
بسم الله الرحمن الرحيم

دانشکده : مهندسی برق و رباتیک

گروه : الکترونیک

عنوان پایان نامه ارشد:

تشخیص حدود جاده با اصلاح لبه‌یابی و پیاده‌سازی آن روی FPGA

دانشجو : هانیه رستمیان

استاد راهنما :

دکتر سید علی سلیمانی ایوری

استاد مشاور:

محسن بیگلری

پایان نامه ارشد جهت اخذ درجه کارشناسی ارشد

ماه و سال انتشار :

شهریور ۱۳۹۵

گروه : الکترونیک

پایان نامه کارشناسی ارشد آقای/ خانم .....هانیه رستمیان.....

تحت عنوان:

تشخیص حدود جاده با اصلاح لبه‌یابی و پیاده‌سازی آن روی FPGA

در تاریخ ..... توسط کمیته تخصصی زیر جهت اخذ مدرک کارشناسی ارشد مورد ارزیابی و با درجه ..... مورد پذیرش قرار گرفت.

اساتید راهنما	امضاء	اساتید مشاور	امضاء
نام و نام خانوادگی : دکتر سید علی سلیمانی ایوری		نام و نام خانوادگی :	
نام و نام خانوادگی :		نام و نام خانوادگی :	

اساتید داور	امضاء	نماینده تحصیلات تکمیلی	امضاء
نام و نام خانوادگی :		نام و نام خانوادگی :	
نام و نام خانوادگی :			
نام و نام خانوادگی :			
نام و نام خانوادگی :			

تقدیم

به خانواده ام

تشکر و قدردانی

شکر و سپاس خدای عزوجل را که همیشه برای بندگانش بهترین‌ها را می‌خواهد....

از زحمات بی‌دریغ استاد راهنمای بزرگوارم، دکتر سید علی سلیمانی، که بدون راهنمایی‌های دلسوزانه ایشان این اثر به انجام نمی‌رسید، قدردانی می‌کنم. ایشان با داشتن پیشینه اطلاعاتی غنی از طراحی و پیاده سازی سخت افزاری، ایده‌ها، نظرات و پیشنهادهای موثری در طول تحقیق به من ارائه نمودند.

از تمامی اساتید گروه الکترونیک دانشگاه صنعتی شاهرود نیز سپاسگزاری و تشکر نموده و برای همه این بزرگواران آرزوی موفقیت و سلامت دارم.

سپاس ویژه از پدر و مادر مهربانم که دعای خیر آنان، همیشه بدرقه راهم بوده....

تعهد نامه

اینجانب .....هانیه رستمیان....دانشجوی دوره کارشناسی ارشد رشته .....الکترونیک-دیجیتال..... دانشکده .....مهندسی برق و رباتیک..... دانشگاه صنعتی شاهرود نویسنده پایان نامه ....تشخیص حدود جاده با اصلاح لبه‌یابی و پیاده‌سازی آن روی FPGA.....تحت راهنمائی.....دکتر سید علی سلیمانی ایوری.....متعهد می شوم .

- تحقیقات در این پایان نامه توسط اینجانب انجام شده است و از صحت و اصالت برخوردار است .
- در استفاده از نتایج پژوهشهای محققان دیگر به مرجع مورد استفاده استناد شده است .
- مطالب مندرج در پایان نامه تاکنون توسط خود یا فرد دیگری برای دریافت هیچ نوع مدرک یا امتیازی در هیچ جا ارائه نشده است .
- کلیه حقوق معنوی این اثر متعلق به دانشگاه صنعتی شاهرود می باشد و مقالات مستخرج با نام « دانشگاه صنعتی شاهرود » و یا « Shahrood University of Technology » به چاپ خواهد رسید .
- حقوق معنوی تمام افرادی که در به دست آمدن نتایج اصلی پایان نامه تأثیرگذار بوده‌اند در مقالات مستخرج از پایان نامه رعایت می گردد.
- در کلیه مراحل انجام این پایان نامه ، در مواردی که از موجود زنده (یا بافتهای آنها) استفاده شده است ضوابط و اصول اخلاقی رعایت شده است .
- در کلیه مراحل انجام این پایان نامه، در مواردی که به حوزه اطلاعات شخصی افراد دسترسی یافته یا استفاده شده است اصل رازداری ، ضوابط و اصول اخلاق انسانی رعایت شده است .

تاریخ

امضای دانشجو

مالکیت نتایج و حق نشر

- کلیه حقوق معنوی این اثر و محصولات آن (مقالات مستخرج ، کتاب ، برنامه های رایانه ای ، نرم افزار ها و تجهیزات ساخته شده است) متعلق به دانشگاه صنعتی شاهرود می باشد . این مطلب باید به نحو مقتضی در تولیدات علمی مربوطه ذکر شود .
- استفاده از اطلاعات و نتایج موجود در پایان نامه بدون ذکر مرجع مجاز نمی باشد.

*** متن این صفحه نیز باید در ابتدای نسخه های تکثیر شده پایان نامه وجود داشته باشد .**

چکیده :

در چند سال اخیر موضوع ایمنی خودروها اهمیت ویژه‌ای پیدا کرده است و به تازگی تحقیقات زیادی بر روی آن انجام گرفته است. سیستم‌های تشخیص خطوط جاده به منظور بهبود ایمنی راننده در جاده‌ها طراحی شده‌اند. ساختار این سیستم‌ها به‌گونه‌ای است که با شناسایی خطوط جاده، به راننده وسیله نقیله‌ای که در حال انحراف از مسیر اصلی حرکت می‌باشد هشدار لازم را می‌دهد. البته با توجه به بسیاری از چالش‌ها و مباحث مربوط به تنوع ظاهری خط، تنوع در وضوح تصویر، تغییر در شرایط دید در جاده‌ها، تشخیص خط مشکلی است که می‌تواند راننده یا سیستم تشخیص با آن روبرو شود.

در این پایان نامه معماری جدید و کارآمدی جهت تشخیص مرزهای جاده ارائه شده است. به منظور لبه‌یابی و کاهش نویز نسبت به طراحی‌هایی که تاکنون در این زمینه صورت گرفته از لبه‌یابی سوبل جهت پیدا کردن لبه‌های قوی استفاده شده است. سپس برای پیدا شدن خطوط، تبدیل هاف و تحلیل اجزای متصل به کار برده شده است.

در مرحله اول، مدار در محیط شبیه ساز متلب پیاده‌سازی شده و عملکرد الگوریتم پیشنهادی مورد ارزیابی قرار گرفته است. پس از کسب موفقیت در طراحی اولیه، از شبیه ساز گرافیکی System Generator و شبیه ساز HDL Coder برای شبیه سازی طرح بر روی FPGA استفاده شده است. طبق نتایج شبیه‌سازی بر روی FPGA، متوسط زمان اجرا ۲۰ms و نرخ تشخیص ۸۷٪ برای سیستم پیشنهادی تشخیص خطوط جاده بدست آمده است. با توجه به این نتایج می توان گفت به نرخ تشخیص قابل قبولی با زمان اجرای کوتاه دست یافته‌ایم.

کلید-واژه : FPGA، پردازش بلادرنگ، تشخیص خطوط خط‌کشی، System Generator ، HDL Coder.

فهرست مطالب

فهرست شکل‌ها

عنوان **صفحه**

فهرست جداول

عنوان **صفحه**

فهرست اصطلاحات اختصاری

عنوان	علامت اختصاری
Advanced Driver Assistance Systems	ADAS
National Highway Traffic Safety Administration	NHTSA
lane departure warning	LDW
Line Hough Transform	LHT
DSP Processors	DSP
Field Programmable Logic Gate Array	FPGA
Xilinx System Generator	XSG
Regien of interest	ROI
Lane Marking Pattern Search	LMPS
Ant Colony Optimization	ACO
Vertical Mean Distribution	VMD
RANdom Sample and Consensus	RANSAC
Symmetrical Local Threshold	SLT
Fast Fourier Transform	FFT
Discrete cosine transform	DCT
Finite Impulse Response	FIR
Infinite Impulse Response	IIR
Input/Output	I/O
Clock	CLK
Hardware Description Language	HDL

فصل اول : مقدمه

۱-۱ پیشگفتار

در چند سال اخیر موضوع ایمنی خودروها اهمیت ویژه‌ای پیدا کرده است و به تازگی تحقیقات زیادی بر روی آن انجام گرفته است. سیستم‌های هوشمند نظارت بر راننده[1] از جمله مواردی است که در ایمنی خودروها مورد توجه قرار گرفته، به‌طوری که این سیستم‌ها با تشخیص هوشمند شرایط حادثه‌ساز، سعی در کمک و هشدار دادن به راننده را دارند. با استفاده از این‌گونه سیستم‌های هوشمند می‌توان حوادث و تصادفات رانندگی را به طور قابل ملاحظه‌ای کاهش داد. سیستم‌هایی که به راننده در طی فرآیند رانندگی کمک می‌کنند به عنوان سیستم‌های کمک راننده پیشرفته[2] (ADAS) شناخته می‌شوند. اکثر حوادث جاده‌ای با توجه به بی‌دقتی راننده اتفاق می‌افتند. سیستم‌های کمک راننده پیشرفته، ایمنی را تضمین کرده و حجم کاری راننده را کاهش می‌دهند.

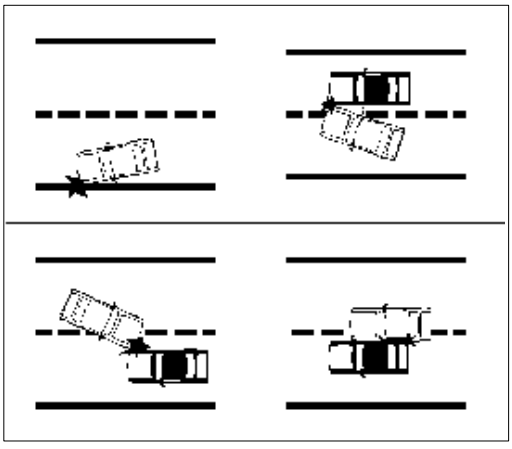
مهم‌ترین تجهیزاتی که تاکنون برای خودروهای هوشمند[3] درنظر گرفته شده است، شامل سیستم‌های اضطراری کمک ترمز، هشدار تصادف از جلو، هشدار خروج از خطوط جاده، تشخیص نقاط کور راننده، چراغ های هوشمند جلو و تشخیص خواب‌آلودگی است.

خواب‌آلودگی که سبب خروج از جاده می‌شود، کابوسی است که همواره بر روح و جان مسافران سایه انداخته است. متأسفانه این کابوس در جاده‌های کشورمان در موارد متعددی به واقعیت پیوسته و همواره در سطح جاده‌های کشور شاهد از بین رفتن هموطنان عزیزمان هستیم. کشور ایران در بحث تلفات جاده‌ای در جهان رتبه نخست را دارد و سالیانه حدود ۳۰ هزار نفر از عزیزانمان در این حوادث جان خود را از دست می‌دهند[1]. گفتنی است این موضوع محدود به کشور ما نیست و در سرتاسر جهان معضل تصادفات جاده‌ای وجود دارد.

[1] Intelligent Driver Monitoring System
[2] Advanced Driver Assistance Systems
[3] Smart Car

بنابر اعلام موسسه[1] NHTSA آمریکا ، هر ساله صد هزار تصادف به علت خواب رفتن راننده رخ می‌دهد که به طور میانگین باعث آسیب شدید به ۷۰۰۰۰ نفر و مرگ ۱۵۵۰ نفر می‌شود [2,3]. به طور کلی خستگی راننده عامل اصلی ۲۵ درصد تصادفات و به طور خاص ۶۰ درصد تصادفات جاده‌ای منجر به مرگ و یا آسیب های جدی می‌باشد[4].

هنگامی که راننده دچار خواب‌آلودگی است تسلط کافی به کنترل خودرو ندارد، بنابراین ممکن است ناگهان از جاده منحرف شده و به مانعی برخورد یا خودرو واژگون شود. در نتیجه سیستم هشدار خروج از خط[2] (LDW) یک ماژول مهم در سیستم کمک راننده پیشرفته است. از مزایای این سیستم این است که اگر راننده بخواهد وارد خط دیگر شده و یا از خط خارج شود و با موانع برخورد کند او را آگاه می‌سازد. در بینایی مبتنی بر سیستم هشدار خروج از خط جاده یک دوربین در جلو وسیله‌نقلیه نصب و تصاویری از جاده گرفته می‌شود. خط‌های روی جاده تفسیر و خط‌کشی‌ها شناسایی می‌شوند. زمانی‌که خودرو به طور ناخواسته بیرون از خط‌کشی حرکت کند، به راننده هشدار داده می‌شود. تشخیص نشانگر خط‌کشی گام اول سیستم هشدار خروج از خط جاده است. در شکل ۱-۱ انواع تصادفاتی که توسط سیستم هشدار دهنده انحراف از مسیر قابل پیشگیری می‌باشد، نشان داده شده است.



شکل۱-۱ : انواع تصادفات قابل پیشگیری توسط سیستم هشدار دهنده انحراف از مسیر.

[1] National Highway Traffic Safety Administration
[2] lane departure warning

بسیاری از چالش‌ها و مباحث مربوط به تنوع ظاهری خط، تنوع در وضوح تصویر، تغییر در شرایط دید در سیستم‌های هشدار دهنده انحراف از مسیر مطرح می‌شود. از جمله این چالش‌ها عبارتند از :

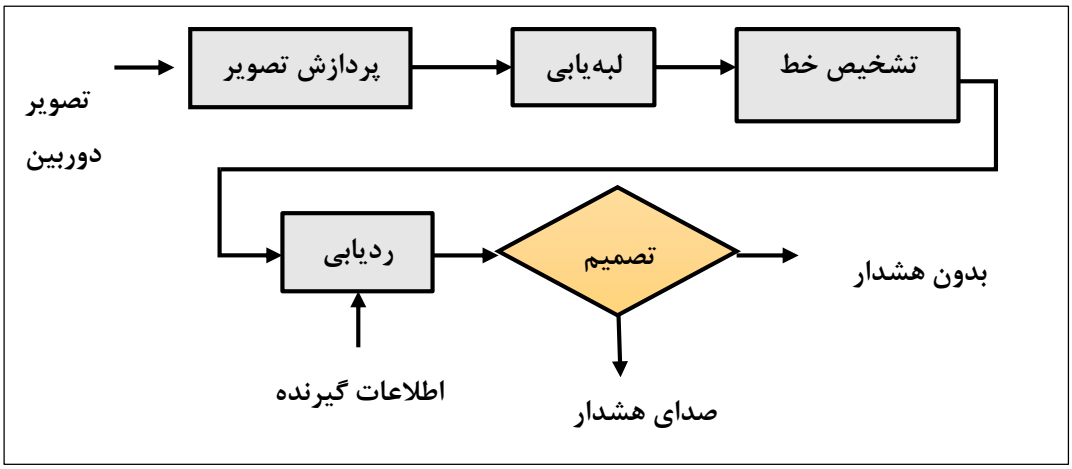
۱. مسائل مربوط به وضوح تصویر با توجه به حضور سایه.
۲. نشانگر خط ممکن است توسط وسیله‌نقلیه جلویی مسدود شده باشد.
۳. ایجاد یک تغییرات ناگهانی در روشنایی هنگامی‌که وسیله‌نقلیه از تونل بیرون می‌آید.
۴. ممکن است دید نشانگر خط به دلیل شرایط آب و هوایی مختلف مانند : باران، مه، برف کاهش یابد.
۵. دید در شرایط شب کمتر می‌باشد.

شکل ۱-۲ چالش‌های موجود در تشخیص خط و ردیابی آن را نشان می‌دهد.

شکل ۱-۲ : چالش‌های مختلف در تشخیص خط. a) مسدود شدن نشانگر خط توسط وسیله نقلیه جلویی، b) درصدی از سایه، c) جاده بارانی، d) نهایت روشنایی در سمت چپ تصویر

دو روش برای تشخیص خطوط خط‌کشی وجود دارد : روش مبتنی بر ویژگی و روش مبتنی بر مدل.

روش مبتنی بر ویژگی از ویژگی‌های سطح پایین مثل لبه‌ها [5-7] استفاده می‌کند. در حالی که روش مبتنی بر مدل برای تشخیص خط پارامترهای هندسی [8-10] را مورد استفاده قرار می‌دهد. شکل ۱-۳ بلوک دیاگرام سیستم هشدار خروج از خط را به طور کلی نشان می‌دهد.



شکل ۱-۳: بلوک دیاگرام یک سیستم هشدار خروج از خط ساده.

سیستم هشدار خروج از خط از تبدیل هاف برای استخراج خط در تصاویر جاده انتقال یافته از دوربین استفاده می‌کنند. تبدیل هاف خطی[1] (LHT) [11] یک روش مشهور برای تشخیص خطوط مستقیم در تصویر است. این روش در برابر نویز و تغییرات سطح روشنایی خیلی مقاوم است.

سیستم هشدار خروج از خط بسیاری از چالش‌ها و مباحث مربوط به تنوع ظاهری خط، تنوع در وضوح تصویر، تغییر در شرایط دید [12] را مطرح می‌کند. براساس کشورها در نشانه‌گذاری خط‌کشی‌های مورد استفاده تفاوت وجود دارد. معمولا رنگ سفید و زرد برای نشانه‌گذاری خط‌کشی استفاده می‌شود.

[1] Liner Hough Transform

۱-۲ انگیزه و چالش‌ها

امروزه الگوریتم‌های زیادی در مباحث پردازش صوت و تصویر، بینایی ماشین، داده‌کاوی[۱] و تشخیص الگو[۲] به صورت توسعه یافته بر روی نرم‌افزار اجرا می‌شود و هر روز نیز پیشرفت جدیدی در توسعه آنها حاصل می‌گردد. بسیاری از این الگوریتم‌ها هنگام پیاده‌سازی نرم‌افزاری به دلیل پیچیدگی‌های بلادرنگ[۳] محاسباتی با نرخ فریم[۴] پایینی اجرا می‌شوند، بنابراین برای پردازش سریع، نیاز به اجرای این الگوریتم‌ها بر روی سخت‌افزار است. با این حال پیاده‌سازی این الگوریتم‌ها بر روی سخت‌افزارهایی مانند ریزپردازنده-ها[۵]، پردازنده‌های DSP[۶] و یا FPGA [۷]ها همچنان چالش مهمی محسوب می‌شود.

از آنجایی که سیستم‌های LDW نیاز به عملکرد سریع در پردازش تصویر دیجیتال دارند، اگر بتوان عملیات را به صورت موازی انجام داد می توان این نیاز را برطرف کرد. بکارگیری FPGAها در این زمینه می تواند کمک موثری باشد. تکنولوژی FPGA یک پیشنهاد برای پیاده‌سازی الگوریتم‌های نرم‌افزاری است.

FPGAها ساختارهای قابل پیکربندی[8] هستند که می‌توانند عملیات پردازش تصویر را با ارائه یک طراحی مناسب سخت‌افزاری به صورت زمان واقعی انجام دهند. مشکل استفاده از FPGAها این است که زمان مورد نیاز برای طراحی و پیاده‌سازی آنها نسبت به نرم‌افزار طولانی‌تر بوده و انعطاف‌پذیری کمتری دارند.

---

[1] Data Mining
[2] Pattern Recognition
[3] Real-Time
[4] Frame Rate
[5] Microprocessors
[6] DSP Processors
[7] Field Programmable Logic Gate Array
[8] Configurable

به کارگیری اعداد ممیز شناور[1] و محاسبات ریاضی روی FPGA به سادگی استفاده از میکروپروسسورها نیست. باید از اعداد ممیز ثابت[2] استفاده کرد و روابط ریاضی به الگوریتم‌هایی به صورت قابل اعمال به FPGA تبدیل و سپس استفاده شوند. همچنین سعی می‌شود هنگام پیاده‌سازی تا جایی که ممکن است عملیات به شکل موازی تبدیل و از امکانات FPGA استفاده شود.

۱-۳ اهداف پایان‌نامه

در این پایان‌نامه از تکنیک ترکیبی نرم‌افزار-سخت افزار برای پیاده‌سازی الگوریتم تشخیص خطوط جاده استفاده شده است، بدین صورت که سیستم پیشنهادی در ابتدا به صورت دو بعدی در محیط نرم‌افزاری متلب اجرا می‌شود و بنابر ضرورت پیاده‌سازی قسمت سخت‌افزاری الگوریتم، برنامه مجدداً در محیط نرم‌افزاری متلب به صورت یک بعدی نوشته و پیاده‌سازی می‌شود. در نهایت از شبیه‌ساز System Generator و شبیه ساز HDL Coder برای پیاده‌سازی طرح مورد نظر بر روی FPGA استفاده می‌گردد.

۱-۴ ساختار پایان‌نامه

پس از بیان مقدمه، در فصل دوم مروری بر مطالعات و تحقیقات انجام شده در زمینه تشخیص حدود جاده بر روی FPGA انجام شده است. در فصل سوم این پایان نامه ابتدا درباره‌ی لبه‌یابی توضیح مختصری داده و سپس یکی از روش‌های لبه‌یابی مورد بررسی قرار گرفته است. بعد از این تبدیل هاف به طور مفصل شرح داده شده است.

در فصل چهارم کاربرد FPGAها در پردازش سیگنال دیجیتال مرور می‌شود. در این فصل تکنیک‌های بهینه سازی طرح‌های سخت افزاری به وسیله FPGA، اصول طراحی دیجیتال و.... تشریح می‌شود.

[1] Floating Point
[2] Fixed Point

در فصل پنجم ابتدا الگوریتم به صورت کلی ارائه شده و سپس اجرای آن به صورت نرم‌افزاری و سخت افزاری شرح داده می‌شود.

و در نهایت فصـل ششم نتایج حاصل از شبیه‌سازی سخت‌افزاری ساختار طراحی شده در فصل ۵ را نشان می‌دهد و پیشنهاداتی برای کارهای آتی در این زمینه ارائه می‌شود.

فصل دوم : مروری بر مطالعات و تحقیقات گذشته

۲-۱ مقدمه

حمل و نقل یکی از زیربناهای اصلی توسعه در هر کشوری به‌شمار می‌آید. در بین شیوه‌های متفاوت حمل و نقل، حمل و نقل جاده‌ای یکی از متداول‌ترین روش‌های جابجایی محسوب می‌شود. سیستم تشخیص خط‌کشی یک جزء مهم بسیاری از سیستم‌های حمل و نقل هوشمند است. در سیستم‌های حمل و نقل هوشمند، وسایل‌نقلیه هوشمند با زیرساخت‌های هوشمند برای دستیابی به یک محیط امن‌تر و شرایط ترافیک بهتر همکاری می‌کنند.

در سال ۱۹۹۶ میلادی موسسه بین المللی ایمنی بزرگراه‌ها گزارشی برای بررسی سیستم‌های هوشمند متشکل از سیستم هشدار دهنده فاصله از خودرو جلویی و سیستم هشدار دهنده انحراف از مسیر منتشر کرد. با وجود نبود این سیستم‌ها در زمان انتشار این گزارش، چنین تخمین زده شد ترکیب چنین سیستم‌هایی می‌تواند سالانه از وقوع ۷۹۱۰۰۰ مورد تصادف از عقب و ۹۰۰۰۰ مورد تصادف بر اثر عدم حرکت در بین خطوط و ۲۹۷۰۰۰ تصادف انحراف از مسیر و چپ شدن اتومبیل در ایالت متحده آمریکا جلوگیری کند. همچنین تخمین زده شد که استفاده از سیستم هشدار دهنده انحراف از مسیر به تنهایی می‌تواند از وقوع ۴۸۳۰۰۰ مورد تصادف در سال در ایالت متحده آمریکا جلوگیری کند[13].

برای نخستین بار در سال۲۰۰۳ میلادی فن‌آوری سیستم هشدار دهنده انحراف از مسیر توسط آیتریس[1] برای کامیون‌ها بکار گرفته شد. در سال ۲۰۰۴ میلادی این سیستم برای خودروهای مسافربر کوچک نیز مورد استفاده قرار گرفت[14].

[1] Iteris

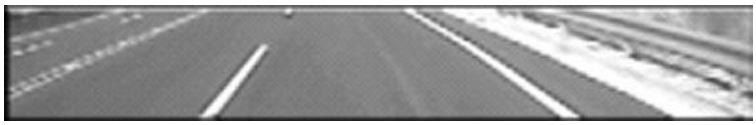
چندین الگوریتم برای استخراج و ردیابی مسیر جاده در چند دهه گذشته پیشنهاد شده است. که تفاوت اصلی آنها شامل : مرحله پیش‌پردازش خط لوله‌ای[1] تصویر ثبت شده، مدل خط، روش تطبیق مدل[2] انتخاب شده و استراتژی ردیابی است.

در این فصل به بررسی تعدادی از تحقیقاتی که در سال‌های اخیر در زمینه تشخیص حدود جاده و پیاده‌سازی آن بر روی FPGAها، برای کاربردهای مختلف، صورت گرفته است می پردازیم.

۲-۲ تشخیص خطوط جاده مبتنی بر ویژگی سه گانه

Yim و Oh در [15] یک الگوریتم تشخیص خط مبتنی بر یک ویژگی سه گانه را توسعه دادند. ویژگی های مورد استفاده موقعیت شروع، جهت و مقدار شدت روشنایی است. در گام اول عملگر سوبل برای بدست آوردن اطلاعات لبه همان طوری که در شکل ۲-۱ دیده می‌شود، اعمال شده است.

الف)



ب)

[1] Pipline
[2] model fitting

ج)

شکل ۲-۱ : پردازش تصویر. الف) تصویر اصلی، ب) تصویر کوچک شده، ج) تصویر لبه‌یابی شده [15].

مرز خط‌کشی به عنوان یک بردار متشکل از ۳ ویژگی (یک بردار ۳ بعدی در فضای ویژگی) ارائه شده است. بردار خط فعلی براساس تصویر ورودی و بردار مدل خط قبلی محاسبه می‌شود. از دو پنجره هر کدام برای مرزهای چپ و راست استفاده می‌شود (مانند شکل ۲-۲). با فرض N پیکسل در هر خط افقی N بردار خط کاندید تولید شده و بهترین کاندید براساس فاصله از بردار خط قبلی با استفاده از یک معیار فاصله وزن‌دار انتخاب می‌شود. برای برابری هر ویژگی یک وزن متفاوت اختصاص داده می‌شود. سپس یک سیستم استنباط خط برای پیش بینی بردار خط جدید به‌کار گرفته می‌شود. اگر پهنای جاده به طور ناگهانی تغییر کند، بردار جاری محاسبه شده دور انداخته و از قبلی به عنوان بردار فعلی استفاده می‌شود.

