آزمایش سوم: جمع و تفریق اعداد ممیز شناور

چکیده

در این آزمایش، با استفاده از تراشه‌های TTL موجود در نرم‌افزار پروتئوس به طراحی یک واحد جمع و تفریق برای اعداد ممیز شناور می‌پردازیم. واحد طراحی شده مطابق استاندارد IEEE-754 single precision است و دارای ۱ بیت برای علامت، ۸ بیت برای نما و ۲۳ بیت برای مانتیس می‌باشد.

فهرست

لیست تراشه‌های استفاده شده در طرح

شمارنده دودویی چهاربیتی: 74191

مقایسه‌کننده cascadable چهاربیتی: 7485

جمع‌کننده‌ی چهاربیتی: 7483

شیفت رجیستر دوجهته هشت بیتی: 74198

همچنین از موارد زیر نیز در طرح استفاده شده است:

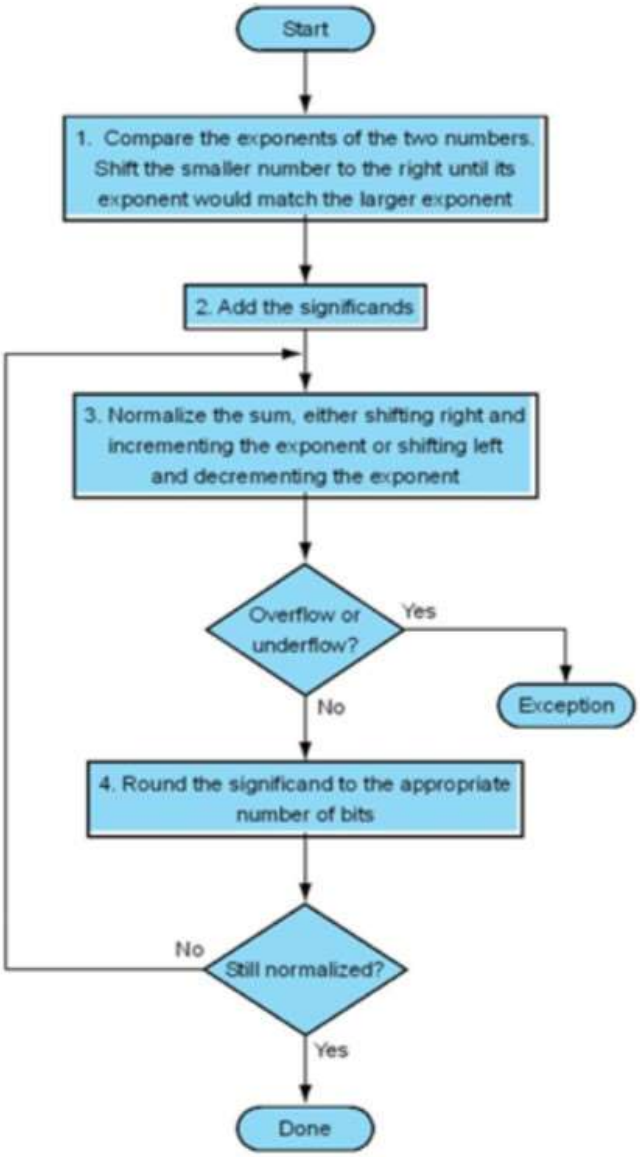
فلیپ‌فلاپ نوع D یک بیتی، گیت AND2 و AND3 و XOR2 و OR2 و OR3 و OR4 و OR8 و NOT و NAND2 .

برای اعمال ورودی‌ها از LOGICSTATE و برای مشاهده‌ی خروجی‌ها از LOGICPROBE استفاده شده است.

برای تولید پالس clock در برخی مراحل آزمایش از LOGICSTATE استفاده شد تا امکان دیباگ کردن بخش‌های مختلف وجود داشته باشد و در برخی مراحل از المان CLOCK موجود در پروتئوس استفاده شد.

الگوریتم مورد استفاده

الگوریتم مورد استفاده برای جمع و تفریق دارای چنین فلوچارتی است:



البته در حالتی که تفریق داشته باشیم در مرحله‌ی .3 بجای جمع، عملیات تفریق و قدرمطلق‌گیری صورت می‌گیرد.

حال قسمت‌های مختلف این الگوریتم را به شکل سخت افزاری پیاده می‌کنیم – برای این منظور، چند بخش مجزا در مدارمان قرار گرفته‌اند که در این گزارش در هر فصل به یکی از این قسمت‌ها و نحوه عملکرد آنها می‌پردازیم.

در طراحی انجام شده، ۱ بیت برای علامت، ۸ بیت برای نما و ۲۳ بیت برای مانتیسا در نظر گرفته شده است.

ورودی‌ها و خروجی‌های مدار

ورودی‌ها

عملوند اول، A: اولین عدد ماست که دارای ۳۲ بیت بوده و عملیات روی آن انجام می‌شود.

عملوند دوم، B: دومین عدد ماست که دارای ۳۲ بیت بوده و عملیات روی آن انجام می‌شود.

start: اگر این بیت ۱ باشد عملیات انجام می‌شود.

add/sub: اگر این بیت ۰ باشد حاصل برابر A+B است و در غیر این صورت A-B است.

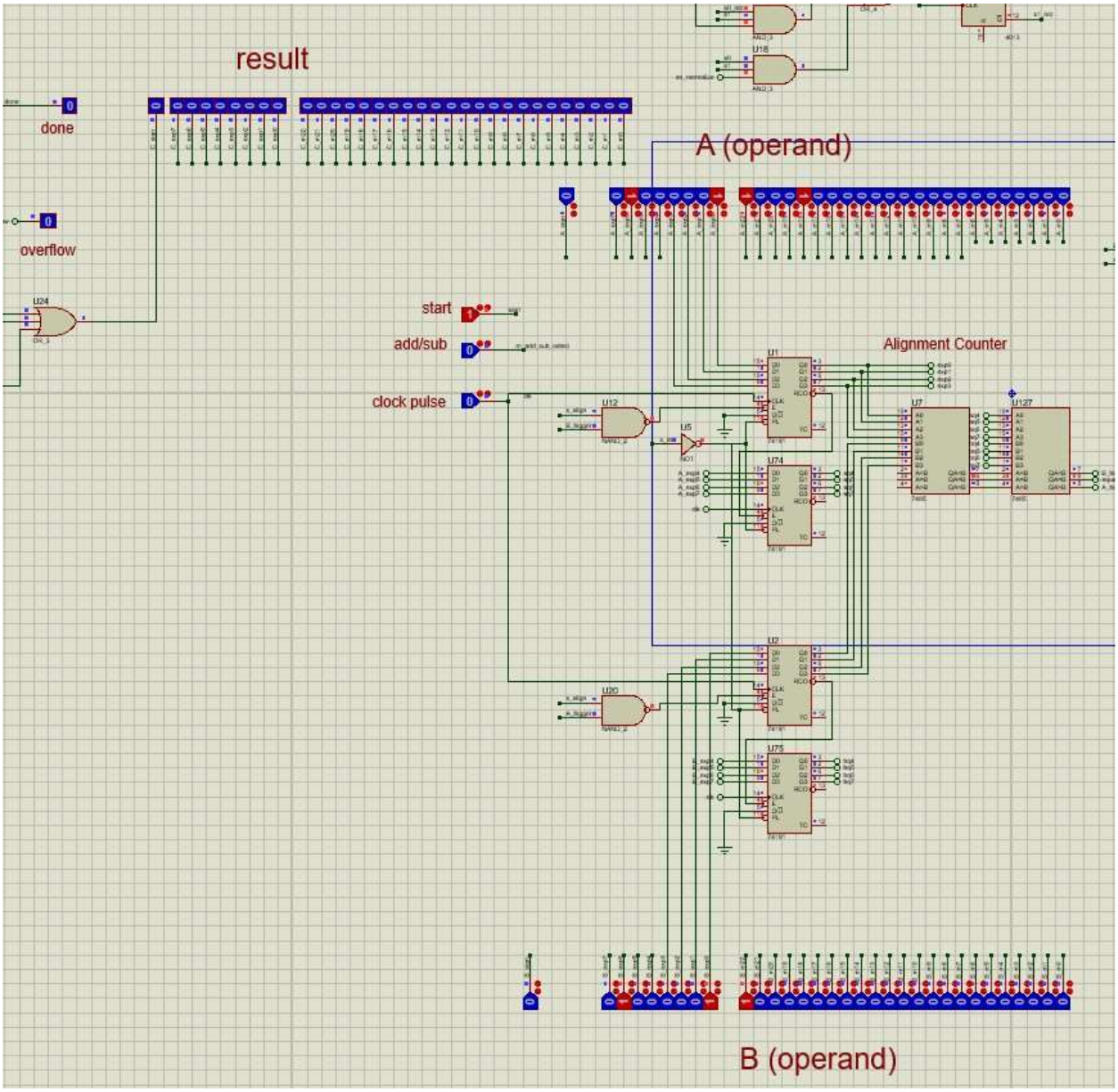
clock pulse: این بیت ورودی کلاک سیستم است، که برای اینکه امکان تحلیل پالس به پالس مدار وجود داشته باشد بجای منبع clock این ورودی قرار داده شده است.

خروجی‌ها

result: این خروجی ۳۲ بیتی متناظر با حاصل عملیات انجام شده است.

done: چنانچه این خروجی ۱ باشد یعنی مقدار قرار گرفته روی result معتبر است.

overflow: این خروجی نشان دهنده سرریز در نمای حاصل است، یعنی چنانچه این خروجی ۱ باشد عدد ۳۲ بیتی حاصل درواقع معتبر نیست زیرا در محاسبه نمای آن سرریز رخ داده است.



بخش محاسبه state مدار

از آنجایی که عملیات جمع و تفریق دارای چندین state است، نیازمند یک state register هستیم که مشخص کند در حال حاضر در کدام state هستیم و با توجه به شرایط فعلی، state مربوط به کلاک بعدی را نیز تعیین کند. برای این منظور یک مدار کوچک طراحی شده است که عملیات مد نظر را انجام دهد.

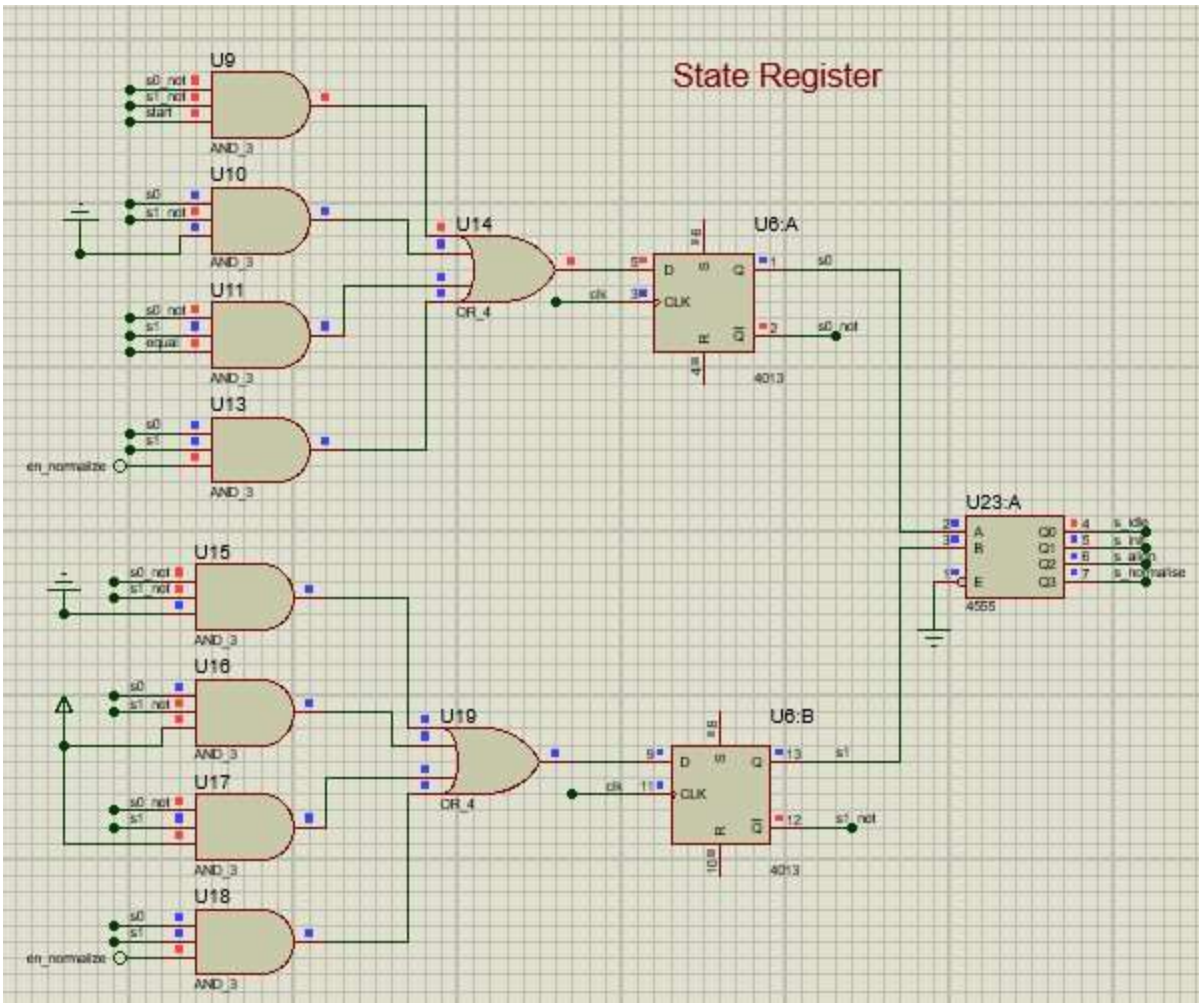
مدار دارای ۴ استیت است:

حالت IDLE: در این حالت عملیاتی در حال انجام شدن نیست و مدار منتظر ورودی و سیگنال استارت است تا محاسبات شروع شود. چنانچه سیگنال start نداشته باشیم در این حالت می‌مانیم. همچنین بعد از پایان عملیات مستقیما به این حالت می‌آییم.

