GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x8 Deney Raporu

Tek Ritimli İşlemci Tasarımı

Hazırlayanlar
1) 171024009 – Ömer Genç
2) 1710240081 - Abdullah Halil Özdal

1. Giriş

Bu deney kapsamında aşağıda ki beceriler test edilmiştir.

- Basitleştirilmiş bir komut setine göre tek ritimli işlemci tasarlamak. GTE20 (Gebze Teknik Elektronik 2020) komut kümesi, 22 komut.
- İşlemci içerisindeki farklı parçaların birbiri ile nasıl bağlandığını öğrenmek.
- Tasarlanan işlemciyi gerçekleyip, belirlenen komut setine göre derlenmiş bir programı çalıştırmak.
- İşlemcinin performansını ölçmek ve doğrulamasını yapmak.

2. Problemler

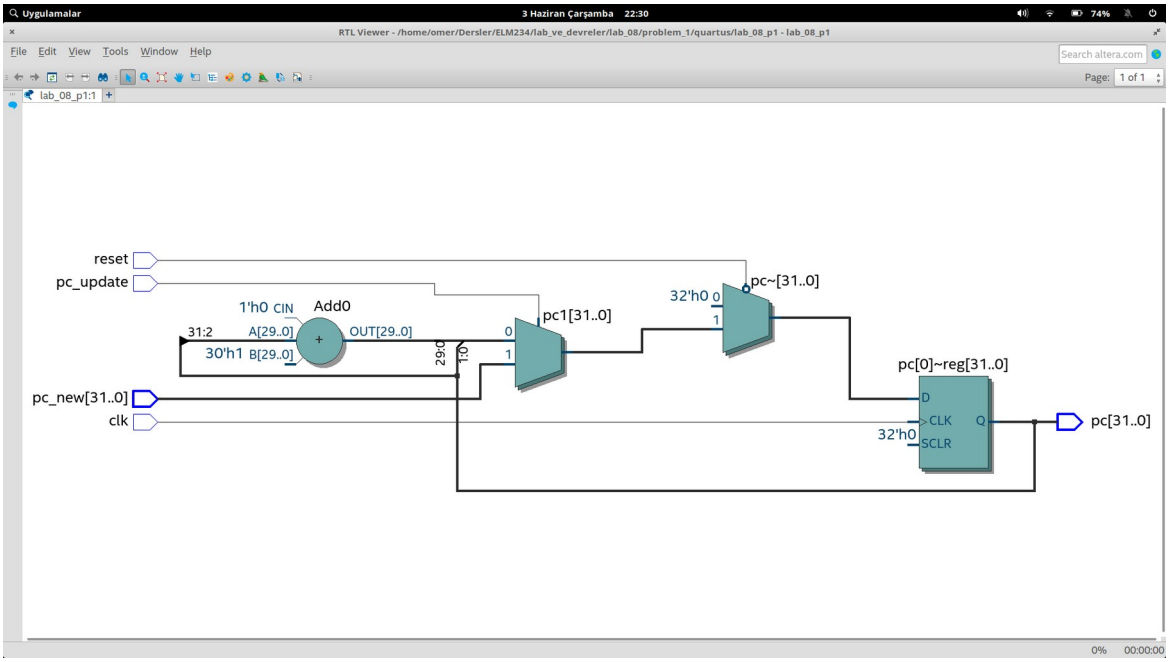
2.1. Problem 1 - Instruction Fetch ünitesi

2.1.1. Teorik Araştırma

Bu problem için teorik bir araştırma bulunmamaktadır.

