به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش‌کار و مستندات پروژه کارشناسی

پیاده‌سازی سخت‌افزاری الگوریتم شبکه عصبی برای تشخیص تصویر مجموعه‌داده MNIST بر روی FPGA

علی شایان پور

۸۱۰۱۹۸۵۳۲

استاد راهنما

دکتر زین‌العابدین نوابی

استاد داور

دکتر بیژن علیزاده

بهار ۱۴۰۲

فهرست

۱- چکیده

امروزه با توجه به کاربردهای فراوان هوش مصنوعی، استفاده از این الگوریتم‌ها و به طور خاص شبکه‌های عصبی بیش از پیش افزایش یافته است. از طرفی پیاده‌سازی سخت‌افزاری این الگوریتم‌ها گام مهمی در تحقق استفاده‌های هر چه بیشتر از آنهاست. فرآیند پردازش و اجرای الگوریتم شبکه های عصبی با توجه به پیچیدگی‌ها و تعداد لایه‌ها، نیازمند انجام محاسبات زیاد و در نتیجه زمانبر است. هدف از این پروژه افزایش سرعت انجام محاسبات می‌باشد. برای این منظور پیاده سازی الگوریتم شبکه عصبی به منظور تشخیص اعداد دست نویس انتخاب شده است. برای آموزش و تست شبکه، مجموعه‌داده MNIST استفاده شده است. برای رسیدن به دقت خروجی مدنظر یک شبکه عصبی کانولوشن با ۲ لایه کانولوشن و ۱ لایه fully connected انتخاب شده است.

لایه‌های مختلف شبکه عصبی در حالت‌های زیر پیاده‌سازی و زمان اجرای آن‌ها اندازه‌گیری شده است:

- نرم‌افزاری در پردازنده PC با مشخصات Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
- نرم افزاری در قسمت PS[1] برد PYNQ-Z2
- سخت‌افزاری در قسمت PL[2] برد PYNQ-Z2 که کنترل آن با استفاده از PS انجام می‌شود.

نتایج بدست آمده نشان می‌دهد که انجام محاسبات زیاد در لایه های مختلف شبکه به صورت سخت‌افزاری نسبت به حالت پیاده‌سازی نرم‌افزاری سرعت بیشتری دارد. با پیاده‌سازی شبکه کانولوشن مدنظر به صورت سخت‌افزاری، به اندازه حدود ۷۶ برابر سرعت نسبت به حالتی که در PC تنها به صورت نرم‌افزاری اجرا می‌شد افزایش سرعت و در صورتی که صرفا لایه‌های Convolution را به صورت سخت‌افزاری پیاده‌سازی کنیم، در PC به اندازه ۴۰ برابر بهبود سرعت خواهیم داشت.

[1] Processing System

[2] Programmable Logic

۲- مقدمه

امروزه با توسعه فراگیر و سریع هوش مصنوعی و پیچیدگی مدل‌ها، کتابخانه‌ها و روش‌های زیادی برای ماژولار کردن و ساده‌سازی عملیات‌های هوش مصنوعی به وجود آمده است. این ساده‌سازی‌ها در زبان‌های سطح بالا مانند پایتون در نهایت منجر به از دست دادن سرعت اجرا در محاسبات می‌شود. هر چند کتابخانه‌ها با انجام موازی‌سازی‌هایی در سطح سیستم، باعث تسریع این فرآیند شده‌اند اما این سرعت باز هم برای مدل‌های بسیار پیچیده و مخصوصا برای پردازش‌های Real Time بسیار پایین است. راهکارهایی که برای این موضوع وجود دارد استفاده حداکثری از پردازش‌های موازی و GPU است اما استفاده از این سخت‌افزارها بسیار هزینه‌بر است.

در این پروژه به پیاده‌سازی سخت‌افزاری شبکه عصبی MLP[3] و CNN[4] پرداخته‌ایم. زمانبندی لایه‌های مختلف محاسبه شده و با یک‌دیگر مقایسه شده است. در این پروژه از پروتکل‌های عمومی و استاندارد استفاده شده است تا پیاده‌سازی آن جامع و ساده باشد. همچنین به دلیل flexibility زیادی که سخت‌افزار دارد می‌توان با اضافه کردن یا کم کردن واحدهای محاسباتی، به سرعت بیشتری (به بهای هزینه بیشتر) رسید. سعی بر آن است که تمامی پیاده‌سازی‌ها در این پروژه با استفاده از حداقل سخت‌افزار بوده که باعث هزینه کمتر و مصرف توان کمتر می‌شود. از طرفی با وجود کمترین واحدهای محاسباتی همچنان سرعت به طرز چشمگیری بیشتر از نرم‌افزار است.

[3] Multilayer perceptron

[4] Convolutional neural network

۳- شرح پروژه

در این بخش به شرح فعالیت‌ها و روند کار می‌پردازیم. همچنین روش‌های انجام آن به صورت کامل شرح داده می‌شود. بخش‌ها به ترتیب روند اجرایی است و نتیجه‌های این بخش‌ها در بخش ۴ (نتایج) موجود است.

۳.۱- شبکه عصبی MLP برای تشخیص اعداد دست‌نوشته

۳.۱.۱- مقدمه‌ای بر شبکه MLP

شبکه‌های عصبی چند لایه[5] یکی از انواع رایج شبکه‌های عصبی است و برای بسیاری از وظایف مورد استفاده قرار می‌گیرد. این شبکه‌ها با توجه به برخی ویژگی‌هایی که دارند، برای مجموعه وسیعی از مسائل قابل استفاده هستند. در زیر به برخی از دلایل استفاده از MLP در کاربردهای شبکه‌های عصبی اشاره خواهیم کرد:

۱- قدرت تقریبی: MLP قدرت تقریبی بالایی دارد، به این معنی که می‌تواند توابع پیچیده را با دقت بالا تقریب بزند. با استفاده از لایه‌های مخفی[6] و با تعداد مناسب نورون‌ها، MLP می‌تواند تقریبی دقیق از توابع غیرخطی را توسط مجموعه‌ای از توابع خطی ارائه دهد.

۲- آموزش قابل تعمیم: MLP معمولاً با استفاده از الگوریتم Backpropagation و روش‌های بهینه‌سازی، قابلیت آموزش را دارد. با این روش، شبکه می‌تواند از رویکرد نمونه‌های آموزشی تعمیم بیاموزد و قادر به پیش‌بینی برای نمونه‌های جدید باشد.

۳- قابلیت استفاده در وظایف تشخیص الگو: MLP به خوبی برای وظایف تشخیص الگو مناسب است. با ترکیب چندین لایه در MLP، شبکه قادر است ویژگی‌های پیچیده‌تر را از داده‌ها استخراج کند و الگوهای پنهان را شناسایی کند.

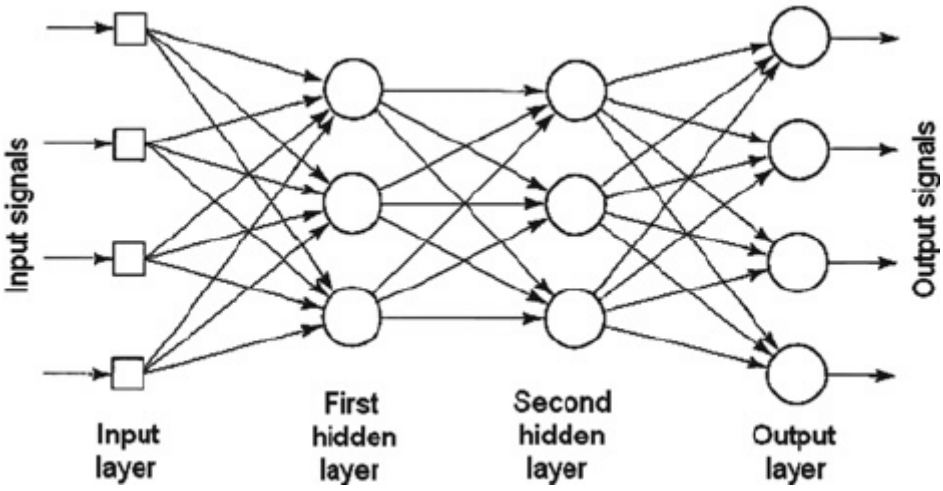
۴- پیچیدگی قابل تنظیم: تعداد لایه‌ها و تعداد نورون‌ها در هر لایه قابل تنظیم است، به این معنی که با تغییر تعداد لایه‌ها و نورون‌ها می‌توان انعطاف پذیری بیشتری در شبکه‌های عصبی داشت و آن‌ها را با دقت و توانایی مناسب برای وظایف مورد نیاز پیکربندی کرد.

۵- قابلیت استفاده در وظایف غیرخطی: MLP قادر است توابع غیرخطی را تقریب بزند. این امکان به MLP می‌دهد که در وظایفی که روابط پیچیده‌تری وجود دارد، مفید باشد.

در شکل ۳.۱.۱.۱ یک نمونه شبکه MLP با دو لایه میانی که هر کدام ۳ لایه مخفی دارند را مشاهده می‌کنیم.

[5] MLP

[6] Hidden layers



شکل ۳.۱.۱.۱- نمونه یک شبکه MLP با دو لایه میانی

۳.۱.۲- پیاده‌سازی

ابتدا برای مدل و آموزش شبکه به سراغ ساده‌ترین معماری برای تشخیص تصویر می‌رویم. این معماری شامل دو لایه میانی است. برای انتخاب اینکه از چند hidden unit استفاده کنیم، باید بر اساس دقتی که به دست می‌آید عمل کنیم. برای همین کار از تعداد لایه‌های کم به لایه‌های زیاد با استفاده از زبان پایتون، کدی می‌نویسیم که ابتدا با استفاده از یک لایه شبکه را آموزش داده و دقت را حساب کند. نتایج در یک فایل csv ذخیره می‌شود؛ پس از این کار تعداد لایه‌ها را یکی زیاد کرده و این کار را از اول تکرار کند. این کار تا زمانی که دقت بیشتر از ۹۸ درصد بشود ادامه پیدا می‌کند.

ورودی شبکه یک عکس ۲۸ در ۲۸ بوده و خروجی ۱۰ عدد که هر کدام احتمال اینکه عدد ورودی از ۰ تا ۹ باشد را مشخص می‌کند.

در نهایت بر اساس نتایج قسمت ۴.۱ انتخاب می‌شود که از ۱۰۰ تا hidden unit استفاده شود تا هم دقت خوبی داشته باشیم و هم Resource بیش از حد استفاده نکنیم.

پس از انتخاب تعداد لایه، مدل train شده شبکه را در یک فایل ذخیره می‌کنیم. سپس به بررسی و تست این شبکه روی داده‌های واقعی و چیزی به جز مجموعه‌داده MNIST می‌پردازیم. این مدل دقت خوبی در تشخیص اعداد نداشته و مجبور به تغییر مدل خود می‌شویم.

۳.۲- شبکه عصبی CNN برای تشخیص اعداد دست‌نوشته

۳.۲.۱- مقدمه‌ای بر شبکه عصبی CNN

شبکه‌های عصبی CNN برای پردازش داده‌های دارای ساختار شبکه‌ای، مانند تصاویر و سیگنال‌های صوتی، استفاده می‌شوند. این شبکه‌ها به خاطر ویژگی‌های خاص خود، در بسیاری از وظایف بینایی ماشین و پردازش

تصویر عملکرد بسیار خوبی دارند. در زیر به برخی از مزایا، معایب و موارد استفاده از شبکه‌های عصبی CNN خواهیم پرداخت:

مزایا:

- استخراج ویژگی‌های سلسله مراتبی: شبکه‌های عصبی CNN، با استفاده از لایه‌های Convolution و لایه‌های Pooling، توانایی استخراج ویژگی‌های سلسله مراتبی از داده‌ها را دارند. این به معنی آن است که شبکه‌ها می‌توانند ویژگی‌های ساده‌تر مانند خطوط و لبه‌ها را در لایه‌های اولیه شناسایی کرده و به ویژگی‌های پیچیده‌تر مانند الگوها و اشیاء در لایه‌های بالاتر بپردازند.
- اشتراک پارامترها: یکی از ویژگی‌های مهم CNN، اشتراک پارامترها است. این به معنی آن است که وزن‌ها و پارامترهای استفاده شده در یک لایه Convolution برای تمام نقاط داده مشترک هستند. این ویژگی باعث می‌شود که تعداد قابل تنظیم پارامترها در CNN به صورت قابل توجهی کاهش یابد و در نتیجه مدل قابلیت یادگیری و تعمیم بیشتری داشته باشد.
- کاهش تعداد پارامترها: استفاده از لایه‌های Pooling در CNN به کاهش تعداد پارامترها و حجم داده ورودی کمک می‌کند. این موضوع به افزایش سرعت آموزش و پیش‌بینی مدل کمک می‌کند.

معایب:

- نیاز به داده آموزش بزرگ: برای آموزش شبکه‌های عصبی CNN، معمولا به مجموعه‌داده‌های بزرگی نیاز داریم که ممکن است در برخی برنامه‌ها محدودیت‌هایی ایجاد کند.
- پیچیدگی محاسباتی: شبکه‌های عصبی CNN، معمولاً پیچیدگی محاسباتی بالایی دارند، به ویژه در موارد استفاده از شبکه‌های عصبی عمیق. آموزش و استفاده از این شبکه‌ها ممکن است زمان و منابع محاسباتی زیادی را مصرف کند.

موارد استفاده:

- تشخیص الگو: به خاطر قابلیت استخراج ویژگی‌های تصویری، در وظایف تشخیص الگو مانند تشخیص چهره‌ها، تشخیص اشیاء و تشخیص افراد استفاده می‌شوند.
- دسته‌بندی تصاویر: برای دسته‌بندی تصاویر در مواردی مانند تشخیص شیء در تصاویر، تشخیص بیماری‌ها از تصاویر پزشکی و تشخیص اشیاء در تصاویر مورد استفاده قرار می‌گیرند.
- ترجمه ماشینی: در وظایف ترجمه ماشینی بر روی متن‌ها و جملات مورد استفاده قرار می‌گیرند.
- تشخیص سیگنال‌های صوتی: برای تشخیص الگوها و ویژگی‌های صوتی در مواردی مانند تشخیص سیگنال‌های صوتی و تشخیص سیگنال‌های زبانی استفاده می‌شوند.

۳.۲.۲- پیاده‌سازی

با توجه به ویژگی‌های مذکور در بخش قبلی بهتر است که برای تشخیص تصویر از مدل CNN استفاده کنیم. برای این کار یک شبکه عصبی با لایه‌ها و مشخصات زیر train می‌کنیم:

۱- لایه Convolution 2D + تابع فعال‌سازی ReLU (ورودی ۲۸ در ۲۸؛ کرنل ۳ در ۳؛ ۳۲ کانال)

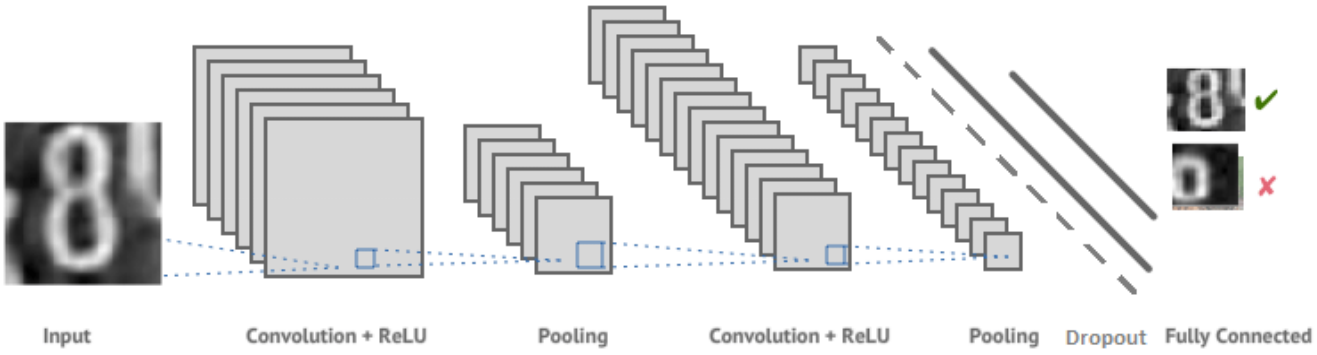
۲- لایه Max Pooling 2D (ورودی ۲۶ در ۲۶؛ ۳۲ کانال)

۳- لایه Convolution 2D + تابع فعال‌سازی ReLU (ورودی ۱۳ در ۱۳؛ کرنل ۳ در ۳؛ ۶۴ کانال)

۴- لایه Max Pooling 2D (ورودی ۱۱ در ۱۱؛ ۶۴ کانال)

۵- لایه Dense یا Fully Connected + تابع فعال‌سازی Softmax (ورودی ۱۶۰۰؛ خروجی ۱۰)

شکل لایه‌ها در شکل ۳.۲.۲.۱ قابل مشاهده است.



