



دانشکده فنی مهاجر اصفهان
گروه مهندسی برق
پایان نامه مهندسی تکنولوژی الکترونیک

تشخیص و ردیابی جسم متحرک در محیط‌های عمومی روی پلتفرم ZYNQ (پردازش بی درنگ)

استاد راهنما
محمدرضا ذاکر حقیقی

نگارندگان
ابوالفضل سجادی هزاوه
سینا شیری

مرداد ۱۳۹۵



تقديم به

سپاس گذاری....

چکیده

این پژوهش یک ایده جدید برای پیاده سازی الگوریتم‌های پردازش تصویر به صورت بلادرنگ^۱ روی سیستم‌های توکار^۲ می‌باشد. در برخی مواقع از پردازش تصویر برای سیستم‌های امنیتی و حساس استفاده می‌شود. این سیستم‌ها نیاز مبرمی به دقت و سرعت زیاد در کنار امنیت بالا دارند. در روش‌های معمول برای پردازش تصویر، کاربر همیشه شاهد عدم حضور حداقل یکی از این سه اصل می‌باشد. هدف از این پژوهش علاوه بر عملی کردن سه اصل اساسی ذکر شده، پردازش آنی ویدئو با کیفیت و امنیت بسیار بالا در کنار قیمت مناسب و کارکرد به صورت مستقل خواهد بود.

برای دست یابی به هدف فوق، پیاده سازی اساسی ترین موضوع پردازش تصویر یعنی تشخیص و تعقیب یک جسم در یک محیط عمومی انجام خواهد گرفت. برای پردازش آنی و بی درنگ احتیاج به سیستمی متفاوت با کارکرد موازی است. با بررسی بین پلتفرم‌های مختلف بهترین گزینه برای پیاده سازی پردازش آنی سیستم‌های مبتنی بر تراشه‌های FPGA می‌باشد، که علاوه بر سرعت بالا بتواند ویژگی‌هایی همچون انعطاف‌پذیری بالا در کد نویسی و مصرف بهینه توان را تأمین نماید. برای این موضوع از تراشه ZYNQ-7000 SoC و نرم افزار Vivado استفاده شده است.

۱- Real Time

۲- Embedded Systems

فهرست مطالب

۱- فصل اول: مقدمه.....	۱
۱-۱- مقدمه	۱
۱-۲- طرح مسئله	۲
۱-۳- ضرورت و اهمیت موضوع	۳
۱-۴- پیشینه	۳
۱-۵- اهداف	۳
۱-۶- روش های تحقیق	۴
۲- فصل دوم: الگوریتم ردیابی شیء.....	۵
۲-۱- مقدمه	۵
۲-۲- پردازش تصویر	۶
۲-۳- بینایی ماشین	۶
۲-۴- بخش های مهم در بینایی در بینایی ماشین	۷
۲-۵- الگوریتم ردیابی با روش k-means خودکار	۱۰
۲-۵-۱- دریافت فریم های ورودی	۱۰
۲-۵-۲- پیش پردازش و استخراج ویژگی	۱۰
۲-۵-۳- تناظرابی با KLT	۱۱
۲-۵-۴- تعیین جهت و اندازه حرکت نقاط ویژگی	۱۱
۲-۵-۵- کشف اشیاء متحرک	۱۲

۱۲	۶-۵-۲- تخمین تعداد اشیاء.....
۱۲	۷-۵-۲- ردیابی با روش k-means اتوماتیک.....
۱۴	۶-۲- الگوریتم ردیابی با روش SIFT.....
۱۴	۶-۲-۱- ساخت فضای شاخص.....
۱۴	۶-۲-۲- گرفتن تفاوت ها از گوس.....
۱۴	۶-۲-۳- تعیین محل اکسترمم DoG.....
۱۴	۶-۲-۴- تعیین محل و مشخصات زیر مجموعه پیکسل ها در محل.....
۱۵	۶-۲-۵- حذف keypoint های ضعیف و لبه ها.....
۱۵	۶-۲-۶- اختصاص جهت به نقاط کلیدی.....
۱۵	۶-۲-۷- ساخت توصیف کننده نقاط کلیدی.....
۱۵	۶-۲-۸- شروع کار با ویژگی های مورد نظر.....
۱۷	۳- فصل سوم: بررسی تراشه ZYNQ.....
۱۷	۳-۱- مقدمه.....
۱۸	۳-۲- پلتفرم های موجود برای انجام پیاده سازی فرآیند.....
۲۱	۳-۳- توضیح مختصر درباره ی ZYNQ.....
۲۲	۳-۳-۱- واحد پردازش کاربردی (PS).....
۲۵	۳-۳-۲- واحد برنامه ریزی منطقی (PL).....
۲۷	۳-۳-۳- نحوه ارتباطات PS و PL.....
۳۸	۳-۳-۴- خصوصیات مهم ZYNQ.....
۳۸	۳-۴- مزیت های ZYNQ نسبت به FPGA.....
۳۹	۳-۵- ارتباط با دیگر شرکت ها.....
۴۰	۳-۶- بررسی نهایی ZYNQ.....

۴۲.....	۷-۳-بررسی دقیق تفاوت بین ZYNQ و SPARTAN 6
۴۲.....	۱-۷-۳-بررسی توان مصرفی
۴۳.....	۲-۷-۳-سرعت انتقالی پایه های I/O
۴۴.....	۳-۷-۳-امکان استفاده از سیستم عاملهای مختلف و پر کاربرد
۴۷.....	۸-۳-بررسی نرم افزارهای مورد استفاده برای برنامه ریزی ZYNQ
۴۷.....	۱-۸-۳-نرم افزار VIVADO
۴۹.....	۲-۸-۳-نرم افزار HLS (High Level Syntheses)
۵۱.....	۳-۸-۳-نرم افزار Xilinx Software Development Kit (XSDK)
۵۳.....	۴-۸-۳-نرم افزار SDSoc (Software-Defined System on chip)
۵۶.....	۴-فصل چهارم: نحوه پیاده سازی
۵۶.....	۱-۴-مقدمه
۵۷.....	۲-۴-مقایسه زبان ها و نرم افزار ها در حوزه پردازش تصویر
۵۸.....	۳-۴-توضیح الگوریتم استفاده شده
۵۸.....	۱-۳-۴-نحوه ردیابی شیء...
۵۹.....	۲-۳-۴-مشخص کردن مختصات شیء
۶۰.....	۳-۳-۴-بدست آوردن فاصله شیء از دوربین
۶۱.....	۴-۴-پیاده سازی الگوریتم مورد نظر در نرم افزار متلب
۶۲.....	۱-۴-۴-بررسی کد
۶۵.....	۵-۴-پیاده سازی الگوریتم مورد نظر در کتابخانه OpenCV
۶۶.....	۱-۵-۴-بررسی کد
۶۹.....	۶-۴-پیاده سازی الگوریتم مورد نظر روی تراشه ZYNQ
۶۹.....	۱-۶-۴-استفاده از سیستم عامل لینوکس و کد OpenCV

۶۹	۴-۶-۲-ساخت هسته اختصاصی مورد نظر و پیاده سازی آن
۸۵	۵-فصل پنجم: نتایج به دست آمده
۸۵	۵-۱-مقدمه
۸۶	۵-۲-نتایج به دست آمده از عملیات سنتز هسته
۸۶	۵-۲-۱-زمان بندی هسته
۸۷	۵-۲-۲-زمان تاخیر (سیکل ساعتی)
۸۸	۵-۲-۳-پیش بینی شرایط بعد از پیاده سازی
۹۰	۵-۲-۴-شرایط ورودی و خروجی ها
۹۲	۵-۳-نتایج به دست آمده از عملیات پیاده سازی سخت افزار و ساخت Bit Stream
۹۳	۵-۳-۱-توان مصرفی
۹۴	۵-۴-مقایسه سرعت اجرا بین روش های موجود
۹۴	۵-۴-۱-نتایج نرم افزار متلب
۹۵	۵-۴-۲-نتایج کتابخانه OpenCV
۹۵	۵-۴-۳-نتایج نرم افزار HLS
۹۷	۶-نتیجه گیری
۹۸	۷-مراجع

فهرست جداول

جدول (۱-۳) بررسی مدارات مجتمع ASI C و ASSP.....	۲۰
جدول (۲-۳) جدول مقایسه ZYNQ با مدارات مجتمع ASSP، ASI C و CHI P Sol ut i on.....	۴۱
جدول (۱-۴) مشخصات سخت‌افزاری.....	۶۲
جدول (۲-۴) نوع متغیرهای ورودی/خروجی هسته.....	۷۳
جدول (۳-۴) آدرس‌دهی بلوک‌ها.....	۷۹
جدول (۱-۵) زمان‌بندی هسته.....	۸۶
جدول (۲-۵) تاخیر در خروجی.....	۸۷
جدول (۳-۵) تاخیر بعد از اعمال حالت Data Fl ow.....	۸۷
جدول (۴-۵) تاخیر بعد از اعمال دو حالت بهینه‌سازی.....	۸۸
جدول (۵-۵) درصد میزان مصرف قسمت‌های مختلف.....	۸۹
جدول (۶-۵) آمار استفاده از قطعات.....	۸۹
جدول (۷-۵) FI FO.....	۹۰
جدول (۸-۵) ورودی خروجی هسته.....	۹۱
جدول (۹-۵) میزان مصرف توان تراشه.....	۹۳
جدول (۱۰-۵) زمان اجرا الگوریتم در نرم‌افزار متلب.....	۹۴
جدول (۱۱-۵) زمان اجرا الگوریتم در کتابخانه OpenCV.....	۹۵
جدول (۱۲-۵) زمان اجرا الگوریتم در نرم‌افزار HLS.....	۹۵

فهرست نمودارها

نمودار (۱-۳) بررسی توان مصرفی پلتفرم‌های مختلف..... ۴۳

نمودار (۱-۵) میزان مصرف قطعات اولیه موجود در تراشه (تخمین)..... ۹۲

نمودار (۲-۵) میزان مصرف قطعات اولیه موجود در تراشه (بعد از پیاده سازی)..... ۹۲

نمودار (۳-۵) میزان مصرف توان..... ۹۳

نمودار (۴-۵) مقایسه حالات پیاده سازی (یک فریم)..... ۹۶

نمودار (۵-۵) مقایسه حالات پیاده سازی (ویدئو ۲۰۰ فریمی)..... ۹۶

فهرست اشکال

شکل (۱-۲) سطوح پردازش تصویر.....	۹
شکل (۲-۲) فلوچارت الگوریتم ردیابی با روش k-means اتوماتیک.....	۱۳
شکل (۳-۲) فلوچارت الگوریتم SI FT.....	۱۶
شکل (۱-۳) انواع روش‌های تولید مدارات مجتمع.....	۱۸
شکل (۲-۳) بلوک دیاگرام ZYNQ.....	۲۱
شکل (۳-۳) نحوه اتصال AXI.....	۲۸
شکل (۴-۳) سمبل ارتباطی مستر در AXI.....	۲۹
شکل (۵-۳) سمبل ارتباطی اسلیو در AXI.....	۲۹
شکل (۶-۳) ارتباط ساده AXI.....	۳۰
شکل (۷-۳) ارتباط کامل در AXI.....	۳۰
شکل (۸-۳) کانال‌های مختلف در AXI.....	۳۱
شکل (۹-۳) ارتباط AXI Interconnect.....	۳۲
شکل (۱۰-۳) مثال ساده AXI Stream.....	۳۴
شکل (۱۱-۳) مثال ارتباط AXI MM و AXI Stream و بلعکس.....	۳۵
شکل (۱۲-۳) خصوصیات ZYNQ.....	۳۶
شکل (۱۳-۳) دایره ارتباطی Xi l i nx با دیگر نرم‌افزارها.....	۳۹
شکل (۱۴-۳) مقایسه سرعت نسل‌های مختلف FPGA.....	۴۳
شکل (۱۵-۳) لوگو لینوکس.....	۴۴

شکل (۱۶-۳) لوگو اندروید.....	۴۴
شکل (۱۷-۳) free RTOS لوگو.....	۴۵
شکل (۱۸-۳) محیط نرم افزار VI VADO.....	۴۹
شکل (۱۹-۳) محیط نرم افزار HLS.....	۵۱
شکل (۲۰-۳) محیط نرم افزار SDK.....	۵۲
شکل (۲۱-۳) فلوچارت طراحی در نرم افزار SDSoC.....	۵۴
شکل (۱-۴) نحوه موقعیت یابی شیء.....	۵۹
شکل (۲-۴) نتیجه پردازش با نرم افزار متلب.....	۶۴
شکل (۳-۴) نتیجه پردازش با کتابخانه OpenCV.....	۶۸
شکل (۴-۴) مراحل پیاده سازی روی تراشه ZYNQ.....	۷۰
شکل (۵-۴) فایل های مورد نیاز هسته.....	۷۱
شکل (۶-۴) تصویر قبل اعمال به هسته.....	۷۸
شکل (۷-۴) تصویر بعد از اعمال به هسته.....	۷۸
شکل (۸-۴) نحوه اتصال بلوک ها.....	۷۹
شکل (۹-۴) نمای جانبی برد.....	۸۳
شکل (۱۰-۴) نمای جانبی برد.....	۸۳
شکل (۱۱-۴) نمای بالایی برد.....	۸۴

فصل اول: مقدمه

۱-۱- مقدمه

برای اولین بار در دهه‌ی ۶۰ میلادی در دانشگاه ماری لند^۱ آمریکا اولین قدم‌ها در راستای علم پردازش تصویر^۲ برداشته شد، که عنوان ارائه شده برای این تحقیقات تصویربرداری ماهواره‌ای و کاربردهای پزشکی بود [۱]. با دقت در زمینه پیشرفت تکنولوژی به صورت واضح می‌توان دریافت که پردازش تصویر به سرعت به یکی از پرکاربردترین علوم در تمامی زمینه‌ها تبدیل شده است. در حال حاضر در اکثر سیستم‌های اتوماسیون از بینایی ماشین^۳ استفاده شده است که نمونه‌هایی از آن کاربرد صنعتی، پزشکی، نظامی و امنیتی، کشاورزی، تردد و ترافیک و کاربرد هوا و فضا را می‌توان نام برد [۲].

۱- Mary Land

۲- Image Processing

۳- Machine Vision

یکی از مسائل مهم و در حال توسعه در پردازش تصویر و بینایی ماشین^۱، مسئله ردیابی اشیاء است. در واقع ردیابی اشیاء، نمایش تغییرات موقعیت یک شیء و دنبال کردن آن در یک دنباله تصاویر ویدئویی با یک هدف خاص می باشد [۳].

۱-۲- طرح مسئله

از سال های پیش نرم افزارهای زیادی برای تسهیل کاربردهای پردازش تصویر توسعه یافته اند که شاید معروف ترین آنها جعبه ابزار پردازش تصویر در نرم افزار MATLAB باشد. اما کسانی که تجربه ی کار با این نرم افزار را دارند به خوبی می دانند که با وجود سهولت برنامه نویسی با آن، سرعت اجرای متلب به خصوص برای کار با ویدئو بسیار آزار دهنده است. همچنین این نرم افزار متن باز^۲ نیست. در مقابل OpenCV یک کتابخانه متن باز برای بینایی رایانه^۳ است. این کتابخانه به زبان های C، C++، Python، JAVA، ANDROID و IOS نوشته شده است و تحت لینوکس^۴، ویندوز و بسیاری پلتفرم های^۵ دیگر قابل اجرا است بنابراین می تواند برای کاربردهای بی درنگ^۶ بسیار مطلوب باشد [۴].

اگر بخواهیم سیستم کنترلی بر پایه پردازش ویدئو با OpenCV داشته باشیم پردازنده های زیادی از جمله (ARM، CPU(PC, Note Book, Raspberry pi) و هر سخت افزار که بتواند یکی از سیستم عامل های Linux، Windows، Android و یا IOS را پشتیبانی کند می تواند به کار گرفته شوند. در

۱- Machine Vision

۲- Open Source

۳- Computer Vision

۴- Linux

۵- Platforms

۶- Real Time

تمامی موارد ذکر شده حجم زیادی از پروسه وقف سیستم عامل می‌شود و سرعت پردازش بسیار کاهش پیدا می‌کند ولی در ZYNQ که پلتفرم انتخابی این پژوهش می‌باشد به دلیل وجود سه پردازنده (دو هسته ARM و یک تراشه FPGA) و تقسیم پردازش میان این بخش‌ها سرعت اجرا از بقیه موارد ذکر شده بالاتر است [۵].

۳-۱- ضرورت و اهمیت موضوع

ردیابی اشیاء در زمینه‌های مختلف مثل کاربردهای صنعتی که برای کنترل کیفیت و تفکیک تولیدات و همچنین کاربردهای نظامی مانند مسائلی چون هدف‌یابی و ردیابی اهداف متحرک که باید دارای سرعت و دقت بسیار بالا باشند، دارای ضرورت و اهمیت بسیار زیادی است.

۴-۱- پیشینه

سابقه ایجاد پدیده ردیابی اشیاء به مسائل نظامی بر می‌گردد که این مقوله و جوانب مختلف آن در سال‌های اخیر (عمدتاً از سال ۱۹۸۰ به بعد) مورد توجه ویژه‌ای قرار گرفته است. اما این پژوهش به این شکل دارای پیشینه قبلی نمی‌باشد و نزدیک‌ترین موضوع در رابطه با ردیابی اشیاء روی پلتفرم ZYNQ، در مقاله [۶] مطرح شده است.

۵-۱- اهداف

در اکثر پروسه‌های پردازش تصویر کاهش سرعت لازمه افزایش دقت سیستم بوده است، ولی در این پژوهش سعی بر این شده است که بدون کم کردن دقت، سرعت پردازش حد الامکان افزایش یابد. هدف از این پژوهش تشخیص و ردیابی یک جسم متحرک در یک محیط به صورت بی‌درنگ می‌باشد که کاربرد های آن شامل:

۱- پیدا کردن جسم گم شده با دوربین‌های مدار بسته

۲- ردیابی و تعقیب یک جسم متحرک

۳- تشخیص و دسته بندی اجسام مختلف در محل‌های گوناگون مانند خط تولیدها و...

می‌باشد.

در این پژوهش مواردی همچون تفاوت بین انواع پردازنده‌ها، مقایسه روش‌های پردازش تصویر، بررسی روش‌های افزایش سرعت پردازش، توضیح برخی از الگوریتم‌های ردیابی شیء و بررسی دقیق الگوریتم انتخابی، بررسی نرم‌افزارهای VIVADO، SDSOC، HLS و SDK و توضیح کلی درمورد روند کار پردازنده مطرح خواهد شد.

۱-۶- روش‌های تحقیق

با توجه به اینکه موضوع پردازش تصویر به صورت بی‌درنگ یکی از مباحث بروز علم الکترونیک می‌باشد، نیاز مبرم به تحقیق پایه‌ای و بررسی دقیق منابع دارد. پس روش تحقیق انتخابی برای پیشروی در این پژوهش روش پایه‌ای عملی-تجربی بوده است.

فصل دوم: الگوریتم ردیابی شیء

۲-۱- مقدمه

امروزه پردازش تصویر به یکی از مهم‌ترین علوم دانش رایانه تبدیل شده است که پردازش تصویر در حال حاضر به صورت دیجیتال انجام می‌شود. این تصاویر توسط دوربین دیجیتال و یا تصاویر پویش شده توسط پویشگر^۱ ثبت می‌شود. این علم شامل دو بخش عمده می‌باشد:

۱- بهبود تصاویر: که شامل مواردی همچون فیلترهای دربرگیرنده‌ی محو کننده^۲ و افزایش تضاد برای بهبود کیفیت تصاویر گرفته شده و نمایش صحیح آن‌ها در مقصد (همانند نمایشگر یا چاپگر) می‌باشد.

۱- Scanner

۲- Blur

۲- بینایی رایانه: روش‌هایی است که به کمک آن‌ها می‌توان معنای تصاویر دیجیتال گرفته شده را به سیستم کامپیوتری فهماند تا از آن‌ها در کارهای مورد نظر بتوان استفاده کرد.

۲-۲- پردازش تصویر

پردازش تصویر روشی برای فهماندن محتوای یک تصویر به رایانه و انجام برخی پردازش‌ها و عملیات روی آن می‌باشد که نهایتاً نتیجه آن نمایش یک تصویر به صورت خروجی یا الگوهای کنترلی است. پردازش ویدئو نیز همان پردازش تصویر است به صورتی که با دریافت هر فریم از ویدئو و پردازش و تصمیم‌گیری روی آن خروجی مورد نظر تولید می‌شود.

علت استفاده زیاد از متدهای پردازش تصاویر شامل بیان انسانی تصویر و پردازش داده‌ها برای ارسال، ذخیره و نمایش به منظور درک ماشین می‌باشد.

درک ماشین: روش‌هایی که به کمک آن می‌توان معنای بیان انسانی یک تصویر را به یک ماشین انتقال داد و پردازش مورد نظر را روی آن انجام داد را درک ماشین می‌گویند.

۲-۳- بینایی ماشین

بینایی ماشین یکی از بخش‌های مهم و پر کاربرد هوش مصنوعی است که با ترکیب متدهای^۱ مربوط به پردازش تصاویر و ابزارهای یادگیری ماشینی دستگاه‌ها را به بینایی اشیاء، مناظر و فهم هوشمند خصوصیات گوناگون آنها توانا می‌کند.

۱- Methods

۲-۴- بخش‌های مهم در بینایی در بینایی ماشین

هدف از پردازش تصویر به ۵ گروه تقسیم شده است. آن‌ها عبارتند از:

۱- تجسم: مشاهده اشیاء که قابل رویت نیست.

۲- ترمیم تصمیم: برای ایجاد یک تصویر بهتر.

۳- بازیابی تصویر: با هدف دست‌یابی به تصویر مورد علاقه.

۴- اندازه‌گیری الگوی: بررسی و اشیاء مختلف در یک تصویر.

۵- شناخت تصویر: تشخیص اشیاء در یک تصویر.

دو نوع از روش‌های مورد استفاده برای پردازش تصویر، حالت‌های آنالوگ و دیجیتال می‌باشد. آنالوگ و یا تکنیک‌های بصری از پردازش تصویر را می‌توان برای نسخه‌های سخت مانند چاپ و عکس استفاده کرد و بیشتر تحلیلگران تصویر استفاده می‌کنند. اصول مختلف تفسیر وجود دارد، در حالی که استفاده از این تکنیک‌های بصری خیلی پر زحمت است. پردازش تصویر محدود به منطقه ای یا قسمتی کوچک از علم نمی‌باشد و بخش وسیعی را در بر گرفته است. یکی دیگر از ابزار مهم در پردازش تصویر از طریق تکنیک‌های بصری است. کاربران با اعمال ترکیبی از دانش شخصی و داده‌های موجود در فضاهای مختلف علمی به پردازش تصویر می‌پردازند.

