گزارش دستورکار ششم آزمایشگاه معماری کامپیوتر

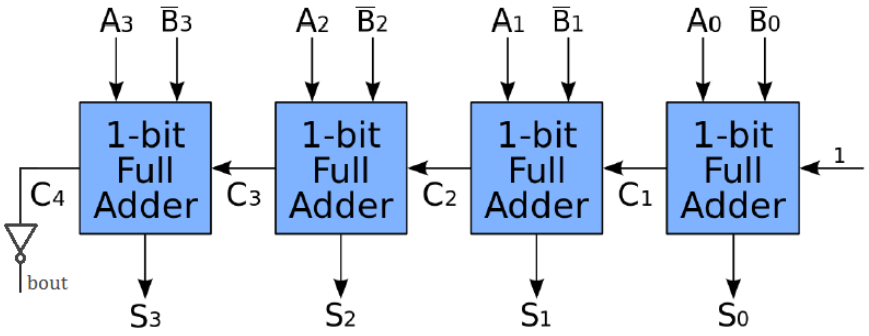
نگار موقتیان، ۹۸۳۱۰۶۲

ماژول complement_subtractor_4bit

در این قسمت از آزمایش می‌خواهیم یک تفریق کننده با استفاده از جمع کنندهٔ آبشاری ۴ بیتی ساخته شده در آزمایش قبل طراحی کنیم. برای این کار ابتدا سعی می‌کنیم رابطهٔ تفریق میان دو عدد را به یک رابطهٔ جمع تبدیل کنیم.

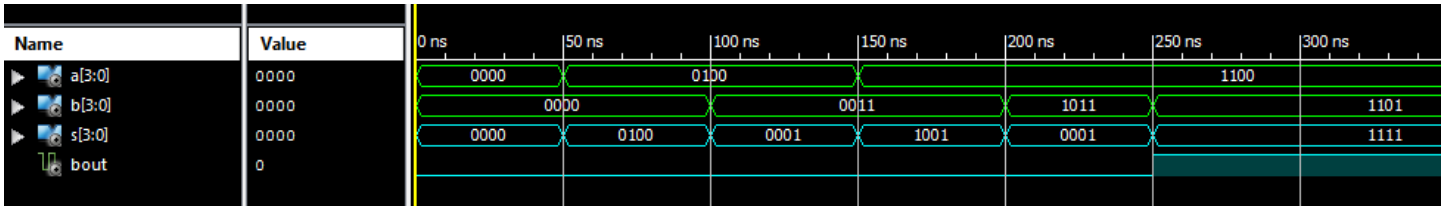
$$A - B \xrightarrow{in\ n\ bits} 2^n + A - B = A + (2^n - B) = A + (B' + 1)$$

بنابراین اگر با استفاده از یک جمع کنندهٔ ۴ بیتی عدد A، مکمل عدد B و عدد ۱ را با یکدیگر جمع کنیم می‌توانیم حاصل A – B را بدست آوریم. مدار این تفریق کنندهٔ ۴ بیتی به شکل زیر خواهد بود.

