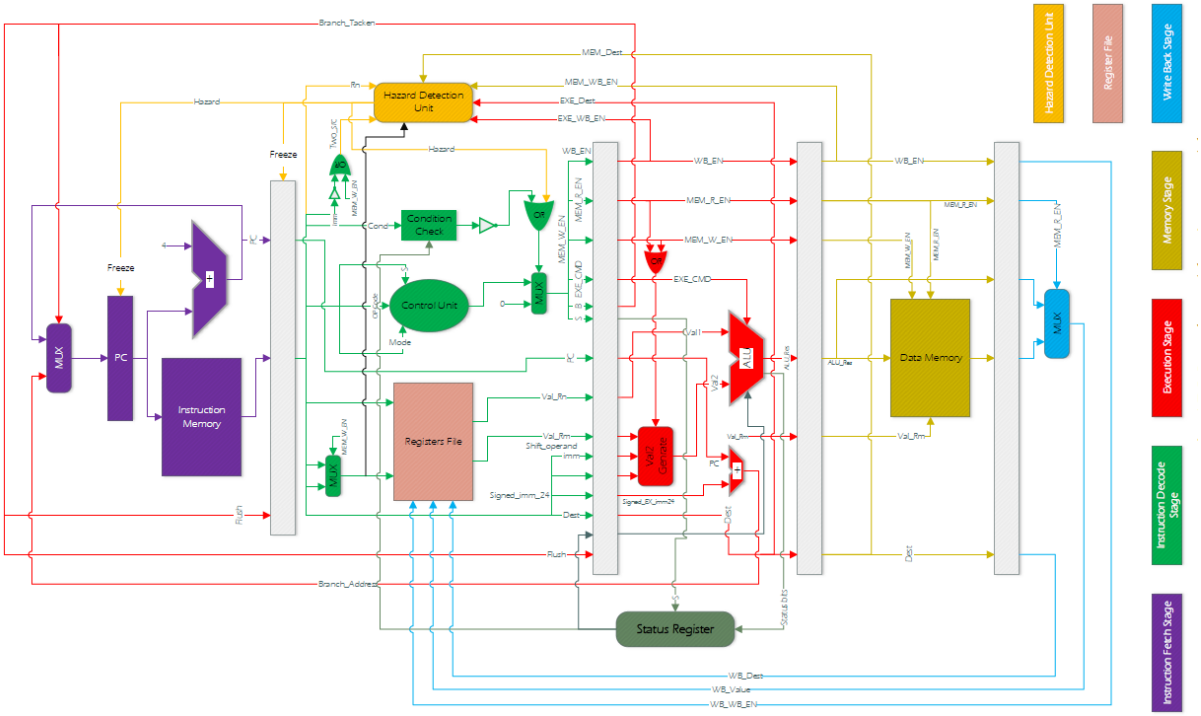
توضیحات آزمایش

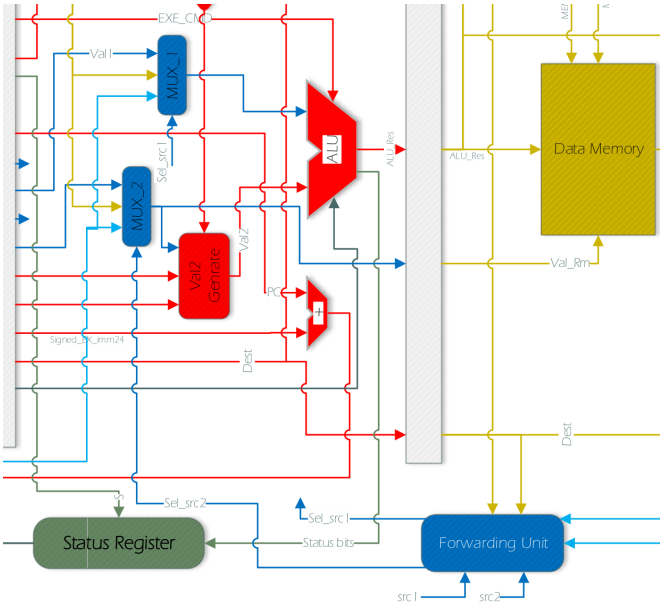
در این آزمایش پردازنده ARM به صورت پایپ‌لاین پیاده‌سازی می‌شود. دیاگرام این پردازنده به صورت زیر است:



