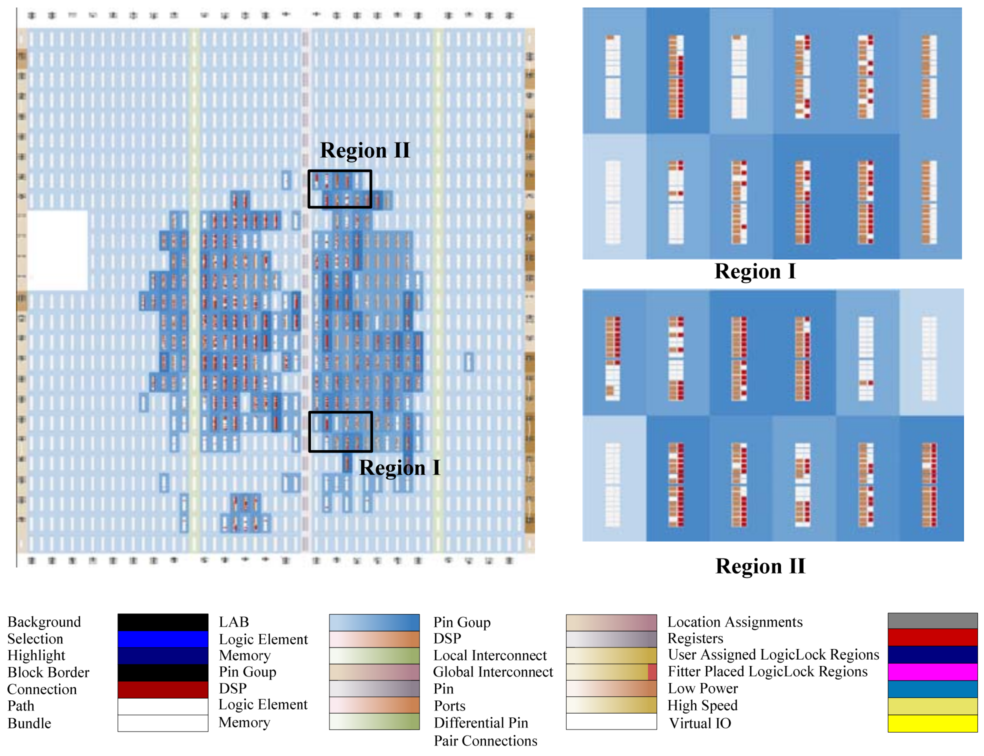


Figure 4.13– Resource utilization of Method I-b on Cyclone II FPGA

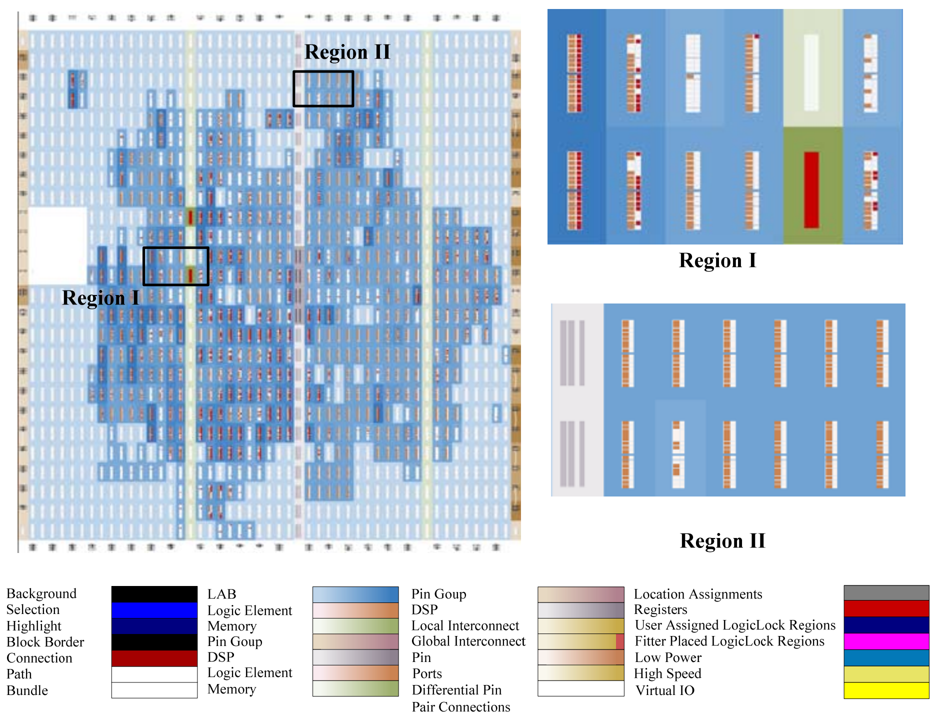


Figure 4.14 – Resource utilization of Method I-c on Cyclone II FPGA

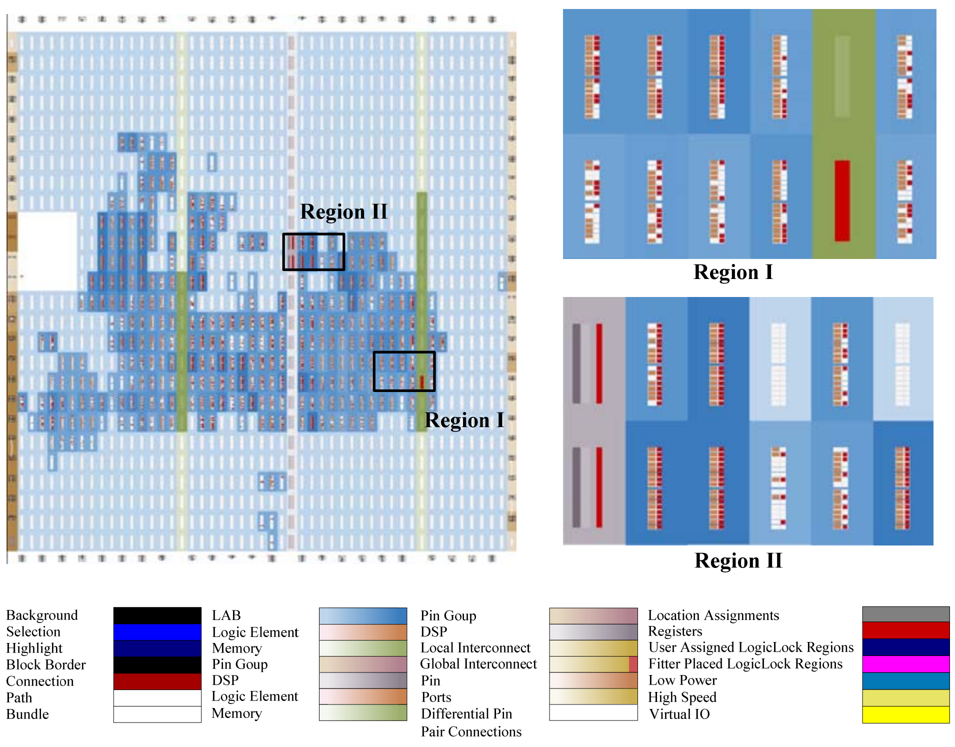


Figure 4.15 – Resource utilization of Methods II-a & II-b on Cyclone II FPGA

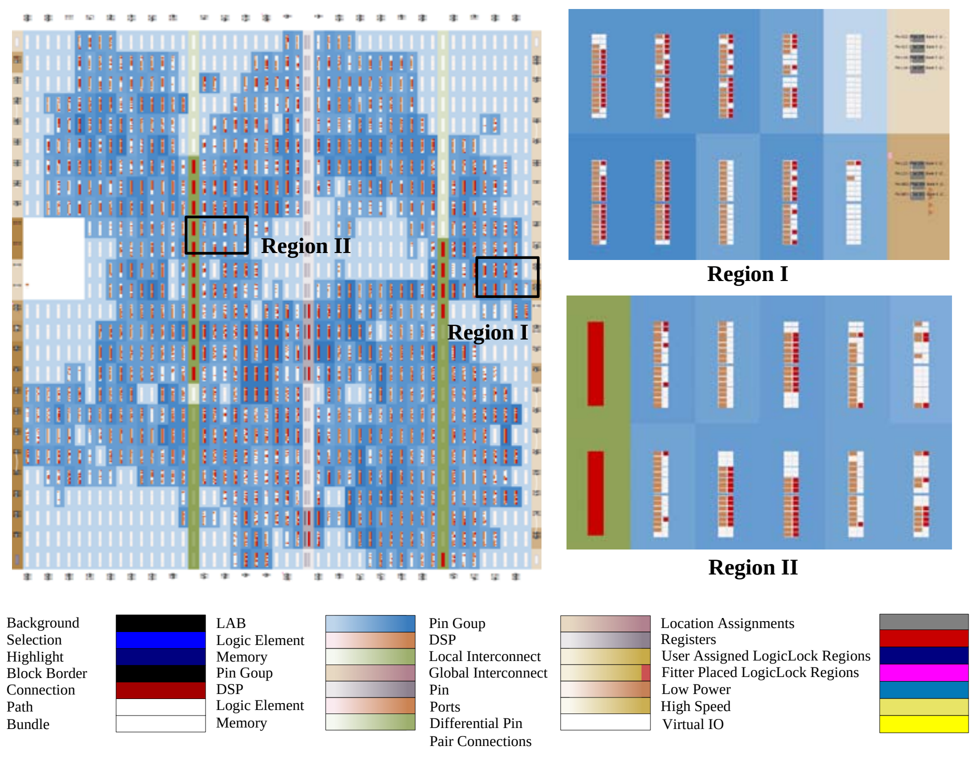


Figure 4.16 – Resource utilization of Method II-c on Cyclone II FPGA

Three illustrations that are provided for Method I-a, I-b and I-c include three different modules employed for different types of computation, which are IMUL, FMUL and FPU modules. On the other hand, in Method II, since a softcore processor is employed, the difference between the illustrations is the difference in the specifications of the processor, which is the floating point arithmetic module. Note that a different module than the FPU module which is used in method I-c. Similarly, resource requirements of each method are summarized in Table 4.3. It is important to notice that resource utilizations (i.e. resource requirements and floor plans) may vary slightly due to the optimization performed by the fitter of the IDE tool provided by the FPGA manufacturer.

Table 4.3 – Resource/Time Costs of Proposed Methods on Altera Cyclone II FPGA chip

Method	Type of Comp.*	Total logic elements	Embedded multipliers	Total clocks required	Max. frequency
I-a	INT	4449 (24%)	4 (8%)	4×32 cycles	400 kHz
I-b	FIXED	3528 (19%)	0 (0%)	32×32 cycles	50 kHz
I-c	FP	7778 (41%)	7 (13%)	20×32 cycles	80 kHz
II-a	INT	4504 (24%)	4 (8%)	3907	12.8 kHz
II-b	FIXED	4504 (24%)	4 (8%)	33185	1.5 kHz
II-c	FP	10595 (54%)	11 (21%)	16051	3.1 kHz

*abbreviations: INT= Integer, FIXED= Fixed point, FP= Floating Point

For method I, clock cycles necessary depend on not only the number of states of the controlled system but also the multiplication algorithm selected. The number of base cycles (N_c) can be expressed as follows:

$$N_c = n(n + 2m + n_y) \tag{4.9}$$

Here, n is the number of states; n_y is the number of measured states and m is the number of manipulated inputs. In this case, since $n = 4$, $n_y = 2$ and $m = 1$; N_c becomes 32 (as indicated in Table 4.3). The duration of each base cycle depends on the design of the multiplier unit.

As can be seen in Table 4.3, the proposed method I increases the attainable sampling rate of the controller significantly. As discussed earlier, this is due to the custom designed and application specific modules. It is also evident that the type of computation also has an effect on the attainable sampling rate, however the slowest frequency attainable by method I is still 3 times faster than the fastest frequency attainable by method II.

It is critical to note that since the speed of the serial communication (RS-232) used in this study is not sufficient for real-time simulation, a non-real time HILS is realized. For the simulation, a sampling frequency of 1 kHz is selected and thus the discrete-time state space representations are evaluated using Eqns. 4.7-4.8.

In terms of resource requirements, it is clear that when floating point operations are involved, resource requirements of the design increase drastically. This is proven by both of the methods with an increase of 3 to 6 thousands of logic elements and 3 to 11 embedded multipliers. While the most resource-wise efficient method seems to be method I-b, all the methods that are not employing floating point arithmetic (method I-a, method II-a and method II-b) are also comparable. This shows that for modules with high complexity, bulk implementation of the controller does not increase the resource requirement significantly, since the customized module would also require a significant amount of FPGA resources. In conclusion, for a mediocre FPGA chip such as *Altera Cyclone II*, utilizing floating point arithmetic is a costly method; especially when it is proven that good control performances can be attained by exploiting certain properties of motion control systems (such as quantized input/output property) as explained in section 4.2, in order to reduce the resource requirements of the design. It is also proven that customized HDL modules are more advantageous in terms of speed, rather than resource utilization, especially in complex tasks. However, complexity of the task is also a vague description and it is necessary to implement both methods to see which one would be more resource-wise efficient.

4.6 Closure

In this chapter, a state space controller with a state observer is designed and implemented for the inverted pendulum problem. In section 4.2, 5 different methods for computation of the necessary calculations are explained thoroughly under 2 different design methods. The HILS results suggest that while the modular approach presented as Method I offers a slight increase in terms of resource requirements, it is definitely faster than the softcore processor method that is presented as Method II.

CHAPTER 5

IMPLEMENTATION OF DIGITAL FILTERS

Generally, one or more filters are usually employed in a typical motion control system. In digital motion control applications, these filters are realized by digital filtering algorithms and can be implemented on various signal processing chips such as: DSPs, ASICs, FPGAs and etc. In this chapter, after a brief overview of digital filters that are commonly used in motion control applications, two different methods (one incorporating the previously mentioned multiplication methods and the other utilizing the *Nios II* softcore processor) for FPGA implementation of a general infinite impulse response (IIR) filter is presented, along with hardware-in-the-loop simulations (HILS) of a cascade control architecture realized by the two alternative methods. The chapter is finalized by a quantitative comparison between the two design methodologies.

5.1 Digital Filters

In this section, a brief overview of digital filters is presented, within a limited scope of digital motion control applications. A digital filter is commonly employed to modify certain aspects of a signal and can be classified into two categories, based on its impulse response or in particular, its feedback property. A finite impulse response (FIR) filter is a digital filter that generates its output signal by using only current and past values of its input. Therefore, it has no internal feedback and the impulse response dies out to zero, as the name implies. On the other hand, an infinite impulse response (IIR) filter contains an internal feedback and therefore its output depends on its past output values as well as past input values. The constant coefficient difference equation of an IIR filter is provided in (5.1).

It is more convenient to discuss the implementation of an IIR filter rather than the FIR, since it is more general and can also be modified to obtain an FIR, as can be observed in (5.1). Therefore, the subsequent discussions are based on the IIR filter.

$$y(k) = \sum_{i=0}^{n} b_i x(k-i) + \sum_{j=0}^{m} a_i y(k-j) \tag{5.1}$$

Using the generalized expression of IIR, it is possible to deduce that the implementation requires multiplication of the filter coefficients with past outputs and past inputs, followed by an addition operation, which is a very similar case to the state-space controller implementation that is presented in the previous chapter. Therefore, it is possible to realize an IIR filter (hence also an FIR filter) on an FPGA, utilizing the previously described methods.

Most of the filters such as: Notch filter, Low-pass filter and High-pass filter and etc. that are commonly used in motion control systems are all IIR filters and can be realized by a general IIR filter module. Furthermore, difference equation of a PID controller, as can be seen in (5.2):

$$m(k) = m(k-1) + b_0 e(k) + b_1 e(k-1) + b_2 e(k-2) \qquad (5.2)$$

where m is the manipulated input, e is the error and k is the time step, is also a variation of (5.1) and can be implemented by the same module. Therefore it is desirable to develop a generic IIR filter module for FPGAs, in order to realize certain controller topologies, as well as digital filters. Implementation details of digital filters are explained in the following section.

5.2 Implementation

In this section, two alternative methods for implementation of digital filters on FPGAs are presented. The first method is development of a generic filter module utilizing a custom multiplication module, which is presented in the previous chapter. The second method presents the implementation of a softcore processor, Nios II and requires a significantly different design approach for the problem, as explained in the previous chapter.

5.2.1 Method I: Generic filter module

Generic filter module is analogous to the matrix multiplier module presented in the previous chapter. Similar to the previous case, this module utilizes a custom multiplication module (which is selected as IMUL in this case) and an adder, in order to calculate the result of (5.1). Figure 5.1 presents a schematic representation of the generic filter module. Here x represents input at the current time step x(k), y represents the output at the current time step y(k), x_n and y_m represents the input at the (k-n)th time step and output at the (k-m)th time step, that are x(k-n) and y(k-m) respectively.

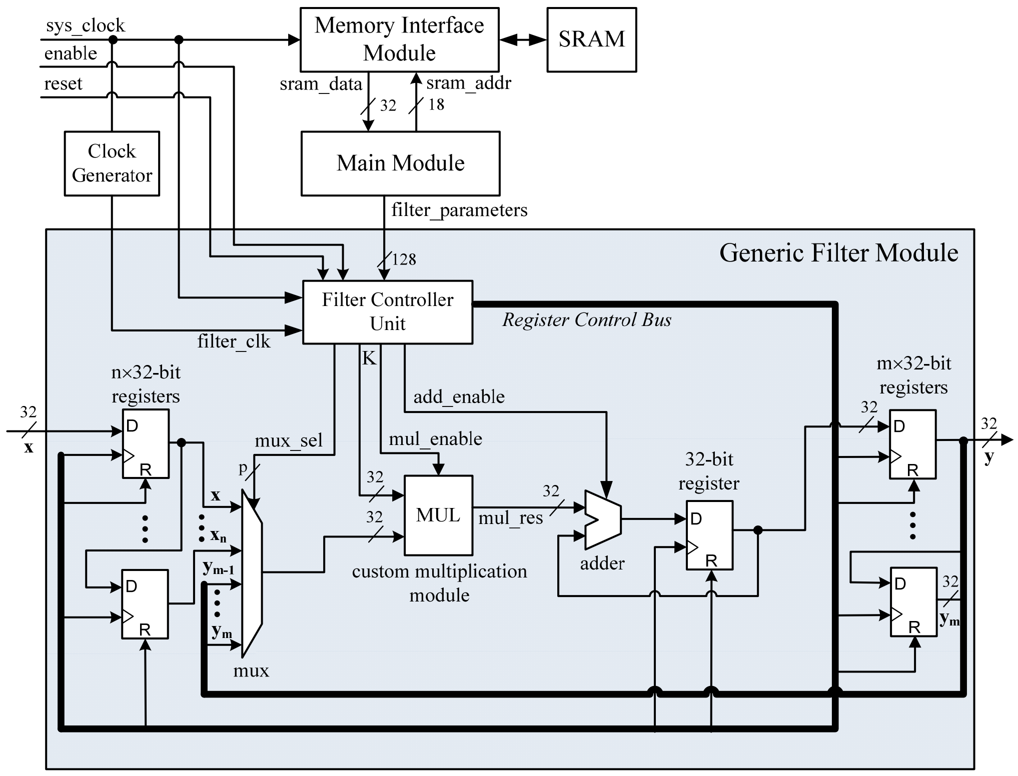


Figure 5.1 – Generic filter module

There are only two design parameters to be selected when utilizing this generic filter module; that are n (the number of past input values) and m (the number of past output values). While these parameters can be selected as relatively large integers, in order to avoid further modifications to the HDL code, it is also possible to select those parameters as required minimum values as required for a certain filter, to minimize resource requirements of the module.

5.2.2 Method II: Softcore processor

Softcore processors are embedded processor IPs developed by FPGA manufacturers, in order to decrease the long and tedious design periods required by low-level (i.e. logic-level) circuit design through hardware description languages (HDL). While shorter and easier design periods are favorable characteristics, the flexibility of the design inevitably decreases due to the bulk implementation of the processor and it may not always be a resource-wise and time-wise beneficial method.

The design approach of a softcore processor is significantly different from the modular approach used in state-space controller implementation and filter implementation via generic filter module. The difference is that, in modular design approach, all the modules are designed with HDLs or schematic design tools and all the connections (data buses, clock signals, enable/reset signals and etc.) between the modules are defined and implemented by the designer. On the other hand, utilizing a softcore processor is via a user-friendly GUI of the design tool provided by the FPGA manufacturer and all the parameters of the processor can be selected easily using this tool. Furthermore, all the peripheral controllers (such as memory management units, serial controllers, GPIOs and etc.) are also provided in this tool, making the overall design process a lot easier and faster.

After implementing the embedded processor along with peripheral controllers to the FPGA, it is possible to develop a C code to realize any algorithm that is supported by the specifications of the implemented processor. In this design method, FPGA implementation of the algorithm is handled by the compiler and therefore the development process is much faster than the modular design approach. Note that, this is one of the most significant differences between the two methods.

Implementation details of the second method is as follows; first the specifications of the processor is selected and peripheral controllers such as SRAM and UART controller are added via SOPC (system on programmable chip) builder tool of *Altera Quartus II*. After the embedded processor is implemented to the FPGA, Nios II IDE is utilized to develop and implement the C code into the processor. Note that the resource requirements of the design does not depend on the algorithm but only the specifications of the processor and peripheral control units. However, execution time of the process inevitably depends on the algorithm. Results of the hardware-in-the-loop simulations are provided in the following section.

5.3 Hardware-in-the-loop simulation results

Hardware-in-the-loop simulation is realized by the inverted pendulum system presented in the previous chapter. In order to demonstrate the performance of the developed methods, a cascade controller utilizing PID controllers as explained in section 5.1 is implemented. The cascade control system consists of a PD controller in the outer loop for control of the position of the cart and a PID controller in the inner loop for control of the angular position of the pendulum.

Figure 5.2 – Cascaded control system illustrates the control system.

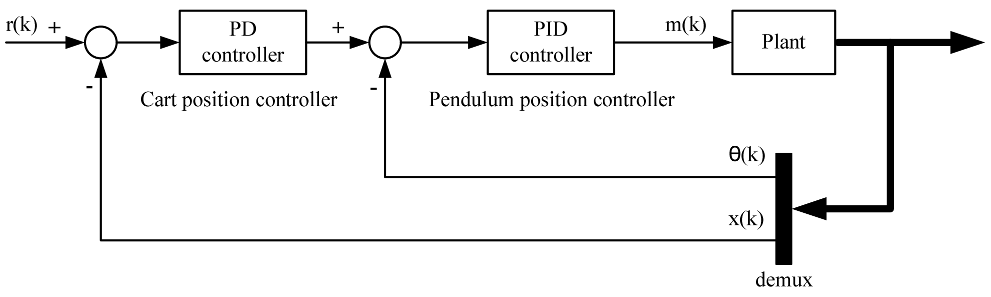


Figure 5.2 – Cascaded control system

In Figure 5.2, r is the reference input, m is the manipulated input, θ is the angular position of the pendulum and x is the linear position of the cart. As can be seen in Figure 5.2, two "generic filter module" instances are required to realize the cascade controller system. On the other hand, resource requirement of the softcore processor doesn't depend on the number of instances required, just as it would be the case in a regular microprocessor.

Results of the HILS using the generic filter module (Method I) with imul and FPU are shown in Figure 5.3 and 5.4 respectively. Results of the HILS using the Nios II softcore processor (Method II) are shown in Figure 5.5 and Figure 5.6 respectively. Note that the initial conditions for all simulations are selected as $x_0 = 0$ m and $\theta_0 = -0.05$ rad.

As can be observed from Figure 5.3, Figure 5.4, Figure 5.5 and Figure 5.6, the cascade control system is able to hold the inverted pendulum in an upward position while the position of the cart is bounded and settling to zero. Furthermore, it can be seen that there is no significant difference between the figures as expected, since the implemented controllers are the same even if the methodologies are drastically different. Having obtained these results, it is possible to evaluate and compare these methodologies in terms of their resource requirement, execution time and ease of implementation; which will be the subject of the following section.

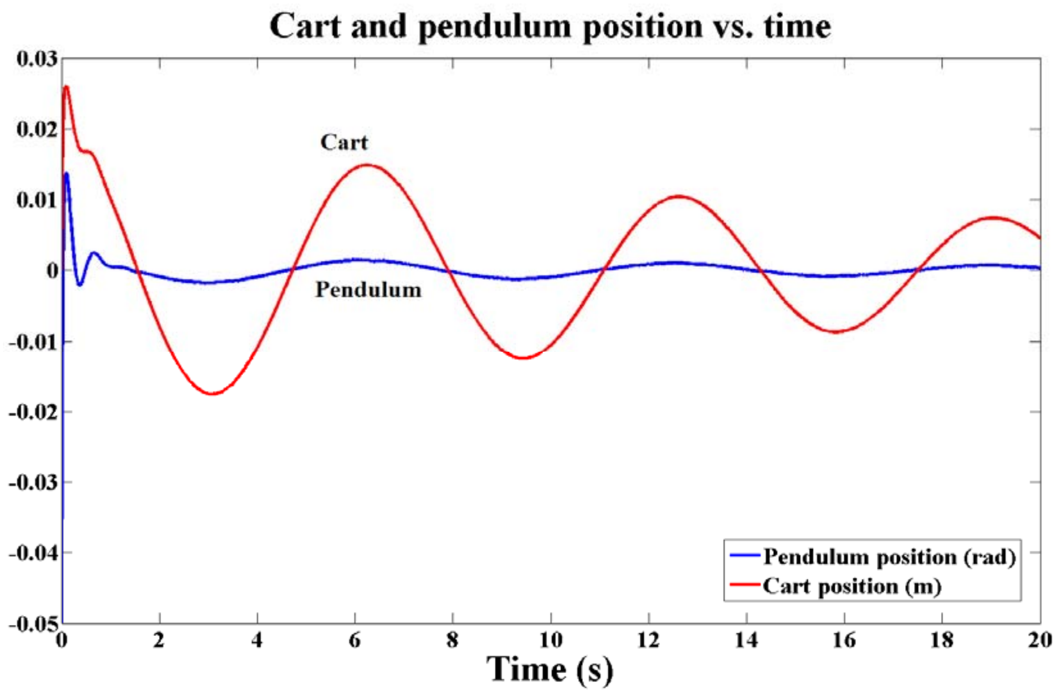


Figure 5.3 – HILS result of method I with IMUL

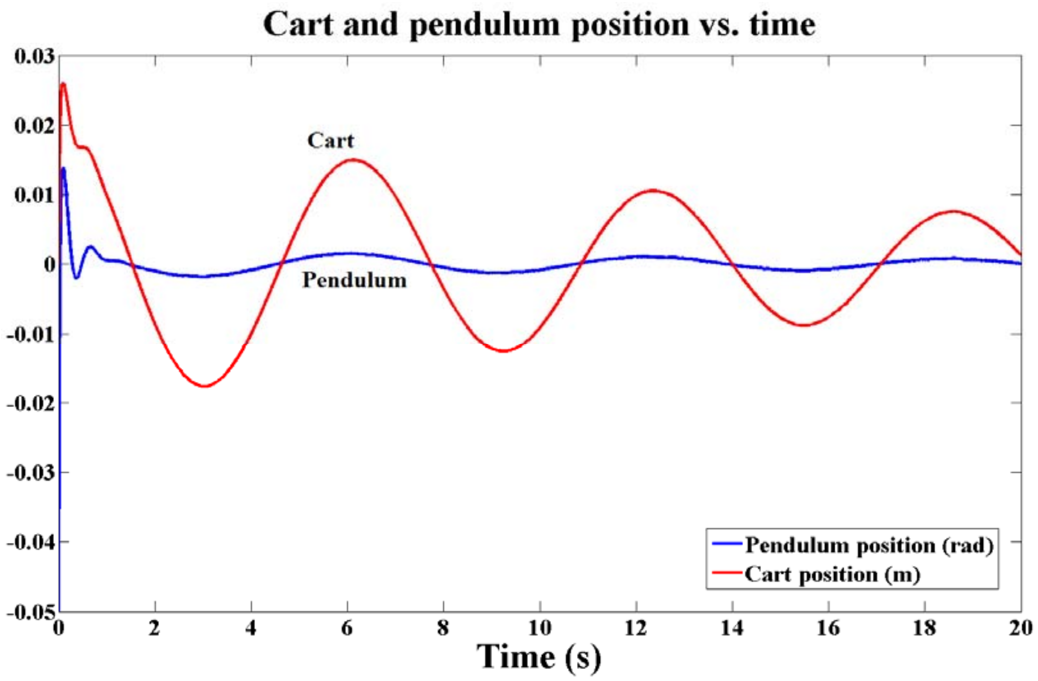


Figure 5.4 – HILS result of method I with FPU

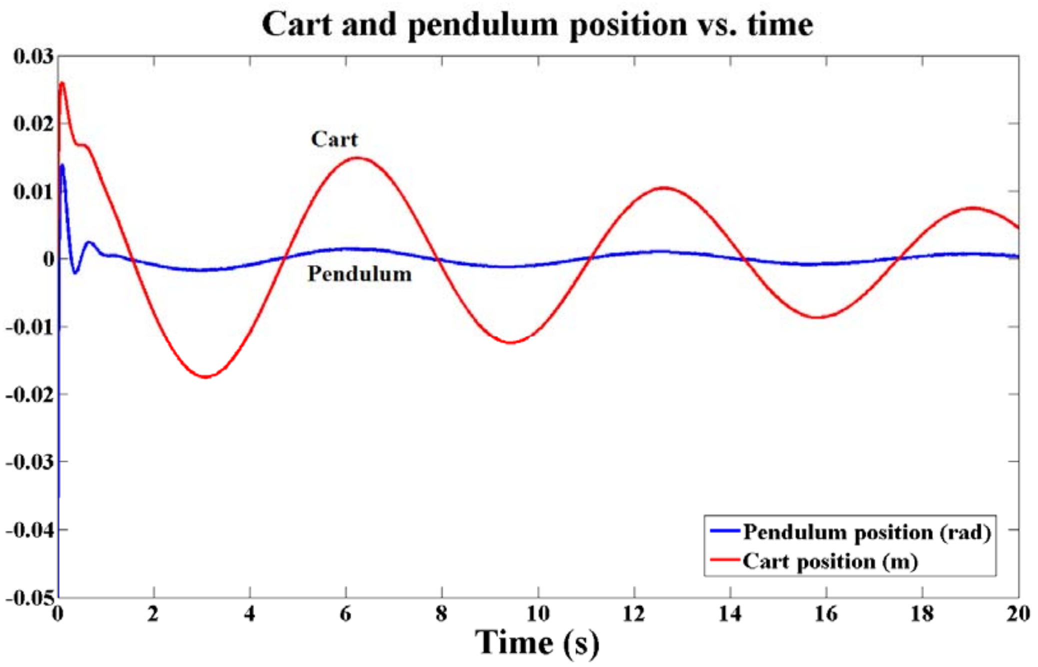


Figure 5.5 – HILS result of method II with IMUL

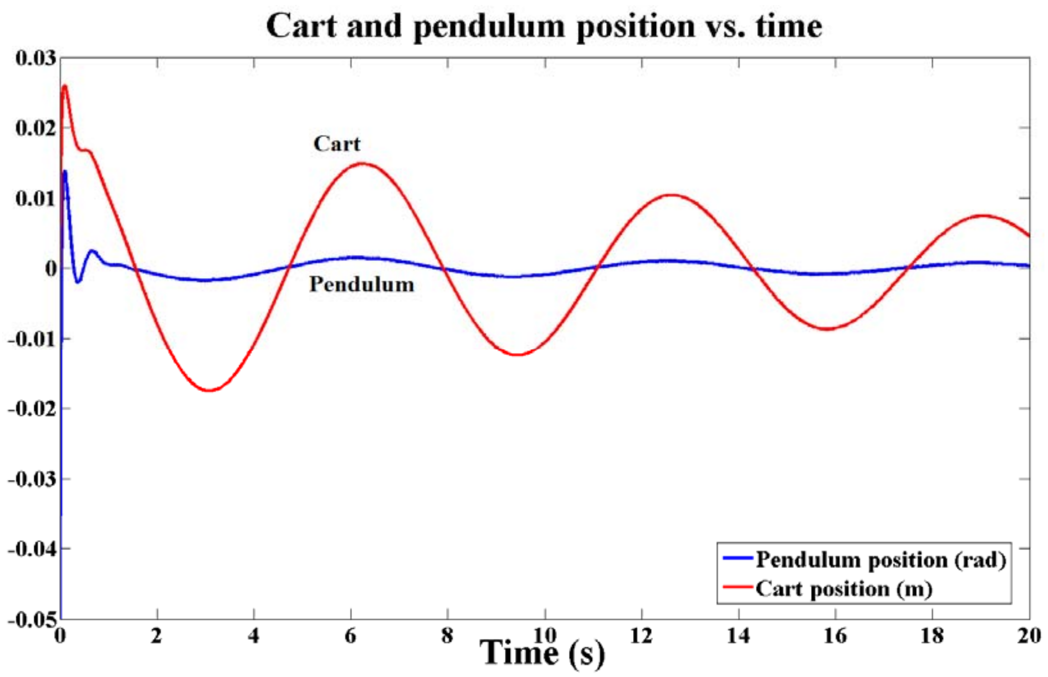


Figure 5.6 – HILS result of method II with FPU

5.4 Comparison and discussion

In the previous section, it has been shown that both method I and method II are feasible options for implementing digital filters on an FPGA. However, certain factors such as: total logic elements used, number of clock cycles to complete loop, hardware multipliers employed and etc. should also be taken into consideration, when developing a design for FPGAs. An illustrative presentation of the resource consumptions of these methods on the *Altera Cyclone II* FPGA chip are obtained using the "*Chip planner*" tool of *Quartus II* software and provided in Figure 5.7, Figure 5.8, Figure 5.9 and Figure 5.10.

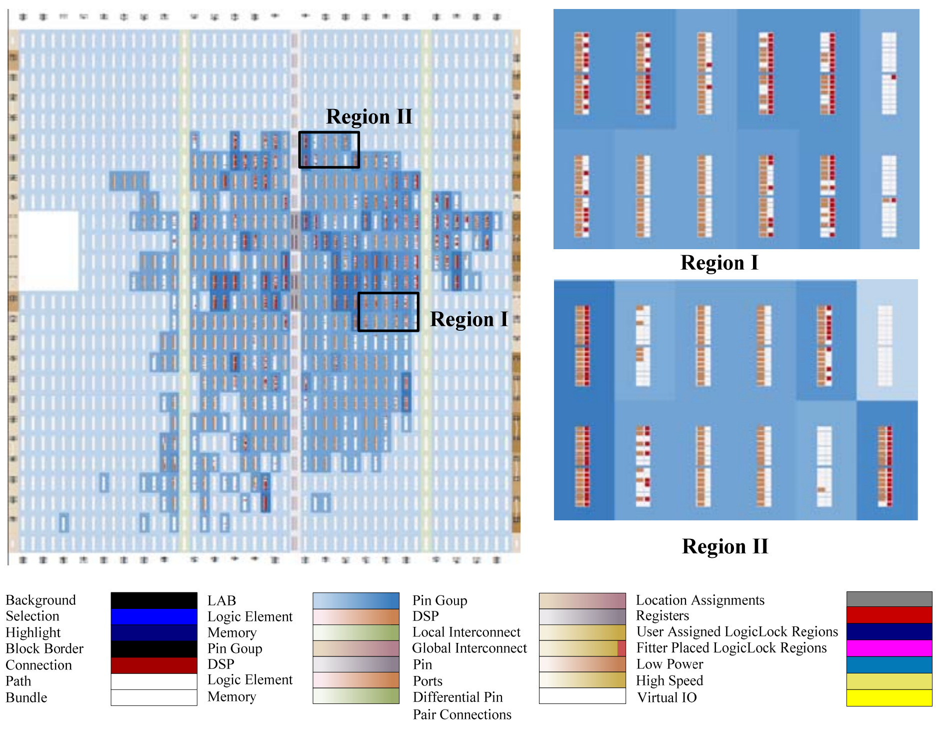


Figure 5.7 – Resource consumption of Method I with IMUL

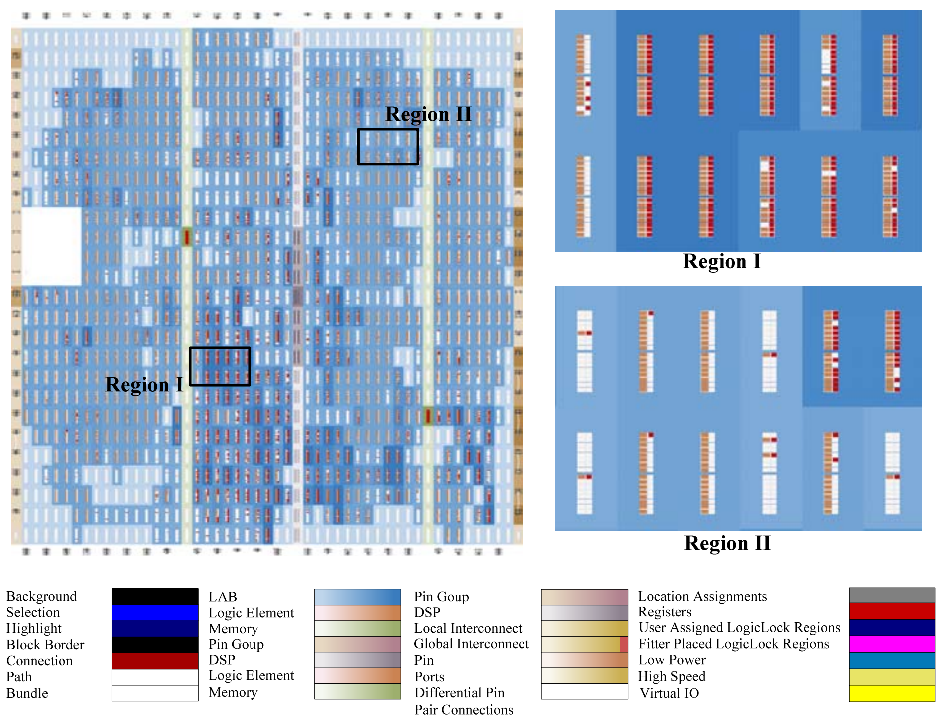


Figure 5.8 – Resource consumption of Method I with FPU

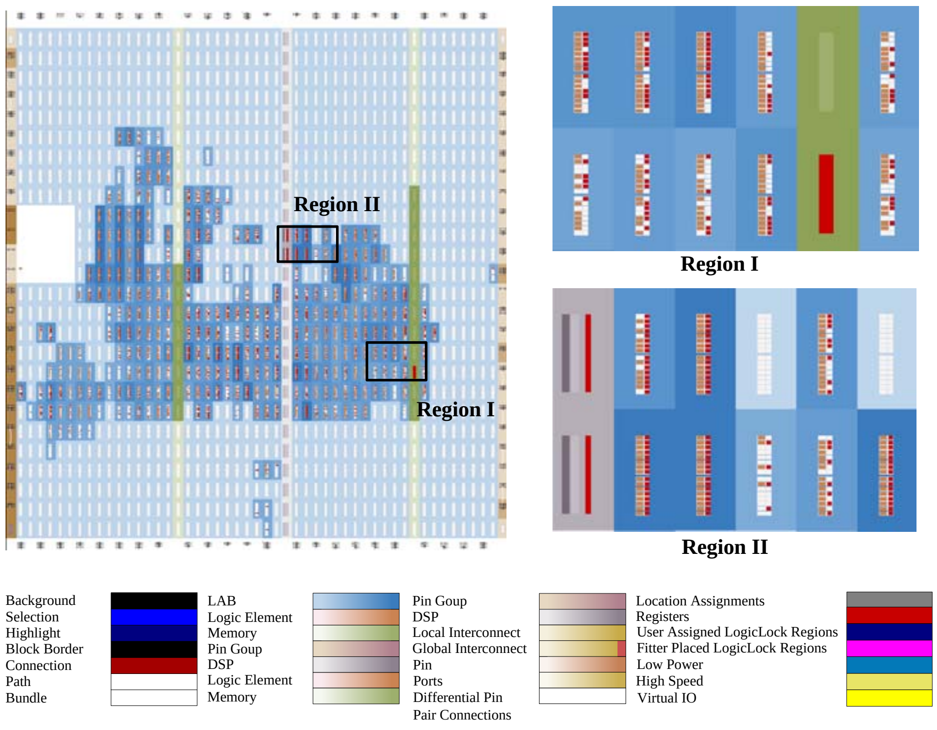


Figure 5.9 – Resource consumption of Method II with integer arithmetic

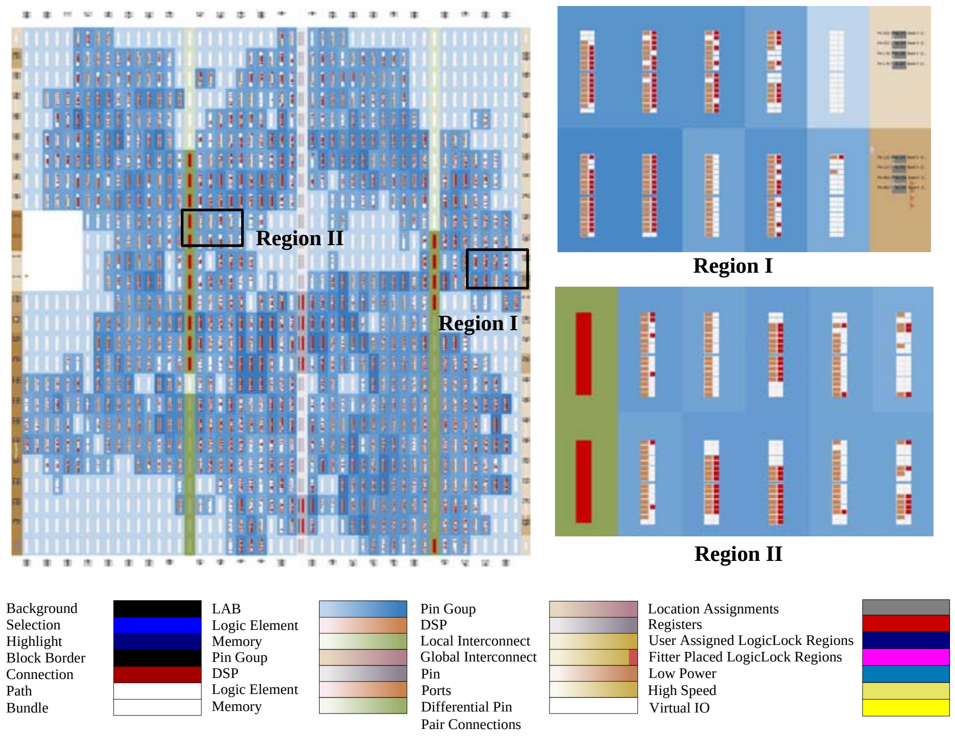


Figure 5.10 – Resource consumption of Method II with FPU

Two illustrations provided for Method I include two different modules employed for two different types of computation, which are IMUL and FPU modules. On the other hand, in Method II, since a softcore processor is employed, the difference between the illustrations is the difference in the specifications of the processor, which is the floating point arithmetic module. Note that this module is not the same module which is used in the generic filter module. The resource/time costs of the proposed methods are tabulated in Table 5.1.

TABLE 5.1 Resource/Time Costs of Proposed Methods on Altera Cyclone II FPGA chip

Method	**Type of Comp.***	**Total logic elements**	**Embedded multipliers**	**Total clock cycles required**	**Max. frequency**
I	INT	5012 (27%)	8 (15%)	4×4 cycles	3 MHz
I	FP	11180 (60%)	14 (27%)	20×4 cycles	625 kHz
II	INT	4504 (24%)	4 (8%)	400	125 kHz
II	FP	10595 (54%)	11 (21%)	3270	15.3 kHz

*abbreviations: INT= Integer, FP= Floating Point

As shown in Table 5.1, in terms of logic elements, resource requirements of both methods are very close. On the other hand, embedded multiplier requirement of the first method (modular approach) is double of the second, owing to the fact that two instances of the generic filter module is required to implement two filters (i.e. it is not possible to use same instance twice for different filters), since past values need to be stored. On the other hand, softcore processor handles the design with 4 embedded multipliers, which is an advantage on its part.

In terms of total clock cycles, method I has a significant advantage over method II, due to the optimizable modular design approach, as expected. It is not possible that this advantage arises due to the increased number of embedded multipliers that are employed, because even if the total clock cycles required by the first method is doubled, it is still significantly faster than the embedded processor. The total clock cycles required can be determined based on the number of multiplications in the filter. In this case, since 4 multiplications are required, total clock cycles required is obtained by multiplying the clock cycles needed by the custom multiplication module by 4.

Observing the simulation results and the resource/time properties of two methods, it can be seen that both methods are convenient options for digital filter implementations on FPGAs. It should be noted that, when the complexity of the design increases (i.e. the number of filters and etc.), the significant increase will be in terms of resource requirements for Method I, since new instances of the generic filter module will be required; on the other hand, for Method II, the significant increase will be in terms of required clock cycles, since the processor is already implemented and the change will only be in the algorithm. Therefore, it can be concluded that Method I -modular design approach- seems more suitable for demanding real time designs where speed is more crucial and Method II -softcore processor- seems more suitable if the design is very complex or a fast and efficient development period is desired.

5.5 Closure

In this chapter, different methodologies for digital filter implementation on FPGAs are presented. These methods are utilized by realizing a cascaded control system to solve a nonlinear inverted pendulum control problem. Furthermore, resource utilization and speed performances are also provided for these methodologies.

CHAPTER 6

ADVANCED CONTROLLERS

This chapter presents a study for some advanced real-time motion controller topologies implemented on the field programmable gate array (FPGA); which are selected as a sliding mode controller and a fuzzy controller. In this chapter, all the aforementioned methods are not implemented and a single implementation method (Method I-c) presented in Chapter 4 is adopted. In the context of Method I-c, controllers are developed using Verilog HDL (i.e. modular hardwired approach is adopted) and an open-core hardwired floating point unit is implemented for the complex calculations.

Unlike other chapters of this study, in this chapter the controllers are implemented on a ML505 development board with a Xilinx Virtex-5 FPGA chip (in other chapters a DE1 board with an Altera Cyclone II FPGA is employed). The tests are performed on a hardware-in-the-loop simulation of a field-oriented induction motor system, which is a similar system to the case provided in Chapter 7, however in this case speed control problem of a CNC turning center is considered.

Finally, simulation results are provided for the test case and implementation results are provided. The controller topologies are evaluated by certain criteria including resource cost (total memory space and logic unit requirement), attainable sampling rate and the success of the controller.

6.1 Controller Topologies

Controller topologies are selected from common advanced controller topologies, considering the suitability of the topology for FPGA implementation.

6.1.1 Fuzzy Controller

Fuzzy control is an intelligent control topology based on fuzzy set theory. It has been applied to many control applications including the control of drives [26]-[27]. Typical method for fuzzy control application in discrete-time control is to calculate error and change in error in each sampling time, then to define a linguistic representation of the error and change in error based on membership functions. As a final step, these linguistic representations having fuzzy memberships go through a defuzzification process to generate a manipulation signal based on a fuzzy rule base. A schematic for the implementation of the fuzzy controller on the FPGA is presented in Figure 6.1.

Membership functions are presented in Figure 6.2. Here, NB, NS, Z, PB, and PS correspond to “Negative big,” “Negative small,” “Zero,” “Positive big,” and “Positive small” respectively. Notice that the functions are formed by the pulse error per sampling time period. Therefore, the data from the encoder can be directly used without further calculations while the fuzzy memberships can be represented as integers.

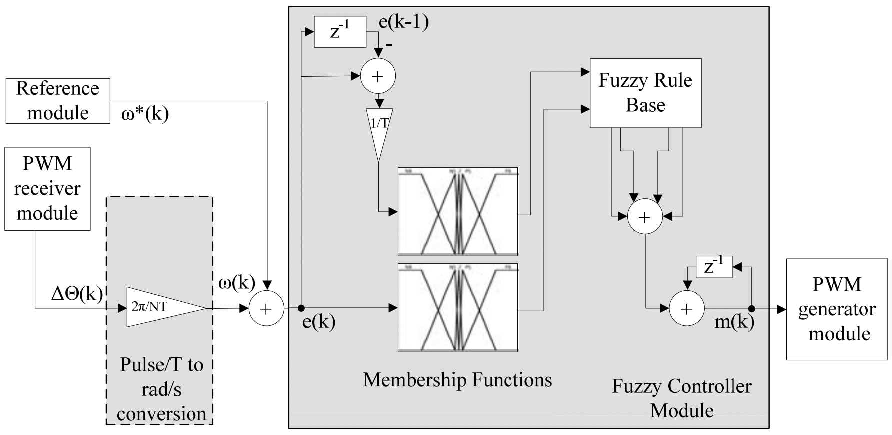


Figure 6.1 – Fuzzy controller implementation

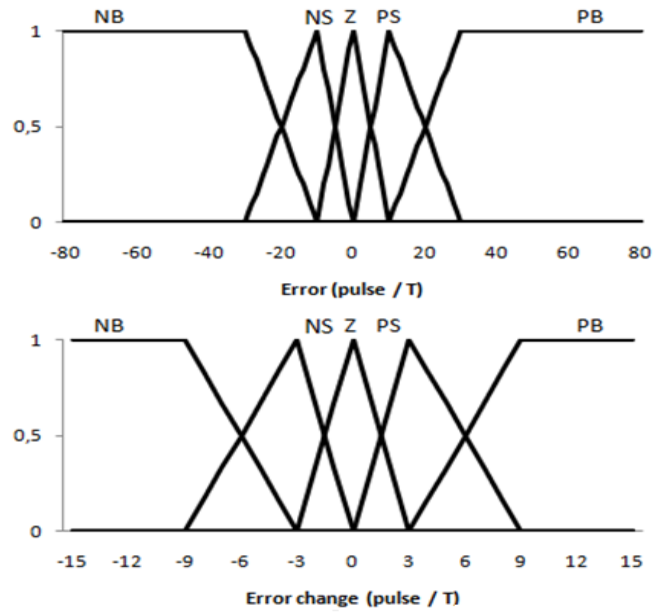


Figure 6.2 – Membership functions of the fuzzy controller

The defuzzification process depends on the fuzzy rule base presented on Table 6.1. It can be observed that the membership functions are formed based on pulse error per sampling time period. Therefore, the data from the encoder can be directly used without further calculations and the fuzzy memberships can be represented in integer forms. The defuzzification process is based on the fuzzy rule base presented on Table 6.1. Fuzzy membership functions and the rule base are selected based on the common experience in the literature and after a trial and error process.

Table 6.1 – Membership functions of the fuzzy controller

error / Δ error	NB	NS	Z	PS	PB
NB	NB	NB	NS	NS	Z
NS	NB	NS	NS	Z	PS
Z	NS	NS	Z	PS	PS
PS	NS	Z	PS	PS	PB
PB	Z	PS	PS	PB	PB

6.1.2 Sliding Mode Controller

Sliding mode control is a robust control method developed to deal with model uncertainties and unknown parameters in the expense of high computational cost [28]. Its applications include position/speed control of servo/induction motor drives [29]-[30]. Sliding mode controller is based on a control law with varying control structures. The basic idea is to force the trajectory of the system state to a sliding surface through switching of the control structures. The most general form of the sliding surface is

$$s = \dot{e} + \lambda e \qquad (6.1)$$

where s is the sliding surface; e is the error of the controlled state and λ is a controller parameter. After calculating the sliding surface, an equivalent control term needs to be calculated such that applying equivalent control would make $\dot{s} = 0$. Hence, the sliding mode control law takes the form:

$$u = u_{eq} + Ksgn(s) \tag{6.2}$$

where u_{eq} is the equivalent control and u is control output. The implementation of the sliding mode controller is presented in Figure 6.3

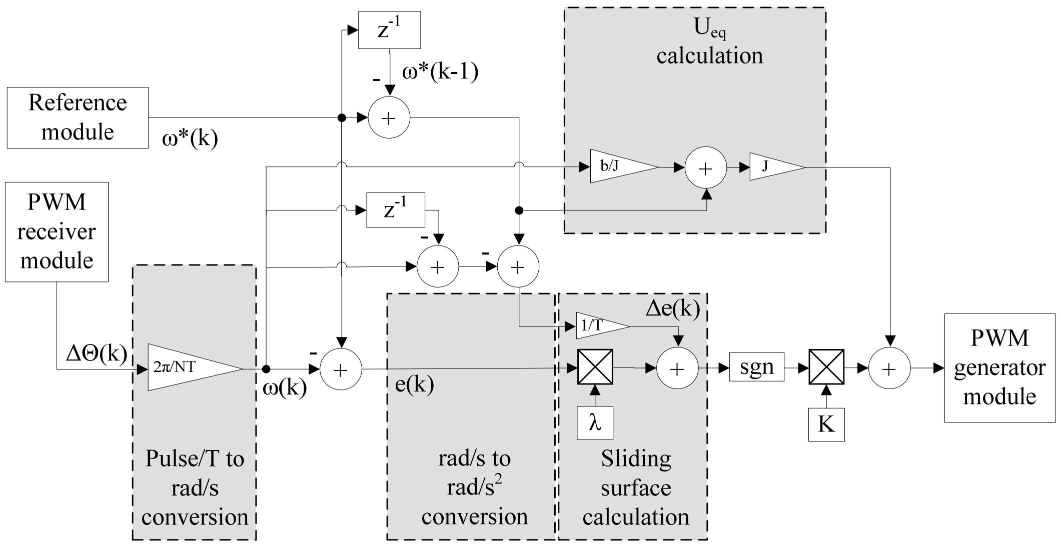


Figure 6.3 – Sliding mode controller implementation

The sliding mode controller parameters are selected as λ = 10000 and K = 15. These parameters are selected based on the disturbance rejection characteristics of the controller and the maximum admissible torque based on the rated torque of the motor.

6.2 Test Setup

The control of CNC turning centers is considered for performance evaluation of the above mentioned controller topologies. In CNC turning centers, the spindle housing the workpiece is the key component of the machine where a constant speed is required during most machining operations. Therefore, to sustain constant speed, the controller must effectively reject the disturbance torque observed in turning operations. Figure 6.4 illustrates the simplified model of a typical turning center. In this system, a field-oriented induction motor, which is further elaborated in [31], is employed. Induction motor parameters are as follows: motor power: 5.5kW; rated motor torque (T_r): 35Nm; rated speed (ω_r): 1500rpm; maximum speed (ω_p): 8000 rpm. It is critical to note that the presented system is realized via a hardware-in-the-loop simulation (HILS) performed on the Altera DE1 FPGA development board. For the sake of implementation, certain simplifications (ideal DTC motor drive, ideal timing belt) are made on the system. Figure 6.5 shows the block diagram of the resulting system.

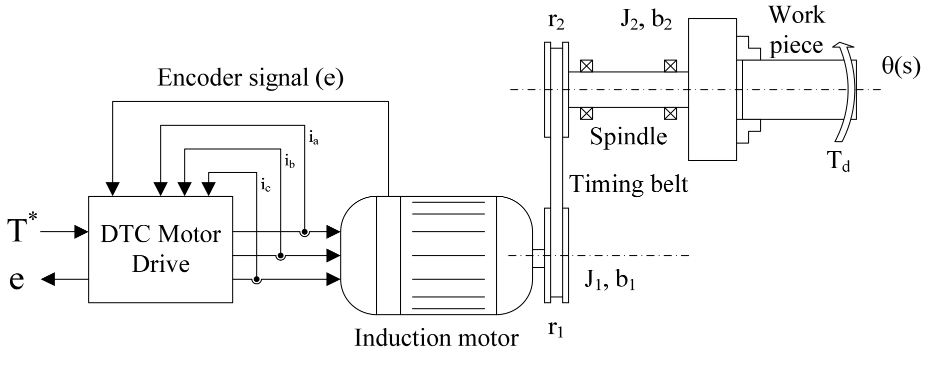


Figure 6.4 – Simplified model of a typical turning center

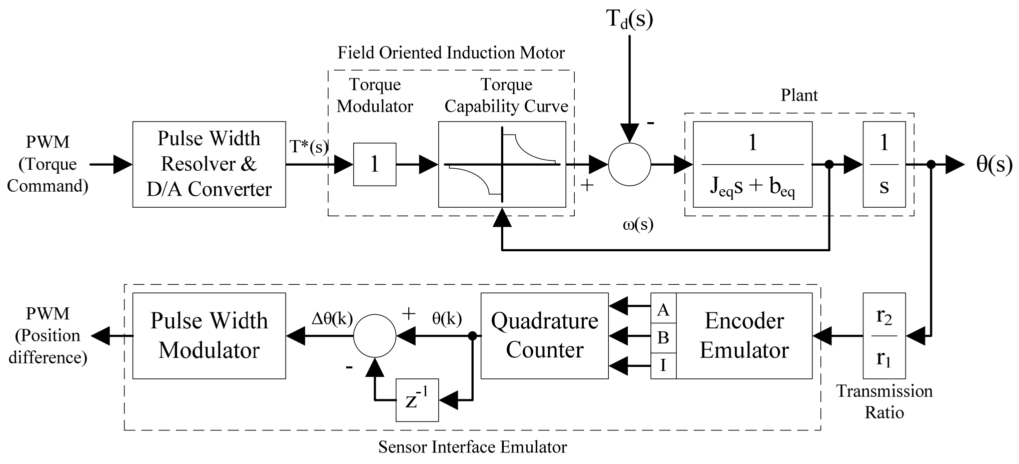
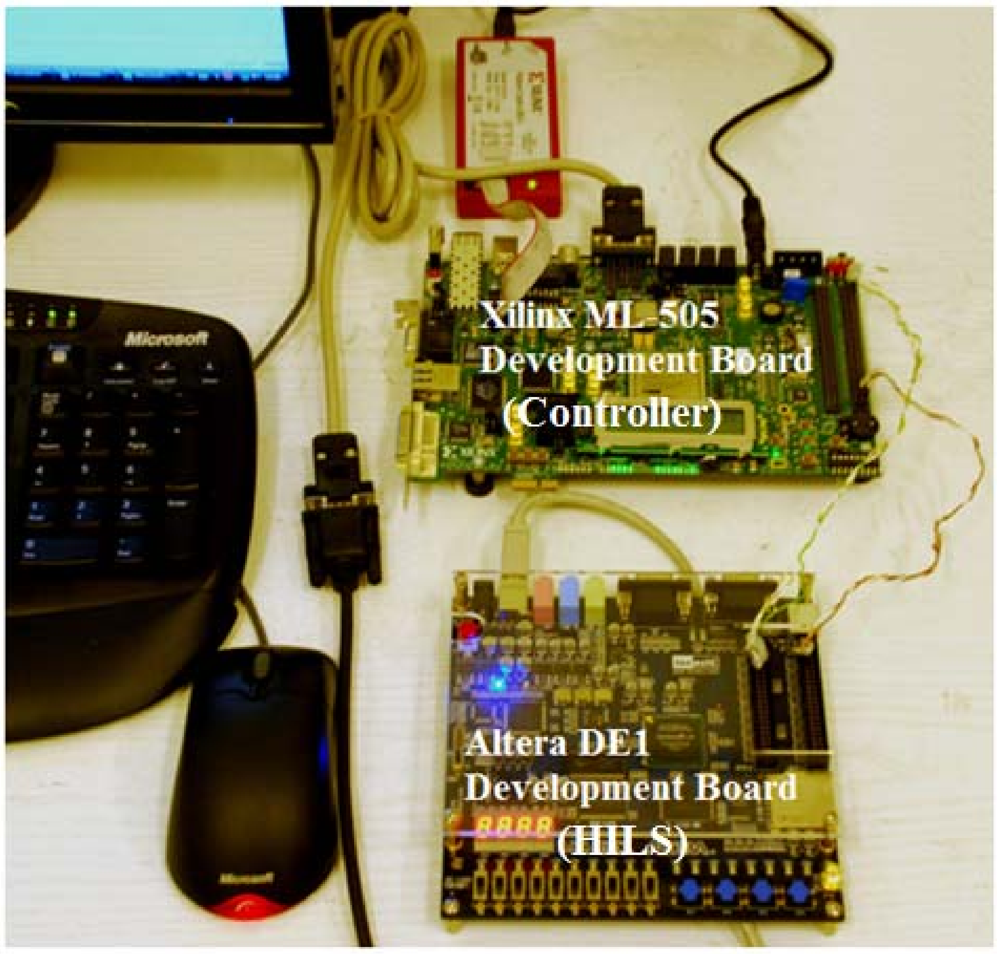


Figure 6.5 – Simplified model of the system



Xilinx ML-505 Development Board (Controller)
Altera DE1 Development Board (HILS)

Figure 6.6 – Test setup

As can be seen in Figure 6.5, the communication with the simulator is accomplished via PWM signals. A 10-bit resolution is selected for the PWM signal to represent torque command and position feedback. In order to prevent problems that could arise from insufficient resolution of the PWM signal, the position difference is transmitted from the HILS setup. Note that the real setup, which is shown in Figure 6.6, consists of two FPGA development boards. One of them is the Xilinx ML-505 development board on which the controller modules are implemented. Likewise, an Altera DE1 development board performs the HILS. Boards are connected through I/O pins, facilitating the PWM connection.

6.3 Results

In this study, different controller topologies are designed and implemented on an FPGA chip. In this section, the results of the HILS are presented and controller topologies are evaluated by their success (tracking performance, disturbance rejection) and their resource cost on an FPGA chip.

The reference input to the system is given as position difference between time samples and a constant 10 Nm torque is applied at t = 1s to simulate the interrupted machining operation. Responses of the system with different controllers are presented in Figure 6.7 and Figure 6.8 highlights the disturbance rejection characteristics of each controller. As can be observed, the controllers achieve similar performances until the disturbance kicks in at 1s. From this point on, the best performance is achieved by the sliding mode controller and while the system under fuzzy control is affected slightly, the controller acts fast enough to compensate the disturbance. Note that from the standpoint of performance, these controllers are comparable.

Table 6.2 – Resource costs of different controller topologies on the Xilinx Virtex-5 FGPA

	Slice LUTs	Slice Registers
Fuzzy	12863 (44%)	2448 (8%)
Sliding mode	4192 (14%)	1675 (5%)

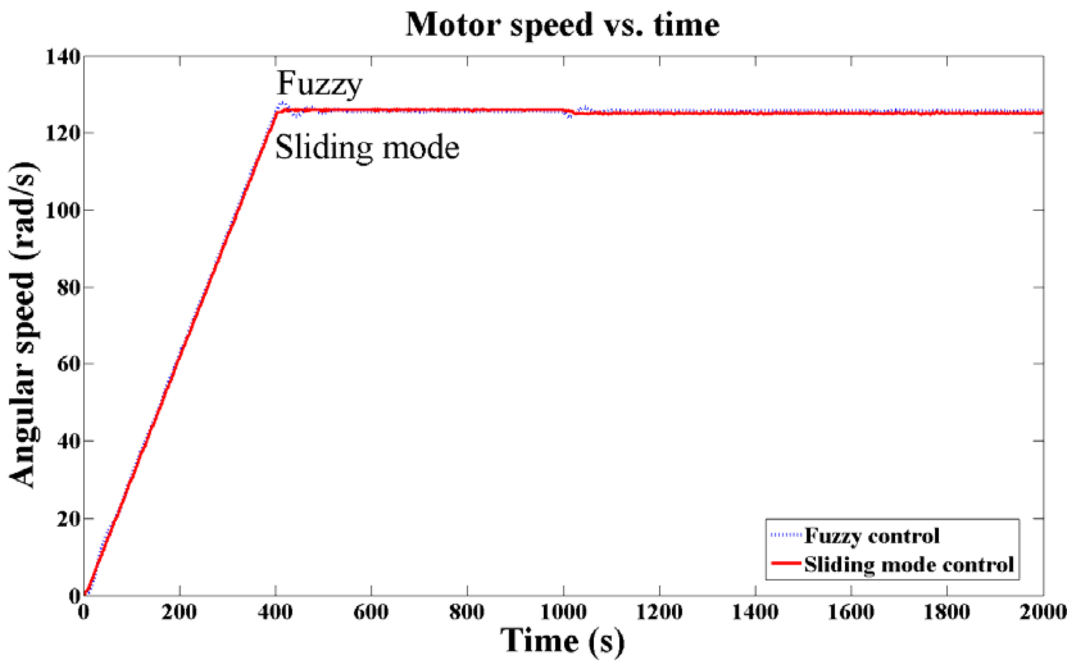


Figure 6.7 – Controller performances under disturbance input

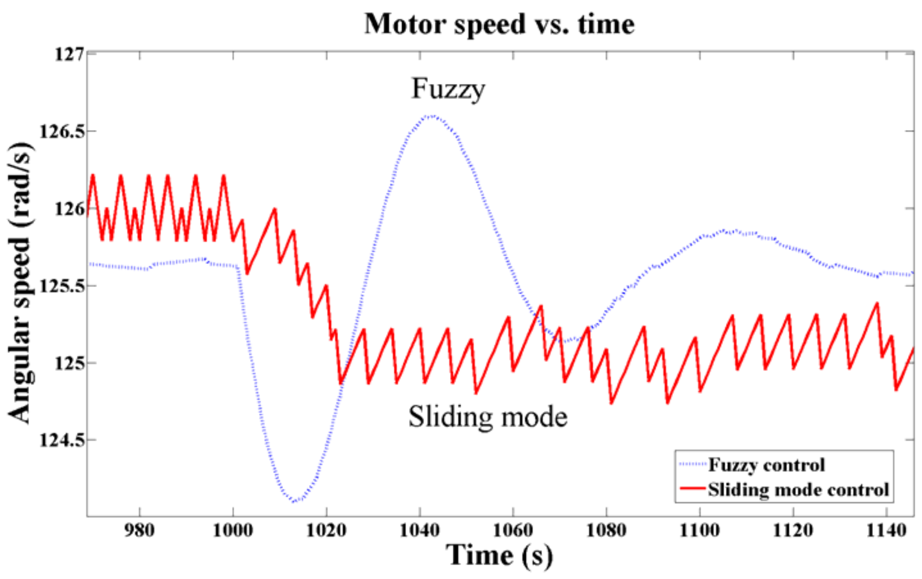


Figure 6.8 – Zoomed controller performances to highlight disturbance rejection

On the other hand, resource costs of the controllers vary in a wide range. These requirements are given in Table 6.2 in terms of slice look up tables (LUTs) and slice registers followed by the percentage consumed out of the available amount on the FPGA chip. At this point, it is critical to note that the resource requirements provided here are **not** in the same terms with the resource requirements provided in the previous chapters for Altera FPGAs, since Xilinx uses a different terminology for different amount of resources.

As can be seen from Table 6.2, the sliding mode controller is consuming around 5% of the registers and 15% of the LUTs available in the FPGA chip. If a multiple-axis solution is sought on the chip, these numbers are sufficient enough for driving 5-axis simultaneously. On the other hand, fuzzy controller requires high amount of resources as Slice LUTs and therefore does not seem applicable for multi-axis solutions.The complexity in the design affects not only the resources but also the attainable sampling period of the controller. In Table 6.3, maximum attainable sampling rates by controllers are presented. As seen in Table 6.3, the controllers can perform the required calculations around 85 cycles, which corresponds to 1.7 μs on an FPGA with a 50 MHz clock.

Table 6.3 – Minimum attainable sampling periods of controllers on the Xilinx Virtex-5 FGPA

Controller Type	Cycles to complete loop	Minimum period
Fuzzy	87 cycles	1.74 μs
SMC	84 cycles	1.68 μs

On the implementation end, sliding mode controller may be implemented with a sufficiently high effort. However, fuzzy controller is complicated to implement on an FPGA chip, especially when an embedded processor is not employed in the design, as in this case.

6.4 Closure

In this chapter, two advanced controller topologies are implemented on a Virtex5 FPGA. In order to take full advantage of the parallel processing capability of the FPGA, a softcore processor (explained as Method II in Chapter 4) is not employed; however as a preliminary study, only Method I-c is utilized for computation. In the results part, control algorithms are compared by their success and their resource cost on an FPGA chip. Results are discussed and important features are highlighted.

CHAPTER 7

HARDWARE-IN-THE-LOOP SIMULATION & RESULTS

The aim of this chapter is to demonstrate the utilization of the modules and methods that are developed in this thesis. Therefore, aforementioned pieces of a motion controller are put together to solve a real-world motion control problem.

As a matter of fact, most of the modules and design methods are investigated *individually* via different testing methods presented at each chapter and they are proven to be successful at their own right. However, it is critical to test the developed methods as a single motion controller design and demonstrate that the proposed design is effectively capable of dealing with real-world motion control problems. Furthermore, testing the overall system would also demonstrate the capabilities and success of the overall design paradigm, which would provide more meaningful results than individual experiments.

As a test case, a CNC machining center is selected. This selection is convenient since CNC machinery is one of the fundamental application areas of motion control systems. Furthermore, it also requires a multi-axis controller; which is also suitable for demonstrating the parallel processing capabilities of the FPGA based design.

The outline of this chapter is as follows: in the first section, the mathematical model of the CNC machining center selected as the test case is presented thoroughly. In the second section, the axis-controllers for this system are designed and the MATLAB/Simulink simulation results are provided. In the third section, the realization details of this problem via hardware-in-the-loop simulation (HILS) are briefly presented. In the fourth section, the HILS results are shown while some key results conclusions on the implementation are discussed in the last section.

7.1 Real system

The real system selected as the test case is a CNC machining center, which is a very important application area of a motion control system; since the success of the controller directly affects the productivity of the machine, as well as the quality of the product. Therefore, a CNC machining center is a suitable candidate for testing the motion controller design. Thus, this section starts with the details of the selected CNC center.

7.1.1 CNC machining center

The selected CNC vertical machining center shown in Figure 7.1 is located at the Machine Shop of the Mechanical Engineering Department of METU. It is a First MCV-1100 3-Axis CNC Machining center by Long Chang Machinery, equipped with an automatic tool changer, coolant and chip removal systems.

Figure 7.1 – First MCV-1100 3-Axis CNC Machining center

The axes of the machine are all mounted on friction (hydrostatic) guideways, and are driven by servomotors via ball screws. The *x*-axis carrying the cart (a.k.a. "table") on which the workpiece is mounted is illustrated in Figure 7.2 and the *y*-axis (a.k.a. "saddle") carries the entire *x*-axis assembly. On the other hand, Z-axis assembly is housed on the column and carries the entire headstock (main spindle shaft, motor, tool changing mechanism) as shown in Figure 7.3.

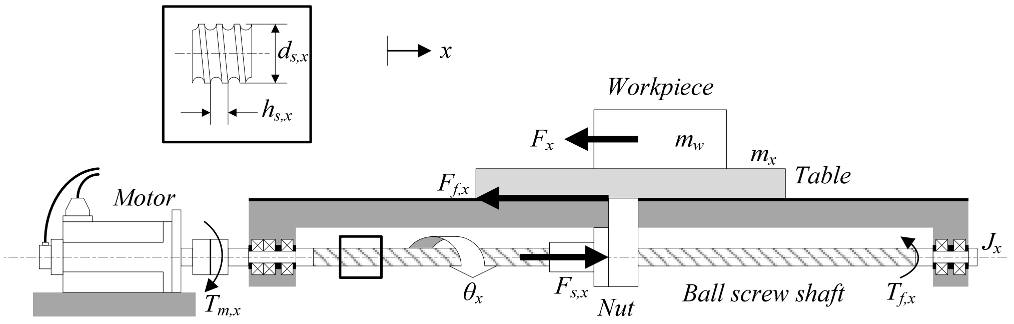


Figure 7.2 – X-axis feed drive for CNC machining center

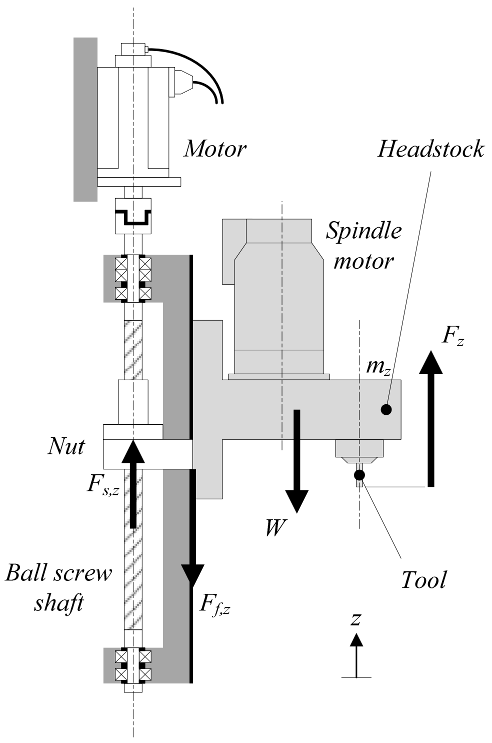


Figure 7.3 – Z-axis feed drive for CNC machining center

With the given information, the equation of motion for the x-axis cart can be written as

$$\ddot{x} = \left(\frac{1}{m_x + m_w}\right)\left(F_{s,x} - F_x - F_{f,x}\,\mathrm{sgn}(\dot{x})\right) \tag{7.1}$$

where m_w stands for the mass of the workpiece, m_x is the mass of the cart, F_x is the cutting force on the axis, $F_{f,x}$ is the friction force (dry) and $F_{s,x}$ is the force exerted on the table by the ball screw nut. The equation of motion for the dynamic system (as reduced to the motor shaft) becomes

$$\left(\frac{1}{J_x}\right)\left(b_x\left|\dot{\theta}_x\right|^{\frac{2}{3}}\right)\mathrm{sgn}\left(\dot{\theta}_x\right) + \ddot{\theta}_x = \left(\frac{1}{J_x}\right)\left(T_{m,x} - \frac{h_{s,x}}{2\pi\eta_{s,x}}F_{s,x} + T_{0,x}\right) \tag{7.2}$$

where J_x is the total moment of inertia of the ball screw and rotor, $T_{m,x}$ is the torque applied by the motor, $T_{f,x}$ is the total (dry and viscous) friction torque on the ball screw and rotor, $h_{s,x}$ is the pitch of the ball-screw shaft and $\eta_{s,x}$ is the ball screw efficiency. When backlash exists in the ball-screw assembly (which is a rare situation in precision parts), equations (7.1) and (7.2) are coupled together with (7.3) as

$$F_{s,x} = \begin{cases} k_x\left(d_x - \frac{b_x}{2}\right) & , d_x > \frac{b_x}{2} \\ 0 & , |d_x| < \frac{b_x}{2} \\ k_x\left(d_x + \frac{b_x}{2}\right) & , d_x < -\frac{b_x}{2} \end{cases} \tag{7.3}$$

where

$$d_x = \frac{h}{2\pi}\theta_x - x_x \tag{7.4}$$

If the ball screw is assumed to be backlash-free, these equations can be reduced to a single equation of motion that uses an equivalent set of parameters. That is, using (7.1) and (7.2) yields

$$\ddot{\theta}_x = \left(\frac{1}{J_{eq,x}}\right)\left[T_{m,x} - \frac{h_{s,x}}{2\pi\eta_{s,x}}F_x - \left(b_x\left|\dot{\theta}_x\right|^{\frac{2}{3}} + T_{0,eq,x}\right)\operatorname{sgn}\left(\dot{\theta}_x\right)\right] \tag{7.5}$$

Here, the equivalent inertia $J_{eq,x}$ is defined as

$$J_{eq,x} = J_x + \frac{h_s^2}{4\pi^2\eta_s}\left(m_x + m_w\right) \tag{7.6}$$

Since the table's position is linearly dependent on the angular position of the ball screw under no-backlash condition, it immediately follows that the velocities are also linearly dependent and $\mathrm{sgn}(\dot{x}) = \mathrm{sgn}(\dot{\theta})$ holds. Hence, utilizing equations (7.1) and (7.2), the equivalent dry friction $T_{f,eq,x}$ can be simply written as

$$T_{0,eq,x} = \frac{h_{s,x}}{2\pi\eta_{s,x}} F_{f,x} + T_{0,x} \tag{7.7}$$

The equations of motion regarding the *y*- and *z*-axes can be similarly obtained as

$$\ddot{y} = \left(\frac{1}{m_y + m_x + m_w}\right)\left[F_{s,y} - F_y - F_{f,y}\,\mathrm{sgn}(\dot{y})\right] \tag{7.8}$$

$$\ddot{\theta}_y = \left(\frac{1}{J_y}\right)\left\{T_{m,y} - \frac{h_{s,y}}{2\pi\eta_{s,y}} F_{s,y} - \left[b_y\left|\dot{\theta}_y\right|^{\frac{2}{3}} + T_{0,y}\right]\mathrm{sgn}\left(\dot{\theta}_y\right)\right\} \tag{7.9}$$

$$\ddot{z} = \left(\frac{1}{m_z}\right)\left[F_{s,z} + F_z - F_{f,z}\,\mathrm{sgn}(\dot{z}) - W\right] \tag{7.10}$$

$$\ddot{\theta}_z = \left(\frac{1}{J_z}\right)\left[T_{m,z} - \frac{h_{s,z}}{2\pi\eta_{s,z}} F_{s,z} - \left(b_z\left|\dot{\theta}_z\right|^{\frac{2}{3}} + T_{0,z}\right)\mathrm{sgn}\left(\dot{\theta}_z\right)\right] \tag{7.11}$$

Under no-backlash condition, these can be expressed in a simpler form similar to (7.5) as

$$\ddot{\theta}_y = \left(\frac{1}{J_{eq,y}}\right)\left\{T_{m,y} - \frac{h_{s,y}}{2\pi\eta_{s,y}}F_y - \left[b_y\left|\dot{\theta}_y\right|^{\frac{2}{3}} + T_{0,eq,y}\right]\mathrm{sgn}\left(\dot{\theta}_y\right)\right\} \tag{7.12}$$

$$\ddot{\theta}_z = \left(\frac{1}{J_{eq,z}}\right)\left\{T_{m,z} + \frac{h_{s,z}}{2\pi\eta_{s,z}}\left(F_z - W\right) - \left[b_z\left|\dot{\theta}_z\right|^{\frac{2}{3}} + T_{0,eq,z}\right]\mathrm{sgn}\left(\dot{\theta}_z\right)\right\} \tag{7.13}$$

Note that the weight of the headstock assembly (*W*) is included to the model of the *z*-axis drive. It is critical to notice that the feed-drive axes are driven by Fanuc α Series AC Servo Motors while the spindle motor is a Fanuc α Series AC Spindle (Induction) Motor. As specified in the user manuals, the speed-torque characteristics of the servo motors have a linearly decreasing tendency in the torque region up to the rated speed [1]. Beyond this point, the motor enters the constant power region as shown in Figure 7.4.

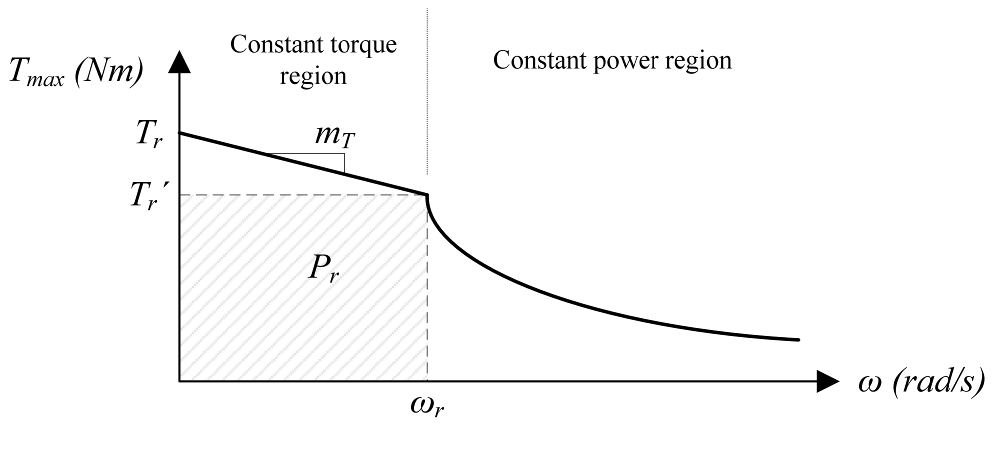


Figure 7.4 – Torque capability curve for CNC machining center axis motors

The torque envelope of the motor, T_{max}, (See Fig. 7.4) is then as follows

$$T_{max} = \begin{cases} T_r + m_T\,|\omega| & ,|\omega| < \omega_r \\ \dfrac{P_r}{|\omega|} & ,|\omega| \ge \omega_r \end{cases} \qquad (7.14)$$

where T_r and ω_r represent the rated torque and rated speed, respectively. T_r' is the torque produced by the motor and P_r is the power output, both at the rated speed while $m_T = (T_r - T_r') / \omega_r$ and $P_r = T_r'\omega_r$. The numerical values for the parameters defining the plant are provided in Table 7.1.

Finally, it should be noted that the motor position data is obtained from the axes via an incremental encoder that generates 10000 ppr which would yield a resolution of 40000 pulses per revolution with quadrature (4X) decoding.

7.1.2 MATLAB/Simulink model

The system's governing equations in terms of equivalent torque and inertia are provided in the previous section in (7.5), (7.12), and (7.13) for x, y and z axes respectively. Using these equations and the system parameters provided in Table 7.1; it is possible to develop a Simulink model for the overall system. Figure 7.5 shows the dynamic model of a single axis (x-axis) of the system developed by the MATLAB/Simulink package.

Table 7.1 – MATLAB/Simulink model of a single axis of the CNC machining center

Parameter	**Sym.**	**Unit**	***X***	***Y***	***Z***
Mass	m	kg	130	331.97	260
Dry friction force	F_f	N	200	200	200
Moment of inertia	J	kg m^2	7.994×10^{-3}	16.484×10^{-3}	19.745×10^{-3}
Dry friction torque	T_f	N	1.1	1.5	2.1
Viscous friction coefficient	B	Nms/rad	0.0005	0.0005	0.0005
Equivalent moment of inertia	J_{eq}	kg m^2	0.00834	0.01737	0.02044
Equivalent dry friction	$T_{f,eq}$	N m	1.435	1.835	2.435
Ball screw lead	h_s	m	0.010	0.010	0.010
Ball screw efficiency	η_s	-	0.95	0.95	0.95
Rated torque	T_r	N m	12	22	30
Rated speed	ω_r	rad/s	209.44	209.44	209.44
Rated power	P_r	W	2094.4	3769.9	4398.2
Torque-speed slope	m_T	Nms/rad	-0.00955	-0.01910	-0.04297

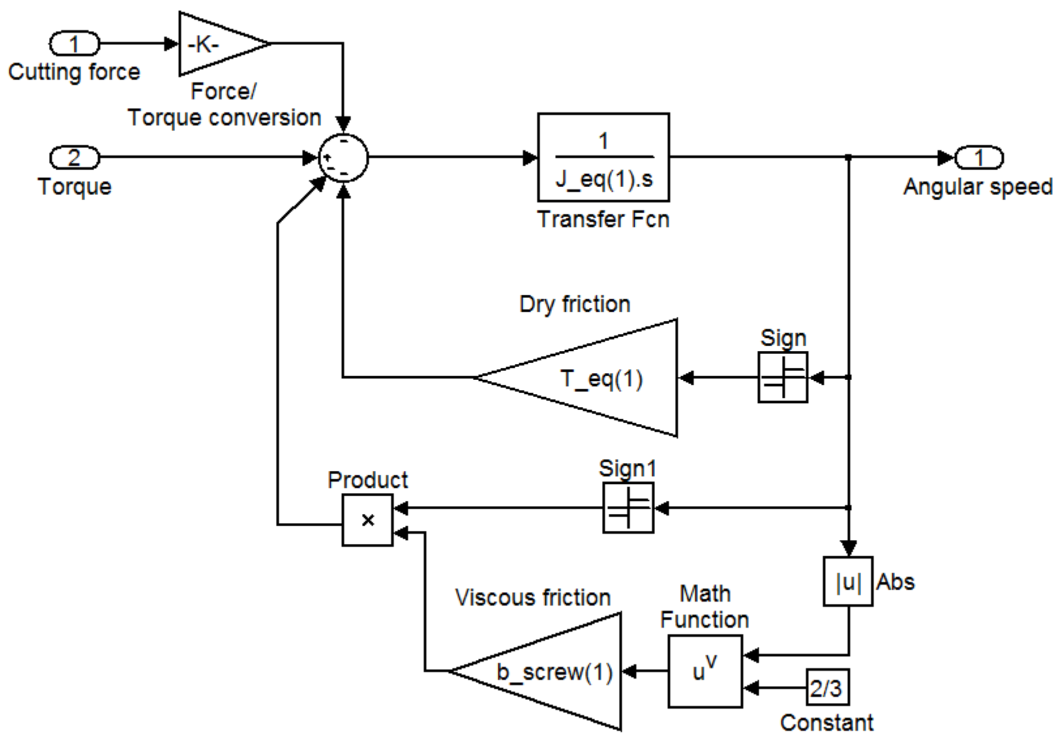


Figure 7.5 – MATLAB/Simulink model of a single axis of the CNC machining center

As can be seen in Figure 7.5, the Simulink model has a transfer function that relates the input torque to the angular speed of the motor shaft. The model includes two different friction models (dry and viscous) and the disturbance input that are exactly represented in the governing equations of the system. This model is employed in all of the three axes of the machining center, with a change in the parameters: equivalent inertia (J_eq), equivalent dry friction coefficient (T_eq), viscous friction coefficient of screw (b_screw), screw efficiency (eff_scr) and pitch of the screw (h). Note that these parameters can easily be changed by changing the index of the parameter (i.e. in J_eq(1) index 1 represents the parameter for the x axis).

While testing/debugging the designed controller, the model in Figure 7.5 is utilized as the controlled plant in the MATLAB/Simulink simulations. The next section explains the details of the controller design.

7.2 Controller design

In section 0, the CNC machining center that is considered as the test case of the motion controller design is introduced and its governing equations, Simulink model, and the relevant system parameters are provided. In this section, a controller is developed based on the introduced model and the simulation results obtained in MATLAB/Simulink are presented.

