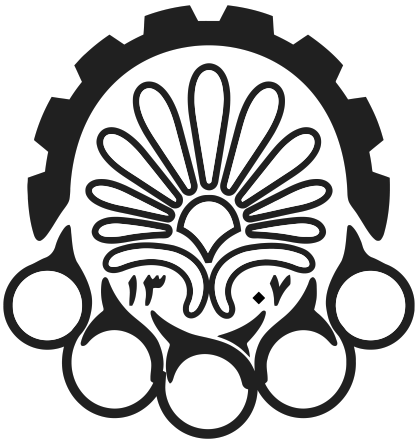
طراحی سیستم‌های قابل بازپیکربندی

دکتر صاحب‌الزمانی



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین سری اول

۸ آبان ۱۴۰۳

طراحی سیستم‌های قابل بازپیکربندی

تمرین سری اول

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

دانشکده مهندسی کامپیوتر

سوال اول

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

۱. در یک پروژه با زمان محدود بهترین راه جهت پیاده‌سازی الگوریتم پردازشی استفاده از تراشه‌های قابل بازپیکربندی است.

پاسخ

نادرست.
زیرا در زمان محدود بهترین راه برای پیاده‌سازی یک الگوریتم پردازشی استفاده از پردازنده‌های مرسوم موجود در بازار مانند CPU است. چون معمولا زمان طراحی و برنامه‌ریزی برای تراشه‌های قابل‌بازپیکربندی مانند FPGA بشتر از CPU های مرسوم است.

۲. طراحی‌های مبتنی بر پردازنده‌های همه منظوره و تراشه‌های خاص منظوره، دو انتهای بردار کارآیی و انعطاف‌پذیری هستند.

پاسخ

نادرست.
اگر ترتیب بیان مهم باشد، این گزاره غلط است و درست آن بدین صورت می‌شود: دو انتهای بردار انعطاف‌پذیری و کارآیی هستند. اما اگر ترتیب مهم نباشد، گزاره درست است.

۳. معماری قابل بازپیکربندی جهت حل مشکل دسترسی حافظه در کامپیوتر فون نیومن ارائه شده است.

پاسخ

نادرست.
این دلیل هم در کنار مصرف انرژی زیاد کامپیوترهای فن نیومن درست است اما دلیل اصلی ارائه معماری بازپیکربندی نزدیک کردن میزان انعطاف پذیری ASIC ها با مصرف انرژی به مراتب کمتر نسبت به کامپیوتر فن نیومن به این نوع کامپیوترها بوده است.

۴. در کاربردهای فضایی و محیط‌های دارای تشعشعات زیاد، تراشه‌های مبتنی بر FLASH بهترین گزینه انتخابی هستند.

پاسخ

نادرست.
تراشه‌های مبتنی بر FLASH در برابر (Single Event Upset (SEU)) ها و (Single Event Latchups (SELs)) های ناشی از تشعشع آسیب پذیرتر هستند و بیشتر دچار BitFlip می‌شوند. در چنین محیط‌هایی بهتر است از تراشه‌های مبتنی بر Anti-Fuse استفاده نمود (البته اگر شرط بازپیکره‌بندی بودن برایمان مطرح نباشد).

ادامه پاسخ ۴

آنتی فیوزها به دلیل ساختار فیکس شده‌شان در برابر تشعشعات و Soft Error ها مقاوم تر هستند.

۵. از تراشه‌های مبتنی بر آنتی‌فیوز به دلیل مقاومت مناسب در برابر دمای بالا در کاربردهای صنعتی استفاده می‌شود.

پاسخ

درست.
تراشه‌های مبتنی بر آنتی‌فیوز به دلیل معماری ای که دارند، در برابر شرایط سخت، از جمله دمای بالا، مقاومت بهتری دارند. این تراشه‌ها به دلیل ماهیت فیزیکی فرآیند آنتی‌فیوز که شامل ایجاد یک اتصال دائم و غیرقابل تغییر است، در برابر تغییرات محیطی مانند دما یا تشعشعات نسبت به سایر تکنولوژی‌ها پایدارتر هستند.
البته این سوال با این فرض درست است که در آن کاربرد صنعتی مورد استفاده نیازی به بازپیکره‌بندی نداشته باشیم.

۶. تراشه‌های CGRA با دارابودن واحدهای خاص منظوره بیشتر، توان کمتری نسبت به FPGAها دارند.

پاسخ

درست.
CGRA ها به دلیل Granularity بزرگتر، معمولاً شامل واحدهای پردازشی بزرگ‌تر و خاص‌منظوره‌تر هستند که می‌توانند برای انجام وظایف خاص بازپیکره‌بندی شوند. اما یکی از مزایای هاCGRA نسبت به هاFPGA این است که مصرف توان کمتری دارند، زیرا این واحدها برای انجام وظایف مشخص بهینه شده‌اند و نیازی به بازپیکره‌بندی در سطح بسیار ریز ((Boolean level (Fine Grain ندارند.

۷. استفاده از FPGAها در مقایسه با تولید یک تراشه خاص باعث کاهش هزینه تولید محصول خواهد شد.

