دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر
گزارش کار چهارم درس آزمایشگاه معماری

عنوان:

جمع/تفریق کننده ممیز شناور

Floating Point Adder/Subtractor

نگارش

کیان قاسمی ۴۰۱۱۰۲۲۶۴
آیین پوست فروشان ۴۰۱۱۰۵۷۴۲
دیبا هادی اسفنگره ۴۰۱۱۱۰۲۴۵

استاد

دکتر حمید سربازی آزاد

دستیار آموزشی

مهندس عطیه غیبی فطرت

تیر ۱۴۰۳

فهرست مطالب

فهرست تصاویر

۱ مقدمه

در این آزمایش قصد داریم یک جمع کنند/تفریق کننده ۳۲ بیتی در فرمت $IEEE754$ با استفاده از نرم افزار پروتئوس شبیه سازی کنیم، اعداد در فالب $IEEE754$ و نرمال ورودی داده می شوند که با توجه به ۳۲ بیتی بودن آنها ۱ بیت برای علامت، ۸ بیت برای *exponent* و ۲۳ بیت برای برای مانتیس در نظر گرفته شده، بیت ورودی add/sub نوع عملیات را نشان می دهد و هر گاه *start* از ۱ به ۰ تغییر داده شد، کلاک شروع به کار می کند و هرگاه خروجی *end* برابر با ۱ شد محاسبات تمام شده و خروجی داده شده خروجی نهایی است، همچنین در این مدار حالت های $underflow$ و $overflow$ نیز بررسی شده و در نهایت در خروجی آورده شده است، سلسله مراتب کلی طراحی این مدار به صورت زیر است:

۱. ساخت قطعات اولیه مانند مالتی پلکسر یا *adder* ۲۵ بیتی
۲. محاسبه علامت ورودی دوم با توجه به حالت تفریق یا جمع بودن عملیات.
۳. پیدا کردن عدد با *exponent* کوچک تر و یکی کردن *exponent* های ورودی با شیفت دادن
۴.عملیات جمع روی مانتیس ها و به دست آوردن مقدار نرمال نشده خروجی و همچنین علامت خروجی نهایی
۵. نرمال کردن خروجی و تشخیص حالت های $overflow$ و یا $underflow$
۶. آزمایش مدار و بررسی درستی عملکرد آن

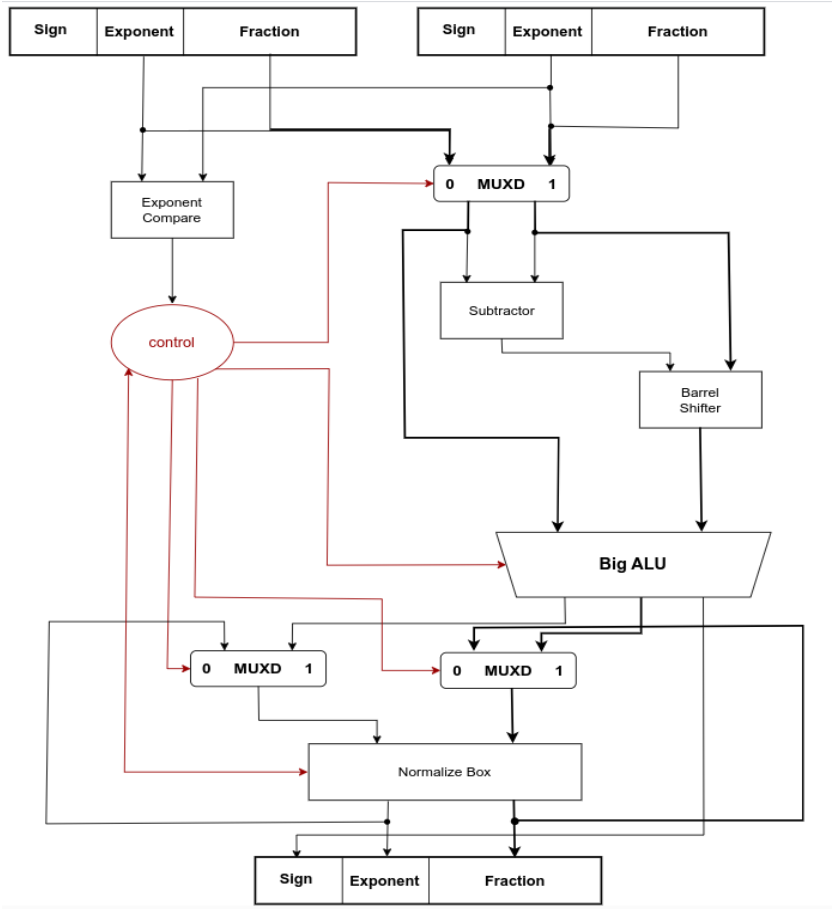
در نهایت بلوک دیاگرام نهایی برای شبیه سازی مدار و قطعات استفاده شده به صورت زیر می باشد:

