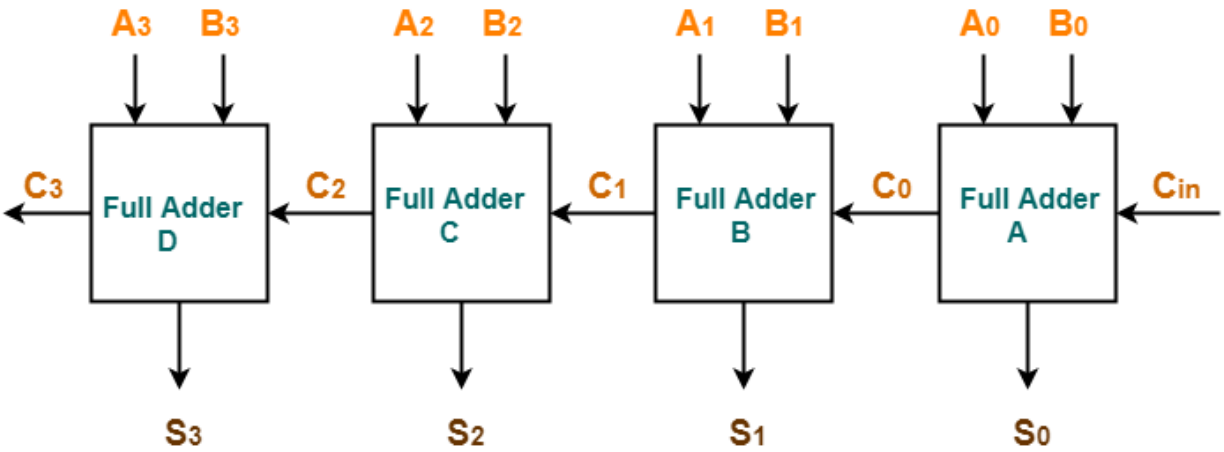
به نام خدا

آزمایش اول

امین ساوه دورودی – محمدرضا اسکینی – کامیاب عابدی

پیاده سازی یک جمع کننده چهاربیتی Ripple Carry Adder

توضیحات اولیه



4-bit Ripple Carry Adder

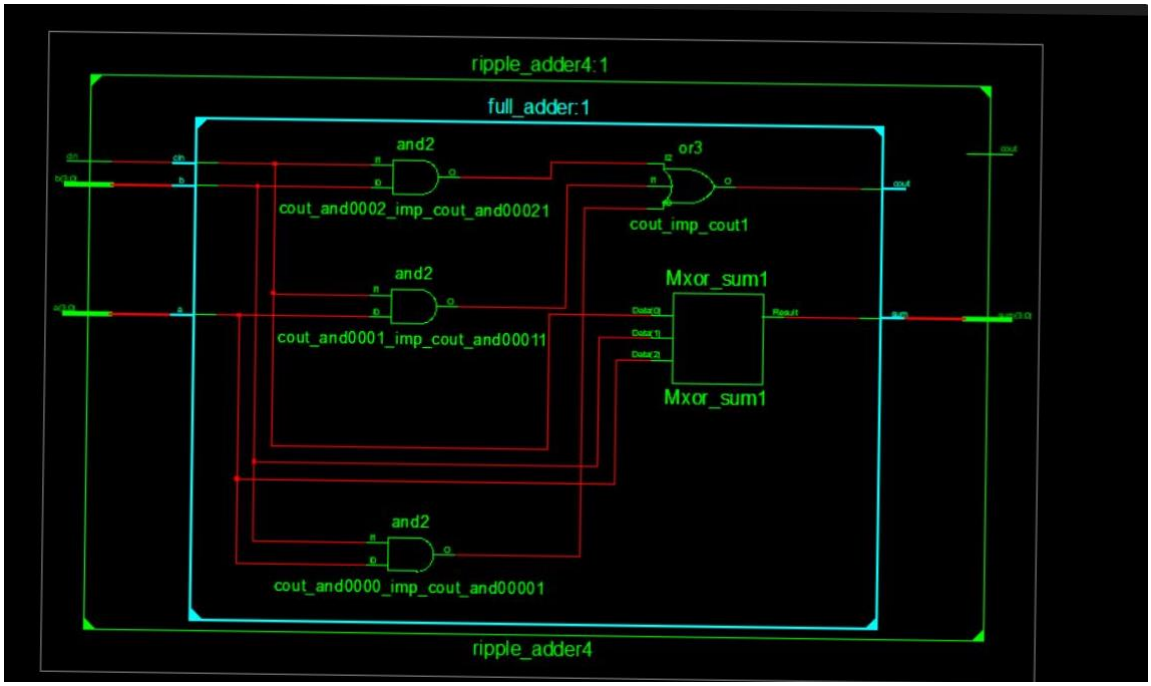
شماتیک یک جمع کننده چهاربیتی Ripple Carry Adder

در یک جمع کننده Ripple Carry Adder , همانند شکل بالا هر جمع کننده کامل به حمع کننده کامل قبلی خود وابسته است و خروجی Cout هر جمع کننده Cin جمع کننده بعدی خواهد بود. پس هر جمع کننده باید صبر کند تا محاسبات جمع کننده قبلی تمام شود و سپس کار خود را شروع کند که از این نظر , برای محاسبات بزرگ, به زمان زیادی نیاز داریم.

