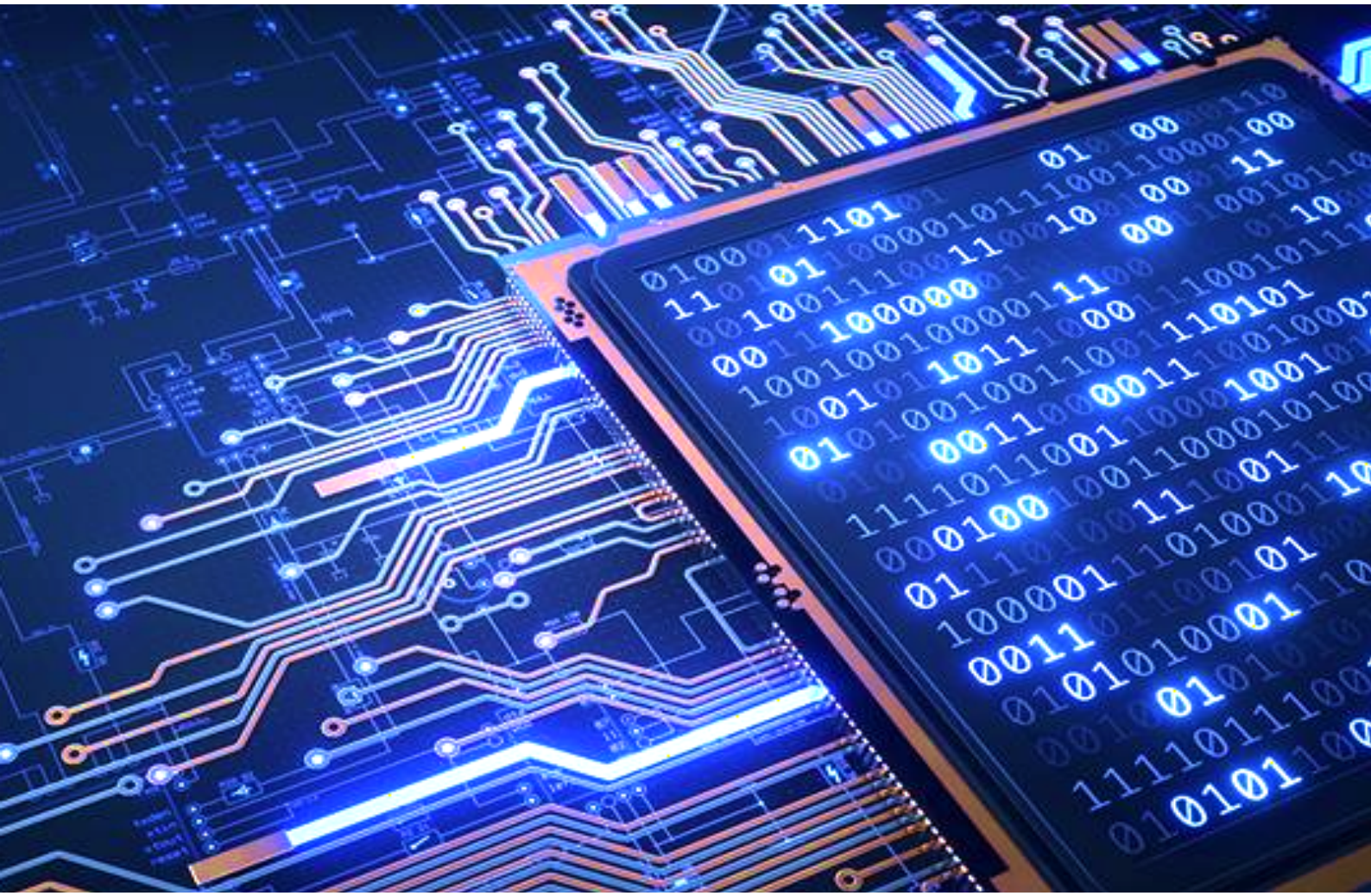
بهار ۱۴۰۳

تمرین کامپیوتری دوم

"مدار های منطقی"

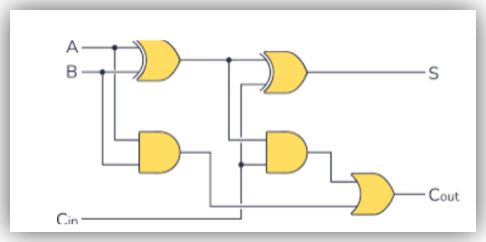
این تمرین با استفاده از ۲ روز Grace تحویل داده شد.



