تمرین طراحی سیستم های دیجیتال

طراحی پردازنده

نام دانشجو: سیدمحمدرضا جوادی

نام استاد: دکتر فصحتی

قالب بندی کلی پوشه ها:

./

./src/

ALUtest

Multb

..

./syn/

Project

..

doc.pdf

مقدمه

پیاده سازی و تست این پردازنده چندین بخش داشت برای همین چند پوشه مختلف برای تست ها و پیاده سازی ها تعبیه شده. اول هربخش پوشه و تست مربوطه را بیان می کنم. توجه کنید صورت تمرین waveform نخواسته، برای تست ها هم باید نشان دهیم درست کار می کند. پوشه اصلی شامل همه کد ها و تست کلی srcاست که در بخش خودش می پردازیم. از آنجا که همه فایل های پیاده سازی آنجا هستند کمی شلوغ است پس با توضیحات پیش بروید. به دلیل زیاد بودن کد و تعداد فایل ها اینجا عکس زیادی نمی گذارم اما در هرپوشه که ذکر می کنم فایل ها پخش شده اند تا راحت پیدا کنید.

پیاده سازی ALU

به پوشه ALUtest بنگرید. فایل ALUTB.v این ماژول را به روش تمامی حالات با هر 4 عملیات تست می کند و نتیجه واقعی را با نتیجه ALU مقایسه می کند. خروجی تست نیز اگر حتی یک خطا داشته باشیم fail می شود وگرنه تست کیس های غلط را نشان می دهد برای دیباگ. توجه کنید چون تست خیلی طول می کشید گام را 103 تعیین کردم که بسیاری از

تست کیس ها و همینطور edge case ها چک شوند، اما اگر مایل بودید می توانید با عوض کردن 103 به 1 همه حالات را بررسی کنید(من سعی داشتم بکنم بعد 3 ساعت هنوز 0.001 هم نرفته بود.خیلی طول می کشد)

```
        for (a_value = -32768; a_value <= 32767; a_value = a_value + 103) begin
            for (b_value = -32767; b_value <= 32767; b_value = b_value + 103) begin
```

می بینید محدوده بسیار وسیعی است.

با این حال تست کیس ها اکثر مرزی ها و تعداد بسیاری از همه حالات را چک شده اند و قطعا ماژول درست کار می کند.

```
javadi@DESKTOP-OVP2LHB:/mnt/c/Users/Asus/Desktop/DSD/ALUtest$ vvp a.out
nice
ALUTB.v:79: $finish called at 34084596 (1s)
```

همانطور که می بینید همین 34,084,596 واحد زمانی طول کشید(لپتاپ خودم مدتی برایش روشن بود(:)

نتیجه تست مثبت بود و به درستی کار می کند. این ماژول با بررسی سیگنال کنترلی خروجی ماژول مد نظر(جمع،...) را به خروجی وصل می کند. در تفریق صرفا عدد دوم را مکمل دو می کند و باز جمع انجام می شود. در ضرب ابتدا طوری ورودی های ماژول ضرب را تعیین می کند تا هردو مثبت باشند، بعد از گرفتن خروجی با توجه به علامت ورودی ها علامت خروجی ضرب را نیز تعیین می کند(مثلا اگر منفی بود مکمل 2 می کند خروجی را). تقسیم نیز همچنین تاحدودی مشابه ضرب است. خود alu در فایل ALU.v پیاده سازی شده. در این فایل از ماژول های مختلف نمونه گرفته شده، ورودی هایشان و خروجی هایشان را بر اساس aluop تعیین می کند. همینطور اگر نیاز داشتند با یک counter به آنها مهلت می دهد یا سیگنال start را فعال می کند(صرفا می گوید شروع کن)

عملیات جمع و تفریق

تفریق را در خود ALU.v با مکمل 2 کردن عدد دوم و جمع انجام می دهیم پس فقط ماژول جمع داریم. این فایل CSA.v است که همانطور که می بینید با استفاده از روش طراحی سطح گیت مطابق صورت پروژه جمع را پیاده کرده(4 بخش دارد که هر بخش به شکل ripple است و در انتهای فایل مولتی پلکسر هارا داریم. همچنین برای هردوحالت carry=1 و 0 برای 12 بیت انتهایی انجام می شود و نتیجه دلخواه از مولتی پلکسر ها انتخاب می شود)