پاسخ

نادرست.
بستگی به مقدار Cross-over volume دارد. اگر ساخت تعداد زیادی آیسی مدنظر باشد، هزینه‌های ساخت ASIC در تیراژ بالا کمتر از FPGA در می‌آید. اما اگر ساخت یک آیسی جهت نمونه مد نظر باشد، گزاره مطرح شده درست است و هزینه ساخت آن با FPGA کمتر است.

۸. یک ASIC همواره سریع‌تر از یک FPGA دستورات پردازشی سطح بالا را انجام خواهد داد.

پاسخ

درست.
FPGA ها به دلیل ساختار Reconfigurable ای که دارند، برای آنکه بتوانند پیاده‌سازی طیف وسیع‌تری از الگوریتم ها و کاربردها را پوشش دهند، از سرعت پردازش کمتری نسبت به ASIC ها که به‌طور ویژه و خاص برای انجام یک کار مشخص به‌صورت Un-Recunfigurable دیزاین شده اند دارند.

۹. افزایش تعداد ورودی یک LUT همواره باعث افزایش سرعت مدار پیاده‌سازی شده با استفاده از آن خواهد شد.

پاسخ

نادرست.
تاخیر کل FPGA به‌عنوان تابعی از اندازه LUT ها معرفی می‌شود. با افزایش تعداد ورودی‌های LUT ها، تعداد حالات پیاده سازی یک Logic یکسان زیاد می‌شود و احتمال Placement سخت‌تر می‌شود. بنابر این تاخیر همواره بیشتر و درنتیجه سرعت کمتر می‌شود.

۱۰. بلوک‌های UltraRAM درکنار بلوک‌های DSP برای پیاده‌سازی الگوریتم‌های هوش مصنوعی به کمک FPGA خانواده Zynq بسیار مناسب هستند.

پاسخ

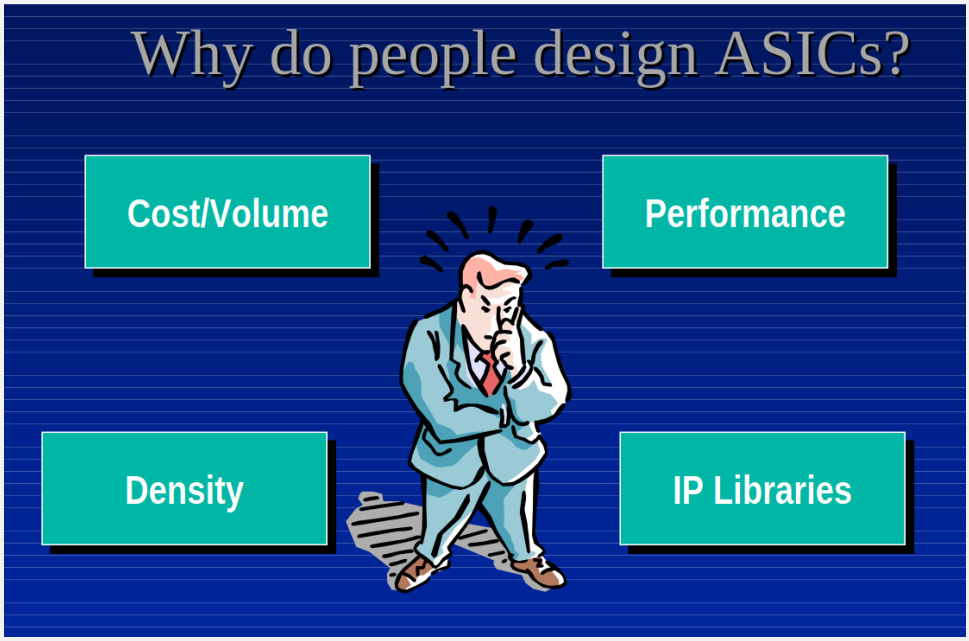
درست.
بلوک‌های UltraRAM به عنوان حافظه‌هایی با ظرفیت بالا و دسترسی سریع در FPGA های خانواده Zynq عمل می‌کنند که می‌توانند حجم زیادی از داده‌ها و وزن‌ها را به سرعت خوانده و برای پردازش توسط بلوک‌های DSP آماده کنند. UltraRAM ها با ارائه حافظه ای با ظرفیت زیاد و تأخیر کم، نقش کلیدی در ذخیره‌سازی و دسترسی سریع به داده‌های مورد نیاز الگوریتم‌های یادگیری ماشین و شبکه‌های عصبی ایفا می‌کند. همچنین بلوک‌های DSP نیز برای انجام عملیات های محاسباتی پیچیده مثل ضرب و جمع که در الگوریتم‌های هوش مصنوعی به وفور استفاده می‌شوند، بهینه شده‌اند. بنابر این درکنار یک حافظه سریع برای انجام محاسبات بسیار مناسب هستند.