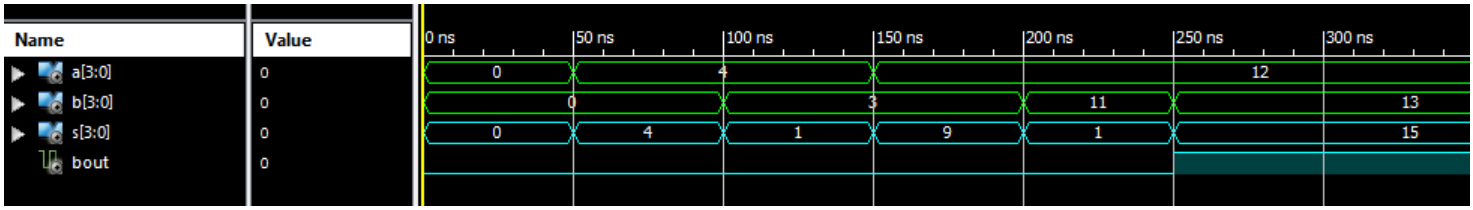


نکته‌ای که باید در این مدار به آن توجه داشت این است که cout مدار جمع کنندهٔ آبشاری حتما باید یک باشد. زیرا طبق روابط بالا یک 2^n به پاسخ مساله اضافه کرده‌ایم که باید خود را در پاسخ حاصل شده نشان دهد. یک بودن cout در مدار جمع کنندهٔ آبشاری معادل این است که در طبقهٔ آخر رقم قرضی (bout) نداشته باشیم. وجود رقم قرضی به این معناست که عدد A از B کوچک‌تر بوده، لذا نتیجهٔ حاصل شده در دنیای اعداد بی‌علامت معتبر نیست.

پس از آن شبیه سازی این مدار توسط test bench نوشته شده و به ازای مقادیر مختلف A و B انجام شد، تا از درستی رفتار مدار ساخته شده اطمینان حاصل شود.



برای سهولت بیش‌تر در بررسی سیگنال‌های ورودی و خروجی تمام اعداد چند بیتی را با فرض بی‌علامت بودن به مبنای ۱۰ می‌بریم. نتیجهٔ تنظیمات گفته شده به صورت زیر است.



همانطور که مشاهده می‌شود در ۲۵۰ نانوثانیهٔ اول شبیه‌سازی خروجی تولید شده به ازای مقادیر A و B معتبر و صحیح است. اما پس از آن عدد A از B کوچک‌تر بوده و سیگنال bout برابر با یک می‌شود. لذا طبق توضیحات داده شده در بالا نتیجهٔ حاصل شده معتبر نیست.

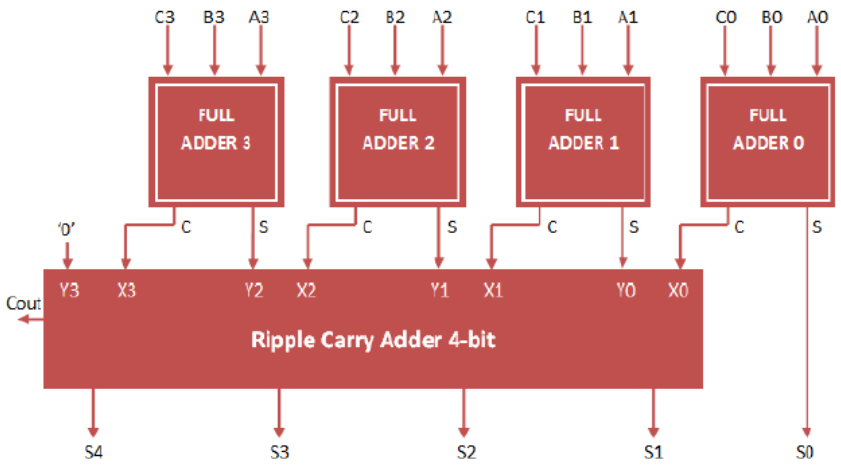
ماژول carry_save_adder_4bit

حال در این قسمت از آزمایش می‌خواهیم یک جمع کنندهٔ ذخیره‌گر نقلی با استفاده از جمع کنندهٔ آبشاری ۴ بیتی ساخته شده در آزمایش قبل طراحی کنیم.

ابتدا باید توجه داشت که می‌توان به دو صورت به یک تمام جمع کننده نگاه کرد. یک دید مانند دیدی است که تا به اینجا داشته‌ایم. یک تمام جمع کننده دو عدد یک رقمی را با یک رقم نقلی جمع می‌زند. اما می‌توان به آن به صورت یک جمع کننده برای ۳ عدد یک رقمی نیز نگاه کرد، رویکردی که در این قسمت از آزمایش از آن استفاده می‌کنیم.

