به نام خدا

طراحی کامپیوتری سیستم‌های دیجیتال - پاییز ۱۴۰۳

پروژه ۲: **طراحی و پیاده سازی بافر ورودی/خروجی**

طراحان: محمد امانلو- مبینا حقی‌زاده

# مقدمه

هدف این تمرین آشنایی با مدیریت داده های ورودی/خروجی به/از یک ماژول سخت افزاری است. در این تمرین شما با یک رویکرد مناسب برای این کار آشنا شده و آن را پیاده سازی خواهید کرد.
توجه کنید که این تمرین جزئی از یک پروژه نهایی است که در ادامه با تمرین های بعدی کامل خواهد شد.

در ابتدا، شبکه‌های عصبی پیچشی و یک معماری سخت‌افزاری جهت پیاده‌سازی آن‌ها به طور خلاصه معرفی می‌شوند. این معماری شامل آرایه‌ای از واحدهای پردازشی است، که شما در این تمرین بخشی از یک واحد پردازشی را پیاده سازی خواهید کرد. بخش مربوطه شامل پیاده سازی یک بافر چرخشی و کنترل داده‌های ورودی/خروجی به/از واحد پردازشی با handshake مناسب است.

### شبکه‌های عصبی پیچشی (Convolutional Neural Networks – CNN)

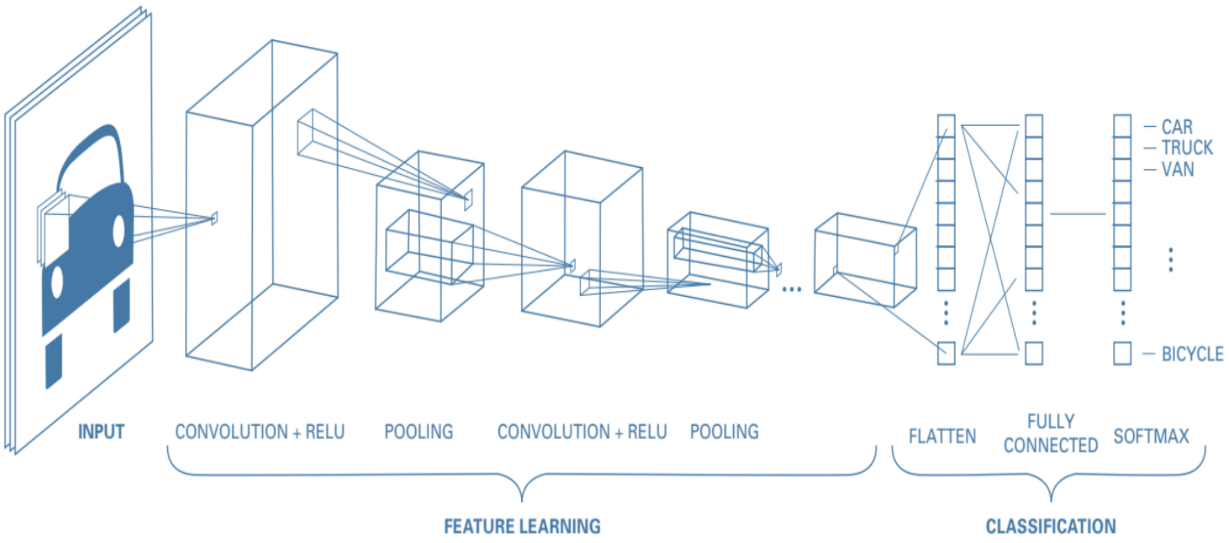
شبکه‌های عصبی پیچشی (CNN) نوعی معماری از شبکه‌های عصبی عمیق است که به‌طور خاص برای پردازش داده‌های تصویری و داده‌های با ساختار فضایی استفاده می‌شود. CNN با بهره‌گیری از ایده پیچش (convolution)، به‌جای پردازش کل تصویر به‌صورت یکباره، به‌صورت محلی و با حرکت دادن فیلترها بر روی قسمت‌های مختلف تصویر، ویژگی‌های ورودی را استخراج می‌کند. این ساختار به مدل اجازه می‌دهد تا الگوهای متنوعی را در سطوح مختلف تصویر، از ویژگی‌های ابتدایی مانند لبه‌ها تا جزئیات پیچیده‌تر، شناسایی کند.