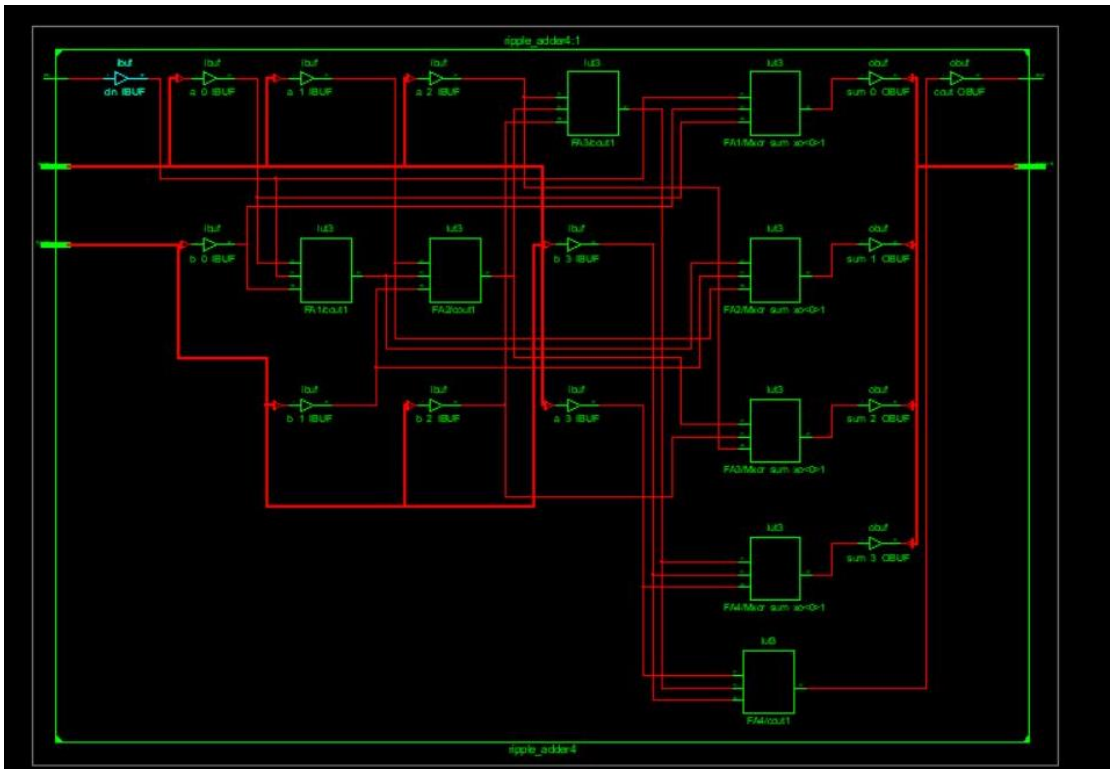
پیاده سازی

برای پیاده سازی به یک نمونه جمع کننده کامل یک بیتی (Full adder) نیاز داریم و چهار نمونه از این جمع کننده میسازیم و مانند تصویر بالا , ورودی ها را به Full adder میدهیم و در آخر انتظار داریم محاسبه به درستی انجام شود.

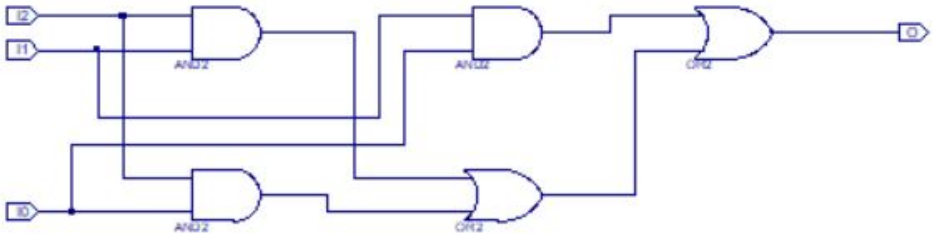
مستندات



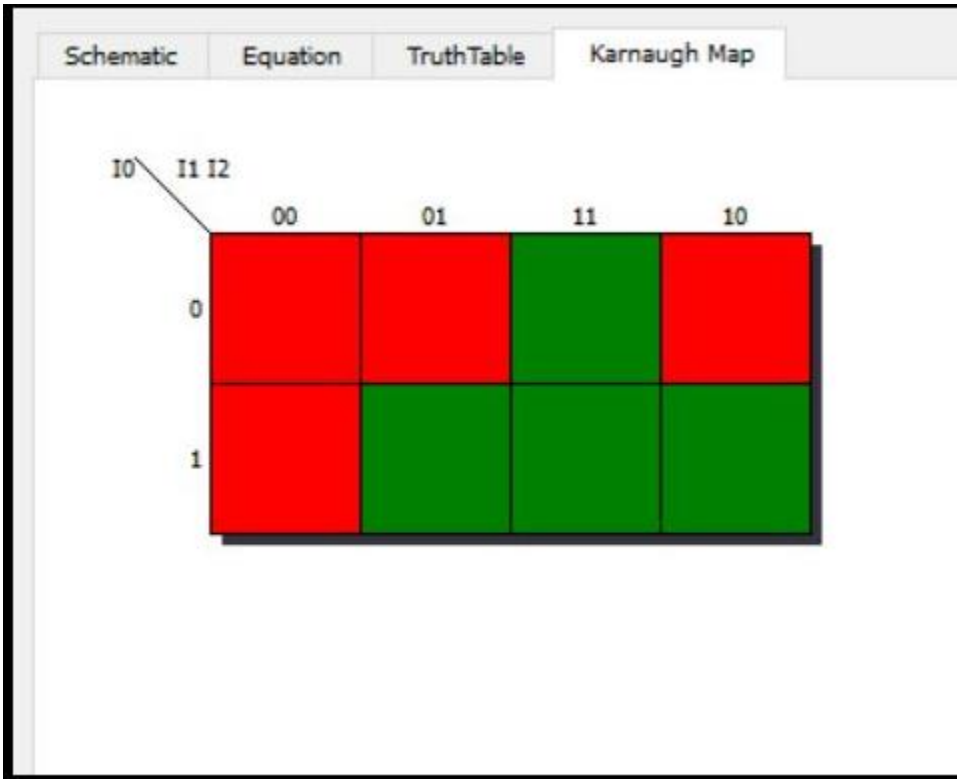
عکس ۱ (RTL)



عکس ۲ (Technology)



عکس ۳ (Schematic)



عکس ۴ (Karnaugh Map)

INIT = E8

Schematic | Equation | TruthTable | Karnaugh Map

I2	I1	I0	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

عکس ۵ (Truth Table)

Schematic | Equation | TruthTable | Karnaugh Map

O = ((I0 * I2) + (I1 * I2) + (I0 * I1));