سوال دوم

در یک سیستم ایمنی مرتبط با خودرو نیاز به طراحی یک سیستم ایمنی با قابلیت اطمینان بالا می‌باشد که بایستی دارای امکان به‌روزرسانی الگوریتم ایمنی نیز باشد. همچنین زمان عملکرد سیستم نیز بایستی به صورت Hard Real-time باشد. برای طراحی این سیستم در صورت نمونه‌سازی و در صورتی که ۱ میلیون نسخه از آن نیاز باشد استفاده از چه نوع بستر پردازشی را پیشنهاد می‌نمایید؟ برای انجام محاسبات، هزینه‌های مربوط به ساخت معماری پیشنهادهای خود را از اینترنت استخراج نمایید.

پاسخ

همیشه یکی از مهمترین پاسخ‌ها در ابتدای هر طراحی انتخاب پلتفرم برای آن است. به طوری که آقای Rajeev Jayaraman در [۱] توضیحات مفصلی در این مورد می‌دهد که مطابق با این سوال از برخی از پاسخ‌های ایشان استفاده می‌کنیم.



شکل ۱: FPGA یا ASIC ؟ مسئله این است.

در ابتدا باید این مورد را عنوان کرد که انتخاب پلتفرم برای طراحی به معیار های زیادی بستگی دارد که این سوال به چند مورد از آن‌ها یعنی قابلیت به‌روزرسانی الگوریتم (بازپیکربندی)، تیراژ ساخت، سرعت بالا (Real-Time) اشاره کرده است. انتخاب پلتفرم طراحی به همین موارد بسنده نمی‌کند و به موارد دیگری همچون میزان سرعت پردازش مورد نیاز (میزان موازی سازی) برای آن اپلیکیشن خاص، توان مصرفی نیز نیاز است توجه کنیم.
با توجه به موارد گفته شده و فرضیات محدود مسئله، پاسخ بسیار دقیقی را نمی‌توان برای این سوال مطرح کرد اما با درنظر گرفتن یک‌سری فرضیات آن‌را تحلیل می‌کنیم.
از آنجایی که می‌خواهیم عملکرد سیستم به‌صورت Hard Real-Time باشد، به پردازنده‌ای نیاز داریم که یا به‌صورت FPGA ها قابلیت موازی سازی بالایی داشته باشد یا مانند CPU ها از فرکانس کلاک بالایی برخوردار باشد تا به کاربر احساس عملکرد Real-Time بودن را بدهد.
موضوع دومی که مورد توجه ما قرار می‌گیرد بحث توان مصرفی است. چون کاربرد ما خودرو است نمی‌توانیم از GPU استفاده کنیم (به دلیل توان مصرفی زیاد) می بایست از تراشه های Low Power استفاده نماییم. چون نیاز داریم که بتوانیم الگوریتممان را در زمان‌های مختلف آپدید کنیم، می‌بایست از تراشه ای استفاده کنیم که بتوان این قابلیت را برای ما فراهم کند.

پاسخ

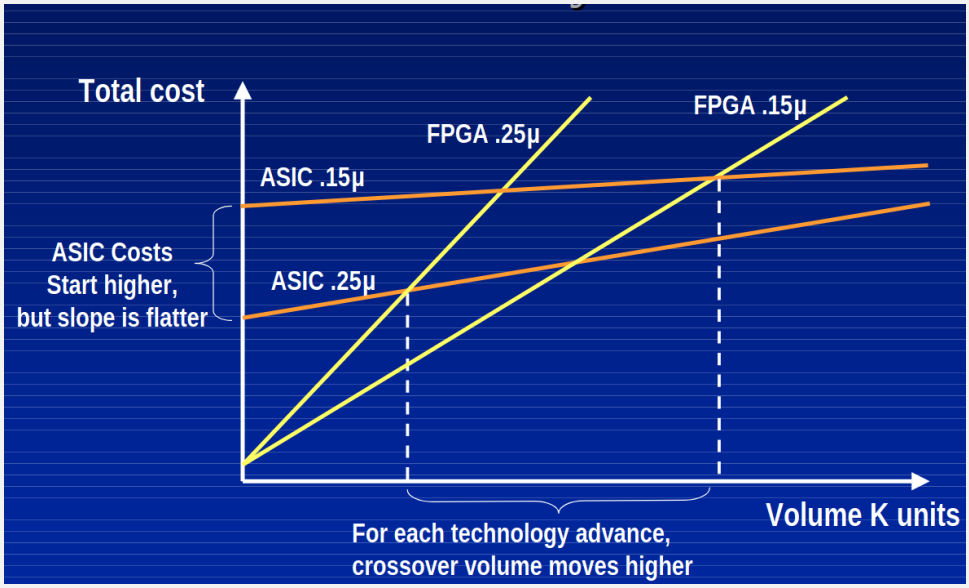
بسته به نوع الگوریتمی که قرار است پیاده سازی شود می‌توان بین CPU و FPGA انتخاب نمود. اگر به موازی‌سازی های بالا در اپلیکیشنمان نیاز داشته باشیم انتخاب ما باید FPGA باشد، درهیر این صورت می‌توان از CPU های مرسوم نیز استفاده نمود.

موضوع بعد بحث هزینه است. بر اساس آنکه قرار است از این تراشه به تعداد یک میلیون قطعه ساخته شود شاید به‌نظر برسد که FPGA به‌صرفه نباشد و هزینه آن بسیار گران بشود. اما باید دید که آیا در این Trade-Off بین هزینه و سرعت پردازش بالا کدام یک بیشتر به نفع ما و کاربرد ماست.

