RISC-V TABANLI İŞLEMCİ TASARIMI

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

Özetçe—

Anahtar Kelimeler — FPGA, CPU, RISC-V, Systemverilog

***Abstract*— A RISC-V CPU design that can run 11 different operation codes written in system verilog language will be developed.**

Keywords — FPGA, CPU,RISC-V, SystemVerilog

I. GİRİŞ

Proje kapsamında başlangıç tasarım verilen bir RISC-V işlemcisinin ALU ve instruction decoder blokları temel SystemVerilog dili özellikleri kullanılarak tasarım ve doğrulama çalışmaları yapılacaktır

II. SİSTEM MİMARİSİ

1) 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

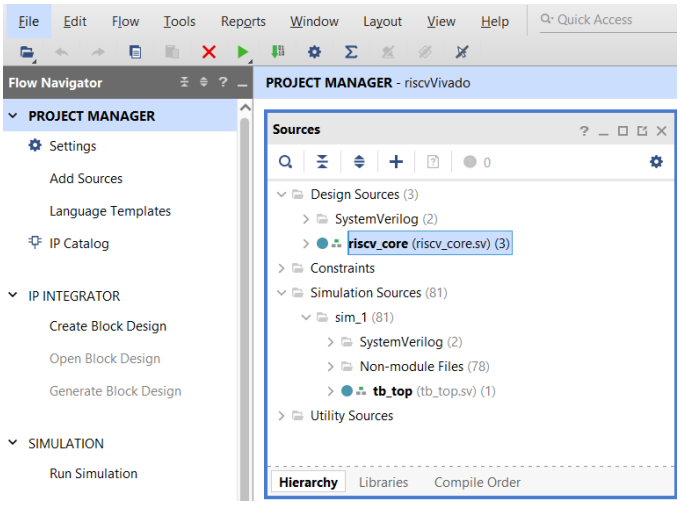
• VHDL Dillerini desteklemektedir. Projede System Verilog dili ile tasarımlar yapılacaktır.

III. KULLANILAN YAZILIM

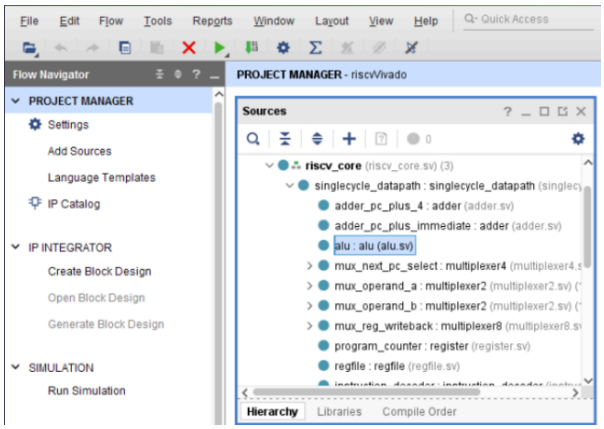
Tasarım Gereksinimleri

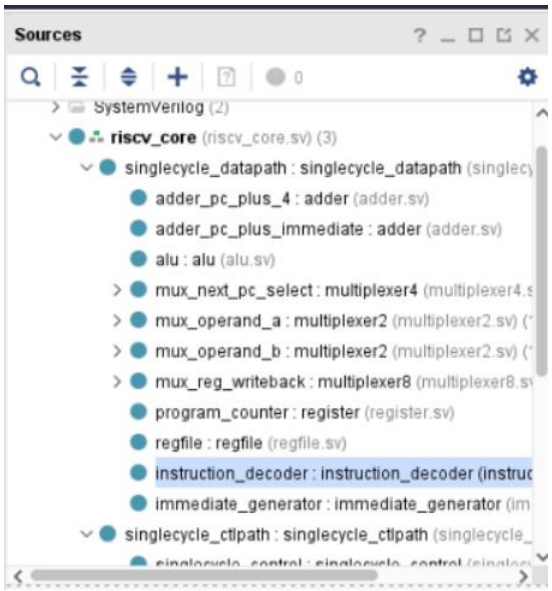
Proje açıldığında verilen başlangıç dosyaları aşağıdaki şekilde görülmelidir. Design source altında başlangıç tasarımının tepe modülü olan riscv_core system verilog dosyası, simulation source bölümünde ise tb_top systemverilog dosyası bulunmaktadır.