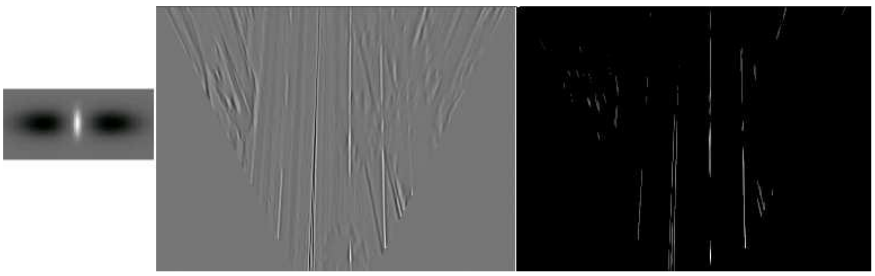
شکل ۲-۲ : مشخص شدن مرزهای خط‌کشی با دو پنجره نشان داده شده [15].

۲-۳ تشخیص بلادرنگ خطوط در جاده‌های شهری

Aly در [16] یک روش مقاوم و بلادرنگ برای تشخیص خط در جاده‌های شهری ارائه کرده است. در ابتدا یک دید از بالا[1] از تصاویر جاده با استفاده از نقشه برداری پرسپکتیو معکوس[2] برای اجتناب از اثر پرسپکتیو تولید می‌شود (همان طوری که در شکل ۲-۳ نشان داده شده است.).

شکل ۲-۳: پرسپکتیو ساده. سمت چپ: تصویر ورودی با ناحیه مورد نظر در پنجره قرمز. سمت راست دید پرسپکتیو[16].

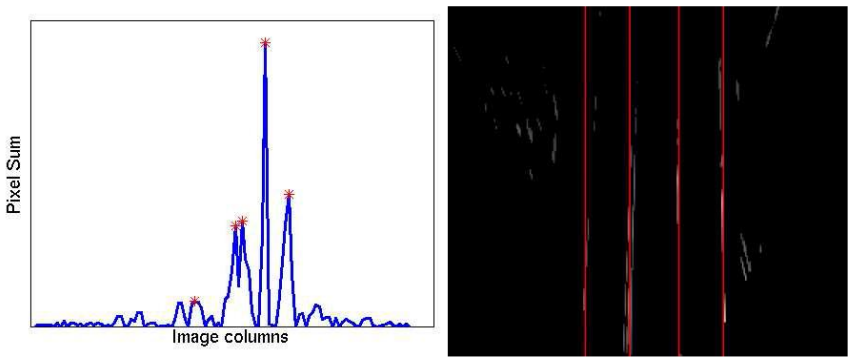
سپس دید از بالا با انتخاب یک کرنل گوسی دو بعدی جهت دار فیلتر می‌شود. فیلتر برای خط‌های روشن با پهنای ویژه در پس زمینه تاریک به طور خاص تنظیم شده است. بنابراین واکنش بالایی نسبت به نشانگرهای خط دارد و فقط بالاترین مقدار با انتخاب ۹٪ از مقدار چارک تصویر فیلتر شده را برمی‌گرداند و همه‌ی مقدارهای زیر آستانه را حذف می‌کند (مانند شکل ۲-۴).



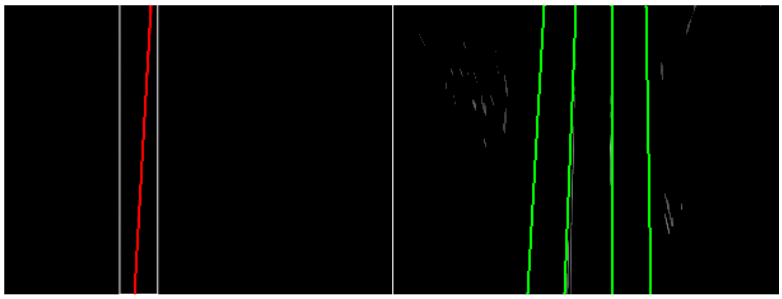
شکل ۲-۴: تصویر فیلتر شده و آستانه گذاری شده. چپ: کرنل مورد استفاده فیلتر کردن. وسط: تصویر بعد از فیلتر گذاری. راست: تصویر بعد از آستانه گذاری[16].

[1] Top view
[2] Inverse perspective mapping
[3] Eigen value

سپس خطوط مستقیم با استفاده از یک تبدیل هاف ساده که توسط فیلتر کردن خط RANSAC که یک گمان اولیه از نوار باریک فیلتر شده RANSAC فراهم شده دنبال می‌شود (شکل ۲-۵ و۲-۶).



شکل ۲-۵: گروه بندی هاف. سمت چپ) نشان دهنده ماکزیمم‌های محلی. سمت راست) خط های تشخیص داده شده بعد از گروه بندی[16].



شکل۲-۶: تطبیق کردن خط RANSAC. سمت چپ)یکی از خط های تشخیص داده شده. سمت راست) نتیجه خط ها از مرحله تطبیق کردن خط RANSAC [16].

یک گام پس پردازش برای متمرکز کردن نوار باریک انجام شده و آن را در تصویر امتداد می‌دهد (شکل۲-۷). الگوریتم به صورت ردیابی عمل نمی‌کند و می‌تواند هر تعداد خط در تصاویر نه فقط خط جاری را تشخیص دهد.

شکل ۲-۷ :گام پس پردازش برای متمرکز کردن خطوط [16].

۲-۴ تشخیص خطوط جاده مبتنی بر روش سلسله مراتبی

Cheng و همکارانش در [17] یک الگوریتم سلسله مراتبی برای تشخیص خط معرفی کرده‌اند. نقاط ویژگی با ابعاد بالا براساس ویژگی رنگ استخراج می‌شوند. آن‌ها برای تشخیص دادن جاده باساختار، از جاده بدون ساختار استفاده کرده‌اند(شکل۲-۸).

شکل۲-۸: سمت چپ) جاده با ساختار، سمت راست) جاده بدون ساختار [17].

سپس بر روی نقاط ویژگی تحلیل اجزای متصل اعمال شده است. یک بردار ویژگی برای نقاط ویژگی با بیش از یک آستانه ساخته می‌شود. تحلیل منظم و مشخص بردار ویژه برای کاهش واریانس مدل انجام شده است. سپس پارامترهای گوسی با حداکثر درست‌نمایی[1] تخمین زده می‌شود. نقاط ویژگی استخراج شده به عنوان خطوط تشخیص داده در جاده‌های باساختار استفاده می‌شود. برای جاده های بدون ساختار کل صحنه براساس ناحیه‌بندی انتقال میانگین[2] ناحیه بندی می‌شود. هر ناحیه به صورت همگن درنظر گرفته می‌شود و نشانگرهای خط با استفاده از قانون بیز تشخیص داده می‌شود(شکل ۲-۹).

[1] Maximum Liklihood
[2] Mean-shift segmentation

شکل۲-۹: نتیجه تشخیص خط در جاده های با ساختار و بدون ساختار [17].

۲-۵ تشخیص خطوط جاده در شب

Borkar و همکارانش در [18] یک روش تشخیص خط، مناسب در شب را پیشنهاد کرده‌اند. الگوریتم اول ناحیه مورد علاقه[1] (ROI) را با از بین بردن آسمان و دیگر اشیاء بی ربط مقداردهی می‌کند (شکل۲-۱۰).

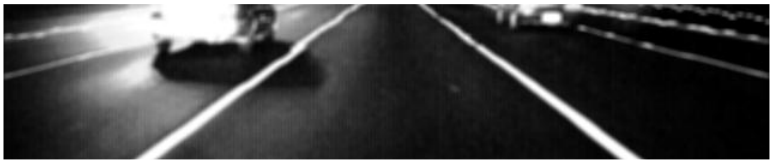
الف)

ب)

شکل۲-۱۰: تصاویر قبل و بعد از بریدن. الف) تصویر اصلی، ب) تصویر بریده شده[18].

[1] Regien of interest

در گام بعدی یک تصویر خاکستری توسط متوسط‌گیری از ۳ کانال رنگ بدست می‌آید. از تاری موقتی در امتداد خط تیره خط‌کشی استفاده شده است. سپس یک آستانه‌گذاری وفقی برای استخراج اشیاء روشن انجام می‌شود (شکل۲-۱۱).



الف)

ب)

شکل۲-۱۱: تصویر قبل و بعد از آستانه‌گذاری وفقی. الف)تصویر با تاری موقتی، ب) تصویر تارشده با اعمال آستانه[18].

تصویر باینری حاصل به دو نیمه چپ و راست تقسیم شده و تبدیل هاف با رزولوشن پایین در هر نیمه برای تشخیص خطوط مستقیم اعمال می‌شود. در نهایت یک کرنل گوسی با تاثیر فیلتر همسان برای استخراج نشانگرهای خط‌کشی استفاده شده است (شکل۲-۱۲).

الف)

ب)

شکل۲-۱۲: تشخیص نشانگرهای خط با موفقیت. الف) تخمین موقعیت نشانگر خط، ب) نشانگرهای خط موقعیت یابی شده [18].

۲-۶ تشخیص بلادرنگ خطوط جاده و پیاده‌سازی آن بر روی FPGA

Roberto و همکارانش در [19] یک الگوریتم برای استخراج و ردیابی مسیر جاده و پیاده‌سازی آن روی FPGA پیشنهاد کرده‌اند. این الگوریتم با ۳ موضوع زیر شروع می‌شود: ۱. استخراج خط از میان

نامزدهای نقشه لبه، زمانی که از لبه‌های جعلی صرف نظر شده است. ۲. پیدا کردن مدل جاده مناسب با داده‌های استخراج شده،۳ . ردیابی پارامترهای مدل برای گرفتن استحکام بیشتر و نتیجه قابل اعتماد تر. برای حل مباحث بالا از یک پیش پردازش خط‌لوله‌ای استفاده شده است. پیش پردازش شامل چندین گام است که می‌تواند به صورت زیر خلاصه شود.

۱. بریدن یک ناحیه موردنظر(ROI) متناظر با جاده که بخش اصلی تصویر در زیر یک سطر افقی است.

۲. اعمال یک فیلتر دو بعدی ۵*۵ برای کاهش نویز گوسی و بهبود عملکرد لبه‌یابی با توجه به نویز.

۳. کشیدگی هیستوگرام برای بهبود کنتراست تصویر.

۴. محاسبه گرادیان‌های افقی و عمودی و کانولوشن با کرنل سوبل دو بعدی ۵*۵.

۵. محاسبه‌اندازه و فاز گرادیان.

۶. لبه‌یابی و نازک سازی در تعیین نقاطی که لبه هستند توسط آستانه گذاری وفقی بر روی اندازه گرادیان.

۷. جستجوی الگوی نشانگر خط[1] (LMPS) : LMPS : یک نوع از فیلتر کردن هوشمند تصویر است که یک زیر مجموعه از نقاط لبه که پیکربندی مناسبی دارند انتخاب می‌کند تا لبه‌های جعلی ناشی از سایه، دیگر وسایل نقلیه عبوری، درختان، علائم و جزئیات دیگر در صحنه را از بین ببرد.

۸. فیلترینگ مورفولوژیکال ۳*۳ دو بعدی.

بعد از انجام مرحله پیش پردازش (شکل ۲-۱۳ (الف)) حالا برای تطبیق کردن یک مدل به جاده از روش RANSAC استفاده می‌شود. ورودی الگوریتم تطبیق مدل یک نقشه باینری است که همان

[1] Lane Marking Pattern Search

خروجی الگوریتم LMPS است (شکل۲-۱۳(ب)). سپس به طور تصادفی یک زیر مجموعه m نقطه‌ای از آن نقشه انتخاب شده و براساس این زیر مجموعه یک مدل ایجاد می‌شود. برای ردیابی هم از یک فیلتر کالمن[1] استفاده شده است (شکل۲-۱۳(ج)).

الف)

ب)

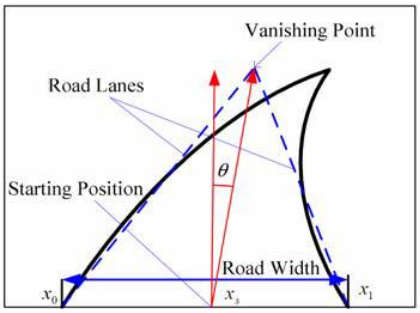
شکل۲-۱۳: نتایج مراحل الف) پیش پردازش، ب)تطبیق کردن و ج) ردیابی[19].

۲-۷ تشخیص خطوط جاده مبتنی بر مدل هندسی و فیلتر گابور

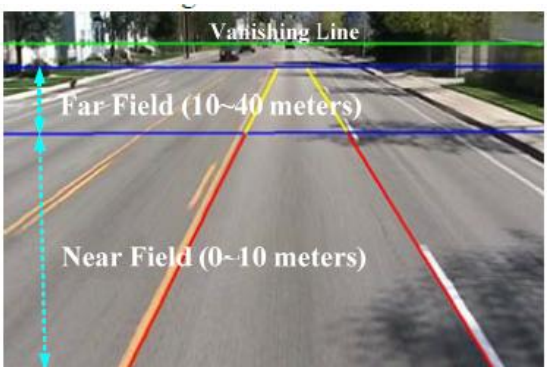
یک الگوریتم تشخیص خط توسط Zhou و همکارانش در[20] ارائه شده است که دارای ۳ بخش است : تولید مدل خط، تخمین پارامترها و سپس تطبیق خط با استفاده از خط وسط و ۳ پارامتر پهنای خط، جهت اصلی و انحنای جاده (شکل ۲-۱۴). پارامترها در مدل با استفاده از مدل خط وسط که محل شروع، جهت و انحنا را فراهم می‌کند کاهش می‌یابد. با استفاده از پهنای خط، مدل خط نهایی بدست می‌آید. این الگوریتم روی میدان دید نزدیک متمرکز شده است (شکل۲-۱۵). برای برآورد کردن

[1] Kalman

پارامترهای مدل خط، نقطه تلاقی[1] تشخیص داده می‌شود. نقطه تلاقی شناسایی شده مبتنی بر تحلیل بافت گابور به برآورد جهت هر پیکسل در ناحیه ROI کمک می‌کند.



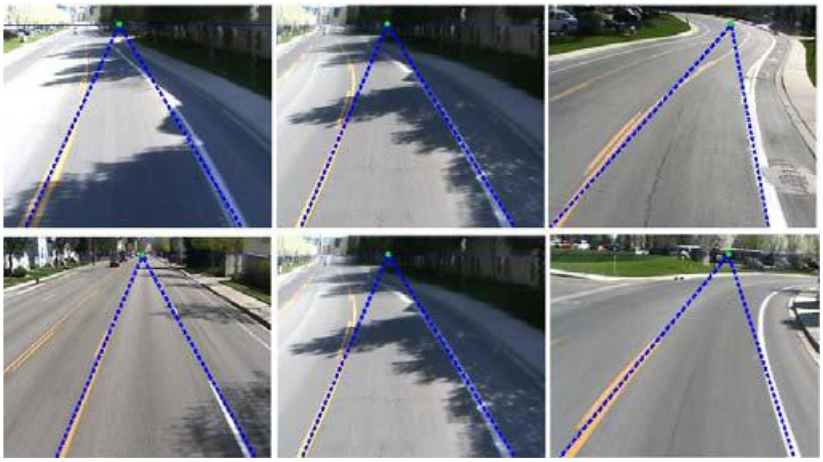
شکل۲-۱۴: تحلیل هندسی خط[20].



شکل۲-۱۵: تعریف میدان دید نزدیک و دور [20].

سرانجام همه جهت گیری های پیکسل‌ها برای محل نقطه تلاقی رای می‌دهند. هر پیکسل برای بیش از یک نقطه تلاقی با توجه به حضور اشیاء رای می‌دهد. هر مدل گوسی برای ماکزیمم کردن درست‌نمایی استفاده شده است. بعد از شناسایی نقطه تلاقی پهنا و جهت خط برآورد شده است. در این مدت مرزهای جاده با استفاده از لبه‌یاب کنی[2] و تبدیل هاف تشخیص داده می‌شود (شکل۲-۱۶).

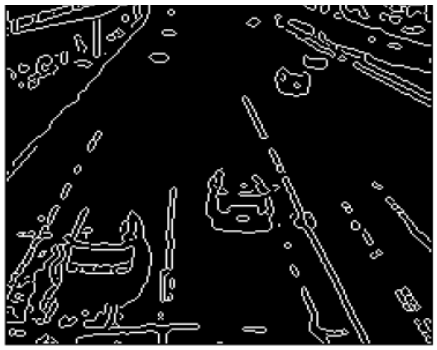
[1] Vanishing Point
[2] Canny



شکل۲-۱۶: نتیجه مرزهای جاده [20].

۲-۸ تشخیص خطوط جاده مبتنی بر لبه‌یابی کنی بهبود یافته با استفاده از بهینه سازی کلونی مورچه

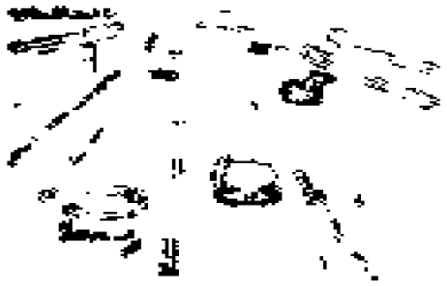
Daigavane و همکاران در [21] یک روش تشخیص خط جاده با لبه‌یابی کنی که با استفاده از بهینه سازی کلونی مورچه[1] (ACO) بهبود یافته، ارائه کرده‌اند. در ابتدا تصویر ورودی به ابعاد 255*255 برای کاهش زمان محاسبه تغییر سایز می‌دهد. تصویر سه کاناله رنگی RGB به تصویر خاکستری یک کاناله تبدیل می‌شود. سپس یک فیلتر میانه[2] برای فیلتر کردن نویز و حفظ لبه استفاده می‌شود. بعد لبه‌ها با استفاده از لبه‌یاب کنی تشخیص داده می‌شوند (شکل۲-۱۷).



شکل ۲-۱۷: لبه‌یابی با لبه‌یاب کنی[21].

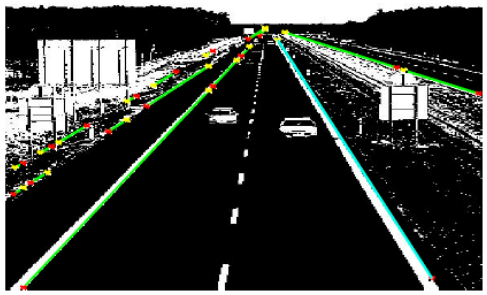
[1] Ant Colony Optimization
[2] Median Filter

تصویر باینری خروجی به عنوان ورودی الگوریتم بهینه سازی کلونی مورچه داده می‌شود و آن اطلاعات لبه اضافی که در جریان لبه‌یابی کنی از دست رفته بودند را تولید می‌کند (شکل۲-۱۸). ACO از یک تعداد مورچه که روی تغییرات شدت روشنایی در تصویر حرکت می‌کنند استفاده می‌کند.



شکل۲-۱۸: لبه‌یابی با بهینه ساز کلونی مورچه [21] .

با استفاده از روش احتمالاتی هر مورچه، تا زمانی که به ناحیه خط دیگر برسد حرکت می‌کند. سپس تبدیل هاف برای پیدا کردن خطوط مستقیم، از لبه‌هایی که به درستی به هم متصل شده‌اند استفاده می‌کند. خطوط مربوط به بالاترین قله‌ها در فضای هاف به عنوان خطوط استخراج می‌شوند (شکل۲-۱۹).

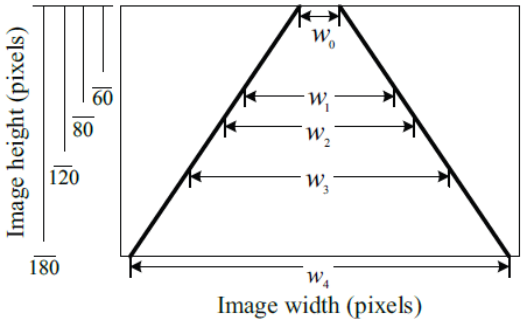


شکل۲-۱۹: تشخیص خط [21] .

۲-۹ تشخیص خطوط جاده مبتنی بر چشم‌انداز در حمل و نقل شهری

Y.-C. Leng و همکاران در [22] یک سیستم تشخیص خط برای جاده‌های شهری پیشنهاد کرده‌اند. لبه‌ها با استفاده از عملگر سوبل تشخیص داده و سپس تبدیل هاف برای تشخیص خطوط مستقیم

پیاده‌سازی می‌شود. در تصاویر جاده خطوط به نظر می‌رسند که قطع شده‌اند. در ارتفاع‌های مختلف تصویر، پهنای خطوط متفاوت است (شکل۲-۲۰). کمترین پهنای خط به عنوان minw و ماکزیمم پهنای خط به عنوان maxw تعریف شده است. پهنای خط در هر ناحیه wi (i=0,1,…,4) باید همیشه بین minw و maxw باشد.



شکل۲-۲۰: تعریف پهنای خط[22].

برای هر کاندید خط چپ و راست یک تطبیق براساس پهنای هر جفت کاندید انجام شده است. اگر پهنای هر جفت کاندید در ارتفاع مختلف رضایت‌بخش نباشد، معیار حذف می‌شود. بعد از استخراج مرزهای چپ و راست، خروج از خط می تواند توسط موقعیت مرز خط‌کشی تعیین شود.

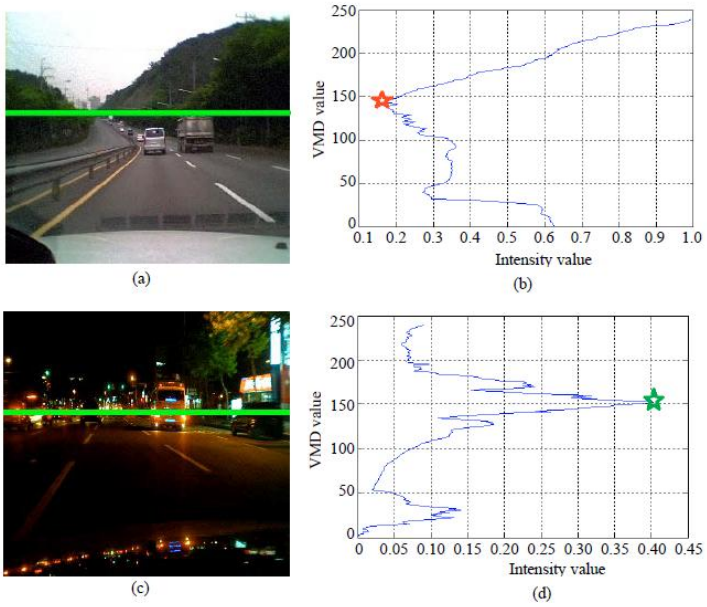
۲-۱۰ تشخیص خطوط جاده در صحنه‌های چالش‌برانگیز

Tran و همکاران در [23] از تشخیص خط افقی برای پیدا کردن زیر ناحیه در تصویر، جایی که خطوط وجود دارند استفاده می‌کنند. از یک روش توزیع متوسط عمودی(VMD)[1] طبق رابطه زیر برای تعیین اولین حداقل[2] که در انحنای بالاتری رخ می‌دهد استفاده کرده‌اند.

$$VMD(i) = \frac{1}{W}\sum_{j=1}^{w} Gr(i,j); i \in (1,H) \qquad (۱-۱)$$

VMD (i) متوسط مقادیر خاکستری سطر i ام است. H ارتفاع تصویر اصلی و W نیز پهنای تصویر اصلی است. Gr (i,j) شدت روشنایی پیکسل در سطر i ام و ستون j ام از تصویر خاکستری است.

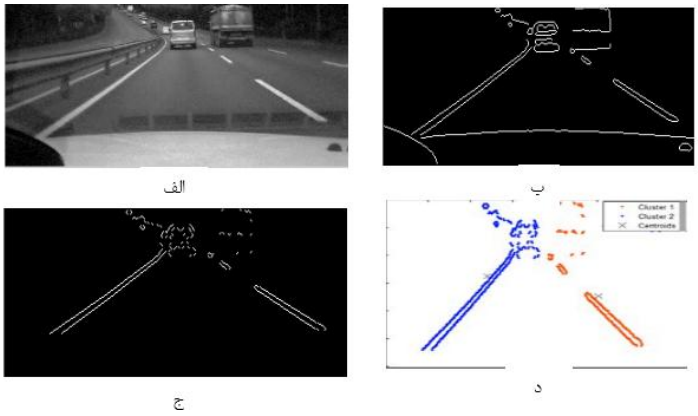
[1] Vertical Mean Distribution
[2] Minimum



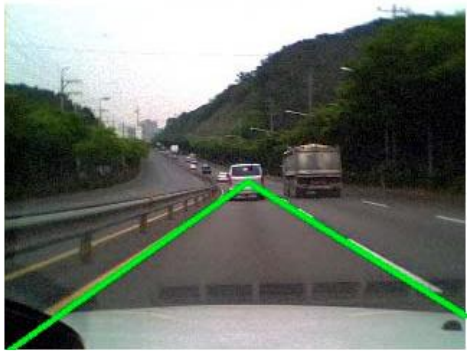
شکل۲-۲۱: (a-d) تشخیص خط افقی [23].

این اولین حداقل به عنوان موقعیت خط افقی در نظر گرفته شده است (شکل۲-۲۱). در روز اولین حداقل و در شب اولین حداکثر دنبال می‌شود.

روی قسمت پایین تصویر لبه‌یاب کنی اعمال می‌شود. بر مبنای یک آستانه ثابت خط‌ها و کاندیدهای نویز در تصویر لبه پیدا خواهند شد. سپس از خوشه‌بندی K-mean (شکل۲-۲۲) و الگوریتم RANSAC برای تشخیص خطوط استفاده شده است. سرانجام مرزهای چپ و راست بدست می‌آیند. نقطه تقاطع بین آن‌ها به عنوان موقعیت خط افقی برای فریم بعدی در دنباله استفاده می‌شود (شکل۲-۲۳).



شکل۲-۲۲: الف) زیر ناحیه تصویر خاکستری، ب) لبه‌یابی کنی، ج) تصویر لبه‌یابی بهبود یافته شده، د) خوشه بندی k-means [23].

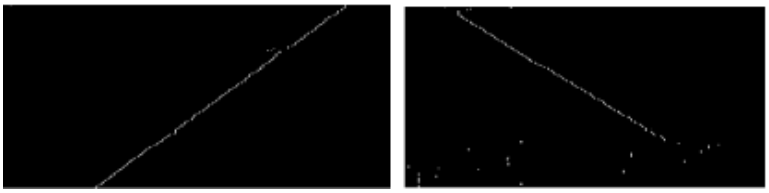


شکل۲-۲۳: نتیجه نهایی و نمایش نقطه تقاطع بین مرزهای چپ و راست[23].