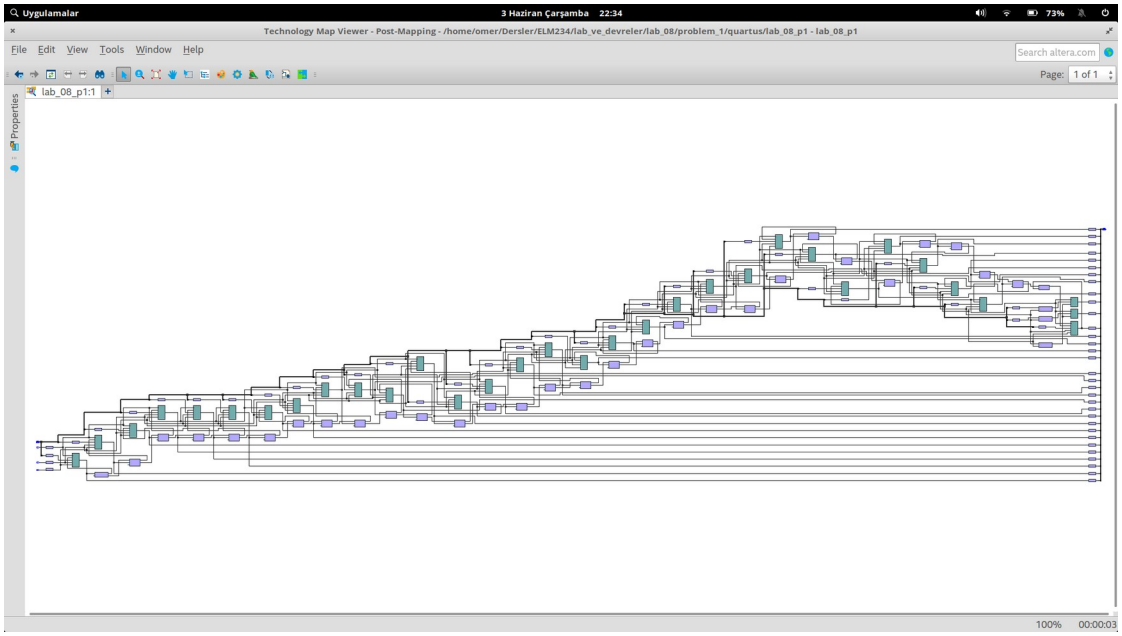
2.1.2. Deneyin Yapılışı

a) RTL Devre Şeması



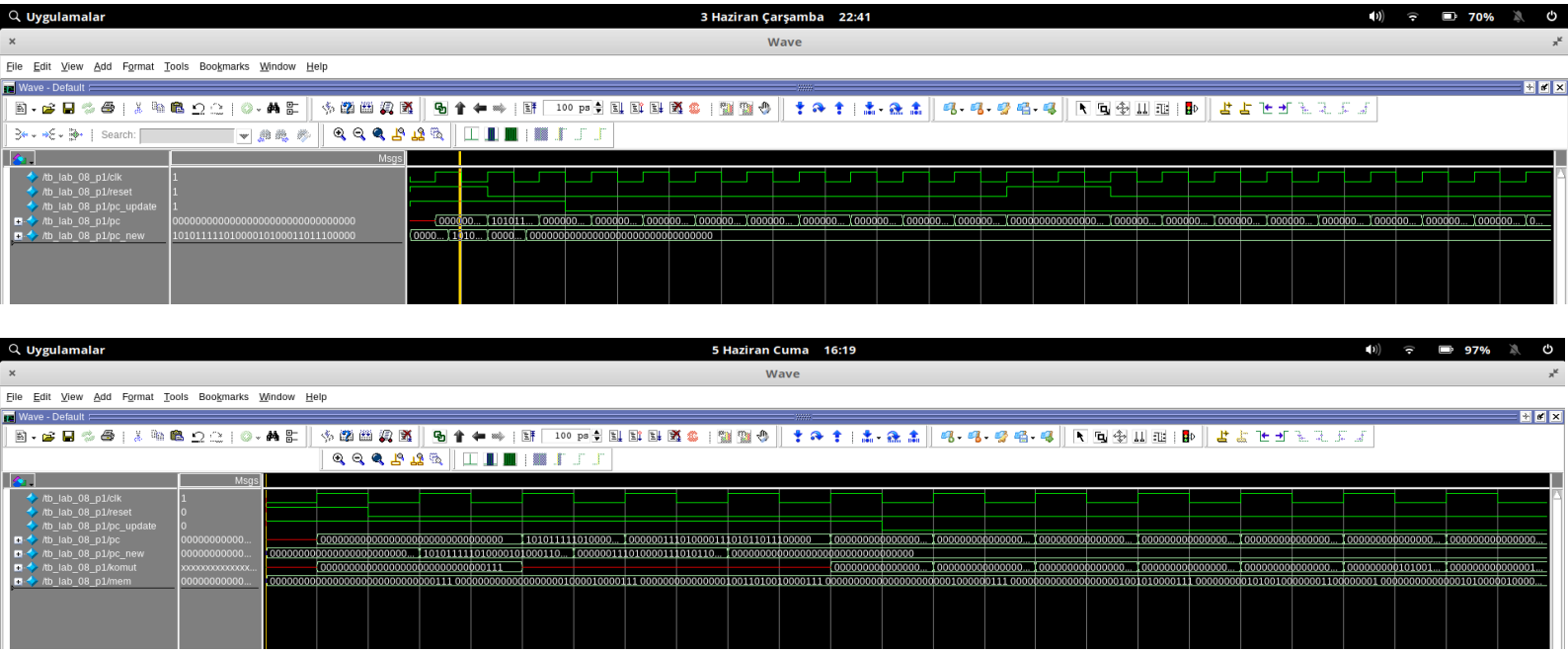
Şekil 1. Lojik devrenin RTL devre şeması.

b) Eşleştirme Ardı Teknoloji Şeması



Şekil 2. Lojik devrenin eşleştirme ardı teknoloji şeması

c) Simulasyon



d) Yorum ve açıklama

Burada bir tür PC(15) yapıldı her clk ta pc değeri 4 artıyor böylece her clk ta bir sonraki komut okunuyor pc_update sinyali geldiğinde PC 4 artmak yerine pc_new değerini okuyor. Simülasyonda başarılı olmuştur. 2. simülasyonda ise pc değeri hafıza adresi olarak alınmış ve bu değerlere göre hafızadan veri çekilmiştir.

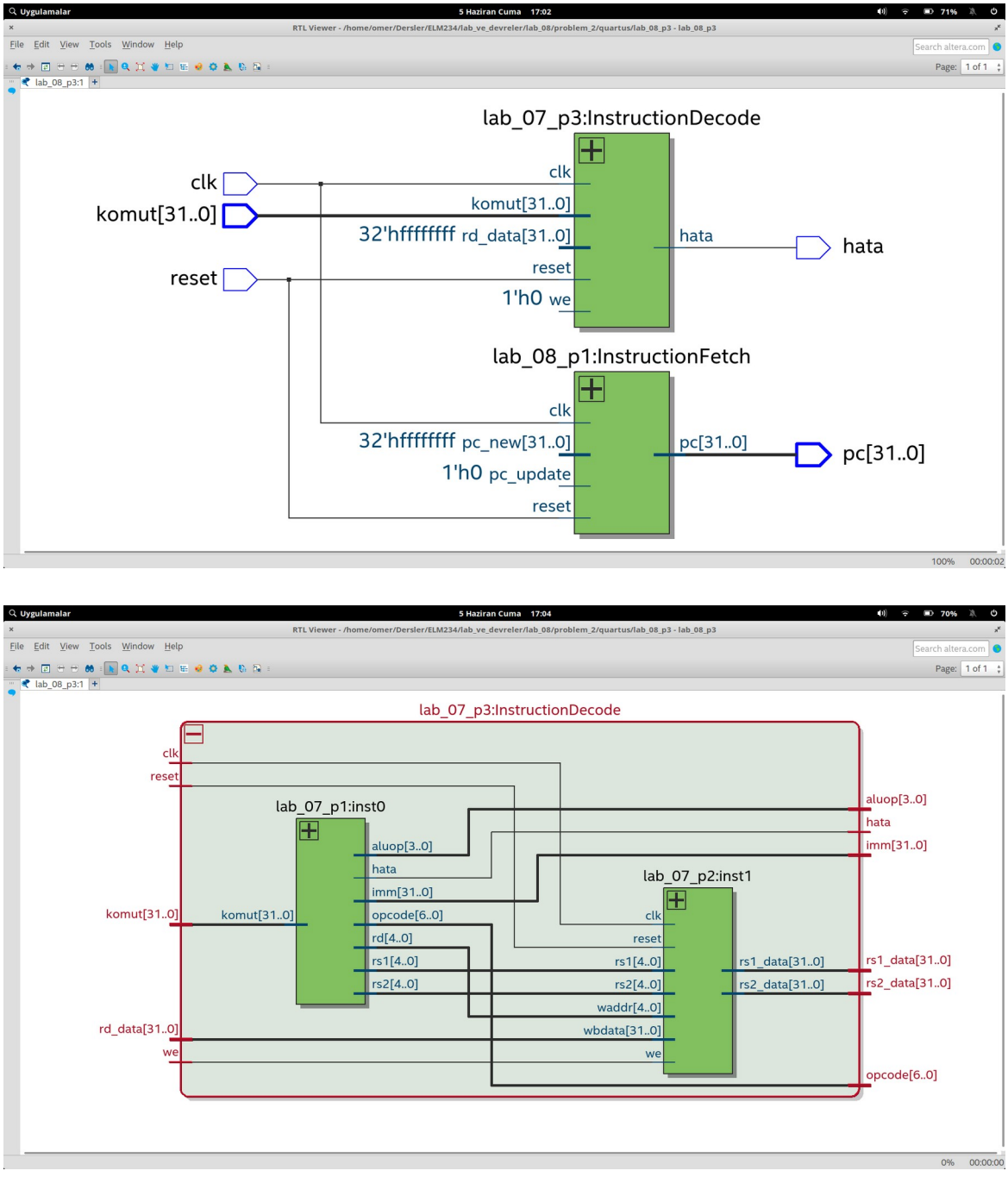
2.2. Problem 2 - Instruction Decode ünitesi

2.2.1. Teorik Araştırma

Bu problem için teorik bir araştırma bulunmamaktadır.

