

**FB-CPU RTL TASARIMI**

Behçet Muhammed, Adem Çolak, Hazem Kataie, Danial Erfani Akhlaghi

Fenerbahçe Üniversitesi

Bilgisayar Mühendisliği

İstanbul, Türkiye

e-mail: {behcet.muhammed, adem.colak, hazem.kataie, [danial.erfani}@stu.fbu.edu.tr](mailto:danial.erfani%7d@stu.fbu.edu.tr),

Özetçe: Bu proje kapsamında makine dilinde yazılan 10 farklı operasyon kodu çalıştırabilen bir işlemci tasarımı geliştirilecektir. 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

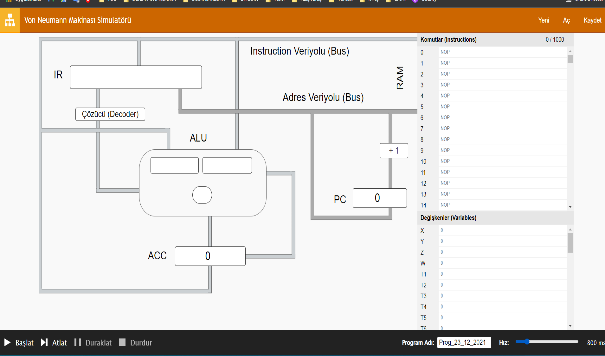
**Abstract:** A CPU design that can run 10 different transaction codes written in machine language will be developed. RTL design of a processor named FB-CPU will be written in Verilog language, and various code snippets written in machine language will be written on the designed processor.

**Keywords:** FPGA, CPU.

**1 – GİRİŞ:** 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. Kullanılacak Basys3 FPGA geliştirme kartı üzerinde FB-CPU demo’su yapılacaktır.

**2- SİSTEM MİMARİSİ:**

**1)- Von Neuman Simülatörü**

****

Von Neumann mimarisi veri ve komutları tek bir birimde bulunduran bilgisayar tasarı örneğidir. FB-CPU’nun mimarisini görselleştiren ve veri akışının gözlemlenebildiği bir simülatördür.

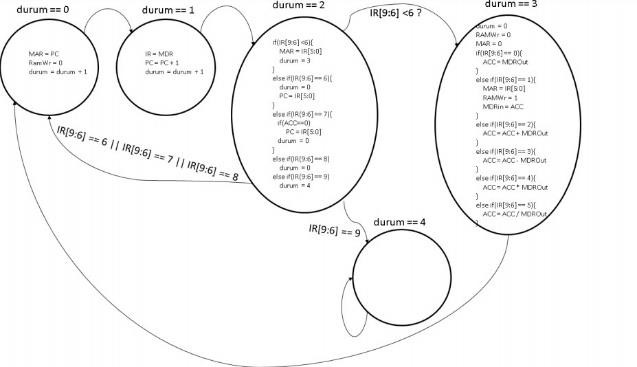
**2)- Xilinx Vivado Design Suite**

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. Vivado Tasarım Aracı, Xilinx’in 7 ve daha yeni jenerasyon FPGA’leri için kullanılabilen bir geliştirme ortamıdır. Bu ortam Xilinx’in sunduğu çeşitli geliştirme ve doğrulama araçlarını barındırır.

Vivado:

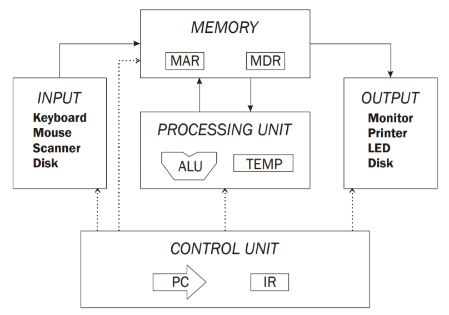
* + Verilog
  + System Verilog
* VHDL Dillerini desteklemektedir. Projede Verilog dili ile tasarımlar yapılacaktır.

**3- Kullanılan Yazılım**



FB-CPU isimli yapacağımız projenin tasarımı şekilde verilmiştir. İstenilen durumlara göre komutların yerine getirilmesi amaçlanmıştır. FB-CPU tasarımı 10 adet komutu yapabilecek şekilde tasarlanmıştır. İşlemci belirtilen komutları yerine getirmek için gerekli durum değerlerini sağlanmalıdır. durum==0, durum==1 ve durum==4 verilerek, bizden durum==2 ve durum==3 işlemlerinin yapılması istenmiştir.