شکل ۳.۲.۲.۱- شبکه عصبی CNN طراحی شده برای تشخیص اعداد دست‌نوشته از روی تصویر

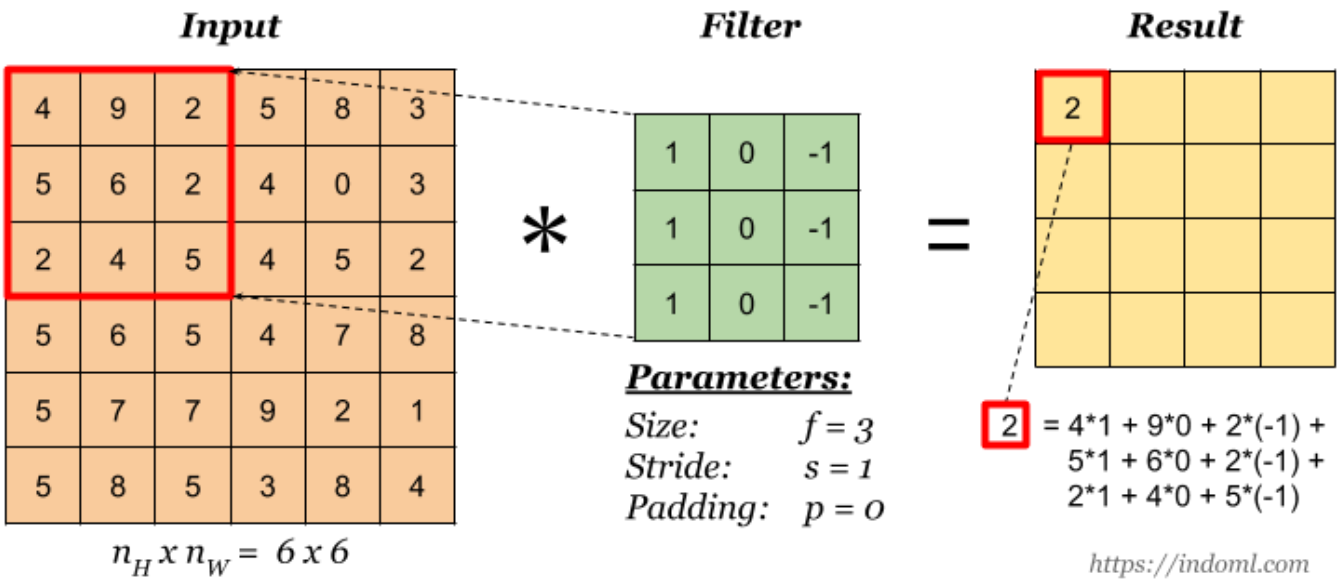
پس از پیاده‌سازی این مدل و train کردن این شبکه عصبی و ذخیره این مدل به عنوان فایل برای استفاده‌های بعدی، باید تست‌های خارج از مجموعه‌داده MNIST را (مانند بخش قبل) انجام دهیم؛ این‌بار (برعکس بخش قبل) دقت بسیار خوبی گرفته و می‌توانیم راه را با همین مدل ادامه دهیم (نتایج در بخش ۴.۴).

۳.۳- استخراج ضرایب و پیاده‌سازی شبکه CNN بدون استفاده از کتابخانه

با توجه به اینکه در پیاده‌سازی سخت‌افزاری به کتابخانه‌های هوش مصنوعی دسترسی نداشته و تنها می‌توانیم از ضرب‌کننده، جمع‌کننده و بقیه عملیات‌های پایه استفاده کنیم، نیاز است تا تمامی لایه‌ها بدون استفاده از کتابخانه و تنها با استفاده از حلقه، ضرب و جمع، انجام شود؛ به همین منظور در این بخش در ابتدا مقدار ضرایب از فایل مدل train شده که قبلا ذخیره کرده بودیم، استخراج می‌کنیم و از آن به منظور انجام عملیات محاسباتی استفاده می‌کنیم.

۳.۳.۱- لایه Convolution

این لایه از یک کرنل (فیلتر) n در n و چند کانال تشکیل شده است. فرآیند آن به این صورت است که کرنل روی تصویر قرار گرفته و تمام نقاط نظیر به نظیر در ورودی ضرب شده و در نهایت همه با هم جمع می‌شوند و خروجی را تشکیل می‌دهند (شکل ۳.۳.۱.۱).

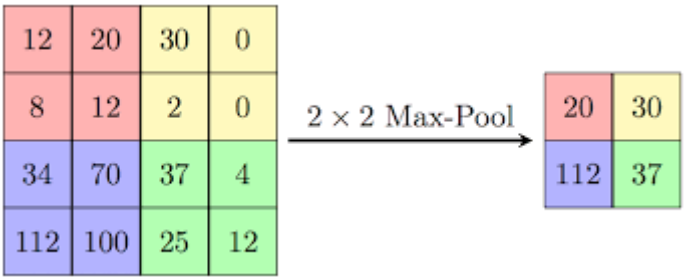


شکل ۳.۳.۱.۱

شکل ۳.۳.۱.۱ یکی از فیلترها را نشان می‌دهد. در لایه اصلی k فیلتر داریم که هر کدام را یک کانال می‌گوییم. برای مثال در لایه اول ۳۲ تا کانال داریم که یعنی در مجموع ۳۲ فیلتر داشته که با اعمال روی ورودی ۳۲ تصویر خروجی خواهیم داشت.

۳.۳.۲- لایه Max Pooling

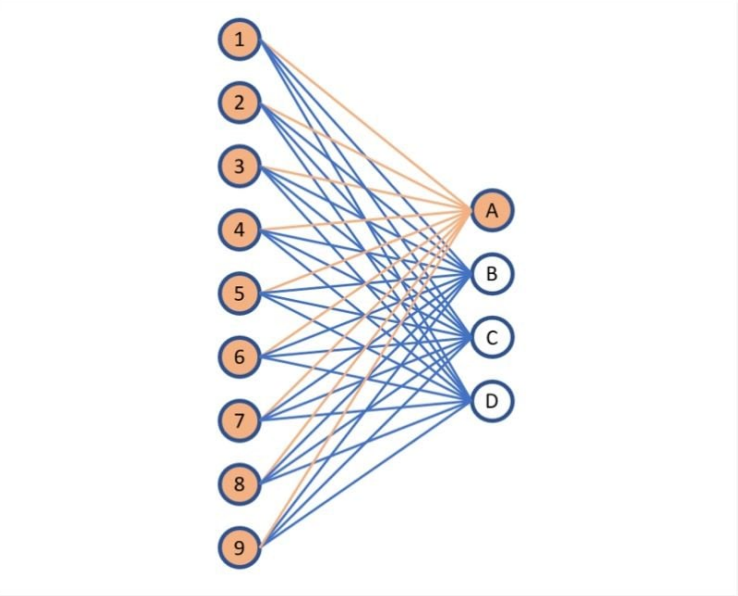
در این لایه یک پنجره ۲ در ۲ وجود دارد که با پیمایش تصاویر ورودی در آن پنجره ۲ در ۲، بیشترین مقدار را انتخاب می‌کند و بعد از آن طوری حرکت می‌کند که همپوشانی با ورودی‌های قبلی نداشته باشد.



شکل ۳.۳.۲.۱- نمونه‌ای از اعمال Max Pooling بر روی ورودی

۳.۳.۳- لایه Dense یا Fully Connected

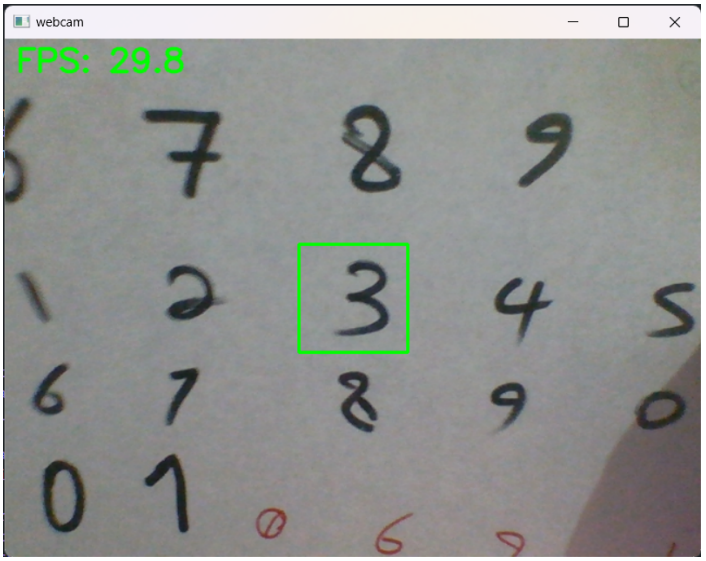
این لایه شامل تعدادی ورودی و خروجی می‌باشد.؛ تمامی ورودی‌ها و خروجی‌ها با یک یال به هم متصل شده‌اند و هر کدام از این یال‌ها دارای وزن می‌باشند. این وزن در ورودی ضرب شده و با ضرب ورودی بقیه یال‌ها در ورودی‌های دیگر (به ازای تمام ورودی‌ها) جمع می‌شود.



شکل ۳.۳.۳- یک نمونه شبکه Dense

۳.۴- اضافه کردن ماژول دوربین

در اینجا برای اینکه ورودی به صورت ثابت نباشد و بتوانیم داده‌های جدید را از بیرون جمع‌آوری کنیم، از ماژول دوربین استفاده می‌کنیم؛ به این صورت که با استفاده از کتابخانه OpenCV یک پنجره باز کرده و با دسترسی به ماژول دوربین تصاویر آن را به صورت frame دریافت کرده و آن را روی ویندوز بازشده نمایش می‌دهیم. سپس یک کادر به اندازه کافی بزرگ با دور سبز رنگ ایجاد می‌کنیم تا مرزی برای عکسی که ذخیره می‌کنیم قرار دهیم (به دلیل بزرگ بودن بیش از حد تصویر هر frame)؛ با این کار تنها کافیست که عدد خود را درون و در مرکز این مربع قرار دهیم و سپس با زدن دکمه enter در زمان مناسب تصویر پردازش شده و خروجی داده شود. در شکل ۳.۴.۱ می‌توان نمونه‌ای از فرآیند گفته شده را مشاهده کرد.



شکل ۳.۴.۱- نمونه‌ای از اتصال ماژول دوربین به مانیتور لپ‌تاپ

۳.۵- پردازش به صورت Real Time

برای اجرای بهتر و داشتن دید بهتر به نحوه انجام پردازش، به جای زدن دکمه enter به این صورت عمل می‌کنیم که به برنامه در قدم اول یک قابلیت Real Time و یک قابلیت Manual اضافه می‌کنیم. پس از این کار به این صورت عمل می‌کنیم:

۱- ابتدا تصویر را از دوربین دریافت کرده و در یک فایل ذخیره می‌کنیم.

۲- سپس با load کردن عکس ذخیره شده، مقادیر ورودی را استخراج می‌کنیم.

۳- ورودی‌ها را به لایه‌ها ارسال کرده و محاسبات روی آن‌ها انجام می‌شود و پس از پایان به اطلاع نرم‌افزار می‌رسد.

۴- از قدم ۱ مجددا تکرار می‌کنیم.

این پیش‌بینی به دو صورت **۱) با استفاده از کتابخانه** و **۲) بدون استفاده از کتابخانه** قابل انجام است.

در حالتی که از کتابخانه استفاده می‌کنیم، FPS[7] بهتری می‌گیریم و این به دلیل موازی‌سازی‌هایی است که در کتابخانه Keras انجام می‌پذیرد. اما پیاده‌سازی بدون استفاده از کتابخانه به دلیل موازی نبودن برخی عملیات‌ها، FPS کمتری می‌گیریم.

۳.۶- اجرا کردن شبکه CNN بر روی بخش PS برد PYNQ

۳.۶.۱- برد PYNQ-Z2

برد PYNQ یک برد توسعه مبتنی بر فریمورک Xilinx Zynq است که برای توسعه و برنامه‌ریزی سیستم‌های الکترونیکی و رایانه‌ای استفاده می‌شود. این برد با همکاری بین شرکت Xilinx و دانشگاه کالیفرنیا در سانتاکروز توسعه داده شده است.

برد PYNQ دارای سخت‌افزاری قدرتمند است که از پردازنده ARM Cortex-A9 و FPGA[8] بر پایه زینک Xilinx تشکیل شده است. این ترکیب اجازه می‌دهد تا برنامه‌هایی با قابلیت‌های پیشرفته و پردازش سریع را اجرا کرده و سخت‌افزارهای قابل برنامه‌ریزی را پیاده‌سازی کند.

این برد با استفاده از محیط توسعه PYNQ، که بر پایه فریمورک Jupyter Notebook است، به برنامه‌نویسان امکان می‌دهد تا به سادگی کدهای Python برای کنترل سخت‌افزار و اجرای الگوریتم‌های پردازشی مستقل از سیستم عامل بنویسند. این محیط توسعه به کاربران اجازه می‌دهد تا از ویژگی‌های PYNQ بهره‌برداری کنند، از جمله این ویژگی‌ها می‌توان پیکربندی سخت‌افزار FPGA، ارتباط با واسط‌های دیگر مانند GPIO و I2C و اتصال به شبکه‌های بی‌سیم مانند Wi-Fi و Bluetooth را نام برد.

با استفاده از برد PYNQ، می‌توان به سادگی پروژه‌های الکترونیکی پیچیده را پیاده‌سازی کرد و از قدرت پردازشی و قابلیت‌های FPGA بهره‌برداری کرد. این برد مناسب برای کاربران حرفه‌ای و آموزشی است و می‌تواند در زمینه‌های

[7] Frame per second

[8] Field programmable gate array

مختلفی مانند سیستم‌های مبتنی بر هوش مصنوعی، اینترنت اشیا، پردازش تصویر و صوت، رباتیک و بسیاری از برنامه‌های الکترونیکی دیگر مورد استفاده قرار گیرد.

۳.۶.۲- پیاده‌سازی

اتصال به پایتون این نرم‌افزار به این طریق است که کابل LAN را به برد وصل می‌کنیم. همچنین یک کابل LAN دیگر که به همان شبکه وصل است را متصل می‌کنیم. با این کار برد PYNQ در شبکه یک سرور به صورت محلی می‌سازد تا وسیله‌های دیگر از جمله PC بتوانند با استفاده از مرورگر به آن متصل شده و وارد محیط Jupyter Notebook شوند.

