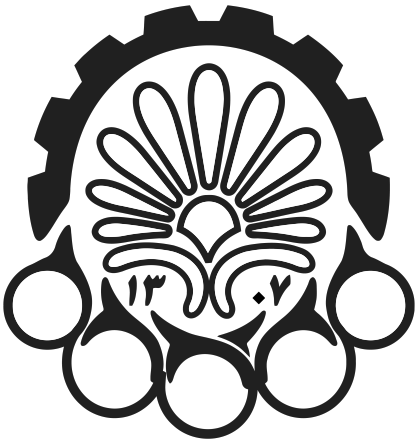
# طراحی سیستم‌های قابل بازپیکربندی

دکتر صاحب‌الزمانی




**دانشگاه صنعتی امیرکبیر**

**( پلی تکنیک تهران )**

دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵


تمرین سری چهارم

۲۷ آذر ۱۴۰۳

# طراحی سیستم‌های قابل بازپیکربندی

تمرین سری چهارم

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

دانشکده مهندسی کامپیوتر

## سوال اول

با ذکر دلیل بیان کنید جملات زیر صحیح هستند یا خیر.

- نگاشت فناوری (technology mapping) می‌تواند بر اساس نوع شبکه ورودی به دو دسته ترکیبی یا ترتیبی طبقه‌بندی شود.

**پاسخ**

**درست.**
این گزاره درست است. بر اساس آنکه شبکه ورودی چه باشد نیاز است تا مشخص شود که فرایند نگاشت قرار است از چه نوعی باشد تا بر اساس آن منابع را اختصاص دهد.

- هدف اصلی نگاشت فناوری FPGA فقط کمینه‌سازی مساحت اشغال شده توسط جداول جستجو است.

**پاسخ**

**نادرست.**
علاوه بر کمینه‌سازی مساحت (یا تعداد LUT) های مصرفی، کمینه‌سازی تاخیر سیگنال ها (افزایش سرعت)، افزایش قابلیت مسیریابی (Routability) و کاهش توان مصرفی نیز جزء اهداف نگاشت فناوری است.

- نگاشت فناوری FPGA عمدتاً از جداول جستجو (LUT) برای عملیات خود استفاده می‌کند و فقط شامل نگاشت LUT است.

**پاسخ**

**درست.**
چون در FPGAها Logic Elementها متشکل است از LUTها، بنابر این در فرایند نگاشت فناوری در FPGAها فقط از LUTها استفاده می‌شود.

- شبیه‌سازی پس از چیدمان (post-layout) اطلاعات کمتری نسبت به شبیه‌سازی قبل از سنتز ارائه می‌دهد.

**پاسخ**

**نادرست.**
در شبیه‌سازی پس از چیدمان، اطلاعات کامل طرح (شامل طول سیم‌ها، تعداد سوییچ‌های موجود در مسیر)، تأخیرهای دقیق (حداکثر فرکانس کلاک) درنظر گرفته می‌شوند. این اطلاعات در شبیه‌سازی قبل از سنتز وجود ندارد، زیرا شبیه‌سازی قبل از سنتز مبتنی بر توصیف منطقی مدار (RTL) است و فاقد اطلاعات فیزیکی دقیق است.

- Chortle-d برای بهینه‌سازی مساحت طراحی شده است.

پاسخ

**درست.**
این الگوریتم از یک رویکرد مبتنی بر DAG (Directed Acyclic Graph) استفاده می‌کند تا گره‌های منطقی مدار را به گره‌هایی با اندازه‌های کوچک‌تر (LUTهای کمتر در FPGA) نگاشت کند. در این فرآیند، تمرکز اصلی بر کاهش تعداد گره‌های نهایی (که به معنای کاهش مساحت است) می‌باشد.

- الگوریتم نگاشت ترتیبی می‌تواند فلیپ‌فلاپ‌ها را در طول فرآیند نگاشت جابجا کند.

پاسخ

**درست.**
قابلیت جابجایی فلیپ‌فلاپ‌ها در نگاشت ترتیبی (با استفاده از Retiming) یکی از ابزارهای اصلی بهینه‌سازی است و امکان طراحی‌های کاراتر و بهینه‌تر را فراهم می‌آورد.

- الگوریتم FlowMap تأخیر سیگنال‌ها را در طراحی‌های نگاشت شده حداقل می‌کند.

پاسخ

**درست.**
FlowMap گره‌های یک مدار منطقی را به گره‌های کوچک‌تر (مانند LUTها در FPGA) تقسیم می‌کند، به گونه‌ای که طولانی‌ترین مسیر بحرانی (Critical Path) کمترین تأخیر ممکن را داشته باشد.

- بهینه‌سازی برای مساحت در نگاشت منجر به کاهش تأخیر نیز می‌شود.

پاسخ

**نادرست.**
این گزاره هم می‌تواند درست باشد و هم نادرست. اگر بهینه‌سازی برای مساحت به معنای کاهش تعداد منابع سخت‌افزاری مورد استفاده (مانند تعداد هاLUT یا گیت‌ها) باشد گزاره نادرست است. اما اگر کاهش مساحت به معنای حذف منطق غیرضروری یا ساده‌تر کردن مدار باشد، مسیرهای بحرانی نیز ممکن است کوتاه‌تر شوند، که منجر به کاهش تأخیر می‌شود و گزاره درست است.

