GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x8 Deney Raporu

KizilarmutPolat Tek Ritimli İşlemci

Lojik Devreler ve Tasarım Laboratuvarı Dersi

Hazırlayanlar
1) 171024086-Berat KIZILARMUT
2) 1801022037-Ömer Emre POLAT

1. Giriş

Bu deneyde Basitleştirilmiş komut setine sahip bir Tek Ritimli İşlemci tasarlanacak. Bu işlemciyi tasarlarken sırasıyla Instruction Fetch Ünitesi, Instruction Decode Ünitesi ve Execute Ünitesi tasarlanacak. Ardından bu üniteler birbiri ile birleştirilecektir.

2. Problemler

2.1. Problem I – Instruction Fetch ünitesi

Bu kısımda bir Instruction Fetch Ünitesi tasarlayacağız. Bu ünitenin işlevi bir sonraki çalıştırılacak olan komutun adresini oluşturmak olacak.

A. HDL Yazılımı

```
/* lab8_g41_p1.sv
* Hazırlayanlar: Berat Kızılarmut,
                 Ömer Emre Polat
* Notlar: Instruction Fetch Unit*/
module lab8_g41_p1 (
    input logic clk, reset,
    output logic [31:0] pc,
    input logic pc_update,
    input logic [31:0] pc_new
);
always_ff @(posedge clk)
begin
    if(reset)
    begin
        if(pc_update)
        begin
            pc <= pc_new;
        end
        else
        begin
            pc <= pc + 32'd4;
        end
    end
    else
    begin
        pc <= 32'b0;
    end
end
endmodule
```

Sıradaki komutun adresini tanımlayan ünite SystemVerilog dilinde yazıldı. Senkron aktif-low reset sinyali bağlandı, bu sinyal çalıştırıldığında pc değerini sıfıra çekmektedir. Güncellenmiş PC_Update sinyali üniteye girdiğinde, PC_New değeri PC'ye atanmakta. Güncellenmiş PC_Update sinyali üniteye girmediği taktirde, PC değeri 4 arttırılmaktadır.

B. Testbench Kurulumu

Kurulan devreyi sınamak için duruma ve devreye uygun iki farklı testbench devresi kuruldu. İlk testbench basit bir test olmaktadır. İkinci testbenche ise içinde bilgiler bulunan bir Program Memory eklendi. Program Memory'nin içine yüklenen bilgiler ise föy ekinde verilen fib20.mem dosyasından readmemb fonksiyonu ile kullanıldı.

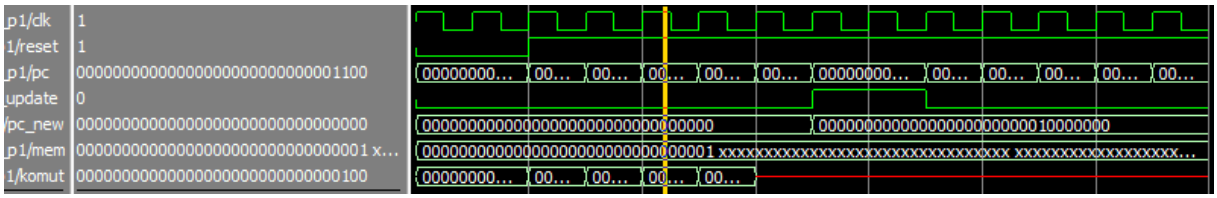
```
/* tb_lab8_g41_p1.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Instruction Fetch Unit Testbench 1 */
`timescale 1ns/1ps
module tb_lab8_g41_p1();
    logic clk, reset;
    logic [31:0] pc;
    logic pc_update;
    logic [31:0] pc_new;
    logic [31:0] mem[0:63]; //5 bitlik giristen dolayı 32x64
    logic [31:0] komut;
    assign komut = mem[pc[29:0]];
    lab8_g41_p1 fetch0(.clk(clk), .reset(reset), .pc(pc),
    .pc_update(pc_update), .pc_new(pc_new));
    always
    begin
        clk = 1; #5;
        clk = 0; #5;
    end
    initial
    begin
        mem[0] = 32'd1;
        mem[4] = 32'd2;
        mem[8] = 32'd3;
        mem[12] = 32'd4;
        mem[16] = 32'd5;
        pc_update = 0; pc_new = 32'b0; reset = 0;
        #20;
        reset = 1;
        #50;
        pc_new = 32'd128; pc_update = 1;
        #20;
        pc_update = 0;
        #50;
        $stop;
    end
endmodule
```

```
/* tb2_lab8_g41_p1.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Instruction Fetch Unit Testbench 2 */
`timescale 1ns/1ps
module tb2_lab8_g41_p1();
    logic clk, reset;
    logic [31:0] pc;
    logic pc_update;
    logic [31:0] pc_new;
    logic [31:0] prog_memory [0:63];
    logic [31:0] komut;
    assign komut = prog_memory[pc[29:0] >> 2];
    lab8_g41_p1 fetch0(.clk(clk), .reset(reset), .pc(pc), .pc_update(pc_update),
.pc_new(pc_new));
    always
    begin
        clk = 1; #5;
        clk = 0; #5;
    end
    initial
    begin
        $readmemb("C:\\Users\\user\\Desktop\\Lojik\\SystemVerilog\\lab8\\fib20.me
m", prog_memory); //loading file
        reset = 0; #3; //resetting
        reset = 1; #2;
        #110; //runtime
        $stop; //end of runtime
    end
endmodule
```

