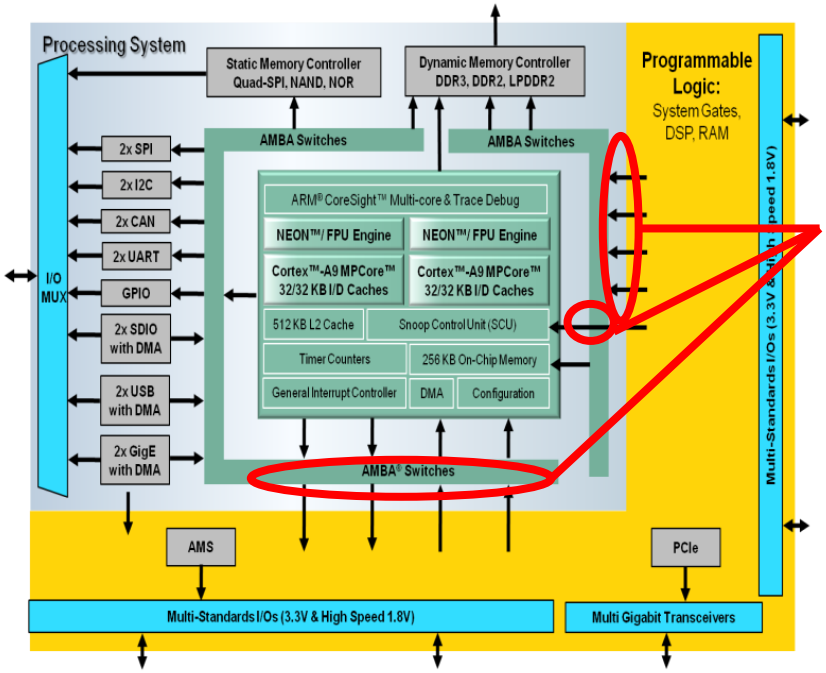
How to Accelerate OpenCV Applications with the Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries

Dec 19, 2013

ZYNQ Overview

- **PS (Dual ARM Cortex™-A9 MPCore™)**
  - Operation clock : 776MHz @speed -2
  - NEON FPU/Vector SP/DP FP
  - Hard macro Peripherals

- **PL (Artix7/Kintex 7)**
  - 28nm HPL process
  - GTX 10.3125Gbps@speed -2



- **AXI Interconnect btw PS and PL**
  - ACP
  - HP Port
  - GP Port

- **Reduce BOM Cost – All-in one solution**
  - Reduce System total power
  - Reduce PCB Layer

Vivado High-Level Synthesis (HLS)

C, C++ or SystemC

VHDL or Verilog

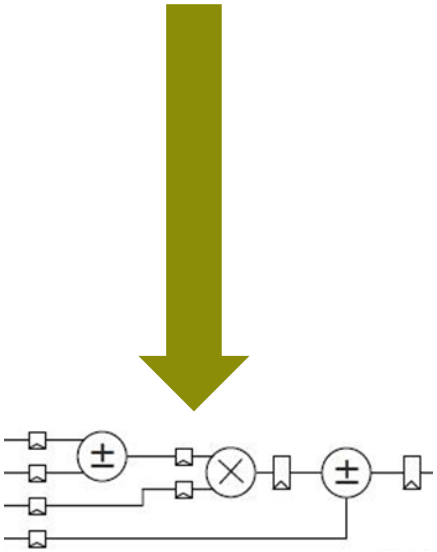
System IP Integration

```
#include "fir.h"

void fir ( data_t *y, coef_t c[N], data_t x )

{
  static data_t shift_reg[N];
  acc_t acc;
  int i;

  acc=0;
  Shift_Accum_Loop: for (i=N-1;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
          shift_reg[i]=shift_reg[i-1];
          acc+=shift_reg[i]*c[i];
    }
  }
  *y=acc;
}
```