- کارایی مسیریابی مستقل از جایابی در طراحی‌های FPGA است.

پاسخ

**نادرست.**
طراحی‌ها درون FPGA به شدت به Placement وابسته است. تصمیمات مربوط به جایابی می‌توانند تأثیر زیادی بر کارایی مسیریابی داشته باشند.

- در شبیه‌سازی تبرید (simulated annealing)، کاهش هزینه همیشه منجر به پذیرش یک حرکت می‌شود.

پاسخ

**درست.**
در شبیه‌سازی تبرید، کاهش هزینه ($\Delta cost < 0$) همیشه منجر به پذیرش حرکت می‌شود، زیرا هدف الگوریتم یافتن نقطه بهینه با کمترین هزینه است. این ویژگی یکی از اصول اساسی این الگوریتم است که به آن اجازه می‌دهد به‌تدریج به سمت بهینه‌سازی حرکت کند.

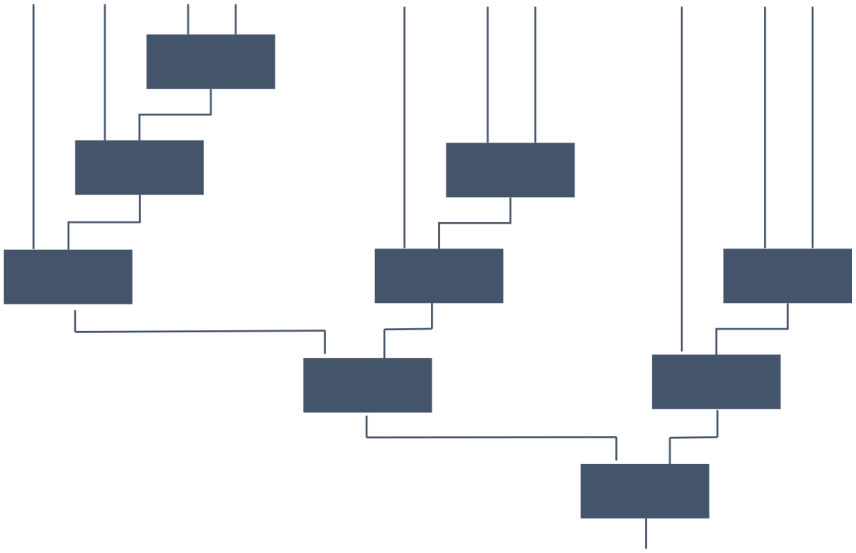
- تابع هزینه در VPR بر اساس طول مسیر و تراکم می‌باشد.

پاسخ

**درست.**
تابع هزینه در VPR یک ترکیب وزن‌دار از طول مسیر و تراکم است. این ترکیب به طراح اجازه می‌دهد که بسته به نیاز، روی کاهش تأخیر یا جلوگیری از تراکم بیشتر تمرکز کند.

## سوال دوم

در کلاس درس، مدار زیر را با هدف حداقل کردن تأخیر به صورت دستی روی LUTهای ۴ ورودی نگاشت کرده‌اید. الگوریتم FlowMap را روی این گراف اجرا کنید و مراحل آن را نشان دهید و نتیجه نگاشت را رسم کنید. هر مستطیل نماینده یک گیت است.



شکل ۱: شکل سوال ۲

## سوال سوم

سه نمونه مختلف از الگوریتم‌های مورد استفاده در نگاشت تکنولوژی FPGA (غیر از الگوریتم‌های تدریس‌شده) را به طور خلاصه توضیح دهید (برای هر کدام یک پاراگراف) و سپس بررسی کنید که چگونه می‌توان آنها را بر اساس توابع هدف و انواع شبکه‌های ورودی طبقه‌بندی کرد.

پاسخ

الگوریتم های مورد بررسی به‌صورت زیر است:

۱. **Performance driven technology mapping for lookup-table based FPGAs using the general delay model [۱]**

در این مقاله، الگوریتمی کارآمد برای نگاشت فناوری مبتنی بر عملکرد برای معماری‌های FPGA مبتنی بر lookup-table با استفاده از مدل تأخیر عمومی ارائه داده شده است. از آنجا که این الگوریتم هیچ محدودیتی بر مقادیر مجاز تأخیر لبه‌ها اعمال نمی‌کند، می‌تواند از اطلاعات تأخیری که در مرحله جایابی تولید می‌شود استفاده کرده و فرآیند نگاشت فناوری را در یک حلقه تکراری به صورت هوشمند هدایت کند.

این الگوریتم از مجموعه‌ای از وزن‌های گره برای اندازه‌گیری میزان بحرانی بودن گره‌ها در یک شبکه بولین استفاده می‌کند. سپس یک جستجوی عمق اول هدایت‌شده به کار می‌رود تا یک ترتیب توپولوژیکی از گره‌ها تولید شود و این ترتیب برای هدایت مرحله خوشه‌بندی استفاده می‌شود. یکی از ویژگی‌های مهم این الگوریتم این است که به‌طور خودکار از مسیرهای همگرا در شبکه استفاده می‌کند.

