آزمایشگاه معماری کامپیوتر

پردازنده ARM

محمد سعادتی (۸۱۰۱۹۸۴۱۰)

سید محمد امین اطیابی (۸۱۰۱۹۸۵۵۹)

فاز اول: پیاده سازی پردازنده ARM

در این فاز به پیاده سازی حالت ساده پردازنده ARM می پردازیم. ماژول های خاص استفاده شده برای عملکرد این فاز به شرح زیر می باشد:

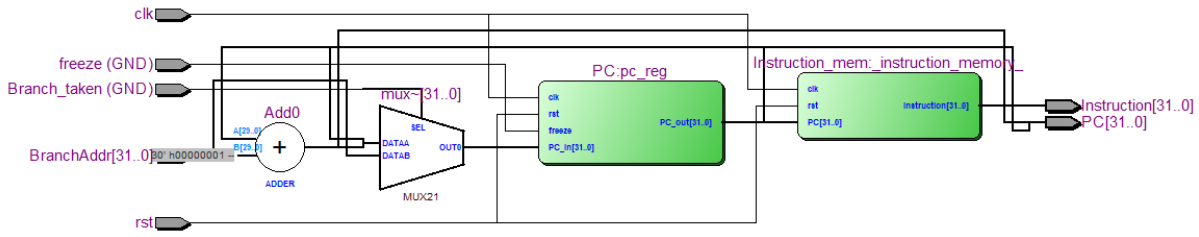
۱- Hazard detection به منظور رفع مخاطره های داده ای
۲- ماژول حافظه با قابلیت read و write

بقیه ماژول ها با توجه عملکرد ویژه این پردازنده ، صرفا برای همین پردازنده پیاده سازی شده اند.

قسمت اول آزمایش ARM : (جلسه اول)

ماژول IF

در ماژول IF از حافظه دستور، دستورات خوانده می شوند و برای decode شدن به ماژول ID می رود. مقدار رجیستر PC با توجه به branch بودن یا نبودن آخرین دستور خوانده شده ، برابر مقداری که branch مشخص کرده است یا PC+4 می شود.



IF block diagram ۱ Figure

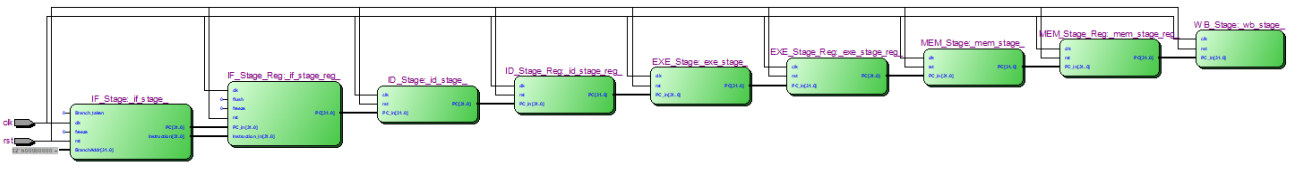


Figure ۲ وضعیت پردازنده در پایان پیاده سازی قسمت اول

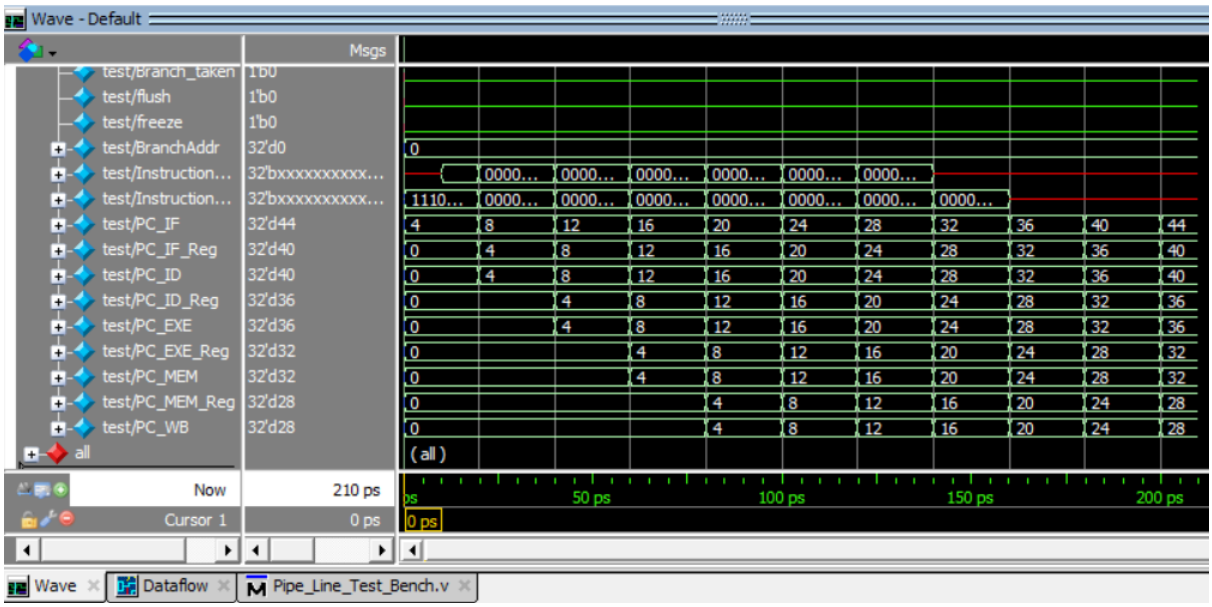


Figure ۳حرکت موج گونه PC

مشاهده می شود که پایپ لاین به درستی عمل می کند.

قسمت دوم آزمایش ARM : (جلسه دوم)

ماژول ID

در ماژول ID دستورات ارسال شده از مرحله IF تحلیل می شود و سیگنال های کنترلی مربوط به آن ایجاد می شود. مقدار های ریجتسر مقصد در مرحله WB و سایر رجیستر های مورد نیاز در این بخش مشخص می شود.