مجید صادقی نژاد ۸۱۰۱۰۱۴۵۹

گام اول:

ماژول خواست شده که در شکل آمده با استفاده از تاخیر های خواسته شده پیاده سازی می کنیم(فایل Q1.sv):



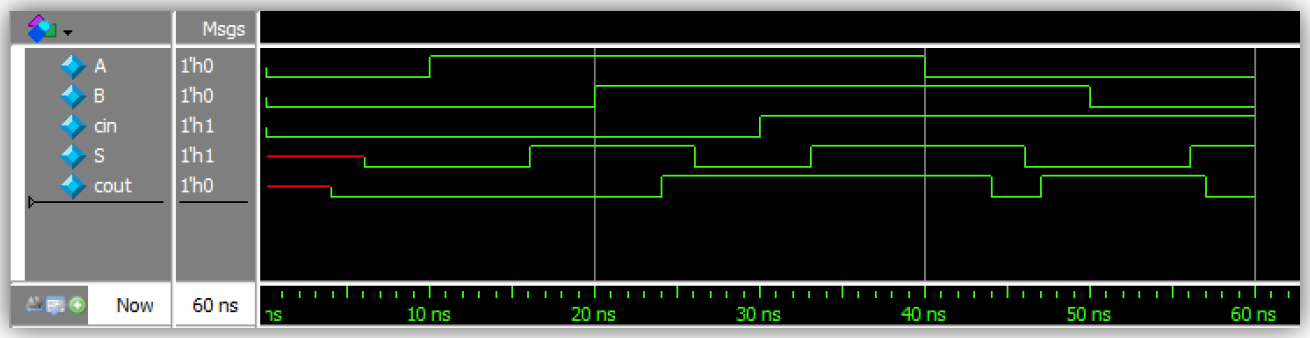

```
`timescale 1ns/1ns
module fullAdder(input A,B,cin, output S,cout);
 wire w1,w2,w3;

 xor #(3) x1(w1,A,B);
 and #(2) a1(w2,A,B);
 xor #(3) x2(S,w1,cin);
 and #(2) a2(w3,w1,cin);
 or #(2) o1(cout,w2,w3);

endmodule
```

سپس در تست طراحی شده (موجود در فایل Q1_TB.sv) حالات زیر را هر ۱۰ نانو ثانیه تغییر می دهیم (A,B,C_{in}):

(۰,۰,۱) -> (۰,۱,۱) -> (۱,۱,۱) -> (۱,۱,۰) -> (۱,۰,۰) -> (۰,۰,۰)

```
`timescale 1ns/1ns
module fullAdd_TB();
 logic A,B,cin;
 wire S,cout;

 fullAdder UUT1(A,B,cin,S,cout);

 initial begin
  A = 1'b0;
  B = 1'b0;
  cin = 1'b0;
  #10;
  A = 1'b1 ;
  #10;
  B = 1'b1 ;
  #10;
  cin = 1'b1;
  #10;
  A = 1'b0;
  #10;
  B= 1'b0;
  #10;
 end
endmodule
```

که با این تست شکل موج زیر ایجاد می شود:



در ۱۰ نانوثانیه اول با توجه به این که C_{out} توسط دو گیت and و یک گیت or تولید می شود و هر دو گیت and یکی از ورودی هایشان دارای مقدار صفر پس تاخیر به حداقل رسیده و برابر ۴ نانوثانیه برای C_{out} می شود و مقدار S با توجه به اینکه توسط دو گیت xor تشکیل می شود و هر گیت دارای تاخیر ۳ نانوثانیه است پس از ۶ نانوثانیه آماده می شود، در ۱۰ نانوثانیه دوم نیز به همین منوال S پس از ۶ نانوثانیه آماده می شود و نتیجه ی C_{out} نیز تاخیر نمی کند برای ۱۰ نانوثانیه سوم نیز S,C_{out} به ترتیب پس از ۴ و ۶ نانوثانیه آماده می شوند که علت آن در ابتدا توضیح داده شده در ۱۰ نانوثانیه چهارم C_{out} بدون تغیر باقی می ماند اما مقدار S برخلاف مراحل قبل پس از فقط ۳ نانوثانیه آماده می شود زیرا از دو گیت xor تشکیل دهنده ی S فقط یکی از آن ها ورودی اش تغییر می کند و فقط یک تاخیر ۳ نانوثانیه ای داریم، در ۱۰ نانوثانیه ی پنجم نتیجه ی S همانند بخش های قبلی پس از ۶ نانوثانیه آماده می شود اما C_{out} دارای یک glitch است و پس از ۷ نانوثانیه آماده می شود که ۷ نانو ثانیه همان زمان بحرانی مدار نیز هست علت نیز این است که تغییر ورودی ها در این بخش به گونه ای است که سبب می شود تمام ۳ گیت and,or,xor که در مسیر هستند فعال شوند همچنین از آنجا که عملکرد xor ۱ نانوثانیه با باقی گیت ها تفاوت دارد سبب ایجاد glitch نیز می شود در ۱۰ نانوثانیه آخر نیز نتیجه S همانند بخش های قبل پس از ۶ نانوثانیه و نتیجه C_{out} نیز دقیقا مانند بخش قبل بعد از ۷ نانوثانیه آماده می شود اما بدون وجود glitch.
همچنین پاسخ مدار در هر بازه به نحوه زیر است که می توان درستی آن ها را تایید کرد(S,C_{out}):