Algorithmic Specification

Micro Architecture Exploration

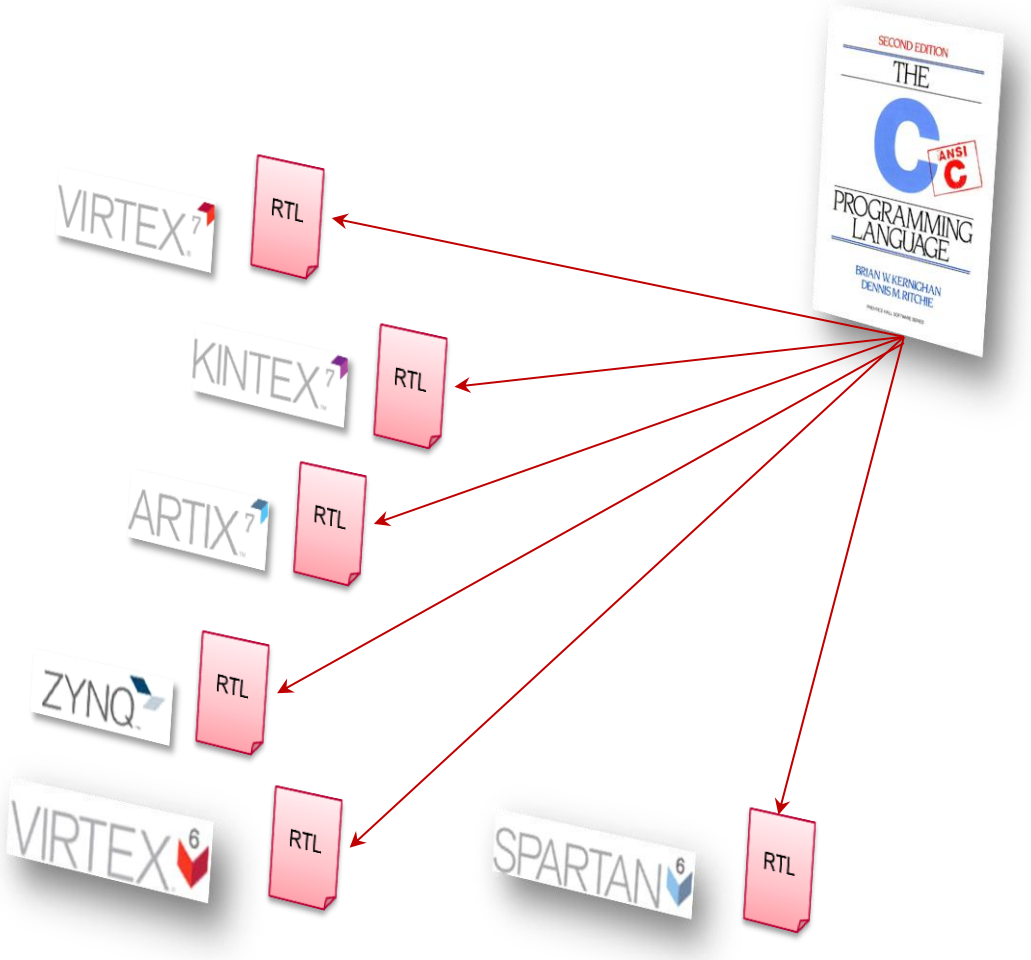
RTL Implementation

Comprehensive Integration with the Xilinx Design Environment

Accelerates Algorithmic C to RTL IP integration

Benefits

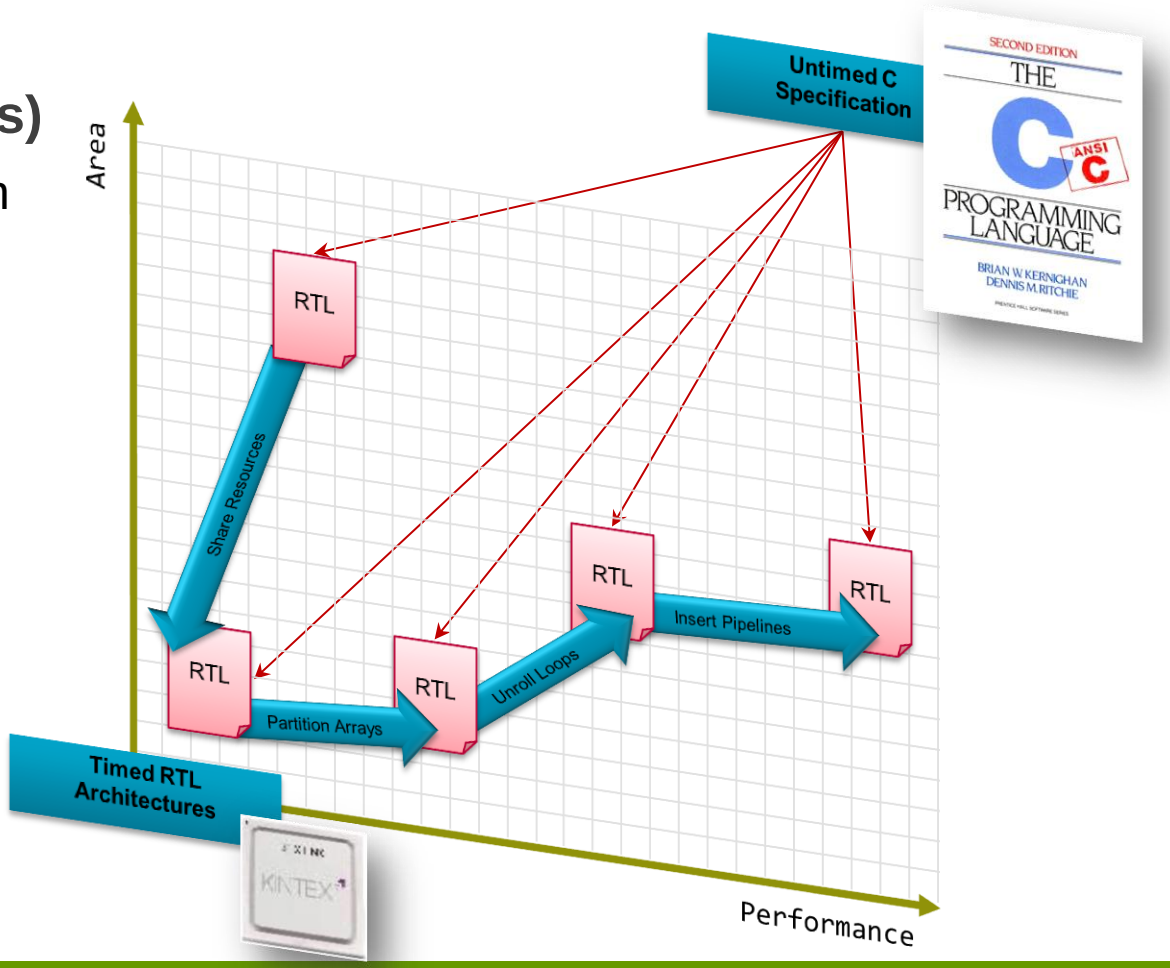
- **Portability/Design Reuse**
  - Technology migration
  - Cost reduction
  - Full Resource/ Performance/Power analysis