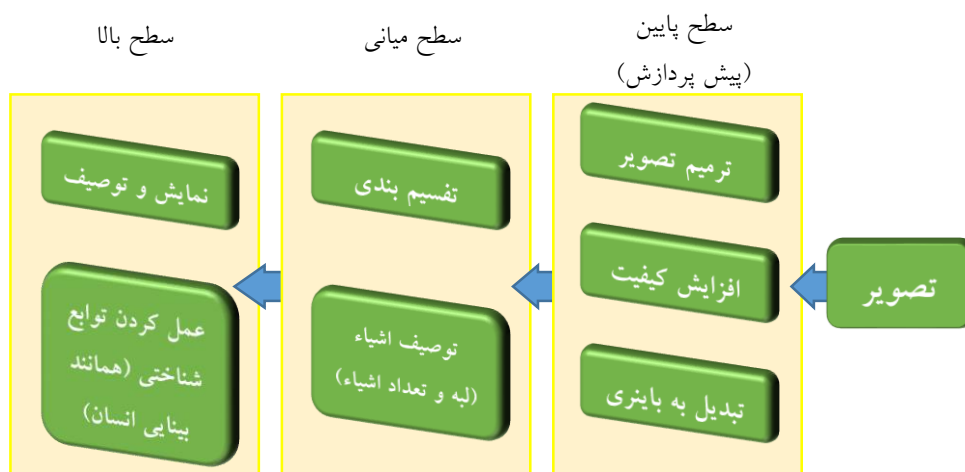
اصولاً پردازش تصویر و بینایی ماشین به یک مفهوم خاص اشاره می‌کند که هدف هر دو یکسان است با این حال می‌توان سه نوع پردازش سطح پایین، سطح میانی و سطح بالا مشخص کرد:

- پردازش سطح پایین شامل پردازش‌های اولیه مثل پردازش‌هایی برای حذف نویز، تصحیح کنتراست و فیلتر کردن تصویر می‌باشد. حاصل این نوع پردازش ورودی و خروجی تصویر است.
- پردازش سطح میانی شامل ناحیه‌بندی تصویر به منظور تقسیم آن به بخش‌ها و اشیاء متخلف، تفسیر اشیاء به گونه ای که برای پردازش رایانه مناسب باشند و طبقه‌بندی و تشخیص اشیاء مختلف است. مشخصه این پردازش این است که ورودی آن معمولاً تصویر و خروجی آن مشخصاتی از اشیاء تصویر مانند لبه‌ها^۱، کانتورها^۲ و تشخیص اشیاء^۳ است.
- پردازش سطح بالا شامل درک روابط میان اشیاء تشخیص داده شده، تفسیر صحنه و انجام پردازش و تشخیص‌هایی است که سیستم بینایی انسان انجام می‌دهد. بسیاری از پردازش‌های سطح بالا در حوزه بینایی ماشین قرار می‌گیرند [۷].

۱- Edges

۲- Counters

۳- Objects Detection



شکل (۱-۲) سطوح پردازش تصویر

با توجه به کاربردهای متفاوت ردیابی اشیاء در حیطه بینایی رایانه، محققان زیادی در طول سالیان گذشته به مطالعه و ارائه روش‌های مختلف برای این کار پرداخته‌اند، معمولاً ردیابی براساس اطلاعات بدست آمده برای برخی ویژگی‌های خاص از اشیاء انجام می‌شود. این ویژگی‌ها شامل: لبه، بافت، رنگ اشیاء، اطلاعات حرکت، نقاط گوشه‌ها و شکل ظاهری اشیاء و غیره می‌باشد. هر روش ردیابی براساس کاربرد می‌تواند از هر کدام از این ویژگی‌ها و یا ترکیبی از این ویژگی‌ها استفاده کند. یکی از مشخصه‌های پرکاربرد برای ردیابی، ویژگی گوشه است که به علت حساسیت کمتر نسبت به تغییرات شدت روشنایی محیط مورد توجه قرار می‌گیرد. مشخصه نقاط گوشه برای ردیابی در روش‌های زیادی استفاده شده است. آشکارسازهای نقاط گوشه در علم ردیابی عبارتند از: Harris, Moravec, KLT و SIFT در [۹و۸] به ارزیابی نقاط ویژگی برای ردیابی پرداخته‌اند. نویسندگان در [۱۰و۱۱] از کم کردن

پس زمینه با بروز رسانی در طول عملیات ردیابی استفاده کرده‌اند. در [۱۲ و ۱۳] از الگوریتم دیگری مبتنی بر خوشه بندی k-means برای ویژگی‌های رنگ و مکان استفاده کردند. ایده اصلی این روش این است که ابتدا ویژگی‌های شی و زمینه نزدیک به شی را خوشه بندی کرده و سپس با حذف خوشه های مشترک، شی را از زمینه اطرافش جدا کرده و ردیابی می‌کند.

در ادامه به بررسی مختصر دو مورد از بهترین الگوریتم‌های شناسایی اشیاء پرداخته خواهد شد:

۲-۵- الگوریتم ردیابی با روش k-means خودکار [۱۴]

۲-۵-۱- دریافت فریم‌های ورودی

از آنجایی که هدف از ردیابی اشیاء تعیین مکان یک شیء به صورت لحظه به لحظه می‌باشد در نتیجه باید تصاویر پی در پی پشت سر هم طبق زمان معین از جسم هدف دریافت شود که به واسطه آن بتوان عملیات پردازش و تشخیص و نهایتاً ردیابی را انجام داد. سیستم‌های مورد نظر برای این کار سیستم‌های مختلفی می‌باشند که به عنوان مثال می‌توان به دوربین‌های حفاظتی نصب شده در یک مکان خاص برای کنترل فرآیند خاص اشاره نمود.

۲-۵-۲- پیش پردازش و استخراج ویژگی

نویز همیشه در تصاویر وجود دارد. زیرا کیفیت یک تصویر تحت تاثیر شرایط زیادی قرار می‌گیرد. به عنوان مثال ابزار فیلم‌برداری، شرایط آب و هوایی و تغییرات شدت روشنایی در طول شبانه روز تصاویر دریافتی را تحت تاثیر قرار داده و متدهای ارائه شده در پردازش تصویر را با مشکل مواجه می‌سازد. بنابراین حذف نویز و یا کاهش آن امری ضروری در پردازش تصویر برای بهتر عمل کردن الگوریتم مورد نظر می‌باشد.

با استفاده از الگوریتم حذف نویز گاوسی، نویز موجود در تصویر ورودی حذف می‌شود، پس از حذف نویز، با استفاده از روش مطرح شده توسط Shi and Tomasi نقاط ویژگی‌های خوب برای ردیابی^۱ را بر روی فریم در لحظه t بدست می‌آید. آنها با استفاده از محاسبه مشتق دوم (با استفاده از عملگر sobel) نقاط ویژگی با مقادیر ویژه خاص را محاسبه کرده و نقاط بدست آمده با این کار را به عنوان نقاط ویژگی خوب برای ردیابی معرفی می‌شود.

۲-۵-۳- تناظریابی با KLT

در این مرحله نقاط ویژگی متناظر بین دو تصویر متوالی را با استفاده از متد KLT که بر پایه سه ویژگی مهم روشنایی^۲، حرکات کوچک^۳ و وابستگی فضایی^۴ می‌باشد، پیدا می‌شود.

بر اساس این سه اصل شدت روشنایی پیکسل‌های مربوط به یک شی در تصاویر متوالی ثابت خواهد بود و پیکسل‌ها در این توالی دارای جابجایی‌های کوچک بوده و همچنین نقاط مربوط به همسایگی یک پیکسل خاص دارای خواص و جابجایی‌های مشابه به همدیگر هستند.

۲-۵-۴- تعیین جهت و اندازه حرکت نقاط ویژگی

از آنجایی که در گام‌های بعدی باید طبق اطلاعات حرکت، اشیاء متحرک را شناسایی و سپس به دسته‌بندی آنها پرداخته شود، لذا در این مرحله اندازه و جهت حرکت هر نقطه محاسبه می‌گردد.

۱- Good Feature For Track

۲- Brightness constancy

۳- persistence or small movements Temporal

۴- Spatial coherence

۲-۵-۵- کشف اشیاء متحرک

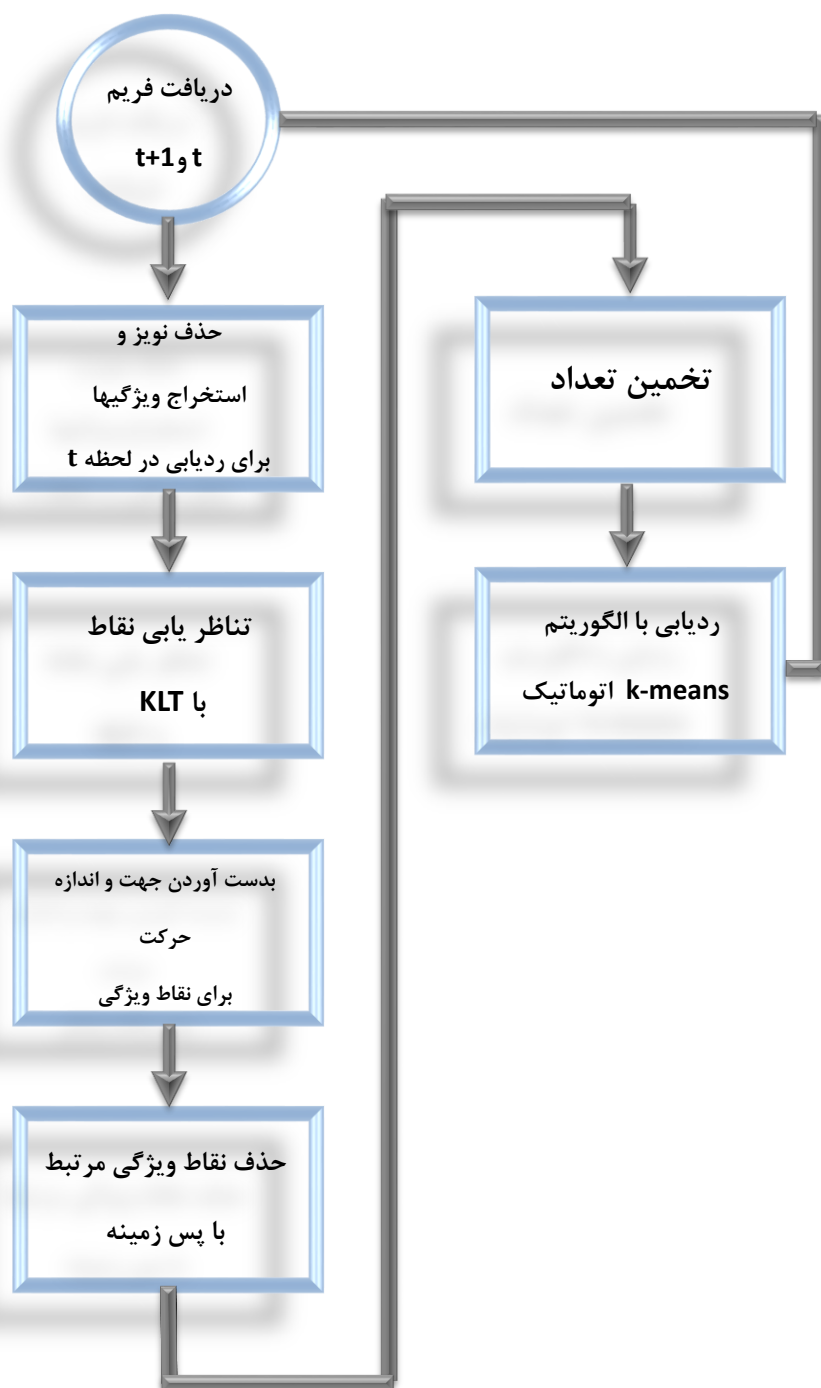
در این مرحله باید اشیاء متحرک موجود در صحنه شناسایی و مشخص شوند. برای این کار با استفاده از اطلاعات حرکت نقاط مورد نظر که دارای جابجایی جزئی یا برابر با صفر دارند، به دلیل اینکه نقاط مربوطه به پس زمینه همواره ثابت بوده و حرکتی ندارند بعنوان نقاط ویژگی مرتبط به پس زمینه در نظر گرفته می شود.

۲-۵-۶- تخمین تعداد اشیاء

در این گام پس از آنکه نقاط مورد نظر مرتبط به اشیاء متحرک مشخص شدند، باید تعداد اشیاء متحرک موجود در صحنه کشف شوند. زیرا در مرحله بعد با استفاده از این تعداد به دسته بندی اشیاء متحرک پرداخته می شود.

۲-۵-۷- ردیابی با روش k -means اتوماتیک

در مرحله ردیابی باید نقاط ویژگی بدست آمده مورد نظر برای اشیاء متحرک به خوشه های مناسب دسته بندی شود بطوری که هر خوشه نشان دهنده یک شی متحرک باشد. پس از عمل خوشه بندی هر خوشه را معادل یک شی در نظر گرفته و میانگین آن را بر اساس بردار ویژگی بعنوان مرکز ثقل شی در نظر گرفته می شود و مستطیلی به گونه ای که تمامی نقاط هر خوشه را در بر بگیرد بعنوان ناحیه شی برای مشخص کردن شیء مربوطه رسم می شود. شکل ۲-۲ فلوچارت مراحل گفته شده می باشد.



شکل (۲-۲) فلوچارت الگوریتم ردیابی با روش k-means اتوماتیک

۶-۲- الگوریتم ردیابی با روش SIFT [۱۵]

۶-۲-۱- ساخت فضای شاخص

پس از دریافت فریم آن را با کرنل گاوسی اسکن کرده و از آن لاپلاس گرفته می‌شود و یک فضای شاخص برای الگوریتم ساخته خواهد شد.

۶-۲-۲- گرفتن تفاوت ها از گوس

با استفاده از گرفتن لاپلاس از تصویر اسکن شده با کرنل گوس و تفریق دو فریم و گرفتن تفاوت‌هایشان، آنها DOG^1 نام گذاری می‌شوند.

۶-۲-۳- تعیین محل اکسترم DoG

با اسکن تصویر هر DoG محل‌های اکسترم آنها تعیین و علامت گذاری می‌شود.

۶-۲-۴- تعیین محل و مشخصات زیر مجموعه پیکسل‌ها در محل

با استفاده از عملیات ریاضی (سری تیلور و گرفتن مشتق از آن) روی نمودار، محل و مشخصات پیکسل‌های فرعی مشخص خواهد شد.

^۱ - Difference of Gaussian

۲-۶-۵- حذف keypoint های ضعیف و لبه ها

یکی از فیلترهای لبه را روی فریم اجرا می شود.

۲-۶-۶- اختصاص جهت به نقاط کلیدی

پس از محاسبه‌ی شیب در فریم، برای هر نقطه کلیدی^۱ یک هیستوگرام ساخته و آن با گوسی ترکیب خواهد شد و سپس برای هر پیک از آن یک نقطه کلیدی جهت ساخته می شود.

۲-۶-۷- ساخت توصیف کننده نقاط کلیدی

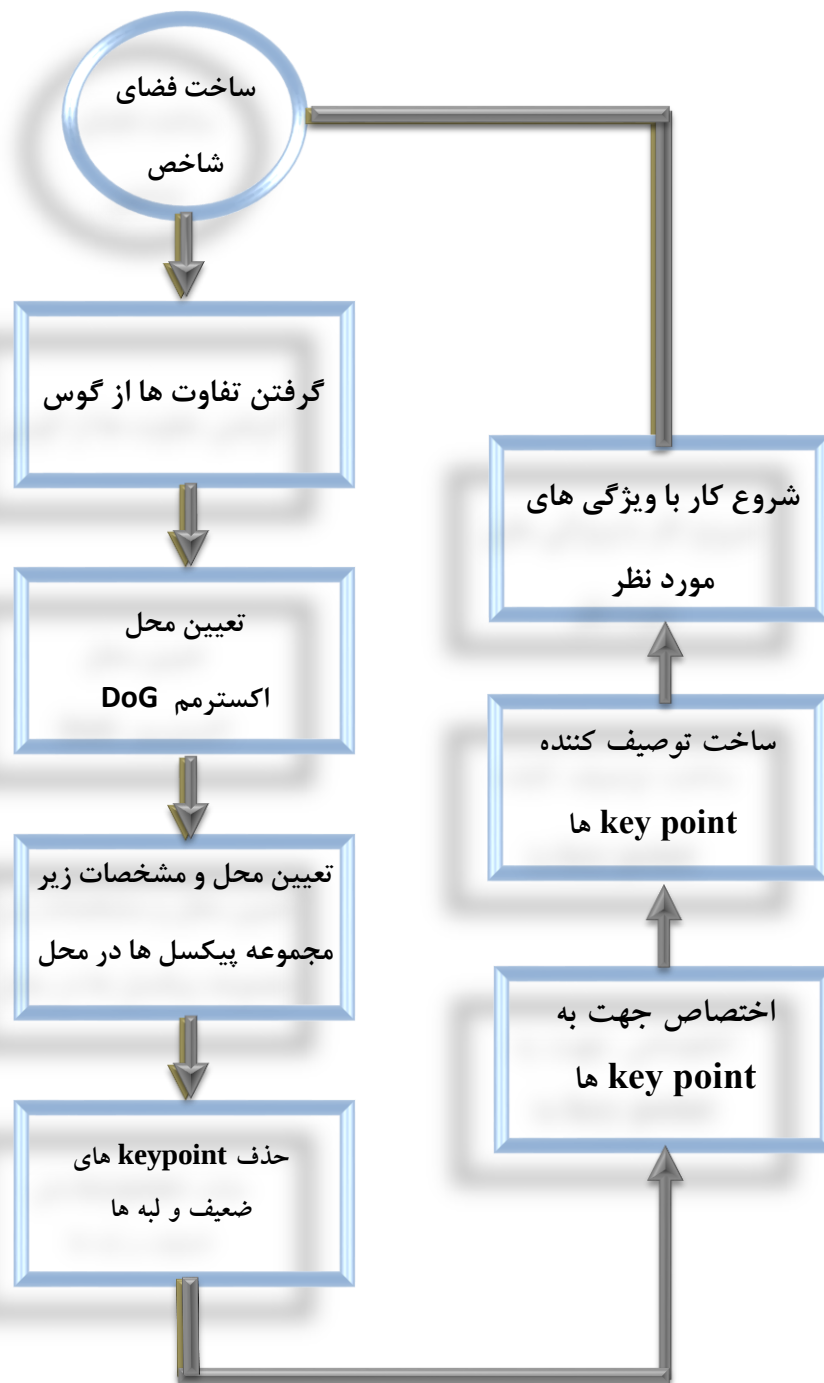
پس از انجام مراحل هر key point جدا شده و در یک جا با جهت قرار می گیرد.

۲-۶-۸- شروع کار با ویژگی های مورد نظر

حال با توجه به الگوریتم مورد نظر و مقایسه ویژگی های شیء مورد نظر با key point های موجود اشیاء پیدا خواهند شد.

تمامی مراحل به شکل فلوچارت در شکل ۲-۳ ذکر شده است.

^۱ - key point



شکل (۲-۳) فلو چارت الگوریتم SIFT

فصل سوم: بررسی تراشه ZYNQ

۳-۱- مقدمه

به طور کلی پلتفرم‌هایی که قرار است روی آنها به نتایج مورد نظر رسید، سیستم‌هایی هستند که نیازمند عملکرد سطح بالا و پشتیبانی کامل از امکانات نسل‌های قبلی می‌باشند. برای مثال پارامترهایی مانند هزینه، توان مصرفی و زمان طراحی و اجرا باید کاهش یابند و همزمان پارامترهایی همانند سرعت اجرا و امنیت افزایش پیدا کند، حتی بایستی به سهولت در نحوه برقراری ارتباط با سخت‌افزار و نرم‌افزار نیز دقت کرد، دو پارامتر مهم در طراحی و ساخت سازه‌های الکترونیکی، انعطاف پذیری و مقیاس پذیری است.

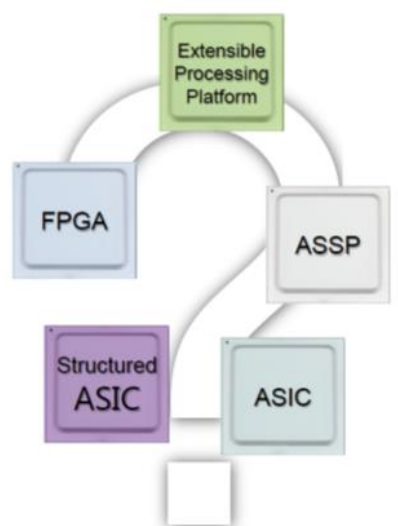
۳-۲- پلتفرم‌های موجود برای انجام پیاده سازی فرآیند [۱۶]

در کل می‌توان تمامی پردازنده‌ها را در پنج گروه زیر قرار داد:

۱-^۱ ASSP : محصول استاندارد با کاربرد ویژه که یک مدار مجتمع می‌باشد که یک عملکرد خاصی دارد و درخواست گسترده‌ای در بازار پیدا می‌کند.

۲-^۲ ASIC : مدار مجتمع با کاربرد ویژه که برای عملکردهای خاص و برای مشتریان خاص تولید می‌شود، یعنی یک سری تراشه سفارشی توسط افراد یا شرکت‌های خاص با پیاده‌سازی منحصر به فرد می‌باشد.

۳-^۳ FPGA: آرایه دريچه‌ای برنامه‌پذیر میدانی یا FPGA یک مدار مجتمع که قابل برنامه نویسی و پیکربندی توسط مشتری یا هر فردی صورت می‌پذیرد.



شکل (۱-۳) انواع روش‌های تولید مدارات مجتمع

۱- Application-Specific Standard Product

۲- Application-Specific Integrated Circuit

۳- Filed Programmable Gate Array

از موارد ذکر شده گزینه اول شامل مدارات مجتمعی می‌باشد که مشتری مشخصی ندارند ولی چون انعطاف‌پذیری مورد نظر را ندارند نمی‌توان زیاد مورد توجه باشند ولی گزینه های دوم و سوم را می‌توان به طور کامل بررسی کرد و به نتایج جالبی دست یافت.