2.2.2. Deneyin Yapılışı

a) RTL Devre Şeması

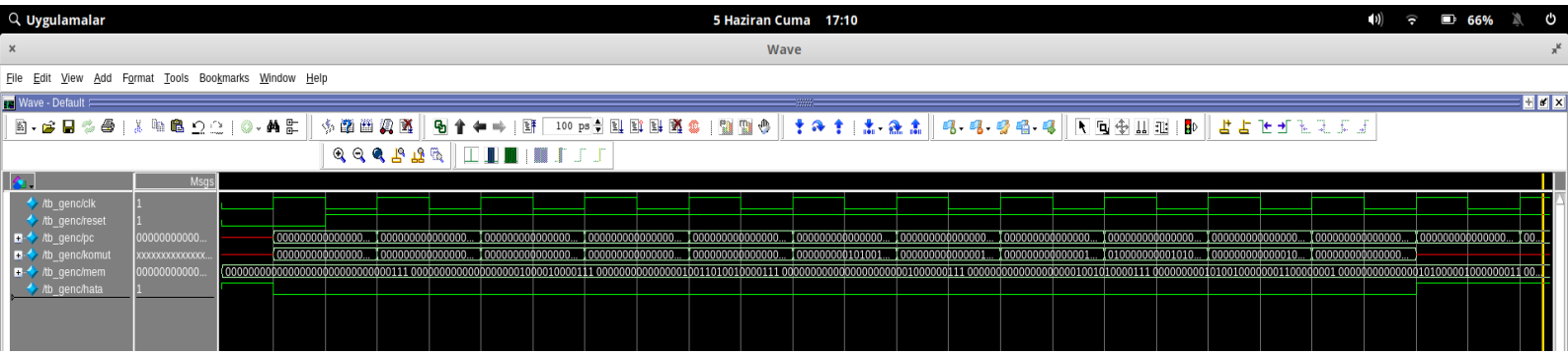


Şekil 1. Lojik devrenin RTL devre şeması.

b) Eşleştirme Ardı Teknoloji Şeması

Şekil 2. Lojik devrenin eşleştirme ardı teknoloji şeması

c) Simulasyon



d) Yorum ve açıklama

Lab 7 de yaptığımız modülü kullanarak gelen komutu decode ettik verilen memory dosyasında ki değerler bitince op code kısmı x olduğu için hata biti pini 1 olmuştur

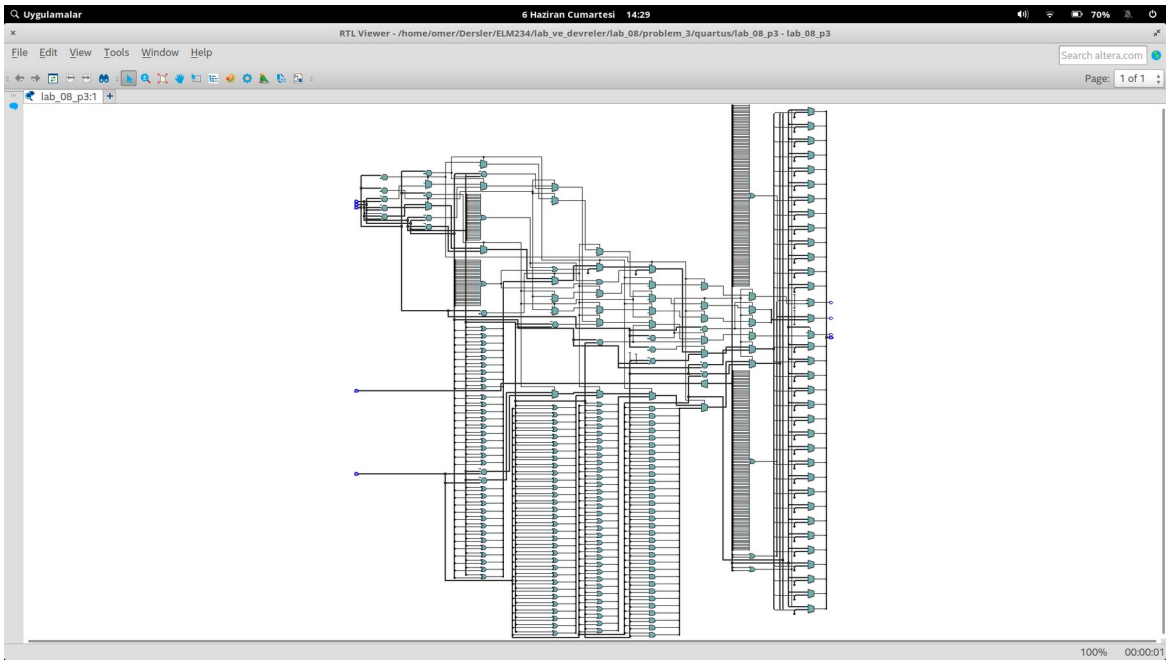
2.3. Problem 3 - Execute ünitesi

2.3.1. Teorik Araştırma

Bu problem için teorik bir araştırma bulunmamaktadır.

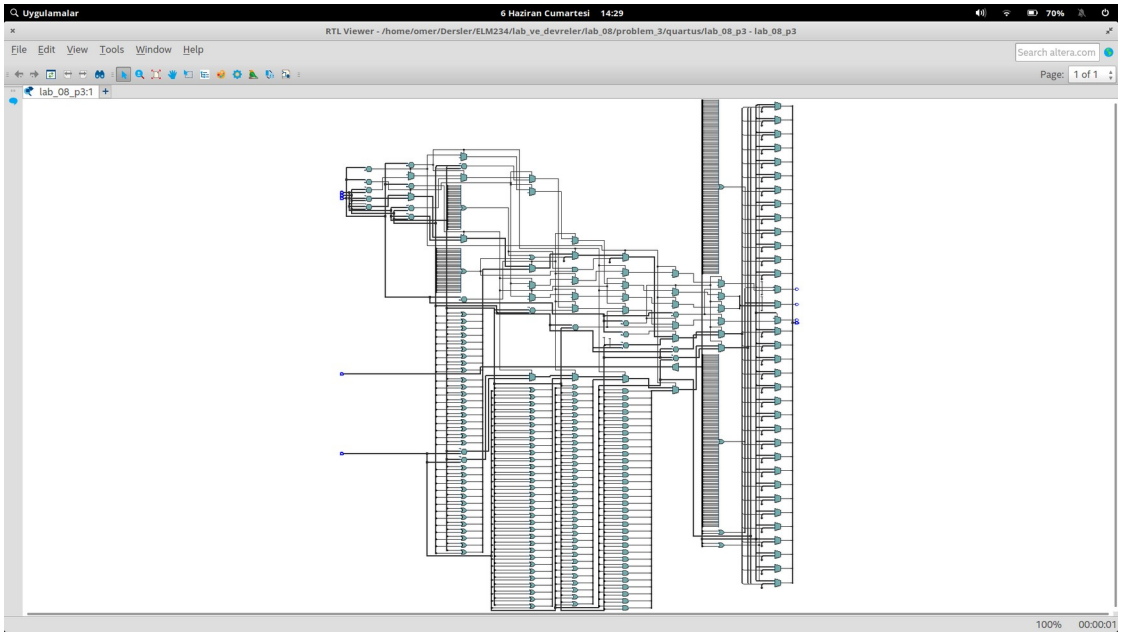
2.3.2. Deneyin Yapılışı

a) RTL Devre Şeması



Şekil 1. Lojik devrenin RTL devre şeması.

b) Eşleştirme Ardı Teknoloji Şeması



Şekil 2. Lojik devrenin eşleştirme ardı teknoloji şeması

c) Yorum ve açıklama

Bu problem de decode ünitesinden gelen komutlar işlenmiştir gelen opcode ve funct değerine göre toplama and leme karşılaştırma branch vs işlemleri yapılmıştır.