Design and IP reuse

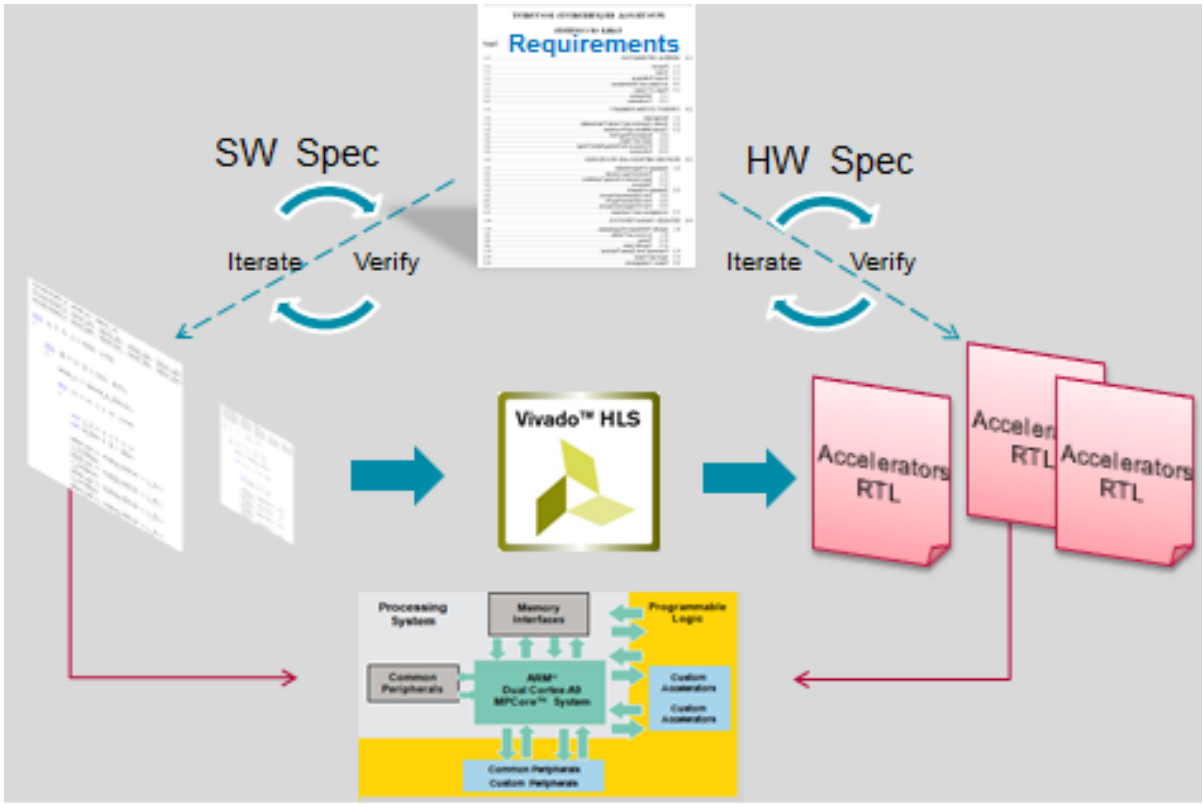
Benefits

- QoR (Quality of Results)
  - Architecture Exploration
    - Timing
      - Parallelization
      - Pipelining
    - Resources
      - Sharing
  - Heuristics



Rapid design exploration delivers QoR rivaling hand-coded RTL

Vivado High-Level Synthesis (HLS)



Accelerate Algorithmic C to Co-Processing Accelerator Integration

OpenCV Overview

- **Open Source Computer Vision (OpenCV) is widely used to develop Computer Vision applications**
  - Library of 2500+ optimized video functions
  - Optimized for desktop processors and GPUs
  - Tens of thousands users
  - Runs out of the box on ARM processors in Zynq

- **However**
  - HD processing with OpenCV is often limited by external memory
  - Memory bandwidth is a bottleneck for performance
  - Memory accesses limit power efficiency

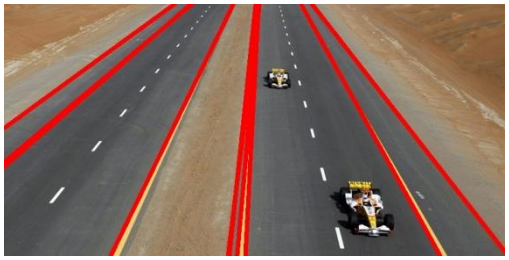
- **Zynq All-programmable SOCs are a great way of implementing embedded computer vision applications**
  - High performance and Low Power

