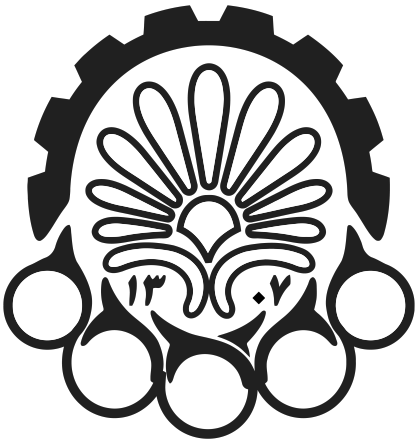
طراحی سیستم‌های قابل بازپیکربندی

دکتر صاحب‌الزمانی



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین سری دوم

۲۴ آبان ۱۴۰۳

طراحی سیستم‌های قابل بازپیکربندی

تمرین سری دوم

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

دانشکده مهندسی کامپیوتر

سوال اول

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

۱. خانواده Cyclone نسبت به Stratix مصرف انرژی کمتری دارد.

پاسخ

درست.
تراشه‌های خانواده Cyclone برای کاربردهایی طراحی شده که نیاز به مصرف انرژی کمتر و هزینه پایین‌تری دارند. از طرف دیگر تراشه‌های خانواده Stratix برای کاربردهای پیشرفته‌تر و پیچیده‌تر مانند پردازش سیگنال دیجیتال، پردازش داده‌های سنگین طراحی شده است که توان محاسباتی بیشتری را برای انجام می‌طلبد و درنتیجه انرژی مصرفی آن نیز بیشتر است.

۲. معماری کلی تراشه‌های برنامه‌پذیر از تولیدکننده‌ای به تولیدکننده دیگر کاملاً متفاوت است.

پاسخ

نادرست.
معماری کلی ساخت تراشه‌های برنامه‌پذیر در شرکت‌های مختلف به‌طور کامل نسبت به دیگری متفاوت نیست. شرکت‌های مختلف بخش‌های کلی تراشه‌را که عمدتا شامل بلوک‌های منطقی و سوئیچ‌ها می‌باشند را (تقریبا) ثابت و مشابه نگه می‌دارند.

۳. مدل‌های Cyclone تولیدی شرکت Intel دارای هسته پردازنده ARM هستند.

پاسخ

نادرست.
تراشه‌های این خانواده دارای هسته پردازشی داخلی نیستند.

۴. بلوک‌های منطقی قابل پیکربندی (CLB) در خانواده اسپارتان دارای sliceهای مشابه هستند.

پاسخ

درست.
در این خانواده Sliceها شامل منابعی مانند LUT و فلیپ‌فلاپ ها هستند که ساختار مشابهی با هم دارند. البته اگر تعداد LUT های هر Slice را به‌عنوان جزئی متمایز کننده درنظر نگیریم.

۵. برای ارتباط دو سیستم مبتنی بر اسپارتان LX25 با سرعت بالا می‌توان از رابط Gigabyte استفاده کرد.

پاسخ

نادرست.
در FPGA های سری Spartan LX، از جمله Spartan-3 LX25، رابط‌های با سرعت بالا مانند Gigabit Ethernet به صورت داخلی وجود ندارند. این سری از FPGA ها به طور خاص برای کاربردهای کم‌هزینه و با پیچیدگی پایین طراحی شده‌اند و معمولاً برای ارتباطات با سرعت بالا مناسب نیستند، زیرا فاقد Transceiver های پرسرعت هستند.

۶. بلوک URAM در اسپارتان قابل پیکربندی به صورت دسترسی تک کاناله و دوکاناله است.

پاسخ

نادرست.
در خانواده اسپارتان، بلوک URAM ای وجود ندارد. این بلوک‌هادر خانواده‌های پیشرفته‌ای مانند UltraScale و +UltraScale وجود دارد.

۷. خانواده Artix-7 دارای بیش از ۷۰۰ ضرب‌کننده سخت‌افزاری است.

پاسخ

درست.
FPGA های موجود در این خانواده‌همگی دارای حدودا ۷۰۰۰ (و بیشتر) ضرب‌کننده سخت‌افزاری هستند.

۸. بلوک‌های MLAB در Cyclone برای پیاده‌سازی FIFO مناسب نیست.

پاسخ

درست.
این بلوک‌ها به طور خاص برای پیاده‌سازی ساختارهای حافظه با تأخیر کم و توان مصرفی پایین طراحی شده‌اند و می‌توانند در ساختارهای FIFO از آن‌ها استفاده نمود اما بستگی به اندازه FIFO نیز دارد. چرا که برای FIFOهایی با اندازه بزرگ معمولا از بلوک‌های M9K و M10K استفاده می‌شود.

۹. معماری FPGAها برای داده‌های پردازشی با سایز مختلف مناسب نیست و برای این منظور GPUها کاربرد بیشتری دارند.

پاسخ

درست.
در FPGA ها معمولا چون قرار است به ازای یک پردازش و محاسبه خاص، واحد سخت‌افزاری طراحی شود، هرچقدر که اندازه ها فیکس باشد از نظر سرعت پردازش و توان مصرفی بهتر عمل می‌کند. اما GPU ها ساخته‌شده اند تا محاسبات برداری را به‌صورت موازی انجام بدهند بنابراین برای کاربردهایی که نیاز به تغییر سایز داده و موازی‌سازی بسیار بالایی دارند (مانند شبکه‌های عصبی عمیق یا پردازش تصویر پیچیده)، GPUها معمولاً کاربرد بیشتری دارند.

۱۰. در Stratix 10 از معماری LUT قابل شکستن استفاده شده است که قادر به تامین دو LUT با ۳ ورودی و یک LUT با ۴ ورودی با ورودی‌های مستقل هستند.

پاسخ

درست.
در خانواده Stratix 10 از معماری Adaptive LUT استفاده شده است.

سوال دوم

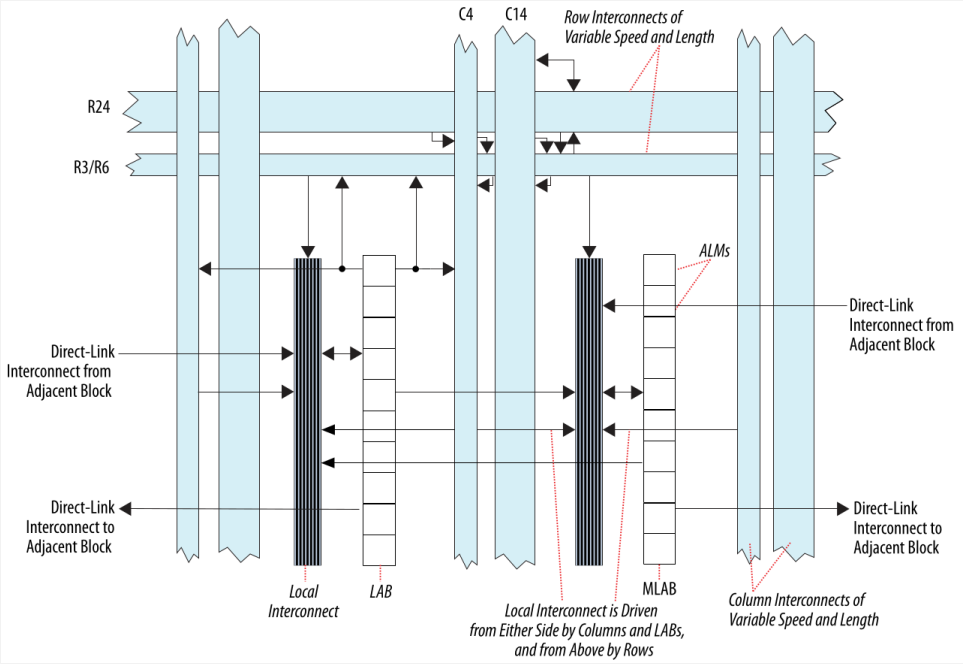
تفاوت‌های اصلی بین خانواده‌های Cyclone و Stratix را توضیح دهید و ذکر کنید در چه شرایطی استفاده از هر کدام مناسب‌تر است؟ همین مقایسه را در خصوص خانواده Stratix و Virtex نیز انجام دهید. موارد را در داخل مدارک فنی شرکت‌های تولیدکننده مشخص کرده و محل آنها را در گزارش خود بیاورید.

