FB-CPU SystemVerilog Testbench

Damla Su KARADOĞAN, Alp Eren GÜRLE, Taha Yasin ÖZTÜRK
Fenerbahçe Üniversitesi
Bilgisayar Mühendisliği
İstanbul, Türkiye
E-mail: {damla.karadogan, alp.gurle, taha.ozturk}@fbu.edu.tr,

***Özetçe*— Bu proje kapsamında dijital tasarım dersinde tamamlanan FB-CPU işlemcisinin SystemVerilog dili ile otonom kontrolünü yapan bir doğrulama ortamı geliştirilecektir.**

Anahtar Kelimeler — FPGA, CPU

***Abstract*—Within the scope of this project, a verification environment that performs autonomous control of the FB-CPU processor with the SystemVerilog language, which was completed in the digital design course, will be developed.**

Keywords — FPGA, CPU.

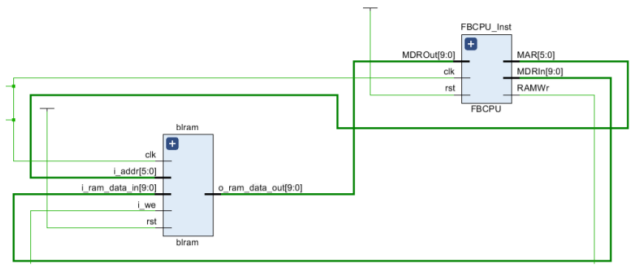
I. GİRİŞ

Geçtiğimiz yıl dijital tasarım dersinde yapmış olduğumuz FB-CPU, geliştirdiğimiz tasarımı yeterince doğrulamıyordu. Bu proje ise, SystemVerilog dili ile otonom kontrol ile profesyonel doğrulama yapan bir projedir. Geçen yıl yazmış olduğumuz işlemcinin üzerine Testbench yazarak denemeler gerçekleştirilecektir.

II. SİSTEM MİMARİSİ

FB-CPU işlemcisinin Testbench'i tasarlanırken Xilinx Vivado Design Suite kullanılmıştır. FPGA geliştirme kartları üzerinde çalışmalar yapmak için gerekli olan tasarımı oluşturmak için kullanılmaktadır. Vivado; Verilog, System Verilog, VHDL gibi donanım tasarım dillerini alarak, FPGA'e konfigüre edilebilecek (Xilinx firması FPGA'leri için .bit uzantılı dosyalar) tasarım dosyasını oluşturur.

FB-CPU'nun mimarisini görselleştiren, veri akışının gözlemlenebildiği "FB-CPU Simülatörü" test yazılımlarının nasıl çalıştığını görmemize yardımcı olmuştur.



Üstteki görüntüde bizim daha önce yapmış olduğumuz FB-CPU ve ona bağlı olan BlockRam'in bağlantıları görülmektedir. Ekran görüntüsünü almış olduğum testbench'te otonom kontrol yoktu. Memory dosyasından case değişkenini 1, 2 ve 3 yaparak test durumunu bram'e yükleyerek test yapmıştık. Bu şekilde gözle bir belleğin adresindeki değere bakıp doğru mu, yanlış mı kontrolü yapılıyordu.

SystemVerilog dili kullanılarak yapacağımız Testbench'te ise her şeyin otonom olması sağlanmıştır. Dosyadan otomatik olarak test dosyalarını okuyup, daha önceden yazmış olduğumuz CPU'ya besleyip sonuçlarını otomatik olarak karşılaştırıp, hata var mı yok mu bilgisini otonom olarak söyleyen bir yapı yapılmıştır. Bu şekilde geçen yılki kodumuzda hata olup olmadığı, yazdığımız yeni testbench ile ortaya çıkabilecektir.