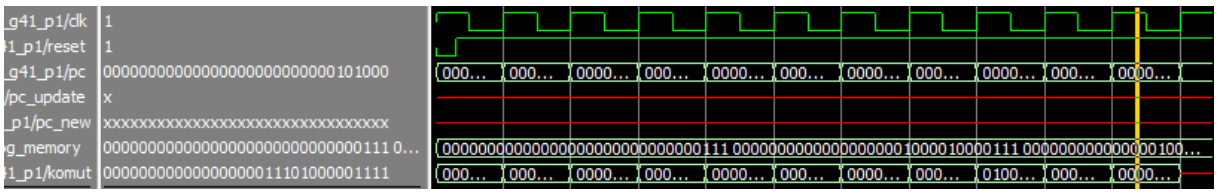
C. Waveshape Analizi

Testbenchlerin sonuçlarını gösteren dalga formları elde edildi.

İlk testbenchimizde, gözlenilebildiği gibi ünite öncelikle reset girişi 0 yapılarak sıfırlanmıştır. Ardından Pc_update sinyali gelene kadar pc_new sinyalinin değeri değişmemiştir. Bu sırada PC sinyalinin değeri 4'er 4'er artmıştır. Basit testbenchte sadece 5 adet komut bulunduğu için, bu komutlar okunduktan sonra sıradaki komutlar tanımsız olduğu için don't care olmuştur.



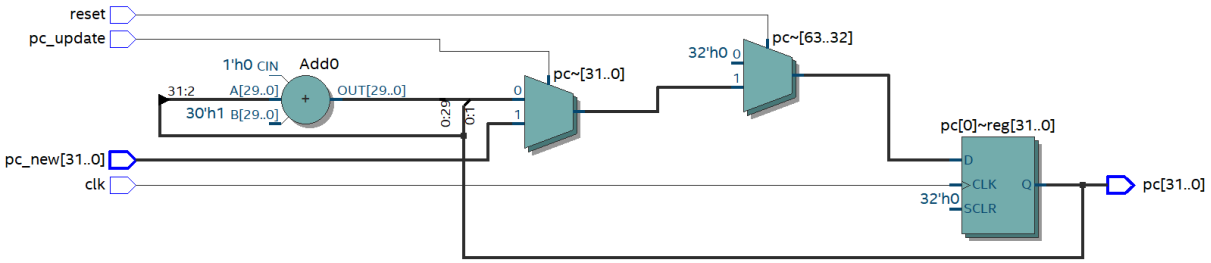
İkinci testbenchimizde ise, ilk testbenchte olduğu gibi en başta reset işlevi sayesinde sıfırlanmıştır. Burada pc_update ve pc_new, basit testhbenchteki gibi manuel değiştirilmediği için tamamiyle don't care gelmiştir. Bu sefer 11 adet tanımlı komut bulunmaktadır ve hepsi okunmuştur.



D. Problem I Analiz ve Yorumlar

	Resource	Usage
1	Estimated Total logic elements	33
2		
3	Total combinational functions	33
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	0
2	-- 3 input functions	0
3	-- <=2 input functions	33
5		
6	Logic elements by mode	
1	-- normal mode	4
2	-- arithmetic mode	29
7		
8	Total registers	32
1	-- Dedicated logic registers	32
2	-- I/O registers	0
9		
10	I/O pins	67
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	rese...nput
15	Maximum fan-out	33
16	Total fan-out	320
17	Average fan-out	1.61

Instruction Fetch Ünitemiz istediğimiz gibi çalışmıştır. Bu ünitede 33 Combinational ALUT ünitesi kullanılmıştır. 32 Adet Register kullanılmıştır. 32 Register gözükmesine rağmen 31 adet Register kullanılmayan durumda olduğu için gerçek aktif kullanılan Register sayısı 1'dir. Ünitenin FMax'i 309.79 MHz olmuştur.



	Fmax	Restricted Fmax	Clock Name	Note
1	309.79 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

2.2 Problem II – Instruction Decode Ünitesi

Tasarlayacağımız olan Instruction Decode Ünitesi, Fetch Ünitesi tarafından çekilen komutların parçalanması ve belirli parçaların istenilen input outputlara gitmesini sağlamaktadır.

Deney föyünde yazanın aksine, Instruction Decode ve Instruction Fetch Üniteleri bu adımda birleştirilmemiştir. Bunun sebebi Execute Ünitesi bulunmadan bu bahsi geçen iki ünitenin birçok input ve output portu boş kalmaktadır, internal nodelar gerekli yerlere bağlanamamaktadır. Fakat bu üniteyi Fetch ünitesi olmadan test edemeyeceğimiz için testbenchimiz iki ünite birleştirilerek yapıldı.

