به نام خدا

طراحی کامپیوتری سیستم‌های دیجیتال - پاییز ۱۴۰۴

پروژه ۲: طراحی و شبیه‌سازی یک واحد محاسباتی

بیت سریال از شتاب‌دهنده **Stripes**

طراحان: آرمین قاسمی – حسین غرابادیان

هدف پروژه :

در این تمرین می خواهیم با نحوه پردازش بیت به بیت داده ها آشنا شویم. در بخش اول یک ماژول برای محاسبه ضرب داخلی دو بردار با استفاده از معماری بیت-سریال می‌سازیم و در بخش بعدی با استفاده از ماژول هایی که در بخش قبل ساختیم، یک ضرب بردار در ماتریس را انجام خواهیم داد.

محاسبات بیت-سریال چیست؟

محاسبات بیت-سریال یک روش طراحی سخت‌افزاری است که در آن، داده‌ها به جای پردازش موازی و یک‌باره، به صورت بیت به بیت و در طی چندین سیکل کلاک پردازش می‌شوند. این فرآیند در عمل بسیار شبیه به ضرب‌کننده Add-and-Shift Multipliersاست که در درس مدار منطقی با آن آشنا شده‌اید. در این روش، مدارهای محاسباتی مانند ضرب‌کننده‌ها بسیار ساده و کوچک می‌شوند و اغلب به چند گیت AND و یک جمع‌کننده نیاز دارند، زیرا در هر لحظه فقط با یک بیت از ورودی سروکار دارند. اگرچه محاسبه یک عملیات واحد ممکن است به چند سیکل کلاک نیاز داشته باشد، اما سادگی مدار به ما اجازه می‌دهد تعداد بسیار زیادی از این واحدهای محاسباتی را به صورت موازی در کنار هم قرار دهیم و به توان پردازشی بالایی دست یابیم.

در ادامه برای آشنایی بیشتر از این نوع معماری در ماژول ضرب کننده یک مثال را بررسی می‌کنیم:

فرض کنید می‌خواهیم عدد A = 6(bin:0110) را در عدد B = 5(bin:0101) ضرب کنیم. روش عادی این دو عدد که به آن موازی-موازی می گوییم به شکل زیر است:

```
    0110   (A = 6)
  x 0101   (B = 5)
  -------
    0110         (Partial Product 1: 0110 * 1)
   0000          (Partial Product 2: 0110 * 0)
  0110           (Partial Product 3: 0110 * 1)
 0000            (Partial Product 4: 0110 * 0)
+----------------
 00011110        (Final Result)
```

در این پروژه می‌خواهیم از روش سریال-موازی یا بیت-سریال برای انجام این ضرب استفاده کنیم. در این روش ورودی B به صورت موازی وارد می شود و همواره در دسترس است ولی ورودی A به صورت بیت-سریال وارد می‌شود. یعنی در هر سیکل کلاک، یکی از بیت‌های آن در دسترس قرار می‌گیرد. در Add-and-Shift Multiplier دیدید که روی ورودی سریال بیت ها را از LSB به MSB پردازش میکردیم، ولی در این پروژه بیت های ورودی سریال از MSB به LSB وارد می شوند و ما هم به همین ترتیب آنها را پردازش میکنیم. عملیات ضربی که بررسی کردیم با روش بیت-سریال به شکل زیر انجام می‌شود:

- این عملیات ضرب به یک Register انباشتگر (Accumulator) هشت بیتی نیاز دارد. این Register در ابتدای کار به مقدار صفر reset می شود.

سیکل ۱

- **بیت ورودی از A:** اولین و پرارزش‌ترین بیت 0110 (MSB) که برابر 0 است.
- **عملیات :**چون بیت ورودی 0است، عدد B با Accumulator جمع نمی‌شود (معادل جمع با صفر است). دقت کنید که در MSB برای اینکه نتیجه علامت درستی داشته باشد باید حاصل را از Accumulator کم کنیم. در واقع اگر بیت MSB عدد A برابر با ۱ بود باید B- را با Accumulator جمع میکردیم.
- **محاسبه:**
  - Accumulator در ابتدا : 00000000
  - نتیجه : 00000000 = 0 + 00000000

سیکل ۲

- **بیت ورودی از A:** ۱
- **عملیات:**

۱. ابتدا مقدار قبلی Accumulator را یک واحد به چپ شیفت می‌دهیم.