حالت INITIALIZE: در این حالت مدار ورودی‌ها را بارگذاری می‌کند. این حالت همواره یک کلاک زمان خواهد برد.

حالت ALIGN: در این حالت چنانچه نمای دو ورودی متفاوت باشد، مانتیس ورودی با توان کمتر به راست شیفت داده می‌شود و توان آن زیادتر می‌شود تا نماها یکسان شوند و عملیات به سادگی صورت گیرد. تعداد کلاک مورد نیاز برای این حالت برابر اندازه‌ی تفاضل دو نمای ورودی است.

حالت NORMALIZE: از آنجایی که پس از محاسبه جمع یا تفریق میان دو مانتیسا ممکن است حاصل بدست آمده نرمال نباشد، لازم است این عملیات روی آن انجام شود تا حاصل ذخیره شده نرمال باشد. تعداد کلاک این حالت متغیر است و بستگی به مانتیس‌ها و نماهای دو ورودی و عملیات انجام شده میان دو مانتیس دارد.



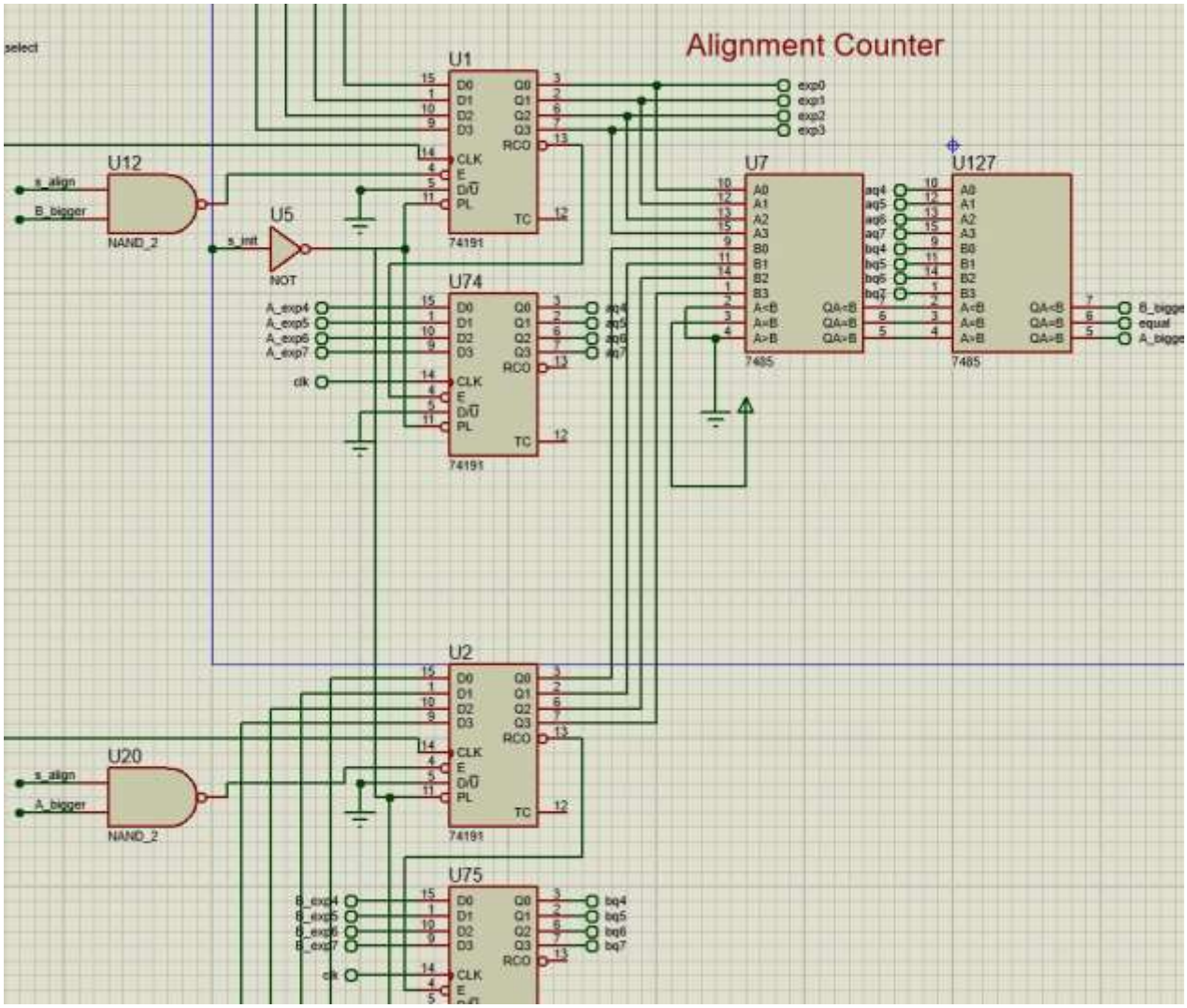
در AND3 های چهارم و هشتم از بالا قابل مشاهده است که ورودی en_normalize داده شده است – همانطور که بعدا خواهیم دید، این امر بخاطر متغیر بودن تعداد کلاک لازم برای عملیات normalization است.

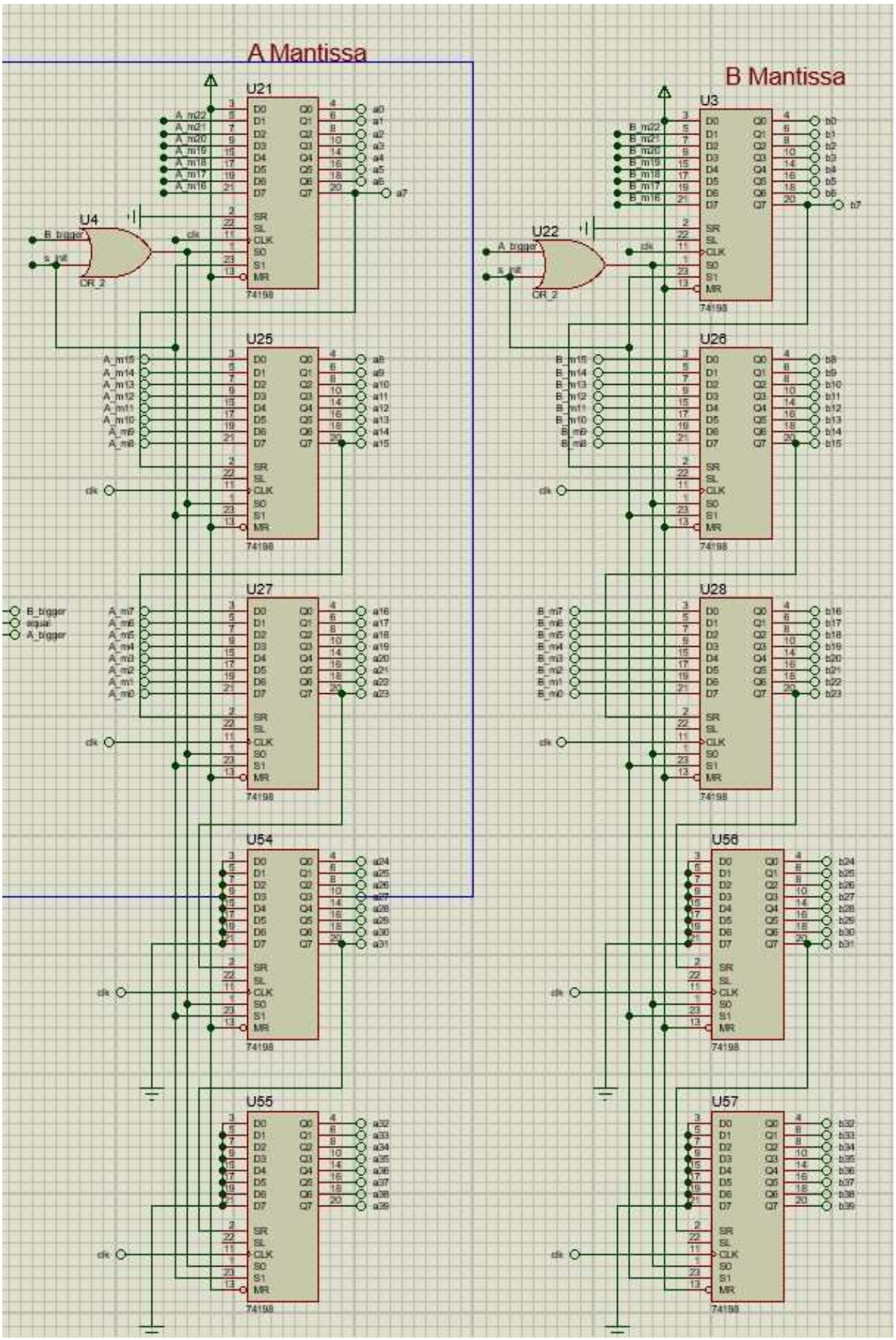
بخش alignment مانتیس‌های دو ورودی

همانطور که اشاره شد، باید قبل از انجام عملیات جمع، دو مانتیس ورودی با همدیگر align شوند: یعنی لازم است که نمای متناظر با دو مانتیس برابر باشد تا بتوانیم عملیات جمع را انجام دهیم. به عنوان مثال، اگر توان یک ورودی ۱ باشد و مانتیس با 1 leading آن 1.1 باشد و دیگری توان ۲ داشته باشد و مانتیس با 1 leading آن برابر 1.01 باشد، نمی‌توان 1.1 و 1.01 را مستقیما جمع زد بلکه باید یکی از این دو مقدار را شیفت دهیم تا توان‌ها برابر شوند و سپس روی مانتیس‌ها عملیات جمع یا تفریق انجام دهیم.

برای این منظور با استفاده از شمارنده و مقایسه‌کننده، مانتیس‌ها شیفت داده می‌شوند تا توان‌ها برابر شوند. سپس حاصل شیفت یافته‌ی مانتیس‌ها (به همراه 1 leading) در ثبات‌هایی ذخیره می‌شوند تا بعدا از آنها استفاده شود.

فلسفه عملکرد به این شکل است که در گام initialization مقدار نماها بصورت parallel load وارد شمارنده‌ها می‌شود و مانتیس‌ها در شیفت رجیسترها بارگذاریم می‌شوند. سپس بر اساس اینکه کدامیک بزرگتر است (طبق خروجی‌های مقایسه‌کننده) شمارنده ها یکی یکی مقدار یکی از نماها را زیاد می‌کنند تا نماها برابر شوند.


Alignment Counter
U1
U12
NAND_2
s_align
B_bigger
U5
NOT
s_int
U74
74191
A_exp4
A_exp5
A_exp6
A_exp7
clk
U7
7485
U127
exp0
exp1
exp2
exp3
aq4
aq5
aq6
aq7
bq4
bq5
bq6
bq7
B_bigger
equal
A_bigger
U2
U20
A_bigger
U75
B_exp4
B_exp5
B_exp6
B_exp7


A Mantissa
B Mantissa
U21
U25
U27
U54
U55
U3
U26
U28
U56
U57
U4
U22
74198

همانطور که توضیح داده شد و در دو تصویر بالا مشاهده می‌شود، نمای دو عدد به ورودی parallel-load شمارنده ها وصل شده‌اند. از آنجایی که شمارنده 74191 چهاربیتی است و توان‌های ما هشت بیتی هستند، برای انجام عملیات alignment دو عدد از این شمارنده‌ها را بصورت ripple–شده به هم وصل می‌کنیم، صرفا کافیست خروجی RCO (Ripple Carry Out) از شمارنده بیت‌های کم‌ارزش‌تر به پایه enable شمارنده بیت‌های پرارزش‌تر وصل شود.