A. HDL Yazılımı

Instruction Decode Ünitesi, LAB7'de tasarladığımı iki ayrı üniteden oluşmaktadır. Bu iki üniteden biri, komutları parçalara bölmektedir, diğer ünite ise bir register file görevi yaparak işlenmiş bilgileri geçici olarak depolamaktadır. LAB7 Problem 3 ünitesine minimal değişiklikler yapılarak, Instruction Decode Ünitesi elde edilmiştir.

```
/* lab7_g41_p1.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Komut Ayırıcı */
module lab7_g41_p1 (
input logic clk, reset,
input logic [31:0] komut,
output logic [6:0] opcode,
output logic [3:0] aluop,
output logic [4:0] rs1,
output logic [4:0] rs2,
output logic [31:0] rs1_data,
output logic [31:0] rs2_data,
output logic [4:0] rd,
output logic [31:0] imm,
output logic hata
);
assign opcode = komut[6:0];
```

```
always_comb
begin
    case(opcode)
        ////////////////R//////////////////////
        7'b0000001:
        begin
            aluop = {komut[30], komut[14:12]};
            rs1 = komut[19:15];
            rs2 = komut[24:20];
            rd = komut[11:7];
            imm = 32'd0;
            hata = 1'b0;
        end
        ////////////////I//////////////////////
        7'b0000011:
        begin
            aluop = {1'b0, komut[14:12]};
            rs1 = komut[19:15];
            rs2 = 5'd0;
            rd = komut[11:7];
            imm = {20'd0, komut[31:20]};
            hata = 1'b0;
        end
        ////////////////U//////////////////////
        7'b0000111:
        begin
            aluop = 4'd0;
            rs1 = 5'd0;
            rs2 = 5'd0;
            rd = komut[11:7];
            imm = {12'd0, komut[31:12]};
            hata = 1'b0;
        end
        ////////////////B//////////////////////
        7'b0001111:
        begin
            aluop = {1'b0, komut[14:12]};
            rs1 = komut[19:15];
            rs2 = komut[24:20];
            rd = 5'd0;
            imm = {19'd0, komut[31:25], komut[11:7], 1'b0};
            hata = 1'b0;
        end
        default:
        begin
            hata = 1'b1;
            aluop = 4'd0;
            rs1 = 5'd0;
            rs2 = 5'd0;
            rd = 4'd0;
            imm = 32'd0;
        end
    endcase
end
endmodule
```

```
/* lab7_g41_p2.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Hafıza oluşturma ve okuma */
module lab7_g41_p2 (
input logic clk, reset,
input logic we,
input logic [4:0] waddr,
input logic [31:0] wbdata,
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 [0:31];
always_comb
begin
    rs1_data = mem[rs1];
    rs2_data = mem[rs2];
end

always_ff @(posedge clk)
begin
    if(we)
        mem[waddr] <= wbdata;
end
endmodule
```

```
/* lab8_g41_p2.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Instruction Decode Unit */
module lab8_g41_p2 (
    input logic clk, reset,
    input logic [31:0] komut,
    output logic [6:0] opcode,
    output logic [3:0] func,
    output logic [31:0] rs1_data,
    output logic [31:0] rs2_data,
    output logic [31:0] imm,
    output logic hata,
    input logic we,
    input logic [31:0] rd_data
);

logic [4:0] rdin, rs1in, rs2in;
logic [31:0] rs1_datain, rs2_datain;

lab7_g41_p1 dut1(.clk(clk), .reset(reset), .komut(komut), .opcode(opcode
), .aluop(func), .rs1(rs1in), .rs2(rs2in), .rs1_data(rs1_datain), .rs2_d
ata(rs2_datain), .rd(rdin), .imm(imm), .hata(hata));

lab7_g41_p2 dut2(.clk(clk), .reset(reset), .we(we), .waddr(rdin), .wbdat
a(rd_data), .rs1(rs1in), .rs2(rs2in), .rs1_data(rs1_data), .rs2_data(rs2
_data));
endmodule
```