**ACC** yapılan işlem sonucunu tutar. **PC** şu anki komutu tutar. **IR** bir sonraki komutu tutar.

****

Durumdan gelen değerlere göre gerekli yapı seçilerek işlemler gerçekleştirilmiş olur.

Temel olarak 4 elemanı vardır.

* Saklayıcılar
* Bellek (RAM)
* İşlem Ünitesi (ALU)
* Kontrol Ünitesi

**Memory**’de işlemcideki kodlar bulunur. İşlemci hangi komutu çalıştıracaksa onu çalıştırıp tekrar Memory’ye yazar.

**Temp**, işlemcideki geçici bellek görevini görür.

**İşlem Ünitesi(ALU)**,işlemlerin gerçekleştiği yerdir.

**Kontrol Ünitesinde** PC ve IR vardır. Bunlar RAM ile bağlantılıdır. PC, RAM üzerinde hangi komutun alınacağını belirler. IR, RAM’den okunan kodun saklandığı saklayıcıdır.

**SAKLAYICILAR** Tasarımda 4 adet saklayıcı bulunmaktadır.

**Durum:** Durum makinasında, hangi durumda olduğunu bilgisi tutulur.

**PC (6 Bit):** RAM üzerinde hangi satırdaki komutun alınacağını belirler. 6 bit olmasının nedeni RAM’in 26 lokasyonu olmasındandır. Dolayısıyla PC değeri RAM’deki her yeri gösterebilmektedir.

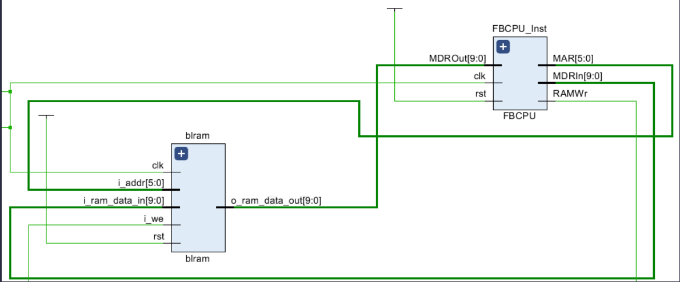
**IR (10 Bit):** Instruction Register, RAM’den okunan kodun (instruction) saklandığı saklayıcıdır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**ACC (10 Bit):** Accumulator, aritmetik işlem sonuçlarının tutulduğu saklayıcıdır.

Diğer tüm saklayıcılar, durum saklayıcısının değişimine göre çalışacaktır. Yani durum’un değerine göre tüm saklayıcıların giriş sinyalleri değişmektedir.  
Diğer bir değiş ile, durum saklayıcısının değerine göre saklayıcıların üzerine başka başka sinyaller atanmakta, sistemin ilerlemesi durum sinyaline bağlıdır.



Yukarıda giriş-çıkış portlarına bağlı olan bellek sinyalleri aşağıda verilmiştir

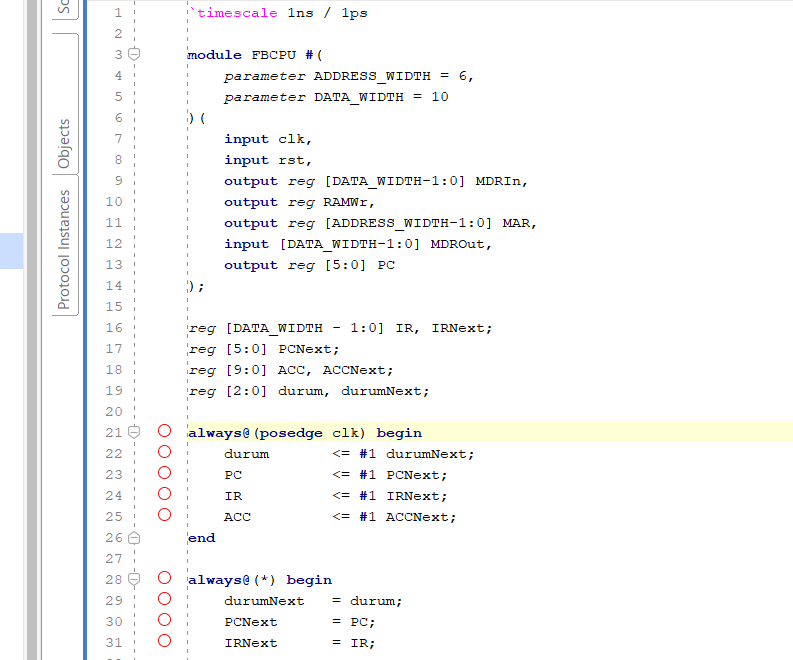
**MAR (6 Bit)**: Memory Address Register isminde bir saklayıcıdır. Bu saklayıcı RAM’in adres girişine bağlanmıştır. RAM’in 26 lokasyonu olduğu için MAR 6 bitliktir. Saklayıcı RAM’in içerisindedir.