7.2.1 Controller selection

As discussed in Chapter 2, there are many motion controller topologies that can be implemented on an FPGA, including both conventional and novel/intelligent controllers. However, in order to demonstrate the presented methods in the previous chapters, the choice for the test case controller is to be made between the state-space controller (presented in Chapter 4) and the filter implementation (discussed in Chapter 5). Even though both of these controllers are equally applicable, the filter implementation seems a more convenient choice since the classical SISO controller topologies (like the industry-standard PID) can be easily realized. Furthermore, it is a more efficient method (in terms of expended resources) on FPGA: As demonstrated in Chapters 4 and 5, two instances of the filter can be realized with 27% of the logic cells (i.e. a single instance consumes 13%) while a state-space controller can realized utilizing 24% of the logic cells (via integer multiplication method). Therefore, filter implementation is selected to build the controller topology for the test case and thus its FPGA implementation is carried out by the method presented in Chapter 5.

7.2.2 Linearized system model

Considering the system model provided in Figure 7.5, the transfer function (G_{pw}) between the input torque and the output angular speed is $G_{pw}(s) = 1 / (J_{eq}\, s)$ when the nonlinear terms (friction) are discarded under the assumption that they could be conveniently visualized as a part of the disturbance. Since a position control is desired, the output of the transfer function G_{pw} needs to be integrated, which would lead to the transfer function between the input torque (manipulation) and the output position of the motor shaft, that is $G_p = 1 / (J_{eq}\, s^2)$. This continuous time transfer function of the plant is considered as the system model for the controller design via root locus method.

Note that since the position feedback is obtained from an encoder, there is an encoder gain of $40000/(2\pi h)$ in the feedback loop. Therefore, the reference trajectory may also obtained in terms of encoder pulses (hence integers) as previously discussed in Chapter 4, while commenting on the advantageous attributes of digital motion control systems. Therefore, the controller is designed according to an encoder gain applied to both the reference and the position feedback.

As a final mark, it is important to note that there exist 3 different axes to be controlled and hence 3 different plant models are obtained. However, the following procedure is presented for a single axis (x), since the plants and the design scheme are very similar for each axis.

7.2.3 Design via root locus technique

After the system model is obtained, the controller may be designed in either continuous-time domain or discrete-time domain. While both approaches are equally acceptable, in this study the latter approach is adopted and therefore as a first step, the system model needs to be converted to an equivalent discrete-time domain (i.e. z-domain) representation.

As an initial step for discretization of the system, a convenient sampling time (t_s) should be selected. In modern real-time control systems, 1 kHz sampling frequency is a highly sufficient for even demanding control applications, thus $t_s = 0.001s$ is selected for the sampling period of the controller. After the selection of t_s, the system is discretized using zero-order hold method to obtain the discrete time model of the system $G_p(z)$ via "*c2d*" function of MATLAB.

Once the discrete-time transfer function is obtained for the plant, the root-locus design is performed via *sisotool,* which is a convenient tool provided by MATLAB for root locus design method. The interface allows the user to select closed-loop pole locations by adjusting the controller gain. That is, the user can modify the open-loop poles and zeros via an interactive root locus plot. Using *sisotool,* the root locus plot of the uncontrolled system is provided in Figure 7.6. Notice that the system is unstable since the closed-loop poles are outside the unit circle and it is not possible to stabilize the system by a simple proportional controller since one of the poles end up at infinity as gain increases. Therefore, another controller needs to be designed to obtain a stable system that yields desirable tracking- and disturbance rejection performances.

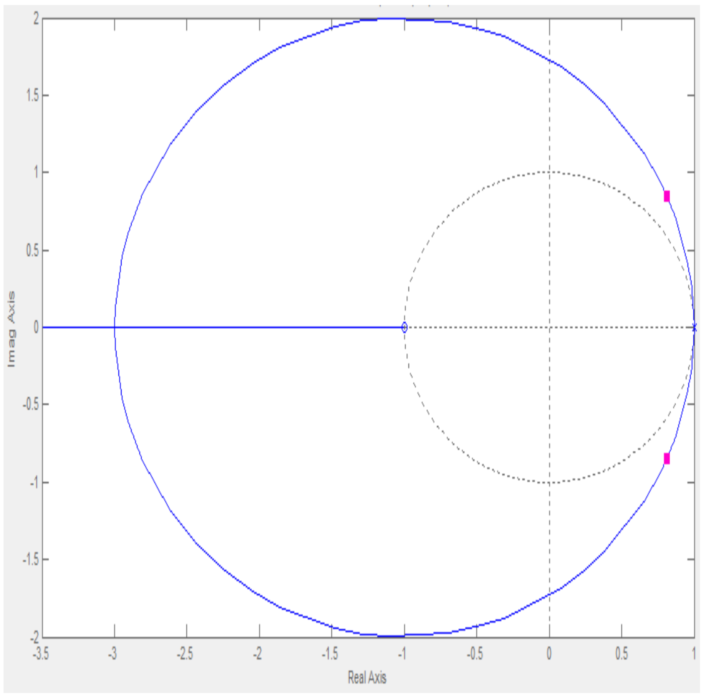


Figure 7.6 – Root locus plot of the uncompensated system

As an initial step, a zero is added at 0.8 on the real axis in order to stabilize the system. After the addition of the zero, the speed of the system is further increased by placing a pole at 0.1. Hence the root locus plot has become as shown in Figure 7.7.

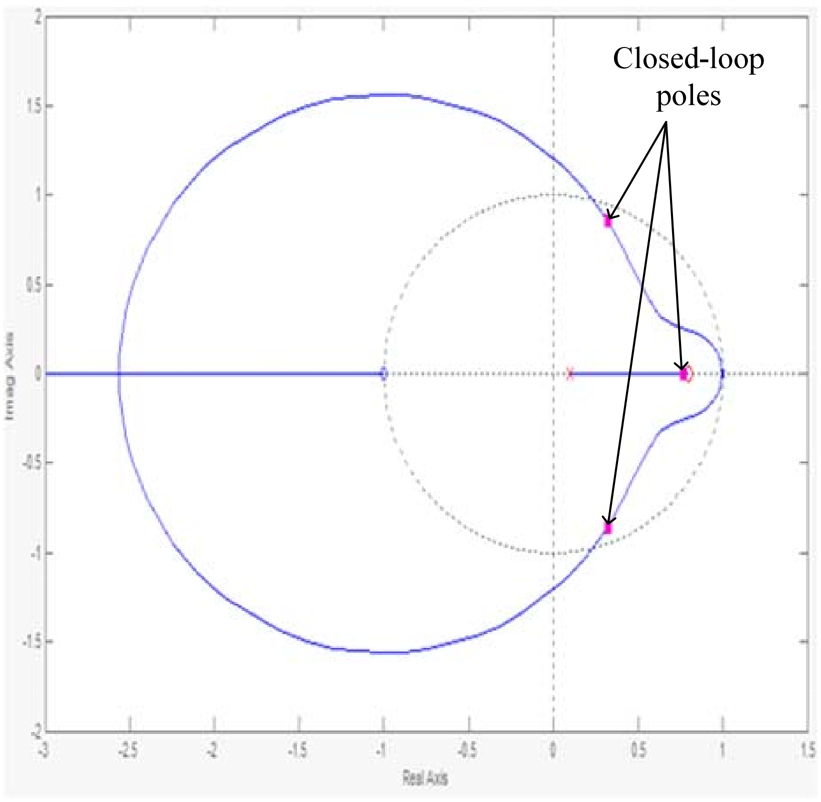


Figure 7.7 – Root locus plot of the system after addition of a pole and a zero

As can be seen in Figure 7.7, the system may be still unstable since two of the closed-loop poles could be outside the unit circle; however after the addition of the pole/zero, now it is possible to stabilize the system by simple gain adjustment. Note that the initial gain is selected as unity by default in *sisotool*, therefore by decreasing the gain, the system can be stabilized.

By decreasing the gain gradually, it can be seen that the closed-loop poles cross the unit circle when gain (K) = 0.492 and therefore values below this value should yield a stable system response. Considering the actual plant, it is known that during machining process, cutting forces act on the system as a disturbance. Therefore, a sufficiently high K value is desired for increasing the system's dynamic stiffness (to disturbances). Hence, K = 0.4 is chosen conveniently for the gain value of the controller. The resulting closed-loop poles are also shown in Figure 7.7

On the other hand, it is also necessary to check the closed-loop bode plot, in order to obtain the bandwidth frequency of the controlled system (which is also provided in the *sisotool* interface). Bode plot of the closed-loop system is shown in Figure 7.8. As can be seen in Figure 7.8, the bandwidth frequency of the system is around 300 Hz (< half the sampling frequency = 500 Hz), which is deemed sufficient for the most CNC vertical machining centers.

Proceeding with the designed controller, its discrete-time domain expression can be obtained as follows:

$$G_c(z) = \frac{2z-1.6}{1.1z-0.1} \tag{7.15}$$

Using (7.15) the developed controller may be tested via Simulink to observe the performance of the controller with the desired trajectory of the x axis.

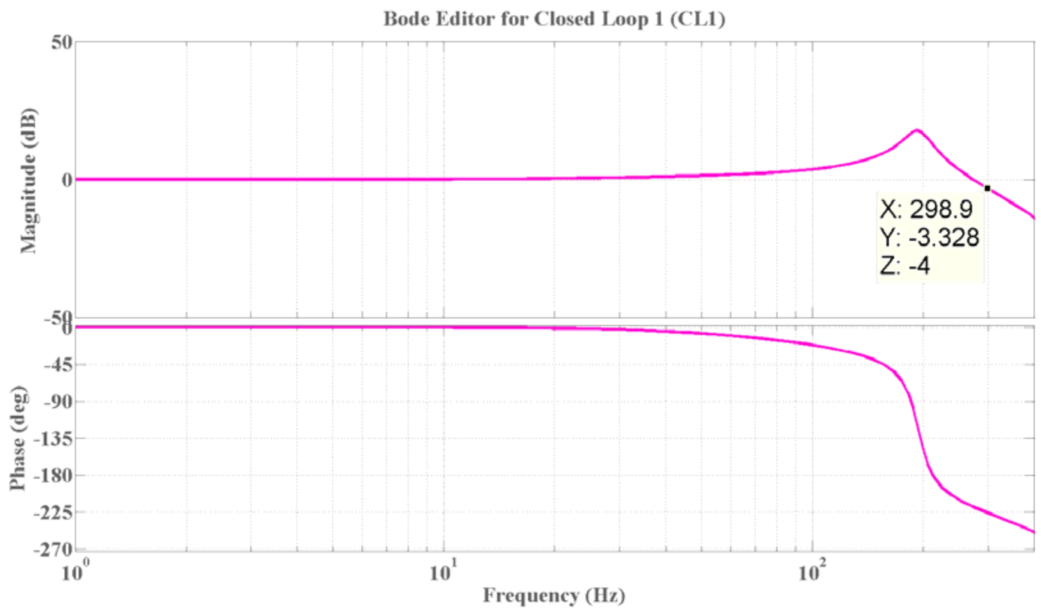


Figure 7.8 – Closed-loop Bode plot of the system

It is important to note that while this design method is explained for the x axis of the CNC machining center, the design method for the other axes are very similar to the x-axis; since the same plant with slightly different parameters are considered. As a matter of fact, the same controller is applied to the all of the axes with a slight change in the controller gains for the y and z axes. The reason behind this choice is that, the root locus plots of the other axes allow more increase in the controller gain, while keeping the closed-loop poles within the unit circle. Therefore, the gain values for y and z axes are selected as 0.6, while the other controller parameters are essentially the same.

Table 7.2 – Controller design parameters for x, y and z axes

Axis	Gain	Closed-loop poles	Bandwidth frequency [Hz]
X	0.4	0.771, 0.321±0.86i	296
Y	0.6	0.755, 0.425±0.69i	257
Z	0.6	0.741, 0.469±0.61i	236

The next section presents the Simulink results of the designed controller.

7.2.4 Simulink simulation results

In Figure 7.9, the overall system model in Simulink is shown. As can be seen, the overall system incorporates the following systems: **i)** the feed-drive (axis) model shown in Figure 7.5 (shown as x-Axis), **ii)** the controller (the lead-lag compensator), **iii)** the feedforward dry friction compensator, **iv)** the torque generation model for the motor. Note that an encoder gain is placed behind the scope to convert the output position signal from radians to encoder counts. To simulate the real encoder behavior, a floor function is placed after the encoder gain. Since the axis model provides the output in terms of angular speed, an integrator is added after the axis model to obtain the angular position. The inputs, which are previously defined reference trajectory and cutting force (disturbance), are obtained from the Matlab workspace. Notice that the provided model represents the x-axis of the machining center; however the model can be applied to all axes with proper changes in the parameter indices.

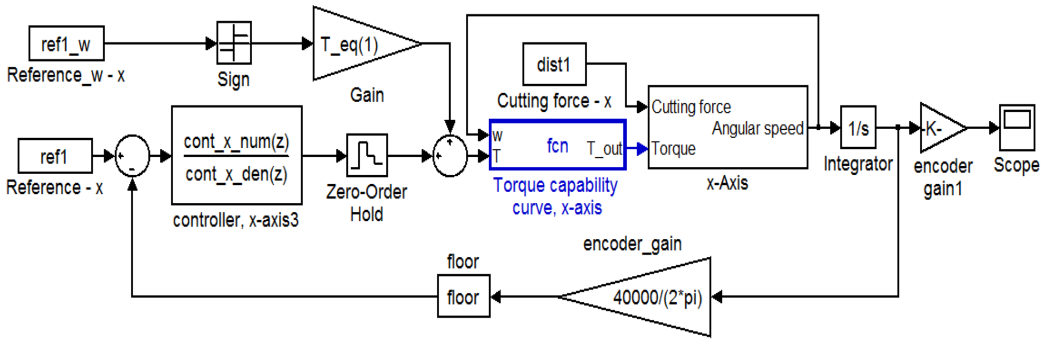


Figure 7.9 – Simulink model of the overall system

The reference trajectory, which has a duration of 20 seconds, is a portion extracted from a real machining operation and is shown in Figure 7.10. As can be seen in Figure 7.10, the reference is provided in terms of encoder counts. On the other hand, the disturbance inputs in Figure 7.11 are not exactly the same as the disturbance on the real system; since the real disturbance (machining) process is relatively hard to model in Simulink environment and is beyond the scope of this thesis. However, an approximation of the actual case, which would provide sufficient information about the disturbance resistance characteristics of the system, can be implemented with ease. Therefore, similar results should be expected from the real test in terms of tracking error. The disturbance inputs corresponding to light- and heavy machining processes are provided in Figure 7.11.

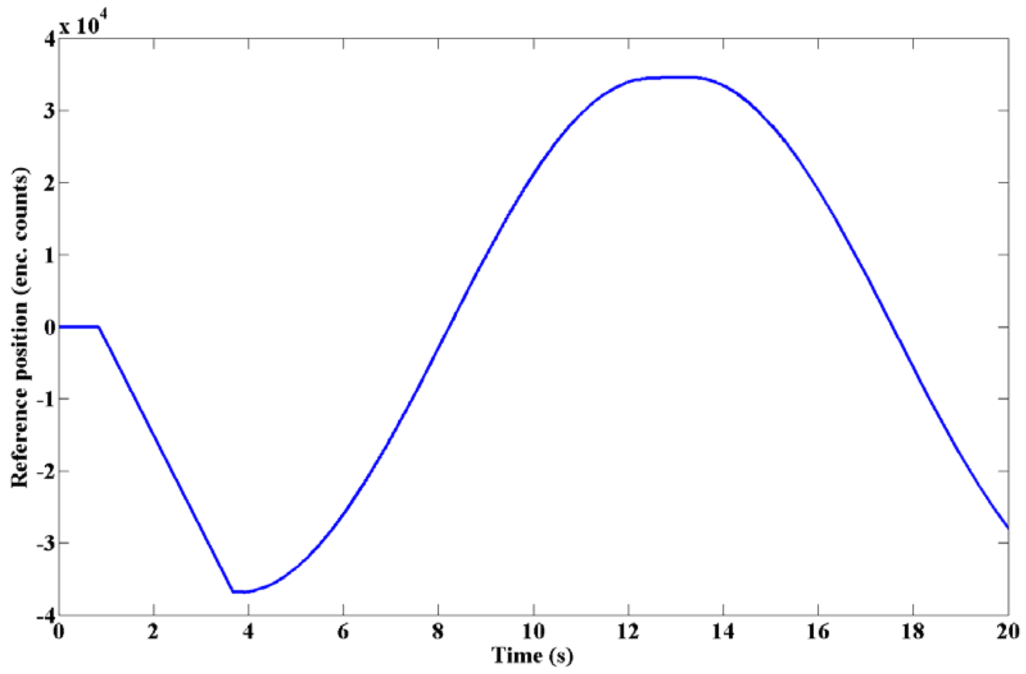


Figure 7.10 – Reference trajectory for the X-axis

It is critical to note that for this particular application, the machining accuracy of the center is designated as ±10μm. Hence, the maximum tracking error of the controlled system under the worst case scenario is expected to be less than 40 encoder counts.

Simulink results are obtained by using the reference trajectory (shown in Figure 7.10), the disturbance input (in Figure 7.11), and the system model provided in Figure 7.9. The results are obtained for both cases of disturbance inputs. The tracking errors are shown in Figure 7.12 in terms of encoder counts.

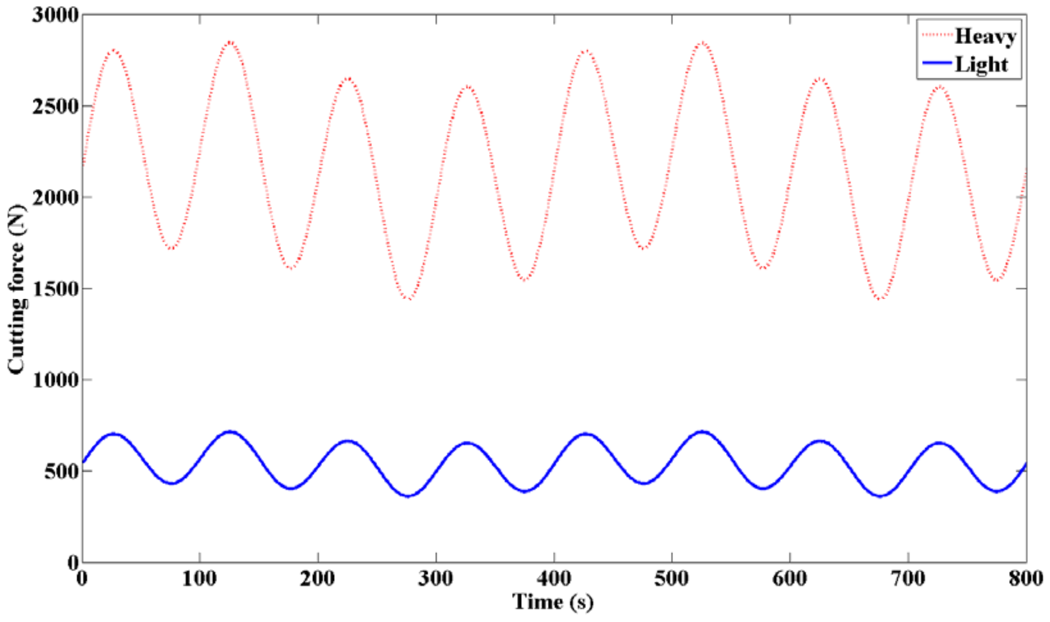


Figure 7.11 – Disturbance inputs for light and heavy machining conditions

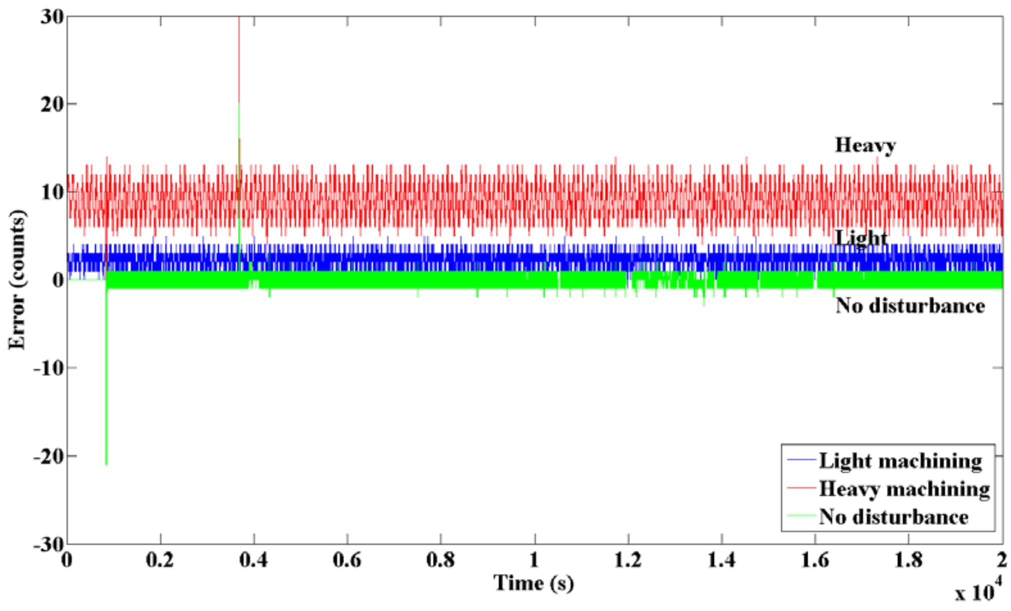


Figure 7.12 – Simulink results of the designed controller in terms of encoder counts

As can be seen in Figure 7.12, when no disturbance is acting on the system, the control system is able to follow the trajectory within an error band of 1 encoder count. However, when a disturbance (i.e. machining force) is present, the tracking error of the system becomes (roughly) 4 and 13 counts for light- and heavy machining (simulation) respectively. Table 7.3 shows the means and standard deviations of error obtained through the simulation study.

Table 7.3 – Mean, max and standard deviation values of the Simulink results (in counts)

Disturbance type	**Mean**	**Max**	**Standard deviation**
No disturbance	0.005	20	0.804
Light disturbance	2.252	23	0.908
Heavy disturbance	9.005	30	1.874

Taking into account the resolution of the encoder (10000 ppr) as well as the pitch of the ballscrew shaft (10 mm), one can determine that 10 encoder counts correspond to a table displacement of 2.5 μm. Thus, Table 7.4 illustrates these table/cart displacement errors.

As can be seen in Table 7.4, the results are quite successful in terms of mean error and standard deviation of the error. The maximum error is also sufficient for control purposes.

Table 7.4 – Mean, max, and standard deviation values of the Simulink results (in μm)

Disturbance type	Mean	Max	Standard deviation
No disturbance	0.001	5	0.201
Light disturbance	0.563	5.75	0.227
Heavy disturbance	2.251	7.5	0.469

If Figure 7.12 is carefully observed, it can be seen that the maximum errors are attained in the sharp transitions of Figure 7.10, which corresponds to "rapid travel/motion" along the trajectory (the phase until t = 4s in Figure). Notice that in CNC technology, rapid travel is a point-to-point motion and position control along the trajectory lying between the initial- and target position is not needed. Therefore, it can be concluded that the designed controller can be applied to the real case.

7.3 Implementation of the system

In the preceding section, the controller design via classical root locus technique is explained and simulation results demonstrating the command tracking and disturbance rejection properties are given for the designed controllers. The results have been proven to be successful and therefore the controller is to be implemented on the FPGA along with other necessary modules for encoder interfacing, PWM generation, etc. In this section, the implementation of the control system and the hardware-in-the-loop simulation (HILS) of the plant are presented.

7.3.1 Implementation of the control system on the FPGA

The control system is implemented on the DE1 FPGA board with an Altera Cyclone II FPGA chip by utilizing the methods and modules provided in the previous chapters. The interface modules that are employed are as follows:

- Incremental encoder decoding module: For gathering angular position information from the axes of the CNC machining center. 3 instances are employed for 3 axes of the machining center.
- PWM transmitter module: For transmitting the calculated torque command to the motor driver (in torque/current modulation mode). 3 instances are employed for 3 axes of the machining center.
- RS-232 controller: For setting the controller parameters and reference values via PC. Single instance is employed.
- SRAM controller: For storing the controller parameters and reference values on the FPGA board. Single instance is employed.

Along with these modules, three controllers are implemented on the chip by employing three instances of the *generic filter module* presented in Chapter 5, using Method I-a (IMUL). Consequently, all the interface modules and three controller modules are implemented on the FPGA chip.

It is important to note that, for control of 3-axis, this design consumes 7550 logic elements (LE), corresponding to 40% of the FPGA's LE resources, along with 12 embedded 9-bit multipliers, corresponding to 23% of the

FPGA's available multiplier resources. On the other hand, if a single axis solution is required, this requirement drops to 3027 logic elements (16%) and 4 embedded 9-bit multipliers (8%). Considering that Altera Cyclone II FPGA is a relatively aged chip (having 18752 total logic elements, 52 embedded 9-bit multipliers), these results prove that the proposed solution is an extremely resource-wise efficient solution. Resource utilization of a single- and 3-axes solution (i.e. synthesized digital circuitry) are presented in Figure 7.13 and Figure 7.14

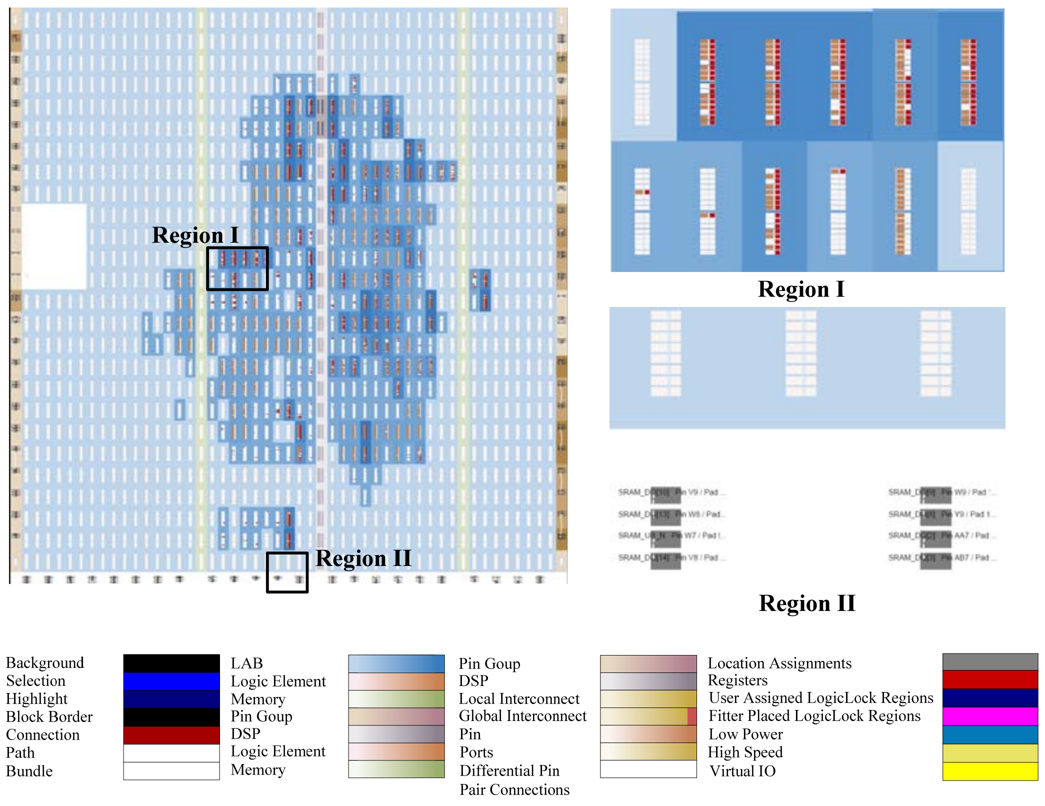


Figure 7.13 – Resource utilization of the single axis solution

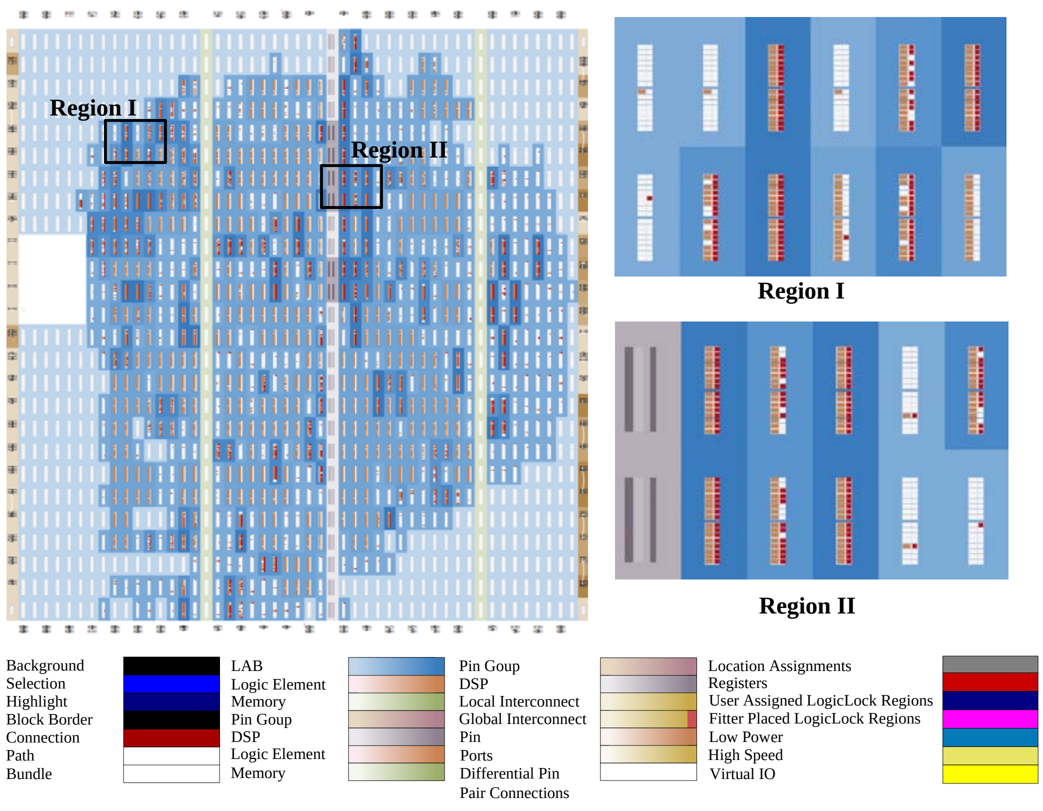


Figure 7.14 – Resource utilization of the implemented 3-axis solution

7.3.2 Realization of the plant via hardware in the loop simulation

HILS of the real system is realized by a hybrid design which includes an Altera FPGA and an Atmel processor. It can provide encoder signals and receive PWM data, exactly the way that a regular CNC machining center would provide and receive. Therefore, within the context of this study, the HILS system is treated as a real plant since there is no difference from the controller's point of view. Figure 7.15 shows the controller system coupled to the HIL simulator.

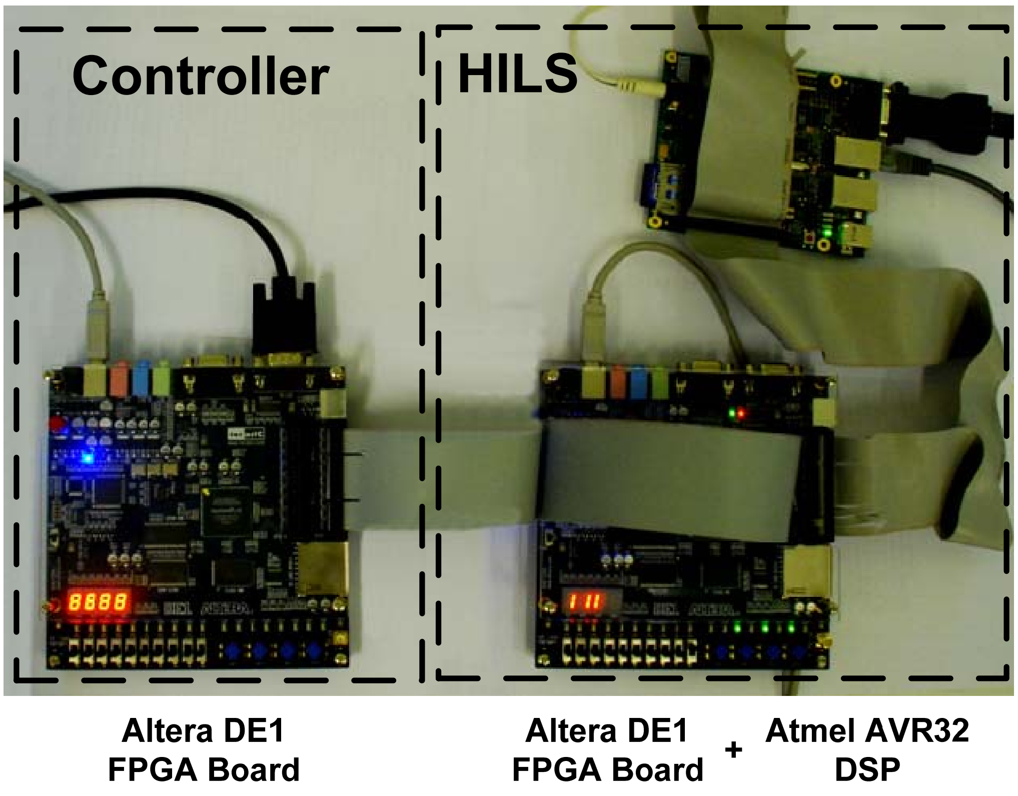


Figure 7.15 – Schematic of the hardware in the loop simulation system

It is critical to notice that the modules within the HILS system is **not** related to the modules presented in Chapter 3. The HILS system is a result of another study [32], and is adopted in this study for the test case simulation of the developed controller.

In chapter 5, the attainable sampling frequency of the digital filter is presented as 385 kHz; therefore the selected sampling frequency (1 kHz) is attainable on the controller end. On the other hand, the computations on the HIL simulator take longer than the selected sampling frequency and inevitably HIL simulation is realized non-real time at 100 Hz. However though, the simulation results represent the real sampling frequency of the system and therefore applicable to a real-time control problem. The results provided in the next section are obtained by utilizing the HILS system.

7.4 HILS Results

HILS results are obtained by using two different reference trajectories, which are portions obtained from a CNC code generated to manufacture a plastic bottle injection mold, as shown in Figure 7.16. The first trajectory is shown in Figure 7.17 and the second one is shown in Figure 7.18.

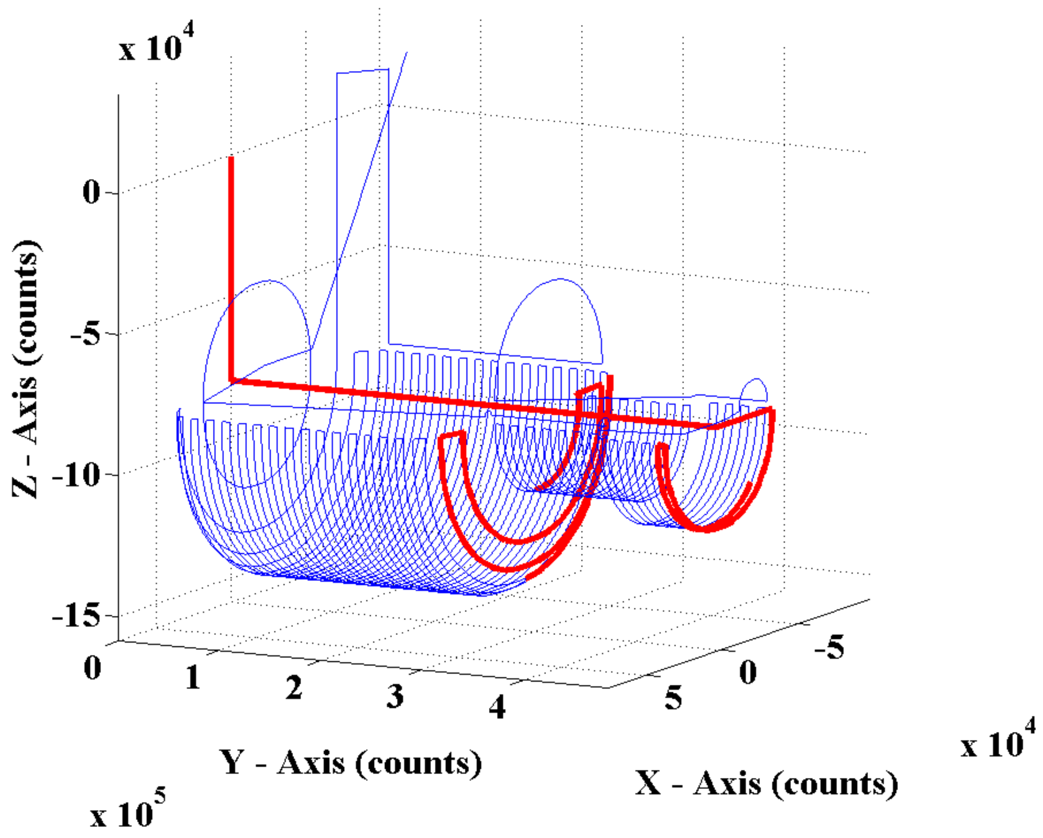


Figure 7.16 – Reference trajectories for a plastic bottle injection mold (selected portions indicated with red color)

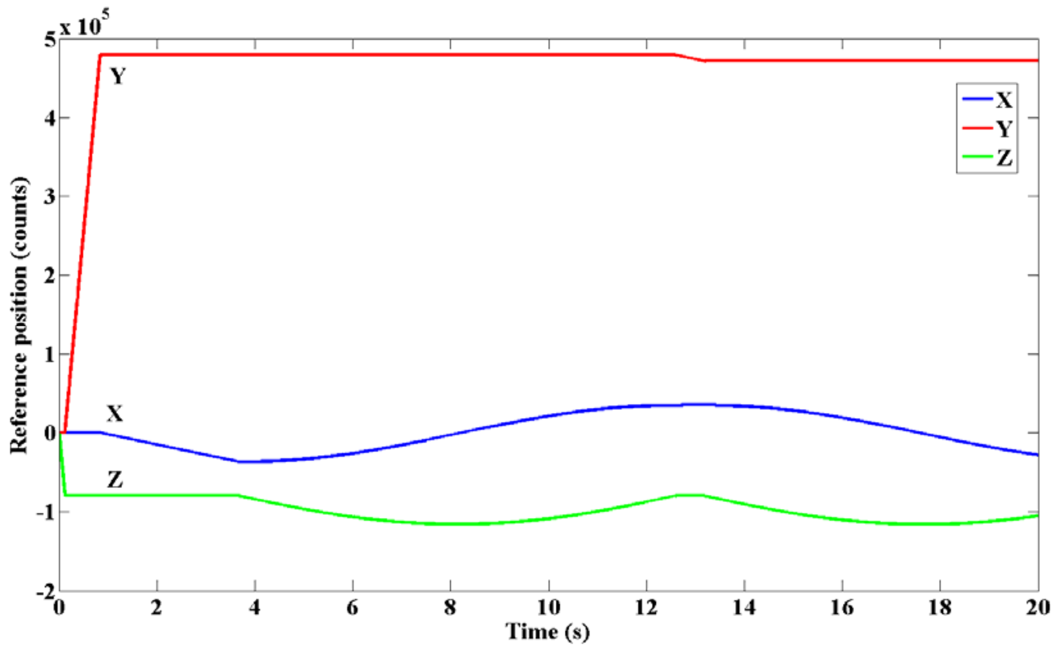


Figure 7.17 – Reference trajectories for X,Y and Z axes (t = 0-20s)

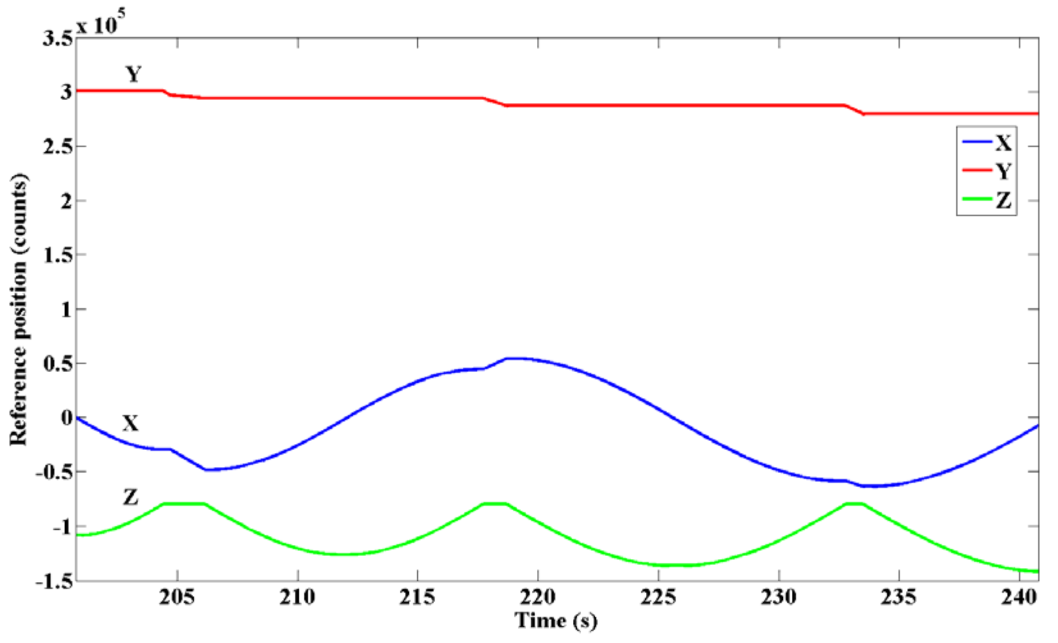


Figure 7.18 – Reference trajectories for X,Y and Z axes (t = 201-241s)

As can be seen from Figure 7.17 and Figure 7.18, the trajectories are portions from a real trajectory set for a CNC machine. The reason why certain portions are selected is that the data to be stored in the SRAM device is limited with 512 kb. However, when a suitable reference command generator is coupled to the design, there is no limit in prolonging the simulation times as the reference data can then be fed to the controller in a continuous fashion. On the other hand, the disturbances that are employed are already discussed in the preceding section and therefore will not be further discussed here.

The results of the HILS are presented in the following order:

1. Figure 7.19 – First trajectory motor position error in X-axis (t = 0-20s)
2. Figure 7.20 – First trajectory motor position error in Y-axis (t = 0-20s)
3. Figure 7.21 – First trajectory motor position error in Z-axis (t = 0-20s)
4. Figure 7.22 – Second trajectory motor position error in X-axis (t = 201-241s)
5. Figure 7.23 – Second trajectory motor position error in Y-axis (t = 201-241s)
6. Figure 7.24 – Second trajectory motor position error in Z-axis (t = 201-241s)
7. Figure 7.25 – Second trajectory cart position error in X-axis with backlash (t = 201-241s)
8. Figure 7.26 – Second trajectory cart position error in Y-axis with backlash (t = 201-241s)
9. Figure 7.27 – Second trajectory cart position error in Z-axis with backlash (t = 201-241s)

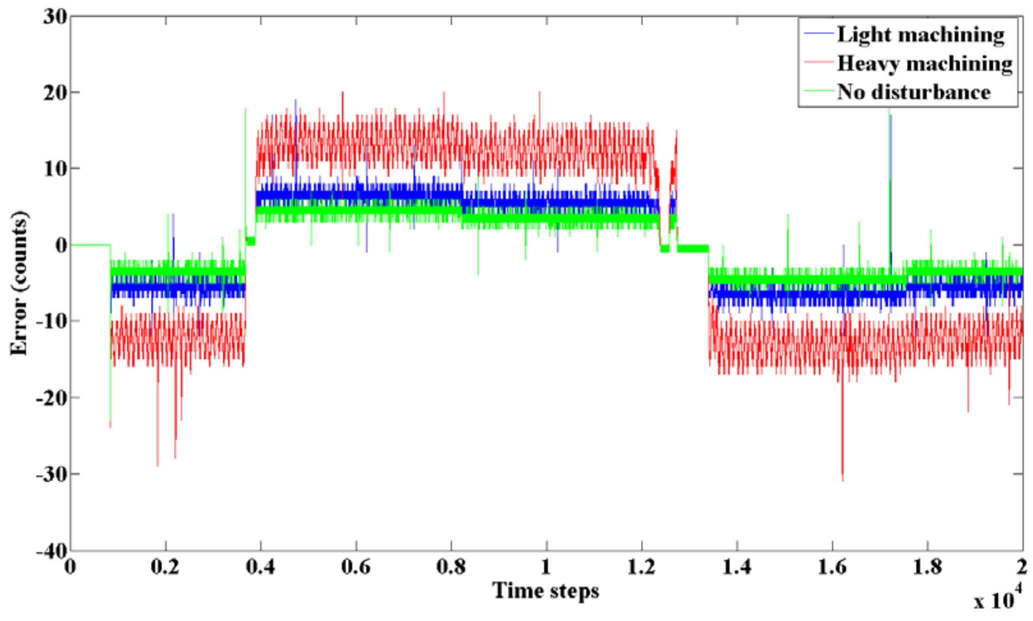


Figure 7.19 – First trajectory motor position error in X-axis (t = 0-20s)

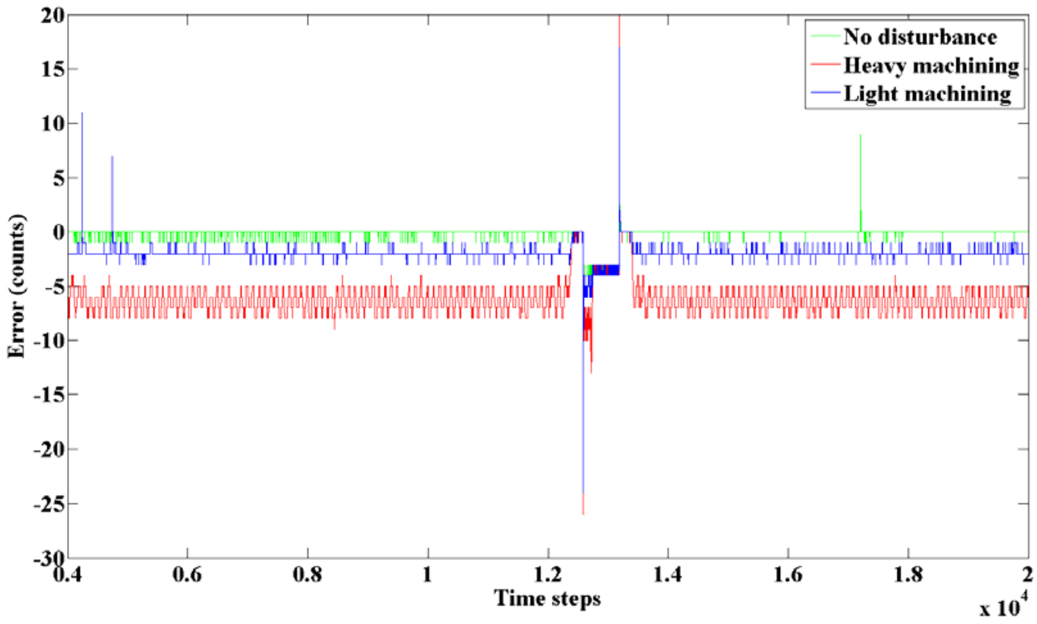


Figure 7.20 – First trajectory motor position error in Y-axis (t = 0-20s)

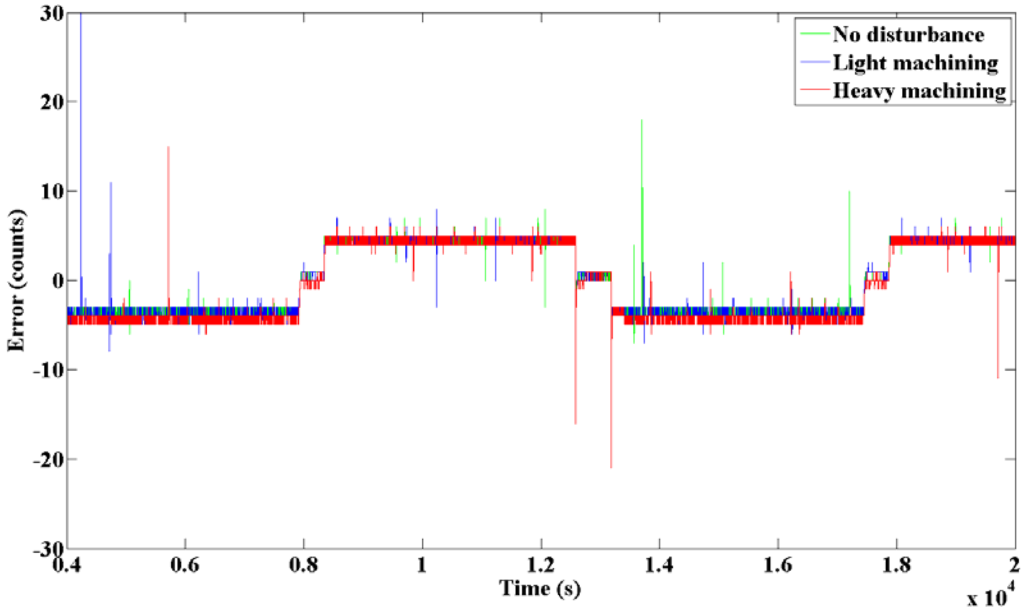


Figure 7.21 – First trajectory motor position error in Z-axis (t = 0-20s)

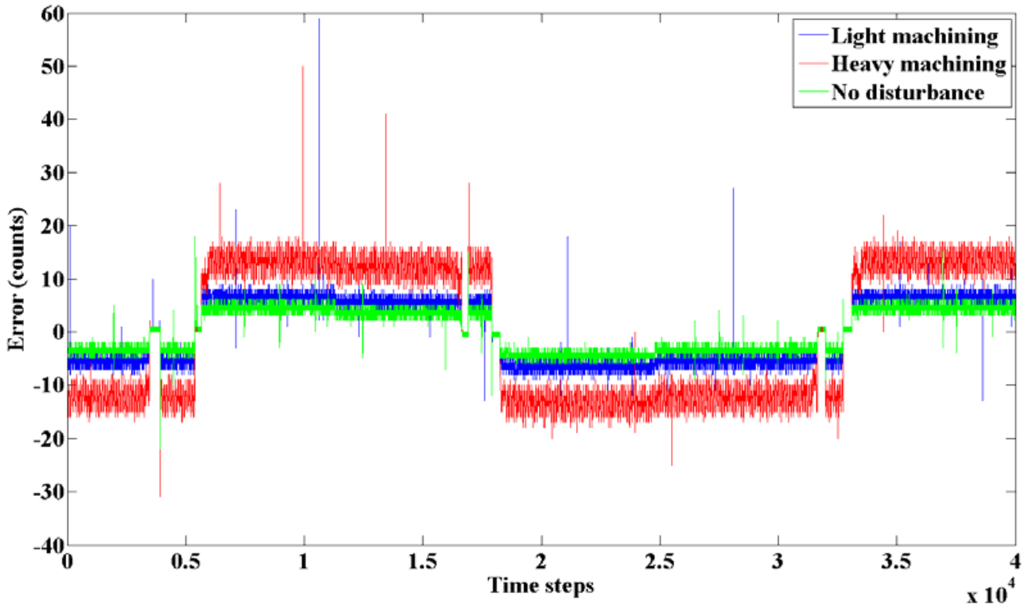


Figure 7.22 – Second trajectory motor position error in X-axis (t = 201-241s)

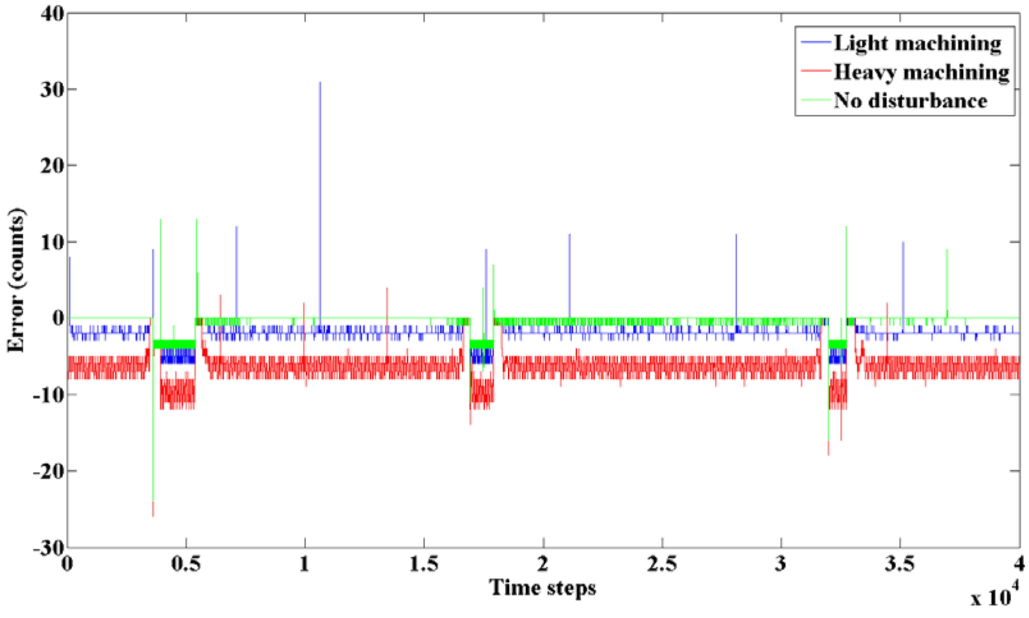


Figure 7.23 – Second trajectory motor position error in Y-axis (t = 201-241s)

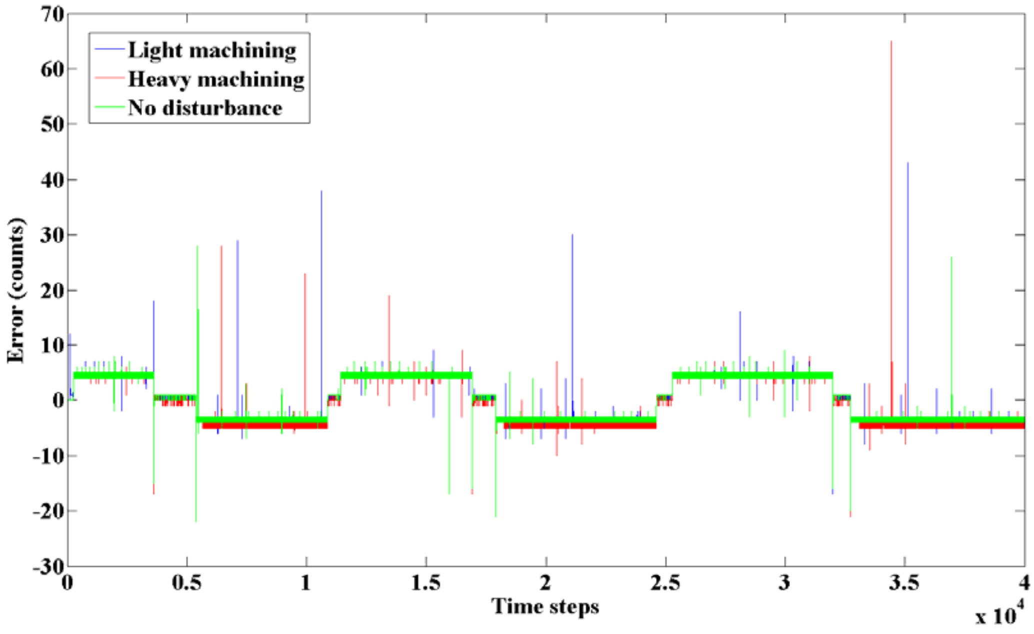


Figure 7.24 – Second trajectory motor position error in Z-axis (t = 201-241s)

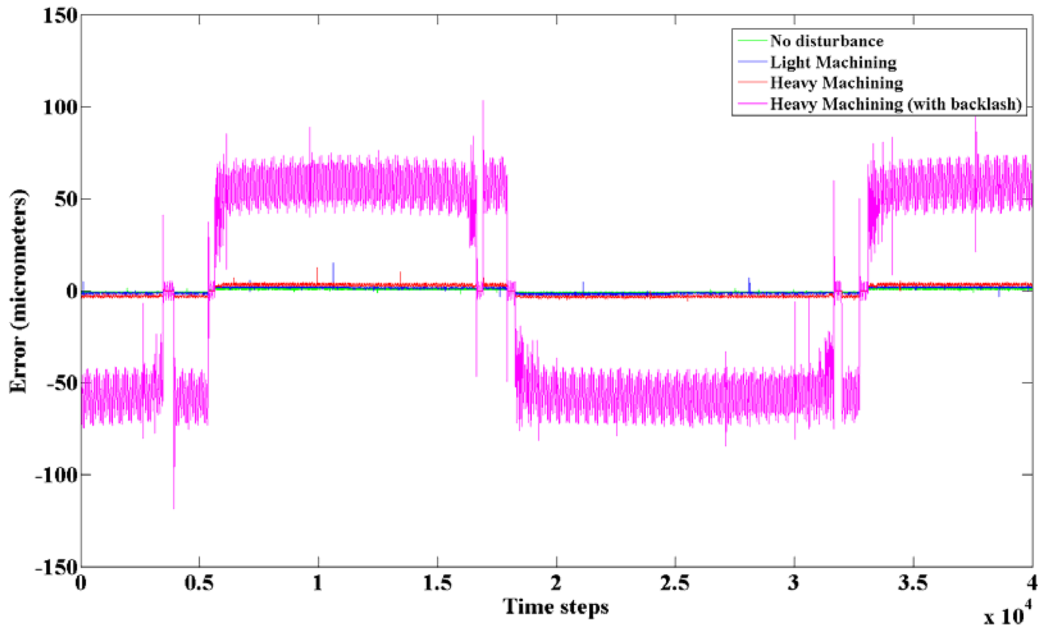


Figure 7.25 – Second trajectory cart position error in X-axis with backlash (t = 201-241s)

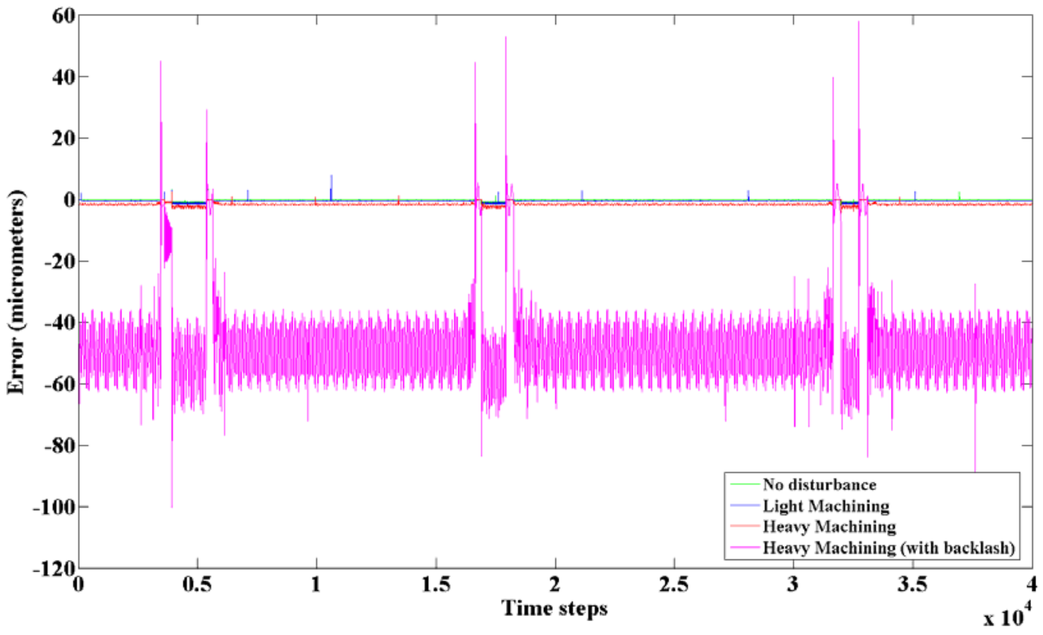


Figure 7.26 – Second trajectory cart position error in Y-axis with backlash (t = 201-241s)

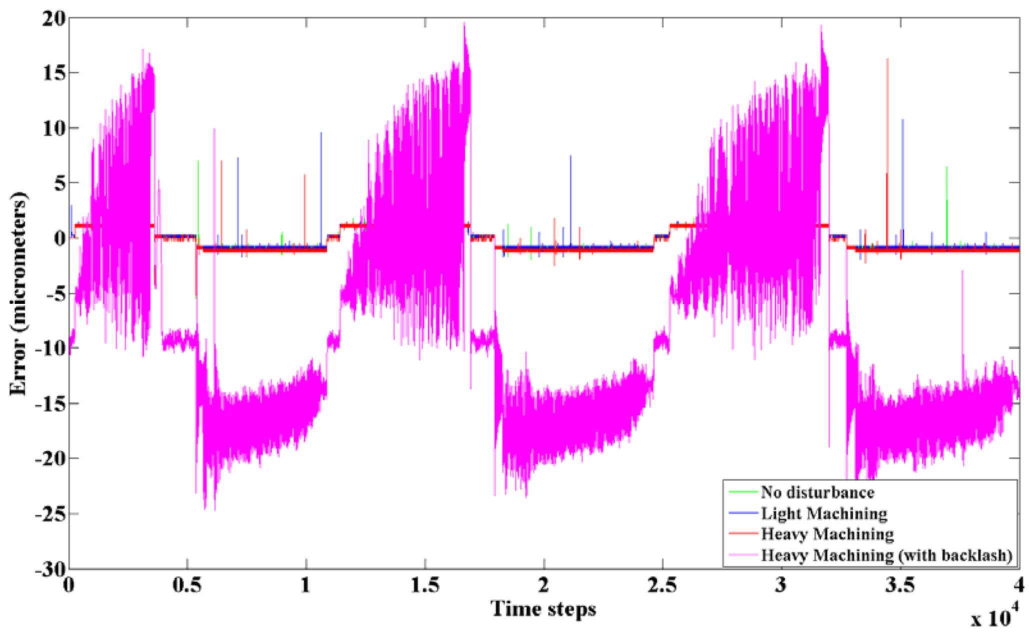


Figure 7.27 – Second trajectory cart position error in Z-axis with backlash (t = 201-241s)

As could be observed from the results, the controller seems successful in the trajectory tracking, as well as disturbance resistance. However, when backlash model is present in the HILS, the error significantly increases in the cart's position. Statistical data provided in Table 7.5 to Table 7.8 would be more useful in interpreting the results. Note that root mean square (RMS) is defined as:

$$\text{RMS} = \sqrt{\frac{1}{K}\sum_{k=0}^{K}[x(k) - x^*(k)]^2} \tag{7.16}$$

where K is the length of the data, x is the real value of the data and x* is the reference value of the data.

Table 7.5 – Root mean square and standard deviation values of the HILS results for first trajectory (in encoder counts)

Axis	**X**		**Y**		**Z**	
Dist. type	**RMS**	**STD***	**RMS**	**STD**	**RMS**	**STD**
No dist.	3.80	3.78	0.87	0.86	4.08	4.08
Light dist.	6.01	5.98	2.12	0.74	4.21	4.21
Heavy dist.	12.59	12.51	6.38	1.56	4.13	4.10

*STD: Standard deviation

Table 7.6 – Root mean square and standard deviation values of the HILS results for first trajectory (in μm)

Axis	**X**		**Y**		**Z**	
Dist. type	**RMS**	**STD**	**RMS**	**STD**	**RMS**	**STD**
No dist.	0.95	0.95	0.22	0.21	1.02	1.02
Light dist.	1.50	1.49	0.53	0.19	1.05	1.05
Heavy dist.	3.15	3.13	1.60	0.39	1.03	1.03

Table 7.7 – Root mean square and standard deviation values of the HILS results for second trajectory (in encoder counts)

Axis	**X**		**Y**		**Z**	
Dist. type	**RMS**	**STD**	**RMS**	**STD**	**RMS**	**STD**
No dist.	3.79	3.78	1.17	1.12	4.03	4.03
Light dist.	6.02	6.01	2.43	1.13	4.17	4.17
Heavy dist.	12.57	12.57	6.65	1.97	4.13	4.10

As can be seen from Table 7.5 to Table 7.8, the results are very successful in terms of mean error and standard deviation of the errors. Even in the heavy machining case, the maximum error appears in the X axis with an RMS value around 12.6 counts (3.15 μm) and a standard deviation around 12.5 counts (3.14 μm), when the HILS has no backlash model. However, when a backlash model between the ball screw and the cart is included in the HIL simulation, it directly increases the error, both in terms of RMS and STD, as can be seen in Table 7.8. This is an expected result since the controller has no effect on backlash compensation.

When the figures are observed, even there exist large errors in rare occasions; they are still within an acceptable range for the CNC machining center control task. Therefore, it can be concluded that the designed controller is successful in trajectory tracking, even under heavy machining condition.

Table 7.8 – Root mean square and standard deviation values of the HILS results for second trajectory including backlash model under heavy machining (in μm)

Axis	**X**		**Y**		**Z**	
Dist. type	**RMS**	**STD**	**RMS**	**STD**	**RMS**	**STD**
No dist.	0.948	0.95	0.29	0.28	1.01	1.01
Light dist.	1.506	1.50	0.61	0.28	1.04	1.04
Heavy dist.	3.14	3.14	1.66	0.49	1.03	1.03
Heavy d.+ BL*	55.94	55.94	49.17	13.56	12.49	9.38

(* d. + BL = disturbance + backlash)

7.5 Closure

In this chapter, most of the aforementioned modules and methods are utilized in an FPGA based motion controller and the results of the test case have proven that the proposed solution is a successful design. The results are comparable with industrial motion controllers in terms of performance and significantly efficient in terms of resource requirements. Therefore, it can be concluded that the proposed solution (FPGA based implementation) have proven to be a useful for motion control (or CNC) applications.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this chapter, an overall assessment of this thesis is presented, along with a number of suggestions that can follow and contribute to this study.

8.1 Conclusion

In this thesis, methods for developing an FPGA based motion control system are investigated. In this perspective, efficient and successful design methods are developed for each element of a typical motion control system, including peripheral interfaces and the controller parts. At certain points, some methods are highlighted as the best design approach; whereas at other points, methods have proven to be compatible with each other. However though, it can be seen that in the majority of this study, hardwired approach is adopted rather than the embedded processor design method.

The reasoning behind this selection can be justified as to obtain the maximum capability of the FPGA chip. Low-level design methodology required by hardwired approach allows the designer to customize a segment of the FPGA, to execute a specific task in an efficient manner. Furthermore, many instances of the designed module can also be used in parallel, as demonstrated in the previous chapters. However, utilizing an embedded processor for a certain task decreases the process time, consumes abundant amount of resources and eliminates the chance of parallel implementation.

On the other hand, employing an embedded processor has also certain advantages such as faster development and easy to use interfaces. Furthermore, when processing speed is not crucial and the design is complex, the embedded processor implementation may be comparable to hardwired design in terms of resource requirements. Therefore, in Chapters 4 and 5 of this thesis, both design methodologies have been evaluated and the results are presented for both approaches. Notice that in those chapters, state-space controller (and observer) and filter designs are presented, where processing speed is not crucial (clearly above a certain limit) and the design is relatively complex. Results approve that the resource consumptions are comparable and the process times significantly differ.

It can be observed that in this thesis two different FPGAs are employed from the two leading FPGA manufacturers; that are Altera (Cyclone II) and Xilinx (Virtex-5). However though, in the majority of the thesis (except Chapter 6) Altera's DE1 board with the Cyclone II (EP2C20F484C7N) chip is employed even though the Xilinx Virtex-5 is a more recent and resourceful chip. The reason for this choice is that development tools that are provided by Xilinx require significantly more time to compile, are more error-prone and harder to debug.

Certain advanced controller topologies are also investigated in this thesis, however due to the shortcomings of the Xilinx FPGA chip and lack of time, only a single method could be implemented to demonstrate the utilization of advanced controllers on an FPGA. Furthermore, the test case provided in Chapter 6 is also different from the test cases provided in Chapter 4 and 5. Nevertheless, the results have been successful and implementation of an advanced controller is demonstrated, even with a single method.

An important aspect of this thesis is that, all the chapters that include a design process contain a test case within itself, to demonstrate the success of the method. However, a test case is also provided in Chapter 7 for the assembled solution, which shows the success of the overall motion control system, including the peripheral interfaces as well as the controller. Therefore, the methods are both proven individually and as an assembled product. Notice that HILS is employed for testing the assembled solution, instead of a real test setup (a CNC machining center), however from the control system's point of view there is no difference between the two, since the simulator provides encoder signals and accepts PWM signal, as it would be in the real case. Although the HILS is not conducted real-time (due to limitations of the simulator) the proposed system is proven to be capable of real-time control, in the respective design chapters of the control system's elements. Therefore the results are evenly applicable to the real-time control case.

8.2 Future work

As addressed in the previous section, an important contribution to this study would be to utilize this system to control a real-world system, preferably a CNC machining center, in order to compare the results obtained via HILS. Although the HIL simulator reflects the real-world application in a very good manner, it is always desirable to utilize the designed control system in a real-world control application.

In this study, it can be deduced that the conventional control methodologies are covered quite thoroughly in Chapters 4 and 5. However in Chapter 6, only a single method could be presented for the advanced controller implementations. As a future study, other methods provided in the previous Chapters could be implemented to obtain a more detailed discussion between different designs. Furthermore, it would also be more meaningful if the study is conducted on an Altera Cyclone II FPGA, to provide a comparison between the conventional and intelligent controllers' implementation on the same chip.

As a final remark, it should also be noted that in Chapter 7, due to the unavailability of a command generator, the reference commands are written to SRAM before the simulation; which resulted in a limitation of the simulation time. However, as a future work, the commands could be received from an outside source to run the full-time simulation, utilizing the developed modules that are presented in Chapter 3.

REFERENCES

[1] R. Usselmann [1], Open Floating Point Unit Manual, www.opencores.org, 2000.

[2] Arbit, A.; Pritzker, D.; Kuperman, A.; Rabinovici, R.; , "A DSP-controlled PWM generator using field programmable gate array," Electrical and Electronics Engineers in Israel, 2004. Proceedings. 2004 23rd IEEE Convention of , pp. 325- 328, 6-7 Sept. 2004

[3] Xu Dong; Wang Tianmiao; Wei Hongxing; Liu Jingmeng, "A new dual-core Permanent Magnet Synchronous Motor Servo System," Industrial Electronics and Applications, 2009. ICIEA 2009. 4th IEEE Conference on , pp.715-720, 25-27 May 2009.

[4] Birou, I.; Imecs, M., "Real-time robot drive control with PM-synchronous motors using a DSP-based computer system," Power Electronics and Motion Control Conference, Proceedings of the Third International IPEMC 2000., vol.3, pp.1290-1295 vol.3, 2000.

[5] Geun-Hyung Lee; Sung-Su Kim; Seul Jung; , "Hardware Implementation of a RBF Neural Network Controller with a DSP 2812 and an FPGA for Controlling Nonlinear Systems," Smart Manufacturing Application, 2008. ICSMA 2008. International Conference on , pp.167-171, 9-11 April 2008

[6] Morales-Caporal, R.; Pacas, M.; , "Digital implementation of a direct mean torque control for AC servo drives based on a hybrid DSP/FPGA controller system," Power Electronics Congress, 2008. CIEP 2008. 11th IEEE International , pp.77-83, 24-27 Aug. 2008

[7] DMC-18x6 reference manual, Galil Motion Control Co., 2010

[8] PMAC2 hardware reference manual, Delta-Tau Co., 2010

[9] Al-Ayasrah, O.; Alukaidey, T.; Pissanidis, G.; , "DSP Based N-Motor Speed Control of Brushless DC Motors Using External FPGA Design," Industrial Technology, 2006. ICIT 2006. IEEE International Conference on , pp.627-631, 15-17 Dec. 2006

[10] Toh, C.L.; Idris, N.R.N.; Yatim, A.H.M.; Muhamad, N.D.; Elbuluk, M.; , "Implementation of a New Torque and Flux Controllers for Direct Torque Control (DTC) of Induction Machine Utilizing Digital Signal Processor (DSP) and Field Programmable Gate Arrays (FPGA)," Power Electronics

Specialists Conference, 2005. PESC '05. IEEE 36th , pp.1594-1599, 16-16 June 2005

[11] Seul Jung; Sung su Kim, "Hardware Implementation of a Real-Time Neural Network Controller With a DSP and an FPGA for Nonlinear Systems," IEEE Transactions on Industrial Electronics, vol.54, no.1, pp.265-271, Feb. 2007.

[12] Kaiping Yu,; Hong Guo,; Dayu Wang,; Lanfeng Li,; , "Design of multi-redundancy electro-mechanical actuator controller with DSP and FPGA," Electrical Machines and Systems, 2007. ICEMS. International Conference on , pp.584-587, 8-11 Oct. 2007

[13] Esmaeli, A.; Li Bo; Sun Li; , "A Novel AC Servo System Implementation," 9th International Multitopic Conference, IEEE INMIC 2005 , pp.1-5, 24-25 Dec. 2005

[14] Ni, F.L.; Jin, M.H.; Xie, Z.W.; Shi, Sh.C.; Liu, Y.Ch.; Liu, H.; Hirzinger, G., "A Highly Integrated Joint Servo System Based on FPGA with Nios II Processor," Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation, pp.973-978, 25-28 June 2006.

[15] Li, Yan; Zhuang, Shengxian; Zhang, Luan, "Development of an FPGA-Based Servo Controller for PMSM Drives,", 2007 IEEE International Conference on Automation and Logistics, vol. 18, no. 21, pp.1398-1403, Aug. 2007.

[16] Ying-Shieh Kung; Rong-Fong Fung; Ting-Yu Tai, "Realization of a Motion Control IC for X-Y Table Based on Novel FPGA Technology," IEEE Transactions on Industrial Electronics, vol.56, no.1, pp.43-53, Jan. 2009.

[17] Das, A.; Banerjee, K.; "Fast prototyping of a digital PID controller on a FPGA based softcore microcontroller for precision control of a brushed DC servo motor," Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE , pp.2825-2830, 3-5 Nov. 2009

[18] Ben Salem, A.K.; Ben Othman, S.; Ben Saoud, S.; Litayem, N.; , "Servo drive system based on programmable SoC architecture," Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE , pp.2961-2966, 3-5 Nov. 2009

[19] Jung Uk Cho; Quy Ngoc Le; Jae Wook Jeon, "An FPGA-Based Multiple-Axis Motion Control Chip," IEEE Transactions on Industrial Electronics, vol.56, no.3, pp.856-870, March 2009.

[20] Chan, Y.F.; Moallem, M.; Wang, W., "Efficient implementation of PID control algorithm using FPGA technology," 43rd IEEE Conference on Decision and Control, vol.5, pp. 4885-4890 Vol.5, 14-17 Dec. 2004.

[21] Yao dong Tao; Hu Lin; Yi Hu; Xiaohui Zhang; Zhicheng Wang, "Efficient implementation of CNC Position Controller using FPGA,".. 6th IEEE International Conference on Industrial Informatics (INDIN 2008), pp.1177-1182, 13-16 July 2008.

[22] Jia Lanping; Zhou Runjing; Liang Zhian, "Realization of position tracking system based on FPGA," 2004. Proceedings. ICSP '04. 2004 7th International Conference on Signal Processing, vol.3, pp. 2588-2591 vol.3, 31 Aug.-4 Sept. 2004.

[23] Ying-Shieh Kung; Ming-Shyan Wang; Chung-Chun Huang; , "Digital Hardware Implementation of Adaptive Fuzzy Controller for AC Motor Drive," Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE , pp.1208-1213, 5-8 Nov. 2007

[24] Faa-Jeng Lin; Ying-Chih Hung, "FPGA-based Elman Neural Network Control System for Linear Ultrasonic Motor," IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency, vol.56, no.1, pp.101-113, January 2009.

[25] Ying-Shieh Kung; Ming-Shyan Wang; Tzu-Yao Chuang; , "FPGA-based self-tuning PID controller using RBF neural network and its application in X-Y table," Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on , pp.694-699, 5-8 July 2009

[26] Liaw, C.-M.; Wang, J.-B., "Design and implementation of a fuzzy controller for a high performance induction motor drive," Systems, Man and Cybernetics, IEEE Transactions on , vol.21, no.4, pp.921-929, Jul/Aug 1991.Arbit, A.; Pritzker, D.; Kuperman, A.; Rabinovici, R.; , "A DSP-controlled PWM generator using field programmable gate array," Electrical and Electronics Engineers in Israel, 2004. Proceedings. 2004 23rd IEEE Convention of , pp. 325- 328, 6-7 Sept. 2004

[27] Alexei, Z.; Sandor, H., "Robust speed fuzzy logic controller for DC drive," Intelligent Engineering Systems, 1997. INES '97. Proceedings., 1997 IEEE International Conference on , pp.385-389, 15-17 Sep 1997.

[28] Sio, K.C.; Lee, C.K., "Identification of a nonlinear motor system with neural networks," Advanced Motion Control, 1996. AMC '96-MIE. Proceedings., 1996 4th International Workshop on , vol.1, pp.287-292 vol.1, 18-21 Mar 1996.

[29] Slotine, J.J.E., and Li, W., Applied Nonlinear Control, Prentice-Hall, 1991.

[30] Brock, S.; Deskur, J.; Zawirski, K., "Modified sliding-mode speed controller for servo drives," Industrial Electronics, 1999. ISIE '99. Proceedings of the IEEE International Symposium on , vol.2, pp.635-640 vol.2, 1999.

[31] Usenmez, S.; Dilan R.A; Dolen M.; Koku A.B., “Real-Time Hardware-in-the-Loop Simulation of Electrical Machine Systems Using FPGAs”, to appear in the Proc. of the International Conference on Electrical Machines and Systems (ICEMS), 2009.

[32] Usenmez, S. “Design of an integrated hardware-in-the-loop simulation system”, Master of Science Thesis, Department of Mechanical Engineering, Middle East Technical University, Graduate School of Natural and Applied Sciences, 2010.

APPENDIX A

LIST OF VERILOG HDL FILES

In this appendix, a list of Verilog HDL files that are employed in different parts of this study is provided, along with their related chapters and corresponding functions.

Table A.1 – List of Verilog HDL files employed in the thesis

Name	**Related chapter**	**Function**
`encoder.v`	Chapter 3	Incremental encoder interface
`pwm_gen_v2.v`	Chapter 3	PWM signal generator
`pwm_transmit_v2.v`	Chapter 3	PWM data transmitter
`pwm_receive_v2.v`	Chapter 3	PWM data receiver
`clk_divider.v`	Chapter 3	Clock generator
`pulse_gen_v2.v`	Chapter 3	Finite pulse generator
`SPI.v`	Chapter 3	SPI slave module
`command_recv.v`	Chapter 3	Custom parallel data receiver
`sasc_top.v`	Chapter 3	RS-232 controller
`sram_ctrl.v`	Chapter 3	SRAM controller
`imul.v`	Chapter 4	Integer multiplication module
`fmul.v`	Chapter 4	Fixed-point multiplication module
`fpu.v`	Chapter 4	Floating point unit
`mul_kx.v`	Chapter 4	Fractional / Quantized multiplication module
`filter_v2.v`	Chapter 5	Generic filter module

APPENDIX B

LIST OF C CODES FOR NIOS II IMPLEMENTATION

In this appendix, a list of C codes developed for Nios II softcore processor implementation of methods presented in Chapter 4 and Chapter 5 is presented.

Table B.1 – List of Nios II C files employed in the thesis

Name	**Related chapter**	**Function**
`nios_ss_imul.c`	Chapter 4	Method II-a implementation
`nios_ss_fmul.c`	Chapter 4	Method II-b implementation
`nios_ss_fpu.c`	Chapter 4	Method II-c implementation
`nios_filter_imul.c`	Chapter 5	Method II-a implementation
`nios_filter_fmul.c`	Chapter 5	Method II-b implementation
`nios_filter_fpu.c`	Chapter 5	Method II-c implementation

APPENDIX C

SAMPLE C CODE FOR NIOS II IMPLEMENTATION

In this appendix, a sample C code developed for Nios II softcore processor implementation of method II-a presented in Chapter 4 is presented.

```
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include "altera_avalon_performance_counter.h"
#define perfctr_base (void*)0x00900020

short readshort();
unsigned short readshortu();
void writeshort(short val);
void writeint32(int val);
int fmul(short k_int, short k_frac, int x);

int main()
{
  signed int perfctr_count;
  unsigned int perfctr_rate = alt_get_cpu_freq();
  int i,j,k;

  short K_ni[4];
  short K_df[4];
  short F_ni[4][4];
  short F_df[4][4];
  short G_ni[4];
  short G_df[4];
  short L_ni[4][4];
  short L_df[4][2];
  short x_read[2];
  int x[4];
  int xkm1[4];

  int u = 0;
  int ukm1 = 0;

      for(k=0;k<4;k++){
        xkm1_f[k] = 0;
        xkm1[k] = 0;
        x_f[k] = 0;
        x[k] = 0;
    }


  while(1){
```

```
  for(i=0;i<4;i++){
  K_ni[i] = readshort();
  K_df[i] = readshort();
  }


  for(i=0;i<4;i++){
      for(j=0;j<4;j++){
        F_ni[i][j] = readshort();
        F_df[i][j] = readshort();
      }
  }

  for(i=0;i<4;i++){
  G_ni[i] = readshort();
  G_df[i] = readshort();
  }

  for(i=0;i<4;i++){
      for(j=0;j<2;j++){
        L_ni[i][j] = readshort();
        L_df[i][j] = readshort();
      }
  }

  for(i=0;i<200;i++){

  x_read[0] = readshort();
  x_read[1] = readshort();

  PERF_RESET(perfctr_base);
  PERF_START_MEASURING(perfctr_base);

// SS with int

  for(k=0;k<4;k++){

      for(j=0;j<4;j++){
      x[k] = x[k] + (F_ni[k][j]*xkm1[j])/F_df[k][j];
      }

      x[k] = x[k] + (G_ni[k]*ukm1/G_df[k]);

      for(j=0;j<2;j++){
      if(j==0) x[k] = x[k] + (L_ni[k][j]*(x_read[j]-
xkm1[0])/L_df[k][j]);
      if(j==1) x[k] = x[k] + (L_ni[k][j]*(x_read[j]-
xkm1[2])/L_df[k][j]);
      }
  }

  for(k=0;k<4;k++){
  u = u + (K_ni[k]*xkm1[k]/K_df[k]);
  }

  PERF_STOP_MEASURING(perfctr_base);
  perfctr_count = perf_get_total_time(perfctr_base);

  writeint32(u);
  writeint32(perfctr_count);

    ukm1 = u;
    u = 0;

    for(k=0;k<4;k++){
        xkm1_f[k] = x_f[k];
        xkm1[k] = x[k];
        x_f[k] = 0;
        x[k] = 0;
    }
  }
  } //End of while(1)
  return 0;
}

short readshort()
{
short val;
short temp;
val = getchar();
temp = getchar();
val = val | (temp << 8);
return val;
}

void writeshort(short val)
{
    putchar((char)val);
    usleep(2000);
    putchar((char)(val >> 8));
}

void writeint32(int val)
{
    putchar((char) val);
    usleep(2000);
    putchar((char)(val >> 8));
    usleep(2000);
    putchar((char)(val >> 16));
    usleep(2000);
    putchar((char)(val >> 24));
}
```

APPENDIX D

LIST OF MATLAB M-FILES

In this appendix, a list of MATLAB M-files developed for HILS employed in Chapter 4 and Chapter 5 is presented.

