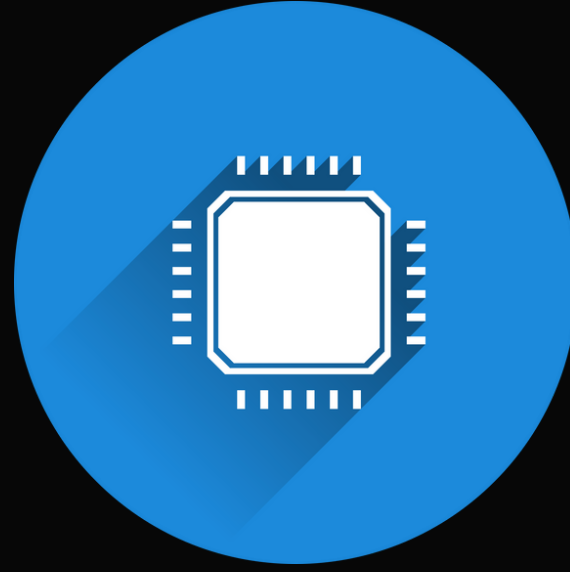


RISC-V Tabanlı İşlemci Tasarımı



Özgür ÇAKMAK ---- Bedirhan İLERİ
200301044 210301501

PROJENİN TANIMI VE AMACI

— Risc-v tabanlı işlemcinin SystemVerilog dili ile RTL tasarımı ve tasarlanan işlemci üzerinde makine dili ile yazılan çeşitli kod parçacıkları yazılacaktır. İşlemcinin her adımı farklı modüllerle yapılacaktır. Proje sonunda basit bir işlemcideki ram, kontrol ünitesi ve saklayıcıların bir arada çalışıp, makine dilindeki kod parçacıklarını nasıl yürütebildiği gözlemlenecektir.

— Risc-v tabanlı bir işlemcinin ALU'sunun destekleyeceği 11 adet işlem vardır. Bu komutlar test yazılımları kullanarak test edilecektir. Bu projenin amacı temel bir işlemcinin çalışma prensibini öğretmektir.

RISC-V Nedir?

(REDUCED INSTRUCTION SET COMPUTER)

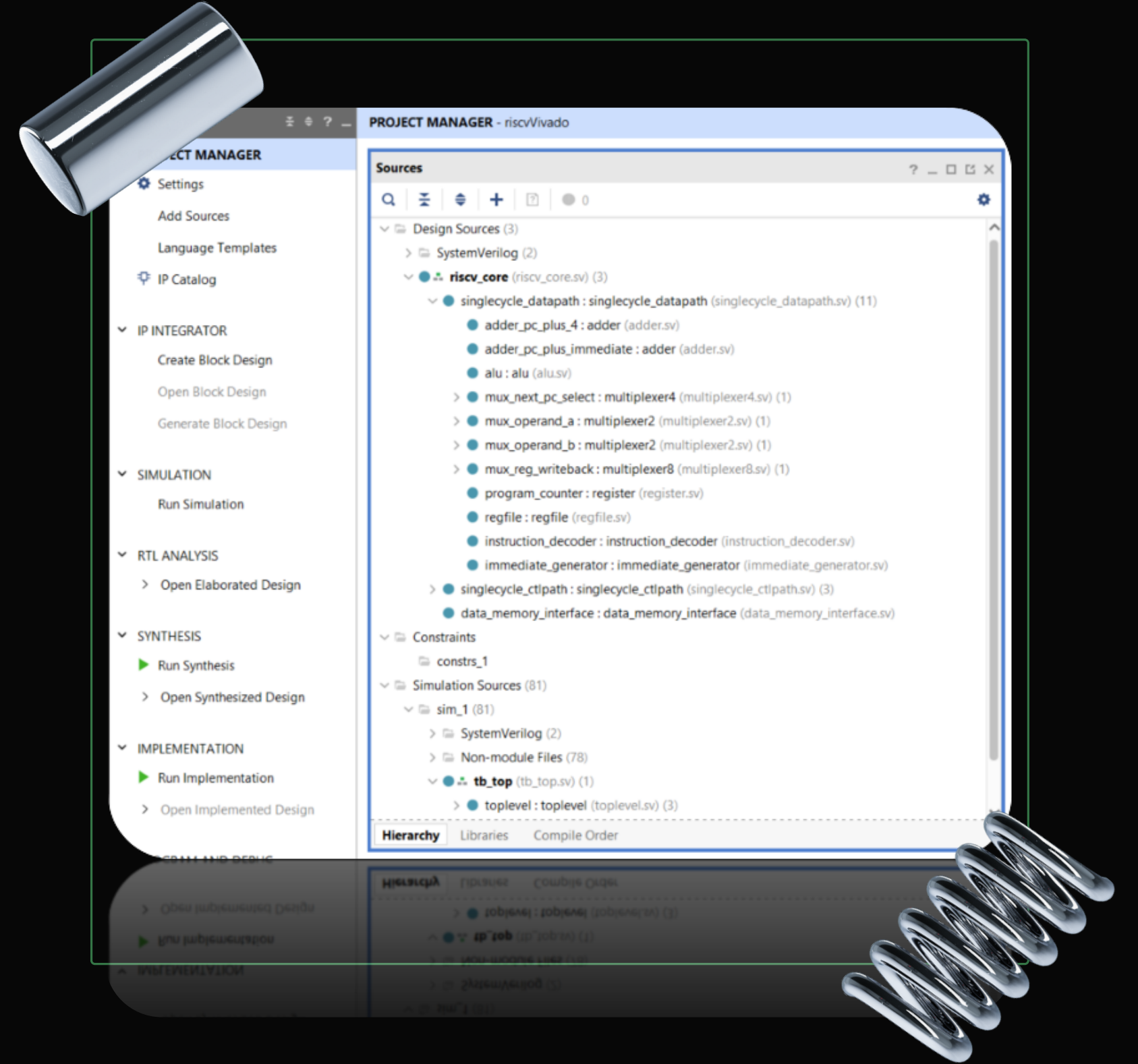
RISC-V, 'İndirgenmiş Komut Seti Bilgisayarı' yani RISC (Reduced Instruction Set Computer) ilkelerine dayanan bir açık standart komut seti mimarisidir. Kaliforniya Üniversitesi tarafından geliştirilen ve herkese açık bir ISA'dır ve herkesin ortak olarak kabul ettiği bir mimaride işlemci üretebilmeyi sağlamaktadır.