اگر این دو نوع از نظر پیاده سازی و پیکربندی بررسی شود می‌توان به نتایج زیر رسید:

- **FPGA:** کاملاً انعطاف لازم برای طراحی را دارند.
 - **ASIC:** کاملاً کاربردی و قابل ارتقا هستند، اما بعد طراحی دیگر نمی‌توان تغییرشان داد.
- یک تکنولوژی به نام **STRUCTURED ASIC** ما بین **ASIC** و **FPGA** است که عملکرد بالا را که از خصوصیات **ASIC** می‌باشد و خصوصیت قیمت پایین را همانند **FPGA** ها ارائه می‌دهد. با استفاده از این نوع آی سی می‌توان سریع و آسان عمل کرد. اما هدف ساخت یک تراشه نمی‌باشد. با دید موشکافانه به گزینه های موجود نتایج جدول صفحه بعد به دست می‌آید:

جدول (۱-۳) بررسی مدارات مجتمع ASIC و ASSP

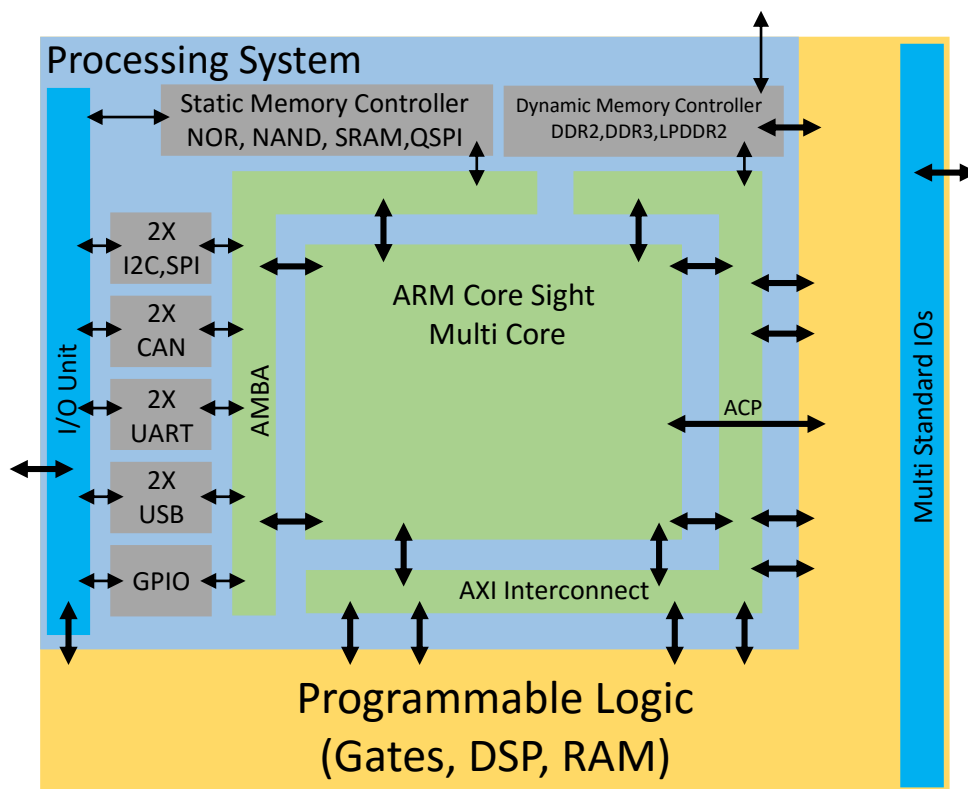
2CHIP SOLUTION (FPGA+ASSP)	ASSP	ASIC	
*	+	+	عملکرد
-	+	+	قدرت
-	+	+	قیمت واحد
+	+	*	هزینه کلی (تولید انبوه)
+	+	-	ریسک توسعه
+	+	-	زمان تولید
+	-	*	انعطاف پذیری ایجاد کردن
+	*	-	مقیاس بندی استفاده از یک پلتفرم برای کارهای مختلف
+: مثبت - : منفی *: بدون تاثیر			

گزینه‌ی Two Chip Solution ترکیبی از دو نسل متفاوت یعنی FPGA و ASSP می‌باشد با اینکه مشکلات تولید حل می‌شود ولی چون ترکیب روی برد صورت می‌پذیرد عملکرد و قدرت خوبی ارائه نمی‌شود. ولی یک نسل بعد که به تازگی به تولید انبوه رسیده است می‌تواند به راحتی مشکلات نام برده شده را حل کند. نام اختصاری این سری Single Chip Solution است که تنها پردازنده این سری ZYNQ All Programmable System On Chip می‌باشد که یک نسل متفاوت و پرکاربرد را روی یک تراشه ارائه می‌دهد. نکته ای که خیلی از افراد به آن دقت نمی‌کنند این ویژگی می‌باشد:

دلیل آنکه نام این تراشه "سیستم با قابلیت تمام برنامه ریزی روی چیپ"^۱ می باشد این است که اولین پلتفرمی می باشد با قابلیت برنامه ریزی نرم افزاری، سخت افزاری و ورودی/خروجی قابل برنامه ریزی را روی یک تراشه ارائه می دهد.

۳-۳- توضیح مختصر درباره ی ZYNQ [۱۷]

ZYNQ تراشه ای با دو نوع پیکربندی می باشد، برنامه ریزی نرم افزاری در پردازنده ARM و برنامه ریزی سخت افزاری یک FPGA یک شتاب دهنده سخت افزاری با ترکیب CPU، DSP، ASSP و عملکردی سیگنال های ترکیبی را ارائه می دهد. در شکل ۳-۲ بلوک دیاگرام این تراشه مشاهده می شود:



شکل (۳-۲) بلوک دیاگرام ZYNQ

۱- All Programmable System on Chip

همان گونه که از بلوک دیاگرام ارائه شده مشخص است، کل تراشه از دو قسمت عمده تشکیل شده است:

• PS^1 : سیستم پردازشگر که شامل قسمت ترتیبی و هسته ARM می باشد.

• PL^2 : سخت افزار قابل پیکربندی که همان FPGA با شرایط ویژه می باشد.

۳-۳-۱- واحد پردازش کاربردی (PS)

این قسمت از تراشه با همکاری شرکت ARM طراحی شده است ولی باید دقت کرد که همان تراشه موجود روی دیگر پلتفرم ها مثلاً گوشی ها، تبلت ها نمی باشد بلکه هسته بسیار بهینه سازی شده است و به راحتی اطلاعات روی پردازنده، هم از طریق PL و هم از طریق کاربر در دسترس می باشد. پردازنده سیستم دارای امکانات زیادی می باشد و بیشتر پروتکل های ارتباطی را پشتیبانی می کند که موارد زیر جزء پروتکل های پشتیبانی شده توسط پردازنده می باشد:

• USB 2.0

• ETHERNET

• CAN 2.0B

• پورت^۳ دوبلکس^۴ SPI

• پورت UARTs (با بیشترین سرعت ۱ Mb/s)

۱- Processing System

۲- Programmable Logic

۳- Port

۴- Duplex

• کنترلر^۱ SD و MMC

• JTAGE

• I2C

• GigE

• پورت ورودی و خروجی HDMI

۳-۱-۱- هسته‌های پردازنده

جنس هسته‌های به کار گرفته شده تقریباً در تمامی سری ZYNQ تفاوت چندانی ندارند که دو

هسته Cortex-A9 از ورژن هفت ARM می‌باشد و خصوصیات زیر را به همراه دارند:

• امکان راه اندازی متقارن یا نامتقارن هسته‌ها (راه اندازی یک هسته به تنهایی نیز امکان پذیر

می‌باشد).

• پشتیبانی از شتاب دهنده^۳‌های:

۱- Java

۲- Jazelle DBX

۳- Jazelle RCT

۴- Thumb-2

۱- Controller

۲- Core

۳- Accelerator

• ۵۱۲۰ کیلو بایت کش^۱ قابل اشتراک گذاری با بیت توازن

• تایمر^۲های اختصاصی و تایمر

۳-۱-۲-۲- رابط کاربری حافظه

رابط کاربری حافظه شامل تکنولوژی‌های حافظه گوناگون می‌باشد:

• کنترل‌کننده DDR:

۱- پشتیبانی انواع حافظه DDR3، DDR3L، DDR2 و LPDDR-2

(مرتب شده بر اساس درجه بندی بر مبنای سرعت و دمای دستگاه)

• قابلیت استفاده از حدوداً ۷۳ پین اختصاصی

• کاهش مصرف DDR به صورت اتوماتیک و تشخیص خودکار خروج از دوره‌های^۳ بیکار

برنامه‌ریزی شده

۳-۱-۳-۳- ورودی ها و خروجی ها

واسط‌های ورودی و خروجی، مجموعه‌ای از رابط‌های کاربری صنعتی استاندارد برای تبادل

اطلاعات با بیرون از پردازنده می‌باشد:

۱- ورودی و خروجی های GPIO

۱- Cache

۲- Timer

۳- Period

۲- کنترل کننده Gigabit Ethernet (دو عدد)

۳- کنترل کننده USB (دو عدد: ۱- به عنوان Host 2- به عنوان OTG)

۴- کنترل کننده SD/SDIO (دو عدد)

۵- کنترل کننده SPI (Master یا Slave)

۶- کنترل کننده CAN (دو عدد)

۷- کنترل کننده UART (دو عدد)

۸- کنترل کننده I2C (دو عدد)

۳-۳-۲- واحد برنامه ریزی منطقی (PL)

PL واحدی می باشد که یک معماری غنی با ظرفیت پیکربندی کاربری ارائه می دهد. به عبارت

دیگر کاربر می تواند با استفاده از ابزار اولیه همانند BRAM، LUT و غیره، سخت افزار مورد نظر خود

را پیاده سازی کند. این ابزار شامل موارد ذکر شده در ادامه می باشد:

• CLB^۱: بلوک های منطقی قابل پیکربندی

۱- دارای LUT^۲ هایی با شش ورودی

۲- دارای حافظه داخل LUT ها

۱- Configurable Logic Block

۲- Look Up Table

۳- عملگر های شیفت رجیستر^۱ و رجیستر

۴- جمع کننده های آبشاری^۲

• BRam: بلوک حافظه با حجم ۳۲ کیلو بایت

۱- دارای دو پورت

۲- امکان ارتقاء به پهنای ۷۲ بیت

۳- دارای مدارات تشخیص خطا^۳

• DSP^۴: پردازشگر سیگنال دیجیتال

۱- پردازنده سیگنال با دقت بالا به عنوان انباشته گر^۵ یا ضرب کننده با مساحت ۲۵ در ۱۸

۲- پیش جمع گر ۲۵ بیت با قابلیت ذخیره توان مصرفی برای بهینه سازی کاربردهای فیلتر متقارن

۳- امکانات پیشرفته شامل: حالت Pipeline انتخابی و ALU^۶ انتخابی

• مدیریت سیگنال پالس ساعت

۱- بافرهای^۷ سرعت بالا

۱- Shift Register

۲- Cascade Adder

۳- Debugger

۴- Digital Signal Processor

۵- Accumulator

۶- Arithmetic Logic Unit

۷- Buffer

۲- ترکیب فرکانسی و شیفیت پالسی

• پیکربندی ورودی ها و خروجی ها (پین ها)

۱- تکنولوژی انتخاب ورودی و خروجی با عملکرد عالی

۲- رنج وسیع در ورودی (۱,۲ ولت تا ۳,۳ ولت)

۳- پشتیبانی ورودی و خروجی با عملکرد بالا

۴- توانایی ارسال داده تا سرعت ۱۲,۵ گیگا بیت در ثانیه (در نسل های ارتقا یافته)

• مبدل آنالوگ به دیجیتال

• بلوک های رابط کاربری پیچیده (PCI Express)

۳-۳-۳- نحوه ارتباطات PS و PL [۱۸]

تراشه های ZYNQ-7000AP از چندین تکنولوژی ارتباط استفاده می کنند، که برای ارتباطات

خاص مورد نیاز بلوک های عملکردی بهینه سازی شده اند.

۳-۳-۳-۱- توضیحاتی درباره ی AXI

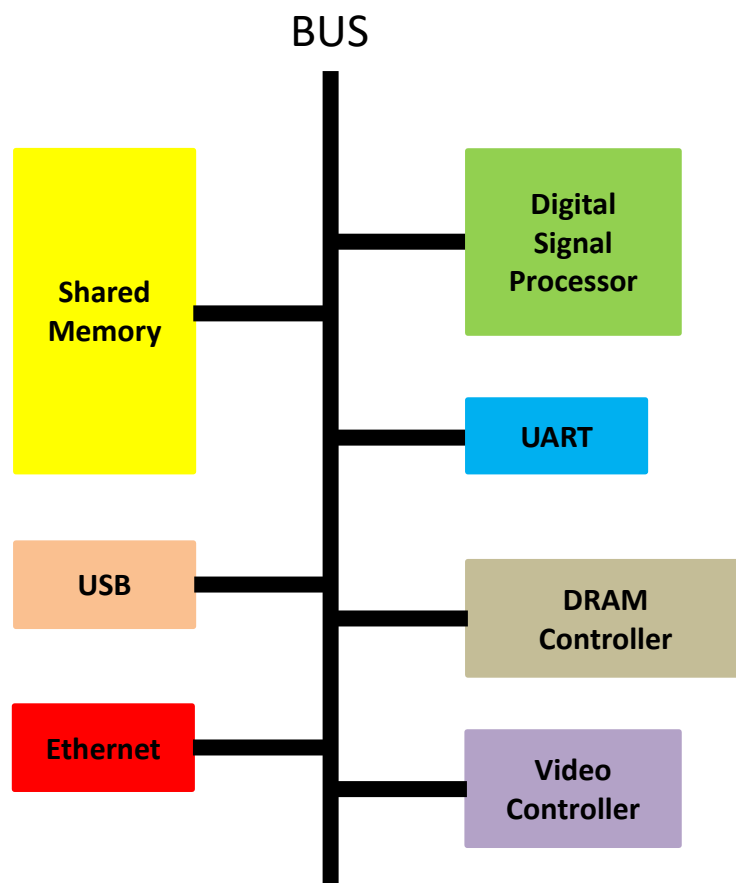
AXI یک گذرگاه^۱ ارتباطی بین اجزای تراشه ZYNQ می باشد که به منظور انتقال اطلاعات مورد

استفاده قرار می گیرد، مثل ارتباط CPU با RAM، USB، CAN و دیگر قسمت های تراشه می باشد.

۱- BUS

۳-۳-۲- ویژگی‌های AXI

- تمامی واحدها به یک زبان مشترک ارتباط برقرار می‌کنند.
- تمامی واحدها می‌توانند به راحتی با هم ارتباط برقرار کنند.
- ساخت، آپدیت، طراحی را آسان می‌کند.
- در طراحی‌های دیگر می‌توان از مدل طراحی شده استفاده نمود و به راحتی امکان اتصال به دیگر طرح‌ها را داراست.



شکل (۳-۳) نحوه اتصال AXI

۳-۳-۳-۳ تبادل اطلاعات در AXI

به انتقال اطلاعات از یک نقطه‌ی سخت افزاری به نقطه‌ی دیگر گفته می‌شود که در کل به دو

المان نیازمند می‌باشد:

• Axi-Master: آغاز کننده ارتباط می‌باشد.



شکل (۴-۳) سمبل ارتباطی مستر^۱ در AXI

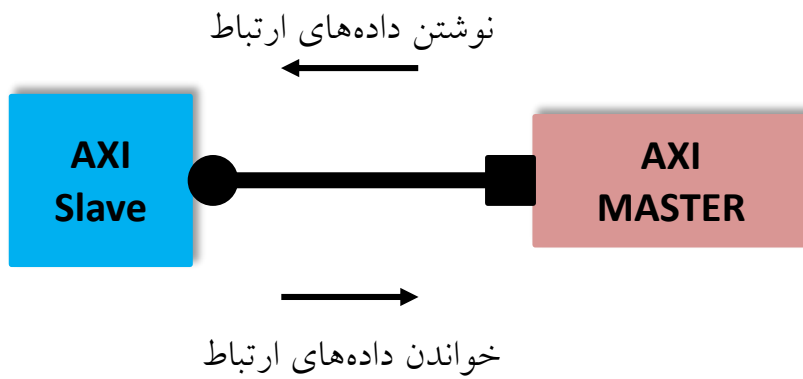
• Axi-Slave: جواب دهنده به ارتباط می‌باشد.



شکل (۵-۳) سمبل ارتباطی اسلیو^۲ در AXI

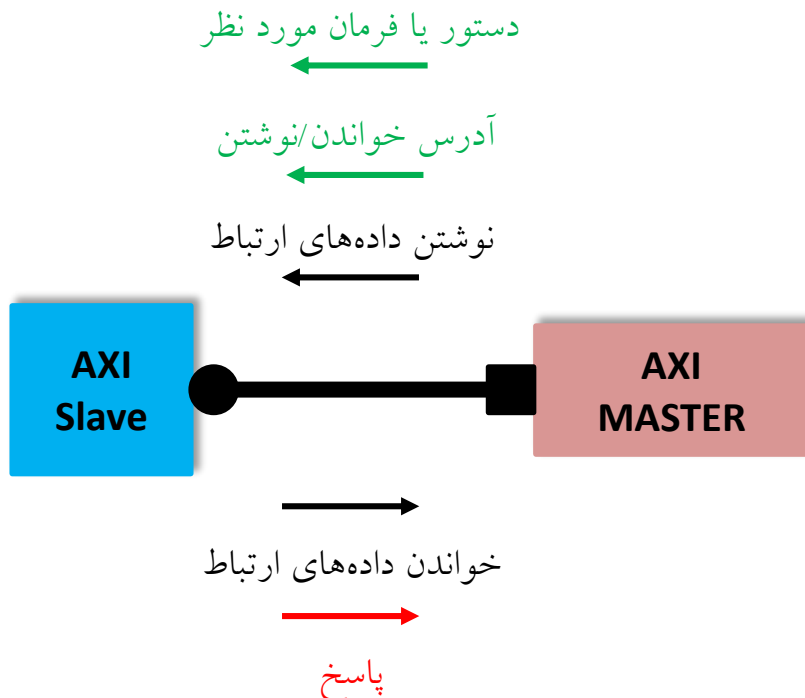
۱- Master

۲- Slave



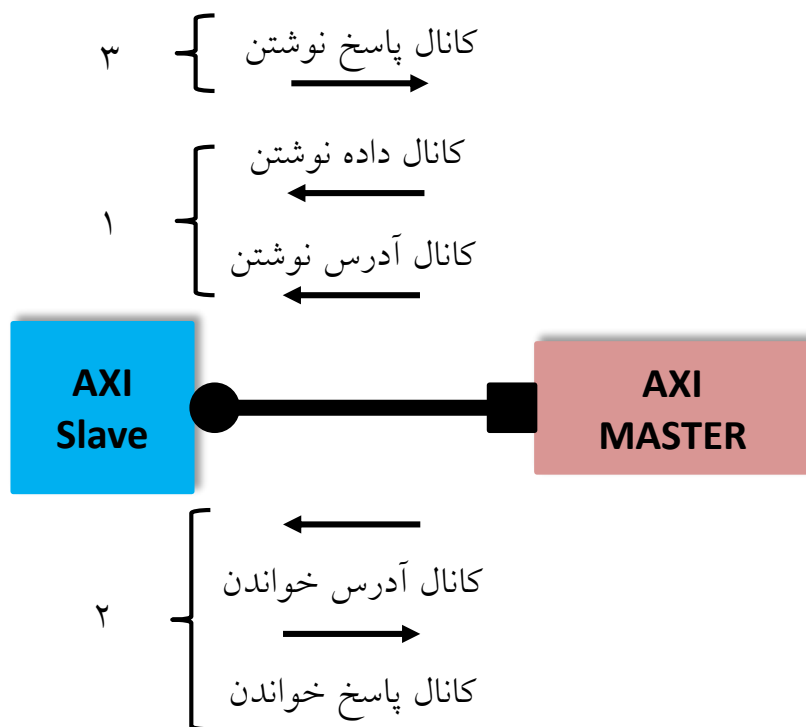
شکل (۶-۳) ارتباط ساده AXI

برای برقراری ارتباط ابتدا مستر، اطلاعاتی شامل خواندن یا نوشتن ارسال کرده و واجب است که مستر، آدرسی نیز برای خواندن یا نوشتن ارسال کند، پس از تایید کردن اسلیو پیام تایید را ارسال خواهد کرد و بدین ترتیب عمل تبادل بین دو قسمت توسط AXI صورت می‌پذیرد.



شکل (۷-۳) ارتباط کامل در AXI

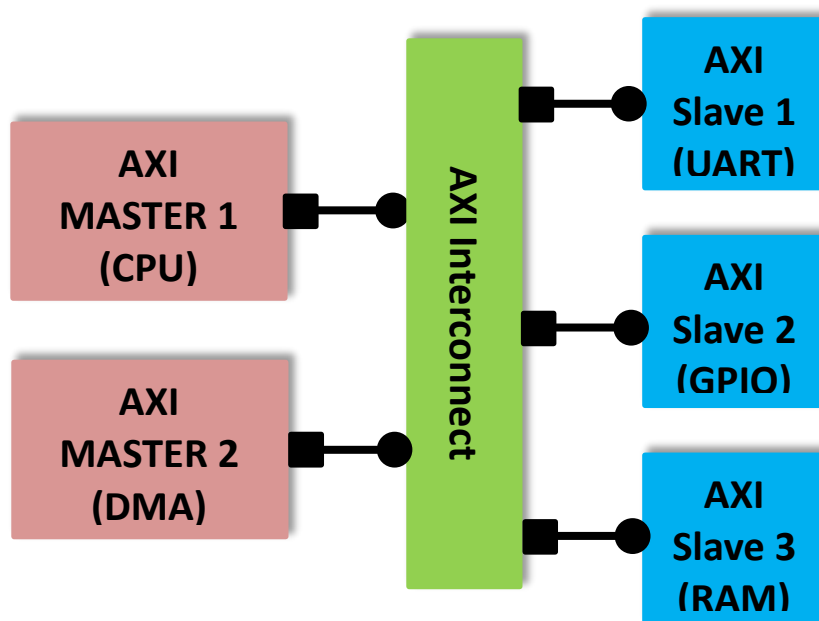
واسط AXI شامل پنج کانال ارتباطی است که ترتیب اجرا شدن هر یک از کانال ها با عدد در شکل مشخص شده است.



شکل ۳-۸) کانال‌های مختلف در AXI

در کل نوع ارتباط AXI از نظر تعداد جز در ارتباط متفاوت می‌باشد که شامل دو نوع AXI Interface و AXI Interconnect می‌باشد. AXI Interface به حالتی گفته می‌شود که فقط یک قطعه

مستر با یک قطعه اسلیو در ارتباط باشد ولی اگر تعداد هر یک از این دو قطعه (مستر یا اسلیو) بیشتر از یک باشد حالت AXI Interconnect به وجود می‌آید.