ورودی Parallel Load این شمارنده ها به نحوی است که در استیت init بارگذاری رخ دهد و ورودی enable به نحوی قرار داده شده است که بر اساس تفاوت داشتن یا نداشتن نماها و استیت alignment عملیات شمارش رخ دهد.

برای مقایسه هم مجددا چون مقایسه‌گر 7485 چهاربیتی است و نماهای ما هشت بیتی هستند، دو عدد از این تراشه‌ها cascade شده‌اند.

برای ذخیره سازی مانتیس و شیفت دادن آن، از رجیسترهای 74198 استفاده شده است که هرکدام هشت بیتی هستند و امکان بارگذاری موازی و شیفت به هر دو جهت را دارند. در مجموع پنج عدد از این رجیسترها قرار داده شده است که امکان ذخیره کردن ۴۰ بیت را برای ما مهیا می‌کند. دقت داریم که اگرچه مانتیس ما ۲۳ بیت است (که با بیت 1 leading می‌شود ۲۴ بیت) ولی بخاطر اختلاف توان‌ها اعدادمان شیفت خواهند خورد. برای از دست ندادن دقت در عملیات، مانتیس پس از alignment با ۴۰ بیت در نظر گرفته شده است (بجای ۲۴ بیت)

ورودی‌های s1 و s0 این رجیسترها به نحوی متصل شده است که در استیت init بارگذاری شوند و در استیت alignment بر اساس وضعیت نماها شیفت به راست روی عدد با نمای کوچکتر انجام شود (و نمای آن زیاد شود)

همچنین ورودی SR رجیسترها که برای تعیین عدد وارد شده به عنوان MSB هنگام شیفت به راست است به رجیستر بعدی وصل شده است. دقت داریم که چون عملیات ما هیچگاه شیفت به چپ نیست، نیازی نیست برای ورودی SL چیزی قرار دهیم.

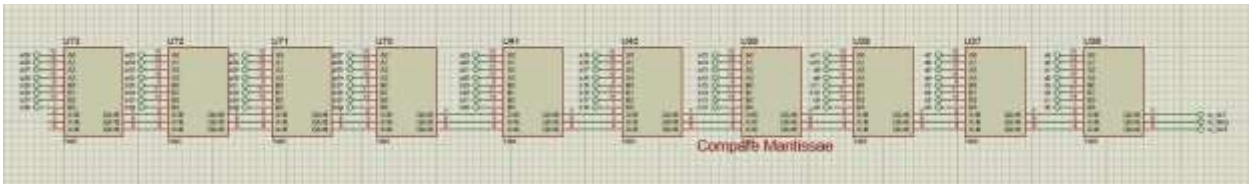
Function Table

Inputs										Outputs			
Clear	Mode		Clock	Serial		Parallel				Q_A	Q_B	Q_C	Q_D
	S1	S0		Left	Right	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	L	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	Q_{An}	Q_{Bn}	Q_{Cn}
H	L	H	↑	X	L	X	X	X	X	L	Q_{An}	Q_{Bn}	Q_{Cn}
H	H	L	↑	H	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	H
H	H	L	↑	L	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	L
H	L	L	X	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

H = HIGH Level (steady state)
L = LOW Level (steady state)
X = Don't Care (any input, including transitions)
↑ = Transition from LOW-to-HIGH level
a, b, c, d = The level of steady state input at inputs A, B, C or D, respectively.
Q_{A0}, Q_{B0}, Q_{C0}, Q_{D0} = The level of Q_A, Q_B, Q_C, or Q_D, respectively, before the indicated steady state input conditions were established.
Q_{An}, Q_{Bn}, Q_{Cn}, Q_{Dn} = The level of Q_A, Q_B, Q_C, respectively, before the most-recent ↑ transition of the clock.

بخش مقایسه کردن مانتیس‌های دو عدد

همانطور که جلوتر خواهیم دید، برای تعیین علامت حاصل عملیات و برای تعیین برخی جزئیات دیگر، لازم است که بدانیم کدام mantissa (پس از alignment) مقدار بیشتری دارد. برای این منظور صرفا از تعدادی مقایسه‌کننده 7485 بصورت cascade شده استفاده می‌کنیم.



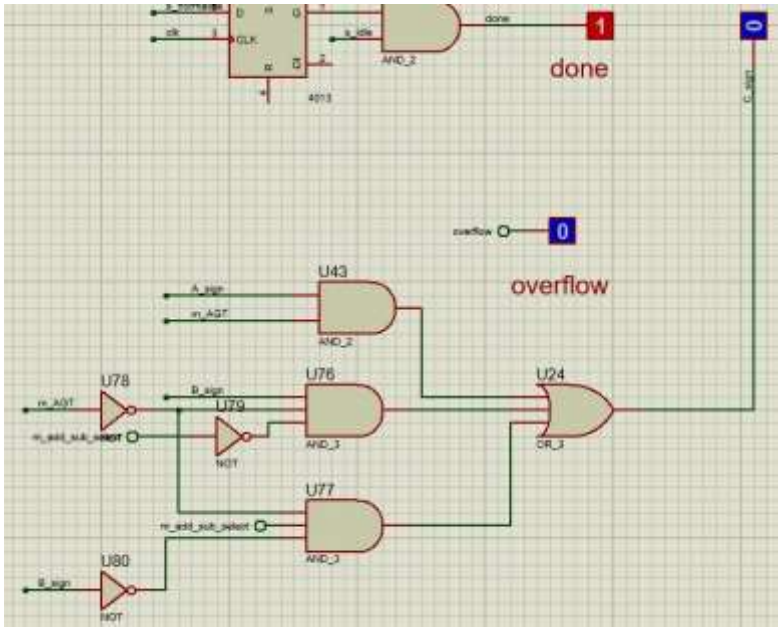
همانطور که گفته شد، مانتیس پس از alignment با ۴۰ بیت در نظر گرفته می‌شود، و مقایسه‌کننده ما چهاربیتی است، پس برای انجام مقایسه نیازمند cascade کردن ده عدد مقایسه کننده هستیم. طبق دیتاشیت چنانچه ورودی‌های A<B، A>B، A=B برای مقایسه‌گر اول high-Z باشند مشکلی ایجاد نمی‌شود.

خروجی‌های m_AGT، m_AEB و m_ALT نشان‌دهنده بزرگتر بودن مانتیس A، تساوی و بزرگتر بودن مانتیس B هستند.

بخش تعیین علامت حاصل

همانطور که گفته شد، خروجی حاصل ما ۳۲ بیت است که یک بیت آن علامت است. عملیات تعیین علامت بعد از alignment و مقایسه شدن مانتیس دو عدد، بصورت کاملا ترکیبی انجام می‌شود و جدای از محاسبات اصلی (که در ادامه خواهیم دید) است.

برای تعیین علامت حاصل کافیست به عملیات مورد نظر (جمع یا تفریق)، علامت دو ورودی و بزرگتر/کوچکتر بودن دو مانتیس توجه کرد. مدار این عملیات در شکل زیر قابل مشاهده است:

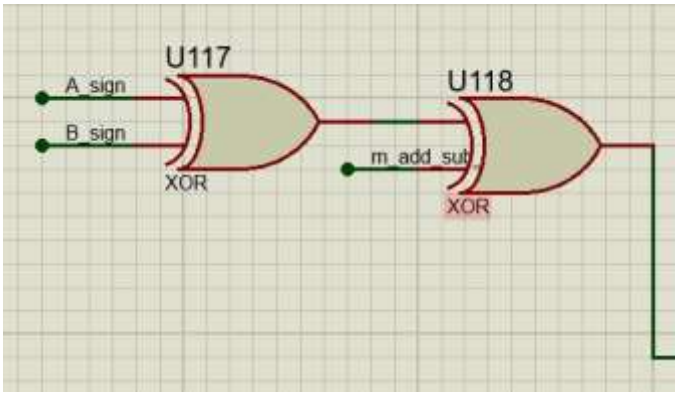


بخش جمع/تفریق مانتیس‌ها

در این بخش، یک مقدار اولیه به عنوان مانتیس حاصل بدست می‌آید. همچنین دقت داریم که مقدار اولیه برای نمای حاصل برابر با بزرگترین نما میان دو ورودی است.

بعدتر در مرحله‌ی normalization ممکن است نیازمند کاهش یا افزایش توان همراه با شیفت دادن مانتیس باشیم، برای همین حاصل این قسمت صرفا اولیه است و لزوما جواب نهایی ما نیست.

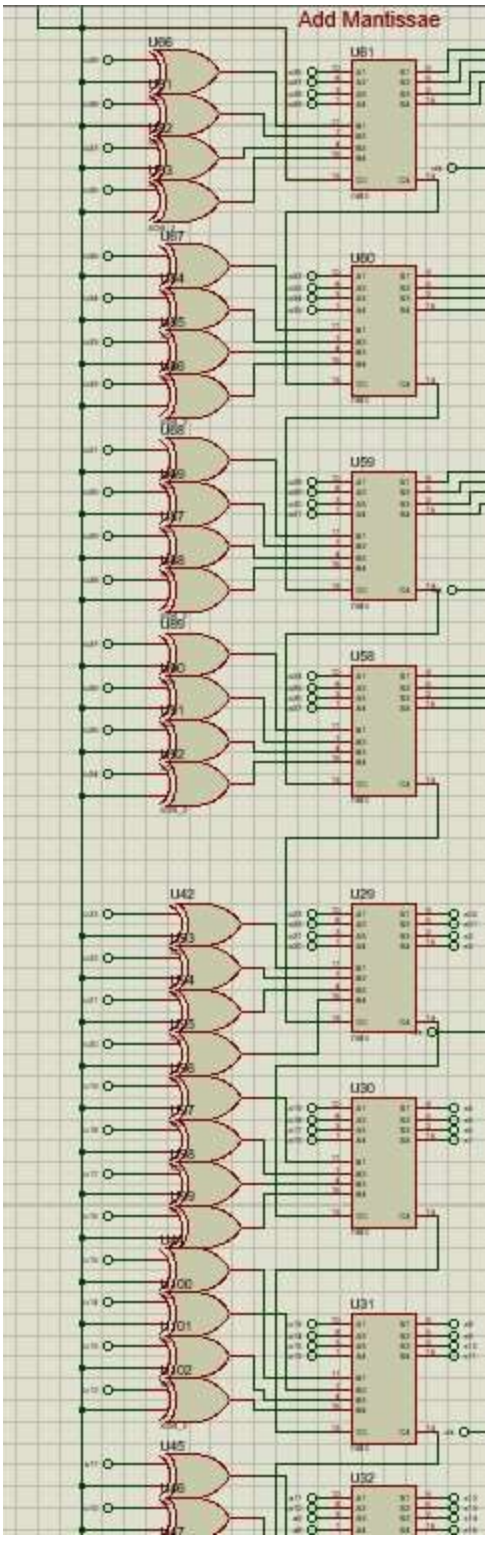
اولا لازم است که مشخص شود باید دو مانتیس با هم جمع زده شوند یا از هم کم شوند. برای این منظور هم باید به عملیات مد نظر دقت داشته باشیم و هم به علامت میان دو مانتیس. بصورت ترکیبی داریم:



چنانچه حاصل اینجا برابر ۱ باشد باید میان دو مانتیس تفریق و در غیر این صورت باید میان دو مانتیس جمع انجام شود. برای جمع هم از ۱۰ عدد تراشه 7483 بصورت ripple-carry استفاده می‌شود (هر جمع‌کننده چهاربیتی است و مانتیس ۴۰ بیت فرض شده است)

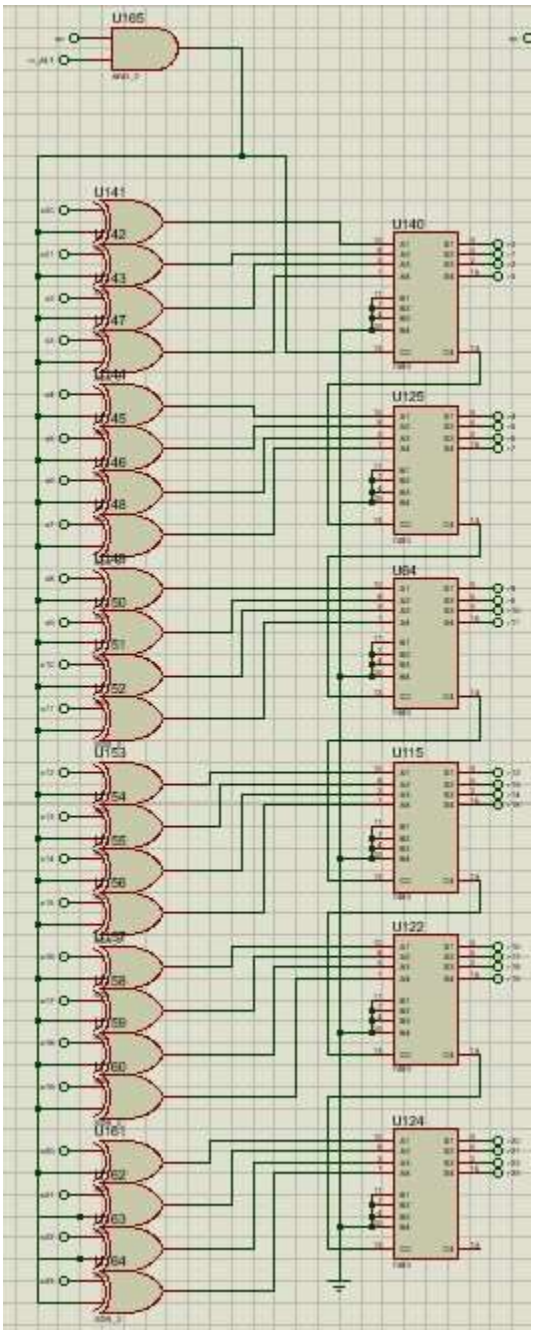
همچنین ورودی دوم یعنی B با حاصل این قسمت xor شده و ورودی cin جمع‌کننده به حاصل این قسمت وصل است، تا درصورت ۱ شدن این حاصل، مانتیس A منهای مانتیس B محاسبه شود (پس از alignment).