۲-۱۱ تشخیص خطوط جاده مبتنی بر تبدیل هاف

Felix و همکاران در [24] یک الگوریتم ساده و توانا در تشخیص خط‌کشی‌ها و مشخصه‌های نشانگر خط پیشنهاد کرده‌اند که گرایش جهت حرکت خودرو را تعیین می‌کند. در ابتدا از تصویر بدست آمده ناحیه مورد نظر (ROI) استخراج می‌شود. سپس این تصویر به منظور تخمین خط وسط[1] ناحیه‌بندی می‌شود. برای همان تصویر اندازه تصویر پردازش شده است. براساس اندازه تصویر و خط وسط تخمین زده، دو تصویر برای خط‌کشی چپ و راست که شامل حاشیه درونی خط است، تولید می‌شوند (شکل۲-۲۴).

الف

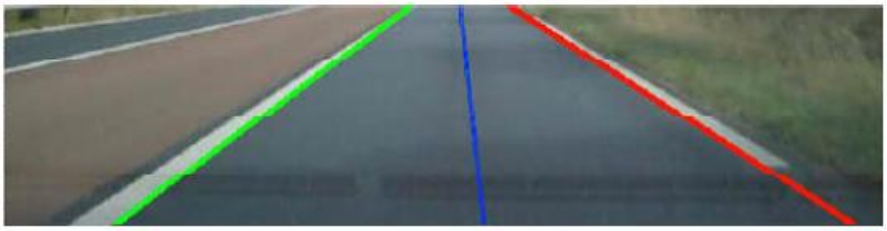


ب

شکل۲-۲۴: مرزهای داخلی خطوط خط‌کشی. الف) اندازه تصویر، ب) مرزهای داخلی برای هر خط[24].

[1] Mid-lane

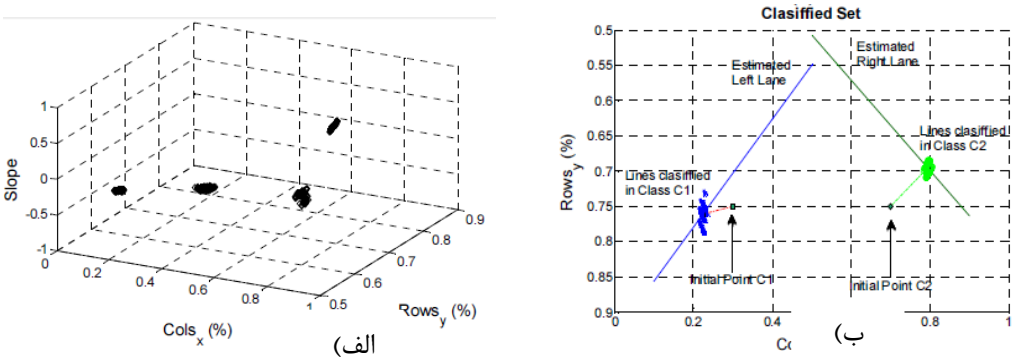
بر روی این تصاویر الگوریتم تبدیل هاف اعمال شده است. در این مرحله دو خط تشخیص داده خواهند شد و روی تصویر اصلی نقاشی می‌شوند. زمانی که خطوط تشخیص داده می‌شوند، الگوریتم مشخصه‌های نشانگرهای خطوط را با استفاده از روش ارائه شده اولیه تشخیص می‌دهد. براساس نشانگرهای خط تشخیص داده شده، خط وسط واقعی با موقعیت‌یابی در فاصله‌ای در وسط آن‌ها می‌تواند محاسبه شود. همچنین این خط وسط هم روی تصویر نقاشی می‌شود (شکل۲-۲۵). با استفاده از این اطلاعات الگوریتم می‌تواند تمایل جهت حرکت را با توجه به روش‌های اولیه ارائه شده تعیین کند.



شکل۲-۲۵: مرزهای جاده و خط وسط تشخیص داده شده [24].

۲-۱۲ تشخیص خطوط جاده مبتنی بر کلاسه‌بند وفقی و بدون نظارت

Herrera و همکاران در [25] یک روش تشخیص خط را براساس کلاسه‌بند وفقی و بدون نظارت معرفی می‌کنند. این کلاسه‌بند به این دلیل انتخاب شده است که پارامترهای خط در جاده را محقق، نمی‌دانسته، اگرچه می‌دانسته که خطوط وجود دارند و فقط آن‌ها نیاز به طبقه‌بندی دارند. معماری این الگوریتم در ۳ فاز انجام شده است. در فاز اول : یک عملیات پیش‌پردازش روی تصویر ورودی انجام می‌شود. به این صورت که روشنایی تصویر اندازه گرفته شده و سپس یک منطقه مورد علاقه (ROI) تعریف شده است.



شکل۲-۲۶: کلاسه‌بند خطی. الف) مجموعه ورودی، ب) خطوط کلاسه بندی شده [25].

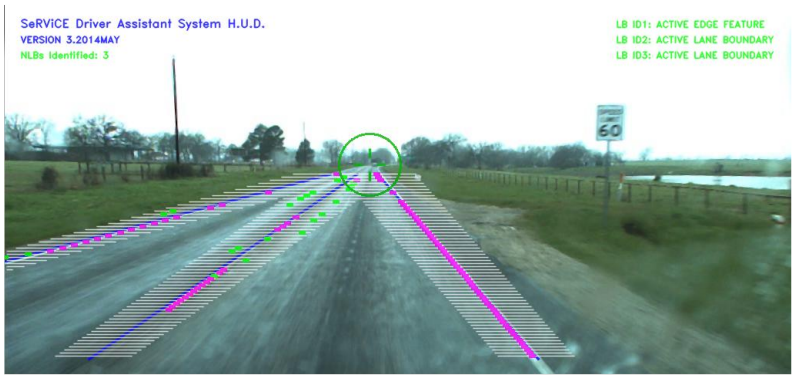
سرانجام یک تصویر جدید برای تشخیص خطوط جاده از فلیترکردن تصویر باینری بدست آمده است. در فاز دوم : یک تبدیل هاف روی تصویر اعمال شده است. نتایج خطوط توسط یک کلاسه‌بند وفقی و بدون نظارت طبقه‌بندی می‌شوند (شکل۲-۲۶). آن‌ها فرض کرده‌اند که وسیله‌نقلیه روی جاده و در مرکز خط‌کشی است. کلاس‌ها به دو نوع تقسیم می‌شوند : اول یکی خط سمت چپ جاده و دوم یکی خط سمت راست جاده. همچنین هر خط شامل ۲ پارامتر است : شیب خط و نقطه وسط خط. این پارامترها در حال تغییر هستند به منظور اینکه خطوط واقعی تنظیم شوند.

در نهایت یک فیلتر کالمن، نویز را در پارامترهای خطوط هر کلاس کاهش می‌دهد. این خطوط در تصویر اصلی نشان داده شده و آنها جاده را می‌سازند. موقعیت وسیله‌نقلیه در فاز آخر تخمین زده می‌شود. از یک ماسک به منظور برجسته‌کردن خطوط استفاده شده تا به کاربر تصویر بهتری نشان داده شود. اشکال سیستم این است که نمی‌تواند تغییرات در خط را تشخیص دهد.

۲-۱۳ تشخیص خطوط جاده مبتنی بر ترکیب ویژگی‌های سطح پایین تصویر

Li و همکاران در [26] یک روش تشخیص خط با استفاده از ترکیب ویژگی‌های سطح پایین تصویر ارائه کرده‌اند. برای تشخیص لبه‌ها در منطقه مورد نظر(ROI) از لبه‌یاب Canny استفاده شده است. خطوط مستقیم از خروجی باینری استخراج‌کننده لبه Canny با استفاده از تبدیل هاف تشخیص داده می‌شود. برای از بین بردن اثر نویز ویژگی‌های ماکزیمم محلی در امتداد مرز خط برآورد شده دنبال

می‌شود. سپس الگوریتم RANSAC برای از بین بردن Outlierها اعمال می‌شود (شکل۲-۲۷). در نهایت ویژگی ماکزیمم محلی به یک خط راست فیت شده است. در مرحله بعدی فیلتر Kalman برای ردیابی خط در فریم‌های باقیمانده به کار می‌رود.



شکل۲-۲۷: نتیجه پالایش ویژگی های ماکزیمم محلی(نقاط سبز نشان دهنده outlierها و نقاط قرمز نشان دهنده inlierها) [26].

۲-۱۴ تشخیص خطوط جاده مبتنی بر الگوریتم آستانه محلی متقارن

Ozgunalp و همکاران در [27] از الگوریتم آستانه محلی متقارن[1] (SLT) برای تشخیص و ردیابی خط استفاده کرده‌اند. این الگوریتم مبتنی بر خاصیت تغییر از یک حالت به حالت دیگر مثلا تیره-روشن-تیره است. برای هر نقطه ورودی متوسط مقدار شدت روشنایی همه پیکسل‌های سمت چپ (در محدوده سمت چپ) همان سطر محاسبه می‌شود و سپس متوسط شدت روشنایی برای سمت راست هم محاسبه می‌شود. اگر شدت روشنایی پیکسلی بزرگتر از متوسط شدت روشنایی محدوده چپ و راست بود. سپس آن به عنوان یک نقطه ویژگی از خط در نظر گرفته و در نقشه ویژگی برچسب زده می‌شود. سپس یک تحلیل اجزای متصل یک بعدی برای حذف نویز نمک و فلفل صورت می‌گیرد. یکی از دو طرف نقاط ویژگی برای مرزهای خط جستجو می‌شود. اگر جهت گیری هر دو مرز نزدیک هم

[1] Symmetrical Local Threshold

باشند، متوسط زاویه‌ها به عنوان جهت مرز خط شناسایی شده در نظر گرفته می‌شود. در غیر این صورت نقطه تخمین زده از نقشه ویژگی حذف خواهد شد. برای کاهش اثر نویز از یک تبدیل فاصله[1] استفاده شده است. فاصله هر نقطه ویژگی از نزدیک‌ترین نقطه غیر ویژگی در نقشه ویژگی محاسبه می‌شود و یک وزن به آن اختصاص داده می‌شود. بنابراین نشانگرهای خط وزن بالاتری از پیکسل‌های جدا شده دارند. همچنین مرکز نشانگر خط یک مقدار بالا دارد. در نهایت تبدیل هاف اعمال شده و مقادیر rho برای مقادیر Theta نزدیک به جهت نقاط ویژگی محاسبه می‌شود (شکل۲-۲۸). در این صورت سرعت محاسبات افزایش یافته و هر دو خط چپ و راست در یک فریم شناسایی و ردیابی می‌شوند.

الف) تصویر ورودی ب) بخشی از نقشه ویژگی استخراج شده

ج) خروجی تبدیل فاصله چ) فضای تبدیل هاف برای همه زوایا

ح) فضای تبدیل هاف فقط برای زوایای نقاط ویژگی خ) ماسک فضای هاف

شکل ۲-۲۸ : مثال تبدیل هاف [27].

[1]Distance transform

فصل سوم : تئوری پژوهش

۳-۱ مقدمه

تشخیص لبه یک گام پیش پردازش برای بسیاری از الگوریتم‌های پردازش تصویر مثل بهبود تصویر، ناحیه‌بندی تصویر، ردیابی و کدینگ تصویر و ویدئو است. لبه‌یابی به طور قابل ملاحظه‌ای مقدار اطلاعات را کاهش می‌دهد و اطلاعات بی‌فایده را فیلتر می‌کند، در حالی که خواص مهم ساختاری یک تصویر را حفظ می‌کند[28]. لبه‌های تصویر به عنوان مهم‌ترین ویژگی‌های تصویر درنظر گرفته می‌شوند که اطلاعات با ارزشی برای درک تصویر انسانی فراهم می‌کنند.

۳-۲- تعریف لبه‌یابی

در الکترونیک به تبدیل یک سیگنال دیجیتال از سطح منطقی بالا به پایین (۱ به ۰) یا پایین به بالا (۰ به ۱) لبه‌ی سیگنال می‌گویند. هدف از آشکارسازی لبه در پردازش تصاویر نشانه‌گذاری نقاطی از یک تصویر است که در آنها شدت روشنایی[1] به تندی تغییر می‌کند. تغییرات تند در خصوصیات تصویر معمولاً نماینده‌ی رویدادهای مهم و تغییرات در خصوصیات محیط هستند.

الگوریتم‌های زیادی به منظور آشکارسازی لبه‌ها در مبحث پردازش تصویر ارائه شده‌اند. اساس بسیاری از این الگوریتم‌ها و روش‌ها مشتق‌گیری از تصویر است. روش‌های مبتنی بر مشتق‌گیری به دو گروه زیر تقسیم می‌شوند :

۱. مبتنی بر گرادیان : روش گرادیان، لبه‌ها را به وسیله جستجو برای اولین حداکثر و اولین حداقل در اولین مشتق تصویر تشخیص می‌دهد. مانند : لبه‌یاب Sobel، Prewitt، Robert .
۲. مبتنی بر لاپلاس : روش لاپلاس نقاط عبور از صفر را در مشتق دوم برای پیدا کردن لبه‌ها جستجو می‌کند.

[1] Intensity

از دیگر الگوریتم‌های کارامد در این حوزه، آشکارساز لبه کنی است که در سال ۱۹۸۳ مطرح و با اقبال شدید صاحب نظران و استفاده کنندگان روبه رو شد. که به خاطر داشتن قابلیت دنبال کردن لبه‌ها و نیز توانایی حذف نویز تصویر با کمک فیلتر گوسی کاربرد زیادی دارد. این الگوریتم نیز از آن دسته روش‌هایی که از مشتق‌گیری روی عکس استفاده می‌کنند، تقسیم‌بندی می‌شود. الگوریتم کنی در آشکارسازی لبه به الگوریتم بهینه معروف است. همچنین الگوریتم‌هایی برای آشکارسازی لبه‌ها در حوزه‌ی فرکانس ارائه شده است. الگوریتم Log در حوزه تبدیل لاپلاس، نقاط عبور از صفر مشتق دوم تصویر را به عنوان لبه در نظر می‌گیرد. این لبه‌یاب به دلیل استفاده از فیلتر گوسی، حساسیت کمی به نویز ضربه دارد. الگوریتم‌هایی نیز وجود دارند که برای آشکارسازی لبه‌های تصویر از تبدیل موجک[1] استفاده می‌کنند. [29] اخیراً تعدادی از عملگرهای لبه‌یابی را در نرم‌افزار MATLAB مقایسه کرده است.

در تمام موارد بالا، پس از اعمال الگوریتم لبه‌یاب، عمل آستانه‌گیری انجام شده، سپس تصویر دوسطحی لبه‌ها برای تک‌پیکسلی شدن، نازک‌سازی می‌شود. روش‌های جدیدی همچون روش مورفولوژیکال[2] نیز وجود دارند که از عملگرهای صریح ریاضی و غیر مشتق گیری استفاده می‌کنند[30].

۳-۳ لبه‌یاب Sobel

اپراتور سوبل، که گاهی اوقات اپراتور سوبل-فلدمن یا فیلتر سوبل نامیده می‌شود، در پردازش تصویر و بینایی کامپیوتر به ویژه الگوریتم‌های تشخیص لبه مورد استفاده است. این اپراتور توسط ایروین سوبل و گری فلدمن در یک سخنرانی در آزمایشگاه هوش مصنوعی دانشگاه استنفورد (SAIL) در سال ۱۹۶۸ مطرح شد. این عملگر از دو کرنل 3*3 استفاده می‌کند که برای محاسبه تقریب‌های مشتق اول در دو

[1] Wavelet Transform
[2] morphological Edge Detector

جهت افقی و عمودی با تصویر اصلی کانولوشن می‌شود. اگر ما A را به عنوان تصویر اصلی تعریف کنیم، G_x و G_y گرادیان‌های افقی و عمودی تصویر به ترتیب هستند.

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A} \qquad ۱-۳$$

علامت ستاره عملگر کانولوشن دو بعدی را انجام می‌دهد.

سپس با استفاده از رابطه زیر، اندازه گرادیان محاسبه می‌شود:

$$|G| = \sqrt{{G_X}^2 + {G_Y}^2} \qquad ۲-۳$$

برای به دست‌آوردن جهت لبه‌های تصویر، با استفاده از مقدار گرادیان در راستای x و y که در مرحله‌ی قبل محاسبه شده است. فرمول زیر برای محاسبه‌ی جهت لبه‌ها استفاده می‌شود.

$$theta = tan^{-1}\left(\frac{G_y}{G_x}\right) \qquad ۳-۳$$

مرحله آخر از الگوریتم لبه‌یابی، انتخاب حد آستانه مناسب است که می توان با صحت بالایی به لبه‌های قوی تصویر دست پیدا کرد.

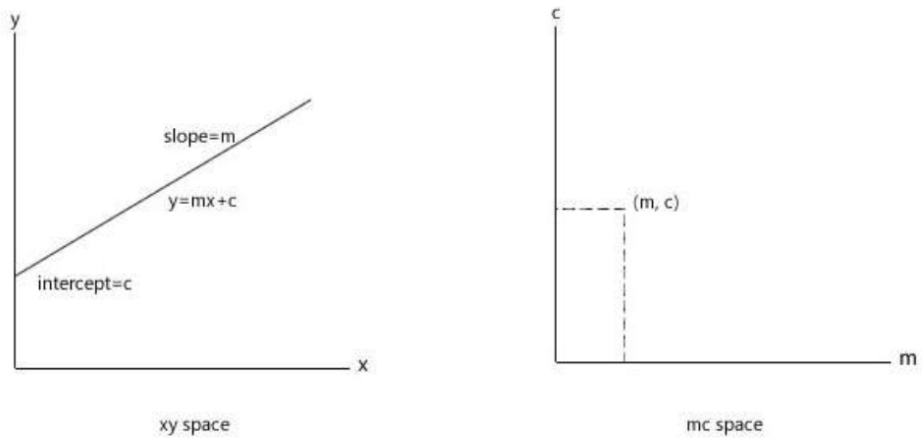
۳-۴ تبدیل هاف

تبدیل هاف[1]، یک تکنیک استخراج ویژگی می‌باشد که در تحلیل تصویر، بینایی کامپیوتر و پردازش تصویر دیجیتال استفاده می‌شود. تبدیل هاف آنطور که امروزه استفاده می‌شود، پس از کار هاف در ۱۹۶۲، توسط هارت[2] در ۱۹۷۲ با نام تبدیل هاف تعمیم یافته، ابداع شد. این تبدیل در بین پژوهشگران بینایی کامپیوتر، توسط کار بالارد[3] در سال ۱۹۸۱، محبوبیت پیدا کرد. عنوان کار بالارد، « تبدیل هاف تعمیم یافته برای آشکار سازی شکل های دلخواه » بود.

[1] Hough Transform
[2] Hart
[3] Ballard

تبدیل هاف ابزاری جادویی می‌باشد که به شما این امکان را می‌دهد که نه تنها خطوط را شناسایی کنید، بلکه شکل‌های دیگر را نیز می‌توانید به خوبی شناسایی کنید. به کمک تبدیل هاف می‌توان شیب و محل تقاطع خطوط را تعیین کرد. تبدیل هاف به هر پیکسل تصویر اجازه می‌دهد که رای دهد. و بخاطر ویژگی‌های ریاضی تبدیل، این رای‌گیری[1] به ما اجازه می‌دهد که خطوط بارز در تصویر را شناسایی کنیم.

یک خط مجموعه‌ای از نقاط است و مدیریت و اداره مجموعه‌ای از نقاط مشکل‌تر از اداره یک نقطه مجرد است. اولین چیزی که ما یاد می‌گیریم این است که چطور یک خط را بدون از دست دادن تمام اطلاعات آن به صورت یک نقطه مجرد نمایش دهیم.

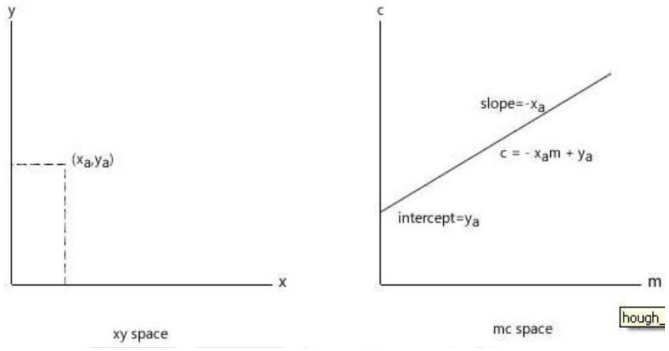


شکل۳-۱ : تبدیل خط در فضایxy به نقطه در فضای پارامتر

همانطور که در شکل بالا نشان داده شده است، هر خطی دو کمیت مختص به خود دارد، شیب[2] و عرض از مبدا[3] ، با این دو عدد ما می‌توانیم یک خط را به طور کامل بیان کنیم. بنابراین فضای پارامتر (یا فضای mc)، به این صورت ساخته می‌شود. بنابراین هر خط در فضای xy برابر است با یک نقطه منفرد در فضایmc.

اما یک نقطه در فضای xy را چگونه می توان در فضای mc نمایش داد؟

[1] Voting
[2] Slope
[3] Intercept



شکل۳-۲ : تبدیل نقطه در فضای xy به خط در فضای پارامتر

می‌دانید که از یک نقطه، تعداد نامحدودی خط می‌تواند عبور کند، بنابراین برای هر خط گذرنده از (x_a,y_a)، نقطه ای در فضای mc وجود خواهد داشت.

معادله خط گذرنده از (x_a,y_a) :

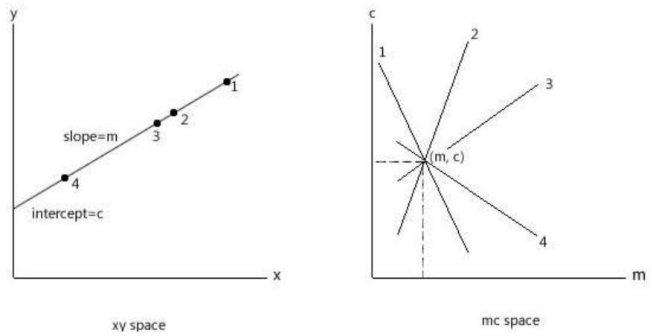
$$y_a = mx_a + c \qquad ۴-۳$$

با آرایش مجدد معادله فوق داریم :

$$c = -mx_a + y_a \qquad ۵-۳$$

رابطه فوق، معادله یک خط در فضای mc می‌باشد. بنابراین یک نقطه در فضای xy برابر است با یک خط در فضای mc.

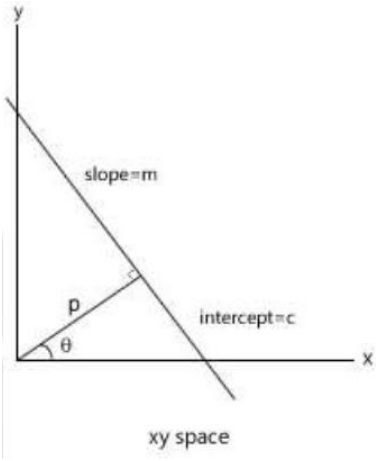
یک خط آشکار شده از تصویر را در نظر بگیرید. برای هر نقطه غیر صفر روی این خط، خطوط در فضای mc را رسم می‌کنیم. مسلما برخی از خطوط همدیگر را قطع خواهند کرد. مکان این تقاطع، پارامترهای خط آشکار شده در فضای xy را نشان می‌دهد.



شکل ۳-۳ : تبدیل چند نقطه در فضای xy به چند خط در فضای پارامتر

اکنون ایده اصلی نحوه عملکرد تبدیل هاف بیان شد. اما این ایده، یک ایراد واضح دارد و آن این است که ممکن است شیب خط بینهایت شود (خط عمودی شود). در این‌صورت استفاده از این تبدیل عملی نخواهد شد.

مشکل فوق با استفاده از روش دیگر برای تعیین پارامتر حل می‌شود. یعنی به جای ترکیب شیب عرض از مبدا خطوط، از ترکیب بردار نرمال(بردار عمود بر خط) استفاده می‌کنیم. در این نمایش، یک خط از ۲ پارامتر تشکیل می‌شود. زاویه Θ و فاصله p. طول بردار نرمال از مبدا تا خط p می‌باشد و Θ زاویه این بردار نرمال با محور X است.



شکل ۳-۴: استفاده از نرمال و زاویه نرمال در تبدیل هاف

زاویه Θ، بین ۹۰- تا ۹۰ درجه می‌تواند باشد و مقدار طول p می‌تواند از صفر تا مقدار قطر تصویر باشد. محور افقی دارای $\Theta=0$ می باشد که p آن مثبت است. محور عمودی یا دارای $\Theta=90$ می‌باشد که p آن مثبت است و یا دارای $\Theta=-90$ می‌باشد که p آن منفی است. توجه کنید که مبدا مختصات هاف، همان مبدا تصویر است.

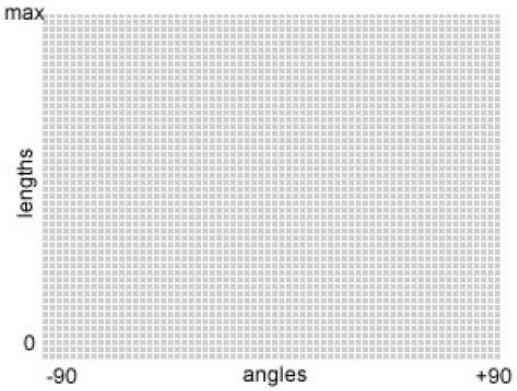
در این نمایش، معادله خط به صورت زیر است :

$$P=x_1\cos\Theta + y_1\sin\Theta \qquad ۶-۳$$

که(x_1,y_1) نقطه‌ای است که خط از آن عبور می‌کند.

با این رابطه جدید، همچنان یک خط در فضای xy برابر است با یک نقطه در فضای pΘ، ولی یک نقطه در فضای xy برابر است با یک منحنی سینوسی در فضای pΘ.

با ایده فوق، اکنون آماده هستیم که تبدیل هاف را پیاده‌سازی کنیم. ایده این است که هر پیکسل رای دهد. بنابراین آرایه‌ای از سلول‌های انباشتگر[1] ایجاد می‌شود. در اینجا ما این آرایه از سلول‌ها را به صورت دو بعدی در نظر می‌گیریم. محور افقی مقادیر مختلف Θ می باشد و محور عمودی مقادیرp است.



شکل۳-۵: ساخت انباشتگر فضای پارامتر در تبدیل هاف

هاف فقط روی تصاویر با لبه آشکار شده کار می‌کند نه روی تصاویر معمولی(در واقع فقط یک تصویر لبه داریم و نمی‌دانیم که اشیای مورد نظر کجا هستند).