(۰,۰) -> (۱,۰) -> (۰,۱) -> (۱,۱) -> (۰,۱) -> (۱,۰)

گام دوم:

ماژول ۴ bit ripple carry adder را طراحی می کنیم (موجود در فایل Q۲.sv):

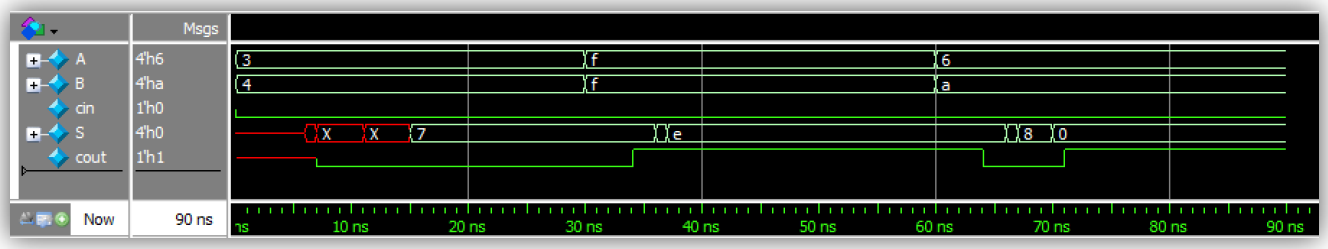
```
`timescale 1ns/1ns
module RCA4(input [3:0] A,B, input cin, output [3:0] S, output cout);
 wire w0,w1,w2;

 fullAdder fa1(A[0],B[0],cin,S[0],w0);
 fullAdder fa2(A[1],B[1],w0,S[1],w1);
 fullAdder fa3(A[2],B[2],w1,S[2],w2);
 fullAdder fa4(A[3],B[3],w2,S[3],cout);

endmodule
```

و سپس با تست های زیر آن را آزمایش می کنیم که هر ۳۰ نانوثانیه به تست بعدی می رویم (A,B,C_{in}):

(۳,۴,۰) , (۱۵,۱۵,۰) , (۶,۱۰,۰)

```
`timescale 1ns/1ns
module RCA4_TB();
 logic [3:0] A,B;
 logic cin;
 wire [3:0] S;
 wire cout;

 RCA4 UUT1(A,B,cin,S,cout);

 initial begin
  A = 4'd3;
  B = 4'd4;
  cin = 1'b0;
  #30;
  A = 4'd15 ;
  B= 4'd15;
  #30;
  A = 4'd6 ;
  B = 4'd10 ;
  #30;
 end
endmodule
```

که شکل موج زیر در نتیجه این تست تولید می شود (موجود در فایل Q۲.sv):



همانطور که مشاهده می شود مجددا به علت این که C_{out} از گیت های and تشکیل شده در برخی مقادیر و برخی از تاخیر مقادیر عملکرد سریع تری را دارد اما گاهی با بعضی از مقادیر شامل زمان تاخیر بحرانی (۲۸ نانوثانیه) می شود و تاخیر بیشتری دارد همچنین به علت این که بیت های مختلف S نسبت به هم با تاخیر تولید می شوند مقدار S در هر بازه ی ۳۰ نانوثانیه چندین بار تغییر می کند و در نهایت درست می شود که تاخیر این بیت های مختلف هم با توجه به مقادیر و تغییر مقادیر رخ می دهد. در نهایت اما جواب نهایی درست می شود و به ترتیب زیر است:

(۷,۰) -> (۱۴,۱) -> (۰,۱) (۷) , (۱۴+۱۶ = ۳۰) , (۰+۱۶ = ۱۶) ۷,۳۰,۱۶

گام سوم:

ابتدا ماژول خواسته شده را طراحی می کنیم(موجود در فایل Q۳.sv):