**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. Saklayıcı RAM’in içerisindedir.

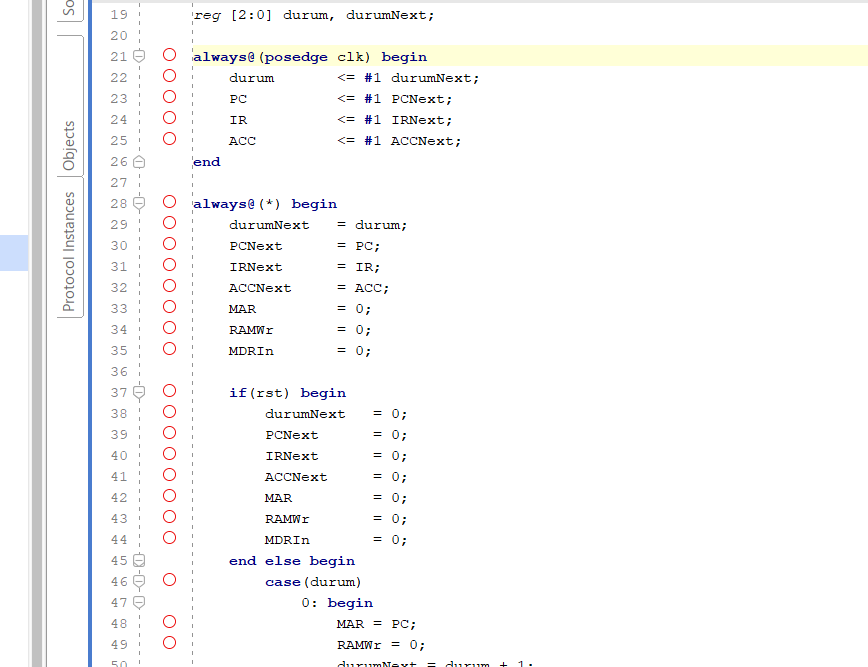
**RAMWr (1 Bit):** RAM’e veri yazılacağı durumlarda aktif edilmektedir. 1 olmadığı durumlarda RAM’e veri yazılmaz. Saklayıcı RAM’in içerisindedir.

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

**fbcpu\_core.v**

****

Buradaki kodlar işlemcinin kendisini barındırır.



Tasarımda giriş-çıkış portlarına bağlı olan bellek sinyalleri,gerekli olan saklayıcılar tanımlanmıştır.

**BELLEK (RAM,Random Access Memory)**

FB-CPU’nun komutları okuyup, hesaplanan değerleri geri yazacağı RAM’e bağlı saklayıcı ve clock sinyali bulunmaktadır. Bu yapı komut ve adreslerin tutulduğu bellektir.

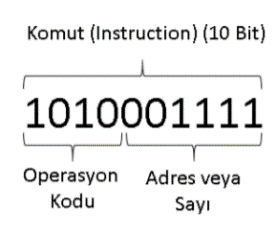
İşlemci ünitesinin belleğe erişimi iki saklayıcı ile olmaktadır:

* MAR: Memory Address Register
* MDR: Memory Data Register
* **İşlem Ünitesi (ALU, Arithmetic Logic Unit):** Aritmetik işlemlerin gerçekleştirildiği bölümdür. FB-CPU’da 4 adet aritmetik işlem vardır. Bunlar toplama, çıkartma, çarpma ve bölmedir, gelen operasyon koduna göre işlemleri gerçekleştirip ACC saklayıcısına yazmaktadır.
* **Kontrol Ünitesi:** Saklayıcılar, Aritmetik İşlem Ünitesi ve RAM’e verilerin birbirleri arasında transferinden sorumludurlar. İşlemci içi veri akışını yönetir. Bellekten program counter’ın göstediği komutu okur. Sistemin geri kalanına, yapılması gereken işlemleri yaptırır. Bir komut birden çok cycle sürebilir.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

Her işlemcinin çalıştırabileceği komut seti vardır.Komutların bellekte okunması ve çözülmesi için durum makinesine 10 bitlik komutunda ilk 4 biti yani [9:6] operasyon kodunu belirtmekte,son 6 biti[5:0] adresi veya sayıyı temsil etmektedir.İşleme sokulacak olan sayı bellekteki bir adresten,komutun içerisinden veya bir saklayıcıdan alınabilir.