۱.۱ قطعات لازم

- گیت های منطقی *OR*، *AND*، *XOR* ۲ بیتی و گیت منطقی *NOT*
- گیت های منطقی *OR*، *AND* و *XNOR* ۸ بیتی و گیت منطقی *OR* ۳ بیتی
- جمع کننده های ۲ بیتی و ۴ بیتی (قطعات 7482 و 7483)
- *D* فلیپ فلاپ ۶ بیتی (قطعه 74174)
- مالتی پلکسر ۴ بیتی (قطعه 74157)

۲.۱ بلوک دیاگرام نهایی

بلوک دیاگرام نهایی و کلی این جمع کننده/ تفریق کننده به صورت زیر می باشد(دقت کنید که واحد کنترلی به عنوان یک بلاک در مدار وجود ندارد و سیگنال های آن به صورت جدا آورده شده است) :



شکل ۱: بلاک دیاگرام کلی

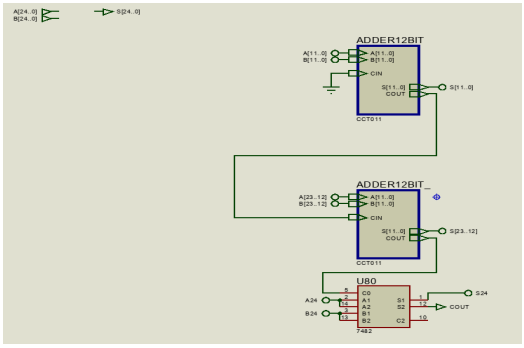
۲ شبیه سازی مدار

ابتدا با استفاده از نرم‌افزار *proteus* مدار مورد نظر را طراحی و آزمایش کردیم. در بخش‌های قبلی با الگوریتم کلی پیاده‌سازی مدار آشنا شدیم. حال طراحی مدار را به صورت مرحله به مرحله نشان می‌دهیم.

۱.۲ مرحله اول: طراحی قطعات اولیه

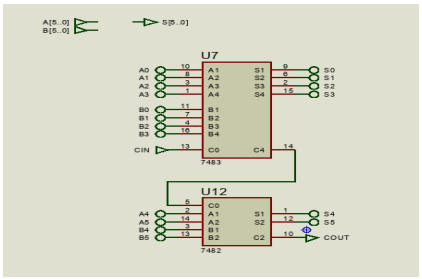
به طوری کلی برای طراحی این مدار نیاز به چند قطعه داریم که به شرح زیر است:

- جمع کننده ی ۲۵ و ۸ بیتی: در تمامی قسمت های مدار از جمع کننده هایی استفاده شده است که عملیات روی مانتیس یا *exponent* ورودی انجام می دهند، در شکل زیر طراحی یک جمع کننده ۲۵ بیتی آورده شده که از دو واحد جمع کننده ۱۲ بیتی تشکیل شده که هر کدام از این واحد ها از دو جمع کننده ۶ بیتی تشکیل شده اند:



شکل ۲: جمع کننده ۲۵ بیتی

داخل یک جمع کننده ۶ بیتی نیز به صورت زیر است که از دو قطعه ۷۴۸۲ و ۷۴۸۳ تشکیل شده است:

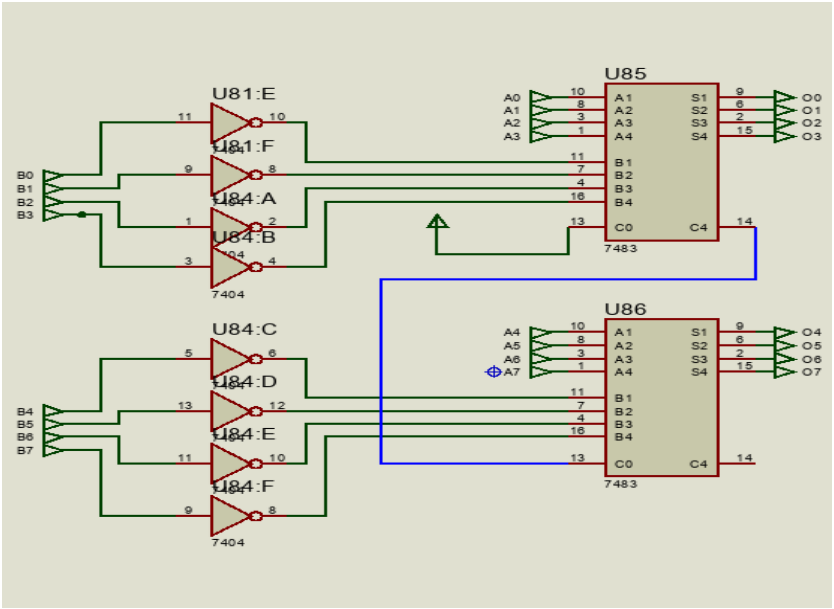


شکل ۳: جمع کننده ۶ بیتی

- مالتی پلکسر: در تمامی قسمت های مدار از مالتی پلسکر های ۳۲، ۲۵۸ بیتی استفاده شده است که مشابه قسمت قبل هر کدام از دو قسمت تشکیل شده اند و در نهایت برای ساخت تمام آن های از مالتی پلکسر ۴ بیتی و قطعه ۱۰۱۵۸ استفاده شده است.

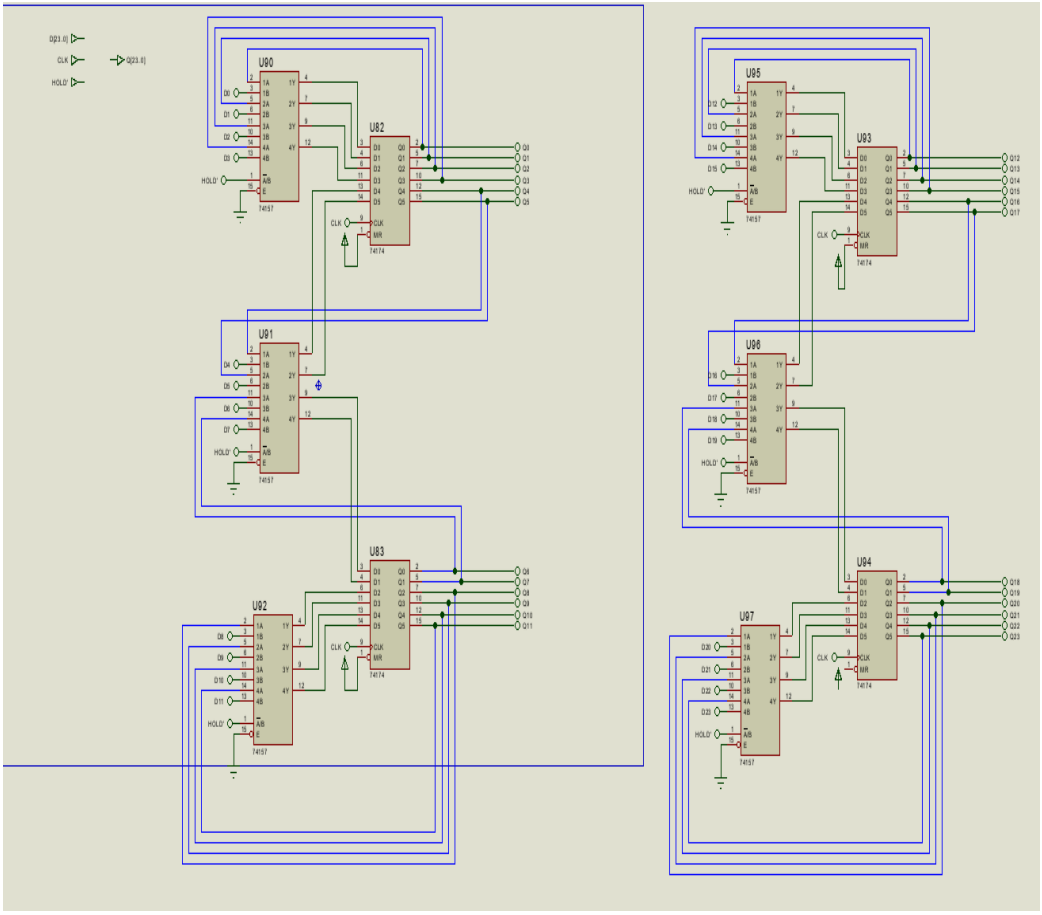
- *Not* ۲۵ بیتی: برای حالت بندی روی ورودی های *ALU* با توجه به علامت ورودی ها و حالت جمع/تقسیم ممکن است نیاز باشد مکمل های ورودی را برای جمع در نظر بگیریم، در این قسمت از *Not* ۲۵ بیتی با یک بیت *Enable* استفاده شده است که هر بیت ورودی با بیت *Xor Enable* شده است.