دقت داریم که ممکن است لازم باشد این حاصل قرینه شود – از آنجایی که مانتیس حاصل باید الزاما مثبت باشد، چنانچه قصد تفریق داشته باشیم و مانتیس B مقدار بیشتری داشته باشد، باید حاصل این بخش قرینه شود که جلوتر این امر هم محقق شده است.



در این تصویر جمع‌کننده‌ها و xorها مشاهده می‌شوند. از ۱۰ جمع‌کننده فقط ۸ جمع‌کننده در این تصویر جا شده‌اند.

حال لازم است در صورتی که m_ALT و بیت عملیات هردو یک بودند – یعنی مانتیس B بزرگتر بود و عملیات تفریق بود – حاصل قرینه شود. با توجه به اینکه عملیات جمع با دقت ۴۰ بیت انجام شده و به اتمام رسیده است و مانتیس (با ۱ اولیه) ۲۴ بیتی است، نیاز نیست این تقارن روی هر ۴۰ بیت انجام شود و کافیست روی ۲۴ بیت انجام شود (برای صرفه جویی در مصرف تراشه‌ها)



البته لازم به ذکر است که این امکان وجود دارد که بدون این تراشه‌ها و با راهکارهای دیگری بحث قرینه کردن انجام شود (یا نیاز به آن از بین برود) ولی مصرف تراشه آن حالت تفاوت خیلی زیادی ندارد زیرا به ازای هر چهار گیت xor یک تراشه 7486 استفاده می‌شود (و سایر گیت‌هایی که برای تحقق یک k-map مصرف می‌شدند نیز نیازمند تراشه اضافه‌تر بودند)

بخش ذخیره حاصل و نرمال‌سازی

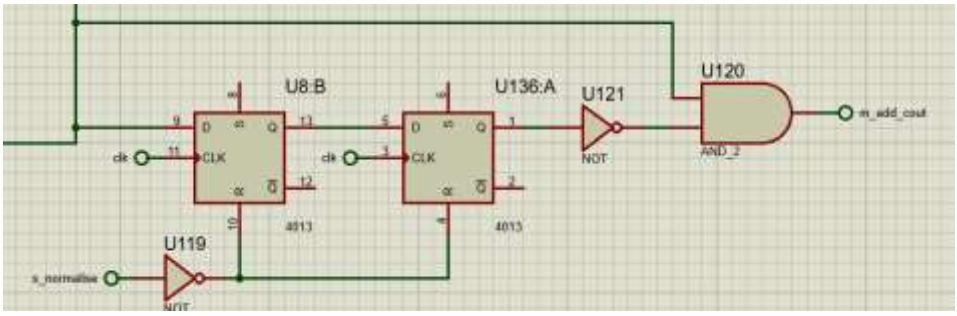
برای نرمال کردن حاصل، لازم است مانتیس بدست آمده در قسمت قبل وارد شیفت رجیستر شود تا به میزان لازم شیفت بخورد.

همچنین لازم است که تعداد شیفت‌ها برایمان مشخص باشد تا توان را اصلاح کنیم – برای این امر یک شمارنده استفاده می‌کنیم، تا مثلا اگر مانتیس سه بیت به چپ شیفت داده شد، از توانمان ۳ تا کم شود.

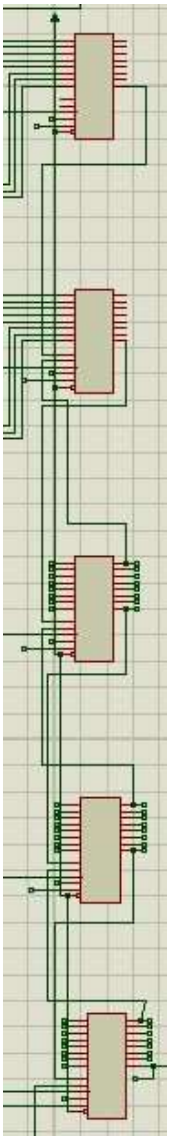
پیش از پرداختن به جزئیات این قسمت، لازم است اشاره شود که خروجی cout جمع‌کننده – که در واقع دومین رقم سمت چپ اعشار مانتیس است – وارد دو فلیپ فلاپ شده است. علت این امر این است که یکی از شروط اتمام فرایند نرمال‌سازی این است که این بیت صفر شود. بنابراین با استفاده از فلیپ فلاپ می‌فهمیم که اگر در گام قبلی این بیت یک بوده و الان هم یک است، قطعا شیفت انجام شده و نیازی نیست که عملیات نرمال‌سازی ادامه پیدا کند.

طبیعتا در استیت‌های غیر از نرمال کردن مقدار این DFF ها صفر می‌شود تا عملکرد این قسمت درست باشد.

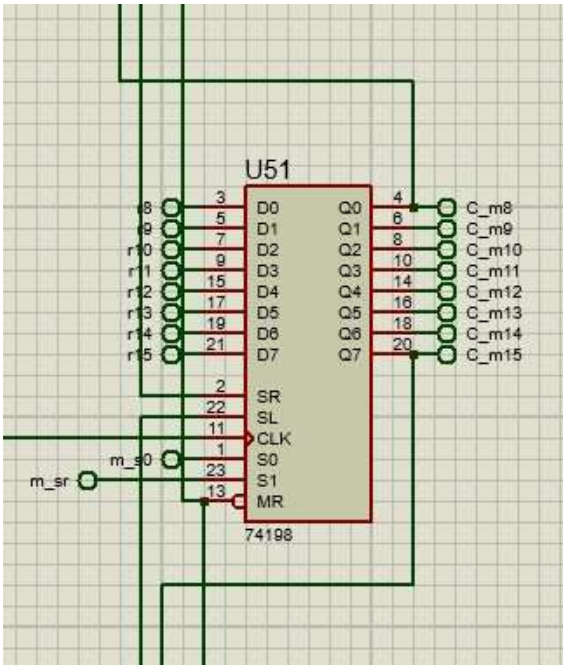
از آنجایی که در اولین سیکل کلاک normalization عملیات بارگذاری انجام می‌شود، لازم است که به اندازه یک سیکل ساعت زیادتر مقدار آن ذخیره شود، بنابراین دو DFF داریم.



همچنین مشابه قسمت‌های ابتدایی، همانطور که گفته شد مقدار حاصل عملیات میان دو مانتیس وارد شیفت رجیستر می‌شود:

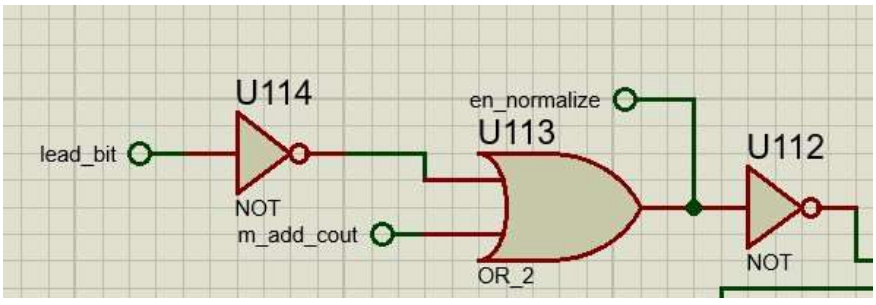


به علت مساحت این قسمت، اسامی تراشه‌ها و اتصالات آنها به درستی مشخص نیست. چنانچه روی یکی از تراشه‌ها زوم کنیم:

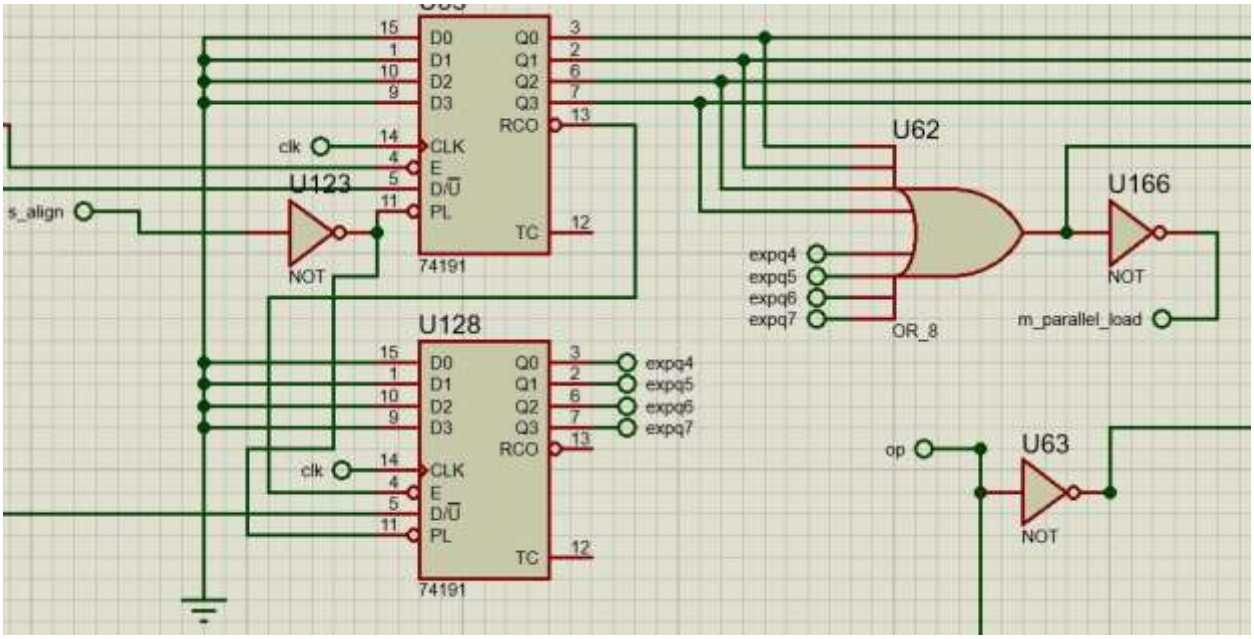


در اینجا نیز ورودی‌هایی برای s1 و s0 تعیین شده‌اند که بر اساس شرایط، عملیات بارگذاری، شیفت به چپ یا راست و یا no-op انجام شود. همچنین برخلاف قسمت ابتدایی، اینجا هم ورودی SL و هم ورودی SR به رجیسترهای بعدی و قبلی وصل هستند زیرا شیفت در هر دو جهت انجام می‌شود.

