FENERBAHÇE ÜNİVERSİTESİ

Bilgisayar Mühendisliği

RISC-V Tabanlı İşlemci Tasarımı

Ahmet Hazar Haspolat

Mustafa Berk Taşkın

Cüneyt Balcı

Ömer Sait Yorulmaz

e-mail: {ahmet.haspolat, mustafa.taskin, cuneyt.balci, omer.yorulmaz}@stu.fbu.edu.tr

İçindekiler

GİRİŞ

İçerik

Proje kapsaminda temel hatları önceden oluşturulmuş olan bir RISC-V işlemcisinin Instruction Decoder ve ALU modüllerini SystemVerilog ile gerçekleştirilip, sistemin doğrulama çalışmaları yapılacaktır.

The Instruction Decoder and ALU modules of a RISC-V processor will be created with SystemVerilog features and the verification studies of the system will be observed.

Keywords: RISC-V, RTL, CPU, FPGA, SystemVerilog

Bağlam

Bu dokümanın kapsamında, önceden tanımlanmış olan RISC-V işlemcisinin çalışma mekanizmalarını detaylıca incelemek ve sistem üzerinde tamamlanması gereken kısımları; Instruction Decoder ile ALU bloklarını oluştururarak, sistemin doğruluğunun test edilmesi ve analiz edilme süreci anlatılacaktır.

Süreç akışında temel olarak;

- *RISC-V işlemcisinin detaylarının incelenmesi ve tamamlanmamış kısımların çıkartılması*
- *Tespit edilen kısımların akış şemalarının oluşturulması ve analizi*
- *Bu analizler çerçevesinde işlemci üzerinde yapılacak gerçeklemelerin oluşturulması*
- *Son aşamada gerçeklenen sistemin doğrulunun test edilmesi üzerine çalışmaların yapılması*

adımları açıklanmıştır.

Bahsi geçen yazılımsal sürecin gerçekleştirilmesi için tasarlanmış sistem mimarisi, kullanılan geliştirme araçları ve geliştirilen proje kapsamında elde edilen sonuçlar da doküman içerisinde detaylandırılmıştır.

SİSTEM MİMARİSİ

Sistem ana mimarisininde toplamde üç adet modül bulunmaktadır. Bunlar;

- Single Cycle Data Path
- Single Cycle Control Path
- Data Memory Interface

Single Cyle Data Path içerisinde ALU ve Instruction Decoder bloklarıyla birlikte multiplexerlar, program counter, regFile gibi bloklar bulunmaktadir. Sistemin çalışabilmesi ve tam performanslı hale gelebilmesi adına bu modül içerisinde olan ALU ve Instruction Decoder blokları çalışır hale getirilecektir. Aşağıdaki görsel ile birlikte system mimarisinin detayları görülebilir.

Tanımlar

RISC-V: RISC(Reduced Instruction Set Computer) prensiplerini kullanan açık kaynak bir Komut Seti Mimarisidir(ISA). İşlemcilerini CISC mantığı ile yapan firmalara örnek olarak AMD ve Intel'i verebiliriz. İşlemcilerini RISC mantığı ile yapan firmalara örnek ise ARM verilebilir.

ALU(Arithmetic Logic Unit): Birimi aritmetik ve mantık işlemlerini gerçekleştiren bir dijital devredir. AMB en basit işlemi gerçekleştiren mikro denetleyiciden, en karmaşık mikroişlemciye sahip bir bilgisayara kadar tüm işlemcilerin yapıtaşıdır.

Instruction Decoder: İşlemcinin yapması gereken kodların icrası için gerekli işlemleri başlatır ve komutun çalıştırılması için gerekli işlemleri belirler.

ALU TASARIMI

Aşağıda 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
);
```

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.

Aşağıdaki tabloda ALU'nun desteklediği işlemler ve operasyon kodları 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.sv dosyasi içerisinde *gerçekleşmiş* olduğumuz ALU komut setleriyle birlikte,

- alu_function değeri okunur. Bu değer operand a ve operand b değerlerini hangi mantık işlemi ya da aritmetik işlem içerisine dahil edeceğimizi gösterir.
- Bu değer belirlendikten sonra işleme tabii tutulacak operandlar ilgili işlem içerisinde konumlandırılır.
- İşlem sonucunda ortaya çıkan değer üzerine result değeri belirlenir
- Result değeri belirlenirken işlem sonucu baz alınır. Gerçeklenen işlem sonucu sıfır ise result değeri bir, tam tersi durumunda yani işlem sonucu sıfırdan farklı ise result değeri sıfır olarak atanır.

```
always_comb begin

     if(alu_function==5'b00001) begin
        result=operand_a + operand_b;

     end else if(alu_function==5'b00010) begin
        result=operand_a - operand_b;

     end else if(alu_function==5'b00011) begin
        result=operand_a << operand_b;

     end else if(alu_function==5'b00100) begin
        result=operand_a >> operand_b;

     end else if(alu_function==5'b00101) begin
        result=operand_a >>> operand_b;

     end else if(alu_function==5'b00110) begin
     if(operand_a==operand_b) begin
        result=31'b1;
        end else begin
        result = 31'b0;
        end

     end else if(alu_function==5'b00111) begin
     if(operand_a < operand_b) begin
        result=31'b1;
        end else begin
        result = 31'b0;
        end

     end else if(alu_function==5'b01000) begin
     if($unsigned(operand_a) < $unsigned(operand_b)) begin
        result=31'b1;
        end else begin
        result = 31'b0;
        end

     end else if(alu_function==5'b01001) begin
        result=operand_a ^ operand_b;

     end else if(alu_function==5'b01010) begin
        result=operand_a | operand_b;

    end else if(alu_function==5'b01011) begin
        result=operand_a & operand_b;
     end
 end
assign result_equal_zero = (result == 32'b0);

endmodule
```

INSTURCTION DECODER TASARIMI

Aşağıda Instruction Decoder ünitesinin giriş ve çıkış sinyalleri 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
);
```