### عملکرد CNN

مدل‌های CNN از رویکرد لایه‌به‌لایه برای تحلیل تصاویر استفاده می‌کنند. این لایه‌ها به‌صورت سلسله‌مراتبی ویژگی‌های تصویر را تحلیل کرده و با پیشرفت به سمت لایه‌های عمیق‌تر، ویژگی‌های

پیچیده‌تر و انتزاعی‌تری را شناسایی می‌کنند. در ابتدا، لایه‌های اولیه به شناسایی الگوهای ساده مثل لبه‌ها و زوایا می‌پردازند؛ سپس لایه‌های بعدی ترکیبی از این ویژگی‌ها را می‌سازند تا بتوانند اشکال، لبه‌ها و اشیاء را تشخیص دهند.

این فرآیند به شکلی انجام می‌شود که مدل بتواند به‌طور خودکار ویژگی‌های مهم و غیرضروری را تشخیص داده و ویژگی‌های مهم را حفظ کند، در حالی که ابعاد داده‌ها کاهش می‌یابد. به این ترتیب، مدل می‌تواند داده‌های بزرگ مثل تصاویر با وضوح بالا را با کاهش تعداد پارامترها و محاسبات مورد نیاز، به‌طور مؤثر پردازش کند.




شکل ۱. ساختار کلی یک شبکه‌ی پیچشی

## شتابدهنده سخت افزاری

شتاب‌دهنده‌های سخت‌افزاری شبکه‌های عصبی ماژول های سخت افزاری هستند که برای افزایش کارایی و سرعت محاسبات شبکه عصبی طراحی شده‌اند. این شتاب‌دهنده‌ها محاسبات سنگینی را که شبکه‌های عصبی نیاز دارند، به صورت موازی و با سرعت بالا انجام می‌دهند. با تسریع در پردازش الگوریتم‌های یادگیری عمیق، این دستگاه‌ها زمان پاسخگویی و مصرف انرژی را به طور قابل توجهی

کاهش می‌دهند، که این امکان را فراهم می‌کند تا برنامه‌های هوش مصنوعی با سرعت هرچه بیشتر و به طور بی‌درنگ اجرا شوند. ادغام آن‌ها در سیستم‌های هوش مصنوعی برای وظایفی از قبیل پردازش زبان طبیعی تا تشخیص تصویر پیچیده حیاتی است، که آن‌ها را به اجزای بنیادی در پیشرفت فناوری هوش مصنوعی تبدیل می‌کند. شتابدهنده‌ها به‌گونه‌ای طراحی می‌شوند که در دستگاه‌هایی مانند موبایل، یا دستگاه‌های با منابع محدودتر مانند FPGAها که محدودیت‌های منابع و انرژی دارند، مدل‌ها را کارآمد اجرا کنند. در ادامه معماری *Eyeriss* به عنوان یکی از شتابدهنده‌های مطرح در حوزه شبکه‌های عصبی معرفی خواهد شد.

## معماری Eyeriss

شتابدهنده سخت افزاری *Eyeriss* یک معماری برای پیاده‌سازی شبکه‌های عصبی پیچشی است که هدف اصلی آن کاهش مصرف انرژی و افزایش کارایی پردازش شبکه‌های عصبی است. این معماری از تعدادی واحدپردازشی یا **Processing Elements (PE)** متعدد برای انجام عملیات ریاضی (MACهای مختلف) و از بافرها و Scratch Padها برای ذخیره‌سازی داده‌ها در حین پردازش استفاده می‌کند. واحد پردازشی شامل قسمت‌های مختلفی هستند که با هم تعامل می‌کنند تا پردازش داده‌ها به طور موثر انجام شود. برای این منظور، بافرها داده‌های ورودی را مدیریت می‌کنند، Scratch Padها داده‌های مورد نیاز را ذخیره می‌کنند، و کنترل‌کننده‌ها هماهنگی بین اجزا را برقرار می‌کنند.