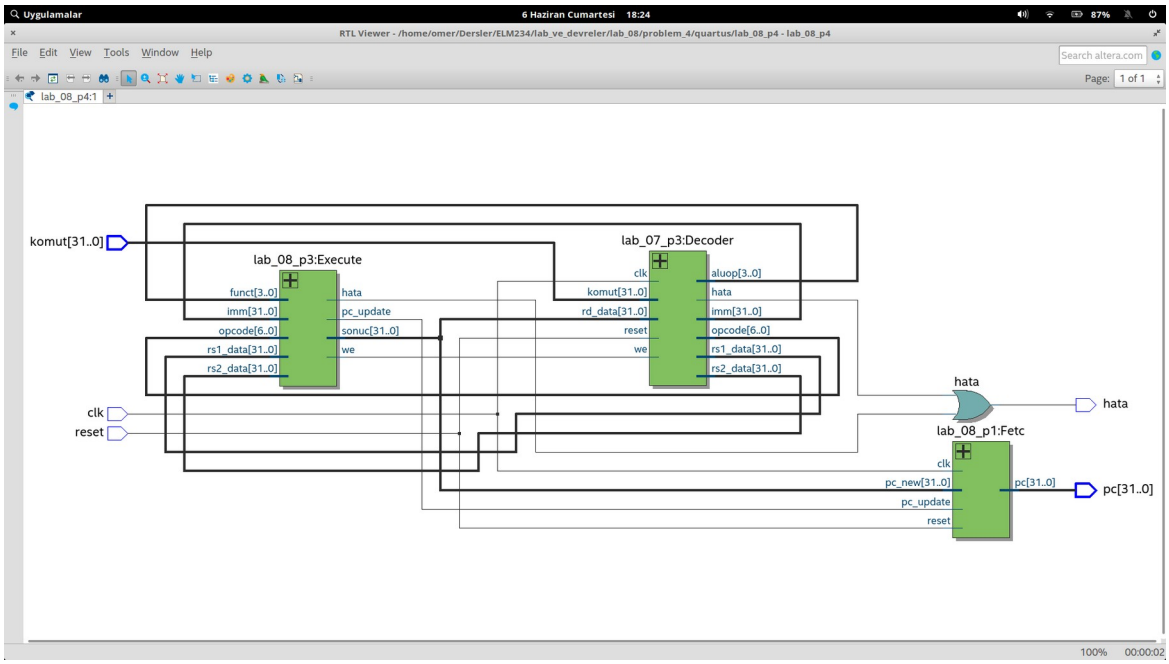
2.4. Problem 4 - Tek ritimli işlemci

2.4.1. Teorik Araştırma

Bu problem için teorik bir araştırma bulunmamaktadır.

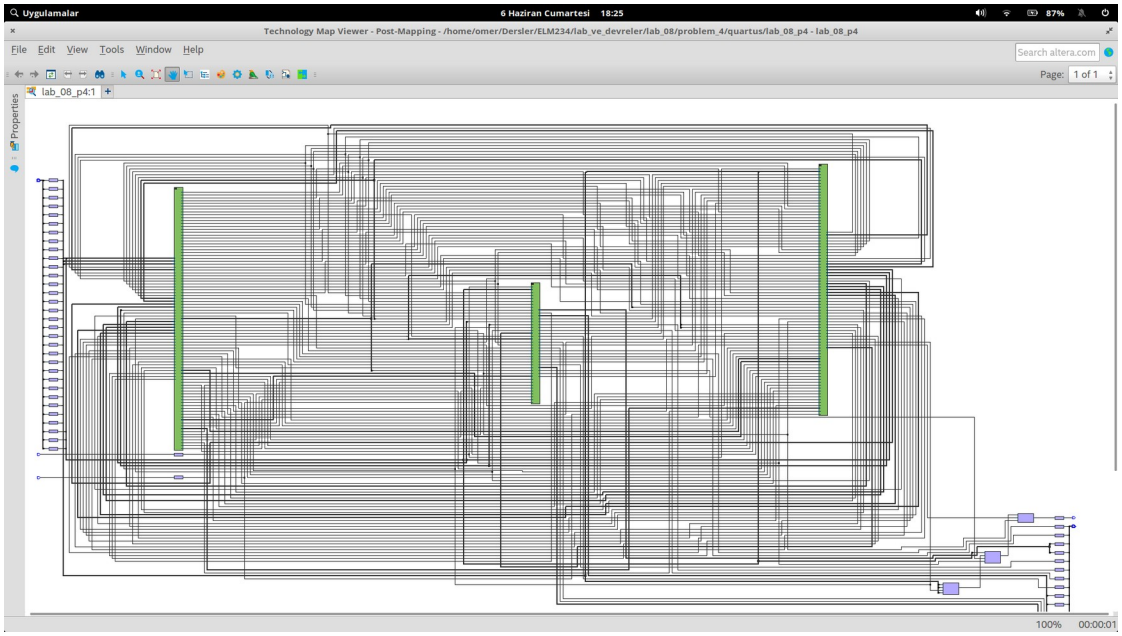
2.4.2. Deneyin Yapılışı

a) RTL Devre Şeması



Şekil 1. Lojik devrenin RTL devre şeması.

b) Eşleştirme Ardı Teknoloji Şeması



Şekil 2. Lojik devrenin eşleştirme ardı teknoloji şeması

c) Analiz ve Sentez Özeti

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Sat Jun 6 18:24:30 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	lab_08_p4
Top-level Entity Name	genc
Family	MAX 10
Total logic elements	2,303
Total combinational functions	1,917
Dedicated logic registers	448
Total registers	448
Total pins	67
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0
UFM blocks	0
ADC blocks	0

Yukarıdaki özette görüldüğü üzere analiz ve sentez başarıyla gerçekleşmiştir ve devrenin gerçeklenebilmesi için 2,303 lojik eleman , 67 pin, 448 register gerekmektedir.