در مقابل این‌ها ASIC ها قرار می‌گیرد که از نظر توان پردازشی، توان مصرفی و سرعت بسار خوب هستند و در تیراژ بالا بسیار ارزان تر از FPGA ها و CPU ها درمی‌آید اما قابلیت آپدیت الگوریتم در آن‌ها وجود ندارد.

مگر آنکه در زمان ساخت از شرکت سازنده درخواست کنیم که یک واحد پردازشی Reconfigurable به‌صورت On Chip درون ASIC ما بگذارند. اینطوری هم در قیمت برای ما به‌صرفه است و هم قابلیت آپدیت کردن الگوریتم را برای ما فراهم می‌کند.

طبق گفته Rajeev Jayaraman نمودار تحلیل هزینه‌های ASIC در مقایسه با FPGA به شکل زیر است.



شکل ۲: نمودار هزینه‌های FPGA و ASIC

مقادیر هزینه و واحدها از نمودار حذف شده‌اند زیرا این مقادیر بسته به فناوری پردازش استفاده شده و با گذشت زمان متفاوت هستند. ASICها دارای هزینه‌های مهندسی غیرقابل تکرار (NRE) بسیار بالایی هستند که ممکن است به میلیون‌ها دلار برسند، در حالی که هزینه واقعی هر تراشه ممکن است تنها چند سنت باشد. در مورد FPGAها، هیچ هزینه NRE وجود ندارد. ما فقط هزینه تراشه FPGA را پرداخت می‌کنیم و پولی هم بابت استفاده از نرم‌افزارهای مربوطه آن نمی‌پردازیم :))) بنابراین، هزینه کل برای ASICها به دلیل هزینه‌های NRE بسیار بالا شروع می‌شود، اما شیب آن کمتر است. به این معنی که نمونه‌سازی ASICها در مقادیر کم بسیار پرهزینه است، اما در حجم‌های بالا، هزینه هر واحد بسیار کاهش می‌یابد. در مورد FPGAها، هزینه تراشه نسبتاً بالاتر است، بنابراین در حجم‌های زیاد، نسبت به ASICها هزینه بیشتری دارد.

بنابر این می‌توان محاسبات تخمینی زیر را نیز برای یک طراحی مشابه بر روی FPGA و ASIC انجام داد.

پاسخ

۱. **برای FPGA:**

- فرض شود یک FPGA به قیمت ۵۰ دلار برای هر واحد داریم و قصد تولید ۱ میلیون نسخه را داریم
- هزینه کل = تعداد نسخه‌ها × هزینه هرواحد
- هزینه کل = $50 \times 1000000 = 50000000$
- هزینه NRE = صفر

۲. **برای ASIC:**

- فرض شود هزینه NRE برای ASIC دو میلیون دلار باشد و هزینه‌ی تولید هر واحد ASIC پس از پرداخت هزینه‌های NRE، ۵ دلار باشد.
- هزینه کل = هزینه NRE + (تعداد نسخه‌ها × هزینه هرواحد)
- هزینه کل = $2000000 + (5 \times 1000000) = 7000000$

بنابر این برای ۱ میلیون نسخه، هزینه FPGA حدود ۷ برابر بیشتر از هزینه تمام شده ASIC است.

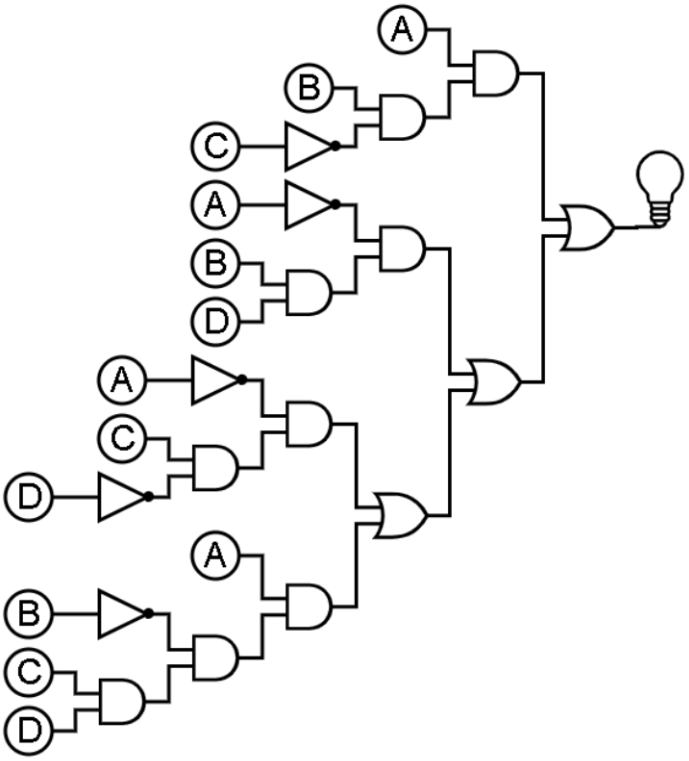
*

References

[1] Rajeev Jayaraman, Xilinx Inc, 2001 https://www.doc.ic.ac.uk/ wl/teachlocal/arch/killa-sic.pdf

سوال سوم

می‌خواهیم مدار زیر را یک بار با LUTهای ۳ ورودی و بار دیگر با LUTهای ۴ ورودی پیاده‌سازی کنیم به طوری که در هر حالت تعداد LUTهای مورد استفاده کمینه باشد.