Durum==0, durumu Ram’den komutun istenildiği yerdir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Durum==1, Verilog dilinde böyle verilmiştir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Durum = = 2, Verilog dilinde yukarıdaki gibidir.

Durum==2 ‘ye baktığımız zaman operasyon koduna göre olaylar gerçekleşir. Eğer operasyon kodu 6’dan küçükse direkt durum==3 ‘e geçilerek işlemler yapılır.MAR içerisine IR’deki sayı yazılır. Ama 6’ya eşit yada 6’dan büyük bir değer almışsa ona göre farklı atamalar olur. Örneğin operasyon kodu 6’ya eşitse operasyon kodu PC’ye aktarılarak yani PC=Sayı olarak JMP işlemi yapılmış olur.

Operasyon kodu 7’ye eşitse JMZ yapılır. ACC =0 ise verilen sayı değerini PC’e atar, değilse işlem yapmaz. Aynı zamanda durum=0 olur.

Operayon kodu 8’e eşitse NOP işlemi gerçekleşir. Yani hiçbir işlem yapılmaz. Durum=0 olur.

Operasyon kodu 9’a eşitse HLT olur. Uygulama durur. Durum=4 olarak döngüye girer.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Durum = = 3,Verilog dilinde yukarıdaki gibidir.

Durum=3 ‘e eşit olduğu zaman girilen operasyon koduna göre gerekli işlemler yapılır. İlk başta durum=0 , RAMWr=0,MAR=0 verilmiştir. Operasyon kodu 0’a eşitse MDROut’un değeri ACC’ye eşit olur. Operasyon kodu 1’e eşitse IR[5:0] kodu MAR ‘a yazılır. RAMWr=1 olur. ACC içerisindeki değer MDRin ‘e yazılır. Operasyon kodu 2’ye eşitse bu toplama işlemi yapılacağı anlamına gelir. Bellekteki verilen adresteki değeri alır, ACC ile toplayarak ACC’nin üstüne yazar. Operasyon kodu 3’e eşitse bu çıkarma işlemi yapılacağı demektir. Bellekteki girilen adresteki değeri alıp ACC ‘den çıkartarak tekrar ACC’nin üstüne yazar. Operasyon kodu 4’e eşitse çarpma işlemi yapılacağı anlamına gelir. Bellekteki verilen adresteki değeri alıp ACC ile çarparak tekrar ACC’nin üstüne yazar. Operasyon kodu 5’e eşitse bölme işlemi yapılacağı anlamına gelir. Bellekteki verilen adresteki değeri alıp ACC ‘ye bölerek tekrar ACC’nin üstüne yazar. (tasarlanan FPGA sadece bit kaydırarak bölme işlemi geçekleştirebilir.)

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Durum 4’te işlemlerimizi bitirerek çıkış yapılır.

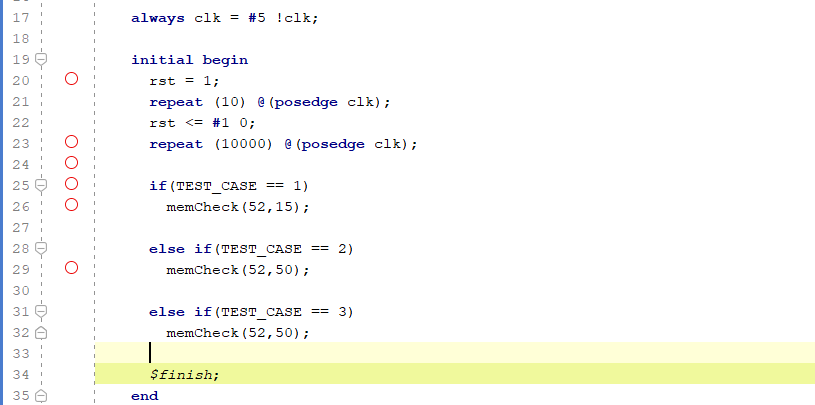
**tb\_fbucpu.v**

Test-bench yazılımında tasarladığımız projenin düzgün çalışıp çalışmadığı kontrol edilir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

TEST\_CASE parametresi, bize verilen test yazılımlarından hangisinin test edilmesini istiyorsak onun test edilmesi için tanımlanmıştır. Test yazılımlarında işlemcinin tasarlanması için belleğe belli verilerin yazıldığı kısımdır.