Bu modülde giriş olarak 32 bitlik instruction word'u alınmaktadır. Çıkışte 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];

Instruction_decoder.sv dosyasi içerisinde *gerçekleşmiş* olduğumuz Instruction Decoder'ın komut setleriyle birlikte,

- inst inputundan gelen 32 bitlik komut seti ilgili formatlara ayrılır. Aşağıdaki görselde 32 bitlik Instruction Formats detayları görülmektedir.
-

```
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_funct3,
8      output [6:0] inst_funct7,
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_funct3 = inst[14:12];
16     assign inst_funct7 = inst[31:25];
17     assign inst_rd     = inst[11:7];
18     assign inst_rs1    = inst[19:15];
19     assign inst_rs2    = inst[24:20];
20
21 endmodule
```

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						

- **opcode (7 bit):** partially specifies which of the 6 types of *instruction formats*
- **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

- İlgili formatların ayrım detayları;
    - Inst_opcode = inst[6:0]
    - Inst_funct3 = inst[14:12]
    - Inst_funct7 = inst[31:25]
    - Inst_rd = inst[11:7]
    - Inst_rs1 = inst[19:15]
    - Inst_ rs2 = inst[2420]

Tanımlar;

- Opcode **(7 bits)**: 6 tip talimat formatını belirtir.
- Funct3 + Funct7 **(10 bits):** Bu iki alan, gerçekleştirilecek işlemi belirtir.
- Rs1 **(5 bits)**: İşleme girecek ilk operandı (operand_a) belirtir.
- Rs2 **(5 bits)**: İşleme girecek ikinci operandı (operand_b) belirtir.
- Rd **(5 bits)**: Hesaplama sonucunun yönlendirileceği hedefi belirtir.

KULLANILAN YAZILIM

Projenin işleyişi aşamasında gerçeklenen ALU ve Instruction Decoder blokları; işlemci içerisinde yapılması planan aritmetik ve mantık işlemlerinin hesaplanabilmesi ve bu işlemlerin anlamlandırılabilmesi için emirlerin formatlanmasıyla birlikte sistemin kullanılabilmesi için Xilinx tarafından geliştirilen Vivado Design Suite yazılımı kullanılmıştır. 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. Biz de SystemVerilog dilini kullanarak Vivado üzerinde tasarımımızı yaptık. 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.

SONUÇLAR

Bu projenin tasarımı ve geliştirilmesi aşamasında öncelikle Vivado Design Suit ile çalışma gereksinimi keşfedildi ve sonrasında bu gereksinim üzerinden araştırma, deneme-yanılma, uygulama konusunda ilerleme kaydedildi. İkincil olarak projenin temel konusu olan ‘RISC-V Tabanlı İşlemci Tasarımı ’nın yazılımsal ve donanımsal temelleri, işlemci içerisinde yer alan blokların çalışma presipleri ve aralarındaki iletişimlerin doğru bir şekilde kurulmaları hususunda kazanımlar sağlandı. Bu bilgiler ışığında son aşamada yapılmış olan testlerinde başarılı bir şekilde sonuçlanmasıyla birlikte, işlemci/donanım alanlarına dair vizyon kazanımı gerçekleştirildi.

PROJE EKİBİ

AHMET HAZAR HASPOLAT – Çamlıca Doğa Koleji’nde lise eğitimimi tamamlamış olup, lisans eğitimime Fenerbahçe Üniversitesi - Bilgisayar Mühendisliği bölümünde devam etmekteyim. Akademik eğitimime görüntü işleme ve yapay zeka alanlarında devam etmeyi hedefliyorum.

MUSTAFA BERK TAŞKIN – Bursa Doğa Koleji Anadolu Lisesi mezuniyetimin ardından, lisans eğitimime Fenerbahçe Üniversitesi - Bilgisayar Mühendisliği bölümünde devam etmekteyim. Python ve Javascipt dillerinde kendimi geliştirmeye devam ediyorum.

CÜNEYT BALCI – 28.08.2000 yılında İstanbul'da doğdum. 2018 yılında Final Temel Lisesi'nden mezun oldum. Şu anda Fenerbahçe Üniversitesi - Bilgisayar Mühendisliği bölümünde ve çift ana dal programı kapsamında Ekonomi (İngilicze) bölümünde lisans eğitimi almaktayım.

ÖMER SAİT YORULMAZ – Fenerbahçe Üniversitesi - Bilgisayar Mühendisliği öğrencisiyim. Akademik eğitimimle eşzamanlı olarak, 2018 yılından beri özel bir şirkette, Software Developer pozisyonunda görev almaktayım.

REFERANS DOSYALAR

https://www.youtube.com/watch?v=6Ug5TEkdGaw

https://github.com/cuneytbalci/riscv_design_project

KAYNAKLAR

- http://www.levent.tc/courses/computer-architecture
- Thomas, Donald, Moorby, Phillip "The Verilog Hardware Description Language" Kluwer Academic Publishers, Norwell, MA. ISBN 0-7923-8166-1
- Janick Bergerdon, "Writing Testbenches: Functional Verification of HDL Models", 2000, ISBN 0-7923-7766-4. (The HDL Testbench Bible)