Real-Time Computer Vision Applications

Computer Vision Applications		Real-time Analytics Function
Advanced Drivers Assist for Safety		Lane or Pedestrian detection
Surveillance for Security		Friend vs Foe recognition
Machine Vision for Quality		High velocity object detection
Medical Imaging For non invasive surgery		Tumor detection

Image Processing Example: Driver Assistance

- Analyze a video frame to detect road lane markings



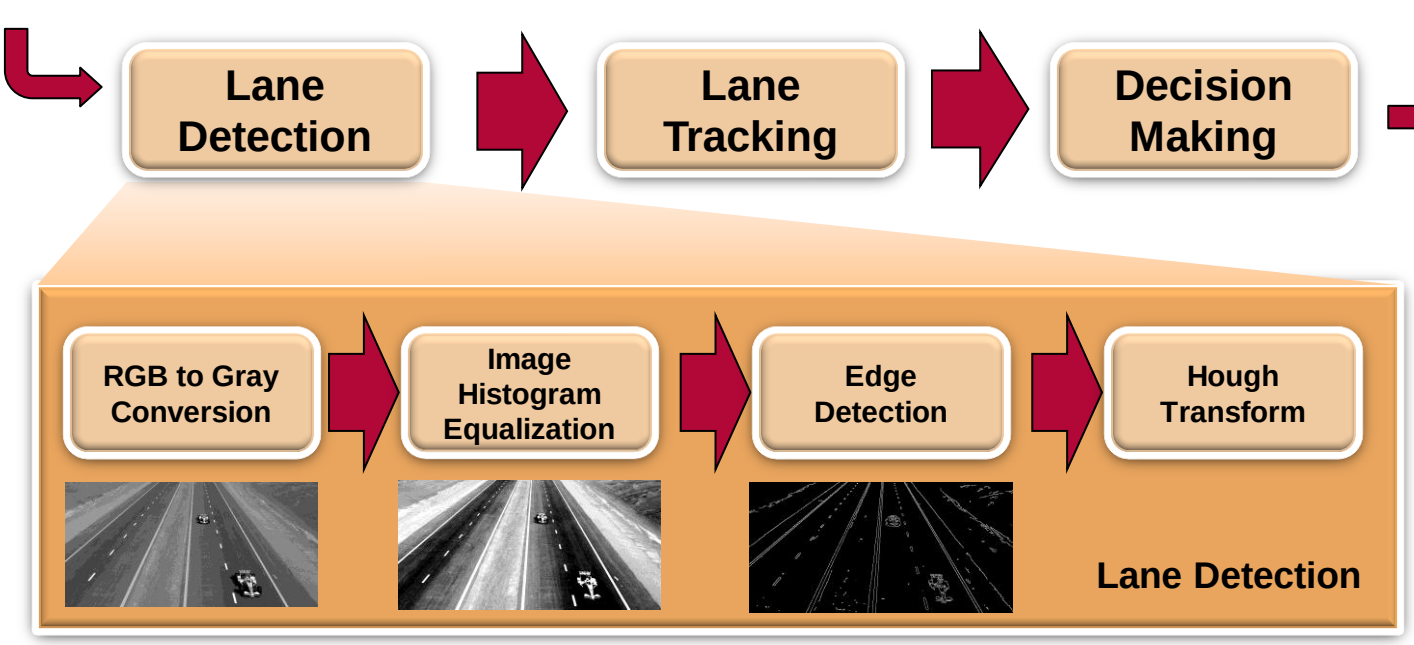
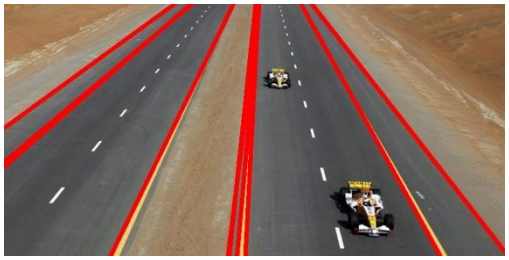


Image Processing Example: Driver Assistance

- Analyze a video frame to detect road lane markings



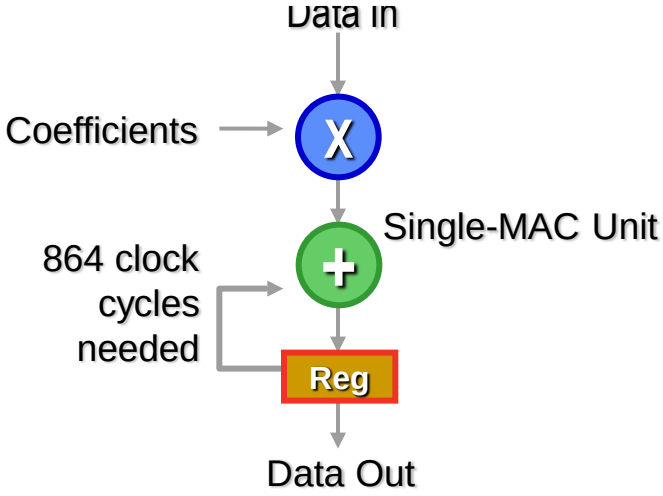
Processing Rate for Real-Time Execution

Requires Hardware	Can be Performed in Software
Pixel Processing	Data or Frame Processing