از پیکسل‌های دارای مقدار صفر صرفنظر می‌کنیم چون آن‌ها نمی‌توانند یک خط باشند.

هر پیکسل غیر صفر روی لبه، یک منحنی سینوسی در فضای pΘ تولید می‌کند. به این ترتیب که ابتدا Θ را 90- درجه می‌گیریم و مقدارp متناظر با آن را محاسبه می‌کنیم. این مقادیر محاسبه شده به سلول انباشتگر(Θ,p) رای می‌دهد. سپس مقدار Θ را یک واحد افزایش می‌دهیم و p بعدی را محاسبه می‌کنیم و باز رای‌گیری انجام می‌شود. این فرآیند را ادامه می‌هیم تا Θ=90 شود و برای هر محاسبه، رای‌گیری را انجام می‌دهیم.

دلیل استفاده از انباشتگر این است که از یک نقطه بینهایت خط عبور می‌کند و ما قادر نیستیم که بینهایت اندازه‌گیری انجام دهیم.

[1] Accumulator

اکنون یک منحنی سینوسی مانند در فضای $p\Theta$ برای هر پیکسل داریم. برخی از سلول‌ها تعداد زیادی رای آورده‌اند. نقاطی که به این سلول رای داده‌اند روی خطی با پارامترهای آن سلول قرار دارند.

به عنوان مثال در شکل زیر، دو مکان خیلی روشن تر می‌باشند که نشان دهنده دو خط برجسته در تصویر می‌باشد.

شکل۳-۶: یک تبدیل هاف نمونه

دقت داشته باشید که تبدیل هاف به تعداد سلول‌های انباشتگر بستگی دارد، که هر چه بیشتر باشد، دقت نیز بیشتر می‌شود.

فصل چهارم : FPGA

۴-۱ مقدمه

امروزه پردازش سیگنال دیجیتال در محدوده بسیار وسیعی از کاربردها مانند پردازش صوت و تصویر، رادیوهای دیجیتال، رادار، موبایل، سیستم موقعیت یاب جهانی[1] و .... استفاده می‌شود. توسعه این کاربردها بر روی سخت افزار یک موضوع فعال در سه دهه اخیر بوده است، بنابراین یک سری از پردازنده‌های خاص برای پردازش سیگنال دیجیتال به نام پردازنده‌هایDSP پدیده آمد. این پردازنده‌های DSP در بسیاری از کاربردهای پردازش سیگنال دیجیتال مورد استفاده قرار می‌گرفت اما در بعضی موارد به خوبی پاسخگوی مصرف توان، فضای سخت‌افزاری و سرعت پردازش نبود. در سالهای اخیر FPGAها به عنوان مناسب‌ترین ساختار مداری برای اجرای الگوریتم‌های پردازش سیگنال دیجیتال به ویژه در کاربردهای بلادرنگ معرفی شدند[31]. در واقع ویژگی موازی بودن عملیات در FPGA است که در حال حاضر آن را به عنوان بهترین انتخاب در پردازش سیگنال دیجیتال با حجم محاسبات زیاد و پیچیده معرفی کرده است. FPGA به طراح اجازه می‌دهد تا طرح دیجیتال خود را آنچنان که می خواهد و با هر حجم و پیچیدگی لازم، طراحی کند. FPGAها مجموعه ای از بلوک‌های دیجیتال ساده تا پیشرفته است که با طراحی مناسب می‌تواند الگوریتم‌های پیچیده را به خوبی حل نماید. اگر عملیات الگوریتم‌های اصلی پردازش سیگنال دیجیتال مانند FFT[2]، DCT[3]، فیلترهایFIR[4]،IIR[5]و ... را به طور دقیق مطالعه کنیم، ملاحظه می‌شود که امکان اجرای این الگوریتم‌ها بر روی سخت‌افزار FPGA به منظور به دست آوردن سرعت محاسبات بیشتر به راحتی امکان‌پذیر است.

FPGA دارای محاسن زیر است:

[1] Global positioning system
[2] Fast Fourier Transform
[3] Discrete cosine transform
[4] Finite Impulse Response
[5] Infinite Impulse Response

۱. مدارهای دیجیتال پیچیده به آسانی در آن پیاده‌سازی می‌شوند.

۲. تست مدار سریع است.

۳. برای تولید کم، ارزان تمام می‌شود.

۴. متناسب با نیاز، تغییرات لازم را در طراحی می‌توان داد و مجدداً FPGA را با ساختار جدید برنامه‌ریزی نمود.

۵. قابل برنامه‌ریزی توسط کاربر است.

همچنین معایب آن عبارتست از:

۱. سطح سیلیکن FPGA به طور بهینه استفاده نمی‌شود.

۲. تأخیر و توان مصرفی آن نسبت به ICهایی که در کارخانه ساخته می‌شوند بیشتر است.

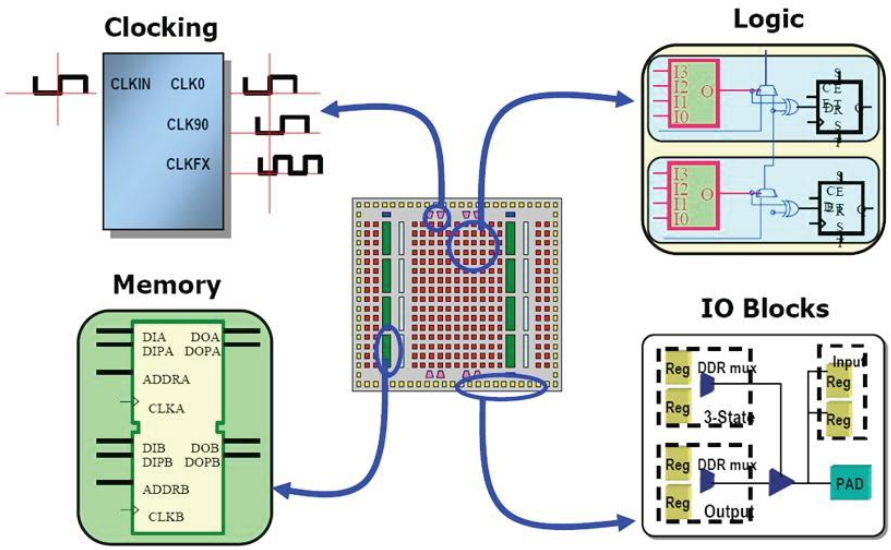
با توجه به محاسن فوق، طراحی سیستم‌های دیجیتال جدید با VHDL و پیاده‌سازی آن بر روی FPGA روز به روز بیشتر می‌شود، به طوری که امروزه سازندگان مختلفی از جمله شرکت‌های Xilinx، Altera، AT&T، Quicklogic، Actel و ... انواع مختلف FPGAها را تولید و با ابزارهای برنامه‌ریزی FPGA، به بازار عرضه نموده‌اند.

۴-۲ ساختار FPGA

ساختار درونی یک FPGA پیشرفته به صورت شکل ۴-۱ می‌باشد. یک FPGA در ساختار داخلی خود، بلوک های دیجیتالی قابل برنامه ریزی را برای طراح فراهم می‌کند که شامل هزاران بلوک ترکیبی و فلیپ فلاپ به همراه I/O[1] برای ارتباط با دنیای خارج، و یک منبع کلاک مستقل است.

[1] Input/Output

یک تراشه FPGA به تنهایی و جدای از ارتباطات با دنیای خارج معنی پیدا نمی‌کند و لازم است که به راحتی با تراشه‌های دیگر و یا سیگنال‌های خارجی ارتباط داشته باشد. به منظور دستیابی به این هدف، سازندگان FPGA تلاش و سرمایه گذاری بسیار وسیعی برای بالا بردن انعطاف پذیری بلوک‌های I/O تراشه‌های خود انجام داده‌اند.



شکل۴-۱ : ساختارداخلی یک FPGA نوعی.

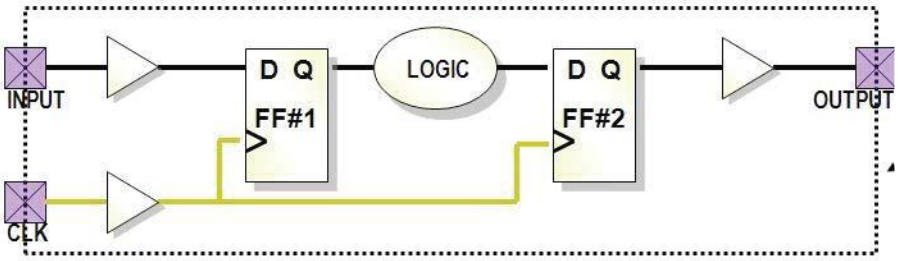
در حال حاضر پورت‌های FPGA می‌توانند به عنوان ورودی، خروجی یا هم ورودی و هم خروجی مورد استفاده قرار گیرند. همچنین تکنیک‌های جدیدی همچون کلاک کردن مدار در هر دو لبه بالارونده و پایین رونده برای افزایش پهنای باند در FPGAها به کار می‌رود.

تمام عناصری که در شکل۴-۱ نشان داده شده است، کمتر از ۲۰٪ فضای سیلیکون درون چیپ FPGA را شامل می‌شود. چیزی که در شکل نشان داده نشده، حجم وسیعی از اتصالات داخلی قابل برنامه ریزی و مدارهای کمکی است که بلوک‌های دیجیتال را برنامه‌ریزی می‌کند تا به یک قطعه دیجیتال قابل استفاده تبدیل شود.

برای حل مشکلات و محدودیت‌های طراحی بر روی قطعات سیلیکونی، سازندگان FPGAها همچنین از هسته‌های هوشمند[1] با اتصالات سخت افزاری در درون تراشه‌ها برای ساخت توابعی که در کاربردهای پردازش سیگنال دیجیتال زیاد کاربرد دارد، استفاده می‌کنند. این بلوک‌های غیر قابل برنامه‌ریزی شامل پردازنده‌های همه منظوره[2]، واسط‌های سریال[3] با سرعت بالا، بلوک‌های محاسباتی ریاضی و واحدهای کنترل دسترسی میانی اترنت[4] است.

۴-۳ اصول یک طراحی دیجیتال

شکل ۴-۲ ساختار پایه‌ای یک طراحی دیجیتال نمونه در درون FPGA ، که از بلوک‌های دیجیتال ترکیبی در بین آرایه‌ای از فلیپ فلاپ‌ها تشکیل شده است، را نشان می‌دهد. این ساختار یک ساختار اصولی است که در بسیاری از کاربردهای پردازش سیگنال دیجیتال از آن استفاده می‌شود.



شکل۴-۲ : ساختار پایه یک طراحی دیجیتال در FPGA.

در شکل ۴-۲ بلوک دیجیتال می‌تواند هر زیر مجموعه‌ای از مدارات دیجیتال باشد، که در آن وضعیت فعلی خروجی تنها به وضعیت فعلی ورودی و کلاک مدار بستگی دارد. بنابراین تمام توابع دیجیتال مانندAND، OR و هر ترکیب پیچیده‌ای از آن‌ها (مانند مالتی پلکسرها، دیکدرها، انکدرها و...) می‌تواند در قسمت بلوک دیجیتال قرار بگیرد. به این نکته باید دقت شود که توابع دیجیتال با هر پیچیدگی

[1] Intellectual Property (IP) core
[2] General purpose processors
[3] Serial Interface
[4] Ethernet Medium Access Control (MAC) unit

دلخواه، مانند مالتی پلکسرها، دیکدرها، انکدرها و... ، با استفاده از این بلوک‌های پایه قابل ساخت است. قسمت INPUT می تواند یک یا چند بیتی باشد. در این مدار همچنین یک بخش کلاک (CLK)[1] وجود دارد که شامل یک نوسانساز موج مربعی با فرکانس ثابت است. هر دو بلوک فلیپ‌فلاپ که می‌تواند شامل چندین هزار فلیپ‌فلاپ باشد، توسط یک منبع کلاک تغذیه می‌شود و هر زمان که سیگنال کلاک در لبه بالارونده باشد، سیگنال را از ورودی D به خروجی Q انتقال می‌دهد.

نکته مهمی که طراح باید آن را مدنظر قرار دهد، این است که تاخیر بین هر کدام از ورودی‌ها به سمت بلوک دیجیتال و خروجی آن، کمتر از یک پریود کلاک مدار باشد. اگر این تاخیر بیش از مقدار یاد شده باشد، مقدار قبلی در ورودی فلیپ‌فلاپ به خروجی آن انتقال داده می‌شود که باعث اختلال در پردازش زمانی سیستم می‌گردد. همانطور که بعداً نشان خواهیم داد این روند به صورت اتوماتیک توسط نرم افزارهای شبیه‌سازی انجام می‌شود و طراح فقط باید مشخصات رفتاری و ساختاری مدار را مدنظر قرار دهد.

در بعضی موارد ممکن است طراح بخواهد کلاک ورودی فلیپ‌فلاپ را از خروجی یک کلاک ترکیبی تغذیه کند، و ممکن است این طراحی نتایج خوبی هم در طول شبیه‌سازی بدهد. اما بعداً با ایجاد تغییرات کوچک در تاخیرهای مختلف سیگنال‌ها، ممکن است زمان‌بندی مدار دچار اغتشاش شود.

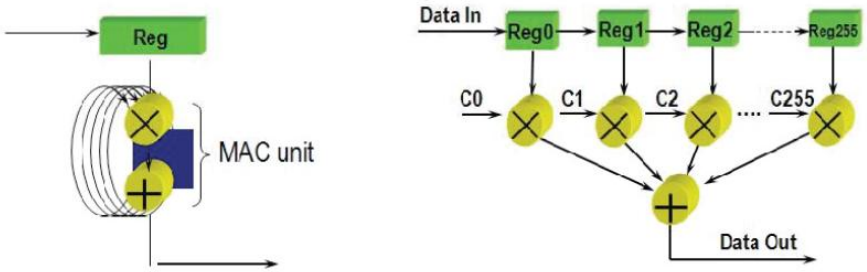
بنابراین باید در مدارات همزمان جهت تغییرات زمانی تاخیرها، برای ماکزیمم زمان ممکن هر سیگنال یک مرز در نظر گرفته شود.

۴-۴ ویژگی‌های برتر FPGAها در پردازش سیگنال دیجیتال

مهمترین ویژگیFPGAها در مقابل پردازنده‌های DSP اجرای عملیات پردازش سیگنال دیجیتال به صورت موازی است، یعنی ترکیب توابع سخت‌افزاری که می‌توانند به‌صورت همزمان در قسمت‌های

[1] Clock

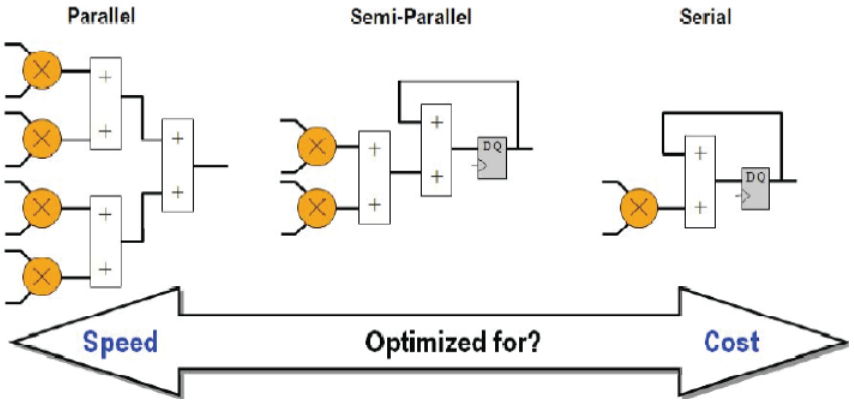
مختلف چیپ عمل کنند. شکل ۴-۳ نشان می‌دهد که چگونه یک فیلتر FIR با ۲۵۶ ضریب توسط هر کدام از این سخت‌افزارها می‌تواند پیاده‌سازی شود.



شکل۴-۳ : مقایسه پیاده‌سازی یک فیلترFIR بر روی دو سخت افزار: سمت راست:FPGA، سمت چپ:DSP.

برای بدست آوردن خروجی فیلتر با استفاده از پردازنده DSP به ۲۵۶ کلاک نیاز است، در حالی‌که FPGA با هر کلاک ورودی، خروجی جدید را تولید می‌کند. هر چند که سرعت کلاک پردازنده‌های DSP نسبتا سریع تر از FPGAها است، اما اگر بخواهیم تعداد زیادی از فیلترهای FIR با ضرایب طولانی را پیاده کنیم و همزمان عملیات جمع‌ها و ضرب‌ها انجام شود، آنگاه خواهیم دید که DSPها قابل مقایسه با FPGAها نیستند[32].

ویژگی مهم دیگر FPGAها، انعطاف‌پذیری آنها برای انتخاب بین سرعت و فضای سخت‌افزاری در فرآیند طراحی است. شکل ۴-۴ سه ساختار متفاوت از پیاده‌سازی مجموع حاصلضرب‌ها روی FPGA را نشان می‌دهد.



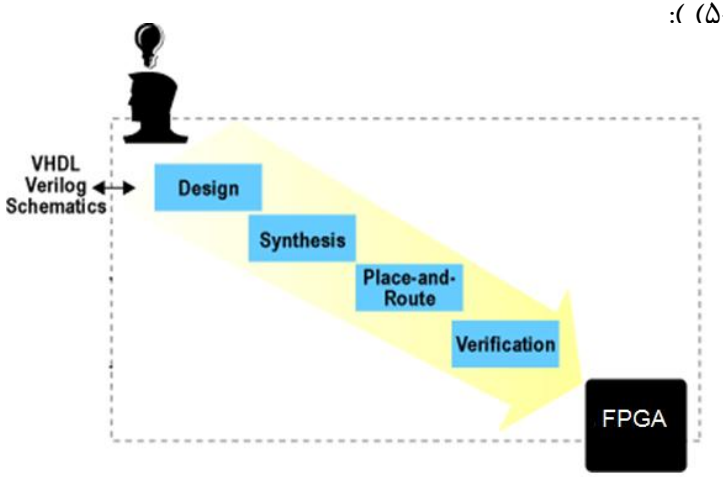
شکل ۴-۴ : انعطاف پذیریFPGA در انتخاب بین سرعت و هزینه.

همانگونه که در شکل ۴-۴ مشاهده می‌شود، با توجه به نوع سیستم مورد طراحی، نوع FPGA و میزان اهمیت بین سرعت و فضا در فرآیند طراحی، می‌توان از حالت سمت چپ، که یک طراحی کاملا موازی است و بیشترین سرعت را دارد، تا حالت سمت راست، که کمترین فضا را اشغال می‌کند، در حالیکه دارای محدودیت سرعت است، مناسب ترین حالت را انتخاب کرد[33].

۴-۵ روش طراحی سیستم‌های دیجیتال با FPGA

طراحی سیستم‌های دیجیتال با FPGA با ابزارهای برنامه‌ریزی FPGA به صورت اتوماتیک انجام می‌شود. این ابزارها توسط سازندگان FPGA و CPLD تهیه و در اختیار کاربران قرار می‌گیرد. به عنوان مثال برای FPGAهای شرکت Altera ابزار برنامه‌ریزی FPGA به نام MAX+PLUS II یا QUARTUS و برای FPGAهای شرکت Xilinx ابزار برنامه‌ریزی FPGA به نام Foundation و ISE و ... استفاده می‌شود.

برای طراحی سیستم‌های دیجیتال با ابزارهای مذکور و پیاده‌سازی آن روی FPGA، مراحل زیر باید طی شود (شکل (۴-۵)):



شکل ۴-۵ : مراحل پیاده‌سازی یک سیستم دیجیتال روی FPGA.

۴-۵-۱ وارد کردن طرح اولیه و کامپایل

طرح دیجیتال مورد نظر ابتدا ممکن است در محیط ابزار برنامه‌ریزی FPGA، به صورت شماتیک[1] یا ادیتوگرافیک[2]، بر روی مانیتور کامپیوتر کشیده شود یا با ویرایش‌گر متن HDL، طرح مذکور با زبان توصیف سخت‌افزار VHDL یا Verilog توصیف می‌گردد. سپس برنامه تفسیر[3] می‌گردد و آماده مرحله‌ی بعدی می‌شود.

۴-۵-۲ شبیه سازی

برای شبیه‌سازی و بررسی عملکرد طرح مذکور، سیگنال‌هایی به ورودی‌های طرح داده می‌شود و خروجی‌های آن بررسی می‌گردد.

۴-۵-۳ سنتز و آماده کردن طرح برای پیاده‌سازی

سنتز[4] یعنی تبدیل برنامه VHDL به معادل عناصر منطقی مانند جمع‌کننده، مولتی‌پلکسر، ثبات، دیکدر، فلیپ‌فلاپ، ... و اتصالات مربوطه که به آن مجموعه گره[5] گفته می‌شود. چون طرح اولیه‌ی منطقی معمولاً به طور بهینه نیست، لذا در ابزارهای برنامه‌ریزی FPGA، الگوریتم‌هایی وجود دارد که طرح اولیه را بهینه می‌کند. به عنوان مثال برای طراحی جمع‌کننده می‌توان جمع‌کننده کامل (FA)، نیم جمع‌کننده (HA)، و ... استفاده کرد. در این مرحله با توجه به مشخصات طرح از نظر سرعت و مقدار فضایی که در سیلیکون می‌گیرد، یکی از جمع‌کننده‌ها برای پیاده‌سازی در FPGA انتخاب می‌شود.

---
[1] Schematic
[2] Editographic
[3] Compile
[4] Synthesis
[5] Netlist

۴-۵-۴ تهیه فایل پیاده‌سازی برای FPGA

در این مرحله نوع FPGA را مشـخص می‌کنیم و با اسـتفاده از فایل سـنتز که شـامل بلوک‌های منطقی و ارتباطات آن می‌باشد، فایلی با یک سری بیت‌های باینری ۰ و ۱ تهیه می‌شود که با استفاده از آن تعدادی سـوئیچ‌ها و بلوک‌های منطقی FPGA می‌توانند برنامه‌ریزی شوند. علاوه بر این، در این مرحله محل بلوک‌های منطقی موردنیاز در FPGA مشخص و ارتباطات آن‌ها در سیلیکن نیز مشخص می‌شود این مرحله جای‌گذاری و سیم‌کشی[1] نیز نامیده می‌شود.

۴-۵-۵ شبیه سازی زمانی و بررسی

بعد از عملیات فوق و مشخص شدن گیت‌ها، بلوک‌های منطقی و مسیرها یا از طریق ارتباطات آنها در FPGA، مقدار تأخیر گیت‌های بلوک‌های منطقی ... مشـخص می‌شـود و در نتیجه زمان‌بندی و تأخیر واقعی مدار و نیز عملکرد نهایی مدار بررسی می‌گردد.

۴-۵-۶ برنامه ریزی FPGA

ابزار برنامه‌ریزی FPGA، با کابلی که از کامپیوتر به بورد FPGA متصـل می‌شود، فایل نهایی برای پیاده‌سـازی را در FPGA برنامه‌ریزی می‌کند، به عبارت دیگر طرح موردنظر را در FPGA پیاده‌سازی می‌کند.

[1] Place and Route

۶-۴ محاسبات ممیز ثابت و ممیز شناور

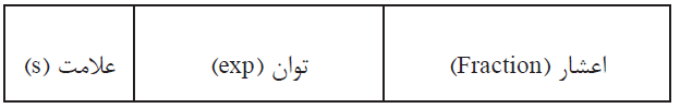
مقادیر محاسباتی در پردازش سیگنال به دو دسته تقسیم می‌شوند، ممیز ثابت[1] و ممیز شناور[2]. در طراحی FPGA، معمولاً از روش ممیز ثابت مکمل دو برای نمایش اعداد استفاده می‌شود. البته نمایش ممیز شناور نیز کاملاً قابل انجام است اما در پیاده‌سازی سخت‌افزاری ممیز شناور، در هر مرحله از عملیات جبری، تعداد بیت‌ها و مکان ممیز اعشاری تغییر خواهد کرد؛ بنابراین باعث افزایش فضای سیلیکونی، زمان تأخیر گیت‌ها و مصرف توان و کاهش سرعت می‌شود و در بخش پیاده‌سازی نرم‌افزاری نیز باعث کاهش سرعت پردازش خواهد شد. بنابراین در کاربردهای با کارآیی بالا، با بکارگیری تعداد بیت محدود و مناسب از نمایش ممیز ثابت استفاده می‌شود، که این نیز از دیگر ویژگی های FPGA محسوب می‌شود؛ یعنی امکان انتخاب مناسب طول کلمه در بخش های مختلف طراحی به منظور دست یافتن به دقت مورد نیاز.

پردازنـده‌های ممیز ثابت معمولاً هر عدد را با کمینه ۱۶ بیت ارائه می‌کنند، هرچند تعداد بیت‌های دیگر نیز می‌تواند اسـتفاده شود. برای عدد صحیح بدون علامت این ۱۶ بیت می‌تواند نماینده ۶۵۵۳۶ $=2^{16}$ عدد از ۰ تا ۶۵۵۳۵ باشد. به طور مشابه عدد صحیح علامت دار، مکمل دو را برای محدوده اعداد منفی از -۳۲۷۶۸ تا ۳۲۷۶۷ استفاده می‌کند. برای اعداد اعشاری بدون علامت، ۶۵۵۳۶ سطح به طور یکنواخت بین ۰ و ۱ پراکنده می‌شـود. و درنهایت صورت اعداد اعشاری علامت دار اعداد منفی، که به طور برابر بین -۱ و ۱ قرار گرفته‌اند را در بر می‌گیرد.

ممیز شـناور ابزار بسیار گسترده‌تری برای نمایش واقعی اعداد فراهم می‌کند و تمایل به استفاده از آن در کاربردهای محاسـبات عددی و خصـوصـاً در کاربردهای پردازش سیگنال دیجیتال بسیار بیشتر

[1] Fixed Point
[2] Floating Point

اســت. در ممیز شناور هدف این است که عدد واقعی را با استفاده از یک علامت، توان[1] و رقم اعشاری[2] مانند شکل ۴-۶ نشان دهیم. در این شکل، s نشان دهنده علامت، exp نمایانگر توان و Fraction بیان کننده رقم اعشار می باشد.