این پردازنده دارای 13 دستور اصلی است. پیاده‌سازی باید در زبان وریلاگ باشد و در نهایت پس از شبیه‌سازی در نرم‌افزار ModelSim، با استفاده از نرم‌افزار Quartus سنتز می‌شود و روی FPGA قرار می‌گیرد. سپس، با استفاده از یک تست‌بنچ، پردازنده پیاده‌سازی شده تست می‌شود. از اهداف این آزمایش می‌توان به یادگیری نحوه عیب‌یابی و تست مدارهای سخت‌افزاری طراحی شده اشاره کرد.

آزمایش دوم

در این جلسه تکنیک ارسال به جلو پیاده‌سازی می‌شود و میزان بهبود کارایی ارزیابی می‌شود:



ماژول Forwarding Unit

این ماژول ورودی forwardEn را گرفته که مشخص می‌کند که ماژول قرار است کار کند یا خیر. در صورت خاموش بودن سیگنال، پردازنده همانند وقتی که انتقال به جلو نداشت کار می‌کند.
در صورت نبود این ماژول، همه تداخل‌ها توسط hazard unit هندل می‌شدند. یعنی مثلا اگر دو دستور جمع پشت سر هم داشتیم که دومی از رجیستری استفاده می‌کرد که اولی تغییر می‌داد، این مورد توسط hazard unit تشخیص داده می‌شد و پایپ‌لاین دو کلاک استال می‌شد.
کنون می‌خواهیم به جای استال پایپ‌لاین، داده درست را مستقیم از مراحل جلوتر پایپ‌لاین به قبلی‌ها ببریم.
این ماژول src1 و src2 دیکد شده دستور که از دیوار رجیستر به مرحله EX آمده را می‌گیرد. این ماژول همچنین از مراحل MEM و WB دو سیگنال wbEn و dest را می‌گیرد.
حال در صورت روشن بودن wbEn در MEM یا WB (با اولویت به MEM)، بررسی می‌کنیم که src1 یا src2 با dest برابر است یا خیر. در صورت برقرار بودن شرایط، یعنی دستور قبلی می‌خواهد در همان src که دستور کنونی می‌خواهد بخواند مقدار جدید بریزد و باید فروارد کنیم.
خروجی این ماژول دو سیگنال selSrc1 و selSrc2 است که به دو Mux در مرحله EX می‌رود و بین انتخاب مقدار عادی از مرحله ID، مقدار در مرحله MEM و مقدار در مرحله WB تصمیم می‌گیرد.
کد این ماژول در ادامه آورده شده است. برای هر کدام از Muxها یک بلاک always نوشته شده که مشابه هم هستند و به موازات هم کار می‌کنند:

```
module ForwardingUnit(
  input forwardEn,
  input [3:0] src1, src2,
  input wbEnMem, wbEnWb,
  input [3:0] destMem, destWb,
  output reg [1:0] selSrc1, selSrc2
);
  always @(forwardEn, src1, wbEnMem, wbEnWb, destMem, destWb) begin
    selSrc1 = 2'b00;
    if (forwardEn) begin
      if (wbEnMem && (destMem == src1)) begin
        selSrc1 = 2'b01;
      end
      else if (wbEnWb && (destWb == src1)) begin
        selSrc1 = 2'b10;
      end
    end
  end

  always @(forwardEn, src2, wbEnMem, wbEnWb, destMem, destWb) begin
    selSrc2 = 2'b00;
    if (forwardEn) begin
      if (wbEnMem && (destMem == src2)) begin
        selSrc2 = 2'b01;
      end
      else if (wbEnWb && (destWb == src2)) begin
        selSrc2 = 2'b10;
      end
    end
  end
endmodule
```

Multiplexerها

دو عدد Mux در مرحله EX اضافه شده‌اند که ورودی‌های ALU را مشخص می‌کنند.
این دو Mux سه ورودی دارند که به ترتیب مقدار رجیستر از مرحله پشتی ID، مقدار آپدیت شده رجیستر از مرحله MEM (یعنی خروجی ALU که به مرحله بعد رفته) و مقدار آن در مرحله WB است.
سیگنال دو بیتی selSrc1/2 که از forwarding unit می‌آید خروجی این muxها را تنظیم می‌کند.
در صورت غیرفعال بودن forwarding، همیشه مقدار مرحله ID انتخاب می‌شود.