Memcheck isimli parametre, TEST\_CASE==1 ise önce 10000 cycle bekleyip sonra 52.adrese 15 sayısının yazılmasını sağlar. TEST\_CASE==2 ise önce 10000 cycle bekleyip sonra 52.adrese 50 sayısının yazılmasını sağlar. TEST\_CASE==3 ise önce 10000 cycle bekleyip sonra 52.adrese 50 sayısının yazılmasını sağlar.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yukarıdaki şekilde, FBCPU ve Memory dosyalarındaki kodlar tanımlanmıştır. İkisinin sinyalleri birbirine bağlanmıştır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

işlemcinin düzgün çalışıp çalışmadığını kontrol etmek için tasarlanan kısımdır.

**Memory.v**

Burada bulunan kodlar Ram’in kendisini tanımlıyor.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yukarıdaki şekilde, örneğin TEST\_CASE==1 durumuna gelirse testCase1.v dosyasının içeriğini buraya aktarmaktır. TEST\_CASE durumuna göre belirlenen dosyanın içeriğine gidilir.

**TEST CASELER**

**testCase1.v**

FB-CPU için bellekte 50 ve 51 adresteki iki sayının toplamını 52 no’lu adrese kaydeden uygulamayı inceleyelim.

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

0.adreste ilk 4 bit operasyon kodunu belirtmekte olup LOD olduğu anlaşılıyor. Son 6 bit ise sayıyı ifade etmektedir. Yani bu aşamada 50 sayısını bellekteki verilen adresten alıp ACC saklayıcısına yerleştirir. 1.adreste ilk 4 bitteki operasyon kodu toplama işlemini ifade etmekte olup sayıyı alıp ACC ile toplayıp ACC ‘ye kaydediyor. 2.adreste ilk 4 bit STO’yu belirtip, son 6 bitteki sayıyı yani ACC’deki değeri alıp bellekte verilen adrese yazar. 3.adreste ilk 4 bitteki operasyon kodu HALT işlemini belirtip uygulamayı durdurur. 50.adreste hexadecimal olarak 5 sayısını belirtir. 51.adreste hexadecimal olarak A yani ondalık sayı olarak 10 ‘yı belirtmektedir.

**testCase2.v**

FB-CPU için bellekte 50 ve 51 adresteki iki sayının çarpımını 52 no’lu adrese kaydeden uygulamayı inceleyelim.

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

0.adreste ilk 4 bit operasyon kodunu belirtmekte olup LOD olduğu anlaşılıyor. Son 6 bit ise sayıyı ifade etmektedir. Yani bu aşamada 50 sayısını bellekteki verilen adresten alıp ACC saklayıcısına yerleştirir. 1.adreste ilk 4 bitteki operasyon kodu çarpma işlemini ifade etmekte olup sayıyı alıp ACC ile çarpıp ACC ‘ye kaydediyor. 2.adreste ilk 4 bit STO’yu belirtip,son 6 bitteki sayıyı yani ACC’deki değeri alıp bellekte verilen adrese yazar. 3.adreste ilk 4 bitteki operasyon kodu HALT işlemini belirtip uygulamayı durdurur. 50.adreste hexadecimal olarak 5 sayısını belirtir.

**testCase3.v**

FB-CPU için bellekte 50 ve 51. adresteki iki sayının çarpımını 52 no’lu adrese kaydeden uygulamayı inceleyelim.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

0.adreste 51.adresteki değer yüklenir.

1.adreste 49.adresteki değer ACC’dan çıkarılır.

2.adreste ACC değeri sıfır olana kadar döngüde kalır. Sıfır olunca 10.adrese atlar. (döngünün kırılmasını bu komut sağlar.)

3.adreste geçici değişkeni saklamak için kullanılan ilk değeri 0 olan 48.adres yüklenir.

4.adreste 50. Adresteki değer ACC’ın üstüne eklenir.

5.adreste ACC değeri geçici değişkeni saklamak amaçlı kullanılan 48. Adrese kaydedilir.

6.adreste, içinde 0 bulunan ve indeks olarak tanımladığımız 49.adresi yükleriz.

7.adreste ACC’a 46.adreste bulunan 1 değeri eklenir.

8.adreste ACC indeks olarak belirlediğimiz 49.adrese kaydedilir.