شکل۴-۶ : نمایش ممیز شناور

در مقایسه با پردازنده های ممیز ثابت، پردازنده‌های ممیز شناور کمینه ۳۲ بیت را برای ذخیره هر مقـدار اســـتفـاده می‌کننـد. این مقـدار الگوی بیتی بســـیـار بیشـــتری از ممیز ثـابـت، یعنی دقیقاً ۴،۲۹۴،۹۶۷،۲۹۶=2^{32}. یک ویژگی کلیدی ممیز شناور این است که اعداد ارائه شده به طور یکنواخت قرار نگرفته‌اند. در عمومی‌ترین شـکل (ANSI/IEEE Std. 754-1985)، بزرگترین و کوچکترین عدد به ترتیب $\pm 3.4\times10^{38}$ و $\pm 1.2\times10^{-38}$ است. مقادیر ارائه شده به طور یکنواخت بین این دو مقدار پراکنده شـده‌اند، که فاصـله بین هر دو مقدار حدود ده میلیون برابر کوچکتر از مقدار اعداد اسـت. اهمیت این موضوع در آن است که فاصله زیاد بین اعداد بزرگ و فاصله کم بین اعداد کوچک در نظر می‌گیرد.

تمـام پردازنـده هـای ممیز شـــنـاور می‌تواننـد بـا اعداد ممیز ثابت نیز کار کنند، که برای اجرای شـمارنده‌ها، حلقه‌ها، و سیگنال‌های خروجی از ADC و ورودی به DAC نیاز است. اما، این بدان معنا نیست که ریاضیات ممیز ثابت می‌تواند به‌اندازه ممیز شناور سریع باشد.

پر اسـتفاده ترین فرم ممیز شـناور، استاندارد IEEE754 برای محاسبات ممیز شناور باینری است، که به طور گسـترده ای در زبان های C وMATLAB اسـتفاده می‌شـود. طبق استاندارد IEEE-754،

[1] Exponent
[2] Mantissa or Fraction

عدد ممیز شـناور با اسـتفاده از یک بیت علامت S، X بیت صـحیح و Y بیت اعشار ارئه می‌شود. بیت علامت اغلب پرارزش‌ترین بیت (MSB) می‌باشد.

بر طبق این اسـتاندارد، یک مقدار ممیز شـناور به صورت نرمالیزه ارائه و ذخیره می‌شود. در شکل نرمالیزه مقدار صـحیح E یک مقدار بایاس شـده نرمالیزه است. این مقدار برابر مجموع مقدار صحیح و بایاس صـحیح است. در شکل نرمالیزه، Y-1 بیت برای ذخیره کردن اعشار استفاده شده است. بیت F_0 اعشار همیشه یک بیت مخفی[1] بوده و مقدار آن یک در نظر گرفته می‌شود.

s علامت عدد را نشـان می‌دهد. اگر S صفر باشد مقدار مثبت ممیز شناور و درغیر اینصورت مقدار منفی می‌شـود. X بیت بعدی برای ذخیره مقدار نرمالیزه صـحیح، و Y-1 بیت آخر برای مقدار اعشـار استفاده می‌شود.

برای عرض صحیح داده شده، بایاس مقدار صحیح از رابطه $bias = ۲^{(x-1)}-۱$ محاسبه می‌شود.

طبق اسـتاندارد IEEE-754 از ۳۲ بیت مقدار ممیز شـناور، ۸ بیت به مقدار صـحیح و ۲۴ بیت به مقدار اعشار نسبت داده می‌شود.

۴-۷ استفاده از FPGA به عنوان بستر پیاده‌سازی پردازش تصویر

با توجه به بحث‌های قبلی برای پیاده‌سازی شبیه‌سازی‌های انجام شده و بررسی نهایی طرح نیاز به یک بسـتر مناسـب برای پیاده‌سازی است. با توجه به کاهش قیمت و همچنین افزایش کارایی مدارات دیجیتال باعث افزایش اسـتفاده از آن‌ها در سـیسـتم‌های پردازشـی شـده است. FPGA در مقایسه با پردازنده‌های ترتیبی مانند میکروکنترلرها از مزیت‌های زمان واقعی بسـیار زیادی برخوردار اسـت و با موازی‌سازی و اجرای قسمت‌های مختلف برنامه به صورت همزمان از چیپ‌هایDSP عملکرد بهتری از

[1] Hidden Bit

خود نشـــان می‌دهد. قابلیت پیکربندی مجدد یکی دیگر از مهمترین قابلیت‌های FPGA اســـت. این قابلیت یک انعطاف‌پذیری سطح بالا فراهم می‌کند که این اجزا را به سیستم بخش‌بندی تصویر می‌دهد که به صورت یک سیستم پردازشی چند کاناله عمل کند و یا حتی چندین سیستم بخش‌بندی مختلف را بر روی یک چیپ پیاده‌سـازی کند. بعلاوه با تعریف قسـمت‌های مختلف یک سـیسـتم پردازشی در سـطح ماژول و تقسـیم‌بندی آن به قسـمت های مختلف در روی یک چیپ FPGA، قابلیت تسـت و آزمایش قسـمت‌های مختلف سـیسـتم پردازش را به صورت مجزا فراهم می‌کند که چیپ‌های DSP با این قابلیت همخوانی ندارند. با داشتن این ویژگی‌ها، به نظر می‌رسد که FPGA پتانسیل بلقوه‌ای برای اســـتفاده در زمینه پردازش تصـــویر و بینایی ماشـــین را دارا می‌باشـــد. به همین دلایل FPGA برای پیاده‌سازی الگوریتم پردازشی پیشنهادی انتخاب شده است.

۸-۴ پیاده‌سازی الگوریتم روی FPGA با نرم افزار System Generator

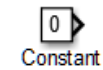
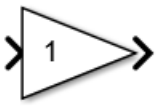
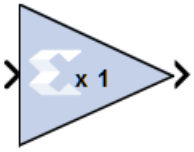
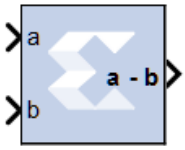
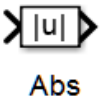
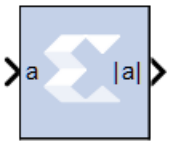
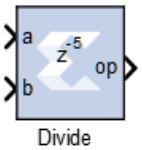
یک ابزار طراحیDSP، نرم افزار System Generator است. این نرم‌افزار توسط شرکت Xilinx معرفی شده و به همراه نرم‌افزار ISE ارائه می‌گردد؛ و مهمترین قابلیت آن ایجاد یک محیط شبیه‌سازی همانند سیمولینک MATLAB برای طراحی FPGA است. برای استفاده از System Generator هیچ نیازی به اطلاعات قبلی در مورد انواع FPGAهای شرکت Xilinx یا متدهای طراحی RTL نیست.

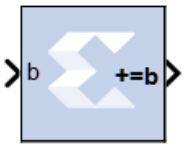
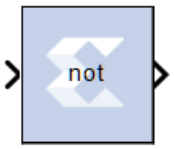
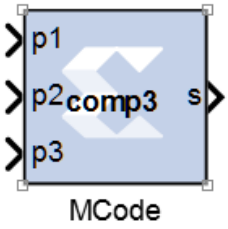
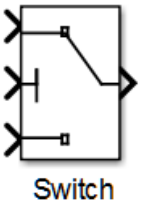
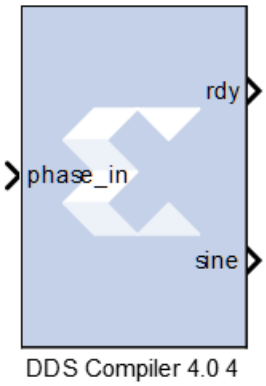
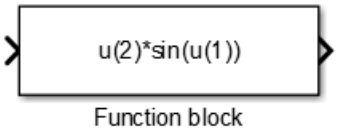
در این نرم افزار طراحی‌های DSP به صـورت شـماتیک، با کمک مجموعه بلوک‌هایی که نرم‌افزار دراختیار کاربر قرار می‌دهد، انجام می‌شـود. محیط این نرم‌افزار کاربر پسـند[1] و بسـیار شـبیه به Simulink در متلب اسـت. از ویژگی‌های منحصـر به فرد این نرم‌افزار این اسـت که تمامی مراحل پیاده‌سازی از قبیل سنتز و آماده‌سازی فایل شبیه‌سازی تا تولید یک فایل برنامه‌ریزی برای FPGA به صورت خودکار انجام می‌شود.

[1] User Friendly

بیش از ۹۰ بلوک DSP در مجموعه بلوک‌های Xilinx DSP برای Simulink وجود دارد که با نصب نرم‌افزار System Generator اضافه می‌شود. این بلوک‌ها شامل بلوک‌های عملیات معمولی مانند جمع و تفریق کننده، ضرب کننده و رجیسترها و همچنین شامل بلوک‌های پیچیده‌تری همچون بلوک‌های تصحیح خطا، تبدیل فوریه، انواع فیلترها و حافظه‌ها می‌باشد. در جدول ۴-۱ بعضی از این بلوک‌ها معرفی و بلوک مشابه آن در متلب نیز نشان داده شده است.

جدول ۴-۱ بلوک‌های معادل متلب در پیاده‌سازی دیجیتالی کنترل کننده

بلوک متلب	وظیفه	بلوک معادل آن در SysGen	وظیفه
	عدد ثابت به مدار وارد می‌کند.		عدد ثابت باینری در حوزه دیجیتال به مدار وارد می‌کند.
	بلوک ضرب کننده است.		عمل ضرب را در FPGA انجام می‌دهد.
	عمل جمع یا تفریق انجام می‌دهد.		جمع یا تفریق باینری انجام می‌دهد.
	قدر مطلق ورودی اش را حساب می‌کند.		مثبت عدد باینری را محاسبه می‌کند.
	عمل تقسیم (همچنین ضرب) را انجام می‌دهد.		عملیات تقسیم در FPGA را انجام می‌دهد.

انباشتگر ورودی‌ها را مرتباً باهم جمع می‌کند.		بلوک انتگرال گیر	
عمل منطقی معکوس کردن علامت را روی عدد باینری انجام می‌دهد.		بلوک منطقی که مقدار ورودی را از نظر علامت معکوس می‌کند.	
در خصوص بلوک سوئیچ و سایر بلوک‌های شرطی می‌توان با استفاده از بلوک MCode یک کد ساده در mfile نوشت و به آدرس کد در مشخصات بلوک ارجاع داد.		یک سوئیچ شرطی است که ورودی اول و آخر را با وسط مقایسه می‌کند و هرکدام بیشتر باشد عبور می دهد.	
دامنه و فاز سینوس ورودی را به صورت لحظه ای حساب می‌کند.		تابع خاص در متلب را می‌توان از طریق این بلوک به مدار داد. مثل توابع نمایی،سینوسی و ...	

System Generator یک بلوک کامپایلر FIR دارد که با استفاده از آن می‌توان سخت‌افزارهای DSP48 در deviceهای virtex4 وvirtex5 که در فرکانس‌های بالای ۵۰۰ مگا هرتز اجرا می‌شوند را مورد هدف قرار داد. با استفاده از توابع استاندارد MATLAB از قبیل fir2 یا fdatool می‌توان ضرایب لازم برای کامپایلر FIR در Xilinx را تولید کرد.

با اســتفاده از AccelDSP در System Generator می‌توان مدل‌های الگوریتمی MATLAB را به کار برد. AccelDSP شـامل سـنتز الگوریتمی قدرتمندی اسـت که ورودی ممیز شناور MATLAB را گرفته و یک مدل ممیز ثابت کامل برای اســتفاده در System Generator ارائه می‌دهد. امکانات AccelDSP عبـارتنـد از تبـدیـل ممیز شـــنـاور به ممیز ثابت، وارد کردن IP خودکار، بیان طرح و برنامه‌ریزی الگوریتمی.

همچنین در System Generator بلوکی به نام Mcode قرار دارد که به کاربر اجازه می‌دهد از MATLAB غیرالگوریتمی برای مدلسازی و پیاده‌سازی عملیات کنترلی ساده استفاده کند.

System Generator یک بلوک به نام Resourse Estimator در اختیار کاربر قرار می‌دهد که این بلوک می‌تواند با سرعت زیادی ناحیه‌ی طرح اولیه تا مکان‌یابی و سیم‌کشی را تخمین بزند.

با اسـتفاده از Co-Simulation سخت‌افزاری در System Generator می‌توان به شبیه‌سازی سریع دسـت یافت. این نرم‌افزار به طور اتوماتیک یک شبیه‌سازی سخت‌افزاری برای یک طرح ساخته شده با اسـتفاده از بلوک‌های DSP تولید می‌کند که یکی از ۲۰ طرح سـخت‌افزاری پشـتیبانی شـده را اجرا می‌کند.

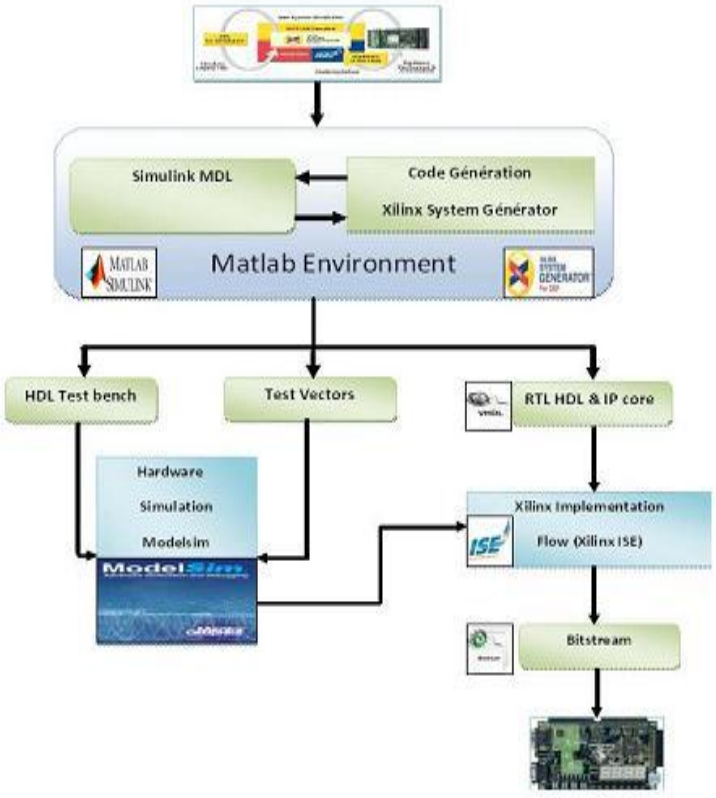
همچنین System Generator دارای یک جعبه بلوک اسـت که اجازه می‌دهد RTL به Simulink تبدیل شده و به طور همزمان در شبیه‌سازهای Modelsim و Xilinx ISE استفاده شود.

یک ویژگی جالب دیگر این نرم‌افزار این اســـت که می‌توان از پردازنده های Microblaze که برنامه‌های C یا C++ بر روی آنها اجرا می‌شود استفاده کرد.

۴-۸-۱ نرم افزار ISE

نرم افزار ISE یک محیط طراحی بسیار قدرتمند است. در زمانی که System Generator در حال اجرای بلوک‌ها در محیط شبیه‌ساز است در پشت صحنه وظیفه پیاده‌سازی این بلوک‌ها را برعهده دارد. محیط ISE متشکل از مجموعه‌ای از بخش‌هایی است که به صورت HDL برنامه‌نویسی شده‌اند که برای تولید، تبدیل، شبیه‌سازی و در نهایت پیاده‌سازی طراحی‌های دیجیتال در FPGAها به کار می‌روند. بعد از سنتز این ماژول‌ها فایل netlist تولید می‌شود، تا به عنوان ورودی مرحله پیاده‌سازی به کار گرفته شود.

بعد از این فایل‌ها، طراحی منطقی انجام شده تبدیل به یک فایل فیزیکی می‌شود که قابل بارگذاری بر روی قطعه مورد نظر خواهد بود. [34] شکل (۴-۷) یک بلوک دیاگرام کامل از شبیه‌سازی تا پیاده‌سازی بر روی سخت‌افزار را نمایش می‌دهد.



شکل۴-۷ : گراف طراحی با System Generator .

۹-۴ پیاده‌سازی الگوریتم روی FPGA با HDL Coder

HDL[1] یک زبان تخصصی توصیف سخت‌افزاری کامپیوتر است که برای توصیف ساختار و رفتار مدارهای الکترونیکی و رایج‌تر از همه مدارهای منطقی دیجیتال کاربرد دارد.

HDL coder یک جعبه‌ابزار متلب است که برای تولید کدهای ترکیبی VHDL و Verilog در انواع مختلف FPGA و تکنولوژی‌های ASIC مورد استفاده قرار می‌گیرد. همچنین می‌تواند یک بلوک Black Box موجود در System Generator را تولید کند. بعد از طراحی کردن یک الگوریتم در متلب برای تولید کد HDL، می‌توان آن را به عنوان یک بلوک Black Box سیستم ژنراتور در سیستم بزرگتر به کار برد.

بلوک Black Box سیستم ژنراتور تولید شده توسط HDL Coder به صورت یک Subsystem است که با بلوک‌هایی از هر دو میحط سیمولینک و Xilinx System Generator کار می‌کند. بنابراین می‌توان با استفاده از بلوک Black Box تولید شده سیستم بزرگتری برای سیمولینک و تولید کد ساخت.

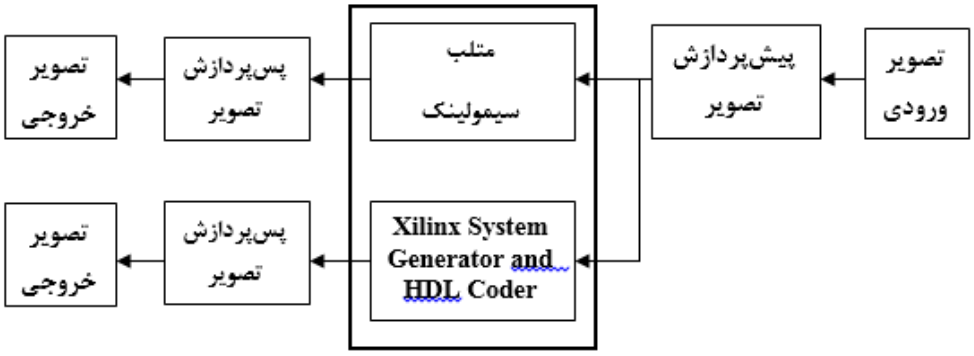
برای تولید کد HDL از متلب، نیاز است کد حاصل به دو فایل تقسیم شود :۱. فایل Test bench، ۲. فایل طراحی. فایل طراحی برای پیاده‌سازی الگوریتم در FPGA یا ASIC مورد استفاده قرار می‌گیرد. فایل Test bench داده ورودی را برای فایل طراحی فراهم و همچنین خروجی فایل طراحی را دریافت می‌کند. جهت آشنایی بیشتر و ایجاد یک پروژه با HDL Coder به پیوست مراجعه شود.

[1] Hardware Description Language

فصل پنجم :الگوریتم پیشنهادی

۵-۱ مقدمه

همانطور که در فصل دوم اشاره شد سیستم تشخیص خطوط جاده از بخش‌های مختلفی تشکیل شده است. شکل(۵-۱) این بخش‌ها و روند آن را نشان می‌دهد [35]. در این فصل نحوه‌ی شبیه‌سازی هر یک از این قسمت‌ها بیان خواهد شد.



شکل ۵-۱ : روند کلی کار.

ابتدا جهت طراحی و سنجش عملکرد طرح از نرم‌افزار متلب استفاده می‌شود. سپس بعد از انجام مراحل فوق، الگوریتم پیشنهادی توسط شبیه‌ساز گرافیکی System Generator و HDL Coder که برای شبیه‌سازی طرح سخت‌افزاری موردنظر مورد استفاده قرار گرفته است، در FPGA پیاده‌سازی می‌گردد.

نرم‌افزارُ System Generator محیطی را فراهم می‌کند که با ایجاد بلوک‌های منطقی، دقیقاً مدل واقعی سیستم قابل پیاده‌سازی را اجرا کرد و مستقیماً آن‌ها را تبدیل به RTL برای پیاده‌سازی در FPGA می‌کند. از این رو ابزار مناسبی برای نزدیک کردن نتایج شبیه سازی به واقعیت است.

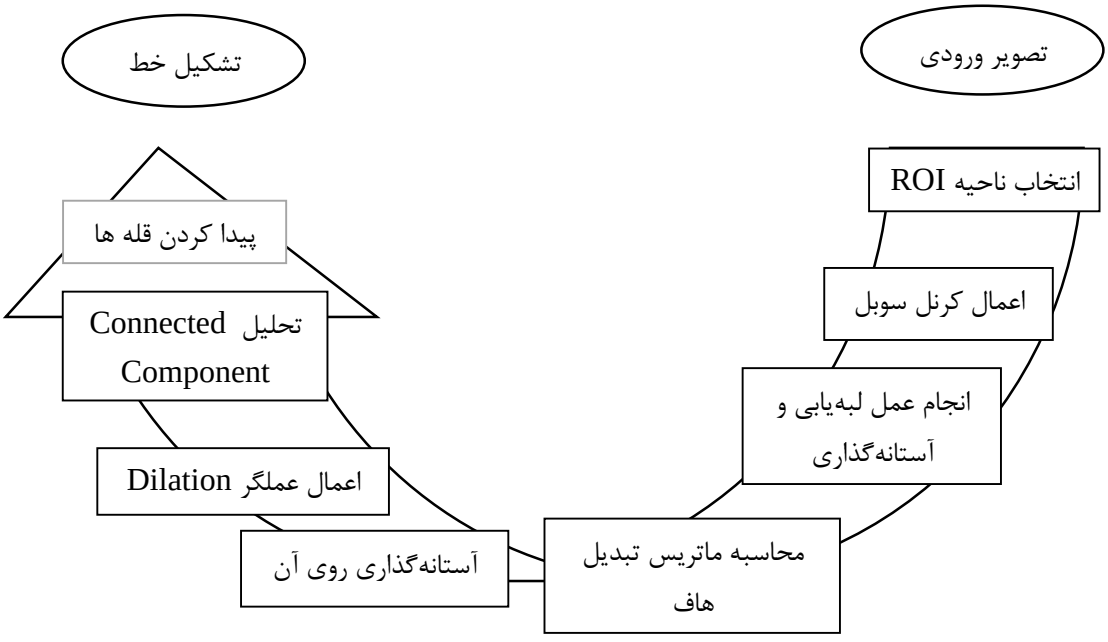
[36] این نرم‌افزار توسط شرکت Xilinx معرفی و به Simulink/Matlab اضافه شد که این اجازه را به طراحان می‌دهد تا یک طراحی گرافیکی سطح بالای قابل پیاده‌سازی سخت‌افزاری در شرایط واقعی را فراهم کنند. برخلاف طراحی‌های انجام شده در محیط شبیه‌ساز که قابلیت پیاده‌سازی مستقیم را

ندارند، طراحی با System Generator شکاف بین طراحی سطح بالا و پیاده‌سازی در شرایط واقعی را پر کرده است و نتایج قابل اعتمادی را به ما می‌دهد.

۵-۲ الگوریتم پیشنهادی

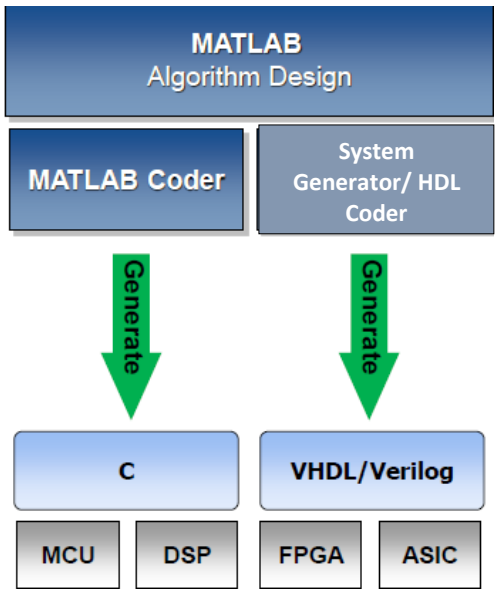
با توجه به فلوچارت رسم شده در شکل (۵-۱) الگوریتم پیشنهادی به چهار صورت انجام شده است:

- در مرحله اول جهت طراحی و سنجش عملکرد طرح از نرم‌افزار متلب استفاده می‌شود. براساس نمودار شکل (۵-۲) الگوریتم پیشنهادی برای تشخیص خطوط جاده به صورت دو بعدی و با استفاده از توابع آماده متلب از جمله :Conv، Find، Hough، Houghpeaks و .... پیاده‌سازی و اجرا شده است. نمودار شکل (۵-۲) الگوریتم پیشنهادی برای تشخیص خطوط جاده را نشان می‌دهد.



شکل ۵-۲ روند الگوریتم پیشنهادی.

- در مرحله دوم به دلیل پیاده‌سازی بر روی FPGA مجدداً برنامه در محیط نرم‌افزاری متلب به صورت دو بعدی و بدون استفاده از توابع آماده متلب پیاده‌سازی و اجرا شد. مثلا برای کانولوشن دو بعدی در قسمت لبه‌یابی سوبل باید یک حلقه For تودرتو تعریف کرده و تک تک المان های تصویر را در کرنل سوبل ضرب و مجموع را محاسبه کنیم.
- در مرحله سوم از آنجایی که باید ورودی نرم‌افزار Xilinx System Generator به صورت سریال باشد. بار دیگر برنامه دو بعدی نوشته شده در محیط متلب که بدون استفاده از توابع آماده بود. به یک برنامه یک بعدی تبدیل و پیاده‌سازی شد.
- در مرحله چهارم برای پیاده‌سازی الگوریتم پیشنهادی بر روی FPGA از شبیه‌ساز System Generator و شبیه‌ساز HDL Coder که در قسمت های بعدی به طور مفصل شرح داده می‌شود، استفاده شده است. در این مرحله با توجه به آنچه که در فصل‌های گذشته بیان شد، برای تولید اتوماتیک کد HDL با توجه به فلوچارت شکل ۵-۳ می‌توان علاوه بر استفاده از نرم‌افزار System Generator از HDL Coder هم استفاده کرد. که با استفاده از آن علاوه بر تولید کد HDL همزمان یک Black Box نرم‌افزار Xilinx System Generator هم تولید می‌شود.



شکل۵-۳: طراحی یک الگوریتم در متلب با دو روش.

۵-۳ شبیه سازی در محیط متلب

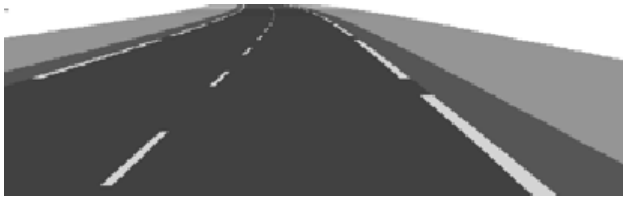
هدف از این الگوریتم صرفاً پیدا کردن ویژگی‌های متعلق به خطوط خط‌کشی است. این الگوریتم ناحیه کوچکتر مورد علاقه در جلوی خودرو در حال حرکت را انتخاب کرده و به عنوان تصویر ورودی مرحله پیش‌پردازش استفاده می‌کند.