Hazard Unit

این ماژول در صورت نبود forwarding، همه تداخل‌ها را تشخیص داده و با استال کردن پایپ‌لاین آنها را درست می‌کند. در صورت فعال بودن forwarding، باید فقط حالاتی را هندل کند که انتقال به جلو برای آنها ممکن نیست.
تنها دستوری که در پردازنده پیاده‌سازی شده این شرایط را دارد دستور LDR است که باید مقداری از data memory بگیرد و آن را به رجیستر برگرداند. از آنجا که پس از خواندن مقدار مموری مقدار آن پایدار نیست (چون خروجی combinational است) نمی‌توان آن را به مراحل قبل فروارد کرد.
بنابراین کد hazard unit به شکل مقابل تغییر داده شده است:

```
module HazardUnit(
    input [3:0] rn, rdm,
    input twoSrc,
    input [3:0] destEx, destMem,
    input wbEnEx, wbEnMem, memREn,
    input forwardEn,
    output reg hazard
);
    always @(rn, rdm, destEx, destMem, wbEnEx, wbEnMem,
             memREn, twoSrc, forwardEn) begin
        hazard = 1'b0;
        if (forwardEn) begin
            if (memREn) begin
                if (rn == destEx || (twoSrc && rdm == destEx)) begin
                    hazard = 1'b1;
                end
            end
        end
        else begin
            if (wbEnEx) begin
                if (rn == destEx || (twoSrc && rdm == destEx)) begin
                    hazard = 1'b1;
                end
            end
            if (wbEnMem) begin
                if (rn == destMem || (twoSrc && rdm == destMem)) begin
                    hazard = 1'b1;
                end
            end
        end
    end
endmodule
```

ماژول TopLevel

ماژول Forwarding Unit و سیگنال forwardEn به تاپ‌لول افزوده شده که به Hazard Unit هم وارد می‌شود و بود و نبود انتقال به جلو در پردازنده را تعیین می‌کند.

```
module TopLevel(
    input clk, rst,
    input forwardEn
);
```

در ماژول ARM هم از یکی دیگر از سوییچ‌های FPGA برای کنترل forwardEn استفاده شده است:

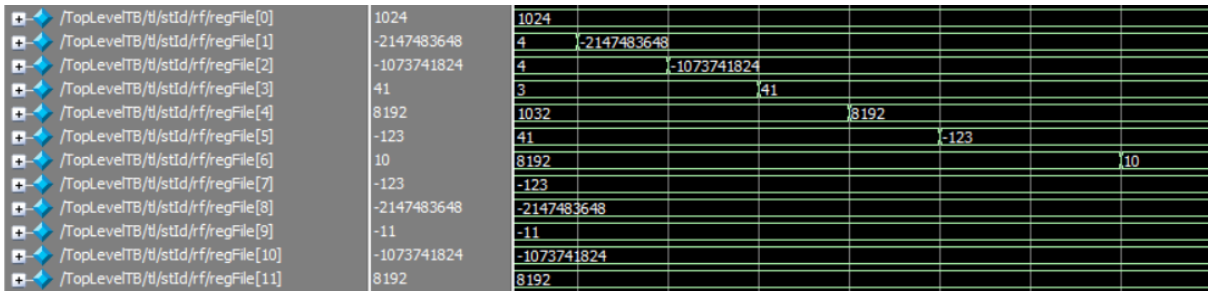
```
TopLevel toplevel(
    .clk(CLOCK_50),
    .rst(SW[0]),
    .forwardEn(SW[1])
);
```

بخش امتیازی

از راه‌های دیگر افزایش کارایی پردازنده، پیاده‌سازی سیستم Branch Prediction و استفاده از Cache است. برای بهبود اجرای برنامه‌ها، می‌توان دستورات بهینه‌ای مانند SIMDها را در پردازنده پیاده‌سازی کرد. می‌توان با اجرای نامنظم دستورات (out of order execution) نیز جلوی استال‌ها را گرفت و utilization را بالا برد.