Testbench tasarımında Verilog yerine SystemVerilog dilinin tercih edilmesindeki en önemli faktör, SystemVerilog'un objeye yönelimli bir dil olmasıdır. Objeye yönelimli bir dil olması daha hiyerarşik, daha kullanılabilir, tekrar yazılması mümkün bir dil yapar. Bu sayede esnek doğrulama projelerinde kullanılabilir. Geliştirme esnekliği; sadece sınıfın barındırdığı fonksiyon veya değişkenlerde bir değişiklik yapıldığında, o değişiklik sınıftan türetilmiş tüm objelere anında aktarılacağı için kodu daha modüler hale getirmesinden gelmektedir. Bu bilgiler ışığı altında "sınıflar" kullanılabilir hale gelmektedir. Sınıflarda veri elemanları ve fonksiyonlar bulunmaktadır. Sınıflardan istendiği zaman dinamik olarak objeler türetilip silinebilir. Sınıflardan bir obje türetmek için türetilecek olan sınıfın ismi ve yanına obje ismi yazılır.

tb_fbcpu.sv yani kodlamasını yapacağımız testbench, FB-CPU modülünü test etmektedir. Bunun için daha önceden tasarlanan FB-CPU modülü ile birlikte proje dosyalarını Vivado aracına ekledik. Daha sonrasında ise yazacağımız testbench'i doğru çalışıp çalışmadığını simülasyon aracında gözlemlemek için simülasyon dosyasının altına tb_fbcpu.sv isimli dosyayı açtık.

```
6        reg clk = 0;
7        reg rst;
```

Bu dosyanın içerisinde clock ve reset tanımlamaları yapılarak başlanır.

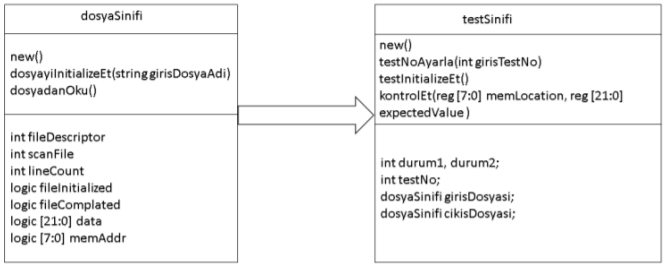
```
9        parameter ADDRESS_WIDTH = 6;
10       parameter DATA_WIDTH = 10;
```

Parametreler yazılarak bram ve FB-CPU arasında yapılan veri transferinin adres ve datalarının bit genişliğinin parametrik olması sağlanır.

```
39       always #5 clk = !clk;
```

Always #5 yazılarak clock 5 nanosaniyede bir 0'dan 1'e çekilir.

Daha sonrasında ise tb_fbcpu.sv dosyasında içerisini doldurmamız istenen iki adet sınıf tanımı yapılmıştır: dosyaSinifi, testSinifi. Bu sınıflardan objeler türetilerek initial begin bloğu içerisinde sınıfın çeşitli fonksiyon ve değişkenleri kullanılarak FB-CPU test edilmektedir. Proje kapsamında dosyaSinifi ve testSinifi isimli sınıfların boş bırakılan sınıf içerikleri doldurulacaktır. Aşağıda dosyaSinifi ve testSinifi isimli sınıfların içerdiği değişken ve fonksiyonlar verilmektedir.



testSinifi isimli sınıf dosyaSinifi sınıfından kalıtım yapılmıştır. Kalıtım, mevcut bir sınıfın değişken ve fonksiyonlarını, başka bir sınıfta kullanılmak istendiğinde, o değişken ve fonksiyonların kopyalarını oluşturmak yerine mevcut sınıftan o özellikleri kalıtım yapılmasıdır.

```
40      class dosyaSinifi;...
103
104     class testSinifi extends dosyaSinifi;...
```

dosyaSinifi'nın amacı, test dosyalarından verileri okuyup CPU'ya beslenebilir hale getirmektir. Otomatik olarak bir dosyadan bir şeyler okumayı sağlayan sınıftır.

testSinifi'nda, dosyaSinifi'ndan kalıtım yapılmıştır. Başka bir sınıftan kalıtım yapıldığını göstermek için "extends" kullanılır. Bu sınıfın içerisinde de değişkenler olmasına rağmen aynı zamanda kalıtım yapıldığı için dosya sınıfının da değişkenlerini içermektedir. Bu sınıfın amacı, dosyaSinifi'nda okunabilmesi için gerekli dosya isimlerini göndermek ve dosyaların doğru okunup/okunmamasına göre simülasyonun başarılı çalışıp çalışmadığını gözlemleyebilmektir.