ضرب

فایلshiftaddmul.v شامل ضرب کننده عادی 8 بیتی است که مطابق صورت پروژه پیاده شده. با استفاده از این ماژول فایل MUL.v الگوریتم کاراتسوبا را پیاده سازی می کند. رجیستر های 16 بیتی S0وS1وS2 برای کمک اضافه شده اند که کامنت هم گذاشتم نقششان چیست، غیر آن بقیه نام ها مطابق صورت تمرین هستند.

از آنجا که خروجی ALU تنها 16 بیت دارد اما خروجی ضرب 32 بیتی است ما 16 بیت پایین را به خروجی می دهیم. برای تست این که خود خروجی ضرب که 32 بیتی است درست است یانه پوشه multb به طور خلاصه تست می کند و فایل multb.v نیز تست بنچ است.(این پوشه خارج از ALUtest است حواستان باشد!)

```
javadi@DESKTOP-OVP2LHB:/mnt/c/Users/Asus/Desktop/DSD/multb$ iverilog *.v
javadi@DESKTOP-OVP2LHB:/mnt/c/Users/Asus/Desktop/DSD/multb$ vvp a.out
A=1234, B=00ff, S=001221cc, real ANS:001221cc
1
A=0bcd, B=0002, S=0000179a, real ANS:0000179a
1
multb.v:60: $stop called at 134 (1s)
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 134 ticks.
>
```

پس می بینید این ماژول نیز به درستی کار می کند.

تقسیم

فایل DIV.v پیاده سازی تقسیم است.(ALUtestپوشه)

ALUtest > DIV.v

```v
module DIV (
    input signed [15:0] A, B,
    input clk, start,
    output reg signed [31:0] S
);
    reg [4:0] count;
    reg [15:0] abs_A, abs_B;
    reg signed [31:0] remainder;
    reg signed [15:0] quotient;
    reg sign;
    always @(posedge clk) begin
        if (start) begin
            // Start division
            abs_A <= A[15] ? -A : A;
            abs_B <= B[15] ? -B : B;
            remainder <= 0;
            quotient <= 0;
            count <= 16;
            sign <= A[15] ^ B[15];
        end else begin
            remainder = {remainder[30:0], abs_A[15]};
            abs_A = abs_A << 1;
            remainder = remainder - {16'b0, abs_B};
            if (remainder[31]) begin
                remainder = remainder + {16'b0, abs_B};
                quotient = quotient << 1;
            end else begin
                quotient = (quotient << 1) | 1'b1;
            end
            count = count - 1;
            if (count == 0) begin
                S <= sign ? -quotient : quotient;
            end
        end
    end
endmodule
```

مطابق آنچه گفته شد پیاده سازی انجام شده. از آنجا که خروجی نهایی alu فقط 16 بیت است ما خارج قسمت را خروجی می دهیم که منطقی است(عملگر / را می خواستیم پیاده سازی کنیم بهرحال نه باقی مانده) توجه کنید ابتدا عملوند هارا مثبت می کنیم و بعد نتیجه را بر اساس علامتشان تعیین علامت می کنیم.

رجیسترفایل

تست این در قالب تست کلی src انجام خواهد شد پس به پوشه src بروید.فایل.registerFile است.

registerFile.v

```
module registerFile (
    input clk,
    input [15:0] wdata,
    output reg signed  [15:0] rdata1, rdata2,
    input [1:0] readAdd1,readAdd2,writeAdd,
    input we,rst
);
    reg signed [15:0] regs [0:3];
    always @(posedge clk) begin
        if (rst) begin
            regs[0]<= 0;
            regs[1]<= 0;
            regs[2]<= 0;
            regs[3]<= 0;
        end
        if (we) regs[writeAdd] <= wdata;
    end
    always @(negedge clk) begin
        rdata1 <= regs[readAdd1];
        rdata2 <= regs[readAdd2];
    end
    // initial begin
    //     $monitor("REGISTER FILE: at time: %t\nr0:%h\nr1:%h\nr2:%h\nr3:%h\n",$time,regs[0],regs[1],regs[2],regs[3]);
    // end
endmodule
```