#### ارتباط CNN با Eyeriss:

CNNها به‌طور گسترده در کاربردهای پردازش تصویر، تشخیص اشیا، و بینایی ماشین استفاده می‌شوند و به دلیل ماهیت محاسبات ماتریسی و پیچشی، به توان محاسباتی بالایی نیاز دارند. Eyeriss به‌عنوان یک شتاب‌دهنده سخت‌افزاری، به‌گونه‌ای طراحی شده است که این عملیات‌ها را به صورت موازی و تا حد امکان بهینه انجام دهد تا مصرف انرژی را کاهش دهد و عملکرد را بهبود بخشد. این معماری قادر است عملیات ماتریسی و پیچشی را به‌طور موازی پردازش کند، که این ویژگی برای CNNها بسیار مفید است چون نیاز به محاسبات ماتریسی زیادی دارند. علاوه بر این، داده‌های مورد نیاز برای پردازش CNN را به‌صورت محلی در حافظه‌های کوچک ذخیره می‌کند تا نیاز به جابجایی داده‌ها از حافظه اصلی را کاهش دهد.

# شرح پروژه

با توجه به فایل پیوست، برای فاز اول پروژه، پیاده‌سازی بافرهای پروژه *Eyeriss* در نظر گرفته شده‌اند. بافرها نقش مهمی در مدیریت داده‌ها و پردازش‌های همزمان دارند و شامل بخش‌هایی برای نوشتن، خواندن و مدیریت وضعیت پر یا خالی بودن هستند. بافرها شامل کنترل‌کننده‌هایی نیز هستند تا تعاملات داده به درستی و بدون از دست رفتگی انجام شود.

در این فاز شما بافرها، ساختار و منطق مربوط به بافرهای **FIFO** و **Circular FIFO** را پیاده‌سازی کرده و عملکرد آن‌ها را بررسی خواهید کرد. این بافرها از طریق سیگنال‌های ورودی و خروجی با دیگر اجزای سیستم ارتباط برقرار می‌کنند و بخش مهمی از سیستم پردازش داده هستند. در ادامه، به تشریح عملکرد  این بافرها و نحوه پیاده‌سازی آن‌ها با جزئیات بیشتر می‌پردازیم.

### 1. بافر معمولی (Buffer)

هسته اصلی بافر چرخشی‌ای که در نهایت پیاده‌سازی خواهید کرد، یک بافر ابتدایی است. این بافر یک نوع حافظه است که برای نگهداری موقت داده‌ها در طول پردازش طراحی شده است. هدف اصلی این بافر، فراهم کردن فضایی است که داده‌ها بتوانند به‌طور همزمان نوشته و خوانده شوند. این کار باعث پردازش بی وقفه داده‌ها می‌شود، به طوری که نیاز به توقف به دلیل نبود داده‌ها یا پر بودن حافظه نیست.

#### نحوه پیاده‌سازی:

بافر معمولی به گونه‌ای طراحی می‌شود که قابلیت خواندن و نوشتن همزمان را داشته باشد. این بافرها پارامترهایی دارند که به ما اجازه می‌دهد عرض داده، عمق (تعداد محل‌های ذخیره‌سازی) و تعداد داده‌های قابل خواندن یا نوشتن به‌صورت همزمان را پیکربندی کنیم.

#### ساختار حافظه‌ای (Memory Array):

بافر از یک آرایه حافظه‌ای تشکیل شده است که هر عنصر آن می‌تواند یک داده را در خود نگه دارد. هر خانه از این حافظه با یک آدرس خاص شناسایی می‌شود.

### عملیات نوشتن:

نوشتن داده‌ها در لبه بالارونده سیگنال کلاک (clk) انجام می‌شود. هنگامی که سیگنال نوشتن (wen) فعال باشد، داده‌های ورودی (din) در آدرس مشخص شده (waddr) در بافر نوشته می‌شوند. این بافرها همچنین از نوشتن همزمان چندین داده در مکان‌های مختلف پشتیبانی می‌کنند.

