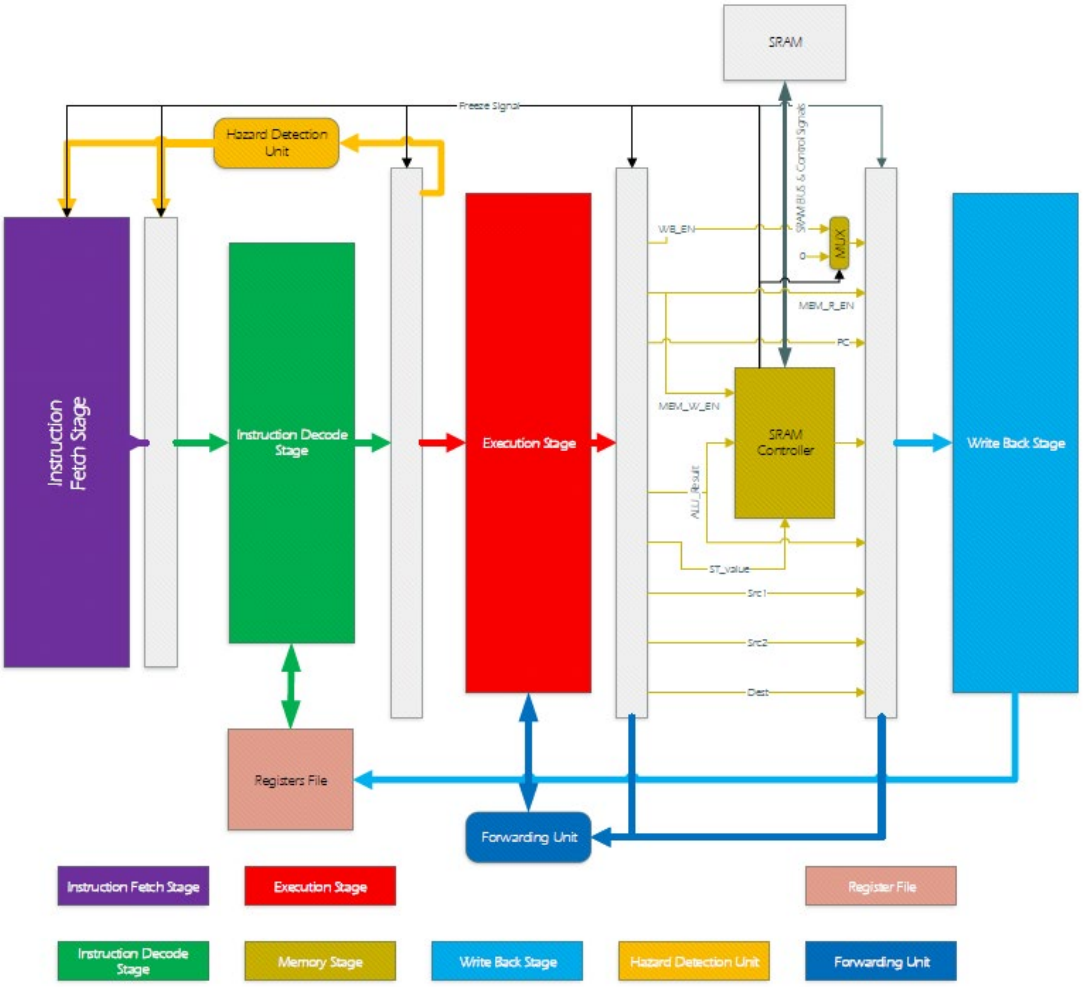
بخش ششم آزمایش

در این آزمایش پردازنده ARM به یک SRAM مجهز شده تا به جای حافظه کوچکی که تا قبل این بخش در خود پردازنده قرار داشت، استفاده شود. دیاگرام این پردازنده با استفاده از SRAM به صورت زیر است:



توضیحات کنترلر:

در این بخش از SRAM-ای که روی برد DE2 قرار دارد استفاده می‌شود و خود ماژول SRAM پیاده‌سازی نمی‌شود. به همین دلیل نیاز داریم از یک کنترلر برای کنترل کردن این SRAM استفاده کنیم. این SRAM یک حافظه 512 کیلوبایتی است که شامل خانه‌های 2 بایتی (16 بیتی) است. به همین دلیل برای آدرس‌دهی خانه‌های این حافظه به 18 بیت آدرس نیاز داریم. از طرفی می‌دانیم که کلمات پردازنده ما 4 بایتی (32 بیتی) است. به همین دلیل لازم است کلمات شکسته شوند و در دو خانه مجاور قرار بگیرند. ورودی و خروجی‌های این SRAM به صورت زیر است:

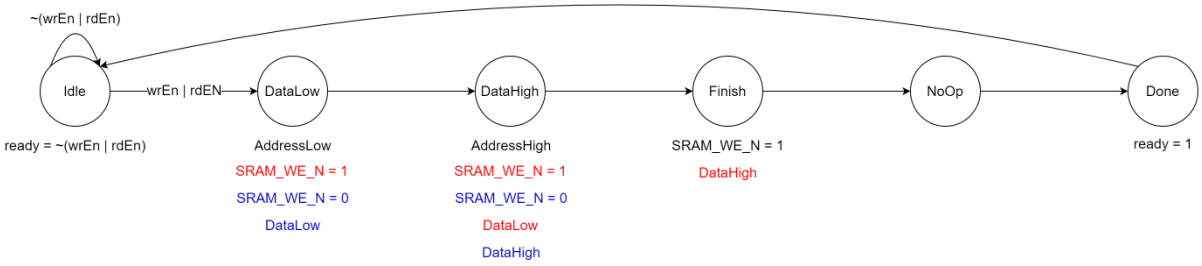
- SRAM_DQ: این پورت که هم خروجی و هم ورودی است (inout)، برای خواندن داده از حافظه و نوشتن داده در آن استفاده می‌شود. هنگامی که از حافظه داده خوانده می‌شود، از طرف کنترلر باید مقدار z (high impedance) قرار بگیرد تا داده به درستی خوانده شود.

- SRAM_ADDR: یک ورودی 18 بیتی که آدرس خانه‌ای که می‌خواهیم از آن داده بخوانیم و یا در آن داده بنویسیم را توسط این ورودی مشخص می‌کنیم.
- SRAM_WE_N: یک ورودی کنترلی به صورت active-low که اگر برابر با 0 باشد، نشان می‌دهد که می‌خواهیم در حافظه داده بنویسیم و اگر برابر با 1 باشد، یعنی می‌خواهیم از آن داده بخوانیم.
- SRAM_UB_N: یک ورودی کنترلی به صورت active-low که اگر برابر با 0 باشد، بایت پر ارزش را فعال می‌کند. در این بخش این سیگنال همواره برابر با 0 است.
- SRAM_LB_N: یک ورودی کنترلی مانند SRAM_UB_N با این تفاوت که اگر برابر با 0 باشد، بایت کم ارزش را فعال می‌کند. این سیگنال نیز همواره به 0 متصل است.
- SRAM_CE_N: یک سیگنال کنترلی به صورت active-low که اگر برابر با 0 باشد، chip حافظه را فعال می‌کند. این سیگنال همواره به 0 متصل است.
- SRAM_OE_N: یک سیگنال کنترلی به صورت active-low که اگر برابر با 0 باشد، خروجی حافظه را فعال می‌کند. مقدار این سیگنال در این بخش همواره برابر با 0 است.

با توجه به اینکه این ماژول حافظه خارج از پردازنده قرار دارد و همچنین خانه‌های آن به صورت 2 بایتی است، نمی‌توانیم در یک کلاک همانند حافظه قبلی، عملیات حافظه را انجام دهیم و حداقل به 3 کلاک نیاز داریم. از طرفی، برای اینکه بعدا بتوانیم کارایی SRAM در کنار حافظه نهان را با حالت فعلی (بدون حافظه نهان) مقایسه کنیم، تعداد کلاک لازم برای انجام عملیات را 6 عدد در نظر می‌گیریم.
حال با توجه به اینکه نیاز داریم در این 6 کلاک پردازنده به طور کامل متوقف شود تا عملیات حافظه کامل شود، توقف یا عدم توقف پردازنده را با یک سیگنال ready مشخص می‌کنیم. زمانی که یک دستور حافظه‌ای (load و یا store) وارد بخش MEM می‌شود، این سیگنال برابر با 0 می‌شود. در بیرون این بخش، یک سیگنال freeze وجود دارد که نقیض سیگنال ready است و در نتیجه در این زمان فعال شده و تمام رجیستربلاک‌های پردازنده را متوقف می‌کند یا به عبارتی اجازه لود به آن‌ها نمی‌دهد.

