FENERBAHÇE ÜNİVERSİTESİ

RISC-V Tabanlı İşlemci Tasarımı

Evrim Arda Kalafat, Doğa Turan

Fenerbahçe Üniversitesi

Bilgisayar Mühendisliği

İstanbul, Türkiye

e-mail: evrim.kalafat@stu.fbu.edu.tr, doga.turan@stu.fbu.edu.tr

Proje özeti: 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.

Anahtar Kelimeler: FPGA, CPU, RISC-V, SystemVerilog, RTL.

Abstract: Within the scope of this project, we are going to design ALU and Instruction decoder blocks of a RISC-V processor by using the basic SystemVerilog language features.

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

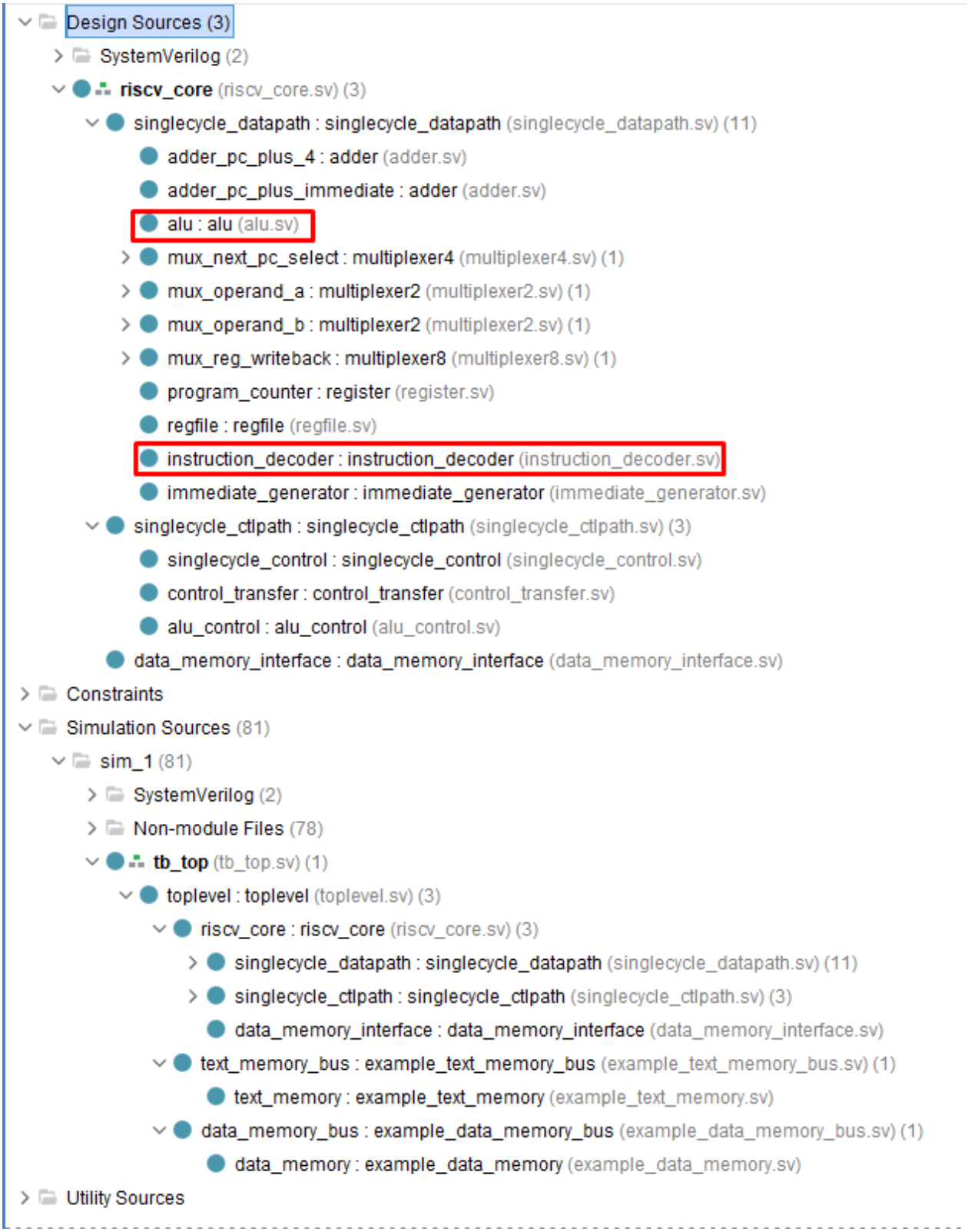
I. Giriş

Projede bize verilen bir RISC-V işlemcisinin ALU ve Instruction decoder bloklarını SystemVerilog dilini kullanarak tasarlamak, bize verilen test kodları ile tamamlanan işlemcimizin doğruluğunu test etmek, temel SystemVerilog dili özelliklerinde kendimizi geliştirmek, RISC-V işlemcisinin yapısını daha iyi tanımak amaçlanmıştır.

