metin, logo, yazı tipi, simge, sembol içeren bir resim

Açıklama otomatik olarak oluşturuldu FB-CPU RTL TASARIMI

Simge Özlem SARP

Bilgisayar Mühendisliği

İstanbul, Türkiye

e-mail: simge.sarp@stu.fbu.edu.tr

*Özetçe*—

Bu proje kapsamında FB-CPU isminde bir işlemcinin Verilog dili ile RTL tasarımı ve tasarlanan işlemci üzerinde makine dili ile yazılan çeşitli kod parçacıkları yazılacaktır. Proje sonunda basit bir işlemcideki RAM, Kontrol Ünitesi ve Saklayıcıların bir arada çalışıp, makine dilindeki kod parçacıklarını nasıl yürütebildiği gözlemlenecektir.

Anahtar Kelimeler — FPGA, CPU

*Abstract*— In this project, RTL design of a processor named FB-CPU with Verilog language and various code snippets written on the computer with machine language will be written. At the end of the project, it will be observed how RAM, Control Unit and Registers in a simple processor can work together and execute code snippets in the machine language.

Keywords — FPGA, CPU.

# Giriş

**Bu proje kapsamında, Verilog dili ile RTL tasarımı ve tasarlanan işlemci üzerinde makine dili ile kod parçacıkları yazılmıştır.**

**Süreç akışında temel olarak;**

**- Von Neumann mimarisi baz alınarak proje geliştirilmiştir.**

**- FB-CPU’nun desteklediği operasyonlar çıkartılmış ve anlamlandırılmıştır.**

**- FB-CPU’nun Durum Diyagramı üzerinden RTL tasarımı yapılmıştır.**

**- Tasarlanan RTL tasarımı üzerinden test yazılımları test edilerek beklenen sonuçlara ulaşılmıştır.**

**Yazılımsal sürecin gerçekleştirilmesi için tasarlanmış sistem mimarisi, kullanılan geliştirme araçları ve geliştirilen proje kapsamında elde edilen sonuçlar da doküman içerisinde detaylandırılmıştır.**

# Sistem Mimarisi

**FB-CPU tasarlanırken Xilinx Vivado Design Suite; Verilog, VHDL vb. donanım tasarım dillerini alarak, FPGA’e konfigüre edilebilecek tasarım dosyasını oluşturur.**

**-FBCPU Simulatörü:**

**FB-CPU’nun mimarisini görselleştiren, veri akışının gözlemlenebildiği “FBCPU Simulatörü” kullanılacaktır.**

**-Xilinx Vivado Design Suite**

**Xilinx Vivado Design Suite, FPGA geliştirme kartları üzerinde çalışmalar yapmak için gerekli olan tasarımı oluşturmak için kullanılmaktadır. Verilog, VHDL vb.. donanım tasarım dillerini alarak, FPGA’e konfigüre edilebilecek (Xilinx firması FPGA’leri için .bit uzantılı dosyalar) tasarım dosyasını oluşturur.**

**FB-CPU RTL tasarımı, Von Neumann mimarisindedir.**

metin, ekran görüntüsü, dikdörtgen, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Temel olarak 4 elemanı vardır; saklayıcılar, bellek(RAM), İşlem Ünitesi (ALU), Kontrol Ünitesi.**

**Von Neumann mimarisinde kullanılan ünitelerin görevleri:**

**Bellek, operasyon komutlarını ve değişkenleri tutmaktadır.**

**İşlemci Ünitesi, aritmetik ve mantık işlemlerini yapmaktadır. Kontrol Ünitesi, komutların çözülmesi için gereklidir. Saklayıcılar, çeşitli görevlerde kullanılan saklama alanları.**

**FB-CPU’nun tasarımda 4 adet saklayıcı bulunmaktadır. Durum; durum makinasında, hangi durumda olduğunu bilgisi tutulur. PC; RAM’deki hangi adresteki komutun çalıştığı bilgisi tutulur. IR; o anda çalışan komutun kendisi tutulur. ACC; geçici saklama alanı.**

metin, yazı tipi, makbuz, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**FB-CPU durum makinaları yöntemi ile gerçeklenecektir. Yani bu işlemci durum ismindeki saklayıcının değerine göre 2³ = 8 farklı durumda çalışan bir tasarımı olacaktır. Diğer tüm saklayıcılar, durum saklayıcısının değişimine göre çalışacaktır. Durumun değerine göre tüm saklayıcıların giriş sinyalleri değişmektedir. Tasarımda giriş çıkış portlarına bağlı olan bellek sinyalleri söyledir.**