نکتهٔ بعدی این است که تا به این‌جا همواره برای جمع چند عدد رقم نقلی را به گونه‌ای تولید می‌کردیم و به ورودی تمام جمع کنندهٔ طبقهٔ بعد می‌دادیم. در حالی که در این نوع جمع کننده ما این رقم‌های نقلی را ذخیره کرده، در ادامه آن را مانند یک عدد جدید در نظر گرفته و در نهایت آن را با حاصل جمع رقم‌ها به صورت مستقل (در این‌جا با استفاده از یک جمع کنندهٔ آبشاری) جمع می‌زنیم.

مدار این جمع کننده مانند زیر می‌باشد.



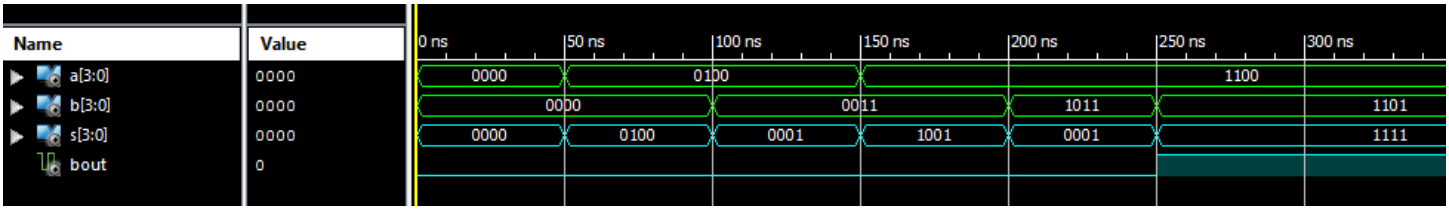
خروجی نهایی از کنار هم گذاشتن نتیجهٔ Cout و S ها بدست می‌آید. زیرا مجموع ۳ عدد ۴ بیتی نهایتاً برابر است با:

$$(1111)_2 + (1111)_2 + (1111)_2 = (101101)_2$$

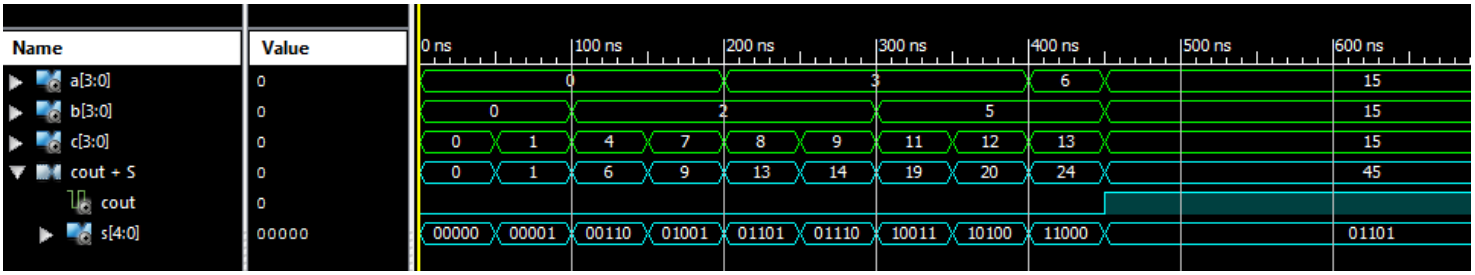
بنابراین برای نشان دادن حاصل جمع این ۳ عدد حداقل نیاز به ۶ رقم داریم.

حال با توجه به این توضیحات این جمع کننده را مطابق مدار داده شده و با استفاده از تمام جمع کننده و جمع کنندهٔ ۴ بیتی آبشاری ساخته شده در آزمایش قبل، به صورت ساختاری پیاده‌سازی می‌کنیم.

در ادامه شبیه سازی این مدار توسط test bench نوشته شده و به ازای مقادیر مختلف A، B و C انجام می‌شود تا از درستی رفتار مدار ساخته شده اطمینان حاصل کنیم.



برای سهولت بیش‌تر در بررسی سیگنال‌های ورودی و خروجی از کنار هم گذاشتن سیگنال‌های خروجی S و cout یک Virtual Bus ساخته و تمام اعداد را با فرض بی‌علامت بودن به مبنای ۱۰ می‌بریم. نتیجهٔ تنظیمات گفته شده به صورت زیر است.