Bize verilen 3 tane giriş/çıkış dosyası bulunmaktadır. Bu dosyalarının mantığı; FB-CPU'da bellek bulunur ve belleğin içerisindeki bazı sayıların başlangıçta değerleri olması gerekir. Bu değerler bize verilen giriş dosyalarında bulunur. İlk yazılan değer adresi; ikinci değer ise, o adresteki içeriğin ne olacağını ifade eder.

Bu dosyaları okutup satır satır içeriklerini data ve adres değişkenlerinin içerisine alacağız.

III. KULLANILAN YAZILIM

```
40      class dosyaSinifi;
41          int fileDescriptor;
42          int scanFile;
43          int lineCount;
44          logic fileInitialized;
45          logic fileComplated;
46          logic [21:0] data;
47          logic [7:0] memAddr;
48          string dosyaAdi;
49          int sayac;
```

testSinifi'ni yazmadan önce dosyaSinifi'ni yazmamız gerekmektedir. Çünkü testSinifi kalıtım yapmaktadır. Bu yüzden öncelikle dosyaSinifi'nin değişkenlerini tanımlayarak başladık;

- fileDescriptor: Bir dosyayı açarken kullanılır. SystemVerilog'da bir şeyler yazar veya okurken, "oku" dediğimizde geriye bir fileDescriptor döner. Onu tutabilmek için bu sınıfın içerisinde int türünde bir fileDescriptor'e ihtiyaç vardır.
- scanFile: Her bir satırı okudukça doğru okuyup okumadığını gösterir.
- lineCount: Okudunan dosyanın kaç satırdan oluştuğunu sayıp, simülasyonun sonucunda kaç satırı test ettiğini/ kaç satırlık bir okuma yapıldığını ekrana bastırmak için kullanılan değişkendir.
- fileInitialized: Hata çıkmasını engellemek için bu sınıfın içerisinde önceden dosya açmamışken bir şeyleri okumaya çalışırsak, daha dosyayı açıp initalize etmediysek patlamamak için yani korumak için 1 bitlik bir fileInitialized isimli bir logic değişkeni tuttuk. Açılabilecek bir dosya olduğunu gördükten sonra değeri 1'e çekilir; değilse 0'da bırakılır. Bu bize henüz dosyayı okuyamadıysa/açamadıysa, dosyadan okumaya çalışmamasını sağlar.
- fileComplated: Dosyanın sonuna erişip erişmediğini anlamak için kullanılan bir değişkendir. Eğer sonuncu satıra geldiysek artık yeni bir şey okumamayı sağlayan bir saklayıcıdır.
- data ve memAddr: Test giriş dosyalarından okudukça değişkenlerin içerisine doldurmayı sağlar. İlk veri adres, ikinci veri ise data'yı temsil etmektir.

```
34 32
```

Bize kullanmamız için verilen bu değişkenler haricinde ayrıca bizim okunabilirliği arttırmak için kendimizin eklediği iki değişken daha bulunmakta. Bunlar;

- dosyaAdi: Başka fonksiyonların içerisinde hangi dosyanın okunduğu bilgisini ekrana bastırabilmek için tanımladığımız global değişken.
- Sayaç: Hangi dosyanın okunduğu mesajını yazdırırken ekranda mesajın daha güzel görünmesini sağlamak için yazdığımız değişken.

```
51      function new();...
61
62      function int dosyayiInitializeEt( string girisDosyaAdi);...
81
82      function int dosyadanOku( );...
```

Bu sınıfta değişkenlerin haricinde 3 tane de fonksiyon bulunmaktadır.