Optimal Solution is a Mix of Hardware and Software

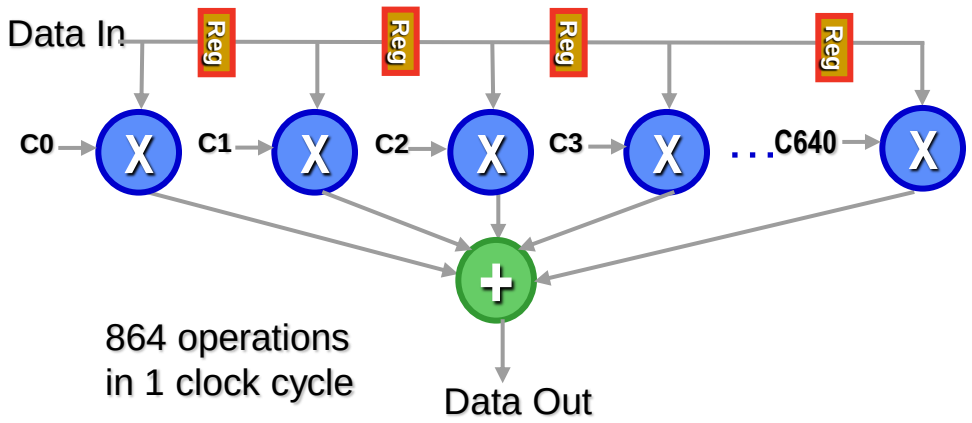
FPGAs allow massively parallel DSP processing

Standard DSP processor – Sequential (generic DSP)



