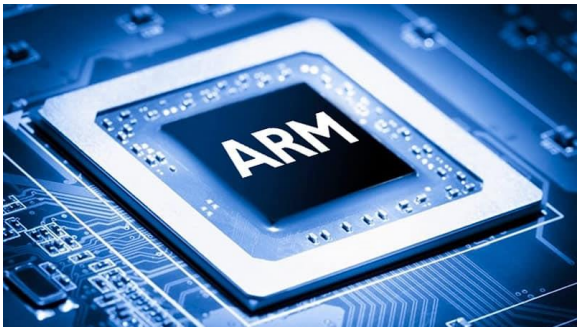
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه معماری کامپیوتر

گزارش دستور کار شماره ۶

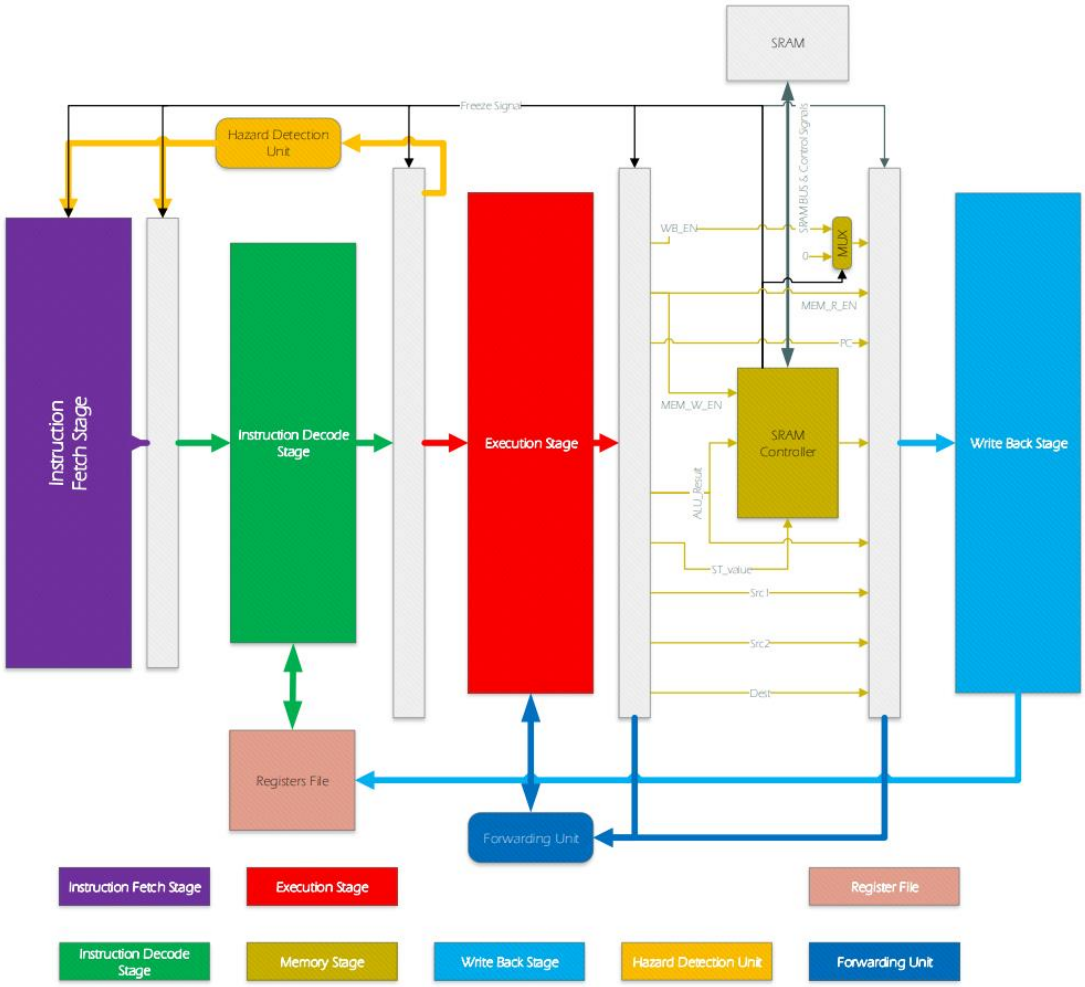
علی پادیاو
۸۱۰۱۹۹۳۸۸
محمد صالح عرفاتی
۸۱۰۱۹۷۵۴۳

بهار۱۴۰۲

مقدمه

در آزمایشات قبلی مراحل IF و ID و EXE و WB و MEM و ماژول Hazard از معماری ARM9 پیاده سازی شد و در انتها ماژول Forwarding Unit به معماری ARM خود اضافه نمودیم.