SİSTEM MİMARİSİ

Her modülün tasarımı system verilog dili ile gerçekleştirilmiştir.

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

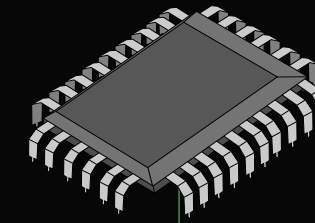


ARİTMETİK LOJİK ÜNİTESİ (ALU)

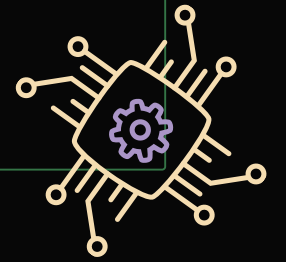
Şekil 1 deki ALU ünitesinin giriş ve çıkış sinyalleri gösterilmiştir.

İşlemcinin ALU'sunun destekleyeceği 11 adet işlem vardır. 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.

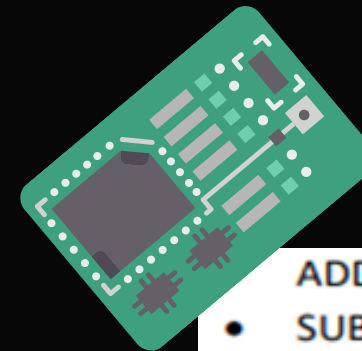
Şekil 3'teki tabloda ALU'nun desteklediği işlemler ve operasyon kodları, şekil 2'de ise operasyonların açıklamaları verilmiştir.



```
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  
);
```



