بخش هفتم آزمایش

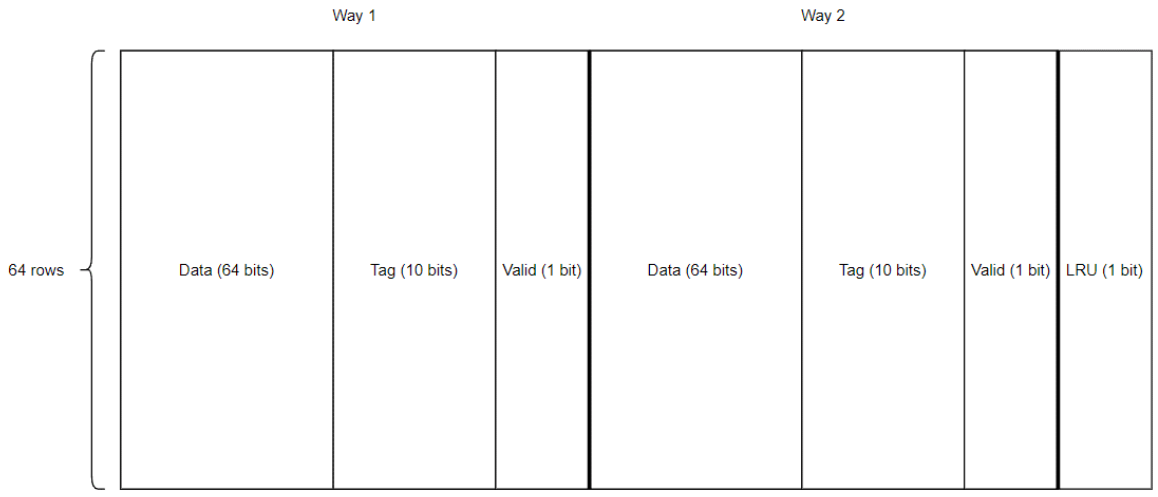
در بخش قبل یک SRAM Controller پیاده‌سازی شد تا بتوانیم به جای یک حافظه بسیار کوچک در خود پردازنده، از یک حافظه بزرگ‌تر در خارج پردازنده استفاده کنیم. در این حالت توانستیم از حافظه 512 کیلوبایتی خود برد DE2 استفاده کنیم و فقط لازم بود که برای آن یک کنترلر پیاده‌سازی کنیم. اما این نوع پیاده‌سازی مشکلی برای ما به وجود آورد؛ آن هم تاخیر بسیار زیاد خواندن و نوشتن در حافظه خارجی بود، که در واقع برای هر عملیات حافظه باید 6 کلاک صبر می‌کردیم و در کل این مدت پردازنده نمی‌توانست کارش را ادامه دهد. حال در این بخش می‌خواهیم با استفاده از پیاده‌سازی حافظه نهان (cache)، این مشکل را تا حد خوبی رفع کنیم.

Cache Controller

در این بخش یک cache controller باید به stage مموری اضافه شود. این کنترلر به صورت کاملا combinational پیاده‌سازی می‌شود تا به خاطر cache کلاکی را از دست ندهیم. با توجه به اضافه شدن این مورد، حال Cache Controller این وظیفه را به عهده می‌گیرد که SRAM Controller را کنترل کند. پس در این حالت، Cache Controller در سر راه SRAM Controller قرار می‌گیرد و فرمان‌های کنترلی آن را صادر می‌کند:

ساختار Cache

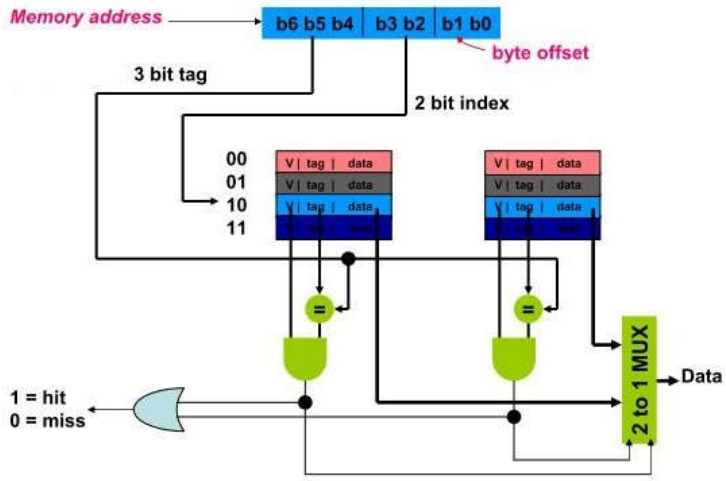
حافظه نهانی که در این آزمایش استفاده شده، یک 2-Way Set Associative Cache است. این حافظه دارای 64 سطر است و در نتیجه index آن 6 بیتی است. از طرفی، هر way دارای 64 بیت است و با توجه به اینکه آدرس‌دهی به صورت بایتی است، 3 بایت آفست خواهیم داشت که بتواند هر یک از 8 بایت یک way را مشخص کند. در نتیجه با توجه به آدرس 19 بیتی، 10 بیت برای tag باقی می‌ماند. در نهایت ساختار cache به صورت زیر خواهد بود:



تگ valid در هر way برای این است که متوجه شویم آیا مقدار درستی داخل cache قرار دارد یا خیر. همچنین، بیت LRU برای سیاست جایگزینی استفاده شده که در ادامه توضیح داده می‌شود.

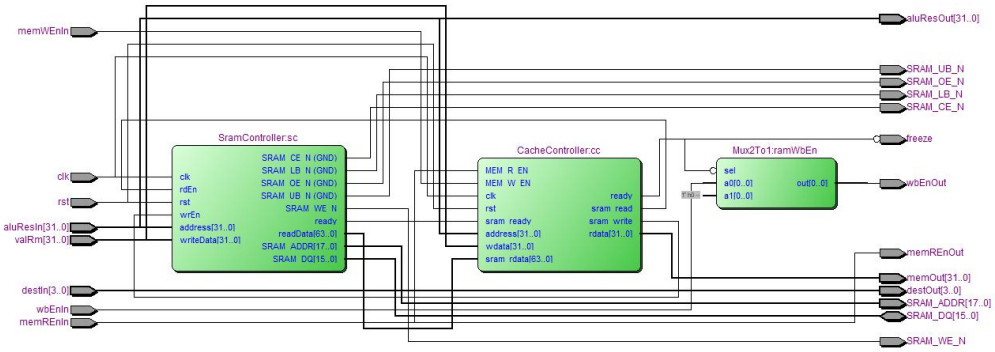
RTL ماژول کنترلر

اگر بخواهیم ساختار RTL این ماژول را رسم کنیم، همچین شکلی خواهد داشت:



در واقع ابتدا index را از آدرس ورودی استخراج می‌کنیم. سپس، در سطر مورد نظر، هر دو way را بررسی می‌کنیم و اگر در یک way، tag داده با tag استخراج شده از آدرس یکسان بود و تگ valid هم 1 بود، hit اتفاق افتاده است.

نتیجه RTL View کوارتس نیز به صورت زیر است:



سیاست جایگزینی

در این حافظه از سیاست Least Recently Used (LRU) استفاده شده است. در این حالت اگر بخواهیم یک داده را جایگزین داده دیگری کنیم، سطر way-ای انتخاب می‌شود که در فاصله دورتری از آن استفاده کردیم. هندل کردن این مورد به این صورت است که هنگامی که داده‌ای از cache خوانده می‌شود، LRU را برابر با way-ای قرار می‌دهیم که از آن داده را خواندیم. زمانی که می‌خواهیم داده جدیدی را در cache بنویسیم، به بیت LRU نگاه می‌کنیم و اگر بیت LRU برابر با 0 بود، way 2 را برای جایگزینی انتخاب می‌کنیم و در غیر این صورت، way 1 را جایگزین می‌کنیم. پس از جایگزینی نیز بیت LRU را برابر با way-ای قرار می‌دهیم که در آن

داده را نوشتیم. زمانی که می‌خواهیم یک داده که در cache وجود دارد را آپدیت کنیم، صرفا فقط بیت valid را 0 کرده و داده را مستقیما در مموری آپدیت می‌کنیم. در نتیجه از حالت تغییر یافته Write Through استفاده می‌شود.

