FB-CPU RTL TASARIMI

Aysen İpek Çakır, Ceyda Uymaz, Deniz Uzun, İrem Kalkanlı, Özlem Çalı
Fenerbahçe Üniversitesi
Bilgisayar Mühendisliği
İstanbul, Türkiye
e-mail: {aysen.cakir, ceyda.uymaz, deniz.uzun, irem.kalkanli, ozlem.cali}@fbu.edu.tr,

***Özetçe*——** **Makine dilinde yazılan 10 farklı operasyon kodu çalıştırabilen bir işlemci tasarımı geliştirilecektir.**

Anahtar Kelimeler — FPGA, CPU

***Abstract*— A CPU design that can run 10 different transaction codes written in machine language will be developed.**

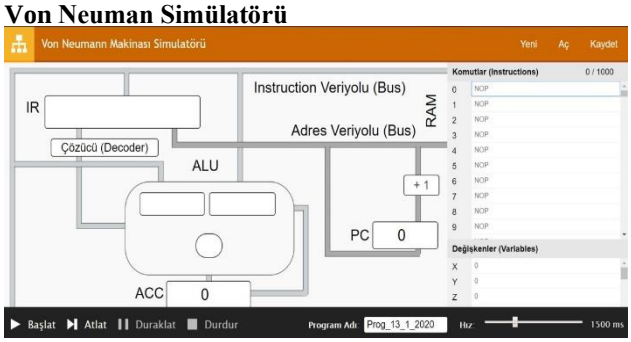
Keywords — FPGA, CPU.

I. 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.

II. 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

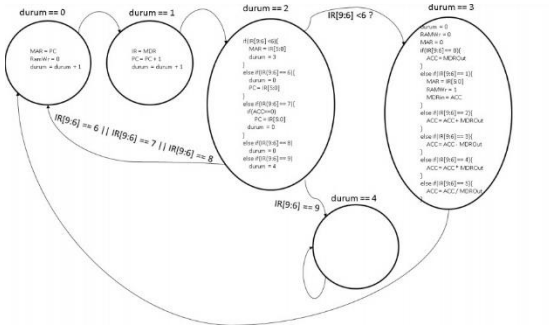
Xilinx Vivado Design Suite, FPGA geliştirme kartları üzerinde çalışmalar yapmak için gerekli olan tasarımı oluşturmak için kullanılmaktadır. Verilog, VHDL vb.. donanım tasarım dillerini alarak, FPGA'e konfigüre edilebilecek (Xilinx firması FPGA'leri için .bit uzantılı dosyalar) tasarım dosyasını oluşturur.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

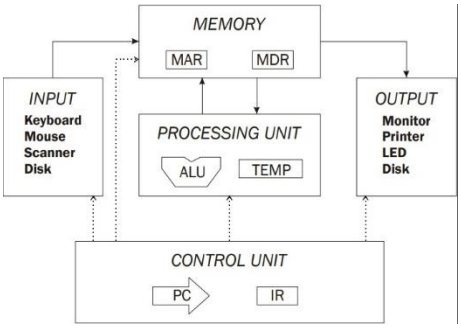
- Verilog
- System Verilog

• VHDL Dillerini desteklemektedir. Projede Verilog dili ile tasarımlar yapılacaktır.

III. KULLANILAN YAZILIM



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

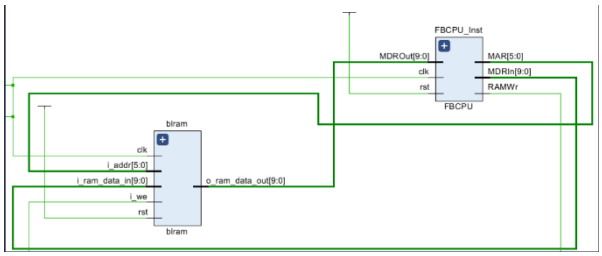
Memory'de işlemcideki kodlar bulunur. İşlemci hangi komutu çalıştıracaksa onu çalıştırıp tekrar Memory'ye yazar.
Temp, işlemcideki geçici bellek görevini görür.
İşlem Ünitesi(ALU),işlemlerin gerçekleştiği yerdir.