Table D.1 – List of MATLAB M-files employed in the thesis

Name	**Related chapter**	**Function**
`sim_ss_imul.m`	Ch. 4	Sending initial parameters to FPGA and realizing HILS for hardwired implementation
`sim_ss_fmul.m`	Ch. 4	
`sim_ss_fpu.m`	Ch. 4	
`sim_ss_nios_imul.m`	Ch. 4	Sending initial parameters to FPGA and realizing HILS for softcore implementation
`sim_ss_nios_fmul.m`	Ch. 4	
`sim_ss_nios_fpu.m`	Ch. 4	
`sim_filter_imul.m`	Ch. 5	Sending initial parameters to FPGA and realizing HILS for hardwired implementation
`sim_filter_fmul.m`	Ch. 5	
`sim_filter_fpu.m`	Ch. 5	
`sim_filter_nios_imul.m`	Ch. 5	Sending initial parameters to FPGA and realizing HILS for softcore implementation
`sim_filter_nios_fmul.m`	Ch. 5	
`sim_filter_nios_fpu.m`	Ch. 5	

APPENDIX E

SAMPLE MATLAB M-FILE FOR HILS

In this appendix, a sample M-file (`sim_ss_nios_imul.m`)is presented.

```
clc;
clear;
s = serial('COM5');
set(s,'BaudRate', 57600, 'DataBits', 8, 'Parity',
'none','StopBits', 1, 'FlowControl', 'none');

state_no = 4; %Hence the total number of 16-bit packages to send
is equal to 8 (state_no*2(for K matrix)).
known_state_no = 2;
simulation_time = 20; %in seconds
t_sample = 0.001; %in seconds

state_history = simulation_time/t_sample;

ctr1=1;
T_new = zeros(state_history,1);
F_new = zeros(state_history,1);
F_read = zeros(state_history,1);
F_dene = zeros(state_history,1);
Y_new = zeros(state_history,4);
Y_new(1,1) = 0; %Set inital position of the cart
Y_new(1,3) = -0.05; %Set inital position of the pendulum
Y_send = zeros(state_history,2);
Y_read = zeros(state_history,3);
states = zeros(state_history,4);
X_current = zeros(4,1);

K = [-0.9975   -2.3195   29.0515   17.6788]; %define K Matrix

[K_n,K_d] = rat(K);

F_matrix = [1.0000    0.0010    0.0000    0.0000;
         0    0.9998    0.0027    0.0000;
         0   -0.0000    1.0000    0.0010;
         0   -0.0005    0.0312    1.0000];

[F_n,F_d] = rat(F_matrix);

G= [0.0000;
     0.0018;
     0.0000;
     0.0045];
```

```
[G_n,G_d] = rat(G);


L=     1.0e+002 * [0.0026   -0.0000;
    1.7286   -0.0153;
   -0.0000    0.0026;
   -0.0110    1.7294];

[L_n,L_d] = rat(L);

F = 0;
u = 0.2;

Y_send(1,1) = round(Y_new(1,1)*10000); %x = cart position
Y_send(1,2) = round(Y_new(1,3)*10000); %theta = angular position
of pendulum

fopen(s);

%IMUL

for i1=1:1:state_no
   fwrite(s, K_n(i1), 'int16');
   fwrite(s, K_d(i1), 'int16');
end

pause(0.01);

for i1=1:1:state_no
   for i2=1:1:state_no
   fwrite(s, F_n(i1,i2), 'int16');
   fwrite(s, F_d(i1,i2), 'int16');
   end
end

pause(0.01);

for i1=1:1:state_no
   fwrite(s, G_n(i1,1), 'int16');
   fwrite(s, G_d(i1,1), 'int16');
end

pause(0.01);

for i1=1:1:state_no
    for i2=1:1:known_state_no
    fwrite(s, L_n(i1,i2), 'int16');
    fwrite(s, L_d(i1,i2), 'int16');
    end
end
pause(0.01);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
for i=t_sample:t_sample:simulation_time

fwrite(s, Y_send(ctr1,1), 'int16');
fwrite(s, Y_send(ctr1,2), 'int16');
Y_read(ctr1,1) = fread(s,1,'int32');
Y_read(ctr1,2) = fread(s,1,'int32');

F = (-Y_read(ctr1,1)/10000);

[T,Y] = ode45(@(t,y) myode(t,y,F),[i-t_sample i],[Y_new(ctr1,1)
Y_new(ctr1,2) Y_new(ctr1,3) Y_new(ctr1,4)]);

Y_new(ctr1+1,1) = Y(size(T,1),1);
Y_new(ctr1+1,2) = Y(size(T,1),2);
Y_new(ctr1+1,3) = Y(size(T,1),3);
Y_new(ctr1+1,4) = Y(size(T,1),4);

X_current(1,1) = Y_new(ctr1+1,1);
X_current(2,1) = Y_new(ctr1+1,2);
X_current(3,1) = Y_new(ctr1+1,3);
X_current(4,1) = Y_new(ctr1+1,4);

Y_send(ctr1+1,1) = round((Y_new(ctr1+1,1))*10000); %x = cart
position
Y_send(ctr1+1,2) = round((Y_new(ctr1+1,3))*10000); %theta =
angular position of pendulum

if(abs(Y_send(ctr1+1,1)) > 32765)
    Y_send(ctr1+1,1) = sign(Y_send(ctr1+1,1))*32765;
end

if(abs(Y_send(ctr1+1,2)) > 32765)
    Y_send(ctr1+1,2) = sign(Y_send(ctr1+1,2))*32765;
end
F_new(ctr1+1,1) = F;
T_new (ctr1,1) = i;
ctr1 = ctr1 + 1;
end
fclose(s);

Y_new = Y_new(1:state_history,1:4);
F_new = F_new(1:state_history,1);
Y_send = Y_send(1:state_history,1:2);

plot(T_new, Y_new(:,3));
hold on;
plot(T_new, Y_new(:,1),'r');
title('Plot of x as a function of time');
xlabel('Time'); ylabel('Y(t)');
```

DESIGN OF ADVANCED MOTION COMMAND GENERATORS UTILIZING FPGA

ULAŞ YAMAN

JUNE 2010

DESIGN OF ADVANCED MOTION COMMAND GENERATORS UTILIZING FPGA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ULAŞ YAMAN

IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

JUNE 2010

Approval of the thesis:

DESIGN OF ADVANCED MOTION COMMAND GENERATORS UTILIZING FPGA

submitted by **ULAŞ YAMAN** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _________________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Suha Oral _________________
Head of Department, **Mechanical Engineering**

Assist. Prof. Dr. Melik Dölen _________________
Supervisor, **Mechanical Engineering Dept., METU**

Assist. Prof. Dr. A. Buğra Koku _________________
Co-Supervisor, **Mechanical Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Mehmet Çalışkan _________________
Mechanical Engineering Dept., METU

Assist. Prof. Dr. Melik Dölen _________________
Mechanical Engineering Dept., METU

Assist. Prof. Dr. A. Buğra Koku _________________
Mechanical Engineering Dept., METU

Assist. Prof. Dr. E. İlhan Konukseven _________________
Mechanical Engineering Dept., METU

Assoc. Prof. Dr. Veysel Gazi _________________
Electrical and Electronics Engineering Dept., TOBB-ETU

Date: _________________

ABSTRACT

DESIGN OF ADVANCED MOTION COMMAND GENERATORS UTILIZING FPGA

Yaman, Ulaş
M.Sc., Department of Mechanical Engineering
Supervisor : Assist. Prof. Dr. Melik Dölen
Co-Supervisor : Assist. Prof. Dr. A. Buğra Koku

June 2010, 143 pages

In this study, universal motion command generator systems utilizing a Field Programmable Gate Array (FPGA) and an interface board for Robotics and Computer Numerical Control (CNC) applications have been developed. These command generation systems can be classified into two main groups as polynomial approximation and data compression based methods. In the former type of command generation methods, the command trajectory is firstly divided into segments according to the inflection points. Then, the segments are approximated using various polynomial techniques. The sequence originating from modeling error can be further included to the generated series. In the second type, higher-order differences of a given trajectory (i.e. position) are computed and the resulting data are compressed via lossless data compression techniques. Besides conventional approaches, a novel compression algorithm is also introduced in the study. This group of methods is capable of generating trajectory data at variable rates in forward and reverse directions. The generation of the commands is carried out according to the feed-rate (i.e. the speed along the trajectory) set by the external logic dynamically. These command generation techniques are implemented in MATLAB and then the best ones from each group are realized using FPGAs and their performances are assessed according to the resources used in the FPGA chip, the speed of command generation, and the memory size in Static Random Access Memory (SRAM) chip located on the development board.

Keywords: Command Generation, Data Compression, FPGA, Polynomial Approximation, Adjustable Feed-rate, Linear Interpolation

ÖZ

FPGA KULLANARAK İLERİ HAREKET KOMUT
ÜRETEÇLERİ TASARIMI

Yaman, Ulaş
Yüksek Lisans, Makina Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Melik Dölen
Ortak Tez Yöneticisi : Yrd. Doç. Dr. A. Buğra Koku

Haziran 2010, 143 sayfa

Bu çalışmada, Alan Programlanabilir Kapı Dizini (Field Programmable Gate Array - FPGA) kullanan evrensel hareket komutu üreteçleri ve Robotik / Bilgisayarlı Sayısal Denetim ("Computer Numerical Control" - CNC) uygulamaları için bir arayüz kartı geliştirilmiştir. Geliştirilen komut üreteç sistemleri, fonksiyon yaklaşıklama ve veri sıkışırtırma tabanlı olmak üzere iki sınıfa ayrılabilir. Fonksiyon yaklaşıklama tabanlı komut üreteç uygulamalarında, komut dizini öncelikli olarak bükülme noktalarından bölümlere ayrılmıştır. Daha sonrasında bu bölümler farklı fonksiyon yaklaşıklama yöntemleri kullanılarak ifade edilmiştir. Yaklaşıklamadan kaynaklanan hata dizini kodlama sırasında saklanarak üretilen komutlara beslenebilir. Diğer komut üretme yöntemlerinde ise, verilen hareket dizininin yüksek dereceden farkı alındıktan sonra kayıpsız veri sıkıştırma teknikleri kullanılarak sıkıştırılır. Bu çalışmada, geleneksel sıkıştırma tekniklerinin yanı sıra yeni bir veri sıkıştırma yöntemi de sunulmuştur. Bu grupta önerilen yöntemler, komutları ileri ve geri yönlerinde farklı hızlarda üretebilme yetilerine sahiptirler. Komut üretim hızı sisteme dışarıdan dinamik olarak beslenmektedir.

Geliştirilen komut üretme teknikleri MATLAB kullanılarak bilgisayar ortamında gerçekleştirilmiş ve her grupta en iyi sonucu veren yöntemler FPGA kullanılarak gerçekleştirilmiştir. Bu yöntemler FPGA kırmığı üzerinde kullandıkları kaynaklar, komut üretim hızı ve geliştirme kartında bulunan Durağan Rastgele Erişimli Bellek'te (Static Random Access Memory - SRAM) depolanan verinin büyüklüğüne göre değerlendirilmişlerdir.

Anahtar kelimeler: Komut Üretimi, Veri Sıkıştırma, Alan Programlanabilir Kapı Dizini, Fonksiyonel Yaklaşıklama, Ayarlanabilir Üretim Hızı, Doğrusal Enterpolasyon

Alevlerin ucunda sönen hayatlara,

"Yaşamayı ciddiye alacaksın,
yani o derecede, öylesine ki,
mesela, kolların bağlı arkadan, sırtın duvarda,
yahut kocaman gözlüklerin,
beyaz gömleğinle bir laboratuvarda,
            insanlar için ölebileceksin,
hem de yüzünü bile görmediğin insanlar için,
hem de hiç kimse seni buna zorlamamışken,
hem de en güzel en gerçek şeyin
            yaşamak olduğunu bildiğin halde."

Nazım Hikmet RAN

ACKNOWLEDGEMENTS

First of all I offer my heartily thanks to my supervisor Assist. Prof. Dr. Melik Dölen and co-supervisor Assist. Prof. Dr. A. Buğra Koku, who have supported me throughout the thesis with their unyielding patience and knowledge.

I would like to thank to the member of the SPARC research group for their technical support throughout the thesis period. This period might have been so painful without their support and advices.

Additionally, I would like to show gratitude to my colleagues at the department for their sincere friendships especially during the writing process of the thesis and the meal times.

I would like to thank TÜBİTAK for the scholarship (BİDEB 2228) and the conference travel grant during my graduate study. My study was also a part of TÜBİTAK project under the contract number 108E048.

Lastly, I am deeply in debt to my parents Şahhanım and Azimet Yaman for their never-ending love and spiritual support at critical and opportune times.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

SYMBOLS

B	Bernstein Polynomials
c_{max}	maximum value of coefficients
c_{min}	minimum value of coefficients
d_{max}	maximum value of original data sequence (counts)
d_{min}	minimum value of original data sequence (counts)
e_{max}	maximum value of error sequence (counts)
e_{min}	minimum value of error sequence (counts)
f	feed-rate (mm/s)
f_{max}	maximum feed-rate (mm/s)
L	Legendre Polynomials
l	binary length of encoded data (bits)
m	group size of Golomb coding
N	length of original data sequence
n	order of finite difference
∇	finite difference

n_0	number of zero magnitude data
N_c	number of coefficients
q	motion-state sequence
r	compression ratio
r_e	compression of error sequence
T	Chebyshev Polynomials
u	decoded command

ABBREVIATIONS

ADC	Analog-to-Digital Converter
ASM	Algorithmic State Machine
BJT	Bipolar Junction Transistor
BP	Bernstein Polynomials
CAM	Computer Aided Manufacturing
CD	Clock Divisor
CLDATA	Cutter Location Data
CMOS	Complementary Metal Oxide Semiconductor
CNC	Computer Numerical Control
CP	Chebyshev Polynomials

CTM	Command Transmit Module
DAC	Digital-to-Analog Converter
DDFS	Direct Digital Frequency Synthesizers
DM	Driver Module
DSP	Digital Signal Processor
DU	Decoding Unit
FPGA	Field Programmable Gate Array
FPOM	Floating Point Operation Module
FSM	Finite State Machine
GTL	Gunning Transceiver Logic
GTLP	Gunning Transceiver Logic Plus
HILS	Hardware-in-the-Loop Simulator
I/O	Input / Output
IP	Intellectual Property
JSD	Joint State Data
LED	Ligth Emitting Diode
LP	Legendre Polynomials
LVDS	Low Voltage Differential Signaling
LVPECL	Low Voltage Positive Emitter Coupled Logic

LZ	Lempel Ziv
MMU	Memory Management Unit
NC	Numerical Control
PWM	Pulse Width Modulation
PWMG	Pulse Width Modulation Generator
RMS	Root Mean Square
SDRAM	Synchronous Dynamic Random Access Memory
SISO	Serial Input Serial Output
SIPO	Serial Input Serial Output
SM	Splitter Module
SR	Shift Register
SRAM	Static Random Access Memory
SW	Sine Wave
TTL	Transistor-to-Transistor Logic
VHDL	Very High Speed Integrated Circuit Hardware Description Language

CHAPTER 1

INTRODUCTION

1.1 Motivation

Modern servo-drive systems employ digital motion controllers (DSPs, micro-controllers) to regulate precisely not only motor currents (electromagnetic torque) but also motor's angular velocity along with the position. If the drive system is configured for (digital) motion control, the relevant reference signals (velocity or position) must be generated by a central controller unit (host) depending on the trajectory to be followed. These signals are eventually transferred to each motor driver via a serial communication protocol (SERCOS, CAN, Profibus, TCP/IP, RS-232, RS-485, etc.). This approach frequently pushes the communication interface to its limits for high-end applications.

Industrial motion controller units utilize vector data tables to represent the trajectory in terms of linear patches. These cards can then perform a linear interpolation between the two consecutive entries in real-time to produce the relevant reference signals for the position servo-control loop. For complex trajectories the size of the vector table may exceed the available resources on the system. The conventional machining approach does not meet the requirements of high speed and high accurate machining in cases where the trajectories are complicated.

In today's technology, memory devices with large capacity as well as multi-core processors running at high clock frequencies are widely available in the market at relatively low cost. Consequently, there is a potential for devising simple yet very effective command generators for computer numerically controlled machinery that benefit fully from the properties of these advanced devices [1].

The fundamental motivation of this study is to develop a direct command generator system with variable feed-rate in forward and reverse directions for servo motor drives where the commands could be produced directly in the drive system without the need for intermittent data transfer from a host controller. This FPGA based system, which could be directly embedded into a motor drive system, is expected to generate the relevant commands by utilizing not only the (dynamically adjusted) speed along the traced trajectory but also decompressed data being produced in advance to represent trajectory to the desired accuracy.

1.2 Scope of the Thesis

The difficulties mentioned in the previous section are overcome by the proposed method in this study. It is implemented into the conventional manufacturing process as illustrated in Figure 1-1. The figure indicates that the method is applicable for both Robotics and CNC Applications. Input to the system is Tool Location Data (TLD) for CNC Applications and Joint State Data (JSD) for Robotics Applications. After the encoder commands are formed according to the incoming data from the previous operations, the encoding algorithm is applied onto the commands structuring the encoded JSD or TLD. Encoding methods applied within the context of this thesis are

- Lossless Data Compression of Higher Order Finite Differences of JSD / TLD
    - Huffman Compression Algorithm
    - Arithmetic Coding

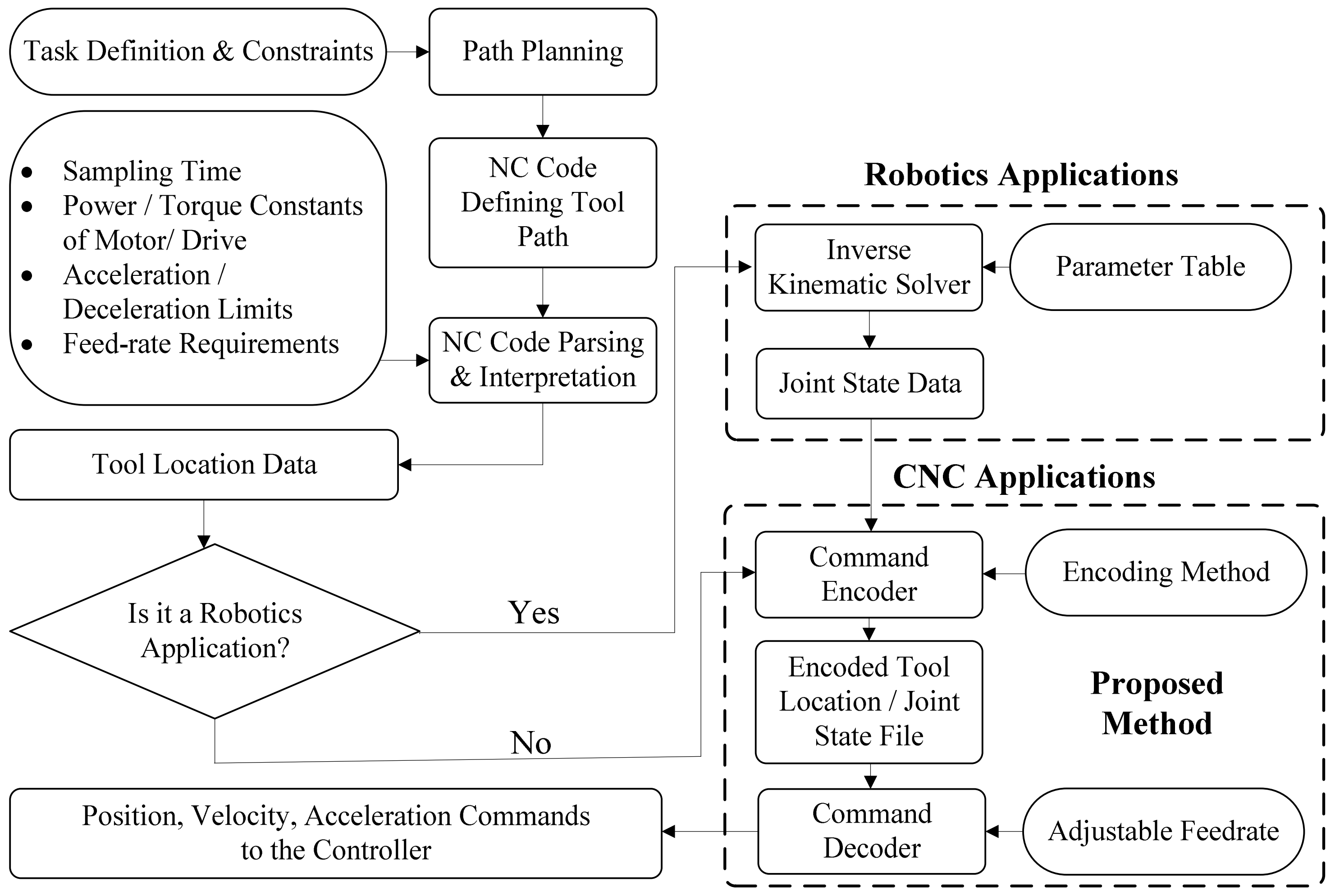

Task Definition & Constraints
Path Planning
Sampling Time
Power / Torque Constants of Motor/ Drive
Acceleration / Deceleration Limits
Feed-rate Requirements
NC Code Defining Tool Path
NC Code Parsing & Interpretation
Tool Location Data
Is it a Robotics Application?
Yes
No
Robotics Applications
Inverse Kinematic Solver
Parameter Table
Joint State Data
CNC Applications
Command Encoder
Encoding Method
Encoded Tool Location / Joint State File
Proposed Method
Command Decoder
Adjustable Feedrate
Position, Velocity, Acceleration Commands to the Controller

Figure 1-1 Flow Chart of the Proposed Method

    - Proposed Compression (ΔY) Algorithm
- Polynomial Representation of Segmented JSD / TLD
    - Chebyshev Polynomials
    - Legendre Polynomials
    - Bernstein Polynomials

The performances of these aforementioned encoding methods are comparatively evaluated according to approximation error, memory requirement, computational complexity, ease of decoding, etc. in MATLAB environment. The best methods for each group are realized on the FPGA Development Board. Once the encoded JSD / TLD is stored into the SRAM of the board, the decoding circuit in the FPGA chip is ready to decode the data in real-time. Consequently, decoded position, velocity, and acceleration commands can be fed to the (centralized or distributed) joint-axis motion controller as the reference signals.

The arrangement of the proposed command generation method in the thesis with the other units of the FPGA based motion control system is illustrated in Figure 1-2. This general motion control system is composed of three main units: FPGA Based Command Generation System, FPGA Based Motion Controller System, and Hardware-in-the-Loop Simulator (HILS). The last two units are not within the scope of the thesis. Only the shaded area in Figure 1-2 is elaborated throughout the thesis.

The generated reference commands (position, velocity, and acceleration) are transferred to the second FPGA board in the system which is responsible for motion control. The velocity and acceleration commands are also sent to the motion control unit, since the control algorithm employed within the FPGA may require these commands for precise control. Then the generated control inputs are fed to the HILS unit through the developed FPGA Input / Output (I/O) Interface Board. This board is also discussed in the thesis. After the manipulated input vector is transferred to the HILS, the control process is simulated and the resulted output vector is fed to the controller unit with the help sensors utilization.

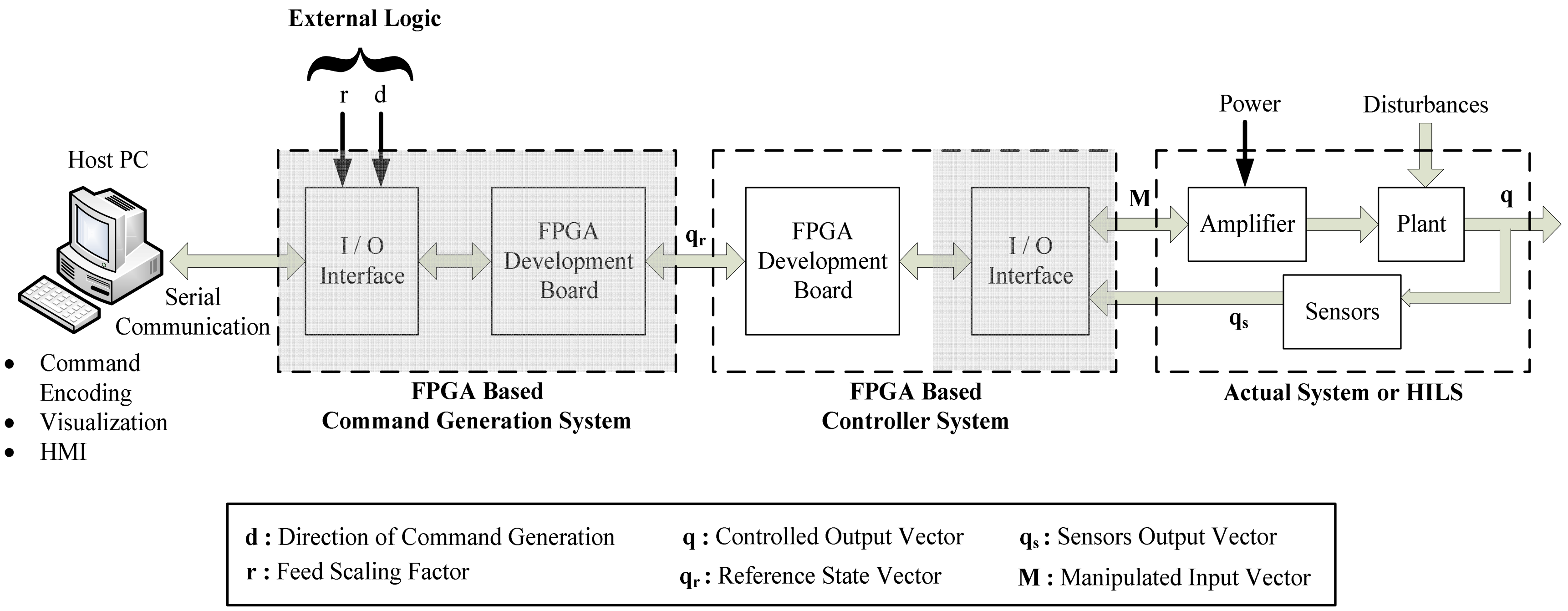


Figure 1-2 FPGA Based Motion Control System

1.3 Organization

The thesis is divided into seven chapters. The second chapter discusses the literature on the relevant topics of the thesis such as differencing, data compression algorithms, polynomial approximation techniques, FPGA implementation of these methods, FPGA-based command generation systems, and command generators with variable feed-rates. In the following chapter, a board developed for FPGA interfacing with CNC and Robotics Applications is investigated. This interface board consists of five modules namely mother board, analog input card, analog output card, digital input card, and digital output card. The fourth chapter elaborates the first group of command generation methods utilizing segmentation and polynomial approximation in MATLAB. The approximation method with Chebyshev Polynomials, which has the smallest compression ratio, is employed on the FPGA Development Board using two different approaches. In the first approach, the algorithm is implemented with a softcore embedded processor in the FPGA chip and in the second approach the algorithm is directly realized in the FPGA chip utilizing Very High Speed Integrated Circuit Hardware Description Language (VHDL). In the fifth chapter the second group of command generation methods utilizing differencing and compression are explained and their performances are evaluated in MATLAB. The most successful one of these compression algorithms, namely the proposed compression method, is realized on the FPGA Development Board in two different ways as done in the previous chapter. In this chapter, the concept of command generation in forward and reverse direction with variable feed-rate is also introduced and evaluated in detail. The method utilizes the command compression algorithm proposed in this chapter. During implementation on the FPGA chip some variations are also considered and compared with each other. In the sixth chapter, the most successful command generation method is employed on a test case and its performance is evaluated. The thesis is concluded by summarizing the key results of this research. Possible future works are also presented in the last chapter.

CHAPTER 2

LITERATURE SURVEY

In this chapter of the thesis, the relevant literature topics are discussed in detailed manner and open research areas are also highlighted for possible further study.

2.1 Differencing

The study on literature starts with a detailed investigation of differencing and its application areas. With the help of this study, the necessity of differencing before compression in command generation is elaborated.

Differencing is commonly used in different fields such as data (video, image, speech, index, etc.) compression, and movement detection in tracking. The major requirement in differencing is that the samples in the data must be coherent so that each sample can easily be predicted according to the difference value and the previous sample. Due to the ease of implementation, differencing without compression schemes may be preferable in video literature [3].

It is very common to use differencing during the storage of files that do not have big variations between the entries. For instance, if the average temperature of the days in a year is necessary for graphical purposes, it is wise to store the difference of temperatures between consecutive entries (i.e., d_1, $d_2 - d_1$, …, $d_{365} - d_{364}$) rather than storing the absolute temperature value of each day (i.e., d_1, d_2, …, d_{365}) [4].

This approach decreases the memory usage since the difference between the two consecutive data points is less than the absolute value of the former entry. There may also be many zeros in the differenced format, which will further decrease the memory size.

Differencing is also used in detection of movement of objects. Balch et al. [5] developed a novel machine vision system that can easily follow hundreds of small insects. In order to demonstrate the usability of their system, they analyzed the behaviours of ant colonies in their study. In detection, they simply subtracted the current image from the back ground image to find the movement.

2.2 Data Compression Techniques

In this part of the chapter, different data compression methods are explained and their main application areas are explained. Although the digital communication system performances and the mass storage density are improving rapidly, data compression algorithms still continue to be an important part of many engineering fields since it can eliminate the disadvantages of data storage and overcome the limitations of transmission bandwidths via enabling devices to send the same amount of data in fewer bits [6].

Two types of data compression techniques exist in the literature: lossless and lossy. Lossless data compression, where the original data is extracted without any loss after decompression, applications have increased over the past years due to the need to improve the storage capacity and data transfer rate [7]. On the other hand in lossy compression, original data can only be approximated after decompression. Lossy compression is usually used in situations when the data do not need to be stored perfectly. For instance, pictures can be restored using lossy compression paradigms without much difference from the original picture. In cases where data loss cannot be afforded lossless compression techniques must be used. This is valid especially in text files, since loss of a character can lead to much different situations [8].

The most commonly used data compression techniques in the literature are Huffman, Arithmetic, and Golomb Coding methods. These three paradigms are discussed in a detailed manner in the following sub-sections.

2.2.1 Huffman Coding

Huffman Coding [9] algorithm is one of the most commonly used lossless data compression method in the literature in various areas. The main concept of the algorithm is that it maps an alphabet (of the same size) to a totally different form composed of strings with variable size. The characteristic properties of these symbols are that the ones having high probability have a smaller representation than those occurring less in the file.

Huffman coding belongs to the group of variable length character encoding methods, since in the resulting code the most common characters would be short and the infrequent ones would be long. For the illustration, assume that it is necessary to encode the characters U, L, A, S, Y, M, and N. If conventional approach were used, only three bits per character were necessary for encoding. Suppose that the relative frequency of these characters is as given in Figure 2-1. In the figure, the characters and their frequencies in the sequence are displayed on the left side. On the right side of the figure Huffman codes are displayed for each character. Huffman tree is formed in between of the frequencies and the Huffman codes. During constructing the tree, after the frequency data of the sequence is determined, two elements with the lowest frequencies are selected as the leaf nodes of the Huffman tree. Then the frequencies of these two elements are added together and the resulting value becomes the frequency for the new node. This approach continues until the Huffman tree is completed or until the last node having a frequency of 100%. Huffman code of each character is found by leading from the top of the tree to the corresponding character.

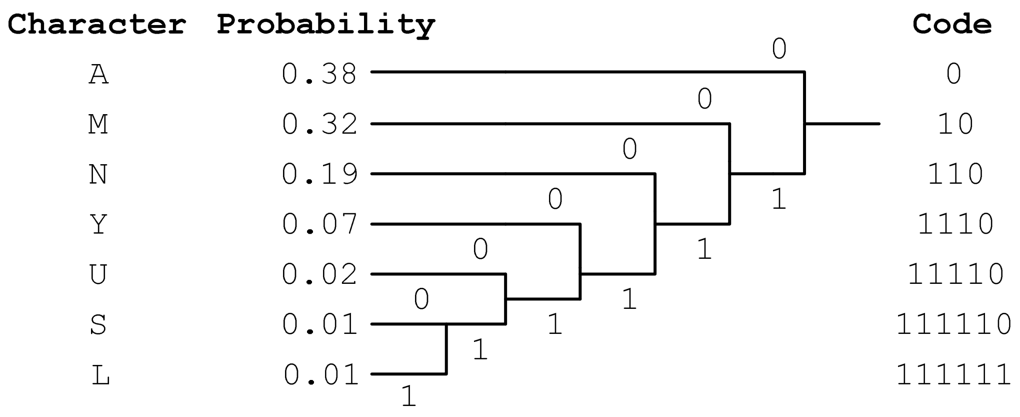


Figure 2-1 Huffman Code for the Given Set of Characters

Conventional Huffman encoding is a non-deterministic one, since a data set can be represented by more than one possible Huffman tree. While constructing the Huffman tree in Figure 2-1, two additional rules described in [8] are applied in order have a unique Huffman tree representation. The first extra rule is that the characters with shorter Huffman codes are placed to the upper branches of the nodes. For instance, in Figure 2-1 S and L are at the bottom part of the tree since they have the longest Huffman codes. Secondly, characters with Huffman codes of same length are placed according to their appearance order. In the figure S and L have the same length of codes. Since S is assumed to be encountered first, it is placed on the upper branch of the node. If these two extra rules are also applied during encoding, it is guaranteed that the Huffman tree is unique for the given set of characters. Due to the word-based memory units, it is difficult to implement Huffman encoding schemes which have variable length of codes. Additional rules described above should be used for fast and lossless recovery in FPGA implementations [8].

A sample encoded data in the Huffman code of Figure 2-1 is decoded in Figure 2-2. It is important to start at the beginning of the data stream in order to decode without any error. When the Huffman codes in Figure 2-1 are examined, it can be

Encoded Message :	1110	0	10	0	110
Decoded Message :	Y	A	M	A	N

Figure 2-2 Huffman Decoding of a Data Stream Encoded in the Huffman Code of Figure 2-1

observed that all of the codes end with a value of '0' except the last Huffman code which is comprised of only 1's. During decoding every bit of the data stream is examined and it is stored to a register until the value in the register matches with a value in the dictionary. After comparing the value in the register with the Huffman codes in the dictionary, original data is generated.

Due to the word-based memory units, it is difficult to implement Huffman encoding schemes which have variable length of codes. Additional codes should be used for fast and lossless recovery in FPGA implementations [8].

2.2.2 Arithmetic Coding

Arithmetic Coding [10] is another commonly used lossless data compression method. According to Witten et al. [11], arithmetic coding is much better than Huffman coding in many cases. Messages are represented by an interval of real numbers between 0 and 1in arithmetic coding. If a long message is going to be encoded, the interval in which the message lies becomes very small and the number of bits representing that interval becomes increases. For the illustration, assume that the letters used in a text are U, L, A, S, Y, M, and N. Their probabilities are given in Figure 2-3. The range, which is [0, 1), is known by both encoder and decoder of arithmetic coding algorithm. It is distributed according to the probabilities of characters as given in Figure 2-3. There is no priority order of range for the characters. It is randomly distributed over the total range. Assume

Character	**Probability**	**Range**
A	0.2	[0,0.2)
L	0.3	[0.2,0.5)
M	0.1	[0.5,0.6)
N	0.2	[0.6, 0.8)
U	0.05	[0.8, 0.85)
S	0.05	[0.85, 0.9)
Y	0.1	[0.9, 1)

Figure 2-3 Probabilities and Ranges of Sample Characters for Arithmetic Coding

that the message “Yaman” is going to be transmitted from the encoder. This encoding process is shown in Figure 2-4. Since the first letter of the message is Y, the encoder starts with narrowing the range to [0.9, 1). The next letter, A, narrows down this range to first one-fifth of it, because the region [0, 0.2) is reserved for A. Proceeding in this way, encoder arrives into the final range, [0.91024, 0.91032).

During decoding, all known about the message is the range [0.91024, 0.91032). Decoder immediately understands that the message starts with the character Y, since it lies in the range [0.9, 1) allocated for Y in Figure 2-3. This clears that the second character is A, since it produces the range [0.9, 0.92), which encloses the range of the original message, [0.91024, 0.91032). The decoder continues in this way to identify the message. The decoder does not need to know the ends of the range set by the encoder. Any number in the range [0.91024, 0.91032) can be used. The main problem decoder faces is that it cannot detect the end of the message. To overcome this drawback a special character may be used which is known to encoder and decoder [11].

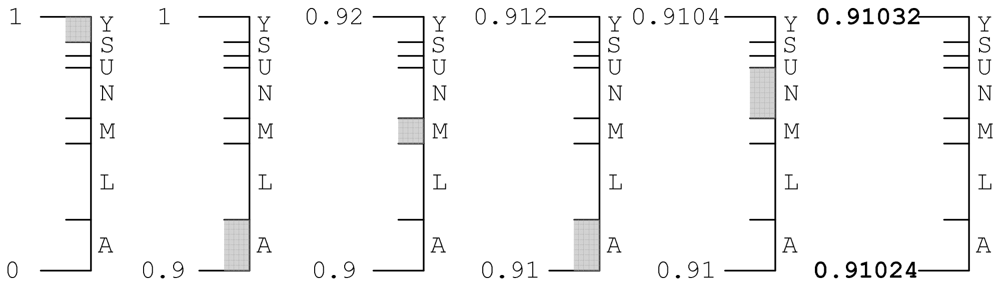


Figure 2-4 Encoding of the message "Yaman" in Arithmetic Coding

2.2.3Golomb Coding

Golomb coding [12] is a lossless data compression paradigm found in the literature. It is not as complex as other compression algorithms but its performance is a bit lower than the previously described data compression methods. The applications of Golomb coding are generally focused on video compression systems such as H.264 video standard [13].

The term group size (*m*) is a very important parameter in Golomb coding. It has a direct effect on the compression ratio. Selecting *m* as the power of two increases the efficiency of coding [12]. After the value of *m* is determined, data set which is to be compressed is divided into subsets of having zeros at the beginning and one at the end. Run lengths of a sample data set are given in Figure 2-5.

```
   Data Set: 00100010100000010010000000010010001110001
    Subsets: 001  0001  01  0000001  001  000000001  001  0001  1  1  001
Run Lengths:  2    3    1      6      2       8       2    3    0  0   2
```

Figure 2-5 Run Length Determination

Before the encoding process, the codes should be formed. An example of Golomb coding for a group size (*m*) of 4 is given in Figure 2-6. Codes are created according to the run lengths which are grouped of size *m* and these groups are named as A_k. Codes appearing on the right of Figure 2-6 have prefixes determined according to their group and tails. Each prefix has (*k* - *1*) number of ones followed by a zero. Tails are the binary representation of integers starting from zero value to (*m* - *1*). It is important to note that the widths of the tails must be the same. For instance, if the group size is selected as eight, the widths of the tails must be three which is $log_2(m)$. The code for each run length is formed by adding the prefix and the tail.

Group	Run Length	Prefix	Tail	Code
A_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A_3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011

Figure 2-6 Golomb Coding with a Group Size of 4

Encoded version of the data set given in Figure 2-5 can be seen in Figure 2-7. When the resulting code in Figure 2-7 is examined, it can easily be observed that the compression ratio is not that much when compared to other compression algorithms.

```
      Subsets: 001  0001 01 0000001 001 000000001 001 0001   1   1  001
Encoded Data : 010  011 001  1010   010   11000   010  011   000 000 010
```

Figure 2-7 Golomb Encoding

2.3 FPGA Implementations of Data Compression Techniques

Lossless data compression, where the original data is extracted without any loss after decompression, applications have increased over the past years due to the need to improve the storage capacity and data transfer rate [7]. There are many examples for the hardware implementations of conventional data compression techniques in the literature. Among these techniques, Huffman [14 - 15], Lempel-Ziv (LZ) [16 - 17], and Golomb [18] compression algorithms are the most popular ones for FPGA implementations.

Rigler et al. [14] implemented Huffman and LZ encoders on an FPGA development board using VHDL in order to serve a basis for hardware implementations of the popular compression program GZIP. Since this implementation is planned to be work with GZIP compression program, while forming the Huffman trees two additional rules are used to make sure that the trees are deterministic. According to the results presented modified Huffman algorithm uses less hardware resources than the LZ algorithm. De Araújo et al. [15] employed a different approach for Huffman Algorithm. They implemented a microprogrammable controller on the FPGA to perform lossless data compression utilizing Huffman method. They claimed that with this flexible architecture, other compression algorithms such as Arithmetic- and Golomb coding algorithm can easily be implemented.

Abd El Ghany et al. [16] realized the LZ encoding and decoding algorithm on the FPGA. In order to increase the efficiency, they used systolic array approach which resulted in a 40% decrease in the compression rate and 30% decrease in resource usage. Cui [17] also implemented LZ compression algorithm on the FPGA by not using the conventional dictionary approach. The dictionary is divided into smaller units. By this approach, the look-up time is decreased and parallel operations become performable.

Among conventional data compression techniques, hardware implementations of different algorithms for compressing specific data structures are also present in the literature. For instance, Yongming et al. [19] have realized the Linear Approximation Distance Threshold algorithm on FPGA to compress the Electrocardiograph signals. Similarly, Valencia and Plaza [20] developed an FPGA-based data compression technique based on the concept of spectral unmixing to compress hyperspectral data. The novel compression method described in the study can be regarded as task-specific since it is developed to compress the signed integer position command sequences. It may not yield better results for text or image compression.

2.4 FPGA Implementations of Polynomial Approximation Methods

Polynomial approximation is not commonly used in command generation systems due to the inevitable errors in approximations. In order to minimize these types of errors, function approximation algorithms are applied to model the segmented trajectory in a piecewise fashion. The aim of this segmentation is to decrease the error resulting from approximation. In the literature, there exist various segmentation approaches. The most common one is uniform segmentation, in which the widths of the segments are equal and the number of segments is limited to powers of two. Since the segment widths cannot be customized according to local function characteristics, a huge amount of segments are necessary to fulfill the error requirement [21]. To overcome this problem, dynamic segmentation depending on the inflection points of the trajectory is proposed in this study.

Selecting an appropriate function approximation technique is very important especially in hardware implementations since errors resulting from the approximation should be stored for lossless reconstruction [22]. Another important aspect in approximation is the degree of polynomials used. When memory resources are limited, higher degree polynomials are commonly applied at the expense of increased computational complexity [23].

Hardware implementations of function approximation techniques are frequently used in developing Direct Digital Frequency Synthesizers (DDFS) in the literature. Ashrafi et al. [24] proposed an FPGA-based method that utilizes Chebyshev polynomial series interpolation. The developed technique unifies the results of ROM-less polynomial approximation methods for sinusoidal DDFS implemented on FPGAs [25] [26].Among various approximation techniques, Chebyshev polynomials are usually preferred for hardware implementation. This is due to the fact that Chebyshev polynomials give better results for non-periodic signals that are limited in range. In the study it is also turned out that the performance of Chebyshev polynomials are superior to Legendre, and Bernstein – Bezier polynomials.

2.5 FPGA-Based Command Generation Systems

Implementations of command generation methods on FPGA chips are not very common in the literature due to high computational complexity involved in the methods. Therefore, the techniques employed on FPGA have simplifications and/or include error compensation modules into the systems. For instance, Su et al. [27] developed a motion command generation chip utilizing FPGA for point-to-point motion applications. They implemented trapezoidal and S-curve motion planning adopting the digital convolution method rather than the complex polynomial type method. With this approach, the computational complexity is significantly decreased. Furthermore, they developed a real-time output pulse compensation algorithm to eliminate the error in the number of output pulses and the results are found to be satisfactory.

Jeon and Kim [28] also used the digital convolution method and designed an FPGA-based acceleration and deceleration circuit for industrial robots and CNC machine tools. Likewise, the method developed by Su et al. [27], they did not use the complex polynomial technique to generate velocity profiles of various acceleration and deceleration characteristics that require extensive computations. Since the current techniques are not satisfactory for generating velocity profiles for industrial robots and CNC machine tools [29], they developed a new method to compensate this deficiency. According to the experimental results given in the paper, they were able to generate unsymmetrical velocity profiles that cannot be generated by digital convolution techniques. Comparing the two works, former one is superior over the latter method. The error in the output pulses is not compensated in Jeon and Kim's study [28] so that the errors are inevitable between the command and response signals. On the other hand, the method proposed in the paper generates commands without any error. Furthermore, it generates position, velocity, and acceleration profiles at the same time.

2.6 Feed-rate Control of CNC Machine Tools

The precision in manufacturing continually improves. In the manufacturing process, the quality of the product is dependent on the functions of CNC machine. Feed-rate control of the machine tool is very important factor for a high-precision CNC machine.

There are various algorithms proposed on feed-rate control in order to increase the surface quality of the product. For instance Cheng et al. [30] employed a predictor-corrector algorithm to estimate the servo command at the next sampling time. In another study of Cheng et al. [31] developed a new interpolator to produce servo commands for real-time control of CNC machining. The main advantage of the proposed interpolator is being capable of generating motion commands for servo controllers at variable feed-rates. In a similar study of Xu et al. [32], they presented variable interpolation schemes for planar implicit curves. They were also able to interpolate in real-time to improve machining efficiency. In the proposed method, the feed-rate is set by the operator according to geometrical state of the surface. In other words, it is decreased when the tool is machining curved parts and increased on planar surfaces.

2.7 Open Research Areas

During the literature survey, not only the current research efforts, but also unexplored areas in the field have been surveyed. With the help of this study, the scope of the thesis has been determined.

First of all, the tool location data are not directly generated in conventional command generation systems (CAD/CAM + CNC processors). Instead, the Numerical Control (NC) code, which represents the trajectory via an ensemble of geometric entities (such as line, arc, helix, NURBS etc.), is parsed and interpreted in real-time to produce the position of the tool accordingly. The reason for this rather meandering approach is due to the fact that data storage and retrieval was extremely costly in the past. With this limitation is greatly circumvented in today’s technology, the tool location data may directly be processed to increase the performance of real-time command generation systems. With this approach, the use of post processors, (machine-depended) NC codes, and complex hardware for real-time data interpolation may be eliminated.

Secondly, the real-time data decompression techniques, which may further alleviate the efficiency of this direct command data generation method, are not fully explored in the literature. For instance, there is no FPGA implementation of Arithmetic coding (regardless of the application area) since it is very difficult to synthesize a digital circuitry that is realizing the algorithm via hardware description languages such as Verilog and VHDL. According to the results obtained in this thesis, the performance of Arithmetic coding is much better than any other compression algorithms when applied to the context of command generation. Thus, the FPGA implementation of Arithmetic coding in command generation methods might be a remarkable contribution.

CHAPTER 3

FPGA INTERFACE

3.1 Introduction

The FPGA Interface to be devised within the scope of the thesis is a board whose port connections can be defined through software and is capable of connecting various sensors and actuators (in any order) to the FPGA based motion control system. The current version of the interface can be regarded as a prototype and it is planned to finalize the interface as a product in the future.

When the devices and their outputs signal used in motion control applications are considered, it is concluded that the interface (as in the data acquisition cards) should have four main communication channels:

1) **Analog Input:** This card of the interface is responsible for the connection of analog devices (sensors, drivers, converters, etc.) to the system. The voltage ranges of these type of devices are

   a. Unipolar 5V $\in$ [0, 5] V

   b. Bipolar 5V $\in$ [-5, 5] V

   c. Unipolar 10V $\in$ [0, 5] V

   d. Bipolar 10V $\in$ [-10, 10] V

2) **Analog Output:** This developed unit is to be used for electromechanical systems and controlling motor drives. The output voltage ranges of this card are the same with the Analog Input card.

3) **Digital Input:** Devices (switches, sensors, etc.) generating digital signals are connected to the motion control system via this card. For generality, the developed card should be compatible with the below digital logic families:

   a. TTL, LS-TTL

   b. CMOS (ACT, HCT, 74C)

4) **Digital Output:** This interface is basically used to generate digital signals in order to sustain digital communication with various devices. The outputs of this card should be compatible with the logic families given above.

The microcontrollers and Digital Signal Processors (DSPs) employed in the motion control industry have internal Analog to Digital Converters (ADCs) and Digital to Analog Converters (DACs) but the FPGA chips have neither of them. Another issue that should be noted is that the most of the FPGA chips operate with 3.3V CMOS/ACT logic families. Thus, in order to evaluate the aforementioned signals there should be signal-converter circuits that can be connected to the digital ports of the FPGA chip on the interface.

The interface shown in Figure 3-2 basically consists of a main board having eight identical slots reserved for four types of daughter cards. The daughter cards are Analog Input, Analog Output, Digital Input, and Digital Output Cards. These boards are to be evaluated in the following subsections in a detailed manner.

When the literature is investigated, there exists no full device that can be utilized with analog and digital peripheral devices having various voltage ranges. The most related works include the programmability of FPGA pins according to the needs of the users and the voltage level converters.

Menon et al. [41] developed a programmable input/output unit composed of three input and one output circuits. These four circuits are coupled to one of the FPGA pins and enabling of these circuits is carried by the programmable bits. With this unit a selected pin of the FPGA can accept TTL, GTL, GTLP, LVPECL, or LVDS voltage levels as input and generate TTL, GTL, or GTLP compatible signals. Goetting et al. [42] designed a similar system, but they implemented the structure within the FPGA chip.

Chang [43] introduced the Application Specific FPGA phrase to the literature in his study. These integrated circuits include at least two functional units, which can be a digital to analog converter, a compressed image decoder, a random access memory, etc. The main purpose of the FPGA chip used in this design is to maintain the communication between the functional units and to connect them.

3.2 Mother Board

The main responsibility of the mother board is to host different types of daughter cards and provide required voltages and signals to these boards. Schematic design of the mother board shown in Figure 3-1 is drawn in Proteus 7 Professional [44]. As can be seen from the figure, the main board has three main functions: **i)** to generate bipolar (reference) triangular waveform, **ii)** to produce reference voltages, **iii)** to supply different voltage ranges to the daughter cards.

Bipolar triangular waveform generator output is used by the Analog Input Cards (AIC) to compare the analog signals with the generated waveform. Triangular waveform generated by the module can be seen in Figure 3-7. The circuit implements the general triangular waveform generator described in [45]. The generated signal should be exactly in between 0V and 3.3V for proper analog to digital conversion. To eliminate the small deviations from the desired voltage levels the resistances (R5 and R6) used in the corresponding circuitry should perfectly be matched. The frequency of the waveform is set by the resistance R6 and the capacitance C2 used in the design, which is about 40 kHz in this case.

On the other hand, the reference voltage generator outputs a constant voltage of 1.65V. This voltage is obtained from the resistance R2 which is serially connected the resistance R1. The capacitance C1 is used to eliminate the noise. The operational amplifier U1 is a voltage follower used not to let the board draw current from the voltage divider. If the voltage follower is not used, during the operation of the board reference voltage can oscillate.

Electrical modules on the upper of the figure are the indicators of available voltage levels supplied by the power source. They also distribute these voltages to the daughter cards. J1, J2, J3, and J4 are connected to 3.3V, 5V, V_{DD}, and 15V voltage supplies, respectively. The capacitances between C3 and C11 are placed to eliminate the noise on the voltage resources. Light Emitting Diodes (LEDs) are also used to indicate the availability of the voltage source on the board.

As can be seen from Figure 3-2, the main board is capable of holding only eight daughter boards. The connectors placed on the left side of the board are used to supply 3.3, 5, 12, and -12 voltages. On the other hand, the connectors on the right side are used to connect peripheral devices to the system. FPGA pins are connected to the headers on the middle of the connectors on the right side.

In Figure 3-3, Altera DE1 FPGA Development Board, designed interface board, and the power supply are shown with their connections. According to the number of pins used for one mother board, FPGA Development Board can support up to four mother boards. This means that thirty-two daughter cards can be placed on the main boards, so that with this set-up an eight axis system can easily be controlled.

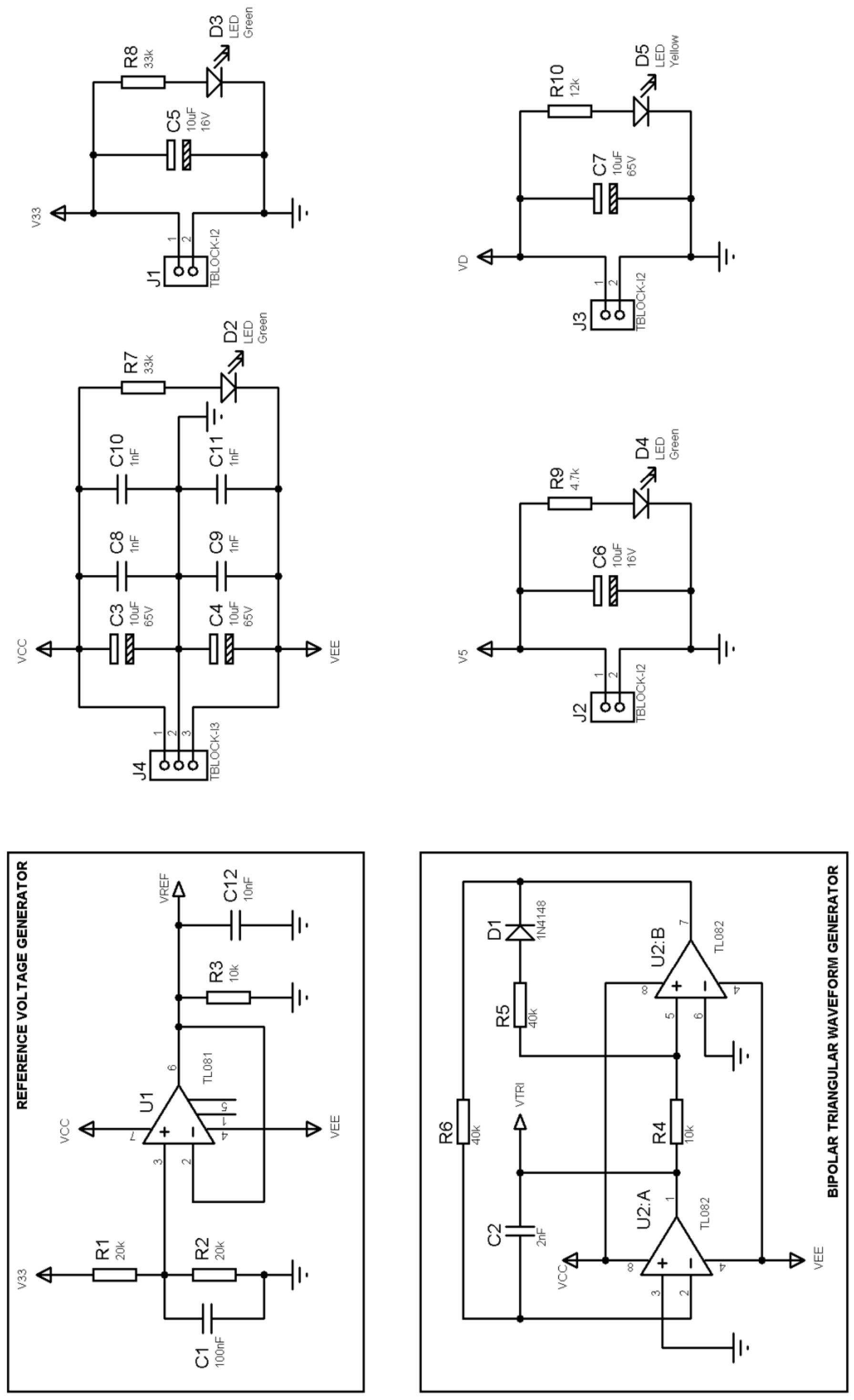


Figure 3-1 Schematic Design of the Mother Board

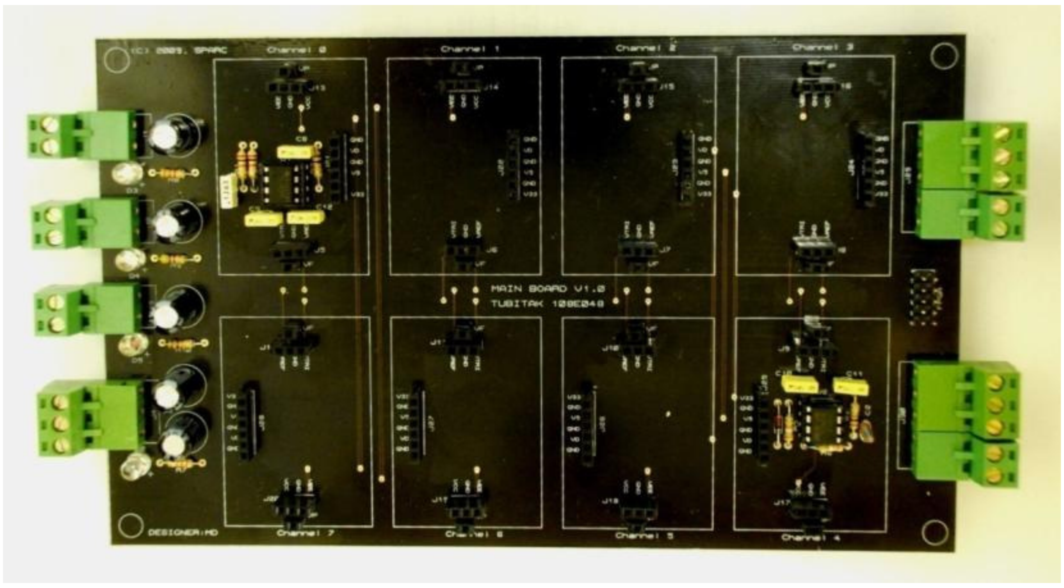


Figure 3-2 Main Board

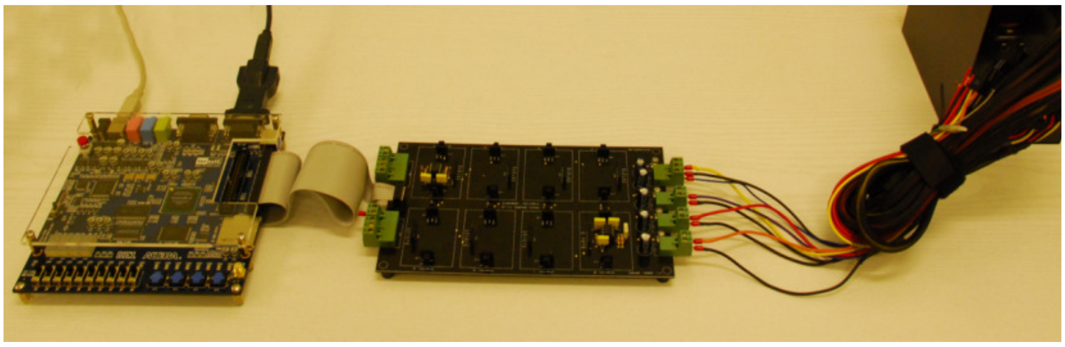


Figure 3-3 FPGA Development Board and Its Interface

3.3 Analog Input Card

The main task of this card is to convert the signals of analog sensors used in the motion control industry to digital signals utilizing Pulse Width Modulation (PWM). According to the block diagram given in Figure 3-4, the input analog signal is first amplified and biased according to the range selection done by the user.

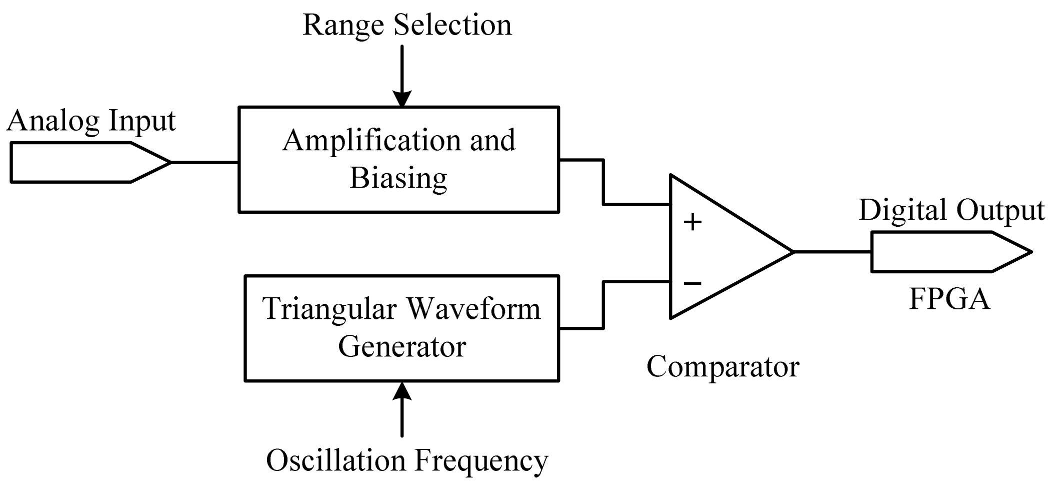


Figure 3-4 Block Diagram of Analog Input Card

The modified input signal is then compared with the triangular waveform on the mother board and the digital output is fed to the corresponding pin of the FPGA chip for further computations.

The analog input card, whose circuit schematic is illustrated in Figure 3-5, is compatible with various voltage ranges as discussed in Section 1. The conversion of the voltage levels are carried out with a dipswitch and a two way jumper placed on the card. With this approach the voltage range of the sensor is to be determined. Voltage ranges are obtained according to the position of the switches and the jumper are given in Table 3-1.

The most important part of the design is placed on the upper part of Figure 3-5. The operational amplifier U1:A is used to modify the range of input signal. The output voltage of U1: A can be defined as

$$V_1 = \left(-\frac{R_1}{R_4/2}\right)V_p \qquad \left(0 \le V_p \le 5V\right) \tag{3.1}$$

$$V_1 = \left(-\frac{R_1}{R_4}\right)V_p \qquad \left(0 \le V_p \le 10V\right) \tag{3.2}$$

$$V_1 = \left(-\frac{R_1}{R_4}\right)V_p \qquad \left(-5 \le V_p \le 5V\right) \tag{3.3}$$

$$V_1 = \left(-\frac{R_1/2}{R_4}\right)V_p \qquad \left(-10 \le V_p \le 10V\right) \tag{3.4}$$

Then the output signal of U1:A is shifted according to the polarity of the input port signal by U1:B. For the unipolar cases, the resistance R3 is connected parallel to the resistance and for bipolar cases it is not connected to the circuit.

After the modification of the voltage range is completed, the modified input signal is compared with the triangular waveform generated from the mother board utilizing a comparator chip (U2:A) and its output is fed to the FPGA chip for further analysis of the signal. The vacant pins of the chip U2 are connected to ground not to cause any problems.

The manufactured card in Figure 3-6 is tested by placing it on one of the slots available on the mother board as shown in Figure 3-3. After all connections are done, by using a function generator, analog signals having different frequencies and magnitudes are fed to the designed analog input card to evaluate the performance of it. In Figure 3-7, signals obtained using the analog input card are shown. As described above, firstly the input analog signal (green) is scaled to the[0, 3.3] voltage range (blue). Then this signal is compared with the triangular waveform and a pulse sequence changing according to the amplitude of the input signal is generated by the card. Hence, it will be possible for the FGPA to determine the value of input signal by measuring the widths of the pulses.

Table 3-1 Voltage Levels

First Switch	Second Switch	Jumper	Voltage Range
1	0	Unipolar	[0, 5] V
1	1	Unipolar	[0, 10] V
0	0	Bipolar	[-5, 5] V
0	1	Bipolar	[-10, 10] V

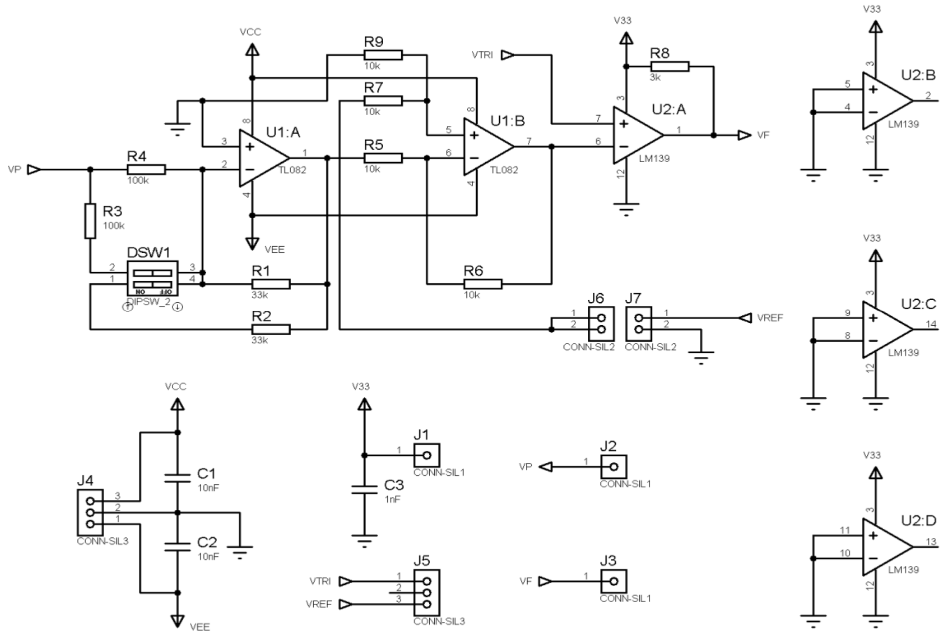


Figure 3-5 Schematic Design of the Analog Input Card

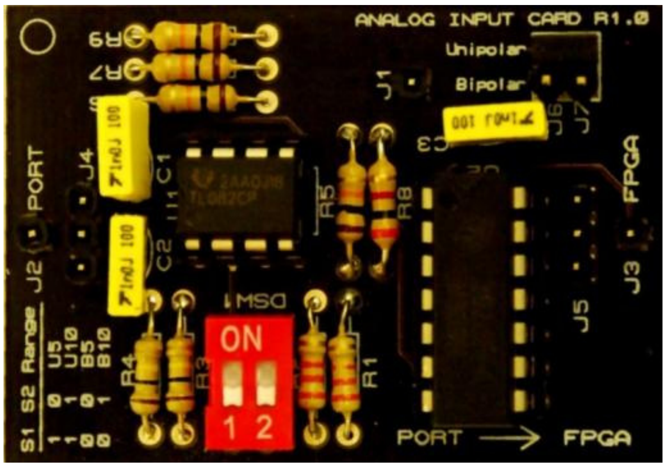


Figure 3-6 Analog Input Card

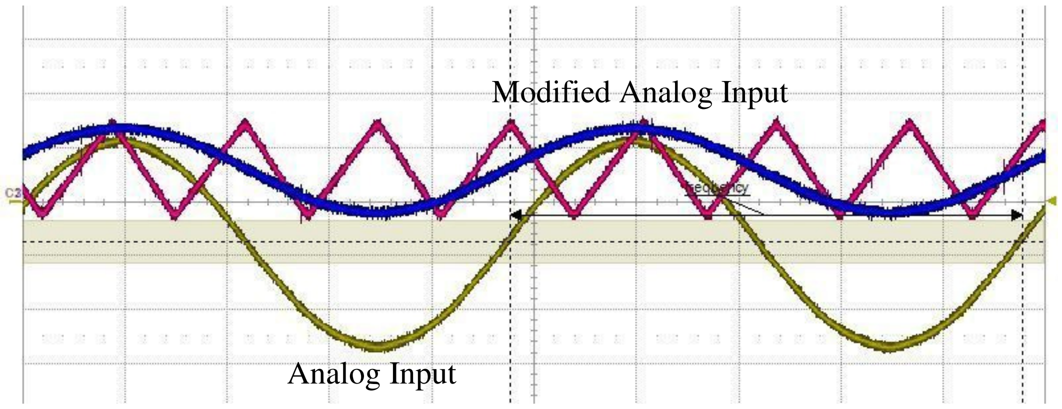


Figure 3-7 Signals of the Analog Input Card

Analog signals having different frequencies (100 Hz, 1 kHz, 10 kHz) are fed to the card and the resulting signals are shown in Figure 3-8, Figure 3-9, and Figure 3-10. It can be inferred from the results that at 40 kHz frequency pulse width modulation is successfully performed. It should also be noted that the resolution of this analog to digital converter is about 10 bits due to the 50 Mhz clock used on the FPGA board.

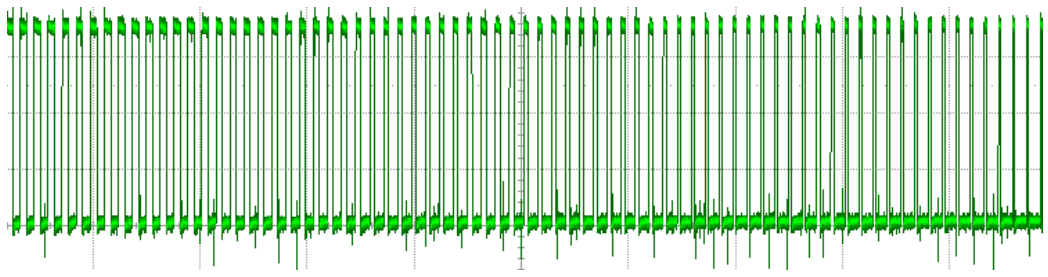


Figure 3-8 Output Signal of the Analog Input Card at 100 Hz

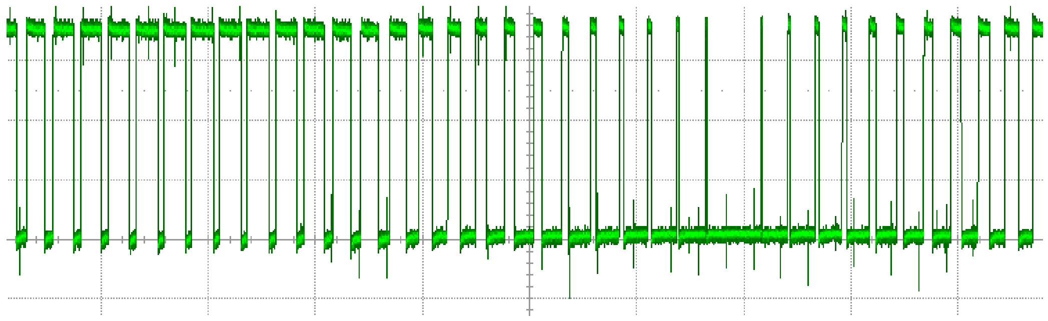


Figure 3-9 Output Signal of the Analog Input Card at 1 kHz

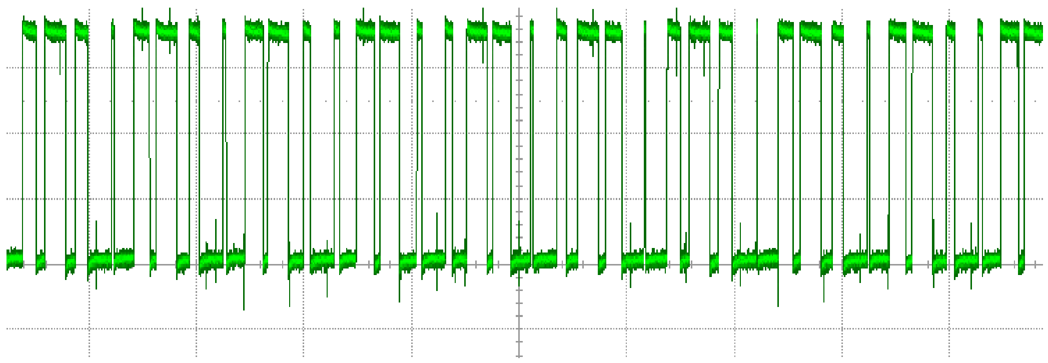


Figure 3-10 Output Signal of the Analog Input Card at 10 kHz

3.4 Analog Output Card

The main function of this card, whose schematic shown in Figure 3-12, is to generate analog signals that are necessary to control motor drives and electromagnetic devices. The voltage ranges of the output signals are modified in a similar fashion described in the previous section. The positions of the switches and the jumper are the same to obtain voltage ranges as given in Table 3-1.

The block diagram of the analog output card is given in Figure 3-11. Firstly, the generated PWM signals from the FPGA development board are transferred to the low-pass filter. According to the cut-off frequency of the filter, digital signals are converted to their analog counterparts. Then these analog signals are amplified and biased according to the voltage range selections of the user.

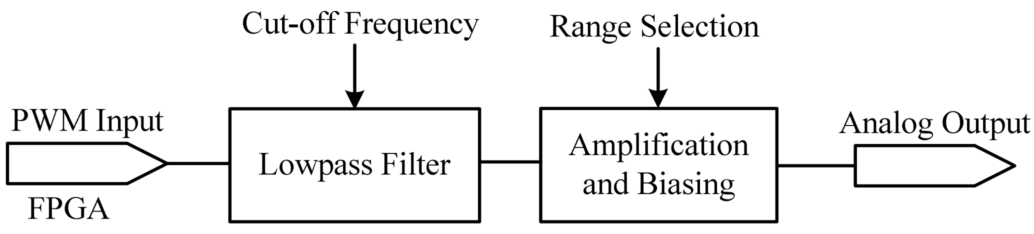


Figure 3-11 Block Diagram of Analog Output Card

As can be seen from Figure 3-12, the incoming PWM signal from the FPGA pin is fed to an active low pass filter (R1 and C1) having a cut-off frequency of 5 kHz. Then the signal is shifted according to the polarity selection by the jumper which is placed on J5 and J6 by the help of U1:B. In the final part of the design another operational amplifier (U1:C) is used to scale the output signal according to the following expression:

$$V_P = \left(-\frac{R_6/2}{R_7}\right) V_9 \qquad \left(0 \leq V_p \leq 5V\right) \tag{3.5}$$

$$V_P = \left(-\frac{R_6}{R_7}\right) V_9 \qquad \left(0 \leq V_p \leq 10V\right) \tag{3.6}$$

$$V_P = \left(-\frac{R_6}{R_7}\right) V_9 \qquad \left(-5 \leq V_p \leq 5V\right) \tag{3.7}$$

$$V_P = \left(-\frac{R_6}{R_7/2}\right) V_9 \quad \left(-10 \leq V_p \leq 10V\right) \tag{3.8}$$

In order to evaluate the performance of the manufactured card in Figure 3-13, a universal sinusoidal signal generator is designed in Quartus II 9 Web Edition software. Its schematic design is shown in Figure 3-14. There are three different modules in the design, namely Clock Divisor (CD), Sine Wave (SW), and PWM Generator (PWMG). SW module can be regarded as the core of this design, since the other two modules are in communication with the SW module. It reads discrete values of a quarter sine wave from a look-up table and sends these values in an alternating manner to complete a full sine wave to the PWMG module. The PWMG module simply generates the output PWM signal according to the incoming value from the SW module. The main task of the CD is to divide the global clock according to the user inputs and pass it to the SW and PWMG modules. With the switches available on the Altera DE1 Development Board, it is possible to generate sinusoidal signals ranging between 90 Hz and 12 MHz. By using this design, sinusoidal signals with frequencies of 100 Hz, 1 kHz, and 4 kHz are generated and fed to the analog output card to observe the performance of it.

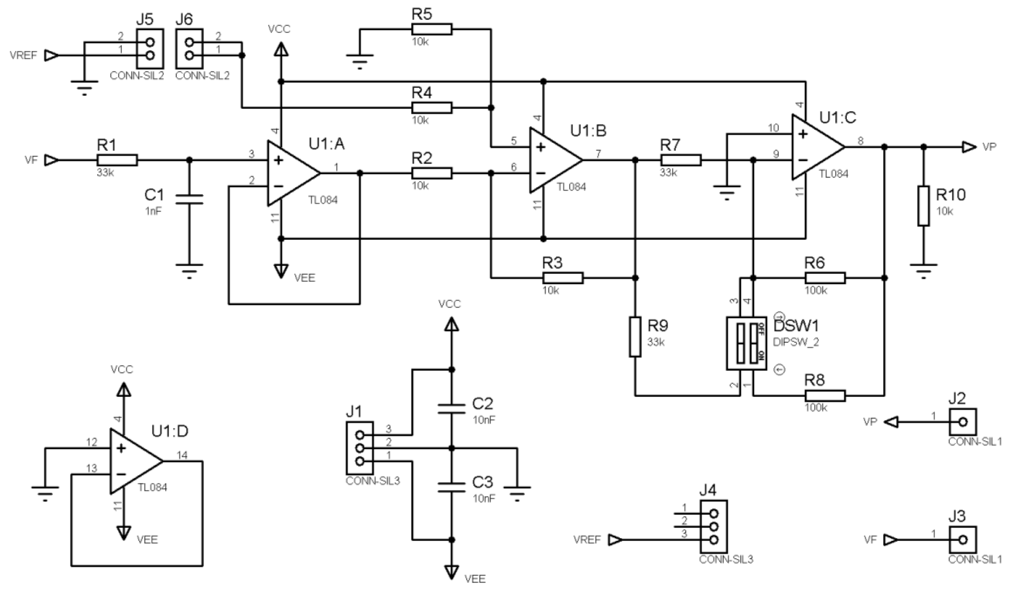


Figure 3-12 Schematic Design of the Analog Output Card

When the results shown in Figure 3-15, Figure 3-16, and Figure 3-17 are considered, it can be concluded that the card works properly at low and intermediate frequencies. On the other hand, it is observed during the tests that at frequencies higher than 4 kHz the magnitude and the frequency of the output signal are deteriorated. This is due to the fact that there is a low pass filter whose cut-off frequency is 5 kHz in the design. Thus, the cut-off frequency of the filter should be determined according to the needs of the application.

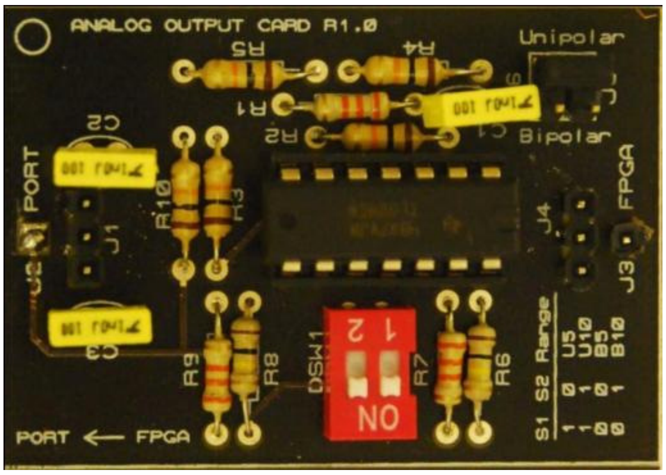


Figure 3-13 Analog Output Card

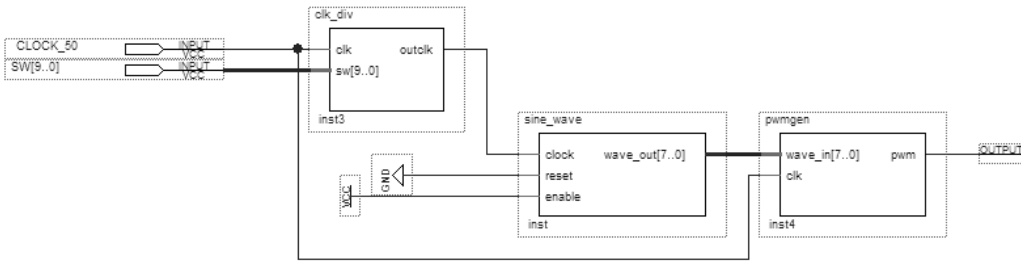
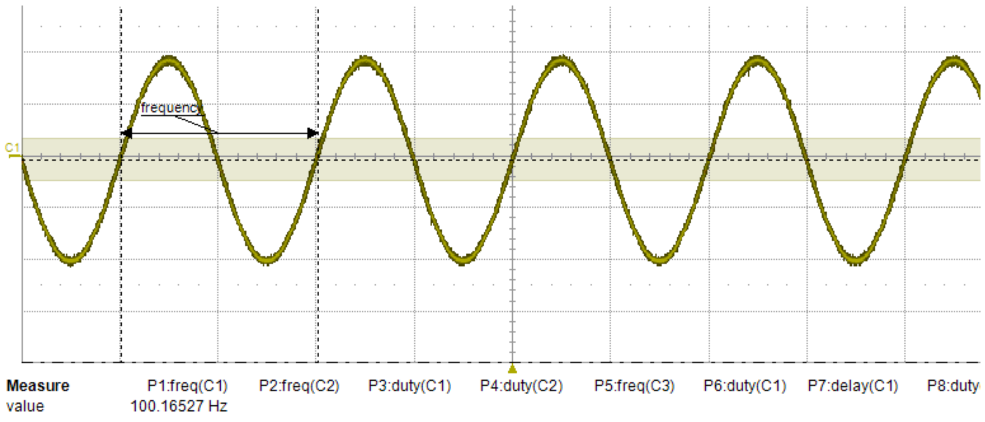
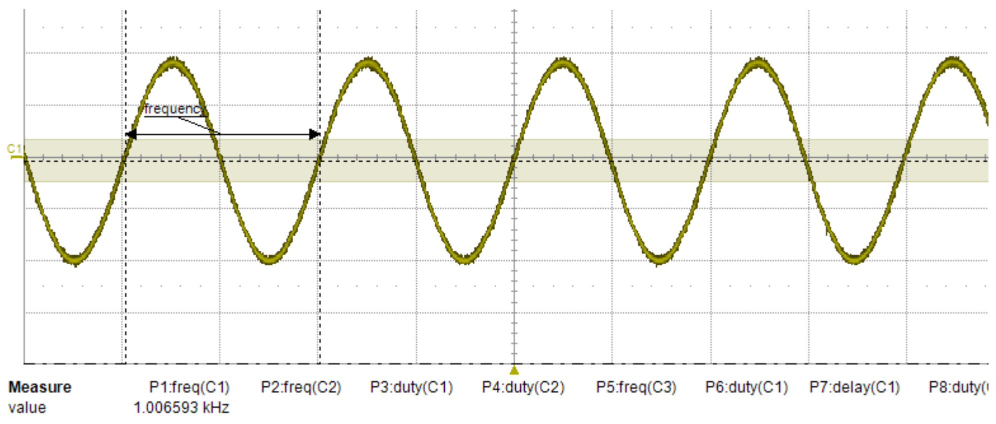


Figure 3-14 Hardwired FPGA Implementation of the Sinusoidal Signal Generator



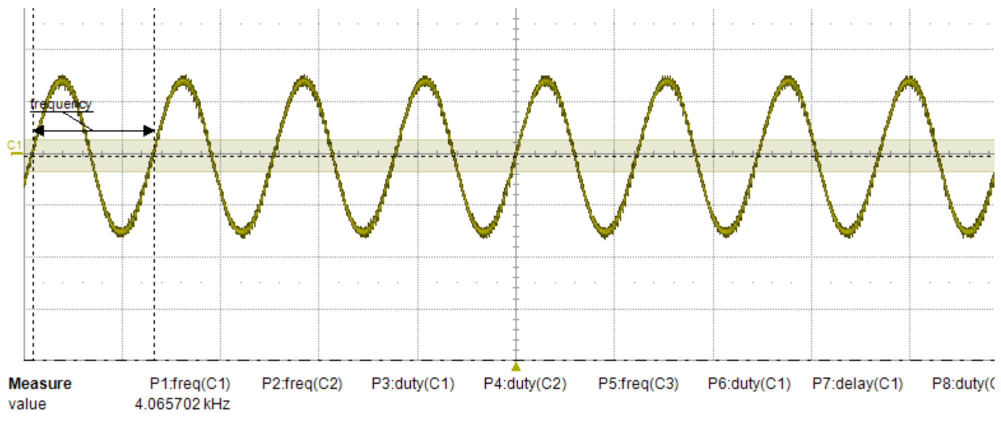
Measure	P1:freq(C1)	P2:freq(C2)	P3:duty(C1)	P4:duty(C2)	P5:freq(C3)	P6:duty(C1)	P7:delay(C1)	P8:duty
value	100.16527 Hz							

Figure 3-15 Output Signal of the Analog Output Card at 100 Hz



Measure	P1:freq(C1)	P2:freq(C2)	P3:duty(C1)	P4:duty(C2)	P5:freq(C3)	P6:duty(C1)	P7:delay(C1)	P8:duty(
value	1.006593 kHz							

Figure 3-16 Output Signal of the Analog Output Card at 1 kHz



Measure	P1:freq(C1)	P2:freq(C2)	P3:duty(C1)	P4:duty(C2)	P5:freq(C3)	P6:duty(C1)	P7:delay(C1)	P8:duty(
value	4.065702 kHz							

Figure 3-17 Output Signal of the Analog Output Card at 4 kHz

3.5 Digital Input Card

The digital input card, whose schematic drawing is provided in Figure 3-18, is basically used to convert digital signals belonging to various logic families such as TTL, LS-TTL, and CMOS to 3.3V CMOS based compatible inputs for FPGA chips. The main components of the design are the two Bipolar Junction Transistors (BJTs). Voltage level shifting is carried out with the help of these high speed (600 MHz) transistors coded as 2N2369. Firstly the digital input is fed to the base of the first transistor and the collector of this transistor is connected to the base of the second transistor. The emitters of the transistors are directly connected to ground and collectors are fed from the 3.3V supply. With this design digital inputs at various logic levels are properly converted to FPGA compatible range. In the literature the circuit is known as the totem pole output circuit. In order to evaluate the performance of the digital input card, square signals (TTL) at different frequencies (1 kHz, 10 kHz, and 100 kHz) are fed to the manufactured card shown in Figure 3-19 via function generator. According to the results given in Figure 3-20, Figure 3-21, and Figure 3-22, at low and intermediate frequencies there is no remarkable change in the form of output signal whose range is [0, 3.3] V. On the other hand, as the frequency of the input digital signal increases, the rise time of the output signal tends to increase. The fundamental reason of this problem is that there exist undesirable capacitances at connection points and between electrical routes. Although it seems that the performance of the card is sufficient for many industrial applications, some improvement should be done on the card to let it operate properly also at high frequencies.