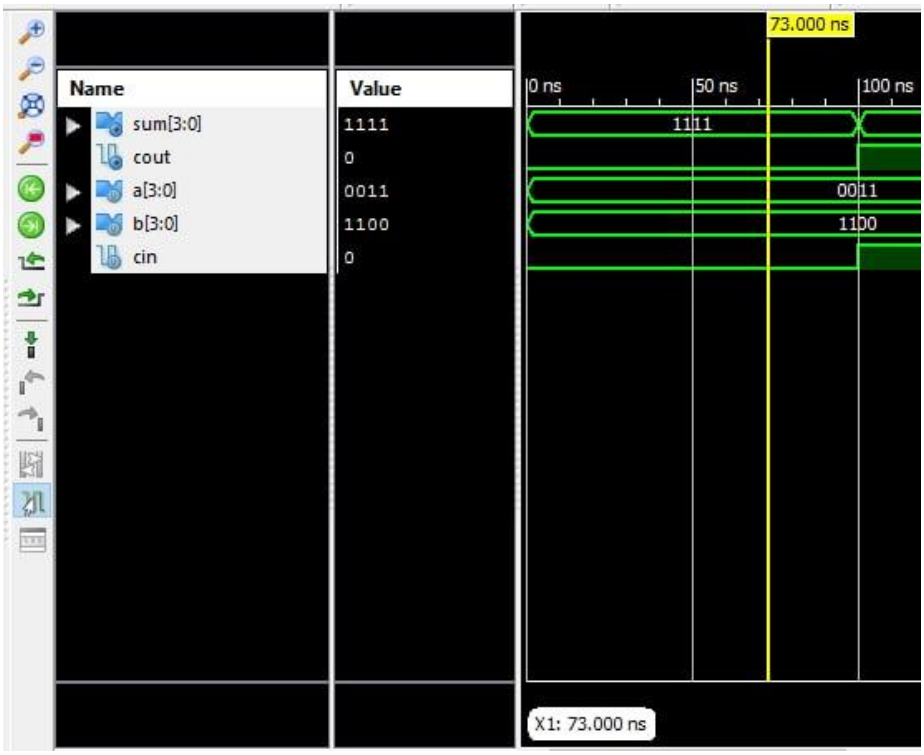
عکس ۶ (Equation)

Pad to Pad

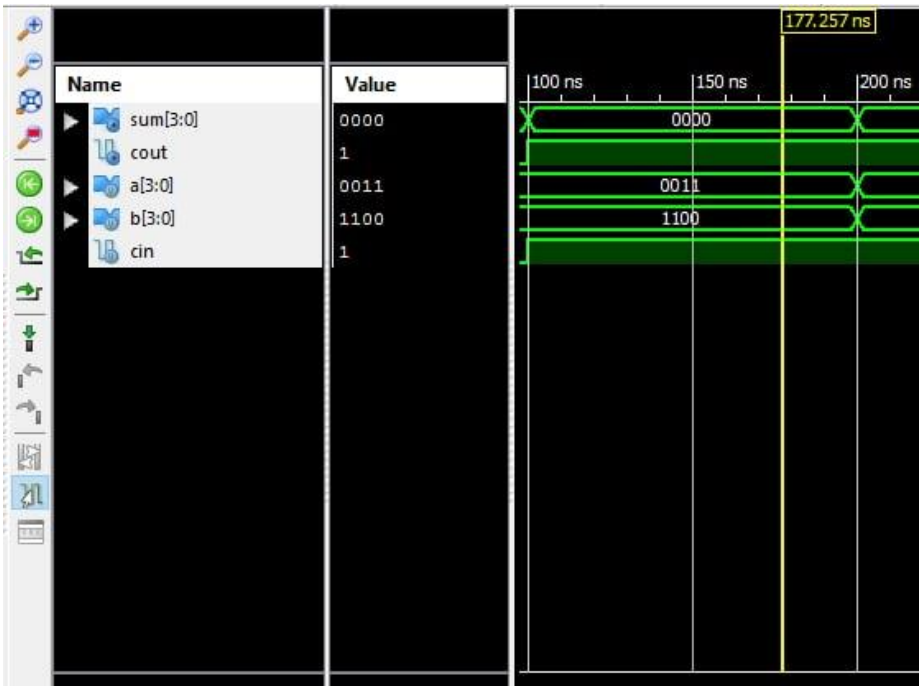
Source Pad	Destination Pad	Delay
a<0>	cout	8.566
a<0>	sum<0>	5.753
a<0>	sum<1>	6.929
a<0>	sum<2>	7.812
a<0>	sum<3>	8.680
a<1>	cout	8.263
a<1>	sum<1>	6.465
a<1>	sum<2>	7.509
a<1>	sum<3>	8.377
a<2>	cout	7.035
a<2>	sum<2>	6.100
a<2>	sum<3>	7.149
a<3>	cout	6.443
a<3>	sum<3>	5.952
b<0>	cout	8.816
b<0>	sum<0>	6.146
b<0>	sum<1>	7.179
b<0>	sum<2>	8.062
b<0>	sum<3>	8.930
b<1>	cout	8.047
b<1>	sum<1>	6.178
b<1>	sum<2>	7.293
b<1>	sum<3>	8.161
b<2>	cout	6.931
b<2>	sum<2>	6.044
b<2>	sum<3>	7.045
b<3>	cout	6.522
b<3>	sum<3>	6.249
cin	cout	8.489
cin	sum<0>	5.771
cin	sum<1>	6.852
cin	sum<2>	7.735
cin	sum<3>	8.603

عکس ۷ (جدول تاخیر)