حال به سیگنال‌های ایجاد شده می‌پردازیم:



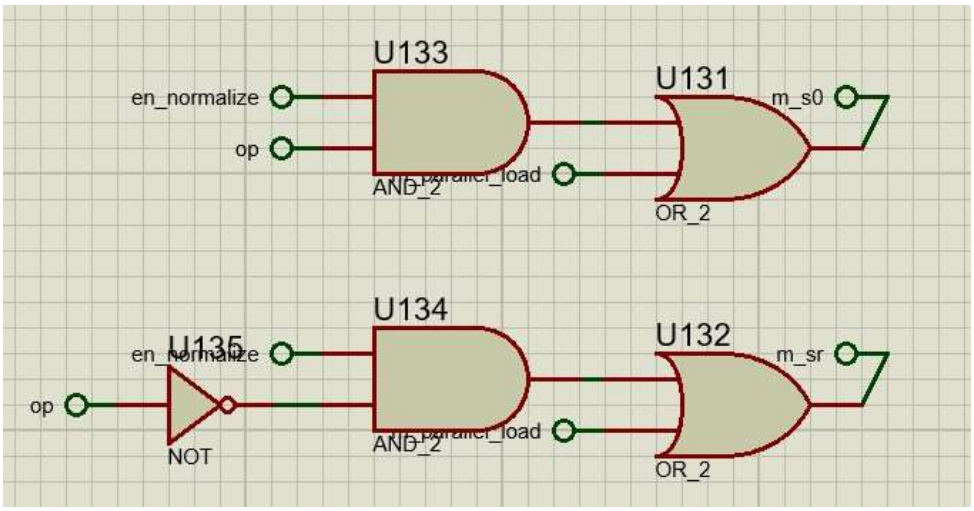
سیگنال en_normalize بر اساس اینکه آیا بیت lead (اولین بیت سمت چپ اعشار) یک باشد و اینکه بیت چپی آن یک نباشد مشخص می‌شود – چنانچه en_normalize یک باشد یعنی normalization ادامه دارد.



در تصویر بالا، دو شمارنده ripple–شده مشاهده می‌شود که تعداد سیکل‌های ساعت که عملیات normalization انجام شده است را می‌شمارند. حاصل این شمارنده‌ها (با کم یا زیاد شدن ۱) به نما اضافه خواهد شد. دقت داریم که چنانچه برای نرمال سازی به راست شیفت بدهیم شمارنده‌ها به سمت بالا می‌شمارند و چنانچه به چپ شیفت بدهیم شمارنده‌ها به سمت پایین می‌شمارند.

همچنین چنانچه در استیت align باشیم (استیت قبل از نرمال‌سازی) مقدار آنها با بارگذاری موازی صفر می‌شود.

چنانچه خروجی شمارنده صفر باشد، سیگنال m_parallel_load یک می‌شود که به منزله بارگذاری موازی برای رجیسترهایی است که مانتیس حاصل در آنها قرار می‌گیرد (و شیفت می‌خورد).



در این تصویر نحوه ایجاد شدن m_sr و m_s0 را مشاهده می‌کنیم که به ترتیب ورودی s1 و s0 رجیسترهای حاوی مانتیس حاصل هستند.

ورودی op عملیات انجام شده میان دو ماتریس است (اگر ۱ باشد تفریق و اگر ۰ جمع) و ورودی en_normalize و m_parallel_load بالاتر توضیح داده شدند.

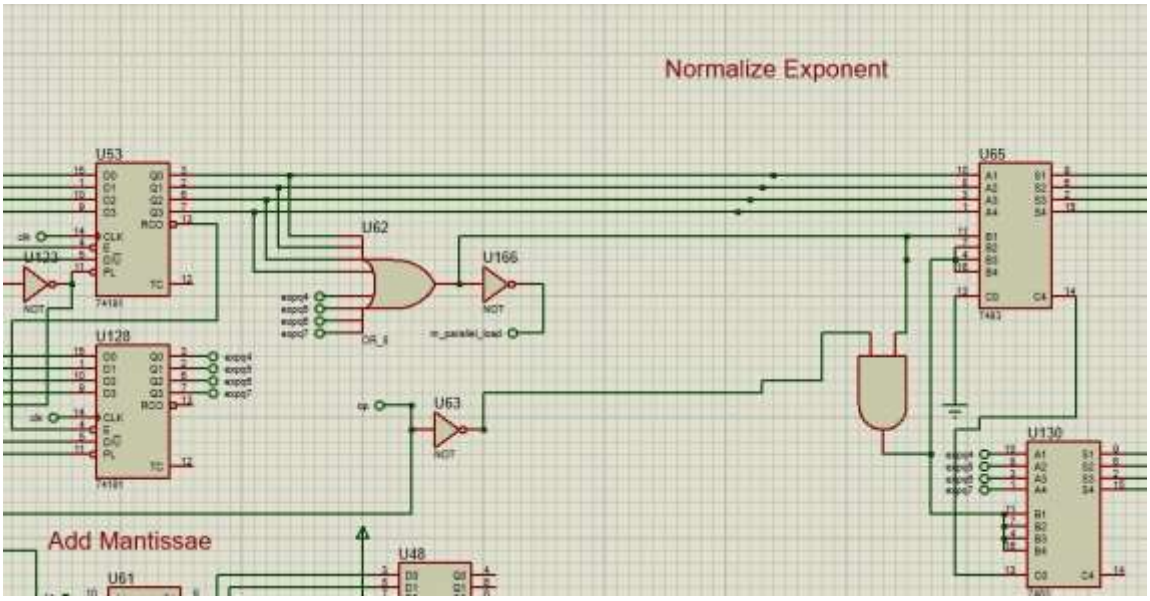
بر این اساس چنانچه در مرحله بارگذاری باشیم مقدار s_1s_0 برابر 11 است، چنانچه قصد نرمالسازی داشته باشیم اگر عملیات‌مان تفریق بوده است (و نیازمند شیفت به چپ برای نرمالسازی هستیم) این مقدار 01 و برای عملیات جمع (و شیفت به راست) 10 و در سایر حالات 00 است.

بر این اساس، عملیات نرمالسازی برای مانتیس انجام می‌شود. حال دقت داریم که توان نیز در این عملیات دستخوش تغییر می‌شود و همانطور که گفته شد یک شمارنده برای این منظور قرار داده بودیم.

از طرفی، وقتی شمارنده صفر بود، عملیات بارگذاری انجام می‌شد و از کلاک بعدی، شیفت انجام می‌شد.

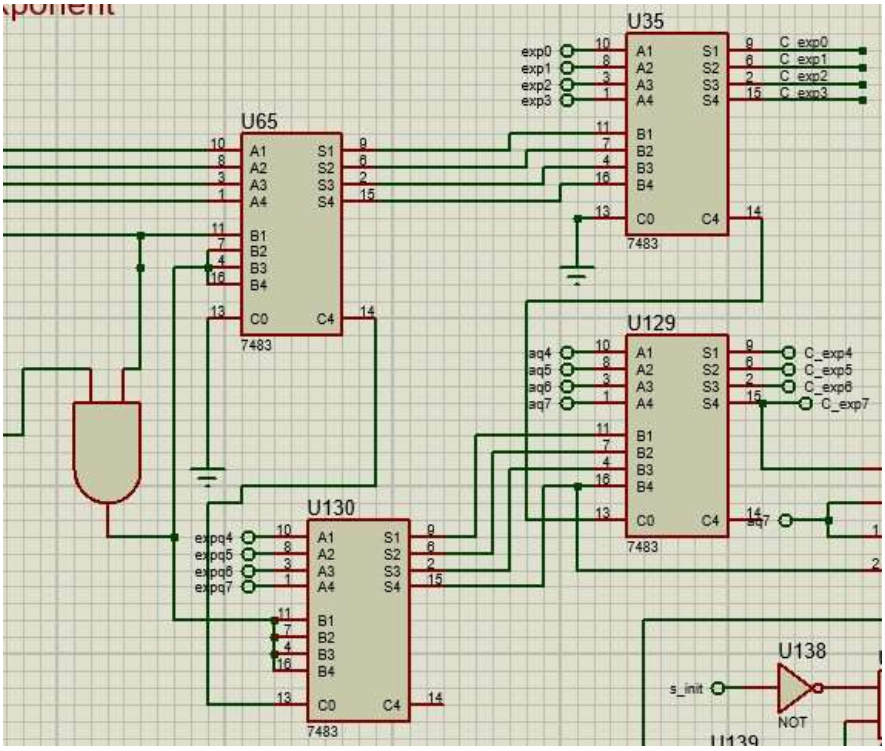
بنابراین مقدار شمارنده به اندازه یک کلاک بیش از حد تغییر می‌کند. چنانچه رو به بالا بشماریم، مقدار نمای حاصل برابر با نما به اضافه‌ی مقدار شمارنده منهای یک است زیرا یک کلاک زیادی به بالا شمرده‌ایم. حالتی که به پایین بشماریم چون یک کلاک زیادی به پایین شمرده‌ایم باید به اضافه یک کنیم.

برای این منظور، از دو تراشه 7483 استفاده شده است:



حاصل بدست آمده در 7483 ها برابر است با عدد شمارنده منهای یک (اگر شمارش به بالا باشد) یا عدد شمارنده منهای یک (اگر شمارش به پایین باشد)

حال کافیست این مقدار با بزرگترین نما میان دو عدد ورودی جمع زده شود تا نمای جواب بدست آید و جواب کامل شود:

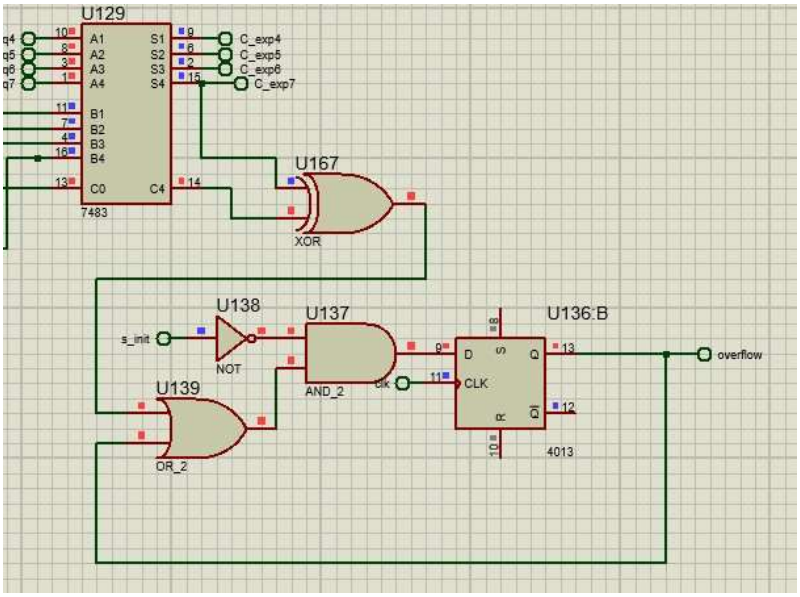


بنابراین به این شکل نما و مانتیس و علامت حاصل بدست می‌آیند.

حال لازم است خروجی‌های دیگر (سیگنال‌های خروجی) را بدست آوریم.

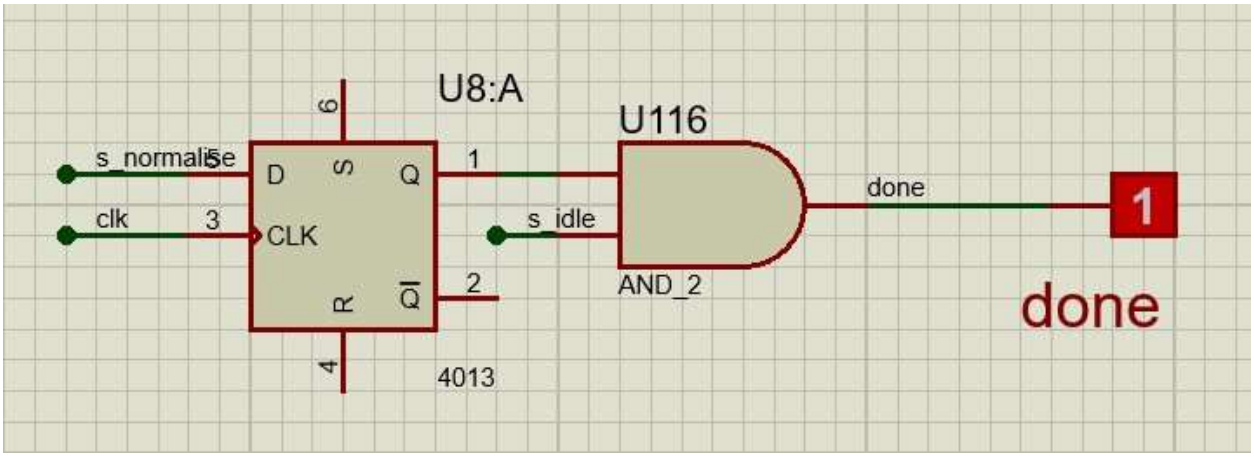
تولید سیگنال overflow

این سیگنال برای مشاهده سرریز روی نما است.