### عملیات خواندن:

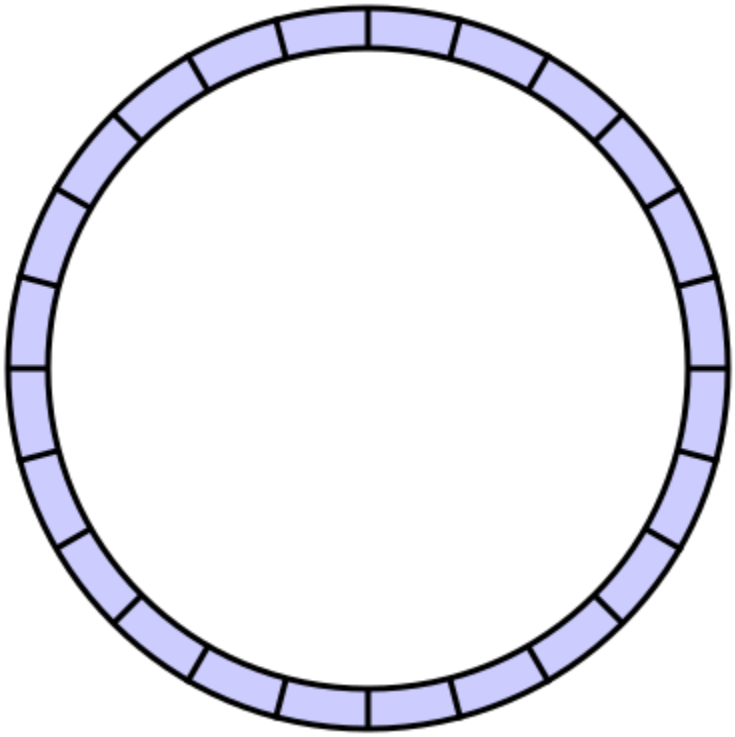
خواندن داده وابسته به سیگنال کلاک نبوده و به صورت ترکیبی انجام می‌شود؛ یعنی با استفاده از آدرس مشخص شده (raddr) داده‌های ذخیره‌شده در بافر به خروجی (dout) منتقل می‌شوند. بافرها امکان خواندن همزمان چندین داده از مکان‌های مختلف را نیز دارند.

در این بافر، دو پارامتر مهم نیز وجود دارند:

- پارامتر PAR_WRITE به صورتی تعیین می‌شود که مشخص کند چند کلمه داده به صورت همزمان، در آدرس‌های پشت سر هم، می‌تواند در بافر نوشته شود.
- پارامتر PAR_READ به صورتی تعیین می‌شود که مشخص کند چند کلمه داده به صورت همزمان، از آدرس‌های پشت سر هم، می‌توانند از بافر خوانده شوند.

### 2. بافر دایره‌ای (Circular FIFO Buffer)

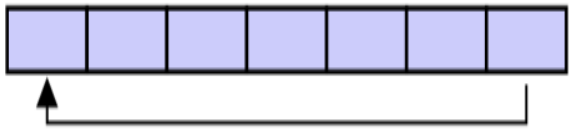
بافر دایره‌ای یک نوع بافر است که برای ذخیره موقت داده‌ها طراحی شده است. این بافر به‌گونه‌ای عمل می‌کند که وقتی فضای آن پر می‌شود، داده‌های جدید از ابتدای بافر در آن ذخیره می‌شوند. از آنجا که داده‌ها به‌صورت چرخه‌ای در آن قرار می‌گیرند، به این نوع بافر "دایره‌ای" می‌گویند.



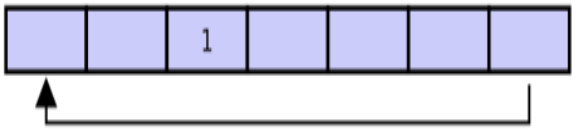
شکل ۲. بافر دایره‌ای

یک حلقه، به‌طور مفهومی یک بافر دایره‌ای را نشان می‌دهد. شکل ۲ به‌صورت بصری نشان می‌دهد که بافر انتهای مشخصی ندارد و می‌تواند درون خود بچرخد. با این حال، چون حافظه هیچ‌گاه به‌طور فیزیکی به‌شکل یک حلقه ساخته نمی‌شود، از نمایش خطی استفاده می‌شود که در ادامه آمده است.