- تفریق کننده ۸ بیتی: برای به دست آوردن قدر مطلق اختلاق دو *exponent* نیاز به یک تفریق کننده ۸ بیتی داریم که پیاده سازی آن نیز به صورت زیر است:



شکل ۴: تفریق کننده ۸ بیتی

- *D Flip flop* ۸ و ۲۴ بیتی : در ساخت مدار برای قسمت نرمال کردن خروجی نیاز به ذخیره سازی *exponent* و مانتیس داریم که برای این کار از دی فلیپ فلاپ استفاده کرده ایم، ساختار کلی *DFF* ۲۴ بیتی به صورت زیر است:



شکل ۵: DFF ۲۴ بیتی

۲.۲ مرحله دوم: محاسبه علامت ورودی دوم

با توجه به این که در واحد ALU تنها عملیات جمع صورت می گیرید، علامت تفریق نیز برای ورودی دوم حساب می شود و درواقع علامت نهایی ورودی دوم به صورت زیر است:

$$Sign(B) = initialsign_B \oplus add/sub$$

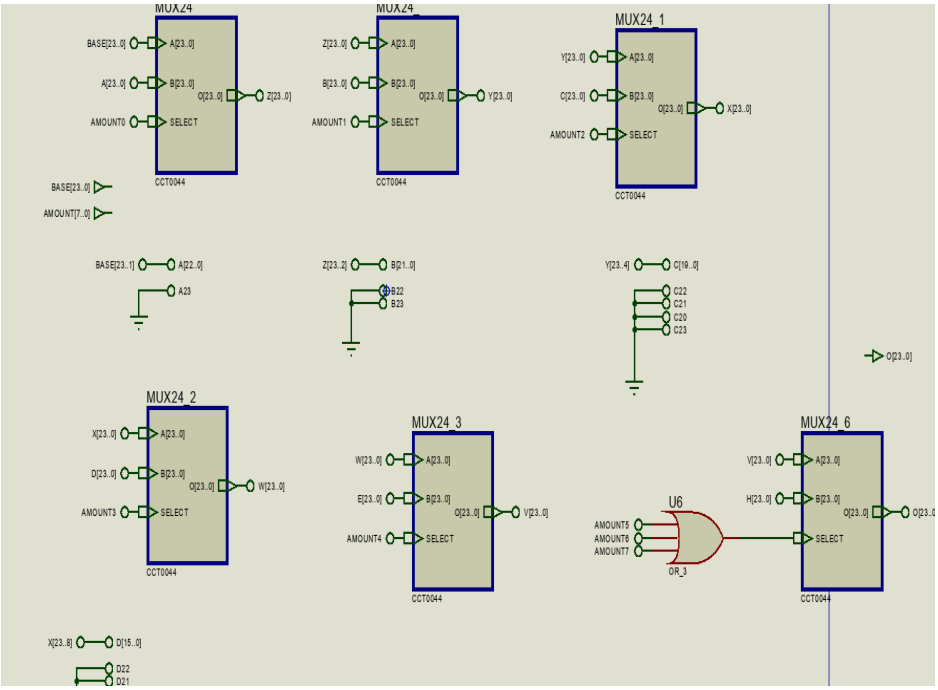
که با توجه به این علامت به ورودی ALU یا خود ورودی دوم و یا مکمل دو آن عدد داده می شود.