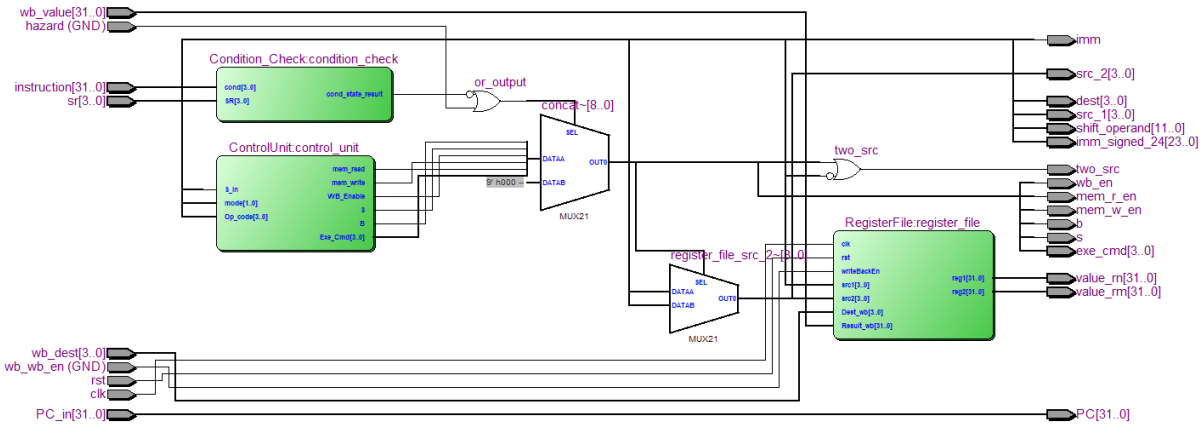
دستور ها در پردازنده ARM به سه دسته کلی

- Arithmetic (محاسباتی)
- Memory (حافظه)
- Branch

تقسیم بندی می شود که براساس این تقسیم بندی ، ControlUnit پیاده سازی شده است که جزئیات پیاده سازی آن در فایل ControlUnit.v قابل مشاهده است.

در شبیه سازی عملکرد این بخش فرض می کنیم که هیچ وقت hazard رخ نمی دهد (در غیر این صورت برای پیاده سازی ماژول hazard نیاز داریم که هر دو ماژول ID و EXE بطور کامل پیاده سازی شده باشد که فعلا هر دوی این ماژول ها را بطور کامل در اختیار نداریم). از آنجایی که پردازنده در این قسمت صرفا

دستور های متناظر برای اجرای هر دستور را تولید می کند، پس چنین فرضی در عملکرد پردازنده اختلالی ایجاد نمی کند.



ID block diagram ۴ Figure

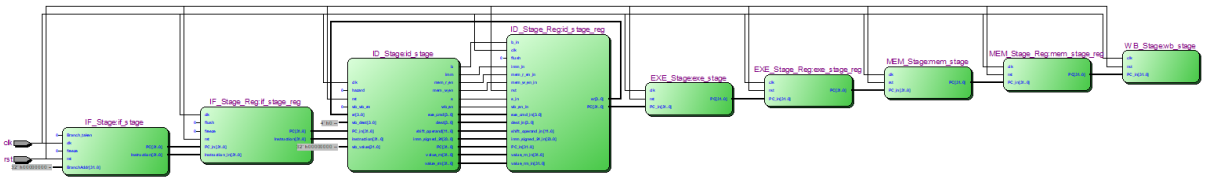
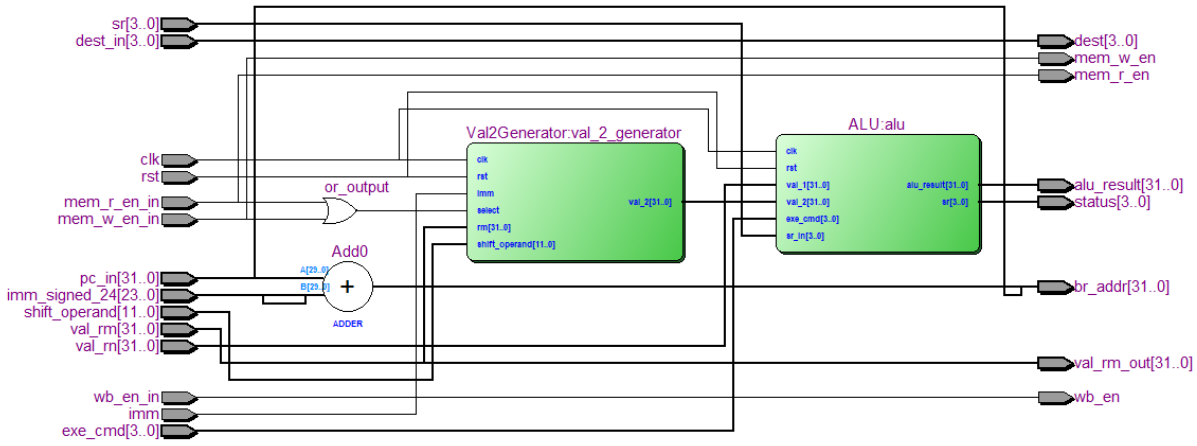


Figure ۵ وضعیت پردازنده در پایان پیاده سازی قسمت دوم

قسمت سوم آزمایش ARM : (جلسه سوم)

ماژول EXE

ماژول EXE از ALU ، Val2 Generator تشکیل شده است. پیاده سازی ALU همانند پیاده سازی ای است که در MIPS صورت گرفته است و تفاوت خاصی ندارد. مقدر بیت های رجیستر state توسط ALU مشخص می شود. ماژول Val2 Generator به منظور نحوه انتخاب دومین ورودی ماژول ALU عمل می کند.