1.2 GHz / 864 clock cycles = 1.38 MSPS

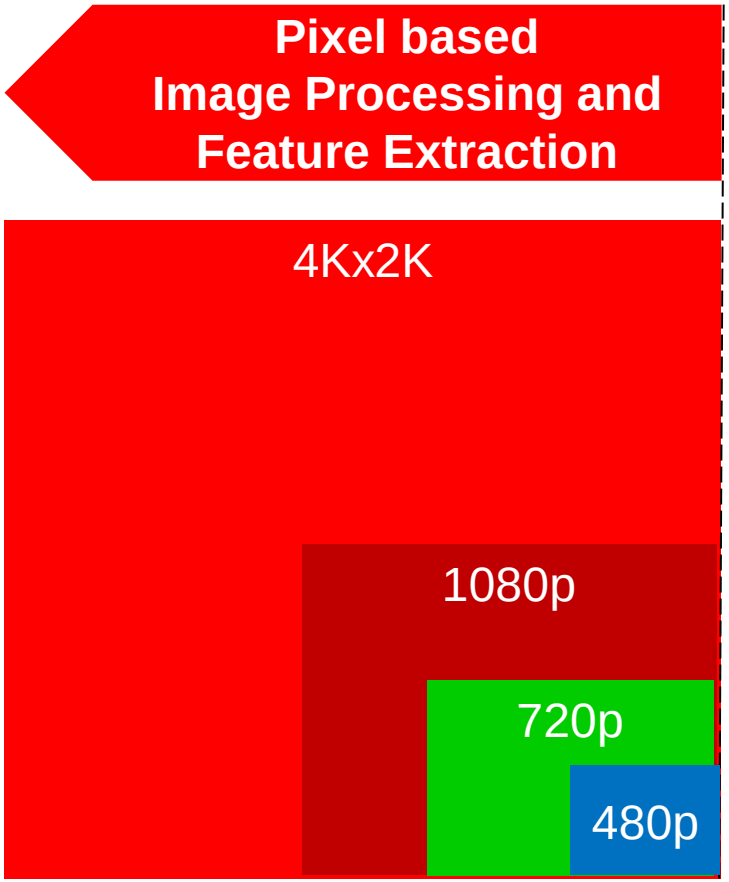
FPGA - Fully Parallel Implementation



600 MHz / 1 clock cycle = 600 MSPS

A 864-tap filter implementation is up to 432 times faster

Real-time Video Analytics Processing



FullHD
1920x1080 ≈ 2K x 1K = 2M pixel

60 frame/sec

60 x 2M = 120M pixel / sec

100 operations / pixel

100x120M = 12G operations/sec