پیاده‌سازی کنترلر:

برای پیاده‌سازی SRAM Controller از یک state machine استفاده شده که به صورت زیر است (سیگنال‌ها و داده‌های مخصوص خواندن با رنگ قرمز و مخصوص نوشتن با رنگ آبی و موارد مشترک با رنگ مشکی مشخص شده‌اند):

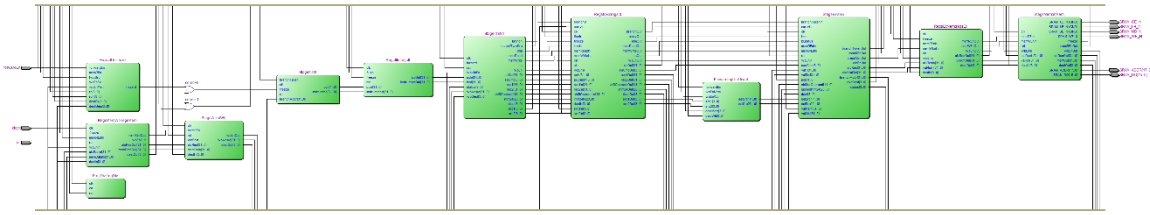


کد این کنترلر به صورت زیر است:

```
module SramController(
    input clk, rst,
    input wrEn, rdEn,
    input [31:0] address,
    input [31:0] writeData,
    output reg [31:0] readData,
    output reg ready,            // to freeze other stages

    inout [15:0] SRAM_DQ,        // SRAM Data bus 16 bits
    output reg [17:0] SRAM_ADDR, // SRAM Address bus 18 bits
    output SRAM_UB_N,            // SRAM High-byte data mask
    output SRAM_LB_N,            // SRAM Low-byte data mask
    output reg SRAM_WE_N,        // SRAM Write enable
    output SRAM_CE_N,            // SRAM Chip enable
    output SRAM_OE_N             // SRAM Output enable
);
    assign {SRAM_UB_N, SRAM_LB_N, SRAM_CE_N, SRAM_OE_N} = 4'b0000;

    wire [31:0] memAddr;
    assign memAddr = address - 32'd1024;

    wire [17:0] sramLowAddr, sramHighAddr;
    assign sramLowAddr = memAddr[18:1];
    assign sramHighAddr = sramLowAddr + 18'd1;

    reg [15:0] dq;
    assign SRAM_DQ = wrEn ? dq : 16'bz;

    localparam Idle = 3'd0, DataLow = 3'd1, DataHigh = 3'd2, Finish = 3'd3, NoOp = 3'd4, Done = 3'd5;
    reg [2:0] ps, ns;

    always @(ps or wrEn or rdEn) begin
        case (ps)
            Idle: ns = (wrEn == 1'b1 || rdEn == 1'b1) ? DataLow : Idle;
            DataLow: ns = DataHigh;
            DataHigh: ns = Finish;
            Finish: ns = NoOp;
            NoOp: ns = Done;
            Done: ns = Idle;
        endcase
    end

    always @(*) begin
        SRAM_ADDR = 18'b0;
        SRAM_WE_N = 1'b1;
        ready = 1'b0;

        case (ps)
            Idle: ready = ~(wrEn | rdEn);
            DataLow: begin
                SRAM_ADDR = sramLowAddr;
                SRAM_WE_N = ~wrEn;
                dq = writeData[15:0];
                if (rdEn)
                    readData[15:0] <= SRAM_DQ;
            end
            DataHigh: begin
                SRAM_ADDR = sramHighAddr;
                SRAM_WE_N = ~wrEn;
                dq = writeData[31:16];
                if (rdEn)
                    readData[31:16] <= SRAM_DQ;
            end
            Finish: begin
                SRAM_WE_N = 1'b1;
            end
            NoOp:;
            Done: ready = 1'b1;
        endcase
    end

    always @(posedge clk or posedge rst) begin
        if (rst) ps <= Idle;
        else ps <= ns;
    end
endmodule
```

