RISC-V TABANLI İŞLEMCİ TASARIMI

Semanur Özyılmaz, Esra Küçükbaş
Fenerbahçe Üniversitesi
Bilgisayar Mühendisliği
İstanbul, Türkiye
e-mail: {semanur.ozyilmaz, esra.kucukbas}@stu.fbu.edu.tr,

***Özetçe*—Bu projede diğer bölümleri tasarlanmış olan RISC-V işlemcisinin ALU ve Instruction Decoder bloklarının tasarımı ve doğrulanması SystemVerilog dili ile yapılmıştır.**

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

***Abstract*— In this project, the design and verification of the ALU and Instruction Decoder blocks of the RISC-V processor, other parts of which were designed, were done with the SystemVerilog language.**

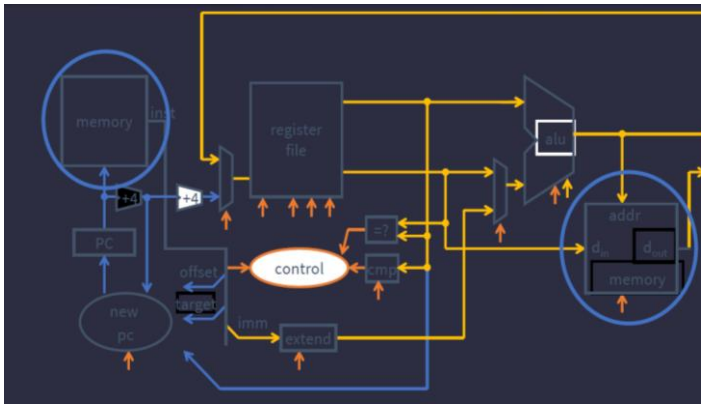
Keywords — FPGA, CPU, RISC-V, SYSTEMVERİLOG

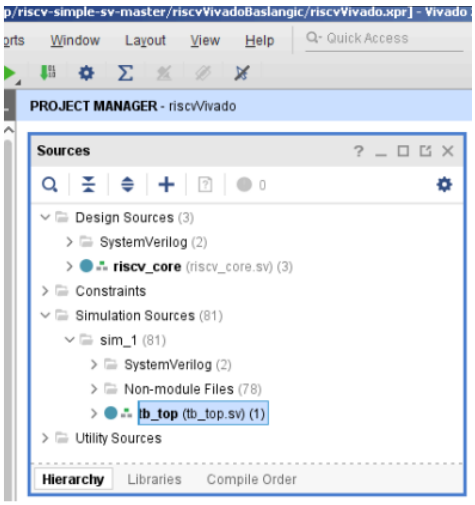
I. GİRİŞ

Bu proje kapsamında başlangıç tasarım verilen bir RISC-V işlemcisinin ALU ve instruction decoder blokları temel SystemVerilog dili özellikleri kullanılarak tasarım ve doğrulama çalışmaları yapılmıştır. Amaç ALU ve instruction decoder bloklarını tasarlayarak işlemcinin verilen başlangıç tasarımı ile senkronize bir şekilde doğru çalışmasını sağlamaktır.

II. SİSTEM MİMARİSİ

Proje SystemVerilog dili ile Xilinx Vivado ortamında tasarlanmıştır. Bu uygulamadaki simülasyon ortamında test edilmiştir. Bu mimaride komutlar ve veriler ayrı belleklerde tutulmaktadır.





Projede Design Sources kısmının altında tepe modülü olarak riscv_core system verilog dosyası, simulation sources kısmının altında ise tepe modülü olarak tb_top system verilog dosyası bulunmaktadır.