Şekil 1



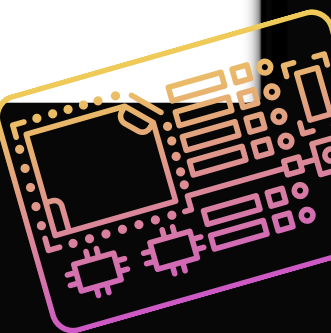
ADD: A + B

- 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

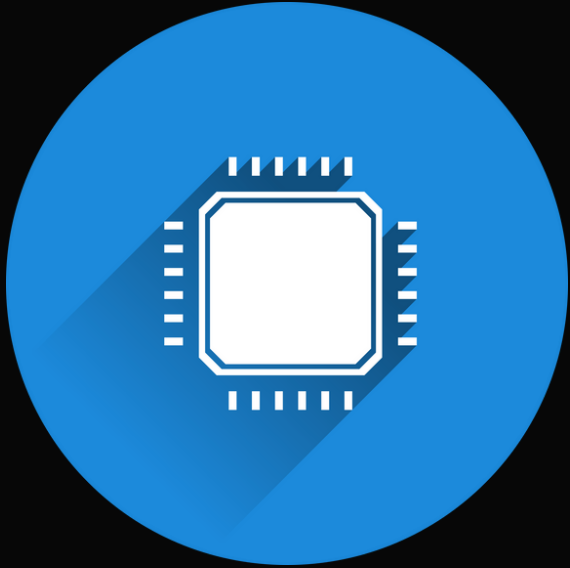
Şekil 2

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

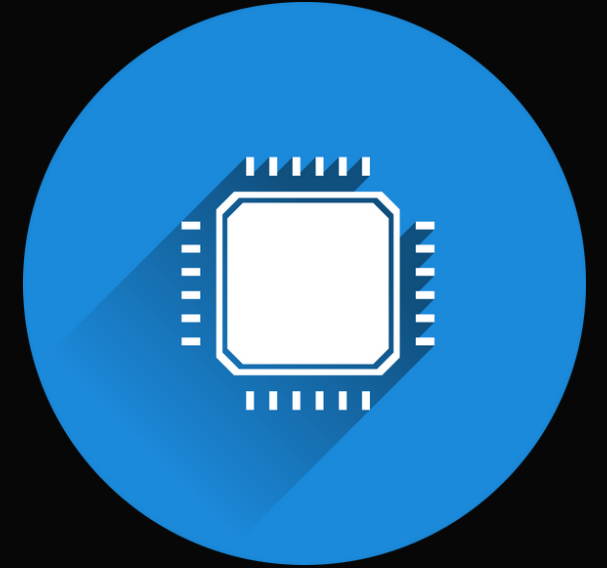
Şekil 3



ARİTMETİK LOJİK ÜNİTESİ (ALU)

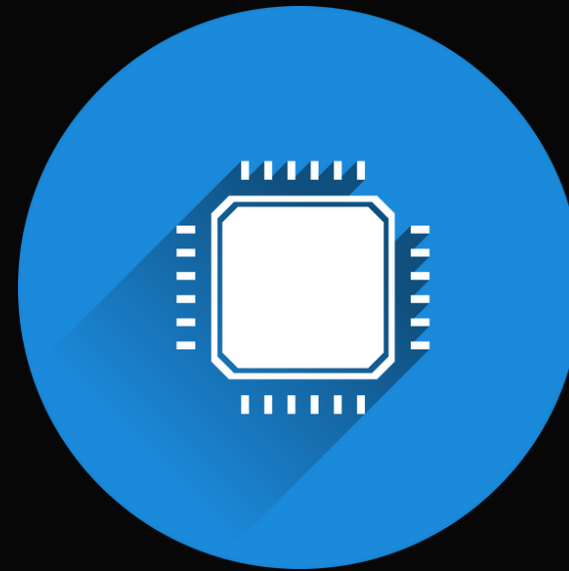


```
logic rez;  
assign result_equal_zero = rez;  
  
always_comb begin  
    case(alu_function)  
        5'b00001:begin //ADD  
            result = operand_a + operand_b;  
            if(result==0)  
                rez = 1;  
            else  
                rez = 0;  
        end  
        5'b00010:begin //SUB  
            result = operand_a - operand_b;  
            if(result == 0)  
                rez = 1;  
            else  
                rez = 0;  
        end  
        5'b00011:begin //SLL  
            result = operand_a << operand_b;  
            if(result == 0)  
                rez = 1;  
            else  
                rez = 0;  
        end  
        5'b00100:begin //SRL  
            result = operand_a >> operand_b;  
            if(result == 0)  
                rez = 1;  
            else  
                rez = 0;  
        end  
    end  
end
```