شکل ۳: مدار مورد نظر

پاسخ

تابع بولی خروجی به‌صورت زیر محاسبه می‌شود:

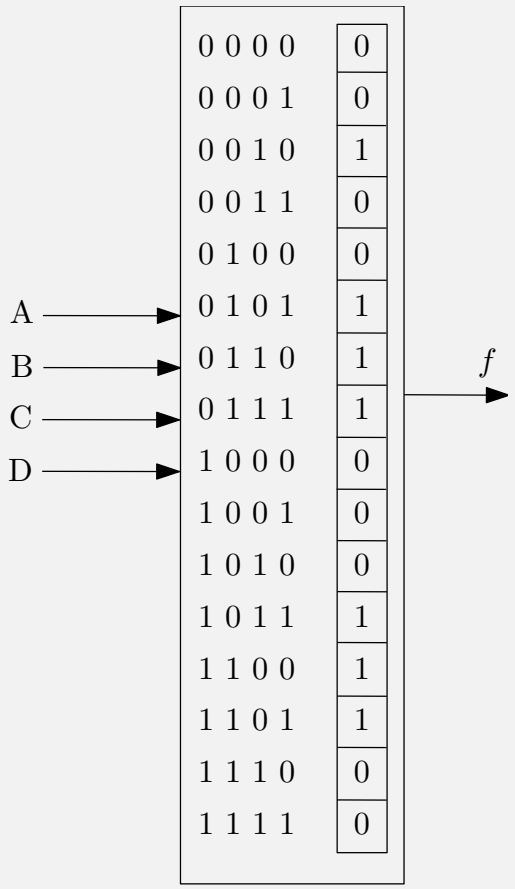
$$f = (A'CD') + (AB'CD) + (A'BD) + (ABC')$$

همچنین جدول درستی این تابع نیز به‌صورت زیر محاسبه می‌شود:

پاسخ

A	B	C	D	$A' \cdot C \cdot D'$	$A \cdot B' \cdot C \cdot D$	$A' \cdot B \cdot D$	$A \cdot B \cdot C'$	f
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0	1
0	0	1	1	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	1	0	0	0	1
0	1	1	1	0	0	1	0	1
1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	0	1	1	0	1	0	0	1
1	1	0	0	0	0	0	1	1
1	1	0	1	0	0	0	1	1
1	1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0

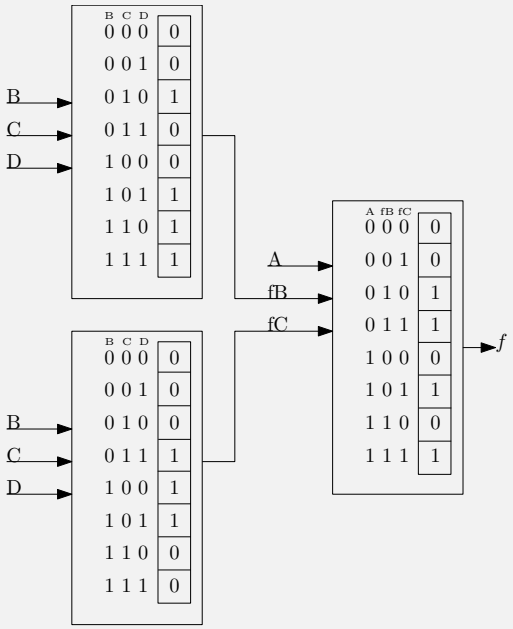
از آنجایی که تابع ۴ ورودی است، برای پیاده‌سازی آن با استفاده از LUT، به یک LUT، ۴ ورودی نیاز داریم. مقادیر خروجی f در سلول‌های SRAM ذخیره می‌شوند و به ازای ورودی‌های مختلف، خروجی های متناظر با آن ورودی را مطابق با جدول درستی نوشته شده می‌دهند. مدار طراحی شده به‌صورت زیر است:



شکل ۴: تابع با LUT، ۴ ورودی

پاسخ

برای طراحی همین تابع با استفاده از LUT های ۳ ورودی، جدول درستی را از وسط نصف می‌کنیم و خروجی های متناظر با ورودی‌های BCD را به ورودی یک LUT، ۳ ورودی می‌دهیم. مطابق با طراحی زیر.



شکل ۵: تابع با LUT های ۳ ورودی

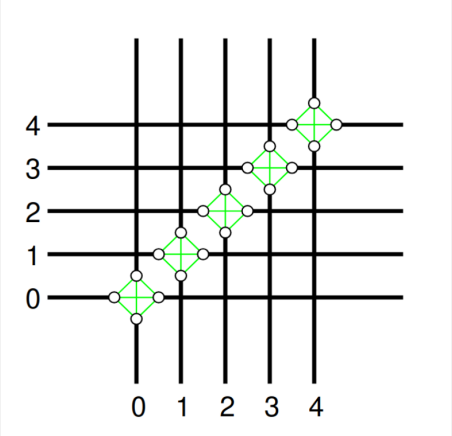
همچنین می‌توان به‌جای LUT آخر، از یک MUX دو ورودی استفاده نمود که خط Select آن به A متصل است.