همانطور که می بینید یک آرایه از 4 رجیستر 16 بیتی رجیستر فایل مارا تشکیل می دهد. با ریست کردن همه مقادیر صفر می شوند. نوشتن در لبه بالارونده و خواندن در لبه پایین رونده انجام می شود تا مشکل structural hazard نداشته باشیم(معماری multi cycle است البته ولی خب محکم کاری خوب است، خود صورت پروژه نیز همچین چیزی خواست.)

ممووری و حافظه

از همان پوشهsrcببینید. پیاده سازی کلی memory.v اینگونه است:

memory.v

```v
module memory (
    input memRead,
    input memWrite,rst,
    input signed [15:0] dataIN,
    input [15:0] address,
    input clk,
    output reg signed [15:0] dataOut
);
    // Declare the memory as signed
    reg signed [15:0] MEM [0:100]; // [0, 2^16 - 1]

    always @(posedge clk) begin
        if (rst) begin
            MEM[0] <= 16'b0000000000000000;
            MEM[1] <= 16'b100_01_11_000001010; // load x1 10(x3)
            MEM[2] <= 16'b100_10_11_000001011; // load x2 11(x3)
            MEM[3] <= 16'b000_00_01_10_0000000; // add x0, x1,x2
            MEM[4] <= 16'b001_00_00_01_0000000; // sub x0, x0,x1
            MEM[5] <= 16'b000_00_00_00_0000000; // add x0, x0,x0
            MEM[6] <= 16'b101_00_11_000001100;  // store x0 12(x3)
            MEM[7] <= 16'b010_11_01_10_0001100;  // mul x3, x1,x2
            MEM[8] <= 16'b000_00_00_00_0000000; // add x0, x0,x0
            MEM[9] <= 16'b011_11_00_10_0000000;  // div x3, x0,x2
            MEM[10] <= 16'h00A5;
            MEM[11] <= 16'h00AA;
            MEM[12] <= 16'b0000000000000000;
            MEM[13] <= 16'b0000000000000000;

        end else if (memWrite) begin
            MEM[address] <= dataIN;
        end
    end
    always @(negedge clk) begin
        if (memRead) begin
            dataOut <= MEM[address];
        end
    end
```