برای پیدا کردن IP مورد نظر جهت اتصال کافی است که با اتصال PC به پورت Micro USB برد PYNQ و اجرا نرم‌افزاری که از پورت COM پشتیبانی کند (مانند Tera Term) به آن متصل شویم. برای دریافت و ارسال درست اطلاعات باید baud rate را در تنظیمات برابر ۱۱۵۲۰۰ قرار دهیم. سپس با دستور `ifconfig` اولین IP را کپی کرده و در مرورگر paste می‌کنیم و پس از آن به محیط Jupyter دسترسی داریم.

۳.۷- ساخت سیستم‌های ساده و ارتباط PS و PL برد PYNQ

در ابتدا برای اطمینان از کارکرد درست ماژول‌ها و اتصال درست PS و PL یک طراحی ساده انجام می‌دهیم تا قسمت PS و PL به هم متصل شده و قسمت PS یک سری داده را به DMA[9] فرستاده و DMA آن را درون یک FIFO قرار داده و FIFO آن را به DMA برگرداند تا در نهایت DMA دوباره آن را در یک قسمت دیگر از حافظه نوشته و PS آن را بخواند. در صورت طراحی درست، PS باید همان داده‌هایی که فرستاده است را بخواند.

قبل از این که این ماژول را پیاده‌سازی کنیم، لازم است در ابتدا با ساختار PL و PS و نحوه ارتباط این دو قسمت آشنا شویم.

۳.۷.۱- ساختار PL و PS برد PYNQ

در برد PYNQ، ساختار PL (برنامه‌پذیر قابل برنامه‌ریزی) و PS (سیستم پردازنده) بر پایه فریمورک Xilinx Zynq قرار دارند.

PS یک بخش سخت‌افزاری است که شامل پردازنده ARM Cortex-A9 است. این پردازنده دارای سیستم‌عامل Linux است و می‌تواند برنامه‌ها را اجرا کند. PS وظیفه کنترل کلی سیستم را بر عهده دارد و مسئولیت‌هایی مانند مدیریت حافظه، کنترل پورت‌های ورودی/خروجی، و مدیریت منابع سیستمی را بر عهده دارد.

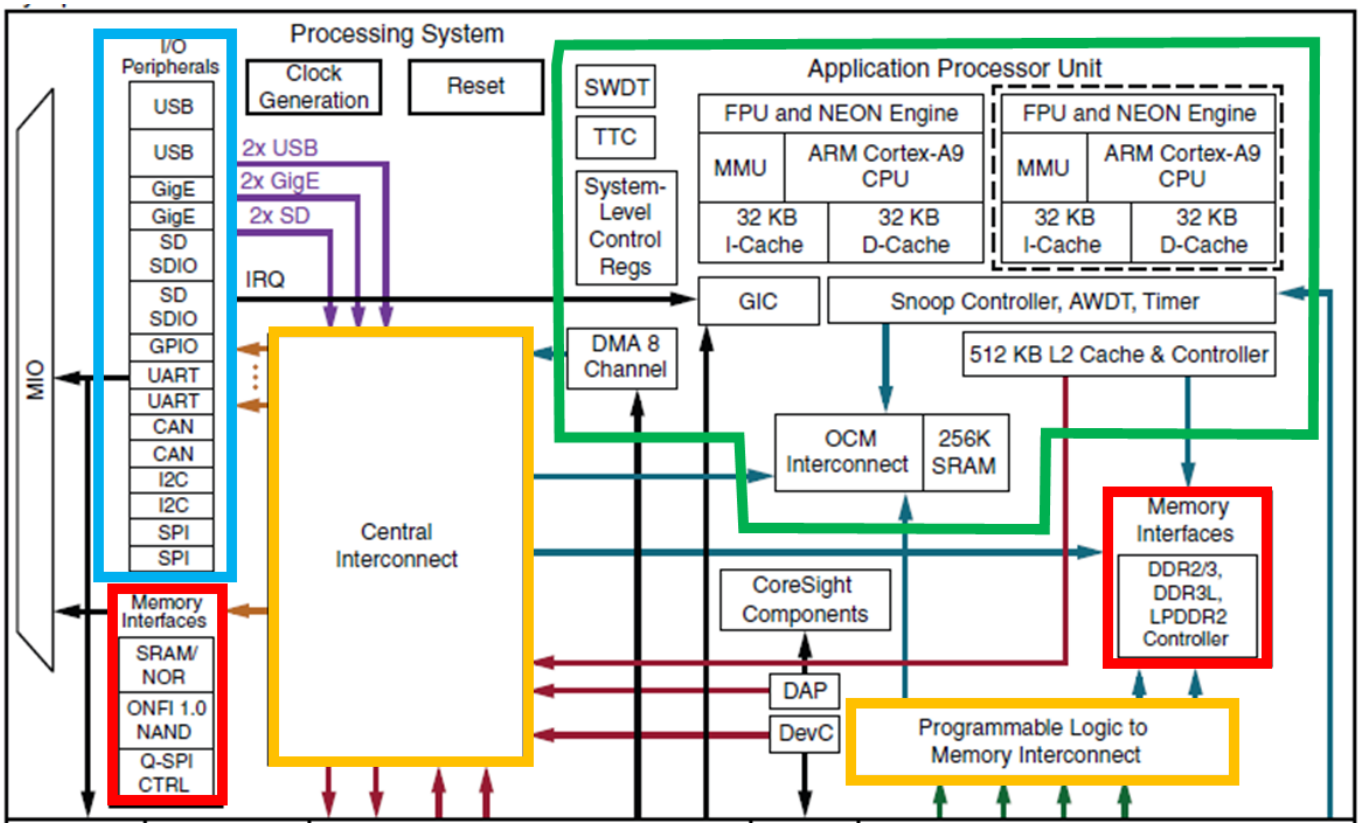
PL یک بخش قابل برنامه‌ریزی است که شامل FPGA است. FPGA قابلیت برنامه‌ریزی و تنظیم مجدد مدارهای منطقی را دارد. با استفاده از زبان های سخت افزاری مانند VHDL یا Verilog، می‌توان سخت‌افزارهای خاصی را در FPGA پیاده‌سازی کرد. بنابراین می‌توانیم عملکرد سخت‌افزار را به طور دقیق و سفارشی تنظیم کرده و برنامه‌های خاصی را اجرا کنیم.

[9] Direct memory access

برای ارتباط بین PL و PS در برد PYNQ، وجود رابط‌های ارتباطی متنوعی از قبیل AXI[10] و رابط‌های DMA[11] امکان‌پذیر است. این رابط‌ها به PS اجازه می‌دهند با PL ارتباط برقرار کند و داده‌ها را به صورت مستقیم، منتقل کند. با استفاده از رابط‌های ارتباطی این دو بخش، می‌توان داده‌ها و سیگنال‌ها را بین PS و PL انتقال داد و در عملکرد سیستم تغییراتی ایجاد کرد.

در مجموع، ارتباط بین PL و PS در برد PYNQ از طریق رابط‌های ارتباطی قابل برنامه‌ریزی برقرار می‌شود، که امکان کنترل و ارسال داده به PL را در اختیار PS قرار می‌دهد و از طرف دیگر، با برنامه‌ریزی FPGA در PL، می‌توان سخت‌افزارهای خاصی را پیاده‌سازی کرده و با PS ارتباط برقرار کرد.

در شکل ۳.۷.۱.۱ می‌توان موارد گفته شده را مشاهده کرد.



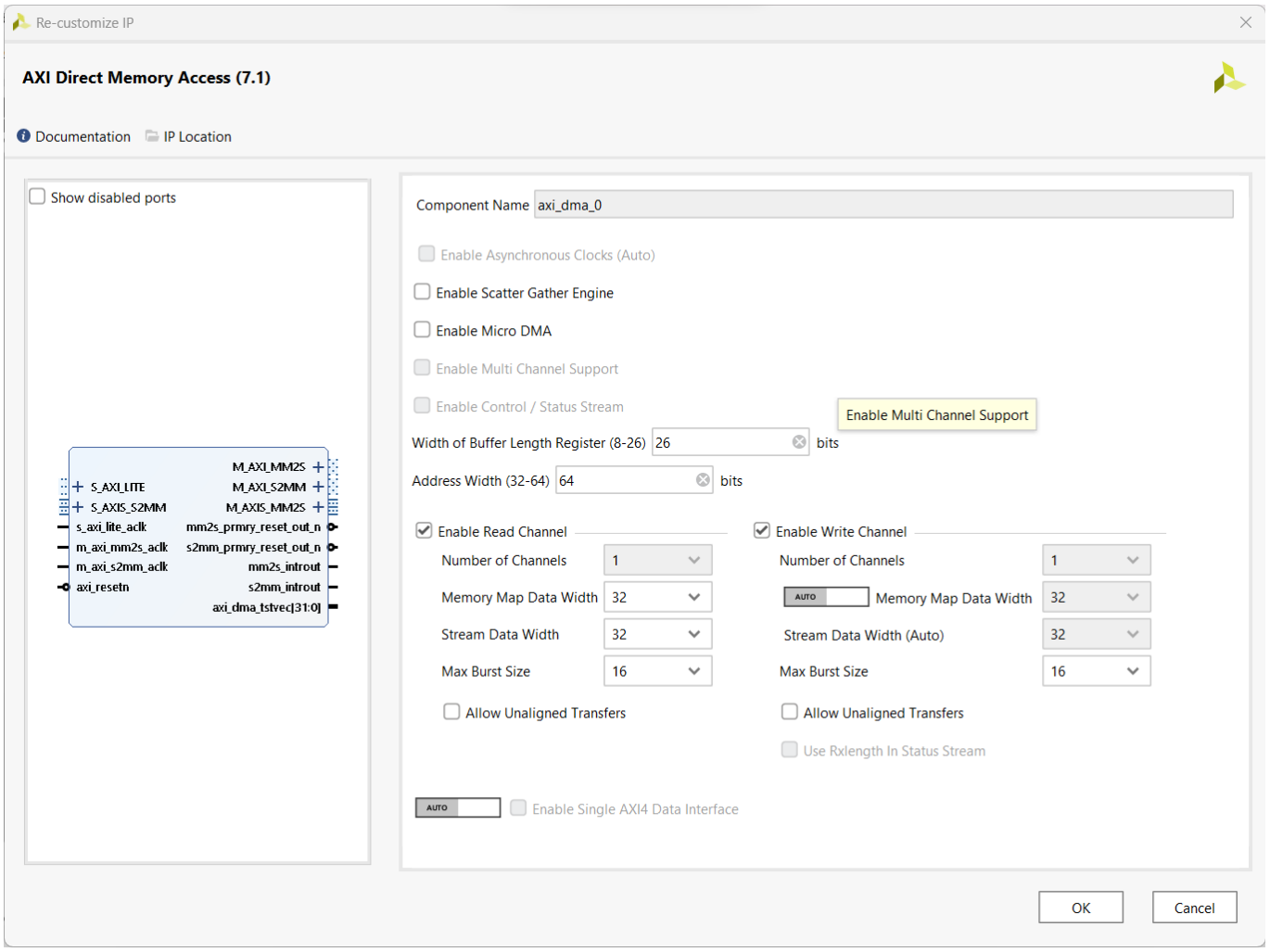
شکل ۳.۷.۱.۱- ساختار درونی برد PYNQ (معماری کلی ZYNQ)

۳.۷.۲- پیاده‌سازی

ابتدا باید طراحی سخت‌افزاری خود را درون نرم‌افزار Vivado انجام دهیم. برای این کار ابتدا یک Block Design جدید درست می‌کنیم؛ سپس به آن ZYNQ Processing System را اضافه کرده و پورت‌های High Performance را فعال می‌کنیم (HP0 و HP2 هر کدام در حالت ۶۴ بیت). سپس DMA را افزوده و پیکربندی آن را مطابق شکل ۳.۷.۲.۱ انجام می‌دهیم.

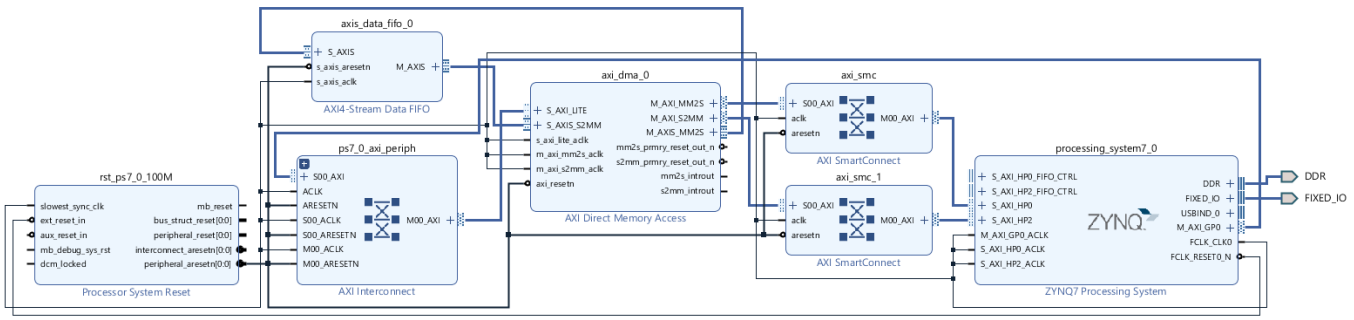
[10] Advanced eXtensible Interface

[11] Direct Memory Access



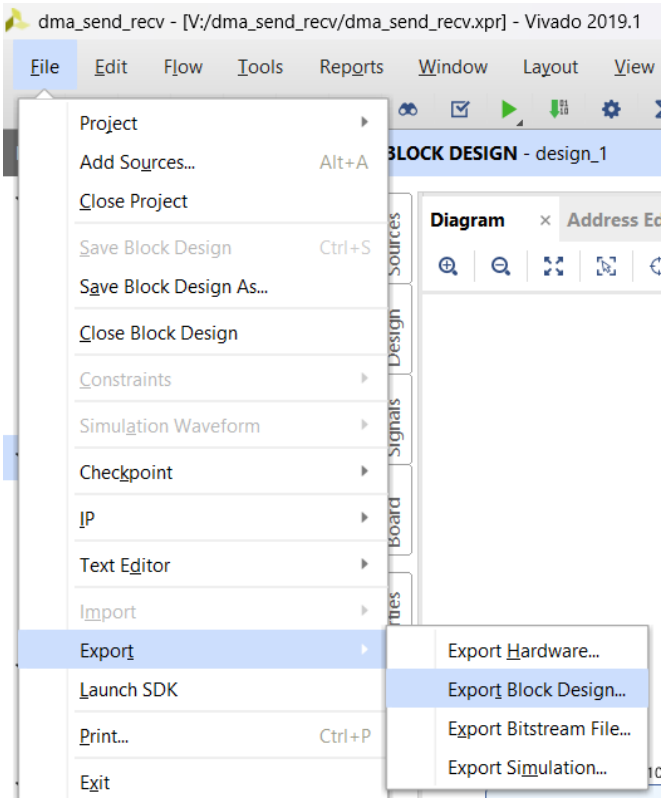
شکل ۳.۷.۲.۱- تنظیمات DMA

سپس یک AXI-4 Stream Data FIFO اضافه می‌کنیم. اتصالات را برقرار کرده تا نهایتا به شکل ۳.۷.۲.۲ برسیم.



شکل ۳.۷.۲.۲- نحوه اتصالات و شمای کلی

پس از انجام این کار یک wrapper برای دیزاین خود درست کرده و در نهایت مرحله Generate Bitstream را انجام می‌دهیم. این عملیات ممکن است زمانبر باشد. پس از اتمام انجام کار و تولید شدن Bitstream، از گزینه‌های مشخص شده در شکل ۳.۷.۲.۳، گزینه‌های Export Block Design و Export Bitstream File را انتخاب می‌کنیم.



