طرّاحی دیجیتال سیستم‌های کامپیوتری
گزارش تمرین کامپیوتری شماره‌ی ۳

محمّد نوروزی ـ ۸۱۰۱۹۳۴۹۹
نازنین صبری ـ ۸۱۰۱۹۴۳۴۶
نوشین شهیدزاده ـ ۸۱۰۱۹۳۴۳۶

در این فاز از پروژه نوع داده‌ی fixed point را جایگزین نوع داده‌ی real کردیم. برای این کار یک نوع داده‌ی جدید با نام fixed_array تعریف کردیم (خطّ ۱۵ فایل nmn_types.vhd) که به‌جای این‌که آرایه‌ای از متغیّرهایی با نوع real باشد آرایه‌ای از متغیّرهای با نوع std_logic_vector(15 downto 1) هستند. این آرایه در واقع یک عدد fixed point را نمایش می‌دهد که در مجموع دارای ۱۶ بیت است و از این ۱۶ بیت ۹ بیت آن بخش صحیح و ۶ بیت آن بخش کسری را نشان می‌دهد و بیت آخر علامت عدد را مشخّص می‌کند.

برای نگهداری اعداد منفی از رویکرد sign-magnitude استفاده کردیم که در آن در چپ‌ترین بیت علامت عدد نگهداری می‌شود و در ۱۵ بیت باقی‌مانده مقدار عدد. در ادامه به شرح عملیات موجود در MAC می‌پردازیم.

در بخش جمع‌کننده که یکی از componentهای بخش MAC است، با توجّه به بیت علامت هر یک از دو عددی که با هم جمع می‌شوند، عملیات مورد نیاز برای جمع انجام شده است. در فایل adder در صورتی که علامت هر دو عدد با هم برابر باشد سایر بیت‌ها را با هم جمع می‌کنیم و با علامت مشترک در res می‌ریزیم (خطوط ۴۵ تا ۴۷ و ۶۸ تا ۷۰). برای این‌که بتوانیم تشخیص دهیم در صورتی که یکی از اعداد منفی و دیگری مثبت بود کدام‌یک را از دیگری کم کنیم از یک nibble comparator استفاده کرده‌ایم که برای طرّاحی آن، طبق مباحثی که در کلاس مطرح شده بود، از چند component از نوع bit comparator استفاده کردیم که کدهای آن به‌ترتیب در فایل های n_bit_comparatoe و bit_comparator موجود است. حال در صورتی که علامت دو عدد برابر نبود سه حالت را در نظر می‌گیریم، اگر خروجی مقایسه‌کننده‌ای که ورودی‌اش دو عدد ما است (خطوط ۳۸ تا ۴۰) a_eq_b بود، res را برابر با ۱۶ بیت صفر می‌کنیم (خطوط ۵۲ و ۵۳ و همچنین ۶۲ و ۶۳)، اگر خروجی a_gt_b بود a - b را (که طبعاً به‌صورت unsigned جمع می‌شود) در res می‌ریزیم (خطوط ۴۹ تا ۵۱ و همچنین ۵۹ تا ۶۱ و برای حالتی که خروجی a_lt_b بود نیز b - a را در res می‌ریزیم. در همه‌ی حالات در صورت overflow مقدار اضافی را بیرون می‌ریزیم و فقط n (=۱۶) بیت سمت راست باقی می‌ماند. (قطعاً علامت صحیح خواهد بود.)

در بخش ضرب‌کننده که دیگر component بخش MAC است، به این صورت عمل می‌کنیم که برای مشخّص کردن بیت علامت نتیجه، بیت‌های علامت دو عدد ورودی را با هم xor می‌کنیم، که به این معناست که اگر هر دو علامت با هم یکی بود (منفی یا مثبت) نتیجه مثبت می‌شود و در غیر این صورت منفی. برای بخش عددی نتیجه نیز حاصل‌ضربِ unsigned دو عدد را (به‌جز بیت علامت) حساب می‌کنیم و این دو بخش را در نهایت با هم concatenate می‌کنیم تا به نتیجه‌ی نهایی برسیم. (خطوط ۱۴ تا ۲۲ از فایل mult) در واقع خروجی نهایی این تابع از point + 2 -width تا point از همه‌ی بیت‌های حاصل‌ضرب خواهد بود. (خطّ ۲۱ از همان فایل)

در بخش activation function نیز فایل را به این صورت تغییر داده‌ایم که از componentی از نوع nibble comparator استفاده کرده‌ایم که عدد ورودی‌مان را با ۰/۵ (یعنی 0000000000100000) مقایسه می‌کند و در صورتی که نتیجه a_eq_b یا a_gt_b بود محتوای عدد را در خروجی می‌ریزد و در غیر این صورت آن را صفر می‌کند. (خطوط ۴۱ تا ۴۵ فایل activationFunction)