تغییرات RTL:

همانطور که قبلا ذکر شد، ابتدا ماژول حافظه قبلی حذف شد و سپس یک کنترلر به این استیج اضافه شد. یک سیگنال freeze که not شده سیگنال ready خارج شده از کنترلر است، به پردازنده اضافه شده که تمام رجیستربلاک‌ها را متوقف می‌کند. در بخش‌هایی که قبلا سیگنال hazard را داشتیم، این سیگنال با سیگنال hazard ترکیب شده و نتیجه or آن‌ها در این بخش وجود دارد. از طرفی یک Multiplexer نیز برای سیگنال WbEn که از استیج MEM به استیج WB می‌رود، قرار داده شد که از نوشتن در رجیسترفایل به طور مداوم، زمانی که منتظر اتمام عملیات حافظه SRAM هستیم، جلوگیری شود. سیگنال سلکت این Multiplexer همان سیگنال freeze است. خروجی RTL Viewer برای پردازنده به صورت زیر است:

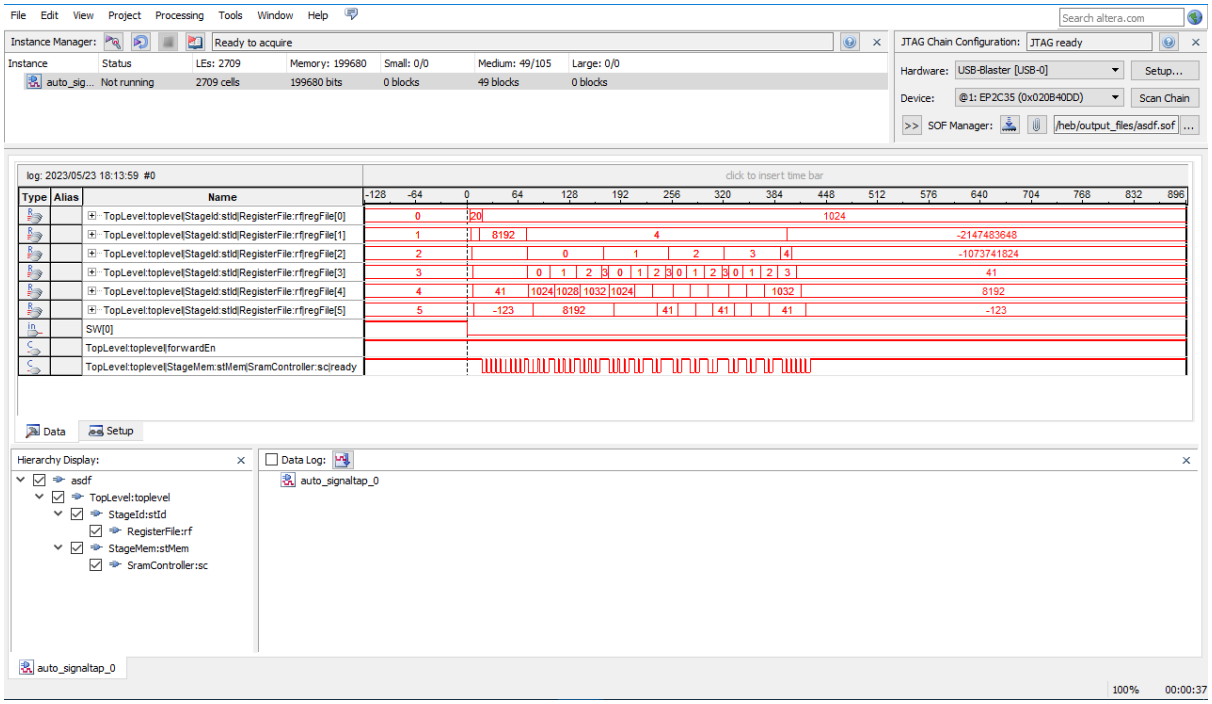


کد کنونی مرحله MEM:

```
module StageMem(
    input clk, rst,
    input wbEnIn, memREnIn, memWEnIn,
    input [31:0] aluResIn, valRm,
    input [3:0] destIn,
    output wbEnOut, memREnOut,
    output [31:0] aluResOut, memOut,
    output [3:0] destOut,
    output freeze,
    inout [15:0] SRAM_DQ,
    output [17:0] SRAM_ADDR,
    output SRAM_UB_N, SRAM_LB_N, SRAM_WE_N, SRAM_CE_N, SRAM_OE_N
);
    assign memREnOut = memREnIn;
    assign aluResOut = aluResIn;
    assign destOut = destIn;

    wire ready;
