گزارش آزمایش طراحی سیستم های دیجیتال

آزمایش پنجم:

ضرب‌کننده بوث

نگارش :

سید ابوالفضل رحیمی ۹۷۱۰۵۹۴۱

امیرحسین باقری ۹۸۱۰۵۶۲۱

استاد :

دکتر علیرضا اجلالی

دستیار آموزشی :

آقای روزبه سیادت زاده

فهرست

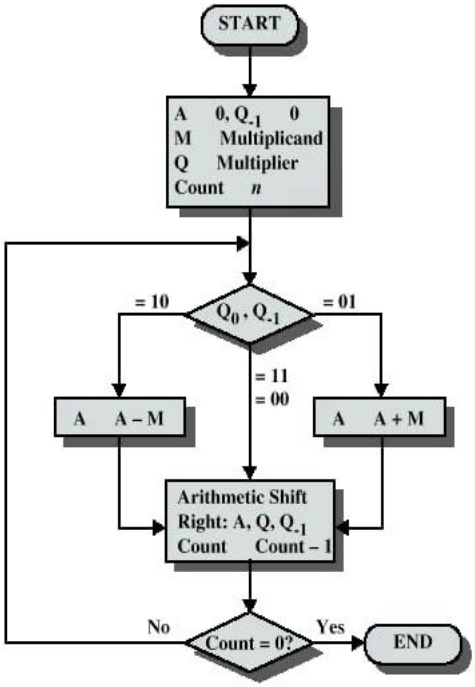
مقدمه

هدف از این آزمایش پیاده‌سازی یک ضرب کننده با استفاده از الگوریتم بوث است. این الگوریتم را با استفاده از آنچه در اسلایدهای دکتر سربازی آمده پیاده‌سازی کردیم و درنهایت توسط تست‌های متعدد این پیاده‌سازی‌مان را تست کردیم.

الگوریتم بوث

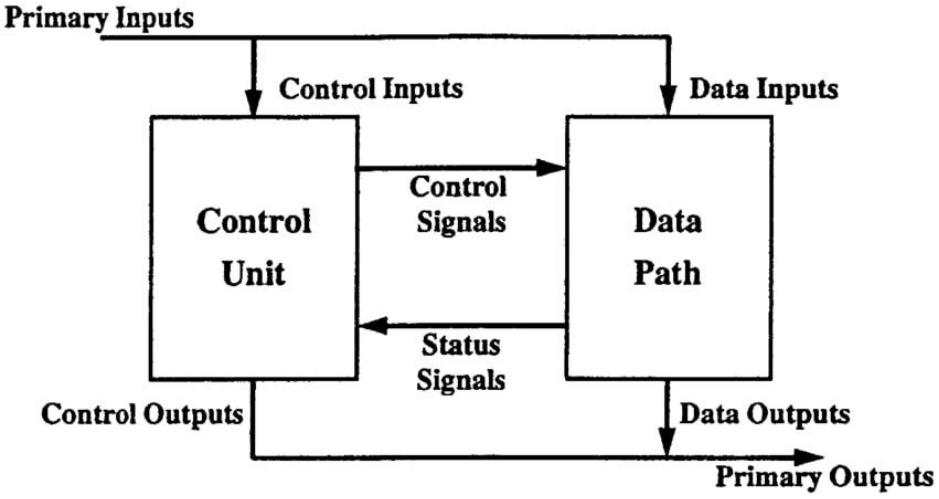
مدار به کمک ابزار مادلسیم کشیده شده اند و همچنین کد stack در ابزار کوارتوس سنتز گشته است.

در زیر می‌توانید شمای کلی از الگوریتم بوث را ببینید:



تصویر 1 شمای کلی الگوریتم بوث که پیاده‌سازی شده است.

ما در پیاده‌سازی همانطور که خواسته شده بود و برای افزایش سرعت امکان شیفت دادن بیش از یک بیت در پالس ساعت را هم فراهم کردیم. حال در نهایت و پیش از ورود به بحث پیاده‌سازی خوب است نگاهی به طراحی کلی هم که دارای یک واحد کنترل و مسیر داده هست هم داشته باشیم. این طراحی دقیقا پیاده شده است.