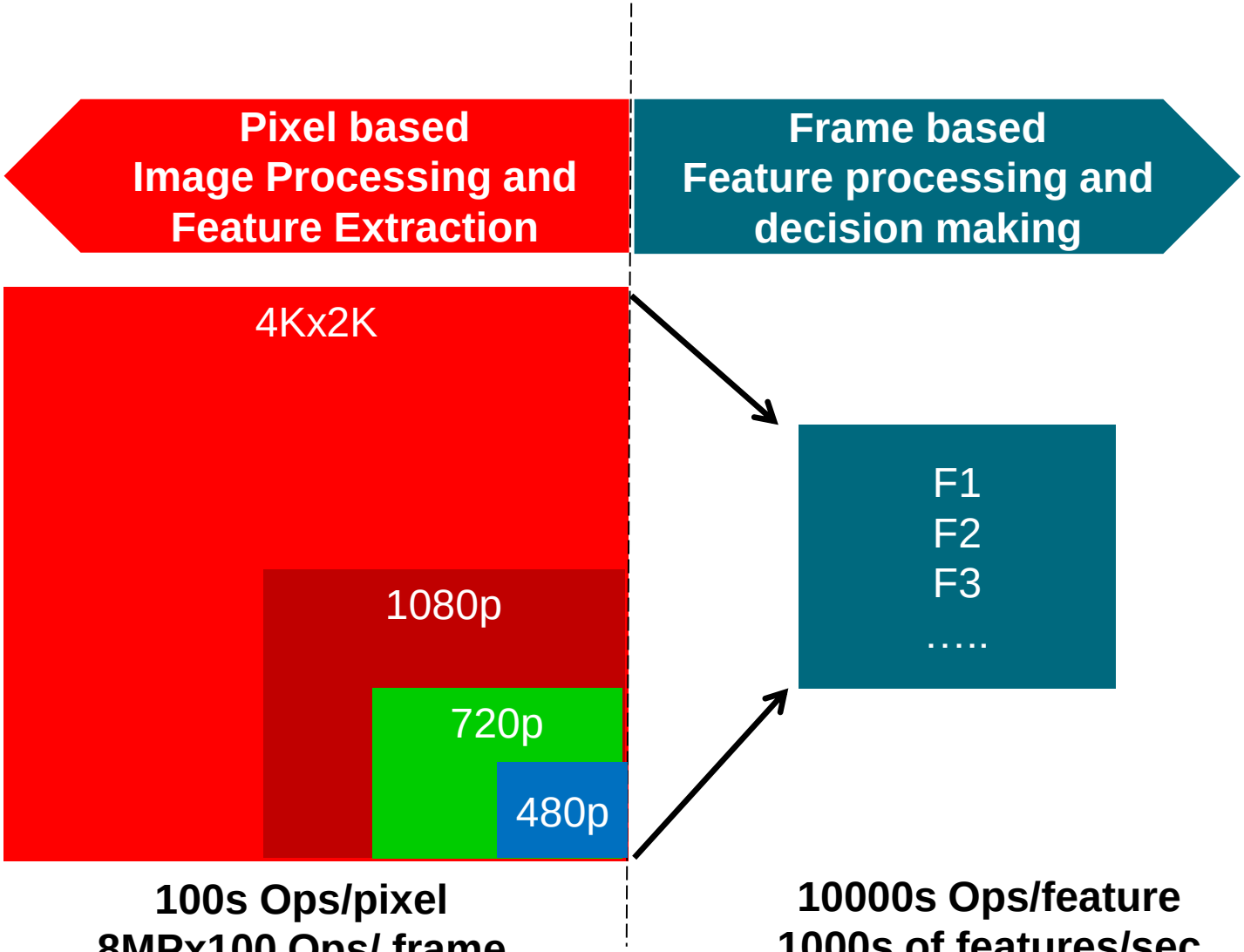
Typical embedded processor
1G ~ 10G ops

500 operations / pixel

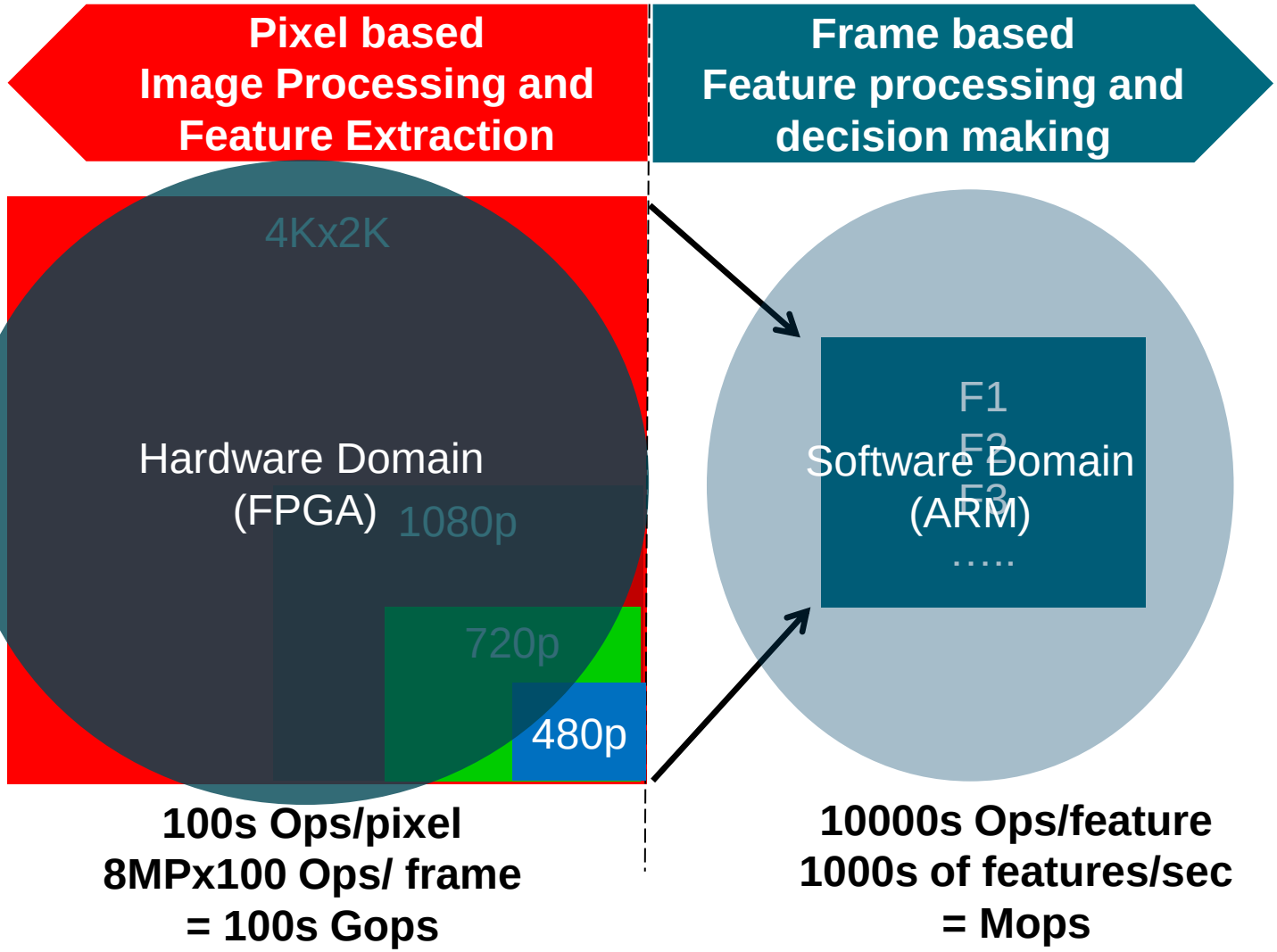
500x120M = 60G operations/sec

UHD
3840x2160 ≈ 4K x 2K = 8M pixel
7680x4320 ≈ 8K x 4K = 32M pixel

Real-time Video Analytics Processing



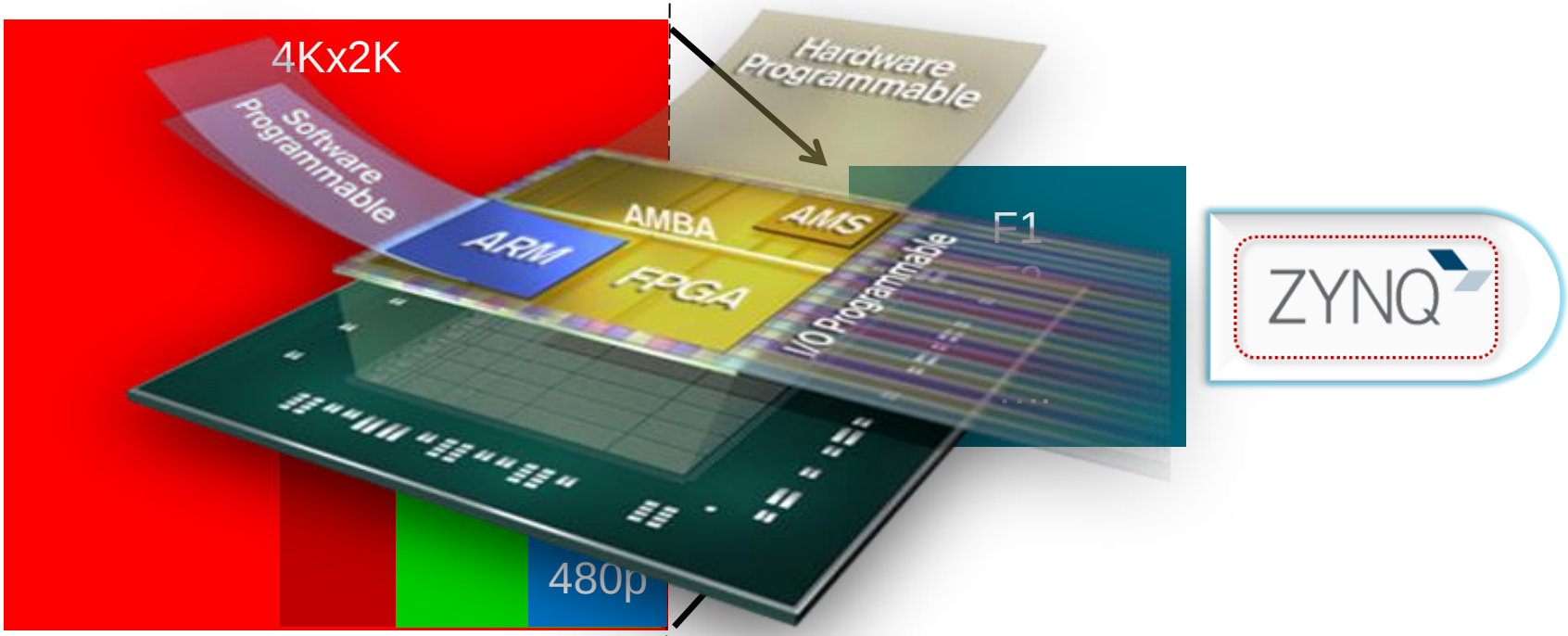
Heterogeneous Implementation of Real-time Video Analytics