**- MAR (6 Bit): Memory Address Register isminde bir saklayıcıdır. Bu saklayıcı RAM’in adres girişine bağlanmıştır. RAM’in 2^6 lokasyonu olduğu için MAR 6 bitliktir.**

**- MDRIn (10 Bit): Memory Data Register In, RAM’e bir veri yazılacağı zaman kullanılan saklayıcıdır. RAM’in bir lokasyonu 10 bitlik olmasından ötürü, saklayıcı 10 bittir.**

**- RAMWr (1 Bit): RAM’e veri yazılacağı durumlarda aktif edilmektedir. 1 olmadığı durumlarda RAM’e veri yazılmaz.**

**- MDROut (10 Bit): Memory Data Register, RAM’den veri okunacağı zaman kullanılan saklayıcıdır. RAM’in bir lokasyonu 10 bit olmasından dolayı, saklayıcı 10 bittir.**

**Tüm saklayıcılar RAM içerisindedir.**

**Bellek (RAM, Random Access Memory); FB-CPU’nun komutları okuyup, hesaplanan değerleri geri yazacağı bir Block RAM mekanizması bulunmaktadır. Test kodunun instantiate ettiği bellek memory.v dosyasında bulunmaktadır. RAM’e bağlı 4 saklayıcı, clock ve reset sinyali bulunmaktadır. Aynı zamanda memory.v içerisinde bulunan include testCase’ler, tb\_fbcpu.v dosyasında parametre ile çağırdığımız test kodunun içeriğini alıp if bloğunun altına getirmektedir.**

metin, makbuz, yazı tipi, cebir içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, cebir, makbuz içeren bir resim

Açıklama otomatik olarak oluşturuldu

**İşlem Ünitesi (ALU, Arithmetic Logic Unit); aritmetik işlemlerin gerçekleştirildiği bölümdür. FB-CPU’da toplama, çıkartma ve çarpma, gibi aritmetik işlemler vardır gelen operasyon koduna göre işlemleri gerçekleştirip ACC saklayıcısına yazmaktadır. Kontrol Ünitesi; Saklayıcılar, Aritmetik İşlem Ünitesi ve RAM’e verilerin birbirleri arasında transferinden sorumludur.**

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

**FB-CPU’nun 10 bitlik komutunun, operasyon ve adres için bitlerinin ayrılması gösterilmiştir.**

**İşlemcimizin tepe modülü olan, tasarımı barındıran fbcpu\_core.v modülü vardır. İşlemcinin kendisini tanımlar. memory.v isimli bir başka dosyada da RAM’in kendisini tanımlanır. Bu iki dosyayı aynı anda test edebilmek ve birbirine bağlamak için tb\_fbcpu.v isimli test bench kodu vardır. Bu test bench kodunda fbcpu\_core ile block RAM’i bulunduran memory.v instance edilmiştir.**

**test bench kodunun içinde TEST\_CASE isminde bir paramatremiz var. İşlemcinin okuyabilmesi için belleklere yazılmış olan programı test etmemizi sağlayan test dosyalarıdır.**

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

**test bench kodundaki parametre ile TEST\_CASE 1’i çağırırsak kodu çalıştırdığımızda makine diliyle yazılmış bu dosya çağırılmış olur. Bu dosyalardan hangisinin o esnada test edileceğini parametreye test casenin değerini vererek söylüyoruz.**

**Test bench’imizin içinde aynı zamanda parametreden girdiğimiz değere göre gerekli değerleri atayan bir if’li seçim mekanizması bulunmaktadır.**

**Bunların doğru olup olmadığını kontrol eden otomatik bir yapı vardır. Simülasyonu başlattığımız zaman ekrana, testin hatalı tamamlanıp tamamlanmadığını gösteren bir çıktı vermektedir.**