9.adreste JMP 0 komutuyla 0.adrese geri dönülür (döngü oluşmasını bu komut sağlar.)

10.adreste 48.adresteki değer yüklenir.

11.adreste ACC’daki değer 52.adrese kaydedilir.

12.adreste HLT komutu olan bitirme işlemi gerçekleşir. 46.adreste 1 değeri bulunur.

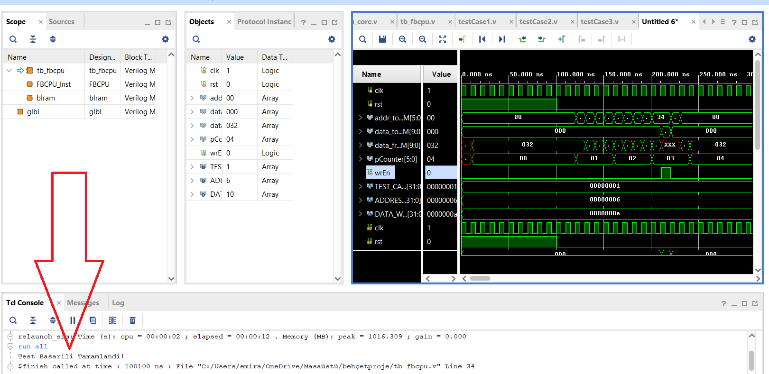
48.adreste ilk 0 değeri bulunur. (Geçici değişken) 49.adreste ilk 0 değeri bulunur.(İndeks)

51.adreste hexadecimal olarak A yani ondalık sayı olarak 10 ‘yı belirtmektedir. 52.adreste de sonuç olarak 10\*5=50

**ÇIKTILAR**

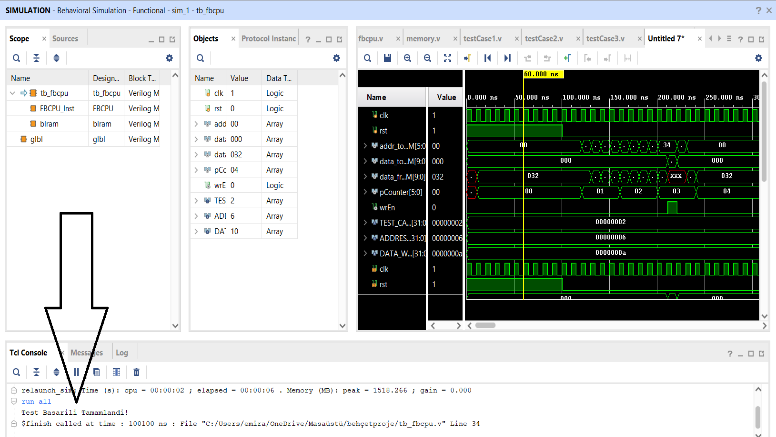
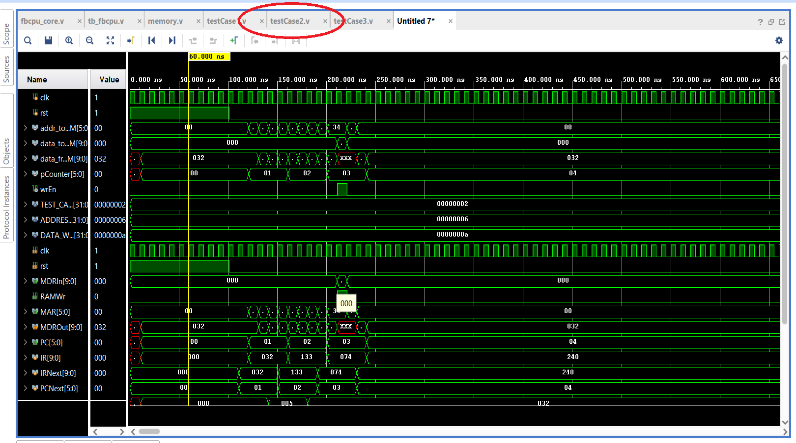
**Test Case 1**