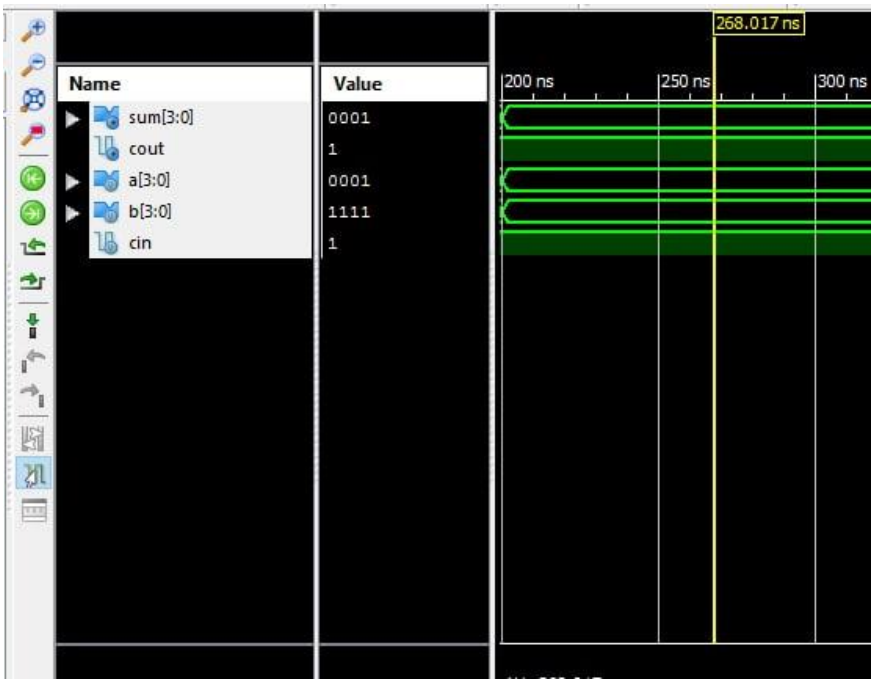
تست



عکس ۸



عکس ۹



عکس ۱۰

تست اول :

A=0011 B= 1100 Cin=0

S = 0011 + 1100 = 1111 Cout=0 انتظار ما :

همانطور که در عکس ۸ مشاهده میشود در بازه زمانی ۰ تا ۱۰۰ نانوثانیه مقدار خروجی برابر انتظارات ما میباشد.

تست دوم :

A=0011 B= 1100 Cin=1

S = 0011+1100 = 0000 Cout=1 انتظار ما :

بعد از ۱۰۰ نانوثانیه مقدار ورودی ها تغییر میکنند.

همانطور که در عکس ۹ مشاهده میشود در بازه زمانی ۱۰۰ تا ۲۰۰ نانوثانیه مقدار خروجی برابر انتظارات ما میباشد.

تست سوم :

A=0001 B= 1111 Cin=1

S = 0001 + 1111 = 0001 Cout=1 انتظار ما :

بعد از ۱۰۰ نانوثانیه مقدار ورودی ها تغییر میکنند.

همانطور که در عکس ۱۰ مشاهده میشود در بازه زمانی ۲۰۰ تا ۳۰۰ نانوثانیه مقدار خروجی برابر انتظارات ما میباشد.

خلاصه طراحی

Device Utilization Summary [-]

Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	8	1,920	1%	
Number of occupied Slices	6	960	1%	
Number of Slices containing only related logic	6	6	100%	
Number of Slices containing unrelated logic	0	6	0%	
Total Number of 4 input LUTs	8	1,920	1%	
Number of bonded IOBs	14	66	21%	
Average Fanout of Non-Clock Nets	1.71			

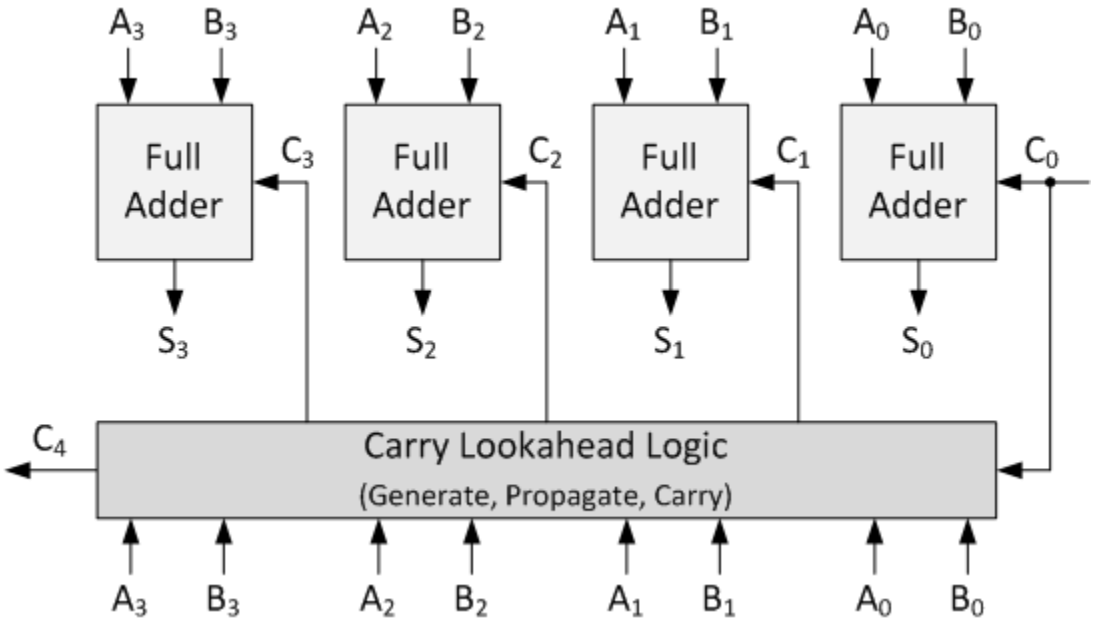
عکس ۱۱

طبق عکس بالا, خلاصه ای از کمیت های طراحی در قالب جدولی نشان داده شده است.

همانطور که مشاهده میشود , این نوع طراحی ۸ LUT را شامل میشود که در ۶ Slice در حال انجام هستند.

پیاده سازی یک جمع کننده چهاربیتی Carry Look Ahead Adder

توضیحات اولیه



شماتیک یک جمع کننده چهاربیتی Carry Look Ahead

در یک جمع کننده چهاربیتی Carry Look Ahead Adder برخلاف Ripple Carry Adder, هر جمع کننده کامل یک بیتی به جمع کننده قبلی خود وابسته نیست. مقدار cin هر جمع کننده کامل یک بیتی طبق فرمول هایی سریعتر از محاسبات جمع کننده کامل به دست آورده میشود و در اختیار جمع کننده های یک بیتی قرار میگیرد. به همین دلیل , فرآیند محاسبه سریعتر خواهد بود.