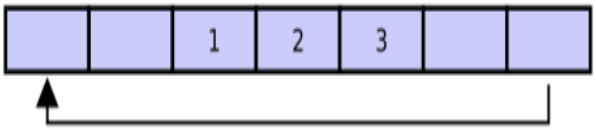
یک بافر دایره‌ای در ابتدا خالی است و طول مشخصی دارد. در شکل زیر، یک بافر ۷ عنصری نشان داده شده است:



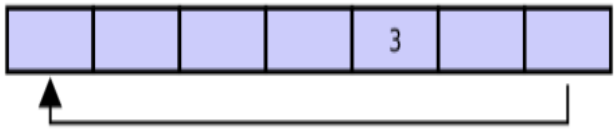
فرض کنید عدد ۱ در مرکز بافر دایره‌ای نوشته شده است (محل شروع دقیق در یک بافر دایره‌ای مهم نیست):



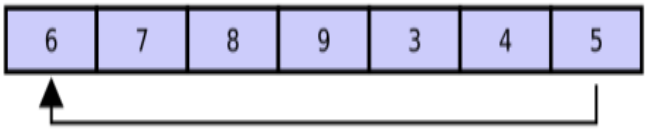

سپس فرض کنید دو عنصر دیگر – ۲ و ۳ – به بافر دایره‌ای اضافه می‌شوند که بعد از ۱ قرار می‌گیرند:



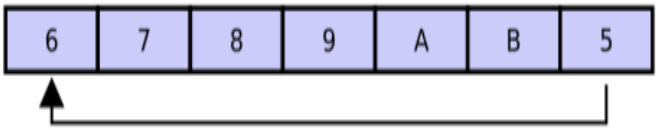

اگر دو عنصر حذف شوند، دو مقدار قدیمی‌تر داخل بافر دایره‌ای حذف خواهند شد. بافرهای دایره‌ای از منطق FIFO (اولین ورودی، اولین خروجی) استفاده می‌کنند. در این مثال، ۱ و ۲ اولین عناصری بودند که وارد بافر دایره‌ای شدند و اولین عناصری خواهند بود که حذف می‌شوند، و ۳ را در بافر باقی می‌گذارند.



اگر بافر ۷ عنصر داشته باشد، به طور کامل پر خواهد شد:

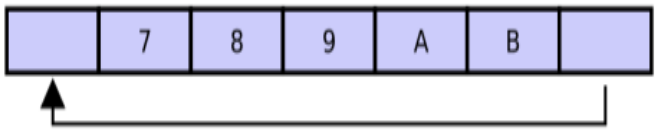



یکی از ویژگی‌های بافر دایره‌ای این است که وقتی پر می‌شود و یک نوشتن بعدی انجام می‌شود، شروع به جایگزینی داده‌های قدیمی‌تر می‌کند. در مثال کنونی، دو عنصر دیگر – A و B – اضافه می‌شوند و آنها جای ۳ و ۴ را می‌گیرند:




به‌طور جایگزین، توابعی که بافر را مدیریت می‌کنند می‌توانند از جایگزینی داده‌ها جلوگیری کرده و یک خطا برگردانند یا استثنا (exception) ایجاد کنند. این‌که آیا داده‌ها جایگزین شوند یا نه، به منطق توابع بافر یا برنامه‌ای که از بافر دایره‌ای استفاده می‌کند، بستگی دارد. در این پروژه، **داده‌های خوانده شده**، جایگزین خواهند شد.

در نهایت، اگر اکنون دو عنصر حذف شوند، آنچه که حذف می‌شود دیگر ۳ و ۴ (یا به عبارت بهتر اکنون A و B) نخواهد بود، بلکه ۵ و ۶ حذف می‌شوند، زیرا ۵ و ۶ اکنون قدیمی‌ترین عناصر هستند، و نتیجه مطابق زیر خواهد بود.