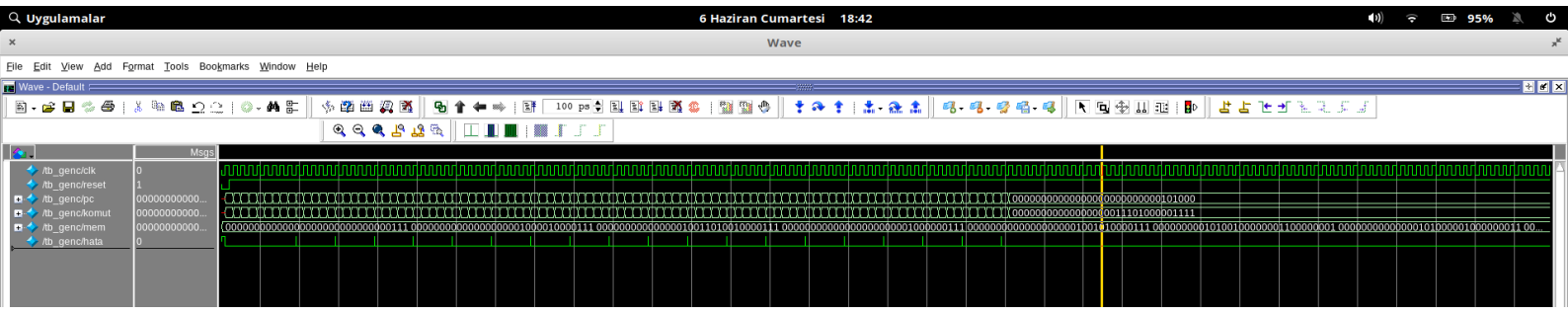
d) Analiz ve Sentez Kaynak Kullanım Özeti

```
+-----------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary         ;
+---------------------------------------------+-------+
; Resource                                    ; Usage ;
+---------------------------------------------+-------+
; Estimated Total logic elements              ; 2,303 ;
;                                             ;       ;
; Total combinational functions               ; 1917  ;
; Logic element usage by number of LUT inputs ;       ;
;     -- 4 input functions                     ; 1241  ;
;     -- 3 input functions                     ; 520   ;
;     -- <=2 input functions                   ; 156   ;
;                                             ;       ;
; Logic elements by mode                      ;       ;
;     -- normal mode                          ; 1733  ;
;     -- arithmetic mode                      ; 184   ;
;                                             ;       ;
; Total registers                             ; 448   ;
;     -- Dedicated logic registers            ; 448   ;
;     -- I/O registers                        ; 0     ;
;                                             ;       ;
; I/O pins                                    ; 67    ;
;                                             ;       ;
; Embedded Multiplier 9-bit elements          ; 0     ;
;                                             ;       ;
; Maximum fan-out node                        ; clk~input ;
; Maximum fan-out                             ; 448   ;
; Total fan-out                               ; 8281  ;
; Average fan-out                             ; 3.31  ;
+---------------------------------------------+-------+
```

e) Fmax Özeti

```
+-----------------------------------------------------+
; Slow 1200mV 85C Model Fmax Summary                  ;
+-----------+-----------------+------------+------+
; Fmax      ; Restricted Fmax ; Clock Name ; Note ;
+-----------+-----------------+------------+------+
; 78.43 MHz ; 78.43 MHz       ; clk        ;      ;
+-----------+-----------------+------------+------+
```

f) Simulasyon



```
// memory data file (do not edit the following line - required for mem load use)
// instance=/tb_genc/genc_i7/Decoder/inst1/mem
// format=mti addressradix=h dataradix=s version=1.0 wordsperline=2
0: 00000000000000000000000000000000 00000000000000000000000000000001
2: 00000000000000000000000000000000 00000000000000000000000000000000
4: 00000000000000000001000001010101 00000000000000000001101001101101
6: 00000000000000000001101001101101 00000000000000000000000000000000
8: 00000000000000000000000000000000 00000000000000000000000000000000
a: 00000000000000000000000000000000 00000000000000000000000000000000
c: 00000000000000000000000000000000
```