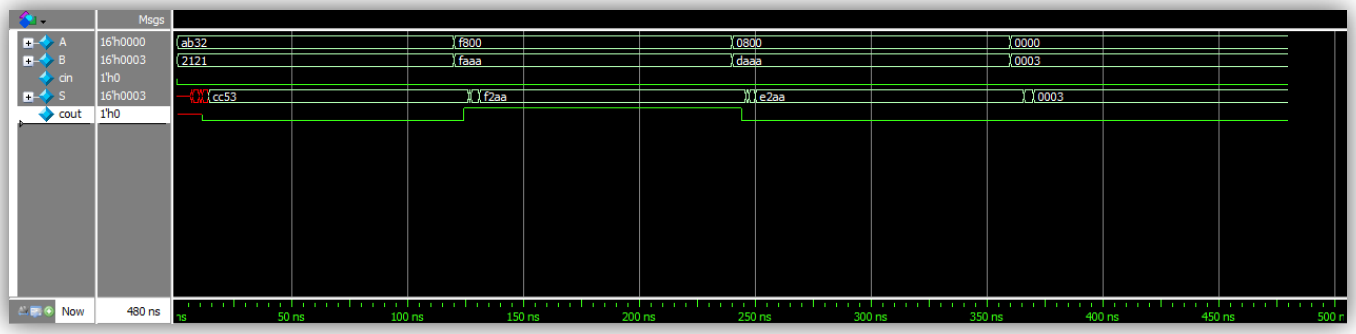
```
`timescale 1ns/1ns
module RCA16(input [15:0] A,B, input cin, output [15:0] S, output cout);
 wire w0,w1,w2;

 RCA4 fa1(A[3:0],B[3:0],cin,S[3:0],w0);
 RCA4 fa2(A[7:4],B[7:4],w0,S[7:4],w1);
 RCA4 fa3(A[11:8],B[11:8],w1,S[11:8],w2);
 RCA4 fa4(A[15:12],B[15:12],w2,S[15:12],cout);

endmodule
```

سپس تستی طراحی می شود (موجود در فایل Q۳_TB.sv) که سه جمع زیر را با فاصله های ۱۲۰ نانوثانیه ای به مدار می دهد توجه شود که اعداد در این تست به صورت hexadecimal هستند و تمام C_{in} ها برابر صفر هستند(A,B):

(ab۳۲ , ۲۱۲۱) -> (f۸۰۰ , faaa) -> (۰۸۰۰ , daaa) -> (۰۰۰۰ , ۰۰۰۳)

```
`timescale 1ns/1ns
module RCA16_TB();
 logic [15:0] A,B;
 logic cin;
 wire [15:0] S;
 wire cout;

 RCA16 UUT1(A,B,cin,S,cout);

 initial begin
  A = 16'hab32;
  B = 16'h2121;
  cin = 1'b0;
  #120;
  A = 16'hf800 ;
  B = 16'hfaaa;
  #120;
  A = 16'h0800 ;
  B = 16'hdaaa;
  #120;
  A = 16'h0000 ;
  B = 16'h0003 ;
  #120;
 end
endmodule
```

و در نتیجه این تست، شکل موج زیر تشکیل می شود:



در ۱۲۰ نانوثانیه ی اول S,C_{out} به ترتیب بعد از ۱۱ نانوثانیه و ۱۴ نانوثانیه آماده می شوند در ۱۲۰ نانوثانیه دوم آن ها بعد از ۴ و ۱۱ نانوثانیه آماده می شوند، در ۱۲۰ نانوثانیه سوم پس از ۴ و ۱۰ نانوثانیه و در ۱۲۰ نانوثانیه چهارم پس از ۰ و ۱۰ نانوثانیه آماده می شوند. همچنین خروجی ها برای هر ورودی به شکل زیر است که درستی مدار را تایید می کند (S,C_{out}):

(cc۵۳ , ۰) -> (f۲aa , ۱) -> (e۲aa , ۰) -> (۰۰۰۳ , ۰)

برای محاسبه تاخیر مدار به صورت تئوری تاخیر مسیر بحرانی مدار را محاسبه می کنیم در گام اول دیدیم تاخیر بحرانی هر full adder برابر ۷ نانوثانیه است بنابراین برای یک ۴ bit ripple carry adder برابر با ۴*۷ = ۲۸ نانوثانیه است و برای یک ۱۶ bit ripple carry adder برابر با ۱۶*۷=۱۱۲ نانوثانیه است همچنین برای محاسبه S طبق گام اول می دانیم تاخیر حداکثر برابر ۶*۱۶=۹۶ ثانیه است اما در ابتدای توضیحات دیدیم که تاخیر مدار ۱۶ بیتی بسیار کمتر از مقدار های محاسبه شده است علت این موضوع به طور کامل در گام اول توضیح داده شد از آن جا که تشکیل دهنده ی C_{out} در مراحل مختلف گیت های and هستند با سرعت عملکرد آن ها در هنگام تغییر وضعیت وابستگی به ورودی های آن ها دارد به طوری که اگر یکی از ورودی های آن ها برابر صفر باشد گیت and عملکرد خود را آغاز می کند همچنین این موضوع برای مقدار S نیز صادق است زیرا ماهیت تشکیل دهنده این خروجی نیز گیت های متصل به هم xor هستند اما در حالت تئوری ما به دنبال محاسبه بدترین تاخیر هستیم که فقط با تعداد محدودی از ورودی ها می توان به این حد بالا از تاخیر رسید.