ARİTMETİK LOJİK ÜNİTESİ (ALU)

```
5'b00101:begin //SRA
    result = operand_a >>> operand_b;
    if(result == 0)
        rez = 1;
    else
        rez = 0;
    end
end
5'b00110:begin //SEQ
    if(operand_a == operand_b) begin
        result = 1;
        rez = 0;
    end else begin
        result = 0;
        rez = 1;
    end
end
5'b00111:begin //SLT
    if(operand_a < operand_b) begin
        result = 1;
        rez = 0;
    end else begin
        result = 0;
        rez = 1;
    end
end
5'b01000:begin //SLTU
    if($unsigned(operand_a) < $unsigned(operand_b)) begin
        result = 1;
        rez = 0;
    end else begin
        result = 0;
        rez = 1;
    end
end
end
```



```
5'b01001:begin //XOR
    result = operand_a ^ operand_b;
    if(result == 0)
        rez = 1;
    else
        rez = 0;
    end
end
5'b01010:begin //OR
    result = operand_a | operand_b;
    if(result == 0)
        rez = 1;
    else
        rez = 0;
    end
end
5'b01011:begin //AND
    result = operand_a & operand_b;
    if(result == 0)
        rez = 1;
    else
        rez = 0;
    end
endcase
end
endmodule
```

INSTRUCTION DECODER

32-bit RISC-V Instruction Formats																																			
Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Register/register	funct7							rs2					rs1					funct3			rd					opcode									
Immediate	imm[11:0]												rs1					funct3			rd					opcode									
Upper Immediate	imm[31:12]																				rd					opcode									
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode									
Branch	[12]	imm[10:5]							rs2					rs1					funct3			imm[4:1]				[11]	opcode								
Jump	[20]	imm[10:1]											[11]	imm[19:12]										rd					opcode						
<ul style="list-style-type: none">• opcode (7 bit): partially specifies which of the 6 types of <i>instruction formats</i>• funct7 + funct3 (10 bit): combined with opcode, these two fields describe what operation to perform• rs1 (5 bit): specifies register containing first operand• rs2 (5 bit): specifies second register operand• rd (5 bit): Destination register specifies register which will receive result of computation																																			

RISC-V işlemcisinde bulunabilecek Instruction formatları yukarıda verilmiştir ve 32 bitlik uzunluğundadır. Proje kapsamında ALU içersinde gerçekleştirilmiş olan operasyon kodları register şeklinde kullanılmıştır.

INSTRUCTION DECODER

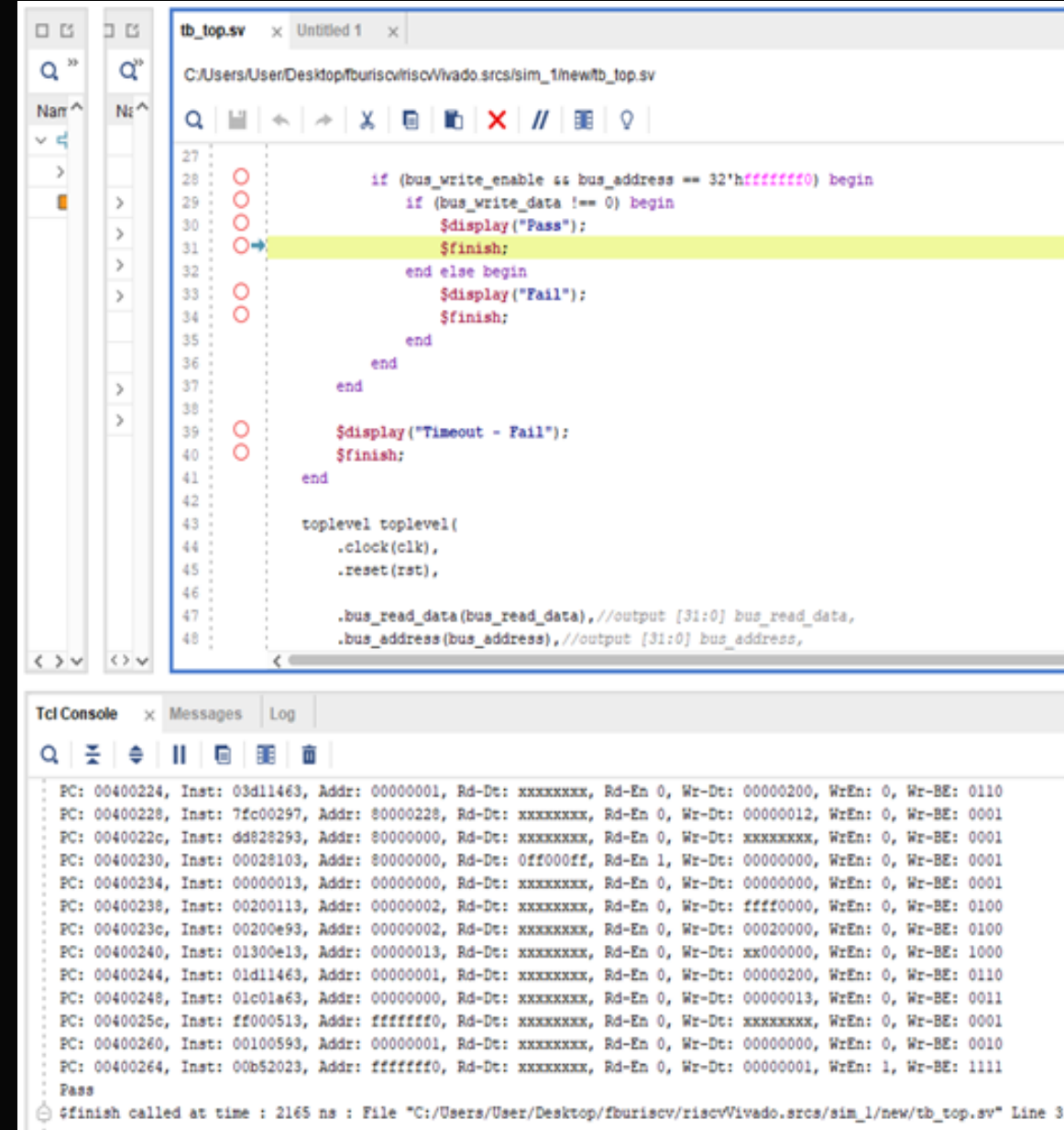
Instruction decoder modülünde giriş olarak 32 bitlik instruction word'u alınmaktadır. Çıkışta ise instruction'un parse edilmiş hali, yani decode edilmiş hali çıkış olarak verilmektedir.

