FB-CPU RTL TASARIMI

Rahaf Jabarin, Merian Zidan, Semanur Özyılmaz, Esra Küçükbaş
Fenerbahçe Üniversitesi
Bilgisayar Mühendisliği
İstanbul, Türkiye
e-mail: {rahaf.jabarin, merian.zidan, semanur.ozyilmaz, esra.kucukbas}@stu.fbu.edu.tr,

***Özetçe*— Bu projede Von Neumann mimarisi ile tasarlanan, 9 adet komutu destekleyen ve durum makinaları yöntemi ile gerçeklenen FB-CPU RTL tasarımı gerçekleştirilmiştir. Tasarlanan FB-CPU üzerinde makine dili ile çeşitli kod parçacıkları yazılmıştır. Basit bir işlemcideki RAM, kontrol ünitesi ve saklayıcıların bir arada çalışmaları ve makine dilindeki kod parçacıklarını yürütmeleri gözlemlenmiştir.**

Anahtar Kelimeler — FPGA, CPU

***Abstract*— This project is designed with von neumann architecture and supports 9 commands FB-CPU RTL design is made by state machines method. Various code snippets are written with machine language on the designed FB-CPU. Ram, controllers and storers in a simple processor have been observed to work together and execute machine-language code snippets.**

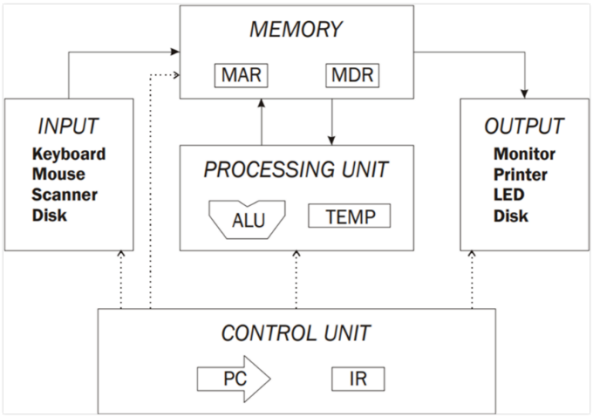
Keywords — FPGA, CPU.

I. GİRİŞ

Bu projede FB-CPU isminde bir işlemcinin Verilog dili ile RTL tasarımı gerçekleştirilmiştir. Tasarlanan işlemci üzerinde makine dili ile çeşitli kod parçacıkları yazılmıştır. FB-CPU RTL tasarımı Von Neumann mimarisindedir. Proje sonunda basit bir işlemcideki RAM, kontrol ünitesi ve saklayıcıların bir arada çalışarak makine dilindeki kod parçacıklarını nasıl yürütebildiği gözlemlenmiştir.

II. SİSTEM MİMARİSİ

FB-CPU, Verilog dili ile Xilinx Vivado ortamında tasarlanmıştır. Bu uygulamadaki simülasyon ortamında test edilmiştir. Ayrıca Von Neumann makinası simülatörü ile işlemcide komutların çalıştırılması esnasındaki veri akışı gözlemlenebilmektedir. Tasarım Von Neumann mimarisi ile gerçeklenmiştir.

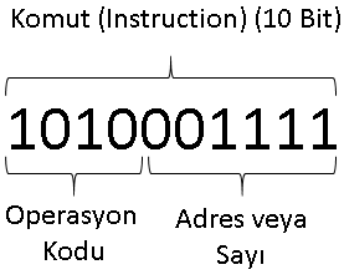


Bu mimaride temel olarak 4 eleman vardır.

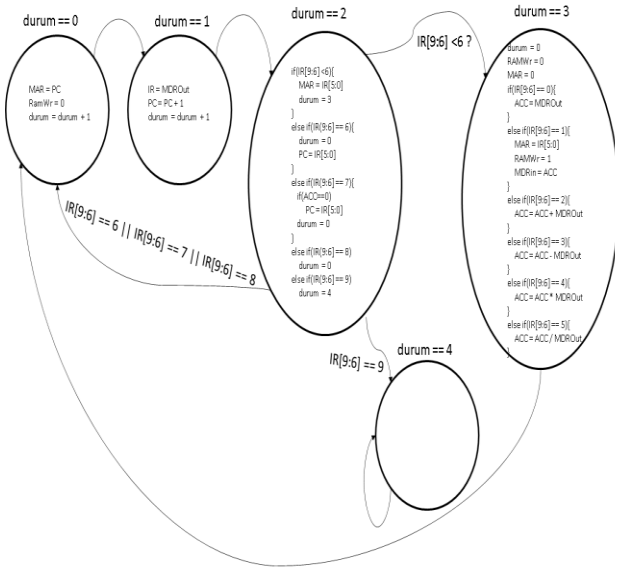
- Saklayıcılar
- Bellek
- İşlem ünitesi
- Kontrol ünitesi

FB-CPU'nun desteklediği 9 komut vardır.