گام چهارم:

در ابتدا ماژول mux۲:۱ و csa ۱۶bit را طراحی می کنیم(موجود در فایل های Q۴.sv , MUX.sv):

```
`timescale 1ns/1ns
module mux2(input a,b,select, output out);
 wire w0,w1,w2,w3;

 not n1(w0,select);
 and a1(w1,select,b);
 and a2(w2,w0,a);
 or o1(w3,w1,w2);
 and #(2) a3(out,w3,w3);

endmodule
```

```
`timescale 1ns/1ns
module CSA16(input [15:0] A,B, input cin, output [15:0] S , output cout);
 wire couts [3:0];
 wire [1:0] co [2:0];
 wire [3:0] w1 [2:0];
 wire [3:0] w2 [2:0];

 RCA4 f1(A[3:0],B[3:0],cin,S[3:0],couts[0]);

 genvar i;
 generate
  for (i = 0; i < 3; i = i + 1) begin : ripple4s
   RCA4 rca1(A[3+(i+1)*4 : (i+1)*4], B[3+(i+1)*4 : (i+1)*4], 1'b0, w1[i], co[i][0]);
   RCA4 rca2(A[3+(i+1)*4 : (i+1)*4], B[3+(i+1)*4 : (i+1)*4], 1'b1, w2[i], co[i][1]);

   mux2 muxs1(w1[i][0],w2[i][0],couts[i],S[(i+1)*4+0]);
   mux2 muxs2(w1[i][1],w2[i][1],couts[i],S[(i+1)*4+1]);
   mux2 muxs3(w1[i][2],w2[i][2],couts[i],S[(i+1)*4+2]);
   mux2 muxs4(w1[i][3],w2[i][3],couts[i],S[(i+1)*4+3]);

   mux2 carryMux(co[i][0],co[i][1],couts[i],couts[i+1]);
  end
 endgenerate

 and a1(cout,couts[3],couts[3]);

endmodule
```

سپس تستی برای آزمایش ماژول طراحی شده و مقایسه آن با rca ۱۶bit طراحی می کنیم(موجود در Q۴_TB.sv):

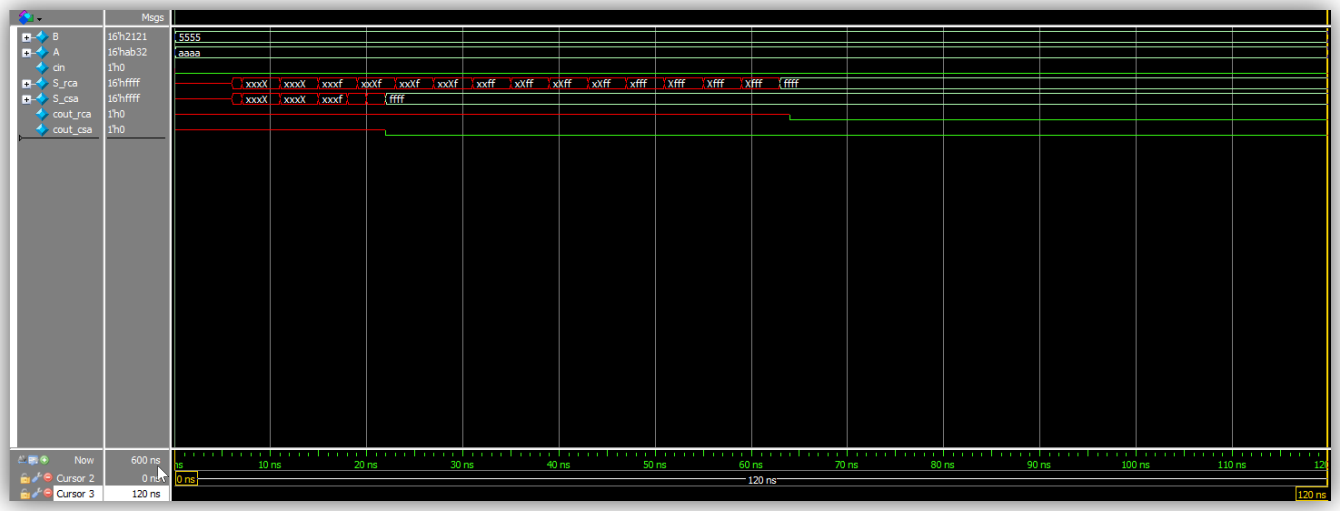
```
`timescale 1ns/1ns
module CSA_RCA16_TB();
 logic [15:0] A,B;
 logic cin;
 wire [15:0] S_rca;
 wire [15:0] S_csa;
 wire cout_rca ;
 wire cout_csa ;

 RCA16 UUT1(A,B,cin,S_rca,cout_rca);
 CSA16 UUT2(A,B,cin,S_csa,cout_csa);

 initial begin
  A = 16'b1010101010101010;
  B = 16'b0101010101010101;
  cin = 1'b0;
  #120
  A = 16'hab32;
  B = 16'h2121;
  #120;
  A = 16'hf800 ;
  B = 16'hfaaa;
  #120;
  A = 16'h0800 ;
  B = 16'hdaaa;
  #120;
  A = 16'h0000 ;
  B = 16'h0003 ;
  #120;

 end
endmodule
```