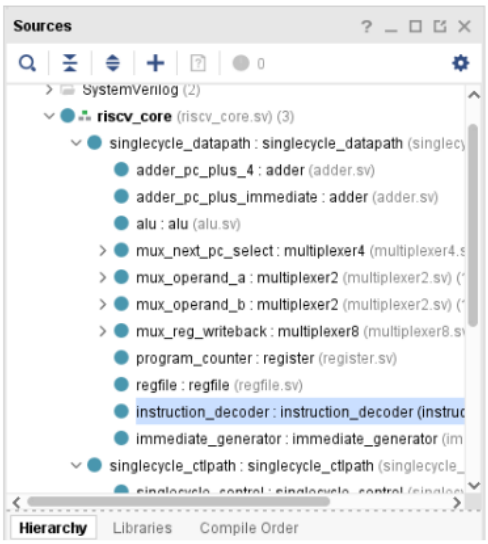
```
module riscv_core (
    input  clock,
    input  reset,

    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 modülünün giriş çıkış sinyalleri yukarıdaki gibidir. clock ve reset sinyalleri gelmektedir. bus_address sinyali RAM'in adres girişini okur. bus_read_data sinyali RAM'den veri okumak için kullanılır. bus_write_data sinyali RAM'e veri yazmak için kullanılır. bus_byte_enable sinyali hangi 8 bitin RAM'e yazılacağını belirtir. bus_read_enable sinyali RAM'den veri okunacağı zaman aktif edilir. bus_write_enable sinyali RAM'e veri yazılacağı zaman aktif edilir. İşlemcinin çalışması için bu sinyaller yeterlidir. inst ve pc sinyalleri işlemcinin o an işlediği komut ile ilgilidir ve testbench için gereklidir.

riscv_core modülü temelde 3 tane modül içermektedir.

singlecycle_datapath modülü içerisinde işlemlerin yapıldığı modüldür. singlecycle_clpath modülü işlemcinin ne yapacağına karar veren modüldür. data_memory_interface modülü ise bellekle ilgili işlemlerin yapılmasını sağlayan modüldür.

Tasarımı yapılacak olan ALU ve Instruction Decoder modülleri riscv_core modülün altındaki singlecycle_datapath modülünün altında bulunmaktadır.




```
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
);
```

ALU'nun giriş çıkış sinyalleri yukarıda belirtildiği gibidir. alu_function girişinden 5 bitlik hangi işlemin yapılacağının bilgisi gelir. Bu tasarımdaki ALU'nun desteklediği 11 adet işlem vardır. operand_a ve operand_b işleme koyulacak olan 32 bitlik sayıların geldiği girişlerdir. İşlem yapıldıktan sonra elde edilen sonuç 32 bit olarak result çıkışından çıkar. result_equal_zero çıkışı ise result 0 ise 1, result 0'dan farklı ise 0 olmaktadır.

ALU_ADD 5'b00001	• ADD: A + B
ALU_SUB 5'b00010	• SUB: A - B
ALU_SLL 5'b00011	• SLL: A << B
ALU_SRL 5'b00100	• SLR: A >> B
ALU_SRA 5'b00101	• SRA: A >>> B
ALU_SEQ 5'b00110	• SEQ: A == B
ALU_SLT 5'b00111	• SLT: A < B
ALU_SLTU 5'b01000	• SLTU: $unsigned(A) < $unsigned(B)
ALU_XOR 5'b01001	• XOR: A ^ B
ALU_OR 5'b01010	• OR: A | B
ALU_AND 5'b01011	• AND: A & B

ALU'nun desteklediği işlemler, işlemlerin operasyon kodları ve işlemlerin açıklamaları yukarıdaki gibidir. Operasyon kodları alu_function girişidir. A ve B de 32 bitlik işleme girecek sayılardır.

```
`include "config.sv"
`include "constants.sv"

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
);