Bu proje ödevi Evrim Arda Kalafat (evrim.kalafat@stu.fbu.edu.tr) ve Doğa Turan'a (doga.turan@stu.fbu.edu.tr) aittir.

II. Sistem Mimarisi

Aşağıdaki şekilde sistemin ana mimarisini görebiliyoruz. Biz bu mimarinin içinde boş bırakılmış “alu.sv” ve “instruction_decoder.sv” dosyalarını tasarlıyoruz.



RISC-V: RISC(Reduced Instruction Set Computer) prensiplerini kullanan açık kaynak bir **Komut Seti Mimarisidir(ISA)**. University of California Berkeley'in oluşturduğu ve herkese açık (ister kişisel, ister ticari, ister akademik) bir ISA olan RISC-V, herhangi bir lisans parası ödemeden herkesin ortak kabul ettiği bir mimaride işlemci üretebilmenizi sağlıyor.

Memory: Saklama alanı. RISC-V işlemcisinde komutları ve verileri tutan 2 tane bellek bulunur.

PC(Program Counter): Hangi adresteki komutun çalıştığını ifade eder.

Register File: Saklayıcıların bulunduğu bir dizidir. RISC-V işlemcisinde her biri 32 bitlik 32 adet saklayıcı bulunur.

ALU(Arithmetic Logic Unit): aritmetik ve mantık işlemlerini gerçekleştiren bir dijital devredir.

ALU Tasarımı:

Aşağıdaki şekilde bize verilen kod parçasını görüyoruz. ALU; alu_function, operand_a ve operand_b olmak üzere 3 adet giriş alıyor. Result ve result_equal_zero olmak üzere 2 adet çıkış veriyor.

```
1  `include "config.sv"
2  `include "constants.sv"
3
4  module alu (
5      input        [4:0]  alu_function,
6      input signed [31:0] operand_a,
7      input signed [31:0] operand_b,
8
9      output logic [31:0] result,
10     output              result_equal_zero
11 );
12
```

Aşağıda bu alu'nun desteklediği işlemler ve operasyon kodları verilmiştir.

```
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
```

ALU'nun işlemlerinin detaylı gösterimi;

- 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

Aşağıda bizim tasarladığımız kod parçasını görüyoruz. ALU bir kombinasyonel devre olduğu için SystemVerilog dilindeki always_comb'u kullandık. Case yapısının içinde tüm operasyon kodlarına karşılık gelen işlemleri yaptırdık ve default olarak result'u 0'a eşitledik. Case yapısından sonra result_equal_zero değerini result'a göre assign ettik.

```
13  always_comb begin
14      case(alu_function)
15          5'b00001: result = operand_a + operand_b; // ADD
16
17          5'b00010: result = operand_a - operand_b; // SUB
18
19          5'b00011: result = operand_a << operand_b[4:0];//SLL
20
21          5'b00100: result = operand_a >> operand_b[4:0];//SRL
22
23          5'b00101: result = operand_a >>> operand_b[4:0];//SRA
24
25          5'b00110: begin if (operand_a==operand_b) result = 31'b1; //SEQ
26          else result = 31'b0;
27          end
28
29          5'b00111: begin if (operand_a<operand_b) result = 31'b1; //SLT
30          else result = 31'b0;
31          end
32
33          5'b01000: begin if ($unsigned(operand_a) < $unsigned(operand_b)) result = 31'b1; //SLTU
34          else result = 31'b0;
35          end
36
37          5'b01001: result = operand_a ^ operand_b;//XOR
38
39          5'b01010: result = operand_a | operand_b; // OR
40
41          5'b01011: result = operand_a & operand_b; // AND
42
43          default:result = 31'b0; // default
44      endcase
45  end
46
47  assign result_equal_zero = (result == 32'b0);
48
49  endmodule
```

Instruction_decoder Tasarımı:

RISC-V Instructionları 32 bit uzunluğundadır. Instruction_decoder, instruction'dan gelen 32 bitlik veriyi parçalayarak çıktı verir. Aşağıda bu 32 bitlik Instruction formatlarını görüyoruz.

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

Aşağıda şekil[1]'de bize verilen kod parçasını, şekil[2]'de ise bizim tasarladığımız kısmı görüyoruz.

Şekil[1]'de instruction_decoder, inst isminde 32 bitlik bir girdi almakta ve bu girdiyi, inst_opcode, inst_funct3, inst_funct7, inst_rd, inst_rs1 ve inst_rs2 olmak üzere 6 parçaya bölmekte.