۲. **Placement-Driven Technology Mapping for LUT-Based FPGAs [۲]**

این مقاله به مطالعه مسئله نگاشت فناوری مبتنی بر جایابی برای معماری‌های FPGA مبتنی بر Table-Lookup به منظور بهینه‌سازی عملکرد مدار پرداخته شده است. کارهای اولیه در حوزه نگاشت فناوری برای FPGAها مانند Chortle-d و Flowmap بر بهینه‌سازی عمق راه‌حل نگاشت‌شده تمرکز داشتند، بدون اینکه تأخیر اتصالات بین‌گِره‌ای را در نظر بگیرند. کارهای بعدی مانند Flowmap-d، Bias-Clus و EdgeMap تأخیر اتصالات را حین نگاشت مد نظر قرار دادند، اما اثرات راه‌حل نگاشت آن‌ها بر جایابی نهایی را لحاظ نکردند. این مقاله به تعامل بین مراحل نگاشت و جایابی تمرکز دارد. ابتدا، اطلاعات مربوط به تأخیر اتصالات از جایابی تخمین زده می‌شود و در فرآیند برچسب‌گذاری استفاده می‌گردد. سپس یک راه‌حل نگاشت مبتنی بر جایابی که هم تراکم سلول‌های Global و هم تراکم سلول‌های Local را در نظر می‌گیرد، توسعه داده می‌شود. در نهایت، یک مرحله Legalization و جایابی دقیق برای پیاده‌سازی طراحی انجام می‌شود.

۳. **LUT-based FPGA technology mapping under arbitrary net-delay models [۳]**

در این مقاله، مسئله نگاشت فناوری مبتنی بر LUT در FPGA تحت مدل تأخیر شبکه دلخواه بررسی شده است. ایده موجود در FlowMap را تعمیم داده شده و الگوریتمی کارآمد توسعه داده شده است که تضمین می‌کند راه‌حل نگاشت بهینه از نظر تأخیر برای شبکه‌های عمومی ارائه شود، به شرط آنکه تأخیر هر شبکه قبل از فرآیند نگاشت مشخص باشد. با محاسبه کارآمد $k-$Feasible Cut با حداقل ارتفاع برای هر گره در شبکه، می‌توان نگاشت بهینه برای هر گره را محاسبه کرد و در نتیجه، راه‌حل نگاشت بهینه برای کل شبکه را با استفاده از برنامه‌ریزی پویا به دست آورد.

*

References

[1] Anmol Mathur and C. L. Liu, ”Performance driven technology mapping for lookup-table

based FPGAs using the general delay model," *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1994. [Link]

[2] Joey Y. Lin, Ashok Jagannathan, and Jason Cong, "Placement-driven technology mapping for LUT-based FPGAs," *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field-Programmable Gate Arrays*, pp. 121–126, 2003. [Link]

[3] Jason Cong, Yuzheng Ding, Tong Gao, and Kuang-Chien Chen, "LUT-based FPGA technology mapping under arbitrary net-delay models," *Computers & Graphics*, vol. 18, no. 4, pp. 507–516, 1994. Published by Elsevier. [Link]

## سوال چهارم

یک مثال از مداری را بزنید که برای مرحله اول Chortle-crf، روش first-fit جواب بهتری نسبت به best-fit می‌دهد.

## سوال پنجم

مفهوم برش k-feasible در نگاشت فناوری FPGA را توضیح دهید و مزایای استفاده از آن در بهینه‌سازی طراحی را در یک پاراگراف شرح دهید.

پاسخ

برش $k-$feasible cut در نگاشت فناوری FPGA به مفهومی اشاره دارد که در آن یک گره در شبکه بولین به همراه تمام گره‌های پیشین آن به یک برش تقسیم می‌شود، به‌طوری که تعداد گره‌های موجود در این برش حداکثر $k$ باشد. این مفهوم زمانی کاربرد دارد که طراحی مدار برای FPGAهای مبتنی بر LUT انجام می‌شود، جایی که هر LUT می‌تواند حداکثر $k-$ورودی داشته باشد. در این روش، گره‌های موجود در برش به ورودی‌های یک LUT نگاشت می‌شوند.
از مزایا مفهوم می‌توان به موارد زیر اشاره نمود:
استفاده از برش $k-$قابل قبول در نگاشت فناوری موجب بهینه‌سازی تأخیر و کاهش عمق مدار می‌شود، زیرا الگوریتم می‌تواند به‌طور موثری مسیرهای بحرانی را شناسایی و نگاشت کند. این روش همچنین به دلیل محدود کردن تعداد گره‌ها در برش، بهره‌وری منابع FPGA را افزایش می‌دهد و استفاده بهینه از LUTها را ممکن می‌سازد. به‌علاوه، الگوریتم‌های مبتنی بر $k-$feasible cut مانند FlowMap به‌طور کارآمد و در زمان چندجمله‌ای عمل کرده و راه‌حل‌های بهینه را با تضمین درستی و عملکرد تولید می‌کنند.