پیاده‌سازی

در کد SRAM Controller تغییر کوچکی ایجاد شد که برای خواندن به جای خواندن 32 بیت، 64 بیت را بخواند و به Cache انتقال دهد. کد این بخش به صورت زیر است:

```
wire [17:0] sramLowAddr, sramHighAddr, sramUpLowAddess, sramUpHighAddess;
assign sramLowAddr = {memAddr[18:3], 2'd0};
assign sramHighAddr = sramLowAddr + 18'd1;
assign sramUpLowAddess = sramLowAddr + 18'd2;
assign sramUpHighAddess = sramLowAddr + 18'd3;

wire [17:0] sramLowAddrWrite, sramHighAddrWrite;
assign sramLowAddrWrite = {memAddr[18:2], 1'b0};
assign sramHighAddrWrite = sramLowAddrWrite + 18'd1;

always @(ps or wrEn or rdEn) begin
    case (ps)
        Idle: ns = (wrEn == 1'b1 || rdEn == 1'b1) ? DataLow : Idle;
        DataLow: ns = DataHigh;
        DataHigh: ns = DataUpLow;
        DataUpLow: ns = DataUpHigh;
        DataUpHigh: 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_WE_N = ~wrEn;
            if (rdEn) begin
                SRAM_ADDR = sramLowAddr;
                readData[15:0] <= SRAM_DQ;
            end
            else if (wrEn) begin
                SRAM_ADDR = sramLowAddrWrite;
                dq = writeData[15:0];
            end
        end
        DataHigh: begin
            SRAM_WE_N = ~wrEn;
            if (rdEn) begin
                SRAM_ADDR = sramHighAddr;
                readData[31:16] <= SRAM_DQ;
            end
            else if (wrEn) begin
                SRAM_ADDR = sramHighAddrWrite;
                dq = writeData[31:16];
            end
        end
        DataUpLow: begin
            SRAM_WE_N = 1'b1;
            if (rdEn) begin
                SRAM_ADDR = sramUpLowAddess;
                readData[47:32] <= SRAM_DQ;
            end
        end
        DataUpHigh: begin
            SRAM_WE_N = 1'b1;
            if (rdEn) begin
                SRAM_ADDR = sramUpHighAddess;
                readData[63:48] <= SRAM_DQ;
            end
        end
        Done: ready = 1'b1;
    endcase
end

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

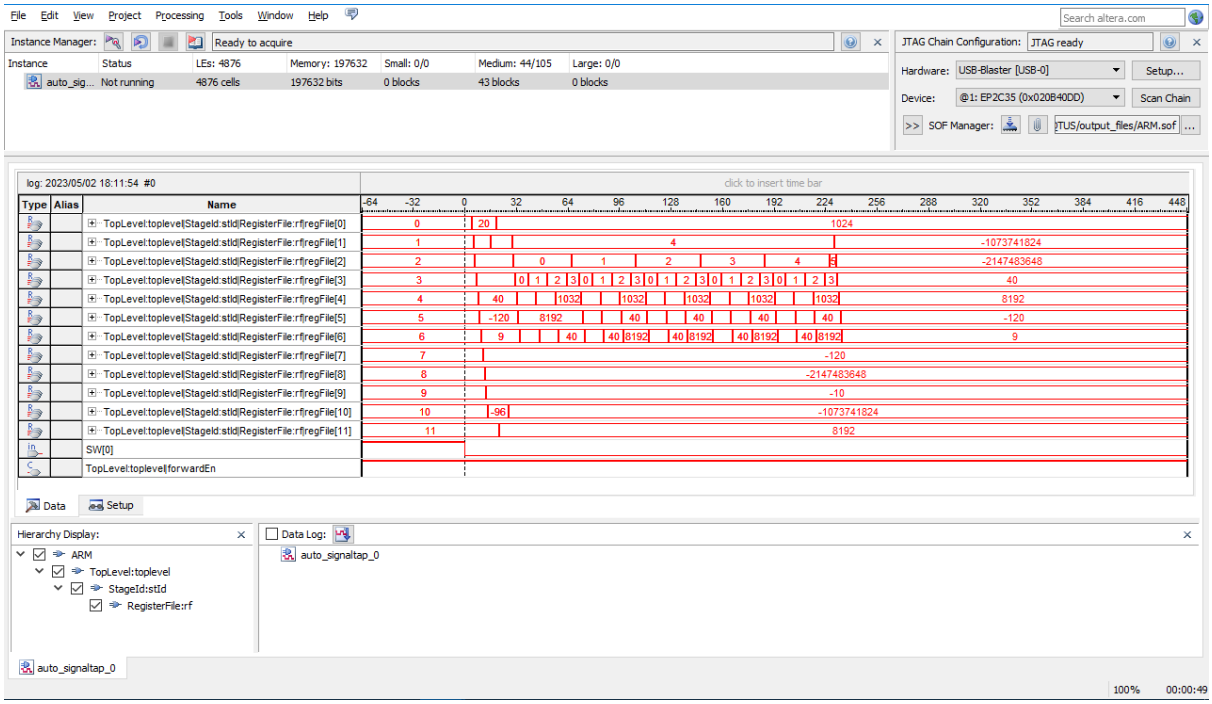
کد بخش Cache Controller نیز به صورت زیر است:

```
module CacheController(
    input clk, rst,
    input rdEn, wrEn,
    input [31:0] address,
    input [31:0] writeData,
    output [31:0] readData,
    output ready,
    // Sram Controller
    input sramReady,
    input [63:0] sramReadData,
    output sramWrEn, sramRdEn
);
    // Cache
    reg [31:0] way0First  [0:63];
    reg [31:0] way0Second [0:63];
    reg [31:0] way1First  [0:63];
    reg [31:0] way1Second [0:63];
    reg [9:0] way0Tag [0:63];
    reg [9:0] way1Tag [0:63];
    reg [63:0] way0Valid;
    reg [63:0] way1Valid;
    reg [63:0] indexLru;

    // Address Decode
    wire [2:0] offset;
    wire [5:0] index;
    wire [9:0] tag;
    assign offset = address[2:0];
    assign index = address[8:3];
    assign tag = address[18:9];

    // Way Decode
    wire [31:0] dataWay0, dataWay1;
    wire [9:0] tagWay0, tagWay1;
    wire validWay0, validWay1;
    assign dataWay0 = (offset[2] == 1'b0) ? way0First[index] : way0Second[index];
    assign dataWay1 = (offset[2] == 1'b0) ? way1First[index] : way1Second[index];
    assign tagWay0 = way0Tag[index];
    assign tagWay1 = way1Tag[index];
    assign validWay0 = way0Valid[index];
    assign validWay1 = way1Valid[index];

    // Hit Controller
    wire hit;
    wire hitWay0, hitWay1;
    assign hitWay0 = (tagWay0 == tag && validWay0 == 1'b1);
    assign hitWay1 = (tagWay1 == tag && validWay1 == 1'b1);
    assign hit = hitWay0 | hitWay1;

    // Data Controller
    wire [31:0] data;
    wire [31:0] readDataQ;
    assign data = hitWay0 ? dataWay0 :
                  hitWay1 ? dataWay1 : 32'dz;
    assign readDataQ = hit ? data :
                    sramReady ? (offset[2] == 1'b0 ? sramReadData[31:0] : sramReadData[63:32]) : 32'bz;
    assign readData = rdEn ? readDataQ : 32'bz;
    assign ready = sramReady;