Register değerleri

g) Sonuç ve Yorum

Bu lab da dönem boyunca öğrendiklerimizi bir araya getirdik ve kendi küçük işlemcimizi tasarladık. Program counter kullanarak komutları memoryden çektik bu komutlar Decode ünitesinde parçalara ayrıldı funct, opcode vs gibi sonra duruma göre registerlardan değer çekildi bu değerler execute unitesinde işlemler gördü.

```
/* lab_08_p1.sv */

module lab_08_p1 (
    input logic clk, reset,
    output logic [31:0] pc,

    input logic        pc_update,
    input logic [31:0] pc_new
);

logic [31:0] pc1;

always_ff @(posedge clk) begin
if(!reset) // active low reset
pc <= 32'b0;
else
pc <= pc1;
end


always_comb
begin
    if(pc_update)
    pc1 = pc_new;
    else
    pc1 = pc + 4;
end

endmodule
```

```
/* tb_lab_08_p1 */
`timescale 1ns/1ps

module tb_lab_08_p1();

logic clk,reset,pc_update;
logic [31:0] pc,pc_new;
logic [31:0] komut;
logic [31:0] mem [0:63];

assign komut = mem[pc >> 2];

lab_08_p1 uut0(.clk(clk),.reset(reset),.pc_update(pc_update),.pc_new(pc_new),.pc(pc));

always
begin
clk = 0; #5; clk = 1; #5;
end

initial begin
$readmemb("fib20.mem",mem);
end

initial begin
pc_new = 32'b0; #15; pc_new = 32'hAFA146E0; #15;
     pc_new = 32'h03A1D6E0; #15; pc_new = 32'b0;
end

initial begin
pc_update = 1; #60; pc_update = 0;
end

initial begin
reset = 0; #10; reset = 1;
end

initial begin
#1000;
$stop;
end

endmodule
```

```
/* genc_ozdal.sv */
module genc(
    input   logic            clk,reset,
    input   logic   [31:0]   komut, // decode yani lab_07_p3 e bağlanacak
    output  logic   [31:0]   pc,    //Fetch e bağlanacak yani lab_08_p1
    output  logic            hata   // Fetch e bağlanacak
);


logic [31:0]    rs1_data,rs2_data,imm;
logic [3:0]     funct;
logic [6:0]     opcode;
logic           we;
logic           pc_update;
logic [31:0]    rd_pcnew_kablo;
logic [31:0]    komutKablo;

assign komutKablo = komut;

lab_07_p3
InstructionDecode(.clk(clk),.reset(reset),.komut(komutKablo),.hata(hata),.rs1_data(rs1_data),
.rs2_data(rs2_data),.imm(imm),.aluop(funct),.opcode(opcode),.rd_data(rd_pcnew_kablo),.we(we));


lab_08_p1
InstructionFetch(.clk(clk),.reset(reset),.pc(pc),.pc_update(pc_update),.pc_new(rd_pcnew_kablo));


endmodule
```

```
/* lab_07_p1.sv */

module lab_07_p1(
    input  logic   [31:0]   komut,

    output logic    [6:0]   opcode,
    output logic    [3:0]   aluop,
    output logic    [4:0]   rs1, rs2, rd,
    output logic   [31:0]   imm,
    output logic            hata
);
parameter R = 7'b0000001;
parameter I = 7'b0000011;
parameter U = 7'b0000111;
parameter B = 7'b0001111;
logic [3:0] funct;
logic [6:0] code;
assign opcode = komut [6:0];
assign aluop   = funct;

always_comb begin
    case(opcode)
        R:
            begin
            rd      = komut [11: 7];
            rs1     = komut [19:15];
            rs2     = komut [24:20];
            imm     = 32'b0;
            {funct[3],funct[2:0]}   = {komut[30],komut [14:12]};

            hata      = 1'b0;
            end

        I:
            begin
            rd       = komut [11: 7] ;
            rs1      = komut [19:15] ;
            imm      = {20'b0,  komut [31:20]};
            funct    = {1'b0,komut[14:12]};


            rs2       = 5'b0;
            hata      = 1'b0;
            end

        U:
            begin
            rd        = komut [11: 7];
            imm       = komut [31:12];

            funct     = 4'b0;
            rs1       = 5'b0;
            rs2       = 5'b0;
            hata      = 1'b0;
            end

        B:
            begin
            funct       = {1'b0,komut[14:12]};
            rs1         = komut [19:15];
            rs2         = komut [24:20];
            imm         = {18'b0,komut [31:25],komut [11:7],1'b0};
            rd          = 5'b0;
            hata        = 1'b0;
            end
    default:
            begin
            funct       = 4'bx;
            rs1         = 5'bx;
            rs2         = 5'bx;
            imm         = 32'bx;
            rd          = 5'bx;
            hata        = 1'b1;
                      end
endcase
end
endmodule
```

```
/* lab_07_p2.sv */

module lab_07_p2(
    input logic clk,reset,

//yazma portları//
    input logic we,
    input logic [4:0] waddr,
    input logic [31:0] wbdata,

//okuma portları//
    input  logic [4:0] rs1,
    input  logic [4:0] rs2,
    output logic [31:0] rs1_data,
    output logic [31:0] rs2_data
);

logic [31:0] mem[0:7];
integer i;
always_ff @(posedge clk)
    if(!reset)
    for (i=0; i<8; i=i+1)
    mem[i] <= 32'b0;
    else if(we)
        mem[waddr] <= wbdata;

always_comb begin

    rs1_data = mem[rs1];
    rs2_data = mem[rs2];

end
endmodule
```

```
/* lab_07_p3 */

module lab_07_p3(
    input  logic            clk,reset,
    input  logic   [31:0]   komut,

    output logic    [6:0]   opcode,
    output logic    [3:0]   aluop,

    output logic   [31:0]   rs1_data, rs2_data, imm, //rs1_data rs2_data p2
    output logic            hata,

    //yazma

    input logic we, //p2 we
    input logic [31:0] rd_data //p2 wbdata
);

logic [4:0] rd_kablo;
logic [4:0] rs1_kablo;
logic [4:0] rs2_kablo;


lab_07_p1 inst0(.komut(komut), .rd(rd_kablo), .rs1(rs1_kablo), .rs2(rs2_kablo),
.opcode(opcode), .aluop(aluop), .imm(imm), .hata(hata));

lab_07_p2 inst1(.clk(clk), .reset(reset), .we(we), .wbdata(rd_data),
.waddr(rd_kablo), .rs1(rs1_kablo), .rs2(rs2_kablo),.rs1_data(rs1_data),
.rs2_data(rs2_data));



endmodule
```

```
/* lab_08_p1.sv */

module lab_08_p1 (
    input logic clk, reset,
    output logic [31:0] pc,

    input logic        pc_update,
    input logic [31:0] pc_new
);

logic [31:0] pc1;

always_ff @(posedge clk) begin
if(!reset)
pc <= 32'b0;
else
pc <= pc1;
end


always_comb
begin
    if(pc_update)
    pc1 = pc_new;
    else
    pc1 = pc + 4;
end

endmodule
```

```
/* tb_genc_ozdal.sv */
`timescale 1ns/1ps

module tb_genc();

logic clk,reset;
logic [31:0] pc;
logic [31:0] komut;
logic [31:0] mem [0:63];
logic        hata;
assign komut = mem[pc >> 2];

genc_ozdal InstructionDecode(.clk(clk),.reset(reset),.komut(komut),.pc(pc),.hata(hata));

always
begin
clk = 0; #5; clk = 1; #5;
end

initial begin
$readmemb("fib20.mem",mem);
end

initial begin
reset = 0; #10; reset = 1;
end

initial begin
#1000;
$stop;
end

endmodule
```

```
/* lab_08_p3.sv */

module lab_08_p3 (
input logic [31:0] imm,rs1_data,rs2_data,
input   logic  [6:0]  opcode,

input   logic  [3:0]  funct, // alu veya branch ops

output logic  [31:0] sonuc, //rd_data veya new pc

output logic         pc_update,we,hata// branch basarili ise pc enable

);
//opcode
parameter R = 7'b0000001;
parameter I = 7'b0000011;
parameter U = 7'b0000111;
parameter B = 7'b0001111;

always_comb begin
    case(opcode)
////////////////////////////////////////////////////////////////////////////////
            R:
            begin
            //add
            if(funct == 4'b0) begin
            sonuc = $signed(rs1_data) + $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //sub
            else if(funct == 4'b1000) begin
            sonuc = $signed(rs1_data) - $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //and
            else if(funct == 4'b0111) begin
            sonuc = $signed(rs1_data) && $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //or
            else if(funct == 4'b0110) begin
            sonuc = $signed(rs1_data) || $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //eor
            else if(funct == 4'b0100) begin
            sonuc = $signed(rs1_data) ^ $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //lsl
            else if(funct == 4'b0001) begin
            sonuc = rs1_data << rs2_data;
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //lsr
            else if(funct == 4'b0101) begin
            sonuc = $signed(rs1_data) >> $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //asr
            else if(funct == 4'b1101) begin
            sonuc = $signed(rs1_data) >>> $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
```

```
            // hata
            else begin
