FB-CPU RTL Tasarımı Projesi

Cüneyt BALCI, Mustafa Berk TAŞKIN, Ahmet Hazar HASPOLAT, Ömer Sait YORULMAZ

Fenerbahçe Üniversitesi

Bilgisayar Mühendisliği

İstanbul, Türkiye

e-mail: [{ cuneyt.balci@stu.fbu.edu.tr](mailto:%7b%20cuneyt.balci@stu.fbu.edu.tr) , mustafa.taskin@stu.fbu.edu.tr , hazar.haspolat@stu.fbu.edu.tr , omer.yorulmaz@stu.fbu.edu.tr}

*Özetçe*— Bu proje kapsamında FB-CPU işlemcisinin Verilog dili ile RTL tasarımı yapılacaktır ve tasarlanan işlemci üzerine makine dili ile kod parçacıkları yazılacaktır. Projede sonuç olarak bir işlemcideki RAM, logic unit ve latchlerin bir arada çalışırken makine dilindeki kod parçacıklarını nasıl yürüttüğü gözlemlenecektir.

Anahtar Kelimeler — FPGA, CPU

*Abstract*— Within the scope of this project, the FB-CPU processor will be designed in Verilog language and RTL will be designed and code snippets will be written on the designed processor with machine language. As a result of the project, it will be observed how RAM, logic unit and latches in a processor execute code snippets in the machine language while working together.

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 iş kırılımları yapılmış ve bu kırılımlar üzerinden RTL tasarımı yapılmıştır.

- Elde edilen RTL tasarımı üzerinden test yazılımları test edilerek beklenen sonuçlara ulaşılmıştır.

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

# Sistem Mimarisi

**Kullanılacak Araçlar:**

Proje kapsamında 2 araç kullanılacaktır.

-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.

**Tasarımın Mimarisi:**

FBÜ CPU durum makinasını gerçeklemeye çalıştık. Durum saklayıcısının aldığı değere göre farklı komutlar çalışıyor.

Durum saklayıcısı 0’a eşit ise; MAR(Memory Adress Register) saklayıcısındaki veri PC(Program Counter)’a besleniyor. Sonra Ram Write(RamWR) sinyali 0 besleniyor ve durum saklayıcısı 1 artırılıyor.

Durum saklayıcısı 1’e eşit ise: IR(Instruction Register) saklayıcısındaki veri MDR(Memory Data Register)’a besleniyor. PC(Program Counter) saklayıcısındaki değer 1 artırılıyor ve durum saklayıcısındaki değer de 1 artırılıyor.

Durum saklayıcısı 2’ye eşit ise: IR’ın 6’dan 9’a kadar olan bitlerinin değerinin 6’dan küçük olup olmadığına bakılıyor. Eğer 6’dan küçük ise IR’ın 0’dan 5’e olan bitleri MAR’a besleniyor ve durum 3’e eşitleniyor. Eğer 6’dan 9’a kadar olan bitlerinin değeri 6’dan küçük değil ise 6’ya eşit olup olmadığı kontrol ediliyor. Eğer 6’ya eşit ise durum saklayıcısı 0’a eşitleniyor. IR’ın 0’dan 5’e olan bitleri MAR’a besleniyor. Eğer 6’dan 9’a kadar olan bitlerinin değeri 6’ya eşit değil ise; 7’ye eşit olup olmadığı kontrol ediliyor ve eğer ACC(Accumulator) de 0’a eşit ise IR’nin 0’dan 5’e kadar olan bitleri PC’a besleniyor ve durum 0’a eşitleniyor. Eğer 6’dan 9’a kadar olan bitlerinin değeri 7’ye eşit değil ise; 8’e eşit olup olmadığı kontrol ediliyor. Eğer 8’e eşit ise durum 0’a eşitleniyor. Eğer 6’dan 9’a kadar olan bitlerinin değeri 8’e eşit değil ise; 9’a eşit olup olmadığı kontrol ediliyor. Eğer 9’a eşit ise durum 4’e eşitleniyor.