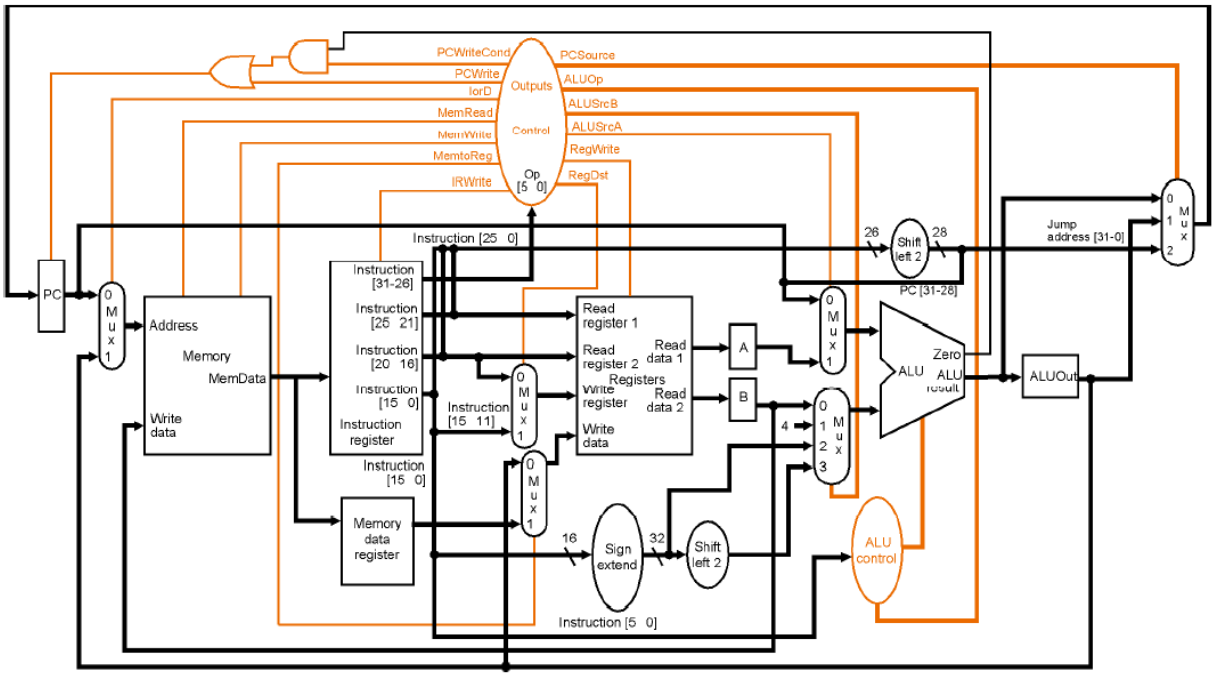
می بینید که این همه یک آرایه ای از خانه های 16 بیتی است. به دلیل مشکل در سنتز اندازه را به 100 تقلیل دادم.در هنگام ریست کردن برخی مقادیر در حافظه قرار می گیرند که شامل دستورات و مقادیر برای loadوstore هستند تا بعدا این بخش را نیز با استفاده از workload و در قالب دستورات تست کنیم، همانطور که در کامنت ها نیز می بینید خودم توضیح دادم دستورات چی هستند همینطور توجه کنید دستورات براساس صورت تمرین پیاده شده اند. دقت کنید چون یک حافظه داریم هم دستورات و هم داده ها در همین قرار می گیرند.

واحد کنترلی

در همان پوشه src ببینید. پیاده سازی این یکی(CU.v) را باید کمی توضیح دهم:

```
module CU (
    input [15:0] instruction,
    input clk,rst,
    output reg  we,memWrite,memRead,ready,regsrc,IRpdate, store,alusrc,
    output reg [1:0] ALUOP
);
```

سیگنال های کنترلی مانند میپس طراحی شده اند تا نیازی به توضیح زیادی نباشد.we برای write enable برای رجیسترفایل است.ready بخاطر صورت پروژه اضافه شده و در کلاک پایانی دستورات فعال می شود و در مرحله اول دستورات یعنی IF غیر فعال می شود. سیگنال IRpdate برای رجیستر instruction است که در ساختار کلی دستور لود شده را نگهداری می کند. در کل ساختار کلی شبیه این عکس از میپس چند چرخه ای است اما به دلیل برخی تفاوت ها مانند نبود دستورات I type یا دستورات branch ساده تر شده، اما برای درک کلی پیاده سازی کنترل یونیت خوب است:



یاد آوری:

در میپس 5 بخش کلی داریم:

IF: instruction fetch, DEC: reading register file and decode instruction, ALU: alu use,

Mem: memory read or memory write, WB: write back in register file.

در کل به همان روشی که در درس یاد گرفتیم بخش کنترلی پیاده سازی شده. در بخش ترکیبی بر اساس حالت فعلی سیگنال ها تولید می شوند و حالت بعدی مشخص می شود و در بخش ترتیبی حالت بعدی درون حالت فعلی قرار می گیرد و آپدیت می شود یک شمارنده هم داریم که در عملیات های ضرب و تقسیم بکار می روند زیرا نمی توان 16 حالت جدید دستی ایجاد کرد اما می توان شمارنده گذاشت که 16 بار در یک حالت بماند. برای محکم کاری عدد بیشتری را شرط رفتن به حالت بعد از محاسبه ضرب و تقسیم گذاشتم تا از درستی اطمینان یابم. طبیعتا تست این ماژول تنها در قالب اجرای دستورات ممکن است. بعدا در بخش تست به آن می پردازیم.