پاسخ

عمده تفاوت بین این دو خانواده، جامعه و کارد مورد استفاده از آن‌هاست که آن هم بده‌دلیل ویژگی‌های خاص هریک از این دو تراشه است. برای مثال تراشه‌های خانواده Cyclone برای کاربردهای کم‌هزینه و با مصرف انرژی پایین طراحی شده است. به دلیل طراحی کم‌هزینه و اندازه کوچک‌تر، بیشتر در کاربردهای تجاری و صنعتی با پیچیدگی نه‌چندان زیاد مانند کنترل‌کننده‌ها، سیستم‌های ساده شبکه و دستگاه‌های مصرفی استفاده می‌شود. اما در مقابل تراشه‌های خانواده Stratix از نظر عملکرد و منابع، در سطح بالاتری نسبت به Cyclone قرار دارد و برای کاربردهای پیچیده و توان پردازشی بالا مناسب است. از Stratix بیشتر در کاربردهای پیشرفته مانند پردازش سیگنال دیجیتال، شبکه‌های مخابراتی سرعت‌بالا، محاسبات سنگین، و کاربردهای هوافضا و نظامی استفاده می‌شود.

در ادامه به‌طور دقیق با ارجاع به دیتاشیت‌های شرکت‌های سازنده این دو خانواده از تراشه‌ها را مقایسه می‌کنیم.

شرکت Intel (Altera سابق) اسم بلوک منطقی و قابل برنامه‌ریزی اش را LAB (Logic Array Block) نام‌گذاری کرده است. ساختار ارتباطات بین این بلوک‌ها برای تراشه های هردوخانواده یکسان و به‌صورت زیر است: (صفحه ۱۰ در [۱] و [۲])

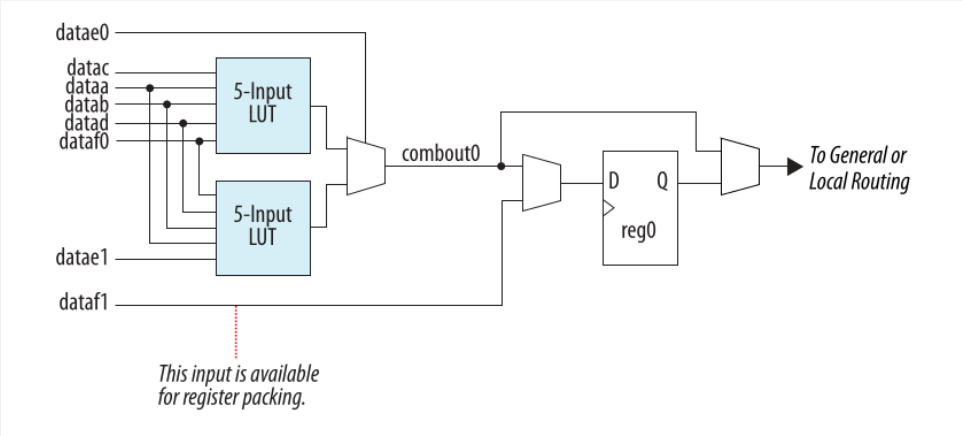


شکل ۱: ساختار ارتباطات میان LAB ها

ساختار LUT ها در این دو خانواده متفاوت هست. در Cyclone از LUT های ۵ ورودی استفاده شده اما در Stratix از LUT های ۶ ورودی. (صفحه ۱۶ در [۱] و صفحه ۱۸ در [۲])

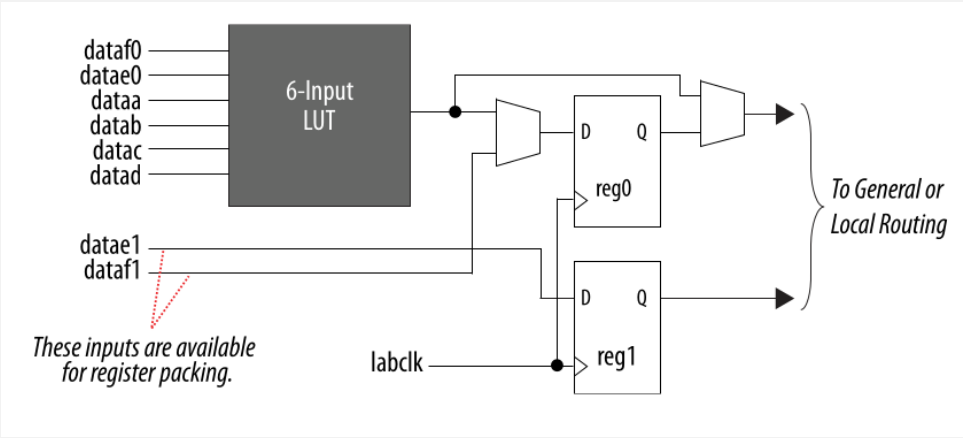
پاسخ

برای Cyclone به‌صورت زیر:



شکل ۲: ساختار LUT ها در Cyclone

و برای Stratix نیز به‌صورت زیر:



شکل ۳: ساختار LUT ها در Stratix

از منظر Memory این این دو خانواده به شدت متفاوت از یکدیگر هستند. در Stratix بلوک‌های پرسرعت حافظه به‌نام M20K داریم درصورتی که در Cyclone تنها M10K داریم. مطابق با جدول شماره ۲-۱ در صفحه ۲۵ [۲] و صفحه ۲۰ از [۱] این دو خانواده از نظر مجموع حافظه نیز بسیار متفاوت هستند. به‌طوری که کمترین میزان حافظه در یکی از مدل‌های خانواده Stratix، ۱۶ کیلو بیت است درصورتی که بیشترین حجم حافظه در خانواده Cyclone تقریبا ۱۴ کیلوبیت است. تصویر این دو جدول در ادامه آورده شده است.

پاسخ

جافظه‌های موجود در Cyclone:

Variant	Member Code	M10K		MLAB		Total RAM Bit (Kb)
		Block	RAM Bit (Kb)	Block	RAM Bit (Kb)	
Cyclone V E	A2	176	1,760	314	196	1,956
	A4	308	3,080	485	303	3,383
	A5	446	4,460	679	424	4,884
	A7	686	6,860	1338	836	7,696
	A9	1,220	12,200	2748	1,717	13,917

شکل ۴: حافظه‌های Cyclone [۱]

جافظه‌های موجود در Cyclone:

Variant	Member Code	M20K		MLAB		Total RAM Bit (Kb)
		Block	RAM Bit (Kb)	Block	RAM Bit (Kb)	
Stratix V GX	A3	957	19,140	6,415	4,009	23,149
	A4	1,900	38,000	7,925	4,953	42,953
	A5	2,304	46,080	9,250	5,781	51,861
	A7	2,560	51,200	11,736	7,335	58,535
	A9	2,640	52,800	15,850	9,906	62,706
	AB	2,640	52,800	17,960	11,225	64,025
	B5	2,100	42,000	9,250	5,781	47,781
	B6	2,660	53,200	11,270	7,043	60,243
	B9	2,640	52,800	15,850	9,906	62,706
	BB	2,640	52,800	17,960	11,225	64,025
Stratix V GT	C5	2,304	46,080	8,020	5,012	51,092
	C7	2,560	51,200	11,735	7,334	58,534
Stratix V GS	D3	688	13,760	4,450	2,781	16,541
	D4	957	19,140	6,792	4,245	23,385
	D5	2,014	40,280	8,630	5,393	45,673
	D6	2,320	46,400	11,000	6,875	53,275
	D8	2,567	51,340	13,120	8,200	59,540
Stratix V E	E9	2,640	52,800	15,850	9,906	62,706
	EB	2,640	52,800	17,960	11,225	64,025

شکل ۵: حافظه‌های Stratix [۲]

از نظر تعداد ضرب‌کننده‌ها نیز در تراشه‌های خانواده Stratix بیشتر هستند این مورد در جدول صفحه ۴۲ در [۱] و صفحه ۴۸ در [۲] آورده شده است.