تصویر 2 شمای کلی واحد اصلی و رابطه میان واحد کنترل و مسیر داده

پیاده‌سازی در وریلاگ

برای پیاده‌سازی در وریلاگ سه ماژول متفاوت که هرکدام را که به ترتیب مربوط به واحد کنترل، مسیر داده و خود ضرب کننده است؛ پیاده سازی کردیم. به ترتیب به هرکدام به صورت جداگانه می‌پردازیم:

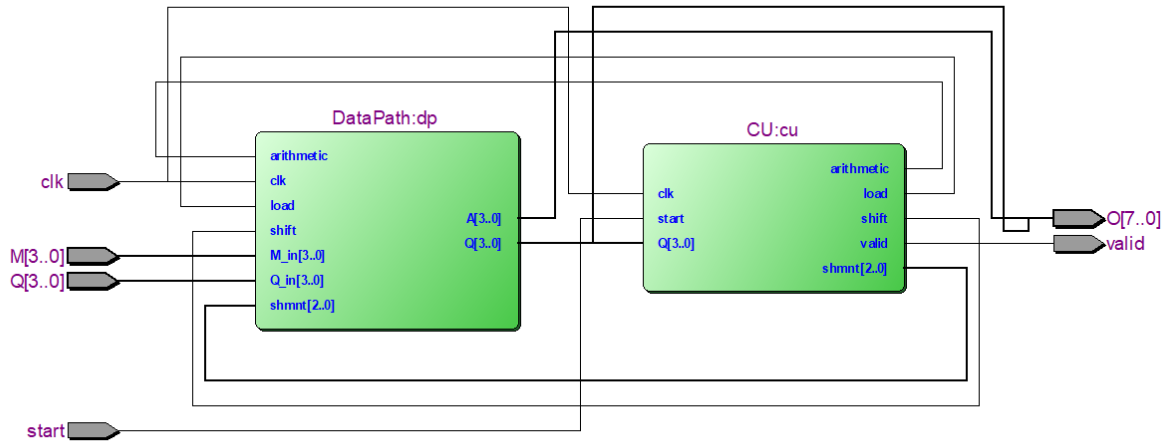
```
module Booth
      #(
            parameter nb = 4
      )
      (
            input clk,
            input start,
            input [nb-1:0] M,
            input [nb-1:0] Q,
            output valid,
            output [2*nb-1:0] O
      );


      // control wires
      wire load;
      wire arithmetic;
      wire shift;
      wire [$clog2(nb):0] shmnt;

      // instance of DataPath
      DataPath #(.nb(nb)) dp
```

```
	(
		.clk(clk),
		.M_in(M),
		.Q_in(Q),
		.A(O[2*nb-1:nb]),
		.Q(O[nb-1:0]),

		.load(load),
		.arithmetic(arithmetic),
		.shift(shift),
		.shmnt(shmnt)
	);

	// instance of CU (Control Unit)
	CU #(.nb(nb)) cu
	(
		.clk(clk),
		.start(start),
		.valid(valid),

		.Q(O[nb-1:0]),
		.load(load),
		.arithmetic(arithmetic),
		.shift(shift),
		.shmnt(shmnt)
	);

endmodule
```

این ماژول از دو ماژول دیگر استفاده شده است. در اینجا به عنوان پارامتر تعداد بیت‌های مضروب و مضروب‌الیه وارد می‌شود. (در تست بنچ آن را برابر با ۱۰ گرفتیم.) ورودی‌ها اعم از کلاک و پرچم شروع است. همچنین دو عددی که باید ضرب شوند با M, Q مشخص شده‌اند. همچنین خروجی O همان حاصل ضرب است که هنگامی که valid یک باشد این خروجی معتبر است.

همچنین یکسری سیم رابط میان این دو قرار داده‌ایم که مشخص کننده عمل انجام شده در کلاک فعلی در مسیرداده است. این سیم‌ها را به ترتیب با load, arithmetic, shift مشخص کرده‌ایم. همچنین shmnt میزان شیفتی است که باید در مسیر داده صورت بگیرد. البته این مقدار حداکثر به اندازه طول است که برابر لگاریتم عدد است که اینجا این‌را هم مشخص کرده‌ایم.