شکل ۳.۷.۲.۳- نحوه خروجی گرفتن دیزاین سخت‌افزار

حال فایل‌های خروجی را در Jupyter Notebook آپلود کرده و با اجرا کد پایتون مربوطه، طراحی خود را تست می‌کنیم.

۳.۸- پروتکل AXI-4 Stream

پروتکل AXI-4 Stream یک پروتکل ارتباطی است که برای انتقال داده‌های پیوسته بین ماژول‌ها در سیستم‌های مبتنی بر FPGA استفاده می‌شود. این پروتکل از سری پروتکل‌های AXI شرکت Xilinx است و برای انتقال داده‌هایی با حجم بالا و نرخ انتقال سریع طراحی شده است.

معماری AXI-4 Stream به صورت یکپارچه و ساده‌تری نسبت به دیگر نسخه‌های پروتکل AXI عمل می‌کند و تمرکز اصلی آن بیشتر بر روی انتقال داده‌ها و کمتر برای کنترل و پیکربندی دارد. ویژگی‌های کلیدی AXI-4 Stream عبارتند از:

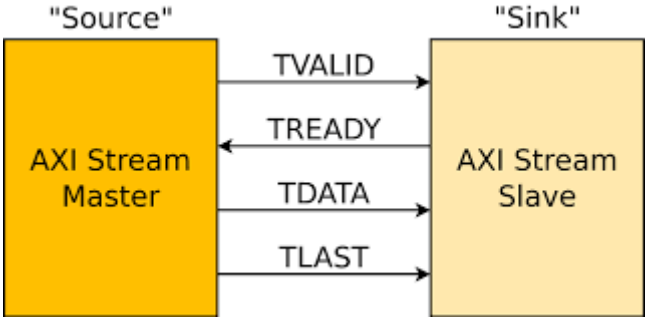
۱- ارتباط پیوسته: AXI-4 Stream برای انتقال داده‌ها از یک سیگنال پیوسته استفاده می‌کند، به این معنی که داده‌ها به صورت پیوسته و بدون وقفه انتقال می‌یابند.

۲- عدم وجود handshake: در AXI-4 Stream، نیاز به مکانیزم handshake برای تایید دریافت یا ارسال داده وجود ندارد. بنابراین، انتقال داده‌ها مستقل از سرعت یا نرخ انتقال برقرار می‌شود.

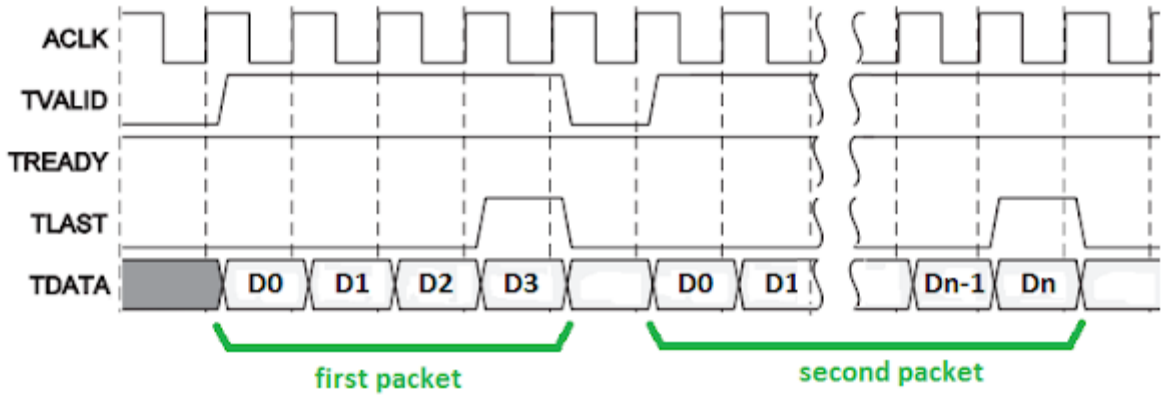
۳- نبود پشتیبانی از چند کانال: AXI-4 Stream تنها یک کانال انتقال داده را پشتیبانی می‌کند و از امکانات چندکاناله پیشرفته AXI-4 و AXI-4 Lite صرف نظر می‌کند.

۴- ساختار ساده: AXI-4 Stream با ساختاری ساده‌تر نسبت به AXI-4 عمل می‌کند، که باعث کاهش پیچیدگی و کاهش مصرف منابع سخت‌افزاری می‌شود.

AXI-4 Stream مناسب برای انتقال داده‌های پیوسته و پرسرعت در سیستم‌هایی است که نیاز به پردازش و انتقال داده‌های بزرگ و پیوسته دارند، مانند فرآیندهای پردازش تصویر، پردازش سیگنال‌های دیجیتال، پردازش سیگنال‌های صوتی و غیره.



شکل ۳.۸.۱- نحوه اتصال دو ماژول Master و Slave

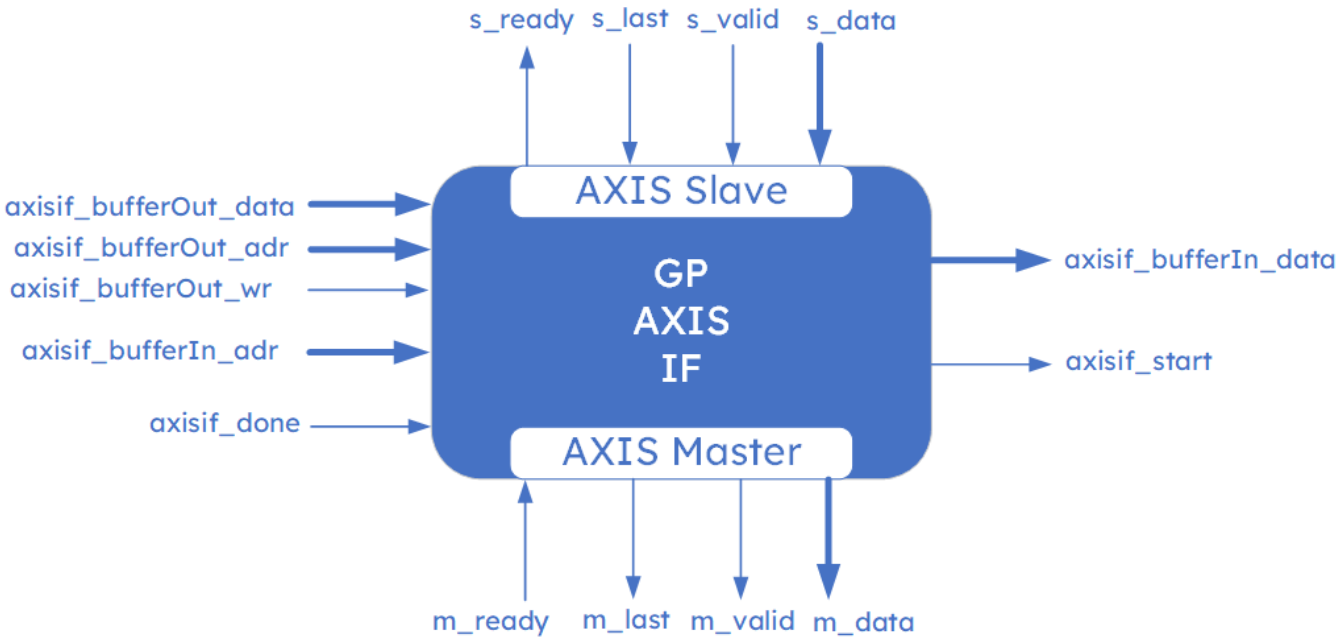


شکل ۳.۸.۲- سیگنال‌ها و شکل موج پروتکل AXIS

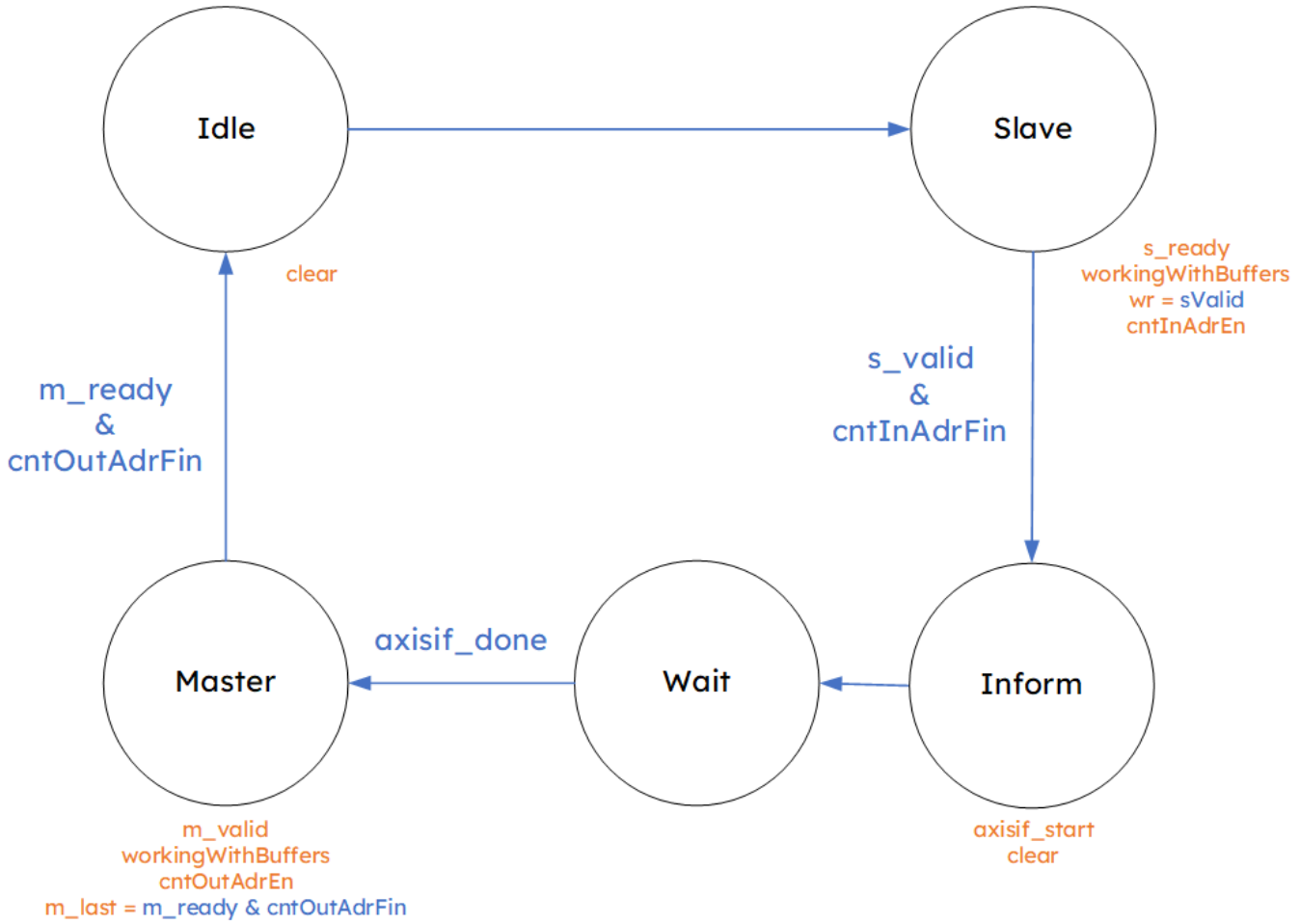
۳.۹- طراحی ماژول General Purpose AXI-4 Stream Interface

به منظور ارتباط ساده‌تر با ماژول‌های دیگر به گونه‌ای که این ماژول عمومیت ساده‌ای داشته و وفق دادن ماژول‌های دیگر با آن ساده باشد و اینکه ما را از پیچیدگی‌های پروتکل AXI-4 Stream دور نگه دارد، این ماژول طراحی و معماری شد. این interface ابتدا حالت Slave دارد؛ به گونه‌ای که تمامی اطلاعات ورودی را درون یک

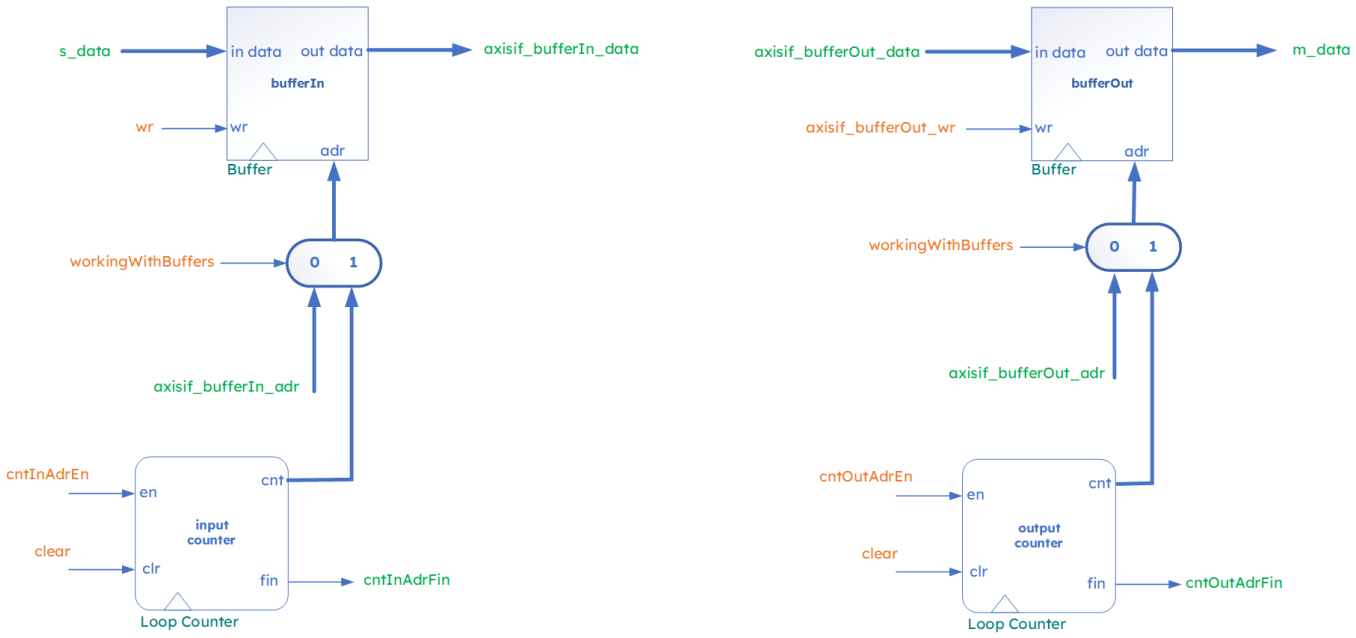
buffer نوشته و در اختیار ماژول متصل قرار می‌دهد و سپس آن ماژول را با یک سیگنال axisif_start فعال می‌کند. این ماژول تمام داده‌ها را در داخل buffer خروجی gp axis interface نوشته و با یک سیگنال axisif_done آن را مطلع می‌کند. پس از آن این interface وارد حالت Master شده و داده‌ها را خروجی می‌دهد. شکل کلی این ماژول و سیگنال‌های ورودی/خروجی آن در شکل ۳.۹.۱ قرار دارد.