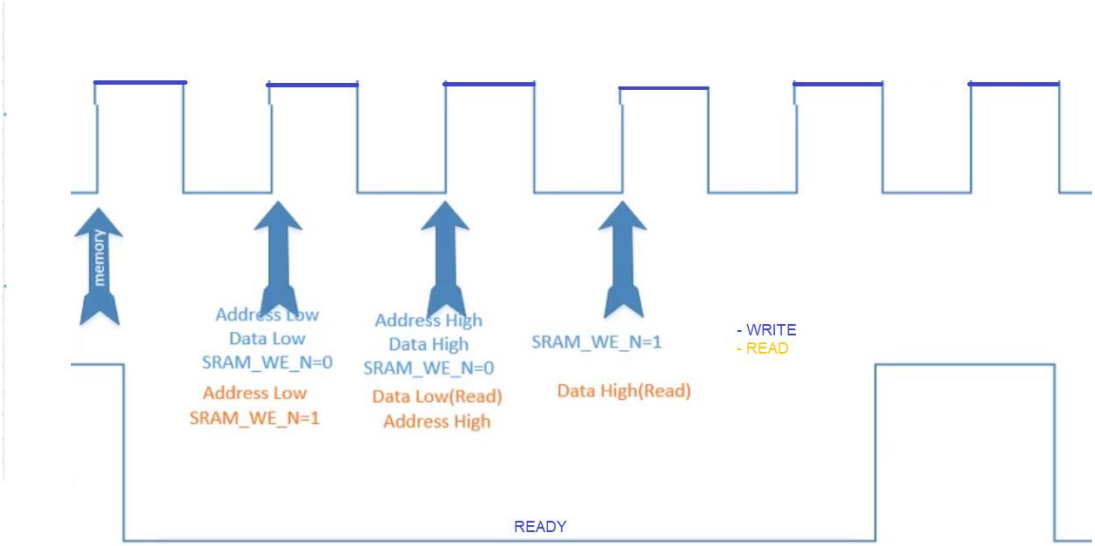
در دنیای FPGA حافظه و موری خیلی ارزشمند است و کمیابق می‌باشد و به همین دلیل از موری RAM جداگانه استفاده می‌کنیم. در معماری ARM ما ماژول memory کاملا از منابع داخلی FPGA استفاده شده بود به همین دلیل این ماژول کامل جداکرده تا در بورد مربوطه از SRAM موجود در دولوپمنت بورد مربوطه برای موری استفاده گردد. بلوک دیاگرام این مرحله به صورت زیر می‌باشد:



شکل ۱

همانطور که در بلوک دیاگرام نیز مشاهده می‌شود به جای ماژول memory داخل FPGA از ماژول SRAM CONTROLLER استفاده می‌شود تا ماژول SRAM که در خارج از FPGA موجود می‌باشد کنترل کنیم.

قابل ذکر است بر این سیگنال FREEZE نیز به Stageها و Regهای مربوطه ارسال میگردد تا هنگام کار با مموری دستوراتی که در حال اجرا هستند متوقف شوند تا SRAM CONTROLLER دیتای مورد نظر را آماده کند(چه نوشتن دیتا و چه خواندن آن).



شکل ۲

به شکل ۲ دقت کنید، عملیات خواندن و نوشتن در ماژول SRAM را به سادگی نشان می‌دهد. که هر دوی این عملیات در پنج کلاک صورت میگیرد(هنگامی‌که کار مربوطه در حال انجام است سیگنال FREEZE ارسال می‌گردد) دقت شود که عملیا نوشتن و خواندن در SRAM، ۱۶ بیتی می‌باشد پس دیتای خود را باید به دو قسمت ۱۶ بیتی تقسیم کرده و اول دیتای کم ارزش و سپس دیتای پرارزش نوشته شود:

عملیات نوشتن:

- سیکل ۱: در سیکل اول، منتظر آنیم که دیتای مورد نظر به درستی شکل بگیرد و سپس سیگنال READY برابر با صفر می‌شود
- سیکل۲: در سیکل۲، آدرس ۱۶ بیت کم ارزش فرستاده شده و سپس دیتای کم ارزش بر روی باس مربوطه گذاشته می‌شود و سیگنال SRAM_WE_N برابر با صفر میشود تا دیتا در ماژول SRAM نوشته شود.