- Opcode, instruction'un ilk 7 bitini yani [6:0];
- 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]; temsil etmektedir.

Buna göre instruction sinyalini parçalayarak ilgili sinyallerin üzerlerine atama yapılmıştır.

```
1  `include "config.sv"
2  `include "constants.sv"
3
4  module instruction_decoder(
5      input  [31:0] inst,
6      output [6:0] inst_opcode,
7      output [2:0] inst_func3,
8      output [6:0] inst_func7,
9      output [4:0] inst_rd,
10     output [4:0] inst_rs1,
11     output [4:0] inst_rs2
12 );
13
14     assign inst_opcode = inst[6:0];
15     assign inst_rd     = inst[11:7];
16     assign inst_func3  = inst[14:12];
17     assign inst_rs1    = inst[19:15];
18     assign inst_rs2    = inst[24:20];
19     assign inst_func7  = inst[31:25];
20
21 endmodule
```

PROJE TESTİ VE SİMÜLASYON



```
tb_top.v  x  Untitled 1  x
C:/Users/User/Desktop/fburiscv/riscv/Vivado/srcs/sim_1/new/tb_top.v

27
28         if (bus_write_enable == bus_address == 32'hffffff0) begin
29             if (bus_write_data != 0) begin
30                 $display("Pass");
31                 $finish;
32             end else begin
33                 $display("Fail");
34                 $finish;
35             end
36         end
37     end
38
39     $display("Timeout - Fail");
40     $finish;
41 end
42
43 toplevel toplevel(
44     .clock(clk),
45     .reset(rst),
46
47     .bus_read_data(bus_read_data), //output [31:0] bus_read_data,
48     .bus_address(bus_address), //output [31:0] bus_address,
```