شکل ۳.۹.۱- ساختار کلی ماژول GP AXIS IF



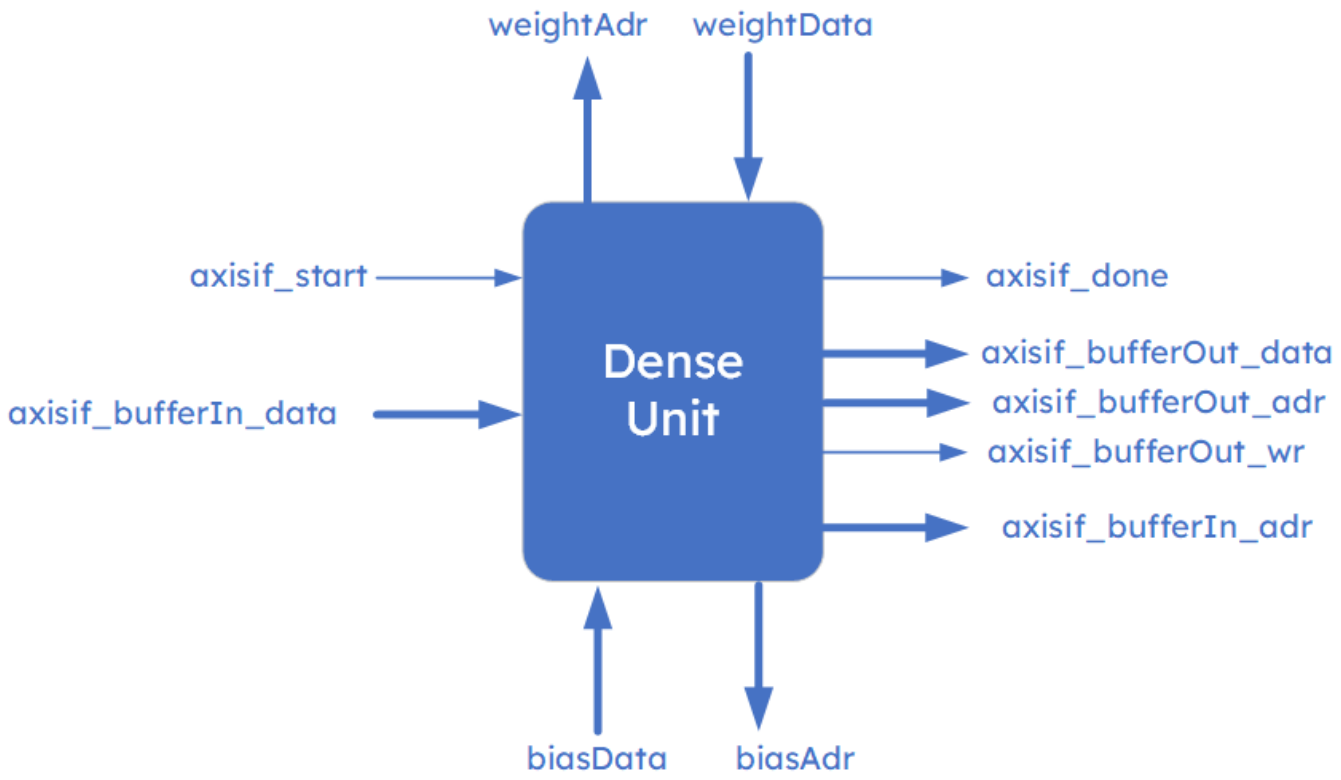
شکل ۳.۹.۲- ساختار Controller ماژول GP AXIS IF



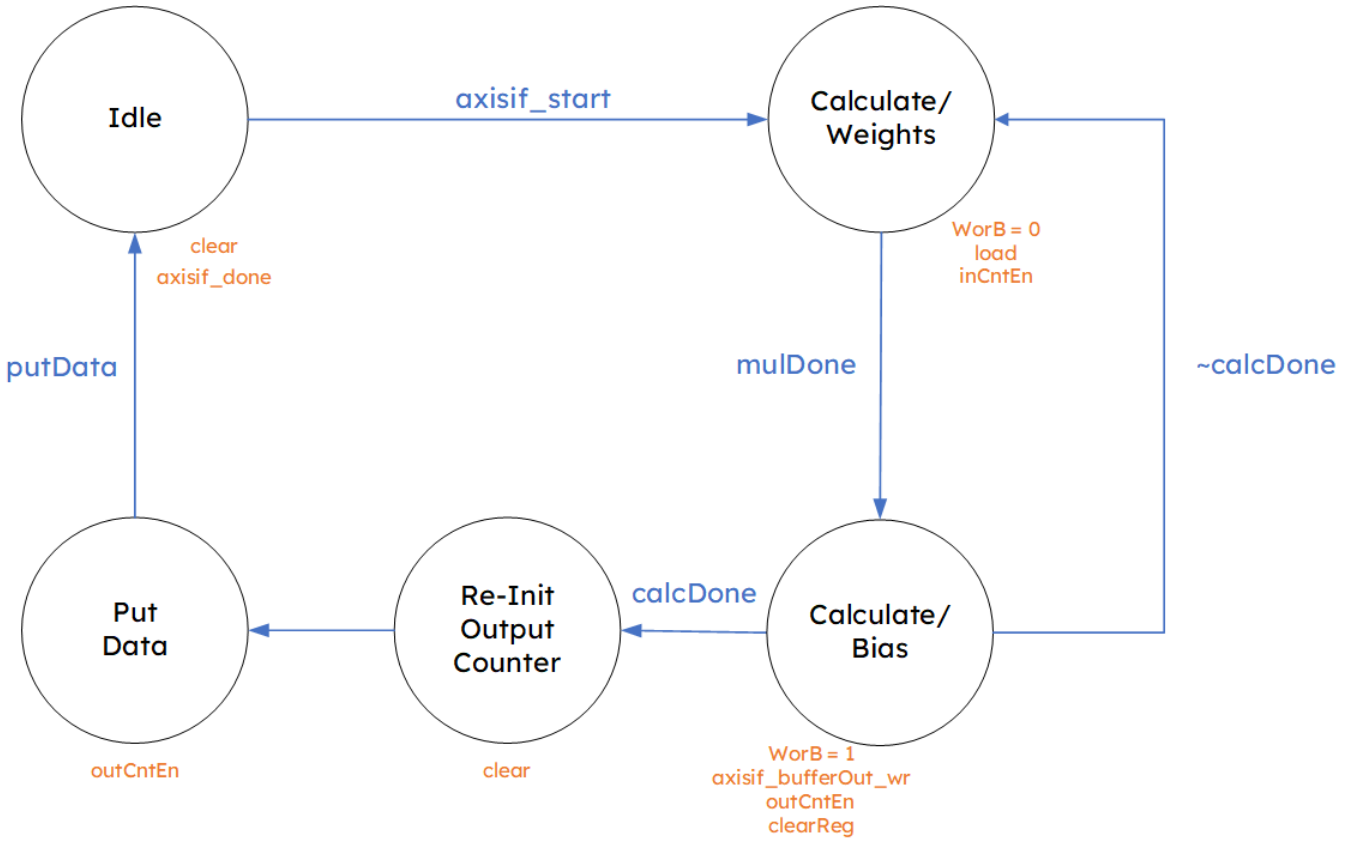
شکل ۳.۹.۳- ساختار Datapath ماژول GP AXIS IF

۳.۱۰- ماژول Dense و ترکیب آن با GP AXIS IF

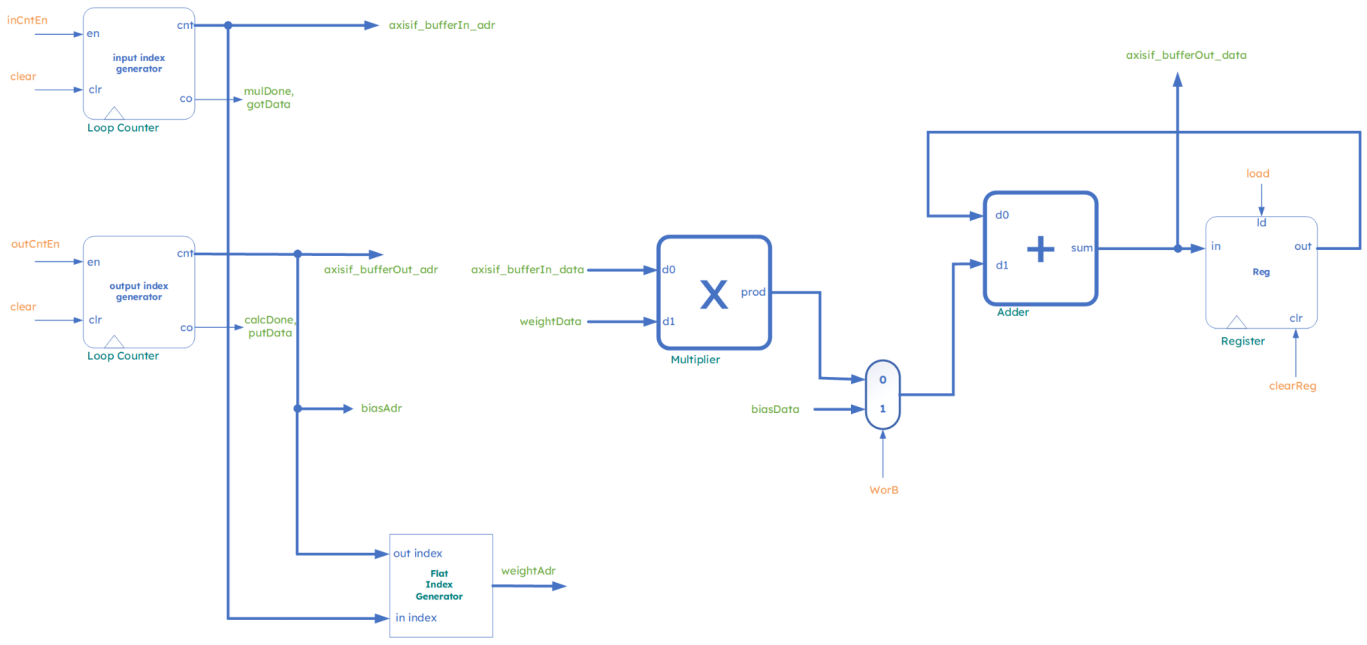
این ماژول تنها یک ضرب‌کننده و جمع‌کننده برای ضرب کردن وزن‌ها در ورودی و جمع کردن آن‌ها با هم و همینطور جمع با bias استفاده می‌کند. این ماژول با GP AXIS IF سازگار است و با استفاده از آن کار می‌کند. به این صورت که ابتدا تمام داده‌ها توسط interface دریافت شده و در بافر ورودی ذخیره می‌شوند. سپس سیگنال start صادر شده و ماژول Dense کار خود را شروع می‌کند و خروجی‌ها را در بافر خروجی interface ذخیره می‌کند. سیگنال‌های ورودی و خروجی ماژول کلی در شکل ۳.۱۰.۱ مشخص شده است.



شکل ۳.۱۰.۱- ساختار کلی ماژول Dense



شکل ۳.۱۰.۲- ساختار Controller ماژول Dense



شکل ۳.۱۰.۳- ساختار Datapath ماژول Dense

۳.۱۱- ماژول SoftMax

این ماژول به این صورت کار می‌کند که ورودی‌های خود را بر روی نمودار نمایی برده و سپس آن‌ها را نرمالایز می‌کند. این لایه درصد اطمینان را به صورت احتمالی تبدیل می‌کند و آن را در یک بازه بین ۰ تا ۱ می‌برد. این ماژول به صورت تخمینی و اعشار ثابت کار می‌کند که جزئیات آن در پیوست ۶.۱ قرار دارد.

۳.۱۲- ماژول Convolution

این ماژول به این صورت کار می‌کند که ورودی‌های خود و مقادیر کرنل خود را به عنوان ورودی گرفته و پس از انجام محاسبات لازم آن را در بافر خروجی می‌نویسد.

۴- نتایج

در این بخش به نتایج گرفته شده از پروژه و مقایسه و تحلیل آن می‌پردازیم که در این راستا سه عنوان زیر در نظر گرفته شده اند:

- از محیط پایتون برای ایجاد مدل شبکه، آموزش شبکه و مشاهده نتایج استفاده می کنیم و در ادامه به اضافه کردن ماژول دوربین و نتایج حاصل از آن اشاره خواهد شد.
- نتایج شبیه سازی در سطح RTL را بررسی خواهیم کرد.
- پیاده سازی سخت افزاری روی برد و مقایسه نتایج پرداخته می شود.

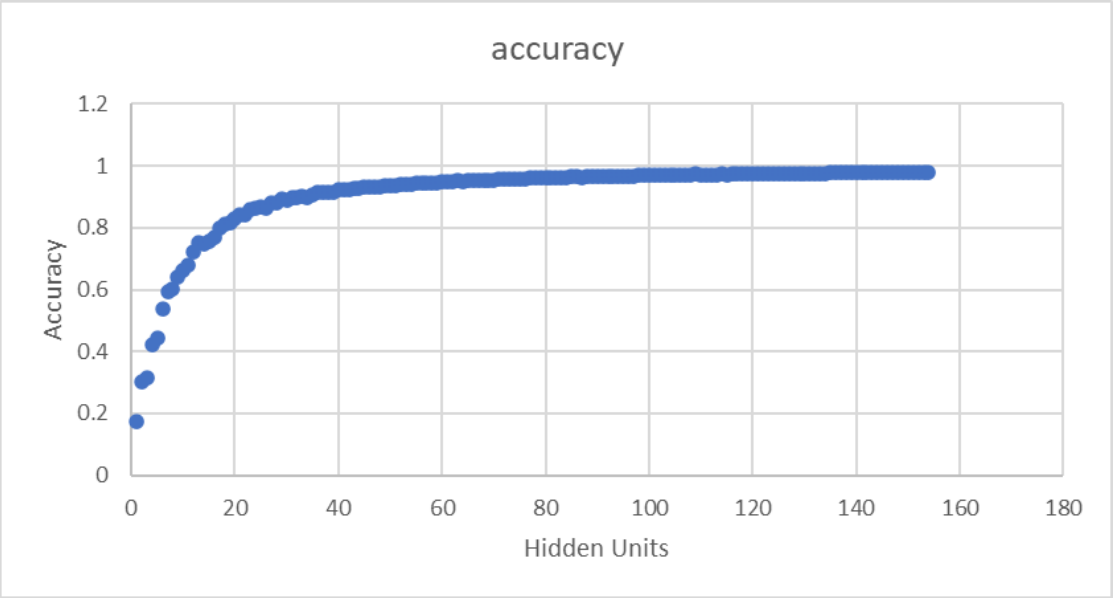
۴.۱- آموزش شبکه MLP برای تشخیص اعداد دست‌نوشته MNIST