Tasarım büyüdükçe, projeyi daha iyi oluşturmak için, yapılan çalışmaları işlevlerine göre modüllere ayırmak kolaylık ve avantaj sağlar. Yapılacak olan işlemcinin birçok modüle ayrılmasının sebebi, daha hızlı ve planlı şekilde projeyi yönetmektir.



Şekilde gösterilen tb_top modülü, riscv_core modülünü içerisinde barındırarak projenin bitiminde doğru çalışıp çalışmadığını kontrol etmek amaçlı yazılmıştır.
Tasarımda tanımı yapılmış ancak gerçeklemesi yapılmamış modüller bulunmaktadır. Bunlar;
• ALU
• Instruction Decoder
modülleridir. Bunlar şekilde gösterilen konumlardadır.





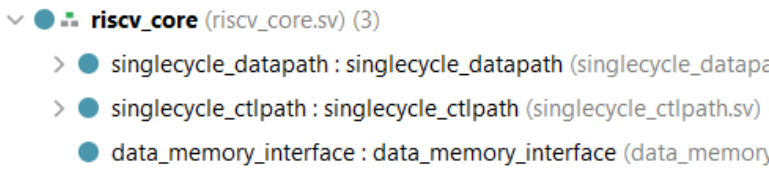
Projenin tepe modülü olan riscv_core modülünün giriş ve çıkış sinyallerine bakalım.

```
module riscv_core (
    input  clock,
    input  reset,

    output [31:0] bus_address,
    input  [31:0] bus_read_data,
    output [31:0] bus_write_data,
    output [3:0]  bus_byte_enable,
    output        bus_read_enable,
    output        bus_write_enable,

    input  [31:0] inst,
    output [31:0] pc
);
```

riscv_core tasarımın tepe modülüdür. Bu modülde clock ve reset girişleri bulunuyor. Bu bir işlemci olduğu için komutları Ram'den alıp Ram'e yazıyor. Bu okuyup yazma işlemlerinin yapılabilmesi için;

bus_address-> Ram'in adres girişini okuyan sinyal
bus_read_data-> Ram'den bir veri okunacağı zaman kullanılan giriş sinyali
bus_write_data-> Ram'e bir veri yazılacağı zaman kullanılam çıkış sinyali
bus_byte_enable-> Ram'e yazılacak olan hangi 8 bitlik kısmın kullanılacağını ifade eden sinyal
bus_read_enable->Ram'den bir şey okunacağı zaman aktif edilen sinyal
bus_write_enable-> Ram'e bir şey yazılacağı zaman aktif edilen sinyal

Ayrıca testbench için gerekli olan, o an işlemcinin hangi komutu işlediğini gösteren giriş var.
32 bitlik inst ve pc sinyalleri testbench'te kullanılıyor.



Tepe modülün içeriğine bakacak olursak 3 tane temel modül görülür.
single_datapath->İçerisinde karar verilen işlemlerin yapıldığı modül. Ne zaman çalışacağını single_ctlpath modülü kontrol ediyor.
single_ctlpath->Aritmetik Lojik Ünitesinin(ALU) ve işlemcinin ne yapacağını karar veren modül
data_memory_interface->Bellekle ilgili işlemlerin yapılmasını sağlayan modül

ALU

Yapılacak olan işlemcinin ALU'sunun destekleyeceği 11 adet işlem bulunuyor. Bu işlemlerden hangisinin yapılacağı alu_function girişinden gelmektedir. İşlemlere göre a ve b sayıları, result isminde sonuç çıkışı ve sonuç eğer sıfır ise, ayrı bir çıkış olarak sonucun sıfır olması durumunda 1 olan bir çıktı vardır.

ALU'nun desteklediği işlemler ve operasyon kodları aşağıdaki şekilde verilmektedir.

```
ALU_ADD   5'b00001
ALU_SUB   5'b00010
ALU_SLL   5'b00011
ALU_SRL   5'b00100
ALU_SRA   5'b00101
ALU_SEQ   5'b00110
ALU_SLT   5'b00111
ALU_SLTU  5'b01000
ALU_XOR   5'b01001
ALU_OR    5'b01010
ALU_AND   5'b01011
```

Operasyonların açıklamaları aşağıda listelenmektedir.