- سیکل۳: در سیکل۳، آدرس ۱۶ بیت پر ارزش فرستاده شده و سپس دیتای پر ارزش بر روی باس مربوطه گذاشته می‌شود و سیگنال SRAM_WE_N برابر با صفر میشود تا دیتا در ماژول SRAM نوشته شود.
- سیکل ۴: در این سیکل، سیگنال SRAM_WE_N را برابر با یک قرار می‌دهیم تا دیتا به اشتباه در آدرس موردنظر ننویسد.
- سیکل ۵: در سیکل ،۵ سیگنال READY برابر یک میشود تا نشان دهد دیتای مورد نظر در ماژول SRAM نوشته شده است و این عملیات تکمیل شده است.

عملیات خواندن:

- سیکل ۱: در سیکل اول، منتظر آنیم که دیتای مورد نظر به درستی شکل بگیرد و سپس سیگنال READY برابر با صفر می‌شود
- سیکل۲: در سیکل۲، آدرس ۱۶ بیت کم ارزش فرستاده شده و سیگنال SRAM_WE_N برابر با یک میشود تا دیتا از ماژول SRAM خوانده شود و در باس مربوطه توسط SRAM نوشته شود تا در سیکل بعد بتوانیم آن را بخوانیم.
- سیکل۳: در سیکل۳، دیتای ۱۶ بیت کم ارزش خوانده شده و سپس آدرس ۱۶ بیت پر ارزش فرستاده شده و سیگنال SRAM_WE_N برابر با یک میشود تا دیتای پرارزش از ماژول SRAM خوانده شود و در باس مربوطه توسط SRAM نوشته شود تا در سیکل بعد بتوانیم آن را بخوانیم.
- سیکل ۴: در این سیکل، دیتای پرارزش از باس مربوطه خوانده میشود.
- سیکل ۵: در سیکل ،۵ سیگنال READY برابر یک میشود تا نشان دهد دیتای مورد نظر از ماژول SRAM خوانده شده است و این عملیات تکمیل شده است.

شرح ماژول‌های اضافه شده:

1-SRAM CONTROLLER

کد:

```
module SramController(
    input clk, rst,
    input wr_en, rd_en,
    input[31:0] address,
    input [31:0] writeData,

    output[31:0] readData,

    output reg ready,

    inout [15:0]SRAM_DQ,
    output [17:0]SRAM_ADDR,
    output reg SRAM_UB_N,
    output reg SRAM_LB_N,
    output reg SRAM_WE_N,
    output reg SRAM_CE_N,
    output reg SRAM_OE_N
  );
```

شکل ۳

شکل ۳، اینترفیس ماژول مخربوطه را نشان میدهد دقت شود که SRAM_DQ از نوع باس INOUT تعریف شده است زیرا دیتا میتواند هم از طریق SRAM و هم از طریق SRAM_CONTROLLER در این باس نوشته شود و هر کدام از این ماژول‌ها که بخواهد دیتا بخواند سیگنال Z (که به معنای Float بودن است) را در باس مربوطه ست میکند تا ماژول دیگر بتواند دیتای خود را بنویسد.

```
reg [2:0] ps, ns;
reg [17:0]addr;
reg [15:0]W_D_O_16;
reg [31:0]R_D_32;

parameter [2 : 0] IDLE = 3'b0;
parameter [2 : 0] WRITE_1 = 3'b001;
parameter [2 : 0] WRITE_2 = 3'b010;
parameter [2 : 0] WRITE_END = 3'b101;
parameter [2 : 0] READ_1 = 3'b001;
parameter [2 : 0] READ_2 = 3'b010;
parameter [2 : 0] READ_3 = 3'b011;
parameter [2 : 0] READ_END = 3'b101;


assign SRAM_DQ = (SRAM_WE_N == 0) ? W_D_O_16 : 16'bzzzzzzzzzzzzzzzz;
assign SRAM_ADDR = addr;
assign readData = R_D_32;

```

شکل ۴

در شکل ۴، PS بیان کننده استیت فعلی و NS بیان کننده استیت آینده می‌باشد(نقش شمارنده برای سیکل‌های موردنظر را ایفا میکنند) و سپس در چند خط بعدی استیت‌ها تعریف شده اند. (مانند IDLE, WRITE_1,...)