شکل (۹-۳) ارتباط AXI Interconnect

• در تعریف address ها در اسلیو هیچ گونه روی هم افتادن^۱ وجود نداشته باشد و کاملاً از هم جدا باشند.

• آدرس برای رنج 2G باید به صورت 0xC0000000 باشد و نمی تواند پشت آدرس های دیگر باشد. به عبارت دیگر باید باند داشته باشد.

تمامی AXI های ذکر شده همه از نوع memory mapped بودند. این AXI ها شامل پنج کانال بودند و آدرس می فرستادند. اما در بسیاری از موارد که آدرس مشخص است و هدف فقط انتقال داده می باشد، احتیاجی به وجود پنج کانال نیست. در این موارد از AXI STREAM استفاده می کنیم AXI STREAM فقط یک کانال ارتباطی دارد و بسیار ساده است.

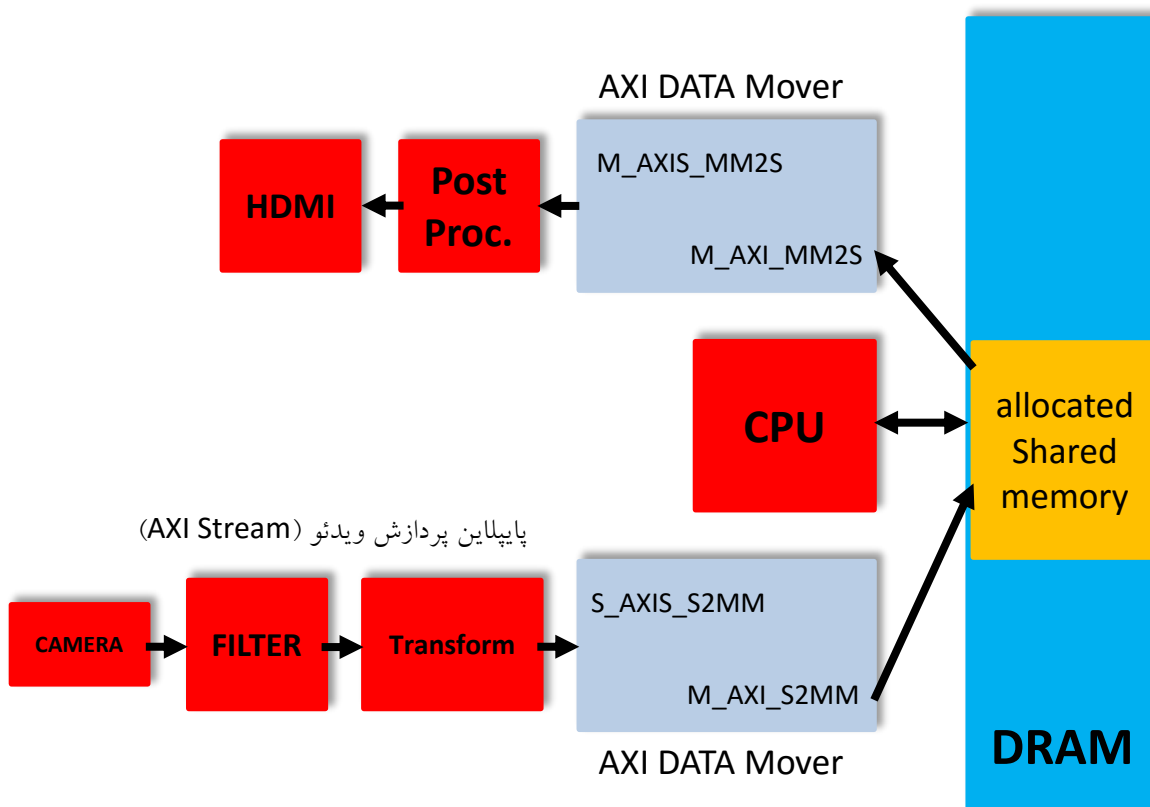
^۱ - Over Flow

پردازش ویدئو



شکل (۱۰-۳) مثال ساده AXI Stream

به طور مثال از دوربین تصویری گرفته می شود و یک فیلتر اعمال شود، تا اینجا تمام بلوک ها به صورت stream به هم وصل شده اند حال تصویر در حافظه ی DRAM ذخیره می شود و سپس از روی حافظه خوانده شده و بعد توسط پروسس روی HDMI نمایش داده می شود. همانطور که قبلاً اشاره شد برای ذخیره در رم احتیاج به آدرس می باشد و کانال های stream شامل آدرس نمی شوند پس اینجا احتیاج به دو بلاک برای تبدیل AXI MM به AXI S و بلعکس است که آن Data Mover نام دارد.



شکل (۱۱-۳) مثال ارتباط AXI MM و AXI Stream و بلعکس

ZYNQ همانند دیگر سیستم های مبتنی بر پردازشگری بوت^۱ شده و رفتار می کند. پردازنده ابتدا بوت می شود و می توان از قسمت FPGA نیز استفاده کرد. یعنی این اجازه را می دهد توسعه دهندگان برنامه ابتدا روی پردازنده برنامه مورد نظر را اجرا کرده و از FPGA نیز کمک بگیرند.

^۱ - Boot



شکل (۳-۱۲) خصوصیات ZYNQ

۳-۳-۳-۵- قطعات سیلیکونی : پلتفرم پردازی مهم (اصلی)

- هزینه کلی کم و عملکرد سطح بالای سیستم

- راه حل انعطاف پذیر و مقیاس بندی شده

۳-۳-۳-۶- پلتفرم های توسعه یافته : پلتفرم های توسعه یافته و ارزشیابی

- گیت های توسعه یافته جامع

- بردهای مبتنی بر ارزیابی کل

- پلتفرم مجازی

۳-۳-۷- ابزارهای توسعه یافته : محیط طراحی استاندارد صنعتی

- مدل برنامه نویسی نرم افزاری که به خوبی تعریف شده است

- فلوهای^۱ سخت افزاری و نرم افزاری کاملاً شبیه

- ابزارهای معماری سیستم

۳-۳-۸- سیستم‌های عامل : پیشنهاد نرم افزار جامع

- سیستم عامل‌های بسیار معروف

- بیشتر راه‌حل‌های میان افزاری

۳-۳-۹- لوازم جانبی و شتاب‌دهنده ها : IP و شتاب‌دهنده‌های قابل انعطاف

- برنامه ریزی منطقی بهینه شده در مقیاس ۲۸ نانو متری در کلاس جهانی

- واسط AMBA AXI استاندارد

۳-۳-۱۰- اکو سیستم^۲ : اکو سیستم جامع

- ابزار، سیستم‌های عامل و میان افزارها

- بلوک‌های IP برای برنامه‌ریزی منطقی

۱- Flows

۲- Eco System

-آموزش و سرویس‌های طراحی [۱۶]

۳-۳-۴- خصوصیات مهم ZYNQ

- عملکرد بالا و مصرف کم همانند ASICها

- قابلیت انعطاف همانند FPGA ها

- برنامه ریزی راحت همانند میکروپروسورها

۳-۴- مزیت‌های ZYNQ نسبت به FGPA

- پردازنده برای برنامه‌نویسی تمام پلتفرم‌ها (PL,DSP,IOs,AMS) و بهبود امنیت در یک راه

حل یکپارچه

- افزایش عملکرد سیستم: ARM با فرکانس یک گیگاهرتز و قسمت FPGA در ZYNQ تقریباً

ده برابر سریع تر از حالت TWO CHIP^۱ عمل می‌کند، که قابل مقایسه با FPGA به تنهایی نمی‌باشد.

- کاهش هزینه ساخت تا حدود ۴۰ درصد

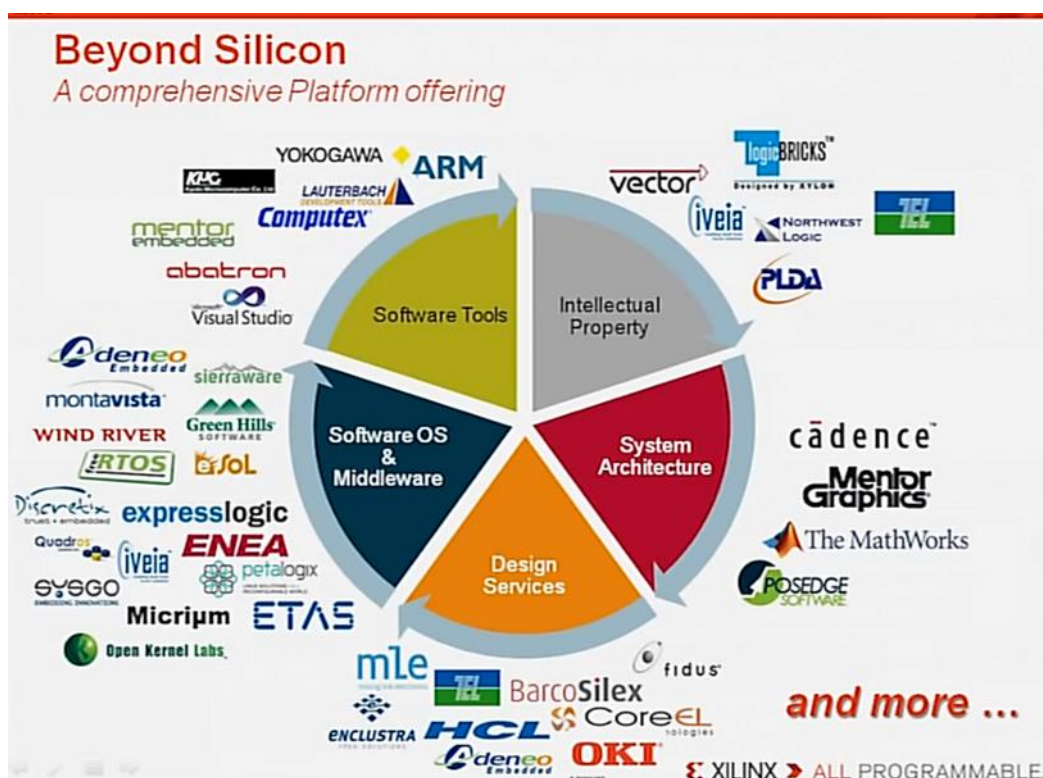
- کاهش ۵۰ درصدی مصرف توان (زیرا قطعات جانبی FPGA به تنهایی زیاد است).

- طراحی انعطاف‌پذیر و بهبود یافته: زیرا هم می‌توان از OS ها و هم IP ها استفاده کرد.

۱- دو پردازنده روی یک برد

۳-۵- ارتباط با دیگر شرکت ها

شرکت Xilinx در سری ZYNQ سعی بر این داشته است تا ارتباط نزدیک و تنگاتنگی با شرکت های تولیدکننده نرم افزار و سخت افزار داشته باشد به نحوی که طبق گزارش تهیه شده توسط خود شرکت در ماه های آخر سال ۲۰۱۵ میلادی شرکت Xilinx با بیش از ۴۵ شرکت نرم افزاری و سخت افزاری در زمینه تراشه های ZYNQ همکاری داشته است. در شکل ۳-۱۳ مشاهده می شود که این شرکت برای ارائه خدمات بهتر حاضر با همکاری شرکت های زیادی شده است [۱۹].



شکل (۳-۱۳) دایره ارتباطی Xilinx با دیگر نرم افزارها

۳-۶- بررسی نهایی ZYNQ

در بعضی کاربردها نیاز به پردازش با سرعت خیلی بالا، امنیت بالا، مصرف کم، حجم کم و قابلیت بروزرسانی دستگاه می‌باشد. گزینه‌های بسیاری پیش رو است ولی هیچ کدام نمی‌تواند به کاملاً پردازنده ZYNQ باشد به عنوان مثال اگر از یک CPU و مادربرد استفاده شود که بتوان در محیط سیستم عامل برنامه‌ریزی شود مشکلاتی مثل امنیت کم، مصرف بالا، حجم بزرگ، سرعت پایین و ... پیش رو می‌باشد. یا استفاده از بردهای ریسپری پای^۱ هم موارد سرعت و امنیت را نقض می‌کنند. همچنین استفاده از FPGA هایی مثل SPARTAN6 با وجود داشتن سرعت، امنیت، حجم کم و ... قابلیت اجرای هر کدی را نداشته و برنامه‌نویسی برخی از وسایل به دلیل برنامه‌نویسی سخت افزاری بسیار طولانی یا در بعضی موارد غیرممکن می‌باشد همچنین به دلیل پشتیبانی نکردن بیشتر پروتکل‌های ورودی/خروجی، نیاز زیادی به مدارات اضافی دست ساز دارد، که باعث بیشتر شدن حجم مدار و حجم کد، مصرف بیشتر و پایین آوردن امنیت می‌شود. پس وسیله‌ای لازم است که در عین سرعت بالا، امنیت زیاد و حجم کم مصرف پایینی داشته باشد و قابلیت مانور نرم افزاری زیادی داشته باشد تا بتوان وسایل پیشرفته‌تر و کارآمدتری ارائه داد.

تراشه ZYNQ تمامی خواسته‌های یک طراح و برنامه نویس را برآورده می‌کند همچنین طراح می‌تواند برنامه خود را روی سیستم عامل خود توسعه داده و با اعمال تغییراتی جزئی آن را روی ZYNQ با سرعتی خیلی بالا اجرا نماید، همچنین داشتن امنیت بسیار بالا (داخل تراشه قسمتی برای بالا بردن امنیت تعبیه شده) حجم کم، قدرت بالا در مقابل مصرف کم، توسعه‌ی آسان و رسیدن به

۱- Raspberry Pi

برنامه‌های پیشرفته‌تر و سریع‌تر در کنار دسترسی به تمامی پروتکل‌های ارتباطی برای ارتباط با انواع وسایل ورودی و خروجی مثل انواع دوربین ، حافظها ، مانیتورها و..... می‌تواند خواسته هر فردی را به طور کامل و قطعی برآورده نماید.

جدول (۲-۳) جدول مقایسه ZYNQ با مدارات مجتمع ASSP، ASIC و 2CHIP Solution

ZYNQ-7000	2 CHIP SOLUTION (FPGA+ASSP)	ASSP	ASIC	
+	*	+	+	عملکرد
+	-	+	+	قدرت
*	-	+	+	قیمت واحد
+	+	+	*	هزینه کلی (تولید انبوه) - برای ما مهم نیست
+	+	+	-	ریسک توسعه
+	+	+	-	زمان تولید- برای ما مهم نیست
+	+	-	*	انعطاف پذیری ایجاد کردن
+	+	*	-	مقیاس بندی استفاده از یک پلتفرم برای کارهای مختلف
+: مثبت -: منفی *: بدون تاثیر				

۷-۳- بررسی دقیق تفاوت بین ZYNQ و SPARTAN 6

به علت اینکه در کشور ایران نزدیک‌ترین تراشه به تراشه مورد نظر (ZYNQ) چیپست^۱ FPGA سری SPARTAN ورژن ۶ می‌باشد خوب است این دو تراشه از نزدیک مورد بررسی قرار گیرند.

در کل این دو پلتفرم قابل مقایسه نیستند زیرا، زیر ساخت های متفاوتی دارند. SPARTAN 6 یک سازه منطقی دارد که برای پیاده سازی سخت‌افزاری برنامه مورد نظر طراحی شده است و بیشتر برای ساخت نسخه اولیه یک آی سی استفاده می‌شود.

در حالی که ZYNQ علاوه بر پیاده‌سازی سخت‌افزاری، قابلیت پیاده‌سازی نرم‌افزاری و برنامه‌ریزی I/O ها وجود دارد. یعنی یک برنامه نوشته شده در زبان های C/C++ را می‌توان بدون کاهش سرعت (در مقایسه با SPARTAN 6) اجرا کرد. یعنی بدون هیچ دردسری این مهم صورت می‌پذیرد.

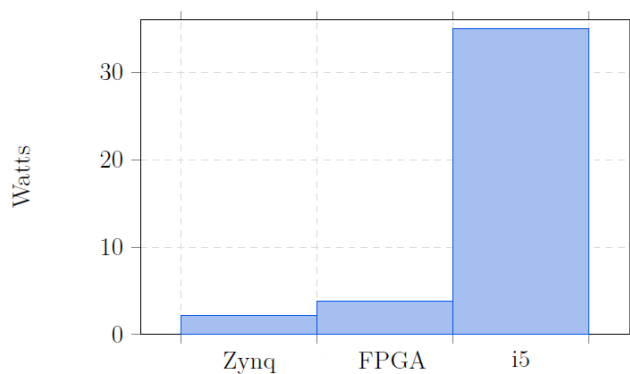
در زیر این دو تراشه در موارد مهم بررسی شده اند:

۷-۳-۱- بررسی توان مصرفی

به علت فاصله زمانی که بین تولید این دو تراشه وجود دارد، بهینه سازی های زیادی برای مصرف کم توان اعمال شده است. البته باید در نظر گرفت FPGA استفاده شده در ZYNQ از نوع VIRTEX است و مصرف خیلی پایینی در مقایسه با SPARTAN 6 دارد. علت اصلی هم تکنولوژی ساخت ۲۸ نانو متری در ZYNQ می باشد. در حالی که Spartan 6 دارای ساخت ۴۵ نانو متری است.

^۱- Chipset

نمودار صفحه بعد در [۲۰] ارائه شده است که میزان مصرف توان را بررسی کرده است:



نمودار (۱-۳) بررسی توان مصرفی پلتفرم‌های مختلف

۳-۷-۲- سرعت انتقالی پایه های I/O

طبق [۲۱] ارائه شده توسط شرکت XILINX، سه نسل متفاوت این شرکت سرعت های زیر

را فراهم می آورند:



شکل (۱۴-۳) مقایسه سرعت نسل های مختلف FPGA [۲۱]

۳-۷-۳- امکان استفاده از سیستم عامل‌های مختلف و پر کاربرد

بر خلاف SPARTAN 6 که اصلاً امکان پیاده سازی سیستم عامل وجود ندارد تا پردازش نرم‌افزاری انجام دهد، ZYNQ به طور شرکتی می‌تواند بیشتر سیستم عامل‌ها را اجرا کند و از طریق این سیستم عامل‌ها پردازش نرم‌افزاری داشته باشد.

۳-۷-۳-۱- سیستم عامل‌های رایگان:

۳-۷-۳-۱-۱- سیستم کد باز لینوکس

- پشتیبانی شده توسط SDK زایلینکس

-دانلود رایگان



شکل (۳-۱۵) لوگو لینوکس

۳-۷-۳-۲- اندروید کد باز

-اندروید ۲,۳ که از IP نمایش دهنده OpenGL ES 1.1 استفاده می‌کند.

-دانلود رایگان



شکل (۳-۱۶) لوگو اندروید

رایگان RTOS ۳-۷-۳-۱-۳-

- سیستم عامل آنی

- برنامه AMP ترکیب شده با پشتیبانی توسط لینوکس



شکل (۳-۱۷) لوگو free RTOS

سیستم عامل های تجاری: ۳-۷-۳-۱-۴-

- Windows Embedded Compact 7

- Linux (distribution)

- Android (distribution)

- VxWorks

- ThreadX

- ulTRON/T-Kernel

- Quadros

- RTA-OS

uC/OS –

OSE –

۳-۷-۳-۲ - راحتی برنامه نویسی

با توجه به اجرای سیستم عامل‌های مختلف روی این دستگاه امکان استفاده از کتابخانه‌های مختلف و برنامه‌نویسی با زبان‌های مختلف فراهم می‌شود که بدین وسیله می‌توان به راحتی برنامه را توسعه داد و در هر ورژن آن را کامل‌تر کرد همچنین می‌توان برنامه‌های بسیار پیشرفته‌ای نوشت و از تمامی امکانات دستگاه استفاده بهینه نمود.

۳-۷-۳-۳ - قابلیت ارتباطی زیاد

با توجه به این که ZYNQ دارای انواع پروتکل‌های ارتباطی در داخل خود است ، مزیت‌هایی مثل کم شدن حجم مدار ، کم شدن هزینه‌های مدار ، ساده شدن برنامه نویسی ، امکان استفاده از ماژول‌های مختلف ، کم شدن مصرف مدار و بالا رفتن امنیت و را ایجاد می‌کند.

ترکیب شدن قدرتمندترین نسل پروسسورهای FPGA در کنار پرسرعت‌ترین میکروکنترلر ARM و کار کردن هماهنگ آنها در کنار پروتکل‌های ارتباطی و رم^۱ و قسمت‌های محافظتی برای کنترل امنیت با سرعت بالا و مصرف کم تنها در تراشه ZYNQ در دسترس هست و می‌تواند به راحتی تمامی انتظارات را از یک پردازنده برآورده کند.

۱- RAM

۳-۸- بررسی نرم افزارهای مورد استفاده برای برنامه ریزی ZYNQ

۳-۸-۱- نرم افزار VIVADO

نرم افزار Vivado در آپریل سال ۲۰۱۲ معرفی شد، هر سال چهار بروزرسانی برای این نرم افزار روی سیستم عامل های Windows و Linux ارائه می شود و جدید ترین نسخه آن ۲۰۱۶,۲ می باشد که در جوئن سال ۲۰۱۶ ارائه شده که این نسخه را با نام Vivado Design Suite HLx Editions معرفی نمود است. این نرم افزار دارای محیط طراحی یکپارچه (IDE^۱) و شامل سیستم یکنواخت الکترونیکی (ESL^۲) و امکانات سنتز و بررسی IP^۳ ها با الگوریتم های مختلف بر پایه C است و در بسته بندی نهایی دارای دو استاندارد الگوریتمی و RTL^۴ برای استفاده دوباره می باشد [۲۲].

شرکت Xilinx نرم افزار Vivado Design Suite را برای طراحی های جدید با Ultrascale, Virtex-7, Kintex-7, Artix-7, and Zynq-7000 معرفی و روانه بازار کرد.

نرم افزار Vivado از قطعات جدید Xilinx با ظرفیت های بالا و طراحی پرسرعت برنامه ریزی منطقی و I/O ها پشتیبانی می کند [۲۳].

هدف شرکت Xilinx از ارائه نرم افزار Vivado طراحی FPGA ها با فضای بزرگتر بود و این نرم افزار با برنامه ی Xilinx ISE همانند یک زنجیره به یکدیگر وصل هستند و برای طراحی CPLD ها

۱- Integrated Development Environment

۲- Electronic System Level

۳- Intellectual Property

۴- Resistor Transistor Logic

و FPGA های Xilinx با ظرفیت کوچک از نرم افزار ISE استفاده خواهد شد و برای طراحی FPGA ها با فضای بزرگتر (نسل ۶ به بعد) از نرم افزار Vivado استفاده می شود.