۲. چون بیت ورودی 1است، مقدار B را با نتیجه شیفت‌یافته جمع می‌کنیم.

- **محاسبه:**

۱. مقدار قبلیAccumulator : 00000000

۲. ۱ .شیفت به چپ : 00000000 = 1 << 00000000

۳. ۲ .جمع با B : 00000101 = 00000000 + 00000101

سیکل ۳

- **بیت ورودی از A:** ۱
- **عملیات :**دوباره ابتدا شیفت و سپس جمع با B .
- **محاسبه:**
  - مقدار قبلیAccumulator : 00000101
  - ۱ .شیفت به چپ : 00001010 = 1 << 00000101
  - ۲ .جمع با B :00001111 = 00000101 + 00001010

سیکل ۴

- **بیت ورودی از A:** چهارمین و کم‌ارزش‌ترین بیت 0110 (LSB) که برابر 0 است.
- **عملیات :**چون بیت ورودی 0است، فقط عمل شیفت را انجام می‌دهیم.
- **محاسبه:**
  - مقدار قبلیAccumulator :00001111
  - ۱ .شیفت به چپ : 00011110 = 1 << 00001111
  - ۲ .جمع با صفر : 00011110 = 0 + 00011110
  - Accumulator در پایان سیکل ۴ : 00011110

با توجه به این مثال، می توانیم با استفاده از این الگوریتم ضرب دو عدد ۴ بیتی در یکدیگر را در ۴ سیکل انجام دهیم. نکته مورد توجه اینجاست که عرض بیت ورودی موازی(یا همان ورودی B) تاثیری در مدت زمان انجام محاسبات ندارد، در حالی که عرض بیت ورودی سریال (همان ورودی A) مستقیما مدت زمان انجام محاسبات را تعیین می‌کند. این ویژگی سبب می‌شود که این اگوریتم قدرت انعطاف بالایی داشته باشد و بتواند مدت زمان محاسبات را بر اساس طول ورودی تنظیم کند.

فرض کنید در یک واحد سخت‌افزاری برای انجام محاسبات شبکه های عصبی، در یک لایه نیاز به دقت ۸ بیتی داشته باشیم و خروجی نرون های ما ۸ بیتی باشند، ولی در لایه دیگر نیاز به ۱۶ بیت دقت داشته باشیم. الگوریتم مطرح شده می‌تواند هر دو مورد را هندل کند و همینطور برای ورودی ۸ بیتی محاسبات را در مدت زمان کوتاه تری انجام دهد.

به بیشترین میزان دقت (یا همان عرض بیت) ممکن برای ورودی A ، MAX_PRECISION یا به اختصار MP میگوییم. برای مثال اگر MP برای یک ضرب کننده ۱۶ باشد، یعنی ضرب کننده می تواند عدد B (که عرض آن ثابت است) را در یک عدد نهایتا ۱۶ بیتی ضرب کند و مدت زمان انجام محاسبات به اندازه تعداد بیت های عدد دوم خواهد بود.

طراحی واحد پردازشی **Stripes** :

در این بخش از پروژه قصد داریم یک ماژول برای محاسبه ضرب داخلی دو بردار در یکدیگر با استفاده از معماری بیت-سریال طراحی کنیم. فرض کنید می خواهیم بردار های A و B را در هم ضرب داخلی کنیم. B یک بردار با N درایه W بیتی، و A یک بردار با N درایه نهایتا MP بیتی است. البته در این مثال فرض میکنیم درایه های A و B هر دو ۱۶ بیتی هستند. یعنی MP = W = 16 . همچنین فرض میکنیم N = 4 است.