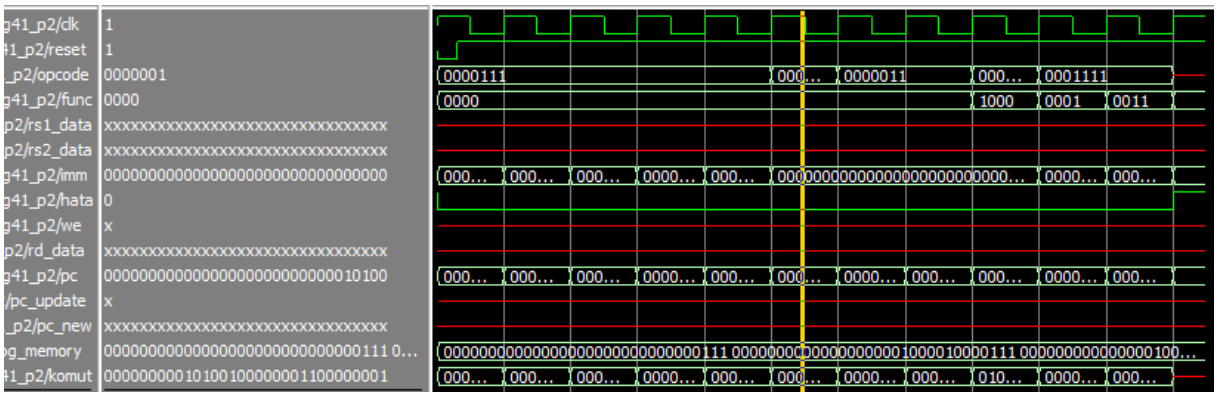
B. Testbench Kurulumu

İkinci Üniteye eklenmemiş olan Fetch Ünitesi, Testbenchte çağırılarak Decode Ünitesi ile gerekli bağlantıları yapılmıştır. Fetch Ünitesi sayesinde Decode Ünitesi sınanabilir hale gelmiştir.

```
/* tb_lab8_g41_p2.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Instruction Encode Unit Testhbench */
`timescale 1ns/1ps
module tb_lab8_g41_p2();
    logic clk, reset;
    logic [6:0] opcode;
    logic [3:0] func;
    logic [31:0] rs1_data;
    logic [31:0] rs2_data;
    logic [31:0] imm;
    logic hata;
    logic we;
    logic [31:0] rd_data;
    logic [31:0] pc; //variables for p1
    logic pc_update;
    logic [31:0] pc_new;
    logic [31:0] prog_memory [0:63]; //program memory for program counter
 to pull the instructions
    logic [31:0] komut;
    assign komut = prog_memory[pc[29:0] >> 2];
    lab8_g41_p2 decode0(.clk(clk), .reset(reset), .komut(komut), .opcode(
opcode), .func(func), .rs1_data(rs1_data), .rs2_data(rs2_data), .imm(imm)
, .hata(hata), .we(we), .rd_data(rd_data));
    lab8_g41_p1 fetch0(.clk(clk), .reset(reset), .pc(pc), .pc_update(pc_u
pdate), .pc_new(pc_new)); //instantiation of p1 so that we can use pc to
pull instructions
    always
    begin
        clk = 1; #5;
        clk = 0; #5;
    end
    initial
    begin
        $readmemb("C:\\Users\\user\\Desktop\\Lojik\\SystemVerilog\\lab8\\
fib20.mem", prog_memory); //loading file
        reset = 0; #3; //resetting
        reset = 1; #2;
        #110; //runtime
        $stop; //end of runtime
    end
endmodule
```

C. Waveshape Analizi

Instruction Decode Ünitesinin, Program Memory ve Fetch Ünitesi sayesinde gelen komutları istenilen şekilde böldüğü ve doğru çıkışları verdiği gözlemlenmiştir.



D. Problem II Analiz ve Yorumlar

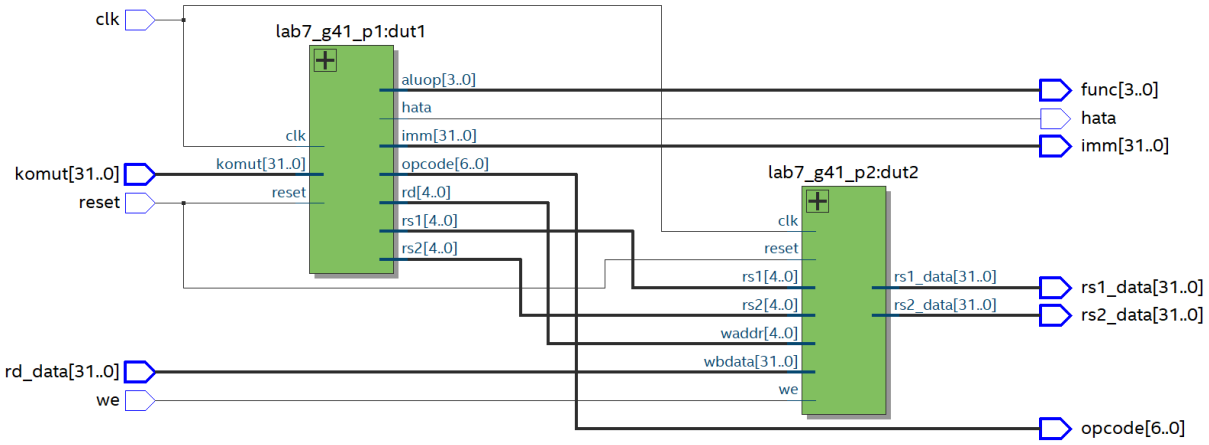
	Resource	Usage
1	Estimated Total logic elements	2,481
2		
3	Total combinational functions	1457
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	1333
2	-- 3 input functions	91
3	-- <=2 input functions	33
5		
6	Logic elements by mode	
1	-- normal mode	1457
2	-- arithmetic mode	0
7		
8	Total registers	1024
1	-- Dedicated logic registers	1024
2	-- I/O registers	0
9		
10	I/O pins	175
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	clk~input
15	Maximum fan-out	1024
16	Total fan-out	9014
17	Average fan-out	3.18

Instruction Decode Ünitesinde, toplam 1457 adet ALUT ünitesi ve toplam 1024 adet Register kullanılmıştır.