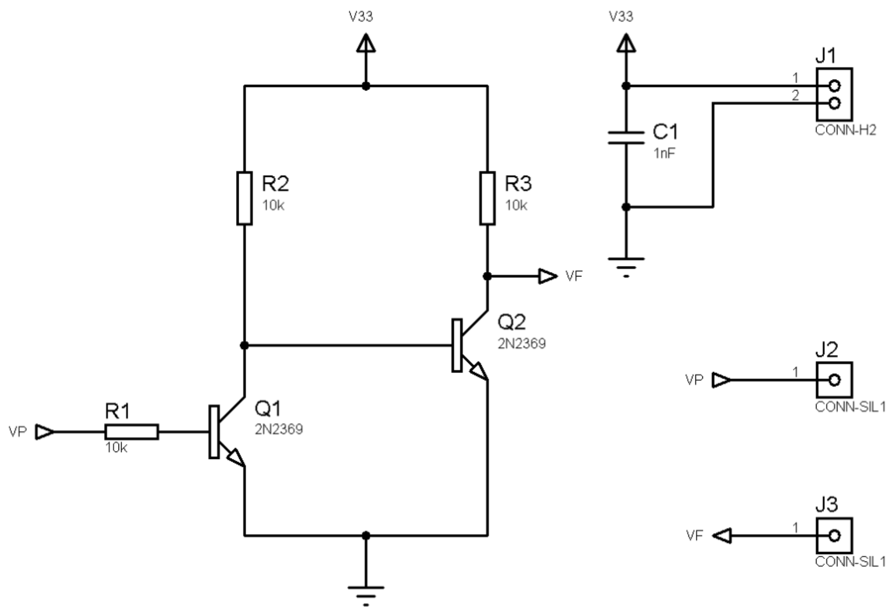


Figure 3-18 Schematic Design of the Digital Input Card

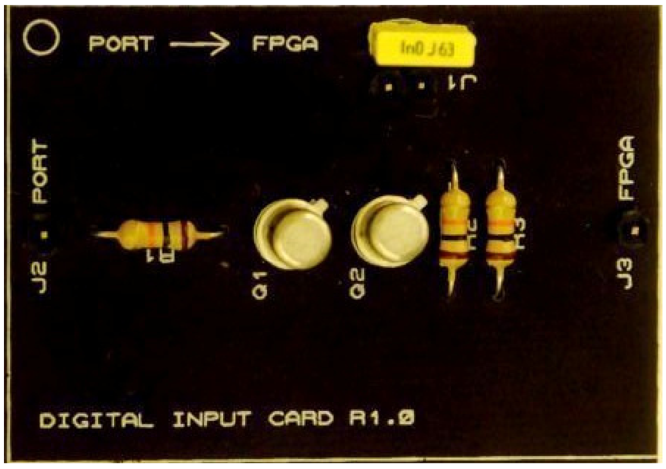
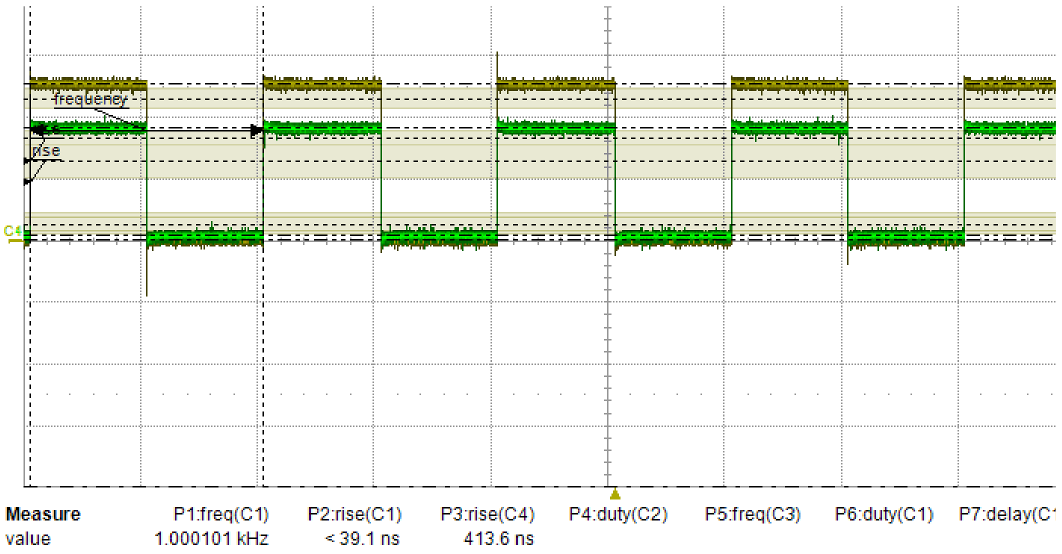
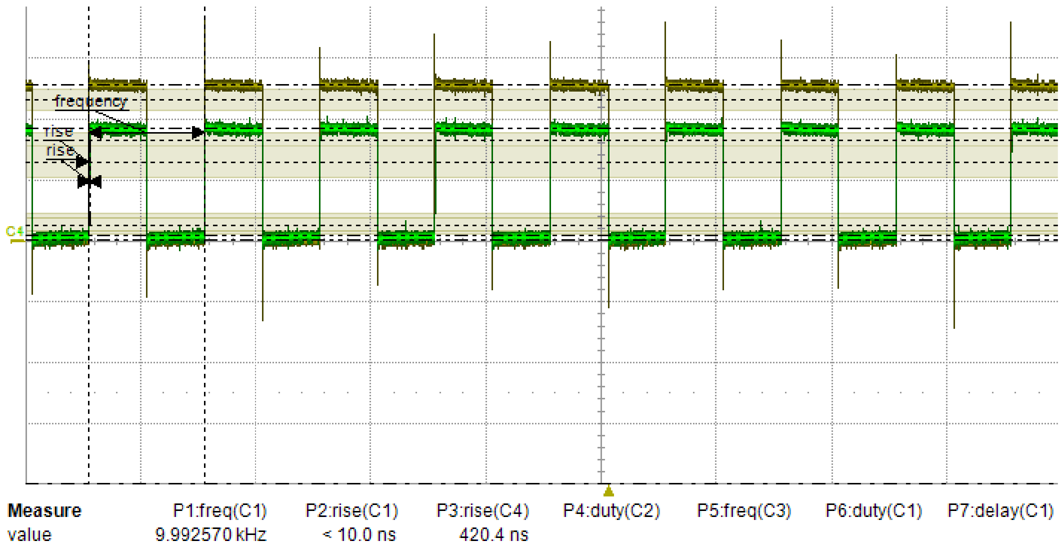


Figure 3-19 Digital Input Card



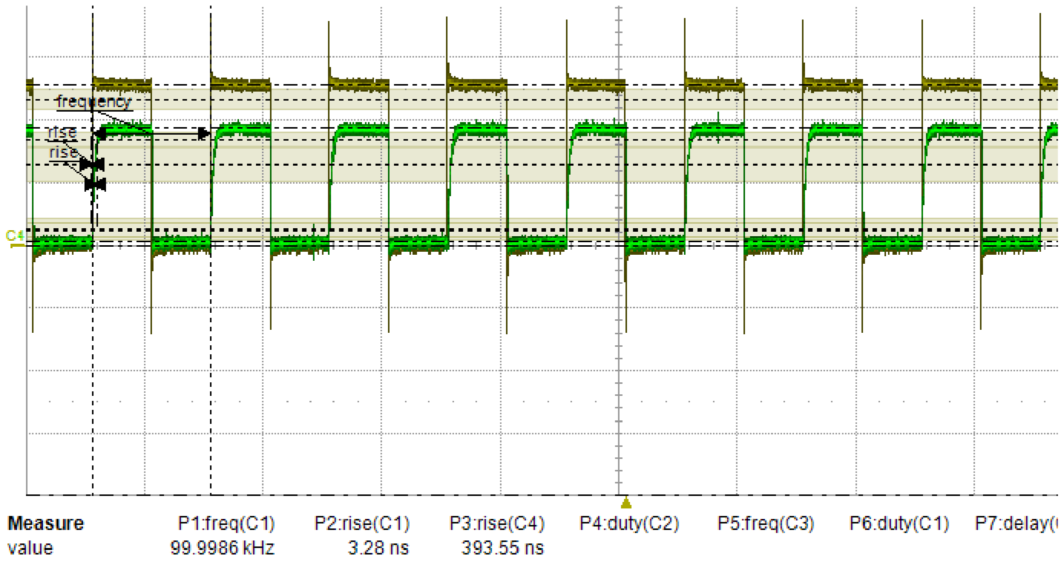
Measure	P1:freq(C1)	P2:rise(C1)	P3:rise(C4)	P4:duty(C2)	P5:freq(C3)	P6:duty(C1)	P7:delay(C
value	1.000101 kHz	< 39.1 ns	413.6 ns				

Figure 3-20 Output Signal of the Digital Input Card at 1 kHz



Measure	P1:freq(C1)	P2:rise(C1)	P3:rise(C4)	P4:duty(C2)	P5:freq(C3)	P6:duty(C1)	P7:delay(C1)
value	9.992570 kHz	< 10.0 ns	420.4 ns				

Figure 3-21 Output Signal of the Digital Input Card at 10 kHz



Measure	P1:freq(C1)	P2:rise(C1)	P3:rise(C4)	P4:duty(C2)	P5:freq(C3)	P6:duty(C1)	P7:delay(
value	99.9986 kHz	3.28 ns	393.55 ns				

Figure 3-22 Output Signal of the Digital Input Card at 100 kHz

3.6 Digital Output Card

The digital output card, whose circuit schematic is shown in Figure 3-23, is basically used to generate digital signals to maintain the communication between the FPGA chip and other devices. The output signals of this card should be compatible with the logic families mentioned in the previous section. The output voltage range of this card is also changed with a jumper as in the other cards. When the schematic design is considered, the applied method is similar to the one in the digital input card. Two BJTs are used and this time the output voltage level is connected to the collector legs of the transistors.

In order to test the digital output card in Figure 3-24, square signals at high frequencies are generated utilizing the FPGA board and fed to the card. The response of the digital output card to the square signal with a frequency of 300 kHz is shown in Figure 3-25. As the case in the evaluation of the digital input card, the rise time of the output signal is much higher than the rise time of the input signal. There is almost no difference between the fall times. For the illustration of the effect of duty cycle on the rise time, signals at different duty cycles are fed to the card and the results are shown in Figure 3-26. As can be inferred from the figure, there occurs delays in the output signals due to the capacitive effects. Although these delays are acceptable for many applications, improvements should be made to decrease these types of delays.

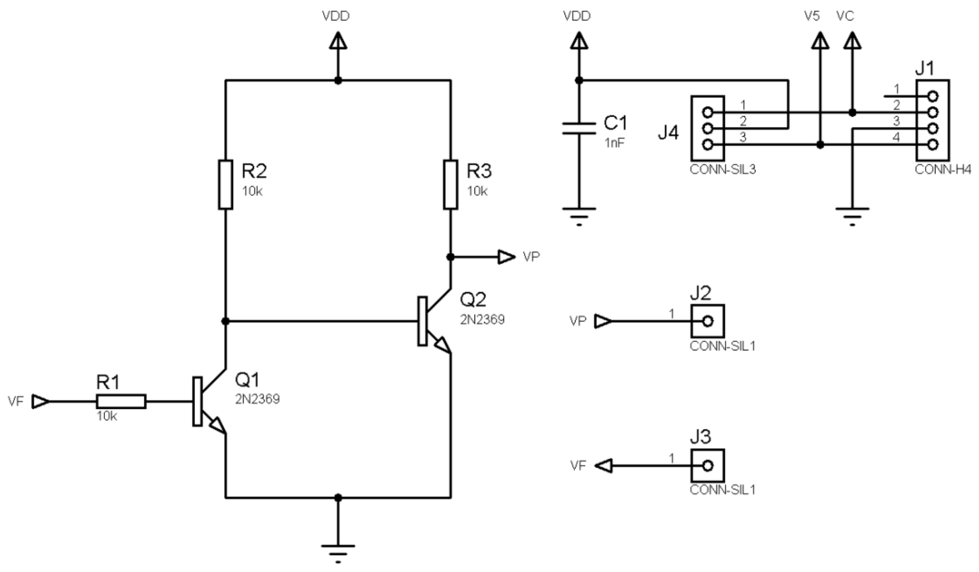


Figure 3-23 Schematic Design of the Digital Output Card

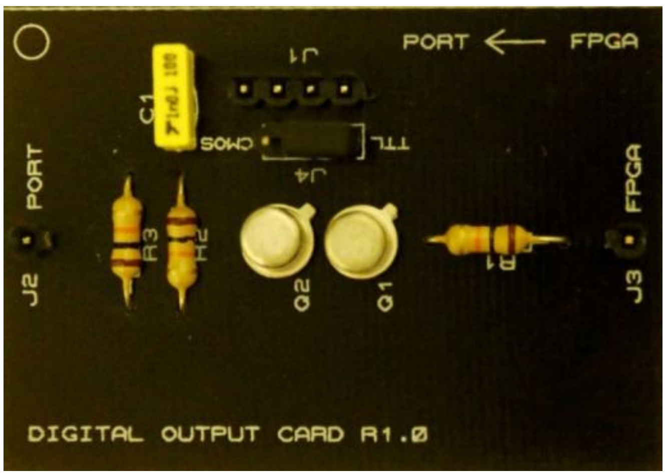
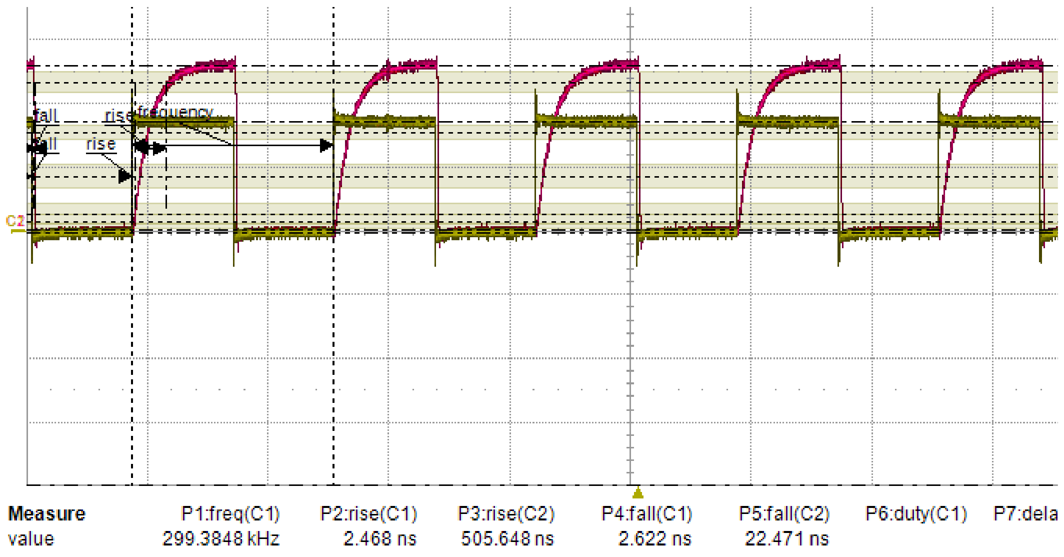


Figure 3-24 Digital Output Card



Measure	P1:freq(C1)	P2:rise(C1)	P3:rise(C2)	P4:fall(C1)	P5:fall(C2)	P6:duty(C1)	P7:dela
value	299.3848 kHz	2.468 ns	505.648 ns	2.622 ns	22.471 ns		

Figure 3-25 Output Signal of the Digital Output Card at 300 kHz

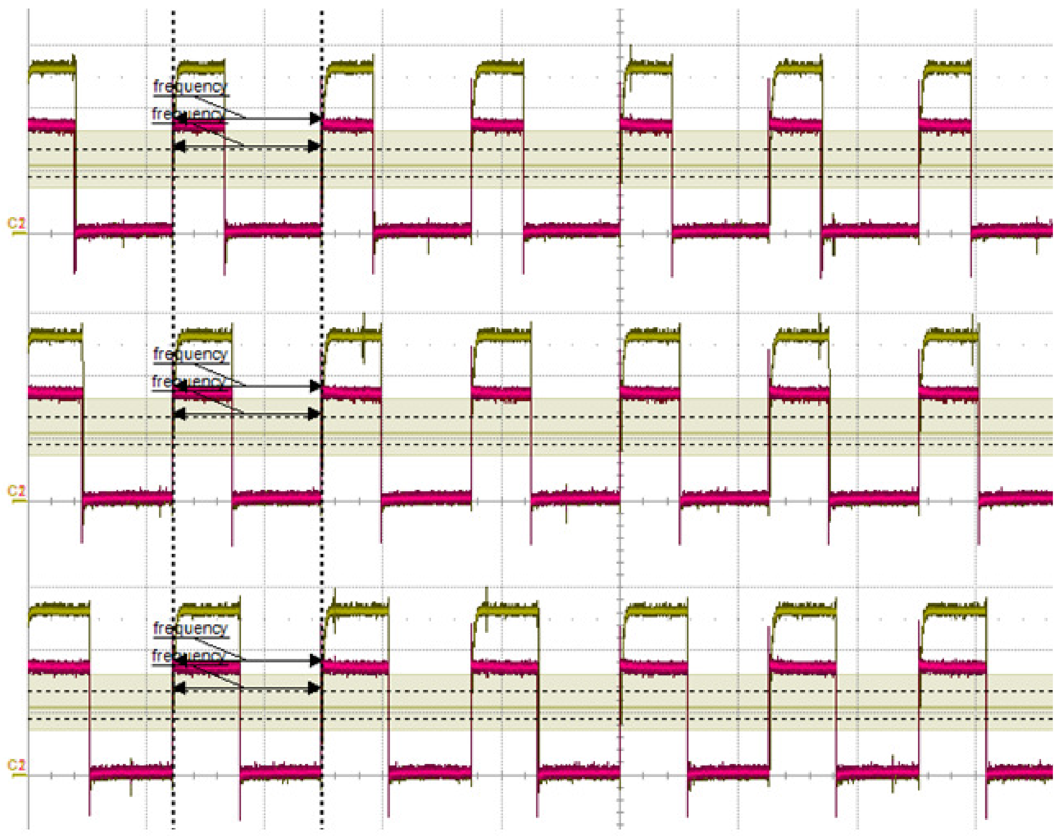


Figure 3-26 Output Signal of the Digital Output Card for Various Duty Cycles

3.7 Closure

In this chapter of the thesis, the developed FPGA interface is elaborated in a detail manner. This interface can be used to connect various peripheral devices to FPGA without any electronic concerns. When the overall design is considered, the modifications on the interface are carried out by hand utilizing the auxiliary switches and jumpers on the daughter cards. In the further designs of the interface, it is planned to replace switches with fast analog switches that can transfer the current in two ways. These analog switches may also be used to configure the channels electronically.

CHAPTER 4

COMMAND GENERATION METHOD UTILIZING SEGMENTATION AND POLYNOMIAL APPROXIMATION

The first developed command generation method utilizes segmentation with respect to the inflection points of the trajectory and then employs function approximation paradigms [33] such as, Chebyshev, Legendre, Bernstein – Bezier, etc. to represent the continuous trajectory efficiently. In this chapter, after the importance of segmentation is discussed, some background information on polynomials is given. According to the performance evaluation of polynomial types in MATLAB, the most successful one is realized on the FPGA Development Board via two different approaches, hardwired and embedded softcore processor.

4.1 Segmentation

Segmentation is preferable when a complex trajectory is to be approximated since a single polynomial (with extremely high order) might not be sufficient to approximate the whole trajectory to the desired accuracy. When the trajectories are segmented before approximation, the magnitude of errors decreases remarkably. On the other hand, segmentation also brings additional computational loads to the approximation methods. Hence, during the evaluation

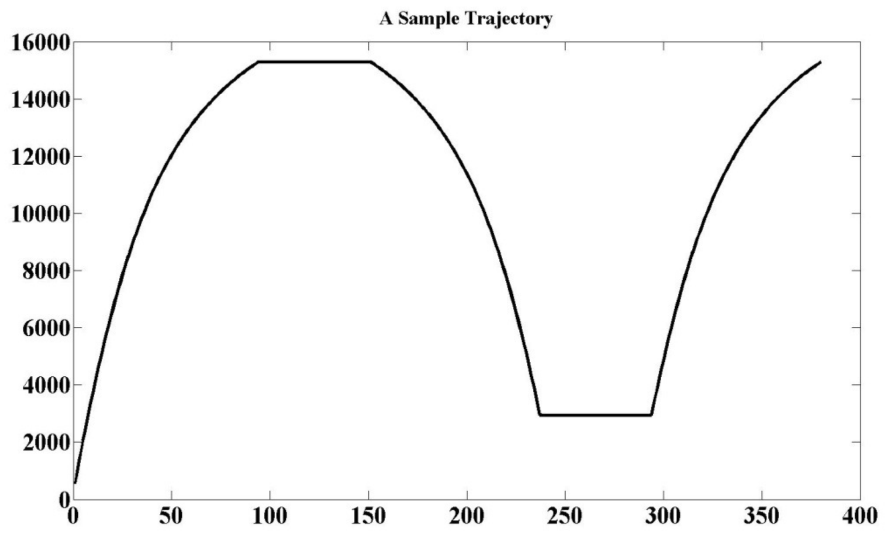


Figure 4-1 A Sample Trajectory

of methods, this extra effort should be also assessed. For illustration, a sample trajectory given in Figure 4-1 is to be approximated with Chebyshev Polynomials (CPs) with two different approaches. In the first approach, the trajectory is directly approximated by ten CPs. In the second approach, it is first divided into sections considering to the inflection points and thus each section is approximated with the same number of CPs.

In MATLAB, the given trajectory is approximated and the resulting sequence in Figure 4-2 is obtained. As can be seen from the figure, the approximated trajectory has significant deviations if compared to the original one. Similarly, the approximation error is demonstrated in Figure 4-3. When it is compared with the original trajectory in Figure 4-1, one can easily perceive that at the inflection points of the original trajectory, the magnitudes of the errors are much greater.

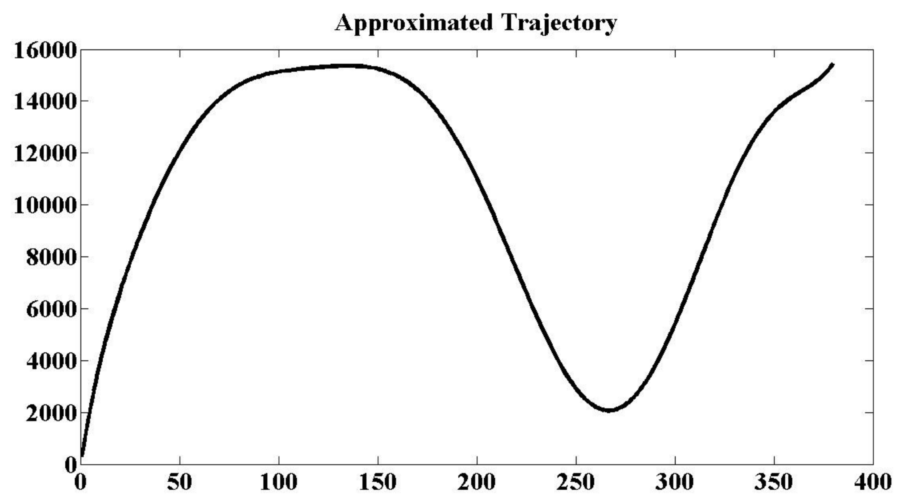


Figure 4-2 Approximation without Segmentation

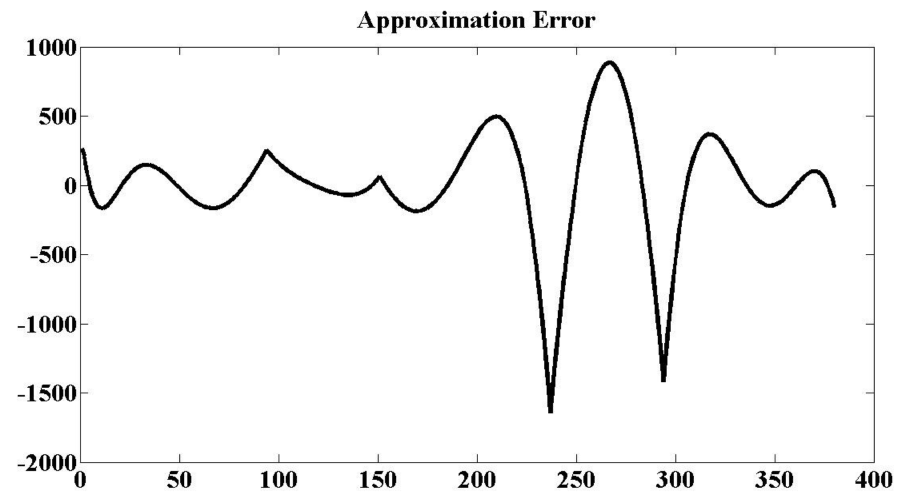


Figure 4-3 Approximation Error without Segmentation

There are two possible ways to decrease the magnitude of errors at these points: **i)** number of polynomials used for approximation can be increased, **ii)** the trajectory can be divided into sections. When the hardware implementations are considered, the former solution is not preferred due to the resulting computational burden. Thus, the sample trajectory is divided into five segments according to the inflection points shown in Figure 4-4. Each segment is approximated utilizing ten CPs. As can be seen from Figure 4-5, the segmentation before approximation gives much better results than the previous method. The error profile is plotted in Figure 4-6 and it is compared to that of its counterpart. Note that the maximum error in Figure 4-6 is about one-sixth of the maximum error in Figure 4-3. Hence, the segmentation before approximation is most feasible approach to approximate curves with C^0 continuity.

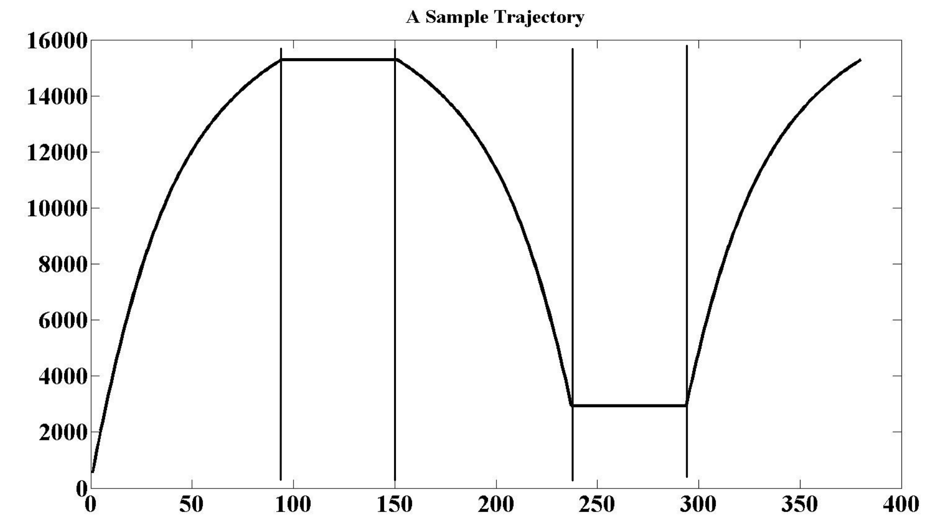


Figure 4-4 Segmented Sample Trajectory

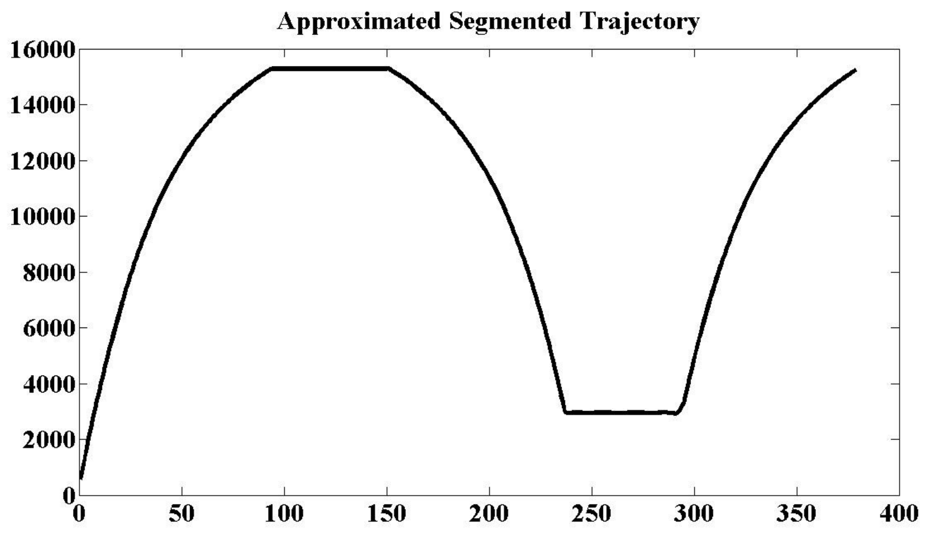


Figure 4-5 Approximated Segmented Trajectory

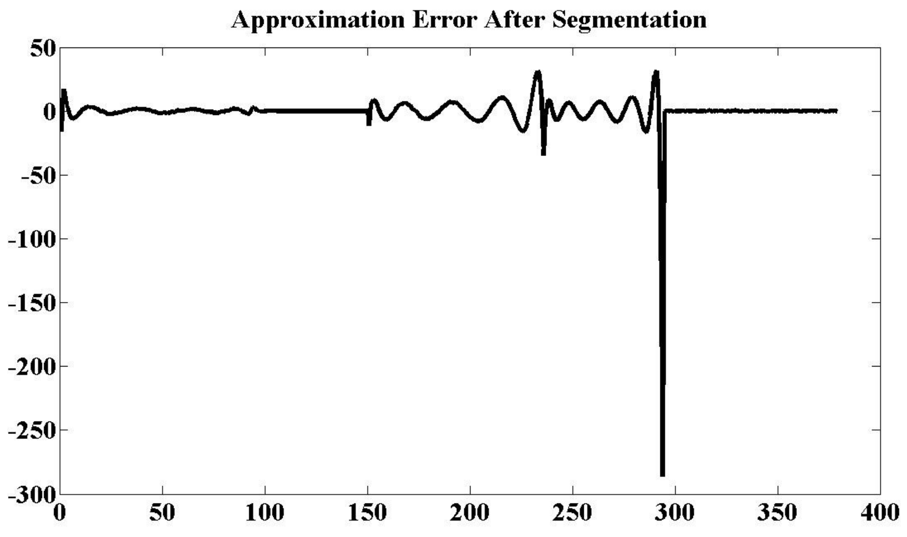


Figure 4-6 Approximation Error After Segmentation

4.2 Polynomial Techniques

In this proposed command generation method, various polynomial approximation techniques are applied onto the segmented trajectory. Before evaluating the different types of polynomials, some background information on such methods will be given to maintain self-containment of this thesis.

Suppose that a command trajectory is to be approximated with an nth order polynomial in the interval [x_{min}, x_{max}]:

$$y(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + a_n x^n \tag{4.1}$$

If there exists sufficient ($m \geq n$) number of data points {(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_m , y_m)} on the trajectory, the unknown polynomial coefficients {a_0, a_1, …, a_n} in (4.1) can be calculated to represent the trajectory. In cases where the number of data points is equal or greater than the number of polynomial coefficients, the remaining coefficients can be determined using the least squares. That is, with these coefficients, (m+1) equations can be written:

$$Y = X \cdot A \tag{4.2}$$

$$A = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{4.3}$$

$$X = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix} \tag{4.4}$$

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{4.5}$$

Polynomial coefficients in (4.2) are determined using the pseudo-inverse method described in [34]:

$$A = (X^T X\ )^{-1} X^T Y \tag{4.2}$$

In polynomial approximation methods, the basis functions are the key factors to approximate the functions more accurately. When (4.1) is considered, the exponential functions $\{x^1, x^2, \ldots, x^{n-1}, x^n\}$ can be treated as the basis functions. This natural selection of basis function does not result in a good representation due to the fact that the employed basis functions are not mutually orthogonal. That is,

$$\int_{x_{min}}^{x_{max}} x^i x^j dt \neq 0 \tag{4.7}$$

where i, j ∈ {0, 1, …, n} and i≠j. In approximations, it is a wise choice to use orthogonal functional forms as the basis functions according to the characteristics of the function to be approximated. On the other hand, these selected basis functions should easily be calculated also converge to the solution with less error [35].

In this polynomial approximation based command generation method, Chebyshev, Legendre, and Bernstein polynomials are elaborated and most suitable one is used in FPGA implementations. When the basis functions of these polynomials are compared, Legendre and Chebyshev polynomials use cosine functions and Bernstein polynomials use binomials.

4.2.1 Chebyshev Polynomials

Chebyshev polynomials (CPs) are usually preferred for hardware implementation, since the CPs approximate non-periodic signals within a limited range better than other polynomial types. This feature is well suited for the commanded trajectories encountered in motion control applications.

CPs, which are strictly defined over an interval $x \in [-1, 1]$, are formed recursively to yield a set of orthogonal polynomials. When a different interval is considered, a change of variables is employed to be able to utilize the CPs. In approximation theory, the CPs are regarded as important polynomials due to the fact that roots of first-kind CPs are used as the nodes in polynomial interpolation. As a result, CPs decreases the problem of Runge's phenomenon and also makes an approximation which is very close to the polynomials of the best approximation for a continuous trajectory under the maximum norm. There is another reason why CPs are good for approximation: When the series is truncated at some term, the error resulted from this cut-off is very close to the first term after the cut-off. This makes the computation of error easy. First five CPs (T_0 – T_4) are given in Figure 4-7, which are formed according to the recurrence formula:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad T_0 = 1, \quad T_1(x) = x, \qquad n \geq 1 \qquad (4.8)$$

Any trajectory can be approximated by

$$y(x) = \sum_{n=0}^{\infty} a_n T_n(x). \qquad (4.9)$$

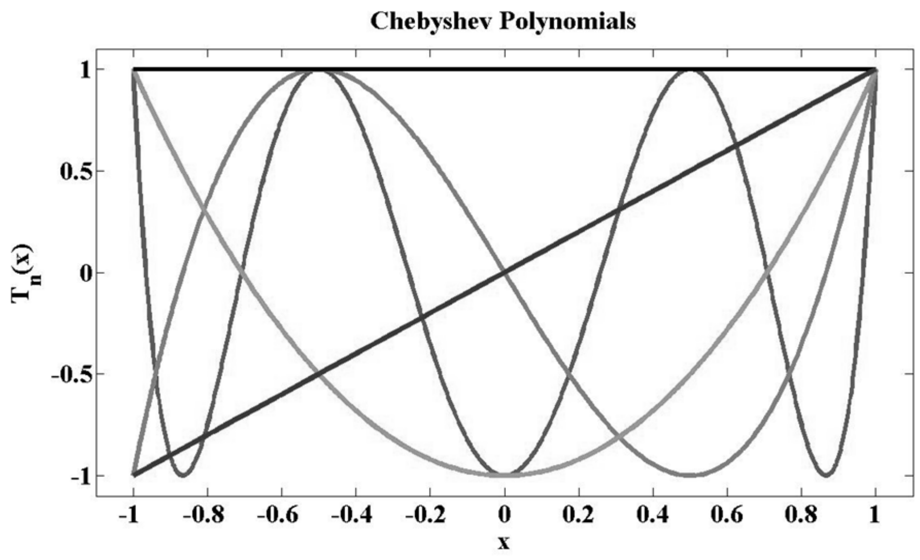


Figure 4-7 First Five Chebyshev Polynomials

When the basis functions of CPs are compared to the basis functions of Fourier, it can be inferred that there is a similarity between them. After employing change of variables, the different basis functions are formed from the trigonometric functions of Fourier via mapping $z = cos(\theta)$.

$$T_n(z) = cos(n\theta) \tag{4.10}$$

Then the following two series become equivalent:

$$f(z) = \sum_{n=0}^{\infty} a_n T_n(z) \tag{4.11}$$

$$f\big(cos(\theta)\big) = \sum_{n=0}^{\infty} a_n \, cos(n\theta) \tag{4.12}$$

Although the series $f(z)$ is not periodic, $f\big(cos(\theta)\big)$ is periodic in θ with a period of 2π. Due to the equivalence of series, the exponential convergence of Fourier series guarantees the convergence of Chebyshev series.

4.2.2 Legendre Polynomials

For non-periodic trajectories, Legendre Polynomials (LPs) can be used instead of CPs in the interval [-1, 1]. If the computational domain is divided into sub domains, the formulation of Legendre basis functions becomes simpler than that of Chebyshev basis functions. The convergence characteristics of these two different polynomial types are the same, but the maximum error of Legendre series is worse than the maximum error of Chebyshev series. Another difference between these polynomials is that the CPs oscillate uniformly over the interval but the LPs are non-uniform and their magnitudes are small when compared to their counterparts [35]. Just like CPs, LPs can be defined as

$$y(x) = \sum_{n=0}^{\infty} a_n L_n(x). \tag{4.13}$$

These polynomials (L_n) are calculated utilizing the recurrence formula and shown in Figure 4-8.

$$(n+1)L_{n+1}(x) = (2n+1)xL_n(x) - nL_{n-1}(x)\,, L_0 = 1, L_1(x) = x, n \geq 1 \tag{4.14}$$

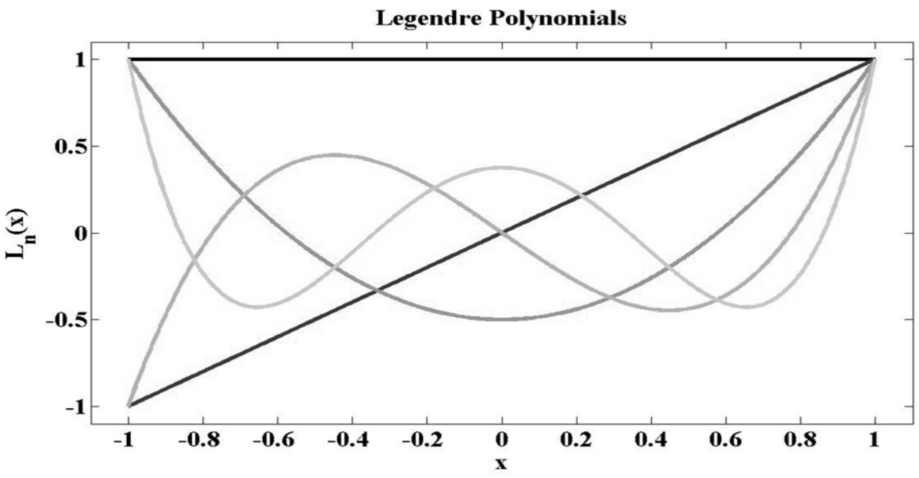


Figure 4-8 First Five Legendre Polynomials

4.2.3 Bernstein Polynomials

The main difference of Bernstein Polynomials (BPs) from Chebyshev and Legendre polynomials is that the BPs are defined in the interval [0, 1] rather than [-1, 1] and always positive. One of the popular application areas of BPs is the generation of Bezier curves in computer graphics. Bernstein basis polynomials are formed using the expression:

$$B_{i,n}(x) = \binom{n}{i} x^i (1-x)^{n-i} \quad (i = 0,1,\cdots,n) \tag{4.15}$$

The Bernstein polynomials are plotted in Figure 4-9 and defined as the linear combinations of Bernstein basis functions given in the equation:

$$y(x) = \sum_{k=0}^{n} a_n B_{k,n}(x). \tag{4.16}$$

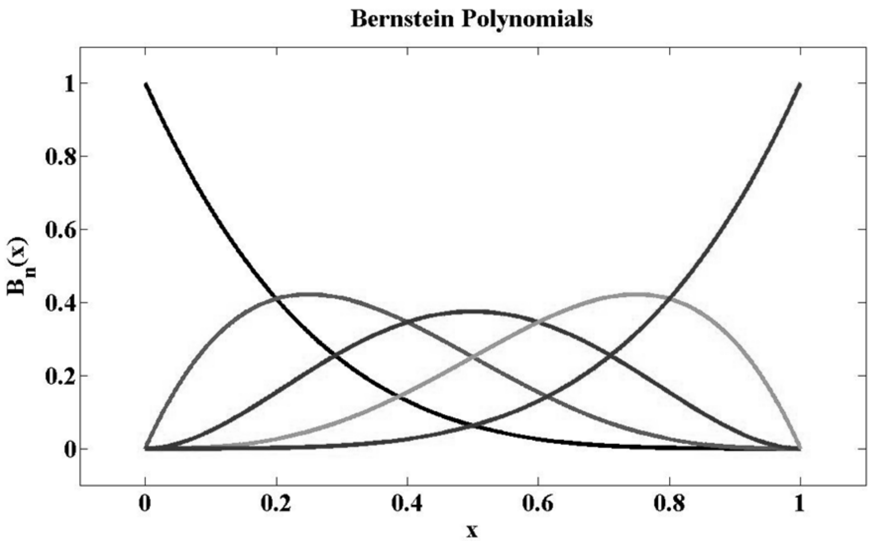


Figure 4-9 First Five Bernstein Polynomials

In order to decrease the computational complexity of Bernstein polynomials, (just like other polynomial techniques) Bernstein polynomials can also be defined in a recursive fashion:

$$B_{k,n}(x) = (1-x)B_{k,n-1}(x) + xB_{k-1,n-1}(x) \tag{4.17}$$

4.3 Performance Evaluation

Before implementing the proposed command generation method on FPGA, polynomial approximation techniques are compared and the most suitable one for command generation is selected for FPGA implementation. This evaluation is carried out in MATLAB, which includes special functions that cannot be easily realizable on the FPGA.

The proposed command generation method based on segmentation and different polynomial approximation methods are applied on the command sequences generated for all six joints of a PUMA 560 manipulator. Figure 4-10 shows these

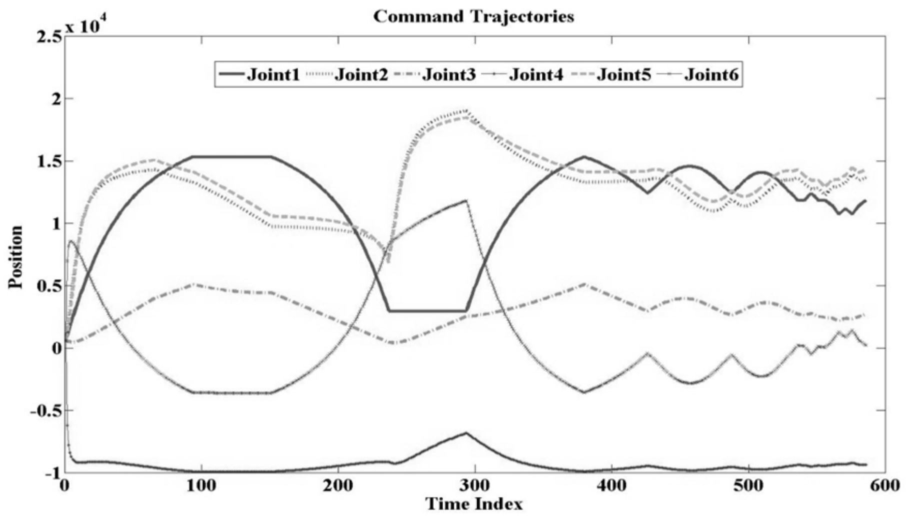


Figure 4-10 Command Trajectories of a PUMA Manipulator

trajectories in encoder counts. It is assumed that the quadrature encoder at each joint has the ability to generate 40000 (= 4 × 10000 counts/rev) counts in one revolution.

The evaluation is carried out for three different variations of the method: **i)** approximation, **ii)** segmentation and approximation, **iii)** segmentation, approximation and error compression.

4.3.1 Single Segment Approximation with Error Sequence Storage

In this version of the command generation method, the trajectories are approximated without any segmentation but with error sequence storage. Chebyshev, Legendre, and Bernstein – Bezier polynomials based approximation method is applied on the command trajectory of the first joint of PUMA 560 manipulator. Plot showing the effect of error root mean square (RMS) value on the compression ratio is provided in Figure 4-11. The number of polynomial

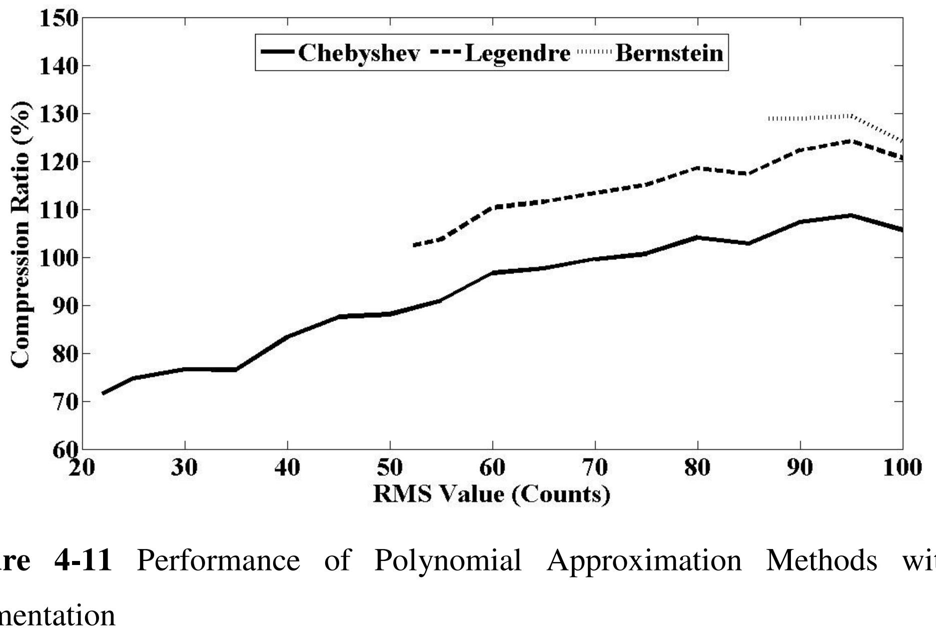


Figure 4-11 Performance of Polynomial Approximation Methods without Segmentation

coefficients used for approximations are also illustrated in Figure 4-12. These methods cannot approximate the trajectory below certain error values, since singularity problems (i.e. the determinant of inverted matrix in (4.6) approaches to zero) occur in the pseudo-inverse technique. Even if the solution of the problem were not ill-conditioned with increasing order, the performance of the approximation methods could not be regarded as acceptable in terms of the large number of coefficients involved to reconstruct the original trajectory. Since the approximation errors mostly lie outside the tolerable range, trajectories must be divided into sections from their inflection points.

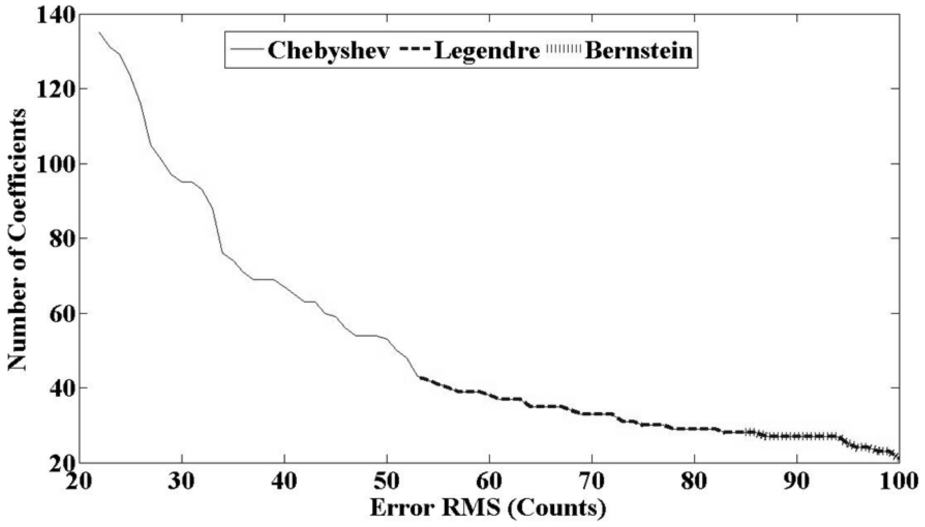


Figure 4-12 Number of Polynomial Coefficients for Different Error RMS Values

During the storage of polynomial coefficients, it is aimed to decrease the computational complexity of the implementation on the FPGA chip. Thus, the coefficients are first rounded to integers and error sequences are formed according to these integer coefficients.

It is critical to notice that while calculating the compression ratios for the above-mentioned techniques, all necessary parameters and error sequence are taken into account. The following expression is used to calculate the compression ratio of pure approximation methods:

$$r = \frac{\left\lceil \frac{N_c}{8} fix \left[\frac{log(c_{max} - c_{min})}{log(2)} + 1 \right] \right\rceil + \left\lceil \frac{N}{8} fix \left[\frac{log(e_{max} - e_{min})}{log(2)} + 1 \right] \right\rceil}{\left\lceil \frac{N}{8} fix \left[\frac{log(d_{max} - d_{min})}{log(2)} + 1 \right] \right\rceil} \tag{4.18}$$

where N is the length of the original data sequence; N_c is the number of coefficients used in the approximation; c_{max}, d_{max}, e_{max} and c_{min}, d_{min}, e_{min} represents the maximum- and the minimum values of the coefficients, the original data sequence, and the error sequence respectively.

4.3.2 Segmentation and Approximation with Error Sequence Storage

In this sub-section of the evaluation, the trajectories in Figure 4-10 are firstly segmented according to the inflection points and then polynomial approximation methods are employed on these segments. The results are given in Figure 4-13. In the method, the most important parameter is the approximation error RMS value. If it is selected to be small, the error in the approximation decreases but the number of polynomials and coefficients increase tremendously. Therefore, a proper RMS value should be selected minimum compression ratio. The formula required to calculate the compression ratio is the same with the one given in Section 4.3.1. As it is inferred from Figure 4-13 that the best performance is achieved for all approximation methods when the error RMS value is four. Relatively high compression ratios at the beginning of the plot are due to the number of polynomials used to achieve corresponding RMS value. After the value of four, compression ratio tends to rise for all polynomial types since the memory required for storing the errors increases. Lastly, when the three approximation methods are considered, Chebyshev Polynomials outperform the Legendre and Bernstein – Bezier polynomials for approximation.

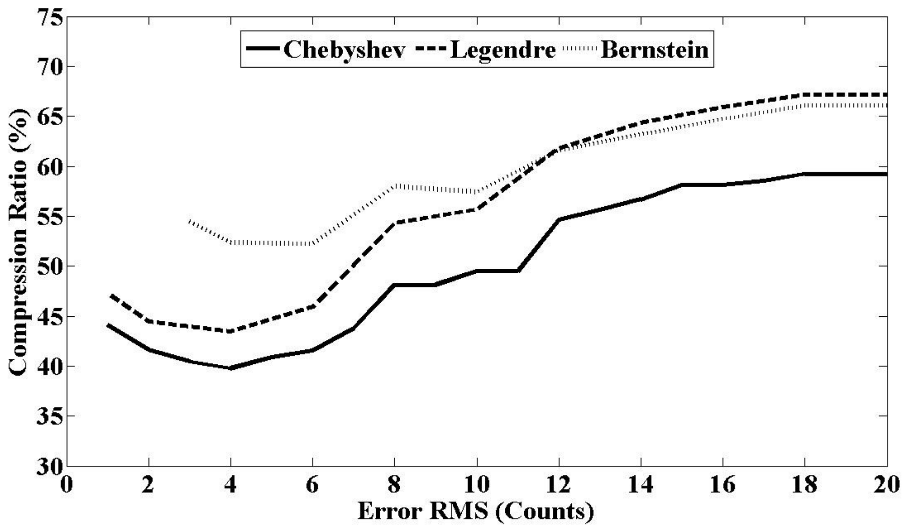


Figure 4-13 Performance of Polynomial Approximation Methods with Segmentation

4.3.3Segmentation and Approximation with Error Sequence Compression

As can be understood from the name of this sub-section, the main difference of this variation of the method is that the errors resulting from the approximation are compressed before storing. For the error compression, the data compression technique proposed in Chapter 5 is used. The performance of this variation is illustrated in Figure 4-14. As in the other variations, CPs outperform the other polynomial types and it can easily be inferred from the figure that this variation is the best of all variations of the command generation method based on polynomial approximation paradigms. By employing CPs, the original trajectory can be compressed to 23% of its original size. It should be noted that during the compression of error sequence, its first order of difference is taken and compressed.

The compression ratio calculation is carried out with the same equation given in Section 4.3.1 with an additional compression ratio parameter (r_e) for the error sequence as

$$r = \frac{\left[\frac{N_c}{8} fix \left[\frac{log(c_{max} - c_{min})}{log(2)} + 1\right]\right] + r_e \left[\frac{N}{8} fix \left[\frac{log(e_{max} - e_{min})}{log(2)} + 1\right]\right]}{\left[\frac{N}{8} fix \left[\frac{log(d_{max} - d_{min})}{log(2)} + 1\right]\right]} \tag{4.19}$$

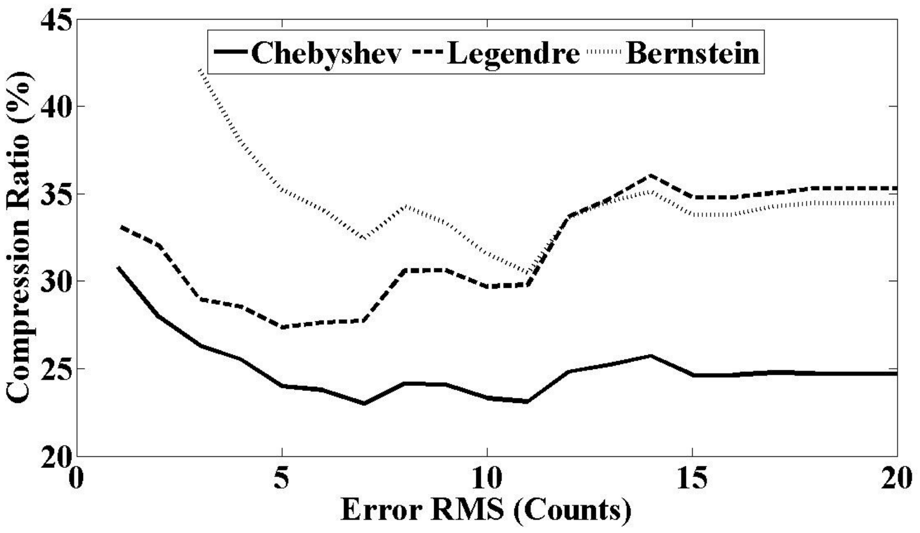


Figure 4-14 Performance of Polynomial Approximation Methods with Segmentation and Error Compression

4.4 FPGA Implementation

After the elaboration of different polynomial techniques employed on the trajectories of the PUMA manipulator (see Figure 4-10), it is turned out that the Chebyshev polynomials give much better results than the Legendre and the

Bernstein polynomials. Thus, the Chebyshev polynomials based command generation method is realized utilizing the Altera DE 1 FPGA Development Board (with Cyclone II FPGA) [36] via two different approaches. In the first approach, the command generation method is directly written in VHDL utilizing the schematic design property of software Quartus II 9.0 Web Edition. This technique will be referred to as "Hardwired" approach. In the second technique, (rather than writing directly in hardware description language) architecture is implemented in NIOS II Embedded Development Environment [37], where a softcore processor IP serving as an embedded microcontroller is deployed on the FPGA. In the following two sub-sections, the differences between these approaches will be discussed and their performances shall be evaluated using the joint-state trajectories generated for the PUMA manipulator.

4.4.1 Hardwired Approach

In the first approach of FPGA implementations, the proposed command generation method is realized by directly writing the algorithm in VHDL. During writing, schematic design property of Quartus II 9.0 Web Edition is used. With this property of the software, it is much easier to sustain and track the communication between different modules performing specific tasks. Schematic design generating the commands for the first joint of PUMA manipulator, whose trajectories are given in Figure 4-10, is provided in Figure 4-15. In this design, there are mainly four different modules and two memory units. The modules are Driver Module (DM), Splitter Module (SM), RS-232 Module, Floating Point Operation Module (FPOM). The first memory unit is used to store Chebyshev polynomials. In this memory unit only hundred discrete values of first eight Chebyshev polynomials are stocked. Thus, in this hardwired implementation there is a restriction on the maximum width of the segments, which is hundred. If the length of the segment is less than hundred, then a proper change of variables is applied to the segment. On the other hand, in the second memory unit polynomial coefficients and the widths of the segments are stored. These coefficients are passed to the FPOMs in a proper order for multiplication with the Chebyshev polynomials. After the multiplications, the results are summed in other FPOMs and sent to the RS-232 module in order to transfer the commands to the computer. Table 4-1 represents the allocated resources in FPGA for the implementation of the method with the hardwired approach. It can be inferred from the table that the almost half of the logic elements are used for this method, which can be regarded as high. The number of pins used is very low, since overall design does not have any communications with peripheral devices. It just internally generates the commands. It uses 37% of the multipliers due to the FPOMs embedded in the design. For a better illustration of resource usage, Figure 4-16 can be viewed. It can also be understood from the figure that about half of the resources are consumed by the architecture. The most important thing of this implementation is that it takes only 750 μs to generate 586 commands, which is fast for command generation systems. In the following sub-sections, the four main modules used in the design are explained in detail.

Table 4-1 FPGA Resources used in Hardwired Approach

Total Logic Elements	11098 (45%)
Total Combinational Functions	10885 (44%)
Dedicated Logic Registers	2644 (11%)
Total Registers	2644
Total Pins	4 (2%)
Total Memory Bits	26880 (4%)
Embedded Multipliers 9-bit Elements	49 (37%)
Total PLLs	0

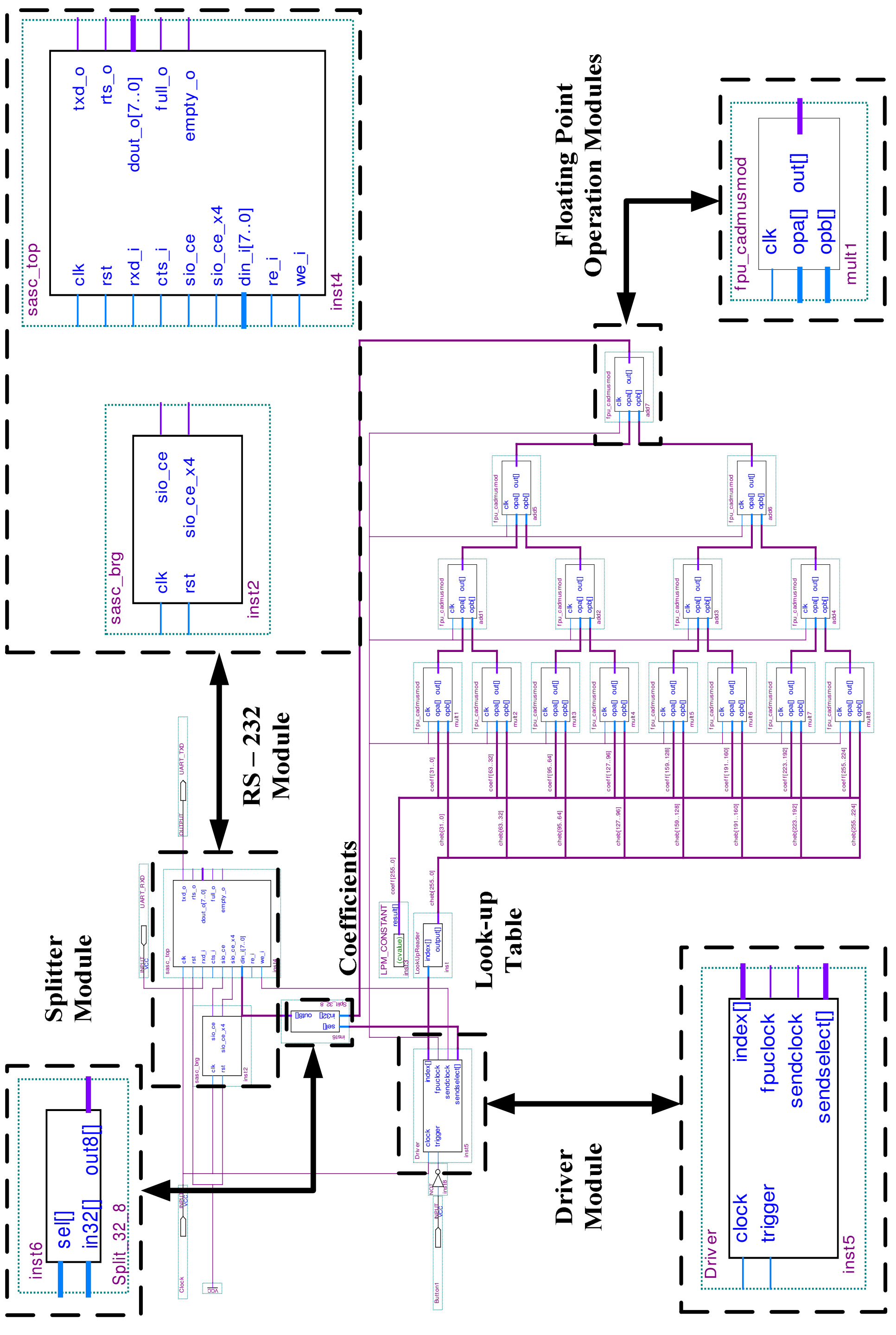
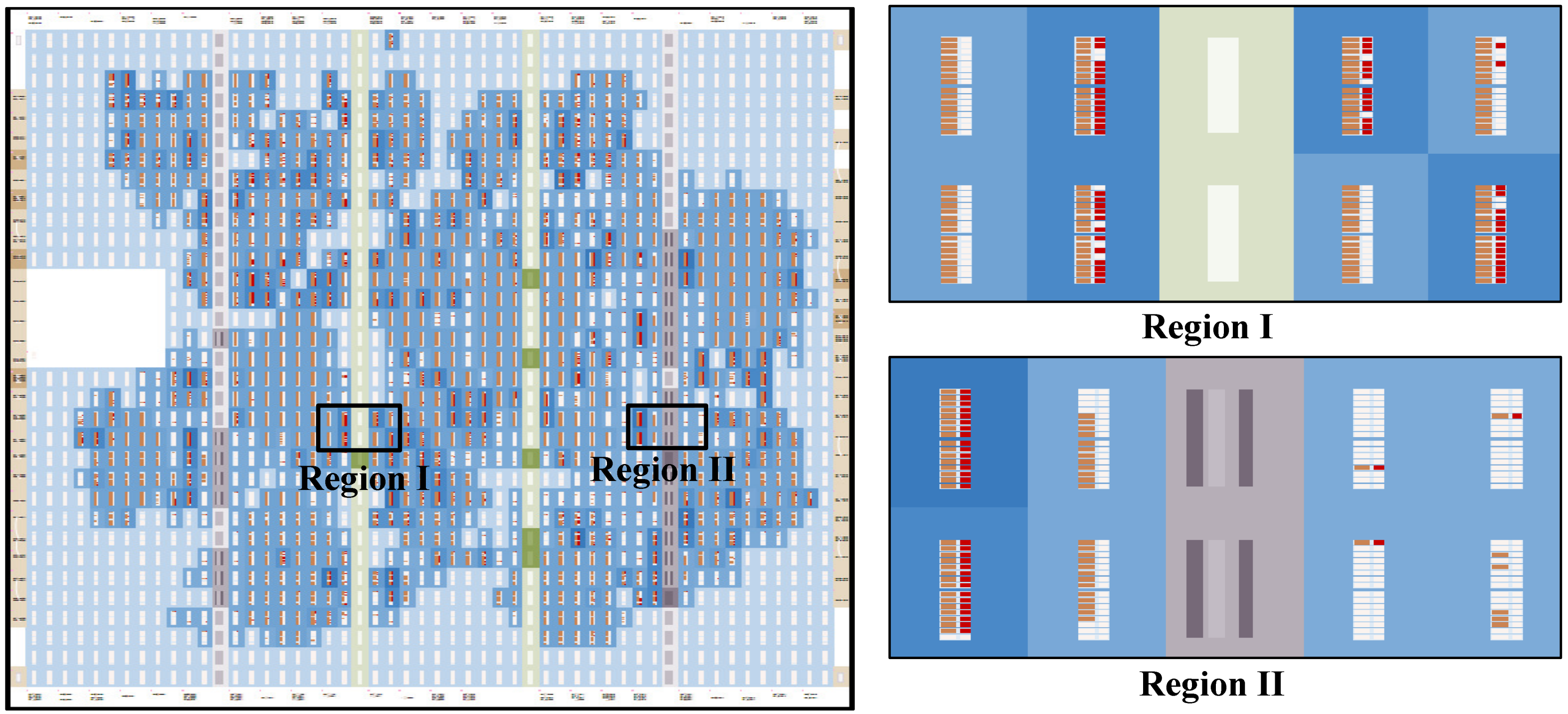


Figure 4-15 Hardwired FPGA Implementation of the Method



Background	⬛	LAB	🟦	Pin Goup	🟫	Location Assignments	⬜			
Selection	🟦	Logic Element	🟧	DSP	⬜	Registers	🟥			
Highlight	🟦	Memory	🟩	Local Interconnect	🟨	User Assigned LogicLock Regions	🟦			
Block Border	⬛	Pin Goup	🟫	Global Interconnect	🟨	Fitter Placed LogicLock Regions	🟪			
Connection	🟥	DSP	⬜	Pin	🟧	Low Power	🟦			
Path	⬜	Logic Element	🟧	Ports	🟨	High Speed	🟨			
Bundle	⬜	Memory	🟩	Differential Pin Pair Connections	⬜	Virtual IO	🟨			

Figure 4-16 Floor Plan of the Synthesized Digital Circuitry for the Hardwired Implementation

4.4.1.1 Driver Module

Driver Module can be regarded as the manager module of the design. It communicates with all the modules and memory units to make the command generator operate properly. As can be seen from Figure 4-15, the DM has two inputs and four outputs. The first input is the global clock available on the FPGA board, which is 50 Mhz in this design, and the second input is the trigger input given by an external logic to the system to generate commands. Among outputs of the DM, `index` is sent to the Look-up Table. With the index value, Look-up Table outputs the values of CPs to the first set of FPOMs. `fpuclock` is generated and transferred to the FPOMs to perform mathematical operations in the desired order of the DM. Communication with the RS-232 Module is sustained with the output `sendclock`. Since the DM knows exactly how many cycles are necessary for mathematical operations, it outputs this clock accordingly. `sendselect` output is given to the SM to split the 32-bit command values into 8-bit values. With all these inputs and outputs the DM operates as the main module of the design.

4.4.1.2 Floating Point Operation Module

There are fifteen FPOMs in the design. Eight of them perform multiplications, and the rest sums the results of these multiplications. The original of this module was developed by Usselmann [38]. This current version is simplified to respond only to the needs of the design. The module, whose schematic is provided in Figure 4-15, has three inputs and an output. The first input `clk` is received from the DM module and the other inputs are the 32-bit floating point data to be used in mathematical operations. The only output `out` is the result of multiplication or addition in this architecture. It is sent to the other FPOMs for further operations or sent to the RS-232 module.

4.4.1.3 Splitter Module

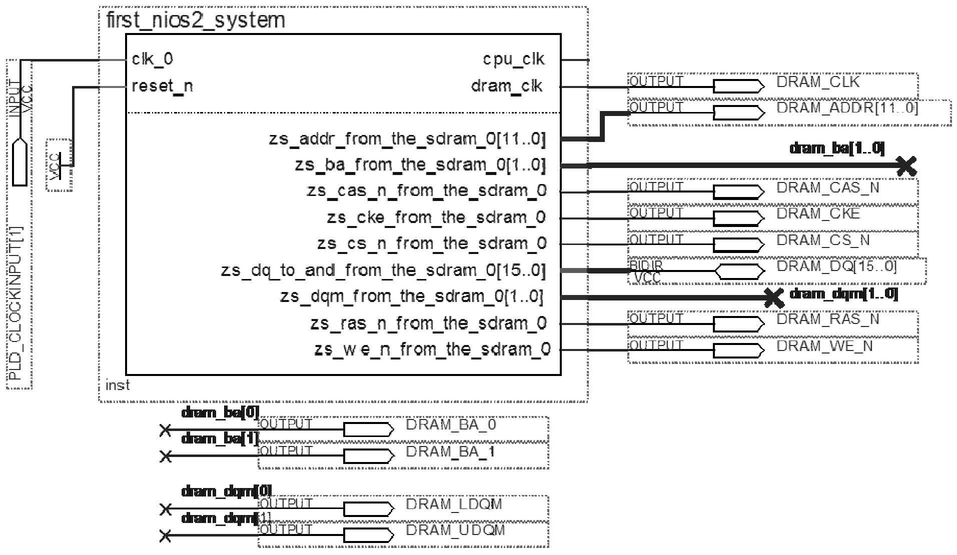
Splitter module, shown in Figure 4-15, is the simplest module used in the design. It just takes the input `sel` as the clock and splits the incoming 32-bit wide floating point data `in32` into four 8-bit wide values. Finally sends these data to the RS-232 module directly. The reason why there is need for such segmentation is that the available RS-232 module can operate with 8-bit data values.

4.4.1.4 RS-232 Module

The RS-232 Module, developed by Usselmann [40], actually consists of two sub-modules; `sasc_brg` and `sasc_top` as named in Figure 4-15. The module on the left side is the baud rate generator. The baud rate is adjustable using the divisor registers in the module according to the global input given as input to the module. The module on the right side is responsible for the communication with the PC and the incoming data.

4.4.2 Embedded Softcore Processor Approach

In the second approach, the command generation algorithm is written in C programming language and the resulting are cross-compiled to run on a softcore processor deployed on the FPGA. The "machine code" is then downloaded to this processor. Note that the embedded softcore is designed in the NIOS II Embedded Development Environment. Schematic design of this *algorithmic state machine* (ASM) is shown in Figure 4-17. The softcore Intellectual Property (IP) has a Synchronous Dynamic Random Access Memory (SDRAM) Unit additional to the basic micro-processor units. SDRAM is used to store Chebyshev polynomials, coefficients, and widths of the segments. Due to the usage of an external memory unit, hardware resources used in the FPGA is less than the ones used in the first



c

Figure 4-17 Softcore FPGA Implementation of the Method

approach as can be seen in Table 4-2. The processor occupies only about one-fifth of the resources. This can also be verified from the chip plan given in Figure 4-18. For the given trajectory, the implementation only uses 61 kB (0.007% of SDRAM) of memory. The time necessary for generating a command sequence of 586 points is around 247 ms, which is much higher than the first approach. It is critical to note that the basis functions (i.e. Chebyshev polynomials) can be generated in advance and it takes roughly 213 ms.

Table 4-2 FPGA Resources used in Softcore Approach

Total Logic Elements	3444 (18%)
Total Combinational Functions	3085 (16%)
Dedicated Logic Registers	1914 (10%)
Total Registers	1966
Total Pins	39 (12%)
Total Memory Bits	28992 (12%)
Embedded Multipliers 9-bit Elements	0
Total PLLs	1 (25%)

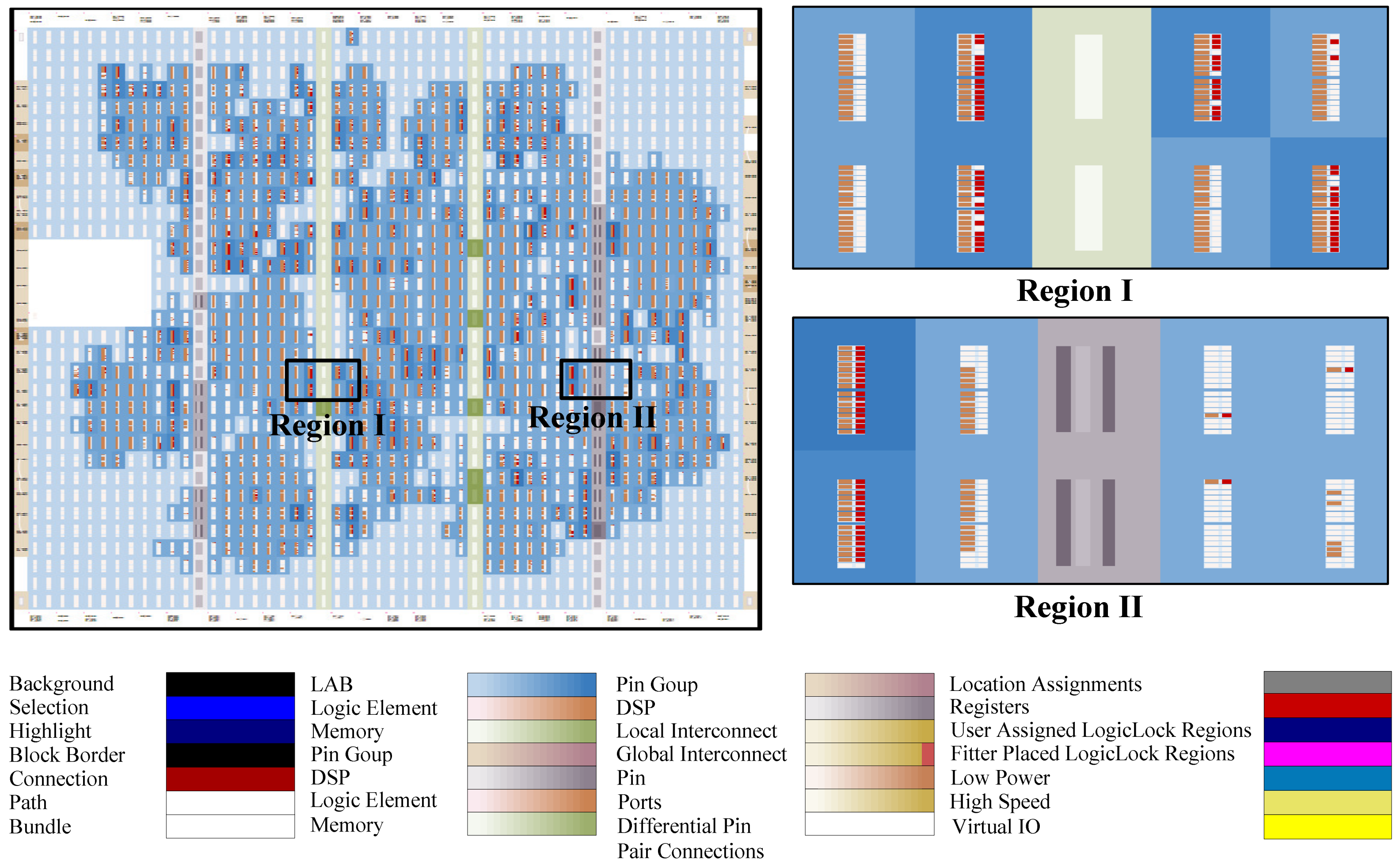


Figure 4-18 Floor Plan of the Synthesized Digital Circuitry for the Softcore Implementation

4.5 Closure

In this chapter, command generation method based on segmentation and polynomial (Chebyshev, Legendre, and Bernstein) approximation was proposed. The method was then implemented on the FPGA development board using two different approaches. Pure polynomial approximation was not used for the implementations, since during the elaboration of the method in MATLAB environment it turned out that the segmentation of the trajectory was inevitable for small magnitudes of errors. The two implementations have their own advantages over the other. Hardwired approach is much faster than the softcore counterpart. On the other hand, the hardware resources used by the embedded softcore is about the half of resources occupied by the hardwired approach. To sum up, if there is no restriction on the usage of hardware resources, it is better to implement hardwired approach which is much faster than the other.

CHAPTER 5

COMMAND GENERATION METHOD UTILIZING DIFFERENCING AND COMPRESSION WITH VARIABLE FEED-RATE

The second developed command generation method consists of two phases: differencing and compression. The main difference of this method from the first one is that there is no approximation during command generation and as a result there are no representation errors. In this chapter, after the need for differencing (before compression) is explained, the proposed data compression technique will be elaborated. In the following section, the performances of various data compression methods are comparatively elaborated. According to the obtained results, variable feed-rate input (i.e. time/velocity scalar) is incorporated to the most successful method. Finally, the command generation algorithm is implemented on the FPGA board using two different approaches: hardwired and embedded softcore processor.

5.1 Differencing

Methods involving higher-order differences of time-sequences are applied to decrease the memory required for storage [4]. Note that in the literature this technique is referred to as differencing or relative encoding [3]. Rather than storing directly the whole command trajectory, it is beneficial to store the differentiated data along with the necessary initial values. Higher-order difference of a sequence can be represented as

$$\nabla q = q(k) - q(k-1), \tag{5.1}$$

$$\nabla^2 q = \nabla q(k) - \nabla q(k-1), \tag{5.2}$$

$$\nabla^{\mathrm{n}} q = \nabla^{\mathrm{n}-1} q(k) - \nabla^{\mathrm{n}-1} q(k-1). \tag{5.3}$$

In these equations, $\nabla^{\mathrm{n}} q$ represents the nth order difference and k is the index. As the order of difference increases, the memory needed for the storage of the sequence decreases. In order to represent the relationship between the memory requirement and the order of difference, several trajectories are formed and their differences are taken up to 7th order [1]. This result is shown in Figure 5-1. As can be seen from the figure, after the third-order difference, there is an increase in the memory usage since the sign of each data point frequently changes in an alternating fashion after the 4th order difference. Hence, the range of data broadens considerably. Therefore, the best solution for data storage is achieved when the order of difference is three or four for most of the motion control applications.

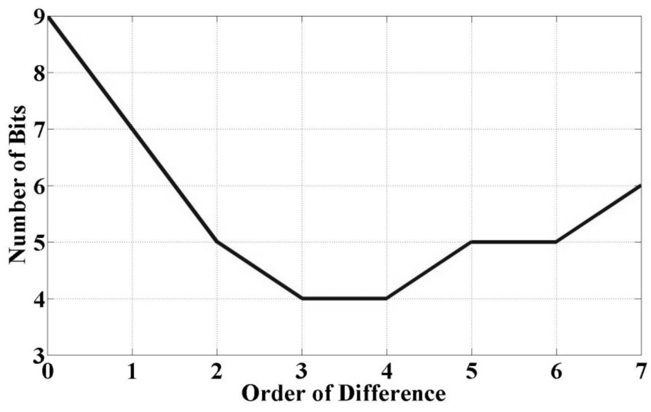


Figure 5-1 Effect of Order of Difference on Memory

Note that the original data can be extracted using the differenced data along with the initial values. For the first order differences, this integration (accumulation) operation can be expressed as

$$q(k) = q(k-1) + \nabla q(k) \tag{5.4}$$

When $k = 1$, an initial value $q(0)$ along with $\nabla q(1)$ is needed to calculate $q(1)$. If the order of difference is greater than one, number of necessary initial values increases. While calculating the compression ratios of the methods utilizing differencing, memory required for the initial values should also be considered. Differencing without compression can also be used as an alternative command generation method. For the illustration of this method, the second joint trajectory of the PUMA manipulator (shown in Figure 4-10) is reconsidered. When the area underneath the trajectory and its differences are plotted in Figure 5-2, it can be observed that the areas decrease remarkably. On the other hand, the

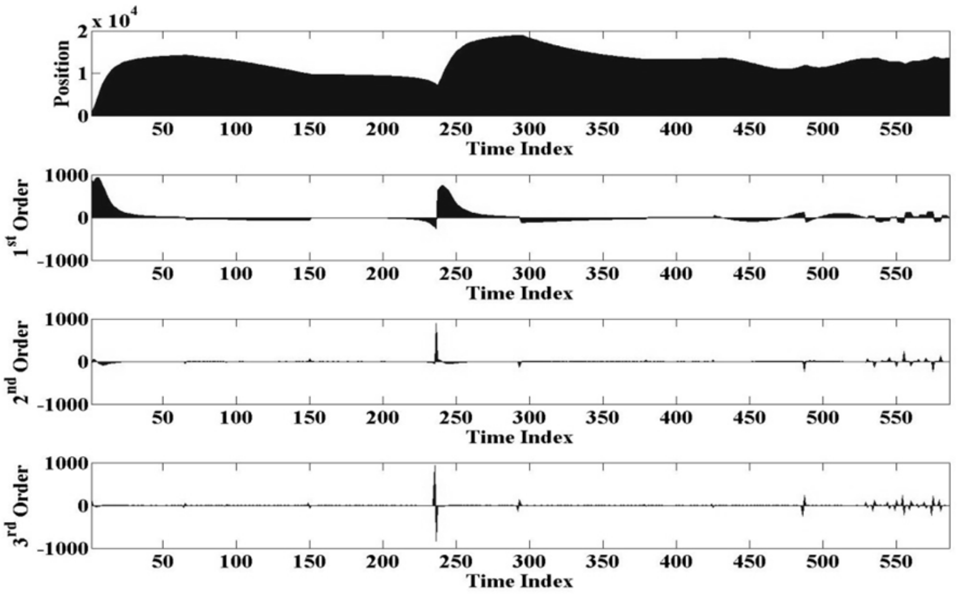


Figure 5-2 Second Joint Trajectory of PUMA Manipulator and Its Differences up to Third Order

compression ratios of differences do not change as remarkably as the areas change. This is due to the reason that there must be constant bit widths for the trajectory and the maximum value of the sequence determines this width. In Table 5-1, compression ratios are given up to sixth order of difference for all the trajectories of the manipulator. It should be noted that in the calculation of compression ratios, the necessary initial values for decoding are also considered.

Table 5-1 Compression Ratios vs Order Difference [%]

	Order of Difference					
Joint Number	**1**	**2**	**3**	**4**	**5**	**6**
1	73.25	66.61	66.52	73.07	79.62	86.17
2	73.25	73.25	79.80	79.71	86.26	92.81
3	61.55	69.22	69.12	76.79	76.68	84.24
4	99.90	92.69	92.59	92.50	92.40	92.30
5	73.25	73.25	79.80	79.71	86.26	92.81
6	93.27	86.53	86.44	86.35	86.26	86.17

5.2 Proposed Data Compression Algorithm

Data compression algorithms described in the second chapter are universal methods. That is, they can be utilized to compress any data type with diverse statistical attributes including text, audio, image, video, etc. While this generality feature can be regarded as an advantage, the implementation of such compression paradigms on FPGA chips can be often times quite complex and could drain considerable resources on a particular FPGA. On the other hand, the compression algorithm proposed in this thesis is specifically developed to deal with optical position encoder commonly encountered in motion control applications. Since the output of these sensors, which satisfy C^0 (and frequently C^1) continuity, can be conveniently represented as (signed or unsigned) integers, one can exploit such (temporal) sequences to come up with an efficient compression technique that is easier to implement on a FPGA chip with modest resources. In the following subsections, encoding and decoding algorithms of this method are to be explained in detail.

5.2.1 Encoding Process

The basic idea behind this technique is that when the higher-order differences of a reference trajectory (i.e position/location sequence) in a typical motion control application is computed, the (integer) values in the resulting sets do decrease considerably. Furthermore, since most motion control applications require constant velocity along the traced trajectory, the majority of the differentiated data is likely to be null (0) while the rest is composed of small integers in which the probability of occurrence is inversely correlated with the magnitude. Considering that a small integer number would require fewer bits, the difference data would take up significantly less memory if compared to the original data set. Unlike entropy-based (general) compression techniques (like Huffman coding), one can directly encode the difference data in this technique without calculating the probability density of the processed data owing to the fact that the special requirements associated with the motion control applications (due to operational concerns) tightly dictate the statistical distribution data beforehand.

Consequently, the proposed compression algorithm (to be referred to as the **ΔY Method** hereafter) is employed on the higher-order differences of the command trajectory (usually position). Once the differenced data of the command sequence is calculated according to the specified order, the resulting data is compressed (a.k.a. “compacted”) utilizing the ΔY algorithm. In Figure 5-3, a sample encoding process for the third-order difference is illustrated. Note that the compressed code consists of three fields (Sign, Amplitude, and Length Fields) and initial value set. The sign field includes sign bits: 0 and 1 represent positive- and negative numbers respectively. Note that if the magnitude of data is zero, no sign bit is assigned for this special case. Hence, the length of the sign field equals to the difference between the number of data points (in the differenced set) and the number of zeros in the data. Similarly, the amplitude field encodes the absolute values of the data sequentially as binary numbers with variable length. To extract the differenced data, another field (a.k.a "length field"), which yields the length of each value in the amplitude field, needs to be formed. As can be seen from Figure 5-3, this field contains sequences of 1's and 0's in an alternating manner. Once can detect the length of a particular number in the amplitude field by simply counting the bits in between two consecutive transitions (0-to-1 or 1-to-0) detected in the length field. Lastly, the order of differencing, and the initial values are needed for lossless decompression. The initial values, which are used to initialize integrator (or accumulator) states, depend on the order of the difference. That is, the number of integrators used in decoding is equal to the order of difference.

Original Command Sequence = {555, 983, 1354, 1710, 2058, 2400, 2736, 3068, 3394, 3715, 4031, 4341, 4646, ...}

3rd **Order Difference** = {42, 7, 2, 0, 2, -2, 1, 0, -1, 1, ...}

Encoded Data

Initial Values = {555, 428, -57}

Sign Field :	0	0	0	-	0	1	0	-	1	0	···
Amplitude Field :	101010	111	10	0	10	10	1	0	1	1	···
Length Field :	111111	000	11	0	11	00	1	0	1	0	···

Figure 5-3 Sample Encoding Process for ΔY Method

5.2.2 Decoding Process

Architecture implementing the decoding process of the ΔY technique is illustrated in Figure 5-4. Decoding starts out with the comparison of the consecutive bits in the length field to detect the transitions in this field. After the length of a particular binary data residing in the amplitude field (l_i) is determined through the bit comparison module and the counter, the information is passed to the left-shift register. Considering the length, the shift register extracts the amplitude of the data from the corresponding field and transfers the data to the differenced data module. After taking the sign value (utilizing another left shift register) from the sign field, the differenced data module completes its task and sends the data to the integration module. In the mean time, the initial values are transferred to the integration module and the original data by accumulating the differenced data in a sequential manner.

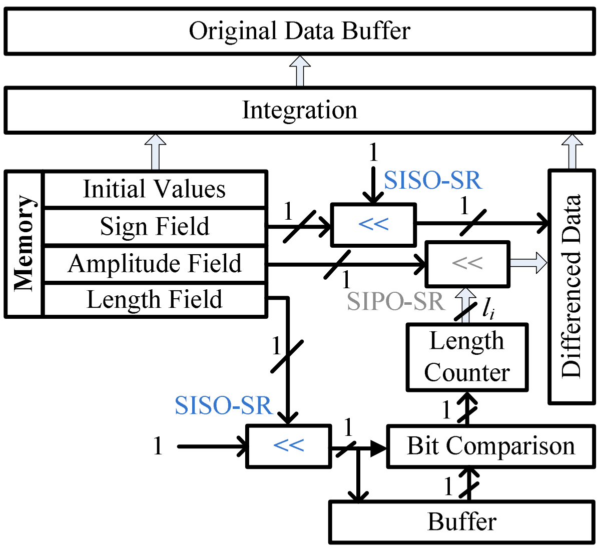


Figure 5-4 Decoding Process of ΔY Decompression Algorithm

It is critical to notice that as indicated in the first section of this chapter, the presented technique generates the original data (i.e. position) by accumulating the finite differences of that sequence in a successive fashion. For instance, when n equals to 3, the third-order difference (i.e. jerk) is accumulated to obtain second-order difference (i.e. acceleration/deceleration). The corresponding results are iteratively accumulated until the position at a particular instant in time is calculated. Hence, the discrete-time derivatives of the command sequence, which are frequently required by modern motion controller topologies, are inherently computed in this technique.

5.3 Performance Evaluation

In this section, Huffman, Arithmetic Coding, and the ΔY compression algorithms are applied on the trajectories of the PUMA 560 manipulator in MATLAB (see Figure 4-10). After the finite differences of the command trajectory for the first joint are computed for various orders, the data compression algorithms are employed to compress the differentiated trajectory data. Resulting compression ratios (in percent) are presented in Table 5-2. In the table, *n* represents the order of finite difference. Note that while calculating the compression ratios for the methods utilizing differencing, the memory required for the initial values is also taken into consideration. As observed from Table 5-2, if n > 1, the compression algorithms yield much better results owing to the fact that the increments of encoder counts from one sampling step to another (i.e. angular velocity) are still quite high as the robot performs a jerky motion throughout the followed trajectory in this particular example. Another conclusion to be drawn from the table is that the performance of the ΔY algorithm is superior to those of the others. With n = 3, the command sequence can be compressed to about one-fourth of its original size. Notice that the results given in Table 5-2 are also in good agreement with Figure 5-1. Up until the third order, the compression ratio decreases and after that there is an increasing trend. This situation may be explained by the decline in the frequencies of the numbers in the sequence and the expansion in the range of the data values.