الگوریتم تشخیص خطوط شامل سه مرحله : پیش‌پردازش، پس‌پردازش و مدل‌کردن خطوط خط‌کشی جاده است. جزئیات این گام‌ها و محاسبات مرتبط در این بخش توصیف شده است.

۵-۳-۱ مرحله پیش‌پردازش

گام اول در سیستم‌های پردازش تصویر سطح پایین برخورد با تصاویر سنسور دید و تولید اطلاعات مفید برای قسمت‌های تشخیص است. در این بخش، تصویر جاده برای قسمت محاسبات کپی می‌شود. این تصویر به ناحیه کوچکتر مورد علاقه برای صرف جویی در زمان محاسبات همان طوری که در شکل ۵-۴ نشان داده شده است، کاهش می یابد.

الف)



ب)

شکل ۵-۴: در مرحله پیش‌پردازش، شکل الف) تصویر اصلی، ب) ناحیه کوچکتر مورد علاقه.

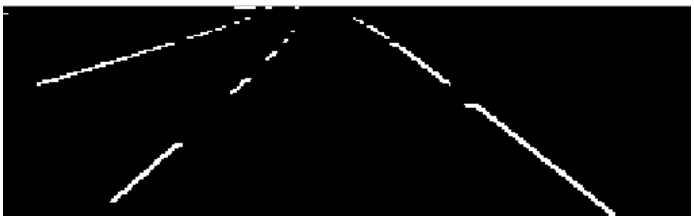
در اینجا برای انتخاب ناحیه ROI فرض کرده‌ایم ناحیه جاده پایین خط افقی است که ناحیه آسمان و انتهای جاده را در بر نمی‌گیرد. می‌توان گفت با انجام آزمون سعی و خطا ۰/۵۹ از کل سطرهای تصویر را به عنوان منطقه ROI برگزیدیم. همچنین همان‌طوری که لبه‌یاب سوبل با تصویر تک‌رنگ کار می‌کند، تصویر به مقیاس خاکستری هم تبدیل شده است

۵-۳-۲ مرحله پس‌پردازش

لبه‌یابی سوبل در مرحله پس‌پردازش پیاده‌سازی شده است. همان‌طوری که در فصل ۳ اشاره شد، لبه‌یاب سوبل از دو کرنل عمودی و افقی استفاده می‌کند. که در الگوریتم پیشنهادی برای انجام عمل لبه‌یابی فقط از یک کرنل سوبل یعنی کرنل افقی استفاده شده است. سپس برای پیدا کردن لبه‌های قوی با انجام آزمون سعی و خطا یک آستانه مناسب انتخاب کرده و به هر پیکسلی که شدت روشنایی آن بیشتر از آستانه باشد، مقدار ۱ اختصاص داده می‌شود. به عنوان مثال تصاویر حاصل از سه آستانه مختلف در شکل ۵-۵ نشان داده شده است. خط‌ها و لبه‌ها در تصویر در مرحله تشخیص ویژگی تشخیص داده شده‌اند.

الف)

ب)

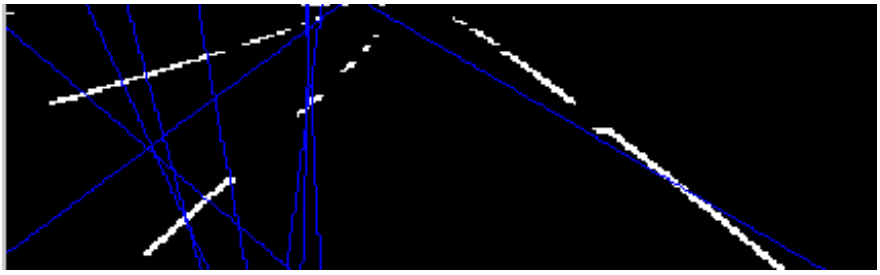


ج)

شکل ۵-۵ : لبه‌یاب سوبل برای استخراج ویژگی های خطوط خط‌کشی، الف) آستانه مساوی ۲۰، ب) آستانه مساوی ۵۰، ج) آستانه مساوی ۱۵۰.

با توجه به شکل‌های ۵-۵ مقدار آستانه ۱۵۰ را انتخاب کردیم. تا به نتیجه مطلوب دست یافتیم.

سپس تبدیل هاف برای اتصال خطوط ناپیوسته و متمایز در بازه ۰ تا ۱۸۰ درجه همان‌گونه که در شکل ۵-۶ دیده می‌شود، پیاده‌سازی شده است. به طور خلاصه، پس‌پردازش یکی از مهم‌ترین گام‌هایی است که بین مرحله استخراج ویژگی و مرحله مدل‌سازی خطوط جاده ارتباط برقرار می‌کند.

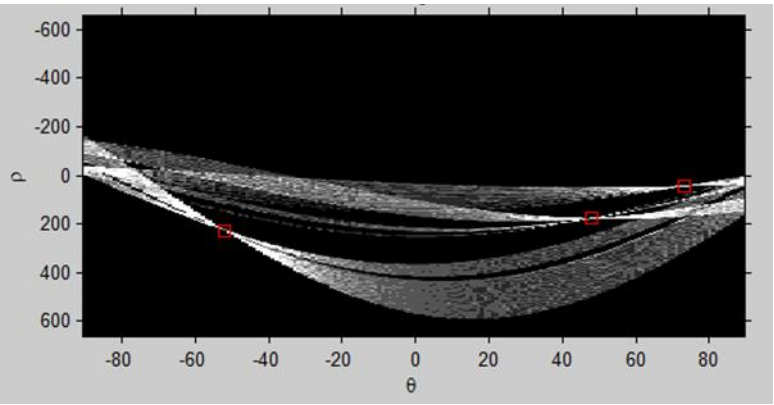


شکل ۵-۶: تبدیل هاف خطوط وابسته که نماینده خطوط خط‌کشی چپ و راست هستند را تشخیص می‌دهد.

۵-۳-۳ مرحله مدل کردن خط جاده[1]

در این مرحله محاسباتی برای تشخیص خطوط خط‌کشی چپ و راست جاده انجام شده است. بعد از انجام تبدیل هاف، نتیجه نهایی که شامل چندین خط که واجد شرایط تبدیل شدن به مرزهای چپ و راست هستند را نشان می‌دهد. با گروه‌بندی کردن همه مرزهای سمت چپ به یک گروه و همه مرزهای سمت راست به گروه دیگر، می توان یک متوسطی را بدست آورد که فقط یک خط در سمت چپ و یک خط در سمت راست تشخیص داده شود.

لازم بذکر است که در الگوریتم پیشنهادی گروه بندی خطوط، ابتدا با آستانه‌گذاری بر روی ماتریس تبدیل هاف و سپس با تحلیل اجزای متصل میسر می‌شود. با این کار قله‌ها در فضای هاف پیدا شده (در شکل ۵-۷ قله ها بخوبی دیده می شوند.) و خط‌هایی که بیشترین رأی را به خود اختصاص داده‌اند، نمایان می‌شوند.

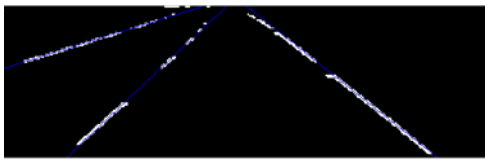
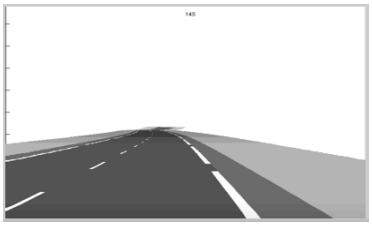
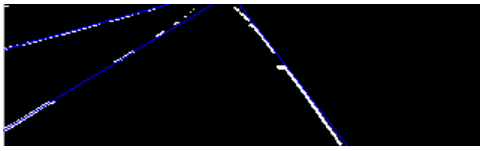
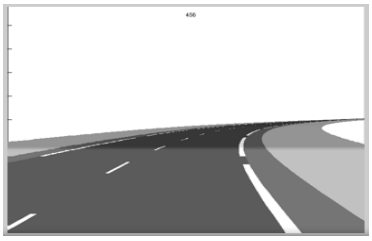
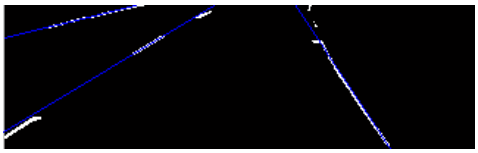
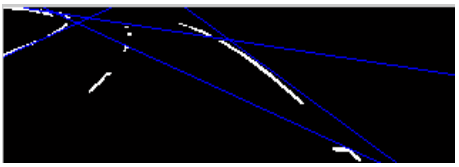
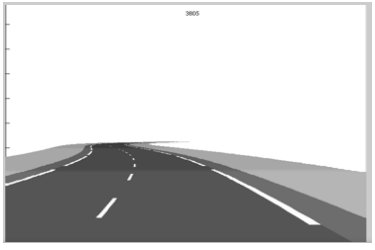
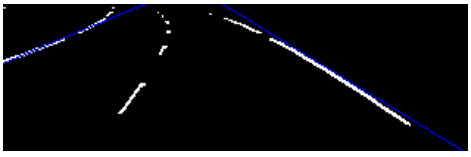


شکل ۵-۷ : فضای تبدیل هاف و نمایش قله‌ها بر روی آن.

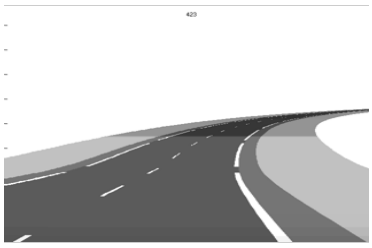
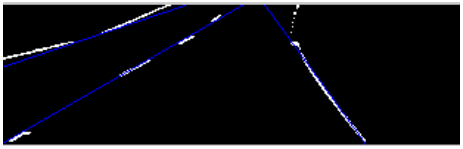
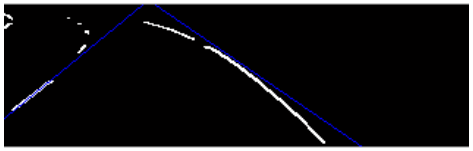
در نهایت نتیجه شبیه‌سازی بر روی چند عکس از پایگاه داده در جدول ۵-۱ نمایش داده شده است.

[1] Road lane modeling

جدول ۵-۱ : نتایج بر روی صحنه‌های متفاوت (موارد۱-۵).

	تصویر اصلی	پس از پردازش
۱		
۲		
۳		
۴		
۵		

جدول ۵-۱ : نتایج بر روی صحنه‌های متفاوت (موارد۶-۷).

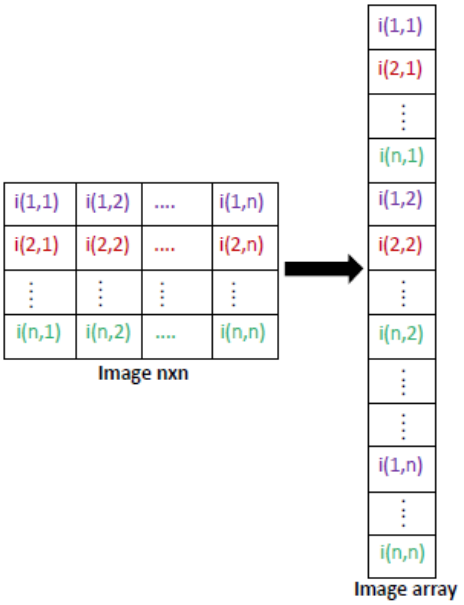
	تصویر اصلی	پس از پردازش
۶		
۷		

در مثال اول که جاده مستقیم و هوا آفتابی است، هر سه خط جاده به درستی تشخیص داده می‌شوند. برای مثال ۲و ۳ هم نتیجه درست است. اما برای مثال ۴ که جاده انحنا دارد، خطوط سمت چپ و سمت راست پیدا شده‌اند ولی خطوط که به اشتباه تشخیص داده شده‌اند هم دیده می‌شود. به عنوان نمونه عکس هایی هم از پایگاه داده انتخاب شده که الگوریتم پیشنهادی می‌تواند دو خط اصلی جاده را بنابر دلایلی از جمله پیچ در جاده، سایه و.... تشخیص دهد. البته در الگوریتم پیشنهادی فرض شده است که از جاده مستقیم و در هوای آفتابی عکس‌برداری می‌شود.

۵-۴ شبیه‌سازی در حوزه متلب با عملیات سریال

همان‌طور که پیشتر اشاره شد در این بخش کل برنامه دو بعدی متلب به یک برنامه یک بعدی تبدیل و پیاده‌سازی شده است. در ابتدا برخلاف پردازش در سطح نرم‌افزاری که تصویر یک آرایه دو بعدی n*m

است و به همین‌صورت پردازش می‌شود، در حالت سخت‌افزاری این ماتریس باید یک آرایه یک بعدی به نام بردار باشد. یعنی تصویر به صورت سریال در اختیار ورودی قرار بگیرد. سپس اطلاعات تصویر در حافظه ROM ذخیره شود. مختصات (i,j) به دنبال تبدیل (j-1)*m+i به مختصات جدیدی تبدیل می‌شود. ما تصویر را به صورت ستونی، از اولین ستون تا آخرین ستون پشت سرهم به صورتی که در شکل ۵-۸ نشان داده شده است چیده ایم. البته می‌توان تصویر را به صورت سطر به سطر هم پشت سرهم چید.



شکل۵-۸ : ذخیره کردن مقادیر تصویر در یک حافظه ROM .

بعد از ذخیره تصویر به صورت سریال، ادامه مراحل الگوریتم مطابق با فلوچارت رسم شده در شکل ۵-۲ هم به صورت یک بعدی پیاده سازی شد. به عنوان مثال عملیات گسترش[1] که در الگوریتم پیشنهادی مورد استفاده قرار گرفته است، در جدول ۵-۲ نحوه تبدیل به برنامه یک بعدی آن نشان داده شده است.

[1] Dilation

جدول ۵-۲: برنامه یک بعدی و دو بعدی عملیات گسترش.

برنامه یک بعدی عملیات گسترش	برنامه دو بعدی عملیات گسترش بدون استفاده از توابع آماده متلب

se=[1 1 1;1 1 1;1 1 1];
L=3;
nS=300; nT=200;
kx=1;
for j2=1:nS-(L-1)
for i2=1:nT-(L-1)
if (TH(kx)==1)
for i =0:L-1
for j=0:L-1
if(newimg((j2+j-1)*nT+i2+i)~=1)
newimg((j2+j-1)*nT+i2+i)=se(i+1,j+1);
end
end
end
end
kx=kx+1;
end
kx=kx+L-1;
end
	[p,q]=size(TH);
	newimg=zeros(p,q);
	structure=ones(3,3);
	for i=2:p-1
	for j=2:q-1
	window=TH(i-1:i+1,j-1:j+1);
	product=structure.*window;
	s=sum(sum(product));
	if s ~=0
	newimg(i,j)=1;
	end
	end
	end
	
	TH ماتریس ورودی که عملیات گسترش روی آن انجام می‌شود.

۵-۵ شبیه‌سازی در حوزه دیجیتال

در این مرحله بعد از کامل‌کردن برنامه‌نویسی در حوزه متلب و رسیدن به نتایج مطلوب، حال نوبت به پیاده‌سازی الگوریتم پیشنهادی بر روی FPGA رسیده است. در ابتدا قبل از هر توضیحی لازم می‌باشد به نکات مهم و ساختاری که در برنامه‌ریزی FPGA به کار برده شده است و نقش مهمی در صحت عملکرد برنامه ایفا می‌کنند اشاره شود. سعی شده است تا جایی که ممکن است از ساختارهای شرطی تودرتو اجتناب کرده و بیش از دو لایه عبارت شرطی در هر بخش نباشد. همچنین تا حد امکان از شرط‌های ساده استفاده شده است.

برای سنتز قسمت لبه‌یابی مدار روی FPGA از نرم‌افزار System Generator استفاده شده است. این نرم‌افزار تنها به محیط شبیه‌ساز متلب چند بلوک دیجیتالی اضافه می‌کند.

جهت استفاده از بلوک‌های این نرم‌افزار لازم است طراح ابتدا با بلوک‌های اصلی مورد استفاده که در طراحی حوزه دیجیتال کمک می‌کند آشنا شود. در فصل ۴ بعضی از این بلوک‌ها معرفی و بلوک مشابه آن در متلب نیز نشان داده شده است.

لازم به ذکر است که جهت ارتباط و اتصال سایر بخش‌ها به این بلوک‌ها در ورودی و خروجی به ترتیب درگاه ورودی[۱] و درگاه خروجی[۲] استفاده شده است. از این بلوک‌ها جهت ورود و خروج داده استفاده می‌شود. بلوکهایی که بین این دو درگاه قرار می‌گیرند الزاماً باید از بلوک‌های محیط System Generator باشند. بلوک درگاه خروجی تنظیمات خاصی نداشته و تعداد بیت‌های خروجی نیز با توجه به سیگنالی که به آن وصل شده به صورت خودکار تعیین می‌گردد. بلوک درگاه ورودی قابلیت تعیین ورودی به شکل‌های بولین[۳] و ممیز ثابت و ممیز شناور می‌باشد. نکته مهم در رابطه با این بلوک این است که زمان نمونه‌برداری از خروجی بلوک را می‌توان تعیین کرد با توجه به این مطلب که این زمان نمی‌تواند از مقدار دوره تناوب شبیه‌ساز سیستم[۴] کمتر باشد.

۵-۵-۱ پیاده‌سازی مرحله لبه‌یابی سوبل در حوزه دیجیتال

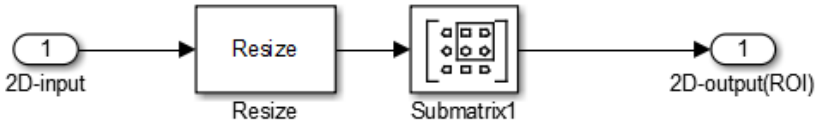
در این قسمت لبه‌یابی سوبل توسط نرم‌افزار System Generator بروی FPGA پیاده‌سازی می‌شود.

---

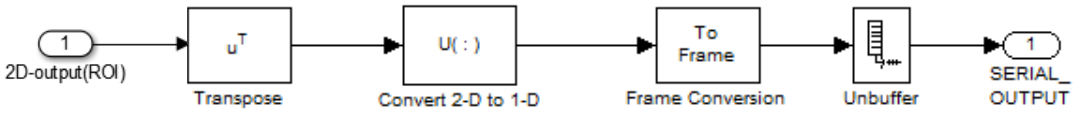
[2]Gateway In
[3] Gateway Out
[1] Boolean
[2] Simulink system Period

۵-۵-۱-۱ بلوک دیاگرام تبدیل تصویر به سریال

در ابتدا لازم است در همین مرحله انتخاب ناحیه مورد علاقه (ROI) توسط بلوک‌های شکل۵-۹ هم اجرا شود سپس همان طور که در بالا توضیح داده شد، از جمله مراحل اولیه با اهمیت پیش‌پردازش تصویر در حالت پیاده‌سازی سخت‌افزاری تبدیل آن به بردار توسط بلوک دیاگرام شکل۵-۱۰ است. تا ورودی بلوک Gateway In فراهم شود.



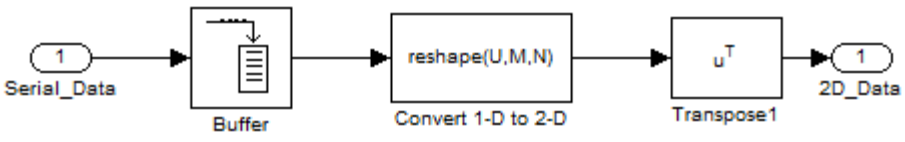
شکل۵-۹: انتخاب ناحیهROI .



شکل۵-۱۰ : بلوک دیاگرام پیش پردازش تصویر.

۵-۵-۱-۲ بلوک دیاگرام تبدیل تصویر از سریال به حالت دو بعدی

از جمله مراحل با اهمیت پس پردازش تصویر ساختن داده قابل استفاده همراه با نرخ داده مناسب برای نمایش در محیط متلب است. بلوک دیاگرام شکل۵-۱۱ بخوبی مراحل پس‌پردازش یا تبدیل تصویر سریال به حالت دو بعدی را نمایش می‌دهد.

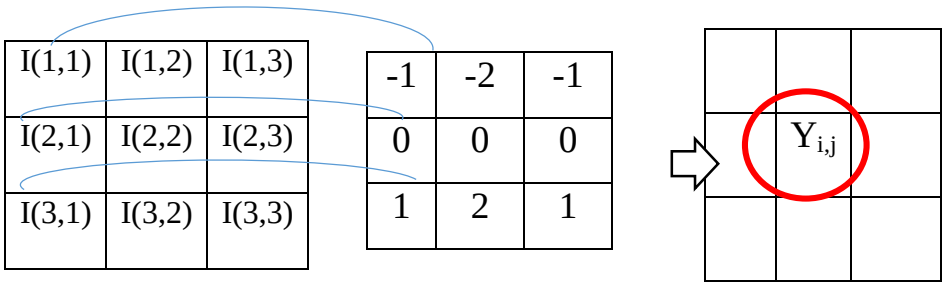


شکل۵-۱۱ : بلوک دیاگرام پس پردازش تصویر.

۵-۵-۱-۳ گرادیان عمودی عملگر سوبل :

همانطور که در فصل۳ اشاره شد عملگر سوبل دارای دو کرنل افقی و عمودی است. که ما در اینجا تنها از کرنل عمودی استفاده کردیم. گرادیان عمودی هم از حرکت دادن کرنل عمودی برای تشخیص لبه سوبل حاصل می‌شود.

کرنل سوبل یک ماتریس ۳*۳ می‌باشد که بر روی ماتریس تصویر حرکت داده می‌شود. حال برای انجام عملیات کانولوشن باید، مجموع حاصلضرب‌های دوبدوی عناصر متناظر از تصویر و کرنل را محاسبه کرده و به عنوان خروجی ثبت کنیم. در شکل ۵-۱۲ عملیات کانولوشن نشان داده شده است.



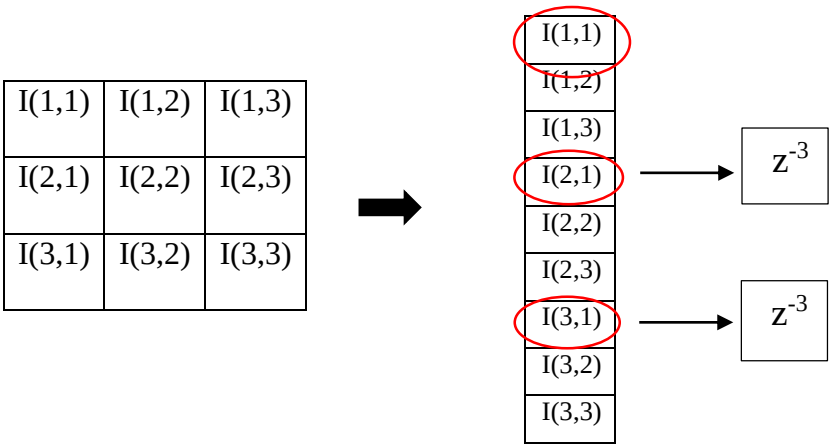
ج)خروجی ب) کرنل عمودی سوبل الف) انتخاب یک ماتریس ۳*۳ از تصویر ورودی

شکل ۵-۱۲ : عملیات کانولوشن.

با توجه به شکل ۵-۱۲ می‌توان عملیات کانولوشن دو بعدی را توسط رابطه ۵-۱ نشان داد.

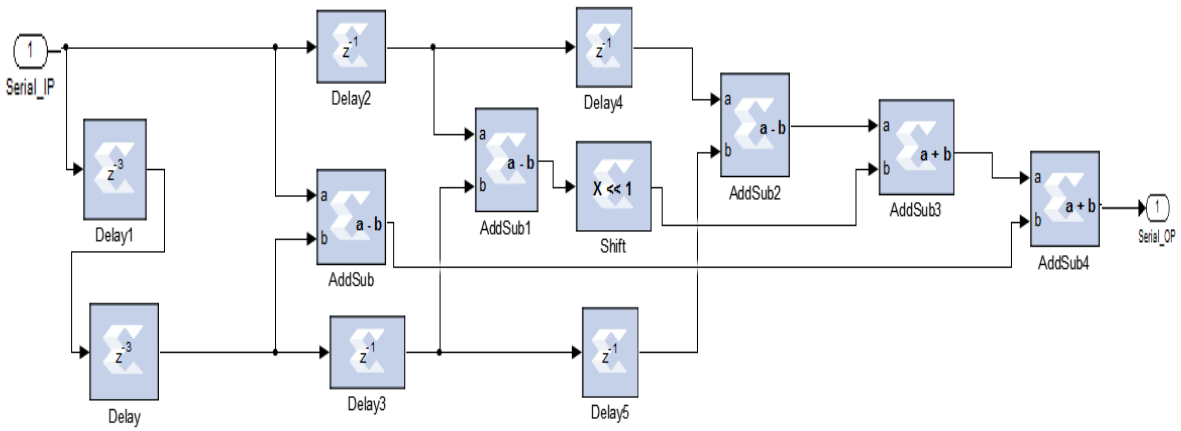
$$Y(i,j)=I(1,1)*(-1) + I(1,2)*(0) + I(1,3)*(1) + I(2,1)*(-2) +\ldots+ I(3,3)*(1) \qquad ۵\text{-}۱$$

همان‌طور که در بخش ۵-۴ بیان شده است، برای تبدیل تصویر به سریال، باید تصویر را سطر به سطر درون یک بردار ذخیره کنیم. سپس جهت دستیابی به مکان هر پیکسل تصویر همان‌طوری که در شکل ۵-۱۳ نشان داده شده است، در نرم‌افزار System Generator از بلوک تأخیر استفاده می‌شود.