این تست به گونه ای طراحی شده که پنج مقدار متفاوت را با تاخیر های ۱۲۰ ثانیه ای به هر دو جمع کنند می دهد اما نکته اصلی در رابطه با مقادیری می باشد که در بار اول به جمع کننده ها داده می شود این مقادیر باعث به وجود آمدن تاخیر بحرانی در rca ۱۶bit می شود چیزی در مراحل قبلی مکررا به آن اشاره شد.اعداد داده شده در مبنای hexadecimal به صورت زیر است و در تمام تست ها C_{in} برابر صفر بوده است.(A,B):

(aaaa , ۵۵۵۵) -> (ab۳۲ , ۲۱۲۱) -> (f۸۰۰ , faaa) -> (۰۸۰۰ , daaa) -> (۰۰۰۰ , ۰۰۰۳)

و بر اساس این تست شکل موج زیر تولید می شود:

نمای کلی شکل موج

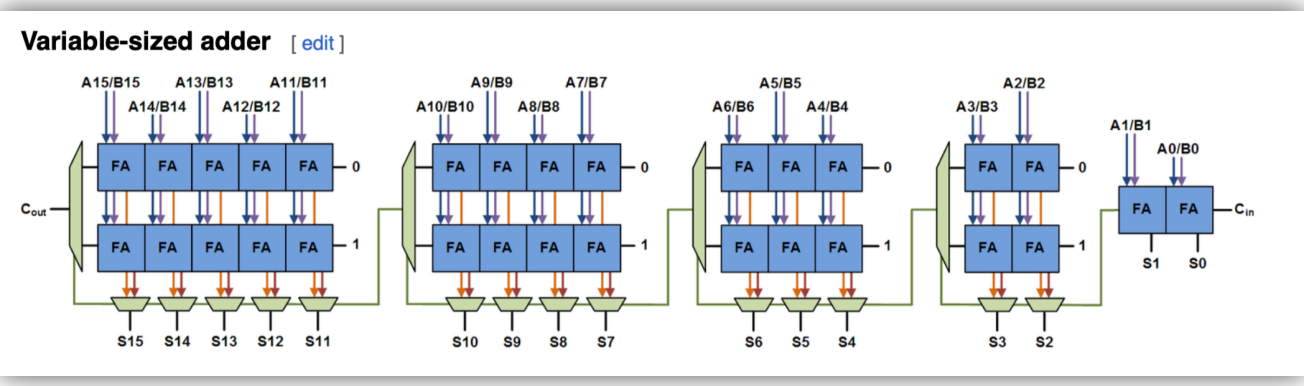


نمای جزئی تر بخش اول شکل موج

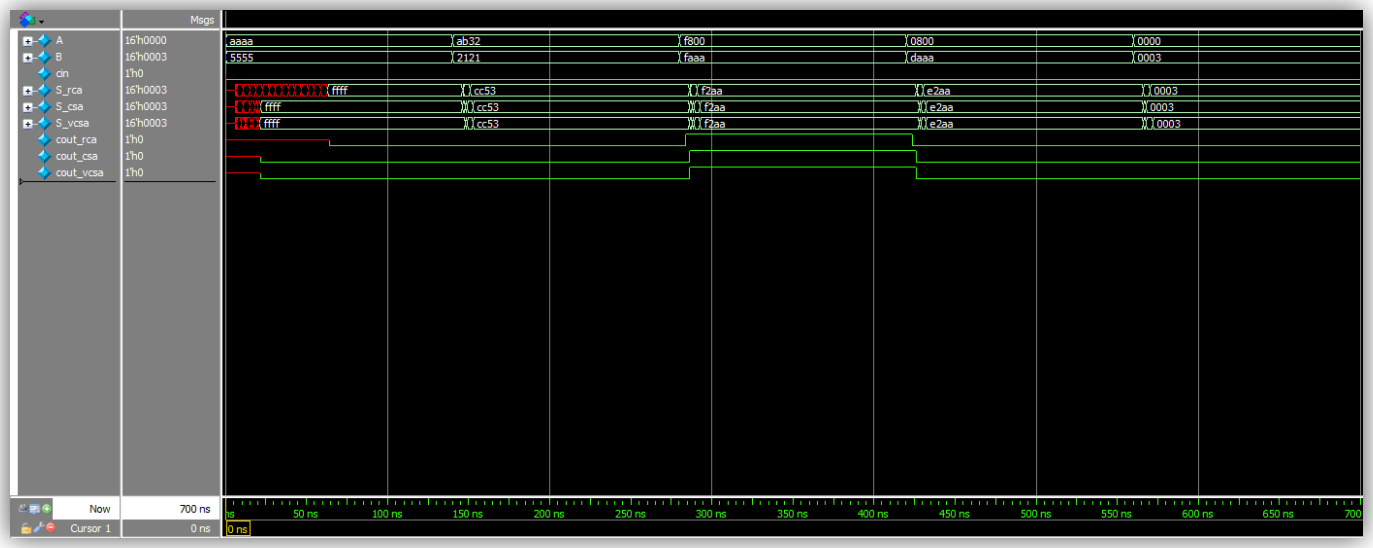
همانطور که در شکل مشخص است هر دو جمع کننده دارای خروجی های کاملا برابر و صحیح هستند اما در رابطه با تاخیر ها نکته ی جالبی وجود دارد همانطور که در ابتدای این گام اشاره شد مقادیر بخش اول تست به دلیل خاص بودن سبب ایجاد تاخیر بحرانی در مدار full adder می شوند و بنابراین به خوبی تفاوت زمانی دو جمع کننده و برتری زمانی جمع کننده ی CSA را نشان می دهند (شکل دوم). اما در بخش های دوم ، سوم و چهارم تفاوت تاخیر دو جمع کننده به شدت اندک بوده و حتی تاخیر زمانی CSA نسبت به RCA بیشتر است که این موضوع در اثر اضافه کردن گیت های MUX دارای تاخیر به مدار جمع کننده ی CSA است که فقط سبب افزوده شدن تاخیر به مدار می شوند در صورتی که مدار RCA به خاطر ماهیت اعداد ورودی که در بخش های قبلی کاملا توضیح داده شد، همان خروجی ها را بدون اعمال تاخیر MUX تولید می کند. و در خروجی پنج مجددا برتری زمانی اندک CSA مشاهده می شود که مجددا به خاطر ماهیت اعداد ورودی می باشد.