علت استفاده از DFF این است که این احتمال وجود دارد که در حین نرمال‌سازی سرریز رخ بدهد، ولی عملیات نرمال‌سازی در آن کلاک به اتمام نرسد. در این صورت در ادامه سرریز قابل مشاهده نخواهد بود (به عنوان مثال هنگام افزایش نما، از 111 به 000 برویم و سپس به 001 برویم، در این حالت سرریز بدون DFF قابل مشاهده نخواهد بود و باید هنگامی که 111 سرریز کند این امر جایی ذخیره شود)

تولید سیگنال done

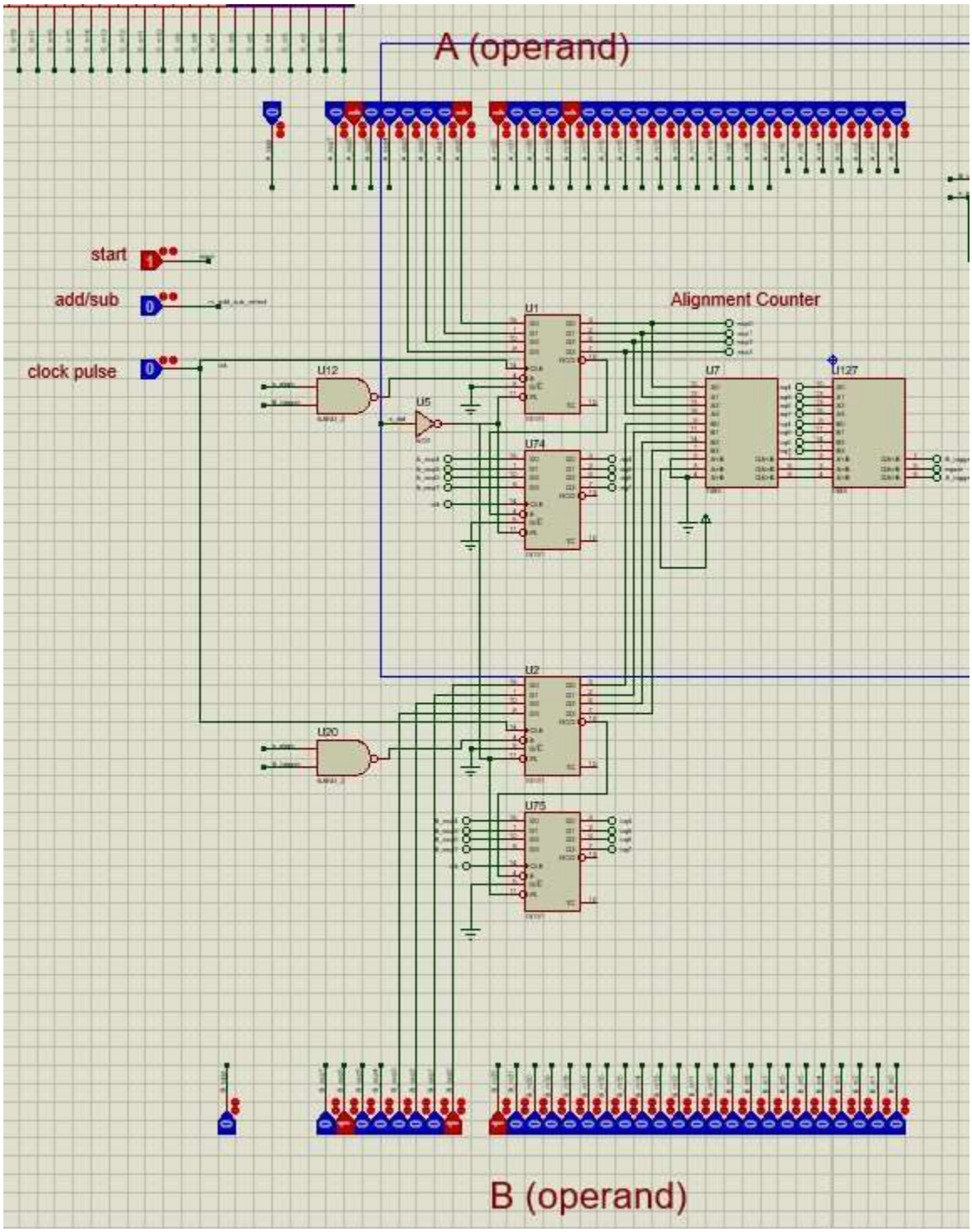


چنانچه استیت قبلی normalize باشد و استیت کنونی idle باشد، عملیات محاسبه به اتمام رسیده است و این خروجی ۱ می‌شود.

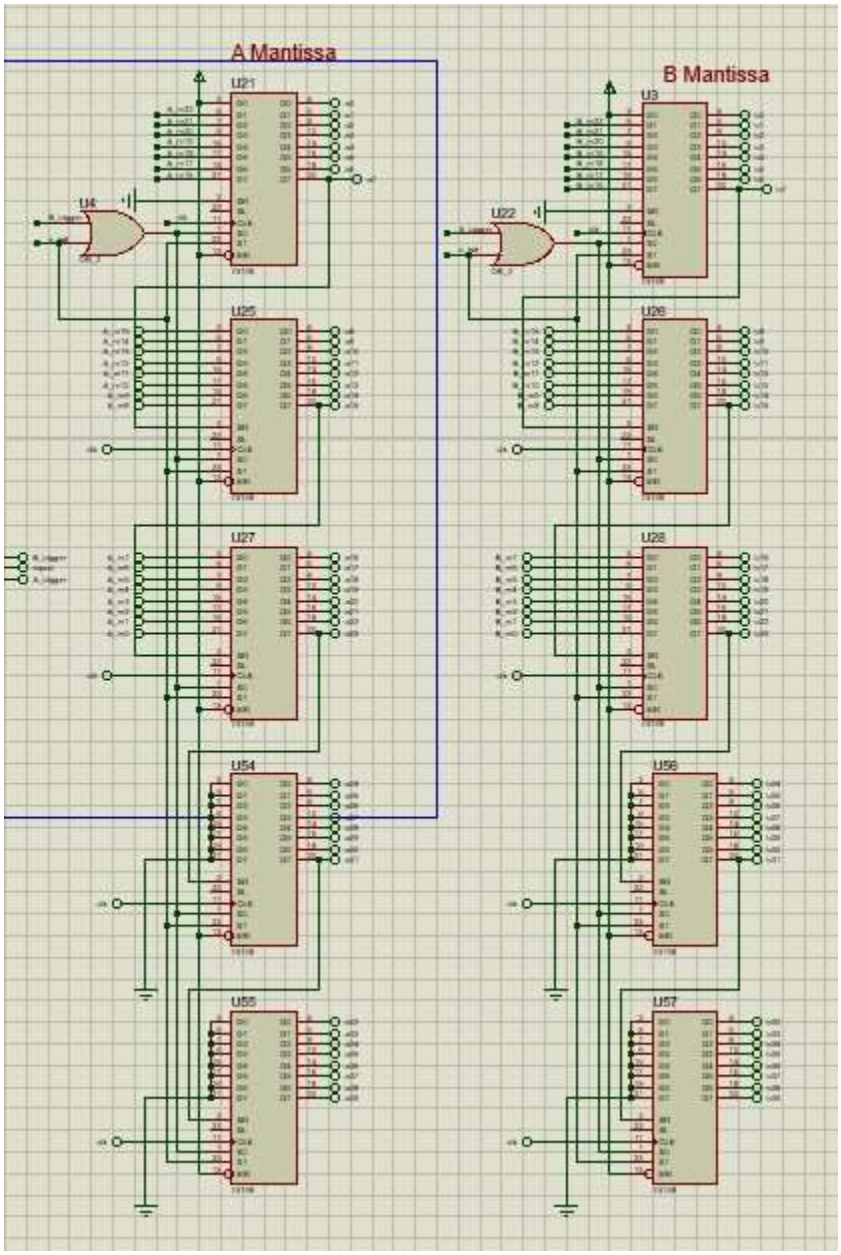
تصاویر کلی از مدار نهایی

در این بخش تصاویری کلی از مدار ایجاد شده را مشاهده می‌کنیم.

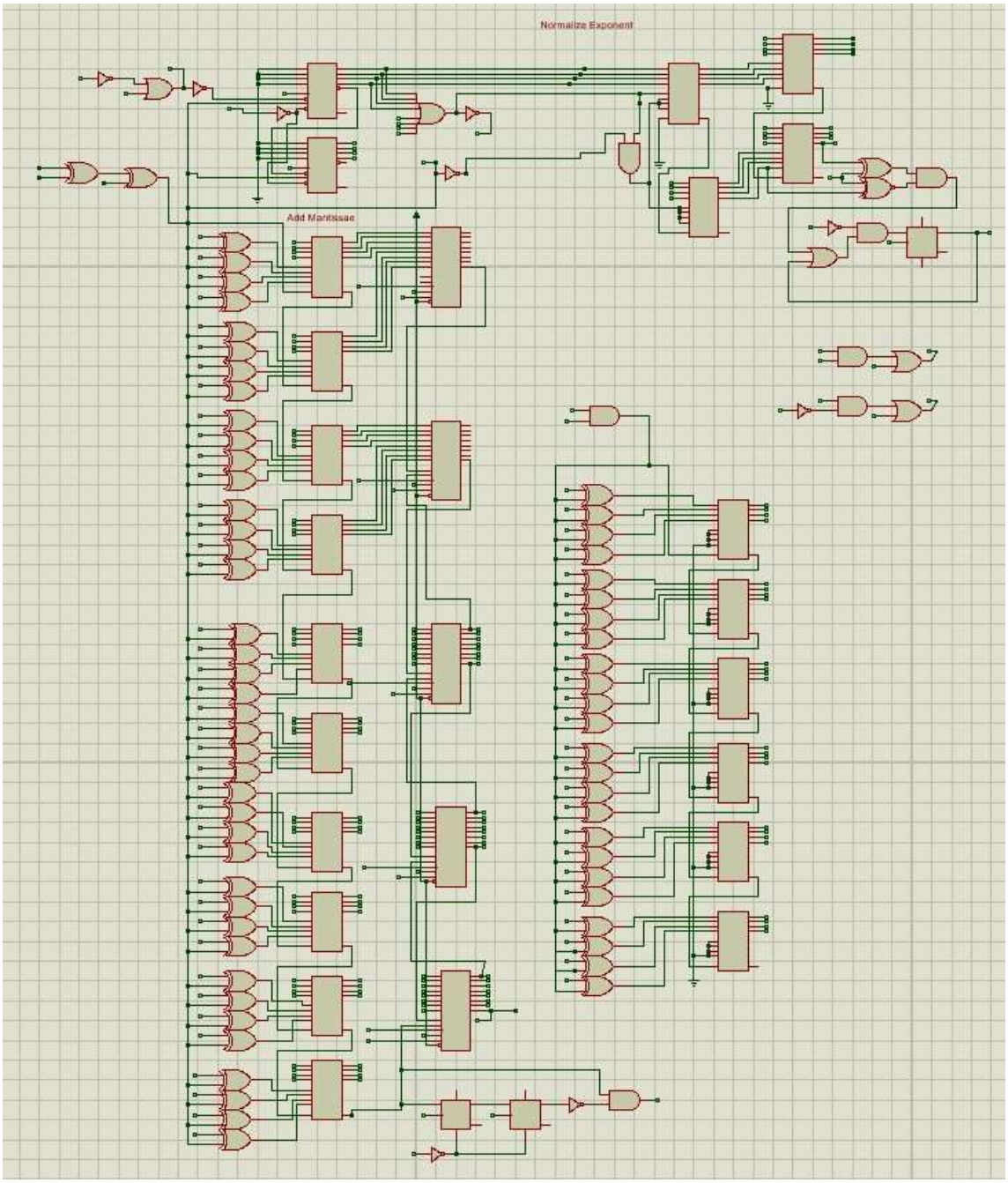
ورودی‌ها و alignment (شمارنده‌ها)