Tcl Console

```
PC: 00400224, Inst: 03d11463, Addr: 00000001, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000200, Wr-En: 0, Wr-BE: 0110
PC: 00400228, Inst: 7fc00297, Addr: 80000228, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000012, Wr-En: 0, Wr-BE: 0001
PC: 0040022c, Inst: dd828293, Addr: 80000000, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xxxxxxxx, Wr-En: 0, Wr-BE: 0001
PC: 00400230, Inst: 00028103, Addr: 80000000, Rd-Dt: 0ff000ff, Rd-En 1, Wr-Dt: 00000000, Wr-En: 0, Wr-BE: 0001
PC: 00400234, Inst: 00000013, Addr: 00000000, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000000, Wr-En: 0, Wr-BE: 0001
PC: 00400238, Inst: 00200113, Addr: 00000002, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: ffff0000, Wr-En: 0, Wr-BE: 0100
PC: 0040023c, Inst: 00200e93, Addr: 00000002, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00020000, Wr-En: 0, Wr-BE: 0100
PC: 00400240, Inst: 01300e13, Addr: 00000013, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xx000000, Wr-En: 0, Wr-BE: 1000
PC: 00400244, Inst: 01d11463, Addr: 00000001, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000200, Wr-En: 0, Wr-BE: 0110
PC: 00400248, Inst: 01c01a63, Addr: 00000000, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000013, Wr-En: 0, Wr-BE: 0011
PC: 0040025c, Inst: ff000513, Addr: ffffffff, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xxxxxxxx, Wr-En: 0, Wr-BE: 0001
PC: 00400260, Inst: 00100593, Addr: 00000001, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000000, Wr-En: 0, Wr-BE: 0010
PC: 00400264, Inst: 00b52023, Addr: ffffffff, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000001, Wr-En: 1, Wr-BE: 1111
Pass
$finish called at time : 2165 ns : File "C:/Users/User/Desktop/fburiscv/riscv/Vivado/srcs/sim_1/new/tb_top.v" Line 31
```

Yukarıdaki tb_top dosyasında hazır olarak bulunan kodu çalıştırdığımızda pass çıktısını vermektedir. O halde kodumuz doğru çalışmıştır.

KULLANILAN YAZILIM

— Tasarımları yapabilmek için Xilinx tarafından geliştirilen Vivado Design Suite yazılımını kullandık. Vivado Design Suite, HDL tasarımlarının sentezi ve analizi için üretilmiş bir yazılım paketidir ve Xilinx ISE'nin yerine çip geliştirme ve üst düzey sentez sistemi için ek özellikler sunar.

— Vivado, tüm tasarım akışının baştan aşağı yeniden yazılmasını ve yeniden düşünülmesini temsil eder. SystemVerilog dili kullanılarak Vivado üzerinde tasarımlar yapıldı. IEEE 1800 olarak standartlaştırılmış SystemVerilog ise elektronik sistemleri modellemek, tasarlamak, simüle etmek, test etmek ve uygulamak için kullanılan bir donanım açıklaması ve donanım doğrulama dilidir.

SONUÇ

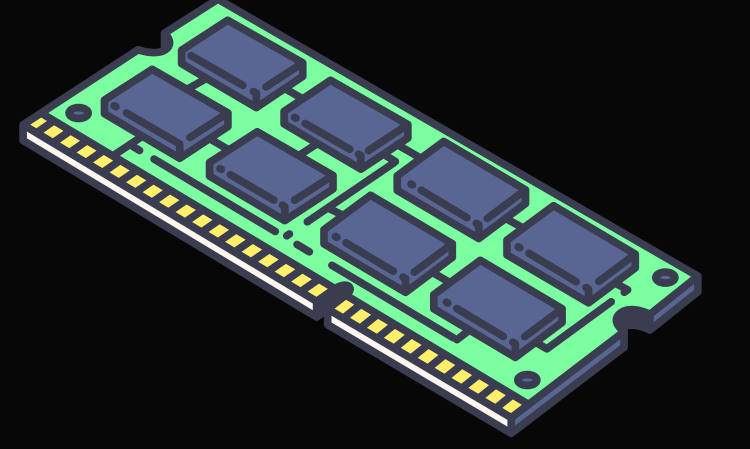
Ve Bulgular

Geliştirilen RISC-V işlemcisi ADD, SUB, SLL, SLR, SRA, SEQ, SLT, SLTU, XOR, OR ve AND işlemlerini destekliyor.

Bu proje ile birlikte RISC-V işlemcisinin tasarımını, nasıl çalıştığını, hangi işlemleri yapabildiğini ve mimarisini öğrenmiş olduk. SystemVerilog dilini iyice kavrayıp kendimizi geliştirdik.

Bu işlemcide tamamlanmış ALU ve Instruction_decoder tasarımını tamamladık.

Bu tasarladığımız işlemciyi Vivado programında test edip doğru çalışıp çalışmadığını kontrol ettik ve tasarımın başarıyla çalıştığını gördük.



Sunumumuzu dinlediğiniz için teşekkürler...