Ünitede bu kadar yüksek sayıda Register bulunmasının sebebi ünitede register file bulunmasıdır. Kalan Combinational Logic'lerin büyük bir kısmını komutları parçalara bölme işlemi kullanmaktadır.

RTL Şemasında da gözüktüğü gibi, LAB7de ki iki ünite doğru şekilde birbirine bağlanmıştır ve Instruction Decode Ünitesini oluşturmuştur.

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
1	|lab8_g41_p2	1457 (0)	1024 (0)	0	0	0	0	0	175
1	|lab7_g41_p1:dut1|	63 (63)	0 (0)	0	0	0	0	0	0
2	|lab7_g41_p2:dut2|	1394 (1394)	1024 (1024)	0	0	0	0	0	0



Instruction Decode Unit RTL

2.3 Problem III – Execute Ünitesi

Execute Ünitesi, Instruction Decode Ünitesinden gelen komutlara göre gelen veriyi gerektiği gibi işleyerek, tekrar Instruction Decode Ünitesine ileterek Register File'a kaydettirmektedir.

A. HDL Yazılımı

Execute Ünitesi, çeşitli opcode ve functionlara göre föy ekinde karşılık gelen işlemleri yapmaktadır. İşlenilen verileri ise çıkıştan vermek ile birlikte, yapılan işleme göre pc_new, pc_update ve write-enable çıkışlarını doğru şekilde vermektedir.

```
/* lab8_g41_p3.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Execute Unit */
module lab8_g41_p3 (
    input logic [31:0] rs1_data, rs2_data,
    input logic [31:0] imm,
    input logic [6:0] opcode,
    input logic [3:0] func, // alu veya branch ops
    output logic [31:0] sonuc, // duruma gore ya rd datası ya yeni pc adres
i
    output logic pc_update, // branch basarili, pc yi güncelle
    output logic we, // rd yi güncelle
    output logic hata
);

always_comb
begin
    case(opcode)
        7'b0000001:
        begin
            if(func == 4'b0000) // ADD
            begin
                sonuc = rs1_data + rs2_data;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
```

```
            if(func == 4'b1000) // SUB
            begin
                sonuc = rs1_data - rs2_data;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0111) // AND
            begin
                sonuc = rs1_data & rs2_data;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0110) // OR
            begin
                sonuc = rs1_data | rs2_data;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0100) // EOR
            begin
                sonuc = rs1_data ^ rs2_data;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0001) // LSL
            begin
                sonuc = rs1_data << rs2_data;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0101) // LSR
            begin
                sonuc = rs1_data >> rs2_data;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b1101) // ASR
            begin
                sonuc = $signed(rs1_data) >>> rs2_data;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            begin
                sonuc = 32'b0;
                hata = 1;
                pc_update = 0;
                we = 0;
            end
        end
        7'b0000011:
        begin
            if(func == 4'b0000) // ADDI
```

```
            begin
                sonuc = rs1_data + imm;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0111) // ANDI
            begin
                sonuc = rs1_data & imm;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0110) // ORI
            begin
                sonuc = rs1_data | imm;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0100) // EORI
            begin
                sonuc = rs1_data ^ imm;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0001) // LSLI
            begin
                sonuc = rs1_data << imm;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            if(func == 4'b0101) // LSRI
            begin
                sonuc = rs1_data >> imm;
                hata = 0;
                pc_update = 0;
                we = 1;
            end
            else
            begin
                sonuc = 32'b0;
                hata = 1;
                pc_update = 0;
                we = 0;
            end
        end
        7'b0000111: //MOV
        begin
            sonuc = imm;
            hata = 0;
            pc_update = 0;
            we = 1;
        end
        7'b0001111:
        begin
            if(func == 4'b0011) // B
            begin
                sonuc = imm;
```

```
                hata = 0;
                pc_update = 1;
                we = 0;
            end
            else
            if(func == 4'b0000) // BEQ
            begin
                sonuc = imm;
                hata = 0;
                we = 0;
                if(rs1_data == rs2_data)
                begin
                    pc_update = 1;
                end
                else
                begin
                    pc_update = 0;
                end
            end
            else
            if(func == 4'b0001) // BNE
            begin
                sonuc = imm;
                hata = 0;
                we = 0;
                if(rs1_data != rs2_data)
                begin
                    pc_update = 1;
                end
                else
                begin
                    pc_update = 0;
                end
            end
            else
            if(func == 4'b0100) // BLT
            begin
                sonuc = imm;
                hata = 0;
                we = 0;
                if($signed(rs1_data) < $signed(rs2_data))
                begin
                    pc_update = 1;
                end
                else
                begin
                    pc_update = 0;
                end
            end
            else
            if(func == 4'b0101) // BGE
            begin
                sonuc = imm;
                hata = 0;
                we = 0;
                if($signed(rs1_data) >= $signed(rs2_data))
                begin
                    pc_update = 1;
                end
                else
                begin
                    pc_update = 0;
                end
            end
            else
            if(func == 4'b0110) // BLTU
```

```
            begin
                sonuc = imm;
                hata = 0;
                we = 0;
                if($unsigned(rs1_data) < $unsigned(rs2_data))
                begin
                    pc_update = 1;
                end
                else
                begin
                    pc_update = 0;
                end
            end
            else
            if(func == 4'b0111) // BGEU
            begin
                sonuc = imm;
                hata = 0;
                we = 0;
                if($unsigned(rs1_data) >= $unsigned(rs2_data))
                begin
                    pc_update = 1;
                end
                else
                begin
                    pc_update = 0;
                end
            end
            else
            begin
                sonuc = 32'b0;
                hata = 1;
                pc_update = 0;
                we = 0;
            end
        end
        default:
        begin
            sonuc = 32'b0;
            hata = 1;
            pc_update = 0;
            we = 0;
        end
    endcase
end
endmodule
```