- LOD ADDR: Bellekteki verilen adresin içerisinden değeri alıp ACC saklayıcısına yerleştirir.
- STO ADDR: ACC'nin içerisindeki değeri alıp bellekte verilen adrese yazar.
- ADD ADDR: Bellekteki verilen adresteki değeri alır, ACC ile toplayıp, ACC'nin üzerine yazar.
- SUB ADDR: Bellekteki verilen adresteki değeri alır, ACC ile çıkartıp, ACC'nin üzerine yazar.
- MUL ADDR: Bellekteki verilen adresteki değeri alır, ACC ile çarpıp, ACC'nin üzerine yazar.
- JMP SAYI: PC = Sayı olur.
- JMZ SAYI: ACC'ın değeri 0 ise, verilen sayı değerini PC'e atar, değilse işlem yapmaz.
- NOP: No Operation, hiçbir işlem yapılmaz.
- HTL: Uygulama durur.



İşlemcinin 10 bitlik komutu operasyon kodu ve adres için yukarıdaki şekildeki gibi bitlere ayrılmıştır.



FB-CPU durum makinası yöntemi ile gerçeklenmiştir. Yukarıda ise durum diyagramı verilmiştir.

İşlemcinin tasarımında 4 adet saklayıcı bulunmaktadır.

- durum: Durum makinasında, hangi durumda olduğunun bilgisi tutulur.
- PC: RAM'deki hangi adresteki komutun çalıştığı bilgisi tutulur.
- IR: O anda çalışan komutun kendisi tutulur.
- ACC: Geçici saklama alanı.

Bu saklayıcılar fbcpu_core.v dosyasında bulunmaktadır. Diğer tüm saklayıcılar, durum saklayıcısının değişimine göre çalışmaktadır. Yani durumun değerine göre tüm saklayıcıların giriş sinyalleri değişmektedir.

FB-CPU'nun komutları okuyup, hesaplanan değerleri geri yazacağı bir Block RAM mekanizması bulunmaktadır. Bu bellek test kodunun instantiate ettiği bellek memory.v dosyasında bulunmaktadır. RAM'e bağlı 4 saklayıcı, clock ve reset sinyali bulunmaktadır.

- MAR: Memory Address Register isminde bir saklayıcıdır. Bu saklayıcı RAM'in adres girişine bağlanmıştır. RAM'in 2^6 lokasyonu olduğu için MAR 6 bitliktir. Saklayıcı RAM'in içerisindedir.
- MDRIn: Memory Data Register In, RAM'e bir veri yazılacağı zaman kullanılan saklayıcıdır. RAM'in bir lokasyonu 10 bitlik olmasından ötürü, saklayıcı 10 bittir. Saklayıcı RAM'in içerisindedir.
- RAMWr: RAM'e veri yazılacağı durumlarda aktif edilmektedir. 1 bitliktir. 1 olmadığı durumlarda RAM'e veri yazılmaz. Saklayıcı RAM'in içerisindedir.
- MDROut: Memory Data Register Out, RAM'den veri okunacağı zaman kullanılan saklayıcıdır. RAM'in bir lokasyonu 10 bit olmasından dolayı, saklayıcı 10 bittir. Saklayıcı RAM'in içerisindedir.

İşlem ünitesi (ALU), aritmetik işlemlerin gerçekleştirildiği bölümdür. FB-CPU'da 3 adet aritmetik işlem vardır. Bunlar toplama, çıkartma ve çarpmadır. Gelen operasyon koduna göre işlemleri gerçekleştirip ACC saklayıcısına yazmaktadır.

Kontrol ünitesi; saklayıcılar, aritmetik işlem ünitesi ve RAM'de verilerin birbirleri arasında transferinden sorumludurlar. İşlemci içi veri akışını yönetir.

Durum makinasının gerçeklenmesi durum saklayıcısının aldığı farklı değerlere göre yapılmıştır.

- Durum saklayıcısı 0' a eşitse => PC'de tutulan bellekteki konum MAR'a atanır. Burada yazma işlemi yapılmayacağı için RAMWr sinyaline 0 değeri atanır. Durum saklayıcısının değeri ise bir artırılır.

```
0: begin
    MAR = PC;
    RAMWr = 0;
    durumNext = durum + 1;
end
```

- Durum saklayıcısı 1'e eşitse => Durum 0'da bellekteki konumdan istediğimiz değer MDROut saklayıcındadır. Bu değeri unutmamak için IR saklayıcına atarız. PC'nin değeri 1 artırılır. Tekrar durum 0'a dönüldüğünde PC'den bir sonraki adresi okuruz. Durum saklayıcısının değeri de bir artırılır.

```
1: begin
    IRNext = MDROut;
    PCNext = PC + 1;
    durumNext = durum + 1;
end
```