این محاسبات به شکل زیر انجام می‌شوند:

$$\vec{A} = \begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \end{bmatrix} \vec{B} = \begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \end{bmatrix} \; ; \; \vec{A} \cdot \vec{B} = a_0 \times b_0 + \ldots + a_{N-1} \times b_{N-1} = result$$

ماژول مورد نظر یکی از بردار های A یا B را به صورت بیت به بیت و سریالی، و بردار دیگر را به صورت موازی و یکجا دریافت می‌کند. در ادامه فرض می‌کنیم که بردار A به صورت سریالی و بیت به بیت دریافت می‌شود. یعنی در هر سیکل ما یک بیت از هر درایه بردار A را دریافت می‌کنیم. همچنین فرض میکنیم بردار B در طول محاسبات در

دسترس است. نکته مهم این است که هر درایه بردار A از msb به lsb وارد میشود. بنابراین می توانیم بگوییم ماژول مورد نظر یک ورودی N بیتی i_vec_a_bits دارد که در سیکل j ام حاوی بیت MP-j ام a_0 تا a_{N-1} است. در مثال ما که MP = 16 است، درسیکل اول بیت ۱۵ ام (msb) در سیکل بعدی بیت ۱۴ ام و... در سیکل۱۶ ام بیت صفرم از هر درایه دریافت می‌شود. به بیان دیگر در سیکل j ام داریم :

$$\text{i\_vec\_bits} = [a_0(MP-j), \dots, a_{N-1}(MP-j)]$$

همچنین این ماژول سیگنال های کنترلی i_valid، i_is_lsb و i_is_msb را نیز دارد. سیگنال i_is_msb در سیکل اول محاسبات فعال می‌شود و نشان دهنده این است که ورودی i_vec_a_bits حاوی بیت های msb اعداد a_0 تا a_{N-1} است. سیگنال i_is_lsb نیز در سیکل آخر محاسبات فعال می‌شود و نشان دهنده این است که ورودی i_vec_a_bits حاوی بیت های lsb اعداد a_0 تا a_{N-1} است. سیگنال i_valid نیز نشان دهنده این است که ورودی i_vec_a_bits در چه سیکل هایی معتبر است.

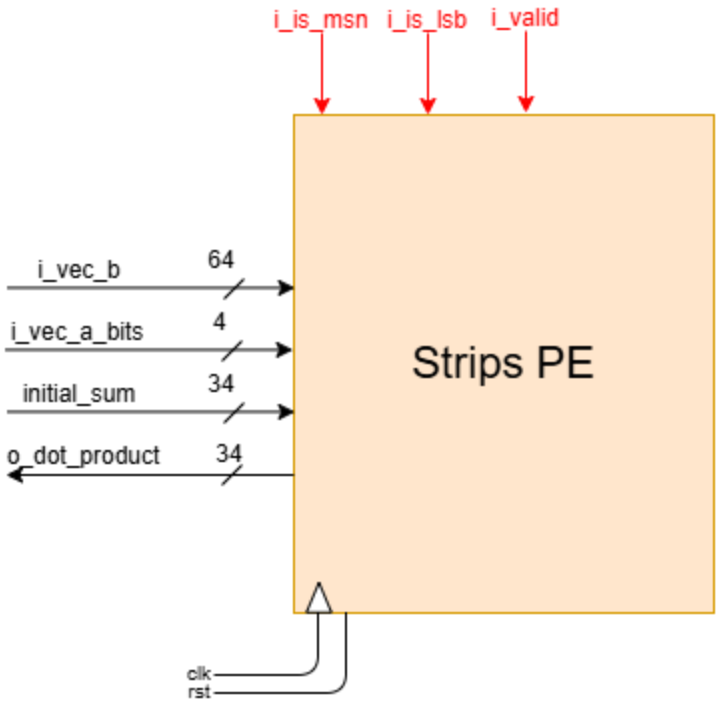
ورودی دیگر مدار i_vec_b است که W×N بیت دارد و شامل تک تک درایه های بردار B است. در مثال ما این ورودی 4×16=64 بیت دارد که در آن ۱۶ بیت اول مربوط بهb_0 ، ۱۶ بیتی بعدی مربوط بهb_1 و... است. فرض ما این است که این ورودی در طول زمان معتبر است و تغییر نمیکند.

یک ورودی دیگر مدار ورودی i_initial_sum است این ورودی صرفا در سیکل آخر که i_is_lsb فعال است معتبر است و با خروجی نهایی حاصل ضرب داخلی جمع میشود. تعداد بیت های این سیگنال به اندازه بیت های سیگنال خروجی مدار است.

همچنین خروجی نهایی مدار سیگنال o_dot_product است که $W + \log_2 N + MP$ بیت دارد. در مثال ما این سیگنال ۳۴ بیتی است.