وصل کردن ماژول ها و ایجاد پردازنده

فایل processor.v همه ماژول هارا سرهم بندی می کند. شکل بالا می تواند شهود خوبی بدهد. فقط به بخش pc می پردازیم چون زیرمجموعه CU است و من برای پیاده سازی بهتر منتقل کردم. باقی کد یا سیم کشی بین ماژول هاست و تعریف آن ها یا همان مولتی پلکسر ها برای سیگنال های کنترلی.

```
93      always @(posedge clk) begin
94          if (rst) begin
95              PC <= 0;
96          end else begin
97              if (ready) PC <= PC + 1;
98          end
99      end
100     always @(negedge clk) begin
101         if (IRpdate)
102         begin
103             instruction <= memDataOut;
104         end
105         MDB <= memDataOut;
106     end
```

در این بخش فرض شده دستورات از خانه 0 ام حافظه شروع می شوند. براساس سیگنال ready رجیستر pc یکی اضافه شده تا دستور بعدی لود شود. در لبه پایین رونده نیز دستورات و داده ها در رجیستر های واسطه مانند memDataOut یا instruction قرار می گیرد. تا که در لبه بالارونده بعدی رجیستر های ما مقادیر درست را در اختیار داشته باشند. پیاده سازی این بخش طوری است که تا حدی شبیه تصویر صفحه قبل است. در کل بیشتر مفاهیم و پیاده سازی ها از میپس و درس معماری کامپیوتر است.

تست

فایلtb.v تست بنچ است. دستورات در حافظه قرار گرفته اند صرفا باید ران کنیم و نتایج را دستی بررسی کنیم. همانطور که می بینید صرفا ریست می کنیم و تا رسیدن pc به 7 صبر می کنیم.

نتایج را به صورت دستی باید چک کنیم. باید ببینم مقادیر درست در رجیستر ها قرار دارند؟
همینطور مقادیر در حافظه درست تغییر کرده اند؟

C: > Users > Asus > Desktop > DSD > tb.v

```
1   module tb;
2       reg clk;
3       reg rst;
4       wire [15:0] pc;
5
6       processor mips(
7           .rst(rst),
8           .clk(clk),
9           .pc(pc)
10      );
11
12      initial begin
13          clk = 1;
14          forever #1 clk = ~clk;
15      end
16
17      initial begin
18          rst = 1;
19          #2;
20          rst = 0;
21          wait (pc == 7)
22          #14
23          $finish;
24      end
25
26      initial begin
27          $dumpfile("tb_.vcd");
28          $dumpvars(0, tb);
29      end
30  endmodule
```

خودتان می توانید تست من را ران کنید و صحت حرف هایی که در ادامه میزنم را چک کنید. نشان می دهم دستور درست انجام شده. دستورات تست شامل همه دستورات صورت پروژه هستند و عملا همه چیز دارد چک می شود که درست کار می کنند یا نه. از TA مربوطه پرسیدم گفتند نیاز به این حد از تست نویسی که work load بنویسیم نیست با این حال این workload کوتاه به تنهایی صحت عملکرد همه را نشان می دهد. ابتدا بیاید دستورات این work load را ببینیم(صفحه بعد)، شیوه تست درستی شبیه تست تمرین های معماری کامپیوتر دکتر اسدی است.

```
// load x1 10(x3)
// load x2 11(x3)
 // add x0, x1,x2
 // sub x0, x0,x1
 // add x0, x0,x0
 // store x0 12(x3)
  // mul x3, x1,x2
 // add x0, x0,x0
  // div x3, x0,x2
```

این دستورات از آدرس 1 تا 9 هستند. در آدرس صفر دستور add r0 r0 r0 قرار دارد که عملا اجرای آن چیزی را تغییر نمی دهد(r0=0 در ابتدا)

بیایید داده هایم را نیز ببینید:

```
MEM[10] <= 16'h00A5;
MEM[11] <= 16'h00AA;
MEM[12] <= 16'b0000000000000000;
MEM[13] <= 16'b0000000000000000;
```