گام پنجم:

پس از طراحی ماژول variable CSA که نحوه پیاده سازی آن در شکل زیر نمایش داده شده است این ماژول را را با تست گام قبلی آزمایش می کنیم.(این بار تاخیر اعمال مقادیر برابر ۱۴۰ نانوثانیه است)(موجود در فایل های Q۵.sv , Q۵_TB.sv):



و به شکل موج زیر می رسیم:



همانطور که در گام قبلی دیدیم بخش اول تست از آنجا که باعث به وجود آمدن تاخیر بحرانی در RCA می شود تاخیر زیادی را بر این جمع کننده اعمال می کند و دو جمع کننده دیگر هر دو این مشکل را رفع کرده اند و خروجی خود را با اختلاف بسیار زیادی، زودتر ارائه می دهند. در مقایسه ی دو جمع کننده CSA و V-CSA مشاهده می شود که تاخیر خروجی آن ها نسبت به هم عملا صفر است علت این موضوع نیز مورد استفاده و کاربرد V-CSA است این جمع کننده تاخیر فول ادر را به تاخیر مالتی پلکسر ترجیح می دهد (این موضوع در بخش سوالات نشان داده خواهد شد)و بنابراین اگر تاخیر فول ادر کوچکتر مساوی دو برابر تاخیر مالتی پلکسر های مصرفی در مدار باشد استفاده از جمع کننده V-CSA به صرفه تر از CSA است اما در مدار های این تمرین چنین شرطی برقرار نیست و لذا استفاده از V-CSA نه تنها باعث افزایش سرعت نمی شود بلکه به میزان اندکی کندتر از CSA عادی می باشد. در باقی تست ها نیز این موضوع مشهود است با این تفاوت که همانطور که در گام های قبلی به توضیح داده شد، به علت ماهیت اعداد ورودی RCA عملکرد سریع تری نسبت به این دو دارد.