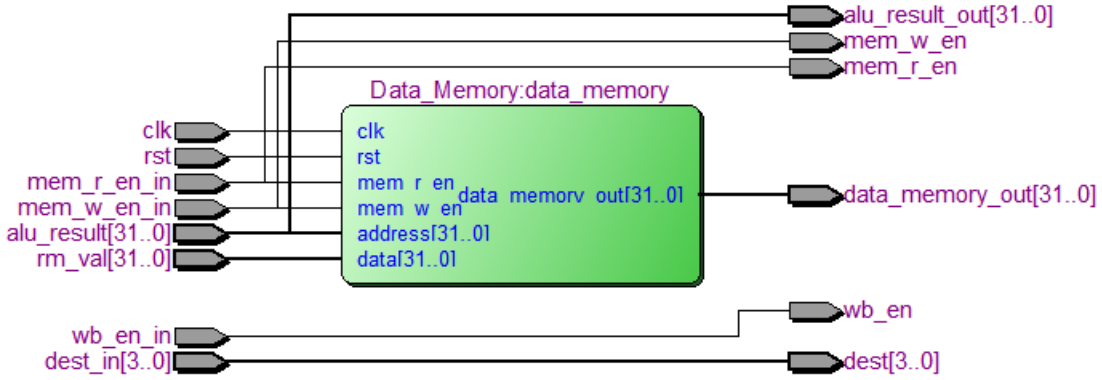


EXE block diagram۶ Figure

قسمت چهارم آزمایش ARM : (جلسه چهارم)

ماژول MEM

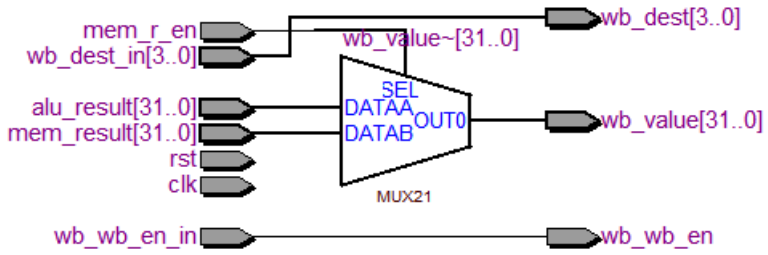
ماژول MEM در واقع همان حافظه ما می باشد. در فاز های بعدی حافظه با SRAM و سپس به SRAM همراه با cache تبدیل خواهد شد.



MEM block diagram۷ Figure

ماژول WB

ماژول WB مقدار نهایی برای نوشتن در RegisterFile را تعیین می کند.



WB block diagram۸ Figure

تست عملکرد پردازنده نهایی

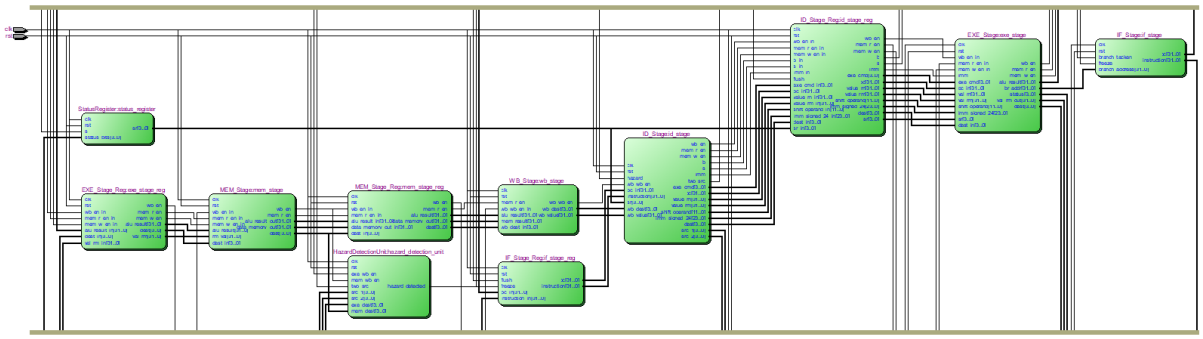
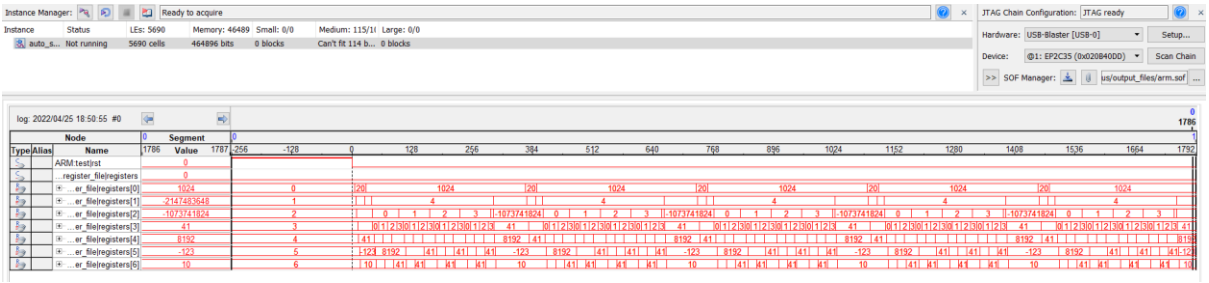


Figure ۹ پردازنده نهایی ARM



وضعیت نهایی رجیستر ها10 Figure

با توجه نتایج بدست آمده تعداد clock-cycle ها برابر است با : ۳۰۸ کلاک

Flow Summary	
Flow Status	Successful - Mon Apr 25 18:30:31 2022
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	6,005 / 33,216 (18 %)
Total combinational functions	3,479 / 33,216 (10 %)
Dedicated logic registers	4,497 / 33,216 (14 %)
Total registers	4497
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	201,760 / 483,840 (42 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure ۱۱ نتیجه سنتز پردازنده ARM

محاسبه زمان اجرا برنامه:

برای محاسبه زمان اجرا برنامه، تعداد سیکل هایی که برنامه طول می کشد تا اجرا شود را در اندازه هر کلاک (ما از کلاک ۵۰ مگاهرتز استفاده می کنیم) ضرب می کنیم.