    assign freeze = ~ready;

    SramController sc(
        .clk(clk), .rst(rst),
        .wrEn(memWEnIn), .rdEn(memREnIn),
        .address(aluResIn),
        .writeData(valRm),
        .readData(memOut),
        .ready(ready),
        .SRAM_DQ(SRAM_DQ),
        .SRAM_ADDR(SRAM_ADDR),
        .SRAM_UB_N(SRAM_UB_N),
        .SRAM_LB_N(SRAM_LB_N),
        .SRAM_WE_N(SRAM_WE_N),
        .SRAM_CE_N(SRAM_CE_N),
        .SRAM_OE_N(SRAM_OE_N)
    );

    Mux2To1 #(1) ramWbEn(
        .a0(wbEnIn),
        .a1(1'b0),
        .sel(freeze),
        .out(wbEnOut)
    );
endmodule
```

مقایسه کارایی پردازنده با حالت حافظه داخلی

خروجی برنامه محک در SignalTap:



برای بدست آوردن تعداد کلاک دقیق برای اتمام برنامه از ModelSim استفاده می‌کنیم.

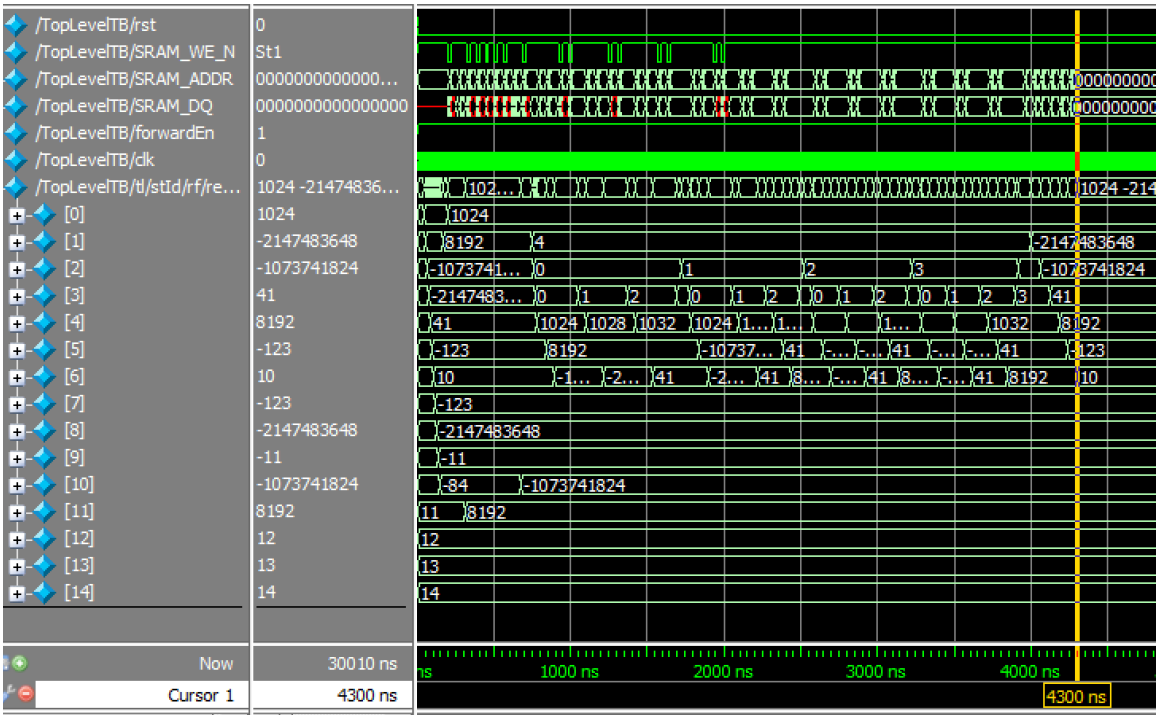
یک ماژول شبیه‌ساز SRAM نوشته شد و از آن در testbench اینستنس گرفته شده است:

```
module Sram(
    input clk, rst,
    input SRAM_WE_N,
    input [17:0] SRAM_ADDR,
    inout [15:0] SRAM_DQ
);
    reg [15:0] memory [0:511];
    assign SRAM_DQ = (SRAM_WE_N == 1'b1) ? memory[SRAM_ADDR] : 16'dz;

    always @(posedge clk) begin
        if (SRAM_WE_N == 1'b0) begin
            memory[SRAM_ADDR] = SRAM_DQ;
        end
    end
endmodule
```