### نحوه پیاده‌سازی:

برای پیاده‌سازی بافر دایره‌ای، ابتدا یک بافر معمولی (مثل آرایه) ایجاد می‌شود و از دو نشانگر (Pointer) برای مدیریت خواندن و نوشتن در آن استفاده می‌شود:

- **نشانگر نوشتن (Write Pointer)**: مکان فعلی برای نوشتن داده‌های جدید.
- **نشانگر خواندن (Read Pointer)**: مکان فعلی برای خواندن داده‌ها.

زمانی که شمارنده نوشتن از شمارنده خواندن یک آدرس عقب‌تر باشد، بافر پر شده و سیگنال full فعال می‌شود؛ یعنی در این حالت، دیگر نمی‌توان داده‌های جدیدی را وارد کرد. همچنین، زمانی که تمام داده‌ها خوانده می‌شوند، سیگنال empty فعال خواهد شد (شمارنده خواندن و نوشتن در یک آدرس باشند).

## مدیریت وضعیت پر و خالی بودن بافر دایره‌ای

همانند بافر معمولی، در این ماژول نیز دو پارامتر مهم وجود دارند:

- پارامتر PAR_WRITE به صورتی تعیین می‌شود که مشخص کند چند کلمه داده به صورت همزمان می‌تواند در بافر نوشته شود. به عبارت دیگر، اگر ما بخواهیم چند کلمه داده را به‌طور همزمان به بافر اضافه کنیم، باید مطمئن شویم که بافر ظرفیت کافی برای این کار دارد.
- پارامتر PAR_READ به صورتی تعیین می‌شود که مشخص کند چند کلمه داده به صورت همزمان می‌توانند از بافر خوانده شوند. به عبارت دیگر، اگر بخواهیم چند کلمه داده را به طور همزمان بخوانیم، باید مطمئن شویم که بافر حداقل به آن اندازه ظرفیت دارد.

### پر بودن بافر (full):

در ماژول بافر دایره‌ای FIFO، زمانی که تمامی خانه‌های حافظه بافر پر باشند، سیگنالی به نام full فعال می‌شود. این سیگنال مانع از نوشتن داده‌های جدید می‌شود تا زمانی که بخشی از داده‌های موجود خوانده شده و فضا آزاد شود.

دقت کنید که در صورتیکه بافر به اندازه تعداد درخواست write، جای خالی نداشته باشد، هیچکدام از داده‌ها نمی‌توانند در بافر ذخیره شوند. برای مثال اگر درخواست نوشتن ۴ داده به صورت همزمان داشته باشیم و در بافر تنها ۳ جای خالی داشته باشیم، هیچکدام از آن ۴ داده ذخیره نخواهند شد و باید تا زمانی که بافر، معادل تعدادشان جای خالی داشته باشد، منتظر بمانند.

برای اینکه بفهمیم آیا بافر پر شده است یا نه، باید تمامی عناصر بافر را بررسی کنیم. به‌طور خاص، ما بررسی می‌کنیم که آیا نوشتن k+1 کلمه داده جدید باعث می‌شود که شمارنده آدرس نوشتن (write_addr_cnt) به شمارنده آدرس خواندن به علاوه یک (read_addr_cnt) برسد یا خیر (در واقع، شمارنده نوشتن باید به اندازه k+1 آدرس از شمارنده خواندن عقب‌تر باشد).

چرا k+1؟: چون ما می‌خواهیم بدانیم آیا با افزودن یک کلمه جدید، بافر به حالت پر می‌رسد یا نه. اگر شمارنده آدرس نوشتن به حداکثر ظرفیت خود برسد، باید دوباره از ابتدای بافر شروع کنیم و بررسی کنیم که آیا می‌توانیم داده جدیدی بنویسیم یا خیر.