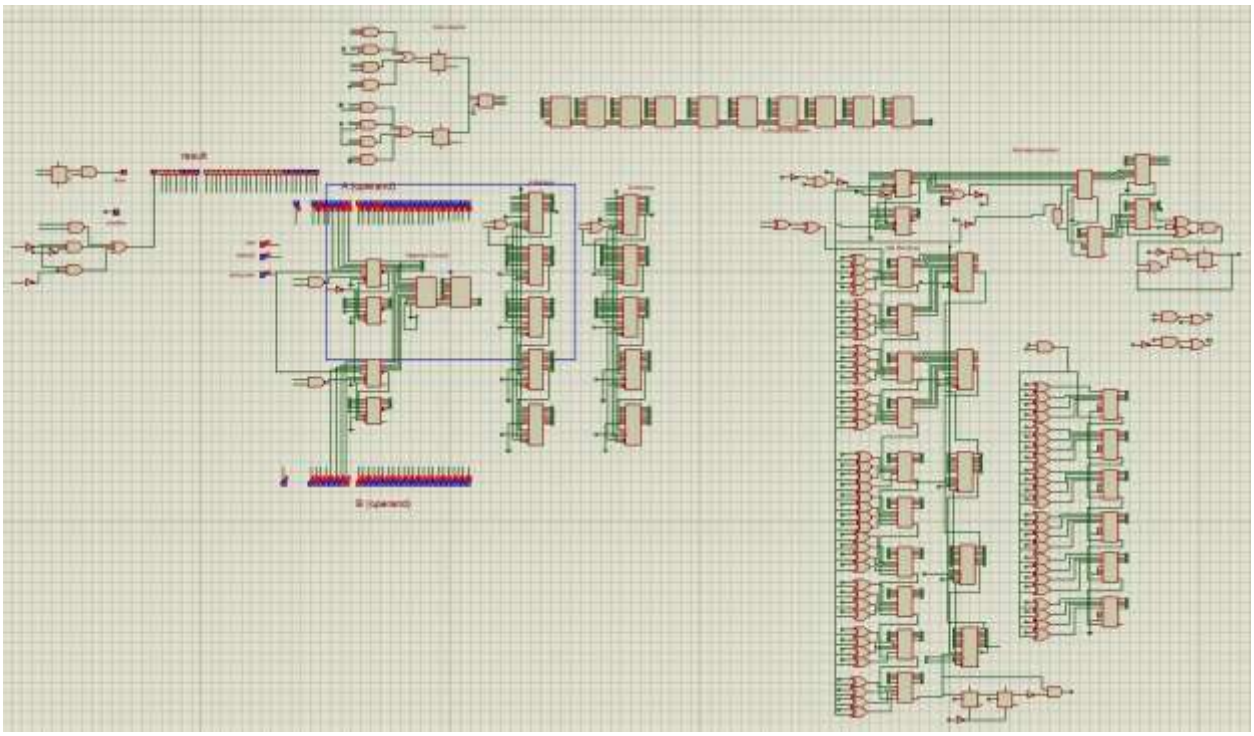
ثبات‌های ذخیره کردن مانتیس‌های ورودی‌ها



جمع مانتیس‌ها و نرمال‌سازی



نمای کلی مدار



در ادامه با دادن ورودی‌های نمونه و مشاهده پاسخ، از صحت عملکرد مدار طراحی شده مطمئن می‌شویم.

بررسی عملکرد مدار با ورودی‌های نمونه

در این قسمت مطابق استاندارد IEEE-754 اعدادی را به عنوان ورودی می‌دهیم و خروجی را مشاهده می‌کنیم.

در هر صفحه یک ورودی نمونه به همراه خروجی و حاصل مورد انتظار و حاصل بدست آمده و تحلیل آن قرار داده شده است.

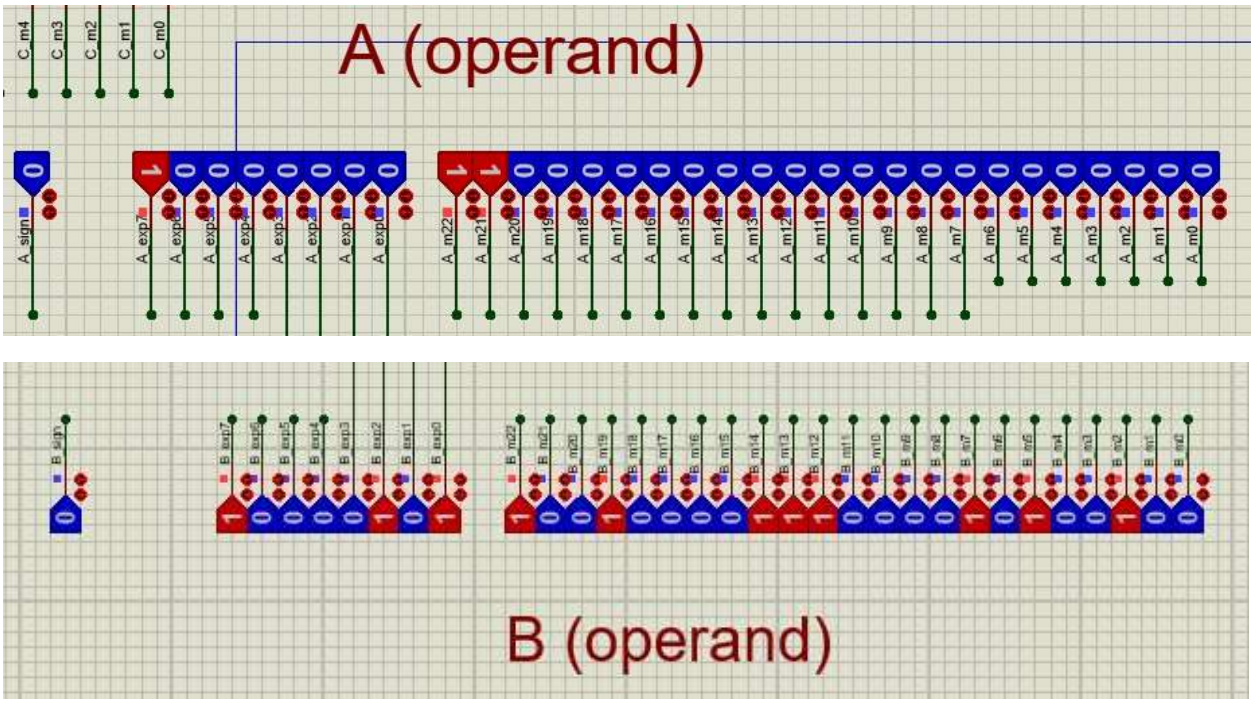
نمونه ورودی اول

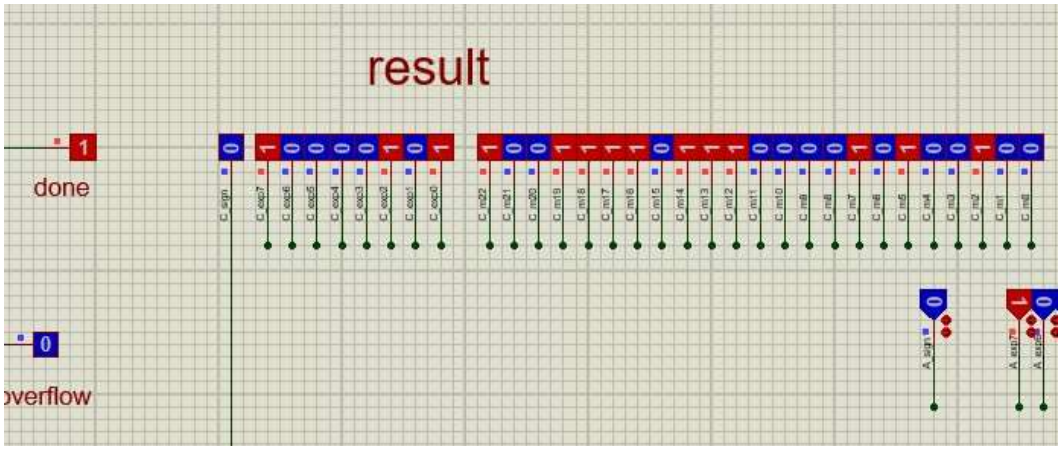
A: 3.5 (Binary: 01000000011000000000000000000000)

B: 100.22 (Binary: 01000010110010000111000010100100)

جواب مورد انتظار:

A+B = 103.72 (Binary: 01000010110011110111000010100100)





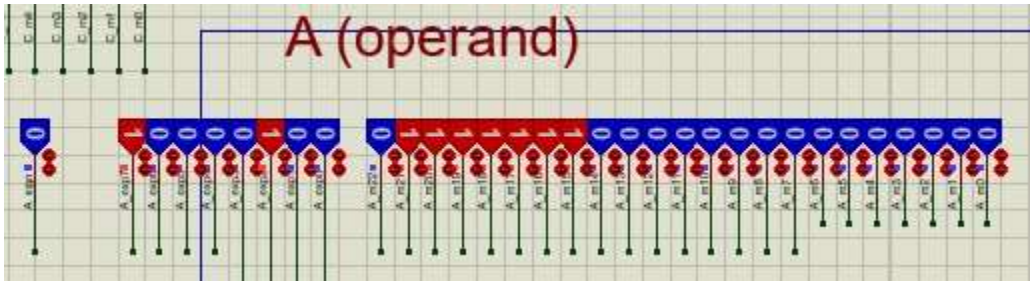
نمونه ورودی دوم

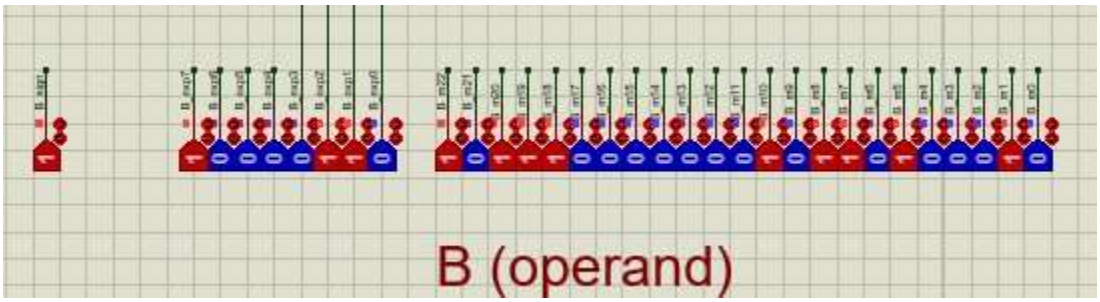
A: 47.875 (Binary: 01000010001111111000000000000000)

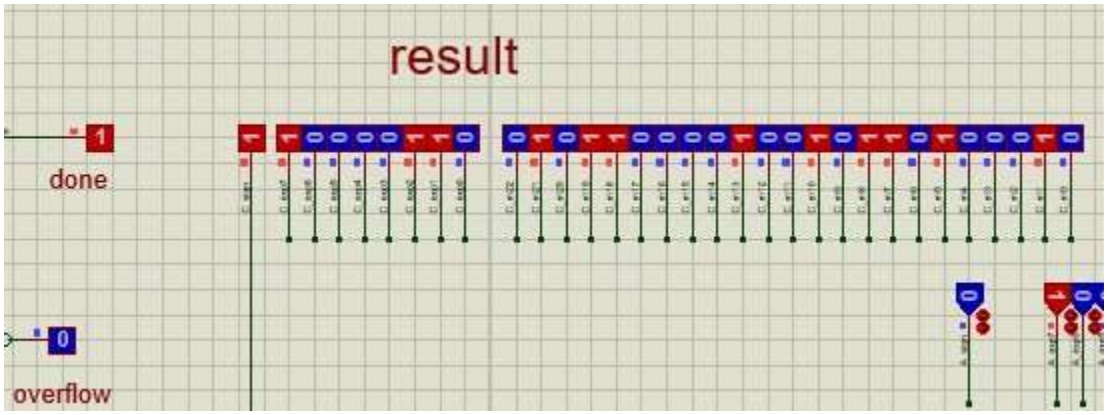
B: -220.022 (Binary: 11000011010111000000010110100010)

جواب مورد انتظار:

A+B = -172.147 (Binary: 11000011001011000010010110100010)