در زیر و پیش از پرداختن به دو ماژول دیگر به شمای RTL که پس از شبیه‌سازی توسط کوارتوس به آن دست یافتیم خوب است توجه کنیم تا عملکرد مدار را بهتر درک کنیم:



تصویر 3 شمای RTL ماژول بوث (Booth)

حال به بررسی ماژول مسیر داده یا همان DataPath می‌پردازیم. کد وریلاگ آن به صورت زیر است:

```
`timescale 1ns/1ns

module DataPath
	#(
		parameter nb = 4
	)
	(
		input clk,

		// input data
		input [nb-1:0] M_in,
		input [nb-1:0] Q_in,

		// output data
		output reg [nb-1:0] A,
		output reg [nb-1:0] Q,

		// control signals
		input load,
		input arithmetic,
		input shift,
		input [$clog2(nb):0] shmnt
	);

	reg [nb-1:0] M;			// Multiplicand
	reg LSB;			// Previous LSB of {A, Q}

	// data path logic
	always @(posedge clk)
	begin
```

```
            // if load=1, update registers
            if (load)
            begin
                  M <= M_in;
                  A <= 0;
                  Q <= Q_in;
                  LSB <= 0;
            end


            // if add/sub=1, do the math and update A
            else if (arithmetic)
            begin
                  if (Q[0] == 1 && LSB == 0)
                        A <= A - M;
                  else if (Q[0] == 0 && LSB == 1)
                        A <= A + M;
            end


            // if shift=1, signed shift right for 'shmnt' bits
            else if (shift)
            begin
                  {A, Q, LSB} <= $signed({A, Q, LSB}) >>> shmnt;
            end
      end
endmodule
```

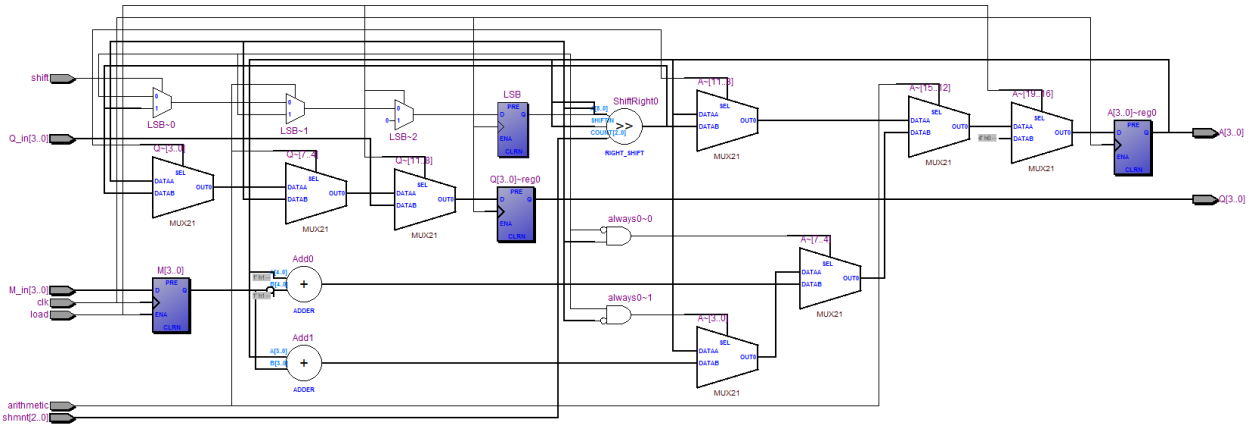
همانطور که در بخش مدار اصلی هم دیدیم ابتدا پارامتر تعداد بیت‌ها مشخص شده است. همچنین ورودی‌ها نیز مضروب و مضروب‌علیه هستند که در صورت نیاز باید اضافه شوند. همچنین سیگنال‌های ورودی از بخش کنترل نیز داده شده‌اند. درنهایت خروجی‌های A و Q هم همانطور که در شکل یک نمایش داده‌شده به عنوان خروجی تعیین شده‌اند.