۳-۸-۱-۱- امکانات

- برنامه نویسی VHDL و Verilog و ساخت ip مربوطه

- وارد کردن ip از نرم افزار های Matlab و HLS

- طراحی و اتصال Ip ها و بلاک ها به یکدیگر

- شبیه سازی

- تصحیح کننده خطا

- تحلیلگر RTL

- پیاده سازی

- سنتز^۱

- ساخت bit stream

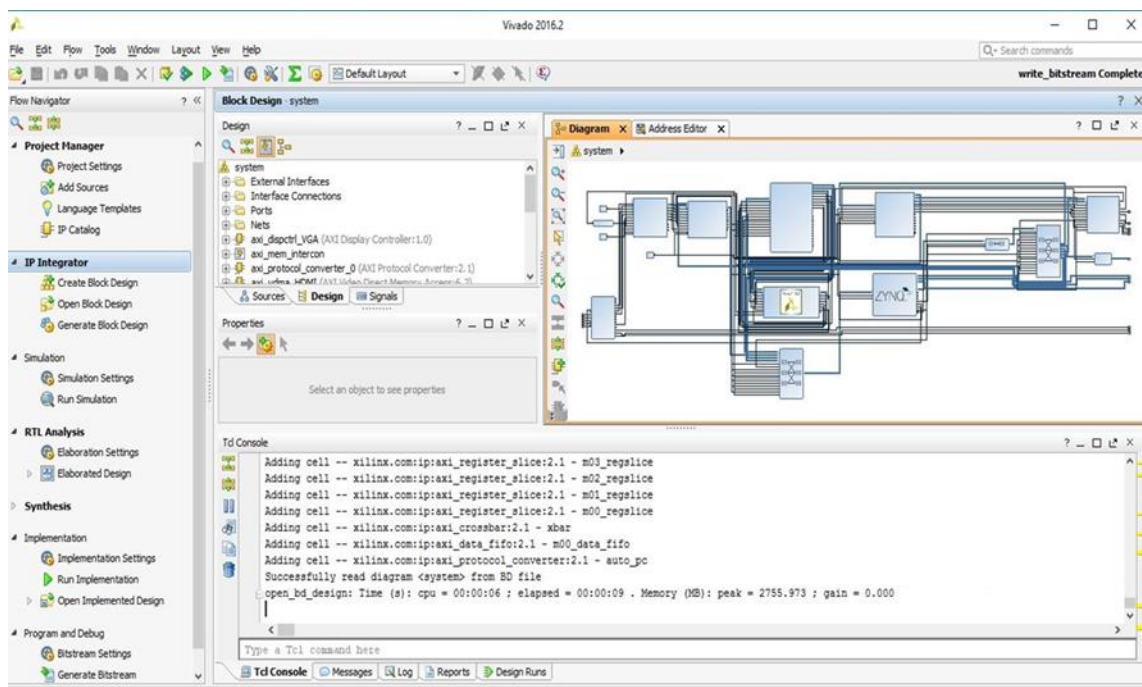
- لینک با نرم افزار XSDK، HLS، SDSOC، SDAccel و SDNet

^۱ - Synthis

هنگام نصب نرم افزار Vivado می توان نرم افزار های HLS و XSDK را نیز در کنار آن نصب

نمود. لوگوی نرم افزار به شکل زیر است:

محیط نرم افزار به صورت شکل ۳-۲۱ می باشد:



شکل (۳-۱۸) محیط نرم افزار VIVADO

۳-۸-۲- نرم افزار HLS (High Level Syntheses)

با توجه به الگوریتم های پیشرفته که امروزه در صنایع بی سیم، پزشکی، دفاع و ... استفاده می شود،

و برنامه های کاربردی که هر روز پیشرفته تر می شوند شرکت Xilinx نرم افزاری به نام Vivado High-

Level Synthesis (HLS) را به صورت رایگان همراه نرم افزار Vivado ارائه داده تا طراحان بتوانند

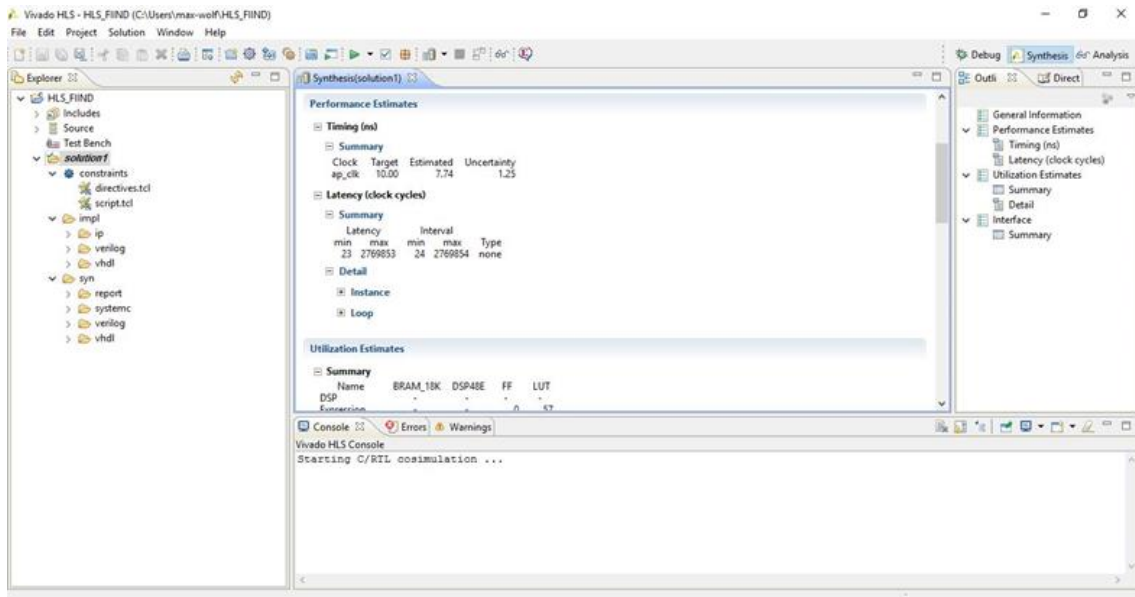
با استفاد از کتابخانه‌های موجود و با زبان‌های C، C++ و System C طرح مورد نظر خود را به صورت یک IP بسازند که این IP با هر دو نرم افزار Vivado و ISE پشتیبانی می‌شود. در واقع به زبان ساده این نرم‌افزار زبان‌های C، C++ و System C را به VHDL یا VERILOG قابل پیاده سازی روی تراشه تبدیل می‌کند [۲۴].

۳-۸-۲-۱- خصوصیات

- کتابخانه گسترده برای انواع داده با دقت دلخواه مثل کتابخانه‌های ، ویدئو، DSP و ...
- پیاده سازی IP با توجه به تراشه و افزایش سرعت کد خروجی
- شبیه سازی برنامه C (در صورت نوشتن Test Bench) و شبیه سازی برنامه‌های VHDL و Verilog به صورت جدا
- شبیه سازی با نرم افزار Modelsim
- ارائه تمامی مشخصات هسته ساخته شده از جمله سرعت اجرا ، میزان فضای استفاده شده توسط هسته و نوع ورودی‌ها و خروجی‌ها
- تصحیح کننده خطا

لازم به ذکر است از کتابخانه OpenCV فقط برای نوشتن Test bench در HLS استفاده می‌شود و قابل سنتز نمی‌باشد اما کتابخانه‌ای با نام OpenCV by Auviz برای فروش در سایت Xilinx قرار گرفته که توسط این کتابخانه اکثر توابع پر کاربرد OpenCv قابل سنتز خواهند شد.

و محیط این برنامه به همانند شکل ۳-۱۹ می باشد:



شکل (۳-۱۹) محیط نرم افزار HLS

۳-۸-۳- نرم افزار (Xilinx Software Development Kit (XSDK

برنامه XSDK یک محیط برنامه نویسی جامع برای ساخت برنامه های جاسازی شده روی همه ی میکروپروسسورهای شرکت Xilinx می باشد که شامل تراشه های Zynq-7000 AP SoCs، Zynq UltraScale+ MPSoC، و برنامه نویسی صنعت پیشواز MicroBlaze می باشد. برنامه XSDK اولین IDE برای طراحی و اشکال زدایی چند پردازنده هم شکل و یا غیر هم شکل واقعی می باشد [۲۵].

۳-۸-۳-۱- خصوصیات

- پشتیبانی از Zynq-7000 AP SoCs، Zynq UltraScale+ MPSoC، و MicroBlaze

• ارتباط داده شده با Vivado Design Suite و قابلیت دانلود جداگانه برای استفاده‌های دیگر

• نوشته شده روی Eclipse 4.3.2 و CDT 8.3.0

• محیط ساده و طراحی ساده

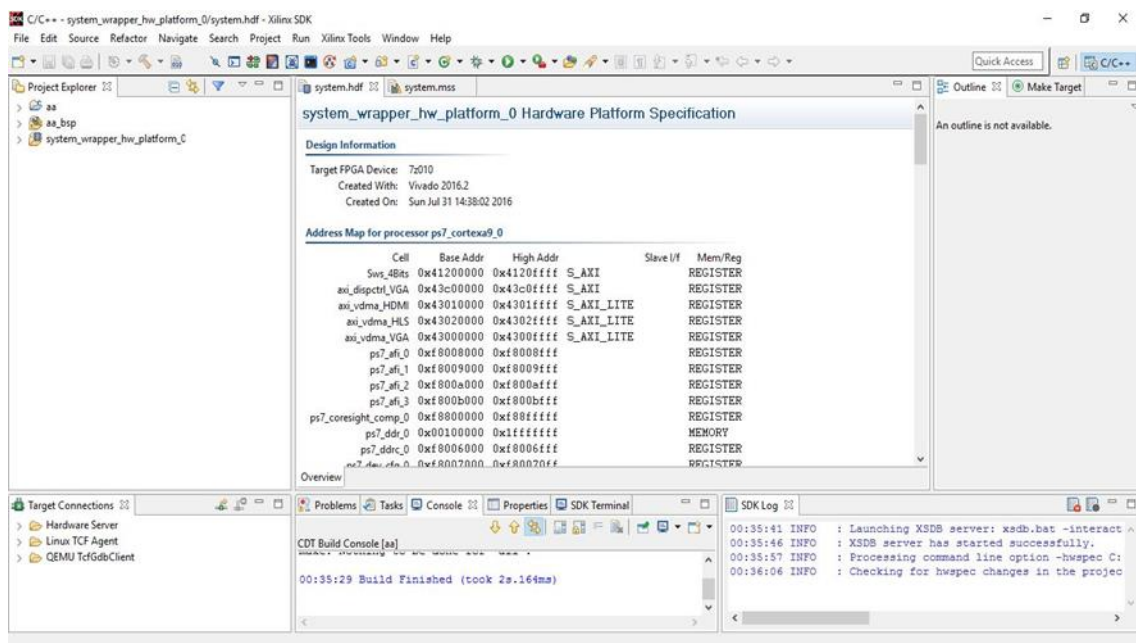
• محیط اشکال‌زدایی کامل با ارتباط JTAG

• دارا بودن کتابخانه‌های مختلف

محیط SDK پس از طراحی سخت افزار در محیط Vivado از قسمت File گزینه‌ی launch

SDK قابل دسترس می‌باشد.

محیط این نرم افزار به همانند شکل ۳-۲۰ است:

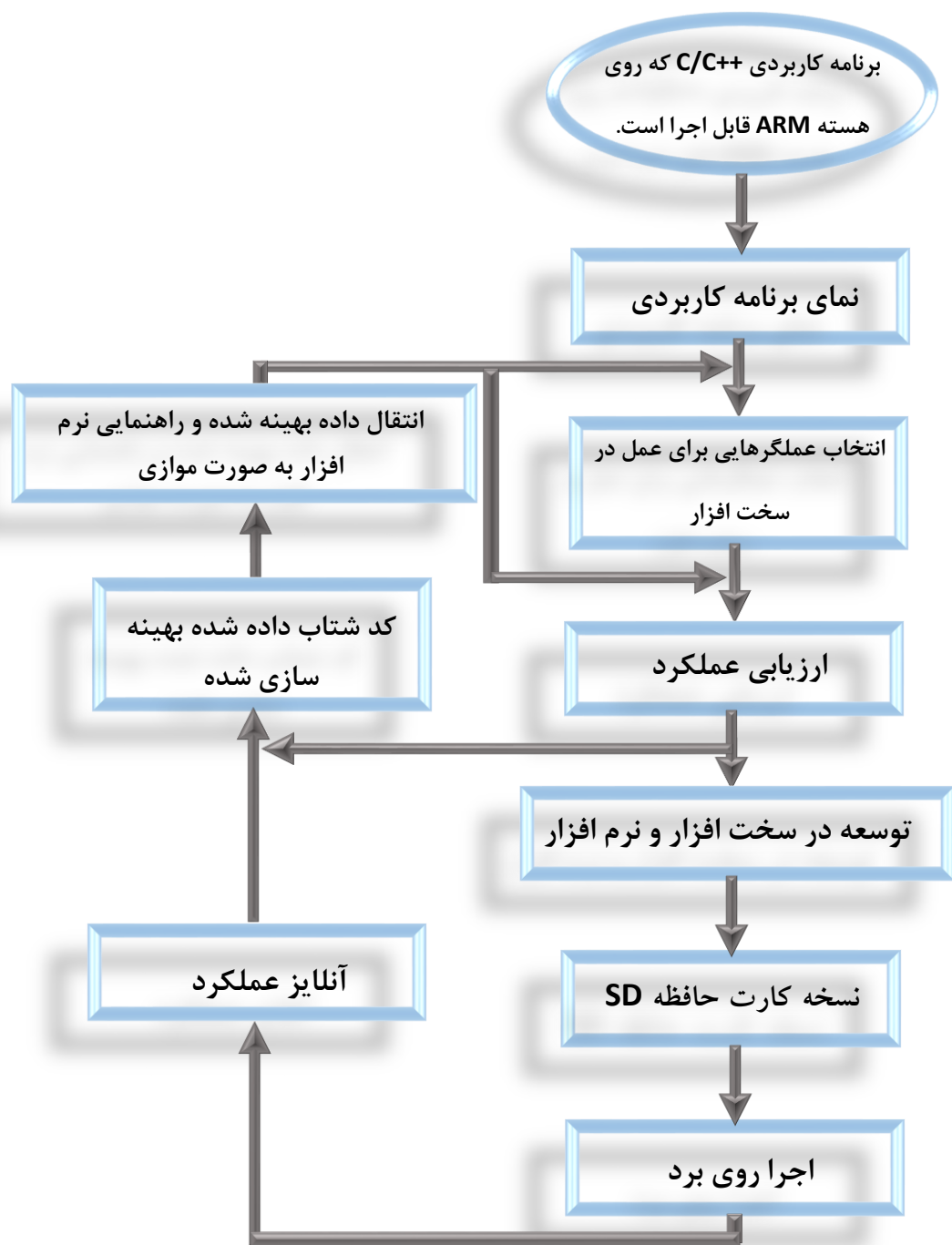


شکل (۳-۲۰) محیط نرم‌افزار SDK

۳-۸-۴- نرم افزار SDSoc (Software-Defined System on chip)

نرم افزار SDSoc یک محیط توسعه یکپارچه می باشد که با همکاری دو شرکت Xilinx و Eclipse معرفی شده است. این نرم افزار برای پیاده سازی و اجرای سیستم های ناهمگن یکپارچه شده می باشد. باید توجه کرد که این نرم افزار فقط برای تراشه های ZYNQ SoC و ZYNQ UltraScale طراحی شده است و برای FPGA ها کاربردی ندارد. محیط SDSoc یک تجربه توسعه کاربردی C++/C یکجا را ارائه می دهد یعنی شامل یک کامپایلر C++/C بهینه سازی شده که شتاب دهنده نرم افزاری را به صورت اتوماتیک در قسمت پیاده سازی منطقی به وجود می آورد.

در صورت استفاده از این نرم افزار به عنوان محیط توسعه، دیگر نیازی به نرم افزارهایی همانند Vivado و SDK نمی باشد، زیرا خروجی این محیط یک فایل بوت برای اجرا از طریق کارت حافظه SD می باشد.



شکل (۳-۲۱) فلوچارت طراحی در نرم افزار SDSoC

- آسان برای توسعه IDE بر روی تراشه های ZYNQ
- شتاب دهنده یک عملگر روی قسمت PL
- ساخت برنامه کاربردی برای سیستم عامل های Linux و FreeRTOS
- کتابخانه های متعدد برای کارهای مختلف

فصل چهارم: نحوه پیاده سازی

۴-۱- مقدمه

تراشه ZYNQ قابلیت اجرای برخی از سیستم عامل ها مانند لینوکس را داراست، در سیستم عامل xilinx که ورژن بهینه شده Linux می باشد. وظیفه مدیریت ورودی ها و خروجی ها بر عهده PL می باشد و بقیه کار ها مرتبط با PS است. حال برای اجرای برنامه ای با دستورات OpenCV احتیاج به سیستم عاملی می باشد، که به دلیل اجرای سیستم عامل روی PS باز هم به سرعت ایده آل نخواهد رسید. با اینکه سرعت اجرا از پلتفرم هایی مثل ریسپری پای بیشتر است، اما با تراشه ی ZYNQ می توان با روش دیگر^۱ به سرعت بسیار بالاتری رسید اما در این حالت امکان پیاده سازی توابع OpenCV به

۱- پیاده سازی کامل روی PL بدون سیستم عامل

طور مستقیم نخواهد بود و باید الگوریتم مورد نظر را برنامه نویسی کرد و یا از برخی توابع پیش فرض استفاده کرد، این کار توسط نرم افزار HLS انجام خواهد شد که در ادامه به بررسی آن پرداخته می شود.

۴-۲- مقایسه زبان ها و نرم افزار ها در حوزه پردازش تصویر

امروزه پردازش تصویر یکی از شاخه های مهم در علم الکترونیک و کامپیوتر تلقی می شود. علت این امر هم کاربردی بودن و انعطاف پذیری بالای این ابزار می باشد. پس نسبت به مهمی مطلب تعداد کاربر زیادی نیز در پی یادگیری و استفاده از این مهم هستند. موازی این مطالب توسعه دهندگان زبان های برنامه نویسی و نرم افزارهای کاربردی سعی بر ارائه نرم افزارهای متفاوت با ویژگی های منحصر به فرد می باشند.

اگر نگاهی کلی به زبان های برنامه نویسی با کنسول های متفاوت شود می توان با مد نظر داشتن معیارهایی همچون تعداد کاربر، رابط کاربری نرم افزار، سرعت و دقت، می توان پیاده سازی پردازش در حوزه تصویر و ویدئو به یکی از سه روش جامع زیر صورت پذیرد:

- استفاده از نرم افزار متلب

- استفاده از کتابخانه OpenCV

- استفاده از زبان VHDL یا VERILOG و پیاده سازی در FPGA

برای دستیابی به سرعت حداکثر و انجام پردازش ویدئو به صورت آنی تصمیم به نوشتن الگوریتمی ساده برای تشخیص و ردیابی جسم و پیاده سازی کامل آن روی PL گرفته شد، در ادامه پس از توضیح این الگوریتم به بررسی آن با شیوه های مختلف پردازش ویدئو پرداخته خواهد شد.

۴-۳- توضیح الگوریتم استفاده شده

در این پژوهش سعی شده است از یک الگوریتم بسیار ساده و عملی استفاده شود تا بدین طریق امکان دستیابی به چهار خواسته مهم یعنی پیاده سازی کامل آن روی PL، رسیدن به پردازش آنی، مقایسه با سایر روش ها و پیاده سازی و اجرای عملی ممکن شود، لازم به ذکر است سخت افزار مدنظر و موجود تراشه ZYNQ می باشد و همانطور که در فصل قبل توضیح داده شد، نرم افزار مرتبط با این تراشه نرم افزار SDSoc می باشد که به علت وجود محدودیت های خرید لایسنس برای کشور عزیزمان ایران، این نرم افزار در دسترس قرار نداشت و به عنوان جایگزین از نرم افزار HLS استفاده شد که بیشتر برای سخت افزار های مبتنی بر FPGA می باشد.

۴-۳-۱- نحوه ردیابی شیء

عدم دسترسی کامل به ماتریس تصاویر و نبود امکان دسترسی رایگان به الگوریتم های مهم همانند لبه یابی، گوشه یابی و دیگر دستورات مهم، دلایلی می باشند که باعث استفاده از الگوریتم نه چندان پیچیده تشخیص شیء با پیدا کردن رنگ منحصر به فرد شدند.

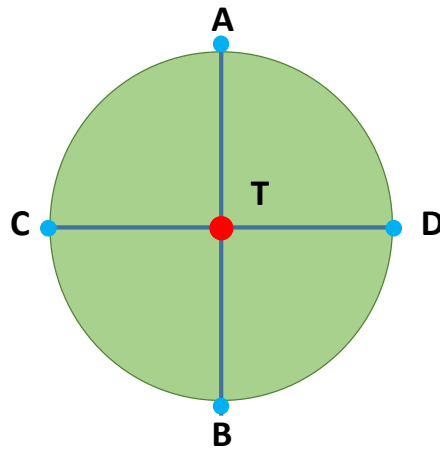
برای توضیح بیشتر درباره الگوریتم هدف باید گفت که فرض بر این است در محیط محدوده رنگی جسم هدف به صورت یکتاست. به عنوان مثال هدف تشخیص و ردیابی توپی سبز رنگ می باشد که در محیط ردیابی تنها این جسم به صورت سبز کم رنگ وجود دارد و محدوده رنگ سبز کم رنگ به صورت یکتا و ویژه جسم هدف مورد نظر می باشد.

الگوریتم استفاده شده بدین اساس است که پس از دریافت فریم مربوطه پیکسل ها را از ابتدا تا انتها اسکن کرده و آنها را با رنج رنگی مورد نظر مقایسه می شود و پیکسل هایی که مورد نظر می باشند

علامت گذاری می شوند، در این مرحله امر تشخیص جسم بر اساس ویژگی مورد نظر یعنی رنگ شیء انجام شد.

۴-۳-۲- مشخص کردن مختصات شیء

حال برای تعیین محل جسم در تصویر به این شیوه عمل خواهد شد که پس از پیدا کردن چهار نقطه روی محیط توپ، مختصات وسط توپ به آسانی با انجام میانگیری اعداد صورت می پذیرد.



شکل (۴-۱) نحوه موقعیت یابی شیء

$$T(row) = A + \frac{B-A}{2} \quad (۴-۱)$$

$$T(column) = C + \frac{D-C}{2} \quad (۴-۲)$$

تشخیص مختصات شیء در تصویر بدین صورت انجام می‌پذیرد که کمترین و بیشترین مقادیر در دو متغیر ردیف و ستون هر فریم که در رنج رنگی مورد نظر می‌باشند، تعیین می‌شوند. و با استفاده از دو رابطه (۱-۴) و (۲-۴) مکان وسط پیکسلی شیء تشخیص داده می‌شود.