پاسخ

مقایسه این دو خانواده از نظر تعداد ضرب‌کننده:

Variant	Member Code	Variable-precision DSP Block	Independent Input and Output Multiplications Operator			18 x 18 Multiplier Adder Mode	18 x 18 Multiplier Adder Summed with 36 bit Input
			9 x 9 Multiplier	18 x 18 Multiplier	27 x 27 Multiplier		
Cyclone V E	A2	25	75	50	25	25	25
	A4	66	198	132	66	66	66
	A5	150	450	300	150	150	150
	A7	156	468	312	156	156	156
	A9	342	1,026	684	342	342	342
Cyclone V GX	C3	57	171	114	57	57	57
	C4	70	210	140	70	70	70
	C5	150	450	300	150	150	150
	C7	156	468	312	156	156	156
	C9	342	1,026	684	342	342	342
Cyclone V GT	D5	150	450	300	150	150	150
	D7	156	468	312	156	156	156
	D9	342	1,026	684	342	342	342
Cyclone V SE	A2	36	108	72	36	36	36
	A4	84	252	168	84	84	84
	A5	87	261	174	87	87	87
	A6	112	336	224	112	112	112
Cyclone V SX	C2	36	108	72	36	36	36
	C4	84	252	168	84	84	84
	C5	87	261	174	87	87	87
	C6	112	336	224	112	112	112
Cyclone V ST	D5	87	261	174	87	87	87
	D6	112	336	224	112	112	112

شکل ۶: ضرب‌کننده‌های Stratix [۲]

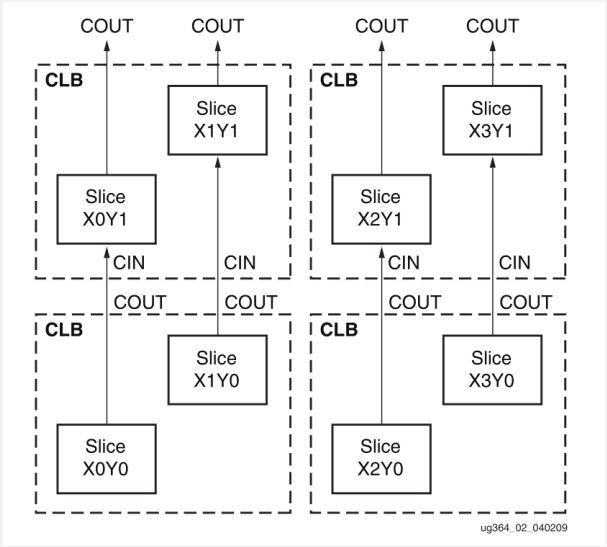
پاسخ

Variant	Member Code	Variable-precision DSP Block	Independent Input and Output Multiplications Operator					18 x 18 Multiplier Adder Mode	18 x 18 Multiplier Summed with 36-bit Input
			9 x 9 Multiplier	16 x 16 Multiplier	18 x 18 Multiplier with 32-bit Resolution	27 x 27 Multiplier	36 x 18 Multiplier		
Stratix V GX	A3	256	768	512	512	256	256	512	256
	A4	256	768	512	512	256	256	512	256
	A5	256	768	512	512	256	256	512	256
	A7	256	768	512	512	256	256	512	256
	A9	352	1,056	704	704	352	352	704	352
	AB	352	1,056	704	704	352	352	704	352
	B5	399	1,197	798	798	399	399	798	399
	B6	399	1,197	798	798	399	399	798	399
	B9	352	1,056	704	704	352	352	704	352
	BB	352	1,056	704	704	352	352	704	352
Stratix V GT	C5	256	768	512	512	256	256	512	256
	C7	256	768	512	512	256	256	512	256
Stratix V GS	D3	600	1,800	1,200	1,200	600	600	1,200	600
	D4	1,044	3,132	2,088	2,088	1,044	1,044	2,088	1,044
	D5	1,590	4,770	3,180	3,180	1,590	1,590	3,180	1,590
	D6	1,775	5,325	3,550	3,550	1,775	1,775	3,550	1,775
	D8	1,963	5,889	3,926	3,926	1,963	1,963	3,926	1,963
Stratix V E	E9	352	1,056	704	704	352	352	704	352
	EB	352	1,056	704	704	352	352	704	352

شکل ۷: ضرب‌کننده‌های Stratix [۲]

تراشه‌های خانواده Virtex از شرکت Xilinx را می‌توان از نظر سرعت و توان پردازشی، همرده تراشه‌های Stratix قرار داد. که در ادامه برخی از ویژگی‌های این خانواده را با خانواده Stratix مقایسه می‌کنیم.

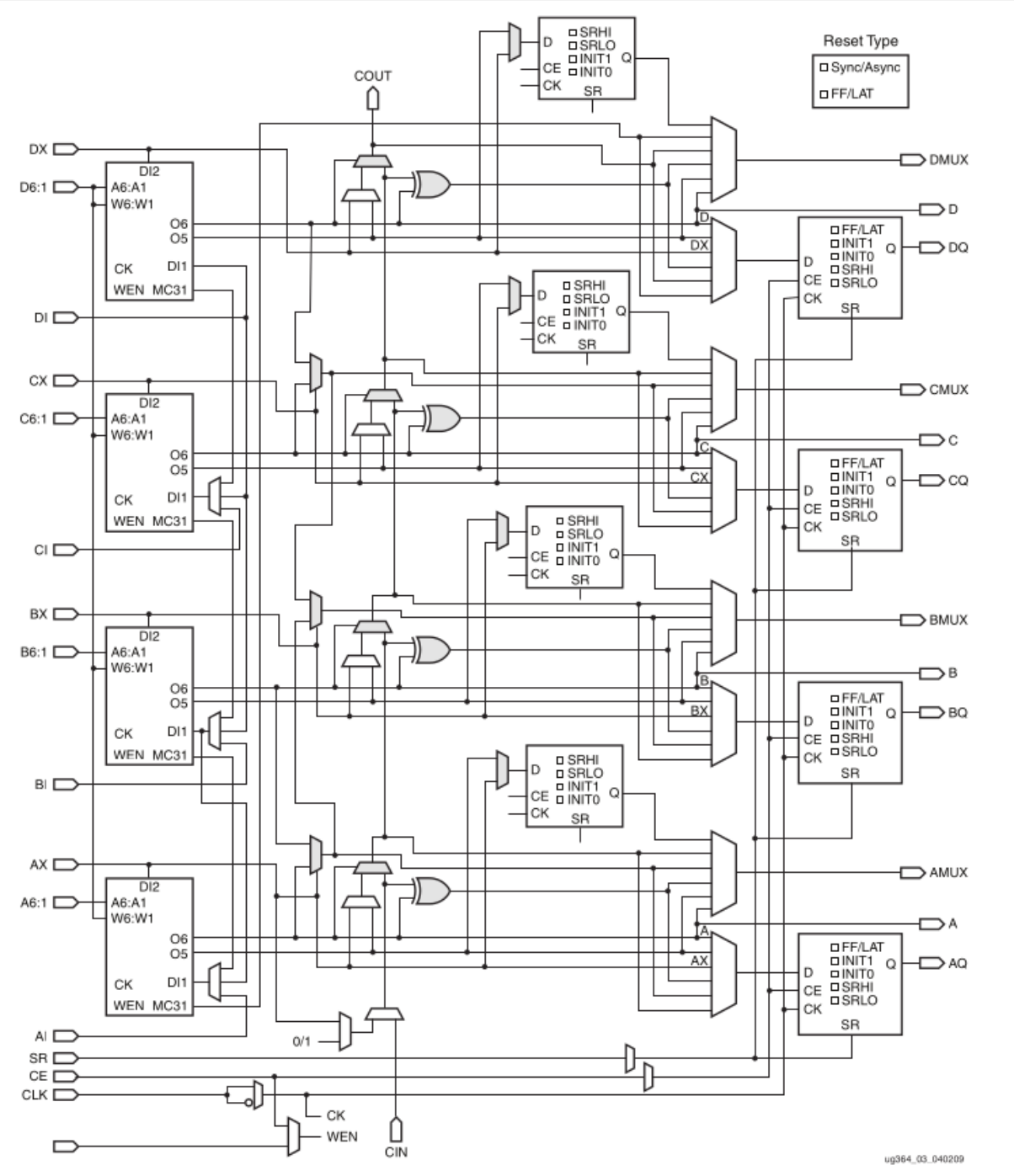
در تراشه‌های شرکت Xilinx نام بلوک‌های قابل برنامه‌ریزی CLB است و ساختار آن تقریبا مشابه است با خانواده Stratix



شکل ۸: ساختار CLB ها در Virtex

پاسخ

ساختار درونی هر CLB به‌صورت زیر است:



شکل ۹: ساختار درونی CLB ها در Virtex [۳]

مطابق با توضیحات موجود در صفحه ۱۱ سند [۳] ورودی‌های LUT ها در این خانواده نیز ۶ عددی هستند. همچنین از نظر مقدار منابع موجود در هر CLB تراشه‌های این خانواده به‌صورت زیر تقسیم‌بندی می‌شوند:

پاسخ

Device	Total Slices	SLICELs	SLICEMs	Number of 6-Input LUTs	Maximum Distributed RAM (Kb)	Shift Register (Kb)	Number of Flip-Flops
XC6VLX75T	11,640	7,460	4,180	46,560	1,045	522.5	93,120
XC6VLX130T	20,000	13,040	6,960	80,000	1,740	870	160,000
XC6VLX195T	31,200	19,040	12,160	124,800	3,140	1,570	249,600
XC6VLX240T	37,680	23,080	14,600	150,720	3,770	1,885	301,440
XC6VLX365T	56,880	40,360	16,520	227,520	4,130	2,065	455,040
XC6VLX550T	85,920	61,120	24,800	343,680	6,200	3,100	687,360
XC6VLX760	118,560	85,440	33,120	474,240	8,280	4,140	948,480
XC6VSX315T	49,200	28,840	20,360	196,800	5,090	2,545	393,600
XC6VSX475T	74,400	48,840	30,560	297,600	7,640	3,820	595,200
XC6VHX250T	39,360	27,200	12,160	157,440	3,040	1,520	314,880
XC6VHX255T	39,600	27,400	12,200	158,400	3,050	1,525	316,800
XC6VHX380T	59,760	41,520	18,240	239,040	4,570	2,285	478,080
XC6VHX565T	88,560	63,080	25,480	354,240	6,370	3,185	708,480

شکل ۱۰: منابع موجود در هر CLB [۳]

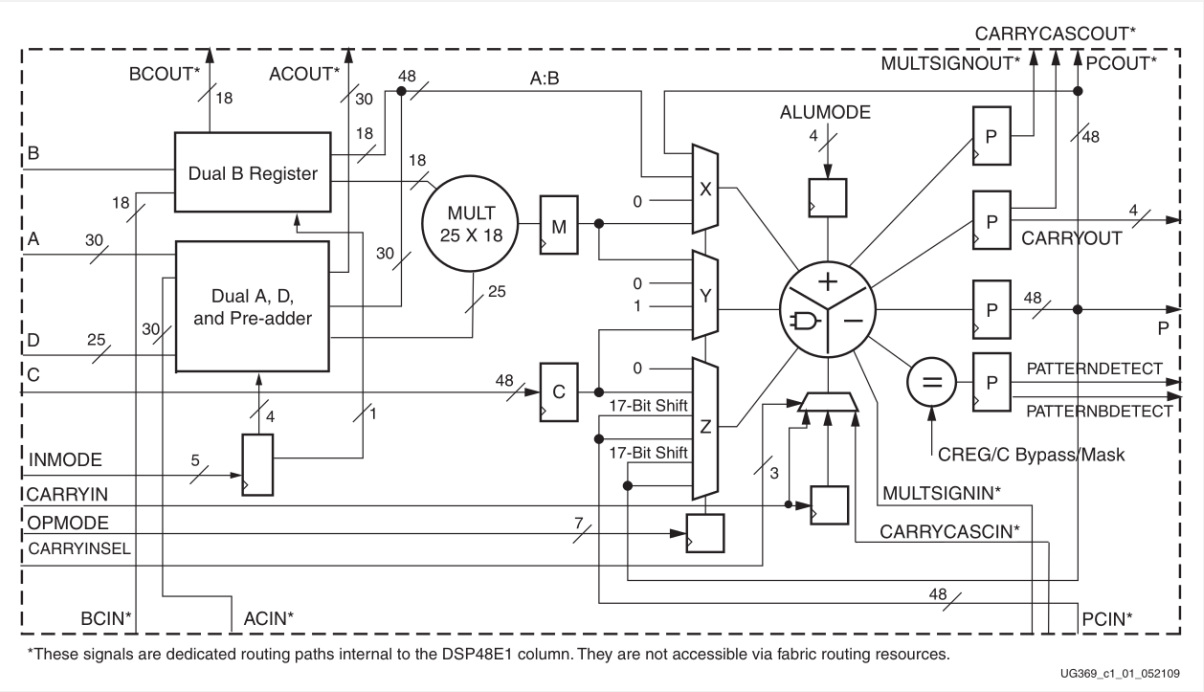
همانطورکه مشخص است، تعداد منابع اعم از تعداد Sliceها، تعداد LUTها و مقدار حافظه تقریبا با تراشه‌های خانواده Stratix مشابه است و در مواردی بیشتر است.
مطابق با جدول ارائه شده در صفحه ۱۳ سند [۴] تراشه‌های این خانواده به‌طور میانگین بیش‌از ۵۰۰ بلوک محاسبات سریع DSP48 را دارند.

Device	Total DSP48E1 Slices per Device	Number of DSP48E1 Columns per Device	Number of DSP48E1 Slices per Column
XC6VHX250T	576	6	96
XC6VHX255T	576	6	96
XC6VHX380T	864	6	144
XC6VHX565T	864	6	144
XC6VLX75T	288	6	48
XC6VLX130T	480	6	80
XC6VLX195T	640	8	80
XC6VLX240T	768	8	96
XC6VLX365T	576	6	96
XC6VLX550T	864	6	144
XC6VLX760	864	6	144
XC6VSX315T	1344	14	96
XC6VSX475T	2016	14	144

شکل ۱۱: تعداد بلوک‌های DSP48 موجود در هر تراشه خانواده Virtex [۴]

که برخلاف تراشه‌های خانواده Stratix این بلوک می‌تواند ضرب سریع، جمع و عملیات‌های منطقی را انجام دهد. هر Slice از این بلوک به‌صورت زیر است:

پاسخ



شکل ۱۲: ساختار درونی بلوک DSP48

و درنهایت برای جمع‌بندی می‌توان گفت اگر در کاربردی نیاز به سرعت و محاسبات خیلی سریع نداشته باشیم و منابع زیادی هم نیاز نداشته باشیم، تراشه‌های خانواده Cyclone با قیمت مناسب و توان مصرفی کمتر نسبت به دو خانواده دیگر انتخاب مناسبی است. اما اگر نیازمند توان پردازشی بالا و منابع زیادی باشیم می‌توانیم از یکی از خانواده‌های][Stratix و یا Virtex استفاده کنیم. از انجایی که این دو خانواده تا حد زیادی شبیه به هم هستند انتخاب میان این دو تراشه کاملا به کاربرد و نیازهای ما از تراشه بستگی دارد.

*

References

[1] Cyclone® V Device Overview [Link]

[2] Stratix® V Device Handbook Volume 1: Device Interfaces and Integration [Link]

[3] 7 Series FPGAs Configuration User Guide [Link]

[4] Virtex-6 FPGA DSP48E1 Slice User Guide [Link]

سوال سوم

آیا با کاهش دقت ذخیره‌سازی برای پیاده‌سازی در FPGAها خصوصاً در شبکه‌های عصبی با استفاده از کوانتیزاسیون همواره دقت کاهش می‌یابد؟ موضوع را تا حد ممکن در حالات مختلف بررسی کنید و با کمک مقالات روز نتایج حاصل را مقایسه کنید. علت استفاده از این روش را نیز به صورت کامل توضیح دهید. در نوشته خود به مقالات مطالعه شده ارجاع دهید.

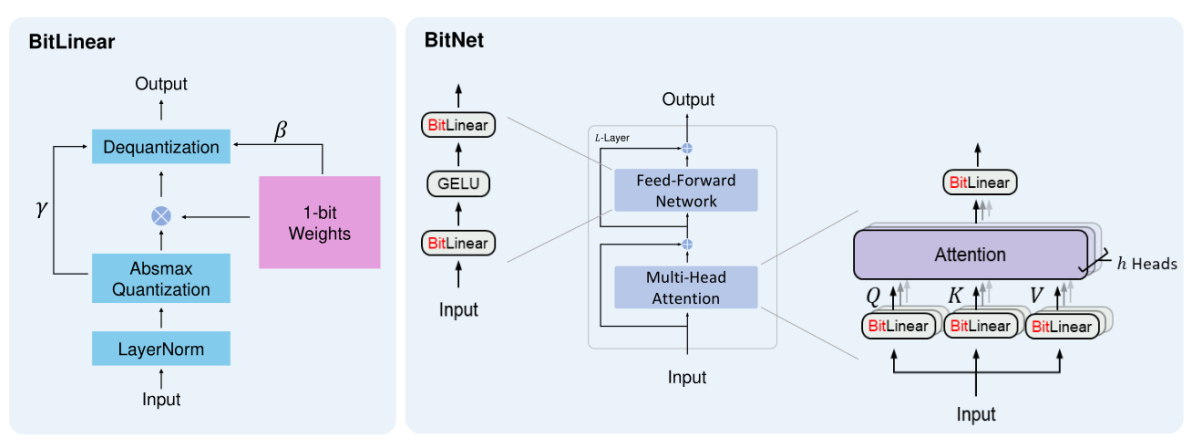
پاسخ

برای پاسخ به این سوال ابتدا نیاز است که دو مفهوم Post-Training Quantization (PQT) و Quantization Aware-Training (QAT) را توضیح دهیم.