همانطور که در الگوریتم بوث هم می‌دانیم در هر مرتبه به مولتی‌پلیکنت نیاز داریم پس آن را در یک reg به نام M می‌ریزیم. همچنین باید به بیت کم‌ارزش قبلی هم نیاز داریم که آن‌را در LSB ذخیره می‌کنیم.

درنهایت در هر مرتبه کلاک صفر به یک(سنکرون) مسیر داده باتوجه به سیگنال‌های کنترلی داده شده توسط واحد کنترل تصمیم می‌گیرید که

- لود کند: یک درخواست جدید از کاربر. پس M, Q باید ست شده و دیگر متغیرها باید به حالت پیشفرض که همان صفر است برگردند.
- عملیات حسابی انجام دهد. اگر از صفر به یک رسیده باشیم باید تفریق صورت گیرد. اگر از یک به صفر رسیدیم باید جمع اتفاق بیفتد و اگر تغییری نداشتیم در بیت کم‌ارزش، کاری نباید صورت گیرد.
- اگر باید شیفت اتفاق بیفتد هم کل AQLsb را بعنوان یک عدد باینری به مقدار مورد نیاز(shmnt) شیفت می‌دهیم.

در انتها نیز خوب است به شمای RTL این قسمت که با کوارتوس بدست آمده توجه کنیم:



تصویر 4 شمای RTL مسیر داده پیاده سازی شده.

درنهایت و پیش از تست مدار، به ماژول واحد کنترل توجه می‌کنیم. کد وریلاگ آن به صورت زیر است:

```
`timescale 1ns/1ns

module CU
	#(
		parameter nb = 4
	)
	(
		// control inputs
		input clk,
		input start,

		// control output
		output valid,

		// status input
		input [nb-1:0] Q,

		// control outputs
		output load,
		output arithmetic,
		output shift,
		output [$clog2(nb):0] shmnt
	);

	// one-hot FSM
	reg[3:0] cs; // holds current state
	reg[3:0] ns; // holds next state

	localparam  LOAD = 0,
				ARTH = 1,
				SHFT = 2,
				DONE = 3;
```

```
// counter of remaining shifts
reg[$clog2(nb):0] counter;

// generate control outputs
assign load = cs[LOAD];
assign arithmetic = cs[ARTH];
assign shift = cs[SHFT];
assign valid = cs[DONE];

// generate next state
always @(cs, counter)
begin
      ns <= 0;

      if(cs[LOAD])
            ns[ARTH] <= 1'b1;

      if(cs[ARTH])
            ns[SHFT] <= 1'b1;

      if(cs[SHFT])
            if (counter > shmnt)
                  ns[ARTH] <= 1'b1;
            else
                  ns[DONE] <= 1'b1;

      if(cs[DONE])
            ns[DONE] <= 1'b1;
end

// at clock, go to next state
always @(posedge clk)
begin

      if (start)
      begin

            //
            cs <= 1;
            counter <= nb;
      end

      else
      begin
            cs <= ns;

            // if old state was a shift, update counter
            if (cs[SHFT])
                  counter <= counter - shmnt;
      end

end
```

```
	// diff_pairs : determine which pair bits of Q are different
	wire [nb-1:0] diff_pairs = ( Q ^ (Q >> 1) ) | (1'b1 << (nb-1'b1));

	// lsb_one : position of right-most '1' bit in diff_pairs
	reg [$clog2(nb):0] lsb_one;

	integer i;
	always @(*)
	begin

		// reset lsb_one
		lsb_one = 0;

		// loop through pairs
		for (i = 1; i <= nb; i = i+1)
		begin
			if (diff_pairs[i-1] && !lsb_one)
				lsb_one = i;
		end

	end

	// shift amount : min(lsb_one, number of shifts left)
	assign shmnt = (counter>lsb_one) ? lsb_one : counter;

endmodule
```

در ابتدا پارامتر تعداد بیت‌ها دوباره به این ماژول هم داده شده است. همچنین سیگنال‌های کنترلی شروع و کلاک به همراه مضروب‌الیه به عنوان ورودی، سیگنال‌های کنترلی برای مسیرداده به همراه ولید بودن یا نبودن نیز به عنوان خروجی مشخص شده‌اند.