محاسبه CPI :

برای محاسبه CPI تعداد کلاک که برنامه طول کشیده تا اجرا شود را تقسیم بر تعداد دستورات یعنی ۴۶ می کنیم.

$$\textbf{CPI = Clock per Instruction} = \frac{clocks}{instruction}$$

Compilation Report	
Total Logic Elements	6005
Total Combinational functions	3479
Dedicated Logic registers	4497
زمان اجرا برنامه	0.00616 sec
CPI	6.695

مشکلات هنگام کد نویسی در فاز اول:

۱- گزاره های if , else برای کامپایل شدن در کوارتز ، باید حتما دارای بلاک begin end باشند.

۲- اگر سیگنالی در بلاک always وجود داشته باشد اما از آن استفاده نشود ، مانند سیگنال rst ای که به عنوان sensitivity list در بلاک always وجود دارد اما در بدنه بلاک always از آن استفاده نشده است، باید حتما در یک گزاره شرطی استفاده شوند بطور مثال برای rst به شکل زیر می توانیم انجام دهیم:

```
If (rst) begin end
```

مشاهده می شود در این حالت سیگنال rst هم حضور دارد و هم تاثیری بر مقادیر موجود ندارد. در واقع در صورت فعال بودن این سیگنال اتفاق خاصی و تغییری بر سایر متغییر ها رخ نمی دهد و از طرفی این سیگنال بلا استفاده نیز نیست.

۳- برای به رسیدن به جواب در سنتز کوارتز، ممکن sample depth موجود در signal tap به دلیل طولانی بودن روند خاتمه مسئله کافی نباشد. به همین منظور باید مقدار sample depth را افزایش دهیم تا به جواب برسیم و پایان برنامه قابل مشاهده باشد.

فاز دوم: افزودن تکنیک ارسال به جلو (Forwarding) به پردازنده ARM

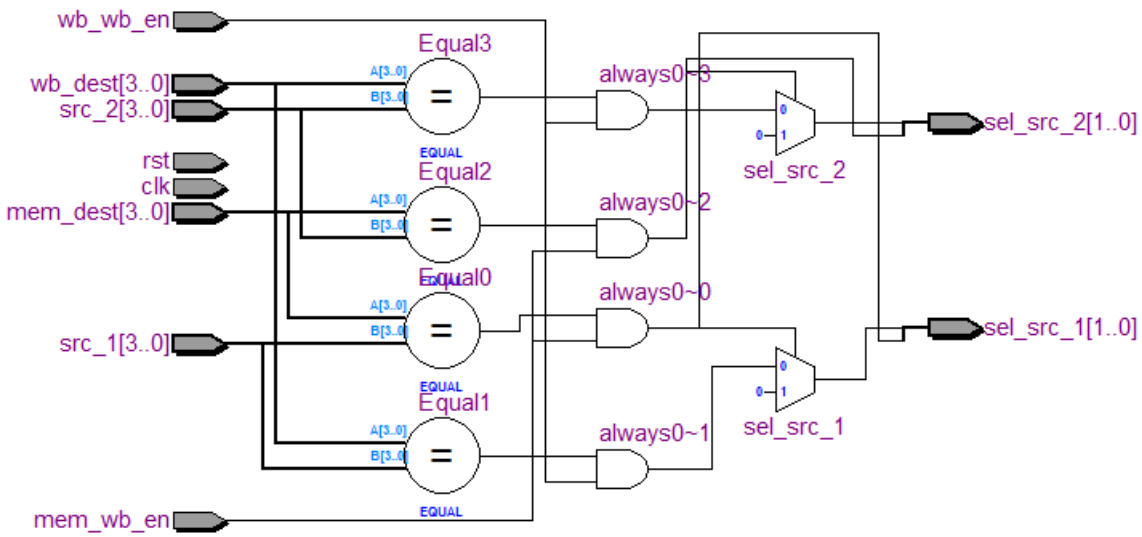
```
always @ (src_1, src_2, mem_dest, mem_wb_en, wb_dest, wb_wb_en) begin
  if ((src_1 == mem_dest) && (mem_wb_en)) sel_src_1 = 2'b01;
  else if ((src_1 == wb_dest) && (wb_wb_en)) sel_src_1 = 2'b10;
  else sel_src_1 = 2'b00;

  if ((src_2 == mem_dest) && (mem_wb_en)) sel_src_2 = 2'b01;
  else if ((src_2 == wb_dest) && (wb_wb_en)) sel_src_2 = 2'b10;
  else sel_src_2 = 2'b00;
end
```

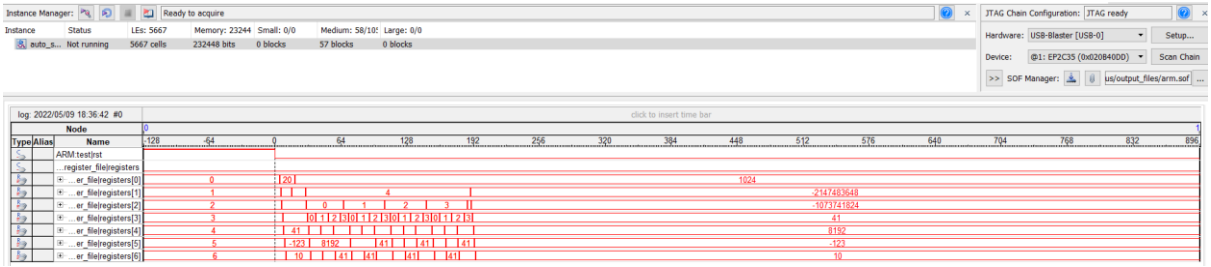
Figure ۱۲ کد بخش Forwarding

مصرف کننده داده در پردازنده ما تنها EXE Stage است . این بخش داده‌ها خود را از RegisterFile می‌خواند و استفاده می‌کند . حال ممکن است دو دستور به صورت پشت سر برای اجرا داشته باشیم که دستور اول باید داده خود را در حافظه یادداشت کند سپس دستور دوم از این داده استفاده کند . همانطور که می‌دانیم این کار نیاز به چند سیکل توقف دارد . ما در این بخش به پیاده سازی تکنیک ارسال به جلو پرداختم . به این صورت که داده‌هایی که در مرحله MEM یا WB قرار دارند زودتر به مصرف کننده می‌رسند . در واقع داده ما در مراحل MEM و WB آماده است و ما داده را قبل از ذخیره شدن در RegisterFile برای مرحله EXE ارسال می‌کنم تا این مرحله از داده استفاده کند و می‌گذاریم پردازنده در سیکل‌های بعدی داده را ذخیره کند .