Table 5-2 Compression Ratios for Various Orders

	Compression Ratio (%)		
n	**Huffman**	**Arithmetic Coding**	**ΔY Method**
0	197.1	181.4	182.9
1	131.0	118.6	77.9
2	34.1	32.7	30.3
3	34.8	31.9	23.6
4	46.4	42.3	30.8
5	60.6	54.1	39.3
6	81.7	70.9	49.6

It is critical to notice that while calculating the compression ratios for the above-mentioned techniques, all necessary parameters to extract the original command sequence (including compressed code, initial values, dictionary tables, etc.) are taken into account. The following expression is used to calculate the compression ratio of the ΔY technique:

$$r = \frac{\left\lceil \frac{1}{8}\left(\sum_{i=1}^{N-n} 2l_i + N - n - n_0\right)\right\rceil + \left\lceil \frac{n}{8}\left[\frac{log(d_{max} - d_{min})}{log(2)} + 1\right]\right\rceil}{\left\lceil \frac{N}{8} fix\left[\frac{log(d_{max} - d_{min})}{log(2)} + 1\right]\right\rceil} \tag{5.5}$$

where *N* is the length of the original data sequence; *n* is the order of finite difference; *l* is the binary length of each data; n_0 is the number of zero magnitude data; d_{max} and d_{min} represents the maximum- and the minimum value of the original data sequence respectively.

To be able to determine which compression algorithm is suitable for command sequences, the aforementioned methods are applied to all trajectories generated for the PUMA 560 manipulator after taking third-order finite differences of the angular position data (in encoder counts). The results are shown in Table 5-3. It is clearly seen from the table that the proposed method leads better results than the contending techniques.

Table 5-3 Compression Ratios for Third Order Differences

		Joint Number					
		1	**2**	**3**	**4**	**5**	**6**
Method	**Huffman**	34.8	43.9	36.0	37.0	46.0	41.3
	Arithmetic	31.9	40.2	33.8	34.8	42.0	37.9
	ΔY	23.6	24.2	24.1	22.2	24.7	25.3

5.4 Command Generation with Variable Feed-rate Input

A novel command generation scheme, where the programmed velocity along the traversed trajectory can be changed dynamically, is elaborated in preceding section. In CNC applications, the speed (i.e. feed-rate) through the course of motion is generally modified by external input (like feed-rate override knob). Under some extreme cases (such as the control scheme of an electro-discharge machine), it might be desirable to reverse the direction of motion as dictated by an external source. Therefore, the proposed command generation method is to be augmented to accommodate a variable feed-rate input.

With this property, the users will be able to change the rate of command generation in both forward and reverse directions. During generation, when there is a need for the intermediate command values, a linear interpolator should be incorporated to the design. That is, this unit is to interpolate between the two decoded command values based on the following expressions:

$$a_k = a_{k-1} + f_k \ (mod \ f_{max}) \tag{5.6a}$$

$$m := \begin{cases} m-1, & a_{k-1} + f_k < 0 \\ m+1, & a_{k-1} + f_k > f_{max} \end{cases} \tag{5.6b}$$

$$u_k = u_{m-1} + \frac{(u_m - u_{m-1})a_k}{f_{max}} \tag{5.7}$$

where u represents the decoded commands at the interval $m \in \{0, 1, \ldots, N\}$; k is the time index. Similarly, $f_k \in \{-f_{max},\ldots -1, 0, 1, \ldots \ f_{max}\}$ indicates the current value of the feed-rate input to the system while $f_{max} \in Z^+$ denotes the maximum feed-rate at which commands could be generated. Note that the variable (a_k) in (4.7) essentially serves as a time scaling factor.

In Figure 5-5, a sample interpolation is carried out with a feed-rate of $(3/8)f_{max}$. That is, if the sampling time is selected as 0.008 s, then with the specified feed-rate the new sampling time becomes 0.003 s. As can be seen from the figure that before the 4th interpolated command is generated, the difference value is updated and the next three commands are generated according to the new difference value.

The shaded area underneath the 2nd original command can be regarded as the error of the interpolation algorithm. After each original data point, extrapolation can also be used rather than interpolation. In the extrapolation case, it is guaranteed to

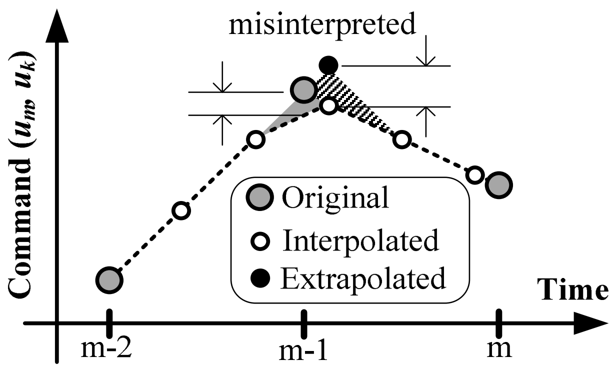


Figure 5-5 Interpolated Data

generate the original command. On the other hand, duration of the representation error of the extrapolation is always larger than the one of interpolation. Another approach to eliminate the representation errors at least at the original command points, the original data can be generated regardless of the sampling time.

The proposed command generation method with variable feed-rate is evaluated in MATLAB before implementing it on an FPGA development board. The feed-rate profile in Figure 5-6 is applied to the original command trajectory illustrated in Figure 5-7. The feed-rate profile is formed such that all the commands are first generated in the forward direction and then in the reverse direction with continuously changing feed-rate. There occur some command representation errors (as shown in Figure 5-8) at each original data points since the interpolation algorithm (described in the previously) is not capable of generating commands at these points. When Figure 5-7 and Figure 5-8 are considered together, it can easily be inferred that the larger errors occur at the inflection points of the trajectory.

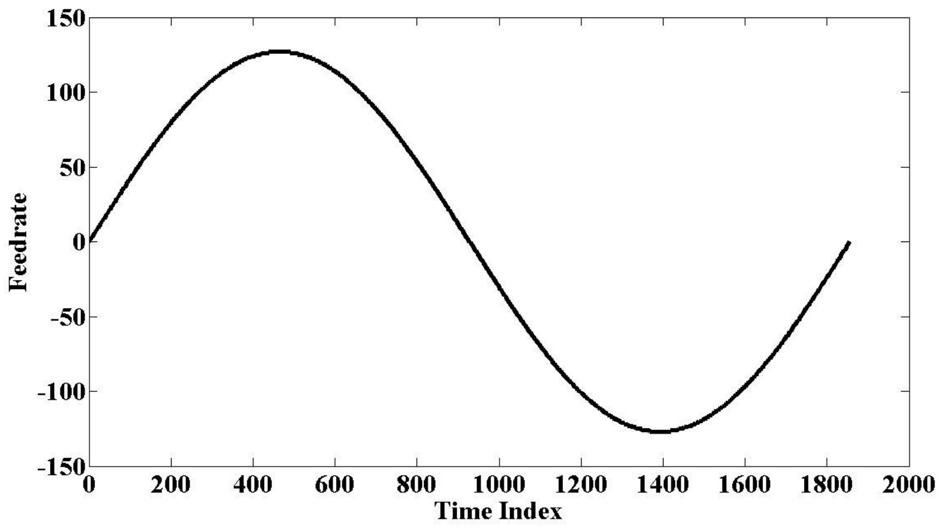


Figure 5-6 Feed-rate Profile

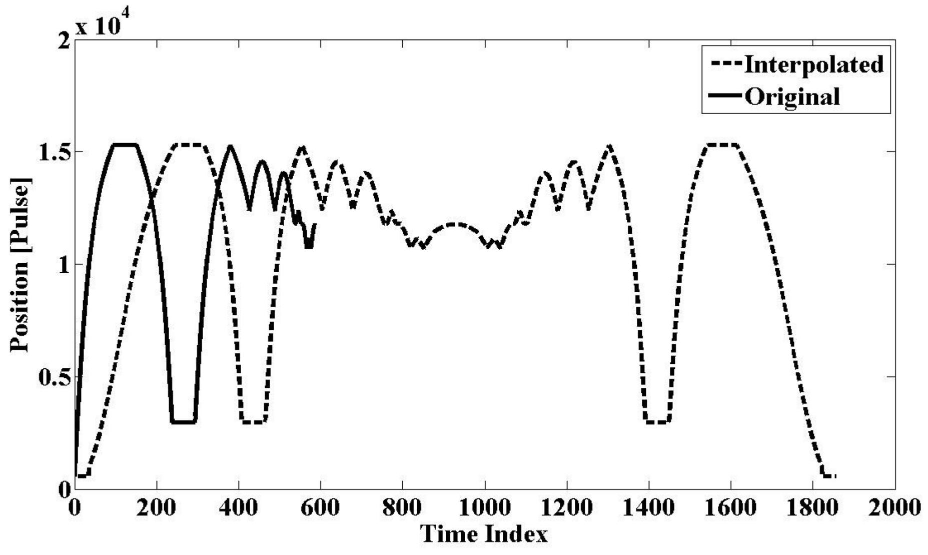


Figure 5-7 Interpolated and Original Command Sequences

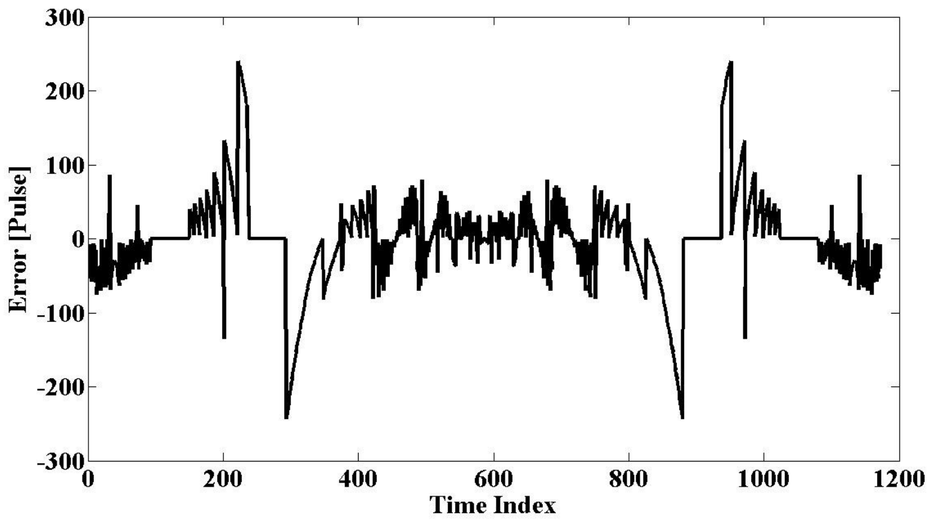


Figure 5-8 Command Representation Errors

5.5 FPGA Implementations

After the elaboration of different data compression techniques employed on the trajectories of the PUMA manipulator, it is turned out that the ΔY compression method exhibits superior performance over the Huffman and the Arithmetic compression techniques. Thus, the ΔY method based command generation paradigm is realized utilizing the Altera DE1 FPGA Development Board (with Cyclone II FPGA) [36] via two different approaches. In the first approach, the command generation method is directly written in VHDL utilizing the schematic design property of software Quartus II 9.0 Web Edition. This technique will be referred to as "hardwired" approach. In the second technique, (rather than writing directly in hardware description language) architecture is implemented in NIOS II Embedded Development Environment [37], where a softcore processor IP serving as an embedded microcontroller is deployed on the FPGA. In the following two sub-sections, the differences between these approaches will be discussed and their performances shall be evaluated using the joint-state trajectories (see Figure 4-10) generated for the PUMA manipulator.

5.5.1 Hardwired Approach

In this approach, the proposed command generation algorithm is realized by a finite state machine (FSM) which is directly designed through the use of VHDL. During this phase, the schematic design property of Quartus II 9.0 Web Edition is used. With this property of the software, it is much easier to sustain and keep track of the communications among different modules performing specific tasks. The schematic of the design is illustrated in Figure 5-9. In this design, there are mainly six modules: SRAM Controller, Memory Management Unit (MMU), Decoding Unit (DU), Accumulators, Interpolator, and RS-232 Module. Before explaining each module, the allocated resources on FPGA while implementing the method are represented in Table 5-4. As can be seen, only 9% of the total logic

Table 5-4 FPGA Resources used in Hardwired Approach

Total Logic Elements	1731 (9%)
Total Combinational Functions	1491 (8%)
Dedicated Logic Registers	911 (5%)
Total Registers	911
Total Pins	50 (16%)
Total Memory Bits	0
Embedded Multipliers 9-bit Elements	2 (4%)
Total PLLs	0

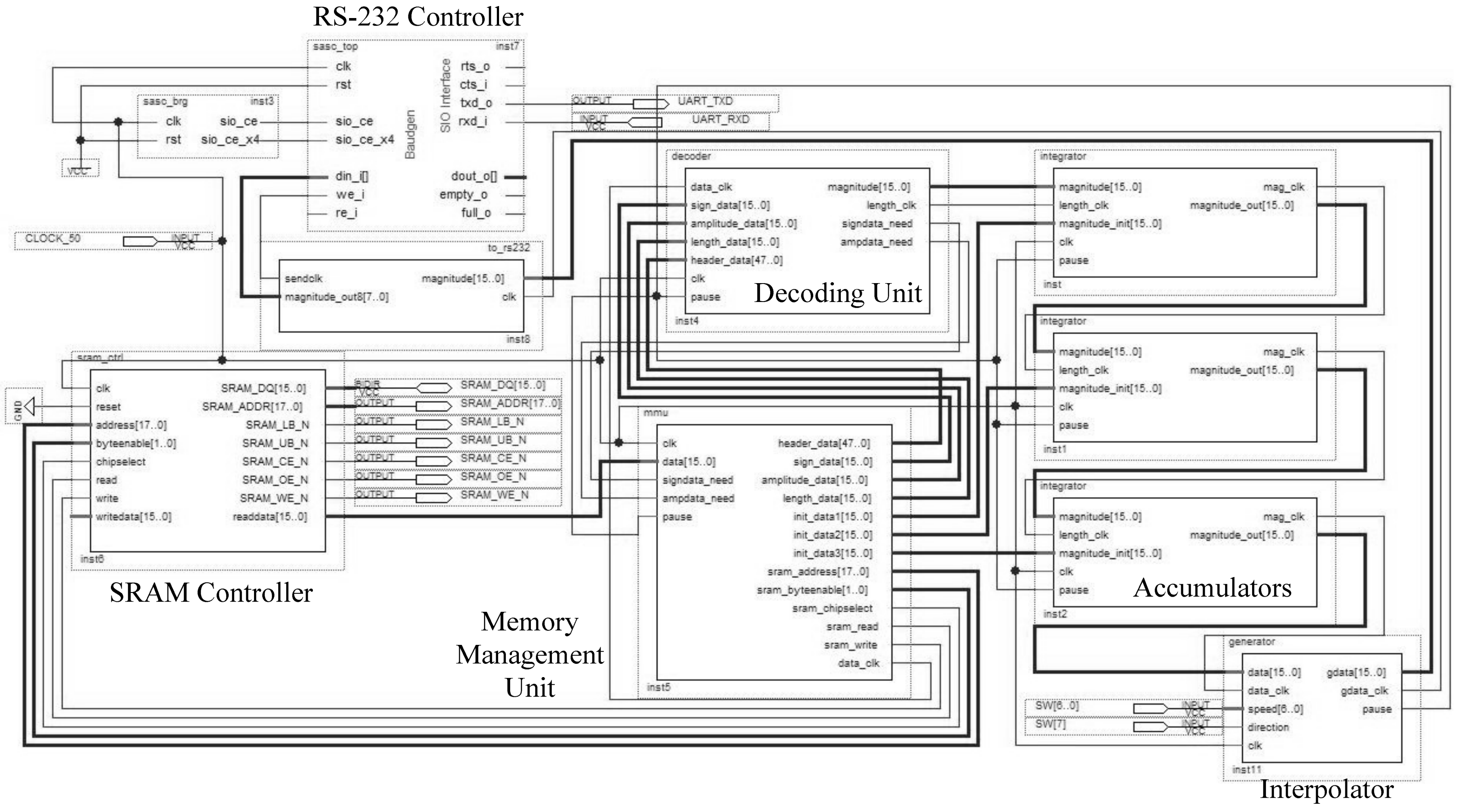


Figure 5-9 Hardwired FPGA Implementation of the ΔY Method

elements available on the FPGA chip are utilized in the design. Note that the FSM is implemented to generate commands for a single manipulator joint controller. Hence, one can implement 6 parallel FSMs to produce the commands for all joints of the PUMA manipulator without exhausting the resources of the chip. The number of pins used is a bit higher than ones in the implementation of the polynomial approximation methods. The reason of this increase is that the FPGA chip needs to be connected to the SRAM chip on the development board to store compressed (joint state) commands. For a better illustration of resource allocation, the floor plan of the "synthesized" digital circuitry on the chip is illustrated in Figure 5-10. As can be seen, only a small portion at the center of the chip is deployed to realize the corresponding architecture [38].

In order to evaluate the performance of the implementation in a detailed manner, the method is applied for all the trajectories of the manipulator (Figure 4-10). The results are given in Table 5-5. In the following sub-sections, the six main modules used in the design are investigated in detail.

Table 5-5 Time for the Generation of Command Sequences

Joint Number	Clock Cycles	Time (ms)
1	14569	0.29
2	15609	0.31
3	14039	0.28
4	13865	0.27
5	16045	0.32
6	15434	0.30

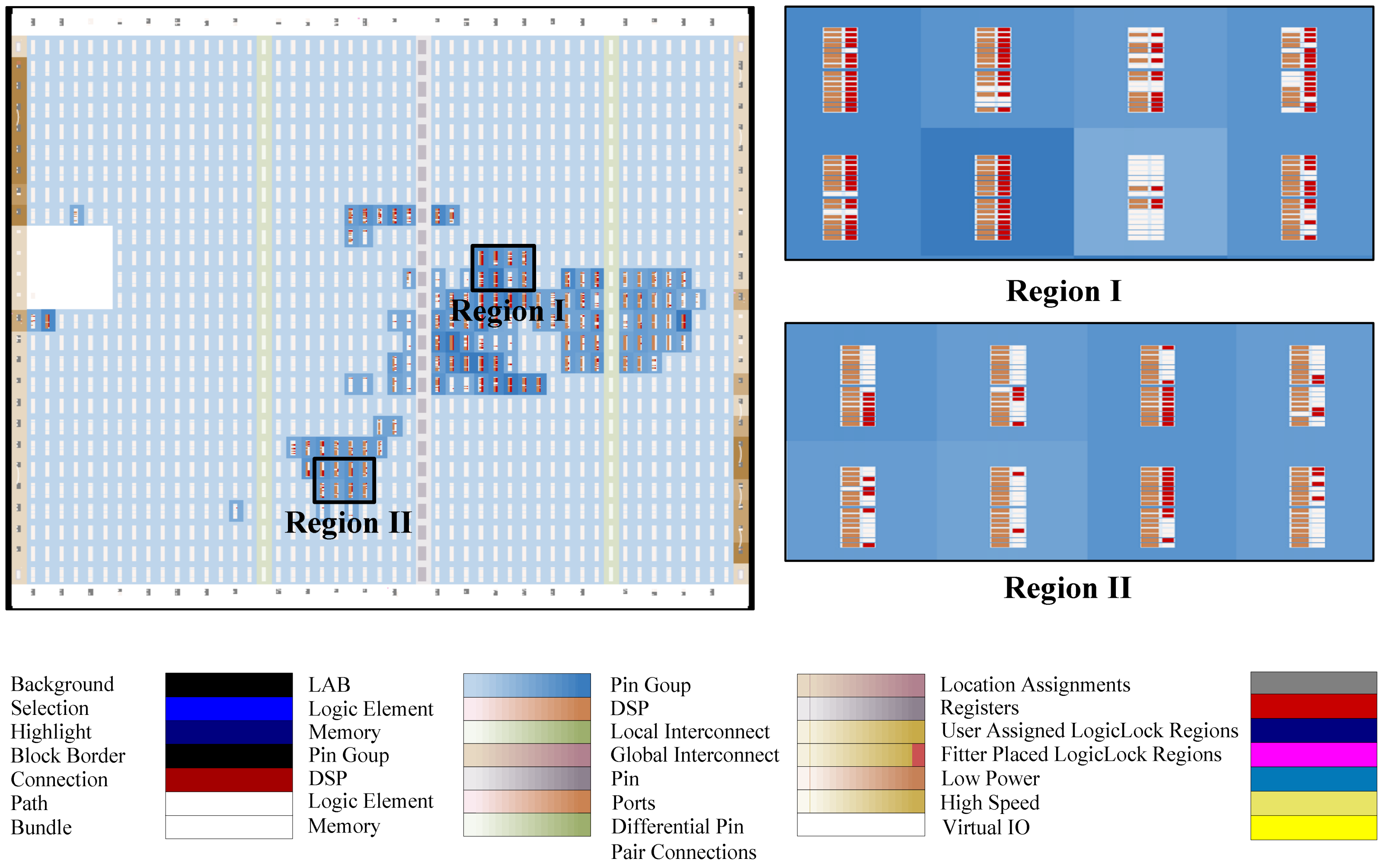


Figure 5-10 Floor Plan of the Synthesized Digital Circuitry for the 1st Architecture

5.5.1.1 SRAM Controller

The main task of the SRAM controller is to maintain the communication between the MMU and the SRAM located on the FPGA Development Board. It sends out the compressed data to the MMU (one by one in this case) according to the address information emanating from the MMU. Schematic version of the module is shown in Figure 5-11. All of the outputs of the module except `readdata` are connected to the SRAM chip on the development board, which is organized as 256K words by 16 bits. The output `readdata` is directly connected to the MMU. This output is responsible for sending the data on the specified address of the memory. The inputs `clk` and `reset` are the global clock and reset pins fed to the

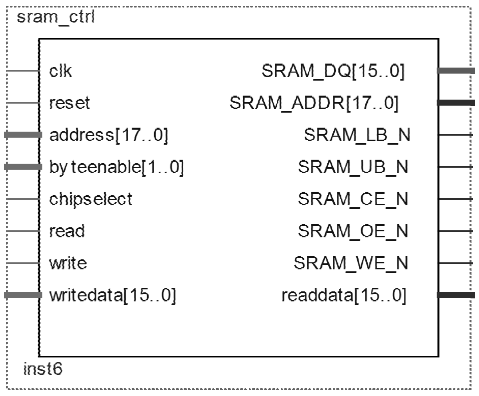


Figure 5-11 SRAM Controller

module. The rest of the inputs are connected to the outputs of the MMU. The `address` input to the module determines the value of output data to the MMU. `byteenable`, `chipselect`, and `read` inputs are set to high during to operation to be able to use the module. `write` input is set to low since there is no writing operation during the decoding process of the compressed data. In order to use the memory efficiently, the compressed code is structured as shown in Figure 5-12 for a generic command sequence. The first three words of

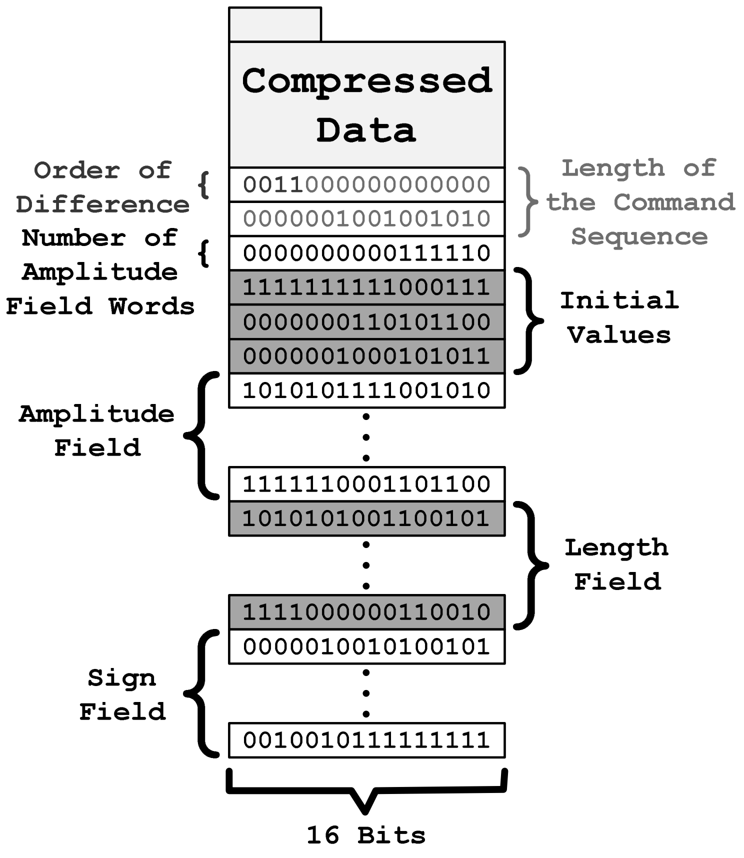


Figure 5-12 Compressed File Format

the compressed data can be regarded as the header. Initial 4 bits of the first word indicate the order of finite difference (where a maximum of 15th order for the differences can be represented). The rest of the first word and the second word (28 bits) are reserved for expressing the length of the command sequence. Finally, the last word of the header is used to specify the number of words reserved for the magnitude field, which indirectly determines the starting address of the sign field. After the header part, the initial values section is located. They are stored in the form of signed binary integers. The number of initial values necessary for integration is set by the order of finite difference which is represented with the first 4 bits of the data. After the information about the compressed data and initial values are given, the amplitude field is then stored in the proceeding words. Since the length of the header part and the number of initial values are known, the starting address of the amplitude field is easily determined during decoding. Note that the length and sign fields are located after the amplitude field. The starting addresses of these two fields are calculated via the number of amplitude field words stored in the third word of the compressed data. With the described data format, the compressed sequences are generated without any error.

5.5.1.2 Memory Management Unit

The MMU, whose schematic is shown in Figure 5-13, can be regarded as the core of the design since it communicates with all modules except the RS-232 Controller. Input signals to this module are limited when the number of output signals is considered. Input signals are only the data sent from the SRAM Controller (`data`), acknowledgment signals (`signdata_need` and `ampdata_need`) coming from the DU indicating that the unit is out of data, and the `pause` input set by the interpolator. Output signals are header data for the compressed file, three fields transferred to the DU, the initial values sent to the accumulators, and the necessary outputs connected to the SRAM Controller.

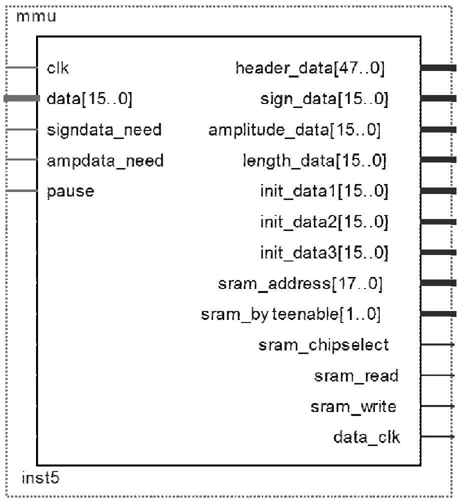


Figure 5-13 Memory Management Unit

The basic operating principles of the MMU are described in Figure 5-14. As can be seen, there are four states of this unit: i) *Get Header*, ii) *Get Initial Conditions*, iii) *Fetch First Set*, iv) *Send & Wait*. After the system is reset, the unit starts acquiring header data (words) from the SRAM and sending them to the DU. In the next state, the initial values are conveyed to the accumulators in a proper order. In the following state, the first set of words from amplitude-, length-, and sign fields are fetched from the SRAM and are sent to the DU to initiate the decoding process promptly. In *Send & Wait* state, the words from each field are sent to the DU. This state is only initiated when the incoming signals `signdata_need` and `ampdata_need` are set. It should be noted that there is no signal indicating the necessity for a data point from the length field. When a word from the amplitude field is needed, the corresponding word from the length field is sent automatically to the DU. This state lasts until all the commands are generated.

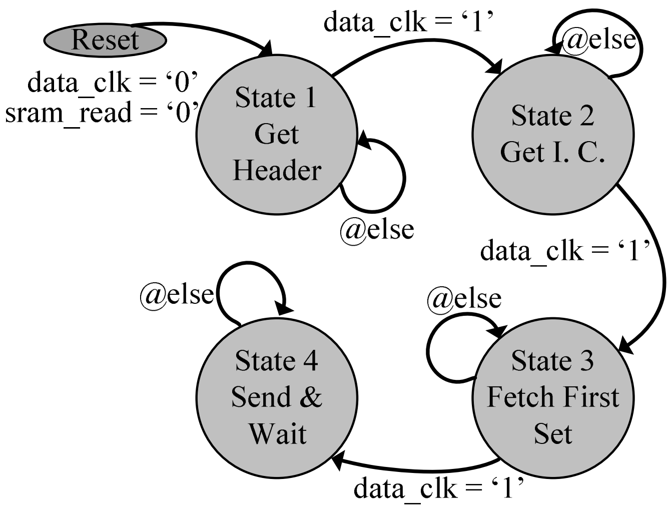


Figure 5-14 State Diagram of Memory Management Unit

5.5.1.3 Decoding Unit

DU is the module where the decoding algorithm is implemented. It communicates with the MMU, the first accumulator module, and the integrator module. The schematic design of this module is provided in Figure 5-15. All the input signals to this unit except the `clk` and `pause` signals are fed from the MMU. Two of the output signals are the acknowledgement signals (`signdata_need` and `ampdata_need`) which are described in the previous sub-section. The remaining two output signals are directly connected to the first accumulator. Thus, the decoded command is transferred to the accumulator in signed integer format at an additional clock indicating that a new command is being submitted.

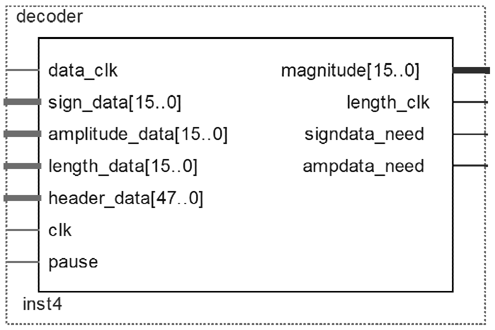


Figure 5-15 Decoding Unit

The basic operating principles of the DU are depicted in Figure 5-16. Decoding that constitutes nine states starts when the header data from the MMU are acquired. Then, the header data (constituting the order of difference, length of the command sequence, and the number of amplitude field words) are divided and stored for further use. In the second state, the first set of words from three different fields is saved. Second set received from the MMU is

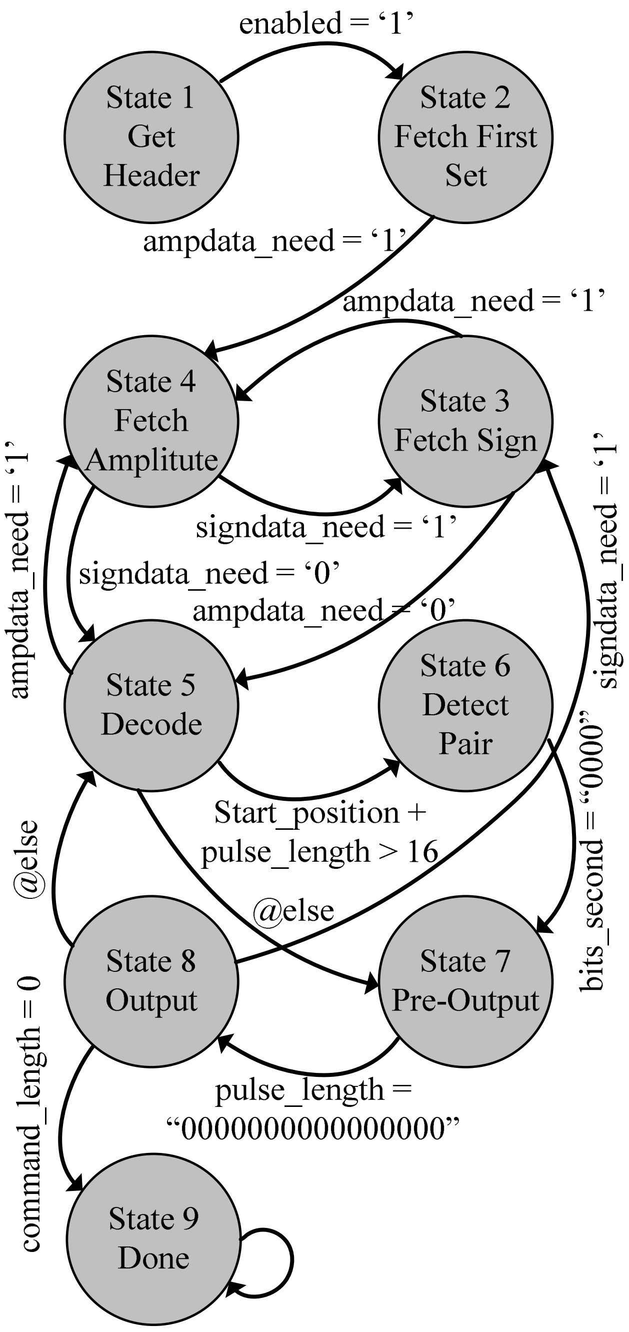


Figure 5-16 State Diagram of Memory Unit

stored in the third and fourth states. While decoding, the second set is necessary since it may turn out that the corresponding command is distributed between two consecutive words. The main task of this unit is executed in the *Decode* state of which is associated with five other states. When there is a lack of data during decoding, the finite-state "decoding" machine moves either on to *Fetch Amplitude* or *Fetch Sign* states to obtain the required data. If the decoding is complete for a given command, the data (in unsigned integer format) are processed in the *Pre-Output* state. In case the corresponding command is stored in two different words, *Detect Pair* state takes over for proper decomposition. Note that in *Pre-Output* state, the decoded command is rolled into a single word and passed onto the *Output* state. The conversion of unsigned to signed integer format is performed in the *Output* state. For this purpose, the data from the sign field must be ready. When sign data run out, the DU moves onto the third state and gets the necessary data. After the decoded command is formed as signed integer, it is sent to the first accumulator instance.

5.5.1.4 Accumulators

Accumulator (integrator) modules (Figure 5-17) are the simplest elements of this design. It gets the input data, sums it with the previous value of the accumulator and outputs the resulting value to the next accumulator. The number of accumulators in the design depends on the order of difference. Note that the given design in Figure 5-9 is hardwired and can decompress data differentiated up to the third order. However, the general design should have 15 accumulator instances (in compliance with the format specified in Section 5.5.1.1). A de-multiplexer unit must be incorporated to the design to deselect the unused accumulator instances. Notice that in the proposed design, the three accumulators yield the acceleration, velocity, and position profiles of the commanded trajectory. This attribute is one of the advantages of the proposed method. Since when a state-space controller is embedded into the system, the velocity and acceleration profiles must be ready for use.

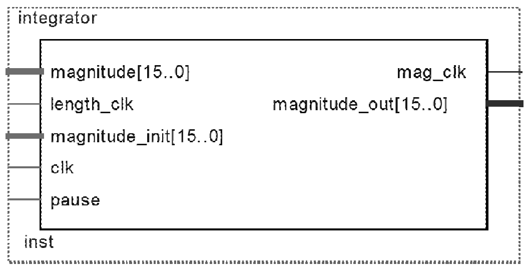


Figure 5-17 Accumulator Module

Instead of using a series of accumulators in the design, only one integrator can be used alternatively, whose schematic design is shown in Figure 5-18. In this design all the initial values are fed to this module at the beginning of decoding process. Integration formula for the third order difference is employed within the module and its result is sent to the interpolator module.

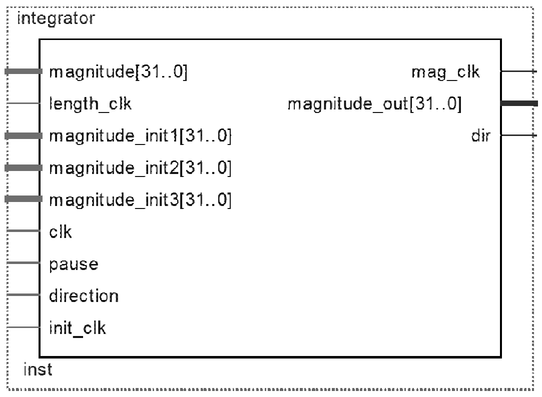


Figure 5-18 Integration Module

5.5.1.5 Interpolator

The interpolator used in the design, Figure 5-19, simply performs the computations described with the equations given in Section 4.4. While generating the commands, it sends `pause` signals to the DU, MMU, and accumulators to stop their operations. When there is need for a new original command, it sets the `pause` signal to low. Internal inputs to this module are data and its clock coming from the last accumulator module, and the global clock used in the system. External inputs are the ones given by the user and these are the feed-rate of generation and its direction. In order to overcome the delays between the generated commands, it also employs a buffer inside.

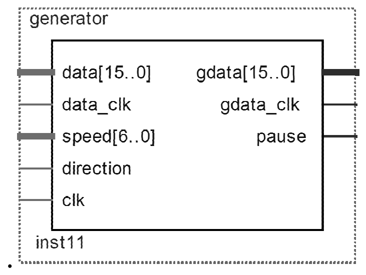


Figure 5-19 Interpolator Module

5.5.1.6 RS-232 Controller

The RS-232 Controller used in this architecture is the same with the one used in the implementation of the method based on Chebyshev polynomials approximations in the previous chapter.

5.5.2 Embedded Softcore Processor Approach

In the second approach of the FPGA implementations, the command generation algorithm is written in C programming language and the resulting are cross-compiled to run on a softcore processor deployed on the FPGA as done in the previous chapter. Then the resulted code is downloaded to the designed processor. Schematic design of this *algorithmic state machine* (ASM) is shown in Figure 5-20. The main difference of this design from the one designed in Chapter 4 is that there is a parallel input-output port in the softcore processor. With this property, a variable feed-rate input can be supplied to the system externally. As can be seen from Table 5-6, the hardware resources of this architecture is twice those of the hardwired architecture. It is critical to note that a sequential ASM is essentially implemented in this approach; there will not be a considerable increase in the resources when other trajectories are also generated. The memory required on the SDRAM will increase. The floor plan of the synthesized logic circuitry is illustrated in Figure 5-21. With the help of the performance counter module of the softcore processor, the time needed for decoding a sequence of 586 data points is roughly 25 ms.

Table 5-6 FPGA Resources used in Softcore Approach

Total Logic Elements	3549 (19%)
Total Combinational Functions	3146 (17%)
Dedicated Logic Registers	1984 (11%)
Total Registers	2036
Total Pins	46 (15%)
Total Memory Bits	28992 (12%)
Embedded Multipliers 9-bit Elements	0
Total PLLs	1 (25%)

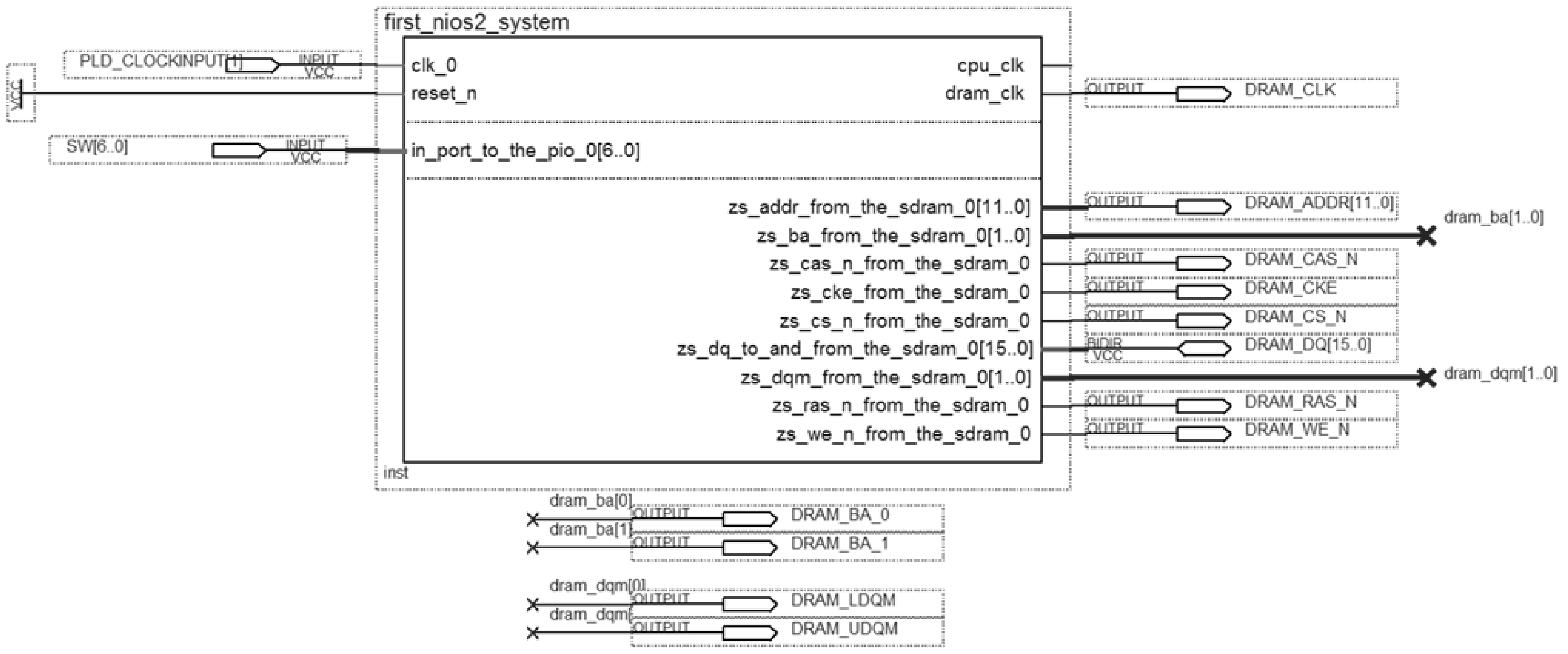


Figure 5-20 Implementation of the Method using Softcore Processor IP

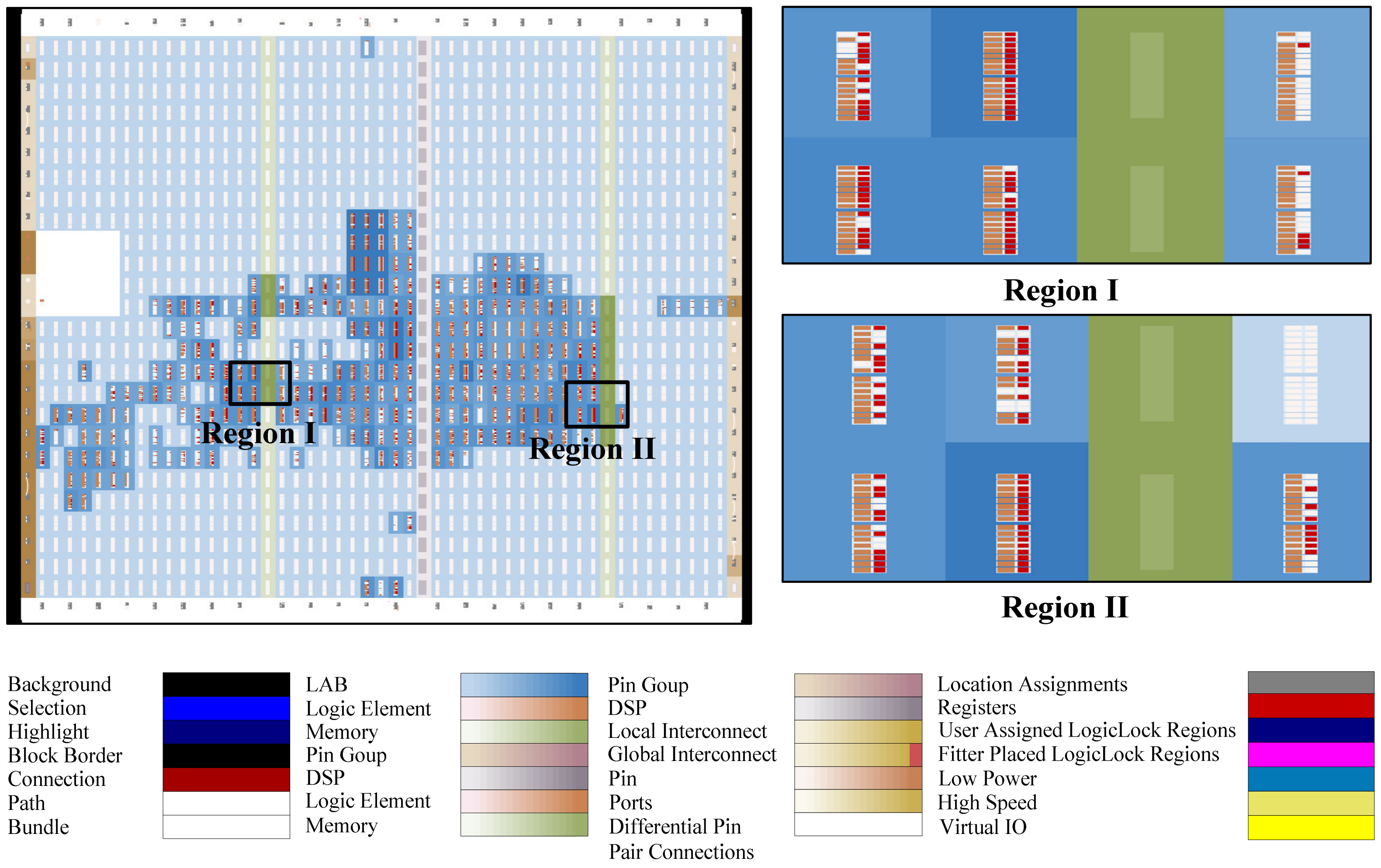


Figure 5-21 Floor Plan of the Synthesized Digital Circuitry for the 2nd Architecture

5.6 Closure

In this chapter, command generation method (a.k.a. ΔY) based on differencing and data compression techniques is proposed and implemented on the FPGA development board using two different approaches. During the performance evaluations, it is turned out that there is no sense in compressing the encoder pulses without taking higher order differences of them and proposed data compression technique is always more successful than the Huffman and Arithmetic coding methods. Taking higher order differences of the trajectory before compression is necessary. After differencing, the frequency of numbers in the sequence increases, (entropy based) data compression makes sense. Note that the novel compression method suggested in paper is not a universal. Its advantages reveal when the command sequence consists of integers showing acceleration and deceleration characteristics.

The hardwired FPGA implementation of the method outperforms the softcore embedded processor approach. Time need to generate same amount of commands for the softcore is about 100 times greater than the time needed by the hardwired architecture.

When the command generation method proposed in this chapter compared with the one proposed in the previous chapter, the former one is the most suitable one. Since in this case, less hardware resources are used and time necessary to generate commands is much lower than the method with polynomial approximation.

CHAPTER 6

CASE STUDY ON COMMAND GENERATION

6.1 Introduction

Up to this chapter, two different command generation methods are proposed and realized utilizing an FPGA development board (Altera DE1 with Cyclone II FPGA): **i)** Command Generation via Segmentation and Polynomial Approximation, **ii)** Command Generation via Differencing and Data Compression. When these two methods are compared, it can be concluded that the command generation method based on differencing and compression circumvents the other technique in terms of speed, resource utilization, compression ratio, and ease of implementation. Thus, during the case study, the performance of differencing and compression based method will be investigated through a detailed case study.

The chapter is organized as follows: the command sequences for a three-axis CNC vertical machining center are introduced. These sequences represent the desired cutting tool position when machining the injection mold of a bottle. After that, the trajectories are compressed with three different compression algorithms (Huffman, Arithmetic Coding, ΔY) after taking higher order differences. Once the compressed commands are generated, the FPGA implementation is carried out after some modifications on previously described schematic design. The results of the methods are compared and discussed at the final section of this chapter.

6.2 Sample Command Trajectory

To test the efficiency of the command generation methods on a realistic case, the manufacturing of a plastic injection mold for a bottle is taken into account. Despite the fact that the mold consists of two complementary parts (male/female), only the machining of the female (or negative) mold is considered in this work.

Before generating the commands for this specific machining task, the NC code, which is given in the Appendix D, are obtained from the sample codes of CncSimulator software [46]. It is critical to notice that an NC program is an industry-standard means of defining the trajectory of a programmable (CNC) machine system. Most CAD/CAM packages (after post-processing for a particular machine tool) do generate the NC code directly to carry out a particular machining task. From the functional point of view, the program describes the trajectory in terms of continuous linear- and circular segments using special (G) functions (such as G0, G1, G2/G2). That is, the code includes only the relevant parameters to define the trajectories in a piecewise fashion. For instance, to define a linear patch, only the destination (end-point) coordinates of this path needs to be specified along with the speed on the trajectory. Similarly, the destination/target coordinates as well as the radius of the curve may be sufficient to define an arc on a specific plane. On the other hand, the reference commands (position, velocity, acceleration) are to be supplied to the motion controller at each sampling period. Hence, the NC code must be processed (or interpolated) to generate the intermediate position data of the tool at equidistant time interval.

Utilizing the MATLAB script (`trajectory_generation.m`) developed by Akıncı [1], the command trajectories for the three axes {x, y, z} of the CNC machining centre are formed. These trajectories are shown in Figure 6-1 as a 3D plot. The trajectories along fundamental axes are illustrated in Figure 6-2, Figure 6-3, and Figure 6-4. Despite the fact that a NC code (by design) guarantees the C^0 continuity of the path, one needs to take into consideration not only the physical limitations of the power generating systems (electrical motors, drivers) but also the requirements of the tasks. The velocity and acceleration/deceleration profiles along the given trajectory must be modified to account for these machine (or task) related issues. Note that, while generating the trajectory data, no attempt is made to maintain the C^1 continuity of the resulting trajectory for this test case. The velocity profiles can be seen in Figure 6-5, Figure 6-6, and Figure 6-7. Hence, some sharp changes in the velocity profile might be observed. As can be seen from the figures, the simulated manufacturing process lasts for 926 seconds. Since the sampling time is selected as 1 ms, 926000 commands are generated for each axis. During the generation of motor commands, it is assumed that one revolution of an axis-motor corresponds to 10 mm of translation along a particular axis.

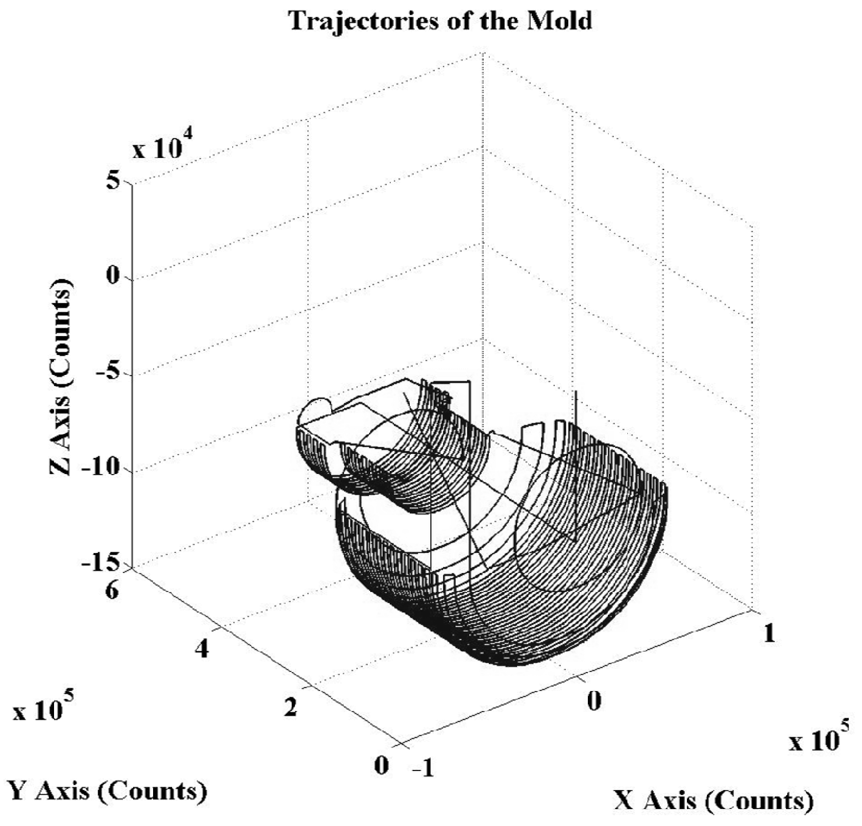


Figure 6-1 Trajectories of the Mold

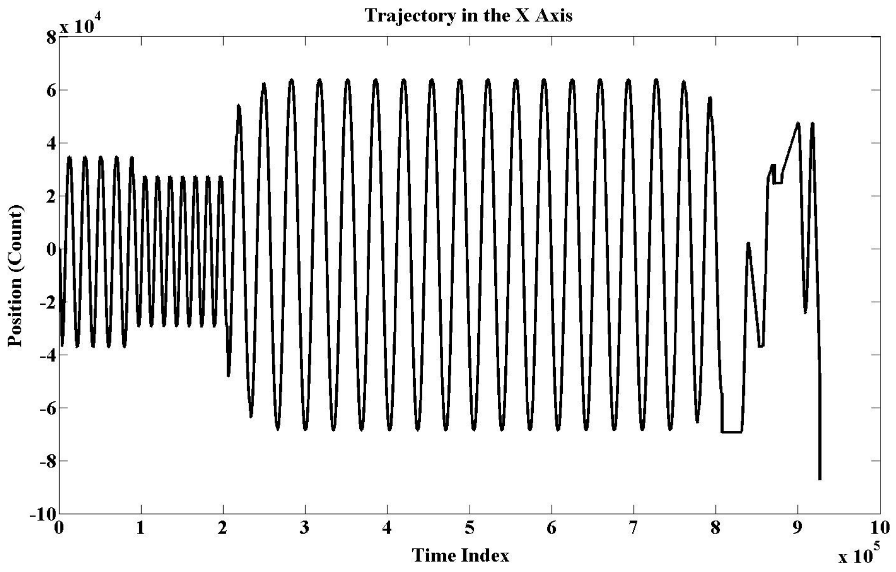


Figure 6-2 Trajectory in the X Axis

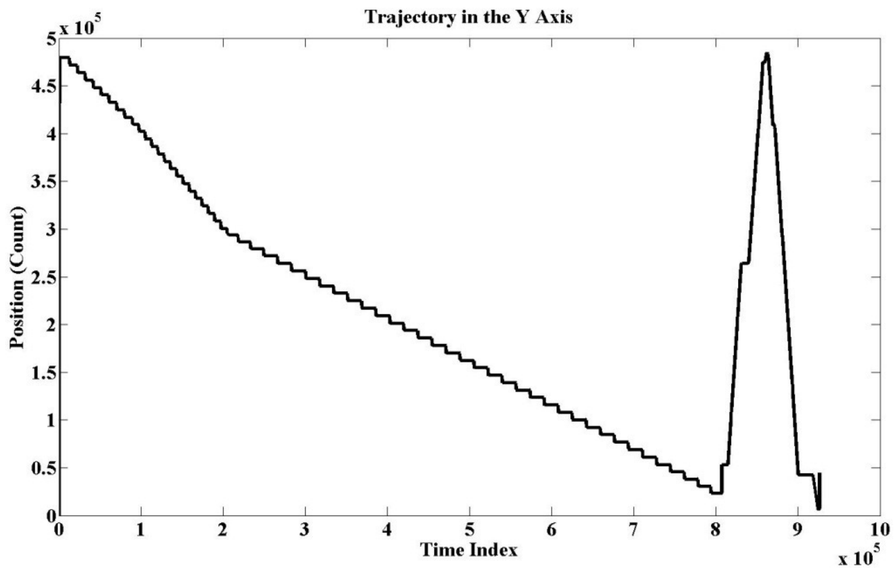


Figure 6-3 Trajectory in the Y Axis

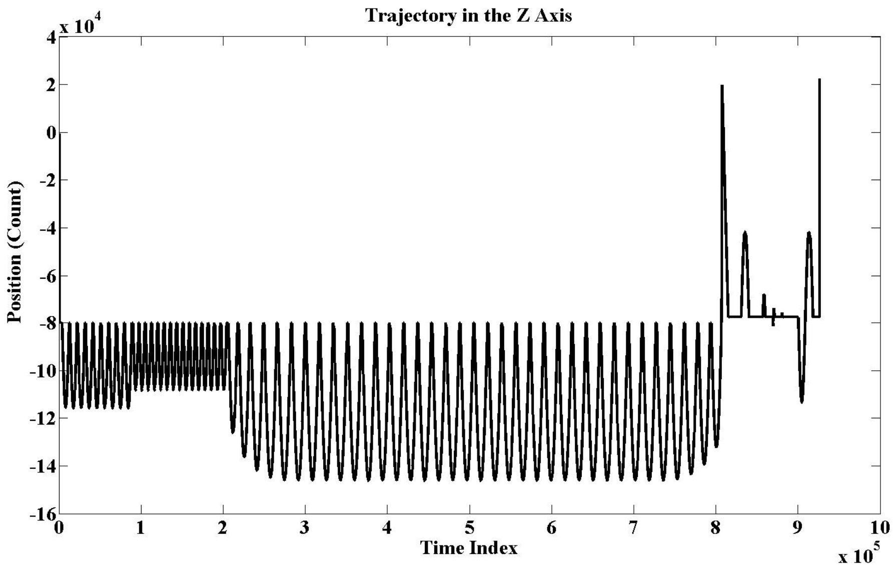


Figure 6-4 Trajectory in the Z Axis

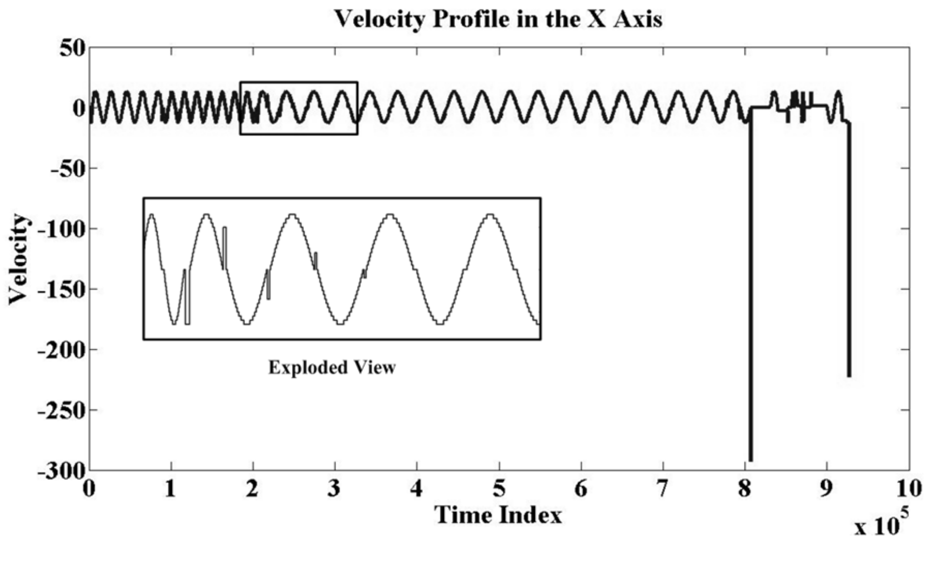


Figure 6-5 Velocity Profile in the X Axis

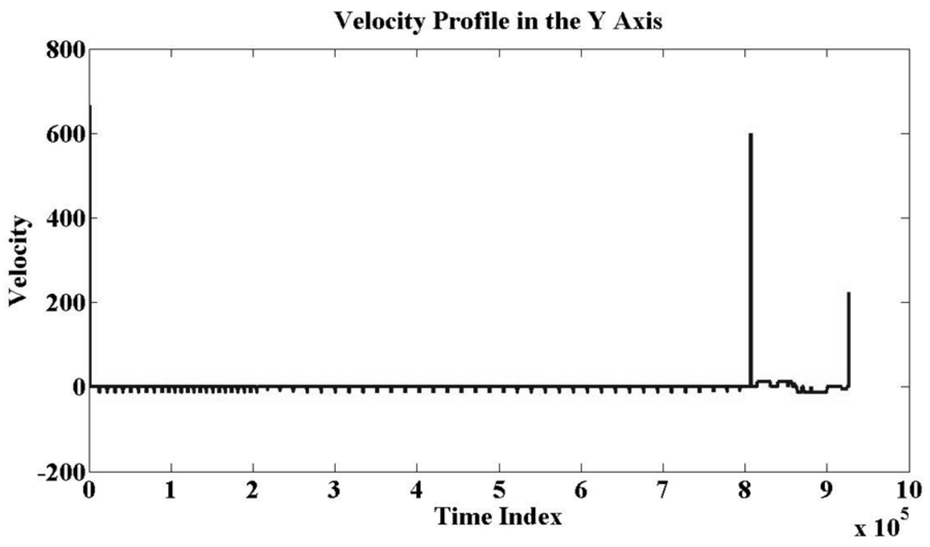


Figure 6-6 Velocity Profile in the Y Axis

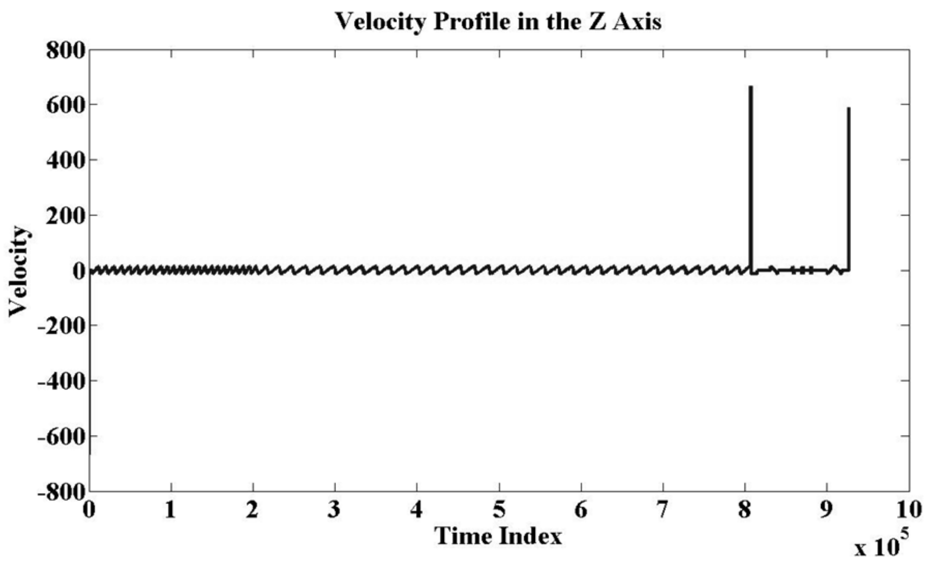


Figure 6-7 Velocity Profile in the Z Axis

6.3 Evaluation of Methods

In the scope of the thesis, two different command generation algorithms are introduced and implemented on the FPGA. To compare the methods (in terms of utilization of resources, compression of command data, and the generation time), the first trajectory in Figure 4-10 is reconsidered and the results are presented in Table 6-1. It can be inferred from the table that the first method uses nine times more FPGA resources than the second one. As stated in previous chapters, the reason of high consumption of the resources is the FPOMs used in the design – especially for multiplication of coefficients with polynomials. When the compression ratios are considered, the first method is also worse than the second approach. The first method cannot compress data as much; since for a reasonable approximation error, the number of polynomials as well as their corresponding coefficients should be kept high. This also causes the implemented design to generate the same amount of commands in longer time duration.

Table 6-1 Implementation Comparison of Proposed Command Generation Methods

	FPGA Resources [%]	**Compression Ratios [%]**	**Duration [ms]**
Segmentation and Approximation [I]	45	39.76	0.75
Differencing and Compression [II]	9	23.56	0.29

The first method generates 586 commands in 750 μs whereas the second method completes the generation process in 290 μs. Furthermore, there occur no representation errors in the commands for the second approach.

It is proven that the second method outperforms the first method in different aspects. Therefore, the trajectories presented in the previous article are encoded according to the second method. Before realizing the decoding on the FPGA, the trajectories are encoded with slight variations on the second method in MATLAB. First, the method is evaluated with only differencing. No further compression is employed on the differenced data. Table 6-2 represents the resulting compression ratios for differences up to sixth-order for all three axes of the given trajectory. The best performances are achieved for the first-order of difference regardless of the selected axis. These results are not totally in agreement with the data presented in Table 5-1 and Figure 5-1. Thus, a careful study on order of difference should be carried out when evaluating the complete method with various compression algorithms.

Table 6-2 Compression Ratios [%] vs Order of Differences for the Test Case

	Order of Difference					
	1	**2**	**3**	**4**	**5**	**6**
X Axis	50.00	55.56	55.56	61.11	66.67	72.22
Y Axis	50.00	55.00	55.00	60.00	65.00	70.00
Z Axis	61.11	61.11	61.11	66.67	72.22	77.78

In order to decide on the order of difference of the method for FPGA implementation, a further study on the trajectories is carried out. The Huffman, Arithmetic Coding, and the proposed compression (ΔY) algorithms are applied on the trajectories of the three axes for an order up to the sixth. The results of compression algorithms for various orders are provided in Table 6-3, Table 6-4, and Table 6-5. Surprisingly, the arithmetic coding seems to be far superior to other compression algorithms regardless of the order and the trajectory.

Table 6-3 Results of Huffman Compression Algorithm for Various Orders of Difference [%]

Axis	**Order of Difference**					
	1	**2**	**3**	**4**	**5**	**6**
X	25.62	5.58	5.59	5.64	5.56	5.72
Y	6.29	5.01	5.01	5.02	5.02	5.03
Z	25.04	5.57	5.60	5.62	5.67	5.70

Table 6-4 Results of Arithmetic Coding Algorithm for Various Orders of Difference [%]

Axis	**Order of Difference**					
	1	**2**	**3**	**4**	**5**	**6**
X	25.36	0.13	0.23	0.36	0.46	0.59
Y	3.96	0.02	0.03	0.05	0.06	0.08
Z	24.72	0.13	0.25	0.36	0.49	0.08

Table 6-5 Results of the ΔY Compression Algorithm for Various Orders of Difference [%]

Axis	**Order of Difference**					
	1	**2**	**3**	**4**	**5**	**6**
X	32.83	10.53	10.53	10.55	10.58	10.66
Y	13.69	10.01	10.01	10.02	10.03	10.04
Z	32.21	10.53	10.54	10.57	10.60	10.69

This is mainly due to the high sampling frequency (1 kHz). As the sampling frequency decreases, the compression ratios of the algorithms approach to each other. The reason behind this fact is that the number of different values increases. Another conclusion to be drawn from the tables is that after the second order of difference, there are not any remarkable changes in the compression ratios.

Since compressing the original data to one-tenth of its original size seems to be adequate, the ΔY compression algorithm is selected for the implementation on the FPGA. Ease of implementation of the algorithm has a strong effect on this selection.

6.4 FPGA Implementation

In the previous section of the chapter, it is concluded that the second or third-order of differences before employing the proposed compression algorithm on the trajectories are the optimum orders for the implementation on the FPGA. In Chapter 5 implementation of the ΔY compression algorithm based command generation method is carried out via two different approaches, namely softcore (processor) and hardwired approaches. When the results of the techniques are considered, it can be concluded that the hardwired approach is faster and expends less resources than the softcore approach.

Before realizing the generator for all axes, a decoder for the trajectory of x-axis is designed in Quartus II 9.0 (Web Edition) in order to compare the utilization of resources with the design given in Chapter 5. In this new design shown in Figure 6-8, several modifications on the modules are done. First of all, the widths of the registers in the modules are increased from 16 to 32 bits since it is not possible to represent the magnitudes of the commands with 16 bits. This conversion of registers affects the resource utilization in the FPGA. When the results given in Table 6-6 and Table 5-4 are compared, it can be inferred that with the doubled register sizes, the new design consumes more than twice the resources of the prior

Table 6-6 FPGA Resources used in Hardwired Approach of the Case Study for the First Axis

Total Logic Elements	4475 (24%)
Total Combinational Functions	3997 (21%)
Dedicated Logic Registers	1549 (8%)
Total Registers	1549
Total Pins	99 (31%)
Total Memory Bits	0
Embedded Multipliers 9-bit Elements	4 (8%)
Total PLLs	0

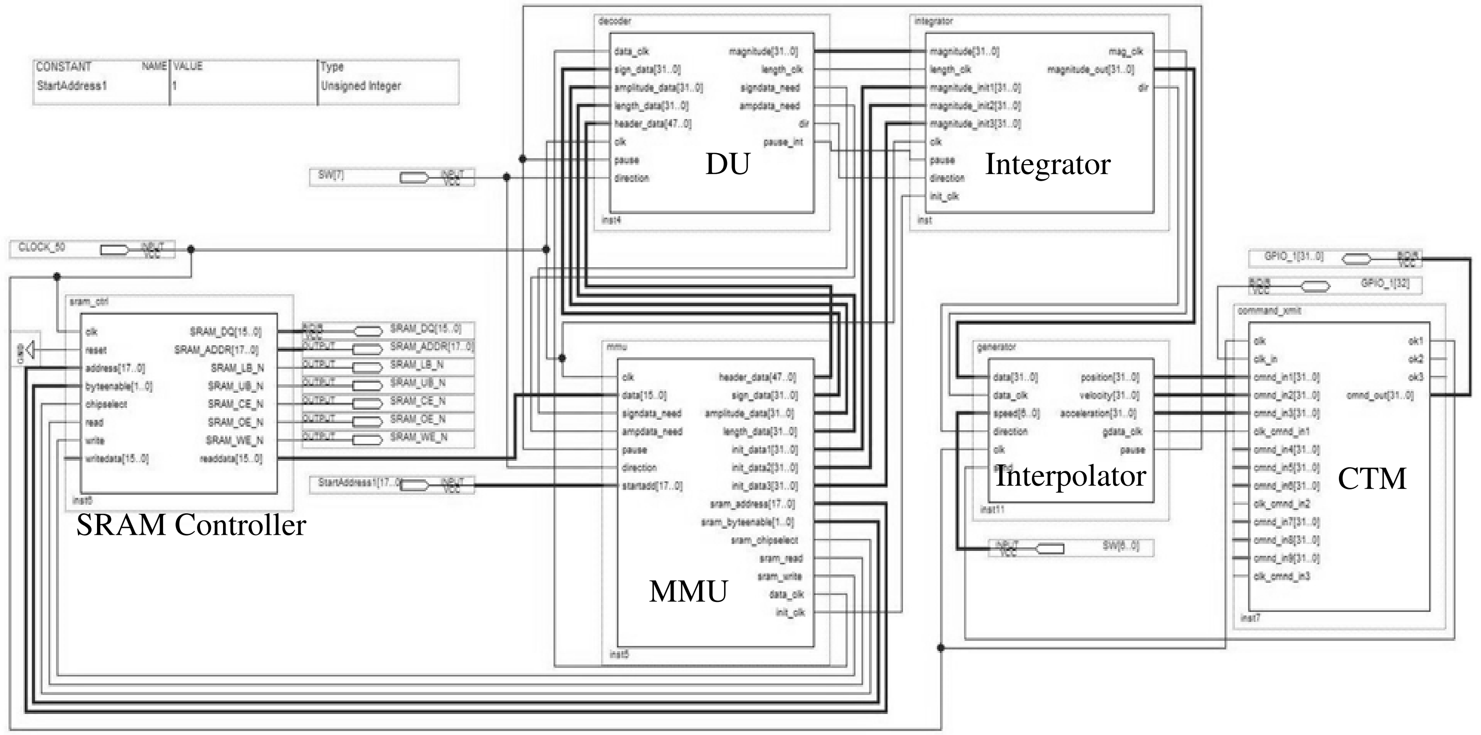


Figure 6-8 Hardwired FPGA Implementation of Command Generator for the X-Axis

design. Another modification in the design with respect to the one in Figure 5-9 is that instead of accumulator modules, an Integrator Module (IM) is embedded to the design. The main difference in this module (which shown in Figure 6-9) is that all of the initial values are directly transferred to this module at the beginning of the decoding process. Position, velocity, and acceleration profiles are fed with a data clock to the Command Transmit Module (CTM) shown in Figure 6-10 according to the acknowlegment signal `ok1` coming from the CTM connected to the `pause` input. The transferred commands are also depended on the current direction of decoding.

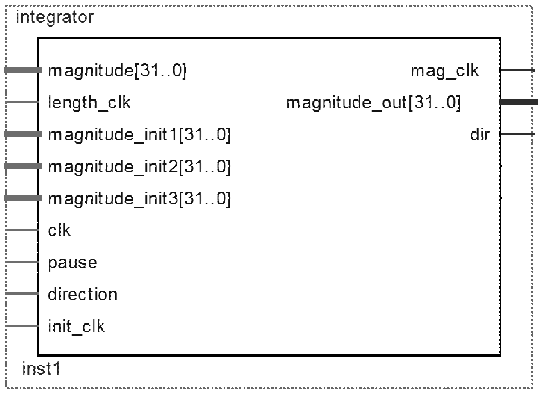


Figure 6-9 Integrator Module

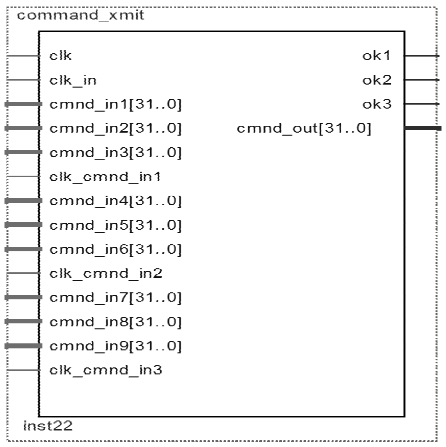


Figure 6-10 Command Transmit Module

Note that this module accepts all the commands from the three axes and sends these signals according to the incoming signal from the controller unit, `clk_in`. The incoming signal shown in Figure 6-11 is comprised of 9 consecutive clocks (at 10 MHz). These pulse sets are periodically generated at the sampling frequency of 1 kHz. At the rising edge of these clocks, the module sends commands to the controller and at the falling edge, the controller receives them.

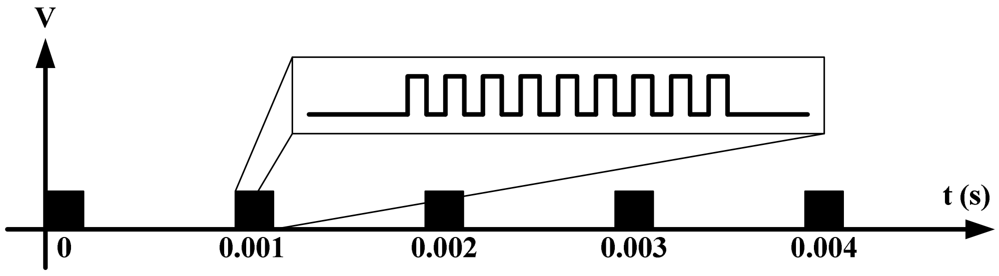


Figure 6-11 Incoming Clock Signal from the Controller

In the full design of the command generator for the case study, there should three separate command generators as illustrated in Figure 6-12. Note that there are three SRAM controllers in this design. However, the additional SRAM controllers do not increase the resource usage significantly since they are connected to the same pins of the FPGA chip. Another issue that should be noted is that the CTM is also responsible for prevents the conflicts that may arise during the communication of SRAM with the MMUs. The output of the CTM is connected to the General Purpose Input Output (GPIO) pins of the FPGA development board. Any controller can be connected to these GPIO pins to receive reference commands for the trajectories. The hardware resources utilized in this design is given in Table 6-7. When the table is compared in terms of the resources used for only x-axis, it can easily be inferred that the complete design almost utilizes three times more resources than the previous one as expected. It is critical to notice that it was stated in Chapter 5 that all six trajectories of the manipulator can be generated using the FPGA development board. However, after changing the size of the registers to 32 bits, it is now possible to generate only four state trajectories.

Table 6-7 FPGA Resources Used in Hardwired Approach of the Case Study for all the Axes

Total Logic Elements	13366(71%)
Total Combinational Functions	12062 (64%)
Dedicated Logic Registers	4568 (24%)
Total Registers	4568
Total Pins	231 (73%)
Total Memory Bits	0
Embedded Multipliers 9-bit Elements	12 (23%)
Total PLLs	0

Finally, the chip floor plan of the synthesized circuit design is given in Figure 6-13. It can be seen that most of the resources are occupied as shown in Table 6-7. For a better illustration of the chip plan, two regions are zoomed. When the colors in the legend are considered, it can be concluded that the most of the chip is reserved for logic elements, connection of elements, and registers.

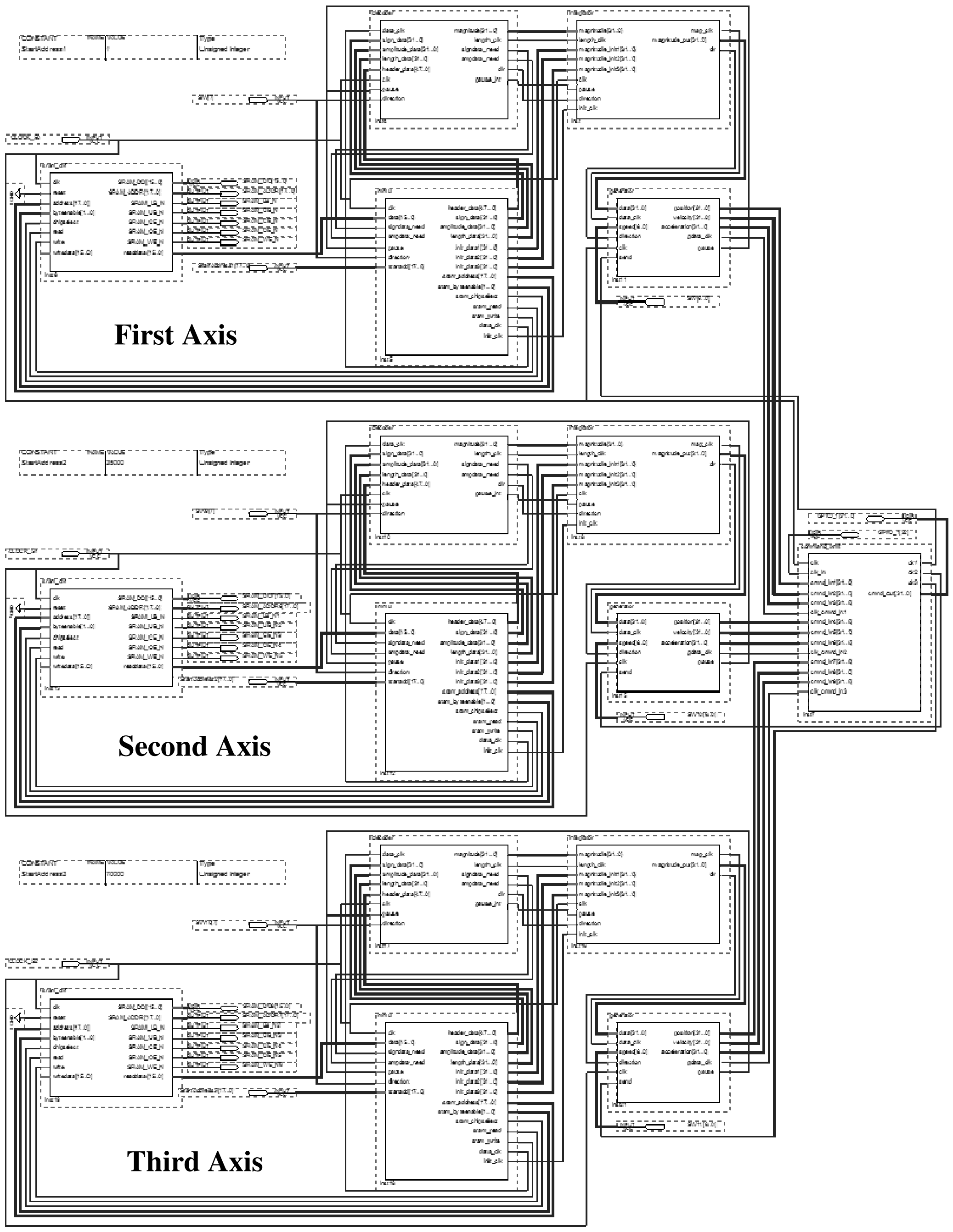


Figure 6-12 Hardwired FPGA Implementation of the Command Generator

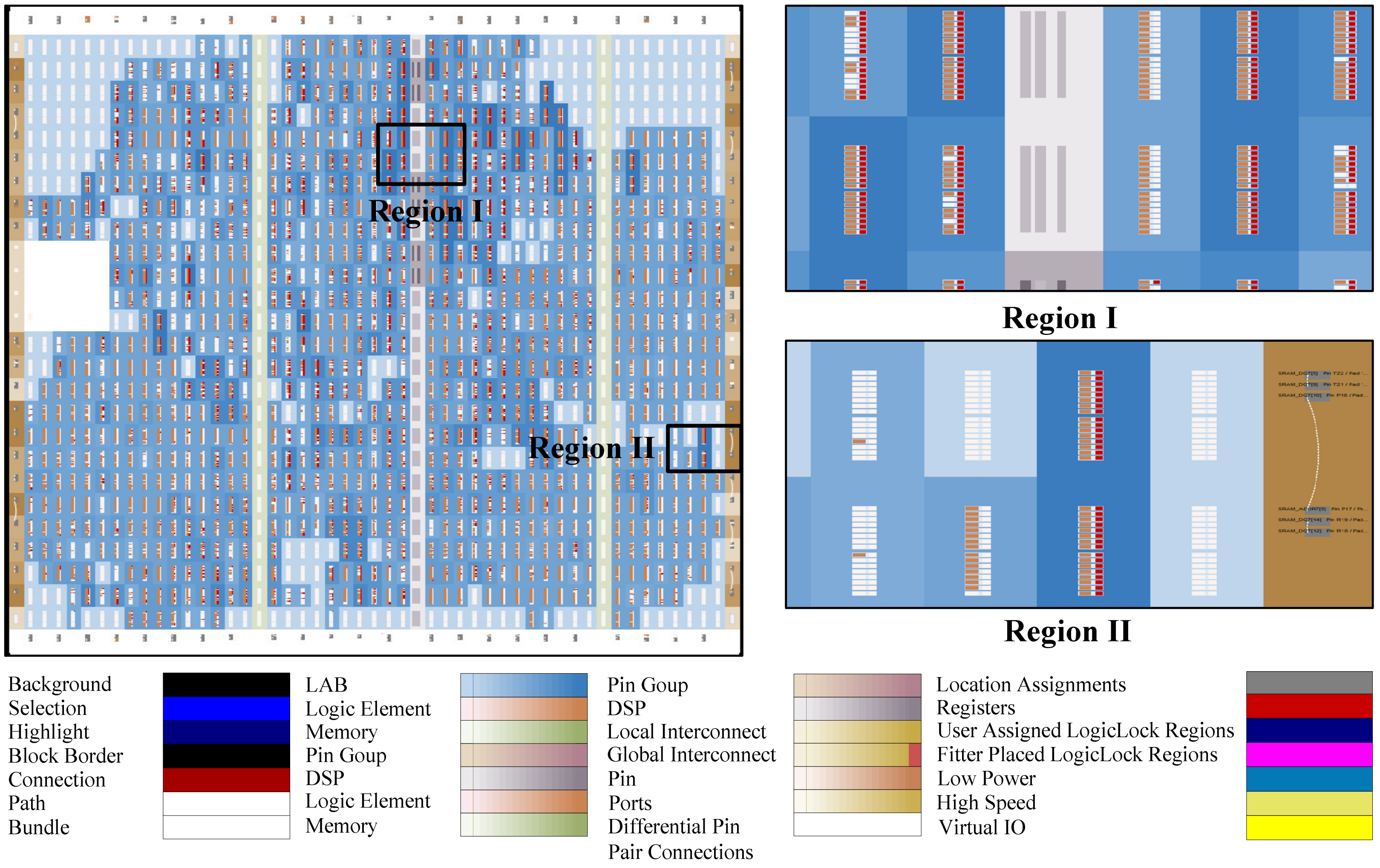


Figure 6-13 The Chip Floor Plan of the Synthesized Circuit Design

6.5 Closure

In this chapter, the most successful command generation method in the thesis, which consists of differencing and the ΔY data compression algorithm, is employed to generate the trajectories of a CNC vertical machining center while machining a plastic injection mold of a bottle. The selected method for FPGA implementation is further investigated in MATLAB and the results are discussed.

During the performance comparison of data compression algorithms, it is turned out that the Arithmetic coding algorithm outperforms its counterparts. The Arithmetic coding is not selected since the algorithm is not very suitable for parallel decoding. On the other hand, the ΔY is specifically developed for parallel decoding and its implementation on the FPGA is advantageous if compared to other compression methods.

The complete design built for the generation of all trajectories of the mold used 71% of the resources on the FPGA chip. If the number of axis increases in the system, the FPGA chip (Cyclone II) should be upgraded to (an advanced one like Cyclone III) generate all command trajectories.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this study, two different advanced command generators utilizing FPGA for computer controlled mechanisms have been developed. In both of the proposed command generators after the encoding of the trajectories are completed, the size of the original data is compressed to at least one-fourth of it.

The FPGA interface developed in the scope of the thesis can be regarded as the most important unit of the command generator system, since without the interface the system cannot be connected to the controller unit and the PC performing encoding operations. Beside providing communication with various devices, the interface is also used to convert digital signals to analog signals and vice versa. During these conversions, the voltage ranges are scaled and shifted according to the ranges set by the user.