            sonuc = 32'bx;
            hata = 1'bx;
            pc_update = 1'bx;
            we = 1'bx;
            end

            end
////////////////////////////////////////////////////////////////////////////////
            I:
            begin

            //addi
            if(funct == 4'b0) begin
            sonuc = $signed(rs1_data) + $signed(imm);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //andi
            else if(funct == 4'b0111) begin
            sonuc = $signed(rs1_data) & $signed(imm);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //ori
            else if(funct == 4'b0110) begin
            sonuc = $signed(rs1_data) | $signed(imm);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //eorı
            else if(funct == 4'b0100) begin
            sonuc = $signed(rs1_data) ^ $signed(imm);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //lsli
            else if(funct == 4'b0001) begin
            sonuc = rs1_data << imm;
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //lsri
            else if(funct == 4'b0101) begin
            sonuc = rs1_data >> imm;
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            else begin
            sonuc = 32'bx;
            hata = 1;
            pc_update = 0;
            we = 0;
            end
            end
////////////////////////////////////////////////////////////////////////////////
            U:
            // mov
            begin
            sonuc = imm;
            hata = 0;
            pc_update = 0;
            we = 1;
            end
////////////////////////////////////////////////////////////////////////////////
            B:
            begin
            //b
            if(funct == 4'b0011)begin
            pc_update = 1;
            sonuc = $signed(imm);
            hata = 0;
            we = 0;
```

```
            end
            //beq
            else if(funct == 4'b0000) begin
                if($signed(rs1_data) == $signed(rs2_data)) begin
                pc_update = 1;
                sonuc = $signed(imm);
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //bne
            else if(funct == 4'b0001) begin
                if($signed(rs1_data) != $signed(rs2_data)) begin
                pc_update = 1;
                sonuc = $signed(imm);
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //blt
            else if(funct == 4'b0100) begin
                if($signed(rs1_data) < $signed(rs2_data)) begin
                pc_update = 1;
                sonuc = $signed(imm);
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //bge
            else if(funct == 4'b0101) begin
                if($signed(rs1_data) >= $signed(rs2_data)) begin
                pc_update = 1;
                sonuc = $signed(imm);
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //bltu
            else if(funct == 4'b0110) begin
                if(rs1_data < rs2_data) begin
                pc_update = 1;
                sonuc = imm;
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //bgeu
            else if(funct == 4'b0111) begin
                if(rs1_data >= rs2_data) begin
                pc_update = 1;
```

```
                sonuc = imm;
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //hata
            else begin
            pc_update = 0;
            sonuc = 32'bx;
            hata = 1;
            we = 0;
            end
            end
            default: begin
            pc_update = 0;
            sonuc = 32'bx;
            hata = 1;
            we = 0;
            end

            endcase
end
endmodule
```

```
/* fmax_genc_ozdal.sv */
//timing analiz için yapılmıştır

module fmax_genc_ozdal(
    input   logic            clk,reset,
    input   logic   [31:0]   komut, // decode yani lab_07_p3 e bağlanacak
    output  logic   [31:0]   pc,    //Fetch e bağlanacak yani lab_08_p1
    output  logic            hata   // Fetch e bağlanacak
);

logic            hata_reg;
logic   [31:0]   komut_reg;
logic   [31:0]   pc_reg;

genc_ozdal gencfmax(.clk(clk),.reset(reset),.komut(komut_reg),.pc(pc_reg),.hata(hata_reg));

always_ff @(posedge clk) begin
komut_reg <= komut;
pc   <= pc_reg;
hata <= hata_reg;
end

endmodule
```

```
/* genc_ozdal.sv */

module genc_ozdal(
    input   logic            clk,reset,
    input   logic   [31:0]   komut, // decode yani lab_07_p3 e bağlanacak
    output  logic   [31:0]   pc,    //Fetch e bağlanacak yani lab_08_p1
    output  logic            hata   // Fetch e bağlanacak
);


logic [31:0]    rs1_data,rs2_data,imm;
logic [3:0]     funct;
logic [6:0]     opcode;
logic           we,hataDecoder,hataExecute;
logic           pc_update;
logic [31:0]    rd_pcnew_kablo;


// execute bağlandı
lab_07_p3 Decoder(.clk(clk),.reset(reset),.komut(komut),.hata(hataDecoder),.rs1_data(rs1_data),
.rs2_data(rs2_data),.imm(imm),.aluop(funct),.opcode(opcode),.rd_data(rd_pcnew_kablo),.we(we));

// fetch e bağlandı
lab_08_p3 Execute(.rs1_data(rs1_data),.rs2_data(rs2_data),.imm(imm),.funct(funct),.opcode(opcode),
.pc_update(pc_update),.sonuc(rd_pcnew_kablo),.we(we),.hata(hataExecute));

// decodere bağlandı
lab_08_p1 Fetc(.clk(clk),.reset(reset),.pc(pc),.pc_update(pc_update),.pc_new(rd_pcnew_kablo));


assign hata = hataDecoder | hataExecute;

endmodule
```

```
/* lab_07_p1.sv */


module lab_07_p1(
    input  logic   [31:0]   komut,

    output logic    [6:0]   opcode,
    output logic    [3:0]   aluop,
    output logic    [4:0]   rs1, rs2, rd,
    output logic   [31:0]   imm,
    output logic            hata
);
parameter R = 7'b0000001;
parameter I = 7'b0000011;
parameter U = 7'b0000111;
parameter B = 7'b0001111;
logic [3:0] funct;
logic [6:0] code;
assign opcode = komut [6:0];
assign aluop   = funct;

always_comb begin
    case(opcode)
        R:
            begin
            rd      = komut [11: 7];
            rs1     = komut [19:15];
            rs2     = komut [24:20];
            imm     = 32'b0;
            {funct[3],funct[2:0]}   = {komut[30],komut [14:12]};

            hata      = 1'b0;
            end

        I:
            begin
            rd       = komut [11: 7] ;
            rs1      = komut [19:15] ;
            imm      = {20'b0,  komut [31:20]};
            {funct[3],funct [2:0]}    = {1'b0,komut[14:12]};


            rs2       = 5'b0;
            hata      = 1'b0;
            end

        U:
            begin
            rd        = komut [11: 7];
            imm       = komut [31:12];

            funct     = 4'b0;
            rs1       = 5'b0;
            rs2       = 5'b0;
            hata      = 1'b0;
            end

        B:
            begin
            {funct[3],funct [2:0]}       = {1'b0,komut[14:12]};
            rs1         = komut [19:15];
            rs2         = komut [24:20];
            imm         = {18'b0,komut [31:25],komut [11:7],1'b0};
            rd          = 5'b0;
            hata        = 1'b0;
            end
    default:
            begin
            funct       = 4'b0;
            rs1         = 5'b0;
            rs2         = 5'b0;
            imm         = 32'b0;
            rd          = 5'b0;
            hata        = 1'b1;
                      end
endcase
end
endmodule
```

```
/* lab_07_p2.sv */

module lab_07_p2(
    input logic clk,reset,

//yazma portları//
    input logic we,
    input logic [4:0] waddr,
    input logic [31:0] wbdata,

//okuma portları//
    input  logic [4:0] rs1,
    input  logic [4:0] rs2,
    output logic [31:0] rs1_data,
    output logic [31:0] rs2_data
);

logic [31:0] mem[0:12];
integer i;
always_ff @(posedge clk)
    if(!reset)
    for (i=0; i<13; i=i+1)
    mem[i] <= 32'b0;
    else if(we)
        mem[waddr] <= wbdata;

always_comb begin

    rs1_data = mem[rs1];
    rs2_data = mem[rs2];

end
endmodule
```

```
/* lab_07_p3 */

module lab_07_p3(
    input  logic            clk,reset,
    input  logic   [31:0]   komut,

    output logic    [6:0]   opcode,
    output logic    [3:0]   aluop,

    output logic   [31:0]   rs1_data, rs2_data, imm, //rs1_data rs2_data p2
    output logic            hata,

    //yazma