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 2^6 lokasyonu olmasındandır. Dolayısıyla PC değeri RAM'deki her yeri gösterebilmektedir.
IR (10 Bit): Instruction Register, RAM'den okunan kodun (instruction) saklandığı saklayıcıdır.

```
always@(posedge clk) begin
    durum       <= #1 durumNext;
    PC          <= #1 PCNext;
    IR          <= #1 IRNext;
    ACC         <= #1 ACCNext;
end
```

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 yanda verilmiştir.

•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. 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

```
`timescale 1ns / 1ps

module FBCPU #(
    parameter ADDRESS_WIDTH = 6,
    parameter DATA_WIDTH = 10
)(
    input clk,
    input rst,
    output reg [DATA_WIDTH-1:0] MDRIn,
    output reg RAMWr,
    output reg [ADDRESS_WIDTH-1:0] MAR,
    input [DATA_WIDTH-1:0] MDROut,
    output reg [5:0] PC
);

reg [DATA_WIDTH - 1:0] IR, IRNext;
reg [5:0] PCNext;
reg [9:0] ACC, ACCNext;
reg [2:0] durum, durumNext;
```

Buradaki kodlar işlemcinin kendisini barındırır.

```
always@(posedge clk) begin
    durum       <= #1 durumNext;
    PC          <= #1 PCNext;
    IR          <= #1 IRNext;
    ACC         <= #1 ACCNext;
end
always@(*) begin
    durumNext   = durum;
    PCNext      = PC;
    IRNext      = IR;
    ACCNext     = ACC;
    MAR         = 0;
    RAMWr       = 0;
    MDRIn       = 0;

    if(rst) begin
        durumNext   = 0;
        PCNext      = 0;
        MAR         = 0;
        RAMWr       = 0;
        IRNext      = 0;
        ACCNext     = 0;
        MDRIn       = 0;
    end else begin
```

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

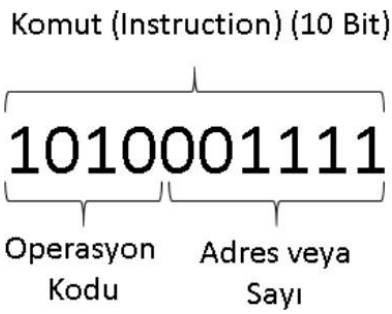
- **İşlem Ünitesi (ALU, Arithmetic Logic Unit):** Aritmetik işlemlerin gerçekleştirildiği bölümdür. FB-CPU'da 4 adet aritmetik işlem vardır. Bunlar toplama, çıkartma, çarpma ve bölmedir, gelen operasyon koduna göre işlemleri gerçekleştirip ACC saklayıcısına yazmaktadır.
- **Kontrol Ünitesi**: Saklayıcılar, 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 1. FB-CPU ISA (Instruction Set Architecture)

Komut Adı	Görevi	Operasyon Kodu
LOD ADDR	Yükleme (Load), Bellekteki verilen adresin içerisinden değeri alıp, ACC saklayıcısına yerleştirir. ACC = *(ADDR)	0000
STO ADDR	Kaydetme (Store), ACC'nin içerisindeki değeri alıp, bellekte verilen adrese yazar. *(ADDR) = ACC	0001
ADD ADDR	Bellekteki verilen adresteki değeri alır, ACC ile toplayıp, ACC'nin üzerine yazar. ACC = ACC +*(ADDR)	0010
SUB ADDR	Bellekteki verilen adresteki değeri alır, ACC ile çıkartıp, ACC'nin üzerine yazar. ACC = ACC - *(ADDR)	0011
MUL ADDR	Bellekteki verilen adresteki değeri alır, ACC ile çarpıp, ACC'nin üzerine yazar. ACC = ACC * (*(ADDR))	0100
DIV ADDR	Bellekteki verilen adresteki değeri alır, ACC ile bölüp, ACC'nin üzerine yazar. ACC = ACC / (*(ADDR))	0101
JMP SAYI	PC = Sayı olur.	0110
JMZ SAYI	ACC'in değeri 0 ise, verilen sayı değerini PC'e atar, değilse işlem yapmaz.	0111
NOP	No Operation, hiçbir işlem yapılmaz.	1000
HLT	Uygulama durur	1001