سوال چهارم

معماری سوئیچ‌های Wilton و Disjoint را توضیح داده و میزان F_s را در هر یک گزارش نمایید. آیا معماری دیگری برای اتصال سوئیچ‌ها می‌شناسید؟

پاسخ

۱. **معماری Disjoint**: در این معماری، اگر اتصالات را شماره گذاری کنیم، فقط آن‌هایی که شماره همنام دارند، مجاز به متصل شدن به یک‌دیگر هستند. برای مثال مطابق با شکل زیر، مسیر شماره صفر فقط می‌تواند به مسیر هایی با همین شماره متصل شود.

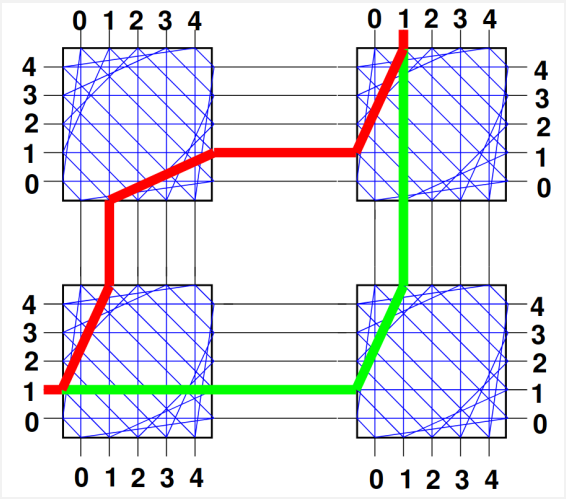


شکل ۶: معماری سوئیچ Disjoint

این معماری انعطاف پذیری مسیریابی را کاهش می‌دهد. و میزان انعطاف‌پذیری این سوئیچ بلاک ها $F_s = 3$ است.

یعنی هر Wire ورودی به سوئیچ بلاک فقط به ۳ Wire هم شماره خودش می‌تواند متصل شود.

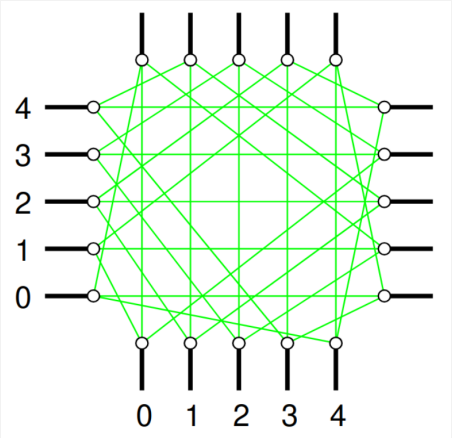
برای مثال می‌توان نحوه اتصال دو Connection Block به یک‌دیگر را با معماری Disjoint به‌صورت زیر نمایش داد:



شکل ۷: تصال دو Connection Block به یک‌دیگر را با معماری Disjoint

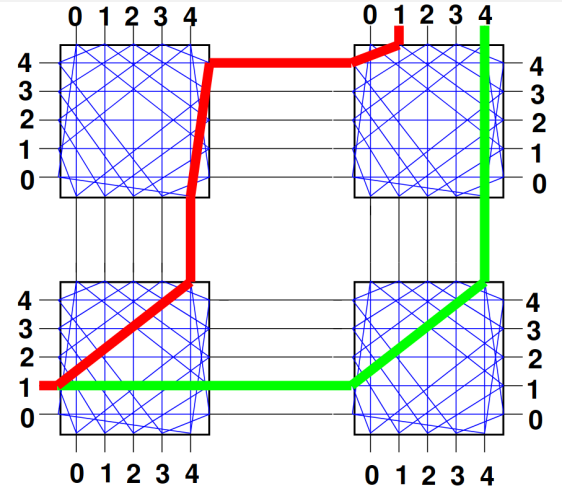
پاسخ

۲. **معماری Wilton**: در معماری Wilton برخلاف معماری Disjoint مقدار انعطاف پذیری در مسیریابی با تغییر ساختار اتصالات بهبود یافته است. در تین نوع معماری، سیم‌ها می‌توانند با یک الگوی مشخص به سیم‌های ناهمنام خود نیز متصل شوند. اما در این معماری نیز همانند معماری قبل $F_s = 3$ است.