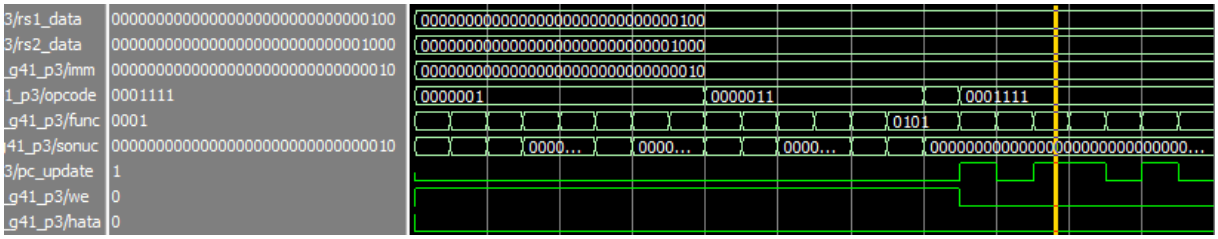
B. Testbench Kurulumu

Kurulan Execute Ünitesinin, giriş yapılan mevcut olan her türlü fonksiyonu belirli girişler ile sınanmıştır. Bu sınanmaların sonuçları istenildiği gibi gelmiştir.

```
/* tb_lab8_g41_p3.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Execute Unit Testbench */
`timescale 1ns/1ps

module tb_lab8_g41_p3();
logic [31:0] rs1_data, rs2_data;
logic [31:0] imm;
logic [6:0] opcode;
logic [3:0] func;
logic [31:0] sonuc;
logic pc_update;
logic we;
logic hata;
lab8_g41_p3 alu0(.rs1_data(rs1_data), .rs2_data(rs2_data), .imm(imm), .
opcode(opcode), .func(func), .sonuc(sonuc), .pc_update(pc_update), .we(
we), .hata(hata));

initial
begin
    rs1_data = 32'd4; rs2_data = 32'd8; imm = 32'd2;
    opcode = 7'b0000001;
    func = 4'b0000; #10;
    func = 4'b1000; #10;
    func = 4'b0111; #10;
    func = 4'b0110; #10;
    func = 4'b0100; #10;
    func = 4'b0001; #10;
    func = 4'b0101; #10;
    func = 4'b1101; #10;

    opcode = 7'b0000011;
    func = 4'b0000; #10;
    func = 4'b0111; #10;
    func = 4'b0110; #10;
    func = 4'b0100; #10;
    func = 4'b0001; #10;
    func = 4'b0101; #10;

    opcode = 7'b0000111;
    #10;

    opcode = 7'b0001111;
    func = 4'b0011; #10;
    func = 4'b0000; #10;
    func = 4'b0001; #10;
    func = 4'b0100; #10;
    func = 4'b0101; #10;
    func = 4'b0110; #10;
    func = 4'b0111; #10;
    $stop;
end
endmodule
```

C. Waveshape Analizi

Föyde verilen fonksiyon setlerinin hepsi sınanmıştır. Branch komutlarında pc_update, immediate kullanılan işlemlerde immediate gelmiştir. Ünite gerektiği gibi çalışmıştır.



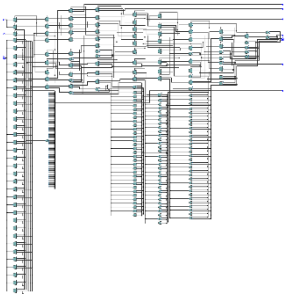
D. Problem III Analiz ve Yorumlar

	Resource	Usage
1	Estimated Total logic elements	1,307
2		
3	Total combinational functions	1307
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	679
2	-- 3 input functions	508
3	-- <=2 input functions	120
5		
6	Logic elements by mode	
1	-- normal mode	1152
2	-- arithmetic mode	155
7		
8	Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0
9		
10	I/O pins	142
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	fun...put
15	Maximum fan-out	121
16	Total fan-out	4657
17	Average fan-out	2.93

Execute Ünitesinde, toplam 1307 adet ALUT ünitesi kullanılmıştır. Ünitede hiç register kullanılmamıştır.