تشخصی نیاز ارسال به جلو با مقایسه مقصد نوشتن و خواند دو دستوری که یکی در مرحله EXE و دیگری در مرحله MEM یا WB است صورت می‌گیرد و با توجه به محل داده ، به کمک مالتی‌پلکسر داده ورودی به ALU را مشخص می‌کنیم .



Forwarding Unit ماژول ۱۳ Figure



Forwarding عملکرد با ۱۴ Figure

با توجه نتایج بدست آمده تعداد clock-cycle ها برابر است با : ۱۹۴ کلاک

Flow Summary	
Flow Status	Successful - Mon May 09 18:32:45 2022
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	7,187 / 33,216 (22 %)
Total combinational functions	4,274 / 33,216 (13 %)
Dedicated logic registers	5,070 / 33,216 (15 %)
Total registers	5070
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	466,976 / 483,840 (97 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Forwarding با ARM نتیجه سنتز پردازنده ۱۵ Figure

Compilation Report	
Total Logic Elements	7187
Total Combinational functions	4274
Dedicated Logic registers	5070
زمان اجرا برنامه	0.00388 sec
CPI	4.217

میزان افزایش کارایی با آزمایش دوم (بدون ارسال به جلو) (حالت پردازنده با Forwarding نسبت به پردازنده پایه):

$$\text{Speed-up} = \frac{194}{308} * 100 \ = 63\ \%$$

نتیجه می گیریم که با استفاده از تکنیک ارسال به جلو، تعداد کلاک حدود ۳۷ درصد کم شده است.

میزان هزینه سخت افزاری (درصد افزایش استفاده از المان های منطقی) (حالت پردازنده با Forwarding نسبت به پردازنده پایه):

$$\frac{7178}{6005} * 100 \ = 119\ \%$$

نتیجه می گیریم که استفاده از تکنیک ارسال به جلو، باعث افزایش ۱۹ درصدی سخت افزار مورد استفاده شده است.

میزان کارایی بر هزینه(Performance per Cost) (حالت پردازنده با Forwarding نسبت به پردازنده پایه):

$$\text{Speed-up} = \frac{4.217}{6.695} * 100 \ = 63\ \%$$

نتیجه می گیریم که با استفاده از تکنیک ارسال به جلو، عملکرد پردازنده حدود ۳۷ درصد بهتر شده است.

فاز سوم: استفاده از SRAM در پردازنده ARM به عنوان حافظه داده

```
parameter one = 3'd0, two = 3'd1, three = 3'd2, four = 3'd3, five = 3'd4;

assign address_t = address - 1024;
assign address1 = {address_t[17:1], 1'b0};
assign address2 = {address_t[17:1], 1'b1};

always @ (posedge clk) begin
  SRAM_UB_N = 0 ; SRAM_LB_N = 0 ; SRAM_CE_N = 0 ; SRAM_OE_N = 0 ; SRAM_WE_N = 1;
  case(ps)
    one: begin
      if(wr_en) begin SRAM_DQ_reg <= writeData[31:16]; SRAM_ADDR_reg <= address2; SRAM_WE_N <= 1'b0; end
      else if(rd_en) begin readData[15:0] <= SRAM_DQ; end
    end
    two: begin
      if(wr_en) begin SRAM_DQ_reg <= writeData[15:0]; SRAM_ADDR_reg <= address1; SRAM_WE_N <= 1'b0; end
      else if(rd_en) begin readData[31:16] <= SRAM_DQ; end
    end
  endcase
end

always @ (ps) begin
  if(ps == five) ns = one;
  else ns = ps + 1;
end

always @ (posedge clk) begin
  if(rst | (!wr_en & !rd_en)) ps <= one;
  else ps <= ns;
end

assign SRAM_DQ = (wr_en) ? SRAM_DQ_reg : 16'bzzzzzzzzzzzzzzzz;
assign ready = ((ps == one || ps == four || ps == three || ps == two) && (wr_en | rd_en)) ? 0 : 1;
assign SRAM_ADDR = (wr_en) ? SRAM_ADDR_reg :
                   (ps == one) ? address1 :
                   (ps == two) ? address2 : 16'b0;
```

Figure ۱۶ کد SRAM Controller

در SRAM با فعال شدن سیگنال rd_en ، آماده به خواندن داده می شود و توسط یک counter در ۶ کلاک داده را می خواند. با فعال شدن سیگنال wr_en ، آماده به نوشتن داده می شود و توسط یک counter در ۶ کلاک داده را می نویسد.

هنگامی در حال خواندن یا نوشتن هستیم، counter با هر کلاک یک واحد می شمارد و در حالتی که مقدار counter به ۶ رسید یا سیگنال های rd_en و wr_en فعال نبودند، مقدار سیگنال ready برابر ۱ می شود.

