

RISC-V Tabanlı İşlemci Tasarımı

EVİRİM ARDA KALAFAT
DOĞA TURAN

Projenin tanımı ve amacı

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.

RISC-V nedir?

RISC, Reduced Instruction Set Computer, yani İndirgenmiş Komut Seti Bilgisayarı bir işlemci mimari çeşididir. RISC-V ise RISC prensiplerini kullanan açık kaynak bir Komut Seti Mimarisi'dir (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 ödemediği herkesin ortak kabul ettiği bir mimaride işlemci üretebilmenizi sağlıyor.

Kullandığımız 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 tasarımıımızı 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.

Sistem mimarisi

```
▼ riscv_core (riscv_core.sv) (3)
  ▼ singlecycle_datapath : singlecycle_datapath (singlecycle_datapath.sv) (11)
    ● adder_pc_plus_4 : adder (adder.sv)
    ● adder_pc_plus_immediate : adder (adder.sv)
    ● alu : alu (alu.sv)
  ▼ mux_next_pc_select : multiplexer4 (multiplexer4.sv) (1)
    ● multiplexer : multiplexer (multiplexer.sv)
  ▼ mux_operand_a : multiplexer2 (multiplexer2.sv) (1)
    ● multiplexer : multiplexer (multiplexer.sv)
  ▼ mux_operand_b : multiplexer2 (multiplexer2.sv) (1)
    ● multiplexer : multiplexer (multiplexer.sv)
  ▼ mux_reg_writeback : multiplexer8 (multiplexer8.sv) (1)
    ● multiplexer : multiplexer (multiplexer.sv)
    ● program_counter : register (register.sv)
    ● regfile : regfile (regfile.sv)
    ● instruction_decoder : instruction_decoder (instruction_decoder.sv)
    ● immediate_generator : immediate_generator (immediate_generator.sv)
  ▼ singlecycle_ctlpath : singlecycle_ctlpath (singlecycle_ctlpath.sv) (3)
    ● singlecycle_control : singlecycle_control (singlecycle_control.sv)
    ● control_transfer : control_transfer (control_transfer.sv)
    ● alu_control : alu_control (alu_control.sv)
  ● data_memory_interface : data_memory_interface (data_memory_interface.sv)
```

```
▼ tb_top (tb_top.sv) (1)
  ▼ toplevel : toplevel (toplevel.sv) (3)
    ▼ riscv_core : riscv_core (riscv_core.sv) (3)
      > singlecycle_datapath : singlecycle_datapath (singlecycle_datapath.sv) (11)
      > singlecycle_ctlpath : singlecycle_ctlpath (singlecycle_ctlpath.sv) (3)
        ● data_memory_interface : data_memory_interface (data_memory_interface.sv)
    ▼ text_memory_bus : example_text_memory_bus (example_text_memory_bus.sv) (1)
      ● text_memory : example_text_memory (example_text_memory.sv)
    ▼ data_memory_bus : example_data_memory_bus (example_data_memory_bus.sv) (1)
      ● data_memory : example_data_memory (example_data_memory.sv)
```

Alu tasarımı

ALU bir kombinasyonel devre olduğu için SystemVerilog dilindeki `always_comb`'u kullandık. Case yapısının içinde tüm operasyon kodlarına karşılık gelen işlemleri yaptırarak 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 | | | | | | |
| <ul style="list-style-type: none">• 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 | );
```

```
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
```

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.

```
initial begin
    #100;
    rst = 0;

    repeat (100000) begin
        @(posedge clk);

        $display("PC: %h, Inst: %h, Addr: %h, Rd-Dt: %h, Rd-En %d, Wr-Dt: %h, WrEn: %d, Wr-BE: %b", pc, inst, bus_address, bus_read_data, bus_read_enable, bus_write_data, bus_write_enable, bus_byte_enable);

        if (bus_write_enable && bus_address == 32'hffffff0) begin
            if (bus_write_data != 0) begin
                $display("Pass");
                $finish;
            end else begin
                $display("Fail");
                $finish;
            end
        end
    end
end

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


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_decoder tasarımını gerçekleştirdik. Bu işlemciyi test kodları ile sınadık ve doğru çalışıp çalışmadığını gözlemledik. İşlemcimiz testleri geçti ve başarıyla çalıştı.

Teşekkürler