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

logic resultzero;
assign result_equal_zero = resultzero;
always_comb begin
    case(alu_function)
        ALU_ADD: begin
            result = operand_a + operand_b;
            if(result==0) resultzero = 1;
            else resultzero = 0;
        end
        ALU_SUB: begin
            result = operand_a - operand_b;
            if(result==0) resultzero = 1;
            else resultzero = 0;
        end
        ALU_SLL: begin
            result = operand_a << operand_b;
            if(result==0) resultzero = 1;
            else resultzero = 0;
        end
        ALU_SRL: begin
            result = operand_a >> operand_b;
            if(result==0) resultzero = 1;
            else resultzero = 0;
        end
```

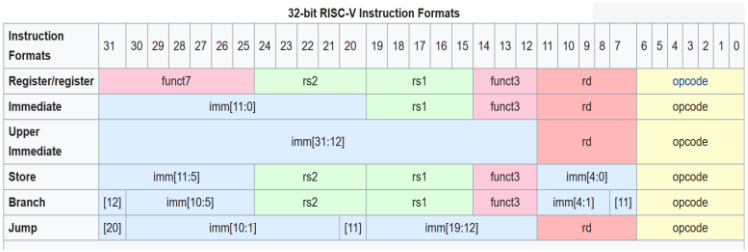
```
          ALU_SRA: begin
               result = operand_a >>> operand_b;
               if(result==0) resultzero = 1;
               else resultzero = 0;
          end
          ALU_SEQ: begin
               result = operand_a == operand_b;
               if(result==0) resultzero = 1;
               else resultzero = 0;
          end
          ALU_SLT: begin
               result = operand_a < operand_b;
               if(result==0) resultzero = 1;
               else resultzero = 0;
          end
          ALU_SLTU: begin
               if($unsigned(operand_a) < $unsigned(operand_b) == 1) beg
                    result = 1;
                    resultzero = 0;
               end else begin
                    result = 0;
                    resultzero = 1;
               end
          end
```

```
          ALU_XOR: begin
               result = operand_a ^ operand_b;
               if(result==0) resultzero = 1;
               else resultzero = 0;
          end
          ALU_OR: begin
               result = operand_a | operand_b;
               if(result==0) resultzero = 1;
               else resultzero = 0;
          end
          ALU_AND: begin
               result = operand_a & operand_b;
               if(result==0) resultzero = 1;
               else resultzero = 0;
          end
     endcase
end
endmodule
```

ALU modülünün tasarımı yukarıdaki gibi yapılmıştır. Gelen alu_function sinyaline göre hangi işlemin yapılacağına karar verilmesi için case bloğu kullanılmıştır. case bloğunda kullanılan işlemlere ait operasyon kodları parametrelere verilerek case bloğunda bu parametreler kullanılmıştır. Bu kodun daha anlaşılır ve okunabilir olmasını sağlar. Hangi işlemin yapılacağına karar verildikten sonra işlem yapılarak sonuç result çıkışına verilmektedir. result değerinin sıfır olup olmamasına göre de resultzero aracılığı ile result_equal_zero çıkışı beslenmektedir. Bu şekilde ALU tasarımı tamamlanmıştır.

```
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
);
```

Instruction Decoder modülünün giriş ve çıkış sinyalleri yukarıdaki gibidir. Modüle giriş olarak 32 bitlik instruction gelmektedir. Modül gelen instruction'ı parçalara ayırarak çözmekte ve çözdüğü parçaları çıkış olarak vermektedir.



Yukarıda RISC-V işlemcisinin 32 bitlik komutlarının formatları görülmektedir. Bu projedeki komutlar register/register türündedir. Bundan dolayı register/register kısmındaki formata göre kod çözümlemesi yapılacaktır.

7 bitlik inst_opcode çıkışından inst girişinin [6:0] arası bitleri çıkarılacaktır.

5 bitlik inst_rd çıkışından inst girişinin [11:7] arası bitleri çıkarılacaktır.

3 bitlik inst_funct3 çıkışından inst girişinin [14:12] arası bitleri çıkarılacaktır.

5 bitlik inst_rs1 çıkışından inst girişinin [19:15] arası bitleri çıkarılacaktır.

5 bitlik inst_rs2 çıkışından inst girişinin [24:20] arası bitleri çıkarılacaktır.

7 bitlik inst_funct7 çıkışından inst girişinin [31:25] arası bitleri çıkarılacaktır.