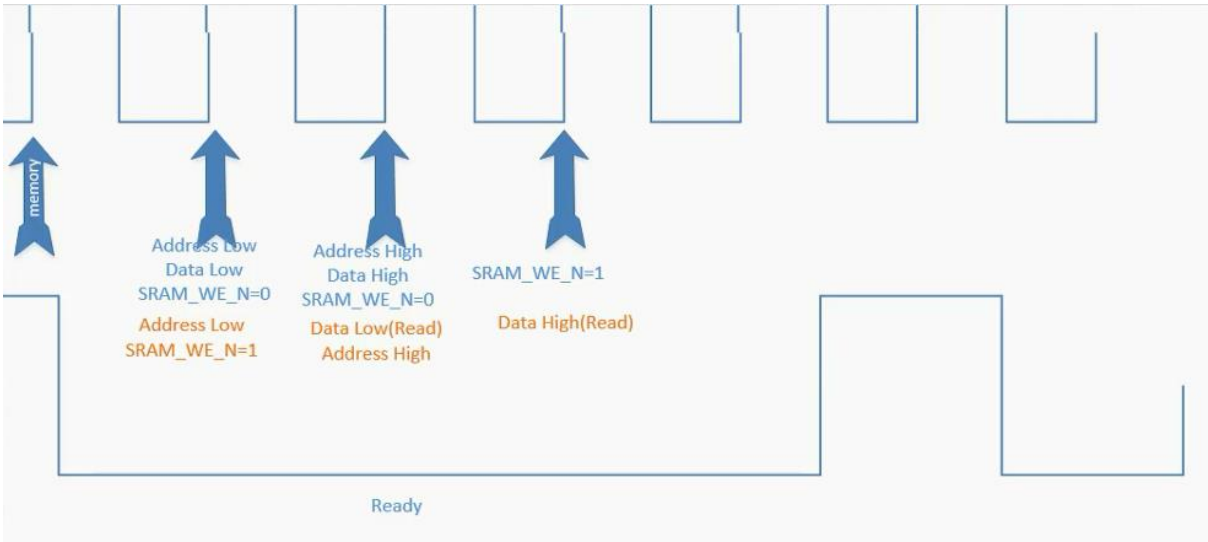


Figure ۱۷ عملکرد SRAM در هر کلاک

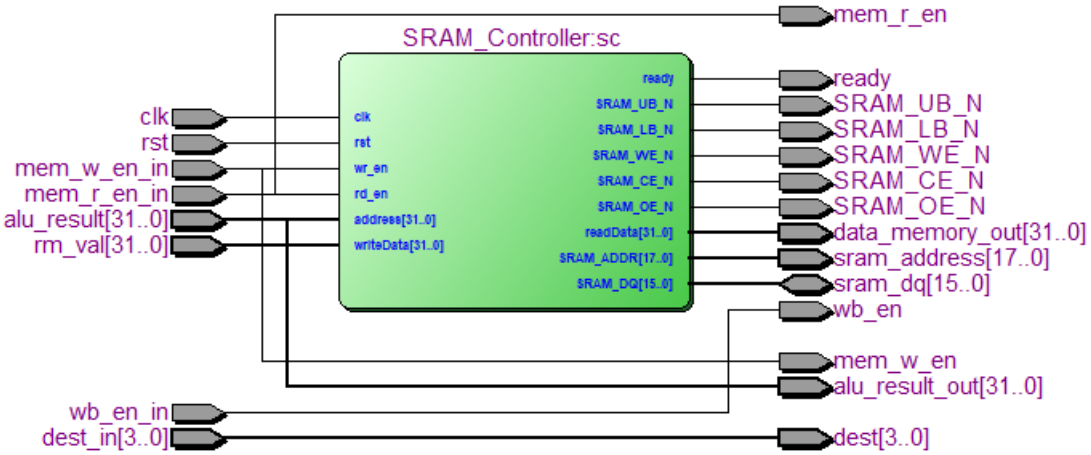


Figure ۱۸ ماژول MEM پس از اضافه شدن SRAM

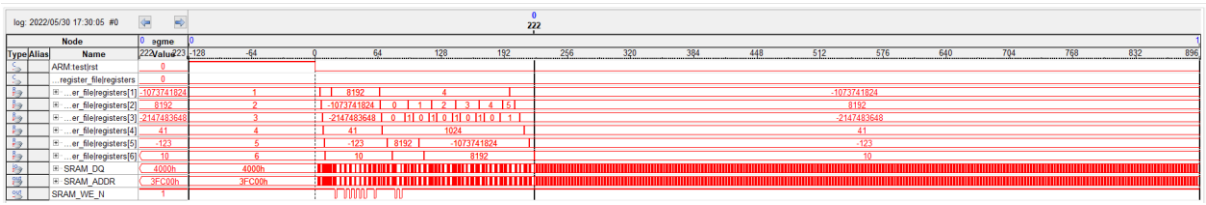


Figure ۱۹ عملکرد با Forwarding

با توجه نتایج بدست آمده تعداد clock-cycle ها در حالت با Forwarding برابر است با : ۲۲۲ کلاک

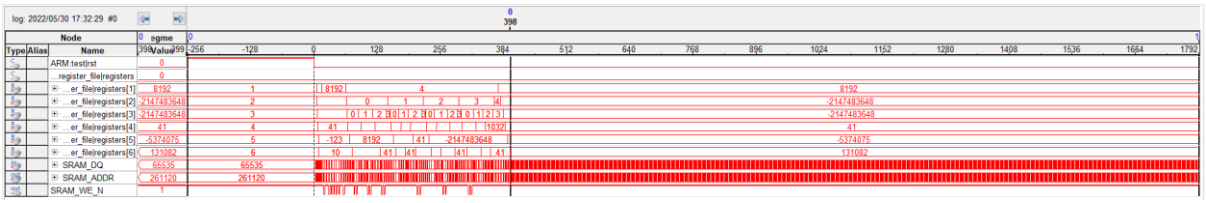


Figure ۲۰ عملکرد بدون Forwarding

با توجه نتایج بدست آمده تعداد clock-cycle ها در حالت بدون Forwarding برابر است با :

۳۹۸ کلاک

Flow Summary	
Flow Status	Successful - Mon May 30 17:29:30 2022
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	6,775 / 33,216 (20 %)
Total combinational functions	3,767 / 33,216 (11 %)
Dedicated logic registers	5,097 / 33,216 (15 %)
Total registers	5097
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	235,520 / 483,840 (49 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure ۲۱ نتیجه سنتز پردازنده ARM با SRAM

Compilation Report	
Total Logic Elements	6775
Total Combinational functions	3767
Dedicated Logic registers	5097
زمان اجرا برنامه	0.00796 sec
CPI	8.652

میزان کاهش کارایی با حالت استفاده از حافظه داخلی (حالت پردازنده با SRAM نسبت به پردازنده پایه):

$$\text{Speed-up} = \frac{398}{194} * 100 \ = 205\ \%$$

نتیجه می گیریم که استفاده از SRAM باعث افزایش ۱۰۵ درصدی تعداد کلاک ها شده است.