- Durum saklayıcısı 2'ye eşitse => IR'ın 6-9 arasındaki bitlerinin değeri 6'dan küçükse burada RAM'den adres okuma ihtiyacı vardır. Burada IR'ın ilk 6 bitinde bulunan adres bitleri MAR'a yazılır. Kalan işlemleri yapmak için de durum 3'e eşitlenir. IR'ın 6-9 arasındaki bitlerinin değeri 6'ya eşitse IR'ın ilk 6 bitindeki adres değeri PC'ye atanır. Bir adrese atlanılmış olunur. Durum ise 0'a eşitlenir. IR'nın 6-9 arasındaki bitlerinin değeri 7 ise ACC'nin değerine bakılır. ACC'nin değeri 0'a eşit ise IR'ın ilk 6 bitindeki adres değeri PC'ye atanır. Bir adrese atlanılmış olunur. ACC'nin değeri 0 olsa da olmasa da durum 0'a eşitlenir. IR'ın 6-9 arasındaki bitlerinin değeri 8'e eşitse hiçbir şey yapmadan durum 0'a eşitlenir. IR'ın 6-9 arasındaki bitlerinin değeri 9'a eşitse durum 4 'e eşitlenir.

```
2: begin
    if(IR[9:6] < 6) begin
        MAR = IR[5:0];
        durumNext = 3;
    end else if(IR[9:6] == 6) begin
        durumNext = 0;
        PCNext = IR[5:0];
    end else if(IR[9:6] == 7) begin
        if(ACC == 0)
            PCNext = IR[5:0];
        durumNext = 0;
    end else if(IR[9:6] == 8) begin
        durumNext = 0;
    end else if(IR[9:6] == 9) begin
        durumNext = 4;
    end
end
```

- Durum saklayıcısı 3'e eşitse => Belirli bir işlem yapıldıktan sonra durum 0'a dönüleceği için durum 0'a eşitlenir. RAMWr ve MAR 0'a eşitlenir. IR'ın 6-9 arasındaki bitlerinin değeri 0'a eşitse burada LOD işlemi yapılacaktır. Durum 2'de MAR'a adres değeri verilerek istenilen içerik bize durum 3'te MDROut'tan gelir. Bu değeri ACC'ye atarız. IR'ın 6-9 arasındaki bitlerinin değeri 1'e eşitse STO işlemi yapılacaktır. Burada bellekteki adrese yazma işlemi vardır. IR'ın ilk 6 bitindeki adres değeri MAR'a verilir. Yazma işlemi yapılacağı için RAMWr 1'e eşitlenir. MDRIn'e ise ACC'nin içeriği verilir. IR'ın 6-9 arasındaki bitlerinin değeri 2'ye eşitse ADD işlemi yapılacaktır. Durum 2'de MAR'a adres değeri verilerek istenilen değer bize durum 3'te MDROut'tan gelir. MDROut'tan gelen değer ACC'nin değeri ile toplanarak ACC'ye yazılır. IR'ın 6-9 arasındaki bitlerinin değeri 3'e eşitse SUB işlemi yapılacaktır. MDROut'tan gelen değer ACC'nin değerinden çıkarılarak ACC'ye yazılır. IR'ın 6-9 arasındaki bitlerinin değeri 4'e eşitse MUL işemi yapılacaktır. MDROut'tan gelen değer ACC'nin değeriyle çarpılıp ACC'ye yazılır. IR'ın 6-9 arasındaki bitlerinin değeri 5'e eşitse durum diyagramında bölme işlemi vardır ancak bölme işlemi Vivado tarafından doğrudan yapılamamaktadır. Tasarımı karmaşık bir konu olduğu için bu durumda hiçbir şey yapılmayacaktır.

```
3: begin
    durumNext = 0;
    RAMWr = 0;
    MAR = 0;
    if(IR[9:6] == 0)
        ACCNext = MDROut;
    else if(IR[9:6] == 1) begin
        MAR = IR[5:0];
        RAMWr = 1;
        MDRIn = ACC;
    end else if(IR[9:6] == 2) begin
        ACCNext = ACC + MDROut;
    end else if(IR[9:6] == 3) begin
        ACCNext = ACC - MDROut;
    end else if(IR[9:6] == 4) begin
        ACCNext = ACC * MDROut;
    end else if(IR[9:6] == 5) begin

    end
end
```

- Durum saklayıcısı 4'e eşitse => HLT operasyonu yapılır ve uygulama durur. Burada hiçbir şey yapılmamaktadır. İşlemcinin gücü kesilip tekrar verilmedikçe buradan çıkılamayacak ve bir işlem yapılmayacaktır.

```
4: begin

end
```

III. KULLANILAN YAZILIM

TEST YAZILIMI 1:

İlk test yazılımında bellekte 50 ve 51 nolu adreslerdeki iki sayı toplanıp 52 nolu adrese kaydedilecektir.