در شکل ۴.۱.۱، می‌توانیم معماری و نحوه چینش لایه‌ها را مشاهده کنیم.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               78500

 activation (Activation)     (None, 100)               0

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 100)               10100

 activation_1 (Activation)   (None, 100)               0

 dropout_1 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 10)                1010

 activation_2 (Activation)   (None, 10)                0

=================================================================
Total params: 89,610
Trainable params: 89,610
Non-trainable params: 0
```

شکل ۴.۱.۱

در این شبکه دقت بر اساس hidden units بررسی شد و نتایج train آن در نمودار ۴.۱.۱ مشخص شده است:



نمودار ۴.۱.۱- دقت اعداد دست‌نوشته MNIST بر اساس تعداد hidden unit های شبکه MLP

در قسمت ابتدایی این نمودار مشاهده می‌شود که با افزایش تعداد hidden units، دقت افزایش می یابد. در نیمه دوم نمودار، از یک تعداد hidden unit به بعد، دقت تغییرات چندانی نداشته و افزایش پیدا نمی‌کند. پس برای استفاده کمتر از resource ها، عدد ۱۰۰ را انتخاب کردیم. این شبکه برای این تعداد hidden unit دارای دقت 0.9736 است.

۴.۲- درستی‌سنجی مدل train شده MLP بر روی داده‌های واقعی دست‌نوشته

۴.۲.۱- اعداد دست‌نوشته با رنگ سیاه

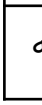
عدد	0	1	2	3	4	5	6	7	8	9
حدس	2	1	3	3	6	5	6	2	3	3
دقت	0.991	0.787	0.999	1.000	1.000	0.524	0.927	0.997	0.509	0.873
نمونه										

جدول ۴.۲.۱.۱- نمونه تصاویر دست‌نوشته و دقت پیش‌بینی مدل

با توجه به جدول ۴.۲.۱.۱ مشاهده می‌کنیم که اعداد زیادی به اشتباه تشخیص داده می‌شوند؛ این در صورتی است که دقت train و test در مجموعه‌داده MNIST برابر با حدود ۹۷ درصد است و این موضوع احتمال overfit شدن را به ما می‌دهد.

در بخش‌های بعدی برای اعدادی با شکل، رنگ و ضخامت مختلف نیز این تست را اجرا می‌کنیم.

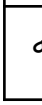
۴.۲.۲- اعداد دست‌نوشته با رنگ آبی، ضخامت نازک

عدد	0	1	2	3	4	5	6	7	8	9
حدس	5	5	5	5	5	5	5	5	5	5
دقت	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
نمونه										

جدول ۴.۲.۲.۱- نمونه تصاویر دست‌نوشته و دقت پیش‌بینی مدل

مشاهده می‌شود که در تمام ستون‌ها، جواب ۵ و دقت ۱ بوده است.

۴.۲.۳- اعداد دست‌نوشته با رنگ آبی، ضخامت کلفت

عدد	0	1	2	3	4	5	6	7	8	9
حدس	5	5	3	5	5	5	5	3	3	5
دقت	1.000	1.000	0.995	1.000	1.000	1.000	1.000	1.000	0.934	1.000
نمونه										

جدول ۴.۲.۳.۱- نمونه تصاویر دست‌نوشته و دقت پیش‌بینی مدل

همچنان ایراد بخش ۴.۲.۲ در این بخش نیز وجود دارد.

۴.۲.۴- نتیجه‌گیری

با توجه به نتایج این قسمت، معماری MLP برای توصیف کردن و پردازش این تصاویر مناسب نیست. علاوه بر آن زیاد کردن تعداد لایه‌های میانی نیز کار را بهتر نکرد.

۴.۳- آموزش شبکه CNN برای تشخیص اعداد دست‌نوشته MNIST

در شکل ۴.۳.۱ نحوه چینش لایه‌های CNN قابل مشاهده است.

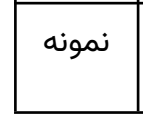
```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential"
_________________________________________________________________
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout (Dropout)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
```

شکل ۴.۳.۱

در نهایت پس از train کردن به دقت 0.9908 رسیدیم.

۴.۴- درستی‌سنجی مدل train شده CNN بر روی داده‌های واقعی دست‌نوشته

۴.۴.۱- اعداد دست‌نوشته با رنگ سیاه

عدد	0	1	2	3	4	5	6	7	8	9
حدس	0	1	2	3	4	5	5	7	0	8
دقت	1.000	1.000	1.000	1.000	0.699	1.000	0.498	0.707	0.677	0.963
نمونه										

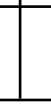
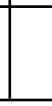
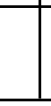
جدول ۴.۴.۱.۱- نمونه تصاویر دست‌نوشته و دقت پیش‌بینی مدل

۴.۴.۲- اعداد دست‌نوشته با رنگ آبی، ضخامت نازک

عدد	0	1	2	3	4	5	6	7	8	9
حدس	2	2	2	2	2	2	2	2	2	2
دقت	0.390	0.392	0.411	0.509	0.456	0.358	0.404	0.322	0.399	0.448
نمونه										

جدول ۴.۴.۲.۱- نمونه تصاویر دست‌نوشته و دقت پیش‌بینی مدل

۴.۴.۳- اعداد دست‌نوشته با رنگ آبی، ضخامت کلفت

عدد	0	1	2	3	4	5	6	7	8	9
حدس	0	1	2	3	4	5	5	2	0	9
دقت	0.999	0.963	0.999	1.000	0.852	0.991	0.508	0.685	0.707	0.822
نمونه										

جدول ۴.۴.۳.۱- نمونه تصاویر دست‌نوشته و دقت پیش‌بینی مدل

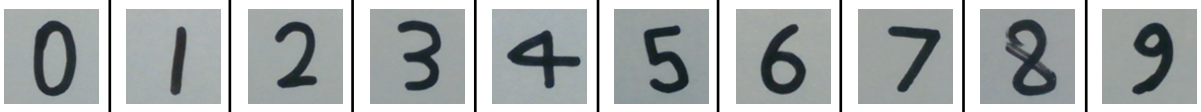
۴.۴.۴- نتیجه گیری

در این بخش مشاهده کردیم که نتایج با معماری CNN به مراتب بهتر از MLP بود؛ در بعضی از تست‌کیس‌های CNN نیز تعداد خیلی کمی خطا وجود داشت که البته درصد دقت غالب آن‌ها کمتر از بقیه اعداد بود.

در بخش ۴.۳.۲ و ۴.۴.۲ که اعداد دست‌نوشته آبی نازک بود، مشاهده کردیم که اعداد به خوبی پیش‌بینی نشدند و این به دلیل واضح نبودن اعداد بود. از طرفی دقت پیش‌بینی در معماری CNN برای این اعداد آبی نازک به مراتب پایین‌تر بود در حالی که این اعداد برای معماری MLP بسیاری بالا بود. به بیان دیگر در معماری MLP با درصد دقت بالایی، عدد را اشتباه حدس می‌زد اما در معماری CNN با درصد پایینی اشتباه تخمین زده می‌شد که این موضوع خود دلیلی برای استفاده از معماری CNN بود.

۴.۵- اضافه کردن ماژول دوربین

در جدول ۴.۵.۱ می‌توان اعداد داخل کادر را به عنوان عکس ثبت شده توسط ماژول دوربین مشاهده کرد. در این اعداد یک سری نکات وجود دارد که به آن‌ها خواهیم پرداخت.

عدد	0	1	2	3	4	5	6	7	8	9
حدس	0	1	2	3	4	5	6	7	8	9
دقت	0.936	0.848	0.967	0.967	0.874	0.976	0.515	0.674	0.800	0.922
نمونه										

جدول ۴.۵.۱- عدد دست‌نوشته ثبت شده توسط ماژول دوربین و پیش‌بینی آن‌ها

تصاویر موجود در جدول ۴.۵.۱ از لحاظ ابعاد مانند داده‌های train مجموعه‌داده MNIST که ۲۸ در ۲۸ پیکسل است، نیست؛ پس برای این موضوع باید عملیات resize بر روی تصاویر ایجاد شود.

نکته دیگری که وجود دارد این است که background تصاویر MNIST باید سیاه بوده و عدد باید با رنگ سفید باشد؛ به همین دلیل باید عملیات invert کردن روی تصاویر صورت بگیرد.

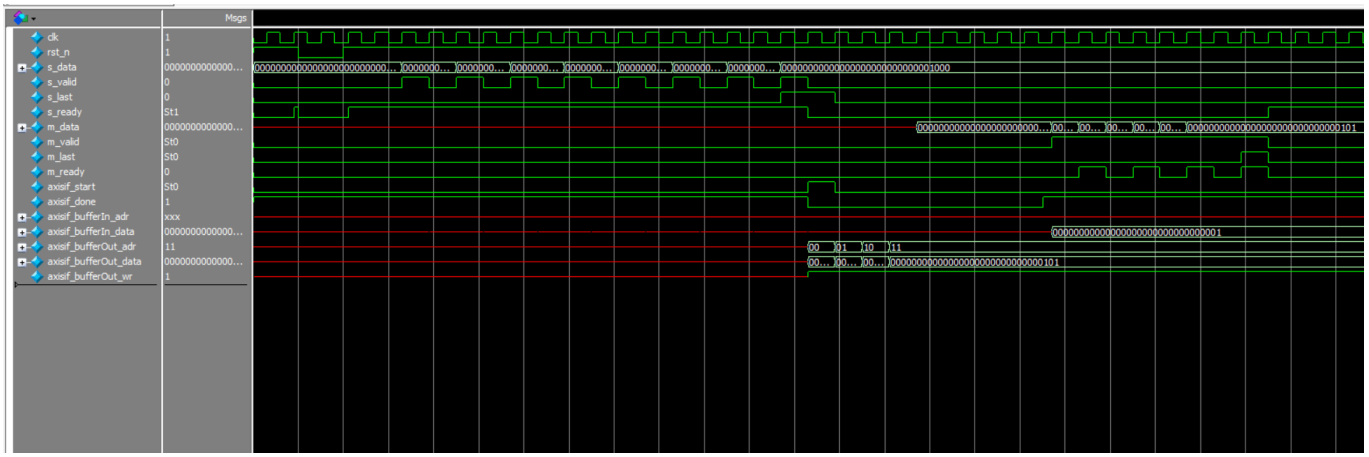
از آن‌جایی که background این تصاویر کاملا سفید نیست و هنگام invert کردن نیز کاملا سفید نخواهد بود ممکن است روی دقت تاثیر بگذارد.

از طرفی این اعداد باید کاملا در مرکز عکس باشند تا شبیه مجموعه‌داده MNIST باشد.

با رعایت این نکات می‌توان به حداکثر دقت در عکس رسید.

۴.۶- شبیه‌سازی ماژول GP AXIS Interface

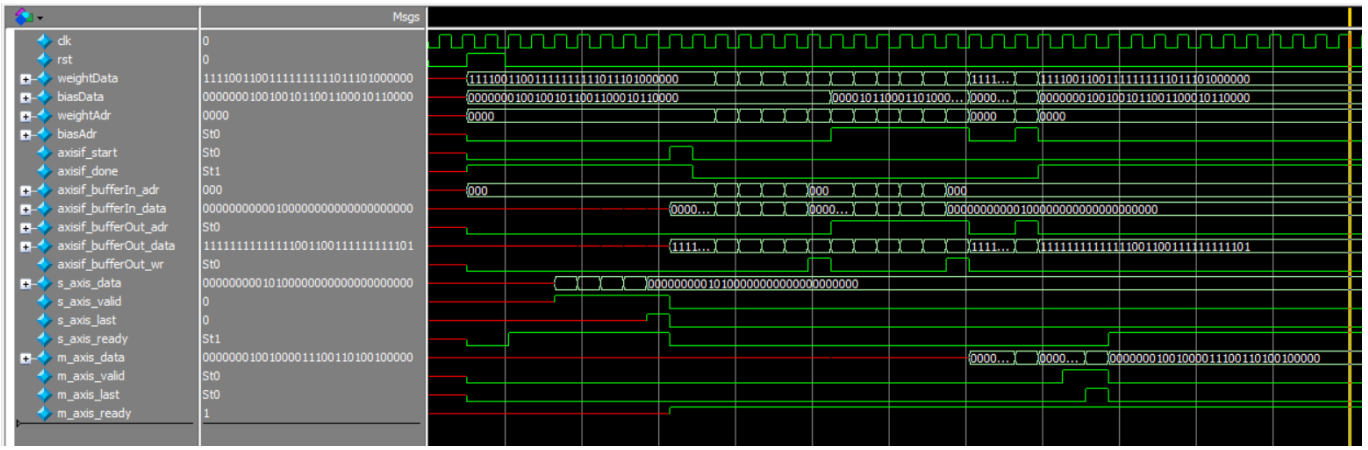
این ماژول به عنوان تست ۸ داده را با برقراری پروتکل AXI-4 Stream گرفته و ۴ داده با همان پروتکل خروجی می‌دهد؛ علاوه بر آن در میان فازهای slave و master، چند سیکل را به محاسبه اختصاص می‌دهد تا رفتار ماژولی که از آن استفاده می‌کند را شبیه‌سازی کند.



شکل ۴.۶.۱- شبیه‌سازی و شکل‌موج ماژول GP AXIS Interface

۴.۷- شبیه‌سازی ماژول Dense یا Fully Connected

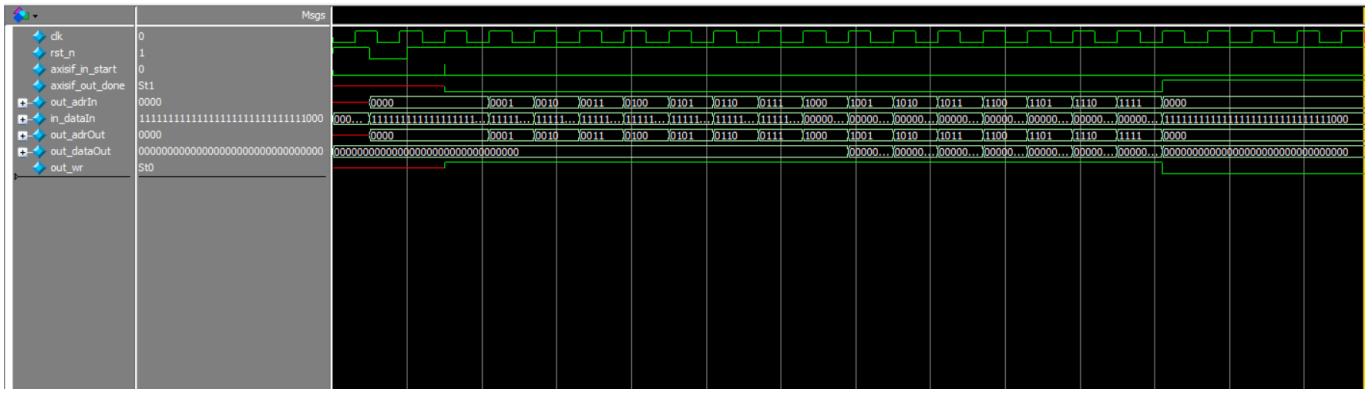
با توجه به تعداد زیاد ورودی برای ماژول Dense -تعداد ۱۶۰۰ تا داده ورودی- زمان simulation نرم‌افزار Modelsim بسیار طولانی شد و باید از یک LUT فیک استفاده کنیم و تعداد ورودی را برابر ۵ و خروجی را برابر ۲ در نظر می‌گیریم. با استفاده از این موضوع ماژول Dense را درستی سنجی می‌کنیم و در شکل ۴.۷.۱ مشاهده می‌شود که ابتدا ۵ داده ورودی از پورت s_axis_data وارد شده و پس از انجام محاسبات لازم آن را در بافر خروجی می‌نویسد و پس از اتمام تمامی محاسبات بر روی پورت m_axis_data برای خروجی ارسال می‌شود.



شکل ۴.۷.۱- شبیه‌سازی و شکل‌موج ماژول Dense

۴.۸- شبیه‌سازی ماژول ReLU

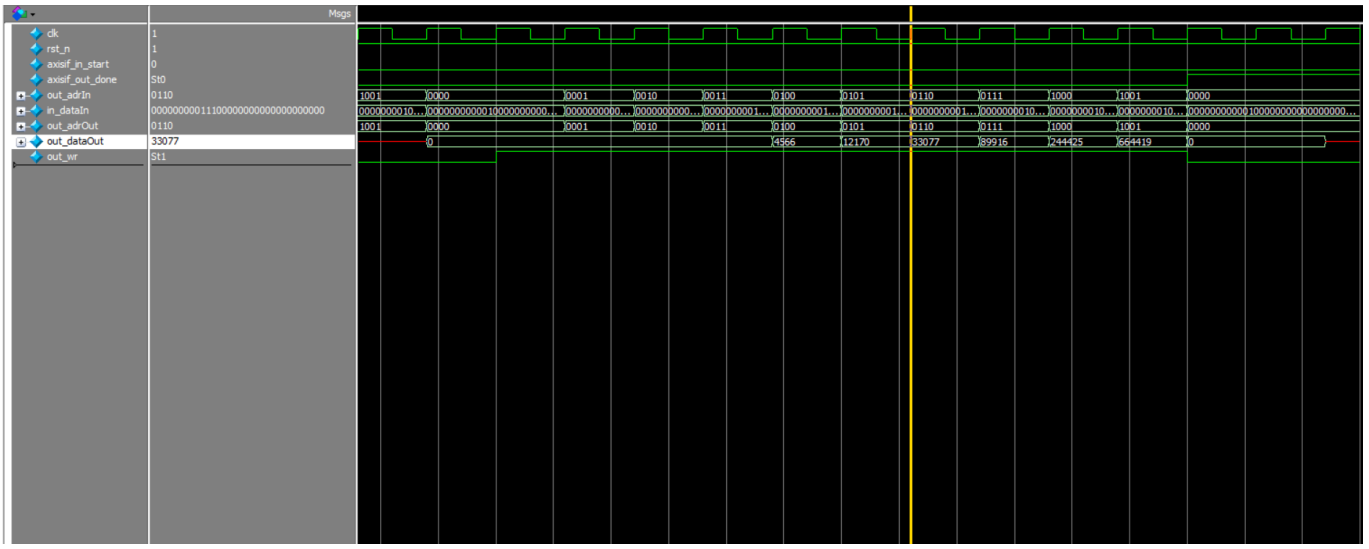
این ماژول به عنوان استفاده عمومی به عنوان activation function طراحی شده است و برای ارتباط با stage های بعدی، پروتکل AXI-4 Stream را پشتیبانی می‌کند. در شکل ۴.۸.۱ نحوه کار این ماژول مشاهده می‌شود.



شکل ۴.۸.۱- شبیه‌سازی و شکل‌موج ماژول ReLU

۴.۹- شبیه‌سازی ماژول SoftMax

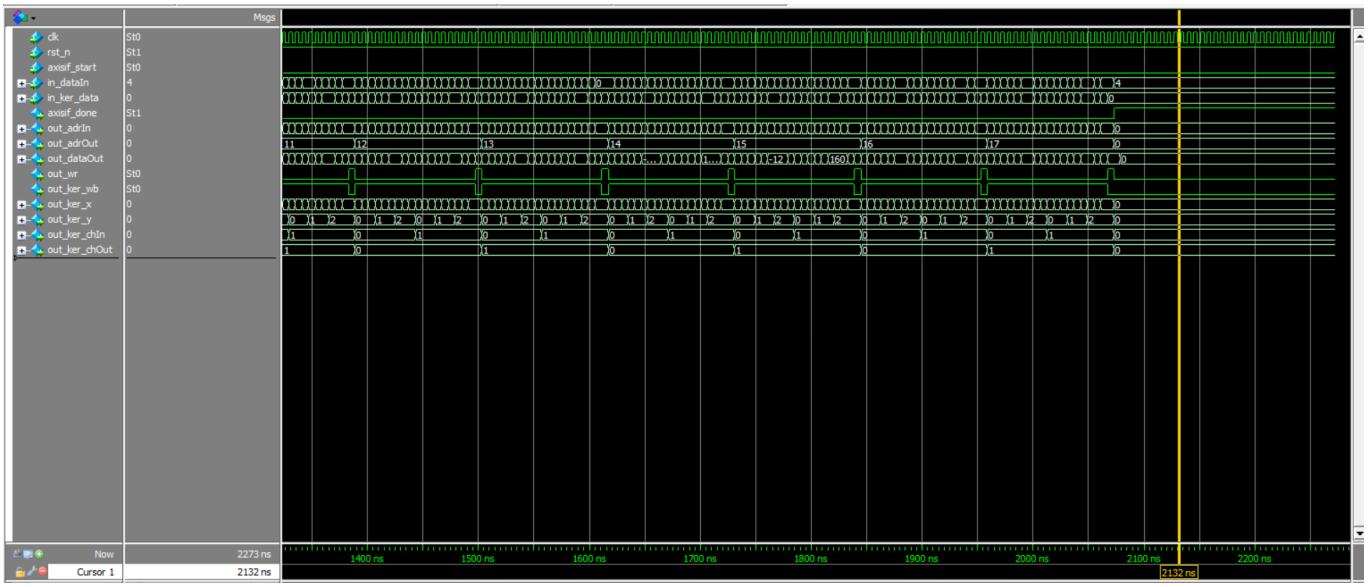
برای شبیه‌سازی ماژول ۱۰ عدد را با تعداد رقم اعشار از پیش تعیین شده، به صورت ورودی دریافت می‌کند و نرمالایز شده نمایی آن‌ها را خروجی می‌دهد (شکل ۴.۹.۱).