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

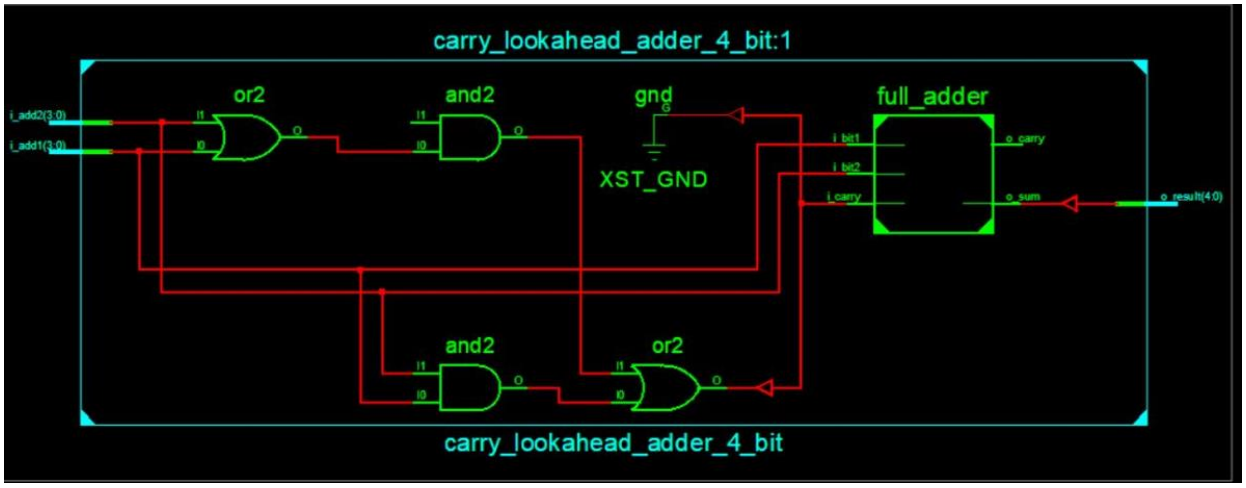
$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

تصاویر بالا نحوه محاسبه سریعتر cin و cout را نشان میدهد که بدون وابستگی به جمع کننده های کامل میتوانیم آن ها را بدست آوریم و به جمع کننده های کامل دهیم تا مجموع بیت های متناظر را محاسبه کنند.

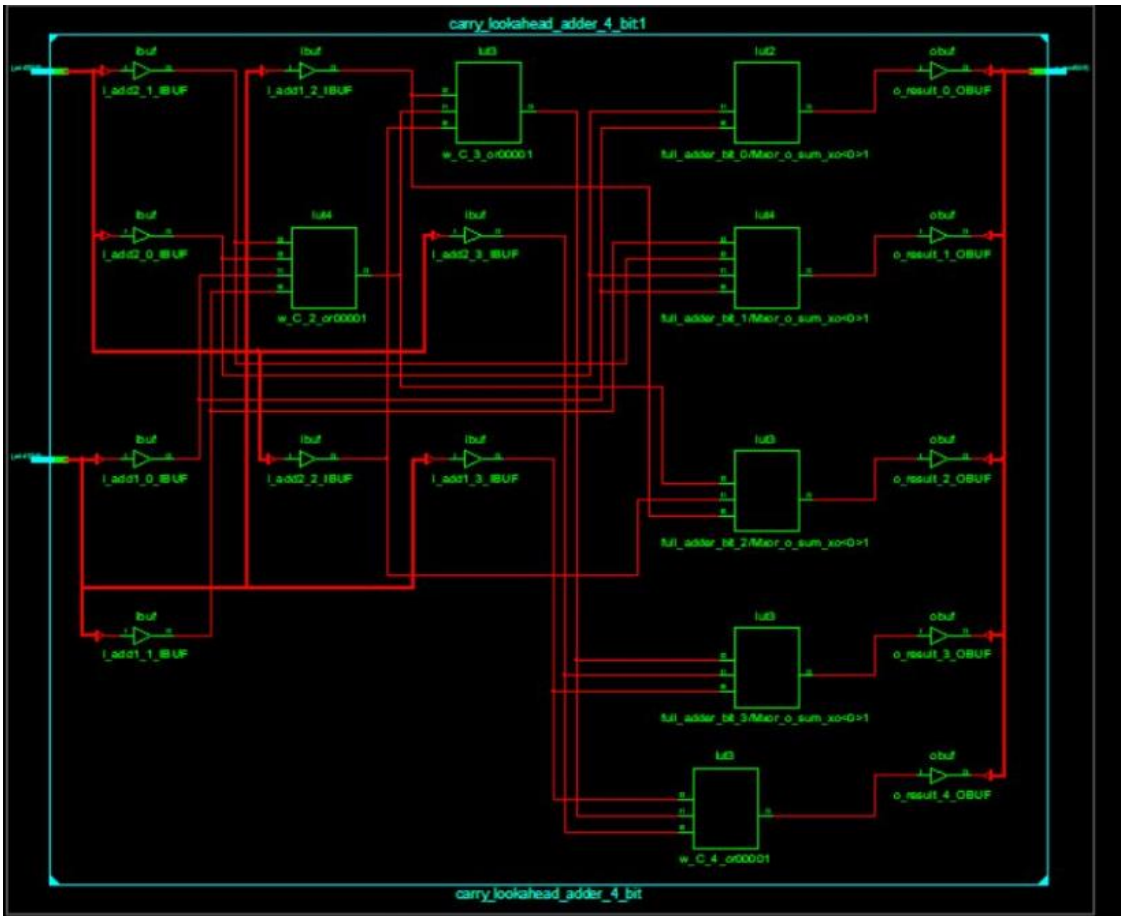
پیاده سازی

برای پیاده سازی به یک نمونه جمع کننده کامل یک بیتی (Full adder) نیاز داریم و چهار نمونه از این جمع کننده میسازیم و با استفاده از فرمول های بالا, ورودی ها را به Full adder میدهیم و در آخر انتظار داریم محاسبه به درستی انجام شود.