در خط ۳۵ بررسی میگردد که آیا دیتا میخواهد برروی باس نوشته و یا خوانده شود اگر میخواهیم بخوانیم باید در باس موردنظر سیگنال Z را بنویسم و در صورت نوشتن در SRAM باید دیتای خودمان را روی باس بگذاریم.

```
39    always@(posedge clk,posedge rst)
40    begin
41      if(rst)
42        ps <= IDLE;
43      else
44        ps <= ns;
45    end
```

شکل ۵

در این بلاک always، استیت بعدی در استیت فعلی نوشته میشود و در شکل ۶ استیت بعدی به دست می‌آید. اگر که کارتمام شده باشد استیت بعدی برابر با IDLE میشود و در غیر این صورت اگر سیگنال rd_en و wr_en یک باشد (که نشان میدهد کاربر میخواهد در مموری بنویسد یا از آن بخواند)، استیت مورد نظر یکی جلوتر می‌رود.

```
47    always@(ps,wr_en,rd_en)
48    begin
49      if(ps == WRITE_END || ps == READ_END)
50        ns = IDLE;
51      else if(wr_en|rd_en)
52        ns = ps + 1;
53      else
54        ns = IDLE;
55    end
```

شکل ۶

```
always@(ps,wr_en,rd_en,writeData)
begin
  {SRAM_WE_N, SRAM_UB_N, SRAM_LB_N, SRAM_CE_N, SRAM_OE_N, ready} = 6'b100000;
  if(wr_en)
  case(ps)
    IDLE:
    begin
      ready=~(wr_en|rd_en);
    end
    WRITE_1:
    begin
      W_D_O_16 = writeData[15:0];
      addr = (address[17:0] - 32'd1024) >> 1;
      SRAM_WE_N = 1'b0;
    end
    WRITE_2:
    begin
      W_D_O_16 = writeData[31:16];
      addr = ((address[17:0] - 32'd1024) >> 1) + 1;
      SRAM_WE_N = 1'b0;
    end
    WRITE_END:
    begin
      ready=1'b1;
    end
  endcase
```

شکل ۷

شکل۷، استیت‌های مورد نظر برای نوشتن را نشان میدهد:

- IDLE: در استیت IDLE، منتظر آنیم که دیتای مورد نظر به درستی شکل بگیرد و سپس سیگنال READY برابر با صفر می‌شود
- WRITE_1: در این استیت، آدرس ۱۶ بیت کم ارزش (۰ تا ۱۵) فرستاده شده و سپس دیتای کم ارزش بر روی باس مربوطه گذاشته می‌شود و سیگنال SRAM_WE_N برابر با صفر میشود تا دیتا در ماژول SRAM نوشته شود.
- WRITE_2: در این استیت، آدرس ۱۶ بیت پرارزش (۱۶ تا ۳۱) فرستاده شده و سپس دیتای پرارزش بر روی باس مربوطه گذاشته می‌شود و سیگنال SRAM_WE_N برابر با صفر میشود تا دیتا در ماژول SRAM نوشته شود.
- WRITE_END: در این سیکل، سیگنال SRAM_WE_N را برابر با یک قرار می‌دهیم تا دیتا به اشتباه در آدرس موردنظر ننویسد و سیگنال READY برابر یک میشود تا نشان دهد دیتای مورد نظر در ماژول SRAM نوشته شده است و این عملیات تکمیل شده است.

عملیات خواندن:

شکل ۸، استیت‌های مورد نظر برای خواندن را نشان میدهد:

- IDLE: در استیت IDLE، منتظر آنیم که دیتای مورد نظر به درستی شکل بگیرد و سپس سیگنال READY برابر با صفر می‌شود
- READ_1: در این استیت، آدرس ۱۶ بیت کم ارزش (۰ تا ۱۵) فرستاده شده تا در استیت بعد بتوانیم دیتای مورد نظر را از باس بخوانیم.
- READ_2: در این استیت، ابتدا ۱۶ بیت کم ارزش از باس خوانده میشود و سپس آدرس ۱۶ بیت پرارزش (۱۶ تا ۳۱) فرستاده شده است تا در استیت بعد آن را بخوانیم.
- READ_3: در این سیکل، دیتای پرارزش از باس مربوطه خوانده میشود.
- سیکل ۵: در این استیت، سیگنال READY برابر یک میشود تا نشان دهد دیتای مورد نظر از ماژول SRAM خوانده شده است و این عملیات تکمیل شده است.

```
83      else if(rd_en)
84      case(ps)
85        IDLE:
86        begin
87          ready=~(wr_en|rd_en);
88        end
89        READ_1:
90        begin
91          addr = (address[17:0] - 32'd1024) >> 1;
92        end
93        READ_2:
94        begin
95          R_D_32[15:0] = SRAM_DQ;
96          addr = ((address[17:0] - 32'd1024) >> 1) + 1;
97        end
98        READ_3:
99        begin
100         R_D_32[31:16] = SRAM_DQ;
101       end
102       READ_END:
103       begin
104         ready = 1'b1;
105       end
106     endcase
107     else
108     case(ps)
109       IDLE:
110       begin
111         ready = 1'b1;
112       end
113     endcase
114   end
```

شکل ۸

2- SRAM:

دقت شود که این ماژول برای SIMULATAION ایجاد شده است و شبیه سازی برای ماژول SRAM موجود در بورد مربوطه می‌باشد.
این ماژول ۱۶ بیت باس مربوط به دیتا دارد (که به SRAM_CONTROLLER نیز متصل می‌باشد) که از نوع INOUT می‌باشد و ۱۸ بیت، بیت آدرس دهی دارد.

کد:

```
module SRAM (
    input clk,
    input rst,
    input SRAM_WE_N,
    input SRAM_UB_N,
    input SRAM_LB_N,
    input SRAM_CE_N,
    input SRAM_OE_N,
    input [17:0] SRAM_ADDR,
    inout [15:0] SRAM_DQ
  );

  reg [15:0] memory[0:511];
  assign #5 SRAM_DQ = SRAM_WE_N ? memory[SRAM_ADDR] : 16'bz;
  always@(posedge clk)
  begin
    if(~SRAM_WE_N)
    begin
      memory[SRAM_ADDR] = SRAM_DQ;
    end
  end
endmodule
```

شکل ۹

3-SIGNAL FREEZE:

قابل ذکر است برای سیگنال فریز ماژولی اضافه نشده و به راحتی باستفاده از یک ماژول OR ساده پیاده سازی شده است.
سیگنال‌های ورودی FREEZE به STAGEها وREGهای مربوطه با NOT READY عملیات OR صورت میگرد تا سیگنال مربوطه تولید گردد. مانند شکل زیر، به سیگنال فریز مربوطه دقت کنید:

```
IF_Stage if_stage (
            .clk(clk),
            .rst(rst),
            .freeze(hazard | ~ready),
            .branch_taken(branch_taken),
            .branch_address(branchAddr),
            .PC(PC_IF),
            .instruction(inst_IF)
          );

IF_Reg if_reg (
          .clk(clk),
          .rst(rst),
          .freeze(hazard | ~ready),
          .flush(branch_taken),
          .PC_in(PC_IF),
          .instruction_in(inst_IF),
          .PC_out(PC_ID),
          .instruction_out(inst_ID)
        );
```

شکل ۱۰

دستورات برنامه:

طبق شرح آزمایش کل برنامه محک را داخل Ins_Mem ذخیره کردیم تا دستورات یکی پس از دیگری اجرا گردند دقت شود که وظیفه این برنامه مرتب سازی رجیسترهای R4 R3, R2, R1, به صورت صعودی می‌باشد.