قرار است از آدرس 10 و 11 load داشته باشیم همینطور در آدرس 12 یک store. برای بررسی صحت این دستور در پایین فایل حافظه شما چند displayخواهید دید.

```
    initial begin
        #18
        $display("MEMORY: at time: %t\nmem[12]:%h",$time,MEM[12]);
        #2
        #2
        #2
        #5
        #5
        #5
        #5
        #5
        #2
        #2
        #2
        #2
        #2
        #2
        $display("MEMORY: at time: %t\nmem[12]:%h",$time,MEM[12]);
    end
```

این برای بررسی همان store است که در خانه 12 انجام می شود. همچنین برای بررسی registerFile نیز عمل مشابهی انجام دادم:

```
initial begin
    $monitor("REGISTER FILE: at time: %t\nr0:%h\nr1:%h\nr2:%h\nr3:%h\n",$time,regs[0],regs[1],regs[2],regs[3]);
end
```

اینگونه می توانیم رجیستر هارا نیز بررسی کنیم.

حال تست را اجرا می کنیم.

```
javadi@DESKTOP-OVP2LHB:/mnt/c/Users/Asus/Desktop/DSD/CPU$ iverilog *.v
javadi@DESKTOP-OVP2LHB:/mnt/c/Users/Asus/Desktop/DSD/CPU$ vvp a.out
VCD info: dumpfile tb_.vcd opened for output.
REGISTER FILE: at time:                   0
r0:0000
r1:0000
r2:0000
r3:0000

REGISTER FILE: at time:                   8
r0:0000
r1:0000
r2:0000
r3:0000

MEMORY: at time:                  18
mem[12]:0000
REGISTER FILE: at time:                  18
r0:0000
r1:00a5
r2:0000
r3:0000

REGISTER FILE: at time:                  28
r0:0000
r1:00a5
r2:00aa
r3:0000

REGISTER FILE: at time:                  36
r0:014f
r1:00a5
r2:00aa
r3:0000

REGISTER FILE: at time:                  44
r0:00aa
r1:00a5
r2:00aa
r3:0000
```

```
REGISTER FILE: at time:                  52
r0:0154
r1:00a5
r2:00aa
r3:0000

MEMORY: at time:                  61
mem[12]:0154
REGISTER FILE: at time:                 120
r0:0154
r1:00a5
r2:00aa
r3:6d92

REGISTER FILE: at time:                 128
r0:02a8
r1:00a5
r2:00aa
r3:6d92

REGISTER FILE: at time:                 188
r0:02a8
r1:00a5
r2:00aa
r3:0004

REGISTER FILE: at time:                 196
r0:034d
r1:00a5
r2:00aa
r3:0004

REGISTER FILE: at time:                 204
r0:03f2
r1:00a5
r2:00aa
r3:0004
```

```
REGISTER FILE: at time:                 212
r0:049c
r1:00a5
r2:00aa
r3:0004

REGISTER FILE: at time:                 220
r0:0938
r1:00a5
r2:00aa
r3:0004

tb.v:23: $finish called at 238 (1s)
javadi@DESKTOP-OVP2LHB:/mnt/c/Users/Asus/Desktop/DSD/CPU$
```

در زمان 8 می بینید دستور 0 اجرا شد و دستور بعدی می رویم و همانطور که گفته شد تاثیری بر رجیستر فایل نداشت(مقادیر ابتدایی رجیستر ها صفر هستند.) بعد از آن در زمان 18 دستور 1 اجرا شده و لود به درستی در رجیستر انجام شده(r1=00a5) بعد آن هم در زمان 28 دستور دوم تمام شد و مقدار aa در r2 لود شد. بعد از آن در 36 دستور سوم تمام شد و r1 و r2 جمع شدند(14f) و حاصل به درستی در r0 قرار گرفت. بعد از آن دستور چهارم دوباره تفریق می