- new():Constructor'dır. Bu mekanizma bir sınıftan obje türetileceğinde, obje türetilirken başlangıçta yapılması istenen bazı işlemler varsa, o işlemleri gerçekleştirmek için kullanılabilir. Buradaki görevi tüm değişkenleri 0'a atamaktadır.

```
51      function new();
52          fileDescriptor = 0;
53          scanFile = 0;
54          lineCount = 0;
55          fileInitialized = 0;
56          fileComplated = 0;
57          data = 0;
58          memAddr = 0;
59          sayac=0;
60      endfunction
```

- dosyayiInitializeEt(string girisDosyaAdi): Kendisine verilen string argümandaki dosya adı ile dosyayı açmaya çalışır. Dosya açıldığında geriye dönen file descriptor'u, fileDescriptor değişkenine atar. Dosya başarılı olarak açılırsa fileInitialized değişkeni 1 olur, diğer durumda ise 0 olur. Fonksiyon geriye başarılı iken 1, değil iken 0 döndürür. Dosya açma işlemi başarısız olduğunda ekrana "Dosya bulunamadı."yazısı bastırılır ve program durur. Başarılı olduğu takdirde ise bulunan dosyanın adına bakılarak kaçıncı giriş/çıkış dosyanın bulunduğunun bilgisi ekrana yazdırılır.

```
function int dosyayiInitializeEt( string girisDosyaAdi);
    fileDescriptor = $fopen(girisDosyaAdi, "r");
    if (fileDescriptor == `NULL) begin
        fileInitialized = 0;
        $display("Dosya bulunamadı. ");
        $finish;
        return 0;
    end
    else begin
        fileInitialized = 1;
        dosyaAdi = girisDosyaAdi;
        sayac=sayac+1;
        if (dosyaAdi=="input1.txt" ||dosyaAdi=="input2.txt" ||dosyaAdi=="input3.txt")begin
            $display("%d. giriş dosyası bulundu.\n" , sayac);
        end else if (dosyaAdi=="output1.txt" ||dosyaAdi=="output2.txt" ||dosyaAdi=="output3.txt") begin
            $display("%d. çıkış dosyası bulundu.\n" , sayac);
        end
        return 1;
    end
endfunction
```

- dosyadanOku(): fileInitialize 1 ve fileComplated 0 ise bulunmuş bir dosya var olduğu anlamına gelmektedir. Dosyadan 1 satır okuyup bunları memAddr ve data değişkenlerine yazar. "%x" olarak okumasının ve yazdırmasının sebebi dosyaların içerisindeki verilerin heksadesimal olmalarından dolayıdır. Dosyadan her başarılı okunan satır için lineCount değişkenini bir arttırır. Dosyanın sonuna erişilip erişilmediği kontrol edilir. Erişilmediyse dosya okunmaya devam edilir. Erişildiyse fileInitialize 0 ve fileComplated 1 olur. Bu iki değişkenin değerine bağlı olarak ekrana bastırmalar gerçekleştirdik. Okuma bittiğinde dosyanın okumasının bittiği ve kaç satır okunduğu bilgisi bastırılır. Eğer ki bu iki değişken de aynı anda 0 değerine sahipler ise dosyanın eklenmesinde başarısız olunduğu anlamına gelmektedir. Başarılı okunmalarda fonksiyon geriye 1, diğer durumda ise 0 döndürür.

```
function int dosyadanOku( );
    if (fileInitialized == 1 && fileComplated==0) begin
        scanFile = $fscanf(fileDescriptor, "%x %x\n", memAddr, data);
        lineCount=lineCount+1;
        $display(" %d ) memAddr: %x -> data: %x \n",lineCount, memAddr, data );
        if ($feof(fileDescriptor)) begin
            fileComplated=1;
            fileInitialized=0;
        end
        return 1;
    end else if (fileInitialized == 0 && fileComplated==1) begin
        $display("%d dosyası okundu. Okunan satır sayısı: %d \n", dosyaAdi, lineCount);
        fileComplated=0;
        lineCount = 0;
        return 0;
    end else if (fileInitialized == 0 && fileComplated==0)begin
        $display("Dosya initialized edilemedi." );
        return 0;
    end
endfunction
```