۱. **Quantization post-training (PQT):**
اگر خیلی کوتاه بخواهم این مدل را توضیح دهم، بدین صورت است که پس از انجام کامل آموزش (در اینجا درمورد LLM ها صحبت می‌کنم) تعداد بیت‌هایی که موردنیازمان نیستند را دور می‌ریزیم. این مدل همواره باعث کاهش دقت شبکه می‌شود. اما مزیتی که این مدل کوانتایز کردن دارد این است که سریع، ساده و آسان است زیرا نیازی به تغییر در ساختار آموزش یا بازآموزی مدل ندارند. [۱]

Quantization aware training (QAT):
روش دیگری که برای کوانتایز کردن شبکه‌های عصبی عمیق استفاده می‌شود، روش QAT است. در مقایسه با PQT این روش معمولاً به دقت بهتری منجر می‌شود، زیرا مدل از ابتدا برای سازگاری با کاهش دقت آموزش داده می‌شود. علاوه بر این، این روش امکان ادامه آموزش یا انجام تنظیمات دقیق را فراهم می‌کند که برای LLM ها ضروری است. چالش اصلی در آموزش QAT ، بهینه‌سازی است. یعنی با کاهش دقت، مدل سخت‌تر به همگرایی می‌رسد. علاوه بر این، مشخص نیست که آیا آموزش QAT از قانون مقیاس‌پذیری مدل‌های زبانی عصبی پیروی می‌کند یا خیر. [۱]

در ادامه به بررسی دو مقاله می‌پردازیم که QAT را با ۲ بیت انجام می‌دهد. شاید در ابتدای کار کمی عجیب به‌نظر برسد اما در این باره بیل گیتس می‌گوید:
” I don’t think there’s anything unique about human intelligence. All the neurons in the brain that make up perceptions and emotions operate in a binary fashion. ”
ساختار ارائه شده در این مدل به‌صورت شکل زیر است:



شکل ۱۳: ساختار ارائه شده در [۱]

پاسخ

بدین‌صورت است که در شبکه Transformer لایه‌های Linear و توابع فعال‌ساز و وزن‌ها را در زمان آموزش به‌صورت باینری درمی‌آورند و به‌صورت باینری آموزش را انجام می‌دهند. این کار نه‌تنها دقت را خیلی خراب نمی‌کند بله سرعت انجام محاسبات را بسیار بالا می‌برد و همچنین حافظه توان مصرفی، حافظه مورد نیاز نیز بسیار کاهش می‌یابد.
در ابتدای کار ابتدا وزن‌ها را به مقادیر ۱+ و ۱− با استفاده از تابع علامت باینری می‌کنند. وزن‌ها را قبل از باینری‌سازی به مقدار میانگین صفر تنظیم می‌کنیم تا ظرفیت در یک محدوده عددی محدود افزایش یابد. از یک ضریب مقیاس‌دهی β پس از باینری‌سازی نیز استفاده می‌شود تا خطای ℓ_2 بین وزن‌های با مقدار حقیقی و وزن‌های باینری‌شده کاهش یابد.
باینری‌سازی یک وزن $W \in \mathbb{R}^{n\times m}$ به صورت زیر فرمول‌بندی می‌شود:

$$\widetilde{W} = \text{Sign}(W - \alpha),$$

$$\text{Sign}(W_{ij}) = \begin{cases} +1, & \text{if} W_{ij} > 0, \\ -1, & \text{if} W_{ij} \leq 0, \end{cases}$$

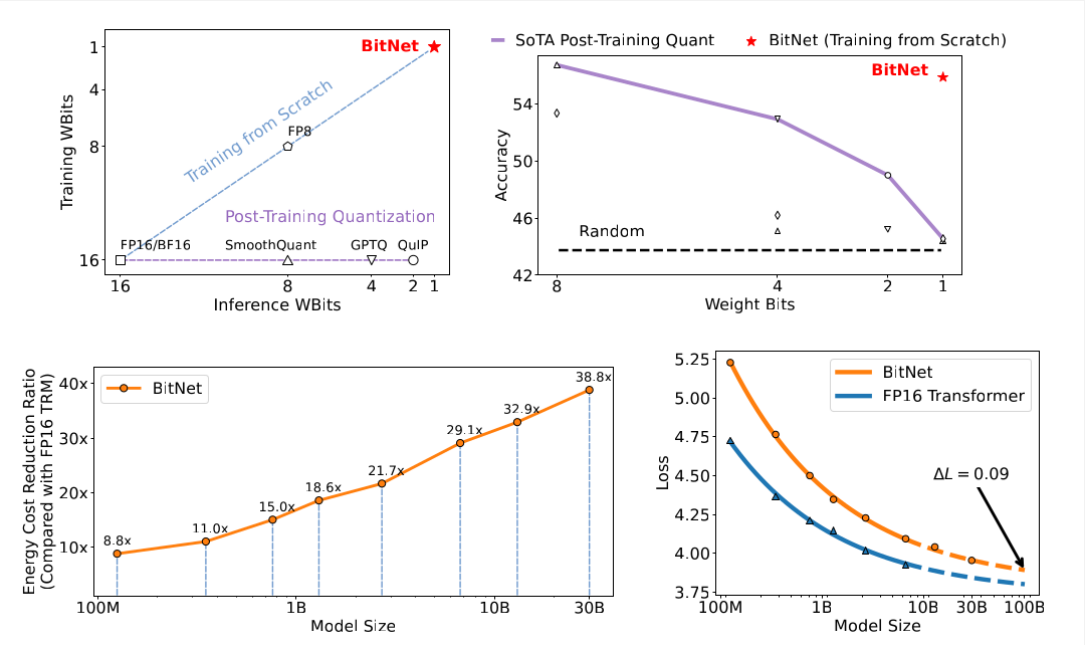
$$\alpha = \frac{1}{nm}\sum_{ij} W_{ij}$$

سپس فعال‌سازی‌ها را به دقت bـبیتی کوانتیزه می‌کنند. سپس فعال‌سازی‌ها را به محدوده $[-Q_b, Q_b]$ (که $Q_b = 2^{b-1}$ است) با ضرب در Q_b و تقسیم بر حداکثر مطلق ماتریس ورودی مقیاس می‌کند:

$$\widetilde{x} = \text{Quant}(x) = \text{Clip}\left(x \times \frac{Q_b}{\gamma}, -Q_b + \epsilon, Q_b - \epsilon\right),$$

$$\text{Clip}(x, a, b) = \max(a, \min(b, x)), \quad \gamma = \|x\|_\infty,$$

نتایج ارائه شده در این مقاله «شکل » نشان می‌دهد که با باینری کردن شبکه، میزان loss شبکه با زمانی که از داده های Floating Point ۱۶ بیتی استفاده می‌کنیم تقریبا برابر است و افزایش چشمگیری ندارد. همچنین از نظر Energy Cost نیز با باینری کردن مدل، انرژی کمتری مصرف شده است.



شکل ۱۴: نتایج ارائه شده در [۱]

پاسخ

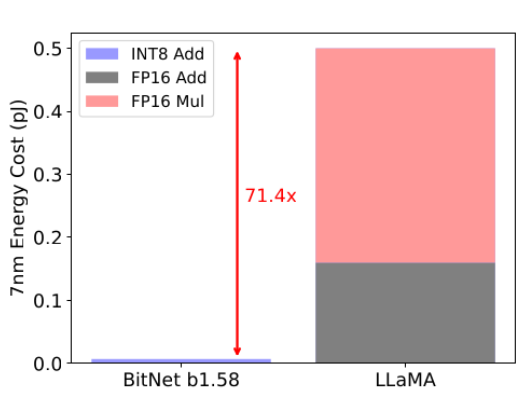
مقاله [۲] که در ادامه مقاله [۱] منتشر شده است، به وزن‌ها مقدار ۱- هم اضافه می‌کند. یعنی شبکه را به سه مقدار ۱+ و ۰ و ۱- آموزش می‌دهد.
فرمول تبدیل وزن‌ها در این مقاله به‌صورت زیر تغییر می‌کند:

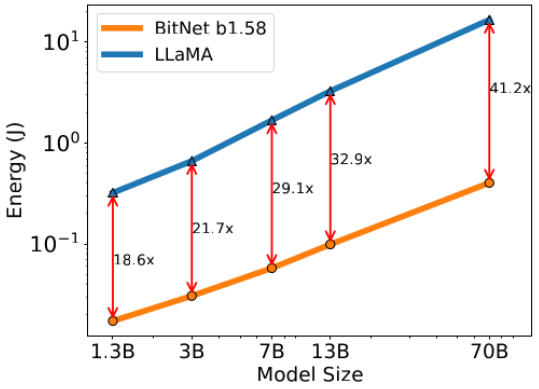
$$\widetilde{W} = \text{RoundClip}\left(\frac{W}{\gamma+\epsilon}, -1, 1\right),$$

$$\text{RoundClip}(x, a, b) = \max(a, \min(b, \text{round}(x))),$$

$$\gamma = \frac{1}{nm}\sum_{ij}|W_{ij}|.$$

در این مقاله نیز گزارش‌ها حاکی از کاهش سایز مدل، کاهش انرژی، کاهش اندک دقت است.