۳.۲ مرحله سوم: مقایسه *exponent* های ورودی

در این قسمت ابتدا *exponent* ورودی ها را به واحد *EXPONENTCMP* می دهیم که خروجی آن به صورت زیر است:

$$AIS\,BIGGER = 1\ if\ \ exponent_A > exponent_B\ \ else\ 0$$

و سپس با توجه به این بیت *exponent* بزرگ تر مشخص می شود و باید عددی که *exponent* کوچک تری دارد را شیفت چپ دهیم تا در نهایت به عدد با *exponent* برابر برسیم، برای این کار ابتدا با توجه به بیت *AIS BIGGER* عدد با *exponent* بزرگ تر را *ANC* و دیگری را *BNC* در نظر میگریم، سپس اختلاف قدر مطلق *exponent* ها را محاسبه و سپس به همین مقدار *BNC* را شیفت چپ می دهیم، عملیا شیفت دادن با استفاده از یک *Barrel shifter* صورت می گیرید که ساختار درونی آن به شکل زیر است:



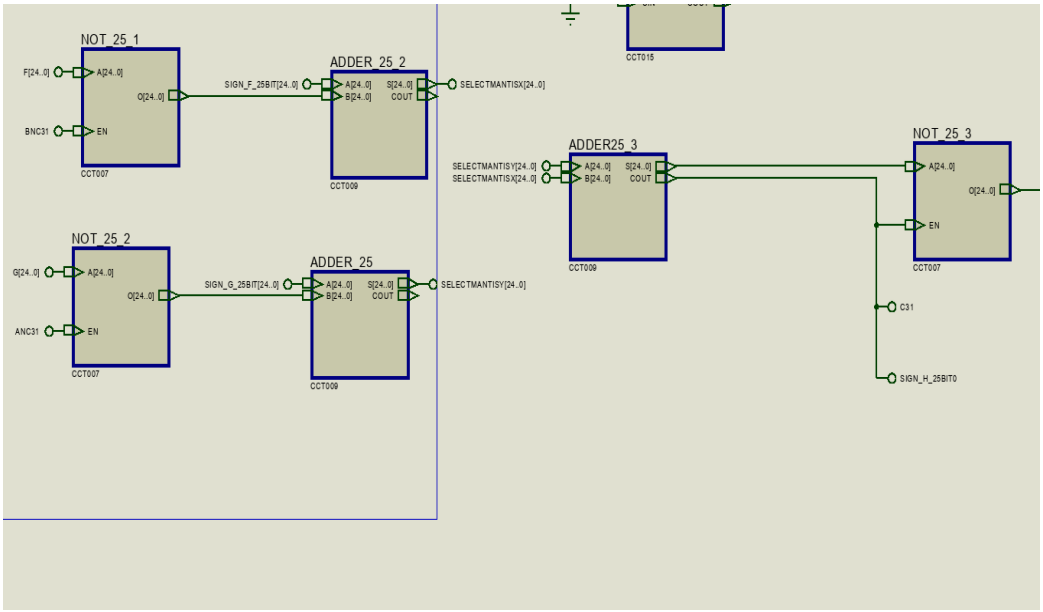
شکل ۶: ساختار *barrel shifter*

۴.۲ مرحله چهارم: عملیات جمع(واحد ALU)

ابتدا یک به ابتدای هر دو مانتیس نهایی قسمت قبل اضافه می کنیم و هر کدام از آن ها را ۲۵ بیت در نظر میگریم سپس همانطور که در قسمت قبل توضیح داده شد اگر ورودی ای منفی بود مکمل دو آن را در نظر میگریم و در نهایت آن ها را به یک جمع کننده ۲۵ بیتی ورودی می دهیم که جمع اولیه دو مانتیس را خروجی می دهد که در نهایت بیت MSB این جمع علامت نهایی خروجی می شود چرا که به طور کلی علامت خروجی به صورت زیر تعیین می شود:

$$sign_{sum} = MSB_{sum} \oplus overflow$$

و چون اگر $overflow$ یا $underflow$ اتفاق بیفتد خروجی درواقع صحیح نیست ، که $overflow$ را در نظر نمیریگم. و در نهایت علامت نهایی را به عنوان بیت $enable$ به بلاک Not میدهیم که اگر علامت منفی بود مکمل جمع نهایی در نظر گرفته شود، این قسمت از مدار در شکل زیر آمده است:



شکل ۷: جمع دو مانتیس با $exponent$ یکسان

۵.۲ مرحله پنجم: نرمال سازی و محاسبه خروجی نهایی

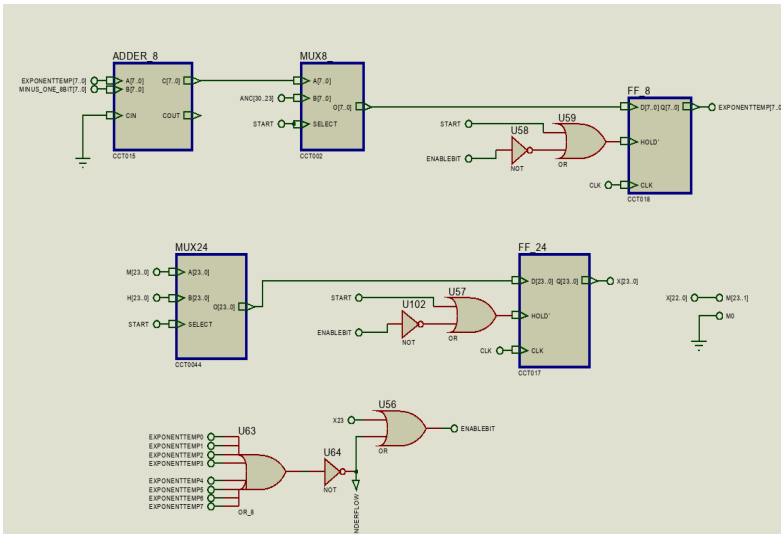
در این قسمت شروع به شیفت چپ دادن مانتیس به ازای هر یک واحد کاهش از $exponent$ می کنیم، و این کار تا زمانی ادامه پیدا می کند که یا $underflow$ اتفاق بیفتد و یا خروجی ۰ باشد و حالت دیگر این که بیت ۲۴ ام مانتیس یک شود که به این معنی است که مقداری که اضافه داشتیم و داخل ۲۳ بیت مانتیس جا نمیشده تمام شده و فرم نهایی مانتیس به صورت نرمال درامد به صورت کلی شرط ذخیره کردن خروجی و انجام ندادن عملیات شیفت چپ در این قسمت به صورت زیر می شود:

$$Enable_{bit} = Zero_{mode}\ OR\ MSB_{mantis}$$

که البته زمانی این بیت استفاده میشود که $start$ ۰ شود و در استیت انجام عملیات باشیم که یعنی بیتی که به DFF ها داده میشود به صورت زیر میشود:

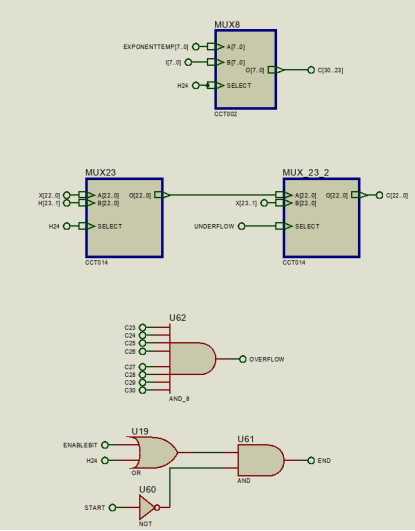
$$Hold = \sim Enable_{bit}\ OR\ Start$$

شکل این قسمت از مدار نیست در قسمت زیر آمده است:



شکل ۸: قسمت نرمال سازی خروجی

در نهایت نیز زمانی $overflow$ داریم که مقدار شیفتی که می دهیم داخل ۳۲ بیت جا نوشد و تمام $exponent$ ها یک باشد و زمانی زمانی محاسبه به پیان می رسید که یا $Enable_{bit}$ در قسمت قبل فعال شده باشد و یا حالت خاص اتفاق افتاده باشد و ۲۵ امین بیت خروجی ۱ شود. شکل ایت قسمت از مدار و خروجی نهایی مدار نیز در قسمت زیر آورده شده است:



شکل ۹: خروجی نهایی مدار و شرط پایان یافتن آن

۶.۲ مرحله ششم: آزمایش مدار و بررسی درستی عملکرد آن

در این قسمت حالت های مختلف ورودی را به مدار دادیم و خروجی آن را برسی کردیم، نتیجه ۵ تست را نیز در این قسمت آورده ایم:

تست اول) در این قسمت اعداد 3.5 و 2.25 را جمع کردیم که همانطور که میبینید حاصل 5.75 شده است:

$A = 0b01000000011000000000000000000000$
$B = 0b01000000000100000000000000000000$
$ADD/SUB = 0,\ C(out) = 0b01000000101110000000000000000000$

تست دوم) در این قسمت اعداد 3.5 و 2.25 را از هم کم کردیم که همانطور که میبینید حاصل 1.25 شده است:

$A = 0b01000000011000000000000000000000$
$B = 0b01000000000100000000000000000000$
$ADD/SUB = 1,\ C(out) = 0b00111111101000000000000000000000$

تست سوم) در این قسمت هر دو عدد $3.4e38$ داده شده که حاصل جمع آن ها اورفلو می کند و همانطور که میبینید بیت $overflow$ روشن شده است:

$A = 0b01111111011111111100100110011110$
$B = 0b01111111011111111100100110011110$
$ADD/SUB = 0,\ overflow = 1$

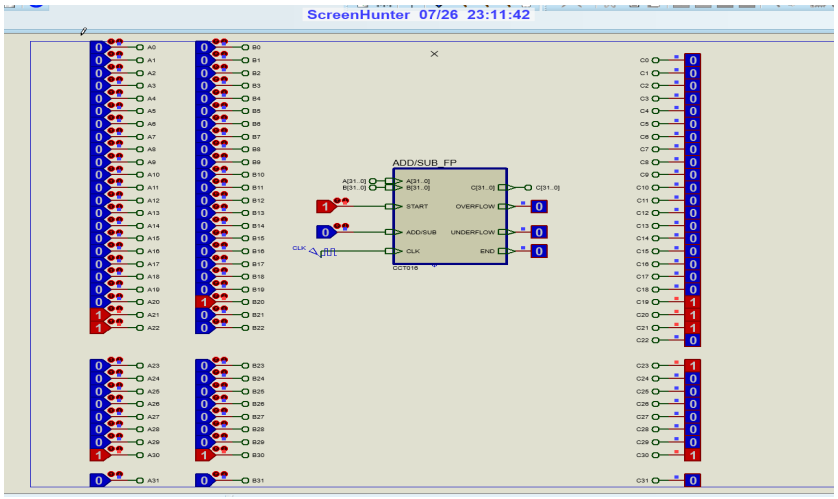
تست چهارم) در این قسمت هر دو عدد $1.0e38$ داده شده که حاصل تفریق آن ها صفر می باشد که همانطور که میبینید خروجی نهایی صفر شده است

$A = 0b01111110100101100111011010011001$
$B = 0b01111110100101100111011010011001$
$ADD/SUB = 1,\ C(out) = 0b00000000000000000000000000000000$

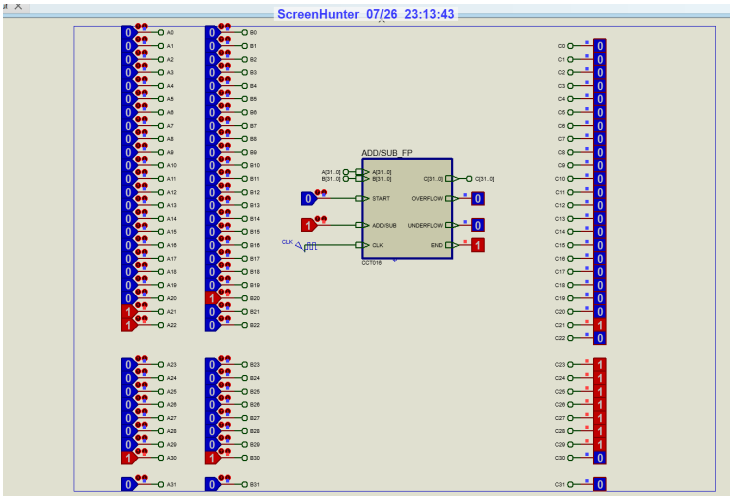
تست پنجم) در این قسمت هر دو عدد $1.4e-45$ و $2.8e-45$ داده شده که حاصل تفریق آن ها حالت $underflow$ اتفاق می افتد که همانطور که میبینید خروجی نهایی نیز، بیت $underflow$ روشن است.

$A = 0b00000000000000000000000000000001$
$B = 0b00000000000000000000000000000010$
$ADD/SUB = 1,\ Underflow = 1$

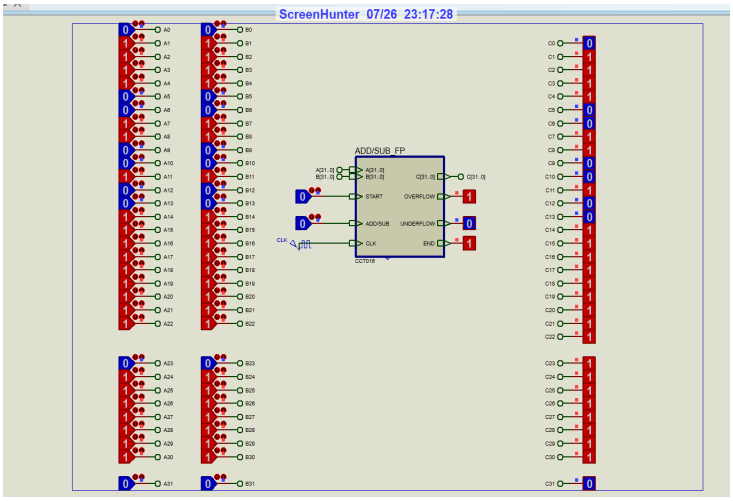
نتایج تست ها به ترتیب آورده شده است:



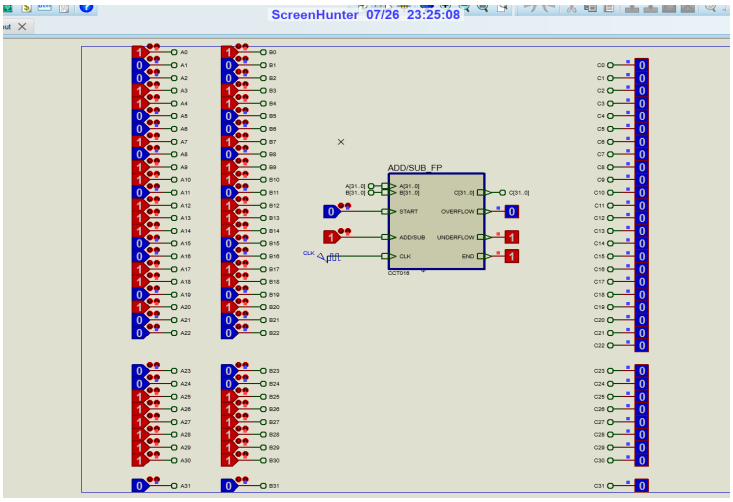
شکل ۱۰: تست اول: جمع دو عدد



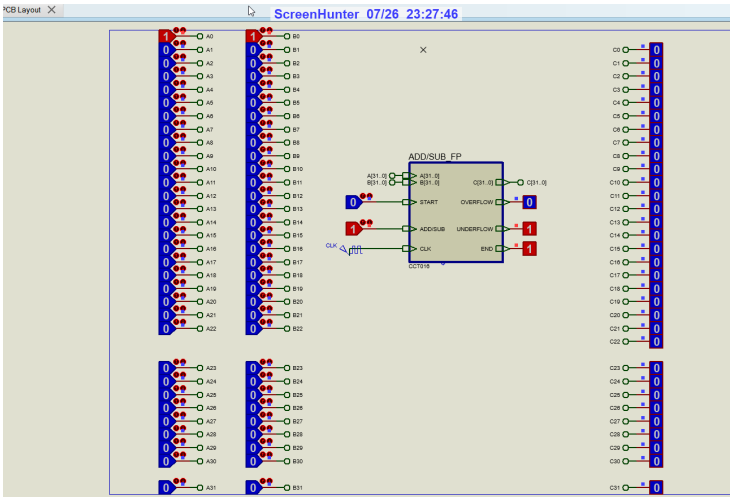
شکل ۱۱: تست دوم: تفریق دو عدد



شکل ۱۲: تست سوم: حالت *Overflow*



شکل ۱۳: تست چارم: حالت *Zero*



شکل ۱۴: تست پنجم: حالت *Underflow*

۳ چالش ها

به طور کلی چالش هایی که در ساخت مدار در پروتئوس وجود داشت شامل دو قسمت بود، یکی حالت های خاص مدار که باید همگی انجام می شدند، برای مثال حالتی که جواب نهایی صفر میشد باید تمامی بیت های *exponent* در انتها صفر نمایش داده می شود. و دیگر چالش ساخت مدار مشکلاتی بود که خود پروتئوس هنگام اجرای مدار داشت و به خاطر نامگذاری و ... اتفاق می افتاد.

۴ نتیجه و بحث

در این آزمایش برای ساخت مدار به شکل اصولی قسمت های مختلف را از هم جدا کردیم و مسیر داده را به چند بخش تقسیم کردیم و توانستیم با خروجی دادن سیگنال ها از واحد کنترل و پیاده سازی اصولی و مرحله به مرحله مدار را به درستی پیاده سازی کنیم، در مدار های نسبتا پیچیده مانند این مدار استفاده از معماری واحد کنترل و مسیر داده و همچنین پیاده سازی مرحله به مرحله و داشتن یک دیاگرام کلی به دقت پیاده سازی و بهینه بودن آن کمک می کند و خطا را کاهش می دهد.