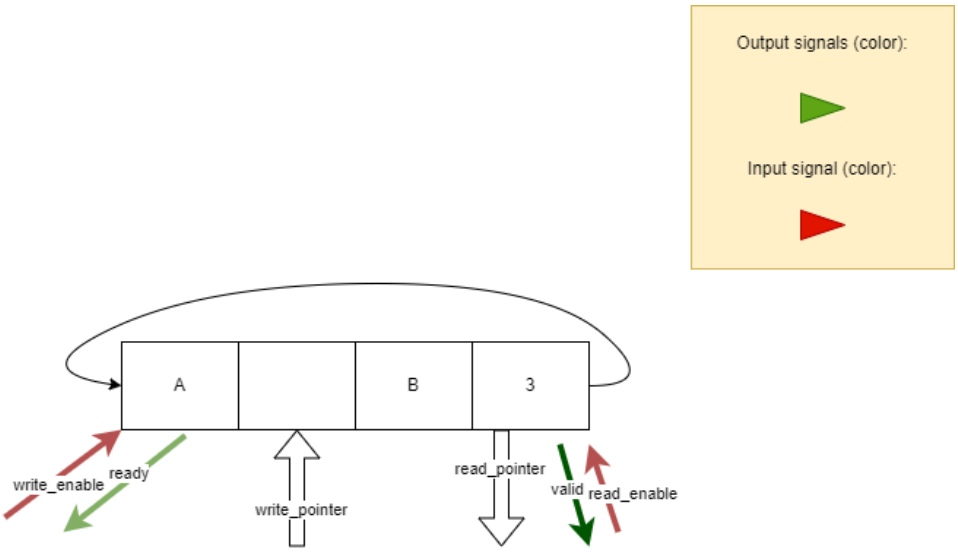
اگر شرط عقب‌تر بودن شمارنده نوشتن به اندازه k+1 آدرس از شمارنده خواندن، به ازای یکی از kهای از صفر تا PAR_WRITE برقرار باشد، به این معنی است که بافر پر شده است و سیگنال full به حالت فعال (1) تغییر می‌کند. در این حالت، دیگر نمی‌توانیم داده‌های جدیدی را به بافر اضافه کنیم مگر اینکه داده‌های قبلی را بخوانیم و از بافر خارج کنیم.

خالی بودن بافر (empty):

سیگنال empty نشان می‌دهد که بافر خالی است و هیچ داده‌ای برای خواندن وجود ندارد. این سیگنال زمانی فعال می‌شود که آدرس‌های خواندن و نوشتن برابر باشند. این شرط نیز همانند سیگنال full، به ازای برابر بودن آدرس‌ها در یکی از kها باید برقرار باشد.

- Handshake: ماژول buffer شامل سیگنال های read_en، empty، full و write_en است. به منظور نوشتن در بافر نیاز به برقراری ارتباط با *سیگنال های write_en که ماژول بیرونی برای بافر* ارسال می کند و *سیگنال ready که بافر برای ماژولی* که می خواهد write کند ارسال می کند. ماژولی که از بافر مقدار میخواهد بر اساس معتبر بودن داده داخل بافر خواندن را انجام می دهد به همین منظور مقدار read_en باید به valid بودن کنترلر بافر ست بشود که نشان

دهنده ی درستی داده موجود در بافر خواهد بود. شکل ۳، نشان دهنده ی سیگنال های مربوط به handshake است.




شکل ۳. نحوه ی handshake بافر

این مکانیزم‌ها اطمینان حاصل می‌کند که بافر به‌درستی وضعیت‌های پر و خالی خود را مدیریت می‌کند و عملیات‌های خواندن و نوشتن موازی به‌صورت بهینه و بدون بروز خطاهای سرریز (overflow) یا کم‌ریزی (underflow) انجام می‌شوند. به این ترتیب، بافر قادر است به‌طور همزمان داده‌ها را بخواند و بنویسد و از بازنویسی داده‌های خوانده‌نشده جلوگیری کند.