شکل ۱۵: نتایج ارائه شده در [۲]

*

References

[1] BitNet: Scaling 1-bit Transformers for Large Language Models [Link]

[2] The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits [Link]

سوال چهارم

با پیشرفت‌های حاصل شده در خصوص شبکه‌های عصبی معماری‌های FPGA جدید نیز برای پاسخ به این نیاز ایجاد شده‌اند. در این خصوص دو معماری Speedster7t و Versal ACAP را با معماری Stratix 10 مقایسه نمایید و مزایای استفاده از هر یک را برای کاربرد شبکه عصبی شرح دهید. موارد مربوطه را در مدارک فنی شرکت‌های مربوطه مشخص کرده و قسمت مشخص شده را در گزارش خود اضافه نمایید.

پاسخ

این سه خانواده را از نظر واحدهای پردازشی تخصصی، شبکه روی تراشه حافظه و پهنای باند و پشتیبانی از انواع داده‌ها بررسی می‌کنیم:

۱. **منابع و واحدهای پردازش تخصصی:**

(آ) Speedster7t: این معماری دارای بلوک‌های پردازش یادگیری ماشین (MLP) است که شامل آرایه‌ای از ضرب‌کننده‌ها، درخت جمع‌کننده و حافظه‌های داخلی می‌باشد. این بلوک‌ها برای عملیات ماتریسی و برداری در شبکه‌های عصبی بهینه شده‌اند. [۱]
واحدهای پردازشی این تراشه‌های این خانواده در [۱] آورده شده است.

Part Number/Name	AC7t800	AC7t850	AC7t1500	AC7t1550	AC7t3000	AC7t6000
6-input LUTs	380k	334k	692k	646k	1300k	2600k
Inline cryptography	–	Yes	–	Yes	Yes	Yes
MLP blocks	288		2,560		880	1760
LRAM (2 kb)	288		2,560		5,000	10,000
BRAM (72 kb)	1,152		2,560		2,600	5,200
Memory	85 Mb		195 Mb		192 Mb	384 Mb
ML TOps: int8 or block bfloat16	6.9		61		30	61
SerDes 112G	24		32		48	64
DDR4/5	1 DDR5 ×64(1)		1 DDR4X64		2 DDR5X64(1)	4 DDR5x64(1)
High-bandwidth memory channels	6 GDDR6(2)		16 GDDR6(3)		16 GDDR6(4)	16 GDDR6(4)
PCI Express Gen5	One ×16		One ×8, one ×16		Two ×16 with CXL	Two ×16 with CXL
Ethernet	8 lanes 2 × 400G or 8 × 100G		16 lanes 4 × 400G or 16 × 100G		16 lanes 2 × 800G, 4 × 400G or 16 × 100G	32 lanes 4 × 800G, 8 × 400G or 32 × 100G
2D NoC bandwidth (Tbps)	12		20		24	48

شکل ۱۶: واحدهای پردازشی موجود در Speedster7t [۴]

پاسخ

ب) Versal ACAP: این معماری از واحدهای هوش مصنوعی (AI Engines) بهره می‌برد که پردازنده‌های برداری و اسکالر با حافظه‌های مجتمع هستند و برای تسریع عملیات شبکه‌های عصبی طراحی شده‌اند. [۲]

مطابق با جدول ارائه شده در صفحه یک سند [۴] منبع مخصوص AI به‌صورت زیر ارائه می‌شود:

Versal ACAP Resources and Capabilities	AI Edge Series	AI Core Series	Prime Series	Premium Series	HBM Series
Programmable Network on Chip (NoC)	✓	✓	✓	✓	✓
Aggregate INT8 TOPs	7–228	57–228	8–57	36–206	107–157
System Logic Cells (K)	44–1,139	540–1,968	329–2,233	1,575–7,352	3,837–5,631
Hierarchical Memory (Mb)	40–177	90–191	54–282	198–994	509–752
DSP Engines	90–1,312	928–1,968	464–3,984	1,904–14,352	7,392–10,848
AI Engines	8–304	128–400	–	–	–
Processing System	✓	✓	✓	✓	✓
Serial Transceivers	0–44	8–44	8–48	48–168	88–128
Max. Serial Bandwidth (full duplex) (Tb/s)	2.5	2.5	7.8	17.6	11.2
I/O	114–530	478–770	316–770	54–780	780
Memory Controllers	1–3	2–4	1–4	3–4	4
HBM (GB)	–	–	–	–	8–32

شکل ۱۷: واحدهای پردازشی موجود در Versal ACAP [۵]

آ) Stratix 10: این معماری دارای بلوک‌های DSP با قابلیت پشتیبانی از عملیات ممیز شناور و ثابت است، اما فاقد واحدهای تخصصی برای شبکه‌های عصبی مانند دو معماری دیگر می‌باشد. [۳] منابع این خانواده در سوال دوم مفصلا بررسی شده است.

۲. **شبکه روی تراشه (NoC):**

آ) Speedster7t: مجهز به شبکه دوبعدی روی تراشه (2D NoC) است که ارتباطات پرسرعت بین واحدهای مختلف را فراهم می‌کند و برای کاربردهای با پهنای باند بالا مناسب است. مطابق با صفحه ۲۵ در [۱]

ب) Versal ACAP: دارای NoC برنامه‌پذیر است که ارتباطات کارآمد بین واحدهای پردازشی و حافظه‌ها را تسهیل می‌کند. [۶]

ج) Stratix 10: فاقد NoC داخلی است و ارتباطات بین واحدها از طریق مسیرهای برنامه‌پذیر استاندارد FPGA انجام می‌شود.

۳. **حافظه و پهنای باند:**

آ) Speedster7t: از رابط‌های GDDR6 با پهنای باند بالا پشتیبانی می‌کند که برای پردازش داده‌های بزرگ در شبکه‌های عصبی مناسب است. [۷]

ب) Versal ACAP: برخی مدل‌ها دارای حافظه HBM هستند که پهنای باند بالایی را ارائه می‌دهد.

ج) Stratix 10: مدل‌های Stratix 10 MX دارای حافظه HBM2 هستند که پهنای باند بالایی را فراهم می‌کند.

پاسخ

۴. پشتیبانی از انواع داده:

(آ) Speedster7t: پشتیبانی از انواع داده مانند `int8`، `float16` و `bfloat16` را ارائه می‌دهد که برای کاربردهای یادگیری ماشین مناسب است.

(ب) Versal ACAP: پشتیبانی از انواع داده متنوع از جمله `int8` و `float16` را دارد

(ج) Stratix 10: پشتیبانی از انواع داده استاندارد مانند `int8` و `float16` را ارائه می‌دهد.

*

References

[1] AI Benchmarking on Achronix Speedster®7t FPGAs [Link]

[2] Tensor Slices to the Rescue: Supercharging ML Acceleration on FPGAs [Link]

[3] Intel® Stratix® 10 Device Datasheet [Link]

[4] Speedster7t FPGA Datasheet (DS015) [Link]

[5] Versal Architecture and Product Data Sheet: Overview [Link]

[6] Versal Architecture and Product Data Sheet: Overview [Link]

[7] https://www.achronix.com [Link]

سوال پنجم

در این تمرین هدف طراحی و پیاده‌سازی بخشی از یک سیستم پردازش تصویر بی‌درنگ بر روی Zynq SoC است. برای انجام این تمرین بایستی مهارت‌های مربوط به نحوه ارتباط بین بخش PS (سیستم پردازنده) و PL (منطق قابل برنامه‌ریزی) و همچنین نحوه استفاده از رابط میان آنها به عنوان مثال AXI برای ارتباط بین PS و PL مطرح شده در تمرین قبلی را به خوبی فراگرفته باشید.