همانطور که در این شکل دیده می‌شود، به ازای مقادیر مختلف مجموع این سه عدد به درستی محاسبه شده است.

ماژول‌های carry_generator_logic و carry_look_ahead_adder_4bit

در این قسمت از آزمایش می‌خواهیم یک جمع کنندهٔ پیش بینی کنندهٔ رقم نقلی با استفاده از تمام جمع کننده‌های ساخته شده در آزمایش قبل طراحی کنیم. انگیزهٔ اصلی برای طراحی این نوع جمع کننده بهبود تاخیر جمع کنندهٔ آبشاری است. همانطور که می‌دانیم در جمع کنندهٔ آبشاری هر طبقه باید منتظر طبقهٔ قبلی بماند و علت اصلی تاخیر، رقم نقلی می‌باشد. حال اگر بتوانیم به طریقی این رقم‌های نقلی را با استفاده از A، B و cin با سرعت بیش‌تری محاسبه کنیم (زیرا این سه ورودی‌اند و در لحظهٔ شروع و بدون هیچ تاخیری آن‌ها را داریم) می‌توانیم جمع کننده‌ای بهینه بسازیم.

برای این کار ابتدا در ماژول carry_genarator_logic این رقم‌های نقلی را تولید کرده‌ایم، به گونه‌ای که اگر در نظر بگیریم:

$P_i = A_i + B_i,\ \ G_i = A_i \cdot B_i$

می‌توان رقم نقلی مربوط به هر طبقه را از روابط زیر محاسبه کرد:

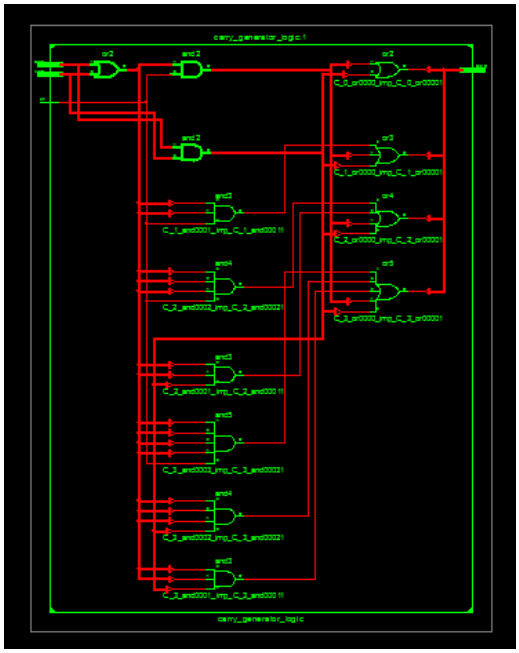
$C_0 = G_0 + (cin \cdot P_0)$

$C_1 = G_1 + (G_0 \cdot P_1) + (cin \cdot P_0 \cdot P_1)$

$C_2 = G_2 + (G_1 \cdot P_2) + (G_0 \cdot P_1 \cdot P_2) + (cin \cdot P_0 \cdot P_1 \cdot P_2)$

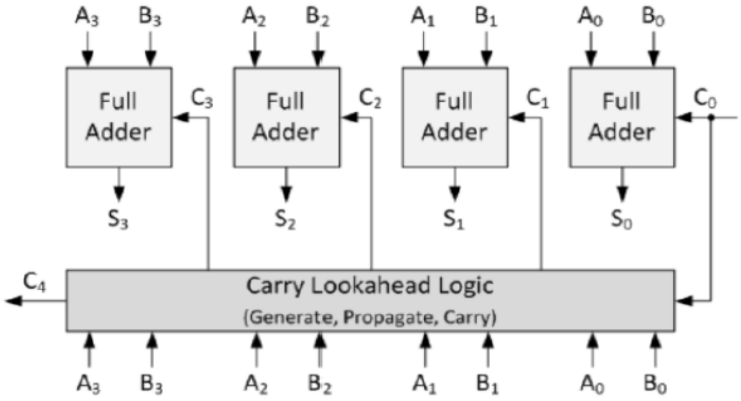
$C_3 = G_3 + (G_2 \cdot P_3) + (G_1 \cdot P_2 \cdot P_3) + (G_0 \cdot P_1 \cdot P_2 \cdot P_3) + (cin \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3)$

طرح شماتیک این مدار که توسط نرم افزار ISE تولید شده در ادامه آمده‌است.