    // Sram Controller
    assign sramRdEn = ~hit & rdEn;
    assign sramWrEn = wrEn;

    always @(posedge clk) begin
        if (wrEn) begin
            if (hitWay0) begin
                way0Valid[index] = 1'b0;
                indexLru[index] = 1'b1;
            end
            else if (hitWay1) begin
                way1Valid[index] = 1'b0;
                indexLru[index] = 1'b0;
            end
        end
    end
```

```
    always @(posedge clk) begin
        if (rdEn) begin
            if (hit) begin
                indexLru[index] = hitWay1;
            end
            else begin
                if (sramReady) begin
                    if (indexLru[index] == 1'b1) begin
                        {way0Second[index], way0First[index]} = sramReadData;
                        way0Valid[index] = 1'b1;
                        way0Tag[index] = tag;
                        indexLru[index] = 1'b0;
                    end
                    else begin
                        {way1Second[index], way1First[index]} = sramReadData;
                        way1Valid[index] = 1'b1;
                        way1Tag[index] = tag;
                        indexLru[index] = 1'b1;
                    end
                end
            end
        end
    end

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            way0Valid = 64'd0;
            way1Valid = 64'd0;
            indexLru = 64'd0;
        end
    end
endmodule
```

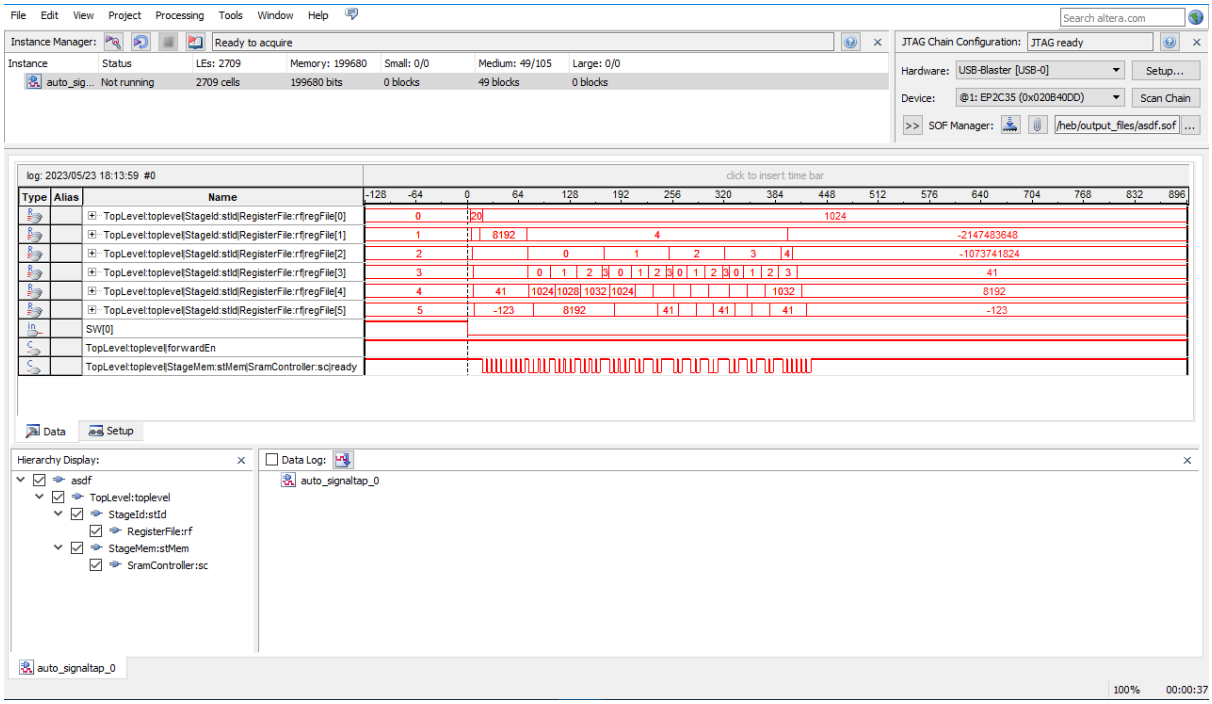
نتایج اجرا

حالت حافظه درون پردازنده



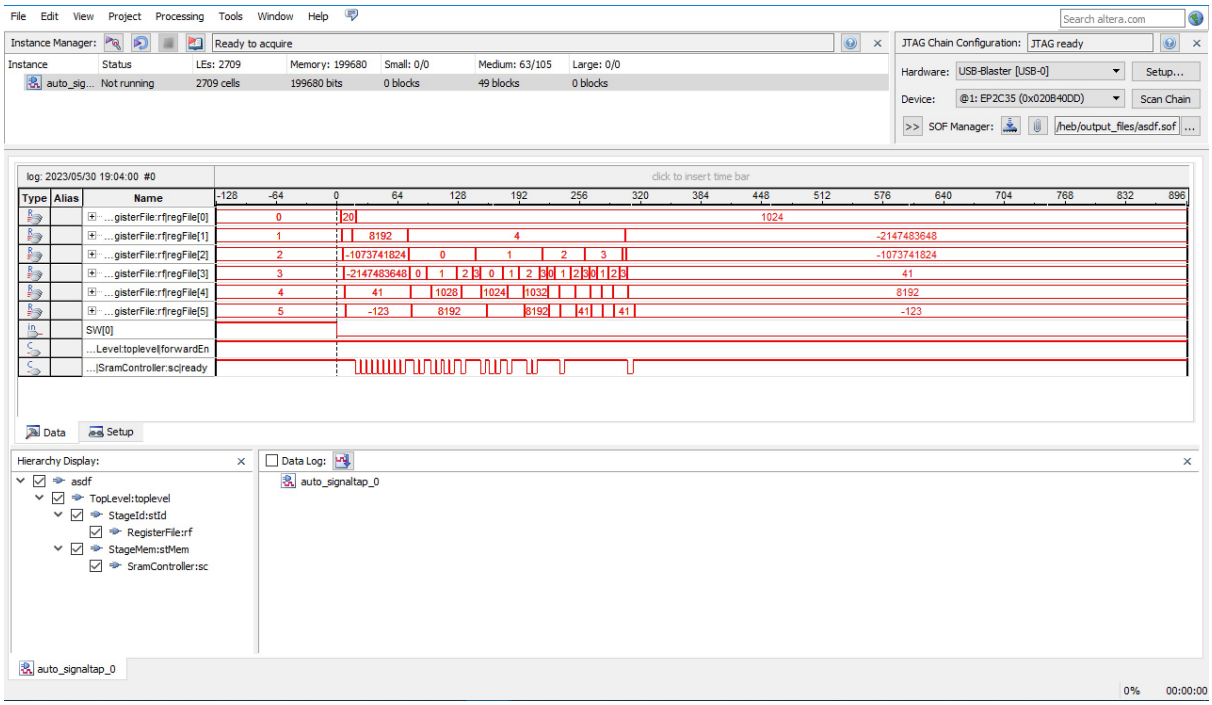
در این حالت اجرای برنامه حدود 230 کلاک نیاز داشته است.

حالت استفاده از SRAM



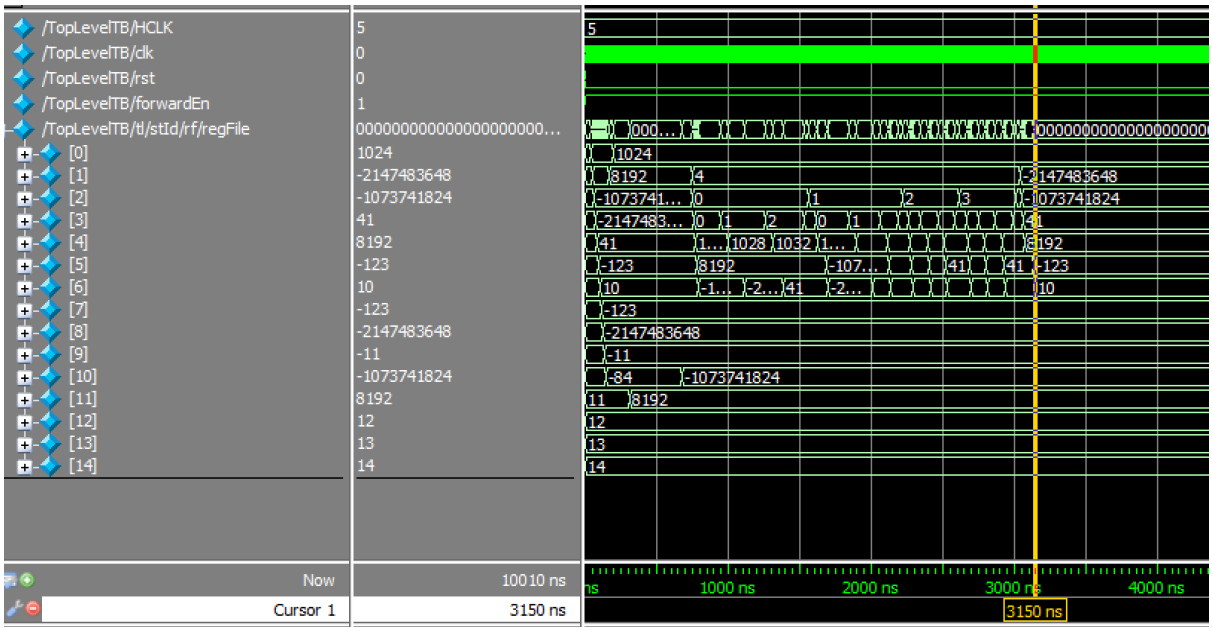
همانطور که مشاهده می‌شود، در این حالت اجرای برنامه حدود 430 کلاک زمان نیاز داشته است. در نتیجه در کل می‌توان گفت کارایی حدود 47 درصد نسبت به حالت قبل کاهش داشته است.

حالت استفاده از Cache



در این حالت نیز اجرای برنامه به حدود 320 کلاک نیاز دارد. در نتیجه نسبت به حالت قبل شاهد 34 درصد افزایش کارایی هستیم. اما همچنان نسبت به حالتی که حافظه در خود پردازنده قرار داشت، کارایی به میزان 29 درصد کمتر است.

اجرا در ModelSim:



نتایج سنتز

حالت حافظه درون پردازنده

Flow Summary	
Flow Status	Successful - Sat May 06 00:42:10 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	ARM
Top-level Entity Name	ARM
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	9,564 / 33,216 (29 %)
Total combinational functions	3,276 / 33,216 (10 %)
Dedicated logic registers	9,086 / 33,216 (27 %)
Total registers	9086
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	263,680 / 483,840 (54 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

همانطور که مشاهده می‌شود، در این حالت 9564 المان منطقی استفاده شده است.

حالت استفاده از SRAM

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 این حالت تعداد 5652 المان منطقی استفاده شده که نشان می‌دهد نسبت به حالت قبل، حدود 41 درصد المان کمتری استفاده شده است.

حالت استفاده از Cache

Flow Summary	
Flow Status	Successful - Tue May 30 19:02:53 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	6,463 / 33,216 (19 %)
Total combinational functions	4,493 / 33,216 (14 %)
Dedicated logic registers	3,972 / 33,216 (12 %)
Total registers	3972
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	209,152 / 483,840 (43 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

در این حالت نیز 6463 المان استفاده شده که نسبت به حالت قبل، شاهد افزایش 14 درصدی استفاده از المان‌ها هستیم. اما همچنان نسبت به حالت اول، حدود 33 درصد المان کمتری استفاده شده است.