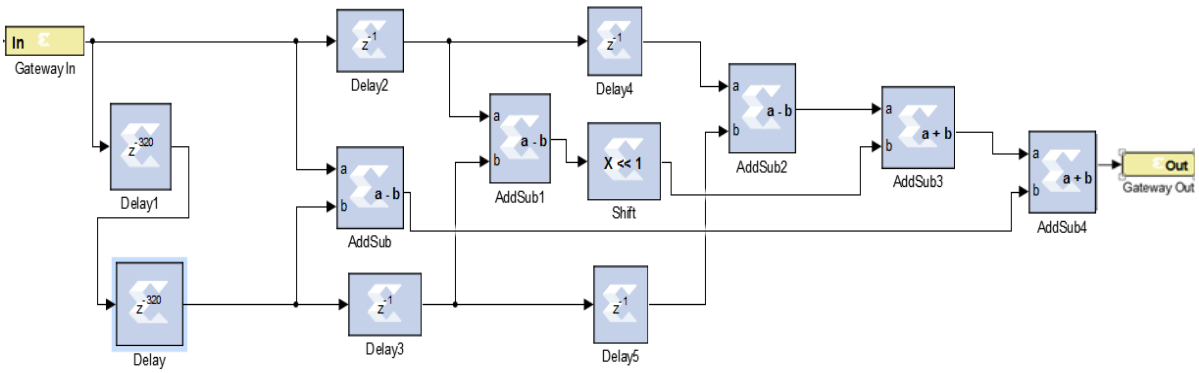
شکل ۵-۱۳ : ذخیره تصویر به صورت سریال و نحوه دستیابی به هر پیکسل تصویر توسط بلوک های تاخیر.

بیت هایی که قرار است در عملیات کانولوشن در اعداد ۱و ۱- کرنل سوبل ضرب شوند، به جای عمل ضرب از هم کم کرده و بیت هایی که می خواهند در اعداد ۲ و ۲- ضرب شوند، را ابتدا یکبار به سمت چپ شیفت داده که معادل ضربدر۲ است و سپس ازهم کم می‌کنیم. در نهایت حاصل را با هم جمع کرده و در خروجی قرار می‌دهیم. در شکل ۵-۱۴ نحوه پیاده‌سازی گرادیان عمودی لبه‌یاب سوبل توسط بلوک‌های محیط شبیه‌سازی System Generator نمایش داده شده است.



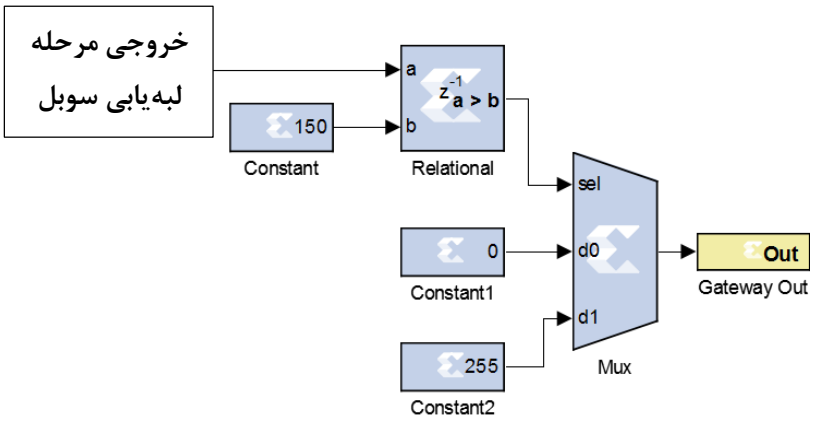
شکل ۵-۱۴: گرادیان عمودی لبه‌یاب سوبل فقط برای یک ماتریس ۳*۳ از ماتریس تصویر (براساس XSG).

از آنجایی که ابعاد تصویر ورودی مرحله لبه یابی ۳۲۰*۹۹ است. در شکل ۵-۱۳ تأخیر Delay و Delay1 را Z^{-320} یعنی تاخیر به اندازه تعداد ستون‌های تصویر مطابق شکل ۵-۱۵ انتخاب می‌کنیم.



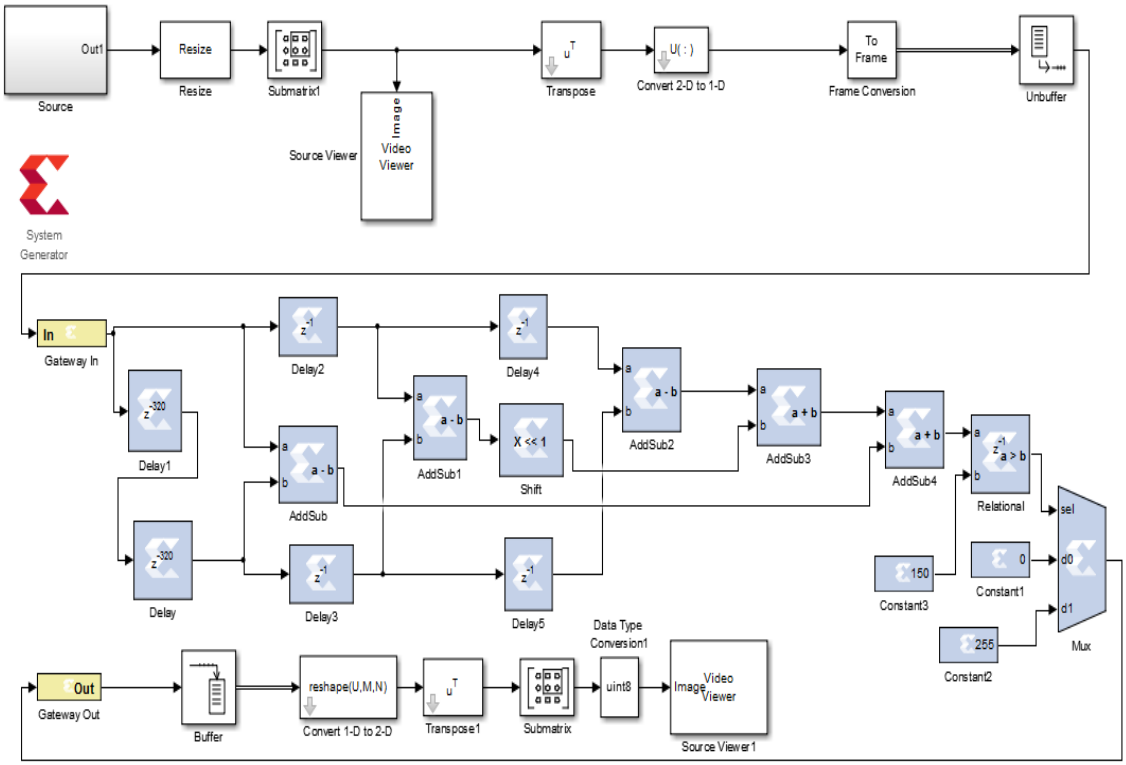
شکل ۵-۱۵: گرادیان عمودی لبه‌یاب سوبل برای کل ماتریس تصویر.

با انجام مراحل بیان شده در قسمت بالا، تمامی لبه‌های تصویر پیدا شده‌اند. حال برای انتخاب لبه‌های قوی نیاز است که از یک آستانه‌گذاری مناسب استفاده شود. برای پیاده سازی عمل آستانه‌گذاری در نرم‌افزار System Generator از بلوک‌های Mux، Relational، Constant مانند شکل ۵-۱۶ استفاده می‌گردد.



شکل ۵-۱۶: بلوک دیاگرام آستانه‌گذاری.

پیاده‌سازی الگوریتم پیشنهادی تا مرحله لبه‌یابی در حوزه دیجیتال در شکل ۵-۱۷ نشان داده شده است.



شکل ۵-۱۷: پیاده سازی مرحله لبه‌یابی و آستانه‌گذاری در حوزه دیجیتال.

۵-۵-۲ پیاده‌سازی تبدیل هاف و مشخص کردن خطوط در حوزه دیجیتال

در حوزه دیجیتال همانطور که گفته شد، تصویر به صورت سریال ذخیره شده و یک پیکسل یک پیکسل در ورودی قرار می‌گیرد. همچنین برای پیاده‌سازی تبدیل هاف و ادامه الگوریتم باید از چندین حلقه For تودرتو (به عنوان مثال همانطوری که در جدول ۵-۲ برای پیاده‌سازی عملگر گسترش در حالت دو بعدی از دو حلقه For تودرتو و در حالت یک بعدی از ۴ حلقه For تودرتو استفاده شده است.) استفاده کنیم.

لذا با توجه به بیان مسائل بالا پیاده‌سازی این مرحله از کار بر روی System Generator به دلیل محدودیت‌های MCode نویسی امکان‌پذیر نبوده یا با مشکلات فراوان و صرف زمان زیادی امکان‌پذیر می‌باشد. بنابراین با تحقیقات در این زمینه به این نتیجه رسیدیم که برای پیاده‌سازی ادامه الگوریتم

پیشنهادی از HDL Coder به جای Xilinx System Generator استفاده کنیم. همانطور که در فصل ۴ توضیح داده شد، یک فایل Testbench و یک فایل طراحی تعریف می‌کنیم. اطلاعات مرحله لبه‌یابی را به عنوان ورودی فایل Testbench قرار می‌دهیم. سپس با استفاده از یک حلقه For تودرتو اطلاعات لبه را به صورت یک پیکسل یک پیکسل توسط فراخوانی تابع، وارد فایل طراحی می‌کنیم. در فایل طراحی ابتدا لازم است برای حفظ حالت بین فراخوانی‌های الگوریتم متلب از متغیرهای محلّی استفاده کنیم. متغیرهای محلّی در واقع ما را قادر به مدل‌کردن ثبات‌ها می‌سازند. قبل از استفاده از متغیرهای محلی باید توسط یک دستور نوع و اندازه آن مشخص و مقداردهی اولیه شود. در مثال زیر نحوه تعریف متغیر محلّی بیان شده است.

مثال ۵-۱:

```
% Initialize with a constant
Persistent p;
if isempty(p)
p = fi(0,0,8,0);
end
```

همچنین متغیرهای محلّی از نوع آرایه هم وجود دارند که ما را قادر به مدل‌کردن RAM می‌سازند. نوع و اندازه متغیرهای محلّی آرایه‌ای هم همانند متغیرهای محلّی باید مشخص و مقداردهی اولیه شوند.

ما در فایل طراحی از دستور شرطی if استفاده کردیم تا بتوانیم با درست بودن شرط if یک عملیات مثلاً اجرا دستور Find یا پیدا کردن تبدیل هاف همه در همان فایل طراحی اوّل اجرا شود و دیگر نیاز به تعریف چندین فایل طراحی نباشد. در جدول ۵-۳ نحوه استفاده از دستور if به طور خلاصه شرح داده شده است.

جدول ۵-۳ : نحوه استفاده از if در فایل طراحی HDL Coder.

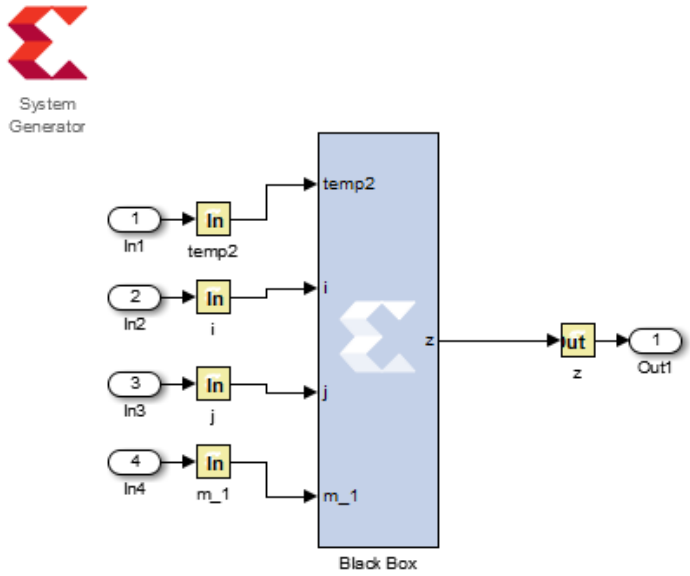
فایل Testbench	فایل طراحی

for m=1:2
for i=1:99
for j=1:320
[z]=fun1(.....,m);
end
end
end
 	
z=fun1(….,3);
	function z=find(…,m)
	persistent
	تعریف همه‌ی متغیرهای محلّی مورد نیاز و تعین اندازه آن‌ها
	if m==1
	دریافت اطلاعات لبه
	z=0;
	elseif m==2
	پیدا کردن پیکسل‌های ۱ (انجام دستور Find)
	z=0;
	elseif m==3
	پیدا کردن ماتریس تبدیل هاف، آستانه‌گذاری و...
	z=0;
	else
	مقداردهی به خروجی
	end
	end

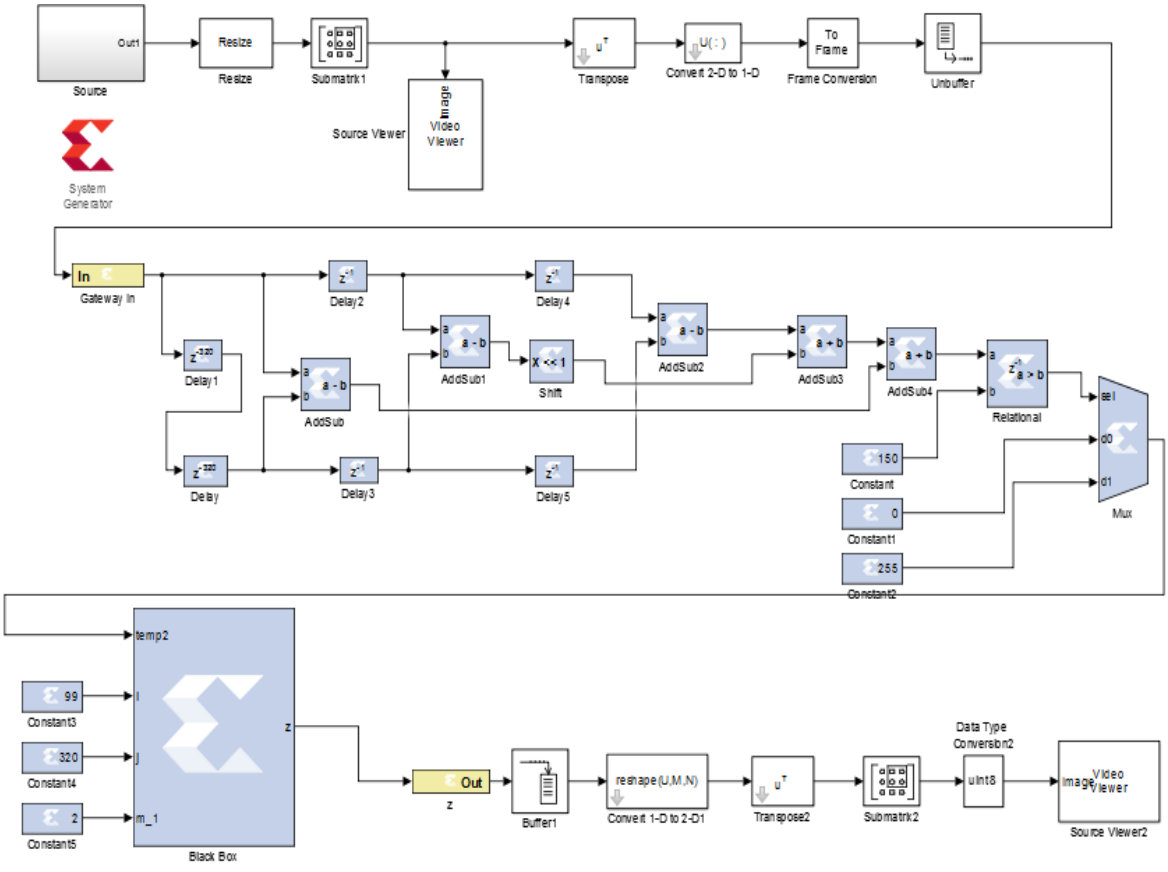
بعد از تعریف متغیرهای محلّی در ابتدا فایل طراحی و ذخیره کردن پیکسل‌های لبه، دستور Find را با استفاده از دو حلقه For تودرتو نوشته تا بتوان پیکسل هایی که ۱ هستند را پیدا کنیم. سپس با توجه به جدول ۵-۳ ماتریس تبدیل هاف را پیدا کرده و روی آن آستانه‌گذاری می‌کنیم. برای پیدا کردن قله‌ها در فضای تبدیل هاف تحلیل اجزای متصل را انجام می‌دهیم. درنهایت خروجی الگوریتم یک پیکسل یک پیسکل به فایل Testbench برگردانده می‌شود. حالا هم فایل Testbench و هم فایل طراحی آماده شده است. همان‌طوریکه که در پیوست نحوه استفاده از HDLCoder بیان شده، می‌توان الگوریتم نوشته شده در متلب را به Fixed point تبدیل کرده و کد HDL آن را تولید کنیم. همچنین این امکان وجود دارد که کد HDL تولید شده را با استفاده از فایل Testbench آزمایش کنیم.

در نتیجه می‌توان با انجام تنظیمات در HDLCoder یک بلوک System Generator برای مرحله تبدیل هاف و نمایش خطوط جاده همانطوری که در شکل ۵-۱۸ نشان داده شده است، تولید کنیم. این بلوک را به مجموعه بلوک‌های System Generator که در بالا عملیات لبه‌یابی را انجام می‌دادند، مطابق شکل

۱۹-۵ متصل کردیم و بنابراین به نتیجه مطلوب یعنی پیاده‌سازی الگوریتم پیشنهادی بر روی FPGA دست یافتیم.



شکل۵-۱۸ : بلوک سیستم ژنراتور تولیدی از HDL Coder (پیاده‌سازی مرحله تبدیل هاف و نمایش خطوط).



شکل ۵-۱۹ : پیاده‌سازی الگوریتم پیشنهادی بر روی FPGA.

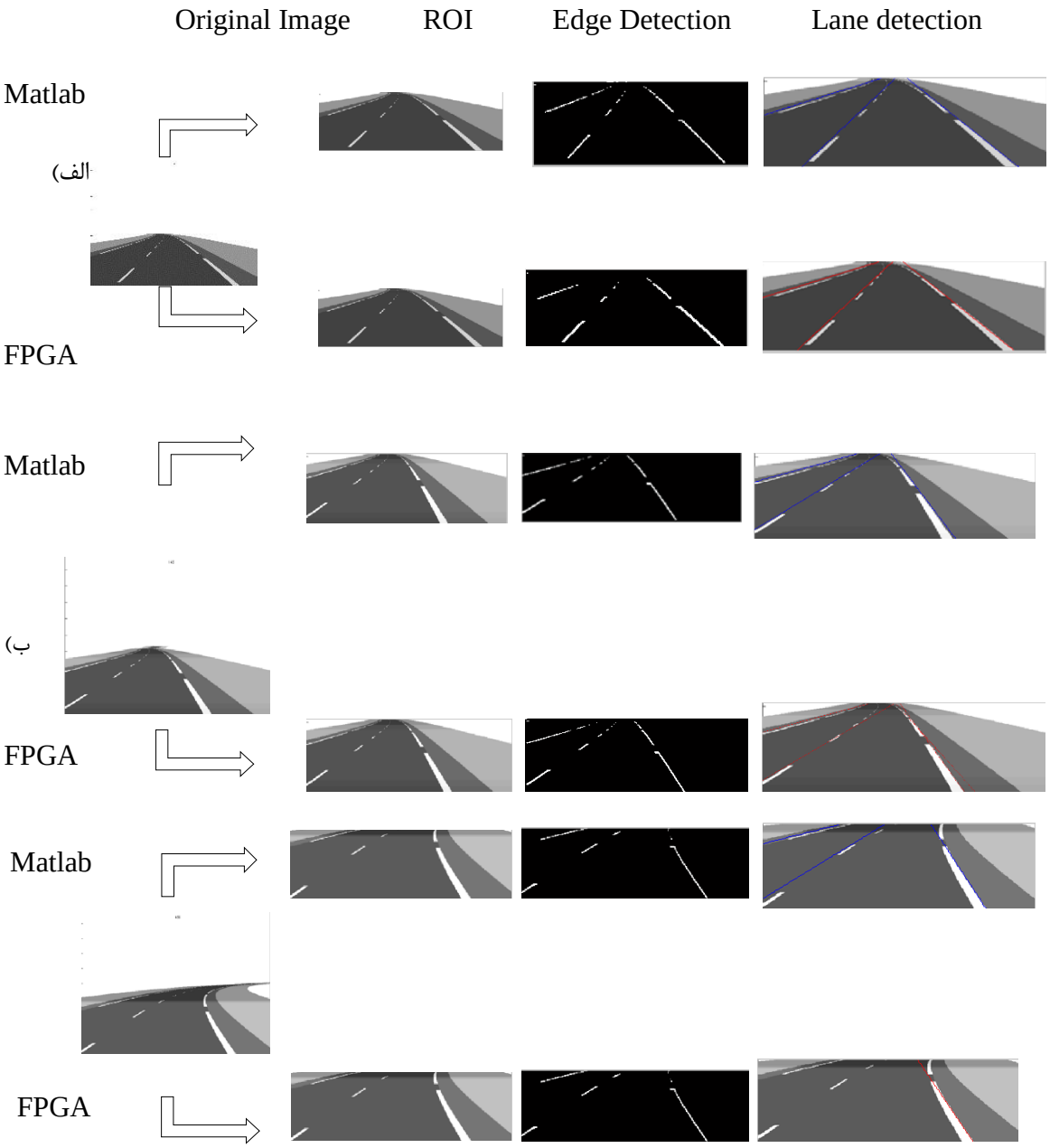
در طول شبیه‌سازی در حوزه دیجیتال دو پرسش مطرح می‌شود.

- دوره تناوب نمونه‌برداری بلوک‌های ورودی FPGA که فرمت داده‌ها را از آنالوگ به دیجیتال تبدیل می‌کنند چگونه باید باشد؟
- هر بلوک از شبیه‌ساز System Generator ورودی‌اش را با چه سطح بیتی دریافت نماید؟

در خصوص پرسش اول بایستی خاطر نشان کرد نه تنها دوره تناوب نمونه‌برداری درگاه ورودی FPGA نمی‌تواند از پریود نمونه‌برداری کل مدار کوچکتر باشد بلکه باید مضرب صحیحی از آن نیز باشد. علاوه بر این باید به این مسأله توجه داشت که هرچه پریود نمونه‌برداری کوچکتر باشد یا به عبارت دیگر فرکانس نمونه‌برداری بیشتر باشد، FPGA در فواصل زمانی کمتر و با دقت بیشتر از سیگنال آنالوگ ورودی‌اش نمونه می‌گیرد.

در مورد سوأل دوم باید گفت که تعداد بیت‌های هر بلوک دیجیتال دقت مدار را تعیین می‌کند. اگر تعداد بیت‌ها کم باشد دقت داده‌های ورودی هر بلوک کم شده و پاسخ خروجی به مقدار نهایی نخواهد رسید.

شکل ۵-۲۰ نتایج حاصل از شبیه‌سازی متلب و پیاده‌سازی بر روی FPGA برای سیستم تشخیص خط پیشنهادی را بر روی چندین تصویر از پایگاه‌داده نشان می‌دهد. در شکل ۵-۲۰ مشاهده می‌شود که مرزهای خط‌کشی با موفقیت استخراج شده‌اند و این حاکی از عملکرد خوب الگوریتم پیشنهادی است.



شکل۵-۲۰ : مقایسه بین نتایج متلب/FPGA.

در شکل ۵-۲۰ تصویر سمت چپ تصویر اصلی است که توسط دوربین گرفته شده، تصویر بعدی نتیجه انتخاب ناحیه مورد علاقه است، و سپس تصویر خروجی بلوک لبه‌یابی دیده می‌شود. آخرین تصویر هم نتیجه اعمال تبدیل هاف و تحلیل اجزای متصل است که خطوط خط‌کشی تشخیص داده شده‌اند. با مقایسه نتایج پیاده‌سازی سخت‌افزاری و شبیه‌سازی نرم‌افزاری سیستم تشخیص خطوط حاضر، تفاوت‌های اندکی در قسمت لبه‌یابی مشاهده می‌شود. که تاثیر چندانی در نتیجه نهایی سیستم ایجاد نمی‌کند. همچنین در قسمت تشخیص خطوط جاده با شبیه‌سازی در حوزه متلب هر سه خط

جاده به درستی تشخیص داده می‌شود اما در پیاده‌سازی سخت‌افزاری در چندین مورد که جاده دارای انحنای بیشتری است، الگوریتم پیشنهادی دو یا یک خط جاده را می‌تواند تشخیص دهد. البته در این پایان‌نامه هدف آزمایش الگوریتم بر روی پایگاه‌داده‌ای است که از جاده مستقیم تصویربرداری شده است. بنابراین ثابت شد که پیاده‌سازی سخت‌افزاری پیشنهادی نتایج خوب و قابل قبولی نسبت به شبیه‌سازی نرم‌افزاری فراهم می‌کند.

از آنجا که به دلیل تفاوت در سرعت کلاک و حجم فضای سخت‌افزاری، انتخاب نوع FPGA در نتایج شبیه‌سازی، آنالیز و سنتز مؤثر است، در این تحقیق طرح سخت‌افزاری خود را بر روی FPGA از خانواده spartan3 اجرا می‌کنیم. از خانواده spartan3 مدل xc3s1500I جهت پیاده‌سازی به کار گرفته می‌شود. همچنین برنامه بر روی کامپیوتر با پردازنده سه هسته‌ای و فرکانس کاری 2.13GHz اجرا و پیاده‌سازی شده است.

برای ارزیابی عملکرد پیاده‌سازی سخت‌افزاری و شبیه‌سازی نرم‌افزاری و تعیین نرخ تشخیص، تعدادی از تصاویر جاده به طور تصادفی از پایگاه‌داده موردنظر انتخاب شدند و به عنوان تصاویر ورودی قرار گرفتند. برای هر آزمایش همان تصویر ورودی در هر دو پیاده‌سازی مورد آزمایش قرار گرفت، نتایج حاصل از هر دو پیاده‌سازی در جدول ۵-۴ ارائه شده است. زمان اجرا در متلب ۱۷,۵ برابر بزرگتر از زمان اجرا بر روی FPGA است. این موضوع منجر به این نتیجه‌گیری می‌شود که پیاده‌سازی سخت‌افزاری بسیار سودمند است به خصوص اگر ما قصد انجام کاری را داریم که با شرایط زمان واقعی انجام می‌شود.

جدول ۵-۴ : عملکرد سیستم.

	FPGA	Matlab
متوسط زمان اجرا	۲۰ms	۳۵۰ms
نرخ تشخیص	٪۸۷	٪۹۰

فصل ششم :جمع بندی، نتیجه‌گیری و پیشنهادات

۶-۱ نتیجه‌گیری

در این پایان‌نامه یک سیستم پیاده‌سازی تشخیص خطوط جاده بر روی FPGA برمبنای الگوریتم تبدیل هاف، تحلیل اجزای متصل و بعضی از تکنولوژی‌های پیش‌پردازش به منظور آماده کردن تصویر گرفته شده برای فرآیند تشخیص خطوط ارائه شده است.