## سوال ششم - پروژه عملی

در ادامه پروژه قبلی دو لایه مخفی کاملاً متصل را به سیستم خود متصل کنید. علاوه بر این یک لایه خروجی با ۱۰ نورون نیز برای خروجی شبکه در نظر گرفته و به شبکه متصل شود.

۱. عملکرد شبکه کاملاً متصل را به صورت مستقل بررسی کنید.

۲. در صورتی که شبکه مشابه در پایتون آموزش داده شده و وزن‌های آن برای تست شبکه در نظر گرفته شود، ۱۰% امتیاز بیشتر برای بخش پروژه در نظر گرفته می‌شود.

۳. در صورتی که کل شبکه (شامل لایه‌های کانولوشن و کاملاً متصل) در پایتون آموزش داده شده و وزن‌های آن برای تست شبکه در نظر گرفته شود ۲۰% امتیاز بیشتر برای بخش پروژه در نظر گرفته می‌شود.

در صورت انجام «۲» یا «۳» نیازی به انجام بخش «۱» نمی‌باشد.

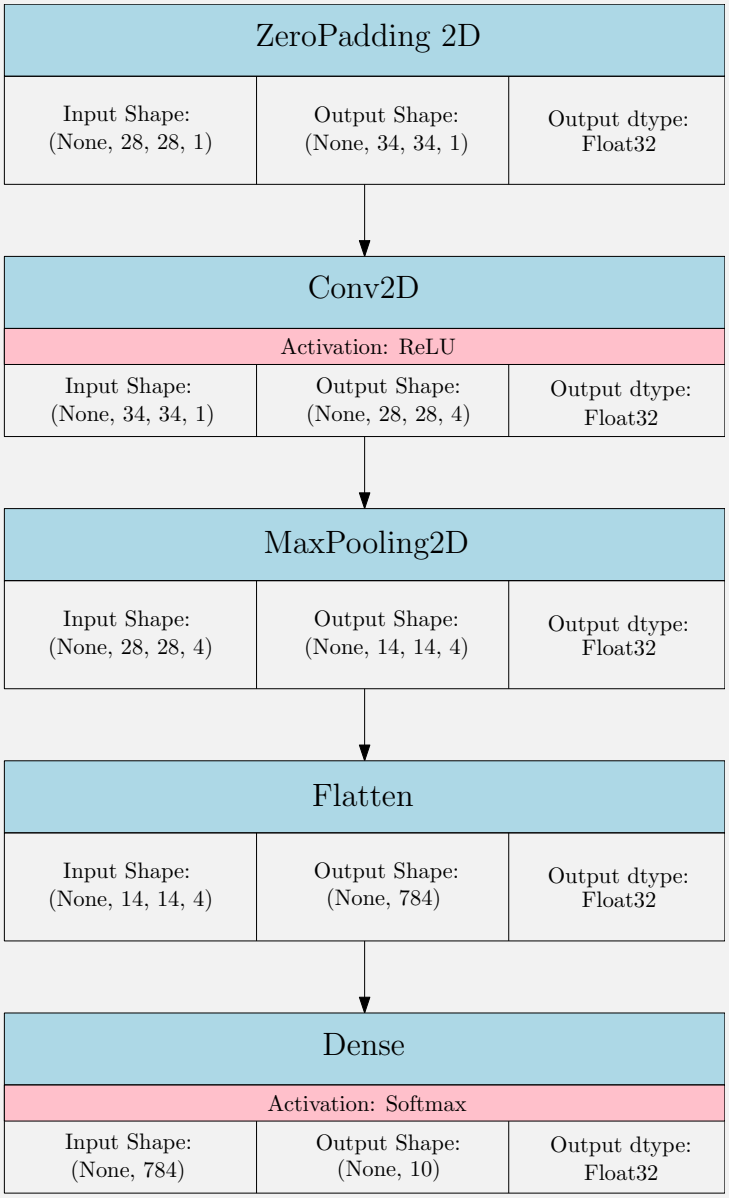
پاسخ

در این پروژه قصد داریم تا پروژه‌های قبلی را با اضافه نمودن یک لایه Fully Connected به آن تکمیل کرده و یک شبکه عصبی CNN را پیاده‌سازی کنیم. بدین منظور پروژه را به دو فاز تقسیم می‌کنیم:

۱. فاز نرم‌افزاری

۲. فاز سخت‌افزاری

**فاز نرم‌افزاری:** در مرحله اول ابتدا به‌صورت نرم افزاری شبکه مورد نظر را تعریف کرده و آن را با داده‌های مجموعه داده MNIST آموزش می‌دهیم تا از وزن‌های آن در فاز سخت افزاری استفاده کنیم.
بدین منظور شبکه ای با معماری زیر را تعریف می‌کنیم:




شکل ۲: معماری شبکه مورد نظر

کد نوشته شده برای پیاده سازی شبکه به‌صورت زیر است:

Listing 1: Model Definition

```
def define_model() -> Sequential:
    # Define model.
    model = Sequential()
    model.add(ZeroPadding2D(padding=pad, input_shape=(input_size[0],
              input_size[1], 1), name="padding_layer"))
    model.add(Conv2D(conv_filter_num, conv_kernel_size, activation="relu",
              padding="valid", kernel_initializer="he_uniform",
              input_shape=(30, 30, 1), name="convolution_layer"))
    model.add(MaxPooling2D(pool_size, name="max_pooling_layer"))
    model.add(Flatten(name="flatten_layer"))
    model.add(Dense(10, activation="softmax", name="dense_layer"))
    # Compile model.
    model.Compile(optimizer=Adam(), loss="categorical_crossentropy",
              metrics=["accuracy"])
    # Return model.
    return model
```

## پاسخ

با کامپایل کردن مدل نوشته شده خروجی شبکه تعریف شده به‌صورت زیر می‌شود:

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| padding_layer (ZeroPadding2D) | (None, 34, 34, 1) | 0 |
| convolution_layer (Conv2D) | (None, 28, 28, 4) | 200 |
| max_pooling_layer (MaxPooling2D) | (None, 14, 14, 4) | 0 |
| flatten_layer (Flatten) | (None, 784) | 0 |
| dense_layer (Dense) | (None, 10) | 7,850 |

Total params: 8,050 (31.45 KB)

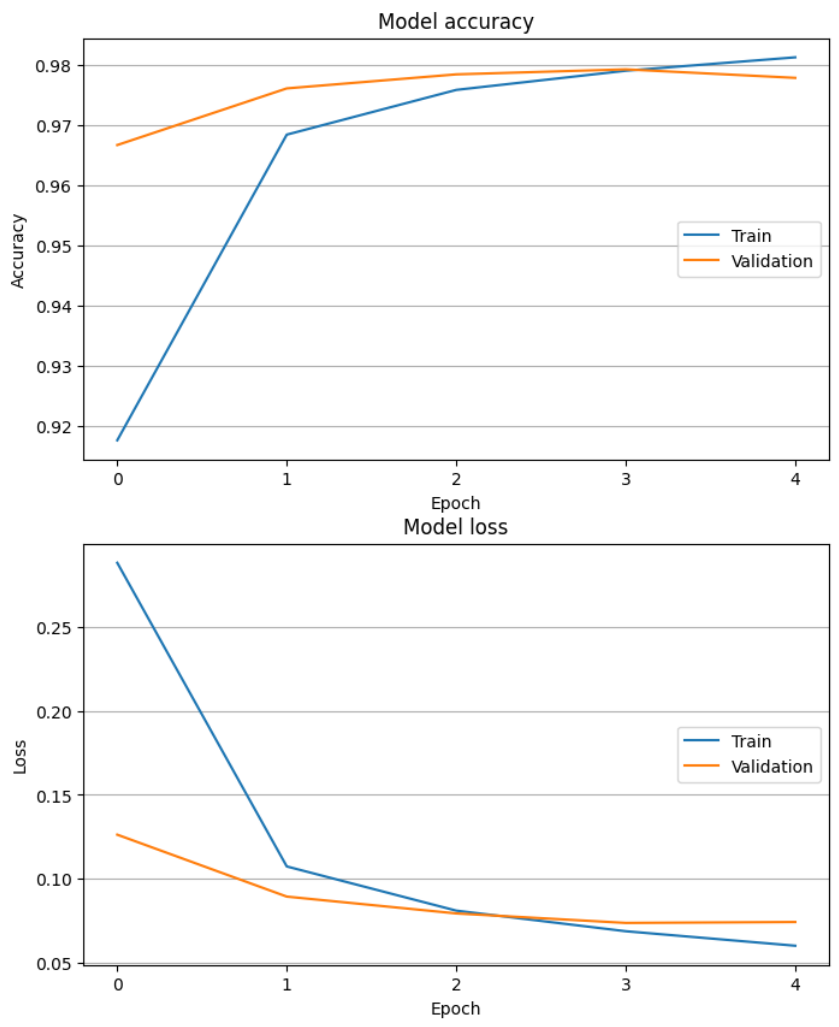
Trainable params: 8,050 (31.45 KB)

Non-trainable params: 0 (0.00 B)

شکل ۳: ساختار شبکه تعریف شده

پس از تعریف ساختار شبکه، شبکه را با استفاده از دیتاست MNIST در ۵ Epoch آموزش می‌دهیم. که نمودار Loss و Accuracy شبکه برای داده‌های Train و Validation به‌صورت زیر به‌دست می‌آید:

پاسخ




شکل ۴: نمودارهای Loss و Accuracy

دقت شبکه بر روی داده‌های آموزش و تست به ترتیب ۰/۹۸۱۷ و ۰/۹۸۰۱ به‌دست آمده است.
همچنین زمان انجام فاز Inference نرم‌افزاری برای ۱۰۰ داده از مجموعه داده MNIST، ۴۶/۸۰۷۲ میلی‌ثانیه به‌دست آمده است.
در نهایت تمامی وزن‌ها و پارامترهای مورد نیاز شبکه برای فاز سخت‌افزاری را در ۳ فایل با نام‌های:

- `conv_weights.h`
- `dense_weights.h`
- `definitions.h`

ذخیره می‌کنیم. فایل `conv_weights.h` شامل وزن‌های به‌دست آمده از لایه‌های کانولوشن، فایل `dense_weights.h` نیز شامل وزن‌های لایه Fully Connected و فایل `definitions.h` شامل برخی از ضرایب ثابت مورد استفاده در فاز سخت‌افزاری است.
در ادامه تمامی داده‌های تست MNIST شامل تصاویر و Label های مربوط به آن را در دو فایل `in.dat` و `out.dat` ذخیره می‌کنیم. مراحل انجام به طور کامل در فایل `gen_data.ipynb` مشخص شده است.