استیت‌های کلی را به صورت دو رجیستر و به صورت یک عدد چهاربیتی در cs(current state) و ns(next state) ذخیره کردیم. بیت‌ها به ترتیب نماینده حالت‌های لود، عملیات حسابی، شیفت و اتمام‌کار هستند. پس از آن یک شمارنده counter ساخته‌ایم که در آن تعداد شیفت‌های باقی‌مانده را نگه‌می‌داریم تا بیش‌تر از مقدار نیاز در الگوریتم پیش نرفته در هنگام پایان سیگنال اعتبار خروجی (valid) را یک کنیم.

پس از این مقدایر سیگنال‌های کنترلی را به وسیله assign مشخص کرده‌ایم.

پس از این دستور، یک بلاک همیشگی تعریف کرده‌ایم (always) که در صورت تغییر تعداد شیفت باقی‌مانده در شمارنده و یا حالت کنونی متغیر استیت بعدی را صفر کرده و حالت‌ها را به صورت زیر تغییر می‌دهد:

- لود به عملیات حسابی
- عملیات حسابی به شیفت
- شیفت به عملیات منطقی و یا به حالت اتمام‌کار با توجه به بزرگ‌تر بودن یا نبودن شمارنده از تعداد شیفت‌های مورد نیاز
- حالت اتمام‌کار به اتمام کار

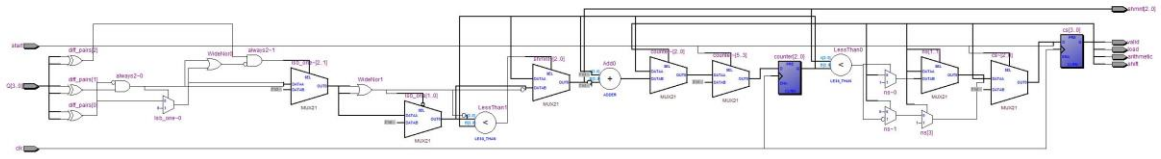
در بلوک بعدی پس از مقدار دهی حالت بعدی، به حالت بعدی می‌رویم. به این‌صورت که:

- اگر سیگنال شروع آمده بود، شمارنده را برابر تعداد اجرای عملیات (همان nb) قرار می‌دهیم. و پس از آن هم حالت کنونی را برابر لود.
- در غیر این‌صورت، متغیر حالت بعدی که در بلوک always قبلی در ns نوشته شده به عنوان حالت کنونی تنظیم می‌شود. همچنین چک می‌شود که اگر حالت کنونی شیفت بود، از شمارنده یکی کم می‌کنیم.

پس از این عملیات نوبت تعیین شیفت است. ابتدا Qو Q>>1 را باهم XoR کرده‌ایم و در diff_pairs ریخته‌ایم. اگر حاصل یک بود یعنی دوبیت کناری متفاوتند و شیفتی درکار نیست و باید پس از محاسبه حسابی از روی آن‌ها رد شویم. این بردار با بردار (1 – nb) >> 1 هم OR شده‌است تا اگر Q صفر بود، بالاترین بیت‌آن یک شود تا shmnt به درستی بعدا ست شود.

در ادامه نیز تنها در lsb_one کم‌ارزش ترین یک diff_pairs را یافته‌ایم تا بتوانیم به کمک آن shmnt را ست کنیم. این کار به سادگی انجام شده. البته دقت کنید که shmnt به مینیمم شمارنده و کم‌ارزش‌ترین ست می‌شود تا الگوریتم به درستی کار کند.

در انتها خوب است به شمای RTL این ماژول هم بپردازیم در زیر آمده است:



تصویر 5 شمای RTL واحد کنترل. برای دیدن بهتر به فایل موجود در نتایج مراجعه کنید.