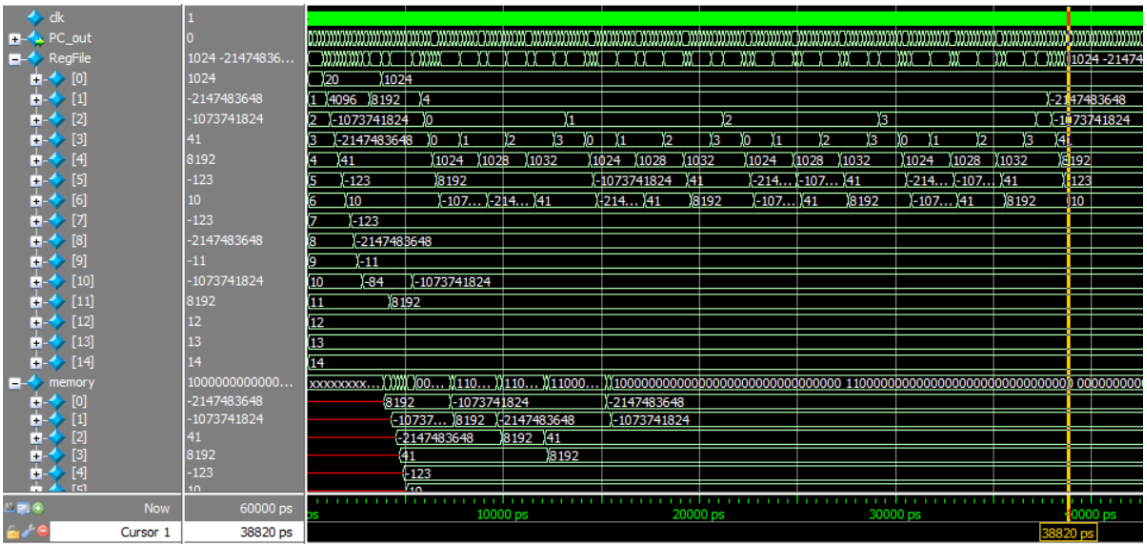
SIMULATION:

برای simulation از تست بنچ زیر استفاده شده است:

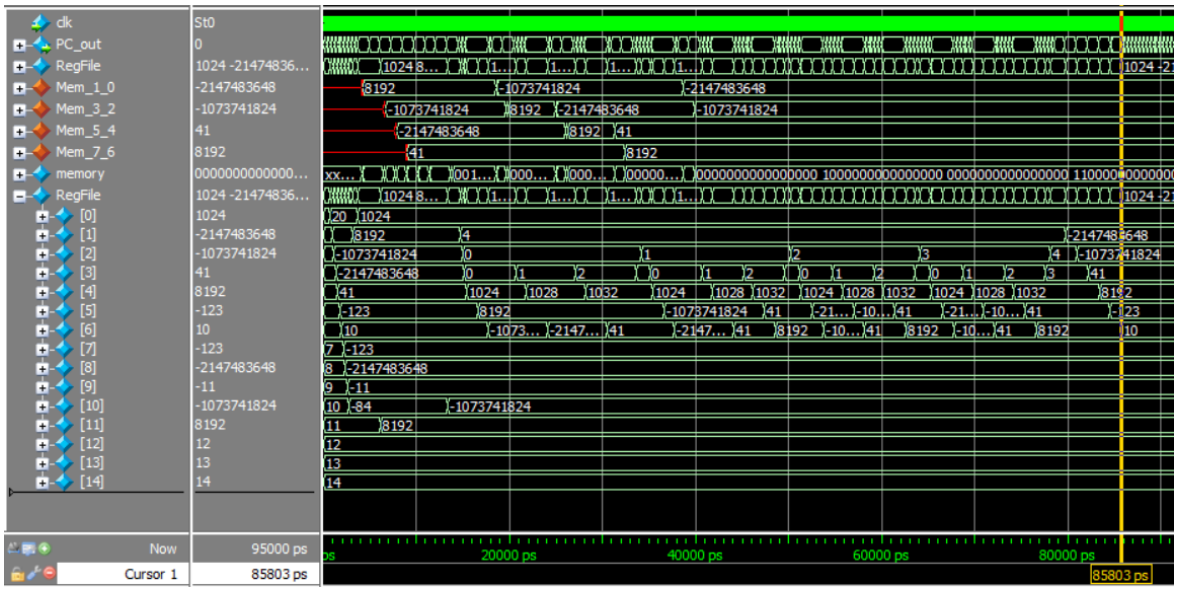
```
module TB_DE2 ();

  reg clk, rst, forward_en;

  System system (
        .clock(clk),
        .rst(rst),
        .forward_en(forward_en)
      );

  initial
  begin
    clk = 1;
    forward_en = 1;
    repeat (1900)
    begin
      #50;
      clk = ~clk;
    end
  end

  initial
  begin
    rst = 0;
    #20 rst = 1;
    #10 rst = 0;
  end

endmodule
```

شکل ۱۱

خروجی بدون SRAM:



خروجی به همراه SRAM:



همانطور که می‌بیند در انتهای هر دو تست این ۴ رجیستر سورت شده می‌باشند. و به دلیل اینکه هر نوشتن و خواندن در SRAM ۶ سایکل طول می‌کشد، در برنامه‌ای که اجرا کردیم، زمان اجرا بیش از ۲ برابر شد.