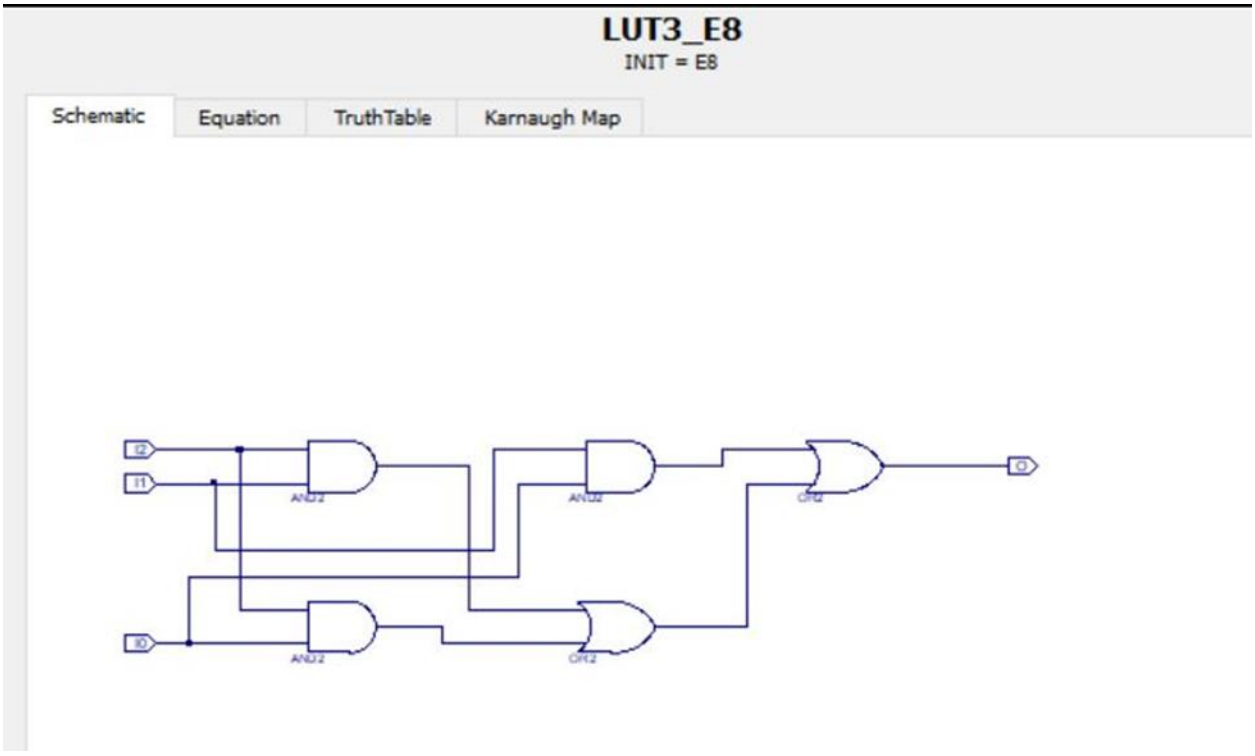
مستندات



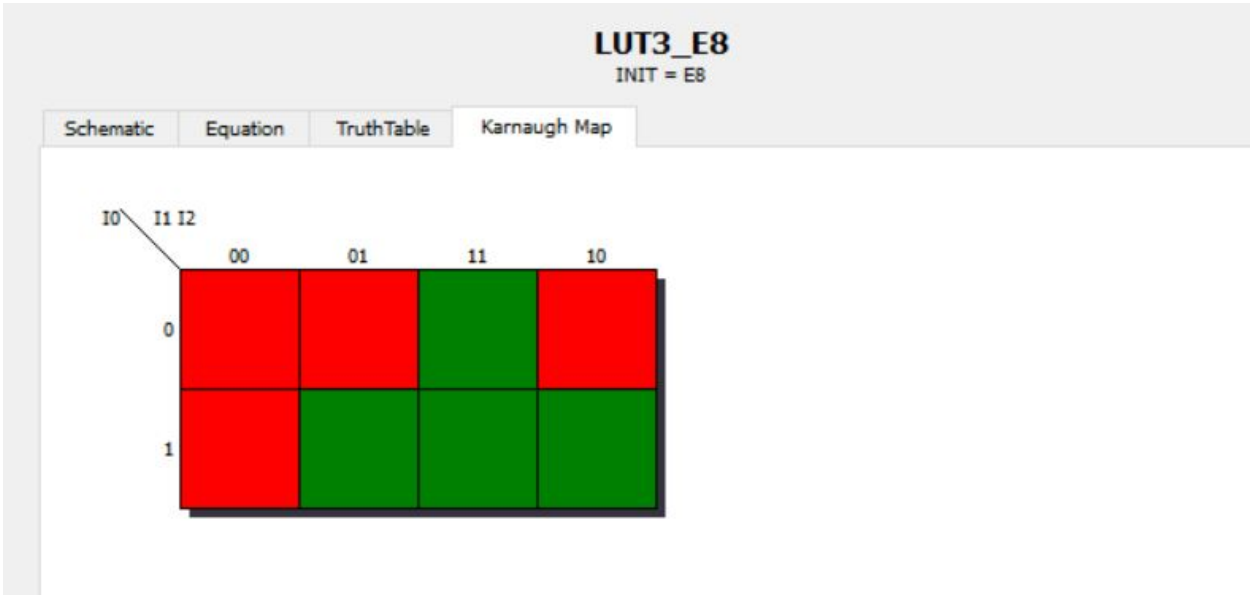
عکس ۱ (RTL)