شکل ۴.۹.۱- شبیه‌سازی و شکل‌موج ماژول SoftMax برای ورودی‌های ۱ تا ۱۰

۴.۱۰- شبیه‌سازی ماژول Convolution

پس از ورودی گرفتن داده‌ها، این ماژول محاسبات را انجام داده و با آماده شدن تدریجی مقادیر خروجی، بافر خروجی را پر می‌کند؛ در انتها پس از اتمام تمامی محاسبات سیگنال done را صادر می‌کند.



شکل ۴.۱۰.۱- شبیه‌سازی و شکل‌موج ماژول Convolution

۴.۱۱- زمان‌بندی‌های نرم‌افزاری بر روی پردازنده PC

Layer	Name	Time (ms)
layer#0	Convolution 2D	396.3614
layer#0 activation	ReLU	7.0078
layer#1	MaxPooling 2D	6.0443
layer#2	Convolution 2D	1520.4479
layer#2 activation	ReLU	2.9997
layer#3	MaxPooling 2D	2.9995
layer#4	Flatten	0
layer#6	Dense	5.0367
layer#6 activation	Softmax	0

جدول ۴.۱۱.۱- زمان‌بندی اجرای لایه‌های مختلف به صورت نرم‌افزاری روی PC

۴.۱۲- زمان‌بندی‌های نرم‌افزاری بر روی قسمت PS برد PYNQ

در جدول ۴.۱۲.۱ تمامی زمان‌بندی‌ها قابل مشاهده است. لازم به ذکر است که تمامی زمان‌های نوشته شده به ثانیه است.

Layer	Name	Time (s)
layer#0	Convolution 2D	14.1355
layer#0 activation	ReLU	0.1697
layer#1	MaxPooling 2D	0.1646
layer#2	Convolution 2D	36.4254
layer#2 activation	ReLU	0.0842
layer#3	MaxPooling 2D	0.0595
layer#4	Flatten	0.0011
layer#6	Dense	0.0888
layer#6 activation	Softmax	0.0001

جدول ۴.۱۲.۱- زمان‌بندی اجرای لایه‌های مختلف به صورت نرم‌افزاری

۴.۱۳- زمان‌بندی‌های سخت‌افزاری

زمان انتقال داده‌ها از Memory واسطه DMA به یک ماژول AXIS برابر با 0.001171 ثانیه معادل 1.171ms و زمان دریافت داده‌ها نیز برابر با 0.001438 ثانیه معادل 1.438ms است. زمان اول برای خواندن از حافظه و زمان دوم برای نوشتن در حافظه است که به همین دلیل بیشتر طول می‌کشد.
لازم به ذکر است که فرکانس کاری مدار 100MHz است.

Layer	Name	Dimension	Time (ms)
layer#0	Convolution	in: 784, out: 194688	1.95472
layer#0 activation	ReLU	in: 21632, out: 21632	0.43264
layer#1	MaxPooling	in: 21632, out: 21632	0.43264
layer#2	Convolution	in: 5408, out: 2230272	22.3568
layer#2 activation	ReLU	in: 7744, out: 7744	0.15488
layer#3	MaxPooling	in: 7744, out: 6400	0.14144
layer#4	Flatten	in: 1600, out: 1600	-
layer#6	Dense	in: 1600, out: 10	0.176
layer#6 activation	Softmax	in: 10: out: 10	0.005

جدول ۴.۱۳.۱- زمان‌بندی اجرای لایه‌های مختلف به صورت نرم‌افزاری

۴.۱۴- نتیجه زمان‌بندی‌ها و مقایسه قسمت PS و PL برد PYNQ

در قسمت ۴.۱۱ مشاهده شد که اجرا بر روی پردازنده PC خیلی سریع‌تر از پردازنده PYNQ بود. این موضوع می‌تواند دلایل مختلفی داشته باشد:

۱- تفاوت‌های سخت‌افزاری: از آنجایی که پردازنده این برد ARM است و این پردازنده دارای معماری RISC است و پردازنده PC ای که روی آن تست انجام شده است (Intel Core i7)، دارای معماری CISC است و معماری RISC به ذاته کندتر از CISC است، ممکن است باعث تفاوت زمانی زیاد PC با قسمت PS باشد.

۲- بهینه‌سازی: کد ممکن است برای سخت‌افزار برد PYNQ-Z2 بهینه نشده باشد. دستگاه‌های مختلف ساختارهای متفاوتی دارند و برای دستیابی به عملکرد بهینه، بهینه‌سازی‌های خاصی ممکن است لازم باشد.

۳- محدودیت حافظه: برد PYNQ-Z2 دارای حافظه محدودتری نسبت به PC است که می‌تواند بر عملکرد تأثیر بگذارد.

۴- مفسر Python: ممکن است مفسر Python بر روی برد PYNQ-Z2 نسبت به PC کندتر باشد. توزیع‌های Python یا تنظیمات مختلف ممکن است ویژگی‌های عملکردی متفاوتی داشته باشند. سیستم‌عامل روی PC ویندوز و سیستم‌عامل روی قسمت PS برد PYNQ لینوکس است و همین تفاوت ممکن است در این تفاوت نقشی داشته باشد.

از طرفی اجرا کردن به صورت سخت‌افزاری روی PL بسیار سریع‌تر از PS بود و در جدول ۴.۱۴.۱ این زمان‌ها با هم مقایسه شده و speedup آن محاسبه شده است.

Layer	Name	PS Time (ms)	PL Time (ms)	Speed up
layer#0	Convolution 2D	14135.5	1.95472	723147%
layer#0 activation	ReLU	169.7	0.43264	39224%
layer#1	MaxPooling 2D	164.6	0.43264	38045%
layer#2	Convolution 2D	36425.4	22.3568	162927%
layer#2 activation	ReLU	84.2	0.15488	54364%
layer#3	MaxPooling 2D	59.5	0.14144	42067%
layer#4	Flatten	1.1	-	-
layer#6	Dense	88.8	0.176	50454%
layer#6 activation	Softmax	0.1	0.005	2000%

جدول ۴.۱۴.۱- مقایسه زمان‌بندی‌های PS و PL

مشاهده می‌شود که بیشترین speedup ها برای لایه‌های Convolution بوده است. در واقع لایه ۰ حدود ۷۲۰۰ برابر سریع‌تر و لایه ۲ حدود ۱۵۰۰ برابر سریع‌تر شده است.
در مجموع کل زمان برای قسمت PS برد PYNQ برابر با 51.129s بوده و برای قسمت PL برابر با 25.65ms بوده است.

۴.۱۵- نتیجه زمان‌بندی‌ها و مقایسه قسمت PL برد PYNQ با پردازنده PC

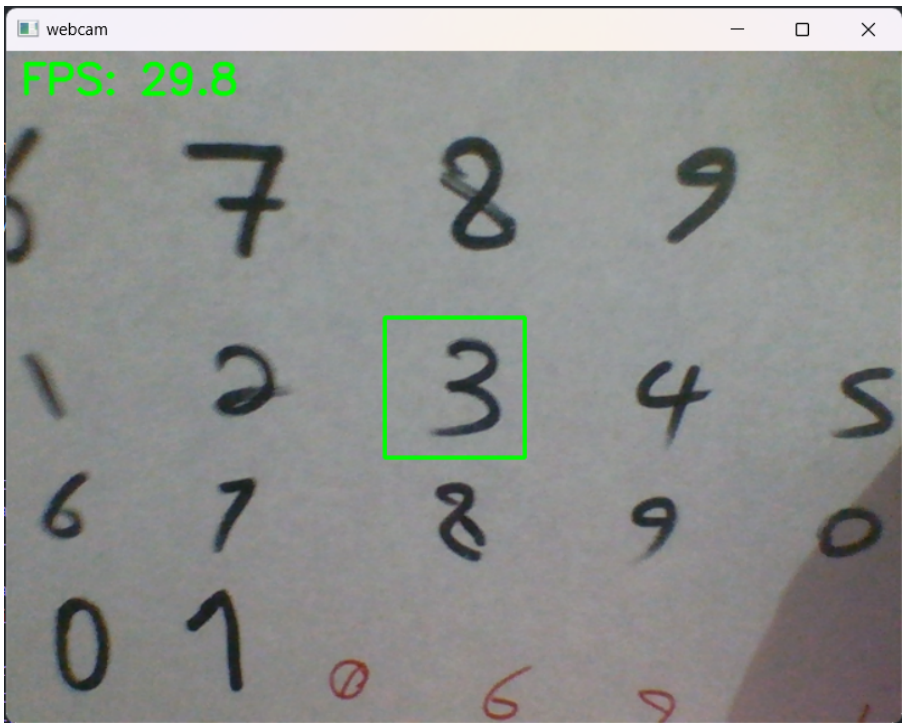
با توجه به پایین بودن سرعت PS برد PYNQ نسبت به پردازنده PC، مقایسه بین پردازنده PC یک لپ‌تاپ با قسمت سخت‌افزاری قابل توجه است.

Layer	Name	PC Time (ms)	PL Time (ms)	Speed up
layer#0	Convolution 2D	396.3614	1.95472	20277%
layer#0 activation	ReLU	7.0078	0.43264	1620%
layer#1	MaxPooling 2D	6.0443	0.43264	1397%
layer#2	Convolution 2D	1520.4479	22.3568	6800%
layer#2 activation	ReLU	2.9997	0.15488	1937%
layer#3	MaxPooling 2D	2.9995	0.14144	2121%
layer#4	Flatten	0	-	-
layer#6	Dense	5.0367	0.176	2862%
layer#6 activation	Softmax	0	0.005	-

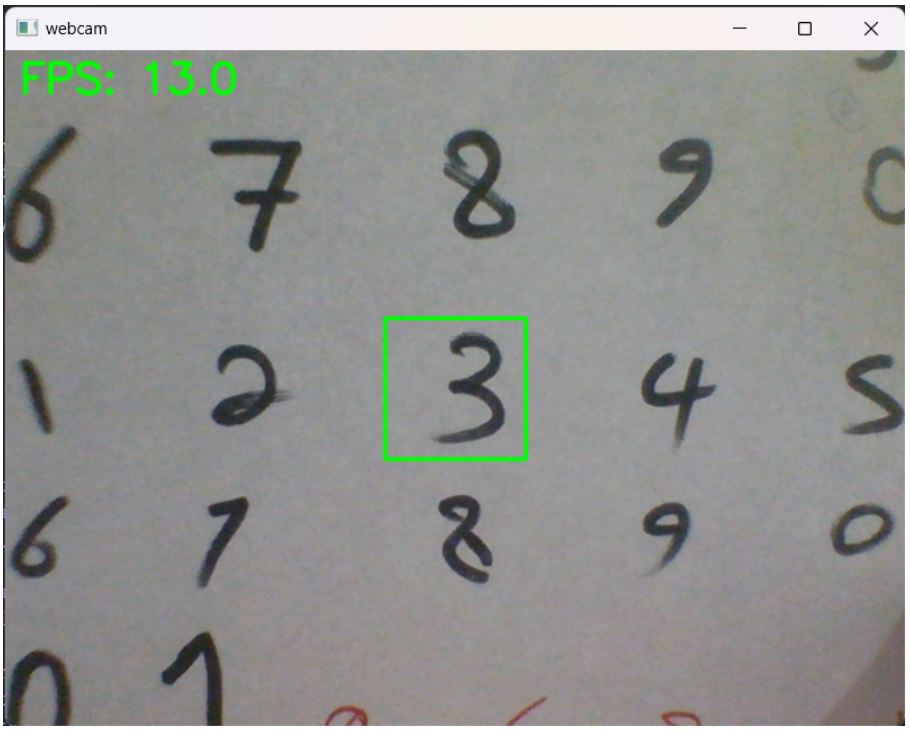
جدول ۴.۱۵.۱- مقایسه زمان‌بندی‌های PL و پردازنده PC

در مجموع مشاهده می‌شود که زمان کلی اجرای PC برابر با 1940.8973ms است.
بیشترین speed up برای لایه‌های Convolution بوده است که در لایه ۰ حدود ۲۰۰ برابر و در لایه ۲ حدود ۷۰ برابر سریعتر شده است.
قابل ذکر است که این تست‌ها صرفا برای یک نمونه عکس است و در صورتی که به صورت realtime کار کنیم، نتایج باز هم به نفع سخت‌افزار باز خواهد گشت زیرا در آنجا قابلیت pipeline هم وجود داشته و می‌توان به صورت همزمان، هنگامی که لایه بعدی در حال کار کردن است لایه قبلی نیز شروع به کار کند اما در نرم‌افزار باید تمام لایه‌ها اجرا شده و سپس محاسبات بر روی فریم بعدی صورت بگیرد.

۴.۱۶- زمان‌بندی اجرا Real Time بر روی پردازنده PC



شکل ۴.۱۶.۱- پردازش Real Time بر روی تصاویر با استفاده از کتابخانه keras و موازی‌سازی



شکل ۴.۱۶.۲- پردازش Real Time بر روی تصاویر با استفاده از حلقه‌های for بدون استفاده از کتابخانه و موازی‌سازی

همانطور که در شکل‌های ۴.۱۶.۱ و ۴.۱۶.۲ مشاهده می‌شود، میزان FPS با استفاده از کتابخانه keras و موازی‌سازی‌هایی که در این کتابخانه در نظر گرفته شده است از ۳۰ فریم بر ثانیه به ۱۳ فریم بر ثانیه، (هنگامی که برای محاسبات لایه‌ها از هیچ کتابخانه و موازی‌سازی‌ای استفاده نمی‌کنیم و صرفا با استفاده از چندین حلقه for تودرتو و ضرایب این محاسبات را انجام می‌دهیم) کاهش می‌یابد. یعنی موازی‌سازی در پردازنده PC بین ۲ تا ۳ برابر می‌تواند کمک کند.

۵- جمع بندی

در این پروژه به بررسی و مقایسه شبکه‌های عصبی MLP و CNN در کاربرد تشخیص تصویر پرداختیم و ماژول‌ها و لایه‌های مختلف را از لحاظ نرم‌افزاری و سخت‌افزاری مقایسه کردیم.

از نکات مهمی که به آن پی بردیم این بود که چه لایه‌هایی Bottleneck طراحی ما بوده و بهتر است آن را به صورت سخت‌افزاری پیاده‌سازی کرد. از طرفی چه لایه‌ای را می‌توان به صورت نرم‌افزاری پیاده‌سازی کرد تا از پیچیدگی زیاد کاسته شود تا در نهایت به یک دیزاین efficient هم از لحاظ زمانی، هم از لحاظ Resource استفاده شده و هم از لحاظ پیچیدگی، رسید. به طور خاص در این پروژه با این معماری CNN دیدیم که یکی از Bottleneck های اصلی سیستم، هر دو لایه Convolution بود و زمان بسیار زیادی را صرف محاسبات می‌کرد؛ در طراحی‌های Real Time این موضوع خیلی بیشتر خود را نشان می دهد. یکی از تکنیک های مورد استفاده برای حل این مشکل طراحی و پیاده سازی سخت افزاری لایه زمانبر و قرار دادن آن به صورت Accelerator در کنار واحد CPU می باشد.

لایه کانولوشن به دلیل داشتن کرنل و چندین کانال باعث افزایش پیچیدگی سیستم شده و به همین دلیل باعث مصرف زمان بسیار زیادی خواهد شد. اگر تعداد سیکل اجرایی مدار را نیز بررسی کنیم، مشاهده می‌شود که هر دو لایه‌ای که بیشترین تعداد سیکل کاری را دارند، Convolution ها هستند.