هدف ایجاد یک هسته برای پردازش تصویر ورودی و تشخیص لبه به صورت بی‌درنگ است. در این تمرین قسمت هسته پردازشی بایستی طراحی شود که یک تصویر را دریافت و خروجی متناظر تشخیص لبه را ایجاد کند. تشخیص لبه یکی از عملیات پایه در پردازش تصویر است که تغییرات ناگهانی در شدت پیکسل‌ها را شناسایی می‌کند. الگوریتم‌های رایج برای تشخیص لبه شامل فیلتر Sobel، Prewitt و Canny هستند. نمونه خروجی تشخیص لبه در تصویر زیر آورده شده است:

شکل ۱۸: تشخیص لبه در تصویر

در این تمرین بایستی تصویر از قسمت PS برای پردازش به قسمت PL ارسال شود و نتایج به قسمت PS جهت نمایش بازگشت داده شود. برای شبیه‌سازی می‌توان قسمت PL را با داده ورودی از طریق Testbench مورد آزمایش قرار داد. برای ورودی، از یک تصویر که شماره دانشجویی شما بر روی آن نوشته شده استفاده نمایید. توضیح کامل نحوه پیاده‌سازی و ایجاد ورودی و خروجی‌ها را در گزارش اضافه کنید و همچنین فایل پروژه خود را با فرمت ZIP در سامانه بارگذاری کنید.

برای الگو گرفتن از یک کد نمونه می‌توانید از این لینک استفاده نمایید.

همچنین الگو گرفتن از کدهای مشابه با ارجاع به منبع، منع ندارد.

پاسخ

این پروژه را به دو قسمت نرم‌افزاری و سخت‌افزاری تبدیل می‌کنیم. بخش سخت‌افزاری نیز به دو زیربخش PL و PS تقسیم می‌شود. در ابتدا توضیحی در مورد عملکر الگوریتم لبه‌یابی Sobel می‌دهیم.
این الگوریتم از دو قسمت اصلی تشکیل شده است:

- ضرایب کرنل عمودی و افقی
- عملیات کانولوشن

برای پیدا کردن لبه‌های عمودی و افقی در یک تصویر ضرایب کرنل در این الگوریتم به‌صورت زیر تنظیم شده است:

$$X\_kernel = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$Y\_kernel = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

این دو کرنل را به‌ترتیب در تصویر ورودی کانوالو می‌کنیم و سپس میانگین خروجی هر دو تصویر کانوالو شده را به‌عنوان لبه‌های تصویر گزارش می‌کنیم.
بدین منظور، در ابتدا تابع کانولوشن را پیاده‌سازی می‌کنیم:

Listing 1: Convolution Function

```python
def convolve(x, kernel):
    x_height = x.shape[0]
    x_width = x.shape[1]

    kernel_height = kernel.shape[0]
    kernel_width = kernel.shape[1]

    H = (kernel_height - 1) // 2
    W = (kernel_width - 1) // 2

    out = np.zeros((x_height, x_width))
    # iterate over all the pixel of image X
    for i in np.arange(H, x_height - H):
        for j in np.arange(W, x_width - W):
            Sum = 0
            # iterate over the filter
            for k in np.arange(-H, H + 1):
                for l in np.arange(-W, W + 1):
                    # get the corresponding value from image and filter
                    a = x[i + k, j + l]
                    w = kernel[H + k, W + l]
                    Sum += (w * a)
            out[i, j] = Sum
        return out
```

پاسخ

سپس تصویر ورودی را می‌خوانیم و آن را به تصویر grayScale تبدیل می‌کنیم:
پ.ن: این تصویر صرفا به دلیل علاقه شدید TA محترم درس به Windows استفاده شده است و جنبه دیگری ندارد :)

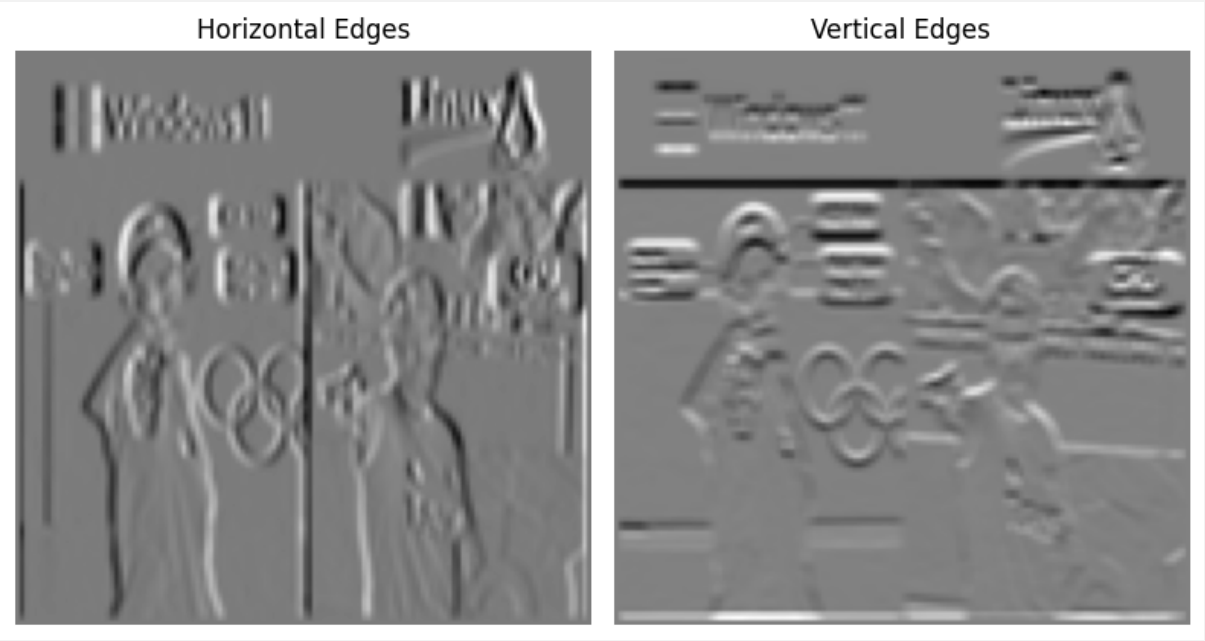
شکل ۱۹: تصویر اصلی ورودی

ابعاد تصویر خاکستری (661 , 1080) است. برای کاهش بار محاسبات و افزایش سرعت، کاهش کیفیت تصویر را به جان می‌خریم و ابعاد تصویر را به (128 , 128) کاهش می‌دهیم. تصویر Resize شده به‌صورت زیر می‌شود:

شکل ۲۰: تصویر Resize شده

سپس هر دو کرنل را در تصویر ورودی ضرب می‌کنیم و خروجی آن به‌صورت زیر می‌شود:

پاسخ



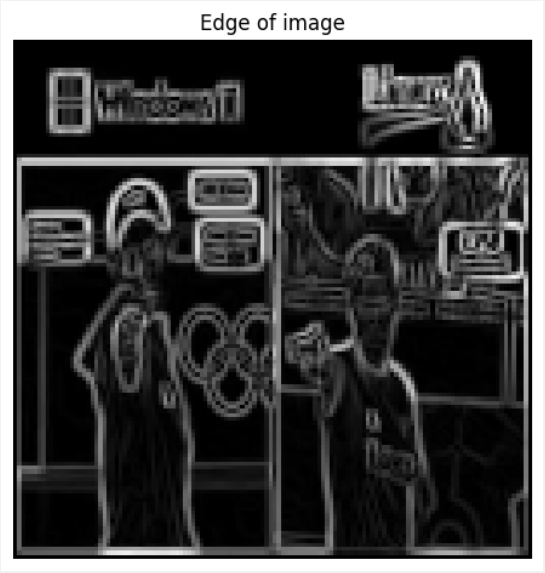
شکل ۲۱: لبه‌های افقی و عمودی پیدا شده

سپس با استفاده از قطعه کد زیر میانگین هر دو لبه را به‌دست می‌آوریم و مقادیر هر پیکسل را بین ۰ تا ۲۵۵ نرمالایز می‌کنیم. تصویر «۲۲» به‌عنوان خروجی نهایی لبه‌های تصویر به‌صورت نرم‌افزاری ارائه می‌شود:

Listing 2: SW Edge Detector

```
edge_out = np.sqrt(np.power(pre_x, 2) + np.power(pre_y, 2))
edge_out = (edge_out / np.Max(edge_out)) * 255
```

پاسخ



شکل ۲۲: لبه‌های نهایی تصویر

پاسخ

در فاز سخت‌افزاری پروژه، همین مراحل را مجددا انجام می‌دهیم تا بتوانیم خروجی‌های هر دو فاز را باهم مقایسه کنیم.
در این فاز از ابزار HLS برای نوشتن سخت‌افزارمان استفاده کرده‌ایم. به دلیل آنکه حجم کد طولانی است، آن را در گزارش نمی‌آوریم اما می‌توانید به فایل `sobel_edge_detector.cpp` موجود در مسیر:
`Codes/HW/Sobel_Edge_Detector_PL/src`
مراجعه کنید. فایل `testBench` ماژول نوشته شده نیز در همین مسیر وجود دارد. فایل تست بدین صورت نوشته شده است که در ابتدا مقادیر پیکسل‌های تصویر خاکستری را از فایل `Linux.txt` می‌خواند و سپس الگوریتم را اجرا می‌کند و نتیجه را در فایل `edge_linux_hls` ذخیره می‌کند. سپس این فایل را به‌صورت زیر به تصویر تبدیل می‌کنیم و آن را نمایش می‌دهیم:

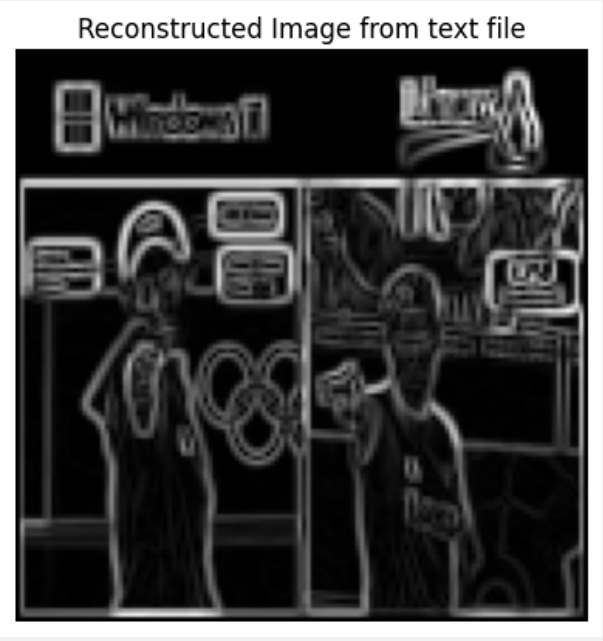
Listing 3: SW Edge Detector

```
img_array = np.loadtxt("Data/edge_linux_hls.txt", dtype=np.uint8)
img = Image.fromarray(img_array)

print("image shape: {}" .Format(img_array.shape))
```

پاسخ

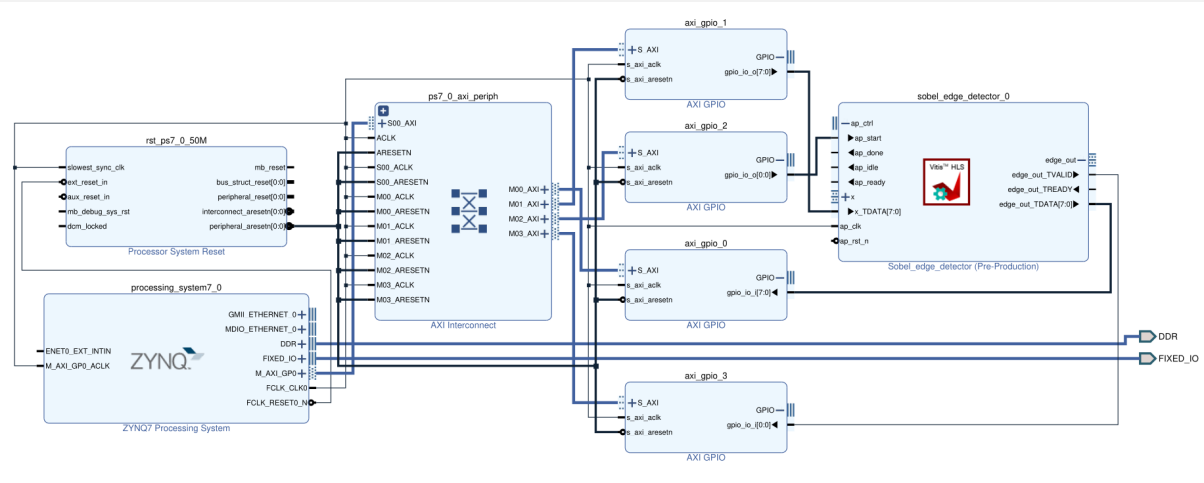
خروجی لبه‌های محاسبه شده در ماژول HLS به‌صورت زیر گزارش می‌شود:



شکل ۲۳: لبه‌های پیدا شده در HLS

در گام بعد نیاز است که ارتباطات بین PL و PS را برقرار کنیم. بدین منظور ابتدا ماژول نوشته شده در HLS را به‌صورت یک IP Core اکسپورت می‌کنیم و آن را در Vivado اضافه می‌کنیم.
برای برقرای این ارتباط از بلوک‌های `AXI_GPIO` استفاده می‌کنیم. ۲ بلوک را برای ارسال داده (عکس) به PL و دریافت داده پردازش شده از PL استفاده می‌کنیم. و از دو بلوک دیگر به عنوان سیگنال‌های کنترلی استفادهکردیم که در بخش مربوطه آن ها را توضیح می‌دهیم. ساختار نهایی طراحی به‌صورت زیر گزارش می‌شود:

پاسخ



شکل ۲۴: دیاگرام طراحی شده برای ارتباط PL و PS

و در نهایت پس از ساخت فایل Wrapper ارتباط بین PL و PS را به‌صورت زیر در Vitis برقرار کردیم:

Listing 4: SW Edge Detector

```
int main()
{
    init_platform();

    XGpio data_input, data_output;
    XGpio start, valid;

    unsigned char edge[ROWS][COLS];

    // GPIO 1, 2: output,
    // GPIO 0, 3: input
    XGpio_Initialize(&data_input, XPAR_AXI_GPIO_0_DEVICE_ID);
    XGpio_Initialize(&valid, XPAR_AXI_GPIO_3_DEVICE_ID);
    XGpio_Initialize(&data_output, XPAR_AXI_GPIO_1_DEVICE_ID);
    XGpio_Initialize(&start, XPAR_AXI_GPIO_2_DEVICE_ID);

    // 0: output, 1: input
    XGpio_SetDataDirection(&data_input, 1, 1);
    XGpio_SetDataDirection(&valid, 1, 1);
    XGpio_SetDataDirection(&data_output, 1, 0);
    XGpio_SetDataDirection(&start, 1, 0);

    while(1)
    {
        if(XGpio_DiscreteRead(&valid, 1) == 1)
        {
            for(int i  = 0; i < ROWS; i++);
            {
                for(int j = 0; j < COLS; j++)
                {
                    XGpio_DiscreteWrite(&data_output, 1, x[i][j]);
                }
            }
            XGpio_DiscreteWrite(&start, 1, 1);
        }
        else
        {
```

```
38              if(XGpio_DiscreteRead(&start, 1) == 1)
39              {
40                  XGpio_DiscreteRead(&data_input, 1);
41              }
42              else
43              {
44
45              };
46          }
47
48      }
49
50      // print("Hello World\n\r");
51      // print("Successfully ran Hello World application");
52      // cleanup_platform();
53
54
55
56      return 0;
57  }
```

پاسخ

در این کد، تصویر خاکستری به‌صورت یک آرایه ۲ بعدی HardCode شده است. سپس با استفاده از دو سیگنال کنترلی valid و start ارسال و دریافت تصویر به PL و PS کنترل شده است.
بدین صور که کاربر درصورتی که بخواهد تصویر را ارسال کنید می‌بایست سیگنال ورودی valid را یک کند سپس تصویر سطر به سطر خوانده می‌شود و مقدار هر پیکسل که ۸ بیتی است به PL ارسال می‌شود. پس از آنکه تصویر به‌صورت کامل ارسال شد، سیگنال start یک می‌شود و این بدین منظور است که الگوریتم می‌تواند شروع به‌کار کند. پس از پیدا کردن لبه‌ها در PL مجددا داده‌ها به‌صورت ۸بیت، ۸ بیت به PS ارسال می‌شود.