دقت کنید که این ماژول هیچ سیگنال کنترلی یا وضعیتی مانند start و done ندارد و همچنین مقادیر بردار B را رجیستر نمی کند.

شمای کلی این ماژول در شکل۱ آورده شده.



شکل ۱ واحد پردازشی Stripes

نهایتا ضرب داخلی بردار های زیر را انجام دهید و در گزارش خود بیاورید.

$$\begin{pmatrix}1\\3\\6\\2\end{pmatrix}\cdot\begin{pmatrix}3\\5\\5\\1\end{pmatrix}=50$$

$$\begin{pmatrix}1\\-4\\6\\3\end{pmatrix}\cdot\begin{pmatrix}2\\6\\-1\\7\end{pmatrix}=-7$$

انجام یک ضرب بردار در ماتریس با استفاده از واحد های پردازشی **Stripes** :

در این بخش می خواهیم ضرب یک ماتریس ۸ در ۸ در یک بردار ۸ در ۱ را با استفاده از PE طراحی شده در بخش قبل انجام دهیم:

$$\begin{pmatrix}b_{11} & \cdots & b_{18}\\ \vdots & \ddots & \vdots\\ b_{81} & \cdots & b_{88}\end{pmatrix}_{8\times8}\times\begin{pmatrix}a_1\\ \vdots\\ a_8\end{pmatrix}_{8\times1}=\begin{pmatrix}o_1\\ \vdots\\ o_8\end{pmatrix}_{8\times1}$$

واضح است که درایه o_i از بردار خروجی از حاصل ضرب داخلی بردار a با ترانهاده سطر iام ماتریس B بدست آمده:

$$(b_{i1}\quad \dots\quad b_{i8})^T\cdot\begin{pmatrix}a_1\\ \vdots\\ a_8\end{pmatrix}=\begin{pmatrix}b_1\\ \vdots\\ b_8\end{pmatrix}\cdot\begin{pmatrix}a_1\\ \vdots\\ a_8\end{pmatrix}=o_i$$

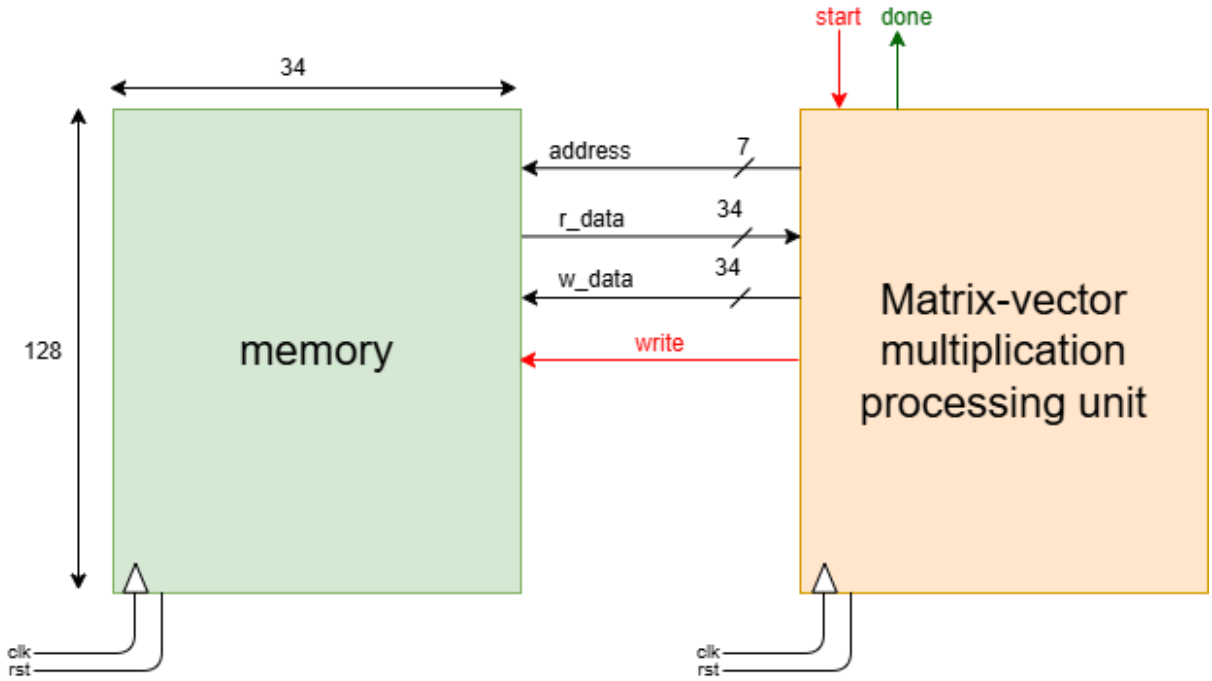
بنابراین اگر به ازای هر سطر از ماتریس B یک داشته PE باشیم که قابلیت ضرب داخلی دو بردار ۸ عضوی را داشته باشد، می توانیم به صورت موازی تمام خروجی ها را پردازش کنیم اما در این بخش می خواهیم این ضرب را با تنها دو PE انجام دهیم.