تست پردازنده

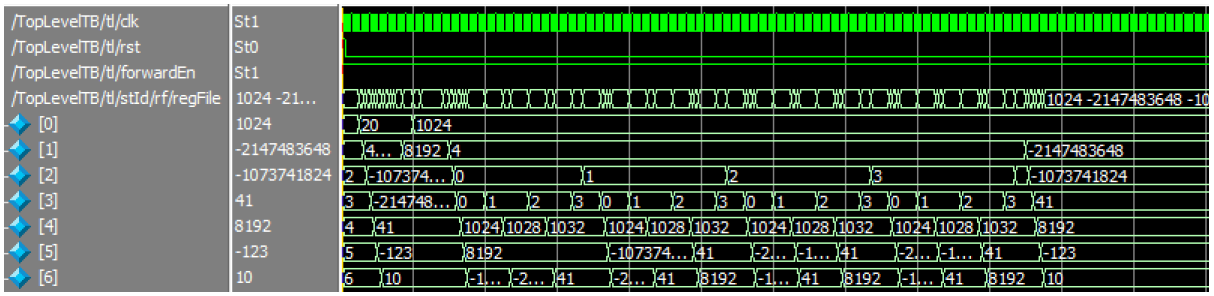
تمامی دستورات محک در حافظه دستورات پردازنده قرار گرفته‌اند تا پردازنده به طور کامل تست شود. هدف نهایی این برنامه این است که عملیات Bubble Sort بر روی رجیسترهای R1 تا R4 صورت بپذیرد.
در گزارش قبلی خروجی تمامی دستورات بررسی شده و اینجا فقط به نتیجه نهایی برنامه می‌پردازیم که در گزارش قبل به صورت زیر بوده و R1 تا R4 سورت شده اند:



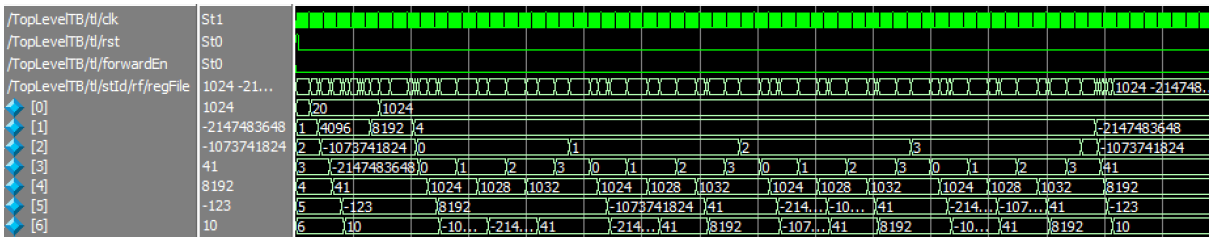
کنون خروجی را در دو حالت forwardEn = 1'b0 و forwardEn = 1'b1 در تست بنچ زیر نگاه می‌کنیم:

```
`timescale 1ns/1ns
module TopLevelTB();
    localparam HCLK = 5;
    reg clk, rst, forwardEn;
    TopLevel tl(clk, rst, forwardEn);
    always #HCLK clk = ~clk;
    initial begin
        {clk, rst, forwardEn} = 3'b011;
        #10 rst = 1'b0;
        #3000 $stop;
    end
endmodule
```

خروجی با forwardEn = 1’b1:



خروجی با forwardEn = 1’b0:



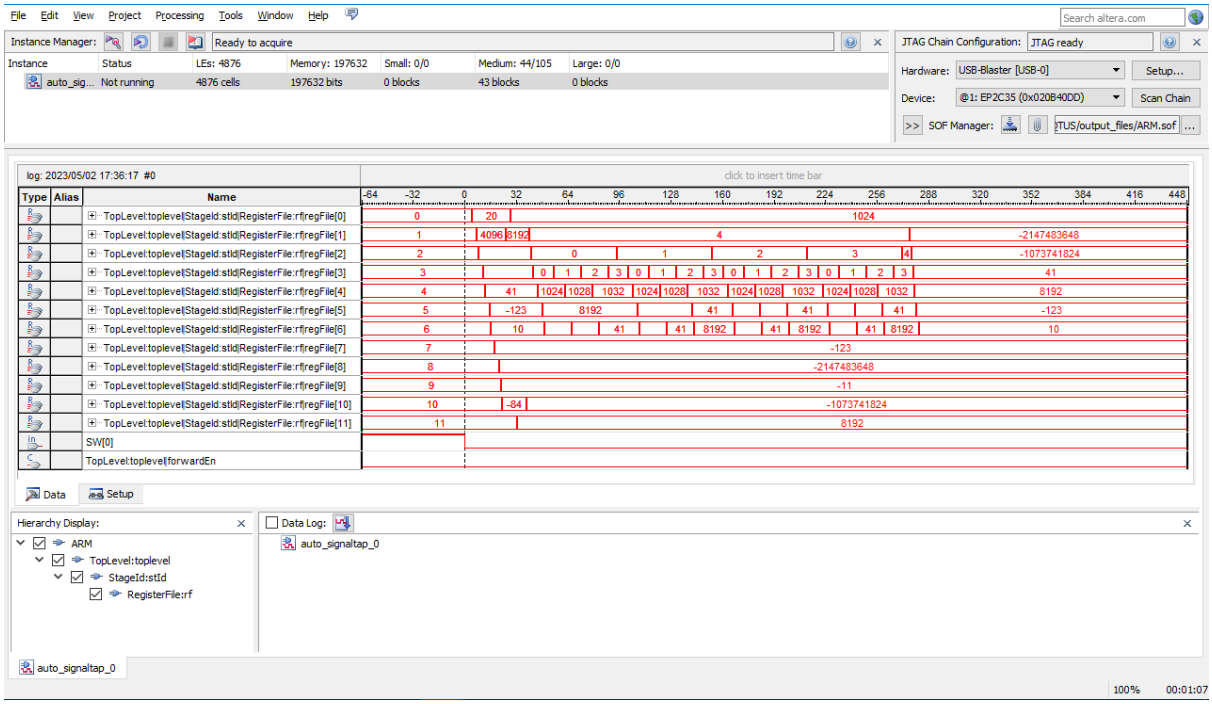
همانطور که می‌بینیم، نتیجه نهایی هر دو حالت یکسان است و R1 تا R4 سورت شده اند.