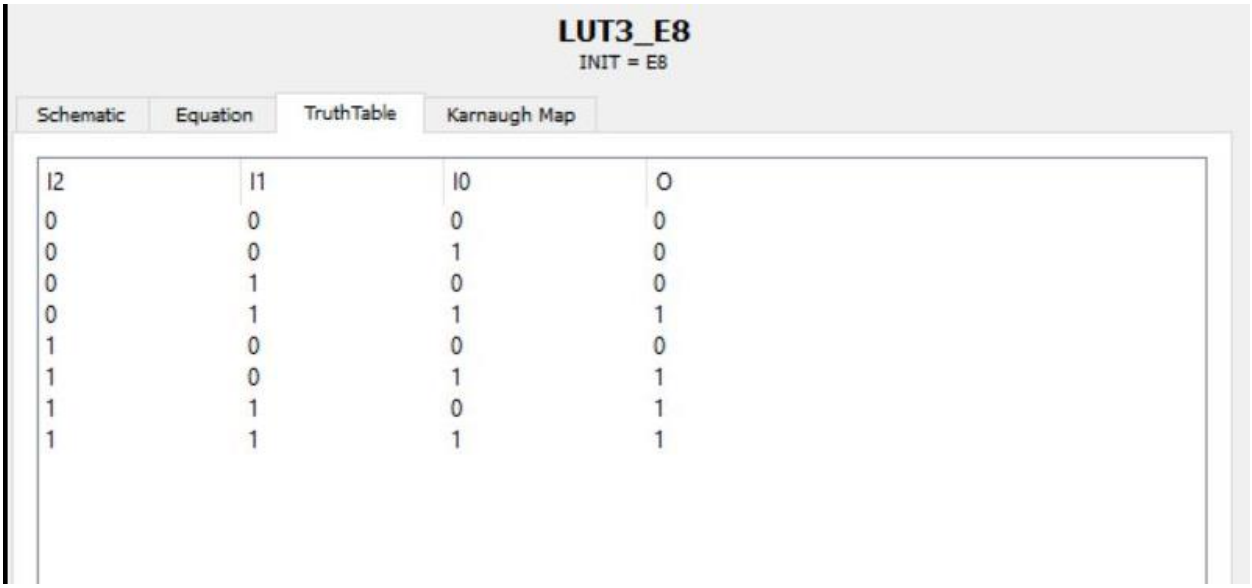
عکس ۲ (Technology)



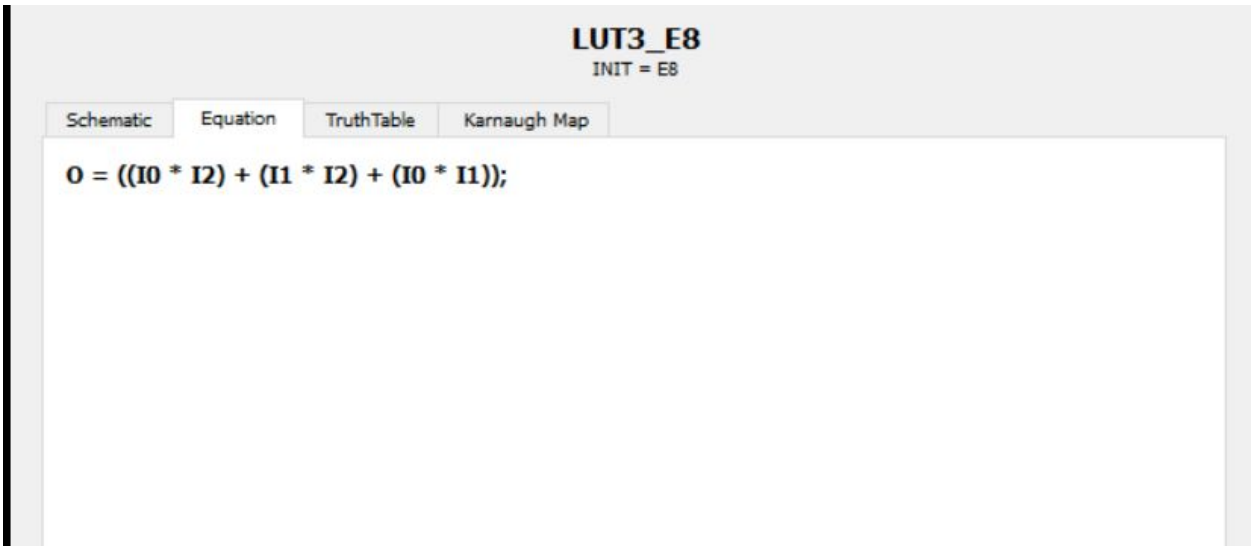
عکس ۳ (Schematic)



عکس ۴ (Karnaugh Map)



عکس ۵ (Truth Table)



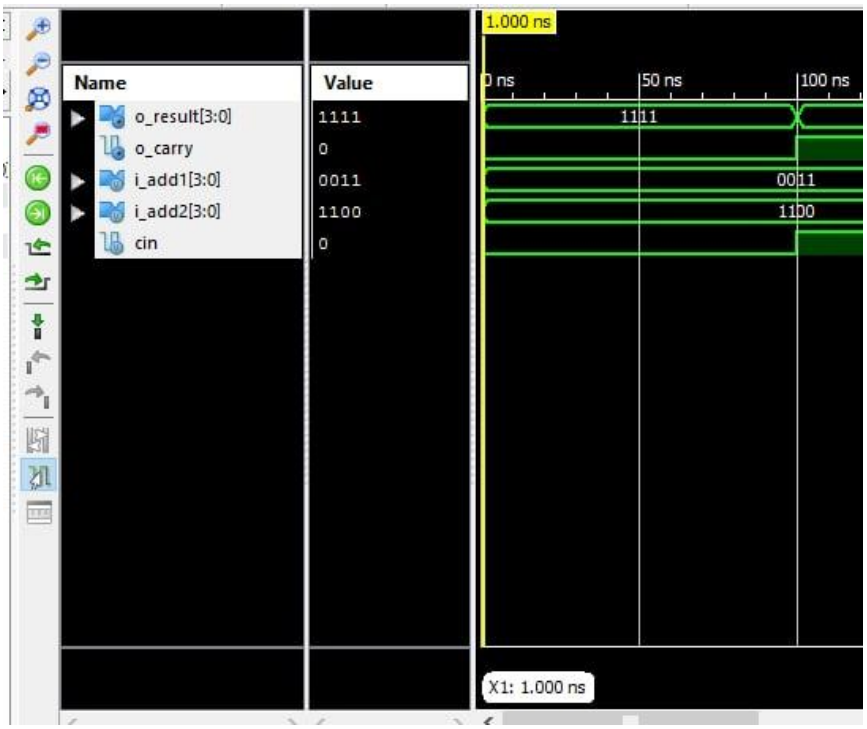
عکس ۶ (Equation)