Execute Ünitesi tamimiyle combinational logicten oluşmaktadır. Toplama çıkarma işlemleri için Quartus Prime üzerindeki üniteler kullanılmıştır. Immediate'lar Zero Padding yapılarak kullanılmıştır.

Devrenin RTL şeması oldukça büyük çıktığı için rapor için fotoğraflanarak incelenebilecek durumda değildir.



	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
1	|lab8_g41_p3	1307 (1307)	0 (0)	0	0	0	0	0	142

2.4 Problem IV – Tek Ritimli İşlemci

Önceki adımlarda tasarlamış olduğumuz Instruction Fetch Unit, Instruction Decode Unit ve Execute Unit birleştirilerek bir Tek Ritimli İşlemci oluşturulacaktır.

A. HDL Yazılımı

Instruction Fetch Unit, Instruction Decode Unit ve Execute Unit üniteleri çağırılarak birbirlerine istenilen gibi bağlanılmıştır.

Föyde istenildiği gibi modülün ismi soyadlarımızın birleşimi olan KizilarmutPolat yapılmıştır.

```
/* KizilarmutPolat.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Single Cycle Processor */
module KizilarmutPolat(
    input logic clk, reset,
    input logic [31:0] komut,
    output logic [31:0] pc,
    output logic hata
);
logic we;
logic pc_update;
logic [31:0] rs1_data, rs2_data;
logic [31:0] imm;
logic [6:0] opcode;
logic [3:0] func;
logic [31:0] sonuc;
logic hata1, hata2;
assign hata = hata1 | hata2;
lab8_g41_p1 fetch0(.clk(clk), .reset(reset), .pc(pc), .pc_update(pc_upd
ate), .pc_new(sonuc));
lab8_g41_p2 decode0(.clk(clk), .reset(reset), .komut(komut), .opcode(op
code), .func(func), .rs1_data(rs1_data), .rs2_data(rs2_data), .imm(imm)
, .we(we), .rd_data(sonuc), .hata(hata1));
lab8_g41_p3 alu0(.rs1_data(rs1_data), .rs2_data(rs2_data), .imm(imm), .
opcode(opcode), .func(func), .sonuc(sonuc), .pc_update(pc_update), .we(
we), .hata(hata2));
endmodule
```

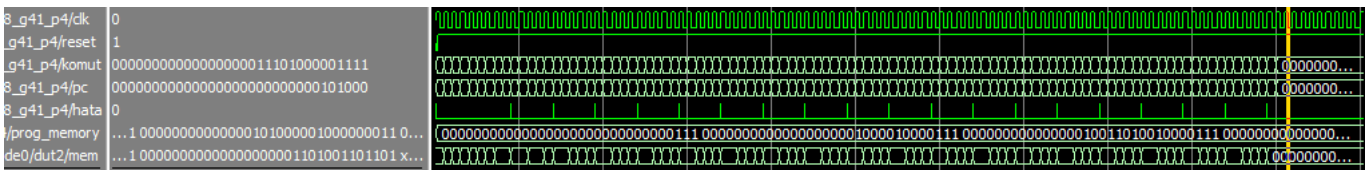
B. Testbench Kurulumu

Tasarlamış olduğumuz Single Cycle ARM Processor çağırılmıştır. Program Memory testbenchte oluşturulup, içerisine föyde verilmiş olan fib20.mem dosyası readmemb fonksiyonu ile yüklenmiştir.

```
/* tb_KizilarmutPolat.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Single Cycle Processor Testbench */
`timescale 1ns/1ps
module tb_KizilarmutPolat();
logic clk, reset;
logic [31:0] komut;
logic [31:0] pc;
logic hata;
KizilarmutPolat arm0(.clk(clk), .reset(reset), .komut(komut), .pc(pc), .hata(hat
a));
logic [31:0] prog_memory [0:63]; //program memory for program counter to pull th
e instructions

assign komut = prog_memory[pc[29:0] >> 2];
always
begin
    clk = 1; #5;
    clk = 0; #5;
end
initial
begin
    $readmemb("C:\\Users\\user\\Desktop\\Lojik\\SystemVerilog\\lab8\\fib20.mem",
 prog_memory); //loading file
    reset = 0; #3; //resetting
    reset = 1; #2;
    #1100; //runtime
    $stop; //end of runtime
end
endmodule
```

C. Waveshape Analizi

Yüklenen program resetten sonra çalışmaya başlamıştır. Program çalışmayı bitirdikten sonra mem(register file) değişkenimizin yedinci elemanına baktığımızda 1101001101101 değerini görmekteyiz. Bu değer decimal olarak 6765 etmektedir. Bu cevap da Fibonacci'nin 20.sayısına eşittir. İşlemcimiz sayıyı doğru hesaplamıştır.