میزان هزینه سخت افزاری (درصد افزایش استفاده از المان های منطقی) (حالت پردازنده با SRAM نسبت به پردازنده پایه):

$$\frac{6775}{6005} * 100 \ = 112\ \%$$

نتیجه می گیریم که استفاده از SRAM ، باعث افزایش ۱۲ درصدی سخت افزار مورد استفاده شده است.

میزان کارایی بر هزینه(Performance per Cost) (حالت پردازنده با SRAM نسبت به پردازنده پایه):

$$\text{Speed-up} \ = \frac{8.652}{4.826} * 100 \ = 205\ \%$$

نتیجه می گیریم که با استفاده از SRAM ، عملکرد پردازنده حدود ۱۰۵ درصد کندتر شده است.

فاز چهارم: استفاده از حافظه نهان (Cache) در پردازنده ARM

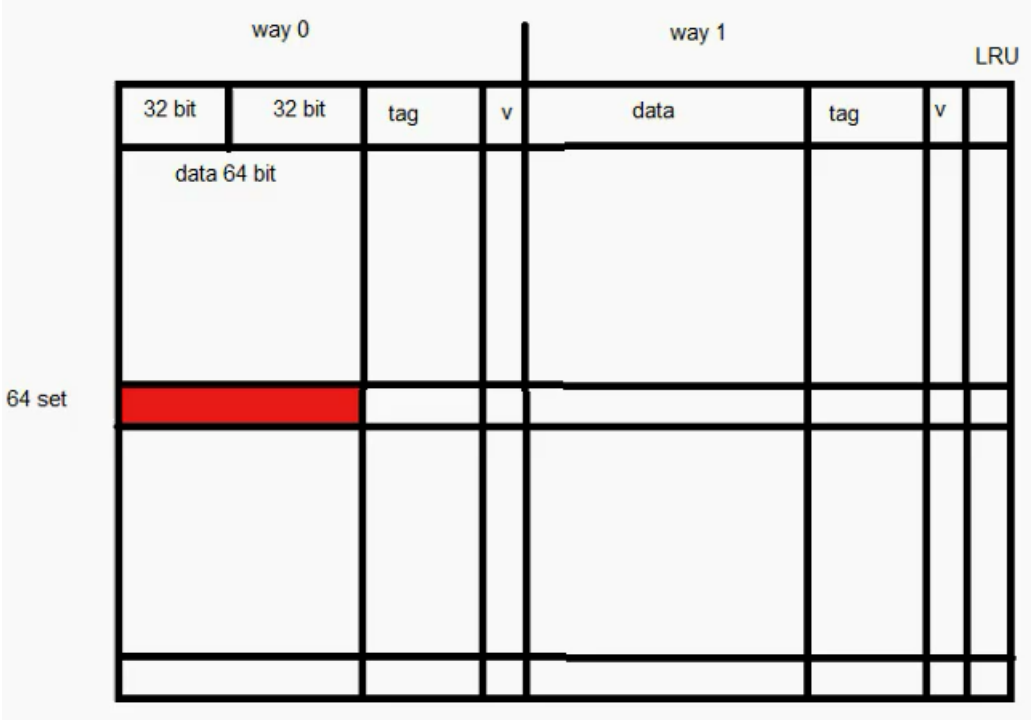


Figure ۲۲ ساختار حافظه نهان

```
assign hit0 =(way0[73:65] == tag_address & way0[0]) ? 1 : 0;
assign hit1 =(way1[73:65] == tag_address & way1[0]) ? 1 : 0;

assign ready = ((hit0 | hit1 | !MEM_R_EN) & !MEM_W_EN);

integer i;
always @ (posedge clk) begin
  if(rst) for(i = 0 ; i < 64 ; i = i + 1) cache[i] = 149'b0;
  else begin
    if(MEM_W_EN) begin
      if(hit0) cache[index_address][74] = 0;
      else if(hit1) cache[index_address][0] = 0;
    end
    if(MEM_R_EN & ready) cache[index_address][148] = hit0 ? 1 : 0;
    read = (!ready & MEM_R_EN);
    write = MEM_W_EN;
    if(!ready & MEM_R_EN & sram_ready)begin
      if(LRU == 0)begin
        cache[index_address][64+74:1+74] = sram_rdata;
        cache[index_address][0+74] = 1;
        cache[index_address][73+74:65+74] = tag_address;
        cache[index_address][148] = 1;
      end
      else if(LRU == 1)begin
        cache[index_address][64:1] = sram_rdata;
        cache[index_address][0] = 1;
        cache[index_address][73:65] = tag_address;
        cache[index_address][148] = 0;
      end
    end
  end
end

assign sram_address = Address;
assign sram_wdata = wdata;
assign rdata = hit0 ? (offset[2] ? way0[64:33] : way0[32:1]) :
               ( hit1 ? (offset[2] ? way1[64:33] : way1[32:1]) :
               (offset[2] ? sram_rdata[63:32] : sram_rdata[31:0]));
```

Figure ۲۳کد Cache Controller

در عکس بالا جزئیات پیاده سازی Cache Controller نشان داده شده است. نوشتن در حافظه Cache توسط سیاست LRU صورت می گیرد که با توجه به مقدار بیت LRU در سطر مد نظر در حافظه کش، نوشتن در way 0 یا way 1 صورت می گیرد.