## مراحل عملکرد بافر دایره‌ای

### نوشتن داده‌ها:

وقتی سیگنال write_en فعال می‌شود، داده‌های ورودی در آدرس مشخص شده با waddr ذخیره می‌شوند. برای بافر دو سیگنال کنترلی valid و ready تعریف می شود که به ترتیب نشان دهنده ی این هست که داده می تواند در کامپوننت های بعدی مورد استفاده قرار گیرد یا خیر و اینکه آیا بافر امکان نوشتن داده ی جدید را دارد. بنابراین نیاز هست که این سیگنال های کنترلی به صورت خروجی بر اساس

وضعیت پر یا خالی بودن بافر و وضعیت write_en به درستی مقدار دهی شوند تا در آینده بقیه کامپوننت ها بتوانند مقدار درست از بافر بخوانند.

### خواندن داده‌ها:

وقتی سیگنال read_en فعال باشد، داده‌های ذخیره‌شده از آدرس مشخص شده با read_addr به خروجی منتقل می‌شوند. ماژولی که میخواهد از بافر بخواند، منتظر قرار گرفتن داده معتبر در بافر است، و همین که داده معتبر وارد آن شد، بافر وارد حالتی میشود که امکان خواندن داده از آن وجود دارد. این ماژول با اولین داده ای که میخواند در یک کلاک بعد، یک سیگنال valid باید به عنوان خروجی تولید کند.

### پشتیبانی از نوشتن و خواندن همزمان:

پارامترهای PAR_WRITE و PAR_READ میزان همزمانی عملیات نوشتن و خواندن را کنترل می‌کنند. PAR_WRITE تعداد کلمات داده‌ای را که می‌توانند به‌طور همزمان در یک سیکل کلاک نوشته شوند، مشخص می‌کند. به همین ترتیب، PAR_READ تعیین می‌کند که چندین کلمه داده به‌طور همزمان خوانده شوند. این ویژگی به بافر این امکان را می‌دهد که حجم بالایی از داده‌ها را به‌صورت کارآمد تبادل کند.

- توجه کنید که پارامترهای PAR_WRITE و PAR_READ، برای بافر معمولی نیز باید تنظیم شوند؛
- می‌توانید برای پیاده‌سازی بخش خواندن و نوشتن موازی در بافر معمولی از generate و for در وریلاگ استفاده کنید؛
- حداکثر عمق بافر را ۸ در نظر بگیرید. همچنین مقادیر خواندن و نوشتن موازی می‌توانند ۱، ۲ و ۴ باشند.

## سایر نکات

- - انجام این تمرین به صورت گروه های دونفره در دو فاز خواهد بود:

1. در فاز اول `controller` و `datapath` را طراحی کرده و در موعد تعیین شده برای فاز اول داخل سایت بارگذاری کنید.

2. در فاز دوم `controller` و `datapath` طراحی شده در فاز اول را در برنامه Modelsim و با زبان verilog پیاده سازی کرده و در موعد معین برای فاز دوم در داخل سایت بارگذاری کنید.

- برای فاز دوم تمرین لازم است فایل های `Testbench` و `HDL` خود را مطابق توضیح داده شده در `trunk` در `subdirectory` های `trunk/doc` بارگذاری کنید. همچنین اطمینان حاصل کنید که با اجرای `trunk/sim/sim_top.tcl` تست پنج شما اجرا می شود. برای اجرای این اسکریپت می توانید از دستور زیر در Modelsim استفاده کنید:

>> do <sim_file>

- فایل ها و گزارش خود را تا قبل از موعد تحویل هر فاز، با نام CAD_HW2_P1_<SID>.zip و CAD_HW2_P2_<SID>.zip به ترتیب در محل های مربوطه در صفحه درس آپلود کنید.

- برای آزمودن کد خودتان در این تمرین تست بنچ‌های مربوطه را خود شما طراحی و پیاده سازی می کنید، اما با توجه به اینکه تمارین بعدی درس مبتنی بر ادامه دادن این تمرین هستند حتما در طراحی و پیاده‌سازی خود به قابلیت مقاومت در برابر تغییر و پارامتری بودن ورودی‌ها و خروجی‌ها توجه لازم را داشته 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