Once the FPGA interface is introduced in the third chapter, the first command generation method is elaborated in the following chapter. This first method is basically an polynomial approximation algorithm. Since pure approximation does not give good results (compression ratios) for the complex trajectories. The method is modified such that the polynomial approximation is employed after segmenting the trajectory according to its inflection points and the representation errors are stored after compressing them via proposed compression algorithm (ΔY). Then after the FPGA implementations, it is turned out that the hardwired approach uses more resources than the embedded softcore processor approach, but the command generation time of the hardwired approach is much better than the embedded softcore counterpart. Thus, the hardwired approach can be preferable for the systems where a high profile FPGA chip (Cyclone III, Stratix IV, etc.) is implemented.

The second FPGA based command generation method is composed of two parts: differencing and compression. During the encoding process, the trajectories are not directly compressed. This is due to the fact that the data values on the trajectory are completely different and compressing them is meaningless. Thus, taking higher order differences of the trajectories are inevitable if remarkable compression ratios are required to be achieved. The optimum order of difference is found to be three for almost all trajectories elaborated. For the second part of the corresponding command generation method, Huffman, Arithmetic Coding, and the ΔY compression algorithms are employed and evaluated. The FPGA implementations are carried out for the ΔY method, since it is much easier to implement and its performance is better than the other compression algorithms. The hardwired implementation of this command generation method outperforms the embedded softcore processor approach in utilization of resources and command generation time.

Comparing the proposed command generation methods in the thesis, it has been observed that differencing and the ΔY compression algorithm based method has presented the best compression ratio and the FPGA resource utilization. Hence, the case study is carried out with this method. In the case study, the manufacturing of a plastic injection mold for a bottle is taken into account. Although the mold has two complementary parts, only the female part is considered. For a detailed elaboration, the three compression algorithms are employed within the method to the trajectories of manufacturing process. It is turned out that the Arithmetic coding algorithm is much better than the other two algorithms, in contrast to the results obtained in the previous chapter of the thesis. Despite the superior performance of Arithmetic coding, in FPGA implementations of the case study the ΔY compression algorithm is applied. The reason of this selection is that the implementation of ΔY method is much easier and its compression ratios are acceptable when the available resources on the FPGA board are considered. The ΔY method is also suitable for parallel decoding which increases the speed of command generation.

To summarize, the proposed FPGA based command generation system is faster than its counterparts and can be implemented to various control systems along with the interface developed in the thesis.

7.2 Future Work

In addition to the scope of the thesis there exist still some contributions that can be made on the topic. These can be classified as the improvements on the FPGA interface, encoded data transfer to the FPGA development board, and implementation of the Arithmetic coding algorithm on the FPGA.

For the FPGA interface part; instead of jumpers and switches used in daughter cards, fast analog switches that can transfer current in two ways are planned to be used. Furthermore, the configuration of channels (the selection of cards) may be completely done electronically via analog switches, and multiplexers. By utilizing the multilayer circuit printing technology and surface mountable electronical devices the final version of the interface can be designed and manufactured.

Another topic on which contributions can be made is the encoded data transfer to the decoder embedded on the FPGA development board. In the scope of the thesis, this topic is not elaborated much. For the data transfer, serial port of the computer is used. Design of a robust data transfer protocol is inevitable when the limited memory resources of the FPGA board are considered. During the command generation, the designed software should overwrite the old data according to the current status of generation. If the direction of command generation is suddenly changed, then the designed software should restore the previous data.

The last further contribution can be done by elaborating the Arithmetic coding with various trajectories and sampling times, since in the case study it is turned out that the Arithmetic coding can compress the trajectories of the plastic injection mold for a bottle up to one-thousandth of their original size. In order to make a strong decision on the validity of the performance of Arithmetic coding, various trajectories should be studied. Implementation of the algorithm may also be a big contribution, since there exists no full decoder implemented on an FPGA chip for the Arithmetic coding algorithm.

REFERENCES

[1] Akıncı, A. "Universal Command Generator for Robotics and CNC Machinery," Middle East Technical University Graduate School of Natural and Applied Sciences, Master of Science Thesis, 2009.

[2] Yaman, U., Mutlu, B. R., Dolen, M., Koku, A. B., "Direct command generation for electrical servo motor drives," Proceedings of the 12th International Conference on Electrical Machines and Systems, IEEJ Industry Applications Society, Tokyo, pp. 1-6, November 2009.

[3] Sayood, K. "Introduction to Data Compression," Elsevier Inc, 2006.

[4] Reghbati, H. K. "Special Feature An Overview of Data Compression Techniques," Computer, vol. 14, no. 4, pp. 71-75, 1981.

[5] Balch, T., Khan Z., Veloso, M., "Automatically Tracking and Analyzing the Behavior of Live Insect Colonies," Proceedings of AGENTS'01, Montreal, pp. 521-528, 2001.

[6] Stearns, S. D. "Arithmetic Coding in Lossless Waveform Compression," IEEE Transactions on Information Theory, IT – 21, pp. 228-230, 1975.

[7] Dickson, K. "Cisco IOS Data Compression," San Jose, CA, USA, 2000.

[8] Rigler, S., Bishop, W., Kennings, A. "FPGA-Based lossless data compression using Huffman and LZ77 algorithms," Canadian Conference on Electrical and Computer Engineering, pp. 1235-1238, 2007.

[9] Huffman, D. "A Method for the Construction of Minimum-Redundancy Codes," Proceedings of the Institute of Radio Engineers, vol. 40, no. 9, pp. 1098-1101, September 1952.

[10] Rissanen, J. J., Langdon G. G., "Arithmetic Coding," IBM J. Res. Develop. vol. 23, pp. 149-162, 1979.

[11] Witten, I. H., Neal, R. M., Cleary, J. G., "Arithmetic coding for data compression," Communications of the ACM, vol. 30, pp. 520-540, 1987.

[12] S. W. Golomb, S. W., "Run Length Encodings," IEEE Transactions on Information Theory, vol. 12, pp. 399-401, 1966.

[13] Ostermann, J., Bormans, J, List, P., Marpe, D., Narroschke, M., Pereirra, F., Stockhammer, T., and Wedi, T., “Video Coding with H.264/AVC: Tools, Performance, and Complexity,” IEEE Circuits and System Magazine, vol. 4, no. 1, pp. 7-28, 2004.

[14] Rigler, S., Bishop, W., and Kennings, A., “FPGA-Based lossless data compression using Huffman and LZ77 algorithms,” Canadian Conference on Electrical and Computer Engineering, pp. 1235-1238, 2007.

[15] De Araujo, T. M. U., Pinto, E. R., De Lima, J. A. G., and Batista, L. V., “An FPGA implementation of a microprogrammable controller to perform lossless data compression based on the Huffman algorithm,” 13th IBERCHIP Workshop, 2007.

[16] Abd El ghany, M. A., Salama, A. E., and Khalil, A. H., “Design and implementation of FPGA-based systolic array for LZ data compression,” IEEE International Symposium on Circuits and Systems, pp.3691-3695, 2007.

[17] Cui, W., “New LZW data compression algorithm and its FPGA implementation,” Picture Coding Symposium 2007, Lisbon (Portugal), Nov. 2007.

[18] H’ng, G. H., Salleh, M. F. M., and Halim, Z. A., “Golomb coding implementation in FPGA,” Elektrika Journal of Electrical Engineering, pp. 36-40, 2008.

[19] Yongming, Y., Jungang, L., and Jianmin, W., “LADT arithmetic improved and hardware implemented for FPGA - Based ECG data compression,” Proceedings of 2nd IEEE Conference on Industrial Electronics and Applications, pp.2230-2234, 2007.

[20] Valencia, D., and Plaza, A., “FPGA-Based hyperspectral data compression using spectral unmixing and the pixel purity index algorithm,” Computational Science, pp.881- 891, 2006.

[21] Lee, D., Luk, W., Villasenor, J., and Cheung, P., “Hierarchical Segmentation Schemes for Function Evaluation,” Proceedings of IEEE International Conference on Field-Programmable Technology, pp. 92-99, 2003.

[22] Lee, D., Cheung, C. C., Luk, W., and Villasenor, J. D., “Hardware Implementation Trade-Offs of Polynomial Approximations and Interpolations,” IEEE Transactions on Computers, vol.57, no.5, pp.686-701, May 2008.

[23] Michard, R., Tisserand, A., and Veyrat-Charvillon, N., “Small FPGA Polynomial Approximations with 3-bit Coefficients and Low-Precision

Estimations of the Powers of X," Proc. 16th IEEE Int. Conf. On Application-Specific Systems,Architecture and Processors, pp.334-339, 2005.

[24] Ashrafi, A., Adhami, R., Joiner, L., and Kaveh, P., "Arbitrary Waveform DDFS utilizing Chebyshev Polynomials Interpolation," IEEE Transactions on Circuits and Systems I: Regular Paper, vol. 51, no. 8, pp. 1468-1475, 2004.

[25] Sodagar, A. M., and Lahiji, G. R., "A Pipeline ROM-Less Architecture for Sine-Output Direct Digital Frequency Synthesizers Using the Second-Order Parabolic Approximation," IEEE Transactions on Circuits and Systems. II, vol. 48, no. 9, pp. 850-857, 2001.

[26] Ashrafi, A., Pan, Z., Adhami, R., and Wells, B. E., "A Novel ROM-less Direct Digital Frequency Synthesizer Based on Chebyshev Polynomial Interpolation," Proc. of the 36th Symposium on System Theory, pp. 393-397, 2004.

[27] Su, K. H., Hu, C. K., and Cheng, M. Y., "Design and Implementation of an FPGA-based Motion Command Generation Chip," Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, 2006.

[28] Jeon, J. W., and Kim, Y. K., "FPGA based acceleration and deceleration circuit for industrial robots and CNC machine tools," *Mechatronics*, vol. 12, pp. 635-642, 2002.

[29] Jeon, J. W., "An efficient acceleration for fast motion of industrial robots," Proceedings of IEEE 21st IECON, pp. 1336–41, 1995.

[30] Cheng, C. W., Tsai, M. C., and Maciejowski, J., "Feed-rate control for non-uniform rational B-spline motion command generation," Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, vol. 220, pp. 1855-1861, 2006.

[31] Cheng, C. W., Tsai, M. C., and Cheng, M. Y., "Real-time variable feed-rate parametric interpolator for CNC machining," 15th IFAC World Congress, Barcelona, Spain, 2002.

[32] Xu, H. Y., Tam, H. Y., Zhou, Z., and Tse, P. W., "Variable feed-rate CNC interpolation for planar implicit curves," Advanced Manufacturing Technology, vol. 18, pp. 794 - 800, 2001.

[33] Cody, W. J., "A survey of practical rational and polynomial approximation of functions," SIAM Rev., vol. 12, no. 3, pp. 400–423, 1970.

[34] Vanicek P., "Approximate Spectral Analysis by Least-squares Fit," Astrophysics and Space Science, pp.387–391, Volume 4, 1969.

[35] Boyd, J. P., "Chebyshev and Fourier Spectral Methods," Dover Publications, Inc., 2000.

[36] Altera DE1 FPGA Development and Education Board User Manual, Altera Co., v1.1, 2006.

[37] NIOS II Processor Reference Handbook, Altera Co., 2009.

[38] U. Yaman, M. Dolen, and A. B. Koku, "A Novel Command Generation Method with Variable Feedrate utilizing FGPA for Motor Drives," Proceedings of 8th IEEE Workshop on Intelligent Solutions in Embedded Systems, Crete, Greece, 8-9 July 2010.

[39] R. Usselmann, "Open Floating Point Unit Manual," http://www.opencores.org, last accessed date: 06/06/2010.

[40] R. Usselmann, "Simple Asynchronous Serial Comm. Device," http://www.opencores.org/cores/sasc/, last accessed date: 06/06/2010.

[41] Menon, S. M., Bobra, Y. K., Ghia, A. V., and Zaliznyak, "Programmable Input/Output Circuit for FPGA for use in TTL, GTL, GTLP, LVPECL, and LVDS Circuits," US-Patent 6218858, 2001.

[42] Goetting, F. E., Frake, S. O., Kondapalli, V. M., and Young, S. P., "FPGA with a Plurality of I/O Voltage Levels," US-Patent 5877632, 1999.

[43] Chang, W. "Application Specific Field Programmable Gate Array," US-Patent 5687325, 1997.

[44] Proteus PCB Design Software, http://www.labcenter.co.uk/index.cfm, last accessed date: 07/06/2010.

[45] Driscoll, F. F., Coughlin, R. F., "Operational Amplifiers and Linear Integrated Circuits," 5th Edition, Prentice Hall College Div, 1997.

[46] CncSimulator, http://www.cncsimulator.com, last accessed date: 13/06/2010.

APPENDIX A

LIST OF VERILOG / VHDL MODULES

In this appendix, the Verilog / VHDL modules used in the study are listed. Brief descriptions of each module along with the related sections of the thesis are also presented.

Table A-1 List of Verilog / VHDL Modules Utilized in the Thesis

Name	Mentioned in	Description
`clk_div`	Chapter 3.4	Modifies the global clock used in the design according to the external input. If this module is paused, then the circuit also stops operating.
`pwmgen`	Chapter 3.4	Generates PWM signals according to the input vector.
`sine_wave`	Chapter 3.4	Generates sine waves by reading the values of quarter sine wave from the look-up table. The frequency of the sine wave is set by changing the global clock of the system.

`sine_package`	Chapter 3.4	It is look-up table holding the values of quarter sine wave according to the pre-defined resolution.
`Driver`	Chapter 4.4	Coordinates the modules used in the approximation architecture design with each other for proper operation.
`Split_32_8`	Chapter 4.4	Splits the 32-bit input to four bytes and sends these bytes to the serial communication module.
`LookUpReader`	Chapter 4.4	Reads the value of Chebyshev coefficients from the previously formed look-up table.
`chebyshev_coef`	Chapter 4.4	Holds the values of Chebyshev coefficients in binary form.
`fpu_cadmusmod`	Chapter 4.4	Performs mathematical floating point operations [38].
`sasc_brg`	Chapter 4.4	It is the baud-rate generator used for serial communication [40].
`sasc_top`	Chapter 4.4	It is the main controller of the serial communication [40].
`sram_ctrl`	Chapter 5.5	It is the SRAM Controller of Altera DE 1 Development Board [36].
`decoder`	Chapter 5.5	Decodes the encoded commands according to the ΔY compression algorithm and transmits the decoded commands to the first accumulator.

`mmu`	Chapter 5.5	This is the Memory Management Unit of ΔY decompression architecture. It receives the necessary data from the SRAM Controller and passes to other modules in the architecture.
`accumulator`	Chapter 5.5	Sums the incoming value with the previous one and sends to the corresponding module.
`interpolator`	Chapter 5.5	Interpolates between the two consecutive command values according to the feed-rate input.
`integrator`	Chapter 6.4	Includes the accumulators according to the order of difference in encoding.
`command_xmit`	Chapter 6.4	Receives the commands for position, velocity, and acceleration for all the axes and transmits these commands to the control unit of the system.

APPENDIX B

NIOS II EDS 9.0 C CODES

In this appendix, the C files used in NIOS II Integrated Development Environment are presented in Table B-1. The parts of the thesis they are used are also mentioned. In Table B-2, NIOS II C file for the ΔY decompression algorithm is presented.

Table B-1 List of NIOS II C Files Utilized in the Thesis

Name	Mentioned in	Description
`DYdecompression.c`	Chapter 5.5.2	Algorithmic state machine of the ΔY decompression algorithm
`ChebyshevApp.c`	Chapter 4.4.2	Algorithmic state machine of the Chebyshev approximation algorithm

Table B-2 NIOS II C file for the ΔY Decompression Algorithm

```c
#include <stdio.h>
#define uint8 unsigned char
#define uint16 unsigned short
#define uint32 unsigned long
#define int8 char
#define int16 short
#define int32 long
#define SECTION1 1
int main(void)
{
    const int twos[16] = { 1, 2, 4, 8, ..., 16384, 32768};
    /*Field Declarations */
    uint8 sign[74] = {143, 128, 0, ..., 166, 1};
    uint8 amp[98] = {232, 67, 147, ..., 237, 0};
    uint8 term[98] = {240, 123, 28, ..., 238, 0};
    uint8 sign_ = 0;
    uint8 amp_ = 0;
    uint8 term_ = 0;
    uint32 original[586];
    float original_we[586];
    int32 l1 = 586; /*Length of the Sign Field */
    int32 l2 = 777; /*Length of Amplitude and Termination Fields*/
    int32 count = 0; /*Counts the length of the amplitude value.*/
    int32 i = 0;    /*For Loop Counter */
    int32 i1 = 0;
    int32 j = 0;   /*For Loop Counter */
    int32 k = 0;
    int32 k1 = 0;
    int32 r = 0;  /*Remainder*/
    int32 r1 = 0;
    int32 l = 0;  /*Original Data Counter*/
    int32 a = 0;  /*Original Data Counter*/
```

```
for(i=0; i<586; i++)
{
    original[i]=0;
}
i = l2 / 8;
r = l2 % 8;
term[i] = term[i] << (8-r);
amp[i] = amp[i] << (8-r);
i = l1 / 8;
r = l1 % 8;
sign[i] = sign[i] << (8-r);


while (k < l2)
{
    i = k / 8;
    r = k % 8;
    k1 = k;
    i1 = i;
    r1 = r;

    if (k == (l2-1))
    {
        if ((((term[i]<<r) & 128)==128))
        {
            term[i]=term[i] & (254<<(7-(r+1)));
        }
        if ((((term[i]<<r) & 128)==0))
        {
            term[i]=term[i] | (1<<(7-(r+1)));
        }
    }

    term_ = term[i] << r;

    if ((term_ & 128) == 128)
    {
            count ++;
            if (((r != 7) & (((term_ <<1) & 128) == 0)) |
((r==7)&((term[i+1] & 128) == 0)))
            {
                for (j=0; j<count; j++)
                {
                    amp_ = amp[i1] << r1;
                    if ((amp_ &  128) == 128)
                    {
                        original[l] += twos[j];
                    }
                    k1--;
                    i1 = k1 / 8;
                    r1 = k1 % 8;
                }
                count = 0;
                l++;
            }
        }
        else
        {
            count ++;
            if (((r != 7) & (((term_ <<1) & 128) == 128)) |
((r==7)&((term[i+1] & 128) == 128)))
            {
                for (j=0; j<count; j++)
                {
                    amp_ = amp[i1] << r1;
                    if ((amp_ &  128) == 128)
                    {
                        original[l] += twos[j];
                    }
                    k1--;
                    i1 = k1 / 8;
                    r1 = k1 % 8;
                }
                count = 0;
                l++;
            }
        }
        k++;
    }

    for (k=0; k<l1; k++)
    {
        i = k / 8;
        r = k % 8;

        sign_ = sign[i] << r;

        if ((sign_ & 128) == 0)
        {
            original[k]= original[k] ;
        }
        else
        {
            original[k]= - original[k] ;
        }
    }
    return 0;
}
```

APPENDIX C

MATLAB M-FILES

In this appendix, the MATLAB scripts (m-files) used in the study are explained briefly in Table C-1. Related sections of these files in the thesis are also noted. In Table C-2, MATLAB M-file for the ΔY compression algorithm is presented.

Table C-1 List of MATLAB M-Files Utilized in the Thesis

Name	Mentioned in	Description
`above_err.m`	Chapter 4	Counts the number of data values above the acceptable error margin.
`acc.m`	Chapter 5	Accumulates the sequence according to the initial values supplied to the function.
`bernstein.m`	Chapter 4	Generates Bernstein polynomials with various orders and lengths.

`bp_approx.m`	Chapter 4	Performs Bernstein approximation on the given trajectory.
`bp_approx_errcomp.m`	Chapter 4	First performs Bernstein approximation and then compresses the approximation errors.
`chebypol.m`	Chapter 4	Generates Chebyshev polynomials with various orders and lengths.
`compviadiff.m`	Chapter 5	Compares the compression ratios of differencing the input sequence for various orders.
`cp_approx.m`	Chapter 4	Performs Chebyshev approximation on the given trajectory.
`cp_approx_errcomp.m`	Chapter 4	First performs Chebyshev approximation and then compresses the approximation errors.
`dacomp.m`	Chapter 5	Performs Arithmetic coding algorithm on the given sequence for the specified order of difference and outputs the compressed code and memory requirements.

`dhcomp.m`	Chapter 5	Performs Huffman compression algorithm on the given sequence for the specified order of difference and outputs the compressed code and memory requirements.
`dycomp.m`	Chapter 5	Performs ΔY compression algorithm on the given sequence for the specified order of difference and outputs the compressed.
`dydcomp.m`	Chapter 5	Decompresses the encoded data with ΔY compression algorithm and outputs the original data sequence.
`dydcompreverse.m`	Chapter 5	Decompresses the encoded data with ΔY compression algorithm in reverse order and outputs the original data sequence.
`gen_enc_pulses`	Chapter 6	Generates sequences of encoder pulses from the tool location data sequences.
`legendre.m`	Chapter 4	Generates Legendre polynomials with various orders and lengths.

`lookupgen.m`	Chapter 4 & 5	Creates look-up file for Chebyshev coefficients or encoded data.
`lp_approx.m`	Chapter 4	Performs Legendre approximation on the given trajectory.
`lp_approx_errcomp.m`	Chapter 4	First performs Legendre approximation and then compresses the approximation errors.
`to16.m`	Chapter 5 & 6	Converts 8-bit data to 16-bit data for SRAM compatibility.
`trajectory_generation.m`	Chapter 5 & 6	Generates command trajectories from the NC code according to the sampling time.
`vsint.m`	Chapter 5 & 6	Performs variable speed interpolation on the given trajectory.

Table C-2 M-file for the ΔY Compression Algorithm

```
%
% This function compresses a given time sequence
% using ΔY compression technique.
% ($ REV 1.4, UY & MD, FEB-2010 $)
%
% Input arguments:
%    q - time sequence (integer)
%    n - order of differences {1,2,3, ...}
%
% Output argument:
%   cdat - compressed data structure with following fields:
%     amp: amplitude (bytes)
%     len: length (bytes)
%     sgn: sign (bytes)
%      ic: initial conditions
%       n: order of difference
%       m: length of original sequence
%
function cdat = dycomp(q,n)
  if (nargin==1), n = 3; end
  m = length(q); q = q(:); y = diff(q,n);
%
% Calculate ICs
%
  ic = zeros(n,1,'int32'); ic(1) = q(1); q = q(1:n+1);
  if(n>1)
    for i = 2:n
      t = diff(q,i-1); ic(i) = t(1);
    end
  end
%
% Sign field
%
```

```
  ns = 8*ceil((m-n)/8); s = [(y>=0); zeros(ns-m+n,1)];
  sf = zeros(ns/8,1,'uint8');
  for k = 1:ns/8
    i = 1 + 8*(k-1); j = i + 7;
    sf(k) = bin2dec(num2str(s(i:j)'));
  end
%
% Amplitude- and length fields
%
  as = []; ts = as; toggle = true; y = abs(y);
  for k = 1:(m-n)
    str = dec2bin(y(k)); L = length(str);
    if (toggle)
      ts = [ts num2str(ones(1,L),'%d')];
    else
      ts = [ts num2str(zeros(1,L),'%d')];
    end
    toggle = not(toggle); as = [as str];
  end
%
% Now, some padding...
%
  na = ceil(length(as)/8); L = 8*na-length(as);
  as = [as num2str(zeros(1,L),'%d')];
  if (toggle)
    ts = [ts num2str(ones(1,L),'%d')];
  else
    ts = [ts num2str(zeros(1,L),'%d')];
  end

  tf = zeros(na,1,'uint8'); af = zeros(na,1,'uint8');
  for k = 1:na
    i = 1 + 8*(k-1); j = i + 7;
    af(k) = bin2dec(as(i:j)); tf(k) = bin2dec(ts(i:j));
  end
  cdat = struct('amp',af,'len',tf,'sgn',sf,'ic',ic,'n',n,'m',m);
end
```

APPENDIX D

NC CODE OF PLASTIC INJECTION MOLD FOR A BOTTLE – CASE STUDY

```
G00 Z-20.01
G00 Y120
G00 X-9.449 Y120.5 T4 (dia 4mm)
G01 Z-20 F200 S150
G18 G03 X9.449 Z-20 I9.449 K0
G01 X9.5 Y120
G01 Y118.5
G18 G02 X-9.5 I-9.5 K0
G01 Y116.5
G18 G03 X9.5 Z-20 I9.5 K0
G01 Y114.5
G18 G02 X-9.5 I-9.5 K0
G01 Y112.5
G18 G03 X9.5 Z-20 I9.5 K0
G01 Y110.5
G18 G02 X-9.5 I-9.5 K0
G01 Y108.5
G18 G03 X9.5 Z-20 I9.5 K0
G01 Y106.5
G18 G02 X-9.5 I-9.5 K0
G01 Y104.5
G18 G03 X9.5 Z-20 I9.5 K0
G01 Y103.5
G01 X7.5 Y102.5
G18 G02 X-7.5 I-7.5 K0
G01 Y100.5
G18 G03 X7.5 Z-20 I7.5 K0
G01 Y98.5
G18 G02 X-7.5 I-7.5 K0
G01 Y96.5
G18 G03 X7.5 Z-20 I7.5 K0
G01 Y94.5
G18 G02 X-7.5 I-7.5 K0
G01 Y92.5
G18 G03 X7.5 Z-20 I7.5 K0
G01 Y90.5
G18 G02 X-7.5 I-7.5 K0
G01 Y88.5
G18 G03 X7.5 Z-20 I7.5 K0
G01 Y86.5
G18 G02 X-7.5 I-7.5 K0
G01 Y84.5
G18 G03 X7.5 Z-20 I7.5 K0
G01 Y82.5
G18 G02 X-7.5 I-7.5 K0
G01 Y80.5
G18 G03 X7.5 Z-20 I7.5 K0
G01 Y78.5
G18 G02 X-7.5 I-7.5 K0
G01 Y76.5
G18 G03 X7.5 Z-20 I7.5 K0
G01 Y74.5
G18 G02 X-7.5 I-7.5 K0
G01 Y73.487
G01 X-12.243 Y72.5
G18 G03 X12.243 Z-20 I12.243 K0
G01 X14.928 Y70.5
G18 G02 X-14.928 I-14.928 K0
G01 X-16.367 Y68.5
G18 G03 X16.367 Z-20 I16.367 K0
G01 X17.165 Y66.5
G18 G02 X-17.165 I-17.165 K0
G01 X-17.487 Y64.5
G18 G03 X17.487 Z-20 I17.487 K0
G01 X17.5 Y64
G01 Y62.5
G18 G02 X-17.5 I-17.5 K0
G01 Y60.5
```

```
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y58.5
G18 G02 X-17.5 I-17.5 K0
G01 Y56.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y54.5
G18 G02 X-17.5 I-17.5 K0
G01 Y52.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y50.5
G18 G02 X-17.5 I-17.5 K0
G01 Y48.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y46.5
G18 G02 X-17.5 I-17.5 K0
G01 Y44.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y42.5
G18 G02 X-17.5 I-17.5 K0
G01 Y40.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y38.5
G18 G02 X-17.5 I-17.5 K0
G01 Y36.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y34.5
G18 G02 X-17.5 I-17.5 K0
G01 Y32.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y30.5
G18 G02 X-17.5 I-17.5 K0
G01 Y28.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y26.5
G18 G02 X-17.5 I-17.5 K0
G01 Y24.5
G18 G03 X17.5 Z-20 I17.5 K0
G01 Y22.5
G18 G02 X-17.5 I-17.5 K0
G01 Y20.5
G18 G03 X17.5 I17.5 K0
G01 Y18.5
G18 G02 X-17.5 I-17.5 K0
G01 Y16.5
G18 G03 X17.5 I17.5 K0
G01 Y14.5
G18 G02 X-17.5 I-17.5 K0
G01 Y12.5
G18 G03 X17.5 I17.5 K0
G01 Y10.5
G18 G02 X-17.5 I-17.5 K0
G01 Y10
G01 X-17.381 Y8.5
G18 G03 X17.381 I17.381 K0
G01 X16.832 Y6.5
G18 G02 X-16.832 I-16.832 K0
G01 X-15.746 Y4.5
G18 G03 X15.746 I15.746 K0
G01 X13.831 Y2.5
G18 G02 X-13.831 I-13.831 K0
G00 Z5
G00 X-17.5 Y10
G01 Z-20
G01 Y64
G18 G02 X-7.5 Y73.487 I9.5 J0
G01 Y102.5
G01 X-8.5
G18 G02 X-9.5 Y103.5 I0 J1
G01 Y120
G18 G02 X-7 Y122.5 I2.5 J0
G01 X0
G01 X7
G18 G02 X9.5 Y120 I0 J-2.5
G01 Y103.5
G18 G02 X8.5 Y102.5 I-1 J0
G01 X7.5
G01 Y73.487
G18 G02 X17.5 Y64 I0.5 J-9.487
G01 Y10
G18 G02 X8 Y0.5 I-9.5 J0
G01 X0
G01 X-8
G18 G02 X-17.5 Y10 I0 J9.5
G00 Z5
M30
```

Implementations of State-Space Controllers using Field Programmable Gate Arrays

B. R. Mutlu[1,2], M. Dolen[1]

[1] Department of Mechanical Engineering, Middle East Technical University, Ankara, Turkey
[2] Department of Mechanical Engineering, Hacettepe University, Ankara, Turkey

Abstract — Field Programmable Gate Arrays (FPGAs) are suitable choices for demanding real-time control applications including CNC machine tools, robotics, advanced automation, aviation, automotive systems. Logic-level design capabilities of FPGAs allow engineers to develop efficient yet flexible methods for motion control applications. In fact, certain properties of FPGAs (parallelism, layout management, logic optimization, etc.) can be exploited to reduce the resources used on the FPGA without sacrificing the performance. This paper focuses on this aspect and presents novel implementation methods for motion controllers using FPGAs. The presented methods are applied to a full-state space controller utilizing a Luenberger-type state observer. This controller topology, which can be easily tailored to any application, is implemented on an Altera Cyclone II FPGA chip utilizing the methods elaborated in the paper. Furthermore, the control performances of the resulting systems are investigated through a hardware in the loop simulation (HILS) of a nonlinear system (inverted pendulum) using MATLAB.

***Index Terms* — State space, full-state feedback, observers, FPGA, controller implementation, motion control.**

I. Introduction

FPGAs have become more resourceful with recent developments in VLSI technology and exhibit some superior qualities over traditional processors (micro-processors, micro-controllers and DSPs) such as parallel processing capability, high sampling rates, flexibility in design, and reliability. Not surprisingly, FPGA chips appear more frequently in controller designs [1, 2].

Despite these favorable attributes, FPGAs require low-level (i.e. logic-level) circuit design through hardware description languages (HDL). Unfortunately, such a design effort using HDLs is a relatively long and tedious process. To overcome this difficulty, FPGA manufacturers offer many intellectual properties (IPs) including embedded processor IPs. However, using an embedded processor quickly consumes the resources of the chip and hinders the actual capacity of an FPGA. Therefore, in order to explore the capabilities of a flexible hardware, all the IPs included in the design should be customized (and optimized) for a specific task at hand.

In this study, different low-level design methods are investigated to realize a typical state-space controller. The proposed techniques are specifically designed to take advantage of the special characteristics of control systems (i.e. input/output attributes). To illustrate the feasibility of the proposed methods, a nonlinear system (inverted pendulum) driven by a servo-motor is considered and a state feedback controller along with a Luenberger-type observer is designed. FPGA implementation of the proposed methods is realized on an Altera Cyclone II FPGA chip. FPGA based controller is then tested on the inverted pendulum system using a hardware-in-the-loop simulation in MATLAB environment.

This work was supported by the Scientific and Technological Research Council of Turkey under the project contract 108E048.

This paper is organized as follows: First, the background work on controller implementation using FPGAs is presented. The following section reviews the state-space controllers as well as state observers. Next section discusses their (customized) hardware implementations on FPGA. The proceeding section presents the inverted pendulum system as the test setup. Next section provides the results of the hardware in the loop simulation of the FPGA based state space controller utilizing the proposed methods. The final section discusses the success and efficiency of the proposed methods and comments on feasibility of implementation of such a design on FPGAs.

II. Background

Since the last decade, FPGAs are becoming gradually dominant in control applications literature. In numerous studies, implementation methods for embedded controller design and application results of different topologies on FPGAs are presented. In this section, a general overview of the relevant technical literature will be given.

A. *Hybrid control systems*

In many previous studies, FPGAs are generally considered as an interface between a DSP and its peripheral units. In such configurations, FPGAs do not usually share the computational burden of the main processor. For instance, in the design proposed by Dong *et al*. [3], FPGA serves as an external circuitry processing position encoder signals while managing some I/O functions whereas all computations are handled by a DSP. A similar scheme appears in the study of Birou and Imecs [4] where the FPGA performs pulse width modulation and encoder interfacing. In the design proposed by S. Jung and S.S. Kim [5], a neural network controller is implemented on a DSP and FPGA is only used to perform rather simple calculations.

B. *Control systems using soft-core processors*

Latter research studies, however, tend to shift the computational load from DSP to FPGA as the capabilities of FPGAs significantly increase in time. A single FPGA is generally employed to implement the entire motion control system. Many designs facilitates a single FPGA that makes good use of an embedded processor IP developed by FPGA manufacturers. While some designs use them only to implement controllers, the others utilize the embedded processor for complex calculations along with the “hardwired” custom modules for simpler tasks. For instance, Ni *et al* [6] design a PID controller on a Cyclone FPGA with a Nios II processor to realize a highly integrated joint servo system. In another study, Li *et al* [7] develops an FPGA-based servo controller for PMSM drives while a sliding mode controller is implemented on a Nios II processor. Jung *et al* [8] adopt a hybrid fuzzy-PI controller to realize a motion control IC and they also used a Nios II embedded processor for the realization of the fuzzy controller. Using an embedded processor IP within the FPGA is proven to be successful in some aspects, but it reduces the flexibility of the controller and full performance of a configurable controller cannot be attained.

C. *Hardwired implementation of control systems*

While the current tendency is to use an embedded processor for complex calculations required by controllers, it is also possible to eliminate the embedded processor and develop a total hardware solution. In a very recent study, Cho *et al* [9] have proposed an FPGA-based multiple axis motion control chip with no embedded processor employed. The chip has all the essential features such as velocity profile generation, interpolation, inverse kinematics calculation and a PID controller which are required to control a multiple axis motion control system such as a robotic manipulator. They managed to avoid floating point calculations by multiplying certain values by constant integers. Using no embedded processor; they attained lower resource costs and power consumption rates.

Controller implementation on FPGA is often a trade-off between resource and execution time of the controllers. The reason for that is; while it is possible to benefit from the parallel processing capability of the FPGA chip by calling many instances of a certain module which would certainly require more resources, it is also possible to use certain modules repeatedly in a sequential manner to increase the time, rather than the resource cost. Chan *et al* [10] have conducted a study on PID controller implementation on an FPGA and they managed to decrease the resources required by a multiplier-based design significantly. What they propose is to replace the multipliers by a distributed arithmetic based design utilizing look up tables and they managed to decrease the resource requirement down to 4 to 13% of the former design. However they increased the computation time from 1 cycle to 13 to 26 cycles. A very similar study by Tao *et al* [11] managed to decrease the logic element requirement from 51.7% to 0.8-1.5%, increasing the computation cycle from 1 to 64-33 cycles. While these studies offer good improvements for a PID controller, there exist many controller algorithms including hybrid topologies and there is no single way to implement each of them more efficiently on an FPGA.

Fuzzy controller designed by Lanping *et al* [12] requires no embedded processor or a floating point unit to perform fuzzy control. On the other hand, the proposed method is similar to a rule based control topology and the method is not generally applicable. An Elman neural network implementation is proposed by Lin *et al* [13] for a linear ultrasonic motor, where a fixed point arithmetic unit is implemented to perform the calculations. A hybrid sliding mode controller is proposed by Li *et al*, where the sliding mode controller is implemented using an embedded processor, and PI regulator with hardware logic.

These studies prove that it is possible to implement different controller topologies on FPGAs utilizing arithmetic logic units or custom hardware solutions. It is also seen that PID controllers are easier to implement by custom hardware logic and in practice usually realized by hardware modules. On the other hand more complex topologies generally require fixed/floating number calculations and in practice they are either realized by an embedded processor or a fixed/floating point unit.

D. *Research Synthesis*

As outlined previously, there exist a large number of controller implementations on FPGAs. Some of these studies avoid low-level (logic circuit) design and rely on embedded processor IPs (a.k.a. soft-core processors) to realize a particular controller topology on FPGAs [14, 15]. In fact, handling all the necessary calculations for a particular application through the use of a soft-core processor (i.e. a sequential algorithmic machine) certainly gives rise to a conventional control system with slightly faster embedded processors. Consequently, much of the potential capabilities of FPGAs are wasted in the process for the sake of easing the design efforts. On the other hand, some research efforts provide hand-tailored solutions to specific applications through the use “hardwired” designs. The resulting solutions, which take full advantage of FPGAs, are often-times very efficient at the expense of (excruciatingly) slow development process. Furthermore, the application of the devised solutions to a general framework is proven to be difficult.

This general overview of the literature highlights the need for developing efficient solutions that make good use of the hardware capabilities of FPGAs. These solutions will hopefully become generalized IPs that could be easily modified to accommodate the requirements of any control application in industry. Hence, the major goal of this work is to focus on the implementation of the state-space controllers (including observers) and is to develop the building blocks (IPs) to implement such controllers on FPGAs with ease.

III. State-space Controller & Observer Design

State space controllers find their use in almost all industrial applications. The control law of a typical state-space controller can be simply expressed as

$$\boldsymbol{u} = -\mathbf{K}[\hat{\mathbf{x}}(k) - \mathbf{x}_{\mathbf{r}}(k)] \tag{1}$$

where $\mathbf{u}$ is the manipulated input vector; $\hat{\mathbf{x}}$ is the estimated state vector; $\mathbf{x}_{\mathbf{r}}$ is the reference state vector while $\mathbf{K}$ refers to the gain matrix. Once the system equations governing the dynamics of the plant are obtained, the gains in (1) are adjusted to yield desired control characteristics using modern control theory. Note that since all states of the plant are not measured for all practical purposes, a full-state observer often times needs to accompany the design in order to estimate the missing components of the state vector. For instance, a Luenberger-type state observer takes the following form:

$$\hat{\mathbf{x}}(k+1) = (\mathbf{F} - \mathbf{LH})\hat{\mathbf{x}}(k) + \mathbf{G}\boldsymbol{u}(k) + \mathbf{L}\boldsymbol{y}(k) \qquad (2)$$

where **F, G, H** correspond to the system matrices formed by the estimated parameters; **y**(k) is the controlled output vector while **L** denotes the gain matrix of the observer.

In industrial motion control, the states of the system are frequently selected as angular position (θ) and (instantaneous) angular velocity (ω). Thus, the general approach is to estimate the velocity using the measured position. There exist versatile estimator algorithms in the literature for velocity estimation in digital systems employing encoders/resolvers [16]. However, no single estimation algorithm exists to cover all applications where dynamic operating conditions do vary considerably such as the one considered in this study. Hence, when good estimates on system parameters are available, it is more desirable to observe the unavailable states, rather than to estimate them using higher-order differencing methods.

IV. Hardware Implementation

As mentioned earlier, many different controller algorithms (conventional/novel/intelligent) could be implemented on FPGAs with a combination of hardwired logic and a floating point arithmetic unit (FPU) without the use of an embedded processor in the design [17]. However, the FPUs are regarded as costly for FPGAs in terms the resources (i.e. the CLBs) expended to realize them. On the other hand, the digital control systems have unique properties that relax the usage of such units and thus there is a potential to develop inexpensive yet high-performance solutions. Some of the afore-mentioned attributes are as follows:

- Outputs of all sensors used in controls technology are essentially amplitude-quantized and can be conveniently represented as (signed/ unsigned) integers.
- Reference signals (i.e. the command vector), which are to be compatible with the sensory data, are to be generated as integer (number) sequences.
- Manipulated outputs of almost all control systems need to be amplitude-quantized while sending them out to the output interface.
- With proper scaling, the controller gains might be cast as integers without a significant change in the overall dynamics of the controlled system dynamics.

For instance, in motion control applications, incremental optical encoders (either linear or rotational) are exclusively employed to measure position of which is commonly characterized as integer counts of pulses being produced by these devices. Similarly, the manipulated output (torque or velocity command to the motor driver) is represented as a finite-length binary number to be latched onto a digital-to-analog converter. Note that within the context of this paper, such systems will be referred to as **"quantized input/output"** control systems. Furthermore, if the control gains could be also cast as integers, the resulting system will be called **"fully quantized system."**

For a fully quantized system, a state-space controller can be a suitable choice since the pole placement techniques offer a margin for manipulation on the controller gains. Note that the casting of these gains as integers is not a straightforward task as the input arguments of the gain matrix must be pre-scaled which may in turn aggravate the quantization noise. Hence, the overall problem requires a fine balance among conflicting objectives.

It is critical to notice that if a system is fully quantized, one may employ integer-arithmetic entirely in all calculations. On the other hand, for a quantized input/output system, the decimal multiplication algorithms of digital signal processing (such as shift-and-add algorithm) can be utilized. These well-known algorithms are easy to implement on FPGAs with the low-level design tools provided by FPGA manufacturers. Since many industrial motion control applications employ the quantized input-output, such methods are expected to reduce resource costs dramatically.

As explained in the previous sections, the implementation methods for state-space controllers are to be developed for quantized input-output or fully quantized systems. In these systems, representing the input, output, and feedback states as floating-point numbers do not have an advantage in terms of accuracy. Therefore, it is appropriate to cast and store these quantities as signed/unsigned integers. Note that the selection of the word size is up to the designer and is to be chosen by considering both the system properties (such as feedback resolution) as well as the features of the implementation method. The methods elaborated in the following section make good use of the special properties for control systems.

A. Proposed Methods:

Two multiplication methods are proposed for quantized input/output control systems. The details on these methods follow.

Method I

In FPGAs, it is possible to develop efficient integer multiplication/division algorithms with the logic-level (i.e. combinational circuit and/or embedded multipliers of the FPGA chip) design. Therefore, the first method, which employs a special multiplier module called "imul", focuses on multiplication of controller gains with system states employing integer arithmetic. To be specific, let us consider the following operation: $y = \lfloor ax \rfloor$ where a ($\in \Re$) is the multiplicand; x and y ($\in \mathbb{Z}$) are the multiplier and the result (product) respectively while $\lfloor\ \rfloor$ refers to floor function. It is obvious that one can represent the fractional number (a) in this operation as the ratio of two integers (N_a, D_a): $y \cong \frac{N_a x}{D_a}$. Hence, the overall problem is reduced to multiplication and division by integers. Note that the success of this approximation is directly correlated to the bit-length of the numerator and denominator.

Method II

Similar to the previous one, this method focuses on performing multiplication/division where the multiplier is essentially an integer. In fact, the proposed method facilitates a fixed-point multiplication unit (called "fmul") where bit-shift/add operations are successively employed to obtain the result. In this paradigm, the fractional number (*multiplicand*) is separated into an integer- and a fractional portion. Two instances of the multiplier module are used to multiply these portions in parallel. When both calculations are complete, the partial products are added to obtain the result.

Overall Architecture

A matrix multiplier module can be designed, utilizing the afore-mentioned multiplication modules (*imul* and *fmul*). Proposed module, which operates on (classical) *multiply-and-accumulate* principle, is illustrated in Fig 1. Note that the matrices are stored in the SRAM and thus the memory interface module sends out the relevant data to the registers of the multiplier controller unit (a finite state machine) on demand. Hence, the architecture provides flexibility in the controller design as the designers can change the data at will.

Similarly, the overall design, which is built on these matrix multiplier units, is presented in Fig. 2. This unit performs the computations in Eqn. (2) sequentially to obtain $\hat{\mathbf{x}}(k)$ and then proceeds to calculate **u(k)** in (1). Note that in the shown architecture all the computations are performed in sequential fashion for the sake of reducing the hardware cost. However, the parallel implementation can be easily realized by eliminating the multiplexer /demultiplexer units in Fig. 1 while using the instances of custom multiplication modules.

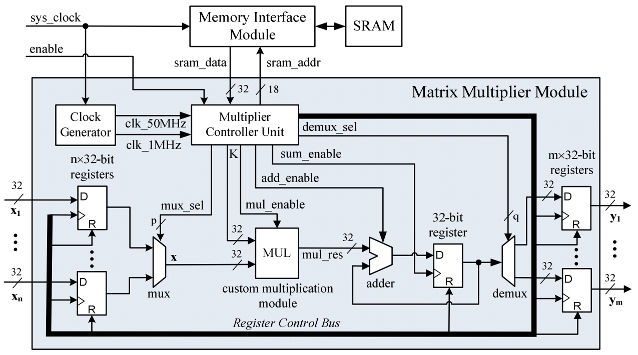


Fig. 1 – FPGA implementation of the matrix multiplication module

V. Test Case

As a benchmark case, an inverted pendulum system shown in Fig. 3 is considered. The system includes an AC servo-motor coupled directly to a timing belt. In this configuration, the motor driver is in the torque regulation mode where the motor along with its driver can be regarded as an ideal torque modulator. Hence, the state-space controller can directly generate the relevant (torque) commands through a digital-to-analog converter. Note that two rotary encoders (which can produce 10000 pulses/rev) are to supply feedback on the position of the carriage as well as the angular position of the pendulum. This choice is also convenient since the system has four states; $(x, \dot{x}, \theta \; and \; \dot{\theta})$ and given that only two states (x, θ) are available via encoder feedback in order to estimate $\dot{x} \; and \; \dot{\theta}$, an observer is required to be implemented in the design that satisfies the testing requirements for the developed methods.

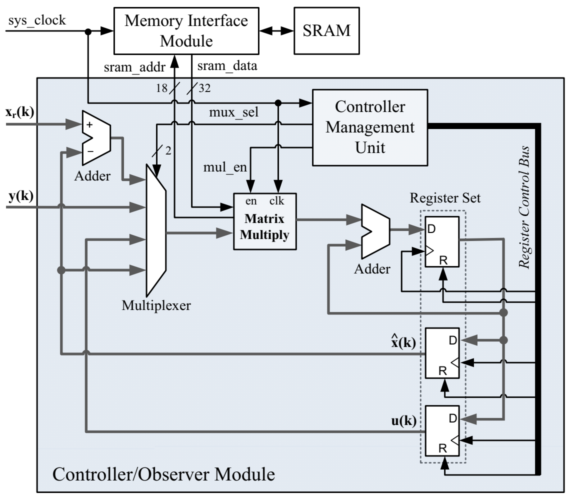


Fig. 2 – FPGA implementation of the controller/observer employing the matrix multiplication modules

Notice that this can be regarded as a "quantized input/output system" since all the sensor feedback is coming from incremental encoders as pulses and that the output of the FPGA is also an integer representing the motor torque. However, the system cannot be considered as a "fully quantized system" since there exist an observer in the design and while the coefficients of the controller can be cast to close integers via pole placement, it is not possible to implement an observer by using only integers.

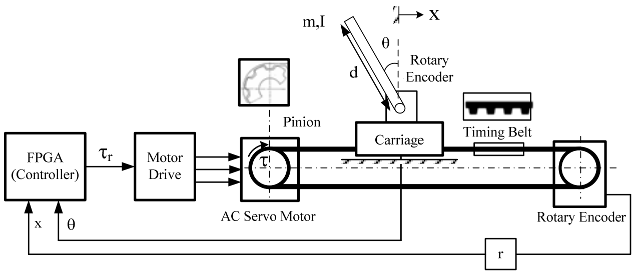


Fig. 3 – Generic model for pendulum drive system.

The force acting on the carriage is related to the motor torque as F = τη/r; here τ is the motor torque [Nm]; r is the pinion (pitch circle) radius [m] and η is the overall transmission efficiency. Equations of motion for this system become

$$(M+m)\ddot{x} + b\dot{x} + m\frac{d}{2}\ddot{\theta}cos\theta - m\frac{d}{2}\dot{\theta}^2 sin\theta = \frac{\tau}{r}\eta \quad (3)$$

$$\left(I + m\frac{d^2}{4}\right)\ddot{\theta} + mg\frac{d}{2}sin\theta = -m\frac{d}{2}\ddot{x}cos\theta \quad (4)$$

Here, M is the mass of the carriage [kg]; b is its viscous damping [Nms]; m is the mass of the pendulum, I is its mass moment of inertia [kgm^2]; τ is the manipulated input (i.e. motor torque). In order to design a controller and a state observer, the system is linearized around an operating point (θ = 0), since the goal of the control system is to hold the pendulum in upright position. Using (3) and (4), the state-space representation of the linearized system can be given as

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \dfrac{-b}{C_2} & -m\dfrac{d}{2}\dfrac{C_1}{C_2}g & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{-C_1}{C_2}b & \dfrac{C_1}{C_2}(M+m)g & 0 \end{bmatrix}}_{A} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \dfrac{\eta}{rC_2} \\ 0 \\ \dfrac{\eta C_1}{rC_2} \end{bmatrix}}_{B} \tau \quad (5a)$$

$$\mathbf{y} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{C}} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} \quad (5b)$$

$$C_1 = -m\frac{d}{2}\left(I + m\frac{d^2}{4}\right)^{-1}, C_2 = M + m + \frac{C_1 d}{2} \quad (6)$$

Having obtained the continuous-time state space representation, the discrete representations of (5) can be simply obtained as

$$\boldsymbol{x}(k+1) = \mathbf{F}\boldsymbol{x}(k) + \mathbf{G}u(k) \quad (7a)$$
$$\boldsymbol{y}(k) = \mathbf{H}\boldsymbol{x}(k) \quad (7b)$$

where

$$\mathbf{F} = e^{\mathbf{A}T}; \quad \mathbf{G} = \left(\int_{\mathbf{0}}^{\mathrm{T}} e^{\mathbf{A}t}\,\mathrm{d}t\right)\mathbf{B}; \quad \mathbf{H} = \mathbf{C} \quad (8)$$

Here, **A**, **B**, **C** denote to the system matrices of (5) while T refers to the sampling period of the controller [s]. With the discrete-time model at hand, a state-space controller (along with an observer) can be designed using modern control theory.

VI. Performance Evaluation via Hardware- in-the-Loop Simulation

In the simulation study, the controller topology presented in section III is implemented on an Altera Cyclone II FPGA chip using Verilog hardware definition language. The nonlinear inverted pendulum system (as explained in Section V) is realized via (non-real-time) hardware-in-the-loop simulation (HILS) in MATLAB environment.

The controller system (controller+observer) is implemented on the FPGA using both of the presented techniques: Method I (integer arithmetic) and Method II (fixed-point artihmetic). Furthermore, as Method III, floating-point computation module developed by Usselmann [19] is also employed as a reference case to evaluate performance. In this scheme, the manipulated torque input (u = τ) generated by the FPGA is transmitted to hardware-in-the-loop simulator running on MATLAB platform. Once the new system states are computed using the dynamic model of the controlled system, they are sent back to the FPGA via RS-232 communication. As explained in section IV, the word size allocated to the matrix elements is a major design parameter. In this study, a 2×16-bit word size is selected for each element of the controller/observer matrices: 32-bit for numerator/ denominator pair for *imul* multiplication module (Method I) or integer/fractional portion pair for *fmul* module (Method II). On the other hand, the system states are stored as 32-bit signed integers as explained in section IV. However, this choice is also a design choice and can be modified according to the needs of a particular application.

Results of the HILS are presented in Figs. 4 through 6. Note that the initial condition of the state vector is selected as [0 0 -0.05 0]^T to start out with a relatively large pendulum angle to create a challenge for the controller. As can be seen in Figs. 4-6, all controllers react fast and bring the carriage around 0.02m to prevent pendulum arm from falling; and after the initial reaction, the implemented controllers can successfully hold the inverted pendulum around $\theta = 0$ and the carriage's position are bounded around its initial position.

Since the system is nonlinear, small differences in calculation of the results between the proposed methods: I, II, and III (baseline) cause a relatively different path for the carriage after t = 2s; Nevertheless, all methods yield acceptable performance. Similarly, resource requirements of each method are summarized in Table I.

TABLE I. Resource/Time Costs of Proposed Methods on Altera Cyclone II FPGA chip

Method	Total logic elements	Embedded multipliers	Total clock cycles required	Max. sampling frequency
I	4449 (24%)	4 (8%)	4×32 cycles	400 kHz
II	3528 (19%)	0 (0%)	32×32 cycles	50 kHz
III	7778 (41%)	7 (13%)	20×32 cycles	80 kHz

As can be seen in Table I, the proposed methods I and II decrease the resource requirement (in terms of logic elements and embedded multipliers) significantly. Clock cycles necessary depend on not only the number of states of the controlled system but also the multiplication algorithm selected. The number of base cycles (N$_c$) can be expressed as follows:

$$N_c = n(n + 2m + n_y) \quad (9)$$

Here, n is the number of states; n_y is the number of measured states and m is the number of manipulated inputs. In this case, since n = 4, n_y = 2 and m = 1; N_c becomes 32 (as indicated in Table I). The duration of each base cycle depends on the design of the multiplier unit. Note that since the system under test is running on a 50-MHz clock, the FPGA is capable of real-time control with a sampling frequency up to 50 kHz (Method II) and 400 kHz (Method I). It is critical to note that since the speed of the serial communication (RS-232) used in this study is not sufficient for real-time simulation, a non-real time HILS is realized. For the simulation, a sampling frequency of 1 kHz is selected and thus the discrete-time state space representations are evaluated using Eqns. 7-8.

VII. Conclusion

Simulation results obtained through HILS suggest that full state-space controllers (employing Luenberger-type observers) can be efficiently implemented on FPGAs (even with limited resources) using the proposed methods. It is proven that a sampling frequency of 50 kHz (which is suitable even for demanding real-time control applications) could be attained. Furthermore, for a more complex system with large number of states, there would be no significant increase in the resource requirement and the only difference would be in terms of attainable sampling frequency that is inversely proportional to the number of states, as indicated by Eqn. (9).

The results suggest that proposed methods can be used to implement state-space controllers (along with full-state observers); using only logic-level design and some open source IPs such as a serial interface module and a SRAM controller.

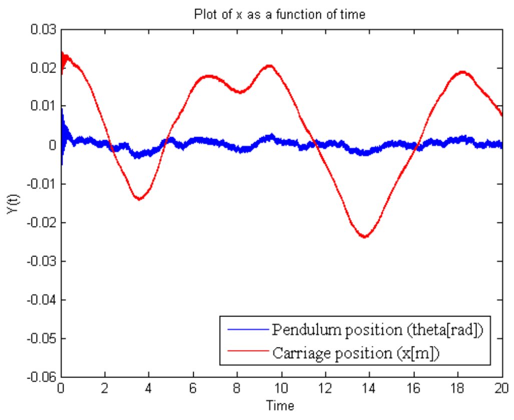


Fig. 4 – HILS result obtained for Method I

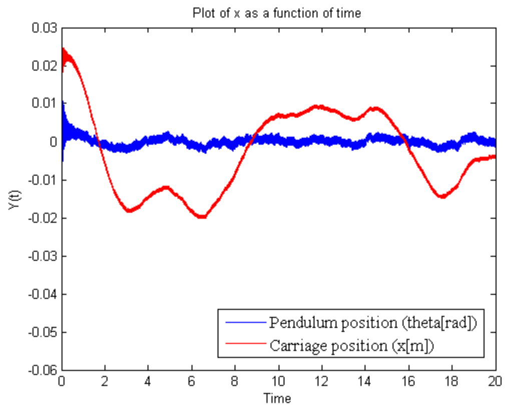


Fig. 5 – HILS result obtained for Method II

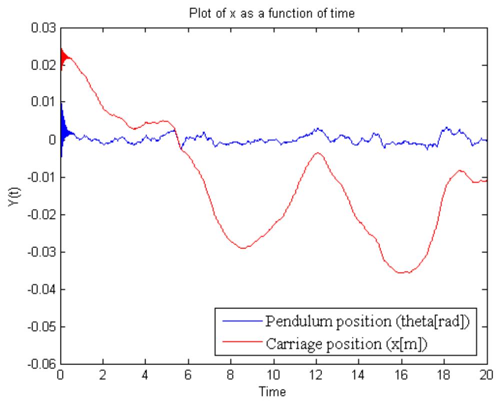


Fig. 6 – HILS result obtained for Method III

VIII. References

[1] Jung Uk Cho; Quy Ngoc Le; Jae Wook Jeon, "An FPGA-Based Multiple-Axis Motion Control Chip," *IEEE Transactions on Industrial Electronics,* vol.56, no.3, pp.856-870, March 2009.

[2] Ying-Shieh Kung; Rong-Fong Fung; Ting-Yu Tai, "Realization of a Motion Control IC for X-Y Table Based on Novel FPGA Technology," *IEEE Transactions on Industrial Electronics,* vol.56, no.1, pp.43-53, Jan. 2009.

[3] Xu Dong; Wang Tianmiao; Wei Hongxing; Liu Jingmeng, "A new dual-core Permanent Magnet Synchronous Motor Servo System," *Industrial Electronics and Applications, 2009. ICIEA 2009. 4th IEEE Conference on* , vol., no., pp.715-720, 25-27 May 2009.

[4] Birou, I.; Imecs, M., "Real-time robot drive control with PM-synchronous motors using a DSP-based computer system," *Power Electronics and Motion Control Conference, Proceedings of the Third International IPEMC 2000.*, vol.3, no., pp.1290-1295 vol.3, 2000.

[5] Seul Jung; Sung su Kim, "Hardware Implementation of a Real-Time Neural Network Controller With a DSP and an FPGA for Nonlinear Systems," *IEEE Transactions on Industrial Electronics,* vol.54, no.1, pp.265-271, Feb. 2007.

[6] Ni, F.L.; Jin, M.H.; Xie, Z.W.; Shi, Sh.C.; Liu, Y.Ch.; Liu, H.; Hirzinger, G., "A Highly Integrated Joint Servo System Based on FPGA with Nios II Processor," *Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation,* vol., no., pp.973-978, 25-28 June 2006.

[7] Li, Yan; Zhuang, Shengxian; Zhang, Luan, "Development of an FPGA-Based Servo Controller for PMSM Drives,", *2007 IEEE International Conference on Automation and Logistics*, vol. 18, no. 21, pp.1398-1403, Aug. 2007.

[8] Ying-Shieh Kung; Rong-Fong Fung; Ting-Yu Tai, "Realization of a Motion Control IC for X-Y Table Based on Novel FPGA Technology," *IEEE Transactions on Industrial Electronics,* vol.56, no.1, pp.43-53, Jan. 2009.

[9] Jung Uk Cho; Quy Ngoc Le; Jae Wook Jeon, "An FPGA-Based Multiple-Axis Motion Control Chip," *IEEE Transactions on Industrial Electronics,* vol.56, no.3, pp.856-870, March 2009.

[10] Chan, Y.F.; Moallem, M.; Wang, W., "Efficient implementation of PID control algorithm using FPGA technology," *43rd IEEE Conference on Decision and Control*, vol.5, no., pp. 4885-4890 Vol.5, 14-17 Dec. 2004.

[11] Yao dong Tao; Hu Lin; Yi Hu; Xiaohui Zhang; Zhicheng Wang, "Efficient implementation of CNC Position Controller using FPGA,".. *6th IEEE International Conference on Industrial Informatics (INDIN 2008)*, pp.1177-1182, 13-16 July 2008.

[12] Jia Lanping; Zhou Runjing; Liang Zhian, "Realization of position tracking system based on FPGA," *2004. Proceedings. ICSP '04. 2004 7th International Conference on Signal Processing,* vol.3, no., pp. 2588-2591 vol.3, 31 Aug.-4 Sept. 2004.

[13] Faa-Jeng Lin; Ying-Chih Hung, "FPGA-based Elman Neural Network Control System for Linear Ultrasonic Motor," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency,* vol.56, no.1, pp.101-113, January 2009.

[14] Ni, F.L.; Jin, M.H.; Xie, Z.W.; Shi, Sh.C.; Liu, Y.Ch.; Liu, H.; Hirzinger, G., "A Highly Integrated Joint Servo System Based on FPGA with Nios II Processor," *Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation,* pp.973-978, 25-28 June 2006.

[15] Li, Yan; Zhuang, Shengxian; Zhang, Luan, "Development of an FPGA-Based Servo Controller for PMSM Drives," *IEEE International Conference on Automation and Logistics, 2007*, pp.1398-1403, 18-21 Aug. 2007.

[16] Brown, R.H.; Schneider, S.C.; Mulligan, M.G., "Analysis of algorithms for velocity estimation from discrete position versus time data," *IEEE Transactions on Industrial Electronics*, vol.39, no.1, pp.11-19, Feb 1992.

[17] Mutlu, B.R.; Yaman, U.; Dolen, M.; Koku, A.B., "Performance evaluation of different real-time motion controller topologies implemented on a FPGA," *International Conference on Electrical Machines and Systems, 2009. ICEMS 2009.*, vol., no., pp.1-6, 15-18 Nov. 2009.

[18] Rahman, Z.; Ehsani, M.; Butler, K.L., "An Investigation of Electric Motor drive Characteristics for EV and HEV Propulsion Systems", SAE 2000-01-3062, 2000.

[19] R. Usselmann, *Open Floating Point Unit Manual*, www.opencores.org, 2000.

Çevrimiçi Donanım Benzetimi için Yeni bir Yazılım Paketi: Cadmus

Serdar Üşenmez[1], *Rasim Aşkın Dilan*[2], *Ulaş Yaman*[3], *Barış Ragıp Mutlu*[4], *Melik Dölen*[5], *Ahmet Buğra Koku*[6]

Makina Mühendisliği Bölümü
Orta Doğu Teknik Üniversitesi, Ankara
[1]serdar.usenmez@gmail.com
[2]adilan@metu.edu.tr
[3]uyaman@metu.edu.tr
[4]brmutlu@gmail.com
[5]dolen@metu.edu.tr
[6]kbugra@metu.edu.tr

Özetçe

Bu makalede ayrık zamanlı kontrol teorisi üzerine verilen derslerdeki teorik bilgilerin uygulamaya konulmasına dair bir "Evdeki Laboratuvar" yöntemi dahilinde öğrencilere sunulabilecek, gerçek doğrulukla benzetim gerçekleştiren "Cadmus" adlı yazılım paketi tanıtılmıştır. Öğrenciler sahip oldukları bilgiler ışığında, mikrodenetleyici içeren bir donanım üzerinde bir denetleyici gerçeklerler. Bunu yazılımla haberleştirerek tasarladıkları denetleyicinin başarımını gözleme ve iyileştirme imkanına sahip olurlar. Bu sayede öğrencilerin donanımlı laboratuvarlara ihtiyaç duymaksızın; bilgilerini uygulama, gözleme ve ilerletme fırsatını yakalayıp, gördükleri eğitimden azami verimi almaları mümkün olur.

Anahtar Sözcükler: Kontrol Eğitimi, Ayrık Zamanlı Kontrol, Çevrimiçi Donanım Benzetimi, Evdeki Laboratuvar

1. Giriş

Ayrık zamanlı kontrol sistemleri üzerine verilen derslerde, öğrencilerin aldıkları teorik bilgileri özümsemeleri için çeşitli sistemlerde kullanılmak üzere denetleyiciler tasarlamaları ve bunları (mümkünse gerçek sistemlere uygulayarak) sınamaları istenir. Böylece, öğrencilerin kontrol edilen sistemin davranışını ayrıntılarıyla kavrayarak, tasarım sürecinde tercih edilen yöntemlerin ve yapılan varsayımların sonuca etkilerini görmeleri mümkün olur. Gerçek uygulamalar üzerine yapılan çalışmalar aracılığıyla söz konusu yetilerin kazandırılması hiç şüphesiz ki en geçerli eğitim tekniğidir.

Ancak bu yöntemin pratik olarak uygulanabilirliği pek yoktur. Herşeyden önce, söz konusu sistemler eğitim kurumundaki laboratuvarlara alınamayacak özelliklerde (boyut, gürültü, titreşim, zehirli madde salımı, vs.) veya maliyette olabilir. Bunun yanı sıra çözümleme amacıyla sistemin içerisinde gerçekleşen olaylarla ilgili veri toplamak, çok sayıda duyucunun uygun şekilde yerleştirilmesini gerektiren zorlu bir süreçtir. Bu sorunlar aşıldığı takdirde dahi, ardışık eğitim-öğretim dönemlerinde aynı sistemin sürekli kullanımı dersin geliştirilebilmesi açısından tercih edilmeyebilir. Bu durumda hem eldeki sistemin uygun şekilde değerlendirilmesi, hem de yeni bir sistemin bulunarak kullanıma hazırlanması gereği ortaya çıkmaktadır. Bu yöntemdeki bir diğer sıkıntı ise, çeşitli sebeplerle laboratuvar ekipmanının kullanım dışı kalması durumunda bütün çalışmaların ertelenmek zorunda kalmasıdır. Laboratuvarda bütün ideal şartlar oluşturulsa bile, öğrencilerin söz konusu sistemi belirli bir zaman dilimi içinde kullanmaları gerekeceğinden, kazanabilecekleri deneyim yapılan harcamalara göre çok sınırlı kalacaktır.

Bu makalede, (ayrık zamanlı) kontrol sistemleri üzerine verilen derslerle ilgili bahsedilmiş olan sıkıntıları aşmak üzere geliştirilmiş yeni bir benzetim yazılımı tanıtılmaktadır. Makalenin ikinci kısmında, ilgili alanda yapılmış çalışmalar kısaca özetlenmiştir. Üçüncü kısımda, söz konusu yazılımın kullanıldığı "evdeki laboratuvar" kavramı tartışılmıştır. Dördüncü bölümde, makalede bahsi geçen yöntemin pilot uygulaması anlatılmış ve elde edilen sonuçlar incelenmiştir. Son kısımda ise tartışma ve sonuçlar bulunmaktadır.

2. Mevcut Çalışmaların İncelenmesi

Çevirimiçi Donanım Benzetimi, (ÇDB, "Hardware-in-the-Loop Simulation") esas itibariyle geliştirilmiş bir donanımın (bilgisayarda benzetimi yapılan) sanal bir sistem üzerinde sınanmasını sağlayan önemli bir tekniktir. Bu nedenle tasarım problemlerine çözümler geliştirilmesi aşamasında; kontrol mühendisliğinde söz konusu teknik oldukça sık olarak kullanılır [1-2]. Geride bıraktığımız 30 sene boyunca çeşitli füze güdüm sistemlerinin denenmesi sırasında da yaygın bir şekilde ÇDB'nin kullanıldığı görülmüştür [3-6].

ABD'de Idaho Üniversitesi'nde karmaşık trafik durumlarının incelenmesi sırasında sadece bilgisayar benzetiminin yeterli olmadığı görülmüş, ÇDB'nin kullanılmasıyla gerçek trafik denetleyicileri ile etkileşen benzetim ortamının daha büyük başarım sağladığı anlaşılmıştır [7]. ABD'de yapılan başka bir araştırmada da trafik modeli geliştirmek için ÇBD kullanımından bahsedilmiştir. Modeller geliştirilirken dış kontrol teçhizatı ile etkileşimin ÇBD sayesinde gerçek zamanlı modele çok yaklaştığı bu nedenle de model geliştirme sırasında kolaylık sağladığı belirtilmiştir. [8].

İngiltere'nin Jaguar firması çeşitli üniversitelerle ortak yürüttüğü çalışmalarda ÇDB'yi otomobillerinin çeşitli otomatik kontrol sistemlerini geliştirmek için kullanılmış.

Geliştirilen model gerçek zamanlı koşacak şekilde değiştirilmiş ve benzetim gerçek bir araç donanımıyla gerçeklenmiştir. Sistem verimi nedeniyle daha sonra Jaguar fabrikalarından bir kaçına da uyarlanmışlardır. [9]. Diğer taraftan, Kanada'nın Uluslararası Uzay İstasyonu için geliştirdiği robot kolun tasarımı sırasında ÇDB'den yararlanılmış ve uzay koşullarının benzetimi yapılarak tasarımın büyük verim ve az maliyetle yapılması sağlanmıştır [10]. Bunun haricinde; ÇDB, Hollanda'da Delft Üniversitesi'nde gerçekleştirilen bir araştırma sırasında uzamsal kısıtlı sürekli dinamik sistemlerin doğasının daha iyi anlaşılmasını sağlamak amacıyla kullanılmıştır. [11]. Bütün bunların yanı sıra, dünyanın çeşitli üniversitelerinde kontrol sistemleri tasarımı ve gerçekleştirimi üzerine projelerde de ÇDB kullanılmaktadır. Laboratuvar olanağı kısıtlı kurumlarda ise öğrencilerin ilgisini canlı tutmak ve aynı zamanda deneysel bir eğitim ortamı sunmak amacıyla ÇDB projelerde kullanılmaya başlanmıştır [1], [7].

Araştırma kurumlarında yapılan çalışmalar ile yüksek lisans ve doktora çalışmalarının yanı sıra, lisans ve yüksek lisans seviyesinde verilen bazı kontrol ve modelleme temelli derslerde de ÇDB kullanılmaktadır. Utah Üniversitesi'nde verilmekte olan ECE5320 kodlu "Mechatronics" dersi ile ECE7330 kodlu "Nonlinear and Adaptive Control" dersi dahilindeki laboratuvar deneyleri sırasında, ÇDB çeşitli doğrusal olmayan sistemlerin test edilmesinde ve bu sistemlere uygun çeşitli kontrol sistemleri geliştirilmesinde kullanılmaktadır. Bu derslerde kurulan ÇDB uygulaması bir düzenek ve kurulan düzeneğe uygulanan kontrolcünün tepkisinin MATLAB / Simulink aracılığıyla grafiksel olarak ifade edilmesinden ibarettir [12].

Benzer olarak, Texas Üniversitesi Makina ve Havacılık Bölümü'nde bazı kontrol derslerinde dinamik sistem çözümlemesi ve kontrol yöntemleri geliştirilmesinden sonra, bunların sonuçlarının değerlendirilmesi için ÇDB kullanılmıştır. Manyetik levitasyon cihazı ile birleştirilmiş ÇDB uygulamasını "Introductory Control" dersine uygulayan yazarlar, öğrencilerin kontrolcülerini değerlendiren bir donanım benzetimi karşısında heyecanlandıklarını ve derse olan ilgilerinin arttığını belirtmişlerdir. Bu uygulamada da benzetim sonuçları sadece MATLAB / Simulink aracılığıyla grafikler üzerinden görüntülenmektedir [13].

3. Geliştirilen Yöntem

3.1. Evdeki Laboratuvar

Denetleyiciler üzerine verilen (lisans veya yüksek lisans) derslerin etkinliğinin arttırılabilmesi için, eğitim faaliyetlerinin birçok geniş çaplı laboratuvar çalışmasıyla desteklenmesi gerekmektedir. Diğer taraftan, en geniş imkanlara sahip eğitim kurumları bile, en iyi şartlar altında öğrencilerine birkaç alışıldık deney düzeneği ile çok sınırlı bir zaman diliminde çalışma yapma fırsatı tanıyabilmektedirler. "Evdeki Laboratuvar" (veya "Cepteki Laboratuvar") kavramı, böylesi sorunların üstesinden gelmek amacıyla ortaya atılmıştır [14].

Bu yöntemde öğrenciler tasarladıkları denetleyicilerini, gerçek sisteme bağlamak yerine, sistemin dinamiğini ileri derecede taklit edebilecek bir bilgisayar sistemine bağlarlar. ÇDB, temelde denetlenen sistemin bir benzetimini yapan güçlü bir bilgisayar ve denetleyici cihazla bu bilgisayar arasında yer alan bir arayüzden oluşmaktadır. Esas itibariyle bu benzetim, sistemin davranışını betimleyen diferansiyel denklemlerin uygun sayısal yöntemler aracılığıyla çözülmesiyle gerçekleşir. Denetleyici tarafından bakıldığında, gerçek sisteme bağlanmış olmak ile benzetim bilgisayarının arayüzüne bağlanmak arasında temelde bir fark bulunmamaktadır. Öğrenciler denetleyicilerini bahsedilen arayüze bağlayarak birçok koşulun benzetimini yapabilir ve çok sayıda sınamayı kolaylıkla gerçekleştirebilirler. Bu sınamalar esnasında sistem parametrelerinin zaman içinde değişimi, doğrusal olmayan elemanlar, bozucular, gürültü, çeşitli elemanlarının arızalanması gibi birçok durumun benzetiminin yapılması mümkündür. Bu sayede öğrenciler gerçek dünyada ortaya çıkabilecek her türlü etki ve sorunla karşı karşıya gelebilecek ve bunlar üzerinde çalışma imkanı bulacaktır. Bu imkanların yanı sıra, yeni bir sistem üzerinde çalışılmak istendiğinde bu sistemi betimleyen denklem ve parametrelerin ilgili yazılıma tanıtılması yeterlidir.

Çevrimiçi Donanım Benzetimi'nin, "Evdeki Laboratuvar" fikrine uyarlanması güç değildir. Halihazırda piyasada bulunan birtakım matematik yazılım paketleri (örneğin; Mathematica, MATLAB / Simulink) zaten diferansiyel denklem çözümü, sonuçların görselleştirilmesi, bilgisayar ile çeşitli donanımlar arasında iletişim sağlamak gibi imkanları sunmaktadır. Yazılım ve donanım alanlarında gerekli düzenlemeler yapıldığı taktirde, bu paketler vasıtasıyla ÇDB yapılabilmesi mümkündür. Diğer taraftan, kullanıcı dostu bir ÇDB ortamının böylesi paketlerle yaratılması oldukça zordur. Ayrıca bu yazılımlar hem çok pahalıdır, hem de elde edilen ürünlerin bilgisayarlar arasında taşınabilirliği oldukça düşüktür.

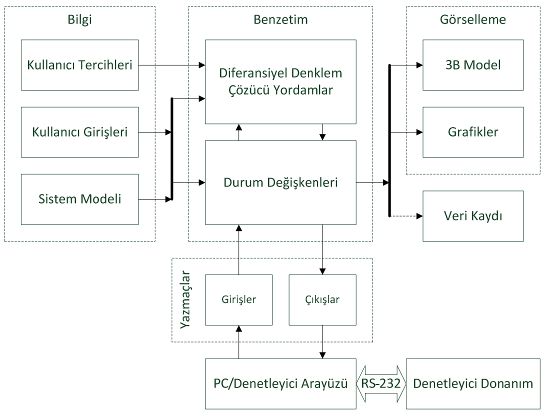
3.2. Önerilen Tümleşik Sistem

Bu makalede, sözü edilen sorunların üstesinden gelebilmek için, ÇDB merkezli bir "evdeki laboratuvar" yöntemi üzerine kurulu, tümleşik bir sistem önerilmektedir. Bu sistemin oluşturulmasındaki temel amaç; öğrencilere laboratuvar ortamı dışında; kendi programları (ve öğrenme hızları) uyarınca çalışma imkanı vererek, denetleyici tasarımına yoğunlaşmalarını sağlamaktır.

Bu bağlamda, denetlenecek dinamik sistemlerin benzetimlerinin yapılabildiği bir ÇDB yazılımı ("Cadmus") geliştirilmiştir. Öğrenciler, kendilerine sunulan bir geliştirme kartı üzerinde uygulamasını yaptıkları denetleyiciyi seri port üzerinden bilgisayara bağlayarak, benzetimi yapılan sistem ile haberleşmektedir. Bu haberleşmenin öğrenci tarafından gerçeklenebilmesi için kullanılan protokolü açıklayan belgeler ve bu protokolün kullanımını sağlayacak yazılım kütüphaneleri öğrencilere sağlanmaktadır. Kontrol edilmekte olan sanal sistemin davranışı, yazılım üzerinde 3 boyutlu katı modeller aracılığıyla gerçekçi olarak görüntülenmekte olup, arzu edildiği takdirde sistemin önemli durum değişkenleri de yine bu yazılım aracıyla (sayısal olarak veya grafik üzerinde) izlenebilmektedir. Ayrıca dışarıdan değiştirilebilmesi istenen sistem parametrelerinin (bozucular, gürültü, yükler vb.) değerleri yazılım aracılığıyla ayarlanabilmektedir. Bu gereçler aracılığıyla öğrenciler, denetleyicilerini sınama, ince ayarlama ve eniyileştirme çalışmalarını gerçekleştirme imkanına sahip olmaktadırlar. Ders eğitmeninin geliştirilmiş olan denetleyicileri değerlendirmesi, yine bu yazılım aracılığıyla olmaktadır.

Cadmus yazılımı C# programlama dili kullanılarak, Microsoft .NET Framework yazılım ortamı üzerinde, Microsoft Windows XP ve Microsoft Windows Vista işletim sistemlerinde işleyebilecek biçimde geliştirilmiştir. Çeşitli bilgilerin sunulması ve alınmasında işletim sistemine has kullanıcı arayüzü formları kullanılmış olup, 3 boyutlu katı modelleme ***Managed DirectX*** grafik kütüphanesi aracılığıyla gerçekleştirilmiştir. Bu ortam ve kütüphanelerin kullanımı, hedeflenmiş işletim sistemlerini kullanan farklı bilgisayarlar arasında yazılımın taşınabilirliğini arttırmıştır [15-16].

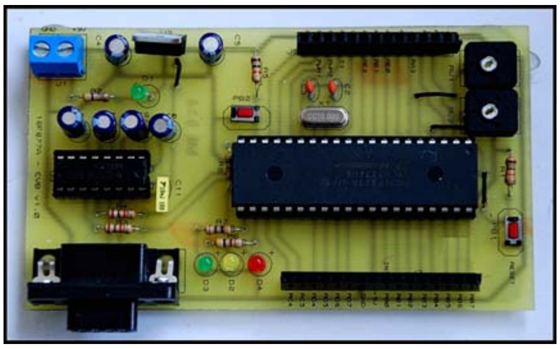
Cadmus yazılımının işlev şeması Şekil 1'de görülebilir. Benzetimi yapılacak olan sistemin matematiksel modeli, sisteme ait sabitler, kullanıcı seçenekleri ve yazılım parametreleri yazılımın *bilgi* katmanını oluşturur. *Benzetim* katmanında sistemin dinamik modelini kullanarak, durum değişkenlerini saptayan sayısal çözümleyici yordamlar bulunur. Sisteme ait giriş ve çıkışlar, durum değişkenlerinden elde edilerek, ilgili *yazmaçlara* aktarılır. *PC / denetleyici arayüzü* üzerinden, ilgili denetleyici donanımın okuma/yazma erişimine sunulur. Sistemin durum değişkenleri, aynı zamanda *görselleme* katmanına iletilerek üç boyutlu ve grafik görselleme için kullanılıp, arzu edildiği takdirde çözümleme için bilgisayarın sabit diskine kaydedilir.



Şekil 1: Cadmus'un işlevsel şeması

Denetleyici algoritmaların geliştirilmesi amacıyla öğrencilere tedarik edilen kart, üzerinde PIC 16F877A mikrodenetleyici taşıyan, düşük maliyetli bir karttır (Şekil 2). Bu kart seri port (RS-232 protokolü) üzerinden kişisel bilgisayarlar ile iletişim kurabilmektedir. Kart üzerinde; bir düğme, iki adet potansiyometre, LED gibi çeşitli elemanlar bulunmakta olup, bunlardan geri bildirim ve hata ayıklama amaçlı olarak faydalanılabilmektedir. Diğer taraftan, mikrodenetleyicinin sahip olduğu bütün giriş ve çıkışlara kartın iki yanındaki dişi soketler aracılığıyla kolayca erişmek mümkündür. Bu geliştirme kartı öğrencinin temel ihtiyaçları düşünülerek tasarlanmış olup, ciddi bir elektronik devre bilgisine sahip olmadan da öğrencinin rahatlıkla denetleyici algoritmasını geliştirebilmesini sağlamaktadır.

Sunulmuş olan sistemin, çeşitli yazılım paketleriyle gerçeklenebilir muadillerine kıyasla en önemli özelliği, 3 boyutlu katı modelleme ile benzetimi yapılan sistemin görselleştirilmesidir. Bu sayede denetleyicinin sınanması, sistem parametrelerinin sayısal değerlerinin (veya bu değerlerden elde edilecek grafiklerin) izlenmesinin ötesine geçmektedir. Öğrenci, sistemi, gerçek bir laboratuvarda gözleyebileceği gibi inceleme imkanı bulmaktadır. Hatta gerçekte çeşitli sebeplerle (yağlama, yalıtım, güvenlik vb.) kapalı geometride bulunan bileşen ve sistemlerin (dişli ve



Şekil 2: PIC 16F877A mikrodenetleyici barındıran geliştirme kartı

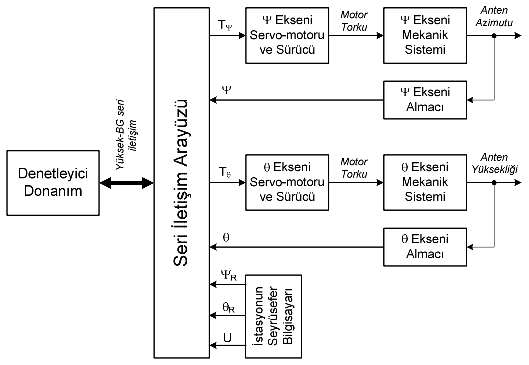
aktarım kutuları, içten yanmalı motorlar, çamaşır makineleri vb.), istenen parçalarının şeffaflaştırılması veya tamamen görünmez kılınmasıyla iç işleyişlerinin gösterilmesi de mümkündür. Bu özellikler sayesinde hem benzetim süreci öğrencinin sebep ve sonuçlarıyla görerek kavrayıp yorumlayabileceği bir hale gelmekte, hem de öğrencinin ilgisi canlı tutulabilmektedir.

Denetlenen sistemin yanı sıra, öğrencinin kullanacağı mikrodenetleyicilerin de bilgisayarlı benzetiminin yapılması mümkündür [17-18]. Ancak sunulmuş olan yöntemde, denetleyici donanımın benzetiminden özellikle kaçınılmış ve gerçek bir mikrodenetleyicinin kullanımı öngörülmüştür. Bundaki temel amaç, tedarik edilen geliştirme kartı sayesinde, öğrencinin bir denetleyicinin prototipinin oluşturulmasında karşılaşacağı elektronik elemanlarla tanışmasının sağlanmasıdır. Öğrenci bu elemanların kullanımında karşılaşılabilecek çeşitli durumlardan bu sayede haberdar kılınmaktadır. Böylece eğitimin fiziksel uygulamalardan tam anlamıyla kopması engellenmiş olmaktadır.

4. Yöntemin Pilot Uygulaması

4.1. Uygulama Projesi

Bu makalede sunulan yöntemin pilot uygulaması, 2007-2008 eğitim-öğretim yılı bahar döneminde, Orta Doğu Teknik Üniversitesi Makina Mühendisliği bölümünün ME 534 (Bilgisayarla Makina Denetimi) dersinde final projesi olarak gerçekleştirilmiştir. Bu projede öğrencilerden alçak bir yörüngede seyreden bir iletişim uydusunu takip etmesi istenen bir anten çanağının iki eksenli hareket denetiminin yapılması istenmiştir. Anten, yükseklik ve azimut eksenlerinde birer servo motora ve motorlardan istenen torku (belirli hız sınırlamaları dahilinde) üretecek motor sürücülerine sahiptir. Ayrıca uydunun herhangi bir anda antene göreceli konumunu hesaplayabilen bir seyrüsefer bilgisayarı ile uydudan alınan sinyalin gücünü ölçen bir almacın mevcut olduğu varsayılmıştır (Şekil 3). Bu sistemlerden gelen bilgi ve antenin her iki eksendeki açısal konumu, PC arayüzünde tanımlanmış 16 bitlik yazmaçlar üzerinden sunulmuştur. Öğrencilerden bu bilgileri kullanarak, bir kontrol döngüsü içinde motor sürücüleri için tork komutlarını (yine 16 bitlik veriler halinde) üretmeleri beklenmektedir. Antenin bulunduğu ortamda bozucu olarak azami hızı 30m/s'ye kadar çıkarılabilen rüzgar mevcuttur. Ayrıca, motor-anten bağlantı mekanizmasındaki dişlilerde boşluk bulunmakta olup, her iki eksende kuru ve viskoz sürtünmelerin etkili olduğu varsayılmaktadır. Bu şartlar altında antenin, belli hata toleranslarını aşmadan, iki dakikalık seyri boyunca uyduyu takip etmesini sağlamak ciddi bir kontrol mühendisliği problemidir. Böylesi (ideal olmayan) şartlar, öğrenciyi klasik denetleyici algoritmaların ötesine geçmeye ve daha gelişmiş yöntemleri araştırarak gerçeklemeye teşvik etmektedir.



Şekil 3: Uygulama projesindeki anten sisteminin fonksiyonel blok diyagramı

Benzetimde kullanılan modellerden de kısaca bahsedilecek olursa; antende her iki eksen (azimut ve yükseklik) için de (tahvil oranı N_g olan) birer dişli kutusunun kullanıldığı kabul edildiğinden, aşağıdaki differansiyel denklemler,

$$J_m\ddot{\theta}_m = T_\theta - T_g \cdot N_g^{-1} \tag{1}$$

$$J\ddot{\theta} + b\dot{\theta} + T_c\,\mathrm{sgn}(\dot{\theta}) = T_g - T_d \tag{2}$$

ilgili ekseninin hareketini tanımlamaktadır. Burada J_m ve J sırasıyla motorun ve anten mili üzerine indirgenmiş kütle atalet momentlerini [kgm^2]; θ_m ve θ, motorun ve antenin açısal konumlarını [rad]; T_g dişli kutusu çıkış torkunu [Nm]; T_θ motor tarafından üretilen momenti; T_c anten milindeki kuru sürtünme momentini [Nm]; T_d rüzgarın anten mili üzerinde yaratmış olduğu bozucu momenti [Nm]; b viskoz sürtünme katsayısını [Nm/(r/s)] ifade etmektedir. Dişli kutusunda boşluk bulunduğu varsayıldığından, T_g dişli kutusunun giriş ve çıkış mili arasındaki konum farkının bir fonksiyonudur:

$$T_g = \begin{cases} k(\theta_m - N_g\theta - h), & (\theta_m - N_g\theta) \geq h \\ k(\theta_m - N_g\theta + h), & (\theta_m - N_g\theta) \leq -h \\ 0, & |\theta_m - N_g\theta| < h \end{cases} \tag{3}$$

Burada, h (> 0) dişli kutusundaki boşluğu [(μ)rad]; k (> 0) temas halinde olan dişlerin eşdeğer rijitliğini temsil eden bir katsayıdır (elastikiyet katsayısının tersidir).