محدودیت سخت افزاری دیگری که داریم این است که هر PE ما می تواند یک ضرب داخلی برداری با نهایتا چهار المان انجام دهد. یعنی پارامتر N در PE باید برابر با ۴ باشد. بنابراین باید هر ضرب داخلی برداری ۸ در ۸ را به دو ضرب داخلی ۴ در ۴ بشکنیم:

$$(b_{i1} \quad \dots \quad b_{i8})^T \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_8 \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_8 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_8 \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_4 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_4 \end{pmatrix} + \begin{pmatrix} b_5 \\ \vdots \\ b_8 \end{pmatrix} \cdot \begin{pmatrix} a_5 \\ \vdots \\ a_8 \end{pmatrix} = o_i$$

بنابراین کنترلر این ماژول باید بتوان ضرب یک ماتریس ۸در۸ در یک بردار ۸در۱ را به ضرب داخلی بردار های ۴در۱ بشکند و نتایج آن‌ها را با هم ادغام کند.

شماتیک نهایی ماژول باید مانند شکل۲ باشد.



شکل ۲واحد پردازشی ضرب ماتریس در بردار

واحد حافظه یک ماژول ساده است که حاوی مقادیر ماتریس B و بردار a است. نتیجه نهایی نیز روی همین واحد حافظه نوشته میشود. با توجه به ابعاد ماتریس و بردار این واحد حافظه می تواند ۱۲۸ کلمه را در خود جا دهد. عرض این کلمات ۳۴ بیت است نتایج خروجی نیز در همین حافظه جا شوند. واحد Matrix-vector multiplication processing unit به این شکل کار میکند که پس از دریافت سیگنال شروع به کار می‌کند، حاصل ضرب ماتریس در بردار را محاسبه میکند و نهایتا بردار خروجی را در حافظه می‌نویسد. فرض کنید ۸ کلمه اول حافظه مربوط به بردارB، ۶۴ کلمه بعدی مربوط به ماتریسA، و ۸ کلمه بعدی مربوط به بردار خروجی باشد که باید

توسط Matrix-vector multiplication processing unit نوشته شود. فرض کنید ورودی ها که ۱۶ بیتی هستند، sign extend شده اند تا در ۳۴ بیت در حافظه قرار بگیرند.
برای این بخش به صورت جداگانه تست کیس در اختیار شما قرار می‌گیرد.

مواردی که باید تحویل دهید:

- طراحی کنترلر و مسیر داده به صورت دستی (برای همه ی ماژول ها)
- کدهای مربوط به زبان وریلاگ (تمام ماژول ها)
- خروجی تست کیس ها

سایر نکات

- انجام این تمرین به صورت گروه های دونفره خواهد بود.
- فایل ها و گزارش خود را تا قبل از موعد تحویل، با نام CAD_HW2_<SID1>_<SID2>.zip در محل مربوطه در صفحه درس آپلود کنید.
- در صورت هرگونه ابهام می توانید از طریق ایمیل یا تلگرام با طراحان در ارتباط باشید.
- نام گذاری صحیح متغیرها، تمیزی کد و توضیحات و پارامتری بودن ورودی‌های ماژول‌ها می تواند تا حدودی کاستی‌های کد را در بخش‌های دیگر جبران کند.
- - هدف این تمرین یادگیری شماست! در صورت کشف تقلب، مطابق با قوانین درس برخورد خواهد شد.

موفق باشید