پاسخ

**فاز سخت‌افزاری:** در این فاز نیاز است کد لایه Fully Connected را نوشته و به به کد کانولوشن قبلی اضاف می‌کنیم و همان ساختار ارائه شده در فاز نرم‌افزاری را در اینجا پیاده‌سازی می‌کنیم.
کد لایه Fully Connected به‌صورت زیر ارائه می‌شود:

**Listing 2: HLS Implementation of Dense Layer**

```
1
2  void dense(hls::stream<float> & flat_to_dense_stream, int filter,
3             hls::stream<float> & dense_to_softmax_stream)
4  {
5      float flat_value;
6      float dense_array[DENSE_SIZE] = { 0 };
7
8      dense_for_flat:
9      for (int i = 0; i < FLAT_SIZE / FILTERS; ++i)
10     {
11         flat_value = flat_to_dense_stream.read();
12
13         for (int d = 0; d < DENSE_SIZE; ++d)
14         {
15             int index = filter * (FLAT_SIZE / FILTERS) + i;
16             dense_array[d] += dense_weights[index][d] * flat_value;
17         }
18     }
19
20     for (int j = 0; j < DENSE_SIZE; ++j)
21     {
22         dense_to_softmax_stream.write(dense_array[j]);
23     }
24 }
```

پاسخ

در این طراحی از ۴ لایه کانولوشن به‌صورت زیر استفاده شده است:

**Listing 3: HLS Implementation of Convolution Layers**

```
1
2  void convolutional_layer(
3      float pad_img0 [PAD_IMG_ROWS][PAD_IMG_COLS],
4      float pad_img1 [PAD_IMG_ROWS][PAD_IMG_COLS],
5      float pad_img2 [PAD_IMG_ROWS][PAD_IMG_COLS],
6      float pad_img3 [PAD_IMG_ROWS][PAD_IMG_COLS],
7      hls::stream<float> conv_to_pool_streams [FILTERS] )
8  {
9      convolution(pad_img0, 0, conv_to_pool_streams[0]);
10     convolution(pad_img1, 1, conv_to_pool_streams[1]);
11     convolution(pad_img2, 2, conv_to_pool_streams[2]);
12     convolution(pad_img3, 3, conv_to_pool_streams[3]);
13 }
14
```

**پاسخ**

همچنین برای Flat کردن Feature Map های به‌دست آمده از خروجی لایه‌های کانولوشنی، ماژولی به‌نام flattening به‌صورت زیر تعریف شده است:

**Listing 4: HLS Implementation of Flatten Layers**

```
1
2  void flattening(hls::stream<float> &  pool_to_flat_stream,
3                  hls::stream<float> & flat_to_dense_stream)
4  {
5      flat_for_rows:
6      for(int r = 0; r < POOL_IMG_ROWS; ++r)
7      {
8          flat_for_cols:
9          for(int c = 0; c < POOL_IMG_COLS; ++c)
10         {
11             flat_to_dense_stream.write(pool_to_flat_stream.read());
12         }
13     }
14 }
```

**پاسخ**

همچنین ماژول‌های دیگری برای انجام عملیات‌هایی مانند Zero Padding، Pooling، Normalization و ... تعریف شده است که به‌علت طولانی نشدن گزارش آورده نشده است. درنهایت تاپ ماژول طراحی به‌نام CNN به‌صورت زیر تعریف شده است:

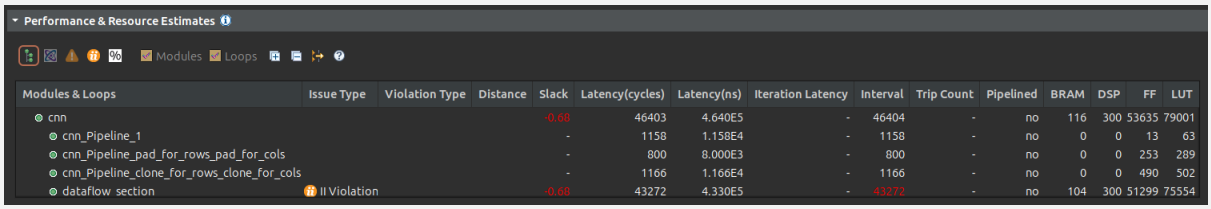
**Listing 5: HLS Implementation of CNN Top Module**

```
1
2  void cnn(float img_in[IMG_ROWS][IMG_COLS], float prediction[DIGITS])
3  {
4      /******** Pre-processing data. ********/
5
6      float pad_img0 [PAD_IMG_ROWS][PAD_IMG_COLS] = { 0 };
7      normalization_and_padding(img_in, pad_img0);
8
9      #if 0
10         #ifndef __SYNTHESIS__
11             printf("Padded image.\n");
12             print_pad_img(pad_img);
13         #endif
14     #endif
15
16     /* Allow parallelism cloning the padded image. */
17     float pad_img1 [PAD_IMG_ROWS][PAD_IMG_COLS];
18     float pad_img2 [PAD_IMG_ROWS][PAD_IMG_COLS];
19     float pad_img3 [PAD_IMG_ROWS][PAD_IMG_COLS];
20
21     float value;
22
23     clone_for_rows:
24     for (int i = 0; i < PAD_IMG_ROWS; ++i)
25         clone_for_cols:
26         for (int j = 0; j < PAD_IMG_COLS; ++j)
27         {
28             pad_img1[i][j] = pad_img0[i][j];
29             pad_img2[i][j] = pad_img0[i][j];
```

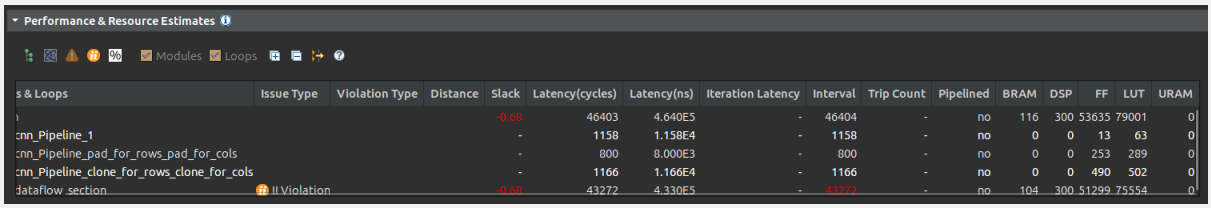
```
30              pad_img3[i][j] = pad_img0[i][j];
31          }
32
33      /* Parallel executions start here. */
34      dataflow_section(pad_img0, pad_img1, pad_img2, pad_img3, prediction);
35  }
```

## پاسخ

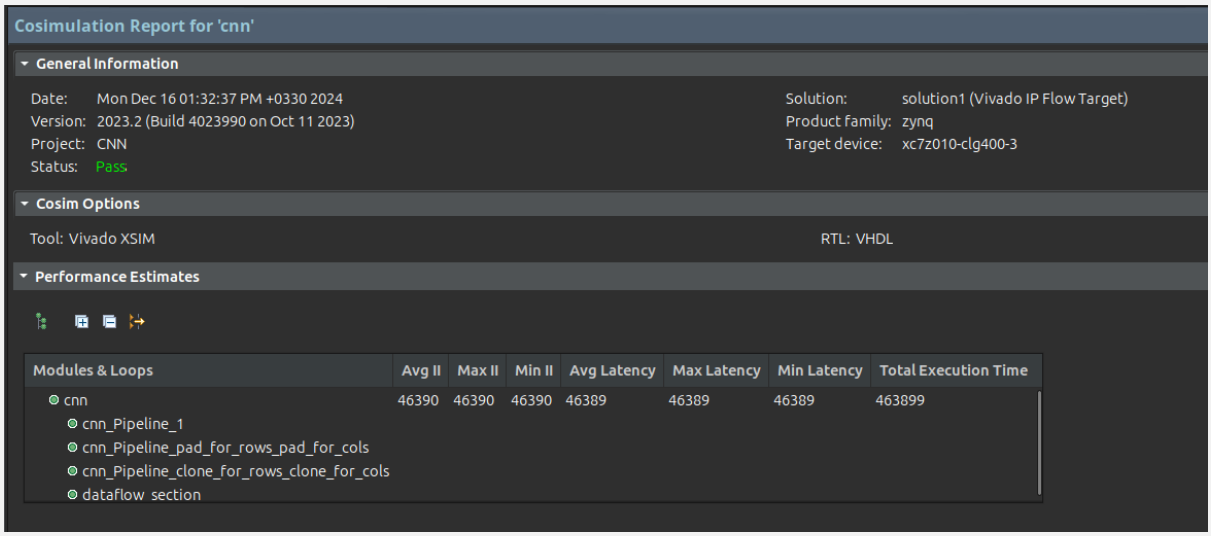
درنهایت ماژول CNN را سنتز می‌کنیم. گزارشات سنتز شبکه به‌صورت زیر ارائه می‌شود:




شکل ۵: منابع مصرفی پس‌از سنتز (الف)




شکل ۶: منابع مصرفی پس‌از سنتز (ب)




شکل ۷: منابع مصرفی پس‌از سنتز (ج)

درنهایت فایل test bench ای برای طراحی نوشته شده است که ۱۰۰ تصویر ابتدایی را از فایل تصاویر تولید شده در فاز قبل را بخواند و برای پردازش به شبکه بدهد. و خروجی شبکه میزان دقت تشخیص اعداد و زمان مصرف شده به‌ازای تمام فرایند می‌باشد.