dosyaSinifi'nın içini doldurduktan sonra testSinifi isimli sınıfın değişken tanımlamalarına başlanır. dosyaSinifi türünde giriş ve çıkış dosyası isiminde iki tane obje üretilir. testNo ile hangi test dosyasının kullanılacağı fonksiyonlar arası iletilir. Durum1- durum2 ile eşitlik karşılaştırmaları yapılır.

```
class testSinifi extends dosyaSinifi;
    int durum1, durum2;
    int testNo;

    dosyaSinifi girisDosyasi;
    dosyaSinifi cikisDosyasi;
```

Bu sınıfın içerisinde değişkenler haricinde fonksiyonlar da bulunmaktadır. Bunlar;

```
function new();...

function int testNoAyarla( int girisTestNo );...

function int testInitializeEt( );...

function int kontrolEt( reg [7:0] memLocation, reg [21:0] expectedValue );...
```

- new(): Constructor'dır. Base class'ta contructor olduğu ve argüman aldığı için, super.new syntax'ı ile base class'ın constructor'una erişilir. testNo değişkenini 0'a atar ve girisDosyasi, cikisDosyasi değişkenlerini new ile initialize eder. Sınıftan bir obje türetildiği zaman henüz aslında RAM'de sınıftaki değişkenler için yer açılmaz. Yani objenin DRAM'de tutulduğu bir adres yoktur. Bu yüzden NULL değerini taşımaktadır. Objenin bellek tahsisi için "new" operatörü ile atama yapılması gerekmektedir. New ifadesi sonucunda, oluşan obje DRAM'de atanan adresi göstermektedir.

```
function new();
    super.new();
    testNo = 0;
    girisDosyasi =new;
    cikisDosyasi =new;
```

- testNoAyarla(int girisTestNo): testNo değişkenine girisTestNo argümanını yazılır.

```
function int testNoAyarla( int girisTestNo );
    testNo = girisTestNo;
endfunction
```

- testInitializeEt(): Sınıfın içinde bulunan testNo değişkenin değerine göre girisDosyasi.dosyayiInitializeEt fonksiyonu ile input1/output1.txt gibi dosyalardan birini açar. Dosyaların açılmasında sorun olursa dosyayiInitializeEt fonksiyonun içerisindeki finish komutu ile simülasyonu durdurulur.

```
function int testInitializeEt( );
    case(testNo)
        0: begin
            girisDosyasi.dosyayiInitializeEt("input1.txt");
            cikisDosyasi.dosyayiInitializeEt("output1.txt");
        end
        1: begin
            girisDosyasi.dosyayiInitializeEt("input2.txt");
            cikisDosyasi.dosyayiInitializeEt("output2.txt");
        end
        2: begin
            girisDosyasi.dosyayiInitializeEt("input3.txt");
            cikisDosyasi.dosyayiInitializeEt("output3.txt");
        end
    endcase
endfunction
```

- kontrolEt(reg [7:0] memLocation, reg [21:0] expectedValue): Kendisine argüman olarak verilen memLocation bilgisini kullanarak, BRAM'deki adres'e bakar. O adresteki içeriğin değeri ile expectedValue değerini karşılaştırır. Aynı ise simülasyon başarılı olarak çıktı verir, değil ise simülasyon hatalı olarak çıktı verir.

```
function int kontrolEt( reg [7:0] memLocation, reg [21:0] expectedValue );
        durum1 = blram.memory[memLocation];
        durum2 = expectedValue;
        if(durum1 !== durum2)begin
            $display("Simülasyon hatalı.");
        end
        else begin
            $display("Simülasyon başarılı.");
        end
    endfunction
```