• ADD: A + B
• SUB: A - B
• SLL: A << B
• SLR: A >> B
• SRA: A >>> B
• SEQ: A == B
• SLT: A < B
• SLTU: $unsigned(A) < $unsigned(B)
• XOR: A ^ B
• OR: A | B
• AND: A & B

ALU ünitesinin giriş ve çıkış sinyalleri gösterilmektedir.

```
module alu (
    input        [4:0]  alu_function,
    input signed [31:0] operand_a,
    input signed [31:0] operand_b,
    output logic [31:0] result,
    output              result_equal_zero
);
```

3 tane girişi, 2 tane çıkışı var.
alu_function-> Operasyonun ne olduğunu belirtir.
operand_a ve operand_b -> İşlem yapılacak 32 bitlik sayıları ifade eder.
result-> Operasyona göre işlem yapılır ve sonuç buraya yazılır.32 bittir.
result_equal_zero -> result'un değerine göre sonuç üretir.
result==0 ise **result_equal_zero** ->1
result==0 değilse **result_equal_zero** ->0

Alu'nun giriş ve çıkış sinyallerine bakarak, alu_function'un 5 bitlik olmasından dolayı bu tasarımda 32 farklı operasyon yapılabileceği söylenebilir.

```
logic  z;
assign result_equal_zero=z;

always_comb begin

case(alu_function)

5'b00001: begin
result=operand_a+operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end


5'b00010: begin
result=operand_a-operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end
```

Modülde yazılan bu fonksiyonda case yapısı kullanılmıştır. İşlemlerden hangisinin yapılacağı alu_function değerine göre gelmektedir.Gelen operasyon koduna göre işlem yapılmaktadır. İşlem sonuçları result'a kaydedilmektedir.

result'un sonucuna göre result_equal_zero sinyalinin değeri belirleniyor.
result==0 ise result_equal_zero değeri 1 oluyor.
result==0 değilse result_equal_zero değeri 0 oluyor.
Bu eşitliği sağlamak için bir bağlantı değeri olan z kullanıldı.

```
5'b00011: begin
result=operand_a<<operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end

5'b00100: begin
result=operand_a>>operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end

end

5'b00101: begin
result=operand_a>>>operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end
```

İstenen 11 tane komut operasyon kodlarına göre, yapılacağı işlem belirlenerek tasarlanır.
result değerine göre de her bir kod bloğunda z'nin değeri belirlenir.

```
5'b00110: begin
result=operand_a==operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end

5'b00111: begin
result=operand_a<operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end

5'b01000: begin
result=unsigned'(operand_a)+ unsigned'(operand_b);
if (result==0) begin
z=1;
end else begin
z=0;
end
end
5'b01001: begin
result=operand_a^operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end

5'b01010: begin
result=operand_a|operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end

5'b01011: begin
result=operand_a&operand_b;
if (result==0) begin
z=1;
end else begin
z=0;
end
end

endcase

end
endmodule
```

Instruction Decoder

Instruction Decoder ünitesinin giriş ve çıkış sinyalleri aşağıda gösterilmektedir.

```
module instruction_decoder(
    input [31:0] inst,
    output [6:0] inst_opcode,
    output [2:0] inst_funct3,
    output [6:0] inst_funct7,
    output [4:0] inst_rd,
    output [4:0] inst_rs1,
    output [4:0] inst_rs2
);
```

Modülde giriş olarak 32 bitlik instruction word'u alınmaktadır. 32 bitlik inst girişine girilen değerleri kısım kısım ayırarak görevlerini yapmak üzere ayırır.

Çıkışta ise instruction'un parse edilmiş hali, yani decode edilmiş hali çıkış olarak verilmektedir.