پاسخ

برای مثال، شبکه ما ۱۰۰ تصویر ابتدایی از مجموعه داده‌های تست دیتاست MNIST را با دقت ٪۱۰۰ تشخیص می‌دهد:

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
make: 'csim.exe' is up to date.

Total predictions: 100
Correct predictions: 100.00 %
Average latency: 4.450780 (ms)

INFO [HLS SIM]: The maximum depth reached by any hls::stream() instance in the design is 784
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

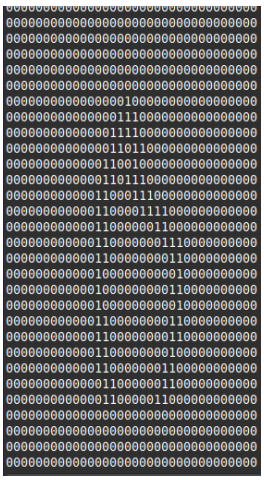
شکل ۸: Accuracy پیش‌بینی به‌ازای ۱۰۰ تصویر

شبکه ما فاز Inference را به ازای ۱۰۰ تصویر در ۴/۴۵ میلی‌ثانیه انجام داده است. درصورتی که در فاز نرم‌افزاری همین تعداد تصویر در مدت زمان ۴۶/۸۰ میلی‌ثانیه انجام شده است.
هرچه تعداد تصاویر را زیاد کنیم دقت شبکه کم می‌شود. برای مثال به ازای ۵۰۰ تصویر دقت تشخیص شبکه ۹۹ درصد به‌دست می‌آید.

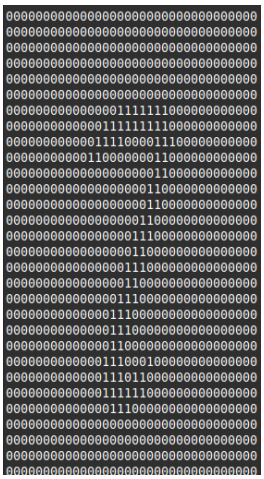
```


Total predictions: 500
Correct predictions: 99.00 %
Average latency: 4.207222 (ms)

INFO [HLS SIM]: The maximum depth reached by any hls::stream() instance in the design is 784
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

شکل ۹: Accuracy پیش‌بینی به‌ازای ۵۰۰ تصویر

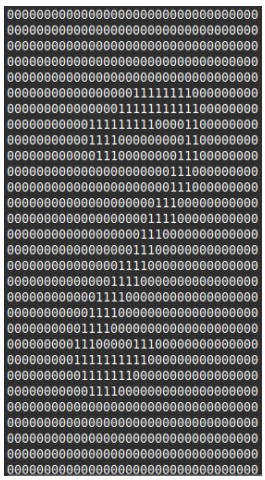
در این حالت شبکه ۵ تصویر زیر را به اشتباه تشخیص داده است:



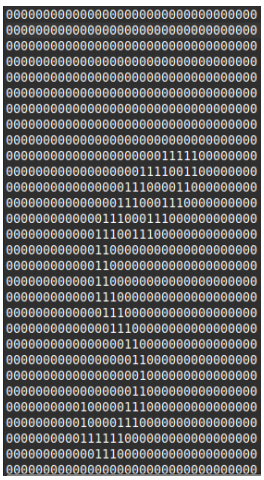
(آ) تشخیص اشتباه عدد ۰



(ب) تشخیص اشتباه عدد ۲



(ج) تشخیص اشتباه عدد ۲



(د) تشخیص اشتباه عدد ۵

(ه) تشخیص اشتباه عدد ۵

شکل ۱۱: تشخیص‌های اشتباه شبکه

**پاسخ**

مقادیری که شبکه برای هر خروجی اشتباه به‌دست آورده است نیز به‌صورت زیر ارائه می‌شود:

```
Prediction for C:
0:  0.268470
1:  0.006770
2:  0.000006
3:  0.000007
4:  0.025127
5:  0.002049
6:  0.006413
7:  0.627097
8:  0.047987
9:  0.016075
```

```
Prediction for B:
0:  0.000006
1:  0.000024
2:  0.078408
3:  0.048098
4:  0.000000
5:  0.000003
6:  0.000000
7:  0.596224
8:  0.271232
9:  0.006006
```

```
Prediction for A:
0:  0.000014
1:  0.000002
2:  0.000000
3:  0.806685
4:  0.001617
5:  0.037086
6:  0.000001
7:  0.000960
8:  0.003560
9:  0.150075
```

پاسخ

```
Prediction for E:
0:  0.000000
1:  0.000000
2:  0.137538
3:  0.003166
4:  0.000000
5:  0.000000
6:  0.000000
7:  0.000023
8:  0.859272
9:  0.000001
```

```
Prediction for D:
0:  0.000006
1:  0.005681
2:  0.000050
3:  0.955743
4:  0.000271
5:  0.037680
6:  0.000232
7:  0.000000
8:  0.000259
9:  0.000077
```