**Fbucpu\_core.v’de önce saklayıcıları tanımladık sonra da always@(posedge clk) bloğunun içerisinde saklayıcılarımızın atamalarını gerçekleştirilmiştir. #1 diyerek de 1 cycle sonra atamanın gerçekleşmesini sağlanır. Yükselen kenar gelmesinden bir nano saniye sonra çıktı vermeye başlar.**

metin, ekran görüntüsü, yazı tipi, cebir içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi, makbuz içeren bir resim

Açıklama otomatik olarak oluşturuldu

**RAM’e giden sinyalleri bir saklayıcıdan geçirmiyoruz. RAM’e giderken içinde saklayıcı var.**

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

**İlk reset anında bu durumlarla beraber beklemede. Rst biter bitmez case’de durum saklayıcısının değerine bakacak şekilde aktifleşir.**

metin, makbuz, cebir içeren bir resim

Açıklama otomatik olarak oluşturuldu

# Kullanılan Yazılım

**Tasarım, bir durum makinası şeklinde kodlandı. Case duruma bakıyor ve durumun içerisindeki yapılara göre sürekli ilerliyor.**

metin, daire, diyagram, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Durum 0’da, Program Counter (PC) isimli saklayıcımız, bellekteki hangi komutu çalıştıracaksak onu tuttuğumuz saklayıcının değerini yani 0’ı, MAR’ın üstüne attı. Bir yazma işlemi yapmadığımız için yazma sinyalimizi RamWr’ı 0’a çektik ve durumu 1 arttırarak 1 cycle sonra durum 1’e atladık. Yani burada PC’nin değerini MAR’a verdik böylelikle RAM’den 0. adresin içeriğini istemiş olduk durumu değiştirdiğimiz zaman RAM bize kendi data outputundan (MDROut’tan) 0. adresin içeriğini döndürecek.**

metin, yazı tipi, makbuz, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Durum 1’de Memory Data Out’tan, gelen instruction bilgisi unutulmaması için Instruction Register (IR)’a yazılır. İşlemler sonrası tekrar durum 0’a döneceğimiz için durum 0’da iken daha önce okuduğumuz 0. adres değil de 1. adresi okumamız gerekmekte. PC’yi 1 arttırdık. Bu işlemi yaptıktan sonra yeni işlemi yapmak için durumumuzu da 1 arttırıyoruz ve durum 2’ye geçiyoruz.**