**metin, elektronik eşyalar, bilgisayar, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**

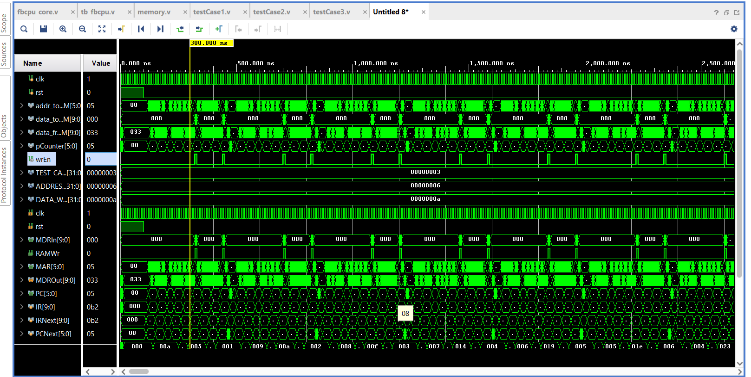
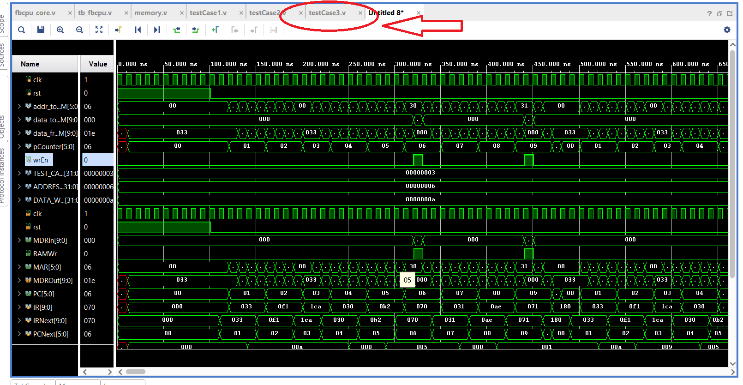
[52] [9:0] adresi 00f (decimal olarak 15) olduğundan ve alttaki konsolda “Test Basarili Tamamlandi” yazısından, işlemcinin yapmasını istediğimiz işlemi başarılı bir şekilde tamamladığını görebiliriz.