معماری سخت‌افزاری با استفاده از کتابخانه Xilinx System Generator تحت سیمولینک متلب طراحی شده که ابزار بسیار سودمندی برای توسعه الگوریتم‌های بینایی کامپیوتر است. استفاده از نرم‌افزار System Generator شکاف میان طراحی سطح بالا و پیاده‌سازی سخت‌افزاری را پر کرده است و نتایج قابل اعتمادی را به ما می‌دهد.

استفاده از FPGA به عنوان بستر سخت‌افزاری با توجه به توان مصرف پایین و سرعت عملکرد بالایی که دارد و همچنین قابلیت‌هایی از قبیل پیکربندی مجدد و موازی‌سازی می‌تواند یکی از بهترین گزینه‌ها به عنوان پردازنده‌های پردازشی باشد که روزبه‌روز گرایش به استفاده از آن‌ها رو به افزایش است.

روش تشخیص خطوط بر روی صحنه‌های مختلف از جاده آزمایش شده و به نرخ تشخیص قابل قبولی با زمان اجرای کوتاه دست یافته‌ایم.

به عنوان پیشنهاد برای تحقیقات آتی می‌توان به موارد زیر اشاره کرد:

- برای دستیابی به نتایج بهتر در جاده‌هایی با انحنا می‌توان الگوریتم تبدیل هاف را به گونه‌ای تغییر داد که علاوه بر تشخیص خطوط مستقیم، انحنا هم تشخیص دهد.
- از تصاویر پایگاه‌داده‌هایی که در آن‌ها هوا ابری، بارانی، مه‌آلود و..... است استفاده کرده و با انتخاب یک روش پیش‌پردازش این موانع را رفع کنیم. تا تشخیص خطوط به درستی انجام گیرد.
- بعد از تشخیص خطوط جاده به دنبال ردیابی خطوط باشیم.

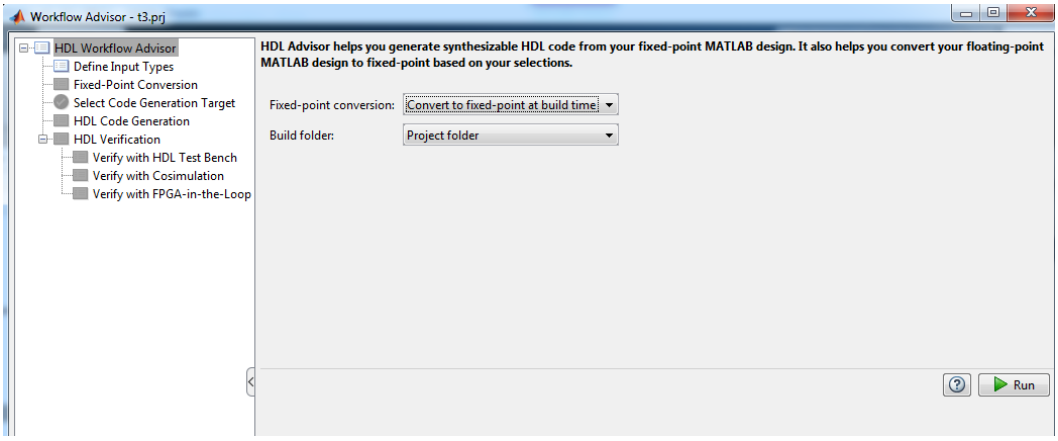
پیوست

در این قسمت به‌طور خلاصه نحوه کار با HDL Coder شرح داده شده است :

برای تولید کد HDL از متلب، نیاز است کد حاصل به دو فایل تقسیم شود :۱. فایل Test bench، ۲. فایل طراحی. فایل طراحی برای پیاده‌سازی الگوریتم در FPGA یا ASIC مورد استفاده قرار می‌گیرد. فایل Test bench داده ورودی را برای فایل طراحی فراهم و همچنین خروجی فایل طراحی را دریافت می‌کند.

برای تولید یک Black Box سیستم ژنراتور با استفاده از HDL Coder، بعد از نصب نرم‌افزار Xilinx System Generator روی سیستم روند کار به ترتیب زیر است :

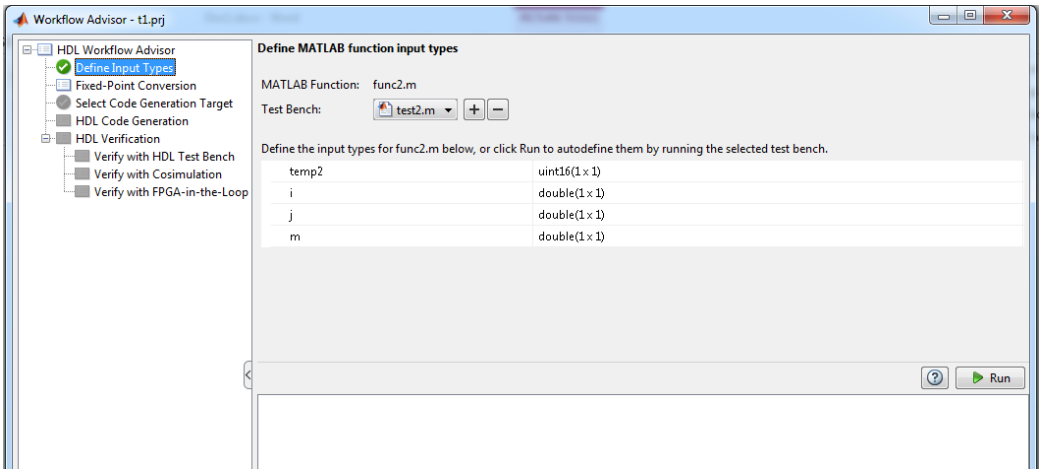
۱- متلب شامل دو فایل است یکی اصل فایل طراحی (مثلاً mlhdlc_heq.m) که به VHDL تبدیل خواهد شد و دیگری فایل تست است (مثلاً mlhdlc_heq_tb.m).
۲- وارد محیط فرمان متلب شده و وارد قسمت APPs می‌شویم و HDL coder را انتخاب می‌کنیم.
۳- یک پروژه باز کرده و فایل اصلی و تست را که قبلا ایجاد کرده و برنامه‌های مورد نظر درآن‌ها نوشته شده است، را اضافه می‌کنیم.
۴- Workflow advisor را کلیک می‌کنیم.
۵- در صفحه باز شده گزینه Fixed point conversion را انتخاب می‌کنیم (شکل پیوست-۱).



شکل پیوست-۱ : سربرگ HDL Workflow Advisor .

۶. تعریف نوع ورودی‌ها

در این مرحله می‌توان نوع ورودی‌ها را به صورت دستی یا حتی با اجرا کردن Test bench تعریف کرد. با کلیک بر روی Run این مرحله اجرا می‌شود (شکل پیوست-۲).

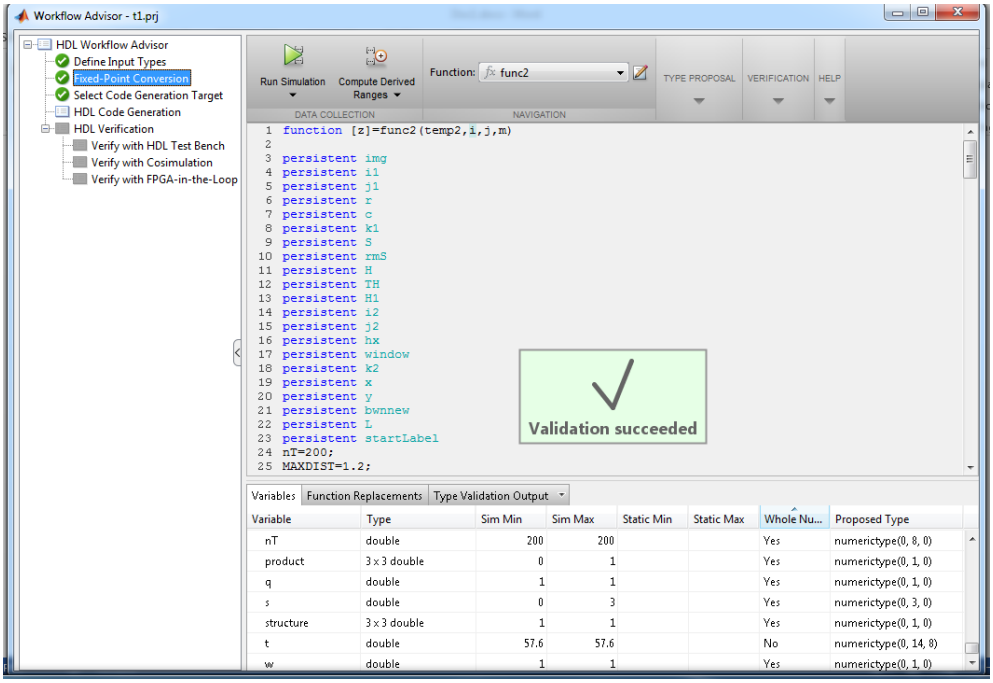


شکل پیوست-۲ : تعریف نوع ورودی‌ها.

۷. اجرای شبیه‌سازی

در مرحله Fixed point conversion با کلیک کردن بر روی Run simulation، طراحی با انواع داده ورودی در مرحله قبل کامپایل می‌شود. حال می توان مشاهده کرد آیا کامپایل موفقیت آمیز بوده یا

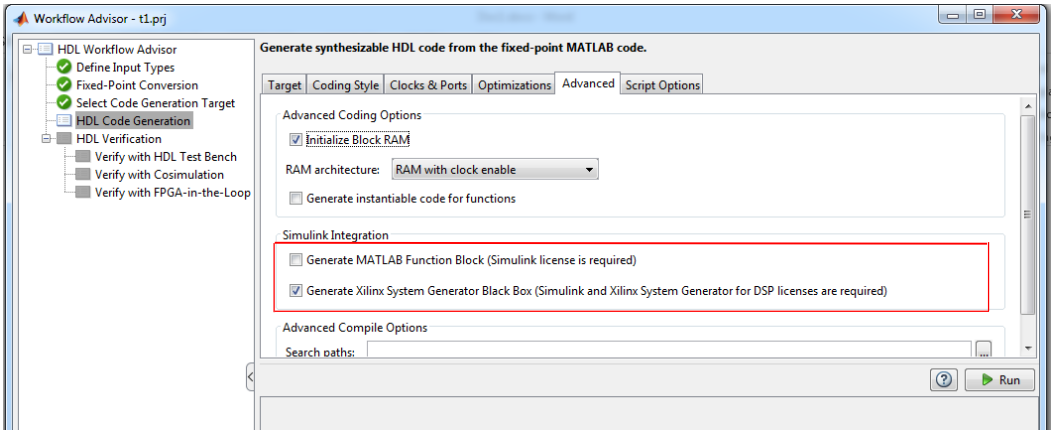
خیر؟ همچنین در این مرحله جدول متغییرها برای همه توابع در طراحی نشان داده می‌شود (شکل پیوست-۳).



شکل پیوست-۳ : اجرای موفقیت آمیز شبیه سازی.

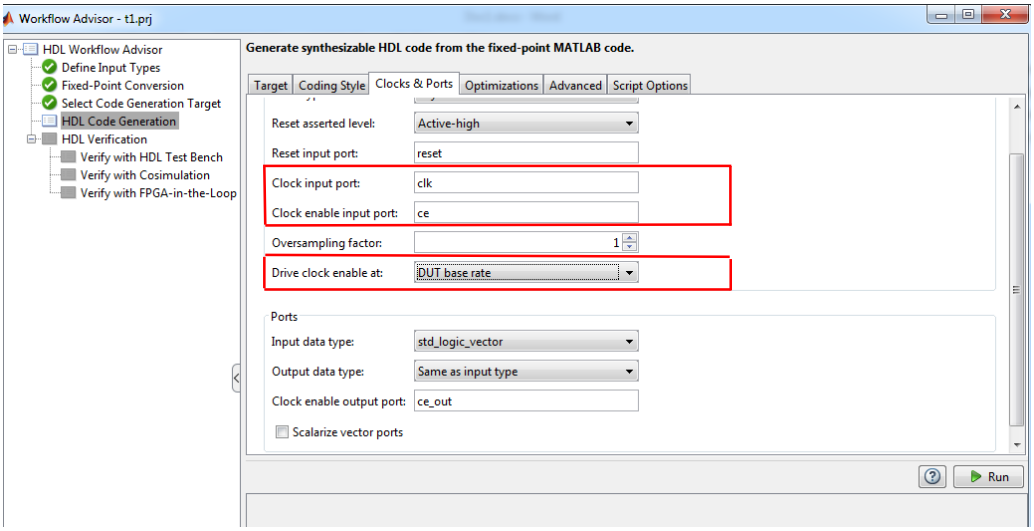
۸. مرحله HDL Code Generation

در این مرحله برای ساختن یک بلوک Xilinx system Generator Black Box به سربرگ advanced رفته و گزینه مربوط را انتخاب می‌کنیم (شکل پیوست-۴).



شکل پیوست-۴: تولید بلوکSystem Generator.

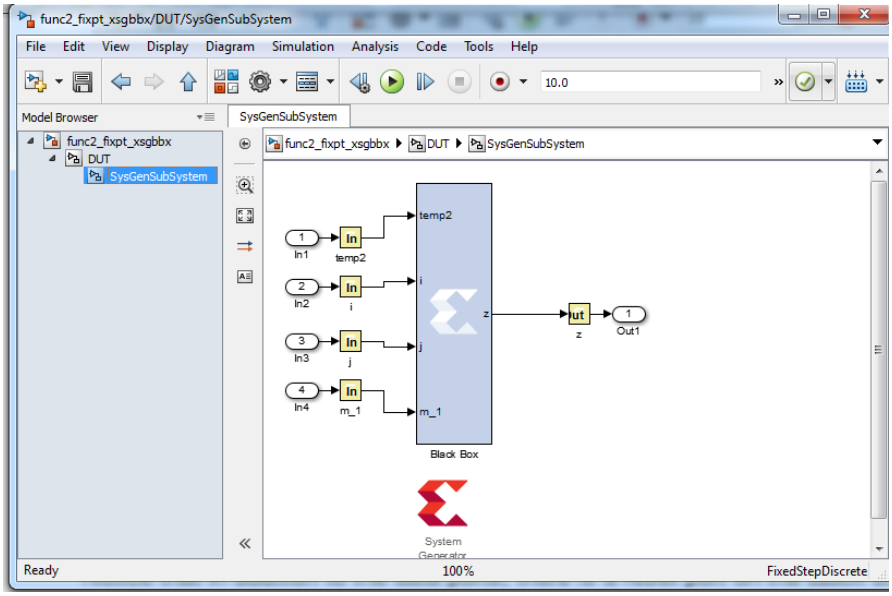
سپس با انتخاب سربرگ cloks & ports می‌توان تنظیمات مربوط به کلاک را با توجه به شکل شکل پیوست-۵ انجام داد.



شکل پیوست-۵: تنظیمات مربوط به کلاک‌ها و پورت‌ها.

با کلیک کردن بر روی Run یک مدل جدید بعد از تولید کد HDL مطابق شکل پیوست-۶ باز می‌شود. که شامل یک Subsystem به نام DUT در سطح بالا است. زیر مجموعه DUT یک Subsystem سیستم ژنراتور به نام SysGenSubSystem دارد که شامل موارد زیر است :

- یک بلوکBlack Box سیستم ژنراتور
- یک بلوک سیستم ژنراتور
- بلوک‌های Gateway-in
- بلوک‌های Gateway-out



شکل پیوست-۶: بلوک Black Box سیستم ژنراتور تولیده شده توسط HDL Coder.

در زمان برنامه‌نویسی در فایل طراحی یک سری محدودیت ها وجود دارد. در جدول پیوست-۱ توابع کتابخانه Fixed-point که HDL Code Generation از آن حمایت می‌کند، آورده شده است.

جدول پیوست-۱: توابع کتابخانه‌ای Fixed-point.

Function	Restrictions
abs	Double data type not supported.
add	None
all	Double data type not supported.
any	Double data type not supported.
bitand	None
bitandreduce	None
bitcmp	None
bitconcat	None
bitget	None
bitor	None
bitorreduce	None
bitreplicate	None
bitrol	None

ادامه جدول پیوست-۱:

Function	Restrictions
bitror	None
bitset	None
bitshift	None
bitsliceget	None
bitsll	None
bitsra	None
bitsrl	None
bitxor	None
bitxorreduce	None
ceil	None
complex	None
conj	None
convergent	None
ctranspose	None
divide	• For HDL Code generation, the divisor must be a constant and a power of two. • Non-fi inputs must be constant; that is, their values must be known at compile time so that they can be cast to fi objects. • Complex and imaginary divisors are not supported. • Code generation in MATLAB does not support the syntax T.divide(a,b).
end	None
eps	• Supported for scalar fixed-point signals only. • Supported for scalar, vector, and matrix, fi single and fi double signals.
eq	None
fi	None
fimath	None
fix	None
floor	None
ge	None
getlsb	None
getmsb	None
gt	None
horzcat	None
imag	None
int8, int16, int32	None
iscolumn	None

ادامه جدول پیوست-۱:

Function	Restrictions
isempty	None
isequal	None
isfi	None
isfimath	None
isfimathlocal	None
isfinite	None
isinf	None
isnan	None
isnumeric	None
isnumerictype	None
isreal	None
isrow	None
isscalar	None
issigned	None
isvector	None
le	None
length	None
logical	None
lowerbound	None
lsb	None
lt	None
max	None
min	None
minus	None
mpower	Both inputs must be scalar, and the exponent input, k, must be a constant integer.
mtimes	None
ndims	None

ادامه جدول پیوست-۱:

Function	Restrictions
ne	None
nearest	None
numberofelements	None
numerictype	None
plus	Inputs cannot be data type logical.
power	Both inputs must be scalar, and the exponent input, k, must be a constant integer.
range	None
real	None
realmax	None
realmin	None
reinterpretcast	None
repmat	None
rescale	None
reshape	None
round	None
sfi	None
sign	None
size	None
sqrt	None
sub	None
subsasgn	Supported data types for HDL code generation are listed in "Supported Data Types" on page 1-2
subsref	Supported data types for HDL code generation are listed in "Supported Data Types" on page 1-2
sum	None
times	Inputs cannot be data type logical.
transpose	None
ufi	None
uint8, uint16, uint32	None
uminus	None
uplus	Inputs cannot be data type logical.
upperbound	None
vertcat	None

مراجع

[1] غفریان حسینی. هادی , حسین زاده. افشین , " نمایشگر خواب آلودگی راننده و سیستم هشدار دهنده", *کنفرانس بین المللی حوادث رانندگی و جاده ای*, دانشگاه تهران, ۱۳۸۴ .

[2] P.Rau,"Drowsy driver detection and warning system for commercial vehicle drivers", National Highway Traffic Safety Administration , 2005.

[3] D.Royal," National Survey on Distracted and Driving Attitudes and Behaviors , " The Gallup Organization, March, 2003.

[4] L.M. Bergasa, J.U. Nuevo, M.A. Sotelo, R.Bavea, and E.Lopez, " Visual Monitoring of Driver Inattention ", *Studies in Computational Intelligence (SCI)*, 2008.

[5] M.Aly, "Real time detection of lane markers in urban streets," *in Proc.IEEE Intell. Veh. Symp*, pp. 7-12, Jun, 2008.

[6] C.W.Lin, H.Y.Wang and D. C. Tseng, "A robust lane detection and verification method for intelligent vehicles," *In Intelligent Information Technology Application*, vol. 1, pp. 521-524, December, 2009.

[7] Y.U. Yim and S.Y. Oh, "Three-feature based automatic lane detection algorithm (TFALDA) for autonomous driving," *IEEE Trans*, vol. 4, No. 4, pp. 219-225, Dec, 2003.

[8] S.Zhou, Y.Jiang, J. Xi, J. Gong, G. Xiong, and H. Chen, "A novel lane detection based on geometrical model and gabor filter," *IEEE*, pp. 59-64, June 2010.

[9] H. Zhao, Z. Teng, H.H. Kim, and D. J. Kang, "Annealed particle filter algorithm used for lane Detection and tracking." *Journal of Automation and Control Engineering 1.1*, 2013.

[10] M. Tan, B. Paula, and C.R. Jung, "Real-time detection and classification of road lane markings," *In Graphics, Patterns and Images (SIBGRAPI), IEEE 26th SIBGRAPI-Conference on*, pp. 83-90, 2013.

[11] W.Kayankit, and W.Suntiamorntut, "Hardware/software co-design for line Detection Algorithm on FPGA," *International conference on Electrical Engineering/Electronics*, vol. 1, pp.604-606, 2009.

[12] J. McCall, and M. Trivedi, "Video-based lane estimation and tracking for driver assistance: Survey, System and Evaluation," *IEEE Trans. On Intelligent Transportation Systems*, vol. 7, pp. 20–37, Mar, 2006.

[13] C. Farmer, "Crash avoidance potential of five vehicle technologies," *Insurance Institute For highway safety*, June 2008.

[14] A. Diop, A. Okpanachi, A. Abayomi, and E. Duke, "Lane departure warning system II," *Senior Design Project*, March 2009.

[15] Y.U. Yim and S.- Y. Oh, "Three-feature based automatic lane detection algorithm (TFALDA) for autonomous driving," *IEEE Trans. Intell. Transp. Syst*, vol. 4, No. 4, pp. 219-225, Dec. 2003.

[16] M.Aly, "Real time detection of lane markers in urban streets," *IEEE Intelligent Vehicles Symposium*, pp. 7-12, June 2008.

[17] H.Y. Cheng, C.C. Yu, C. C. Tseng, K. C. Fan, J. N. Hwang, and B. S. Jeng, "Hierarchical lane detection for different types of roads," *IEEE International Conference*, pp. 1349-1352, 2008.

[18] A.Borkar, M. Hayes, M. Smith, and S. Pankanti, "A layered approach to robust lane detection at night," *IEEE Intelligent Vehicles Symposium*, pp. 51-57, 2009.

[19] R. Marzotto, P. Zoratti, D. Bagni, A. Colombari, and V. Murino," A real-time versatile roadway path extraction and tracking on an FPGA platform", *ELSEVIER*, Volume 114, Issue 11, pp.1164–1179, November ,2010.

[20] S.Zhou, Y.Jiang, J. Xi, J. Gong, G. Xiong, and H. Chen, "A novel lane detection based on geometrical model and gabor filter," *IEEE Intelligent Vehicles Symposium (IV)*, pp. 59-64, June, 2010.

[21] P.Daigavane and P. Bajaj, "Road lane detection with improved canny edges using ant colony optimization," *in 3rd International Conference on Emerging Trends in Engineering and Technology (ICETET)* , pp. 76-80, 2010.

[22] Y.-C. Leng and C.-L. Chen, "Vision-based lane departure detection system in urban traffic scenes," *11th International Conference*, pp.1875- 1880, 2010.

[23] T.T. Tran, H.M.Cho, and S.B.Cho, "A robust method for detecting lane boundary in challenging scenes", *Information Technology Journal. Image Process*, vol. 10, No.12, pp. 2300-2307, 2011.

[24] F. Mariut, C.Fosalau, and D.Petrisor,"Lane mark detection using Hough transform," *International Conference and Exposition on Electrical and Power Engineering*, October, 2012.

[25] F. Cela, M. Bergasa, L. Sánchez, and A. Herrera, “Lanes detection based on unsupervised and adaptive classifier”, *IEEE*, 2013.

[26] Y. Li, A. Iqbal, and N.R. Gans, “Multiple lane boundary detection using a combination of low-level image features,” *17th International Conference on*, pp. 1682-1687, 2014.

[27] U.Ozgunalp and N.Dahnoun, "Robust lane detection and tracking based on novel feature extraction and lane categorization," *Speech and Signal Processing (ICASSP) International Conference on*, pp. 8129-8133, May,2014.

[28] D. Divya and S. Sushmap," FPGA Implementation of a distributed canny edge detector," *International Journal of Advanced Computational Engineering and Networking*, Vol. 1, Issue. 5, pp.46-51, July, 2013.

[29] G.T. Shrivakshan, and Dr.C. Chandrasekar,"A Comparison of various Edge Detection Techniques used in Image Processing," *IJCSI International Journal of Computer Science Issues*, Vol. 9, Issue 5, No. 1,pp.269-276, September 2012.

[30] R. E. R. C.González, Digital image processing, 3rd Edition: Prentice Hall, 2008.

[31] R. Woods, J. McAllister, G. Lightbody, and Y. Yi, “FPGA-based Implementation of Signal Processing Systems,” *John Wiley & Sons Ltd, United Kingdom*, 2008.

[32] U. Meyer-Baese, “Digital Signal Processing with Field Programmable Gate Arrays,” 3rd ed., *Springer*, 2007.

[33] J. Serrano, CERN, Geneva, and Switzerland, “Digital signal processing using Fied Programmable Gate Arrays,” *13th Beam Instrumentation Workshop*, pp.29-38, 2008.

[34] R. Faraji, "Design and manufacturing of industrial controllers based on FPGA," *Graduate University of Advanced Industrial Science and Master's Thesis,* 2012**.**

[35] G. Kavitkar ,and L. Paikrao," FPGA based Image Feature Extraction Using Xilinx system Generator," *(IJCSIT) International Journal of Computer Science and Information Technologies*, Vol. 5 (3) ,pp. 3743-3747, 2014.

[36] Y. Said, T. Saidani, F. Smach, and M. Atri, "Real time hardware co-simulation of edge detection for video processing system," *16th IEEE Mediterranean*, pp. 852-855, 2012**.**

Abstract :

In recent years a growing body research has been devoted to the issue of vehicle safety. Road detection systems are among new facilities improving a driver safety and alerting the driver when a vehicle strays it however the challenges such as lane appearance diversity, variation in clarity of image, changes in visibility conditions are still problematic for roadways detection systems.

In the present project, a new and efficient architecture is presented for road boundary detection. Opposed to previous drawings, in the current research the Sobel edge detection was used which improves edge detection and noise reduction by finding strong edges. After that Hough transform and Connected Component analysis were used for finding the lines.

In the first phase, circuit was implemented in the Matlab simulation environment and the operation of the proposed algorithm was evaluated. After the success of the initial design, System Generator graphic simulator and HDL Coder simulator were used to simulate the design on FPGA. The simulation results on FPGA, the average execution speed is 20ms and the detection rate is 87% were obtained the proposed system. According to these results can be said to have achieved a detection rate acceptable short running time.

Keywords : FPGA, Real-time processing, Lane detection, System Generator, HDL Coder.

Shahrood University of Technology

Faculty of Electrical and Robotic Engineering

Detection of roads path with modified edge detection and implementation on FPGA

Hanie Rostamian

Supervisor(s):

Dr.Ali Soleimani

Adviser:

Mohsen Biglari

September 2016