نتایج

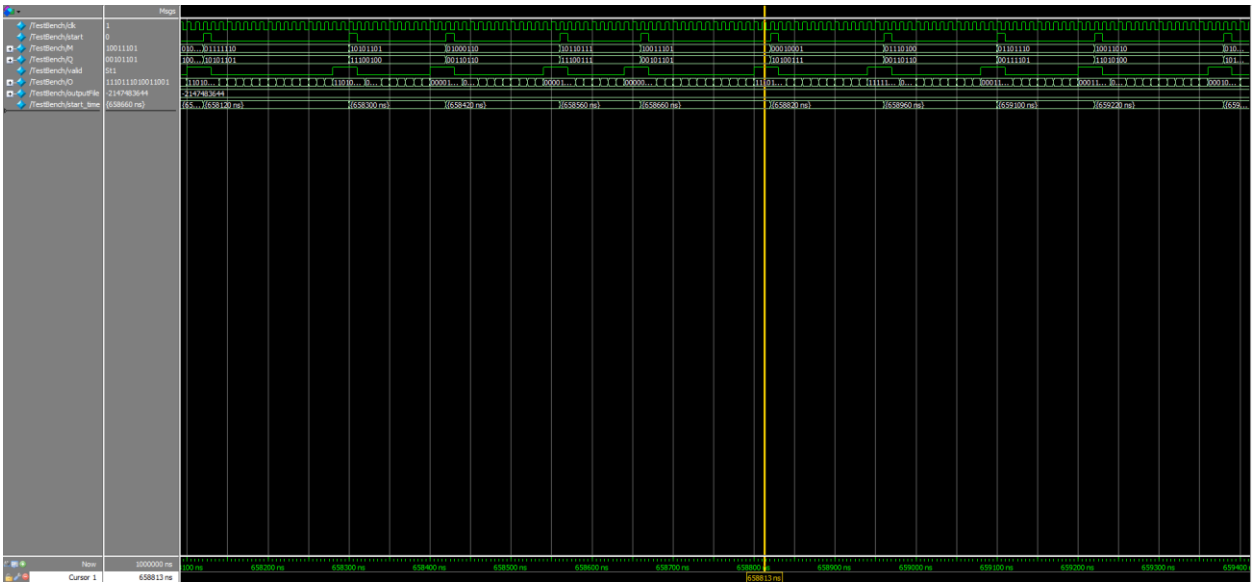
در تست‌بنچ با درنظر گرفتن ورودی‌ها به عنوان اعداد ۸ بیتی ضرب‌ها را انجام دادیم. همچنین در این ماژول خروجی‌ها را در یک فایل هم ریختیم تا راحت‌تر بتوانیم عملکرد مدارمان را تست کنیم:

```
`timescale 1ns/1ns

module TestBench
	#(
		parameter nb = 8
	);

	reg clk;
	reg start;
	reg [nb-1:0] M;
	reg [nb-1:0] Q;
```

```
	wire valid;
	wire [2*nb-1:0] O;

	integer outputFile;

	Booth #(.nb(nb)) booth
	(
		.clk(clk),
		.start(start),
		.M(M),
		.Q(Q),
		.valid(valid),
		.O(O)
	);

	// clock generator
	initial
		clk = 1;
	always
		#5 clk = ~clk;

	// variable to save start-time of each test
	time start_time;

	// first test
	initial
	begin

		outputFile = $fopen("output.txt", "w");

		start <= 1;
		M <= 0;
		Q <= 0;
		start_time = $time;

		#10;
		start <= 0;
	end

	initial begin
		#1000000
		$fclose(outputFile);
	end

	// always at end of the test


	always @(posedge valid)
	begin
		// display result of previous test when it's done
		$fdisplay(outputFile, "%d, %d, %d, %0t",
				$signed(M), $signed(Q), $signed(O), $time -
start_time);
```

```
            // generate next test
            #20;
            start <= 1;
            M <= {$random};
            Q <= {$random};
            start_time = $time;

            #10;
            start <= 0;
        end


endmodule
```

خروجی موج را در فایل زیر می‌بینید:



*تصویر 6 خروجی موج تست بنچ. حاصل ضرب -99 * 45 = -4455*

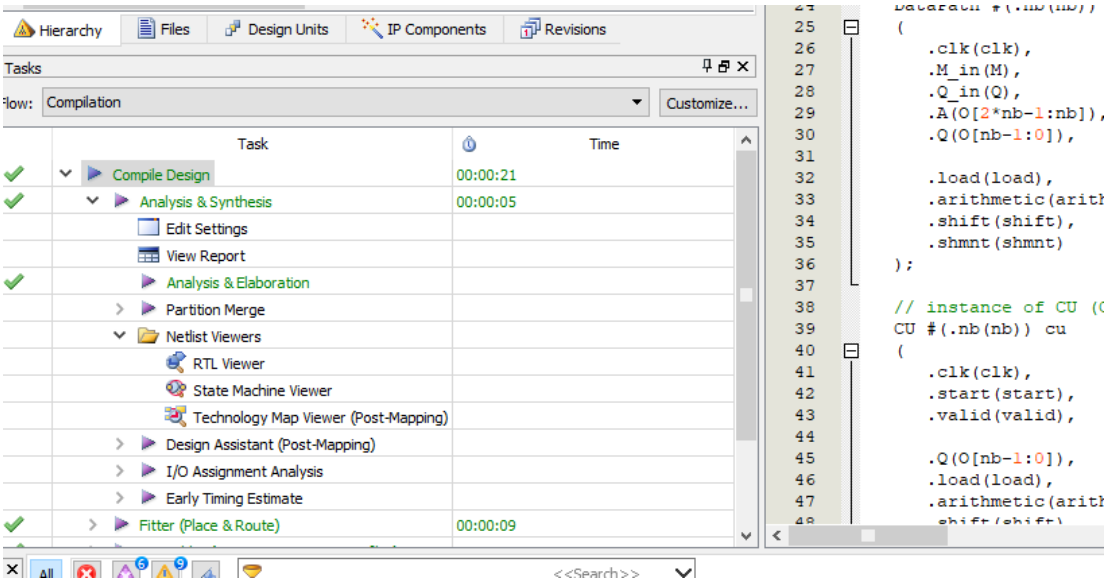
چون تست‌بنچ ما اعداد را به صورت رندم در هم ضرب می‌کند، برای همین خروجی بسیار زیادی (۷۶۰۰ نمونه متفاوت) از آن گرفتیم و آن‌ها را در یک فایل ذخیره کردیم. پس از این هم این فایل‌ها را درون یک فایل اکسل ریختیم که در فولد نتایج می‌توانید آن را مشاهده کنید.

G2 =COUNTIF(F2:F7659,TRUE)/7658

A	B	C	D	E	F	G
Q	M	Our Q*M	time	real Q*M	Our == real ?	accuracy
36	-127	-4572	80	-4572	TRUE	1
9	99	891	100	891	TRUE	
13	-115	-1495	120	-1495	TRUE	
101	18	1818	120	1818	TRUE	
1	13	13	100	13	TRUE	
118	61	7198	100	7198	TRUE	
-19	-116	2204	100	2204	TRUE	
-7	-58	406	100	406	TRUE	
-59	-86	5074	180	5074	TRUE	
-27	119	-3213	100	-3213	TRUE	
18	-113	-2034	80	-2034	TRUE	
-14	-50	700	100	700	TRUE	
-24	-59	1416	120	1416	TRUE	
92	-67	-6164	120	-6164	TRUE	
45	101	4545	140	4545	TRUE	
99	10	990	120	990	TRUE	
-128	32	-4096	80	-4096	TRUE	
-86	-99	8514	120	8514	TRUE	
-106	19	-2014	100	-2014	TRUE	
13	83	1079	140	1079	TRUE	
107	-43	-4601	160	-4601	TRUE	
2	-82	-164	140	-164	TRUE	
29	-49	-1421	80	-1421	TRUE	
35	10	350	120	350	TRUE	
-54	60	-3240	80	-3240	TRUE	
-14	-118	1652	140	1652	TRUE	
65	-40	-2600	100	-2600	TRUE	
120	-119	-14280	120	-14280	TRUE	
-21	-74	1554	140	1554	TRUE	
-58	-82	4756	140	4756	TRUE	
-68	42	-2856	160	-2856	TRUE	
11	113	1243	100	1243	TRUE	

تصویر 7 نتایج و مقایسه آن‌ها با مقدار واقعی (توجه کنید که دقت برای ۷۶۰۰ تست یک است.)

بدیهی است که مدار سنتز شده و شماهای RTL بالا گواه این مطلبند، اما نتیجه کامپایل را می‌توانید در زیر هم ببینید:



تصویر 8 نتیجه کامپایل کوارتوس