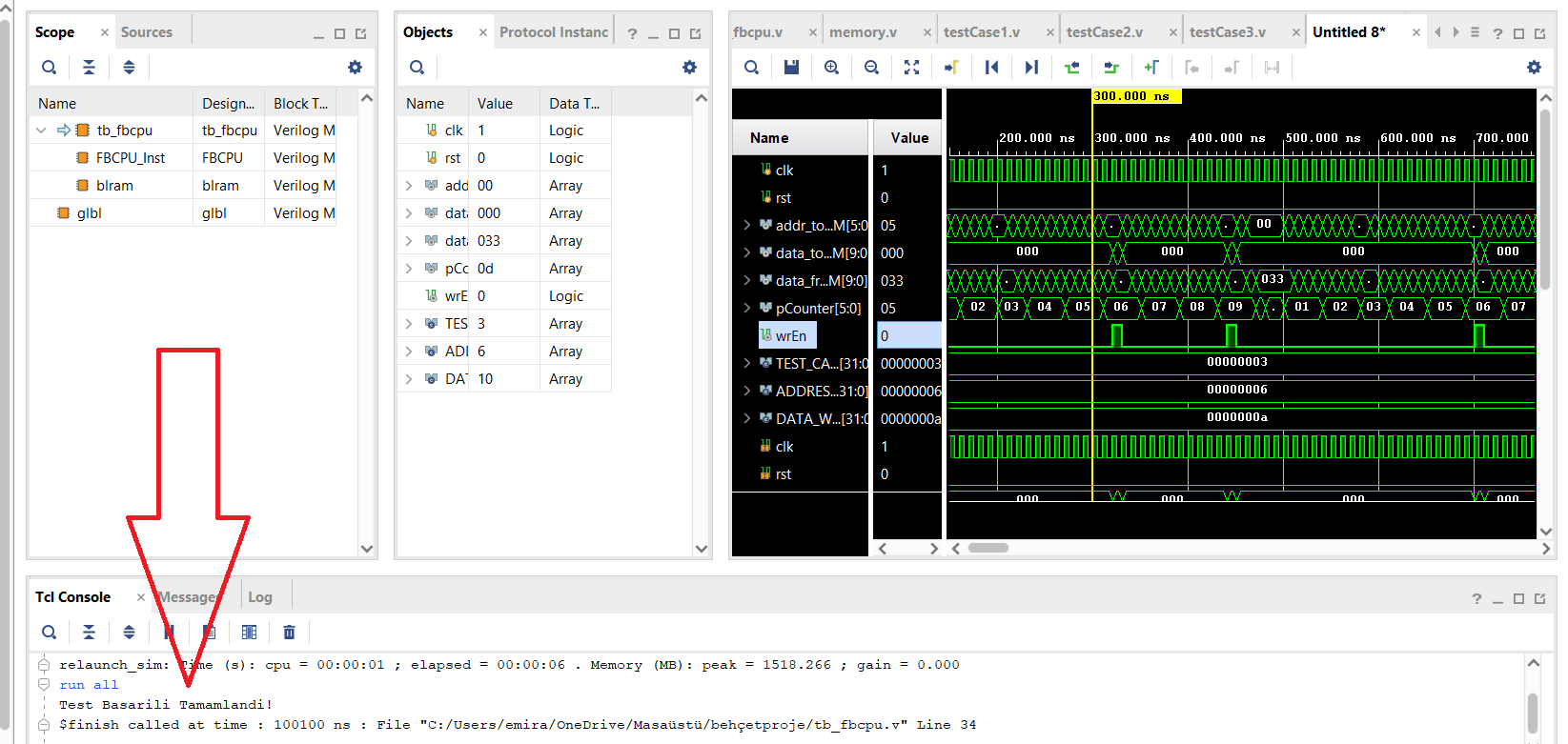
**Test Case 2**

****

[52] [9:0] adresi 032(decimal olarak 50) olduğundan ve alttaki konsolda “Test Basarili Tamamlandi” yazısından, işlemcinin yapmasını istediğimiz işlemi başarılı bir şekilde tamamladığını görebiliriz.

**Test Case 3**

****

****

[52] [9:0] adresi 032(decimal olarak 50) olduğundan ve alttaki konsolda “Test Basarili Tamamlandi” yazısından, işlemcinin yapmasını istediğimiz işlemi başarılı bir şekilde tamamladığını görebiliriz.

**4- SONUÇLAR:**

Geliştirilen FB-CPU işlemcisi gerekli durum koşullarını sağladığında 10 adet komutu yerine getirip, 4 adet işlem yapabilmektedir. Elde ettiğimiz kazanımlara gelirsek, bu Sonuç olarak FB-CPU işlemcisi makine dilindeki kodlarını istenen operasyonları düzgün bir şeklide gerçekleştirebilmektedir

**5-PROJE EKİBİ:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AD VE SOYAD** | **OKUL NUMARASI** | **DOĞUM YERİ** | **DOĞUM TARİHİ** | **MEZUN OLDUĞU LİSE** | **İNGİLİZCE SEVİYESİ** |
| **Behçet Muhammed**  **(Proje Ekip Sorumlusu)** | **200301109** | **Halep, Suriye** | **01/01/1997** | **Münif Paşa Çağdaş Lisesi** | **A1** |
| **Adem Çolak** | **200301037** | **Kadıköy, İstanbul** | **17/08/2002** | **Gökyüzü Koleji Anadolu Lisesi** | **A2** |
| **Hazem Kataie** | **200301115** | **Halep, Suriye** | **27/07/2001** | **Akşamsettin AİHL** | **A2** |
| **Danial Erfani** | **200301105** | **Meshhad, İran** | **02/01/2001** | **Kemal Hasoğlu Anadolu lisesi** | **C1** |

**6- REFERANS DOSYALAR:**

* **YOUTUBE: https://youtu.be/4UjSwiZQBxw**
* **GİTHUB:**

**7- KAYNAKLAR:**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

* Levent, Vecdi Emre (2019) “Von Neumann Mimarisi”, Bilgisayar Mühedisliğine Giriş-Ders Notları.
* Levent, Vecdi Emre (2021) “Durum Makinaları”, Mantıksal Sistem Tasarımı-Ders Notları.
* Levent, Vecdi Emre (2021) “Veriyolu Elemanları”, Mantıksal Sistem Tasarımı-Ders Notları.
* Levent, Vecdi Emre (2021) “Bellekler”, Mantıksal Sistem Tasarımı-Ders Notları.
* Levent, Vecdi Emre (2021) “FB-CPU RTL Tasarım”, Mantıksal Sistem Tasarımı-Ders Notları.

***ÖNEMLİ NOT1: HOCAM EN FAZLA 5 SAYFA DEMİŞTİNİZ ANCAK MAALESEF 5 SAYFAYA SIĞDIRAMADIK.***

***ÖNEMLİ NOT2: HOCAM SLAYTI SIKIŞTIRDIĞIMIZ İÇİN SLAYTA EKLEDİĞİMİZ EKRAN GÖRÜNTÜLERİNİ DAHA İYİ GÖRÜNTÜLEMENİZ İÇİN SİZE AYRICA RAR DOSYAYI OLARAK İLETTİK.***