4.2. Uygulama İçin Geliştirilen Yazılım

Cadmus yazılımının erken bir örneği, Bölüm 4.1'de açıklanan uygulama projesinde kullanılmak üzere özelleştirilmiştir. Bu sürümde, öğrencilere yardımcı olmak amacıyla, grafik kullanıcı arayüzünde; anten durum bilgilerinin yanında uydunun gerçek konumu ve antene uygulanan bozucu tork büyüklükleri gibi birtakım önemli bilgiler de yer almaktadır. Bununla beraber, benzetimde elde edilen veriler hem grafikler aracılığıyla betimlenir, hem de daha sonra analiz için bir bilgisayar dosyasına kaydedilir. Benzetim süresince antenin üç boyutlu imgesiyle bunun iki temel düzleme izdüşümü kullanıcı arayüzünde görüntülenir. Ayrıca, bu görüntü üzerinde yer alan imleçler uydunun antene göre konumunu da gösterirler. Rüzgarın şiddeti ve yönünü gösteren bir rüzgar gülü de antenle beraber görüntülendiğinden, rüzgarın antene olan etkisinin daha rahat anlaşılabilmesi mümkün olmaktadır.

Dikkat edilirse, öğrencinin elindeki geliştirme kartı sınırlı kaynaklara sahiptir ve kontrol döngüsü içinde yer alan hesapları gerçek zamanlı olarak yapamaz. Bunun yerine, arayüze bağlı donanımından gelen komutlar (tork komut yazmaçlarına yapılan yazma işlemi), benzetimin öğrenci tarafından seçilen örnekleme zamanı kadar ilerletilmesini sağlar. Bunun arkasından benzetim duraklatılır ve denetleyici donanımdan gelecek olan bir diğer komut seti beklenir. Sonuçta, iki komut arasında geçen süre gerçekte ne olursa olsun, benzetim belirlenmiş örnekleme süresine göre yürütülür. Böylece öğrencinin geliştirme kartının yarattığı kısıtlamalardan bir ölçüde kurtulması sağlanarak, geliştirdiği algoritmaya ve onun başarımına odaklanması sağlanır.

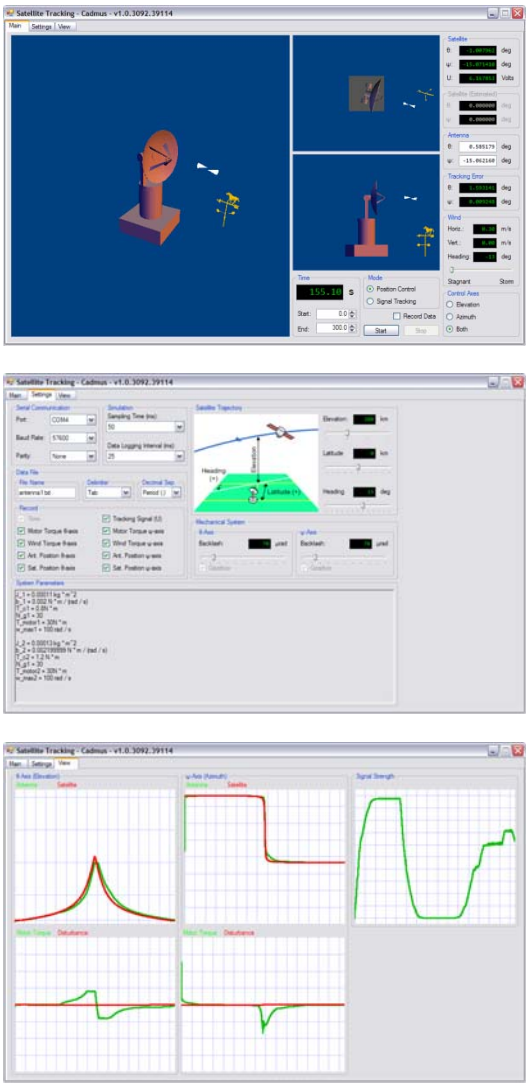
Şekil 4'te gösterildiği üzere; yazılım üç formdan oluşmaktadır. Benzetimin başlatılıp durdurulabildiği *ana* formda ("*Main")*; anten ve uydunun konum bilgilerinin yanı sıra rüzgar (bozucu) durumu ve benzetim süresi gibi önemli değerler gösterilir. Ayrıca bu formda sunulan seçeneklerle, kullanıcının benzetimi istediği bir zaman aralığı için gerçekleştirmesi mümkün kılınmıştır. Geliştirme sürecini kolaylaştırmak için, sadece istenen bir eksenle çalışma imkanı da öğrencilere sunulmaktadır.

Ayarlar ("Settings") formunda RS-232 iletişimine dair birtakım parametreler sunulmuştur. Bunun yanında, benzetimin (daha önce bahsedildiği üzere) gerçek zamanlı işlemeyen bir denetleyiciyle uyumlu çalışabilmesi için, denetleyicinin örnekleme periyodu bu form üzerinden yazılıma bildirilir. Yine bu formda, benzetimde elde edilen verilerin bilgisayarın sabit diskine kaydedilmesiyle ilgili ayarlar mevcuttur. Burada hangi durum değişkenlerinin sabit diske yazılacağı, oluşturulacak dosyanın adı ve bu dosyaya yazım biçimi (formatı) belirtilebilir. Bunların yanı sıra, takip edilen uydunun yörüngesine ait parametreler (yükseklik, uydu antenle aynı boylamdan geçerken anten ile arasındaki enlem farkı, yörüngenin anten enlemini kesme açısı) yine bu ekrandan değiştirilebilir. Ayrıca, antenin eksen kontrolünü gerçekleştiren motorlara bağlı olan dişli kutularındaki boşluğun büyüklüğünü mikroradyan cinsinden belirlemek de mümkündür.

Görüntüleme ("View") ekranında ise benzetimde kullanılan durum değişkenlerinin zaman içinde değişimi gösterilir. Bu formda her iki eksen için ikişer grafik bulunmaktadır. İlk grafik, uydunun ve antenin o eksendeki konumlarını göstermektedir. İkinci grafik ise ilgili eksende uygulanan motor torku ile rüzgar tarafından o eksene uygulanan bozucu torku temsil etmektedir. Bu ekrandaki beşinci bir grafikte ise, o anda anten tarafından alınmakta olan takip işareti, gerilim cinsinden gösterilmektedir. Daha yakından inceleme için, her grafikte istenen bölgeye yakınlaşmak mümkündür. Söz konusu bütün grafikler benzetim esnasında sürekli olarak güncellenir ve arzu edilen herhangi bir anda öğrenci tarafından izlenebilir.

Yazılım içerisinde, anten, her iki eksen ve bunlara ait denetleyiciler tek bir sınıf (“class”) altında toplanarak modellenmiştir. Bu sınıf, motor ve rüzgarın oluşturduğu torkları hesaplayarak, zamanın da bir fonksiyonu olan durum denklemlerini (adi diferansiyel denklem setlerini) iyi bilinen sayısal tümlevleme yöntemleriyle çözerek, ilgili eksenlerdeki açısal konumları elde eder.

Bir uydunun istenen bir yörüngeye yerleştirilmesi için gereken hesaplamaların karmaşıklığı nedeniyle, uydu, anten gibi bir dinamik sistem olarak modellenmemiştir. Bunun yerine, dünya etrafındaki hareketi dairesel kabul edilerek; basit kinematik bağıntılar aracılığıyla antene göre konumu (yükseklik ve azimut açıları) zamanın bir fonksiyonu olarak hesaplanır.



Şekil 4: Pilot uygulama yazılımının formları

Diferansiyel denklem çözücü yordamlar, yazılım içerisinde ayrı bir sınıf altında yer alır ve belirli bir şablona uygun fonksiyonları kabul edecek şekilde tasarlanmıştır. Bu şablona uygun olmak kaydıyla; herhangi bir dereceden durum denklemi (veya denklemleri) çözücü yordamlara sunularak ilgili değişkenlerinin (azimut ve yükseklik açıları ile bunların zamana göre türevleri) çözülmesi mümkündür. Halihazırda böylesi denklemlerin çözülmesi amacıyla 4. dereceden sabit adımlı Runge-Kutta yöntemi kullanılmaktadır [19].

Yazılım, esas itibariyle, bilgisayarda iki bağımsız izlek (“thread”) yürütmektedir. Birinci izlekte, bölüm 3.2’de anlatılmış olan *benzetim* ve *yazmaçlar* katmanlarına dair işlemler yürütülmektedir. Benzetimin başlatılmasıyla beraber denetleyici donanımdan (seri port aracılığıyla) gelecek olan komutlar beklenir. Bu komutlar alındığında benzetim, seçilmiş olan örnekleme süresi kadar ilerletilir. Ardından başa dönülerek denetleyiciden gelecek sonraki sinyaller beklenir. İkinci izlek ise, yine bölüm 3.2’de anlatılmış olan görselleme katmanına dair işlemlerden sorumludur. Kullanılmış olan bu çok izlekli (“multithreaded”) yapı sayesinde, gerçekleştirilme hızı grafik donanımına son derece bağlı olan (dolayısıyla yüksek süratle gerçekleştirilmesi her bilgisayarda sağlanamayan) görselleme işlemlerinin, benzetim işlemlerinin önünü tıkaması engellenmiştir.

Kullanılan yazılımın bir diğer özelliği de çalıştırıldığı bilgisayar üzerinde etkin bir internet bağlantısı bulunduğu takdirde; belirli aralıklarla derse ait Web sitesine bağlanarak güncelliğini teyit edebilmesidir. Böylece, yazılım daha güncel bir sürümün varlığını öğrendiğinde; kullanıcıyı bu konuda uyararak, yeni sürümün indirilebileceği bir web sayfasına bağlanma seçeneği sunar. Dersi veren öğretim üyesi veya araştırma görevlileri tarafından yapılacak herhangi bir değişiklik, böylece en çabuk biçimde öğrencilere ulaştırılabilmektedir.

4.3. Uygulama İçin Geliştirilen Arayüz Protokolü

Daha önce belirtildiği üzere, öğrencilerin geliştirdikleri denetleyiciler ile Cadmus arasında 16-bit (2 bayt) uzunluğunda verilerin aktarımı söz konusudur. Bu sabit veri uzunluğu sayesinde, geliştirme kartındaki mikrodenetleyici üzerinde hızlı çalışacak ve düşük bellek kullanacak bir iletişim protokolü oluşturulmuştur. Bu protokolü gerçekleştiren CCS C kaynak kodları da öğrenciye sağlanmıştır [20]. Burada, denetleyicinin erişimine açık olan PC yazmaçlarına okuma ve yazma işlemleri, CCS C’nin seri port yerleşik yordamları vasıtasıyla gerçekleştirebilmektedir:

```
void ReadRegister(int8 id, int16 *data) {
 putc(97 + id);
 *data = getc();
 *data <<= 8;
 *data += getc();
}
void WriteRegister(int8 id, int16 *data) {
 putc(65 + id);
 putc((int8)(*data >> 8));
 putc((int8)(*data));
}
```

4.4. Uygulama Sonuçları

Pilot çalışma sonunda, ilgili dönemde dersi almış olan öğrencilere bir anket uygulanarak; ÇDB’nin başarısı sorgulanmıştır. Bu bölümde anket sonuçlarının değerlendirilmesi ve akabinde ÇDB uygulamasının eğitim alanındaki etkilerinden bahsedilecektir.

Uygulamanın yapıldığı dönemde dersi alan 20 öğrenciden 14’ü ankete katılmış olup, bunlardan biri lisans, 9’u yüksek lisans, 4 tanesi ise doktora öğrencisidir. Dersi daha önce alan iki öğrenciden ayrıca söz konusu uygulamanın yapıldığı final projesiyle önceki yıllardaki projeleri karşılaştırılmaları da istenmiştir.

Ankete katılan öğrencilerin büyük bölümü (%78.6) ÇDB uygulamasının derse etkisini çok olumlu olarak değerlendirmiş; 11 öğrenci tasarlanan denetleyicilerin (MATLAB / Simulink gibi) salt sayısal bir benzetim yerine; 3 boyutlu grafiklerle görselleştirilmiş gerçekçi bir benzetim kullanılarak değerlendirilmesinin çok daha akılda kalıcı olduğunu vurgulamışlardır. 10 öğrenci denetleyiciyi geliştirirken böylesi bir benzetimden yararlanmanın hata ayıklama ve denetleyicilerin ince ayarları konusunda iyi bir yönlendirici olduğunu belirtmişlerdir. Öğrencilerin %57.1'lik bir kısmı, uygulamanın final projesi sürecini daha keyifli ve cazip kıldığını düşünmektedir. Dersi daha önce almış olan iki öğrenciden biri ÇDB uygulamasının derse yaklaşımını olumlu olarak etkilediğini belirtmiştir.

Benzer bir uygulamanın öğrencilerin eğitimleri sırasında aşina olduğu bir yazılım paketi olan MATLAB ile gerçekleştirilmesi durumunda öğrencilerin %42.9'u böyle bir uygulamanın daha başarılı olacağını düşünürken, %35.7'si yapılmış uygulamanın daha olumlu olduğunu ifade etmişlerdir. Burada önemle vurgulanması gereken nokta; projenin uygulaması esnasında yazılımda birtakım hatalar ile karşılaşılmış olduğudur. Bu hatalar doğrultusunda öğrenciden yazılım kullanımının iyileştirilmesine yönelik birçok öneri ve geri bildirim gelmiştir. Dolayısıyla, proje uygulanırken, güncellenmiş yazılım sürümleri öğrenciye dinamik olarak sunulmuştur. Bu uygulamanın olumsuz olduğunu düşünen öğrenciler de dahil olmak üzere; 10 öğrenci, hataları ayıklanmış bir Cadmus sürümünün proje başında sağlanması durumunda, fikirlerinin çok daha olumlu yönde değişeceği şeklinde görüş bildirmişlerdir.

Son olarak öğrencilerin büyük bir çoğunluğu (%92.9), ÇDB ve benzeri uygulamaların kontrol tasarımı ve sistem modelleme temelli dersler ve uygulamalarda kullanımının süreci kolaylaştıracağı ve kaynakların daha verimli kullanılmasına olanak sağlayacağını düşünmektedir.

Öğrencilere yapılan anket ve ilgili öğretim üyelerinin değerlendirmeleri sonucu olarak; ÇDB'nin örnek verdiğimiz kontrol dersinde uygulanmasının olumlu tepkiler aldığı ve eğitim alanında uygulanmasının büyük potansiyele sahip olduğu görülmektedir.

5. Tartışma ve Sonuçlar

Kontrol sistemleri üzerine verilen eğitimi iyileştirmek ve verimini arttırmak amacına yönelik olarak, bu makalede bir ÇDB yazılımı tanıtılmıştır. İleride bir paket program haline getirilmesi düşünülen bu yazılım halen geliştirme aşamasında olup, değişikliklere açıktır.

Pilot uygulama, öğrencilerin 2 eksenli bir uydu anteninin kontrol çevrimi (32-bitlik kayar-noktalı aritmetik) işlemlerini 19 ila 25 milisaniye içinde gerçekleştirdiklerini göstermiştir. Bu da doğal olarak, PIC 16F877A yerine daha fazla kaynağa sahip mikrodenetleyicilerin (PIC 18F4520 veya PIC 18F4550 gibi) kullanılması gereğine işaret etmektedir. Bu sayede öğrencilerin çok daha karmaşık sistemler için denetleyiciler geliştirilmelerinin önü açılmış olacaktır. İletişim için Evrensel Seri Veriyolu (USB) kullanımına geçilerek daha büyük bir bantgenişliği ve modern ev bilgisayarlarıyla daha iyi uyumluluk sağlanması hedeflenmektedir. Burada vurgulanması gereken çok önemli bir nokta da geliştirilen Cadmus yazılımının; donanımdan bağımsız olarak çalıştığı gerçeğidir. İlgili seri haberleşme protokolüne uyulduğu sürece, yazılım herhangi bir denetleyici donanımla beraber uyumlu olarak çalışacaktır.

Pilot uygulama için ortaya konmuş olan özelleştirilmiş yazılım üzerinde, benzetimin gerçekleştirme süresini ölçmek amacıyla, çeşitli durum denklemleri ve benzetim elemanları kombinasyonlarıyla denemeler yapılmıştır. Bu denemelere dair ayrıntı ve sonuçlar Tablo 1'de sunulmuştur. Bu tablodaki sonuçlar incelendiğinde, pilot projede geliştirilmiş olan yazılımın bir benzetim döngüsünü 181µs ila 582µs arasında kapatabileceği görülmektedir. Bu bilgiden yola çıkılarak, söz konusu pilot proje bağlamında, örnekleme süresi 1 milisaniye olan bir denetleyici için gerçek zamanlı bir benzetimin yapılması mümkündür. Ancak, bu sonuçlar Cadmus yazılımının nihai sürümüyle ilgili yalnızca bir ön fikir verebilmektedir. Daha yüksek dereceli durum denklemlerinin yahut (bilgisayarlar tarafından hesaplanması külfetli olan) trigonometrik / logaritmik / üstel fonksiyonların kullanımını gerektiren benzetimlerde; söz konusu sürelerin büyük değişiklik göstereceği açıktır. Bu noktada kullanılan programlama dili, yazılım ortamı ve işletim sistemine has yöntemlerin ve eniyileştirmelerin kullanımı araştırılmalıdır. Bunların yanı sıra, gerçek zamanlı icrayı taahhüt eden işletim sistemlerinin ve donanım mimarilerinin de incelenmesi gerekmektedir. Bu konularla ilgili diğer ayrıntılar, bu makalenin kapsamı dışında kaldığından burada yer verilmemektedir.

Tablo 1: Benzetim Süreleriyle ilgili Sonuçlar

	Süreler (milisaniye)			
Bilgisayar	A	B	C	D
1	0.006	0.058	0.096	0.181
2	0.022	0.193	0.347	0.582

A: Benzetim yapılmaz (iletişim ve diğer işlemlerden gelen ek yükün ölçümü)
B: Yalnızca tek eksende anten benzetimi
C: Yalnızca iki eksende anten benzetimi
D: İki eksende anten, uydu ve rüzgar benzetimi

PC 1: Intel Core 2 Duo 3.00GHz 6MB önbellekli işlemci, 2GB bellek, NVIDIA GeForce 9600GT grafik kartı, Microsoft Windows Vista işletim sistemi
PC 2: Intel Core Duo 2.00GHz 2MB önbellekli işlemci, 1GB bellek, NVIDIA GeForce Go 7400 grafik kartı, Microsoft Windows XP işletim sistemi

Yakın bir gelecekte, Cadmus benzetim yazılımının daha modüler bir yapıya kavuşturulması ve istenen herhangi bir dinamik sistem (robot/otomasyon, üretim, havacılık/uzay, otomotiv, savunma, vb.) için biçimlendirilebilmesi hedeflenmektedir. Son halinde yazılım; benzetim, görselleme, kayıtlama ve iletişim işlemlerini yürüten bir *temel yazılım* olarak düzenlenecektir. Bu temele, benzetimi yapılan sistemle ilgili denklemler, değişkenler ve görsel nesneleri içeren *modüller* eklenecektir. Ayrıca, eğitimcinin istediği sistem modüllerini rahatlıkla üreterek öğrencilerine sunmakta kullanabileceği bir *tasarım yazılımı* da bu paketin tümleşik bir parçası olacaktır. Bu yazılım, sistemi tanımlayan değişken ve denklemlerin basit ancak etkili bir programlama dili aracılığıyla belirtilmesine olanak tanıyacaktır. Görsellemenin gerçekleştirilebilmesi için ise basit geometrik şekillerin (prizma, küre, silindir vs.) kullanımının yanı sıra, yaygın bilgisayarlı grafik ve çizim programlarınca benimsenmiş dosya biçimlerinden modellerin alınması da mümkün olacaktır. Böylece eğitmen, halihazırda sahip olduğu (veya çeşitli kaynaklardan edinebileceği) modellerin yanı sıra, kendi yaratacağı modelleri de görsellemede kullanabilecektir. Bu modellerin çeşitli özelliklerinin (konum, büyüklük vb.) yine programlama dili aracılığıyla sistem değişkenlerine bağlanmasıyla sistemin görselleştirilmesi gerçekleştirilmiş olacaktır.

Cadmus üzerinde gerçekleştirilmesi olası bir diğer değişiklik, yazılımın desteklediği işletim sistemlerinin çeşitlendirilmesidir. Yaygın olarak kullanılan Linux ve UNIX tabanlı işletim sistemlerinin desteklenmesi, bu işletim sistemlerini tercih eden öğrencilerin de bu yazılımı kullanabilmelerini sağlayacaktır. Bu durumda halihazırda var olan .NET Framework ihtiyacının ortadan kaldırılması ve görselleştirme için farklı grafik kütüphanelerinin kullanımına geçilmesi gerekecektir.

Bu makalede sunulmuş olan Çevrimiçi Donanım Benzetimi ve Evdeki Laboratuvar yöntemi sayesinde öğrenciler gerçek bir laboratuvarın mekansal ve zamansal kısıtlamalarından kurtarılarak tamamen kendi kullanımlarına ayrılmış, sanal bir laboratuvara bireysel olarak sahip olabileceklerdir. Bu yolla öğrencilerin kontrol üzerine almış olduğu eğitimin pekiştirilmesi hiç şüphesiz ki kolaylaşacaktır. Denetleyici tasarımı sırasında yapılması olası hatalar (yanlış denetleyici tasarımı, uygun olmayan parametre seçimi, ölçeklendirme yanlışları, vs) sonucunda sisteme ve/veya insanlara zarar verme olasılığı kalkacaktır. Böylece, bu gibi durumlarda laboratuvar çalışmalarının sekteye uğramasının da önüne geçilecektir. Gerçek bir sistemin satın alınması, kullanıma hazırlanması ve bakımının yanı sıra olası tamir ihtiyaçları da böylece ortadan kaldırılmış olacağından eğitim kurumunun üzerinden önemli bir mali yük kaldırılmış olacaktır. Bütün bu olumlu gelişmeler, kontrol üzerine verilen derslerin verimliliğini ve öğrencilerin bu derslerden sağladığı faydayı arttıracaktır.

6. Teşekkür

Bu makalenin yazarları; sunulan yöntemin geliştirilmesinde ve uygulanmasında emeği geçen herkese, özellikle de 2007-2008 yılı bahar döneminde “ME 534 – Computer Control of Machines” dersini alan ODTÜ Makina Mühendisliği Bölümü öğrencilerine içtenlikle teşekkür ederler.

7. Kaynakça

[1] W. Grega, “Hardware-in-the-loop simulation and its application in control education”, 29th ASEE/IEEE Frontiers in Education Conference, San Juan, Puerto Rico, s:12b6 7-12b6 12, 1999.

[2] M. Bacic, ”On hardware-in-the-loop simulation”, Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference, Seville, Spain, s:3194-3198, 2005.

[3] M. E. Sisle and E. D. McCarthy, "Hardware-In-The-Loop Simulation For An Active Missile", Simulation, Cilt: 39, Sayfa: 159-167, 1982.

[4] M. Bailey and J. Doerr, "Contributions of hardware-in-the-loop simulations to Navy test and evaluation", Proceedings Of The Society Of Photo-Optical Instrumentation Engineers (SPIE), Cilt: 2741, Sayfa: 33-43, 1996.

[5] H. Eguchi and T. Yamashita, "Benefits of HWIL simulation to develop guidance and control systems for missiles", in Technologies For Synthetic Environments: Hardware-In The-Loop Testing V, Cilt: 4027, Sayfa: 66-73, Proceedings Of The Society Of Photo-Optical Instrumentation Engineers (SPIE), 2000.

[6] J. S. Cole and A. C. Jolly, "Hardware-in-the-loop simulation at the US Army Missile Command", in Technologies For Synthetic Environments: Hardware-In-The-Loop Testing, Cilt: 2741,Sayfa: 14-19 Proceedings Of The Society Of Photo-Optical Instrumentation Engineers (SPIE), 1996.

[7] R. B. Wells, J. Fisher, Y. Zhou, B. K. Johnson, M. Kyte, “Hardware and Software Considerations for Implementing Hardware-in-the-Loop Traffic Simulation”, Industrial Electronics Society, 27th Annual Conference of IEEE, Denver, Colorado, A.B.D. s:1915-1919, 2001.

[8] D. Bullock, B. Johnson, Richard B. Wells, Micheal Kyte, Zhen Li, ”Hardware-in-the-loop simulation”, Burdock et.al Transportational Research Part C, 2004.

[9] J.R Kendall, R.P. Jones, “An investigation into the use of hardware-in-the-loop simulation testing for automotive electronic control system”, Elsevier Science Ltd. Control Engineering Practice, Cilt: 7, Sayfa: 1343-1356, 1999.

[10] J. de Carufel, E. Martin, ve J.-C.Piedboeuf, “Control strategies for hardware-in-the-loop simulation of flexible space robots”, IEE Proc.- Control Theory Applications, Cilt: 147, No: 6, s:569-579 2000.

[11] Z. Papp, M. Dorrepaal ve D.J. Verburg, “Distributed Hardware-in-the-loop simulator for autonomous continuous dynamical systems with spatially constrained interactions”, Proceedings of IEEE International Parallel and Distributed Processing Symposium, Nice, France, s:119.1, 2003.

[12] Y. Tarte, Y. Chen, W. Ren ve K. Moore, “Fractional Horsepower Dynamometer – A General Purpose Hardware-In-The-Loop Real-Time Simulation Platform for Nonlinear Control Research and Education”, Proceedings of the 45th IEEE Conference on Decision & Control, San Diego, CA, A.B.D., s:3912-3917, 2006.

[13] P. S. Shiakolas ve D. Piyabongkarn, “Development of a Real-Time Digital Control System with a Hardware-in-the-Loop Magnetic Levitation Device for Reinforcement of Controls Education”, IEEE Transactions on Education, Cilt: 46, No: 1, 2003.

[14] M. Grimheden ve M. Törngren, “How should embedded systems be taught? Experiences and snapshots from Swedish higher engineering education”, ACM SIGBED Review, Cilt: 2, No: 4, s:34-39, 2005.

[15] I. A. Drumm, “The Application of Adaptive Beam Tracing and Managed DirectX for the Visualisation and Auralisation of Virtual Environments”, Proceedings of the Ninth International Conference on Information Visualisation (IV’05), Londra, İngiltere, s:961-965, 2005.

[16] Microsoft Developer Network, http://msdn.microsoft.com

[17] I. Gorton, J. Kerridge, B. Jervis, “Simulating microprocessor systems using occam and a network of transputers”, IEEE Proc.- Cilt: 136, No: 1, 1989.

[18] Y.L. Lu, C.T. Lin, C.F. Wu, S.A. Hwang ve Y. H. Lin, “Microprocessor Modeling and Simulation with SystemC”, IEEE Proc., 2007.

[19] W.H. Press, S:A: Taukosky, W.T. Vattening, B.R. Ronnery, “Numerical Recipes in C: The Art of Scientific Computation”, 1992.

[20] Custom Computer Services Inc., http://www.ccsinfo.com

Kayan Kipli DC Motor Konum Denetiminin FPGA ile Gerçekleştirilmesi

Barış Ragıp Mutlu[1,2], *Ulaş Yaman*[2], *Melik Dölen*[2], *A. Buğra Koku*[2]

[1]Makina Mühendisliği Bölümü
Hacettepe Üniversitesi, Ankara
brmutlu@hacettepe.edu.tr

[2]Makina Mühendisliği Bölümü
Orta Doğu Teknik Üniversitesi, Ankara
uyaman@metu.edu.tr
dolen@metu.edu.tr
kbugra@metu.edu.tr

Özetçe

Bu makalede bir DC servo motor konum kontrolu ***alan programlanabilir kapı dizini*** (Field programmable gate array - FPGA) tabanlı kayan kip denetleyici ile gerçekleştirilmiştir. Öncelikle konum denetimi yapılacak olan motor modellenmiştir. Ardından tasarlanan denetleyici FPGA kırmığına üzerinde sentezlenmiştir. Kayan kip denetleyicisinin başarımını değerlendirmek amacıyla aynı sistem için ek olarak PID ve komut ileri beslemeli denetleyiciler tasarlanmış olup, bunların komut takip edebilme yetilerine ve FPGA kırmığı üzerinde kullandıkları kaynakların miktarlarına göre karşılaştırılmışlardır.

1. Giriş

Kayan kipli denetim, tahmin edilemeyen değişikliklere karşı olan gürbüzlüğü nedeniyle elektromekanik sistemlerin denetimi için yıllarca kullanılmıştır ve halen de kullanılmaktadır. Kayan kipli denetimin en ayırıcı özelliği, durumdan duruma geçmesinden dolayı, çıkış denetim işaretinin süreksiz olmasıdır. Getirilerinin yanı sıra, kayan kipli denetimin dezavantajları da mevcuttur. En temel sorun olarak çıkış işaretinin yüksek sıklıklarda değişmesi gösterilebilir. Diğer sorun ise 20 kHz ve üzerindeki frekanslarda güç kaynaklarının durumlarının değiştirilmesiyle ortaya çıkan çınlamadır. Günümüz teknolojisinde 150 kHz ve üzeri frekanslarda durumu değiştirilebilen güç kaynakları mevcuttur fakat genel olarak sayısal işaret işlemcileri bu tip çalışma frekanslarını desteklememektedirler. Bu güç kaynaklarının getirilerinden mümkün olduğunca yararlanabilmek için FPGA gibi yüksek frekans-bantlarında çalışma olanağı sağlayan sayısal ürünlere ihtiyaç duyulmaktadır. FPGA kullanımıyla birlikte, çınlama sorununda azalmalar görülmektedir [1].

Son yıllarda çeşitli denetim topolojilerinin donanımsal olarak uygulanması üzerine olan araştırmalar artmaktadır. Dış belirsizlik ve gürültülere karşı duyarsız olması kayan kipli denetimin diğer denetim algoritmaları arasında popüler olmasını sağlamıştır.

Kayan kipli denetimin FPGA ile gerçekleştirilmesi sırasında çeşitli geliştirmeler yapılmıştır. Bunlara örnek olarak, bulanık kayan kipli denetim [5] ve uyarlanabilir geri adımlı kayan kipli denetim [3] gösterilebilir. Bu makalelerde tasarlanan denetleyiciler, bir indüksiyon motorunun konum denetimini yerine getirmektedirler. Kayan kipli denetleyicilerin tasarlanmasının temel nedeni sürtünme kuvveti de dahil olmak üzere sistemdeki belirsizlikleri telafi etmektir. Kullanılan bulanık kayan kipli denetleyici [5] sistemdeki toplu belirsizliğin üst sınırını kestirmektedir. Tasarlanan bu denetleyici ile sistem belirsizliklere rağmen peryodik referans yörüngelerini takip edebilmektedir. Diğer makalede [3] ise toplu belirsizliklerin değerini gerçek zamanlı olarak uyarlamak için uyarlanabilir bir yasa elde edilmiş ve böylece uyarlanabilir geri adımlı kayan kipli denetim sonuçlandırılmıştır. Daha sonra bu iki denetleyici sistemin başarımı FPGA ile gerçekleştirilen deneyler sonucunda doğrulanmıştır.

Kayan kipli denetime yapılan diğer bir değişiklik ise efendi-hizmetli tipi kayan kipli denetimdir. Tasarlanan bu denetleyici, N adet koşut bağlanmış tezgah tabanlı tek çeviricilerden oluşan birimsel bir sisteme FPGA aracılığıyla yerleştirilmiştir [2]. Böylece çıkış gerilimi düzenlenmesi ve çeviriciler arasındaki dengeli akım dağıtımı sağlanmış olmaktadır. Öte yandan genel çevirici sistemin kayma alanı her bir çeviricinin çıkış süzgeçlerinin aktarım fonksiyonu cinsinden ifade edilmiş olur.

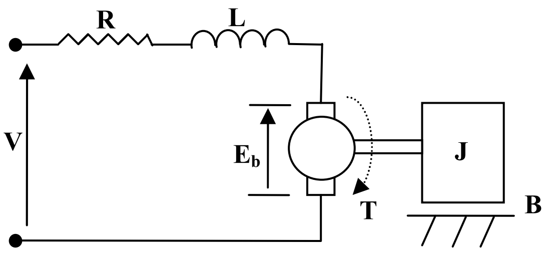
Günümüz FPGA kırmık derleyicileri göz önüne alındığında, matematiksel işlemlerin FPGA ile gerçekleştirmenin oldukça zor olduğu görülmektedir. Denetim algoritmaları matematiksel hesaplamalar içerdiği için bir matematiksel ünitenin FPGA kırmığında bulunması gerekliliği kaçınılmazdır. Raygoza ve diğerleri [1] tarafından, sinüs ve kosinüs gibi farklı matematiksel fonksiyonları hesaplayan bir FPGA ünitesi tasarlanmıştır. Tasarlanan bu ünitenin en temel özelliği aritmetik ve trigonometrik işlemleri bir saat çevriminde gerçekleştirebilmesidir. Daha sonra bu ünite manyetik kaldırma sistemi için tasarlanmış olan bir kayan kipli denetleyicide kullanılmıştır. Denetleyici, sistemin nominal durumlarda çalışırken gerekli hız ve kesinlik ihtiyaçlarını başarıyla yerine getirmiştir.

Kayan kipli denetimin zamanda ayrık sürümü ise diğer bir makalede [4] FPGA üzerinde gerçekleştirilmiş ve test edilmiştir. FPGA kullanılmasının temel nedeni geniş bant aralığına sahip elektriksel gücün yüksek durum değiştirme sıklıklarına ihtiyaç duymasıdır. Makalede de belirtildiği gibi FPGA kullanmak denetim algoritmasının gerçekleştirilmesine zaman anlamında büyük getiriler sağlamaktadır. Böyle bir katkıyı mikrodenetleyicilerden beklemek oldukça zordur. Durum değiştirme kararının her adımda alındığı -kayan kipli denetim örneğindeki gibi- bir durumda bu katkı önem arz etmektedir.

Bu makalede, bir DC servo motor için tasarlanmış olan kayan kipli denetleyicinin FPGA üzerinde gerçekleştirilmesi anlatılmaktadır. Makalenin ikinci kısmında DC servo motor modellenmesine yer verilmiştir. Üçüncü kısımda ise denetleyicilerin sınandığı düzenek ayrıntılı bir şekilde anlatılmıştır. Makalenin dördüncü kısmında kayan kipli denetleyicinin tasarımı anlatılmış, beşinci kısımda ise kayan kipli denetleyicinin ve karşılaştırma amacıyla geliştirilen PID denetleyicinin FPGA aracılığıyla nasıl gerçekleştirildiğinden bahsedilmiştir. Elde edilen sonuçlar ise makalenin altıncı kısmında tartışılmıştır.

2. Sistemin Modellenmesi

Servo motorlar endüstride yaygın olarak kullanılmaktadır. Bu nedenle motorların konum denetimi yüksek doğruluk gerektirmektedir. Çalışma sırasında denetimi yapılan DC servo motorun yapısı Şekil 1'de gösterilmiştir. Sisteme verilen referans komutu motor milinin açısal konumunu belirlemektedir. Tasarlanan denetleyicilerin amacı sistem çıkışı ve referans konumları arasındaki hatayı asgari seviyeye indirmektir.



Şekil 1: DC Servo Motor Modeli

DC servo motor dinamiği aşağıdaki denklemlerle elde edilebilmektedir:

$$V = RI + L\frac{dI}{dt} + E_b \qquad (1)$$

$$T = J\frac{d^2\theta}{dt^2} + B\frac{d\theta}{dt} \qquad (2)$$

$$T = K_T I \qquad (3)$$

$$E_b = K_b\frac{d\theta}{dt} \qquad (4)$$

Denklemlerde kullanılan fiziksel parametreler aşağıdaki gibidir:

V : Giriş Gerilimi [V]
E_b : Geri Elektromotor Kuvveti Gerilimi [V]
I : Armatür Akımı [Amp]
L : Armatür Endüktansı [H]
R : Armatür Direnci [Ω]
J : Rotor Eylemsizlik Momenti [kgm^2]
T : Motor Torku [Nm]
Θ : Mil Konumu [rad]
B : Viskoz Sönüm Katsayısı [Nms]
K_T : Tork Sabiti [Nm/A]
K_b : Hız Sabiti [Vs/rad]

Uluslararası birim sistemi birimleri ile hız ve tork sabitlerinin birbirlerine eşit oldukları bilinmektedir. Bu eşitlik Tablo 2'deki hız ve tork sabitinin birimleri eşitlenerek de görülebilir. Dolayısıyla;

$$K_b = K_T = K \qquad (5)$$

(1) ve (5) arasındaki denklemler kullanılarak sistemin durum uzayı denklemleri elde edilmiştir:

$$\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\\ \dot{\theta}_3\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 0 & \frac{-RB-K^2}{LJ} & \frac{-B}{J}-\frac{R}{L}\end{bmatrix}\begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\end{bmatrix} + \begin{bmatrix}0\\ 0\\ \frac{K}{JL}\end{bmatrix} V \qquad (6)$$

Durum değişkenleri olarak milin konumu, hızı ve ivmesi alınmıştır.

$$\theta_1 = \theta \qquad (7)$$

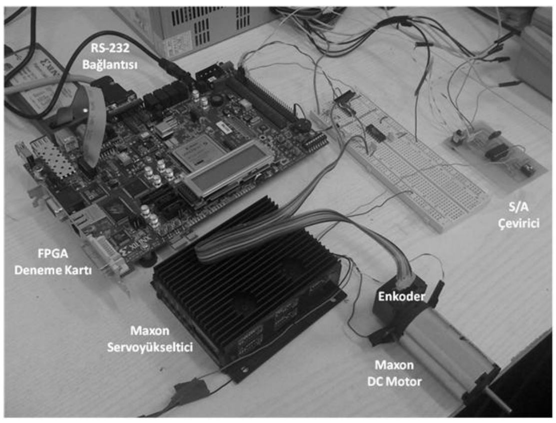
$$\theta_2 = \omega \qquad (8)$$

$$\theta_3 = \alpha \qquad (9)$$

Sistemde servoyükseltici giriş olarak sayısal/analog çeviriciden aldığı ±10 V gerilimi birebir motora besleyecek bir gerilim kaynağı olarak kullanılmaktadır.

3. Deney Düzeneği

Deneyler bir DC Servo motor üzerinde yapılmıştır. Düzenek Şekil 2'de gösterilmiştir. Şekil 2'den de görüldüğü üzere, düzenek Virtex-5 FPGA kırmığına sahip bir ML-505 geliştirme kartı, bir sayısal analog çevirici, TTL-CMOS ve CMOS-TTL düzey kaydırma devresi, bir Maxon LSC 30/2 servoyükseltici ve bir Maxon A-max DC motordan oluşmaktadır. Bu elemanların yanı sıra düzenekte servoyükselticiyi besleyen bir güç kaynağı, mantıksal düzey kaydırma ve sayısal/analog çeviriciyi besleyen başka bir güç kaynağı da bulunmaktadır. Düzenekteki servoyükselticinin ve motorun özellikleri Tablo 1 ve Tablo 2'de sunulmuştur.



Şekil 2: Deney Düzeneği

Tablo 1 – Maxon LSC 30/2 Servoyülseltici'nin Özellikleri

Besleme gerilimi	12-30 VDC
Maksimum çıkış gerilimi	25 V
Maksimum çıkış akımı	2 A
Maksimum çıkış gücü	50 W

Tablo 2 – Maxon A-max 32 DC Motor'un Özellikleri

Armatür direnci	7.19 Ω
Armatür endüktansı	1.04 mH
Tork sabiti	38.2 mNm / A
Hız sabiti	250 rpm / V
Sönüm oranı	2.027 mNm.s
Rotor eylemsizliği	44.0 gcm^2

Virtex-5 FPGA kırmığında üretilen düzeltici işaret, ML-505 üzerindeki çıkış bacaklarına kullanılarak sayısal analog çeviriciye beslenmektedir. Bu çevirici, gelen sayısal işareti $\pm$10V aralığında değişen bir gerilime dönüştürmektedir. Bu gerilim, servoyükselticiye ayar gerilimi olarak beslenmekte ve servoyükseltici de gelen bu gerilimin seviyesine göre motoru sürmektedir. Motorun pozisyon bilgisi artımlı enkoder işaretleri ile düzey kaydırma devresine bağlanmakta ve TTL'den CMOS düzeyine düşürülmektedir. CMOS düzeyindeki bu işaretler ML-505 geliştirme kartının iki bacağına bağlanmaktadır. Deney tamamlanana kadar veriler FPGA içerisinde saklanmakta, deney sonlandıktan sonra ise ML-505 geliştirme kartı üzerindeki RS-232 portu aracılığıyla bilgisayara aktarılmaktadır.

4. Kayan Kipli Denetleyici Tasarımı

Bu bölümde deney düzeneğinde yer alan Maxon DC servo motor için tasarlanan kayan kipli denetim anlatılmıştır. Sistemin derecesi üç (*n =3*) olduğundan dolayı kayma yüzeyinin aşağıdaki biçimde olması gerekmektedir [6].

$$s = \left(\frac{d}{dt} + \lambda\right)^2 (\theta - \theta_r) \qquad (10)$$

Denklemdeki θ_r milin referans konumunu temsil etmektedir. λ ise kayan kipli denetleyicinin parametrelerinden birisidir. Denklem (10) genişletildiği takdirde kayma yüzeyinin son denklemi aşağıdaki gibi olmaktadır.

$$s = (\ddot{\theta} - \ddot{\theta}_r) + 2\lambda(\dot{\theta} - \dot{\theta}_r) + \lambda^2(\theta - \theta_r) \qquad (11)$$

Durum yörüngelerinin yüzeyde kayabilmesi için denklem (11) ve türevinin sıfıra eşit olması gerekmektedir.

$$\dot{s} = s = 0 \qquad (12)$$

Denklem (12)'de belirtilen koşulları sağlayan eşdeğer gerilimi gerekli sadeleştirmelerden sonra aşağıdaki denklem ile ifade edilebilir.

$$V = \dot{\theta}\left(\frac{RB + K^2 - \lambda^2 LJ}{K}\right) + \ddot{\theta}_r\left(\frac{2\lambda LJ}{K}\right) + \dot{\theta}_r\left(\frac{\lambda^2 LJ}{K}\right) - K_s sat\left(\frac{s}{\phi}\right) \qquad (13)$$

Motorun konum denetimini yapmak için gerekli olan denetim giriş işareti literatürde [6]

$$V = V_{eş} - K_s sat(\tfrac{s}{\emptyset}) \qquad (14)$$

şeklinde ifade edilmektedir. (14)'te kullanılan K_s, maksimum denetleyici çıkış işaretini temsil eden bir denetleyici parametresidir. ϕ ise çınlamayı gidermek için kayma yüzeyi etrafında bulunan sınır katmanının kalınlığını belirtmektedir [6].

Geliştirilen kayan kipli denetleyicinin parametreleri elle ayarlanmıştır. Yapılan MATLAB – Simulink benzetimlerinde tasarlanan denetleyicinin bant genişliği 10 Hz civarında bulunmuştur. Denetleyici parametreleri ise Tablo 3'teki gibi belirlenmiştir;

Tablo 3 – Kayan kipli denetleyici parametreleri

λ	K_s	$\emptyset$
500	5	0.01

5. Denetleyicilerin FPGA ile Gerçekleştirilmesi

Denetleyicilerin FPGA üzerinde gerçekleştirilmesi için, Verilog HDL donanım tanımlama dili (hardware description language) kullanılarak çeşitli modüller geliştirilmiştir. Bu modüller arasında PID, komut ileri beslemeli ve kayan kipli denetleyici modüller olduğu gibi artımlı enkoderden gelen pozisyon bilgisini çözümleyen bir quadrature enkoder çözümleyici modülü, sayısal / analog çeviriciye girecek olan darbe genişlik kiplenimi işaretini üreten PWM üretici modül, sinüsoid referans işaretini üreten bir referans modülü ve gelen açısal pozisyon bilgilerinden açısal hız hesabı yapan bir hız hesaplama modülü de bulunmaktadır. Bu modüllerin yanı sıra hesaplamaları gerçekleştirmek için R. Usselmann tarafından geliştirilen, 32-bit kayan nokta hesaplamaları ve tamsayı/kayan nokta çevrimleri yapabilen bir kayan noktalı hesaplama modülü de kullanılmıştır [7]. Verilog HDL ile geliştirilmiş bir quadrature enkoder çözümleyici modülünün kodu, Ek-1 olarak makalenin sonunda sunulmuştur.

5.1. PID Denetleyici

FPGA üzerinde denetim gerçekleştirmek için öncelikle denetimcinin tasarlanması, sonra da sonlu fark denklemlerinin elde edilmesi gerekmektedir. Geliştirilen PID algoritmasından sonlu fark denklemleri elde edildiği zaman, ortaya çıkan denklem şu şekildedir;

$$m(k) = m(k-1) + b_0 e(k) + b_1 e(k-1) + b_2 e(k-2) \qquad (15)$$

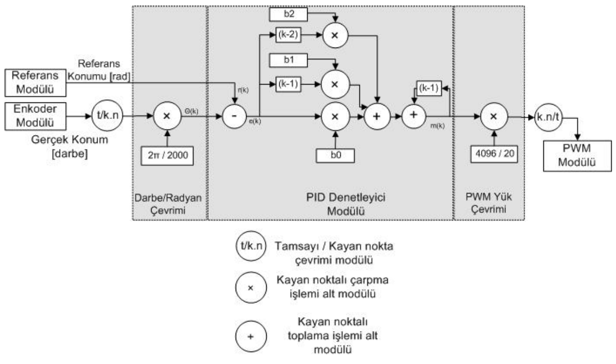
Burada, m(k) denetleyici algoritmanın ürettiği düzeltici işareti; e(k) hata işaretini; b_0, b_1 ve b_2 ilgili sonlu fark denkleminin sabitlerini; k ise zaman indisini göstermektedir. Bu denklemdeki sabitler, PID denetleyicinin katsayılarıyla aşağıdaki şekilde hesaplanmaktadır;

$$b_0 = K_P + \frac{K_d}{T} + K_i T \quad (16)$$

$$b_1 = -K_P - \frac{2K_d}{T} \quad (17)$$

$$b_2 = \frac{K_d}{T} \quad (18)$$

Bu işaretler, FPGA üzerinde 32-bit kayan noktalı sayı olarak temsil edilmektedir. Hesaplamalar ise bu bölümün başında bahsedilen kayan noktalı hesaplama modülü yardımıyla gerçekleştirilmektedir. PID denetleyicinin FPGA üzerindeki uygulaması Şekil 3'te gösterilmiştir.



Şekil 3: PID denetleyici uygulaması

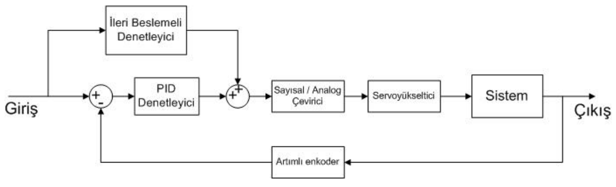
Geliştirilen PID denetleyicinin bant genişliği 350 Hz olarak belirlenmiştir. Denetleyici parametreleri ise Tablo 4'teki gibi belirlenmiştir;

Tablo 4 – PID denetleyici parametreleri

K_p	K_d	K_i
40	0.4	5000

5.2. Komut İleri Beslemeli Denetleyici

Bu uygulamada konum yörüngesi önceden belirli olduğu için, sistemde komut ileri besleme kullanmak mümkündür. Dolayısıyla, karşılaştırma açısından komut ileri beslemeli bir denetleyici de geliştirilmiştir. Komut ileri beslemeli denetleyici sistemi Şekil 4'te gösterilmiştir.

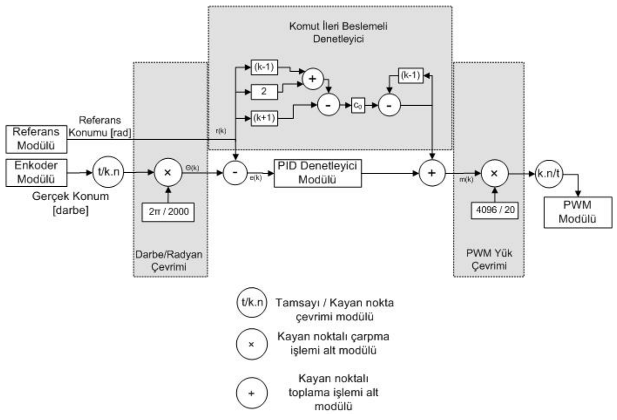


Şekil 4: Kayma yüzeyi hesaplama modülü

Geliştirilen ileri beslemeli denetleyicinin sonlu fark denklemi ise şu şekilde hesaplanmıştır;

$$m(k) = -m(k-1) + c_0(r(k+1) - 2r(k) + r(k-1)) \quad (19)$$

Bu denklemdeki parametreler (15) ile aynı olup, denklemde komut ileri beslemeden gelen bir $r(k+1)$ terimi bulunmaktadır. Komut ileri beslemeli denetleyicinin FPGA üzerindeki uygulaması Şekil 5'te gösterilmiştir.



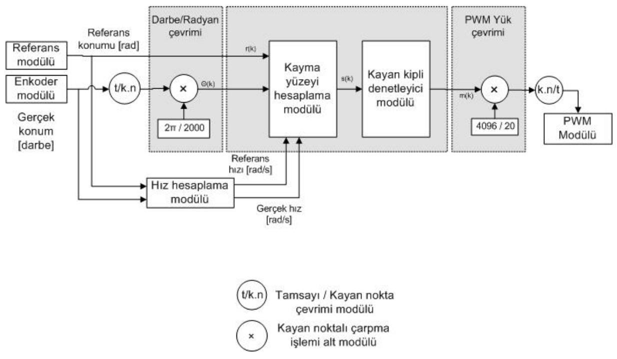
Şekil 5: Komut İleri Beslemeli Denetleyici uygulaması

5.3. Kayan Kipli Denetleyici

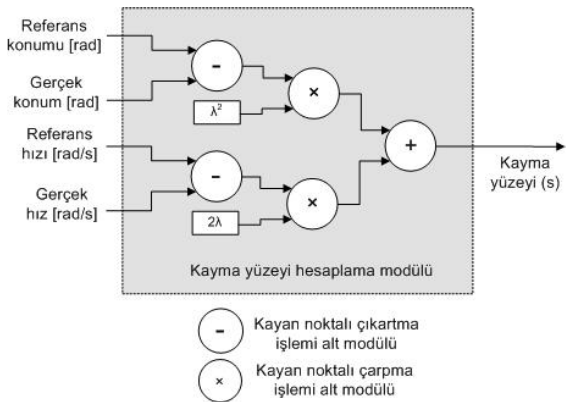
Kayan kipli denetleyici tasarımı, makalenin 4. bölümünde anlatılmıştır. Öte yandan, benzetimlerde (11) no'lu denklemdeki ivme terimlerinin kayma yüzeyine katkısının oldukça az olduğu görülmüştür. Dolayısıyla, geliştirilen denetleyici tasarlanandan daha basit bir biçimde uygulanmıştır ve (11) no'lu denklem şu şekilde basitleştirilmiştir:

$$s = 2\lambda(\dot{\theta} - \dot{\theta}_r) + \lambda^2(\theta - \theta_r) \quad (20)$$

Bu formdaki kayan kipli denetleyicinin FPGA üzerindeki uygulaması Şekil 5.a'da verilmiştir. Şekil 5.b'de ise kayma yüzeyi hesaplama modülünün detayı sunulmuştur.



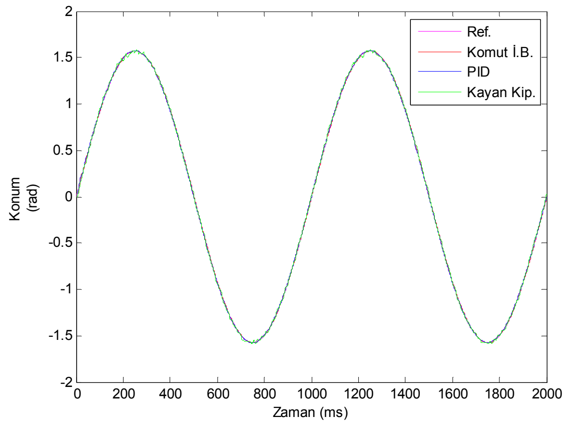
Şekil 6.a: Kayan kipli denetleyici uygulaması



Şekil 6.b: Kayma yüzeyi hesaplama modülü

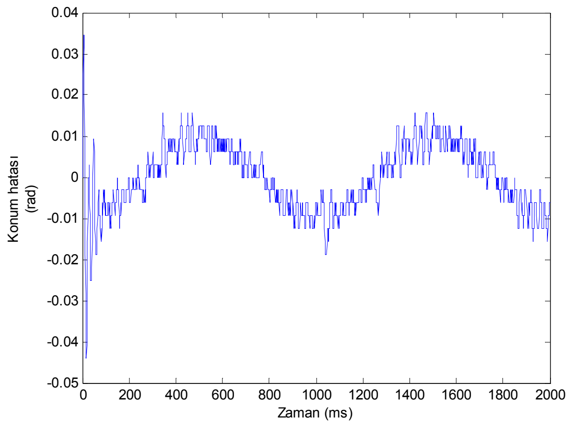
6. Tartışma ve Sonuçlar

Bu kısımda, makalenin üçüncü bölümünde anlatılan deney düzeneği ile gerçekleştirilen deneylerin sonuçları verilmiştir. Deneylerde, beşinci bölümde anlatılan PID, komut ileri beslemeli ve kayan kipli denetleyicilere referans yörüngesi olarak 1 Hz frekansta bir sinüsoid dalga üretilmiştir. Referans yörüngesi ve beşinci kısımda anlatılan denetleyicilerin konum takip başarımı Şekil 7'de sunulmuştur.

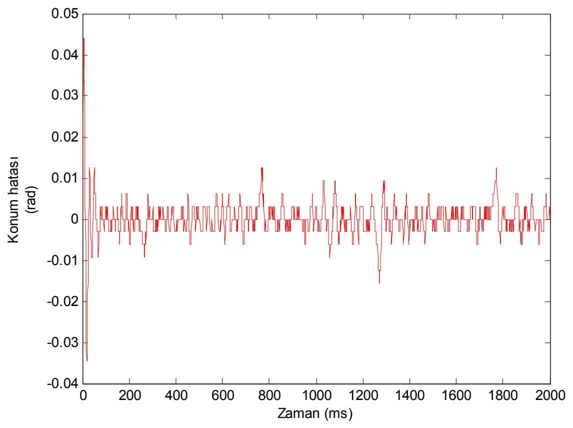


Şekil 7: Referans yörüngesi ve denetleyicilerin başarımı

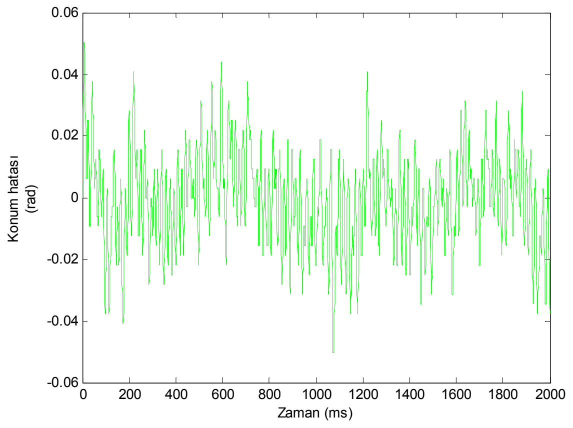
Denetleyicilerin takip hataları ise PID, komut ileri beslemeli ve kayan kipli denetleyici sırasıyla Şekil 8, 9 ve 10'da sunulmuştur.



Şekil 8: PID denetleyicinin konum hatası – zaman grafiği



Şekil 9: Komut ileri beslemeli denetleyicinin konum hatası – zaman grafiği



Şekil 10: Kayan kipli denetleyicinin konum hatası – zaman grafiği

Denetleyicilerin başarımı hakkındaki önemli bilgiler Tablo 4'te derlenmiştir.

Tablo 4 – PID, Komut ileri beslemeli ve Kayan kipli denetleyicilerin başarımı

	PID	Komut ileri besleme	Kayan kipli
Ortalama Hata [rad] (Daimi Rejim)	0.00645	**0.00228**	0.0120
Azami Hata [rad] (Daimi Rejim)	0.0188	**0.0157**	0.0502

Denetleyicilerin FPGA kırmığı üzerinde harcadıkları kaynaklar ve minimum döngü zamanarı ise Tablo 5'te verilmiştir.

Tablo 5 – PID, Komut ileri beslemeli ve Kayan kipli denetleyicilerin FPGA üzerinde harcadığı kaynak miktarları

	PID	Komut ileri besleme	Kayan kipli
Yazmaç Sayısı	**2476**	3813	3496
LUT Sayısı	**13140**	18868	19277
Asgari döngü zamanı	**160ns**	200ns	370ns

Şekil 8, 9 ve 10'da ve Tablo 3'te görüldüğü üzere, kayan kipli denetleyicinin yaptığı konum hatası, PID ve komut ileri beslemeli denetleyiciye oranla daha fazladır. Bunun yanısıra, Tablo 4'te görüldüğü üzere kayan kipli denetleyici, PID denetleyiciye oranla yaklaşık %50 daha fazla kaynak harcamıştır. Komut ileri beslemeli denetleyici ile ise yakın mertebededir. Bu, kayan kipli ve komut ileri beslemeli denetleyicilerdeki hesaplama sayısının çokluğu nedeniyle beklenen bir sonuçtur. Aynı sebeple, asgari döngü zamanı da PID denetleyiciye oranla daha yüksektir. Komut ileri beslemeli denetleyici ise PID'den daha fazla kaynak harcamasına karşın, en başarılı yörünge takibi performansını göstermiştir.

Bu çalışmada, kayan kipli denetleyici FPGA üzerinde başarıyla uygulanmış ve DC motor denetimi için tatmin edici sonuçlar elde edilmiştir. Alınan sonuçlara göre, kayan kipli denetleyiciyi PID denetleyiciye oranla %50 oranında fazla kaynak harcayarak modern bir FPGA kırmığı üzerinde uygulamak mümkündür. Daha fazla kaynak harcamasına karşın, bozucu etkenlerin etkin olduğu sistemlerde kayan kipli denetleyicinin daha sağlıklı bir karşılaştırmasını yapmak mümkündür.

7. Teşekkür

Bu çalışma, TÜBİTAK'ın 108E048 kodlu projesi kapsamında yapılmıştır. Makalenin yazarları sağladığı mali destekten ötürü TÜBİTAK'a teşekkür ederler.

8. Kaynakça

[1] J.J. Raygoza-Panduro, S. Ortega-Cisneros, J. Rivera ve A. De la Mora, "Design of a Mathematical Unit in FPGAs for the Implementation of the Control of a System of Magnetic Levitation," Proceedings of 4th Southern Conference on Programmable Logic, s: 37-42, 2008.

[2] R. Ramos, D. Biel, F. Guinjoan ve E. Fossas, "Master-slave sliding-mode control design in parallel-connected inverters," *Automatika*, Cilt: 42, No: 1-2, s:37-44, 2001.

[3] F. J. Lin, C. K. Chang ve P. K. Huang, "FPGA-Based Adaptive Backstepping Sliding-Mode Control for Linear Induction Motor Drive," *IEEE Transactions on Power Electronics*, Cilt: 22, s:1222-1231, 2007.

[4] S. Lentijo, S. Pytel, A. Monti, J. Hudgins, E. Santi ve G. Simin, "FPGA based sliding mode control for high frequency power converters," Proceedings of 35th Power Electronics Specialists Conference, Cilt:5, s:3588-3592, 2004.

[5] F. J. Lin, C. K. Chang ve P. K. Huang, "FPGA-Based Fuzzy Sliding-Mode Control for a Linear Induction Motor Drive," IEE Proc.-Electr. Power Appl., Cilt: 152, No: 5, s:1137-1148, 2007.

[6] J.-J.E. Slotine ve L. Weiping, *Applied Nonlinear Control*, Prentice Hall, Englewood Cliffs, N.J., 1991.

[7] R. Usselmann, *Open Floating Unit Manual*, www.opencores.org, 2000.

9. Ekler

Ek 1 : Quadrature enkoder modülü

```
module encoder(clk, inA, inB, out_count,
    reset);

input clk;
input reset;
input inA;
input inB;
output[31:0] out_count;

reg [2:0] quadA_delayed, quadB_delayed;
always @(posedge clk) quadA_delayed <=
    {quadA_delayed[1:0], inA};
always @(posedge clk) quadB_delayed <=
    {quadB_delayed[1:0], inB};

wire count_enable = quadA_delayed[1] ^
    quadA_delayed[2] ^ quadB_delayed[1] ^
    quadB_delayed[2];
wire count_direction = quadA_delayed[1] ^
    quadB_delayed[2];

integer count;
always @(posedge clk)
begin
if(~reset)
begin
  if(count_enable)
  begin
    if(count_direction) count<=count+1;
    else count<=count-1;
  end
end
else count<=0;
end

assign out_count = count;

endmodule
```

Bir Doğru Akım Motorunun FPGA Üzerinde Gerçek Zamanlı Benzetiminin Gerçekleştirilmesi

Serdar Üşenmez[1], *Rasim Aşkın Dilan*[2], *Melik Dölen*[3], *A. Buğra Koku*[4]

Makina Mühendisliği Bölümü
Orta Doğu Teknik Üniversitesi, Ankara
[1]serdar.usenmez@gmail.com
[2]adilan@metu.edu.tr
[3] dolen@metu.edu.tr
[4] kbugra@metu.edu.tr

Özetçe

Bu makalede bir doğru akım (DC) motorunun alan-programlanabilir kapı dizini (“Field Programmable Gate Array”, FPGA) kırmığı üzerinde, çevrimiçi donanım benzetiminde kullanılabilmesi amacıyla gerçek zamanlı benzetiminin gerçekleştirilmesi incelenmiştir. Söz konusu motorun matematiksel modeli, cebirsel yöntemlerle sabit katsayılı sonlu fark denklemine dönüştürülür. Bu denklemin çözümü FPGA kırmığı üzerinde yüksek hızlı olarak gerçekleştirilir. Çözüm esnasında motora ait durum değişkenleri (hız ve konum), kırmıkla iletişim halinde olan bir kişisel bilgisayarda görüntülenir. Geliştirilen uygulamanın uygun bir denetleyiciyle birleştirilmesi halinde bu denetleyicinin çevrimiçi donanım benzetimi yöntemiyle sınanması mümkündür.

Anahtar sözcükler: Alan programlanabilir kapı dizinleri, çevrim-içi donanım benzetimi, dinamik sistem modelleme.

1. Giriş

Çeşitli sistemler için denetleyici donanımların geliştirilmesi sürecinde, tasarlanan denetleyicilerin başarımının araştırılması ve parametrelerinin ayarlanması gerekmektedir. Bu sayede denetleyicinin kullanıldığı nihai üründen mümkün olan en yüksek faydanın elde edilmesi amaçlanır.

Bu çalışmaları gerçekleştirmenin en geçerli yolu, geliştirilmekte olan denetleyicinin, ele alınan sisteme doğrudan doğruya bağlanmasıdır. Ancak, denetleyicinin gerçek bir sisteme bağlanması sistem maliyeti, hasar riski gibi sebepler dolayısıyla tercih edilmeyebilir. Bunun yanı sıra, gerçek bir sistemin sınamaya sokulmak üzere çeşitli alıcılarla donatılması ve bu alıcıların bağlantılarının doğru şekilde gerçekleştirilmesi başlı başına çözülmesi gereken bir problemdir.

Bu makalede, bir denetleyicinin sınanması sürecinde bu sorunları aşmak için; tercih edilen çevrimiçi donanım benzetimi yönteminin bir parçası olarak kullanılacak bir model anlatılmaktadır. Makalenin ikinci kısmında, benzer sorunları çözmek üzere geliştirilmiş çözümlerin kısa bir özeti verilmiştir. Üçüncü kısımda geliştirilmiş olan çözüme dair bilgiler sunulmuştur. Son kısımda ise buna dair sonuçlar ve tartışmalar yer almaktadır.

2. Mevcut Çalışmaların İncelenmesi

Çevirimiçi Donanım Benzetimi, (ÇDB, “Hardware-in-the-Loop Simulation”) esas itibariyle geliştirilmiş bir donanımın ya da yazılımın (bilgisayarda ya da çeşitli kırmıklar kullanılarak benzetimi yapılan) sanal bir sistem üzerinde sınanmasını sağlayan önemli bir tekniktir. Bu nedenle tasarım problemlerine çözümler geliştirilmesi aşamasında; kontrol mühendisliğinde çeşitli gerçeklerinde sınanması çok zor ya da maliyeti yüksek olan sistemlerin sınanmasında söz konusu teknik oldukça sık olarak kullanılır [1-2]. Son on yıl içinde ÇDB, kişisel bilgisayar tabanlı ya da DSP kırmıkları kullanılarak gerçekleştirilmiştir [3-4].

Yakın zamanda yapılan ÇDB uygulamalarında yavaş yavaş FPGA kırmığı da kullanılmaya başlamıştır. FPGA kırmığı, paralel işlem yapabilme konusundaki üstün becerisi sayesinde, geniş bir uygulama alanı bulmuştur. Bu kırmıklar ÇDB gibi gerçek zamanlı uygulamalar geliştirmek isteyen araştırmacıların tercihi olmaya başlamıştır. Çek Teknik Üniversitesi'ndeki (Prag) araştırmacılar çeşitli sistemler için sınama araçları geliştirirken ÇDB uygulaması geliştirmiştir. Motor kontrolü, hava yastığı kontrolü ve baskı makinesi kontrolü gibi projelerde ÇDB uygulamalarını sınayan araştırmacılar FPGA kırmığını uygulamalarında kullanarak MATLAB/Simulink, UPPAAL gibi benzetim uygulamalarıyla karşılaştırıldığında hız konusunda büyük bir başarım sağladıklarını göstermişlerdir [5]. Benzer olarak, üç fazlı paralel devre etkin süzgeçlerinde ortaya çıkan arızaların sezimi ve dengelemesi için ÇDB uygulaması geliştiren araştırmacılar, FPGA kırmığı kullanımıyla arıza oluşumu ve arıza sezimi arasında geçen zamanı enküçülterek 10 µs'ye kadar indirebilmiştir [6]. Başka bir ÇBD uygulaması da kalıcı mıknatıslı senkron motorların benzetimi (KMSM) için geliştirilmiştir. Bu uygulamada KMSM modeli sonlu eleman çözümlemesi kullanılarak yapılmış ve FPGA kırmığı üzerinde uygulanmıştır. Bu uygulamada zaman adımı FPGA kırmığı kullanımıyla 1.3 µs'ye kadar düşürülmüştür [7-8].

Elektrik sistemlerinin benzetimi içinde çeşitli araştırmalarda FPGA kırmığı kullanılmaktadır. Bir elektrik sisteminin gerçek zamanlı benzetimi için FPGA kırmığından faydalanılmış, kırmığının paralel işlem yapabilme yeteneği kullanılarak çoklu örnekleme yaklaşımı uygulanmıştır.

Böylece, zaman adımı 2 µs'ye kadar indirilerek gerçek zamanlı bir davranış elde edilebilmiştir [9]. DC makinelerin modellerinin gerçek-zamanlı benzetimi bir başka araştırmaya konu olmuş, FPGA kullanımı ile yapılan benzetimin ticari diğer benzetim yazılımlarıyla karşılaştırılması sonucu çok daha başarılı olduğu görülmüştür [10].

Gerçek zamanlı güç sistemlerinin benzetiminde FPGA kırmığı üzerinde kurulan çeşitli yeni devre mimarilerinin benzetim performansını hatrı sayılır bir şekilde iyileştirdiği bu sırada söz konusu sistemin maliyetinin eski sistemlere göre çok da fazla değişmediği gözlenmiştir [11]. Güç elektroniği çeviricilerinin gerçek-zamanlı benzetimi için geliştirilen mimaride de yine FPGA kırmığı kullanılmıştır. Uygulamanın başarımı, MATLAB uygulaması ile yapılan koşut bir uygulamayla karşılaştırılmış ve daha başarılı olduğu vurgulanmıştır [12]. Kanada'da Laval Üniversitesi'nde yapılan bir araştırmada havacılık ve uzay sistemlerinde kullanılan güç sistemlerinin gerçek zamanlı benzetimi FPGA kullanılarak yapılmış ve sistemin benzetim adımı 0.4 µs mertebesine kadar düşürülebilmiştir [13].

Çeşitli gerçek sistemlerin benzetiminin bu yordamla gerçekleştirilmesinin yanı sıra yeni sistemlerin yaratılıp sınanmasında da ÇDB kullanılmaktadır. Yapılan bir çalışmada, kullanıcı için elektronik devrelerin işlevsel ve mantıksal etkileşimli benzetimi için bir FGPA kırmığı tabanlı bir mimari oluşturulmuştur. Bu mimari de kullanıcı sıradüzensel devre elemanlarını seçip, istediği devreyi oluşturarak gerçek zamanlı benzetimini yapabilmektedir [14].

3. Uygulama

3.1. Sistem Modeli

Benzetimi gerçekleştirilen DC motor, Şekil 1'de verilen blok diyagramı ile modellenmiştir. Bu modele ait parametrelerin sayısal değerleri Tablo 1'de sunulmuştur.Benzetim esnasında, motorun hem açısal hız, hem de açısal konum ifade eden durum değişkenlerinin gözlenmesi istendiğinden, her iki durumu da motora uygulanan gerilimden elde etmeyi sağlayacak ayrık-zamanlı transfer fonksiyonları MATLAB/Simulink yazılım paketi yardımıyla (1kHz'lik bir örnekleme frekansı için) yaklaşık olarak elde edilmiştir. Modele ait yaklaşık açısal hız transfer fonksiyonu şu şekildedir:

$$\frac{\omega(z)}{v(z)} = \frac{1.012z+0.1166}{z^2-0.9173z+0.007312} \quad (1)$$

Bu denklemin çözümünün FPGA kırmığı üzerinde rahatça gerçekleştirilebilmesi için basit cebirsel düzenlemelerle bir sabit katsayılı fark denklemine dönüştürülmesi gereklidir:

$$\frac{\omega}{v} = \frac{1.012+0.1666\,q^{-1}}{q-0.9173+0.007312\,q^{-1}} \quad (2)$$

Bir başka şekilde

$$\omega(q-0.9173+0.007312q^{-1}) = v(1.012+0.1666q^{-1}) \quad (3)$$

ifade edilebilir. Böylece, sonlu fark denklemi

$$\omega_{k+1} = 1.012v_k + 0.1666v_{k-1} + 0.9173\omega_k - 0.007312\omega_{k-1} \quad (4)$$

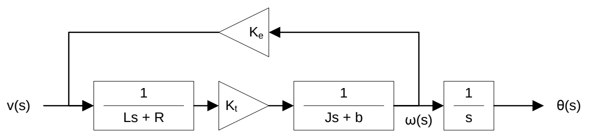
halini alır. Burada k çözüm esnasında mevcut zaman adımı olup; bir sonraki zaman adımında açısal hızın sahip olacağı değer, açısal hızın ve uygulanan gerilimin daha önceki zaman adımlarında sahip olduğu değerler kullanılarak Denklem 5 sayesinde hesaplanabilir. Benzer şekilde, motora ait yaklaşık açısal konum transfer fonksiyonu ve ilgili sabit katsayılı fark denklemi ise

$$\frac{\theta(z)}{v(z)} = \frac{4.481\times10^{-4}z^2+7.073\times10^{-4}z+2.351\times10^{-5}}{z^3-1.917z^2+0.9246z-0.007312} \quad (5)$$

şekildedir. Buna karşılık gelen sonlu fark denklemi

$$\theta_{k+1} = 4.481\times10^{-4}v_k + 7.073\times10^{-4}v_{k-1} + 2.351\times10^{-5}v_{k-2} + 1.917\theta_k - 0.9246\,\theta_{k-1} + 0.007312\theta_{k-1} \quad (6)$$

olur.



Şekil 1: Motor modeli blok diyagramı

Tablo 1: Motor model parametreleri

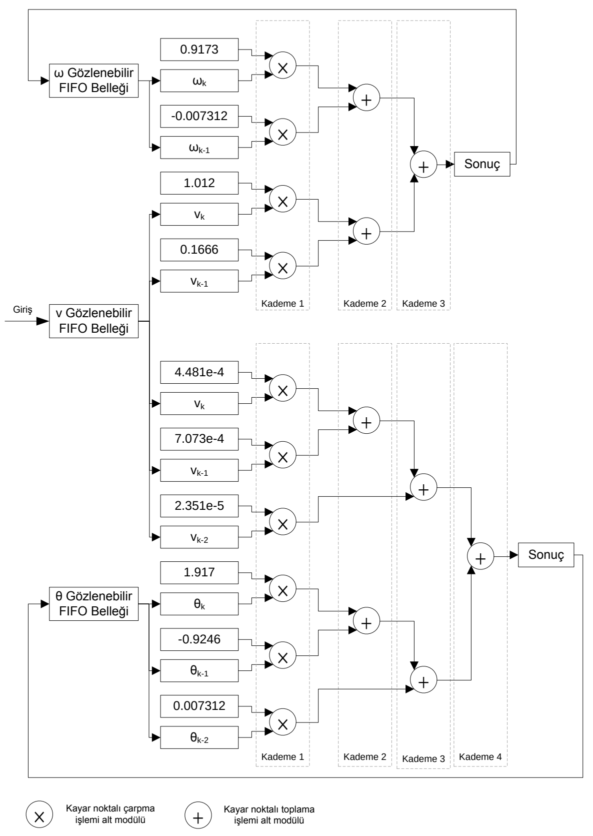
Parametre	İsim	Değer
L	Armatür endüktansı	1.04mH
R	Armatür direnci	7.19Ω
Kt	Tork sabiti	38.2mNm/A
Ke	Hız sabiti	250rpm/V
J	Rotor eylemsizliği	44.0gcm^2
b	Sönüm oranı	2.0275mNm·s

3.2. Modelin FPGA Üzerinde Sentezi

FPGA ("Field Programmable Gate Array", "Alanda Programlanabilir Kapı Dizisi"), tasarlanmış olan bir sayısal devrenin kolayca gerçeklenip çalışır hale getirilmesini sağlayan programlanabilir bir kırmıktır. Tasarlanmış olan devreler; VHDL, Verilog gibi donanım tanımlama dilleriyle ve/veya kırmık üreticilerinin sunduğu çeşitli gereçler aracılığıyla tanımlanır. Ardından yine üreticinin sağlamış olduğu sentezleme ve derleme gereçleri aracılığıyla, kırmık içerisinde bulunan ve özel olarak tasarlanmış mantık elemanlarının tasarlanan devrenin işlevini gerçekleştirmesini sağlayacak bir bağlantı listesi oluşturulur. Bu listenin bir programlayıcı aracılığıyla kırmığa yüklenmesi neticesinde söz konusu sayısal devre kırmık üzerinde gerçeklenmiş olur.

FPGA kırmıklarının en büyük yararlarından biri, karmaşık mantık devrelerinin uzun ve masraflı tasarım ve üretim aşamalarına gerek kalmaksızın gerçeklenebilmesini sağlamalarıdır. Bir diğer avantaj ise tasarlanan bir devrenin birden fazla örneğinin aynı kırmık üzerinde gerçeklenebilir oluşudur. Bu sayede birçok devrenin birbirine paralel olarak çalıştırılabilmesi mümkündür. Bunlara ek olarak; çeşitli mikroişlemciler de dahil olmak üzere halihazırda bulunan birçok sayısal devre elemanı, tüm işlevleriyle aslına uygun olarak FPGA kırmıkları üzerinde gerçeklenebilmektedir. Bu sayede karmaşık devre fonksiyonlarının tek bir kırmık üzerinde, paralel işlem gücünden faydalanılarak gerçekleştirilmesi mümkündür.

Bölüm 3.1'de elde edilen sabit katsayılı fark denklemlerinin çözümünü mümkün olan en yüksek hızda gerçekleştirebilmek için de FPGA kırmığının sağladığı paralel işlem gücünden faydalanılmıştır. (4) ve (6) numaralı denklemlerde bulunan her bir aritmetik işlemi için kayar noktalı bir hesap modülü, işlemleri mümkün olduğunca eş zamanlı gerçekleştirebilecek biçimde kademeler halinde birbirine bağlanmıştır. Bunların yanında, bulundurduğu tüm elemanların aynı anda okunabilir olduğu FIFO (“First In, First Out”, “İlk Giren İlk Çıkar”) yapısında arabellek modülleri bulunmaktadır. Bu modüller, fark denklemlerine verilen girdilerin ve hesaplanan sonuçların geçmiş değerlerini, ihtiyaç duyulan zaman adımına kadar tutar. Bu şekilde elde edilmiş olan yapı, Şekil 2'de gösterilmiştir. Bunlara ek olarak, modüllerin çalışabilmesi için gerekli saat atımlarını gerekli yerlere gönderen bir yönetici modül de bulunmaktadır.



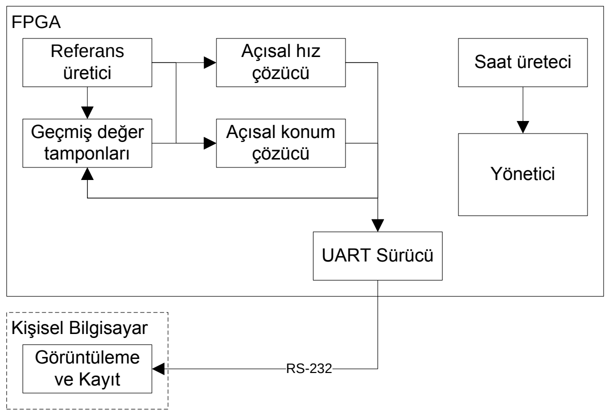
Şekil 2: Sabit katsayılı fark denklemi çözücü sistemin şeması.

Sabit katsayılı fark denklemlerinin çözülmesi, yönetici modül tarafından kayar noktalı çarpma modüllerinin eş zamanlı işletilmesiyle başlar. Ardından elde edilen sayıların toplanması için, diğer kademelerdeki kayarnoktalı toplama modülleri sırasıyla işletilerek sonuç elde edilir. Son olarak, girdiler ve elde edilen sonuçlar ilgili arabelleklerde saklanarak, bir sonraki saat çevriminde yapılacak hesaplama için ihtiyaç duyulan değerler kaydedilir.

Fark denklemlerinin çözülme süresi, çözümü gerçekleştirecek kayar noktalı hesap modüllerinin sayısına logaritmik olarak bağlıdır. Kullanılan modüller dört saat çevrimi içerisinde bir işlem gerçekleştirebilmektedir [15]. Girdi ve sonuçların ilgili arabelleklere itilmesi ise bir saat çevriminde gerçekleşmektedir. Buradan yola çıkılarak, kullanılmış olan yapıyla bir sabit katsayılı fark denkleminin çözümü için gereken saat çevrimi sayısı

$$n_c = \lceil \log_2(n_a + 1) \rceil + 5 \tag{7}$$

şekildedir. Burada n_a denklemdeki toplama işareti sayısı, n_c ise hesaplama için gereken saat çevrimi sayısıdır. Buradan hareketle (4)'te verilen denklemin çözümü 13, (6)'de verilen denklemin çözümü ise 17 saat çevriminde gerçekleştirilebilir.



Şekil 3: Geliştirilen sistem için sınama düzeneği.

3.3. Sınama

Gerçeklenen sistemin doğru çalıştığının sınanması amacıyla, Bölüm 3.2’de açıklanmış olan modüllere bir referans üretici modül, bir de RS-232 iletişim modülü eklenmiştir (Şekil 3). Referans üretici, fark denklemlerine sabit değer olarak 1 verir. Bu, büyüklüğü 1 ve geçiş anı $t = 0$ olan bir adım fonksiyonuna denktir. İletişim modülü ise (4) ve (6) numaralı denklemlerden her zaman adımı için elde edilen sonuçları alarak uygulama donanımının bağlı bulunduğu kişisel bilgisayara göndermekle yükümlüdür. Gönderilen bilgiler, benzetim esnasında kişisel bilgisayar üzerinden takip edilebilir ve sonradan inceleme yapılmak üzere kaydedilebilir.

4. Tartışma ve Sonuçlar

Tasarlanan sistemin sınama düzeneğiyle beraber gerçeklenmesi, Altera “Nios II Embedded Evaluation Kit, Cyclone III Edition” geliştirme kiti [16] üzerinde yapılmıştır. Sistemin bu donanım üzerinde harcadığı çeşitli kaynakların listesi Tablo 2’de verilmiştir.

Kullanılan sınama düzeneği sayesinde elde edilen denklem çözümlerinin doğruluğunu sınamak amacıyla Bölüm 3.1’de açıklanan transfer fonksiyonlarının adım girdisine verdikleri cevaplar, MATLAB/Simulink yazılım paketi kullanılarak bilgisayar üzerinde hesaplanmıştır. Bu yolla elde edilen değerler, sistem tarafından gerçekleştirilen denklem çözümlerinden gelen sonuçlarla karşılaştırılmıştır. Ortaya çıkan hataların grafikleri, Şekil 5 ve 6’da sunulmuştur.

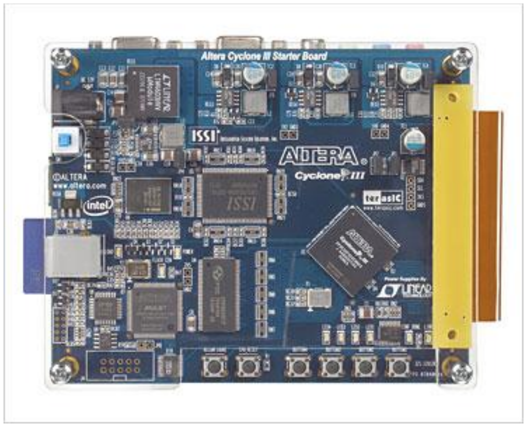
Karşılaştırma sonuçları incelendiğinde, hız durum değişkeninin hesaplanmasında 6.875×10^{-6} radyan/saniyelik, konumda ise 14.018 radyanlık bir azami hata olduğu görülmüştür. Hızdaki hatanın -9.431×10^{-8} radyan/saniyede sabitlendiği, konumdaki hatanın ise zaman içinde artmaya devam ettiği gözlenmiştir. Daha fazla inceleme yapıldığında MATLAB yazılım paketinin çift duyarlı kayan noktalı işlemci ile çalıştığı öğrenilmiştir [17]. Buna karşılık uygulamada tek duyarlı kayan noktalı işlemci kullanıldığı göz önünde bulundurularak, fark denklemlerinin çözümleri bilgisayar üzerinde tek ve çift duyarlı değişkenler kullanılarak tekrarlanmıştır. Elde edilen sonuçlar; benzetimde ortaya çıkan hatanın söz konusu duyarlık farkından kaynaklandığını, çift duyarlık kullanılması durumunda bu farkların hız için 10^{-14} ve konum için 10^{-8} mertebesine indiğini göstermiştir. Bu sonuçtan hareketle benzetimin mümkün olan en yüksek doğrulukla gerçekleştirilebilmesi için çift duyarlı kayan noktalı işlemcilerin kullanılması gerektiği açıktır.

Tablo 2: Sistem tarafından harcanan donanım kaynakları

Toplam mantık elemanı	Toplam mantık işlemi	Yazmaç sayısı
15,431 (%63)	14,088 (%57)	6,450 (%26)

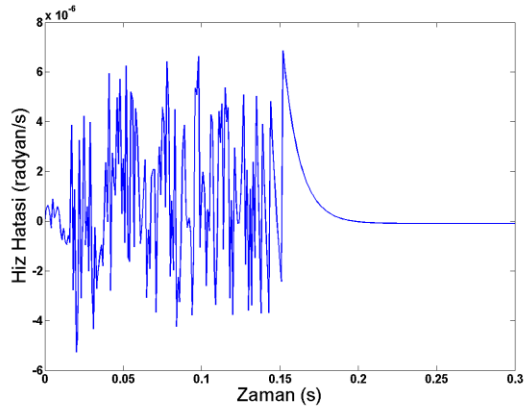
EM9b*	Bellek bit'leri**
39 (%30)	69,800 (%11)

* 9-bit Embedded Multiplier, çarpma işlemlerini hızlandırmak amacıyla FPGA kırmığının içerisinde bulunan özelleştirilmiş donanım.
** FPGA kırmığı tarafından gerçeklenen devre tarafından kullanılan bellek elemanlarıdır. Denklem çözümünden elde edilen değerler, geliştirme kiti üzerinde ayrıca bulunan SDRAM modüllerinde saklanmıştır.
† Verilen kullanım değerleri ve yüzdeler, sistemin Altera Cyclone III EP3C25F324C6 marka ve model FPGA kırmığı için derlendiğinde kullandığı kaynaklardır. Farklı ürünler kullanıldığında bu değerler bir miktar değişebilir.