D. Problem IV Analiz ve Yorumlar

	Resource	Usage
1	Estimated Total logic elements	3,863
2		
3	Total combinational functions	2871
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	2116
2	-- 3 input functions	638
3	-- <=2 input functions	117
5		
6	Logic elements by mode	
1	-- normal mode	2687
2	-- arithmetic mode	184
7		
8	Total registers	1056
1	-- Dedicated logic registers	1056
2	-- I/O registers	0
9		
10	I/O pins	67
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	clk~input
15	Maximum fan-out	1056
16	Total fan-out	13868
17	Average fan-out	3.41

Single Cycle Processor'da, toplam 2871 adet ALUT ünitesi ve 1056 Register kullanılmıştır. Burada kullanılan registerların çoğunluğu, 1024 adet, register file tarafından kullanılmıştır.

Decode Ünitemiz, Execute Ünitemizden daha fazla ALUT kullanmıştır. Bunun sebebi Decode Ünitemizin her opcode için çeşitli fonksiyonlar içermesidir.

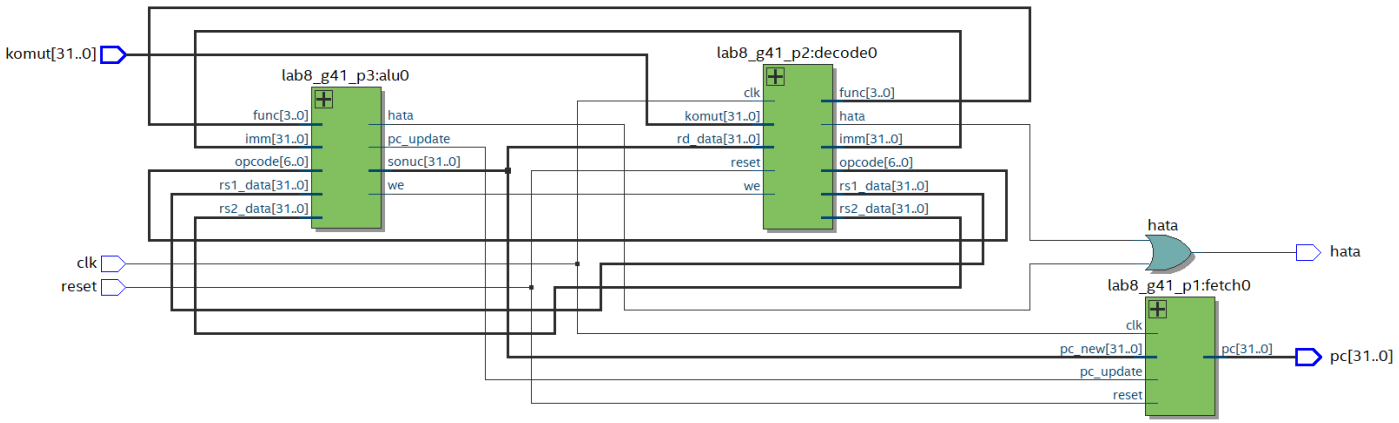
Hata çıkışı, Decode ve Execute ünitelerinin hata sinyallerini OR'lanması olarak verilmiştir.

Maksimum Clock Speedimiz FMax 58.37 MHz olmuştur. Bu Clock Speedi arttırmak için Pipeline Seviyeleri eklenebilir. Ekleyeceğimiz her bir pipeline Clock Speedimizi arttırmak ile birlikte latency'mizi de arttırmaktadır.

İşlemcimize, Data Memory eklenirse Register File dışında, değişkenlerimizi saklayabileceğimiz bir ünite oluşmaktadır. Data Memory, Register File'a oranla çok daha yüksek kapasiteye sahiptir.

	Fmax	Restricted Fmax	Clock Name
1	58.37 MHz	58.37 MHz	clk

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
1	|lab8_g41_p4	2871 (4)	1056 (0)	0	0	0	0	0	67
1	|lab8_g41_p1:fetch0|	64 (64)	32 (32)	0	0	0	0	0	0
2	|lab8_g41_p2:decode0|	1465 (0)	1024 (0)	0	0	0	0	0	0
1	|lab7_g41_p1:dut1|	71 (71)	0 (0)	0	0	0	0	0	0
2	|lab7_g41_p2:dut2|	1394 (1394)	1024 (1024)	0	0	0	0	0	0
3	|lab8_g41_p3:alu0|	1338 (1338)	0 (0)	0	0	0	0	0	0



3. Genel Yorumlar ve Kazanımlar

Bu deneyde basit bir Tek Ritimli İşlemcinin alt ünitelerini daha yakından inceleyip, tasarlayarak Tek Ritimli İşlemci'lere daha farklı bir yaklaşım sağladık. Structural Design sayesinde Tek Ritimli İşlemcilerde bulunan Instruction Fetch, Instruction Decode ve Execute Ünitlerini ayrı ayrı tasarlayıp, bu tasarlamış olduğumuz kalıpları birbirine bağlayarak bir Tek Ritimli İşlemci oluşturduk.

4. Kaynaklar

1. Harris and D. Harris, Digital Design and Computer Architecture: ARM Edition, 1st edition. Morgan Kaufmann, 2015