metin, makbuz, yazı tipi, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Durum 2’de bir önceki adımda IR’ye atadığımız saklayıcının 9’dan 6. biti yani soldan ilk 4 biti, 6’dan küçük mü diye bakıyoruz. Bakma sebebimiz 6’dan küçük olan işlemlerinin hepsinin RAM’den bir adres okuma ihtiyacı(SUB, ADD vs.) olmasıdır. Bir tek STO’da içerik bilmeye ihtiyacımız yoktur. İlk 6 operasyonda içeriği okuma ihtiyacımız olduğu için hepsinin içeriğini okuma ve sonrasında durum 3’e götürme ve kalan işleri durum 3’de bitirme mantığıyla işlemci tasarlanmıştır. 9’dan 6’ya olan bitler 6’dan küçükse, adrese ve adresin içeriğine ihtiyacımız olacağı için adres IR’nin ilk 6 bitinde tutuluyor. Bunu RAM’in MAR’ına yazdık. Yani RAM’den tekrar bir adres istiyoruz ama bu sefer istediğimiz adres, oradaki komutun yanında yazılan adres. Bu işlemlerden sonra durumu 3’e götürdük. Yani ilk 6 işlemden(operasyondan) birisiyse, adres içeriğini tekrar okumamız gerektiği için RAM’den o bilgiyi istedik. Yani durum değişimimiz ilk 6 bite bağlı. Makine kodumuzun içindeki soldan ilk 4 biti 6’dan küçük geldiyse durumu 3’e geçiyoruz.**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Eğer gelen değer 6 ise, PC değerini IR’nin 5’ten 0’a olan değerine karşılık gelen sayıya eşitliyoruz. Durumu da tekrar 0’a döndürüyoruz. Yani aslında JMP komutu geldi. PC’nin yeni değerini yazdık, durumu tekrar 0’a götürdüğümüz zaman artık PC değeri güncellenmiş olacağı için bizim RAM’den okuyacağımız işlem artık RAM’den çekeceğimiz kodun yeri, güncellenmiş yerden olacak ve biz JMP işlemi yapmış olacağız. 7. komutun (operasyonun) özelliği koşullu JMP işlemidir. ACC değeri 0 ise PC değerini, IR’nin ilk 6 bitinin değerini atıyoruz ve durum 0’a gidiyoruz; değilse de PC değerini değiştirmiyoruz, hiç bir işlem yapmadan durumu 0’a gönderiyoruz. (ACC’ın 0 ya da 1 olmasından bağımsız olarak durum 0’a gidiyor) Eğer komut 8 geldiyse NOP komutu yani “no operation”. Hiçbir işlem yapılmaz, durumu sadece direkt 0’a döndürüyoruz. Operasyon kodu 9 gelirse HLT yapılır. Yani kod durur. Durması için durum 4’e atlatıyoruz. - Durum 3’ü, ilk 6 operasyonu RAM’den okuduktan sonra işlem yapmak için kullanıyoruz. Durum 3’e geçtiğimizde doğrudan gelecek durumumuzu 0’a götürüyoruz yani yaptığımız işlemlerden bir cycle sonra direkt durum 0’a gidiyoruz. Yukarıda RAMWr ve MAR’ı 0’a eşitledik ama alt taraftaki işlemlerde değerlerini güncelleyebiliriz. IR 9’dan 6’ya, 0’a eşitse LOD işlemi yaptığımız anlamına gelir. LOD, o adresin içeriğini alıp ACC içine atıyor. Biz aslında o durumun içeriğini durum 2’de MAR’a vererek istedik ve bu bize bir cycle sonra MDROut’tan geliyor. Bunu ACC’a atıyoruz. Operasyon kodu 1 olduğunda STO komutu gelmiştir. Bu komutla ACC’nin içerisindeki değeri, belleğin içindeki değere atamamız lazım. Onu yapmak için de hangi adrese değeri yazacaksak, adres içeriğini söylediğimiz yer MAR’dır. MAR’a, IR 5’ten 0’ına söylediğimiz yerin adresini verdik. RAMWr’ı 1 yaptık. Aynı zamanda da RAM’in data içeriğini yüklüyoruz. MDRIn’ine de ACC değerini verdik. Burada ACC içeriğini Memory’nin Data portundan verdik WriteIn’i 1 yaptık aynı anda adresi de STO işleminden gelen adresi vermiş olduk. STO’nun tamamı 10bitlik bir sayı, onu da IR saklayıcısında tutuyorduk. IR saklayıcısının 5’den 0’ını verdiğimiz zaman 6 bitlik o adresi MAR’a vermiş olduk**

**Operasyon kodu 2 olduğunda ADD işlemi yapılır. Bir cycle önce adres içeriğini okuma işlemini yapmıştık. Okuduğumuz adres, Memory data read register outtan geldi, ACC ile toplayıp tekrar ACC yazdık. Operasyon kodu 3 olduğunda SUB yani çıkartma işemi geldiğinde ADD işlemindeki adımları çıkartma işlemine uyarlamış olduk. Yani Memory data read register outtan gelen değeri ACC ile çıkartıp tekrar ACC’a yazdık. Operasyon kodu 4 olduğunda MUL komutuyla çarpma işlemi gerçekleştirilir. MDROuttan gelen değer ACC ile çarpılıp tekrar ACC’ın üstüne yazılır. Operasyon kodu 5 olduğunda DIV komutuyla bölme işlemi yapılması istenir ama bizim düzeyimizin üstünde bir işlem karmaşıklığı içerdiği için biz operasyona kodu 5 geldiğinde hiçbir işlem yaptırmadan durumu 0’a göndereceğiz.**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Durum 4’te hiçbir şey kontrol edilmiyor, hiçbir şey yapılmaz sadece üzerinde döner. Bir çıkış komutu da yoktur. HLT işlemi geldiğinde işlemcinin gücü kesilip yeniden güç verilmediği takdirde durum 4’ten çıkması mümkün değil.**