LOD komutu ile 50 nolu adresin içeriği yüklenir ve ACC'ye atanır. Daha sonra 51 nolu adresteki değer ACC ile toplanarak ACC'ye yazılır. Sonra da ACC'nin değeri STO komutu ile 52 nolu adresin içeriğine verilir. HLT komutu ile uygulama durur.

TEST YAZILIMI 2:

İkinci test yazılımında bellekte 50 ve 51 nolu adreslerdeki iki sayı çarpılıp 52 nolu adrese kaydedilecektir.

LOD komutu ile 50 nolu adresin içeriği yüklenir ve ACC'ye atanır. Daha sonra 51 nolu adresteki değer ACC ile çarpılarak ACC'ye yazılır. Sonra da ACC'nin değeri STO komutu ile 52 nolu adresin içeriğine verilir. HLT komutu ile uygulama durur.

TEST YAZILIMI 3:

Üçüncü test yazılımında bellekte 50 ve 51 nolu adreslerdeki iki sayının çarpımını 52 nolu adrese kaydeden uygulama geliştirilecektir. Burada çarpma operasyonunu kullanılmayacaktır. Çarpma işlemi için 50 nolu adresteki sayı 51 nolu adresteki sayı defa toplayıp 52 nolu adrese yazılacaktır.

LOD komutu ile 51 nolu adresin içeriği yüklenir ve ACC'ye atanır. SUB komutu ile ACC'den 49 nolu adresteki değer çıkarılarak ACC'ye yazılır. JMZ komutu ile eğer ACC'nin değeri 0 ise 10. satıra atlanır. LOD komutu ile 48 nolu adresin içeriği yüklenir ve ACC'ye atanır. ADD komutu ile 50 nolu adresteki sayı ACC ile toplanarak ACC'ye yazılır. STO komutu ile ACC'nin değeri 48 nolu adrese atanır. LOD komutu ile 49 nolu adresin içeriği yüklenir ve ACC'ye atanır. ADD komutu ile 46 nolu adresteki sayı ACC ile toplanarak ACC'ye atanır. STO komutu ile ACC'nin değeri 49 nolu adrese atanır. JMP komutu ile 0. satıra atlanır. ACC'nin değeri 0 olduğunda JMZ komutu ile 10. satıra atlanılır ve bu döngü biter. Döngü bittiğinde LOD komutu ile 48 nolu adresin içeriği ACC'ye atanır. STO komutu ile ACC'nin değeri 52 nolu adrese kaydedilir. HLT komutu ile uygulama durur.

IV. SONUÇLAR

Geliştirilen FB-CPU işlemcisi 9 adet komutu desteklemektedir. Bunlar bellekten bir değeri alma, belleğe bir değer yazma, bellekteki iki sayıyı toplama, bellekteki iki sayıyı çıkarma, bellekteki iki sayıyı çarpma, bir satırdaki komuta atlama, ACC'nin değerine bağlı olarak bir satırdaki komuta atlama, hiçbir şey yapmama ve uygulamayı durdurma komutlarıdır. Bu proje ile basit bir işlemcide RAM, kontrol ünitesi ve saklayıcıların bir arada çalışıp kod parçacıklarını yürütmelerini gözlemlemiş olduk. İşlemcinin çalışma mantığı ve bileşenlerini anlamış olduk.

PROJE EKİBİ

Semanur Özyılmaz: 2019 yılında Küçükçekmece Anadolu Lisesinden mezun oldum. Lisans eğitimime Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde devam etmekteyim. C, C++, C# ve Python dilleri ile ilgilenmekteyim.

Esra Küçükbaş: 2019 yılında Hatice Kemal Kayalıoğlu Fen Lisesinden mezun oldum. Lisans eğitimime Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde devam ediyorum. C, C++ ve Python dilleri ile ilgilenmekteyim.

Rahaf Jabarin: 2018 yılında Filistin'de Tsamoh Lisesinden mezun oldum. Lisans eğitimime Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde devam etmekteyim. C, C++, C# ve Python dilleri ile ilgilenmekteyim.

Merian Zidan: 2018 yılında Filistin'de Madyan Lisesinden mezun oldum. Lisans eğitimime Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde devam etmekteyim. C, C++, C# ve Python dilleri ile ilgilenmekteyim.

REFERANS DOSYALAR

Youtube Video Linki: https://youtu.be/n4LOaCiMasM

Github Linki: https://github.com/semanurozyilmaz/FB-CPU

KAYNAKLAR

[1]http://www.levent.tc/files/courses/digital_design/project/BLM201_proje_spesifikasyonlari.pdf