سوالات:

۱ – با توجه به شکل دو مدار و در نظر گرفتن این موضوع که تقریبا در اکثر موارد فول ادر تاخیر بیشتری نسبت به مالتی پلکسر دارد می توان مسیر و تاخیر بحرانی هر یک از این دو مدار را بدین گونه حساب کرد:

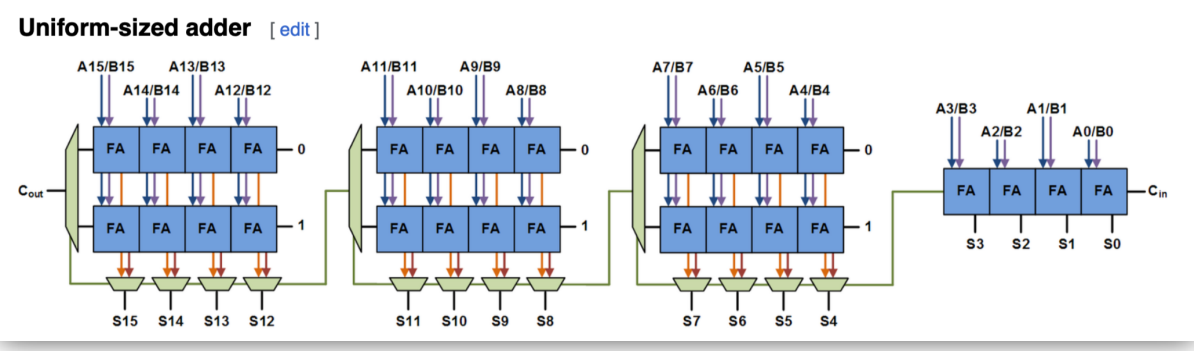
CSA: ۴*FA + ۳*MUX

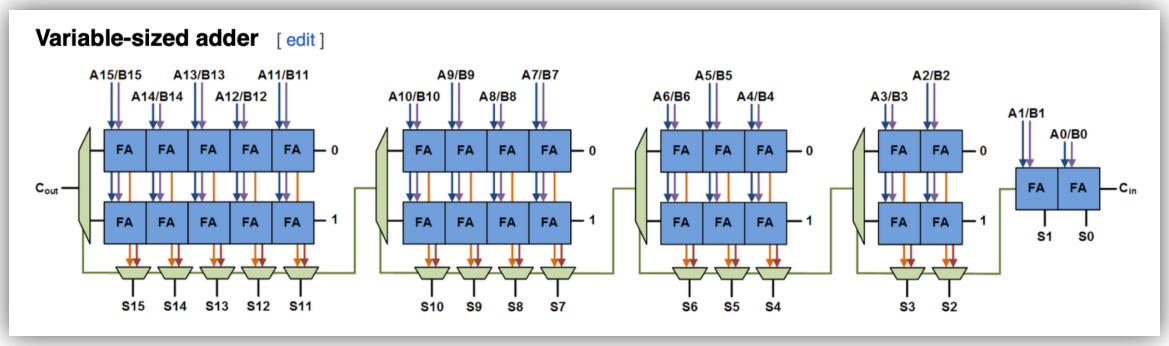
V-CSA: max(۵*FA+MUX , ۲*FA+۴*MUX) = ۵*FA+MUX

و اگر بخواهیم تفاوت تاخیر بین این دو جمع کننده را به دست آوریم:

Delay(CSA) – Delay(V-CSA) = -FA + ۲*MUX

که نشان می دهد در صورتی که تاخیر فول ادر کمتر از دو برابر تاخیر مالتی پلکسر باشد با V-CSA تاخیر کمتری را خواهیم داشت.





در رابطه با مزایای این جمع می کننده می توان به تاخیر کمتر در صورت وجود شرط به دست آمده اشاره کرد اما در رابطه با معایب بحث اضافه شدن تعداد مالتی پلکسر ها و بزرگتر شدن و پیچیده شدن مدار نسبت به حالت عادی اشاره کرد.

۲- از تست پنجم نتیجه می شود مدار در هر مرحله به شرط داشتن شرط هایی که در هر گام بیان شد عملکرد بهتری از لحاظ سرعت و کاهش تاخیر را نشان می دهد اما باید توجه داشت که ماهیت اعداد مورد استفاده نیز در تاخیر جمع کننده ها تاثیر گذار است. در ارتباط با بحث trade off می دانیم که با کاهش تاخیر مدار در هر مرحله به اندازه ، تعداد گیت و پیچیدگی مدار اضافه می کنیم که این یک مورد منفی محسوب می شود.