۴-۳-۳- بدست آوردن فاصله شیء از دوربین

برای تشخیص فاصله جسم از دوربین هم بدین صورت عمل شده است که ابتدا قطر توپ در مقیاس واقعی اندازه‌گیری شده است و بعد توپ در فاصله ای مشخص (برای مثال یک متر) از دوربین قرار می‌گیرد و تعداد پیکسل هایی که در تصویر قطر توپ را مشخص می‌کنند را به دست می‌آید. حال با استفاده از رابطه (۳-۴) درجه کانونی لنز دوربین تعیین می‌شود:

$$F = W \times D \quad (۳-۴)$$

- F = درجه کانونی لنز دوربین
- W = قطر توپ (سانتی متر)
- D = تعداد پیکسل مشخص کننده قطر توپ در تصویر در فریم استاندارد

حال با توجه به رابطه (۴-۴) فاصله شی تا دوربین در هر فریم به صورت مجزا معین می‌شود:

$$D' = \frac{W \times F}{P} \quad (۴-۴)$$

- F = درجه کانونی لنز دوربین
- W = قطر توپ (سانتی متر)
- P = تعداد پیکسل مشخص کننده قطر توپ در تصویر در هر فریم
- D' = فاصله توپ تا دوربین (سانتی متر)

۴-۴- پیاده سازی الگوریتم مورد نظر در نرم افزار متلب

متلب یک نرم افزار پر کاربرد برای استفاده در ساخت نسخه های آزمایشی سریع است و باید در نظر گرفت کد های متلب برای اشکال زدایی^۱ آسان هستند. و مطالب و پشتیبانی خوبی از سمت شرکت MathWork صورت می گیرد. اما متلب حالت کد باز ندارد، لایسنس کامل آن قیمت خیلی بالایی دارد و این نرم افزار حالت پرتابل ندارد.

متلب یک زبان ترجمه شده است و این موضوع به صورت منفی در کارکردش تاثیر میگذارد. اما مشکل اصلی متلب سرعت پایین این نرم افزار در تحلیل تصاویر است که بعضی مواقع نمی شود به صورت آنی (Real time) استفاده کرد. درست است سرعت پردازش رابطه مستقیم با میانگین نمره سخت افزار سیستم دارد، ولی در یک سیستم یکسان رقبای این نرم افزار همانند OpenCV عملکرد خیلی بهتری دارند.

در این بخش به بررسی الگوریتم مربوطه توسط نرم افزار متلب پرداخته خواهد شد، لازم به ذکر است کلیه تست ها و سرعت سنجی ها روی متلب با ورژن 2016a و سخت افزاری با شرایط زیر انجام شده است.

۱- Debug

جدول (۱-۴) مشخصات سخت‌افزاری

مدل		نوع
Intel Core i5 -2410 2.3GHz/2.3GHz		پردازنده
6GB		حافظه
GeForce GT520M	اصلی : ۱۰۲۴	گرافیک
	اشتراکی: ۳۰۶۲	
	کل: ۴۰۸۶	

۴-۴-۱- بررسی کد

در ابتدا قبل از هر چیز صفحه کار را پاک کرده و تمام صفحه‌های باز را بسته و متغیرها پاک

خواهند شد:

```
clc
clear all
close all
```

سپس توسط دستور زیر ویدئوی مورد نظر را اجرا و فریم‌ها را درون متغیری با نام `img` ریخته

می‌شوند.

• عملیات خواندن ویدئو می‌تواند توسط دستورهای دیگر به واسطه دوربین انجام پذیرد که دستور

`while` برای اجرای پشت سر هم فریم‌ها و همچنین متغیر `counter` برای شمارش تعداد فریم‌های

ورودی می‌باشد که در پایان ویدئو توسط شرط تعریفی برنامه را به اتمام برساند. البته در تست اولیه

فقط یک فریم بررسی می‌شود و در فصل بعد نتایج حالت ویدئو بررسی خواهد شد.

```
v=VideoReader('Test.mp4');
counter=0;
while hasFrame(v)
counter=counter+1;
img = readFrame(v);
```

توسط دو دستور For موجود کل فریم را پیکسل به پیکسل اسکن کرده و مقادیر R و G و B را به تفکیک درون متغیرهای جدا ریخته می‌شوند:

```
for icol=1:1:1280
    for irow=1:1:720
        R=img(irow,icol,1);
        G=img(irow,icol,2);
        B=img(irow,icol,3);
```

توسط دستور زیر رنج کلی رنگ سبز از پیکسل ها جدا می‌شود، در صورتی که پیکسل مورد نظر دارای رنج رنگی سبز باشند حاصل فرمول بزرگ تر از ۴۵ خواهد بود:

```
X=G-B/2-R/2;
if X>45
```

در دستور بعد رنج سبز جدا شده را محدودتر کرده تا فقط رنج رنگی هدف مورد نظر از تصویر جدا شود، سپس پیکسل تشخیص داده شده با رنگ سیاه علامت‌گذاری می‌شود:

```
if G>150 && G<255 && B>0 && B<95 && R>0 && R<143
    img(irow,icol,1)=0;
    img(irow,icol,2)=0;
    img(irow,icol,3)=0;
end

imshow(img);
```

و در آخر توسط دستور فوق تصویر نهایی نمایش داده خواهد شد:



شکل (۲-۴) نتیجه پردازش با نرم افزار متلب

- لازم به ذکر است که از توضیح پیدا کردن مختصات جسم به دلیل مشابه بودن در تمامی قسمت ها در این بخش و بخش OpenCv خودداری شده و در قسمت HLS به طور مفصل به این موضوع پرداخته شده است، همچنین کدهای هر سه نرم افزار به صورت کامل پیوست شده اند.

۴-۵- پیاده سازی الگوریتم مورد نظر در کتابخانه OpenCV

OpenCV نام یک کتابخانه است که توسط شرکت ایتل شروع به تولید شده و کمپانی willow ادامه دهنده روند رو به رشد ایتل بود و اخیراً یک شرکت مستقل به نام OpenCV شده است.

این کتابخانه حالت کد باز دارد و فاصله بین دو بروز رسانی این نرم افزار تقریباً چند سال می شود که به صورت سه تا چهار ماه می باشد. یک مزیت خیلی مهم OpenCV این است که تعداد کاربر و انجمن گفتمان زیادی دارد و راحت می شود مشکلات در برنامه نویسی را حل کرد.

OpenCV الگوریتم و تابع زیادی در مقایسه با متلب دارد. این کتابخانه قابل استفاده در زبان های C++، C، Python و Java می باشد.

در مورد سرعت پردازش این کتابخانه می توان گفت، خیلی سریع تر از متلب می باشد چون توابع هر زمانی قابل بهینه سازی می باشد و در هر به روز رسانی توابع جدید با سرعت پردازش بهتر ارائه می شود.

برای استفاده از کتابخانه OpenCV نیاز به یک نرم افزار کامپایلر می باشد که واسط استفاده شده در این پژوهش نرم افزار Microsoft Visual Studio 2013 است. نسخه کتابخانه مربوطه برای دسترسی به توابع ۳,۱,۰ می باشد و QT نصب شده نسخه ۳,۶,۱ از این نرم افزار است.

لازم به ذکر است سخت افزار استفاده شده برای پیاده سازی الگوریتم در این نرم افزار، همان سخت افزاری می باشد که برای نرم افزار متلب استفاده شده است. پس هر دو تست در سیستم ۶۴ بیتی اجرا شده اند و به یک اندازه به پردازنده و حافظه رم دسترسی دارند.

زبان استفاده شده برای برنامه نویسی در کتابخانه OpenCV، زبان ++C می باشد و همانطور که ذکر شد، از کامپایلر Visual Studio استفاده شده است. پس در ابتدای کد نیاز به معرفی فایل های سرآیند^۱ مربوطه می باشد:

در سه خط اول هدر های مورد نیاز برای داشتن ورودی و خروجی در زبان ++C و خط چهارم برای استفاده از دستورات این کتابخانهها بدون نیاز به اعلام کلمه std در اول دستورات می باشد. قسمت بعد هم هدر فایل هایی برای داشتن خروجی تصویر و استفاده از هسته اصلی کتابخانه OpenCV می باشد:

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

//OpenCV Header Files
#include "opencv2/core/core.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;
```

تعریف دو متغیر از جنس اعداد صحیح دوبرابر حالت پیش فرض^۲ برای ذخیره زمان به دست آمده از تایمر برای اندازه گیری مدت زمان اجرای توابع که در سطر بعدی هم با استفاده از تابع تایمر به زمان فعلی تایمر دسترسی دارد:

```
double start_time, finish_time;
start_time = getTickCount();

Mat frame=imread("Test.jpg");
/*VideoCapture cap("Test.mp4");
//Mat frame;*/
Vec3b pixel;
```

۱- Header File

۲- Double Integer

- باید به این نکته اشاره کرد که کل برنامه همانند قسمت نرم افزار متلب در دو حالت یک فریم و دویست فریم تست می شود پس به دو حالت ورودی گرفته می شود یکی به حالت فایل تصویر و در حالت بعدی که به صورت کامنت مشاهده می شود به شکل فایل ویدئویی با کیفیت 720p است.

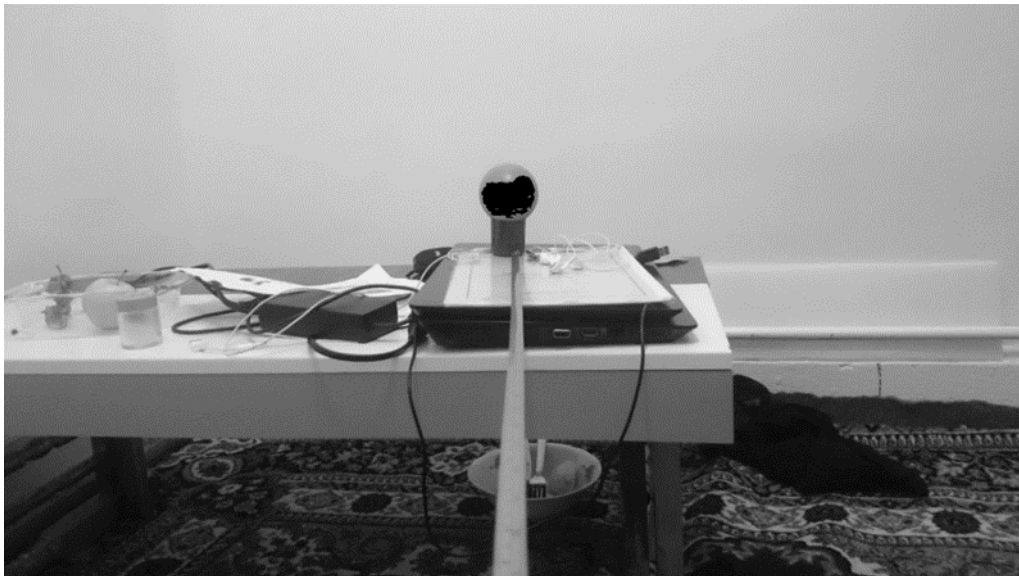
- متغیر pixel از جنس Vec3b تعریف شده است که یک متغیر مختص کتابخانه OpenCV می باشد و یک متغیر سه مقداره منحنی وار برای ذخیره سه مقدار رنگ اصلی است. قسمت اصلی کد می باشد که شامل دو حلقه اصلی و شرایط جداسازی رنگ است. مقادیر حلقه ها به تعداد پیکسلها در طول و ارتفاع تصویر می باشند و در مجموع ۹۲۱۶۰۰ بار تکرار می شود و کل پیکسلهای تصویر را چک می کند و هر کدام در رنج رنگی مورد نظر بود را با رنگ سیاه مشخص می کند:

```
for (int ir = 0; ir < 1280; ir++)
{
    for (int ic = 0; ic < 720; ic++)
    {
        pixel = frame.at<Vec3b>(Point(ir, ic));
        pixel[2] = pixel[2];
        pixel[1] = pixel[1];
        pixel[0] = pixel[0];
        int D = pixel[1] - pixel[0] / 2 - pixel[2] / 2;
        if (D>45)
        {
            if (pixel[0]>=0 & pixel[0]<=95 & pixel[1]>=150 &
                pixel[1]<=255 & pixel[2]>=0 & pixel[2]<=143)
            {
                pixel[2] = 0;
                pixel[1] = 0;
                pixel[0] = 0;
                frame.at<Vec3b>(Point(ir, ic)) = pixel;
            }
        }
    }
}
```

در قسمت آخر باید نتایج پردازش مشاهده شود که با استفاده از دستور imshow تصویر مربوطه نمایش داده می‌شود و در ادامه زمان اجرا در کنسول نوشته می‌شود که در فصل بعدی تحلیل خواهند شد:

```
imshow("img", frame);
finish_time = getTickCount();
cout << "Time per frame: " << (finish_time - start_time) / getTickFrequency()
<< "secs" << endl;
```

حال میتوان شکل خروجی که به دست آمده است را مشاهده نمود. همان طور که از تصویر خروجی مشخص است شیء مورد نظر به درستی تشخیص داده شده است و هیچ مشکلی در اجرای کد نوشته شده وجود ندارد:



شکل (۴-۳) نتیجه پردازش با کتابخانه OpenCV

۴-۶- پیاده سازی الگوریتم مورد نظر روی تراشه ZYNQ

همانطور که به طور مفصل در فصل قبل توضیح داده شد، بهترین پلتفرم برای پیاده سازی عملیات پردازش تصویر در محیط های حساس و غیر آزمایشگاهی و انجام این عملیات با سرعت خیلی بالا، تراشه ZYNQ می باشد. پس در این بخش به توضیح مفصل درباره نحوه پیاده سازی در این حوزه پرداخته خواهد شد.

در کل دو روش برای انجام پردازش تصویر با تراشه ZYNQ وجود دارد:

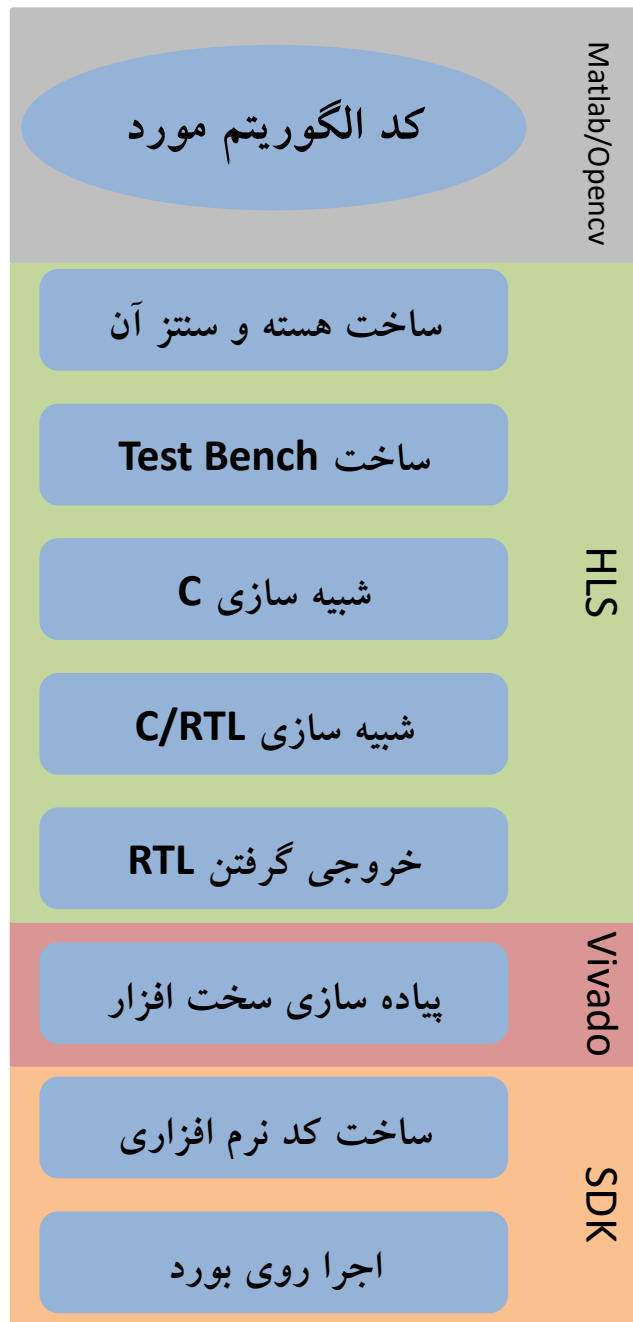
۴-۶-۱- استفاده از سیستم عامل لینوکس و کد OpenCV

در این روش از سیستم عامل Xilinx استفاده می شود که همان سیستم عامل Linux Ubuntu می باشد ولی با بهینه سازیهای بسیار ریز که این امکان را به کاربر می دهد تا بتواند به قسمت PL آی سی و هسته های مربوطه دسترسی آسان داشته باشد و به راحتی بتواند محاسبات را انجام دهد. ولی چون هدف از این پژوهش پیاده سازی الگوریتم در قسمت PL می باشد، فقط همانند دو قسمت قبل توضیح کوتاهی درباره پیاده سازی گفته می شود.

۴-۶-۲- ساخت هسته اختصاصی مورد نظر و پیاده سازی آن

این حالت دارای سرعت پردازش خیلی بالایی است ولی نسبت به حالات قبل باید قسمتهای زیادی انجام شود که در کل پیاده سازی شامل مراحل زیر می باشد:

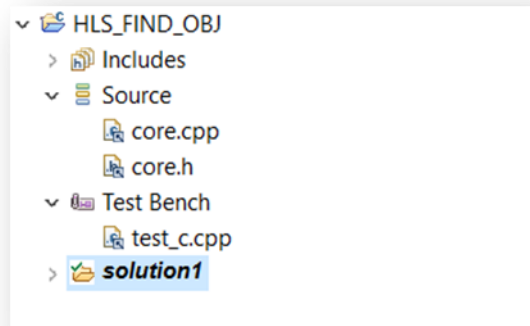
- ساخت هسته الگوریتم مورد نظر در نرم افزار HLS
- پیاده سازی بلوکی سخت افزار های مورد نیاز در نرم افزار Vivado
- توسعه قسمت نرم افزاری الگوریتم با زبان C و در نرم افزار SDK



شکل (۴-۴) مراحل پیاده‌سازی روی تراشه ZYNQ

۴-۶-۲-۱- ساخت هسته الگوریتم مورد نظر در نرم افزار HLS

برای ساخت هسته اختصاصی در نرم افزار HLS ابتدا باید یک پروژه جدید ساخت و تراشه هدف را به دقت انتخاب کرد و بعد اقدام به ساخت سورس کد و میز تست شود:



شکل (۴-۵) فایل های مورد نیاز هسته

همانطور که در شکل بالا ملاحظه می شود در قسمت منبع هسته مورد نظر دو فایل موجود می باشد که فایل با نام `core.cpp` دربرگیرنده کد اصلی می باشد و `core.h` هدر فایل هسته می باشد.

۴-۶-۲-۱- بررسی کد

فایل سرآیند مورد نظر شامل کدهایی می باشد که تابع اصلی، انواع کتابخانه های استفاده شده و انواع متغیرهای ورودی و خروجی را مشخص می کند. کتابخانه های مورد نظر `hls_video.h` و `ap_axi_sdata.h` می باشد که هدر اولی شامل کتابخانه های است که دربرگیرنده دستورات ترجمه شده کتابخانه OpenCV به HDL می باشد و کاربر هیچ نقشی در این دستورات ندارد و در حقیقت خود

شرکت Xilinx این کتابخانه را برای راحتی کاربران ایجاد کرده است ولی مشکل عمده در این کتابخانه، کمی توابع و عدم گسترش کامل این بخش می باشد. هدر فایل دوم برای تعریف نوع متغیرهای ورودی و خروجی می باشد که عهده دار انتقال سیگنال های تصویر در هسته می باشند:

```
#include<hls_video.h>
#include<ap_axi_sdata.h>
```

تعریف بیشترین و کمترین تعداد پیکسل در طول و ارتفاع تصویر:

```
#define MAX_WIDTH 1280
#define MAX_HEIGHT 720
```

ابتدا دو متغیر ایجاد شده است که اولی از جنس اسکالر و سه مقداره برای ذخیره سه رنگ اصلی می باشد و متغیر دومی یک ماتریس به اندازه تصویر ورودی می باشد. سطر سوم نوع متغیر استفاده شده برای انتقال تصاویر دریافتی و ارسالی از طریق AXI Stream می باشد. تا این قسمت فقط هدر فایل هسته توضیح داده شد که شامل کلیات می باشد در ادامه به توضیح کد اصلی پرداخته می شود:

```
typedef hls::Scalar<3, unsigned char> RGB_PIXEL;
typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> RGB_IMAGE;
typedef ap_axiu<32,1,1,1> uint_32_ch;
```

شده برای انتقال تصاویر دریافتی و ارسالی از طریق AXI Stream می باشد. تا این قسمت فقط هدر فایل هسته توضیح داده شد که شامل کلیات می باشد در ادامه به توضیح کد اصلی پرداخته می شود.

ابتدا ورودی و خروجی هسته و نوع آنها بررسی می شود:

جدول (۴-۲) نوع متغیرهای ورودی/خروجی هسته

نام متغیر	ورودی/خروجی	نوع متغیر	وظیفه
instream	ورودی	stream<uint_32_ch>	دریافت فریم
outstream	خروجی	stream<uint_32_ch>	ارسال فریم
rows	ورودی	int	تعداد سطر فریم
cols	ورودی	int	تعداد ستون فریم
GL	ورودی	int	کمترین مقدار رنگ سبز
GH	ورودی	int	بیشترین مقدار رنگ سبز
BL	ورودی	int	کمترین مقدار رنگ آبی
BH	ورودی	int	بیشترین مقدار رنگ آبی
RL	ورودی	int	کمترین مقدار رنگ قرمز
RH	ورودی	int	بیشترین مقدار رنگ قرمز
TL	ورودی	int	کمترین مقدار فیلتر دوم
TH	ورودی	int	بیشترین مقدار فیلتر دوم
EN	ورودی	int	فعال کردن حالت تست
return	خروجی	unsigned long int	بازگرداندن مکان و فاصله

• برای هر یک از ورودی و خروجی ها باید از قسمت Directive خصوصیات مشخصی تعریف

شود تا هسته در نرم افزار Vivado با مشکل مواجه نشود.

ابتدا باید تمامی متغیرهای مورد نیاز را قبل حلقه اصلی معرفی کرد. دو متغیر اولی از جنس

ماتریس بوده و در تعداد سطر و ستون آنها از قسمت نرم افزاری تعیین می شود. متغیر pix از جنس

اسکالر^۱ برای ذخیره سازی مقادیر رنگ‌ها می‌باشد و Pos یک متغیر ۱۰ رقمی برای انتقال موقعیت و فاصله شیء به بیرون از هسته است. row و col برای نمایش موقعیت پیکسل مورد بررسی در حلقه می‌باشد که علت استفاده از این روش در قسمت توضیحات حلقه گفته می‌شود. چهار متغیر آخر هم همان نقاط A، B، C و D روی محیط شیء می‌باشند:

```
RGB_IMAGE img_0(rows,cols);
RGB_IMAGE img_1(rows,cols);
RGB_PIXEL pix;
//Variable for Return
unsigned long int Pos=0;
//Variable for Position in Pixels
int row=1;
int col=1;
//Variable for min/MAX of Row and Column
int TRL=1;
int TRH=1;
int TCL=1;
int TCH=1;
```

استفاده از کتابخانه hls_video برای گرفتن فریم از ورودی و تبدیل آن به ماتریس:

```
hls::AXIvideo2Mat(instream, img_0);
```

حلقه اصلی کد به تعداد کل پیکسل‌های تصویر اجرا می‌شود و علت استفاده از یک حلقه، استفاده از حالت PipeLine می‌باشد تا اجرای حلقه را کاهش داد. پس با استفاده از شرط نوشته شده می‌توان موقعیت سطر و ستون را به دست آورد:

^۱ - Scalar

```

for (int i=0;i<(rows*cols);i++)
{
    #pragma HLS PIPELINE
    if (col==1280)
    {
        col=0;
        row++;
    }
    col++;
}

```

در خطر اول مقدار پیکسل مربوطه از ماتریس `img_0` دریافت می‌شود و بعداً هر یک از مقادیر `pix` در متغیرهای `B`، `G` و `R` قرار می‌گیرد. و توسط دو شرط ذکر شده در قسمت های قبلی رنگ شیء مورد نظر تعیین می‌شود:

```

pix=img_0.read();
unsigned char R,G,B;
B=pix.val[0];
G=pix.val[1];
R=pix.val[2];
X=G-(B/2)-(R/2);
if (G>=GL && G<=GH && B>=BL && B<=BH && R>=RL && R<=RH)
{
    if (X>=TL && X<=TH)
    {

```

شروط لازم برای تعیین موقعیت شیء:

```

if (TCL==1) TCL=col;
if (TRL==1)
{
    TRL=row;
    DR=row;
    DC=col;
    ENS=1;
}
if (TRL>row) TRL=row;
if (TRH<row) TRH=row;
if (TCL>col) TCL=col;
if (TCH<col) TCH=col;

```

دستوری برای رسم یک شکل مربعی در مکان شیء در تصویر خروجی:

```
if (EN==0 && ENS==1 && row>=(DR+2) && row<DR+22 && col>=(DC-10) && col<=(DC+10))
{
    pix.val[0]=0;
    pix.val[1]=255;
    pix.val[2]=255;
}
img_1.write(pix);
```

محاسبات لازم برای پیدا کردن موقعیت و فاصله شیء :

```
long int Trow=(TRH-TRL)/2;
unsigned long int Tcol=(TCH-TCL)/2;
Tcol=Tcol+(TCL-1);
Trow=Trow+(TRL-1);
int Dis=TCH-TCL;
Trow=Trow*1000;
Tcol=Tcol*1000000;
Pos=Tcol+Trow+Dis;
```

تابعی برای ارسال تصویر و تبدیل ماتریس به AXI و بازگرداندن مقدار متغیر Pos :

```
hls::Mat2AXIvideo(img_1,outstream);
return Pos;
```

تا این قسمت هسته مورد نظر ساخته شد ولی باید قبل خروجی گرفتن از نرم افزار HLS یک

کد دیگر برای تست هسته باشد، پس در ادامه کد قسمت Test Bench توضیح داده می شود:

همانند کد OpenCV ابتدا باید کتابخانه های مورد نظر فراخوانی شوند. اولین هدر فایل مربوط

به هسته مورد نظر می باشد. هدر فایل بعدی هسته اصلی کتابخانه OpenCV می باشد که استفاده از آن

در قسمت test bench مجاز است. سومین خط مربوط به توابع پایه زبان C++ بوده و دو خط آخر

شامل توابع ترجمه شده توسط شرکت می باشد:

```
#include "core.h"
#include "opencv2/core/core.hpp"
#include<stdio.h>
#include<hls_video.h>
#include<hls_opencv.h>
```

تعریف آدرس فریم ورودی و خروجی و اندازه تصویر خروجی:

```
#define INPUT_IMAGE_ADDR      "C:\\Users\\...\\Desktop\\Test.jpg"
#define OUTPUT_IMAGE_ADDR    "C:\\Users\\...\\Desktop\\img.jpg"
char outImage[1280][720];
```

در خط اول همانند قسمت OpenCV تصویر ورودی در یک ماتریس ریخته می‌شود. در دو سطر بعدی متغیر هایی تعریف شده است تا بتوان ماتریس ساخته شده در سطر قبل را به هسته مورد نظر فرستاد. در ادامه هم ماتریس مورد نظر به تابع مربوطه به هسته (find) با تنظیمات مورد نظر ارسال می‌شود و بعد نتایج از هسته دریافت شده و به صورت فایل ذخیره می‌شود:

```
cv::Mat inImg=cv::imread(INPUT_IMAGE_ADDR);

hls::stream<uint_32_ch> inputstream;
hls::stream<uint_32_ch> outputstream;

cvMat2AXIvideo(inImg,inputstream);
find(inputstream,outputstream,64,64,45,255,100,255,0,95,0,143,1);
AXIvideo2cvMat(outputstream,imgCvout);

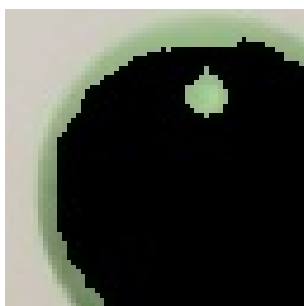
saveImage(std::string(OUTPUT_IMAGE_ADDR),imgCvout);
```

بعد نوشته شدن کد توضیح داده شده، شبیه سازی های C و C/RTL انجام می‌شود که نتایج این

شبیه سازی ها در ادامه قابل مشاهده می‌باشد:



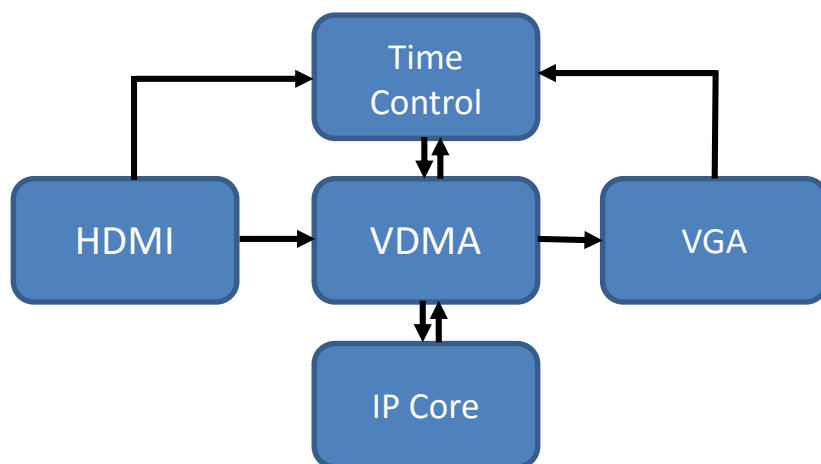
شکل (۶-۴) تصویر قبل اعمال به هسته



شکل (۷-۴) تصویر بعد از اعمال به هسته

۲-۲-۶-۴ طراحی سخت افزار مورد نظر در نرم افزار Vivado

بعد انجام این مراحل هسته به شکل RTL ذخیره می شود و برای پیاده سازی و تعیین سخت افزار مورد نظر باید مراحل را در نرم افزار Vivado ادامه داد. در نرم افزار Vivado ابتدا باید یک پروژه نسبت به مورد مورد نظر ساخت و بعد یک بلوک طراحی ایجاد کرد و با استفاده از بلوک های آماده و Core هایی که توسط شرکت یا کاربر ساخته شده اند روند زیر را پیاده کرد:



شکل (۸-۴) نحوه اتصال بلوک ها

آدرس دهی تمامی بلوک ها به صورت اتوماتیک و توسط نرم افزار صورت می گیرد ولی آدرس هسته ساخته شده (find_0) باید به صورت دستی انجام شود:

جدول (۳-۴) آدرس دهی بلوک ها

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
axi_vdma_VGA					
Data_MM2S (32 address bits : 4G)					
processing_system7_	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
axi_vdma_HDMI					
Data_S2MM (32 address bits : 4G)					
processing_system7_	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
axi_vdma_HLS					
Data_MM2S (32 address bits : 4G)					
processing_system7_	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
Data_S2MM (32 address bits : 4G)					
processing_system7_	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_vdma_VGA	S_AXI_LITE	Reg	0x4300_0000	64K	0x4300_FFFF
axi_vdma_HDMI	S_AXI_LITE	Reg	0x4301_0000	64K	0x4301_FFFF
axi_vdma_HLS	S_AXI_LITE	Reg	0x4302_0000	64K	0x4302_FFFF
axi_dispcctl_VGA	S_AXI	S_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF
Sws_4Bits	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
find_0	S_AXI_CONTROL_BUS	Reg	0x43C1_0000	64K	0x43C1_FFFF

شماتیک کامل سخت افزار ضمیمه شده است. بعد ترسیم، باید خروجی های مورد نظر ساخته شود که شامل مراحل زیر می باشد:

- شبیه سازی Wave form (اختیاری)

- سنتز

- پیاده سازی سخت افزار

- ساخت bit stream

- خروجی برای نرم افزار SDK

در صورت انتخاب گزینه ساخت bit stream گزینه های دوم، سوم و چهارم خودکار انجام می شوند. بعد از گرفتن خروجی برای نرم افزار SDK با انتخاب گزینه launch SDK می توان وارد نرم افزار SDK شد.

۴-۶-۲-۳- کد نویسی هسته در نرم افزار SDK

طراحی مورد نیاز برای این قسمت که شامل مدیریت VDMA و هسته ساخته شده است، بسیار آسان می باشد. می توان گفت VDMA مهم ترین بلوک سخت افزاری است. کار این بلوک مدیریت انتقال داده بین بلوک های HDMI، VGA و هسته ساخته شده می باشد و بدون استفاده از هسته ARM به حافظه DDR دسترسی دارد و این امر باعث افزایش سرعت پردازش می شود. در نرم افزار SDK

همانند نرم افزار های قبلی باید یک پروژه با پایه برنامه نویسی زبان C ساخت و کتابخانه های مورد

نیاز به پروژه اضافه شوند:

```
#include "display_demo.h"
#include "xaxivdma.h"
#include "timer_ps.h"
#include "xparameters.h"
#include "Vdma.h"
#include "xscutimer.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xil_printf.h"
#include "xsdps.h"
#include "xfind.h"
#include "xgpiops.h"
```

نصب و پیکربندی هسته مورد نظر و گزارش وضعیت عملیات در خروجی:

```
XFind *coreFind;
XFind_Config *coreFindCfg;
Status = XFind_Initialize(&coreFind, FIND_ID);
xil_printf("Initialize of Core:%d\n", Status);
coreFindCfg = XFind_LookupConfig(FIND_ID);
Status = XFind_CfgInitialize(&coreFind, coreFindCfg);
xil_printf("Configure Core:%d\n", Status);
```

مقدار دهی به دو فیلتر موجود در هسته:

```
XFind_SetBl(&coreFind,0);
XFind_SetBh(&coreFind,95);
//GREEN:
XFind_SetGl(&coreFind,150);
XFind_SetGh(&coreFind,255);
//RED:
XFind_SetRl(&coreFind,0);
XFind_SetRh(&coreFind,143);
//2nd FILTER:
XFind_SetTl(&coreFind,30);
XFind_SetTh(&coreFind,255);
```

تنظیمات مربوط به اندازه تصویر خروجی و فعال کردن نهایی هسته:

```
// Enable TEST mode:
XFind_SetEn(&coreFind,1);
//////////

//Set ROW and COLUMN SIZE:
XFind_SetRows(&coreFind, 720);
XFind_SetCols(&coreFind, 1280);
//////////

//Start Core:
XFind_Start(&coreFind);
XFind_EnableAutoRestart(&coreFind);
//////////
```

راه اندازی بلوک VDMA، معرفی HDMI و VGA به بلوک VDMA و به شروع به کار پروسه

مورد نظر که در شکل ۴-۸ اشاره شده بود:

```
//Initialize VMDA HLS

Status = startVdmaHls(HLS_VDMA_ID, CfgVdmaHls, VdmaHls, StpRead, StpWrite,
hdmIPtr, vgaPtr, vgaCtrl);

if (Status != XST_SUCCESS)
{
    xil_printf("VDMA HLS Status:%d\n", Status);
}

//Configure VGA out (VDMA)

Status = DisplayDemoInitialize(&vgaCtrl, VGA_VDMA_ID, SCU_TIMER_ID,
VGA_BASEADDR, DISPLAY_NOT_HDMI, vgaPtr);

if (Status != XST_SUCCESS)
{
    xil_printf("Configure VGA out status:%d\n", Status);
}

//Initialize VMDA HDMI

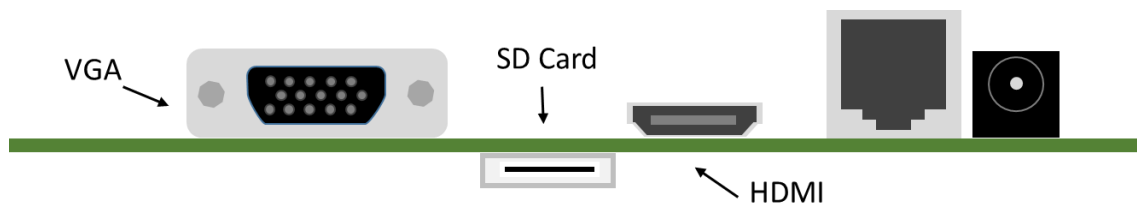
Status = startVdmaHdmI2(HDMI_VDMA_ID, CfgVdmaHdmi, VdmaHdmi, StpVdmaHdmi,
hdmIPtr, vgaCtrl);