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.

```
case (durum)
    0: begin//0.durum
        MAR = PC;
        RAMWr = 0;
        durumNext = durum + 1;
    end
```

Durum==1, Verilog dilinde böyle verilmiştir.

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

```
2: begin//2.durum
   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) begin
   PCNext=IR[5:0];
   end
   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 = = 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.

```
3: begin//3.durum
durumNext=0;
RAMWr=0;
MAR=0;
if(IR[9:6]==0)begin
ACCNext=MDROut;
end 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
 ACCNext=ACC/MDROut;
end
end
```

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.)

```
            4: begin//4.durum
            durumNext=4;
            end
        endcase
    end
end

endmodule
```

Durum 4’te işlemlerimizi bitirerek çıkış yapılır.

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

```
`timescale 1ns / 1ps

// FB - CPU Testbench Dosyasi

module tb_fbcpu;

    parameter TEST_CASE = 3;

    parameter ADDRESS_WIDTH = 6;
    parameter DATA_WIDTH = 10;

    reg clk = 1;
    reg rst;

    wire [ADDRESS_WIDTH-1:0] addr_toRAM;
    wire [DATA_WIDTH-1:0] data_toRAM, data_fromRAM;
    wire [ADDRESS_WIDTH-1:0] pCounter;
    wire wrEn;

    always clk = #5 !clk;
```

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.

```
    always clk = #5 !clk;

    initial begin
      rst = 1;
      repeat (10) @(posedge clk);
      rst <= #1 0;
      repeat (10000) @(posedge clk);

      if(TEST_CASE == 1)
        memCheck(52,15);

      else if(TEST_CASE == 2)
        memCheck(52,50);

      else if(TEST_CASE == 3)
        memCheck(52,50);

      $finish;
    end
```

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.

```
    FBCPU #(
        ADDRESS_WIDTH,
        DATA_WIDTH
    ) FBCPU_Inst(
        .clk(clk),
        .rst(rst),
        .MDRIn(data_toRAM),
        .RAMWr(wrEn),
        .MAR(addr_toRAM),
        .MDROut(data_fromRAM),
        .PC(pCounter)
    );

    blram #(ADDRESS_WIDTH, 64, TEST_CASE) blram(
      .clk(clk),
      .rst(rst),
      .i_we(wrEn),
      .i_addr(addr_toRAM),
      .i_ram_data_in(data_toRAM),
      .o_ram_data_out(data_fromRAM)
    );
```

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

```
    task memCheck;
        input [31:0] memLocation, expectedValue;
        begin
          if(blram.memory[memLocation] != expectedValue) begin
                $display("Test Hatali Tamamlandi!");
          end else begin
                $display("Test Basarili Tamamlandi!");
          end
        end
    endtask

endmodule
```

İşlemcinin düzgün çalışıp çalışmadığını kontrol etmek için tasarlanan kısımdır.

MEMORY.V
Burada bulunan kodlar Ram’in kendisini tanımlıyor.