```
`include "config.sv"
`include "constants.sv"

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
);

assign inst_opcode = inst[6:0];
assign inst_rd = inst[11:7];
assign inst_funct3 = inst[14:12];
assign inst_rs1 = inst[19:15];
assign inst_rs2 = inst[24:20];
assign inst_funct7 = inst[31:25];

endmodule
```

Instruction Decoder tasarımı yukarıdaki gibi yapılmıştır. 32 bitlik inst girişi yukarıda belirtildiği şekilde uygun bitlerinden ayrılarak ilgili çıkış sinyallerine atanmıştır. Böylece 32 bitlik komutlar çözülmüş olur.

III. KULLANILAN YAZILIM

RISC-V işlemcisini test etmek için verilen başlangıç test kodları vardır. Test kodunda makine diline dönüştürülmüş test uygulamasını işlemciye besleyerek sonucu kontrol eden bir

uygulama vardır. Tasarımın simülasyonu başlatılıp play butonuna basıldığında testin tamamlanması için en fazla 10000 cycle beklenmektedir. Bu süre sonunda sonuç hesaplanamamış ise tasarımda bir hata var demektir.

```
            if (bus_write_enable && bus_address == 32'hfffffff0) begin
                if (bus_write_data !== 0) begin
                    $display("Pass");
                    $finish;
                end else begin
                    $display("Fail");
                    $finish;
                end
            end
        end

        $display("Timeout - Fail");
        $finish;
    end
```

Bu süre sonunda if koşulu ile belirtilen adrese gelinmişse işlemci takılmadan çalışmış demektir ancak istenilen sonuç elde edilememişse konsolda 'Fail' çıktısı verilir. İstenilen sonuç elde edilmişse konsolda 'Pass' çıktısı verilir. Bu işlemcinin başarılı bir şekilde çalıştığını gösterir. Eğer belirtilen sürede doğru veya hatalı bir sonuç üretilemediyse işlemci bir noktada takılı kalmış demektir ve konsolda 'Timeout – Fail' çıktısı verilir.

```
PC: 00400260, Inst: 00100593, Addr: 00000001, Rd-Dt:
PC: 00400264, Inst: 00b52023, Addr: fffffff0, Rd-Dt:
Pass
$finish called at time : 2165 ns : File "C:/Users/De
```

Tasarım sonucunda işlemci başarılı bir şekilde çalışmış ve konsolda 'Pass' yazısı görülmüştür.

IV. SONUÇLAR

Tasarlanan RISC-V işlemcisi 11 adet işlemi desteklemektedir. Bunlar toplama, çıkarma, sola bit kaydırma, sağa bit kaydırma, sağa işareti koruyarak bit kaydırma, eşitlik kontrolü, küçüklük kontrolü, unsigned küçüklük kontrolü, xor, or ve and işlemleridir. Bu proje ile bu 11 adet işlemi gerçekleştirebilen RISC-V işlemcisinin ALU ve Instruction Decoder modülleri tasarlanarak başlangıç olarak verilen modüllerle birlikte tam bir işlemci elde etmiş olduk. RISC-V işlemcisinin tasarımı, çalışması, komut seti hakkında bilgi edinmiş ve tasarımını deneyimlemiş olduk.

PROJE EKİBİ

Semanur Özyılmaz: 2019 yılında Küçükçekmece Anadolu Lisesinden mezun oldum. Lisans eğitimime Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde devam etmekteyim. Veri bilimi, makine öğrenmesi ve derin öğrenme alanları ile ilgilenmekteyim. Python, SQL, R, HTML5 ve CSS3 ile çalışmaktayım.

Esra Küçükbaş: 2019 yılında Hatice Kemal Kayalıoğlu Fen Lisesinden mezun oldum. Lisans eğitimime Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde devam ediyorum. C, C++ ve Python dilleri ile ilgilenmekteyim.

REFERANS DOSYALAR

https://youtu.be/THC1t-MJFLs

https://github.com/semanurozyilmaz/RISC-V-TABANLI-ISLEMCI-TASARIMI

KAYNAKLAR

[1]http://www.levent.tc/files/courses/computer_architecture/lectures/lec6/BLM202_hafta6_riscv.pdf

[2]http://www.levent.tc/files/courses/computer_architecture/lectures/lec9/BLM202_hafta9_RISCV_design.pdf

[3]http://www.levent.tc/files/courses/computer_architecture/project/BLM202_proje_spesifikasyonlari.pdf