**FB-CPU için bellekte 50 adrese hexadecimal karşılığı 5 olan sayıyı ACC saklayıcısının içine LOD komutuyla beraber yüklenir. ADD komutuyla ACC saklayıcımızın içinde bulunan 50. adresin değeri, 51 adresteki hexadecimal karşılığı A olan ifade toplanır. STO komutuyla ACC saklayıcısının değer (50 ve 51. adreslerin toplamı F’dir) 52. adrese kayıt edilmiş olur. HLT komutuyla test durdurulur.**

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

FB-CPU için bellekte 50 adrese hexadecimal karşılığı 5 olan sayıyı ACC saklayıcısının içine LOD komutuyla beraber yüklenir. MUL komutuyla ACC saklayıcımızın içinde bulunan 50. adresin değeri, 51 adresteki hexadecimal karşılığı A olan ifade çarpılır. STO komutuyla ACC saklayıcısının değer (50 ve 51. adreslerin çarpımı 32’dir) 52. adrese kayıt edilmiş olur. HLT komutuyla test durdurulmuştur.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

**FB-CPU için bizden yapılması istenen işlem çarpma işlemidir bunu döngülerle yapmamız gerekmektedir. Bellekte 51. adresteki değer LOD komutuyla ACC’a kayıt edilir. SUB komutuyla ACC içinde olan değerden 49 çıkartılır ve tekrar ACC saklayıcısının içine kaydedilir. JMZ komutu ile bir önceki basamakta yaptığımız işlemin (ACC-49) sonucu 0’a eşit mi değil mi diye kontrol edilir. 0’a eşit ise döngüden çıkar ve 10. adresteki işlemi gerçekleştirmeye gider. Yani 48. adresteki temp’in değerini ACC saklayıcısın içine yükler, STO ile ACC içindeki değeri 52. adrese kayıt eder ve sonra da HLT ile kodu bitirir. MUL komutu kullanmadan sonuç belleğe kayıt edilir. Ama JMZ 0’a eşit değilse ACC-49, 0 oluncaya kadar dönmeye devam eder. Kontrolden sonra 48. adrese temp değerini LOD ile yükler. 50. adresteki değeri ACC’ın üstüne ADD ile ekler. 48. adrese gider ve ACC’nin değerini STO temp’e kayıt eder. 49. adresteki değeri ACC’nin içine atar ve 46. adresteki değer ile toplar ve 49. adrese tekrar kayıt eder. JMP komutuyla tekrar 0. Satıra yani döngünün başına döner.**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

# Sonuçlar

**Geliştirilen işlemcinin desteklediği işlemler, FB-CPU ISA (Instruction Set Architecture) tablosunda verilmiştir. Gelen operasyon koduna göre işlemler yapılıp ACC saklayıcısına yazılır. Test yazılımı(test\_case)1, test yazılımı 2 ve test yazılımı 3 tek tek denenerek, Vivado üzerindeki simülasyon üzerinden incelenmiştir.**

**Bu proje vasıtasıyla bir işlemcinin temel bileşenleri , çalışma prensipleri ve çalışma mantığını anlamış aynı zamanda algoritmik düşünce yeteneğim gelişmiş ve verilog diline olan hakimiyetimin arttığını söyleyebilirim.**

##### Proje Ekibi

**Simge Özlem SARP, 01.03.2002 yılında doğdum. 2020 yılında Mustafa Kemal Anadolu Lisesinden mezun oldum. Şu anda Fenerbahçe Üniversitesinde Bilgisayar Mühendisliği bölümünde lisans eğitimini almaktayım.**

##### Referans Dosyalar

SUNU:

Github: https://github.com/simgeozlemsarp/FB-CPU-RTL-TASARIMI-

##### Kaynaklar

1. Smith, J. O. and Abel, J. S., ``Bark and ERB Bilinear Trans­forms'', *IEEE Trans. Speech and Audio Proc*., 7(6):697-708, 1999.
2. Lee, K.-F., *Automatic Speech Recognition: The Development of the SPHINX SYSTEM,* Kluwer Academic Publishers, Boston, 1989.
3. Rudnicky, A. I., Polifroni, Thayer, E H., and Brennan, R. A. "Interactive problem solving with speech", *J. Acoust. Soc. Amer*., *Vol. 84, 1988, p S213(A).*