Şekil[2]'de ise inst girdisinden gelen 32 biti 6 parçaya ayırıyoruz. Bunun için SystemVerilog dilindeki assign'ı kullandık.

```
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 );
```

Şekil[1]

```
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
22  endmodule
```

Şekil[2]

III. Kullanılan Yazılım

Tasarımlarımızı yapmak 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. 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.

Alu ve Instruction_decoder tasarımlarımızı bitirdikten sonra Simulation sekmesinin altından Run Simulation > Run Behavioral'a tıklayarak simülasyonumuzu çalıştırdık. Sonra yukarıdaki Run All(F3) tuşuna basarak testlerimizi gerçekleştirdik.

```
19      initial begin
20          #100;
21          rst = 0;
22
23          repeat (100000) begin
24              @(posedge clk);
25
26              $display("PC: %h, Inst: %h, Addr: %h, Rd-Dt: %h, Rd-En %d, Wr-Dt: %h, WrEn: %d, Wr-BE: %b",
27
28              if (bus_write_enable && bus_address == 32'hfffffff0) 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
```

```
Tcl Console  ×  Messages  Log

PC: 00400238, Inst: 00200113, Addr: 00000002, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: ffff0000, WrEn: 0, Wr-BE: 0100
PC: 0040023c, Inst: 00200e93, Addr: 00000002, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00020000, WrEn: 0, Wr-BE: 0100
PC: 00400240, Inst: 01300e13, Addr: 00000013, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xx000000, WrEn: 0, Wr-BE: 1000
PC: 00400244, Inst: 01d11463, Addr: 00000001, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000200, WrEn: 0, Wr-BE: 0110
PC: 00400248, Inst: 01c01a63, Addr: 00000000, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000013, WrEn: 0, Wr-BE: 0011
PC: 0040025c, Inst: ff000513, Addr: fffffff0, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xxxxxxxx, WrEn: 0, Wr-BE: 0001
PC: 00400260, Inst: 00100593, Addr: 00000001, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000000, WrEn: 0, Wr-BE: 0010
PC: 00400264, Inst: 00b52023, Addr: fffffff0, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000001, WrEn: 1, Wr-BE: 1111
Pass
$finish called at time : 2165 ns : File "C:/Users/Arda/Desktop/Bilgisayar Mimarisi/Proje/Proje_Blm/riscvVivado
```

Yukarıda da gözüktüğü gibi "Pass" çıktısını aldık. Tasarımımız testleri başarıyla geçti.

IV. Sonuçlar

Geliştirilen işlemci ADD, SUB, SLL, SLR, SRA, SEQ, SLT, SLTU, XOR, OR ve AND işlemlerini yapabilmekte. Bu projeyle birlikte basit bir şekilde bir RISC-V işlemcinin nasıl çalıştığını ve mimarisini öğrenmiş olduk. SystemVerilog dilinde kendimizi geliştirdik. Bir işlemcide Alu ve Instruction_decorder tasarımını gerçekleştirdik. Bu işlemciyi test kodları ile sınadık ve doğru çalışıp çalışmadığını gözlemledik. İşlemcimiz testleri geçti ve başarıyla çalıştı.

V. Proje Ekibi

Evrim Arda KALAFAT, 25.09.2001 yılında istanbulda doğdu. 2019 yılında Kadıköy Final Temel Lisesi'nden mezun oldu. Şu anda Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde lisans eğitimi almakta. Bilgisayar Mühendisliği Bölüm Temsilciği yapmaktadır. C, C++ ve Pyhton dillerinde bilgili. Programlama, yapay zeka ve siber güvenlik ile ilgileniyor.

Doğa TURAN, 18.09.2001 yılında Gölcük'te dünyaya geldi. 2019 yılında Şehit Özcan Kan Fen Lisesi'nden mezun oldu. Fenerbahçe Üniversitesi'nde Endüstri Mühendisliği anadal ve Bilgisayar Mühendisliği ikinci ana dal olmak üzere lisans eğitimi almaktadır. C ve Pyhton dilleri hakkında donanımlıdır.

VI. Referans Dosyalar

https://youtu.be/S0RCmj7MAG0

https://github.com/rhgod/RISC-V_Project

VII. Kaynaklar

[1] Asanović, K., & Patterson, D. A. (2014). Instruction sets should be free: The case for risc-v. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146.

[2] Waterman, A. S. (2016). Design of the RISC-V instruction set architecture (Doctoral dissertation, UC Berkeley).

[3] Traber, A., Zaruba, F., Stucki, S., Pullini, A., Haugou, G., Flamand, E., ... & Benini, L. (2016, January). PULPino: A small single-core RISC-V SoC. In 3rd RISCV Workshop.

[4] Sutherland, S., Davidmann, S., & Flake, P. (2006). SystemVerilog for Design Second Edition: A Guide to Using SystemVerilog for Hardware Design and Modeling. Springer Science & Business Media.