نمونه ورودی سوم

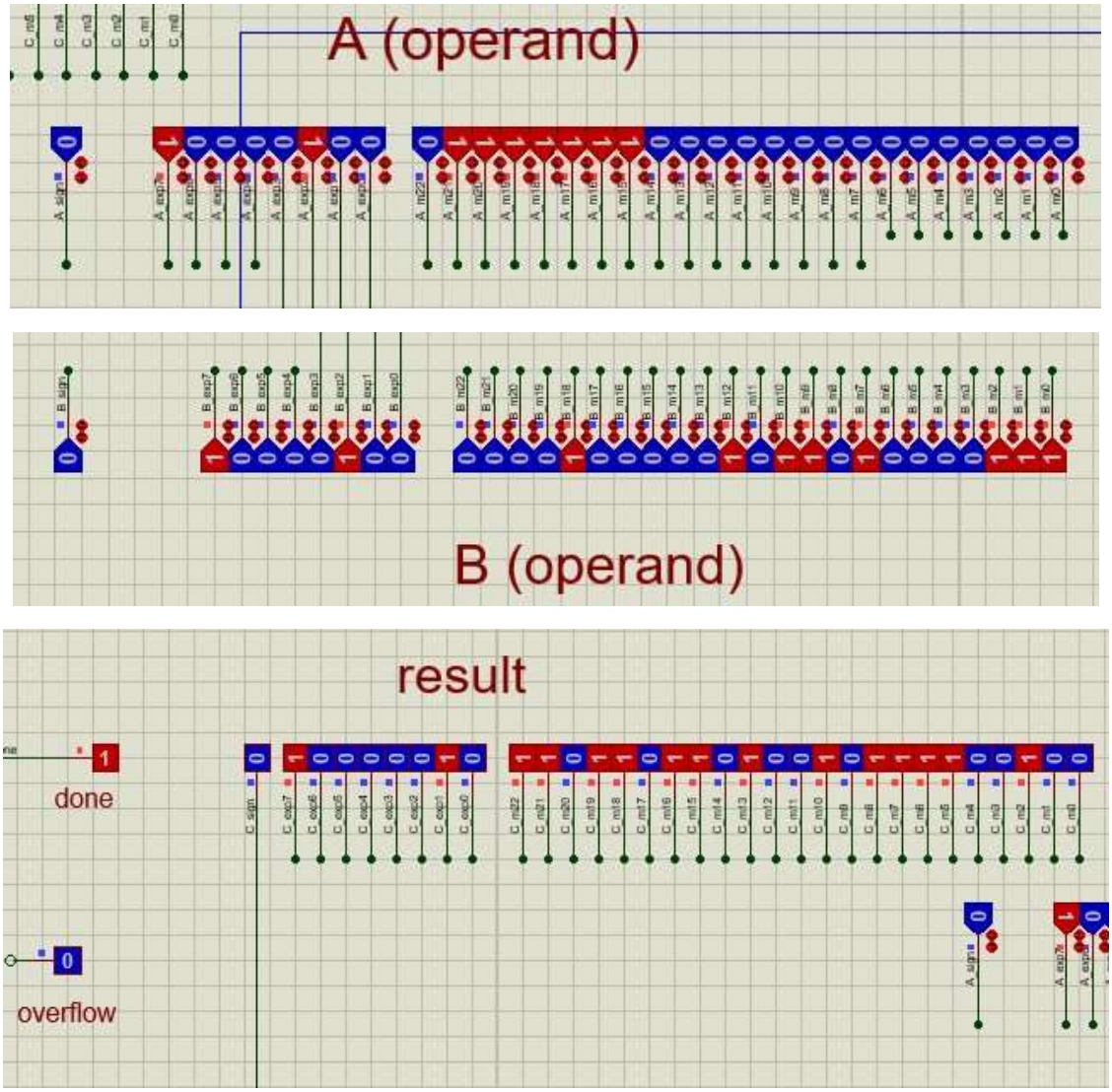
A: 47.875 (Binary: 01000010001111111000000000000000)

B: 33.022 (Binary: 01000010000001000001011010000111)

جواب مورد انتظار:

A-B = 14.853 (Binary: 01000001011011011010010111100011)

دقت داریم که در اینجا به اندازه 2^{-21} خطای محاسباتی داریم و سه بیت آخر بجای 011 برابر 100 هستند.
علت این امر، سیاست truncate بجای roundup در محاسبه جمع دو مانتیس است.



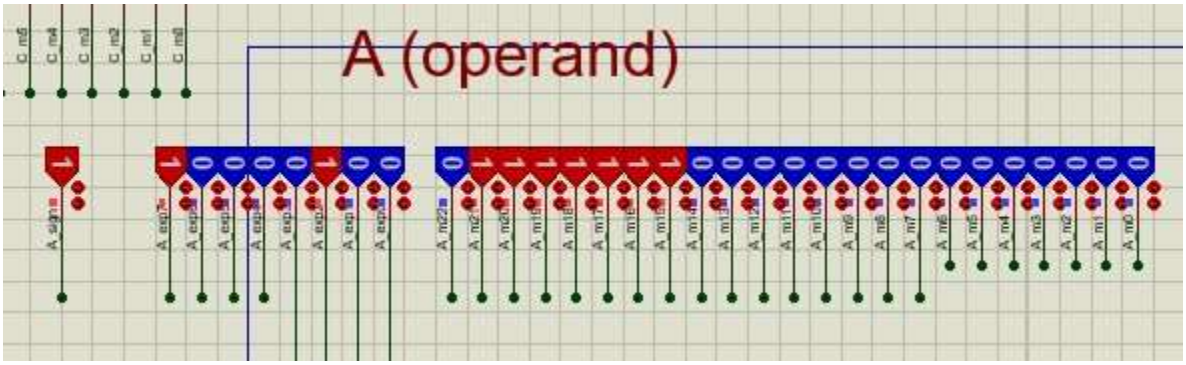
نمونه ورودی چهارم

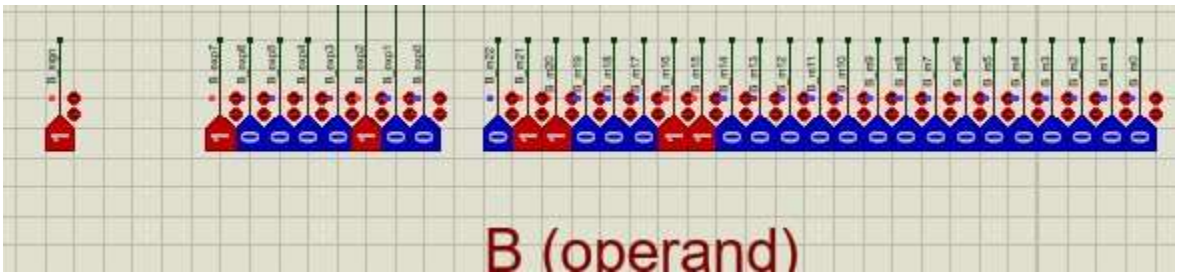
A: -47.875 (Binary: 11000010001111111000000000000000)

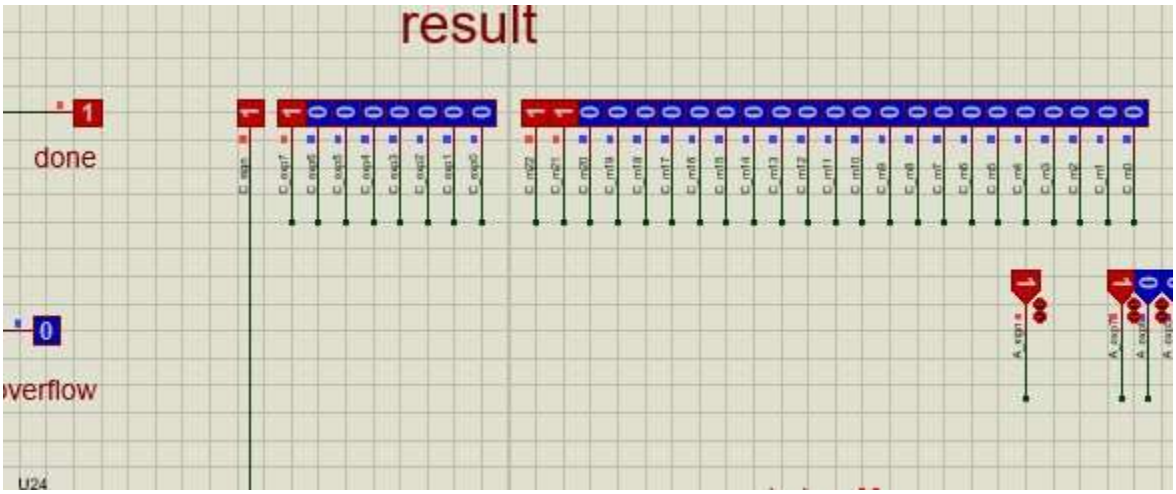
B: -44.375 (Binary: 11000010001100011000000000000000)

جواب مورد انتظار:

A-B = -3.5 (Binary: 11000000011000000000000000000000)







نمونه ورودی پنجم (بررسی اورفلو)

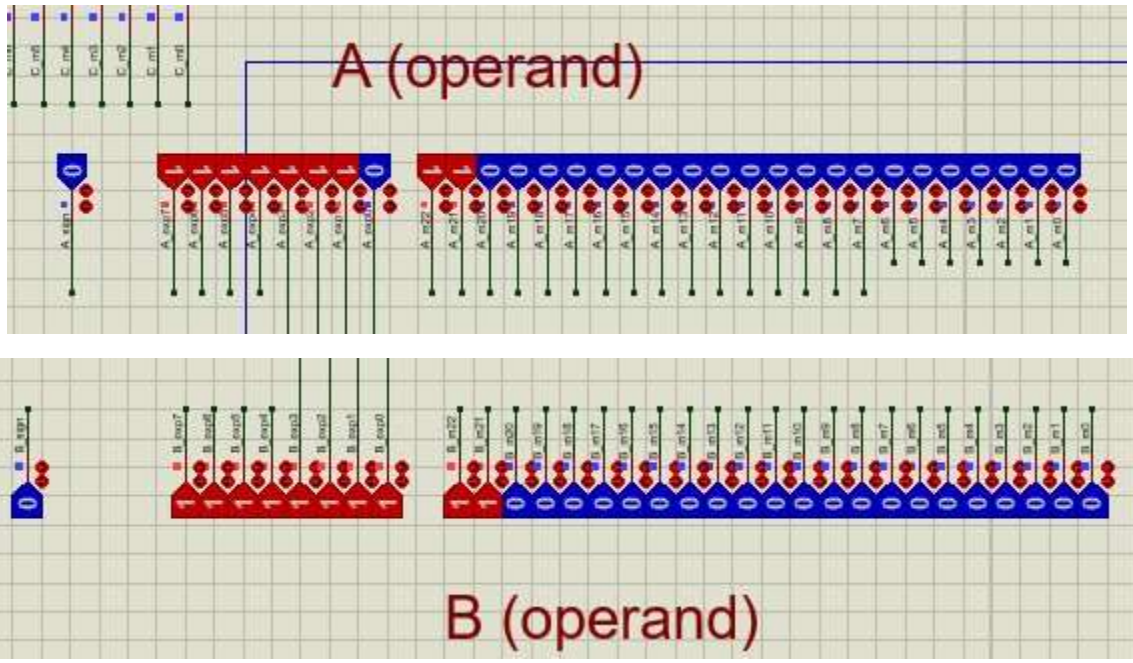
در این نمونه عملکرد اورفلو را بررسی می‌کنیم.

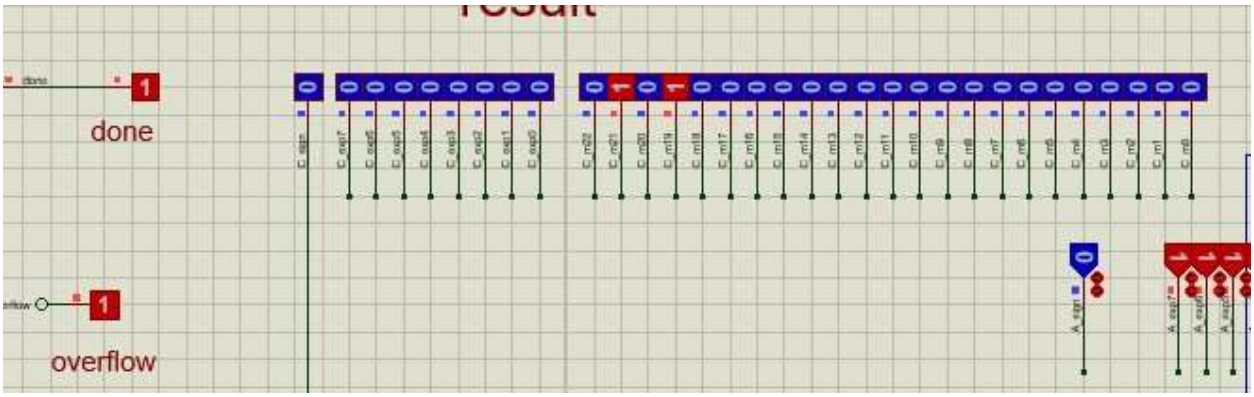
A: 2.97747071056e+38 (Binary: 01111111011000000000000000000000)

B: NaN (Binary: 01111111111000000000000000000000)

واضح است که چنانچه آنها با هم جمع زده شوند، نما سرریز خواهد کرد، زیرا در ابتدا نمای حاصل 11111111 است و مانتیس آن برابر جمع 1.11 و 0.111 که برابر است با 10.101 . سپس باید نما یکی زیاد شود و مانتیس یکی به راست شیفت داده شود که منجر به اورفلو می‌شود.

نمای نهایی برابر 00000000 خواهد شد و مانتیس برابر 00...010100 خواهد شد.





مشاهده می‌شود که خروجی overflow به درستی عمل کرده است.