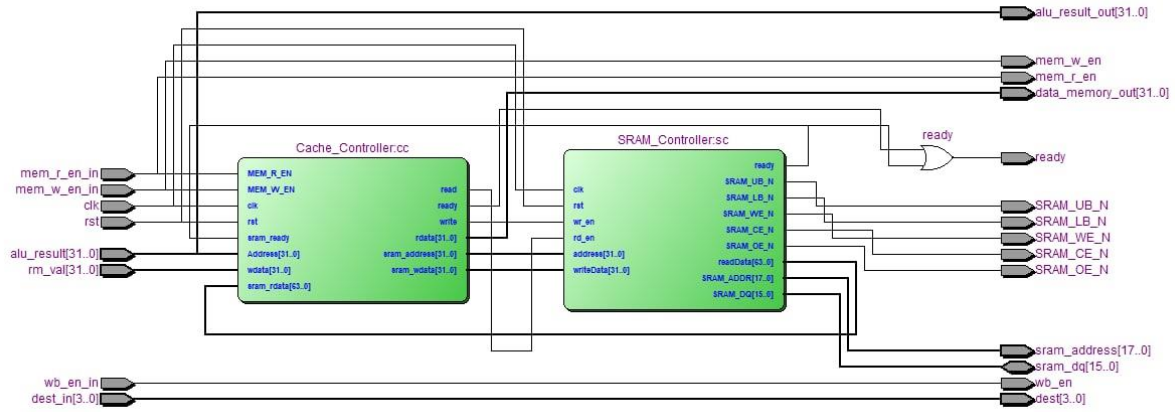


Figure ۲۴ ماژول MEM پس از اضافه شدن Cache

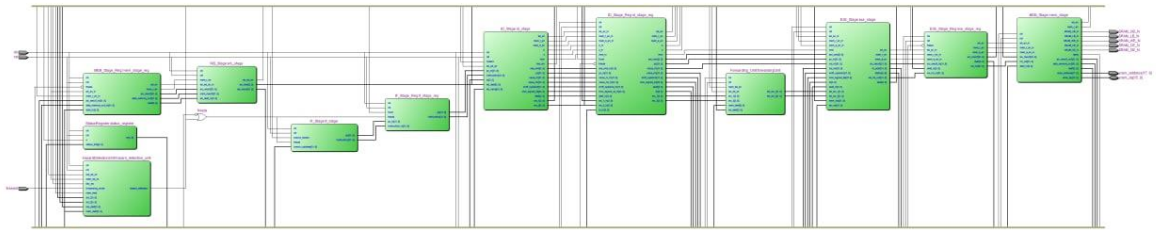


Figure ۲۵ کل پردازنده نهایی ARM

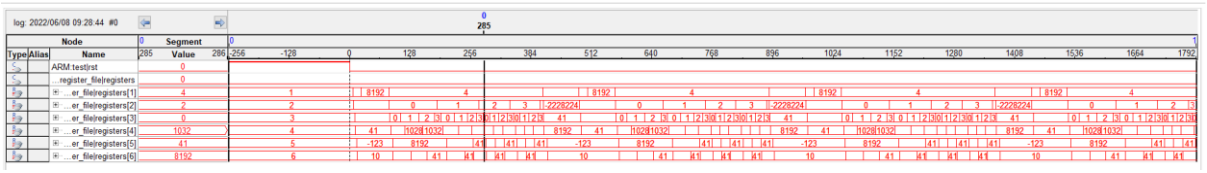


Figure ۲۶ عملکرد Cache

با توجه نتایج بدست آمده تعداد clock-cycle ها در حالت استفاده از حافظه نهان برابر است با :

۲۸۵ کلاک

Flow Summary	
Flow Status	Successful - Mon Jun 13 09:58:45 2022
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	17,640 / 33,216 (53 %)
Total combinational functions	10,508 / 33,216 (32 %)
Dedicated logic registers	14,092 / 33,216 (42 %)
Total registers	14092
Total pins	42 / 475 (9 %)
Total virtual pins	0
Total memory bits	399,360 / 483,840 (83 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure ۲۷ نتیجه سنتز پردازنده ARM با Cache

Compilation Report	
Total Logic Elements	17640
Total Combinational functions	10508
Dedicated Logic registers	14092
زمان اجرا برنامه	0.0057
CPI	6.195

میزان کارایی با حالت استفاده از حافظه داخلی (حالت پردازنده با Cache نسبت به پردازنده پایه):

$$\text{Speed-up} = \frac{285}{308} * 100 \ = 92\ \%$$

نتیجه می گیریم که استفاده از SRAM باعث کاهش ۷ درصدی تعداد کلاک ها شده است.

میزان هزینه سخت افزاری (درصد افزایش استفاده از المان های منطقی) (حالت پردازنده با Cache نسبت به پردازنده پایه):

$$\frac{17640}{6005} * 100 \ = 293\ \%$$

نتیجه می گیریم که استفاده از SRAM ، باعث افزایش ۱۹۳ درصدی سخت افزار مورد استفاده شده است.

میزان کارایی با حالت استفاده از SRAM (حالت پردازنده با Cache نسبت به پردازنده با SRAM):

$$\text{Speed-up} = \frac{285}{398} * 100 \ = 71\ \%$$

نتیجه می گیریم که استفاده از SRAM باعث کاهش ۲۹ درصدی تعداد کلاک ها شده است.

میزان هزینه سخت افزاری (درصد افزایش استفاده از المان های منطقی) (حالت پردازنده با Cache نسبت به پردازنده با SRAM):

$$\frac{17640}{6775} * 100 \ = 260\ \%$$

نتیجه می گیریم که استفاده از SRAM ، باعث افزایش ۱۶۰ درصدی سخت افزار مورد استفاده شده است.