• Opcode, instruction'un ilk 7 bitini yani [6:0]'ı temsil etmekte
• Func3, instruction'un 14-12 bitleri arasını [14:12];
• Func7, instruction'un 31-25 bitleri arasını [31:25];
• Rd, instruction'un 11-7 bitleri arasını [11:7];
• RS1, instruction'un 19-15 bitleri arasını [19:15];
• RS2, instruction'un 24-20 bitleri arasını [24:20];

Buna göre instruction sinyali parçalanarak ilgili sinyallerin üzerlerine atama yapılmalıdır.

```
logic [6:0] OP;
logic [2:0] Func3;
logic [6:0] Func7;
logic [4:0] RD;
logic [4:0] RS1;
logic [4:0] RS2;

always_latch begin
OP=inst[6:0];
Func3=inst[14:12];
Func7=inst[31:25];
RD=inst[11:7];
RS1=inst[19:15];
RS2=inst[24:20];
end
```

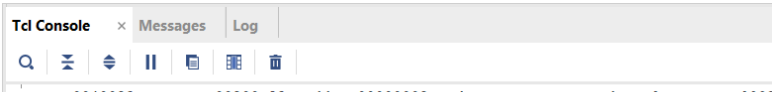
Modülde verilen her output için bir logic değişkeni oluşturduk. Logic veri türü sayesinde sentezleyici output'un reg veya wire olmasını sentez aşamısında belirler. Always_latch bloğunda logic değişkenlerimize inst inputundaki atanması gereken bit aralıklarını atıyoruz.

```
assign inst_opcode=OP;
assign inst_funct3=Func3;
assign inst_funct7=Func7;
assign inst_rd=RD;
assign inst_rs1=RS1;
assign inst_rs2=RS2;

endmodule
```

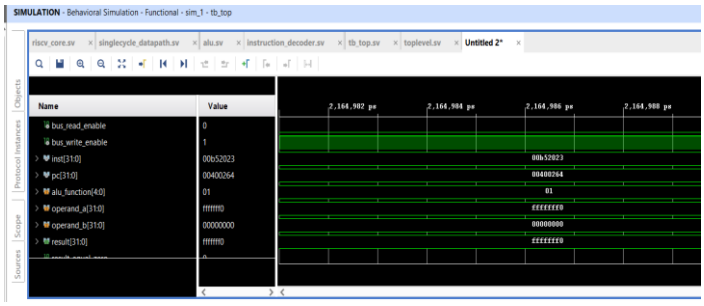
Assign ifadesi ile outputlar, logic değerlerine atanır.

Proje Testi

Projenin doğru çalıştığının test edilmesi için başlangıç test kodları verilmiştir. Test kodunda daha önceden hazırlanmış olan bir CPU test uygulamasının makine diline döndürülmüş halini, işlemciye besleyip sonucunu kontrol eden bir uygulama bulunmaktadır.
Tasarım simülasyonu başlatılıp play tuşuna basıldığında, en fazla 10000 cycle bekleyip, sonuç hesaplanmış ise aşağıdaki şekilde görülebilen **PASS** çıktısını vermektedir. Aksi takdirde **FAİL** çıktısı verecektir.



Tasarımın testbench modülleriyle test edilmesi sonucu **PASS** sonucunu vermesi doğru çalıştığını göstermektedir.



Projenin simülasyon kısmı yukarıda şekildeki gibidir.

IV. SONUÇLAR

Geliştirilen RISC-V CPU işlemcisi belirlenen koşulları sağladığında 11 adet komutu yerine getirip işlem yapabilmektedir.Yapılan işlemcinin risc-v temelli olmasının avantajları görülmüştür. İnstructor ve alu ünitelerinin farklı bölümlere ayrılması projenin yönetilmesinin daha kolay olmasını ve hızlı çalışmasını sağlamıştır.

Sonuç olarak RISC-V CPU işlemcisi makine dilindeki kodlarla 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

İşlemciyi yapabilmek için saklayı

https://github.com/iremkalkanli/BLM-202-RISC-V-islemci

https://www.youtube.com/watch?v=XEuCqNeZfPQ

KAYNAKLAR

[1] Levent, Vecdi Emre (2021) "Rısc-V Proccessor'', *Bilgisayar Mimarisi-Ders Notları*.

[2] Levent, Vecdi Emre (2021) "Rısc-V CPU Design'', *Bilgisayar Mimarisi-Ders Notları*.