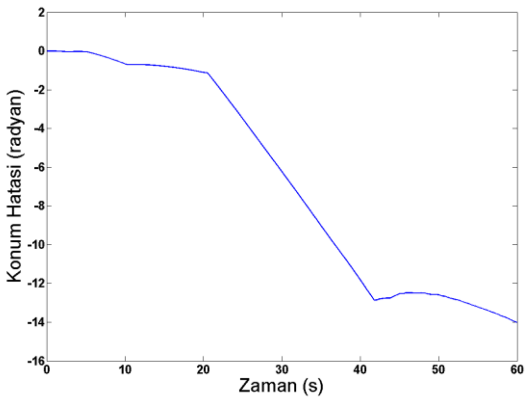


Şekil 4: Altera Nios II Embedded Evaluation Kit, Cyclone III Edition geliştirme kiti

Harcanan kaynakların listesi incelendiğinde, iki durum değişkenine sahip basit bir sistem için dahi eldeki kaynakların yaklaşık %63'ünün kullanıldığı görülmüştür. Benzetimi yapılacak sistemin karmaşıklaşması durumunda ihtiyaç duyulacak kaynak miktarının, eldeki kaynakları kolayca aşabileceği kolayca tahmin edilebilir. Sistem tasarlanırken paralel işlem gücünden azami derecede yararlanmak amacıyla denklemlerde bulunan her cebirsel işlem için ayrı bir kayar noktalı işlem modülü kullanıldığı göz önünde bulundurulduğunda, yüksek kaynak kullanımının sebebinin kullanılan modül sayısı olduğu kolayca anlaşılabilmektedir. Diğer yandan; kullanılan donanım üzerinde bulunan 50MHz'lik saat kaynağı sayesinde, (4) ve (6)'deki denklemlerin çözümü, sırasıyla 260ns ve 340ns'de tamamlanmaktadır. Bu denklemlerin 1kHz'lik bir örnekleme frekansı için oluşturulduğu göz önünde bulundurulduğu takdirde; hesaplamaların ihtiyaç duyulandan çok daha kısa bir sürede tamamlandığı anlaşılmaktadır. Hatta, yüksek başarımlı sistemlerde karşılaşılabilen 10kHz'lik örnekleme frekansları için dahi bu süreler oldukça kısadır. Bu bulgulardan hareketle, paralel işlem gücünden ödün vermek suretiyle benzetim süresinin (ihtiyacı karşılayacak kadar kısa kalmak kaydıyla) uzamasına izin verilerek kullanılan kaynak miktarını azaltmanın mümkün olabileceği görülmektedir. Bunu gerçekleştirebilmek amacıyla; sınırlı sayıdaki kayar noktalı hesap modülünü kullanarak, fark denklemlerinin çözümünü gerçekleştirebilecek özelleştirilmiş bir işlemci modülünün tasarlanmasına karar verilmiştir.



Şekil 5: MATLAB ile yapılan karşılaştırmada hız hatası



Şekil 6: MATLAB ile yapılan karşılaştırmada konum hatası

Sınama esnasında, iletişim için kullanılan RS-232 protokolünün yüksek hızlarda ve/veya geniş veri paketleriyle aktarım hızının yetersiz kaldığı görülmüştür. Bu sebeple benzetim esnasında elde edilen değerlerin, kullanılan donanım üzerindeki bellek birimlerinde (SDRAM) depolanarak, benzetim sonrasında bilgisayara aktarılması yoluna gidilmesi gerekmiştir. Ancak, bu değerlerin, gerçek zamanlı olmasa dahi benzetim esnasında ve zaman ölçeklemesine maruz kalmadan izlenebilmesi istenmektedir. Bunu gerçekleştirebilmek üzere Ethernet gibi daha yüksek hızlı veri aktarım protokollerinin kullanılması gerekeceği öngörülmüştür.

Sonuçta, bu makalede bir elektrik motorunun, matematiksel modelinin düzenlenmesi yoluyla elde edilen denklemler aracılığıyla bir FPGA kırmığı üzerinde gerçek zamanlı benzetimini mümkün kılan bir sistem ve bunun uygulaması açıklanmıştır. Sistemin, eklenecek uygun iletişim, arayüz ve eşzamanlama modülleri (darbe genişlik kiplenimi/PWM, değer/frekans çeviriciler, enkoder işaret üreteçleri v.b.) aracılığıyla uygun bir denetleyiciye bağlanarak bu denetleyicinin çevrimiçi donanım benzetimi yoluyla sınanmasını sağlaması mümkündür. Bu sayede denetleyici donanım tasarlama ve geliştirme sürecinin büyük ölçüde kolaylaştırılması ve hızlandırılması sağlanabilecektir.

5. Teşekkür

Bu çalışma, TÜBİTAK'ın 108E048 kodlu projesi kapsamında tamamlanmıştır. Makalenin yazarları mali desteğinden ötürü TÜBİTAK'a içtenlikle teşekkür ederler.

6. Kaynakça

[1] W. Grega, "Hardware-in-the-loop simulation and its application in control education," 29th ASEE/IEEE Frontiers in Education Conference, 1999.

[2] M. Bacic, "On hardware-in-the-loop simulation", Proceedings of the 44th IEEE Control Conference, 2005.

[3] R. Crosbie, J. Zenor, R. Bednar, N. Hingorani ve T. Ericsen, "High-Speed, Scalable, Real-Time Simulation Using DSP Arrays," IEEE Proceedings of the 18th Workshop on Parallel and Distributed Simulation, (PADS'04), Kufstein, Avusturya, Cilt:2, s:52-59, 2004.

[4] P. Baracos, G. Murere, C.A. Rabbath ve W. Jin, "Enabling PC-Based HIL Simulation for Automotive Applications," IEEE International Electric Machines and Drives Conference, (IEMDC 2001), Boston, U.S.A., Cilt:1, s: 721-729, 2001.

[5] J. Krakora ve Z. Hanzalek, "FGPA Based Tester Tool for Hybrid Real-time Systems," Microprocessors and Microsystems - Embedded Hardware Design, Cilt: 32, s:447-459, 2008.

[6] S. Karimi, P. Poure ve S. Saadate, "FPGA-based Hardware in the Loop Validation for Fault Tolerant Three-Phase Active Filter," IEEE International Symposium on Industrial Electronics, (ISIE 2008), Cambridge, UK, Cilt:1, s:2189-2194, 2008.

[7] C. Dufour, J. Belanger, V. Lapointe ve S. Abourida, "Real-Time Simulation on FPGA of a Permanent Magnet Synchronous Machine Drive Using a Finite-Element Based Model," 9th International Symposium on Power Electronics, Electrical Drives, Automation and Motion, (SPEEDAM 2008), Ischia, İtalya, Cilt:6, s: 19-26, 2008.

[8] S. Abourida, J. Belanger ve C. Dufour, "Real-Time HIL Simulation of a Complete PMSM Drive at 10 µs Time Step," 2005 European Conference on Power Electronics and Applications, Dresden, Almanya, Cilt:1, s:9-P.9, 2005.

[9] I. Bahri, M-W. Naouar, E. Monmassın, I. Slama-Belkhodja ve L. Charaabi, "Design of an FPGA-Based Real-Time Simulator for Electrical System," 13th IEEE Power Electronics and Motion Control Conference, (EPE-PEMC 2008), Poznan, Polonya, Cilt:4, s:1365-1370, 2008.

[10] K. Jayalakshmi ve V. Ramanarayanan, "Real-Time Simulation of Electrical Machines on FPGA Platform," IEEE Proceedings of India International Conference on Power Electronics, Chennai, Hindistan, Cilt:3 , s:259-263, 2006.

[11] J.C.G. Pimentel, "Implementation of Simulation Algorithms in FPGA for Real Time Simulation of Electrical Networks with Power Electronics Devices," IEEE International Conference on Reconfigurable Computing and FPGA's, (.ReConFig 2006), San Luis Potosi, Meksika, Cilt:1, s: 1-8, 2006.

[12] P. Le-Huy, S. Guerette, L.A. Dessaint ve H. Le-Huy, "Dual-Step Real-Time Simulation of Power Electronic Converters Using an FPGA," IEEE International Symposium on Industrial Electronics, Montreal, Quebec, Kanada, Cilt: 7, s: 1548-1553, 2006.

[13] J.C.G. Pimentel ve Y.G. Tirat-Gefen, "Real-Time Hardware in the Loop Simulation of Aerospace Power Systems," 5th International Energy Conversion Engineering Conference and Exhibit, (IECEC 2007), St.Louis, Mossouri, U.S.A., Cilt:2 , s:8-17 , 2007.

[14] D. Castells-Rufas ve J. Carrabina, "Jumble: A Hardware-in-the-Loop Simulation System for JHDL," 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, (FCCM 2007), Napa Valley, California, U.S.A., Cilt:1, s: 345-348, 2007.

[15] R. Usselmann, "Open Floating Point Unit Manual", http://www.opencores.org/, 2009.

[16] Altera "Nios II Embedded Evaluation Kit, Cyclone III Edition" geliştirme kiti ürün bilgi sayfası, http://www.altera.com/products/devkits/altera/kit-cyc3-embedded.html, 2009.

[17] The MathWorks Product Support "1108-Troubleshooting Common Floating-Point Arithmetic Problems", http://www.mathworks.com/support/tech-notes/1100/1108.html , 2009,

Endüstriyel Kontrol Uygulamaları için Komut Üretim Yöntemleri

Ulaş Yaman[1], *Melik Dölen*[2], *A. Buğra Koku*[3]

Makine Mühendisliği Bölümü
Orta Doğu Teknik Üniversitesi, Ankara
[1]uyaman@metu.edu.tr
[2]dolen@metu.edu.tr
[3]kbugra@metu.edu.tr

Özetçe

Bu çalışmada, çeşitli robotik ve otomasyon uygulamalarında kullanılabilecek komut üreteçleri tasarımları incelenmiştir. Literatürde RS-274D tabanlı bir dil ile desteklenen enterpolasyon yöntemleri yaygın olarak kullanılmaktadır. Bu çalışmada ise günümüz bilgisayar teknolojisinin tüm imkanlarından yararlanan iki yeni yöntem paketi önerilmiştir. Birinci yöntem paketinde oluşturulmuş olan komutlar, yüksek dereceden sonlu farklarının hesaplanmasından sonra çeşitli veri sıkıştırma yordamları kullanılarak, kayıpsız olarak saklanmaktadır. Önerilen ikinci paket yöntemde ise komut yörüngesi öncelikli olarak doğrusal bölümlere ayrılarak, kalan veriler Chebyshev polinomları tabanlı fonksiyon yaklaşıklama (“approximation”) teknikleriyle temsil edilmektedir. Ayrıca, makalede bu tekniklerin başarımları karşılaştırmalı olarak değerlendirilmiştir.

Anahtar Sözcükler: Komut Üretimi, Veri Sıkıştırma, Fonksiyon Yaklaşıklama.

Abstract

In this study, several command generation techniques that can be used in various robotics and automation applications are investigated. In the literature, interpolation methods that are supported with a RS-274D based language are commonly used. Two groups of novel command generation methods using computer facilities are proposed in the paper. In the first group of methods after taking higher order differences of command sequences, different data compression techniques are employed to store command sequences in a smaller size of memory. On the other hand, in the second group of methods command trajectories are first divided into linear segments and then the remaining trajectory is approximated using Chebyshev based polynomials. Then all the methods are evaluated according to their performances.

1. Giriş

İlerleyen bilgisayar teknolojisiyle birlikte, farklı yapıdaki robotlar ve manipülatörlerin otomotiv, uzay/havacılık, tüketici elektroniği gibi endüstrinin çeşitli alanlarında yaygın bir şekilde kullanılmasına başlanmıştır. Manipülatörler, modülerlikleri ve programlama kolaylıklarından dolayı kaynak, boyama, silme/parlatma, parça iletimi ve montajı gibi temel üretim görevlerinin yerine getirilmesinde vazgeçilmez araçlar olarak öngörülmektedirler. Bunların yanı sıra yüksek hızlarda hassas konumlama yapabildikleri için yetenekli iş gücü gereksinimini de büyük ölçüde azaltmışlardır. Böylelikle dolaylı olarak daha kaliteli ürünlerin ve daha fazla ürünün ortaya çıkmasına sebep olmuşlardır.

Manipülatörlerin yukarıda bahsedilen alanlara uygulanması oldukça kolaydır. Öncelikle manipülatör yörüngesinin yapılacak olan işe göre tanımlanması gerekmektedir. Daha sonrasında manipülatörü çalıştırmadan önce ters kinematik model kullanılarak bağlantı noktalarında bulunan eyleyicilerin açısal konumları hesaplanmaktadır. Bu verilerle birlikte programlanan hareket denetleyicisinin amacı açısal bağlantı konumlarını hassas bir biçimde denetlenmektedir.

Endüstride kullanılan hareket denetleyici kartları, genel olarak karışık yörüngeleri doğrusal parçalarla ifade etmek amacıyla vektör veri tabloları kullanmaktadırlar. Bunu yaparken yörüngedeki hatanın kabul edilebilir sınırlar içerisinde olması gerekmektedir. Daha sonrasında bu kartlar, konum denetim döngüsü için gerekli olan referans işaretlerini üretmek amacıyla ardışık tablo değerleri arasında, gerçek zamanlı olarak, doğrusal enterpolasyon yapmaktadırlar. Eğer robot kolunun karmaşık bir yörüngeyi takip etmesi gerekirse, oluşturulan doğrusal parçaların uzunlukları kısalmakta ve dolayısıyla doğrusal enterpolasyonun yararı ortadan kalkmaktadır. Aynı zamanda bu durum tablolar için ayrılmış olan bellek kaynaklarının tükenmesine de yol açmaktadır. Bu sorunlarla karşılaşmamak amacıyla NURBS enterpolasyon yöntemini uygulama yetisine sahip, maliyet açısından yüksek, ileri düzey denetleyici üniteleri kullanılabilmektedir. Öte yandan, denetlenen eksenlerin sayısı arttığında matematiksel yükü fazla olan böyle bir enterpolasyon yönteminin kullanılması hem teknik hem de ekonomik açıdan uygun değildir. Önerilen yöntemlerle birlikte yüksek sayıda eksenin denetimi aynı kaynakları kullanarak daha rahat bir şekilde yapılabilmektedir [1].

Verinin öncelikle yüksek dereceden sonlu farkının alınıp sonrasında sıkıştırılması yöntemi literatürde genellikle video işlemek için özel olarak geliştirilmiş donanımsal ürünlerde kullanılmaktadır [2]. Benzer işlemlerin yörünge komutlarının üretiminde kullanılmasının temel nedeni farkı alınan komut serisinde yüksek sıklıkta görülen aynı değere sahip yörünge komutlarıdır. Daha sonrasında Huffman [3] veya Aritmetik kodlama [4] gibi veri sıkıştırma yordamlarının uygulanmasıyla yörünge komutlarının boyutunda büyük ölçüde daralma görülmektedir. Veri alanındaki bu daralmayla birlikte gerçek zamanlı olarak yapılmak istenen komut iletimi kolaylaşmaktadır. Kullanılan bu iki sıkıştırma yordamının temelinde yatan mantık aynıdır. Sıkıştırılmak istenen veri setinde sıklıkça kullanılan verilerin daha az bit ile ifade edilmesi ve az kullanılan verilerin ise daha fazla bit ile gösterilmesi amaçlanarak sonuçta daha az bit kullanılmak istenmektedir.

Ortaya konulan ikinci paket yöntemlerde ise öncelikle üretilmek istenen komut dizini doğrusal bölümlere ayrılmaktadır. Oluşturulan bölümlerin birbirlerinden temel farkı, ardışık komutlar arasındaki artım miktarı ve başlangıç-bitiş noktalarıdır. Bu veriler bir tablo halinde saklanmakta ve komut üretilmek istendiğinde bu tablodan sırasıyla okunmaktadır. Komut üretmek veya herhangi bir fonksiyonun değerini hesaplamak amacıyla benzer tablo yöntemleri literatürde geniş bir biçimde kullanılmaktadır [5-7]. Komut yörüngesi hatasız bir biçimde doğrusal bölümlerle ifade edilemeyeceği için kalan veriler Chebyshev tabanlı fonksiyon yaklaşıklama tekniği kullanılarak ifade edilmiştir [8-9]. Yaklaşıklama yöntemi uygulanırken aynı zamanda hatanın istenilen sınırlar içerisinde kalmasına da özen gösterilmiştir. Doğrusal bölümleme yönteminin başarısını karşılaştırmak amacıyla komut yörüngesi, hatanın belirli sınırlar içerisinde kalması koşuluyla bölümlere ayrılarak Chebyshev fonksiyon yaklaşıklama yöntemi kullanılarak ifade edilmiştir. Benzer yöntemler literatürde de mevcuttur [9-10].

Makalenin genel akışı şu şekildedir: Bu bölümde önerilen yöntemlerin literatürdeki yerlerinden kısaca bahsedildikten sonra makalenin ikinci kısmında yüksek dereceden farklara ve veri sıkıştırmaya bağlı olan yöntemlerden bahsedilmiştir. Üçüncü bölümde ise bölümleme ve fonksiyon yaklaşıklama yordamlarının kullanıldığı ikinci paket yöntemler anlatılmıştır. Sonraki bölümde sonuçlar karşılaştırmalı olarak tartışılmış ve ileride yapılabilecek çalışmalara değinilmiştir.

2. Farkı Alınmış Verinin Sıkıştırılması

Makalenin bu bölümünde önerilen birinci yöntem paketi ayrıntılı olarak değerlendirilmiştir. İlk olarak komut yörüngesinin yüksek dereceden sonlu farkının alınmasının yararlarından bahsedildikten sonra, kullanılabilecek veri sıkıştırma yordamlarına değinilecek ve yöntemler MATLAB ortamında sınanacaktır.

2.1. Yüksek Dereceden Sonlu Fark Alma

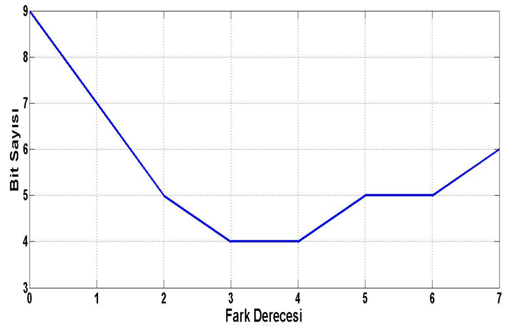
Bellek boyutundan kazanmak amacıyla yüksek dereceden sonlu fark alma yöntemleri veri depolamak üzere kullanılabilir. Özgün komut yörüngesini tanımlayan noktaları doğrudan saklamak yerine, farkı alınmış komut dizinini ve ilgili ilk değerlerini depolamak yeterli olacaktır. Hatırlanacağı üzere, farklar

$$\nabla q = q(k) - q(k-1) \tag{1a}$$

$$\nabla^2 q = \nabla q(k) - \nabla q(k-1) \tag{1b}$$

$$\nabla^n q = \nabla^{n-1} q(k) - \nabla^{n-1} q(k-1) \tag{1c}$$

şeklinde tanımlanabilir. Burada kullanılan $\nabla^n q$, n. dereceden farkı temsil etmektedir. Bu uygulamayla birlikte farkı alınan büyüklüğün erimi artan dereceyle birlikte azalmaktadır. Alınan farkın derecesine göre bellek gereksinimlerinin nasıl değiştiğini göstermek amacıyla farklı yörüngeler oluşturulmuş ve 7. dereceye kadar farkları hesaplanmıştır. Şekil 1 bu değişimi göstermektedir.



Şekil 1: Temsil edilen sayıların bit cinsinden uzunluğunun Fark Derecesine Göre Değişimi

Şekilden de anlaşılacağı üzere, 3. derecenin üstünde alınan farklarda bellek gereksiniminde bir artış görülmektedir. Bunun temel nedeni, dördüncü dereceden sonra verinin işaretinin almaşık şekilde pozitif ve negatif olarak değişmesidir. Böylece sonlu fark alındığında verinin eriminde artış olmakta ve daha fazla belleğe gereksinim duyulmaktadır. Dolayısıyla bellek açısından en uygun çözümü sağlayan fark derecesinin 3 ya da 4 olduğu görülmektedir.

Özgün komut yörüngesini oluşturabilmek için sonlu farkların uygun bir biçimde düzenlenmesi gerekmektedir. Bu düzenleme, (1) denkleminde yapılan işlemlerin tersini ("tümleme") içermektedir.

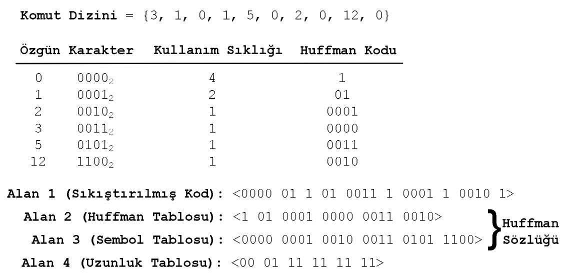
$$q(k) = q(k-1) + \nabla q(k) \tag{2}$$

Yukarıdaki denklemde görüldüğü gibi, k = 1 iken q(1) değerin hesaplanabilmesi için ∇q(1) haricinde q(0) ilk değerine ihtiyaç vardır. Daha yüksek dereceli sonlu farkların alınması durumunda gerekli olan ilk değerlerin sayısı da artmaktadır. Algoritmanın sıkıştırma oranları hesaplanırken, ilk değerlerin saklanabilmesi için gerekli olan belleği de dikkate almak gerekir.

2.2. Veri Sıkıştırma Yöntemleri

Veri sıkıştırması genel olarak belirli uzunluktaki veri dizisinin daha az bellek kullanarak ifade edilmesi şeklinde tanımlanabilir. Birinci paket yöntemler çerçevesinde kullanılan iki (kayıpsız) veri sıkıştırma yordamının (Huffman ve Aritmetik Kodlama) temelinde, sıklıkla kullanılan sayıların (karakterlerin) ikili düzende daha kısa kodlarla ve daha az kullanılan sayıların ise uzun kodlarla ifade edilmesi yatmaktadır.

Bu çalışmada önerilen birinci yöntemde; Huffman sıkıştırma algoritmasından yararlanılmaktadır. Bu iyi bilinen veri sıkıştırma tekniğiyle ilgili detaylı bilgiler [3] ve [11] numaralı kaynaklardan edinilebilir. Dikkat edilirse, bu yöntemde komut dizininde yer alan sayılar için ilgili serinin istatistiksel özelliklerine bağlı olarak uzunluğu değişen Huffman kodları yaratılmaktadır. Bu da doğal olarak algoritmanın daha karışık bir hal almasına ve bellek yönetim problemlerine neden olmaktadır. Bu sorunun üstesinden gelmek amacıyla değişik araştırmacılar kullanımı sınırlandırılmış Huffman kodlarını önermektedirler [11]. Ancak, böylesi değişiklikler doğal olarak algoritmanın verimini önemli ölçüde düşürmektedir. Bu makalede önerilen teknikte ilgili komut dizini 4 alana ayrılmaktadır. Bu alanların daha rahat bir şekilde ifade edilebilmesi için örnek bir veri sıkıştırma işlemi ve kullanılan veri yapıları Şekil 2'de gösterilmiştir.



Komut Dizini = {3, 1, 0, 1, 5, 0, 2, 0, 12, 0}

Özgün Karakter		Kullanım Sıklığı	Huffman Kodu
0	0000_2	4	1
1	0001_2	2	01
2	0010_2	1	0001
3	0011_2	1	0000
5	0101_2	1	0011
12	1100_2	1	0010

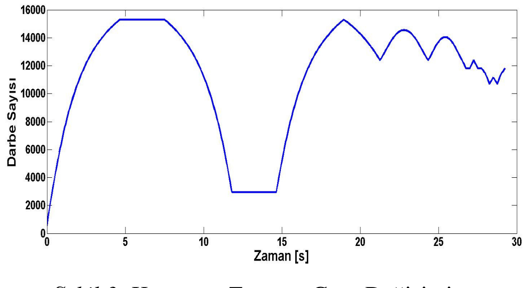
```
Alan 1 (Sıkıştırılmış Kod): <0000 01 1 01 0011 1 0001 1 0010 1>
  Alan 2 (Huffman Tablosu): <1 01 0001 0000 0011 0010>              } Huffman
   Alan 3 (Sembol Tablosu): <0000 0001 0010 0011 0101 1100>         } Sözlüğü
  Alan 4 (Uzunluk Tablosu): <00 01 11 11 11 11>
```

Şekil 2: Huffman Algoritmasıyla Veri Sıkıştırma Örneği

Birinci alanda Huffman kodlarıyla temsil edilen özgün komut bilgileri yer almaktadır. İkinci alan Huffman tablosunu barındırmakta olup, üçüncü kısımda ise sabit bit sayısıyla ifade edilmiş olan özgün karakterler (semboller) bulunmaktadır. İkinci ve üçüncü alan birlikte Huffman sözlüğünü temsil etmektedir. Son alanda ise, özgün karakterlere karşılık gelen Huffman kodunu ikinci alan içerisinde işaret eden indisler yer almaktadır.

2.3. Yöntemin Uygulanması

Önerilen yöntemin uygulanabilirliğini göstermek amacıyla bir PUMA 560 manipülatörü için oluşturulmuş olan komut yörüngesi kullanılmıştır [1]. Bu yörüngenin takip edilebilmesi için manipülatörün 1 numaralı ekleminde bulunan eyleyiciye bağlı olan optik konum duyucusunun darbe sayılarını içeren verinin öncelikle sonlu farkları alınmıştır. Sonrasında Huffman ve Aritmetik kodlayıcı veri sıkıştırma algoritmaları kullanılarak, komut dizisi sıkıştırılmıştır. İlgili eklem için üretilen komut dizisinin (optik konum duyucusunun ürettiği darbe sayısı cinsinden) zamana göre değişimi Şekil 3'te gösterilmiştir. Kullanılan konum duyucusu "quadrature" tip optik konum enkoderi olup, bir devirde 40000 (= 4×10000 darbe/devir) darbe üretebilme yetisine sahiptir.



Şekil 3: Komutun Zamana Göre Değişimi

Sözü edilen (açısal) konum yörüngesinin öncelikle değişik derecelerden sonlu farkları alınmış ve sonrasında Huffman ve Aritmetik kodlama yöntemleri ile veriler sıkıştırılmıştır. Sonuçlar, Tablo 2'de gösterilmiştir. Bu tablodaki sıkıştırma oranı

$$r = \frac{\sum_{m=1}^{4} L_m}{ceil\left(\frac{N-n}{8}\,fix\left(\frac{\log(d_{\max}-d_{\min})}{\log(2)}+1\right)\right)} \qquad (3)$$

şeklinde tanımlanmıştır. Burada, N orijinal komut dizininin uzunluğunu; n sonlu farkın derecesini; d_{max} ve d_{min} sırasıyla sonlu farkı alınmış serideki azami ve asgari değere sahip büyüklükleri ifade etmektedir. Bu ifadede **fix** tamdeğer fonksiyonu olup; **ceil** ise argümanını en yakındaki büyük tamsayıya yuvarlatan matematiksel bir fonksiyondur. Benzer olarak, (3)'te L_m (byte cinsinden) ilgili sıkıştırma tekniğinin kullanmış olduğu dört ayrı alandaki (bkz. Şekil 2) verinin uzunluğunu temsil etmektedir.

Hatırlanması gereken önemli bir nokta da, sıkıştırılmış veriden tümlenerek elde edilen yörüngenin özgün yörüngeden herhangi bir farkının olmayışıdır. Başka bir deyişle, önerilen yöntem kayıpsız bir komut yörüngesi oluşturma yöntemidir.

Tablo 2: Sıkıştırma Oranları

	Sıkıştırma Oranları [%]	
Fark Derecesi	Huffman	Aritmetik Kodlama
0	200	183
1	133	119
2	36	34
3	36	33
4	48	44
5	63	56
6	84	73

Tablo 2'den de görüldüğü üzere komut yörüngesinin sonlu farkının alınmadan sıkıştırılması iyi sonuçlar vermemektedir. Bunun nedeni komut yörünge dizisinde yer alan sayıların birbirlerinden çok farklı olmasından ötürü sıkıştırma algoritması sonucu oluşan sözlüğün boyutunun oldukça büyük olmasıdır. Tablodan genel olarak çıkarılabilecek diğer bir sonuç ise Aritmetik kodlama yönteminin Huffman sıkıştırma algoritmasına göre başarımının biraz daha yüksek olmasıdır. Üçüncü dereceden sonlu farkın alınmasıyla birlikte, kodlayıcının komut yörünge verisinin boyutunda yaklaşık olarak 2:3 oranında azalma görülmektedir. Dikkat edilirse, Tablo 2'de elde edilen sonuçlar, Şekil 1'de verilen sonuçlarla uyum içerisindedir. Üçüncü fark derecesine kadar azalma gösteren sıkıştırma oranı dördüncü dereceden itibaren artmaktadır. Bu durumu verideki sayı (karakter) sıklıklarının azalmasıyla birlikte ilgili verilerin yer aldığı alanların genişlemesine bağlamak mümkündür.

3. Veri Bölümleme ve Fonksiyonel Yaklaşıklama Teknikleri

Bu bölümde veri bölümleme ve Chebyshev polinomları tabanlı fonksiyon yaklaşıklamasına dayalı yöntemler önerilmiş ve MATLAB yazılımı aracılığıyla uygulanabilirliği sınanmıştır. Öncelikle Şekil 3'te verilen komut yörüngesi bölümleme yapılmaksızın sadece Chebyshev yaklaşıklaması yordamıyla ifade edilmiştir. Başarımın istenilen ölçüde olmamasından dolayı komut yörüngesi kesişim noktalarından itibaren bölümlere ayrıldıktan sonra Chebyshev yaklaşıklama yöntemi uygulanmıştır. Yaklaşıklama sonucunda oluşan hatalar ise önerilen sıkıştırma algoritması aracılığıyla saklanmıştır. Bu işlemler neticesinde özgün komut yörüngesi hatasız bir şekilde yeniden oluşturulabilmektedir.

Chebyshev fonksiyon yaklaşıklama yöntemi literatürde yaygın olarak kullanılmaktadır [7-8]. [-1,+1] arasında tanımlı olan Chebyshev polinomları, genel olarak sınırlı bir alanda peryodik olmayan verilerin modellenmesi için kullanılmaktadır. Esas itibariyle, bir Chebyshev fonksiyonu

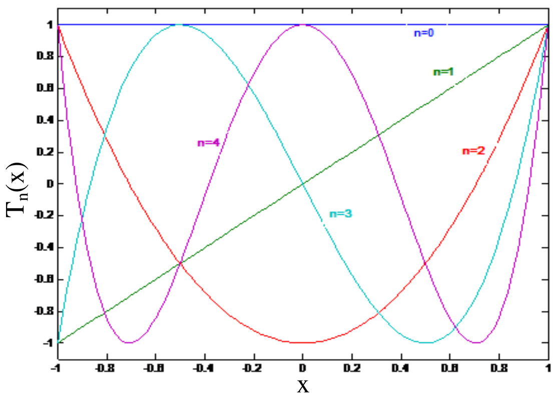
$$f(x) = \sum_{n=1}^{N} c_n T_n(x), \quad x \in [-1,\ 1] \tag{4}$$

şeklinde tanımlanmış olup, burada temel fonksiyonları teşkil eden polinomlar şöyledir:

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \tag{5a}$$

$$T_0(x) = 1, \quad T_1(x) = x \tag{5b}$$

Şekil 4'te dördüncü dereceye kadar olan Chebyshev polinomları (temel fonksiyonları) gösterilmiştir.

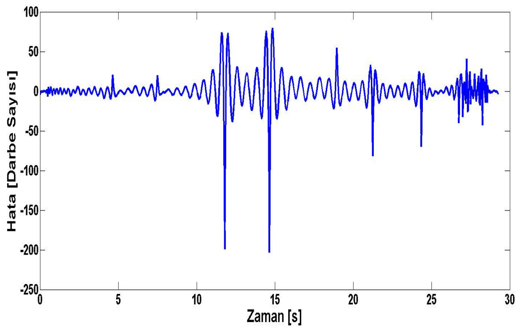


Şekil 4: Chebyshev Polinomları (Temel Fonksiyonları)

Dikkat edilirse, Chebyshev fonksiyon yaklaşıklama yönteminin kullanılabilmesi için öncelikli olarak verinin [-1,+1] aralığına indirgenmesi gerekmektedir. Komut yörüngesinin yaklaşıklamasının yapılabilmesi için gerekli olan katsayılar, MATLAB ortamında "Pseudo-Inverse" tekniği kullanılarak hesaplanmıştır. Yöntemin uygulanmasıyla Şekil 3'te gösterilen yörünge, farklı sayıdaki Chebyshev polinomlarıyla modellenmiş ve özgün yörünge ile arasında oluşan azami hatalar Tablo 3'te gösterilmiştir. Tablodan da anlaşılacağı gibi, kullanılan Chebyshev polinomlarının sayısı arttıkça iki yörünge arasındaki hata da azalmaktadır. Hatanın değişimini görmek amacıyla 150 adet Chebyshev polinomu kullanılarak yapılan yaklaşıklama sonucu ortaya çıkan hata, Şekil 5'te verilmiştir.

Tablo 3: Chebyshev Polinomları Sayısına Göre Oluşan Azami Hata

Chebyshev Polinomları Sayısı	Azami Hata [Darbe]
5	3443
10	1764
50	301
100	119
150	79



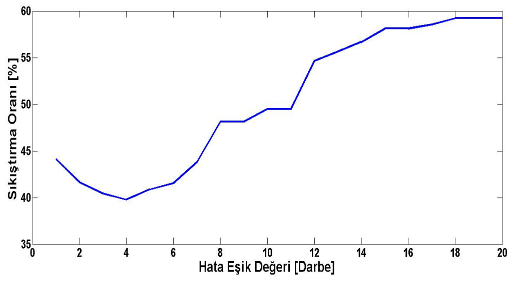
Şekil 5: Yaklaşıklama Sonucu Oluşan Hata

Kullanılan yöntem bir yaklaşıklama olduğundan dolayı hatanın sıfıra inmesi beklenmemektedir. 150 adet yüksek sayıda Chebyshev polinomu kullanılmasına rağmen elde edilen yörüngedeki hatanın makul değerlerde olmadığı görülmektedir. Bu nedenle yörüngenin tamamının bir fonksiyon yaklaşıklama yöntemiyle ifade edilmesi başarılı bir çözüm olarak değerlendirilememektedir.

Makul sayıda Chebyshev polinomu kullanarak, herhangi bir yörüngeyi hatasız olarak modellemek oldukça zordur. Şekil 5'te gösterilen hata grafiği, orijinal yörünge (Şekil 3) göz önüne alındığında, söz konusu yörünge parçalı kısımların bir araya gelmesinden oluştuğundan; yüksek hataların değişik kısımların kesişim noktalarında meydana geldiği görülür. Eğer yörünge bu noktalarından uygun bölümlere ayrılıp, her bölüm için ayrı ayrı Chebyshev polinomlarıyla yaklaşıklama yapılırsa, çok daha az polinom kullanımı ile birlikte makul hata seviyelerine inilebileceği anlaşılmaktadır. Her durumda hata kaçınılmaz olduğu için kayıpsız bir şekilde yörüngeyi yeniden temsil etmek amacıyla hataların bir tablo halinde saklanması gerekmektedir. Kullanılacak olan bu hata tablosunun daha az bellek kullanmasını sağlamak için, bu tablodaki verilerin basit bir algoritma ile sıkıştırılması öngörülmüştür. Hata verilerine uygulanan bu sıkıştırma yöntemi iki bölümden oluşmaktadır. Birinci kısımda veriler (değişken bit uzunluklarıyla) ardışık olarak sıralanmıştır. İkinci alanda ise, çözümlemenin yapılabilmesi için verilerin uzunluklarının ifade edildiği bir bit dizini kullanılmaktadır. Önerilen bu teknik üç temel adımda özetlenebilir:

1. Komut yörüngesinin bölümlenmesi;
2. Seçilmiş bir hata eşik değeri (rms) için, her bir bölümün Chebyshev polinomları vasıtasıyla modellenmesi;
3. Kalan hata verisinin özel bir veri şıkıştırma tekniği vasıtasıyla kayıpsız olarak temsil edilmesi.

Önerilen teknikte en önemli parametre seçilmiş olan hata eşik değeridir. Bu değerin küçük seçilmesi durumunda her ne kadar modelleme hatası düşüp, ilgili hata dizisinin genel boyutu azalsa da; Chebyshev polinomlarının ve ilgili katsayıların sayısı dikkate değer ölçüde arttığı görülmektedir. Dolayısıyla, uygun hata eşik değerinin belirlenmesi belirli bir eniyileme çalışmasının sonucunda ortaya çıkacaktır. Yöntemin farklı hata eşik değerlerine göre sağlamış olduğu sıkıştırma oranları Şekil 6'da gösterilmiştir.

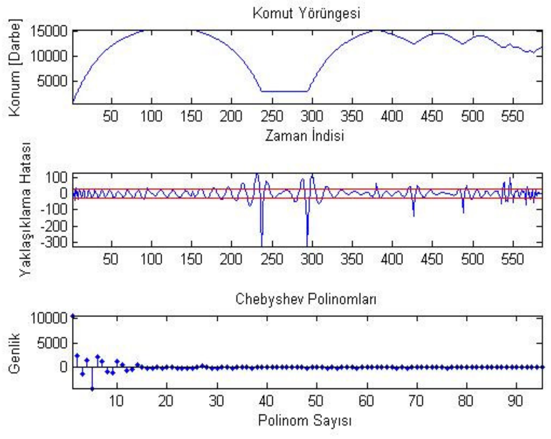


Şekil 6: Sıkıştırma Oranının Hata Eşik Değerine Göre Değişimi

Şekilden de görüleceği üzere, en düşük sıkıştırma yüzdesi hata eşik değerinin 4 (enkoder darbesi) olarak belirlendiği durumda ortaya çıkmaktadır. Şeklin başında görülen yüksek sıkıştırma oranların nedeni, istenilen hata ortalamasının karşılanabilmesi için yüksek sayıda Chebyshev katsayısı kullanılmasıdır. Hata eşik değerinin 4 darbe sayısını geçtiği durumlarda sıkıştırma yüzdesinin yükselişe geçmesinin nedeni ise hatayı içeren dizi için gerekli olan belleğin artışıdır.

4. Yöntemlerin Karşılaştırılması

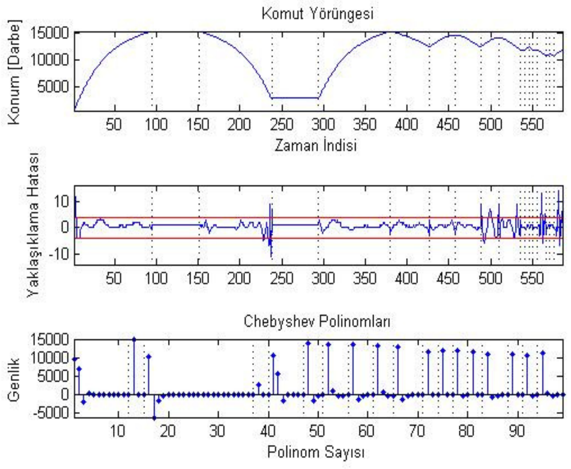
Üçüncü bölüm çerçevesinde önerilen yöntemlerin birbirlerinden temel farkları ikinci yöntem içerisinde uygulanan bölümleme ve yaklaşıklama sonucu ortaya çıkan hatanın sıkıştırılmasıdır. Herhangi bir bölümleme yapılmadan yörüngenin tamamının Chebyshev polinomları yardımıyla yaklaşıklamasının yapıldığı birinci yöntemin sonuçları Şekil 7'de gösterilmiştir.



Şekil 7: Chebyshev Yaklaşıklaması Sonuçları

Üçlü bileşik grafikten oluşan bu şekilde karşılaştırmanın rahatça yapılabilmesi için komut yörüngesi en üstte verilmiştir. Yaklaşıklama sonucu oluşan hatalar ortadaki grafikte gösterildikten sonra en altta ise Chebyshev katsayılarının büyüklüklerine yer verilmiştir. Şekilden de anlaşılacağı üzere oluşan yüksek hatalar ve katsayıların büyüklükleri yöntemin dezavantajı olarak görülmektedir. Bu sebeplerden ötürü de genel sıkıştırma oranı hiçbir zaman %75'in altına düşmemektedir.

Diğer taraftan, hata eşik değeri 4 olarak seçilip, bu makalede önerilmiş olan yöntem kullanıldığında; %44 civarında genel sıkıştırma oranı elde edilmiştir. İlgili sonuçlar Şekil 8'da sunulmuştur.



Şekil 8: Bölümleme ve Chebyshev Yaklaşıklaması Sonuçları

Oluşan hataların genel olarak hata eşik değerinin arasında olmasından dolayı sıkıştırılma sırasında hata büyüklüklerin daha az bit kullanılarak ifade edilmesine olanak sağlamaktadır. Bunun yanı sıra, Şekil 8 ve 9 beraber değerlendirildiğinde kullanılan polinom sayısının ve genliklerinin çok farklı olmadığı görülmektedir.

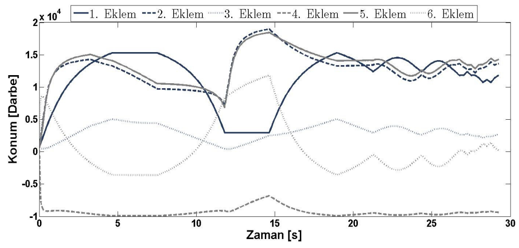
Hata eşik değerinin sıkıştırma oranına olan etkisini göstermek amacıyla farklı hata eşik değerleri için uygulanan yöntem neticesinde alınan sonuçlar Tablo 4'te verilmiştir.

Tablo 4: Hata Eşik Değerine Göre Bellek Kullanımı

Hata Eşik Değeri	5	10	15	20
Katsayılar için Gerekli Bellek [Bayt]	182	146	120	114
Hata Dizisi için Gerekli Bellek [Bayt]	297	434	561	580
Sıkıştırma Oranı [%]	40.9	49.5	58.1	59.2

Tablodan da anlaşıldığı üzere hata eşik değerinin artmasıyla beraber genel sıkıştırma oranı da yükselmektedir. Bunun temel nedeni büyüklüğü artan hataların sıkıştırılması için gerekli olan bellek gereksiniminin artmasıdır. Öte yandan artan hata eşik değeriyle birlikte Chebyshev yaklaşıklaması için gerekli olan belleğin azalması ise kullanılan polinomlarının sayısının düşüşüne bağlanabilir.

Makalenin ikinci kısmında önerilen yöntemlerden öncelikle üçüncü dereceden sonlu farkın alınıp sonrasında Aritmetik kodlamanın yapıldığı teknik, yöntemlerin en başarılısı olarak görülmektedir. Üçüncü kısımda önerilen yöntemlerden ise bölümlemenin ardından Chebyshev polinomlarıyla yapılan fonksiyon yaklaşıklaması tekniği bariz bir şekilde öne çıkmaktadır. Bahsedilen bu iki yöntemin başarımlarını karşılaştırabilmek amacıyla yöntemler, Şekil 9'da gösterilen PUMA 560 manipülatörüne ait yörüngeler üzerinde sınanmıştır.



Şekil 9: Eklem Komutlarının Zamana Göre Değişimi

Her bir eklem yörünge komutları için ayrı ayrı uygulanan yöntemlerin sıkıştırma sonuçlarına Tablo 5'te yer verilmiştir.

Tablo 5: Sıkıştırma Oranlarının [%] Karşılaştırılması

Eklem Numarası	Yöntem 1	Yöntem 2
1	32.9	40.8
2	41.4	47.5
3	35.4	38.7
4	36.2	42.2
5	43.4	48.4
6	39.2	42.2

Bu tabloda ***Yöntem 1***, 3. derecededen farkın Aritmetik kodlama tekniğiyle sıkıştırılması sonucu elde edilen sonucu; ***Yöntem 2*** ise hata eşiği 5 (darbe olarak) şeçildiğinde üçüncü bölümde önerilen tekniğin sonuçlarını göstermektedir. Tablo 5'ten de görüleceği üzere, kayıpsız olarak yörüngenin yeniden oluşturulmasını sağlayan bu iki yöntemin başarımı birbirleriyle kıyaslanabilir durumdadır. Tablo ayrıntılı bir şekilde incelendiğinde yörünge farklılığına bağlı olmaksızın birinci yöntemin başarımının diğerine göre biraz daha iyi olduğu görülmektedir. Sonuç olarak, istenilen yörüngenin mümkün olduğunca az bellek (ve işlem gücü) kullanılarak oluşturulabilmesi için, Yöntem 1'in diğerine göre daha avantajlı olduğu anlaşılmaktadır.

5. Tartışma

Bu çalışmada herhangi bir komut yörüngesini daha az bellek alanı kullanarak temsil edebilecek yöntemlerden en önemlileri sunulmuştur. Temel olarak veri sıkıştırma ve fonksiyon yaklaşıklama yordamlarının kullanıldığı bu yöntemler vasıtasıyla genel olarak 1:2 oranında sıkıştırmanın yapılabileceği gösterilmiştir.

Sıkıştırma yapılmadan önce komut yörüngesinin üçüncü dereceden sonlu farkının alınması, sıkıştırma oranlarının büyük ölçüde azalmasına neden olmuştur. Farkı alınmadan yapılan sıkıştırmaların hiçbir anlam ifade etmediği denemeler sonucunda ortaya çıkmıştır.

Chebyshev polinomları tabanlı fonksiyon yaklaşıklama yordamının genel olarak kullanıldığı yöntemlerde özgün yörüngenin hatasız bir şekilde ifade edilmesi mümkün olmadığından, yörüngenin hatasız biçimde üretilebilmesi için, modelleme hatalarının ayrıca saklanması gerekmektedir. Dolayısıyla, ikinci tip yöntemler çerçevesinde yörüngeye uygulanan dinamik bölümleme, yaklaşıklama sonucunda oluşan hataların büyük ölçüde azalmasına olanak sağlamıştır. Bununla birlikte sıkıştırma oranı, birinci paket yöntemlerle elde edilen sıkıştırma oranlarıyla karşılaştırılabilecek duruma gelmiştir.

Gelecek çalışmalarda, bu makalede önerilen yöntemler sonucunda oluşan sıkıştırılmış verilerin çözümlenebilmesi için ***alan programlanabilir kapı dizini*** (Field Programmable Gate Array - FPGA) tabanlı donanımsal çözümlere ağırlık verilecektir.

6. Teşekkür

Bu çalışma, TÜBİTAK'ın 108E048 kodlu projesi kapsamında yapılmıştır. Makalenin yazarları sağladığı finansal destekten ötürü TÜBİTAK'a teşekkür ederler.

7. Kaynakça

[1] A. Akıncı, "Universal Command Generator for Robotics and CNC Machinery," Orta Doğu Teknik Üniversitesi Fen Bilimleri Enstitüsü Yüksek Lisans Tezi, 2009.

[2] D. Maroulis, N. Sgouros ve D. Chaikalis, "FPGA-based architecture for real-time IP video and image compression," IEEE International Symposium on Circuits and Systems Proceedings, ISCAS, s: 5582, 2006.

[3] D. A. Huffman, "A method for the construction of minimum redundancy codes," Proceedings of IRE, Cilt: 40, No: 10, s:1098-1101, 1952.

[4] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," Communications of the ACM, Cilt: 30, s: 520-540, 1987.

[5] H. Hassler ve N. Takagi, "Function evaluation by table look-up and addition," 12th IEEE Symposium on Computer Arithmetic, s: 10-16, 1995.

[6] J. A. Pineiro, J. D. Bruguera ve J.-M. Muller, "Faithful powering computation using table look-up and a fused accumulation tree," Proceedings of 15th IEEE Symposium on Computer Arithmetic, s: 40-47, 2001.

[7] F. de Dinechin ve A. Tisserand, "Some improvements on multipartite tables methods," *IEEE Transactions on Computers,* Cilt: 3, s: 319-330, 2005.

[8] A. Ashrafi, Z. Pan, R. Adhami ve B. E. Wells, "A novel ROM-less direct digital frequency synthesizer based on Chebyshev polynomial interpolation," Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory, s: 393-397, 2004.

[9] A. Ashrafi, R. Adhami, L. Joiner ve P. Kaveh, "Arbitrary waveform DDFS utilizing Chebyshev polynomials interpolation," *IEEE Transactions on Circuits and Systems I: Regular Paper*, Cilt: 51, No: 8, s: 1468-1475, 2004.

[10] J. Jiang ve E. K. F. Lee, "Segmented sine wave digital-to-analog converters for frequency synthesizer," Proceedings of International Symposium on Circuits Systems, ISCAS'01, Cilt: 1, s: 520-523, 2001.

[11] H. K. Reghbati, "Special Feature An Overview of Data Compression Techniques," *Computer* , Cilt: 14, No: 4, s: 71-75, 1981.

A Novel Command Generation Method with Variable Feedrate utilizing FGPA for Motor Drives

Ulas Yaman, Melik Dolen, A. Bugra Koku
Department of Mechanical Engineering
Middle East Technical University
Ankara 06531, Turkey
{uyaman, dolen, kbugra}@metu.edu.tr

Abstract—**This paper focuses on a novel command generator for servo-motor drives to be used as an integral part of their motion controllers. The method, which incorporates a new data compression algorithm, is capable of generating trajectory data at variable rates. In this paradigm, higher-order differences of a given trajectory (i.e. position) are first computed and thus the resulting data are compressed via the proposed technique. The generation of the commands is carried out according to the feed-rate (i.e. the speed along the trajectory) set by the external logic dynamically. The paper discusses the implementation of the method on a Field Programmable Gate Array (FPGA). During implementation Very High Speed Integrated Circuit Hardware Description Language (VHDL) is used rather than using embedded processors on the FPGA chip. The performance of the method is assessed according to the resources used in the FPGA chip on the development board and also compared with the same approach without an interpolator.**

Keywords-command generation; data compression; FPGA; adjustable feedrate; linear interpolation

I. Introduction

Modern servo drive systems employ digital motion controllers (DSPs, micro-controllers) to regulate precisely not only motor currents (electromagnetic torque) but also motor's angular velocity along with the position. If the drive system is configured for (digital) motion control, the relevant reference signals (velocity or position) must be generated by a central controller unit (host) depending on the trajectory to be followed. These signals are eventually transferred to each motor driver via a serial communication protocol (SERCOS, CAN, Profibus, TCP/IP, RS-232, RS-485, etc.). This approach frequently pushes the communication interface to its limits for high-end applications. Consequently, the objective of this study is to develop a direct command generator system with variable feedrate for servo motor drives where the commands could be produced directly in the drive system without the need for intermittent data transfer from a host controller. This FPGA based system, which could be directly embedded into a motor drive system, is expected to generate the relevant commands by utilizing not only the (dynamically adjusted) speed along the traced trajectory but also decompressed data being produced in advance to represent trajectory to the desired accuracy.

Industrial motion controller units utilize vector data tables to represent the trajectory in terms of linear patches. These cards can then perform a linear interpolation between the two consecutive entries in real-time to produce the relevant reference signals for the position servo-control loop. For complex trajectories the size of the vector table may exceed the available resources on the system. The conventional machining approach does not meet the requirements of high speed and high accurate machining. To overcome these difficulties, the proposed method is implemented into the conventional manufacturing process as illustrated in Fig. 1. One of the major advantages of the method is that it can send position, velocity, and acceleration commands simultaneously to the controller. The purpose of compressing the servo commands is to decrease the data transfer load and feedrate is given as an input to the proposed method to reduce the error along the curves where the radius of curvature is small.

In today's technology, memory devices with large capacity as well as multi-core processors running at high clock frequencies are widely available in the market at relatively low cost. Consequently, there is a potential for devising simple yet very effective command generators for computer numerically controlled machinery that benefit fully from the properties of these advanced devices [1].

This paper focuses on a novel technique composed of three

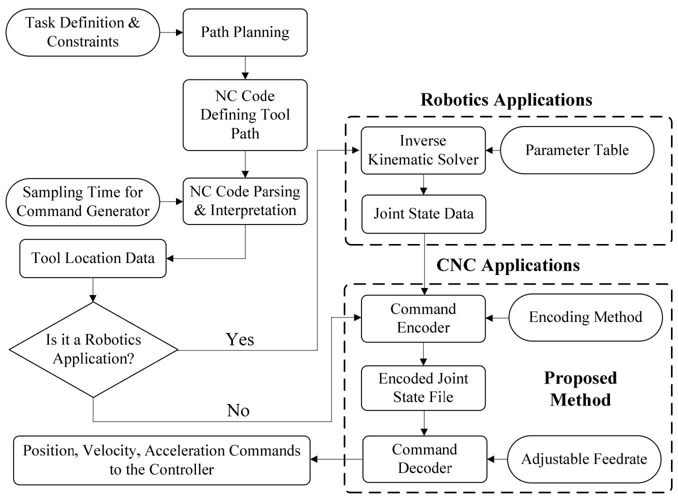


Figure 1. CNC / Robotics Applications

This work was supported by the Scientific and Technological Research Council of Turkey under the project contract 108E048.

consecutive parts: the first part, which is commonly referred as differencing in literature [2], computes the higher order differences of the trajectory data (e.g. position). Second part reduces the size of the resultant data utilizing a novel (lossless) compression technique. The last part of the method generates the commands according to the feedrate input.

The rest of the paper is organized as follows: Section II represents a review of the relevant research fields. Section III describes the FPGA-based command generation method using differencing plus compression and its hardware implementation is presented in Section IV. Section V presents the results of the proposed method employed on the test case and compares the results without the feedrate input. Finally, Section VI concludes this paper.

II. Background

In this section, the literature laying the basis of the proposed technique is presented in three different topics. In the first part, examples of various data compression methods implemented on FPGA are reviewed. In the second part, FPGA-based command generation systems as well as methods are presented. Finally, the CNC applications with variable feedrate are investigated.

A. FPGA Implementations of Data Compression Methods

Data compression is regarded as the crucial component in high-speed data transfer and storage. Two types of compression exist in the literature: lossless and lossy. In lossless data compression, original data is encoded to a smaller data set that can later be decoded back to recover the original data. Whereas in lossy compression, the original data can only be approximated after decompression. Lossless data compression, where the original data is extracted without any loss after decompression, applications have increased over the past years due to the need to improve the storage capacity and data transfer rate [3]. There are many examples for the hardware implementations of conventional data compression techniques in the literature. Among these techniques, Huffman [4] [5], Lempel-Ziv (LZ) [6] [7], and Golomb [8] compression algorithms are the most popular ones for FPGA implementations. For instance, Rigler, Bishop, and Kennings [4] implemented Huffman and LZ encoders on an FPGA and concluded that modified Huffman algorithm uses less hardware resources than the LZ algorithm. On the other hand, Abd El Ghany, Salama, and Khalil [6] also realized the LZ encoding and decoding algorithm on FPGA. In order to increase the efficiency, they used systolic array which resulted in a 40% decrease in the compression rate.

Among conventional data compression techniques, hardware implementations of different algorithms for compressing specific data structures are also present in the literature. For instance, Yongming, Jungang, and Jianmin [9] have realized the Linear Approximation Distance Threshold algorithm on FPGA to compress the Electrocardiograph signals. Similarly, Valencia and Plaza [10] developed an FPGA-based data compression technique based on the concept of spectral unmixing to compress hyperspectral data. The novel compression method described in the paper can be regarded as task-specific since it is developed to compress the integer position command sequences. It may not yield better results for text or image compression.

B. FPGA-Based Command Generation Systems

Implementations of command generation methods on FPGA chips are not very common in the literature due to high computational complexity involved in the methods. Therefore, techniques employed on FPGA have simplifications and/or include error compensation modules into the systems. For instance, Su, Hu, and Cheng [11] developed a motion command generation chip utilizing FPGA for point-to-point motion applications. They implemented trapezoidal and S-curve motion planning adopting the digital convolution method rather than the complex polynomial type method. With this approach, the computational complexity is significantly decreased. Furthermore, they developed a real-time output pulse compensation algorithm to eliminate the error in the number of output pulses and the results are found to be satisfactory.

Jeon and Kim [12] also used the digital convolution method and designed an FPGA-based acceleration and deceleration circuit for industrial robots and CNC machine tools. Likewise the method developed by Su, Hu & Cheng [11], they did not use the complex polynomial technique to generate velocity profiles of various acceleration and deceleration characteristics that requires much computations. Since the current techniques are not satisfactory for generating velocity profiles for industrial robots and CNC machine tools [13], they developed a new method to compensate this deficiency. According to the experimental results given in the paper, they were able to generate unsymmetrical velocity profiles that cannot be generated by digital convolution techniques. Comparing the two works, former one is superior over the latter method. The error in the output pulses is not compensated in Jeon and Kim [12]'s study, so the errors are inevitable between the command and response signals. On the other hand, the method proposed in the paper generates commands without any error. Furthermore, it generates position, velocity and acceleration profiles at the same time.

C. Feedrate Control of CNC Machine Tools

The precision of the final mechanical product is getting better as the manufacturing technology improves. In the manufacturing process, the quality of the product is dependent on the functions of CNC machine. Feedrate control of the machine tool is very important factor for a high-precision CNC machine.

There are various algorithms proposed on feedrate control in order to increase the surface quality of the product. For instance Cheng, Tsai, and Maciejowski [14] employed a predictor-corrector algorithm in order to estimate the servo command at the next sampling time. In another study of Cheng, Tsai, and Cheng [15] developed a new interpolator to produce servo commands for real-time control of CNC machining. The main advantage of the proposed interpolator is being capable of generating motion commands for servo controllers at variable feedrates. In a similar study of Xu, Tam, Zhou, and Tse [16], they presented variable interpolation schemes for planar implicit curves. They were also able to interpolate in real-time to improve machining efficiency. In the proposed method, the feedrate is set by the operator according to geometrical state of the surface. In other words, it is decreased when the tool is machining curved parts and increased on planar surfaces.

III. Proposed Method

In this section of the paper, the method suggested for command generation is elaborated. After the advantages of taking higher order differences of the command trajectory are mentioned, the encoding algorithm is explained and the compressed file format is given. Then the decoding algorithm of the suggested command generation method is explained and finally the interpolator is investigated.

A. *Encoding Algorithm*

According to the previous study of authors [1], taking third order differences of the command trajectory before compressing does increase the compression ratio. In the method after the difference of the command (position) sequence is calculated according to the specified order, data is compressed utilizing the suggested algorithm in [1]. In Fig. 2, a sample encoding process for the third-order difference is illustrated. In order to use the memory efficiently, a compressed code structure is developed in the previous study of authors [17].

B. *Decoding Algorithm*

Decoding starts with the comparison of the consecutive two bits of the length field. After the binary length of the data is determined, it is passed to a left shift register. Considering the length of the data, register forwards the amplitude of the data from the corresponding field to the differenced data module. After the sign value is obtained from the sign field, the data is sent to the accumulator module. Then the initial values are transferred to the integration module in the proper order and original data is generated from the buffer.

C. *Interpolator*

The interpolator used in the method is a linear type. It interpolates between the two original command values, decoded from the compressed code, according to the current status of the feedrate according to the below equations.

$$u_k = u_{m-1} + \left(u_m - u_{m-1}\right)\left(a\right) / f_{\max} \qquad (1)$$

$$a_k = \begin{cases} a_{k-1} + f_k, & a_{k-1} < 1 \\ \left(a_{k-1} - 1\right) + f_k, & a_{k-1} > 1 \end{cases} \qquad (2)$$

Original Command Sequence = {555, 983, 1354, 1710, 2058, 2400, 2736, 3068, 3394, 3715, 4031, 4341, 4646, ...}

3rd Order Difference = {42, 7, 2, 0, 2, -2, 1, 0, -1, 1, ...}

Encoded Data

Initial Values = {555, 428, -57}

Sign Field :	0	0	0	0	0	1	0	0	1	0	···
Amplitude Field :	101010	111	10	0	10	10	1	0	1	1	···
Length Field :	111111	000	11	0	11	00	1	0	1	0	···

Figure 2. Encoding Process of the Proposed Method

In the above equations u represents the commands according to their indices. k and m are the indices of the generated and decoded sequences, respectively. f_k is the current feedrate given as input to the system and f_{max} is the maximum feedrate at which commands can be generated. Another variable in the equations is the difference multiplier, a. It accumulates after each generation according to (2). During the implementation of this equation, the next original command should be fed to the interpolator when the interpolated command is out of the difference region. In Fig. 3, a sample interpolation is carried out with a feedrate of (0.375)(f_{max}). As can be seen from the figure that before the 4th interpolated command is generated, the difference value is updated and the next three commands are generated according to the new difference value. The shaded area underneath the 2nd original command can be regarded as the error of the interpolating algorithm. In future works, it is aimed to develop new algorithms to decrease the error at original command points.

IV. Implementation

The implementation of the proposed method is realized on an Altera FPGA DE1 Development Board, which contains a number of memory devices (SRAM, SDRAM, SD Card, Flash) ready to be used in conjunction with the FPGA chip (Cyclone II). Among these chips ISSI IS61LV25616 SRAM is selected due to its ease of control. It is organized as 256K words by 16 bits. To decrease the complexity of the coding in VHDL, the schematic design property of Quartus II 9.0 Web Edition is employed. The schematic of command generator shown in Fig. 4 basically consists of 6 different modules: SRAM Controller, Memory Management Unit, Decoding Unit, Accumulators, Interpolator and RS-232 Controller. The following sections elaborate the design of these units.

A. *SRAM Controller*

The main task of the SRAM controller is to maintain the communication between the Memory Management Unit and the SRAM located on the FPGA Development Board. It sends out the compressed data to the Memory Management Unit (one by one) according to the address information emanating from the Memory Management Unit.

B. *Memory Management Unit*

The Memory Management Unit (MMU) can be regarded

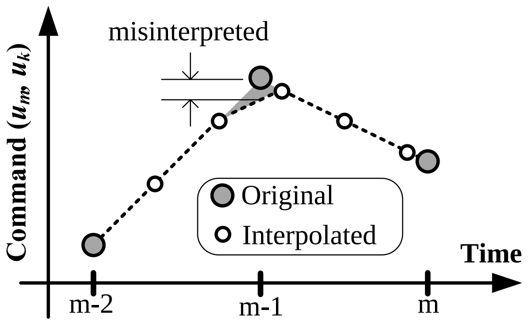


Figure 3. Interpolated Data

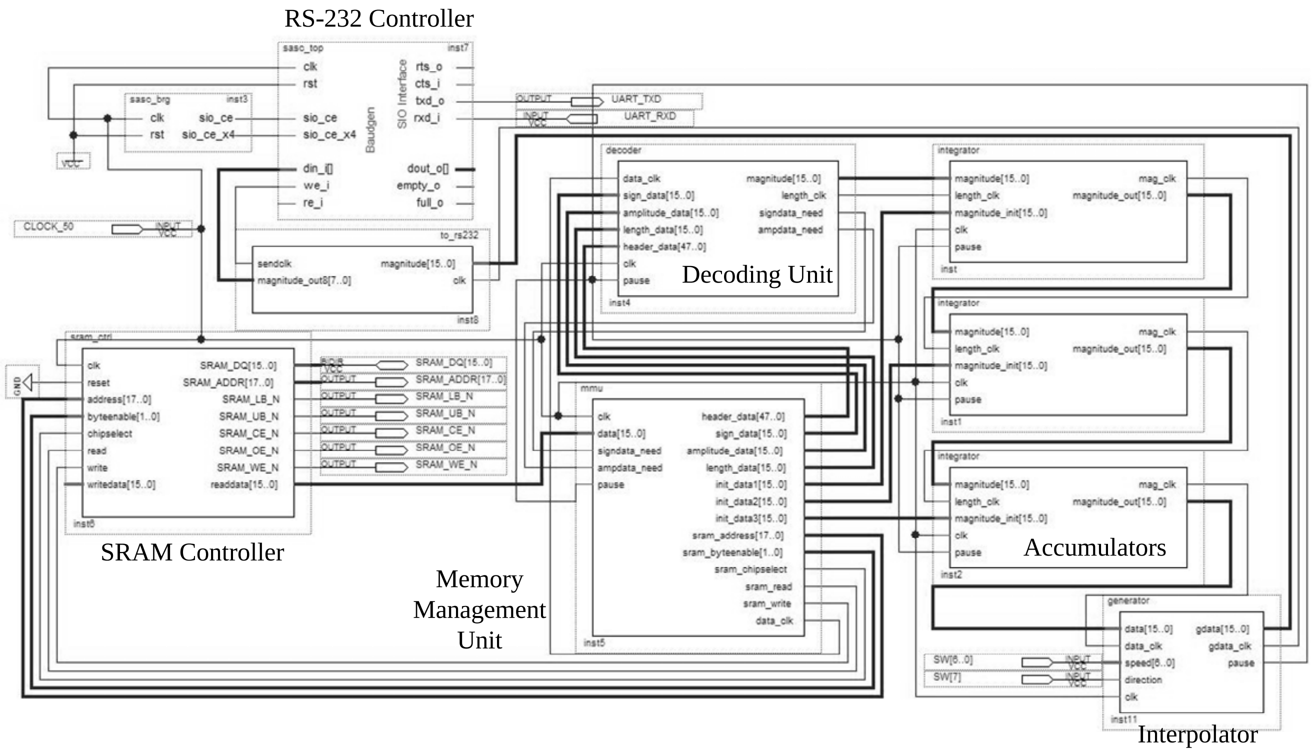


Figure 4. Schematic Design of Command Generator with Variable Feedrate

as the core of the design since it communicates with all modules except the RS-232 Controller. Input signals to this module are limited when the number of output signals is considered. Input signals are only the data sent from the SRAM Controller and acknowledgment signals coming from the Decoding Unit (DU) indicating that the unit is out of data. Output signals are

- the initial values sent to the accumulators,
- header data for the compressed file,
- three fields transferred to the DU.

The basic operating principles of the MMU are described in Fig. 5. As can be seen, there are four states of this unit: i) get header, ii) get initial conditions, iii) fetch first set, iv) send & wait. After the system is reset, the unit starts acquiring header data (words) from the SRAM and sending them to the DU. In the next state, the initial values are conveyed to the accumulators in a proper order. In the following state, the first set of words are fetched from the SRAM and are sent to the DU to initiate the decoding process promptly. In last state, the words from each field are sent to the DU. This state is only initiated when the incoming signals ‘signdata_need’ and ‘ampdata_need’ are set. Note that there is no signal indicating the necessity for a data point from the length field. When a word from the amplitude field is needed, the corresponding word from the length field is sent automatically to the DU. This state lasts until all the commands are generated.

C. Decoding Unit

DU is the module where the decoding algorithm is implemented. It communicates with the MMU and the first accumulator module. All the input signals to this unit are fed from the MMU. Two of the output signals are the acknowledgment signals which are described in the previous article. The remaining two output signals are directly connected to the first accumulator. Thus, the decoded command is transferred to the accumulator in signed integer format at an additional clock indicating that a new command is being submitted. The basic operating principles of the DU are depicted in Fig. 6. Decoding that constitutes nine states starts when the header data from the MMU are acquired. Then, the header data (constituting the order of difference, length of the command sequence, and the number of amplitude field words) are divided and stored for further use. In the second state, the first set of words from three different fields is saved. Second set received from the MMU is stored in the third and fourth states. While decoding, the second set is necessary since it may turn out that the corresponding command is distributed between two consecutive words. The main task of this unit is executed in the ‘Decode’ state of

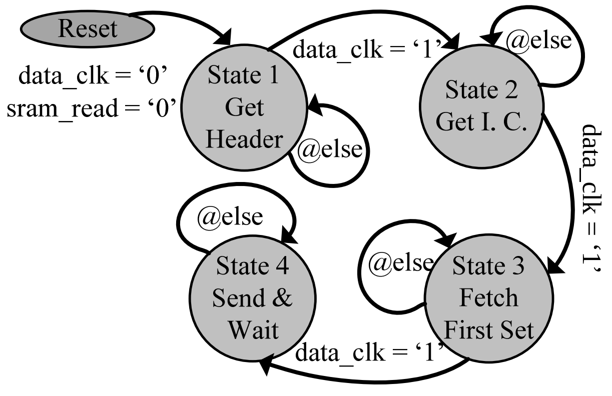


Figure 5. State Diagram of Memory Management Unit

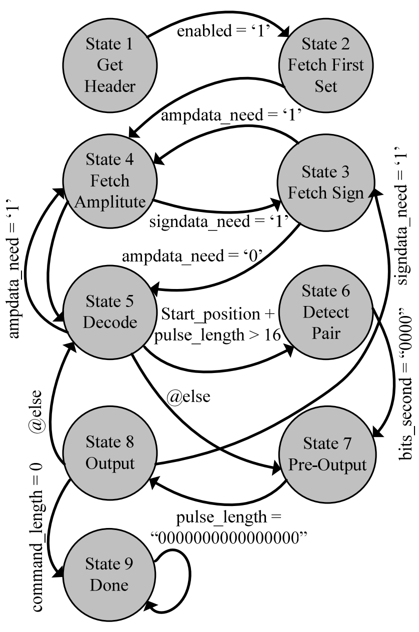


Figure 6. State Diagram of Decoding Unit

which is associated with five other states. When there is a lack of data during decoding, the finite-state “decoding” machine moves either on to ‘Fetch Amplitude’ or ‘Fetch Sign’ states to obtain the required data. If the decoding is complete for a given command, the data (in unsigned integer format) are processed in the ‘Pre-Output’ state. In case the corresponding command is stored in two different words, ‘Detect Pair’ state takes over for proper decomposition. The conversion of unsigned to signed integer format is performed in the ‘Output’ state. For this purpose, the data from the sign field must be ready. When sign data run out, the DU moves onto the third state and gets the necessary data. After the decoded command is formed as signed integer, it is sent to the first accumulator instance.

D. Accumulators

Accumulator (integrator) modules are the simplest elements of this design. It gets the input data, sums it with the previous value of the accumulator and outputs the resulting value to the next accumulator. The number of accumulators in the design depends on the order of difference. Note that the given design in Fig. 7 is hard-wired and can decompress data differentiated up to the third order. However, the general design should have 15 accumulator instances (in compliance with the format specified in Section 3.1). A demultiplexer unit must be incorporated to the design to deselect the unused accumulator instances. Notice that in the proposed design, the three accumulators yield the acceleration, velocity, and position profiles of the commanded trajectory. This attribute is one of the advantages of the proposed method. Since when a state-space controller is embedded into the system, the velocity and acceleration profiles must be ready for use.

E. Interpolator

The interpolator in the schematic design simply performs the computations described with (1) and (2). While generating the commands, it sends ‘pause’ signals to the DU, MMU, and accumulators to stop their operations. When there is need for a new original command, it sets the ‘pause’ signal to low. In order to overcome the delays between the generated commands, it also employs a buffer inside.

F. RS-232 Controller

The RS-232 Controller is implemented in the design for monitoring purposes. Instead of the RS-232 Controller, the output of the architecture may be connected to any other serial data transfer controller (RS-485, TCP/IP, CAN, etc.). Since the controller is designed to transfer bytes, a splitter module is implemented after the output of the last accumulator (of 16 bits).

V. RESULTS AND COMPARISON

In a previous study [1], the proposed compression algorithm is rigorously tested in MATLAB environment. Its performance is compared to those of two other (popular) compression methods: Huffman and Arithmetic Coding. The proposed compression algorithm is far more superior to the above-mentioned (lossless) algorithms in terms of compression ratios within the given context.

The proposed command generation method with variable feedrate is evaluated in MATLAB before implementing it on an FPGA development board. The feedrate profile given in Fig. 7 is applied on the original command trajectory illustrated in Fig. 8 with straight line. There occur command representation errors (Fig. 9) at each original data points since the interpolation algorithm, described in the previous sub-section, is not capable of generating commands at these points. In further study, it is planned to develop algorithms to eliminate or decrease these type errors in the trajectory. When Fig. 8 and Fig. 9 are considered together, it can easily be inferred that larger errors occur at the inflection points of the trajectory.

The proposed command generation method described in the previous sections is then applied to the command sequences generated for all six joints of a PUMA 560 manipulator. Fig. 10 shows the trajectory of the first joint in terms of encoder pulses. Note that the optical position encoders used in this study can generate 40000 pulses/rev (= 4 × 10000 pulse/rev) in 4X quadrature decoding mode. Decompressor algorithm of the proposed method is realized utilizing Altera Cyclone II FPGA DE1 Development Board. Table I shows the obtained results in terms of allocated resources in the FPGA chip for the proposed method with and without the interpolator. As can be understood from the table that the main difference between the two cases is that the one with the interpolator uses 2 embedded multipliers in order to perform interpolation. This computational effort also results in an increase in the number of total logic elements.

TABLE I. FPGA RESOURCES USED

Resources	**With Interpolator**	**Without Interpolator**
Total Logic Elements	1728 (8%)	1105 (6%)
Total Combinatorial Functions	1482 (8%)	825 (4%)
Dedicated Logic Registers	910 (5%)	577 (3%)
Embedded Multipliers	2 (4%)	0

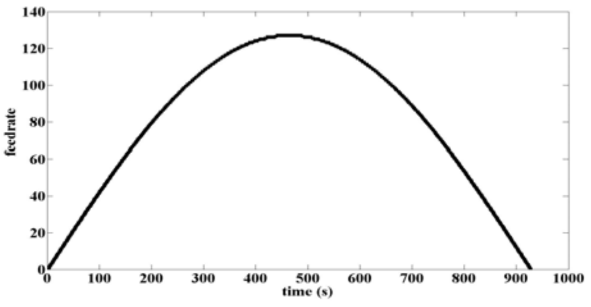


Figure 7. Feedrate Profile

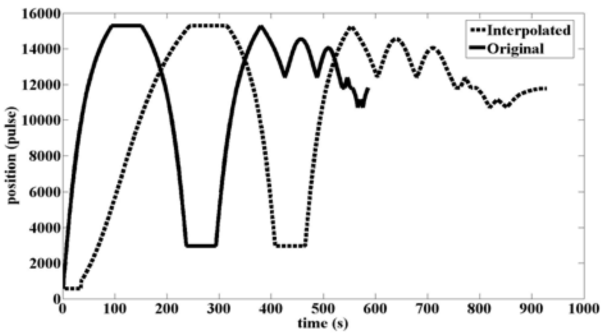


Figure 8. Interpolated and Original Command Sequences

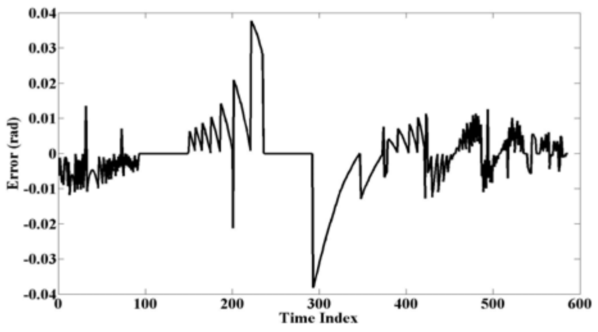


Figure 9. Command Representation Errors

VI. Conclusions

In the paper, an efficient method that is able to generate command sequences with variable feedrate for motion control applications is presented. Basically, differencing and data compression techniques are utilized to represent the command trajectories without the loss of generality. With the proposed method, original data sequence can be compressed to approximately one-third of its initial size and can be generated faster than any other methods. With the advantage of the variable feedrate property, during manufacturing processes the operator can increase or decrease the feedrate according to the geometric status of the surface of the product, which results in an increase in the machining quality and precision.

Taking higher differences of the trajectory before compression is essential for efficiency. After differencing, the frequency of numbers in the sequence increases and data compression techniques make more sense. The novel compression method suggested is not a universal technique. The method takes advantage of the fact that the command sequences in motion control are composed of integers.

The implementation results showed that the addition of an interpolator to the hardware design resulted in an increase in the use of FPGA resources due to the computational effort. In further studies, the proposed command generation method will be modified to generate in the reverse direction also. With this improvement in manufacturing processes the quality will further be enhanced.

References

[1] U. Yaman, B. R. Mutlu, M. Dolen, and A. B. Koku, “Direct command generation for electrical servo motor drives,” Proceedings of the 12th International Conference on Electrical Machines and Systems, IEEJ Industry Applications Society, Tokyo, pp. 1-6, November 2009.

[2] H. K. Reghbati, “Special feature an overview of data compression techniques,” Computer, vol. 14, no. 4, pp. 71-75, 1981.

[3] K. Dickson, “Cisco IOS Data Compression,” Cisco Syst., San Jose, CA, 2000.

[4] S. Rigler, W. Bishop, and A. Kennings, “FPGA-Based lossless data compression using Huffman and LZ77 algorithms,” Canadian Conf. on Electrical and Computer Engineering, pp. 1235-1238, 2007.

[5] T. M. U. De Araujo, E. R. Pinto, J. A. G. De Lima, and L. V. Batista, “An FPGA implementation of a microprogrammable controller to perform lossless data compression based on the Huffman algorithm,” 13th IBERCHIP Workshop, 2007.

[6] M. A. Abd El ghany, A. E. Salama and A. H. Khalil, “Design and implementation of FPGA-based systolic array for LZ data compression,” IEEE Int. Symposium on Circuits and Systems, pp.3691-3695, 2007.

[7] W. Cui, “New LZW data compression algorithm and its FPGA implementation,” Picture Coding Symposium 2007, Portugal, 2007.

[8] G. H. H’ng, M. F. M. Salleh and Z. A. Halim, “Golomb coding implementation in FPGA,” Elektrika Journal of Electrical Engineering, pp. 36-40, 2008.

[9] Y. Yongming, L. Jungang, and W. Jianmin, “LADT arithmetic improved and hardware implemented for FPGA - Based ECG data compression,” Proceedings of 2nd IEEE Conference on Industrial Electronics and Applications, pp.2230-2234, 2007.

[10] D. Valencia, and A. Plaza, “FPGA-Based hyperspectral data compression using spectral unmixing and the pixel purity index algorithm,’ Computational Science, pp.881- 891, 2006.

[11] K. H. Su, C. K. Hu, M. Y. Cheng, “Design and implementation of an FPGA-based motion command generation chip,” Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, 2006.

[12] J. W. Jeon, and Y. K. Kim, “FPGA based acceleration and deceleration circuit for industrial robots and CNC machine tools,’ Mechatronics, vol. 12, pp. 635-642, 2002.

[13] J. W. Jeon, “An efficient acceleration for fast motion of industrial robots,” Proceedings of IEEE 21st IECON, pp. 1336–41, 1995.

[14] C. W. Cheng, M. C. Tsai, and J. Maciejowski, “Feedrate control for non-uniform rational B-spline motion command generation,” Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, vol. 220, pp. 1855-1861, 2006.

[15] C. W. Cheng, M. C. Tsai, and M. Y. Cheng, “Real-time variable feedrate parametric interpolator for CNC machining,” 15th IFAC World Congress, Barcelona, Spain, 2002.

[16] H. –Y. Xu, H. –Y Tam, Z. Zhou, and P. W. Tse, “Variable feedrate CNC interpolation for planar implicit curves,” Advanced Manufacturing Technology, vol. 18, pp. 794 - 800, 2001.

[17] U. Yaman, M. Dolen, and A. B. Koku, “A novel FPGA-based command generation method for servo-motor drives,” To appear in the Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics, Funchal, Portugal, 2010.

TÜBİTAK
PROJE ÖZET BİLGİ FORMU

Proje No: 108E048	
Proje Başlığı: Kişisel Bilgisayar Tabanlı Evrensel Hareket Denetleyici Sistemlerinin Geliştirilmesi	
Proje Yürütücüsü ve Araştırmacılar: Y. Doç. Dr. Melik DÖLEN, Y. Doç. Dr. A. Buğra KOKU, Mak. Müh. Serdar ÜŞENMEZ, Mak. Müh. Barış R. MUTLU, Mak. Müh., Ulaş YAMAN, Mak. Y. Müh. Ergin KILIÇ	
Projenin Yürütüldüğü Kuruluş ve Adresi: Orta Doğu Teknik Üniversitesi, Makina Mühendisliği Bölümü İnönü Bulvarı, 06531 Ankara	
Destekleyen Kuruluş(ların) Adı ve Adresi: İlgili Değil.	
Projenin Başlangıç ve Bitiş Tarihleri: 01.07.2008 – 01.07.2010	

Öz (en çok 70 kelime)

Bu proje kapsamında her türlü endüstriyel sistemin denetimine imkan sağlayan ve uygulamanın gerekleri doğrultusunda biçimlendirilebilir bir evrensel denetim sistemi ortaya konmuştur. Denetleyici sisteminin bir prototipi çeşitli FPGA geliştirme kartları ve projeye has tasarlanmış giriş/çıkış kartları vasıtasıyla meydana getirilmiş olup, herhangi bir denetim sisteminin FPGA kırmığı üzerinde sentezlenmesine imkan tanıyacak çok sayıda düşünce iyeliği modül (Verilog ve VHDL gibi donanım tanımlama dilleri kullanılarak) oluşturulmuştur. Proje kapsamında yapılan deneyler ve çevrimiçi donanım benzetimi sonucunda sistemin piyasada bulunan ve hemen hemen her tür ticari denetim sistemi (hareket kontrol kartları, PC104- veya PLC tabanlı sistemler) ile başa baş bir denetleme başarımı sergileyeceği anlaşılmıştır.

Anahtar Kelimeler:

Hareket Denetleyici Sistemler, Alan Programlanabilir Kapı Dizinleri, Gerçek-zamanlı Kontrol, Çevrimiçi Donanım Benzetimi, Arayüz Kartları, Kontrol Uygulamaları.

Fikri Ürün Bildirim Formu Sunuldu mu? **Evet** ☐ [✓ **Hayır** ☒] **Gerekli Değil** ☐

Fikri Ürün Bildirim Formu'nun tesliminden sonra 3 ay içerisinde patent başvurusu yapılmalıdır.

Projeden Yapılan Yayınlar:

1. Mutlu, B. R., *Real-time Motion Control using Field Programmable Gate Arrays*, Yüksek Lisans Tezi, ODTÜ Fen Bilimleri Enstitüsü, Haziran 2010.

2. Üşenmez S., *Design of an Integrated Hardware-in-the-Loop Simulation System*, Yüksek Lisans Tezi, ODTÜ Fen Bilimleri Enstitüsü, Haziran 2010.

3. Yaman, U., *Design of Advanced Motion-Command Generators utilizing Field Programmable Gate Arrays*, Yüksek Lisans Tezi, ODTÜ Fen Bilimleri Enstitüsü, Haziran 2010.

4. Yaman, U., Dölen, M., and Koku, A. B., "A Novel Command Generation Method with Variable Feedrate utilizing FGPA for Motor Drives," *Proc. of the 8th IEEE Workshop on Intelligent Solutions in Embedded Systems (WISES)*, Heraklion, Crete (Yunanistan), 8-9 Temmuz 2010.

5. Mutlu, B. R., Dölen, M., "Novel Implementation of State-Space Controllers using Field Programmable Gate Arrays," *Proc. of the Int'l Symposium on Power Electronics, Electrical Drives, Automation, and Motion (SPEEDAM)*, Pisa (İtalya), 14-16 Haziran 2010.

6. Mutlu, B. R., Yaman, U., Dölen, M., and Koku, A. B., "Performance Evaluation of Different Real-Time Motion Controller Topologies Implemented on a FPGA," *Proc. of the 12th International Conference on Electrical Machines and Systems (ICEMS)*, Tokyo (Japonya), 15-18, Kasım 2009.

7. Üşenmez, S., Dilan, R. A., Dölen, M., Koku, A. B., "Real-time Hardware in the Loop Simulation of Electrical Machine Systems using FPGAs," *Proc. of the 12th International Conference on Electrical Machines and Systems (ICEMS)*, Tokyo (Japonya), 15-18 Kasım 2009.

8. Yaman, U., Mutlu, B. R., Dölen, M., Koku, A. B., "Direct Command Generation Methods for Servo-Motor Drives," *Proc. of the 12th International Conference on Electrical Machines and Systems (ICEMS)*, Tokyo (Japonya), Nov. 15-18, 2009.

9. Mutlu, B. R., Yaman, U., Dölen, M., Koku, A. B., "Kayan Kipli DC Motor Konum Denetiminin FPGA ile Gerçekleştirilmesi," *Türkiye Otomatik Kontrol Ulusal Toplantısı 2009 (TOK 2009) Bildiri Kitabı*, Istanbul,13-16 Ekim 2009.

10. Üşenmez, S., Dilan, R. A., Dölen, M., Koku, A. B., "Bir Doğru Akım Motorunun FPGA Üzerinde Gerçek Zamanlı Benzetiminin Gerçekleştirilmesi," *Türkiye Otomatik Kontrol Ulusal Toplantısı 2009 (TOK 2009) Bildiri Kitabı*, Istanbul,13-16 Ekim 2009.

11. Yaman, U., Dolen, M., Koku, A. B., "Endüstriyel Kontrol Uygulamaları için Komut Üretim Yöntemleri," *Türkiye Otomatik Kontrol Ulusal Toplantısı 2009 (TOK 2009) Bildiri Kitabı*, Istanbul,13-16 Ekim 2009.

12. Üşenmez S., Dilan R. A., Yaman, U., Mutlu, B. R., Dölen, M., Koku B. A., "Çevrimiçi Donanım Benzetimi için Yeni bir Yazılım Paketi: Cadmus," *Türkiye Otomatik Kontrol Ulusal Toplantısı 2008 (TOK 2008)*, cilt 2, sf. 685-691, 2008.

13. Üşenmez, S., Koku, A. B., Dölen, M., "A New Hardware-in-the-Loop Simulator for Control Engineering Education," *IEEE Transactions on Education* isimli dergiye gönderilecektir.

14. Yaman, U., Dölen, M., Koku, A. B., "FPGA Implementation of a Novel Motion-Command Generator for Motor Drive Systems," *IEEE Transactions on Industrial Electronics* isimli dergiye gönderilecektir.

15. Mutlu, B. R., Dölen, M. "Evaluation of Digital Motion Controllers Implemented on Field Programmable Gate Arrays," *Control Engineering Practice* isimli dergiye gönderilecektir.

16. Üşenmez, S., Mutlu, B. R., Dölen, M., Koku, A. B., "Hardware-in-the-Loop Simulation of a Three-axis Vertical Machining Center utilizing Field Programmable Gate Arrays," *Control & Automation* isimli dergiye gönderilecektir.