کد تست‌بنچ به صورت زیر است:

```
`timescale 1ns/1ns

module TopLevelTB();
    localparam HCLK = 5;

    reg clk, rst, forwardEn;

    wire SRAM_WE_N;
    wire [17:0] SRAM_ADDR;
    wire [15:0] SRAM_DQ;

    Sram sram(
        .clk(clk), .rst(rst),
        .SRAM_WE_N(SRAM_WE_N),
        .SRAM_ADDR(SRAM_ADDR),
        .SRAM_DQ(SRAM_DQ)
    );

    TopLevel tl(
        .clock(clk), .rst(rst), .forwardEn(forwardEn),
        .SRAM_ADDR(SRAM_ADDR),
        .SRAM_DQ(SRAM_DQ),
        .SRAM_WE_N(SRAM_WE_N),
        .SRAM_UB_N(),
        .SRAM_LB_N(),
        .SRAM_CE_N(),
        .SRAM_OE_N()
    );

    always #HCLK clk = ~clk;

    initial begin
        {clk, rst, forwardEn} = 3'b011;
        #10 rst = 1'b0;
        #30000 $stop;
    end
endmodule
```

نتیجه شبیه‌سازی:



در اینجا همانطور که می‌بینیم forwardEn همانند شکل SignalTap روشن است و از آنجا که هر کلاک در تست‌بنچ 10ns است، 430 کلاک تا اتمام برنامه گرفته شده است (که در شکل SignalTap هم مشهود است).
در حالتی که از مموری داخل پردازنده استفاده می‌شد، طبق گزارش قبلی، 195 کلاک نیاز بود.
پس کارایی پردازنده کاهش یافته است و برای اجرای برنامه محک به 235 کلاک بیشتر نیاز دارد.

$$\frac{195}{430} = 0.45 \approx 50\%\ decline\ in\ performance$$

نتایج سنتز و مقایسه هزینه سخت‌افزار با حالت حافظه داخلی

نتیجه ستنز:

Flow Summary	
Flow Status	Successful - Tue May 23 18:12:55 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	asdf
Top-level Entity Name	arm
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	5,652 / 33,216 (17 %)
Total combinational functions	3,680 / 33,216 (11 %)
Dedicated logic registers	3,549 / 33,216 (11 %)
Total registers	3549
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	199,680 / 483,840 (41 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

در اینجا 5652 المان FPGA استفاده شده است.
در مقایسه، طبق گزارش قبلی، با وجود مموری داخل پردازنده 7753 المان FPGA استفاده شده بود.
پس المان‌ها طبق انتظار کاهش یافته‌اند (چون که از مموری خارج از پردازنده استفاده می‌شود و مموری داخلی حذف شده است) و 2101 المان کمتر استفاده شده است.

$$\frac{5652}{7753} = 0.73 \approx 30\%\ less\ elements$$