شکل ۸: معماری سوئیچ Wilton

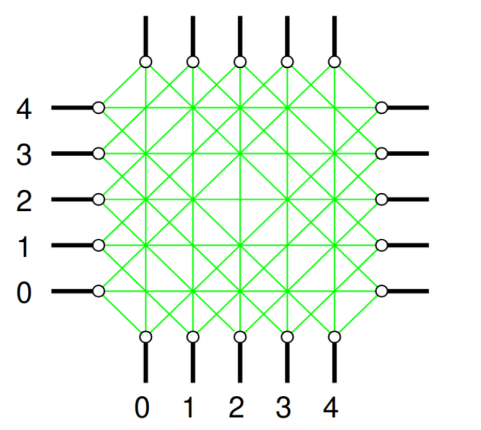
برای مثال می‌توان نحوه اتصال دو Connection Block به یک‌دیگر را با معماری Wilton به‌صورت زیر نمایش داد:



شکل ۹: تصال دو Connection Block به یک‌دیگر را با معماری Wilton

درکنار این دو معماری، معماری‌های مختلف دیگری معرفی شده است که یکی از معروف ترین آن‌ها معماری Universal است.
در این معماری که شکل آن به‌صورت زیر است، انعطاف پذیری در اتصالات باهم بیشتر شده است. در این معماری نیز $F_s = 3$ است.

پاسخ



شکل ۱۰: معماری سوئیچ Universal

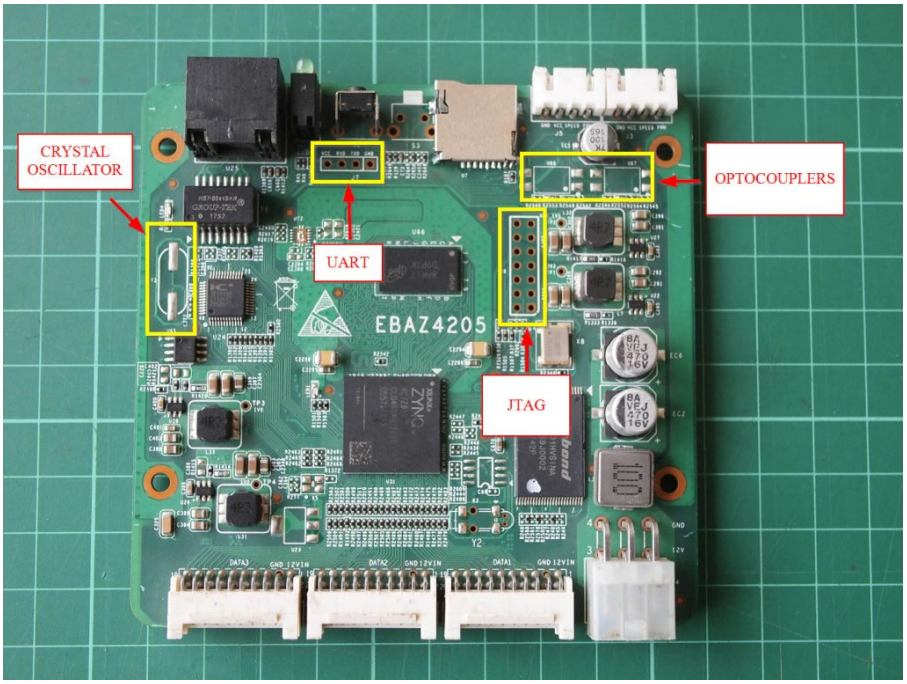
سوال پنجم

آشنایی اولیه با ابزار ویوادو: در این درس دانشجویان با استفاده از ابزار ویوادو از شرکت زایلینکس به انجام پروژه‌ها خواهند پرداخت. هدف از انجام پروژه‌ها، آشنایی عملی با طراحی توأم بر روی سیستم‌های قابل بازپیکربندی است. برای این منظور در این بخش در ابتدا دانشجویان می‌بایست نرم‌افزار ویوادو را بر روی سیستم خود نصب کنند. سپس با بررسی لینک زیر در ارتباط با نحوه طراحی توأمان و نحوه کار با ابزار آشنایی لازم را کسب کرده و توضیحات موردنیاز را در ارتباط با این نوع طراحی ارائه دهند.

- Link (I)
- Link (II)

پروژه مشابه موارد یاد شده در دو ویدئو نیز بایستی به همراه پاسخ تمرین‌ها بارگذاری شود. جهت دانلود نرم‌افزار ویوادو از این لینک استفاده نمایید. نسخه پیشنهادی ۲۰۲۰.۲ به بعد می‌باشد. به دلیل مشکل احتمالی در فعال‌ساز بهتر است از نسخه ۲۰۲۴ استفاده نشود.

همانطور که در ویدئو نیز بیان شد، هدف در این قسمت، طراحی Co-Design است. بدین منظور، برای طراحی یک گیت NAND ساده، گیت AND را با استفاده از Logic Block های FPGA طراحی می‌کنیم و ماژول NOT را در هسته پردازشی یعنی CPU طراحی کرده و اتصالات بین این دو طراحی را برقرار می‌کنیم.
ذکر این نکته الزامی است که در این تمرین ما از بورد EBAZ4205 که تراشه موجود بر روی آن Zynq 7000 است استفاده نمودیم. این آیسی در نرم‌افزار Vivado با پارت‌نامبر xc7z010clg400-3 شناخته می‌شود. تصویر این برد در «شکل ۱۱» آورده شده است:



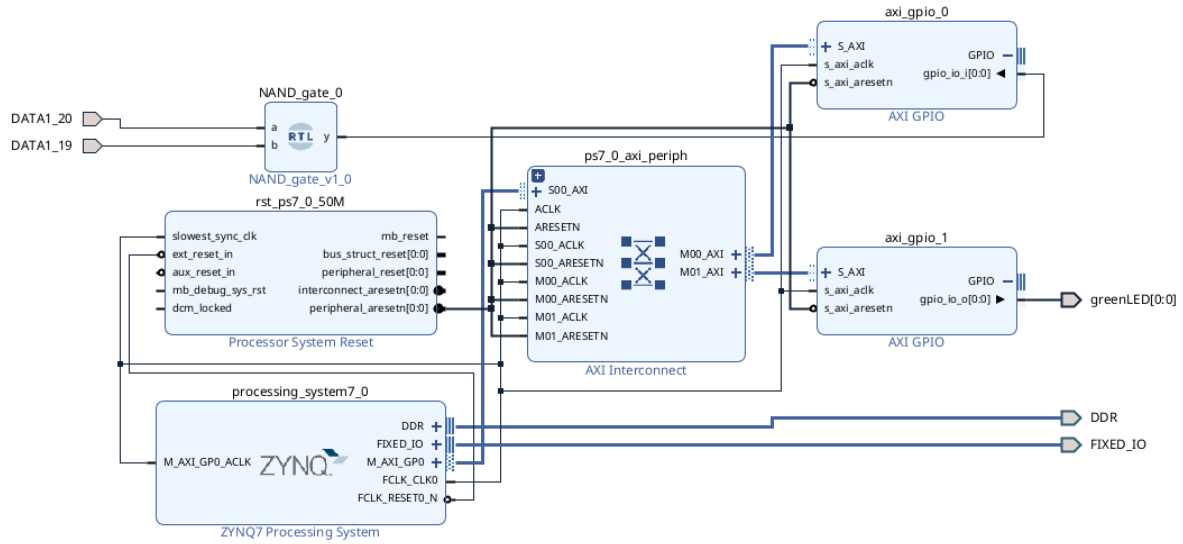
شکل ۱۱: بورد مورد استفاده در این تمرین

همچنین فایل Constrain مربوط به این بورد را می‌توان از اینجا دانلود کرد.
در ابتدا پس از نصب نرم‌افزار و انتخاب آیسی، طراحی سمت PL را انجام می‌دهیم. در این قسمت صرفا یک گیت AND را طراحی می‌کنیم. کد نوشته شده برای ماژول AND به‌صورت زیر است:

Listing 1: AND Module for PL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity NAND_gate is
    Port ( a, b: in std_logic;
           y: out std_logic );
end NAND_gate;

architecture Behavioral of NAND_gate is

begin

    y <= a and b;

end Behavioral;
```

پس از طراحی ماژول AND می‌بایست نحوه Interconnection سمت PL و PS را در درون تراشه برقرار کنیم.
بدین منظور از قسمت طراحی دیاگرامی Vivado پردازنده ZYNQ را انتخاب می‌کنیم و همچنین از ماژول AND خودمان نیز یک بلوک می‌سازیم.
با استفاده از بلوک‌های AXI GPIO می‌توانیم ارتباطات بین PL و PS را برقرار کنیم.
درنهایت سیستم طراحی شده به‌صورت «شکل ۱۲» می‌شود.



شکل ۱۲: طراحی PL و PS انجام شده

پس از تکمیل شدن طراحی، می‌بایست گیت NOT را نیز به‌صورت نرم‌افزاری (به زبان c) طراحی کنیم و سپس طراحی را سنتز نهایی کنیم.
برای انجام این کار، نرم‌افزار Vitis Clasic را اجرا می‌کنیم و یک Application Project جدید می‌سازیم و فایل xsa. ساخته شده در مرحله قبل را به این پروژه اضافه می‌کنیم.
پس از ایجاد پروژه کد گیت NOT را به‌صورت زیر می‌نویسیم:

Listing 2: Not Module for PS

```c
#include <stdio.h>
#include "platform.h"
#include "xgpio.h"
#include "xparameters.h"
#include "xil_printf.h"
```

```
6
7  int main()
8  {
9      init_platform();
10
11     XGpio input, output;
12     int a;
13     int y;
14
15     XGpio_Initialize(&input, XPAR_AXI_GPIO_0_DEVICE_ID);
16     XGpio_Initialize(&output, XPAR_AXI_GPIO_1_DEVICE_ID);
17
18     XGpio_SetDataDirection(&input, 1, 1);
19     XGpio_SetDataDirection(&output, 1, 0);
20
21     print("debug the code");
22
23     while(1)
24     {
25         a = XGpio_DiscreteRead(&input, 1);
26
27         if(a == 1)
28         {
29             y = 0;
30         }
31         else
32         {
33             y = 1;
34         }
35
36         XGpio_DiscreteWrite(&output, 1, y);
37     }
38
39     cleanup_platform();
40     return 0;
41 }
```

سپس بعد از سنتز کد، با قرار دادن فایل Bitstream ایجاد شده در مرحله PL به عنوان فایل پروگرام، طراحی Integrate شده را بر روی بردمان پروگرم می‌کنیم.