Pad to Pad

Source Pad	Destination Pad	Delay
cin	o_carry<0>	9.452
cin	o_result<0>	6.482
cin	o_result<1>	7.692
cin	o_result<2>	8.433
cin	o_result<3>	9.395
i_add1<0>	o_carry<0>	8.913
i_add1<0>	o_result<0>	6.206
i_add1<0>	o_result<1>	7.153
i_add1<0>	o_result<2>	7.894
i_add1<0>	o_result<3>	8.856
i_add1<1>	o_carry<0>	8.039
i_add1<1>	o_result<1>	5.965
i_add1<1>	o_result<2>	7.020
i_add1<1>	o_result<3>	7.982
i_add1<2>	o_carry<0>	7.252
i_add1<2>	o_result<2>	6.281
i_add1<2>	o_result<3>	7.195
i_add1<3>	o_carry<0>	6.176
i_add1<3>	o_result<3>	6.092
i_add2<0>	o_carry<0>	8.750
i_add2<0>	o_result<0>	6.067
i_add2<0>	o_result<1>	6.990
i_add2<0>	o_result<2>	7.731
i_add2<0>	o_result<3>	8.693
i_add2<1>	o_carry<0>	7.996
i_add2<1>	o_result<1>	5.997
i_add2<1>	o_result<2>	6.977
i_add2<1>	o_result<3>	7.939
i_add2<2>	o_carry<0>	6.690
i_add2<2>	o_result<2>	5.814
i_add2<2>	o_result<3>	6.633
i_add2<3>	o_carry<0>	6.085
i_add2<3>	o_result<3>	5.978

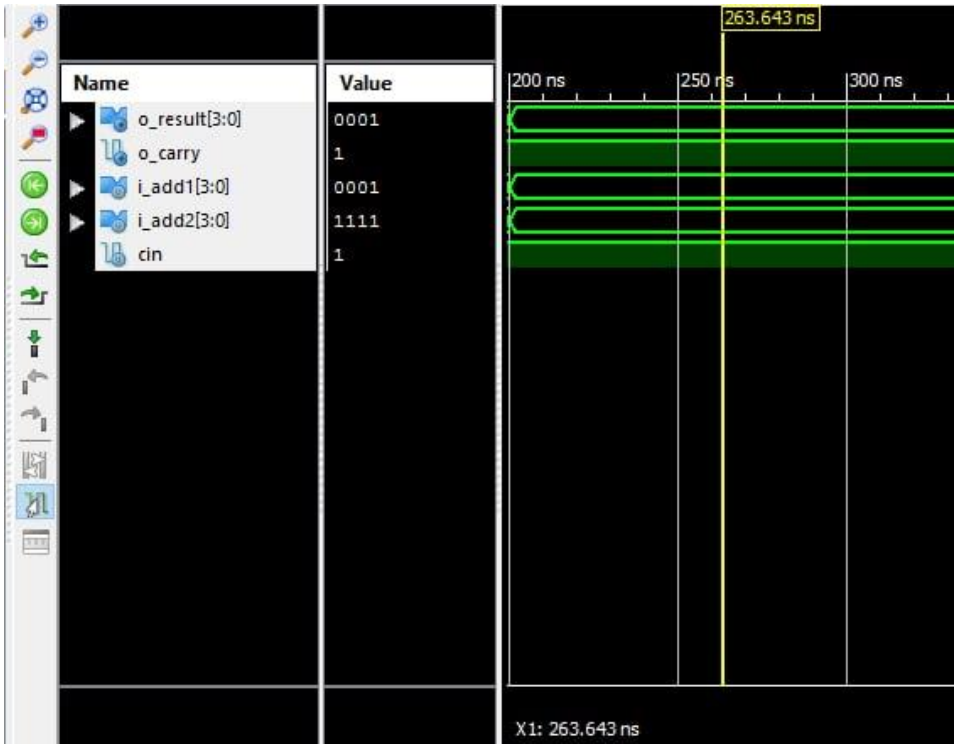
عکس ۷ (جدول تاخیر)

تست



عکس ۸

عکس ۹



عکس ۱۰

تست اول :

A=0011 B= 1100 Cin=0

S = 0011 + 1100 = 1111 Cout=0 انتظار ما :

همانطور که در عکس ۸ مشاهده میشود در بازه زمانی ۰ تا ۱۰۰ نانوثانیه مقدار خروجی برابر انتظارات ما میباشد.

تست دوم :

A=0011 B= 1100 Cin=1

S = 0011+1100 = 0000 Cout=1 انتظار ما :

بعد از ۱۰۰ نانوثانیه مقدار ورودی ها تغییر میکنند.

همانطور که در عکس ۹ مشاهده میشود در بازه زمانی ۱۰۰ تا ۲۰۰ نانوثانیه مقدار خروجی برابر انتظارات ما میباشد.

تست سوم :

A=0001 B= 1111 Cin=1

S = 0001 + 1111 = 0001 Cout=1 انتظار ما :

بعد از ۱۰۰ نانوثانیه مقدار ورودی ها تغییر میکنند.

همانطور که در عکس ۱۰ مشاهده میشود در بازه زمانی ۲۰۰ تا ۳۰۰ نانوثانیه مقدار خروجی برابر انتظارات ما میباشد.

خلاصه طراحی :

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	7	1,920	1%	
Number of occupied Slices	4	960	1%	
Number of Slices containing only related logic	4	4	100%	
Number of Slices containing unrelated logic	0	4	0%	
Total Number of 4 input LUTs	7	1,920	1%	
Number of bonded IOBs	16	66	24%	
Average Fanout of Non-Clock Nets	1.80			

عکس ۱۱

طبق عکس بالا, خلاصه ای از کمیت های طراحی در قالب جدولی نشان داده شده است.

همانطور که مشاهده میشود , این نوع طراحی ۷ LUT را شامل میشود که در ۴ Slice در حال انجام هستند.

نتیجه گیری

طبق مقایسه جدول های خلاصه طراحی هر دو جمع کننده یعنی تصویر های ۱۱ هر دو بخش , طبق انتظارات ما همانطور که در توضیحات اولیه آورده شده است, جمع کننده Carry Look Ahead Adder تعداد کمتری LUT در طراحی خود دارد و همینطور slice های کمتری نسبت به Ripple Carry Adder اشغال میکند. و هم چنین طبق توضیحات اولیه Carry Look Ahead Adder در محاسبات سنگین به دلیل انجام محاسبات به صورت موازی در جمع کننده های کامل آن , سریعتر میباشد که با مقایسه تصویر های ۷ هر دو بخش به این نتیجه به صورت عملی خواهیم رسید.