    input logic we, //p2 we
    input logic [31:0] rd_data //p2 wbdata
);

logic [4:0] rd_kablo;
logic [4:0] rs1_kablo;
logic [4:0] rs2_kablo;


lab_07_p1 inst0(.komut(komut), .rd(rd_kablo), .rs1(rs1_kablo), .rs2(rs2_kablo),
.opcode(opcode), .aluop(aluop), .imm(imm), .hata(hata));

lab_07_p2 inst1(.clk(clk), .reset(reset), .we(we), .wbdata(rd_data),
.waddr(rd_kablo), .rs1(rs1_kablo), .rs2(rs2_kablo),.rs1_data(rs1_data),
.rs2_data(rs2_data));



endmodule
```

```
/* lab_08_p1.sv */

module lab_08_p1 (
    input logic clk, reset,
    output logic [31:0] pc,

    input logic        pc_update,
    input logic [31:0] pc_new
);

logic [31:0] pc1;

always_ff @(posedge clk) begin
if(!reset)
pc <= 32'b0;
else
pc <= pc1;
end


always_comb
begin
    if(pc_update)
    pc1 = pc_new;
    else
    pc1 = pc + 4;
end

endmodule
```

```
/* lab_08_p3.sv */

module lab_08_p3 (
input logic [31:0] imm,rs1_data,rs2_data,
input   logic  [6:0] opcode,

input   logic  [3:0] funct, // alu veya branch ops

output logic  [31:0] sonuc, //rd_data veya new pc

output logic         pc_update,we,hata// branch basarili ise pc enable

);
//opcode
parameter R = 7'b0000001;
parameter I = 7'b0000011;
parameter U = 7'b0000111;
parameter B = 7'b0001111;

always_comb begin
    case(opcode)
////////////////////////////////////////////////////////////////////////////////
            R:
            begin
            //add
            if(funct == 4'b0) begin
            sonuc = $signed(rs1_data) + $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //sub
            else if(funct == 4'b1000) begin
            sonuc = $signed(rs1_data) - $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //and
            else if(funct == 4'b0111) begin
            sonuc = $signed(rs1_data) && $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //or
            else if(funct == 4'b0110) begin
            sonuc = $signed(rs1_data) || $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //eor
            else if(funct == 4'b0100) begin
            sonuc = $signed(rs1_data) ^ $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //lsl
            else if(funct == 4'b0001) begin
            sonuc = rs1_data << rs2_data;
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //lsr
            else if(funct == 4'b0101) begin
            sonuc = $signed(rs1_data) >> $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //asr
            else if(funct == 4'b1101) begin
            sonuc = $signed(rs1_data) >>> $signed(rs2_data);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
```

```
            // hata
            else begin
            sonuc = 32'bx;
            hata = 1'bx;
            pc_update = 1'bx;
            we = 1'bx;
            end

            end
////////////////////////////////////////////////////////////////////////////////
            I:
            begin

            //addi
            if(funct == 4'b0) begin
            sonuc = $signed(rs1_data) + $signed(imm);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //andi
            else if(funct == 4'b0111) begin
            sonuc = $signed(rs1_data) & $signed(imm);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //ori
            else if(funct == 4'b0110) begin
            sonuc = $signed(rs1_data) | $signed(imm);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //eorı
            else if(funct == 4'b0100) begin
            sonuc = $signed(rs1_data) ^ $signed(imm);
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //lsli
            else if(funct == 4'b0001) begin
            sonuc = rs1_data << imm;
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            //lsri
            else if(funct == 4'b0101) begin
            sonuc = rs1_data >> imm;
            hata = 0;
            pc_update = 0;
            we = 1;
            end
            else begin
            sonuc = 32'bx;
            hata = 1;
            pc_update = 0;
            we = 0;
            end
            end
////////////////////////////////////////////////////////////////////////////////
            U:
            // mov
            begin
            sonuc = imm;
            hata = 0;
            pc_update = 0;
            we = 1;
            end
////////////////////////////////////////////////////////////////////////////////
            B:
            begin
            //b
            if(funct == 4'b0011)begin
            pc_update = 1;
            sonuc = $signed(imm);
            hata = 0;
            we = 0;
```

```
            end
            //beq
            else if(funct == 4'b0000) begin
                if($signed(rs1_data) == $signed(rs2_data)) begin
                pc_update = 1;
                sonuc = $signed(imm);
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //bne
            else if(funct == 4'b0001) begin
                if($signed(rs1_data) != $signed(rs2_data)) begin
                pc_update = 1;
                sonuc = $signed(imm);
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //blt
            else if(funct == 4'b0100) begin
                if($signed(rs1_data) < $signed(rs2_data)) begin
                pc_update = 1;
                sonuc = $signed(imm);
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //bge
            else if(funct == 4'b0101) begin
                if($signed(rs1_data) >= $signed(rs2_data)) begin
                pc_update = 1;
                sonuc = $signed(imm);
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //bltu
            else if(funct == 4'b0110) begin
                if(rs1_data < rs2_data) begin
                pc_update = 1;
                sonuc = imm;
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //bgeu
            else if(funct == 4'b0111) begin
                if(rs1_data >= rs2_data) begin
                pc_update = 1;
```

```
                sonuc = imm;
                hata = 0;
                we = 0;
                end
                else begin
                pc_update = 0;
                sonuc = 32'bx;
                hata = 0;
                we = 0;
                end
            end
            //hata
            else begin
            pc_update = 0;
            sonuc = 32'bx;
            hata = 1;
            we = 0;
            end
            end
            default: begin
            pc_update = 0;
            sonuc = 32'bx;
            hata = 1;
            we = 0;
            end
            endcase
end
endmodule
```

```
/* tb_genc_ozdal.sv */
`timescale 1ns/1ps

module tb_genc();

logic clk,reset;
logic [31:0] pc;
logic [31:0] komut;
logic [31:0] mem [0:63];
logic        hata;
assign komut = mem[pc >> 2];

genc_ozdal genc_i7(.clk(clk),.reset(reset),.komut(komut),.pc(pc),.hata(hata));

always
begin
clk = 0; #12; clk = 1; #12;
end

initial begin
$readmemb("fib20.mem",mem);
end

initial begin
reset = 0; #10; reset = 1;
end

initial begin
#10000;
$stop;
end

endmodule
```

```
/* fib20.mem */
00000000000000000000000000000111
00000000000000000001000010000111
00000000000000010011010010000111
00000000000000000000000001000000111
00000000000000000001001010000111
00000000010100100000001100000001
00000000000000010100000010000011
00000000000000011000000010100000011
01000000000101001000010010000001
00000000000001001001010100001111
00000000000000000001110100001111
```

```
/* register file da ki registerların çıktısı
   r6 da ki değer  00000000000000000001101001101101 = 6765
*/

// memory data file (do not edit the following line - required for mem load use)
// instance=/tb_genc/genc_i7/Decoder/inst1/mem
// format=mti addressradix=h dataradix=s version=1.0 wordsperline=2
0: 00000000000000000000000000000000 00000000000000000000000000000001
2: 00000000000000000000000000000000 00000000000000000000000000000000
4: 00000000000000000001000001010101 00000000000000000001101001101101
6: 00000000000000000001101001101101 00000000000000000000000000000000
8: 00000000000000000000000000000000 00000000000000000000000000000000
a: 00000000000000000000000000000000 00000000000000000000000000000000
c: 00000000000000000000000000000000
```