```
module blram(clk, rst, i_we, i_addr, i_ram_data_in, o_ram_data_out);

parameter SIZE = 6;
parameter DEPTH = 64;
parameter TEST_CASE = 1;

input clk;
input rst;
input i_we;
input [SIZE-1:0] i_addr;
input [9:0] i_ram_data_in;
output reg [9:0] o_ram_data_out;

reg [9:0] memory[0:DEPTH-1];

always @(posedge clk) begin
  o_ram_data_out <= #1 memory[i_addr[SIZE-1:0]];
  if (i_we)
      memory[i_addr[SIZE-1:0]] <= #1 i_ram_data_in;
end
```

```
initial begin
    if(TEST_CASE == 1) begin
        `include "testCase1.v"
    end else if(TEST_CASE == 2) begin
        `include "testCase2.v"
    end else if(TEST_CASE == 3) begin
        `include "testCase3.v"
    end
end

endmodule
```

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.

```
memory[0] = 10'b0000_110010; // LOD 50, (ACC = *50), Hex = 32
memory[1] = 10'b0010_110011; // ADD 51, ACC = ACC + (*51), Hex = B3
memory[2] = 10'b0001_110100; // STO 52, (*52) = ACC, Hex = 74
memory[3] = 10'b1001_000000; // Halt, Hex = 240
memory[50] = 10'b0000_000101; // Hex = 5
memory[51] = 10'b0000_001010; // Hex = A
```

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.

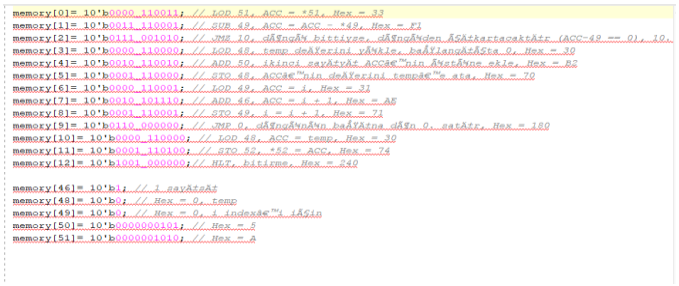
testCase2.v
FB-CPU için bellekte 50 ve 51 adresteki iki sayının çarpımını 52 no'lu adrese kaydeden uygulamayı inceleyelim.