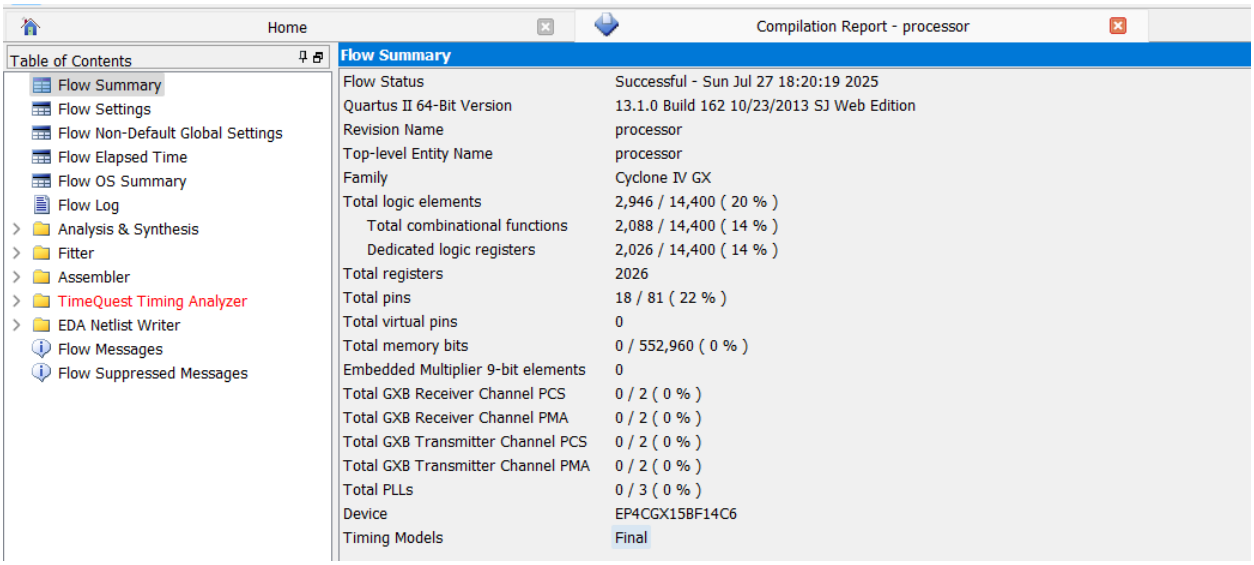
کند در زمان 44. منطقا تفریق معادل r1+r2-r1 است و باید حاصل r2 یعنی aa شود. که می بینیم حاصل همین شد و درست حساب شده. بعد از آنr0 با خودش جمع می شود در 52 یعنی باید دوبرابر شود که می شود154 پس این هم درست بود. در ادامه دستور store انجام خواهد شد تا 0x0154 در خانه 12 قرار گیرد. می بینیم در زمان 61 به نظر می رسد در خانه درست قرار گرفته چون ماژول حافظه چاپ کرده و درست قرار گرفته. بعد از آن نوبت دستور ضرب است. این مورد با اینکه کمتر از 16 کلاک نیاز دارد اما برای اطمینان من 25 کلاک وقت دادم(صورت تمرین هیچ اجباری ندارد در این موضوع) و در زمان 120 ضرب انجام می شود. می بینید مقدار درست در r3 قرار گرفته. بعد از آن دوباره جمع r0با خودش در 128. توجه کنید r0 اکنون در حقیقت x2 * 4 است. پس اگر این دو را تقسیم کنیم نتیجه می شود 4. دستور نهایی که تقسیم است در زمان 188 انجام شد و r3=4 را می توانید مشاهده کنید. تست ما به اتمام رسیده بقیه خانه های مقادیرشان چیز های رندومی است و نیاز نیست حاصل آن هارا بررسی کنیم.

پس دیدید تمامی دستورات صورت تمرین به درستی اجرا شدند و نتایج درستی داشتند.

سنتز

با استفاده از کوارتوس سنتز کردم.

نتیجه موفقیت آمیز بود:



فایل های خروجی نیز در اختیارتان قرار گرفته در پوشه syn که گفته بودید. برای اینکه اگر می خواهید از صحت سنتز مطمئن شوید خود پوشه project که درون syn است شامل خود پروژه کوارتوس است. کافی است بر فلش بالا بزنید و دوباره سنتز خواهد شد(فکر کنم تنظیمات نیز عوض نشود) و نتیجه را با فایل ها مقایسه کنید تا درستی آن ها کاملا تایید شود(راستش نمی دانم با چه ابزاری می توان خروجی هارا چک کرد به همین خاطر برایتان این روش را قرار می دهم.)