if (Status != XST_SUCCESS)
{
    xil_printf("VDMA HDMI Status:%d\n", Status);
}
```

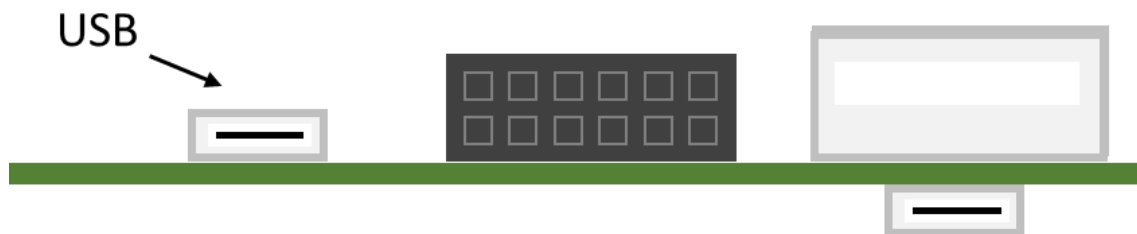
۴-۶-۲-۴- راه اندازی عملی فرآیند روی برد

در مرحله پایانی بعد از ساختن نسخه قابل بوت کد نوشته شده، فایل مربوطه در حافظه SD کارت کپی می‌شود و آماده استفاده در برد است.

شکل های زیر مربوط به دید های جانبی از برد می‌باشند. که باید ابتدا کارت حافظه SD جای گذاری شود، بعد کابل های VGA، HDMI و USB برای برقراری ارتباط UART و JTAG متصل می‌شوند. حال برد آماده استفاده می‌باشد.

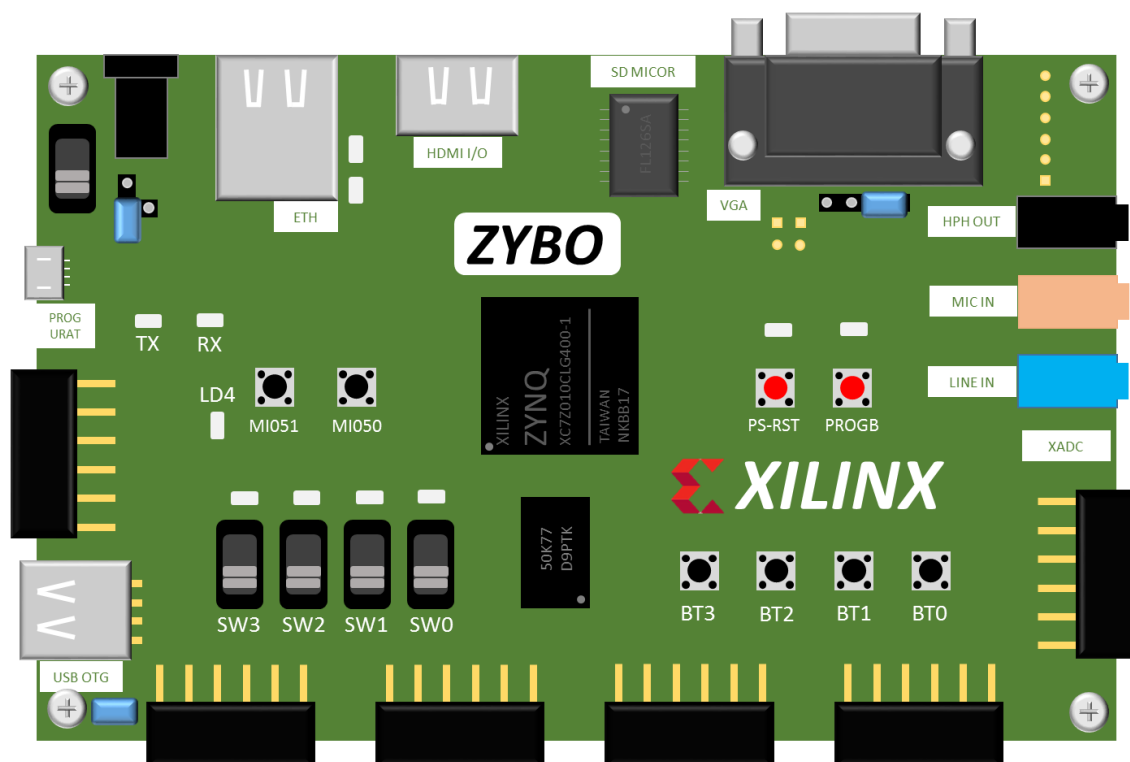


شکل (۹-۴) نمای جانبی برد



شکل (۱۰-۴) نمای جانبی برد

بعد از اتمام اتصالات، با روشن کردن بورد LED سبز رنگ به معنی راه اندازی کامل روشن می شود:



شکل (۴-۱۱) نمای بالایی مورد

فصل پنجم: نتایج به دست آمده

۵-۱- مقدمه

در فصل سوم پلتفرمهای مختلف و با قابلیت دسترسی و خرید بررسی شده‌اند که شامل انواع پردازنده با تنوع در برنامه نویسی بودند. در ادامه و در فصل چهارم الگوریتم مورد نظر معرفی شد و ابتدا در نرم افزار متلب و بعد با استفاده از کتابخانه OpenCV عمل پیاده سازی انجام پذیرفت. و در نهایت با کد نویسی در نرم افزار HLS ساخت هسته مورد نظر و با استفاده از نرم افزار های Vivado و SDK پروسه مورد نظر روی بورد انتقال یافت.

هدف از این فصل، بررسی تمامی روش های ممکن بر اساس آمار به دست آمده در هر روش و انتخاب بهترین حالت با توجه به شرایط و خواسته های متفاوت می باشد. در آغاز به طور کامل آمار

مربوط به ساخت هسته و روش استفاده از تراشه ZYNQ ارائه می‌گردد و در ادامه مقایسه کلی انجام خواهد گرفت.

۵-۲- نتایج به دست آمده از عملیات سنتز هسته

بعد از اقدام به عمل سنتز در نرم افزار HLS و پایان آن، نتایج مربوطه توسط نرم افزار ارائه می‌گردد :

۵-۲-۱- زمان بندی هسته

تراشه اصلی مورد استفاده XC7Z010CLG400-1 می‌باشد که دارای پایین‌ترین سرعت در سری ZYNQ شرکت Xilinx است و به صورت پیش‌فرض بخش PL این تراشه با فرکانس پایه ۱۰۰ مگاهرتز شروع به کار می‌کند. البته در نرم افزار Vivado و در قسمت طراحی سخت افزار می‌توان این فرکانس را به ۲۵۰ مگا هرتز رساند.

با توجه به سنتز انجام گرفته برای مقدار کلاک و مدت زمان انجام هر دوره سه مقدار ارائه شده است که حالت قطعی داشتن ۱۰ نانو ثانیه پالس می‌باشد ولی امکان کاهش این مقدار به ۸,۶۹ نانو ثانیه و داشتن سرعت بالاتر نیز امکان دارد. حالت ایده آل نیز ۱,۲۵ نانو ثانیه می‌باشد:

جدول (۵-۱) زمان‌بندی هسته

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.69	1.25

۵-۲-۲- زمان تاخیر (سیکل ساعتی)

این بخش حالتی که تاخیر در اجرا ایجاد می کنند را بررسی می کند و نتایج زمانی را ارائه می دهد.

اگر در حالت عادی هسته را سنتز بگیریم مقادیر زیر به دست می آیند:

جدول (۵-۲) تاخیر در خروجی

Latency		Interval		Type
min	max	min	max	
28	5534655	29	5534656	none

همان طور که از نتایج مشخص است، این حالت اصلا خوب نیست زیرا در بدترین حالت امکان

دارد بالای ۵ میلیون بار تاخیر داشته باشیم پس با استفاده از حالت Data Flow سعی بر کاهش این

تاخیرات می شود:

جدول (۵-۳) تاخیر بعد از اعمال حالت Data Flow

Latency		Interval		Type
min	max	min	max	
30	2769860	31	2769861	DataFlow

با اعمال حالت گفته شده تاخیرات به ۲ میلیون کاهش یافت ولی باز به اندازه کافی بهینه نمی‌باشد. البته علت اصلی این امر استفاده از حلقه در هسته می‌باشد. پس باید از روشی استفاده کرد تا تعداد دفعات تکرار این حلقه را کاهش داد. یکی از این روش‌ها استفاده از حالت PipeLine می‌باشد. بعد از اعمال این گزینه مقادیر زیر به دست می‌آید:

جدول (۵-۴) تاخیر بعد از اعمال دو حالت بهینه‌سازی

Latency		Interval		Type
min	max	min	max	
17	924483	8	924484	DataFlow_& PipeLine

نتایج بعد از آخرین تغییرات نشان دهنده بهینه سازی بسیار خوب در هسته می‌باشد. زیر تعداد تاخیرات تقریباً برابر با تعداد پیکسل تصویر مورد نظر می‌باشد.

۵-۲-۳- پیش بینی شرایط بعد از پیاده سازی

این قسمت بیشتر تلاش دارد تا تعداد قطعات استفاده شده برای سنتز را نمایش دهد. یعنی به چه تعداد LUT، FF، DSP و... برای پیاده سازی الگوریتم مورد نظر استفاده شده است:

جدول (۵-۵) درصد میزان مصرف قسمت‌های مختلف

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2
FIFO	0	-	50	296
Instance	-	5	1217	3311
Memory	-	-	-	-
Multiplexer	-	-	-	8
Register	-	-	10	-
Total	0	5	1277	3617
Available	120	80	35200	17600
Utilization (%)	0	6	3	20

با توجه به جدول بالا بیشترین قسمت استفاده شده جداول صحت یا LUT می‌باشد، البته آن در

مقایسه نسبت به کل فضای موجود فقط بیست درصد از کل است. حال بهتر است گزینه‌ی Instance

که بیشترین مصرف از سخت افزار را دارد را بررسی شود:

جدول (۶-۵) آمار استفاده از قطعات

Instance	Module	BRAM_18K	DSP_48E	FF	LUT
find_AXIvideo2Mat_U0	find_AXIvideo2Mat	0	0	234	261
find_Block_proc1_U0	find_Block_proc1	0	4	231	354
find_HLS_FIND_OBJ_core_cpp_line11_U0	find_HLS_FIND_OBJ_core_cpp_line11	0	0	34	33
find_Loop_1_proc_U0	find_Loop_1_proc	0	1	676	2594
find_Mat2AXIvideo_U0	find_Mat2AXIvideo	0	0	42	69
Total	5	0	5	1217	3311

همان گونه که پیش بینی می شد، پیاده سازی سخت افزاری حلقه موجود در هسته بیشترین آمار را در بر می گیرد.

جدول دیگری که باید بررسی شود، جدول نتایج استفاده از FIFO می باشد:

جدول (۷-۵) FIFO

Name	BRAM_18K	FF	LUT	Depth	Bits	Size:D*B
Pos_loc_channel_U	0	0	0	2	32	64
TCH_loc_channel_U	0	5	44	2	32	64
TCL_loc_channel_U	0	5	44	2	32	64
TRH_loc_channel_U	0	5	44	2	32	64
TRL_loc_channel_U	0	5	44	2	32	64
img_0_data_stream_0_V_U	0	5	20	1	8	8
img_0_data_stream_1_V_U	0	5	20	1	8	8
img_0_data_stream_2_V_U	0	5	20	1	8	8
img_1_data_stream_0_V_U	0	5	20	1	8	8
img_1_data_stream_1_V_U	0	5	20	1	8	8
img_1_data_stream_2_V_U	0	5	20	1	8	8
Total	0	50	296	16	208	368

بخشی از FPGA که برای انتقال داده از قسمتی به قسمت دیگر به کار گرفته می شود. یک ساختار نگهداری داده ها که دسترسی به آنها با شرایط خروجی به ترتیب ورودی می باشد.

۵-۲-۴- شرایط ورودی و خروجی ها

جدول زیر هم تعداد و اطلاعات مربوط به ورودی و خروجی ها را نمایش می دهد. چون ارسال و دریافت فریم ها با استاندارد AXI می باشد به همراه داده های تصویر باید سیگنال هایی برای بیان نحوه و وضعیت ارسال وجود داشته باشد که شامل سیگنال های TSTRB، TKEEP، TDATA، TREADY و TVALID، TDEST، TID، TLAST، TUSER می باشد:

جدول (۸-۵) ورودی خروجی هسته

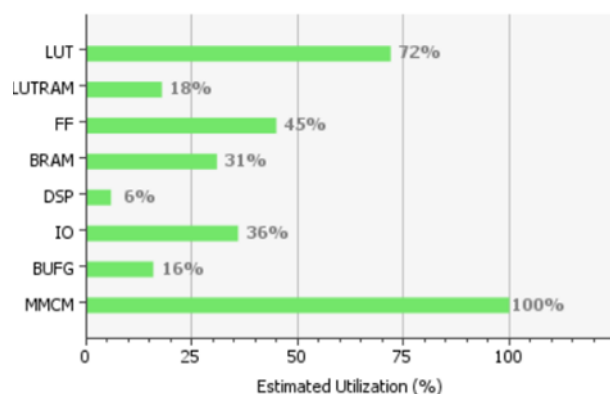
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
instream_TDATA	in	32	axis	instream_V_data_V	pointer
instream_TKEEP	in	4	axis	instream_V_keep_V	pointer
instream_TSTRB	in	4	axis	instream_V_strb_V	pointer
instream_TUSER	in	1	axis	instream_V_user_V	pointer
instream_TLAST	in	1	axis	instream_V_last_V	pointer
instream_TID	in	1	axis	instream_V_id_V	pointer
instream_TDEST	in	1	axis	instream_V_dest_V	pointer
instream_TVALID	in	1	axis	instream_V_dest_V	pointer
instream_TREADY	out	1	axis	instream_V_dest_V	pointer
outstream_TDATA	out	32	axis	outstream_V_data_V	pointer
outstream_TKEEP	out	4	axis	outstream_V_keep_V	pointer
outstream_TSTRB	out	4	axis	outstream_V_strb_V	pointer
outstream_TUSER	out	1	axis	outstream_V_user_V	pointer
outstream_TLAST	out	1	axis	outstream_V_last_V	pointer
outstream_TID	out	1	axis	outstream_V_id_V	pointer
outstream_TDEST	out	1	axis	outstream_V_dest_V	pointer
outstream_TVALID	out	1	axis	outstream_V_dest_V	pointer
outstream_TREADY	in	1	axis	outstream_V_dest_V	pointer
rows	in	32	ap_stable	rows	scalar
cols	in	32	ap_stable	cols	scalar
TL	in	32	ap_stable	TL	scalar
TH	in	32	ap_stable	TH	scalar
GL	in	32	ap_stable	GL	scalar
GH	in	32	ap_stable	GH	scalar
BL	in	32	ap_stable	BL	scalar
BH	in	32	ap_stable	BH	scalar
RL	in	32	ap_stable	RL	scalar
RH	in	32	ap_stable	RH	scalar
EN	in	32	ap_stable	EN	scalar
ap_clk	in	1	ap_ctrl_hs	find	return value
ap_rst_n	in	1	ap_ctrl_hs	find	return value
ap_start	in	1	ap_ctrl_hs	find	return value
ap_done	out	1	ap_ctrl_hs	find	return value
ap_return	out	32	ap_ctrl_hs	find	return value
ap_idle	out	1	ap_ctrl_hs	find	return value
ap_ready	out	1	ap_ctrl_hs	find	return valu

۳-۵- نتایج به دست آمده از عملیات پیاده سازی سخت افزار و ساخت Bit Stream

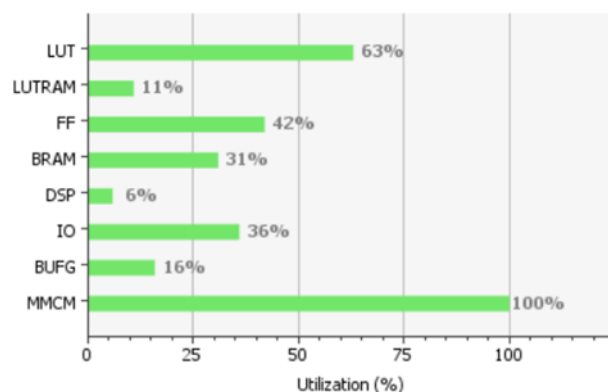
بعد از انجام پیاده سازی سخت افزاری بلوکهای زیادی همچون VGA، HDMI، VDMA و

Find به قسمت PL اضافه می شوند پس میزان مصرف از PL افزایش می یابد. جدول زیر درصد های

دقیق تعداد قطعات استفاده شده را نشان می دهد:



نمودار (۱-۵) میزان مصرف قطعات اولیه موجود در تراشه (تخمین)



نمودار (۲-۵) میزان مصرف قطعات اولیه موجود در تراشه (بعد از پیاده سازی)

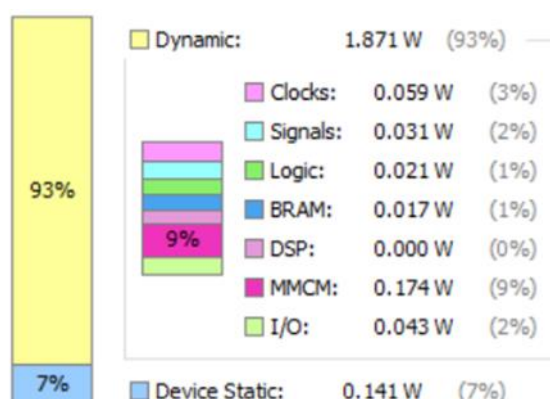
همانطوری که از نتایج پیداست مقادیر تخمین زده شده در بعضی از موارد بیشتر از مقادیر واقعی شده اند.

۵-۳-۱- توان مصرفی

حال توان مصرفی و دمای داخلی تراشه بررسی می‌شود، در سری‌های ULTRA Scale بلوکی به نام MPSoCs می‌باشد که وظیفه کنترل و مدیریت توان مصرفی را بر عهده دارد. و در سری‌های دیگر می‌توان با توجه به شرایط در نرم‌افزار Vivado مقدار ماکزیمم توان مورد نظر را تعیین نمود تا پردازنده با توجه به آن فعالیت خود را کنترل کند.

جدول (۵-۹) میزان مصرف توان تراشه

پارامتر	مقدار
بیشترین توان مصرفی تراشه	۲,۰۱۲ وات
دمای اتصالات	۴۸,۲ سانتی گراد



نمودار (۵-۳) میزان مصرف توان

در شکل گرافیکی بالا هم میزان مصرفی قسمت های مختلف بیان شده است. البته کل توان مصرف مقایسه سرعت اجرا بین روش های موجودی خیلی پایین می باشد و می توان گفت کمترین توان مصرفی بین پلتفرم های مختلف را داراست.

۵-۴- مقایسه سرعت اجرا بین روش های موجود

در این قسمت سرعت اجرای الگوریتم مورد نظر در سه روش پیاده سازی با استفاده از متلب، OpenCV و تراشه ZYNQ بررسی خواهد شد. برای تست الگوریتم در هر روش دو نوع ورودی وجود دارد:

- یک فریم به اندازه 720×1280 پیکسل

- ویدئو ۲۰۰ فریمی با کیفیت 720p

۵-۴-۱- نتایج نرم افزار متلب

جدول (۵-۱۰) زمان اجرا الگوریتم در نرم افزار متلب

نوع تست	نمایش نتایج در خروجی	زمان اجرا (ثانیه)
یک فریم	فعال	۰,۵۴۶۹۲۱
	غیر فعال	۰,۱۹۰۷۶۲
ویدئو (۲۰۰ فریم)	فعال	۴۸,۲۲۹۶۰۱
	غیر فعال	۲۸,۸۱۹۷۹۷

- نحوه اندازه گیری زمان اجرا در هر قسمت با استفاده از دستور tic toc انجام گرفته است.

جدول (۵-۱۱) زمان اجرا الگوریتم در کتابخانه OpenCV

نوع تست	نمایش نتایج در خروجی	حالت نرم افزار	زمان اجرا (ثانیه)
یک فریم	فعال	Release	۰,۳۳۱۷۷۸
	غیر فعال	Release	۰,۰۳۸۰۸۸
ویدئو (۲۰۰ فریم)	فعال	Release	۴,۹۹۲۷۲
	غیر فعال	Release	۲,۷۸۸۲۸

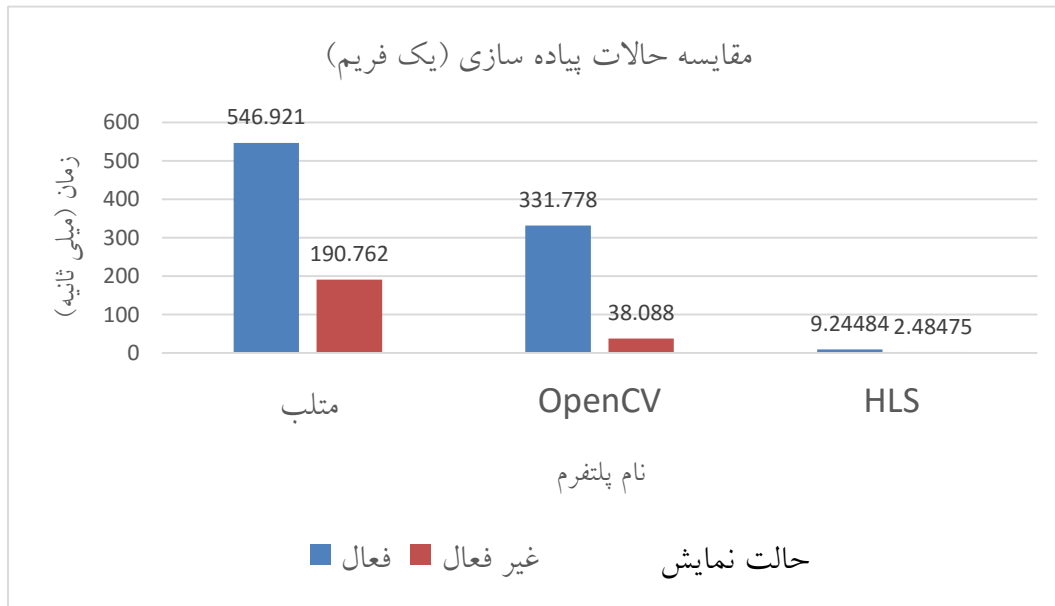
• نحوه اندازه گیری زمان اجرا با استفاده از دستورات get_Tick_Count صورت گرفته است.

جدول (۵-۱۲) زمان اجرا الگوریتم در نرم افزار HLS

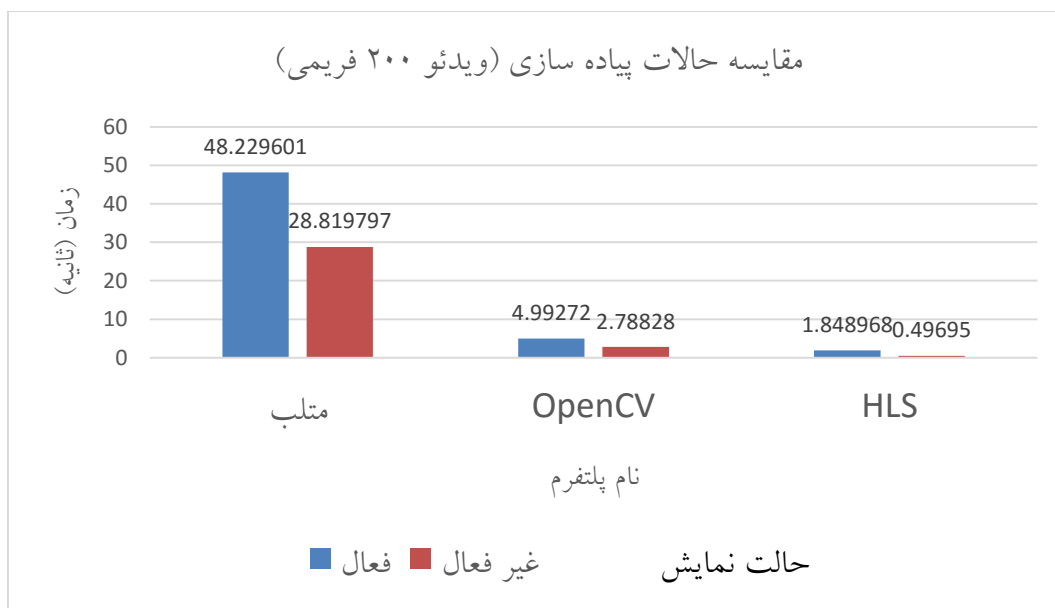
نوع تست	نمایش نتایج در خروجی	زمان اجرا (ثانیه)
یک فریم	فعال	۰,۰۰۹۲۴۴۸۴
	غیر فعال	۰,۰۰۲۴۸۴۷۵
ویدئو (۲۰۰ فریم)	فعال	۱,۸۴۸۹۶۸
	غیر فعال	۰,۴۹۶۹۵

• نحوه اندازه گیری زمان اجرا با استفاده از نتایج خروجی سنتز صورت گرفته است.

از نتایج به دست آمده می‌توان به راحتی گفت که نتایج پیاده سازی در تراشه ZYNQ خیلی بهتر از دیگر حالات می‌باشد و سرعت انجام محاسبات به شدت بالا است:



نمودار (۴-۵) مقایسه حالات پیاده سازی (یک فریم)



نمودار (۵-۵) مقایسه حالات پیاده سازی (ویدئو ۲۰۰ فریمی)

نتیجه گیری

با مقایسه‌های انجام شده می‌توان به این نتیجه رسید که پلتفرم ZYNQ-7000 SoC برای برآورده ساختن هدف این پژوهش بهترین انتخاب می‌باشد. از مزیت‌های استفاده از این تراشه می‌توان به سرعت فوق العاده بالا، دقت زیاد، توان مصرفی کم، امنیت خوب، قیمت مناسب سخت‌افزار، پشتیبانی از پروتکل‌های ارتباطی مختلف و کارکرد به صورت مستقل اشاره کرد. و معایت بکارگیری این تراشه، قیمت بالای لایسنس نرم افزارهای مورد نیاز، فروشی بودن هسته‌های مورد استفاده، عدم دسترسی آسان به منابع نرم‌افزاری و سخت‌افزاری به دلایل نظامی و سیاسی، نوپا بودن مبحث بررسی شده و کمبود اطلاعات مورد نیاز می‌باشد.

با توجه به معایت و مزایای بیان شده، پیشنهاد می‌شود از این پلتفرم در مواردی که احتیاج به سرعت بالا در کنار دقت و امنیت زیاد است، استفاده شود ولی در شرایطی که سرعت جز فاکتورهای اساسی مورد نظر نیست، بهتر است از سیستم‌های متداول دیگر بهره برد.

مراجع

- [1] Rosenfeld, A. 1969. Picture Processing by Computer. New York: Academic Press.
- [2] Bradski, G. 2008. Learning OpenCV: Computer Vision with the OpenCV Library. Boston: O'Reilly Publication.
- [۳] پایان نامه کارشناسی ارشد پردازش تصویر، دانشگاه آزاد واحد جنوب تهران، مهدی کوهی
- [4] Prateek Joshi, David Millán Escrivá, Vinícius Godoy. 2016. OpenCV by example. Packt Publication.
- [5] Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz, Robert W. Stewart. 2014. The Zynq Book: Embedded Processing With the ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 All Programmable SoC. Strathclyde Academic Media.
- [6] Kapela, R. 2015. Embedded platform for local image descriptor based object detection. Applied Mathematics and Computation. Elsevier. Volume 267. p. 419-427.
- [7] Rafael C. Gonzalez, Richard Eugene Woods. 2008. Digital Image Processing. Prentice Hall.
- [8] Mikolajczyk, K. and C. Schmid. 2003. A performance evaluation of local descriptors. in International Conference on Computer Vision & Pattern Recognition (CVPR'03).
- [9] Mikolajczyk, K. and C. Schmid, 2005. A performance evaluation of local descriptors. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 27(10): p. 1615-1630.
- [10] Stauffer, C. and W.E.L. Grimson, 2000. Learning patterns of activity using real-time tracking. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 22(8): p. 747-757.
- [11] Zhang, R. and J. Ding. 2012. Object Tracking and Detecting Based on Adaptive Background Subtraction. Procedia Engineering. 29: p. 1351-1355.

[12] Chunsheng, H., et al., 2007. Object tracking with target and background samples. IEICE transactions on information and systems. **90**(4): p. 766-774.

[13] Hua, C., et al. . 2008. K-means Clustering Based Pixel-wise Object Tracking. IPSJ Online Transactions 1. p. 66-79

[۱۴] ردیابی دقیق اشیاء متحرک بر اساس اطلاعات حرکت و الگوریتم k_means اتوماتیک، بیستمین کنفرانس ملی سالانه انجمن کامپیوتر ایران ، دانشگاه فردوسی مشهد سال ۱۳۹۳، عزیز کرمانی

[15] Jason Clemons. SIFT: SCALE INVARIANT FEATURE TRANSFORM BY DAVID LOWE. from http://web.eecs.umich.edu/~silvio/teaching/EECS598/lectures/lecture10_1.pdf [Accessed 14 August 2016]

[16] ZHAW Winterthur. 2012. Zynq-7000 – The new embedded processing platform. Embedded Computing Conference. from http://embeddedcomputingconference.ch/pdf_2012/1A2_Heimlicher.pdf [Accessed 14 August 2016]

[17] UG821 (v12.0) September 30, 2015. Zynq-7000 All Programmable SoC Software Developers Guide. from http://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf [Accessed 14 August 2016]

[18] UG761 (v13.1) March 7, 2011. AXI Reference Guide. from http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf [Accessed 14 August 2016]

[19] Member List. Xilinx company website from <http://www.origin.xilinx.com/alliance/memberlocator.html> [Accessed 14 August 2016]

[20] Mark Will. 2013. Real-Time Image Processing. The University of Waikato. from

http://markwill.me/wp-content/uploads/2014/06/honours_thesis_compressed.pdf [Accessed 14 August 2016]

[21] XMP100 (v1.0), 2014. Low-end-portfolio-product-selection-guide . from

<http://www.xilinx.com/support/documentation/selection-guides/low-end-portfolio-product-selection-guide.pdf> [Accessed 14 August 2016]

[22] The Vivado Design Suite accelerates programmable systems integration and implementation by up to 4X. from

<http://www.edn.com/electronics-products/other/4375467/The-Vivado-Design-Suite-accelerates-programmable-systems-integration-by-up-to-4X> [Accessed 14 August 2016]

[23] Brian Bailey, EE Times. Second generation for FPGA software.

<http://www.eetimes.com/electronics-products/ip-eda-products/4371701/Second-generation-for-FPGA-software> [Accessed 14 August 2016]

[24] Vivado High-Level Synthesis. Xilinx company website. from

<http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html> [Accessed 14 August 2016]

[25] Xilinx Software Development Kit (XSDK). Xilinx company website. from

<http://www.xilinx.com/products/design-tools/embedded-software/sdk.html> [Accessed 14 August 2016]