```
memory[0] = 10'b0000_110010; // LOD 50, (ACC = *50), Hex = 32
memory[1] = 10'b0100_110011; // MUL 51, ACC = ACC * (*51), Hex = 133
memory[2] = 10'b0001_110100; // STO 52, (*52) = ACC, Hex = 74
memory[3] = 10'b1001_000000; // Halt, Hex = 240
memory[50] = 10'b0000_000101; // Hex = 5
memory[51] = 10'b0000_001010; // Hex = A
```

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

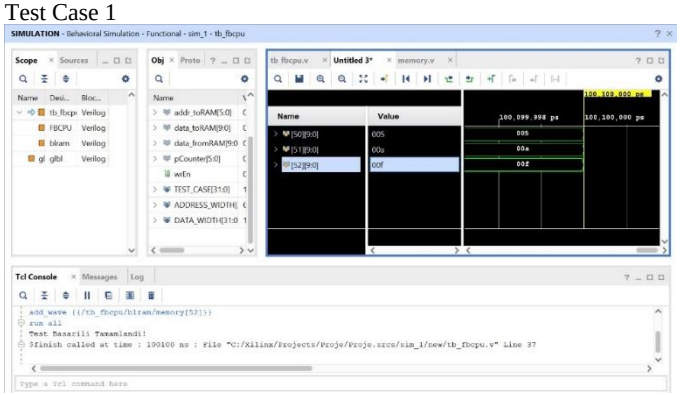
FB-CPU için bellekte 50 ve 51. adresteki iki sayının çarpımını 52 no'lu adrese kaydeden uygulamayı inceleyelim.



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

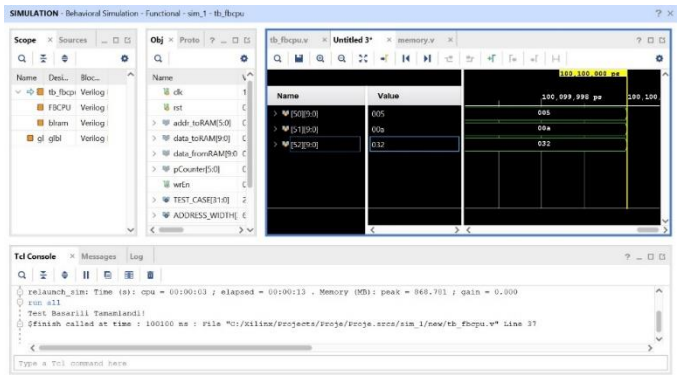
ÇIKTILAR

Test Case 1



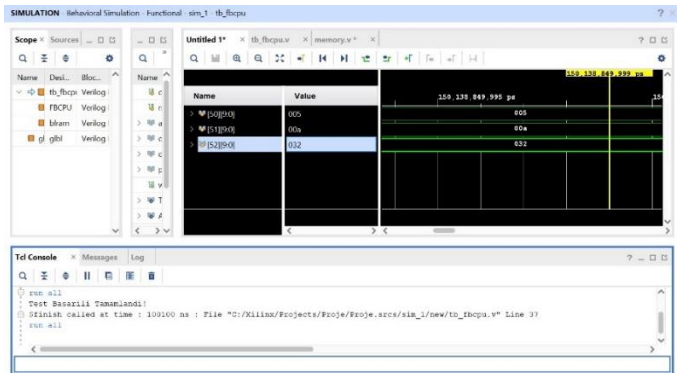
[52] [9:0] adresi 00f (decimal olarak 15) olduğundan ve alttaki konsolda "Test Basarili Tamamlandi" yazısından, işlemcinin yapmasını istediğimiz işlemi başarılı bir şekilde tamamladığını görebiliriz.

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.

IV. 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

PROJE EKİBİ

İrem KALKANLI (Proje Ekip Sorumlusu):

Okul numarası:190301007

Doğum Tarihi:15.01.2000

Doğum Yeri: İstanbul

Mezun Olduğu Lise: Ataşehir 3 Doğa Koleji

Deniz UZUN:

Okul numarası:190301015

Doğum Tarihi:08.04.2001

Doğum Yeri: İstanbul

Mezun Olduğu Lise: Kavacık Uğur Anadolu Lisesi

Özlem ÇALI:

Okul numarası:190301002

Doğum Tarihi:19.05.2000

Doğum Yeri: Hatay

Mezun Olduğu Lise: Necmi Asfuroğlu Anadolu Lisesi

Aysen İpek ÇAKIR:

Okul numarası:190301001

Doğum Tarihi:20.03.2001

Doğum Yeri: Malatya

Mezun Olduğu Lise: Fethi Gemuhluoğlu Fen Lisesi

Ceyda UYMAZ:

Okul numarası:200301503

Doğum Tarihi:26.08.2000

Doğum Yeri:İstanbul

Mezun Olduğu Lise: Celal Aras Anadolu Lisesi

REFERANS DOSYALAR

Youtube: https://www.youtube.com/watch?v=TfiC3wqjeZY&feature=youtu.be

Github: https://github.com/iremkalkanli/BLM-201-FBU-CPU-RTL-Tasarimi

KAYNAKLAR

[1] Levent, Vecdi Emre (2019) “Von Neumann Mimarisi”, *Bilgisayar Mühedisliğine Giriş-Ders Notları*.

[2] Levent, Vecdi Emre (2020) “Durum Makinaları”, *Mantıksal Sistem Tasarımı-Ders Notları*.

[3] Levent, Vecdi Emre (2020) “Veriyolu Elemanları”, *Mantıksal Sistem Tasarımı-Ders Notları*.

[4] Levent, Vecdi Emre (2020) “Bellekler”, *Mantıksal Sistem Tasarımı-Ders Notları*.

[5] Levent, Vecdi Emre (2020) “FB-CPU RTL Tasarım”, *Mantıksal Sistem Tasarımı-Ders Notları*.