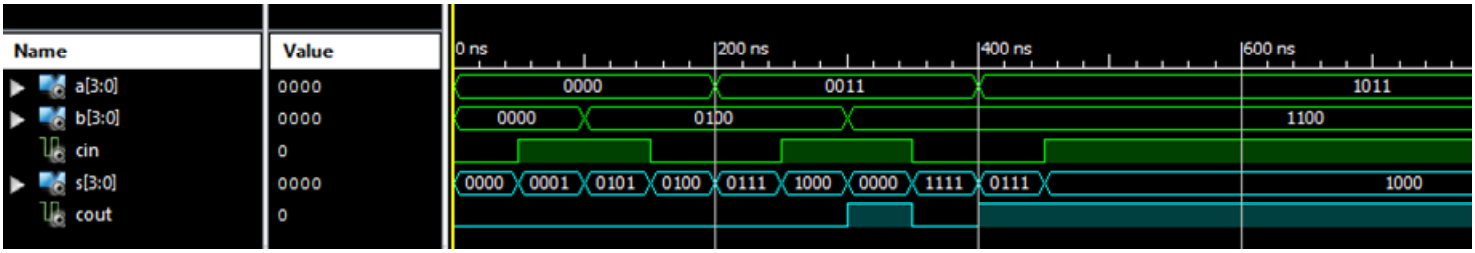


(همانطور که در این طرح شماتیک دیده می‌شود گیت‌هایی با تعداد ورودی بالا داریم. این قضیه ساخت جمع کنندهٔ پیش بینی کنندهٔ رقم نقلی با ورودی‌های بزرگ را عملاً ناممکن می‌سازد)

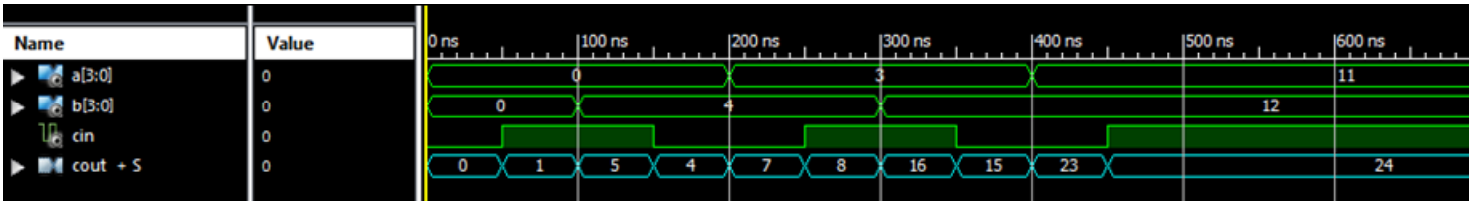
سپس در ماژول carry_look_ahead_adder_4bit این رقم‌های نقلی را مطابق شکل زیر به تمام جمع‌کننده‌ها می‌دهیم تا حاصل نهایی را بدست آوریم.



پس از آن شبیه سازی این مدار توسط test bench نوشته شده و به ازای مقادیر مختلف A، B و cin انجام شد، تا از درستی رفتار مدار ساخته شده اطمینان حاصل شود.



سپس مانند بخش‌های قبل برای سهولت بیش‌تر در بررسی سیگنال‌های ورودی و خروجی از کنار هم گذاشتن سیگنال‌های خروجی S و cout یک Virtual Bus ساخته و تمام اعداد را با فرض بی‌علامت بودن به مبنای ۱۰ می‌بریم. نتیجهٔ تنظیمات گفته شده به صورت زیر است.



همانطور که دیده می‌شود جمع‌ها به درستی انجام شده‌اند.