Tüm bu fonksiyon ve sınıflar initial begin bloğunun içerisinden çağrılmaktadır. Burada testSinifi sınıfından test isminde bir obje türetilir ve testSinifi'nın çağrılmasında kullanılır. For döngüsü ile testNo ataması yapılır ve while

döngüsü ile simülasyonun doğru çalışıp çalışmadığını kontrol etmek için fonksiyonlara gidilir.

```
initial begin
    testSinifi test;
    test = new;
    clk = 0;
    rst = 0;
    for (int i = 0; i<3 ;i = i+1) begin
        $display("Su anki Test no:  %d\n",i );
        test.testNoAyarla(i);
        test.testInitializeEt();
        while(test.girisDosyasi.dosyadanOku() == 1) begin
            blram.memory[test.girisDosyasi.memAddr] = test.girisDosyasi.data;
            //@(posedge clk );
        end

        rst <= #1 1;
        repeat(10) @(posedge clk );
        rst <= #1 0;
        repeat(10000) @(posedge clk );
        while (test.cikisDosyasi.dosyadanOku() == 1) begin
            $display("TEST SONUCU:" );
            test.kontrolEt(test.cikisDosyasi.memAddr, test.cikisDosyasi.data);
```

IV. SONUÇLAR

TestBench'i yazdıktan sonra kodu simüle ettik. Çıktılarımızı gözlemledik.

```
Su anki Test no:            0

         1. giriş dosyası bulundu.

         1. çıkış dosyası bulundu.

          1 ) memAddr: 00 -> data: 000032
          2 ) memAddr: 01 -> data: 0000b3
          3 ) memAddr: 02 -> data: 000074
          4 ) memAddr: 03 -> data: 000240
          5 ) memAddr: 32 -> data: 000005
          6 ) memAddr: 33 -> data: 00000a

input1.txt dosyası okundu. Okunan satır sayısı:          6

          1 ) memAddr: 34 -> data: 00000f

TEST SONUCU:
Simülasyon başarılı.
output1.txt dosyası okundu. Okunan satır sayısı:          1

Bitirilen Test            0
```

```
0 32
1 B3
2 74
3 240
32 5
33 A
```

```
34 F
```

İlk olarak test numarası 0 geldiği için 1. giriş dosyası olan "input1.txt" bulundu ve okundu. Okunan içeriği txt dosyaları ile biz de karşılaştırdık ve doğru okunmuş olduğunu gözlemledik. Okunan toplam satırı rahat gözlemleyebilmek için her çıktıya madde numarası atadık ve hesaplanan okunmuş satır numarası ile karşılaştırdığımızda her satırın okunmuş olduğunu gördük. Test sonucunun başarılı olduğu yazısını gördük ve sonrasında test numarası 0 olan okumanın gerçekleştiği bilgisi basıldı. Aynı çıktıları test numarası 1 ve 2'de de gözlemledik. Test numarası 1;

```
Su anki Test no:            1

         2. giriş dosyası bulundu.

         2. çıkış dosyası bulundu.

          1 ) memAddr: 00 -> data: 000032
          2 ) memAddr: 01 -> data: 000133
          3 ) memAddr: 02 -> data: 000074
          4 ) memAddr: 03 -> data: 000240
          5 ) memAddr: 32 -> data: 000005
          6 ) memAddr: 33 -> data: 00000a

input2.txt dosyası okundu. Okunan satır sayısı:          6

          1 ) memAddr: 34 -> data: 000032

TEST SONUCU:
Simülasyon başarılı.
output2.txt dosyası okundu. Okunan satır sayısı:          1

Bitirilen Test            1
```

```
0 32
1 133
2 74
3 240
32 5
33 A
```

```
34 32
```