نتایج سنتز

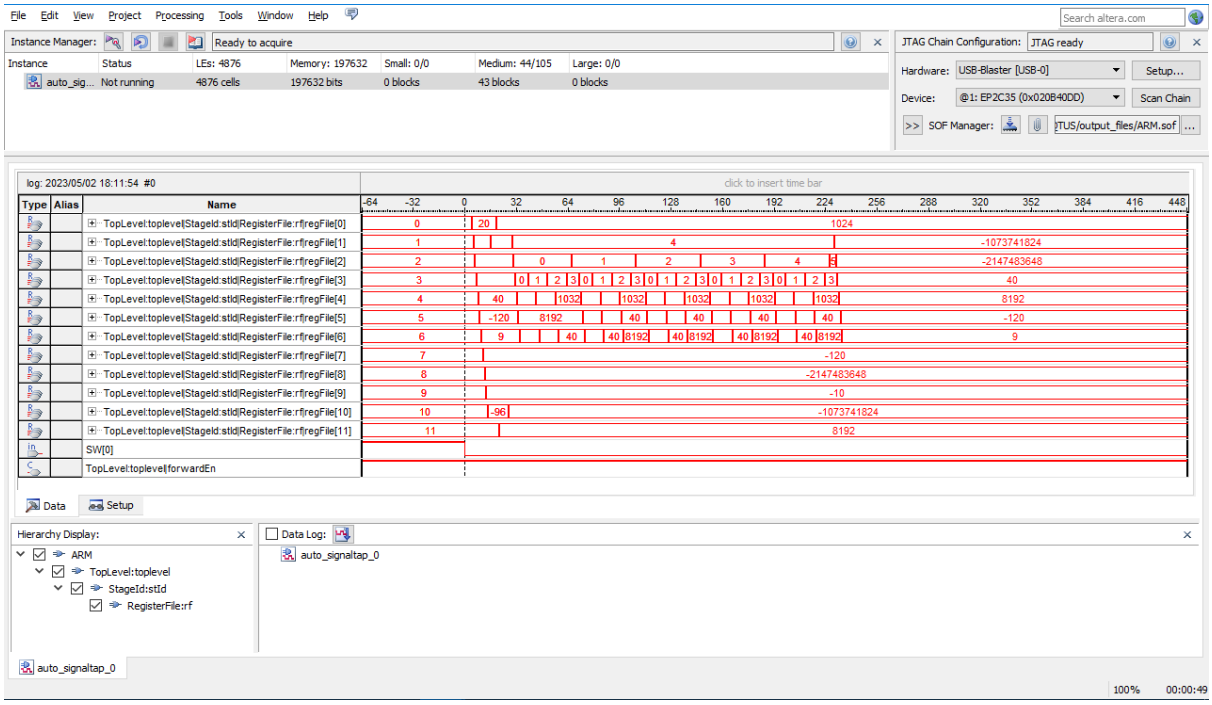
Flow Summary	
Flow Status	Successful - Sat May 06 00:42:10 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	ARM
Top-level Entity Name	ARM
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	9,564 / 33,216 (29 %)
Total combinational 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 %)

خروجی برنامه محک در SignalTap

خروجی با forwardEn = 1’b0:



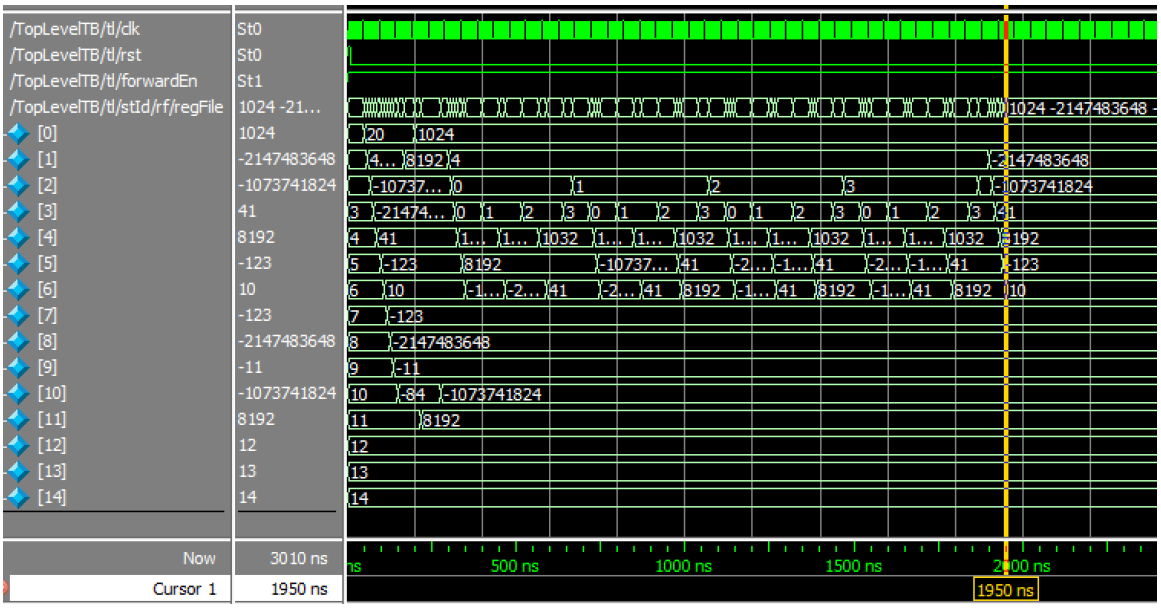
خروجی با forwardEn = 1’b1:



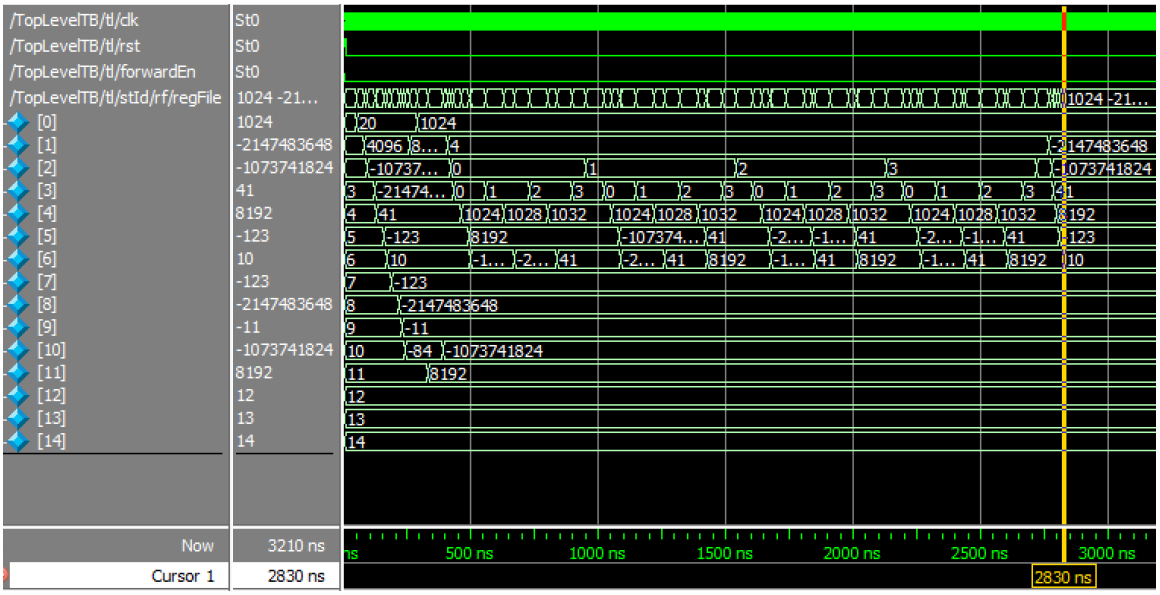
محاسبه کارایی

برای محاسبه کارایی وجود انتقال به جلو، زمان پایان یافتن برنامه محک (که آخرین آپدیت رجیسترفایل در نظر گرفته شده) را در دو حالت بود و نبود فروواردینگ نگاه می‌کنیم:

با فروارد:



بدون فروارد:



همانطور که مشاهده می‌کنیم، در حالت فروارد 1950ns طول کشیده و در حالت نبود فروارد 2830ns.

از آنجا که در تست‌بنچ هر کلاک 10ns در نظر گرفته شده، این یعنی برنامه در حالت فروارد‌دینگ 195 کلاک و در نبود فروارد‌دینگ 283 کلاک طور انجامیده است.

$$\frac{195 - 283}{283} = -0.311 = 31.1\%\ improvement$$

برای بررسی سخت‌افزاری، مقدار المان‌های منطقی استفاده شده در سنتز بر روی FPGA را در نظر می‌گیریم. در حالتی که forwarding unit و muxهای مرحله EX و چیزهای مرتبط آن وجود نداشت (گزارش قبل) این نتیجه را داشتیم:

Flow Summary	
Flow Status	Successful - Thu Apr 20 23:10:01 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	7,753 / 33,216 (23 %)
Total combinational functions	4,066 / 33,216 (12 %)
Dedicated logic registers	5,853 / 33,216 (18 %)
Total registers	5853
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	396,288 / 483,840 (82 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

که یعنی 7753 المان استفاده شده است. در بخش نتایج سنتز دیدیم که در حالت کنونی 9564 المان استفاده کرده ایم. یعنی 1811 المان بیشتر شده است.

$$\frac{1811}{7753} = 0.233 = 23.3\%\ more\ elements$$