Durum saklayıcısı 3’e eşit ise: Durum saklayıcısı 0’a eşitleniyor, RamWR sinyali 0’a eşitleniyor, MAR değeri 0’a eşitleniyor. IR’ın 6’dan 9’a kadar olan bitlerinin değerinin 0’a eşit olup olmadığına bakılıyor. Eğer eşit ise MDRout saklayıcısı ACC’ye eşitleniyor. Eğer 6’dan 9’a kadar olan bitlerinin değeri 0’ya eşit değil ise; 1’ye eşit olup olmadığı kontrol ediliyor. Eğer 1’e eşit ise IR’ın 0’dan 5’e kadar olan bitleri MAR’a eşitleniyor ve RamWR sinyali 1’e eşitleniyor ve ACC değeri MDRin’e besleniyor. IR’ın 6’dan 9’a kadar olan bitlerinin değerinin 1’a eşit olup olmadığına bakılıyor. Eğer 1’e eşit değil ise 2’ye eşit olup olmadığına bakılıyor.Eğer 2’ye eşitse ACC ve MDRout toplanıp ACC’e yazılıyor. IR’ın 6’dan 9’a kadar olan bitlerinin değerinin 1’a eşit olup olmadığına bakılıyor. Eğer 1’e eşit değil ise 2’ye eşit olup olmadığına bakılıyor.Eğer 2'ye eşitse ACC ve MDRout toplanıp ACC’e yazılıyor. IR’ın 6’dan 9’a kadar olan bitlerinin değerinin 2’a eşit olup olmadığına bakılıyor. Eğer 2’e eşit değil ise 3’ye eşit olup olmadığına bakılıyor.Eğer 3'ye eşitse ACC’den MDRout çıkarılıp ACC’e yazılıyor. IR’ın 6’dan 9’a kadar olan bitlerinin değerinin 3’a eşit olup olmadığına bakılıyor. Eğer 3’e eşit değil ise 4’ye eşit olup olmadığına bakılıyor.Eğer 4'ye eşitse ACC ve MDRout çarpılıp ACC’e yazılıyor. IR’ın 6’dan 9’a kadar olan bitlerinin değerinin 4’a eşit olup olmadığına bakılıyor. Eğer 4’e eşit değil ise 5’ye eşit olup olmadığına bakılıyor.Eğer 5'ye eşitse ACC MDRout ‘a bölünüp ACC’e yazılıyor.

Durum saklayıcısı 4’e eşit HLT operasyonu çalışır.

# Kullanılan Yazılım

Test Case 1:

Öncelikle accumulator(ACC)’daki 50. Adres LOD komutu ile yükleniyor ve sonrasında (ADD 51 komutu ile) 51. Adres ile toplanıp (STO 52 komutu ile) 52. adrese kaydediliyor ve sonrasında da halt komutu ile durduruluyor.

Test Case 2:

Öncelikle ACC’deki 50. Adresi LOD komutu ile yükleniyor ve sonrasında da MUL 51 komutu ile 51. Adresteki değer ile çarpılıyor. Ardından da son olarak STO 52 komutu ile 52. Adrese kaydedilip halt komutuyla durduruluyor.

Test Case 3:

Öncelikle ACC’deki 51. Adres LOD 51 komutu ile yüklenir ve ardından SUB 49 komutu ile 49. Adresteki veri 51. Adresteki veriden çıkarılır. Bu işlemden sonra eğer değer 0’a eşitse 10. Satıra atlanır değilse devam edilir. Sonrasında LOD 48 komutu ile 48. Adrese temp değeri yüklenir ve ardından ADD 50 komutu ile 50. Adresteki değerle toplandıktan sonra da STO 48 komutu ile 48. Adrese kaydedilir. Sonrasında LOD 49 komutuyla ACC’ye 49. Adres yüklenir ve sonrasında ADD 46 komutu ile 46. Adresteki değerle toplanıp ardından da STO 49 komutuyla 49. Adrese kaydedilir. JMP 0 komutu döngünün başına dönme komutu vermektedir. Ardından LOD 48 komutuyla 48. Satır yüklenir ve STO 52 komutuyla 52. Adrese kaydedilir. En son da HLT komutuyla işlem sonlandırılır.

# Sonuçlar

Bu proje vasıtasıyla bir işlemcinin temel bileşenleri , çalışma prensipleri ve çalışma mantığını anlayarak ve elde edilen RTL tasarımındaki iş akış süreçlerindeki gereksinimleri edinmiş olduk.

##### Proje Ekibi

Cüneyt BALCI:

28.08.2000 yılında İstanbul'da doğdum. 2018 yılında Final Temel Lisesi'nden mezun oldum. Şu anda Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde lisans eğitimi almaktayım.

Mustafa Berk Taşkın:

Bursa Doğa Koleji Anadolu Lisesinden mezun olup, lisans eğitimine Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde devam etmekteyim. Python ve Javascript dillerinde kendimi geliştirmeye devam ediyorum.

Ahmet Hazar HASPOLAT

Çamlıca Doğa Koleji’nde lise eğitimimi tamamlamış olup, lisans eğitimime Fenerbahçe Üniversitesi - Bilgisayar Mühendisliği bölümünde devam etmekteyim. Akademik eğitimime görüntü işleme ve yapay zeka alanlarında devam etmeyi hedefliyorum.

Ömer Sait YORULMAZ

Fenerbahçe Üniversitesi Bilgisayar Mühendisliği öğrencisiyim. Ayrıca özel bir şirkette software developer olarak çalışmaktayım.

##### Referans Dosyalar

Youtube linki:

<https://youtu.be/i1bsEqPimvI>

Github linki:

<https://github.com/cuneytbalci/fb_cpu_verilog_rtl_project>

##### Kaynaklar

1-<https://verilogguide.readthedocs.io/en/latest/>

2-<http://www.levent.tc/courses/digital-design>