نکته‌ای که در ابتدا تصور می‌شد این بود که لایه Dense، یک لایه با مصرف زمانی بالا باشد اما در ادامه بعد از پروفایل کردن متوجه این موضوع شدیم که این زمان خیلی هم زیاد نبوده و در order تابع‌های فعال‌سازی است.

در آینده می‌توان علاوه بر قسمت Evaluation، قسمت Train کردن را نیز به صورت سخت‌افزاری پیاده‌سازی کرد؛ زیرا یک بخش عظیمی از زمان در train کردن سپری می‌شود و می‌توان از این موضوع به خوبی استفاده کرد و زمان train کردن را فوق‌العاده کاهش داد. همچنین با استفاده از پارامتری بودن تمام طراحی‌های انجام شده می‌توان با تغییر دادن پارامترها و کم و زیاد کردن لایه‌ها و همچنین استفاده از شبکه‌ها و معماری‌های دیگر، زمان اجرا را برای حالت‌های دیگر محاسبه کرد تا این اطلاعات دید خوبی برای طراحان داشته باشد.

۶- پیوست

۶.۱- الگوریتم تخمینی به کار رفته در محاسبات SoftMax

تعریف تابع SoftMax به شکل زیر است:

$$softmax\left(x_1, \ldots, x_n\right) = \left(y_1, \ldots y_n\right)$$

که

$$y_1 = \frac{exp\left(x_i\right)}{\sum_{j=1}^{n} exp\left(x_j\right)}$$

تعریف می‌کنیم:

$$x_m = \left\{x_j\right\}$$
$$z_i = x_i - x_m$$

که می‌دهد:

$$z_i = x_i - x_m \leq x_m - x_m = 0$$
$$z_m = x_m - x_m = 0$$
$$y_i = \frac{exp\left(z_i + x_m\right)}{\sum_{j=1}^{n} exp\left(z_j + x_m\right)} = \frac{exp\left(z_i\right) exp\left(x_m\right)}{\sum_{j=1}^{n} exp\left(z_j\right) exp\left(x_m\right)} = \frac{exp\left(z_i\right)}{\sum_{j=1}^{n} exp\left(z_j\right)}$$

فرض کنید e_j خروجی سخت‌افزاری که مقدار e^{z_j} را حساب می‌کند باشد و خطای آن را نیز δ در نظر بگیرید.

$$\left|e_j - exp\left(z_j\right)\right| \leq \delta$$
$$\Leftrightarrow exp\left(z_j\right) - \delta \leq e_j \leq exp\left(z_j\right) + \delta$$

همچنین فرض کنید که هنگامی که ماژول محاسبه exponential به جای مقدار دقیق $exp\left(z_i\right)$، عدد e_i را خروجی دهد، ماژول SoftMax که به صورت درونی از ماژول exponential استفاده می‌کند، به جای مقدار دقیق y_i، مقدار تقریبی f_i را خروجی می‌دهد. خطای مطلوب برای f_i را با ε نمایش می‌دهیم.

$$\left|f_i - y_i\right| \leq \varepsilon$$
$$\Leftrightarrow y_i - \varepsilon \leq f_i \leq y_i + \varepsilon$$

هدف محاسبه حداکثر خطای δ برای مقدار e_j است به طوری که خطای f_i از ε تعیین شده بیشتر نشود. می‌توان مساله را به $\delta \in (0, 1)$ محدود کرد.

از روابط زیر استفاده می‌کنیم:

$$exp\left(z_j\right) = exp\left(x_j - x_m\right) \leq exp\left(x_m - x_m\right) = exp\left(0\right) = 1$$
$$\sum_{j=1}^{n} exp\left(z_j\right) \leq \sum_{j=1}^{n} 1 = n$$

$$\sum_{j=1}^{n} exp\left(z_j\right) > exp\left(z_m\right) = 1$$

$$y_i = \frac{exp\left(z_i\right)}{\sum_{j=1}^{n} exp\left(z_j\right)} \leq \frac{exp\left(z_i\right)}{1} = exp\left(z_i\right)$$

$$y_i = \frac{exp\left(z_i\right)}{\sum_{j=1}^{n} exp\left(z_j\right)} \geq \frac{exp\left(z_i\right)}{n}$$

داریم:

$$f_i = \frac{e_i}{\sum_{j=1}^{n} e_j} \geq \frac{exp\left(z_i\right)-\delta}{\sum_{j=1}^{n}\left(exp\left(z_j\right)+\delta\right)} = \frac{exp\left(z_i\right)-\delta}{\sum_{j=1}^{n} exp\left(z_j\right)+n\delta}$$

از این قضیه ریاضیاتی استفاده می‌شود: به ازای $0 \leq x < A$:

$$\frac{x^2}{A^2} \geq 0 \Rightarrow 1 \geq 1 - \frac{x^2}{A^2} = \left(1 - \frac{x}{A}\right)\left(1 + \frac{x}{A}\right) \Rightarrow \frac{1}{1+\frac{x}{A}} \geq 1 - \frac{x}{A}$$

به عنوان محدودیت بیشتر، در نظر می‌گیریم:

$$1 - n\delta \geq 0 \Leftrightarrow 1 \geq n\delta \Leftrightarrow \delta \leq \frac{1}{n}$$

سپس تضمین می‌شود که:

$$n\delta \leq 1 < \sum_{j=1}^{n} exp\left(z_j\right)$$

در نتیجه:

$$f_i \geq \frac{exp\left(z_i\right)-\delta}{\sum_{j=1}^{n} exp\left(z_j\right)+n\delta} = \frac{exp\left(z_i\right)-\delta}{\sum_{j=1}^{n} exp\left(z_j\right)\left(1+\frac{n\delta}{\sum_{j=1}^{n} exp\left(z_j\right)}\right)} \geq \frac{exp\left(z_i\right)-\delta}{\sum_{j=1}^{n} exp\left(z_j\right)}\left(1 - \frac{n\delta}{\sum_{j=1}^{n} exp\left(z_j\right)}\right)$$

$$= \left(\frac{exp\left(z_i\right)}{\sum_{j=1}^{n} exp\left(z_j\right)} - \frac{\delta}{\sum_{j=1}^{n} exp\left(z_j\right)}\right)\left(1 - \frac{n\delta}{\sum_{j=1}^{n} exp\left(z_j\right)}\right) = \left(y_i - \frac{\delta}{\sum_{j=1}^{n} exp\left(z_j\right)}\right)\left(1 - \frac{n\delta}{\sum_{j=1}^{n} exp\left(z_j\right)}\right)$$

$$= y_i - \frac{n\delta}{\sum_{j=1}^{n} exp\left(z_j\right)} y_i - \frac{\delta}{\sum_{j=1}^{n} exp\left(z_j\right)} + \frac{n\delta^2}{\left[\sum_{j=1}^{n} exp\left(z_j\right)\right]^2} \geq y_i - \frac{n\delta}{\sum_{j=1}^{n} exp\left(z_j\right)} \times 1 - \frac{\delta}{\sum_{j=1}^{n} exp\left(z_j\right)}$$

$$= y_i - \frac{(n+1)\delta}{\sum_{j=1}^{n} exp\left(z_j\right)} \geq y_i - \frac{(n+1)\delta}{1} = y_i - (n+1)\delta$$

$$\Rightarrow f_i \geq y_i - (n+1)\delta$$

از طرفی:

$$f_i = \frac{e_i}{\sum_{j=1}^{n} e_j} \leq \frac{exp\left(z_i\right)+\delta}{\sum_{j=1}^{n}\left(exp\left(z_i\right)-\delta\right)} = \frac{exp\left(z_i\right)+\delta}{\sum_{j=1}^{n} exp\left(z_i\right)-n\delta}$$

به عنوان محدودیت بیشتر:

$$1 - n\delta \geq n\delta \Leftrightarrow 1 \geq 2n\delta \Leftrightarrow \delta \leq \frac{1}{2n}$$

در نتیجه:

$$f_i \le \frac{exp\left(z_i\right)+\delta}{\sum_{j=1}^{n} exp\left(z_i\right)-n\delta} \le \frac{exp\left(z_i\right)+\delta}{1-n\delta} = \frac{exp\left(z_i\right)}{1-n\delta} + \frac{\delta}{1-n\delta} \le \frac{exp\left(z_i\right)}{n} + \frac{\delta}{1-n\delta} \le y_i + \frac{\delta}{1-n\delta}$$

که می‌دهد:

$$f_i \le y_i + \frac{\delta}{1-n\delta}$$
$$f_i \ge y_i - (n+1)\delta$$

در نتیجه محدودیت‌ها را به این صورت قرار می‌دهیم:

$$\frac{\delta}{1-n\delta} \le \varepsilon$$
$$(n+1)\delta \le \varepsilon$$

که باعث می‌شود:

$$f_i \le y_i + \frac{\delta}{1-n\delta} \le y_i + \varepsilon$$
$$f_i \ge y_i - (n+1)\delta \ge y_i - \varepsilon$$

پس به صورت کلی باید نامعادلات زیر را حل کنیم:

$$\delta \le 1$$
$$\delta \le \frac{1}{n}$$
$$\delta \le \frac{1}{2n}$$
$$\frac{\delta}{1-n\delta} \le \varepsilon$$
$$(n+1)\delta \le \varepsilon$$
$$\Rightarrow \delta \le min\left\{1, \frac{1}{n}, \frac{1}{2n}, \frac{\varepsilon}{1+n\varepsilon}, \frac{\varepsilon}{n+1}\right\}$$

در پروژه از این اعداد استفاده می‌شود:

$$n = 10$$
$$\varepsilon = 5\% = 0.05$$

که می‌دهد:

$$\delta = 0.0045$$

برای امکان‌پذیر کردن پیاده‌سازی سخت‌افزار، توجه می‌کنیم:

$$exp(-6.0) = 0.0025 < \delta$$

بنابراین هرگاه $z_j \le -6.0$، می‌توان حاصل نمایی را با عدد ۰ تخمین زد.

همچنین نیاز است که برای $-6.0 < z_j \le 0$، تخمینی از تابع نمایی ارائه شود. این تخمین را به صورت بسط تیلور تا x^{18} انجام می‌دهیم. این‌گونه خطای ماژول exponential در این حالت نیز محدود به δ می‌شود.

$$\left| \sum_{p=0}^{18} \frac{z_j^{\,p}}{p!} - exp\left(z_j\right) \right| \le \delta$$

قابل ذکر است که عدد ۱۸ در معادله فوق به طرق آزمون و خطای عددی به دست آمده است.

۷- مراجع

[1] PYNQ

[2] Zynq

[3] XUP PYNQ-Z2

[4] PYNQ Introduction — Python productivity for Zynq (Pynq) v1.0

[5] Verilog AXI stream components for FPGA implementation

[6] AXI4-Lite Interface Wrapper for Custom RTL in Vivado 2021.2 - Hackster.io

[7] AXI4-Stream Interfaces

[8] Visualization of MLP weights on MNIST — scikit-learn 1.2.2 documentation

[9] Serialization and saving | TensorFlow Core

[10] How to predict with your Keras models

[11] Simple MNIST convnet