Test numarası 2;

```
Su anki Test no:            2

         3. giriş dosyası bulundu.

         3. çıkış dosyası bulundu.

          1 ) memAddr: 00 -> data: 000033
          2 ) memAddr: 01 -> data: 0000f1
          3 ) memAddr: 02 -> data: 0001ca
          4 ) memAddr: 03 -> data: 000030
          5 ) memAddr: 04 -> data: 0000b2
          6 ) memAddr: 05 -> data: 000070
          7 ) memAddr: 06 -> data: 000031
          8 ) memAddr: 07 -> data: 0000ae
          9 ) memAddr: 08 -> data: 000071
         10 ) memAddr: 09 -> data: 000180
         11 ) memAddr: 0a -> data: 000030
         12 ) memAddr: 0b -> data: 000074
         13 ) memAddr: 0c -> data: 000240
         14 ) memAddr: 2e -> data: 000001
         15 ) memAddr: 30 -> data: 000000
         16 ) memAddr: 31 -> data: 000000
         17 ) memAddr: 32 -> data: 000005
         18 ) memAddr: 33 -> data: 00000a

input3.txt dosyası okundu. Okunan satır sayısı:         18

          1 ) memAddr: 34 -> data: 000032

TEST SONUCU:
Simülasyon başarılı.
output3.txt dosyası okundu. Okunan satır sayısı:          1

Bitirilen Test            2
```

```
0 33
1 F1
2 1CA
3 30
4 B2
5 70
6 31
7 AE
8 71
9 180
A 30
B 74
C 240

2E 1
30 0
31 0
32 5
33 A
```

```
34 32
```

En son ise tcl konsolda simülasyonun tamamlandığının yazısını gördük.

```
Simulasyon Tamamlandı
```

Tüm test dosyaları başaralı bir şekilde okunmasının haricinde waveform ekranını incelediğimizde verilerimizin okunduğunu gözlemlemiş olduk.

Bu proje sonrasında geçen yıl dijital sistem tasarımında yazmış olduğumuz FB-CPU kodunun doğruluğunu otonom şekilde doğrulamış olduk. SystemVerilog diline olan hakimiyetimizin artması haricinde bu dildeki objeye yönelimli programlama yöntemlerini, sınıf kullanımını öğrenmiş ve deneyimlemiş olduk.

PROJE EKİBİ

Damla Su KARADOĞAN, 11.02.2001 yılında doğdu. 2019 yılında Özel Envar Anadolu Lisesinden mezun oldu. Şu anda Fenerbahçe Üniversitesinde Endüstri Mühendisliği bölümünde lisans eğitimini almakta ve Bilgisayar Mühendisliğinde ÇAP eğitimi alıyor. Öğrenci numarası, 190302016.

Alp Eren Gürle, 13.10.2000 yılında doğdu. 2018 yılında Denizli Uğur Anadolu Lisesinden mezun oldu. Şu anda Fenerbahçe Üniversitesinde Bilgisayar Mühendisliği bölümünde lisans eğitimi almakta. Öğrenci numarası, 190301028.

Taha Yasin Öztürk, 20.11.2000 yılında doğdu. 2019 yılında Erbakır Fen Lisesinden mezun oldu. Şu anda Fenerbahçe Üniversitesinde Bilgisayar Mühendisliği bölümünde lisans eğitimi almakta. Öğrenci numarası, 190301027.

REFERANS DOSYALAR

Youtube Linki: https://youtu.be/n_iTWcmLa7s

Github Linki: https://github.com/damlasu/FB-CPU-SystemVerilog-Testbench

KAYNAKLAR

[1] *http://www.levent.tc/courses/electronic-circuits*

[2] *https://www.chipverify.com/systemverilog/systemverilog-functions*

[3] *http://www.levent.tc/courses/electronic-circuits*