Xilinx Real-time Image Analytics Implementation: Zynq All Programmable SoC

Pixel based
Image Processing and
Feature Extraction

Frame based
Feature processing and
decision making



100s Ops/pixel
8MPx100 Ops/ frame
= 100s Gops

10000s Ops/feature
1000s of features/sec
= Mops

Accelerating OpenCV Applications

Driver Assist

HD Surveillance

Video Conferencing

Studio Cinema Camera

Office-class MFP

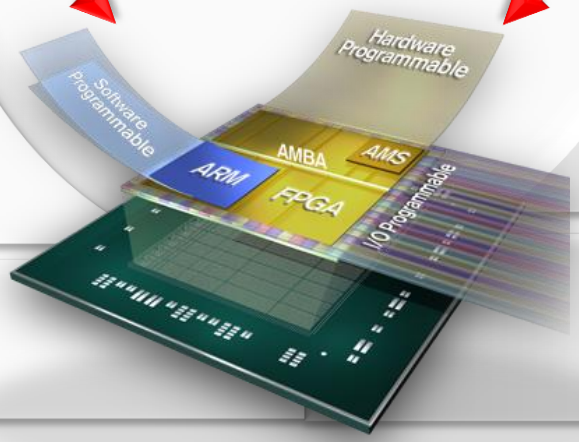
Machine Vision

OpenCV

Frame-level processing Library for PS

Pixel processing interfaces and basic functions for analytics

Vivado HLS



Broadcast Monitor

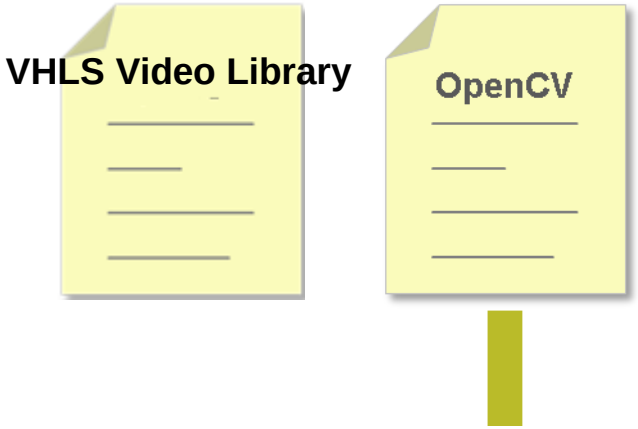
Cinema Projection

Digital Signage

Consumer Displays

Medical Displays

Vivado: Productivity gains for OpenCV functions



- C simulation of HD video algorithm ~1 fps
  - RTL simulation of HD video 1 frame per hour
- Real-time FPGA implementation up to 60fps

Video Library Functions

- C++ code contained in `hls` namespace.
  - `#include "hls_video.h" & "hls_opencv.h"`
- Similar interface, equivalent behavior with OpenCV, e.g.
  - OpenCV library:
  - HLS video library:

```
cvScale(src, dst, scale, shift);
```

```
hls::Scale<...>(src, dst, scale, shift);
```

- Some constructor arguments have corresponding or replacement template parameters, e.g.
  - OpenCV library:
  - HLS video library:

```
cv::Mat mat(rows, cols, CV_8UC3);
```

```
hls::Mat<ROWS, COLS, HLS_8UC3> mat(rows, cols);
```

- ROWS and COLS specify the maximum size of an image processed

Video Library Core Structures

OpenCV	HLS Video Library
`cv::Point_<T>, CvPoint`	`hls::Point_<T>, hls::Point`
`cv::Size_<T>, CvSize`	`hls::Size_<T>, hls::Size`
`cv::Rect_<T>, CvRect`	`hls::Rect_<T>, hls::Rect`
`cv::Scalar_<T>, CvScalar`	`hls::Scalar<N, T>`
`cv::Mat, IplImage, CvMat`	`hls::Mat<ROWS, COLS, T>`
`cv::Mat mat(rows, cols, CV_8UC3);`	`hls::Mat<ROWS, COLS, HLS_8UC3> mat (rows, cols);`
`IplImage* img = cvCreateImage(cvSize(cols,rows), IPL_DEPTH_8U, 3);`	`hls::Mat<ROWS, COLS, HLS_8UC3> img, (rows, cols);`
	`hls::Mat<ROWS, COLS, HLS_8UC3> img;`
	`hls::Window<ROWS, COLS, T>`
	`hls::LineBuffer<ROWS, COLS, T>`

HLS Video Libraries

- **OpenCV functions are not directly synthesizable with HLS**
  - Dynamic memory allocation
  - Floating point
  - Assumes images are modified in external memory
- **The HLS video library is intended to replace many basic OpenCV functions**
  - Similar interfaces and algorithms to OpenCV
  - Focus on image processing functions implemented in FPGA fabric
  - Includes FPGA-specific optimizations
    - Fixed point operations instead of floating point
    - On-chip Linebuffers and window buffers
  - Not necessarily bit-accurate
- **HLS Video Libaries**
  - Data Type - Memory Line Buffer
  - Memory Window - Video Functions

Xilinx HLS Video Library 2013.2

`Hls::Mat <-> AXI-4 Stream`

Video Data Modeling	
Linebuffer class	Window class

AXI4-Stream IO Functions	
AXIvideo2Mat	Mat2AXIvideo

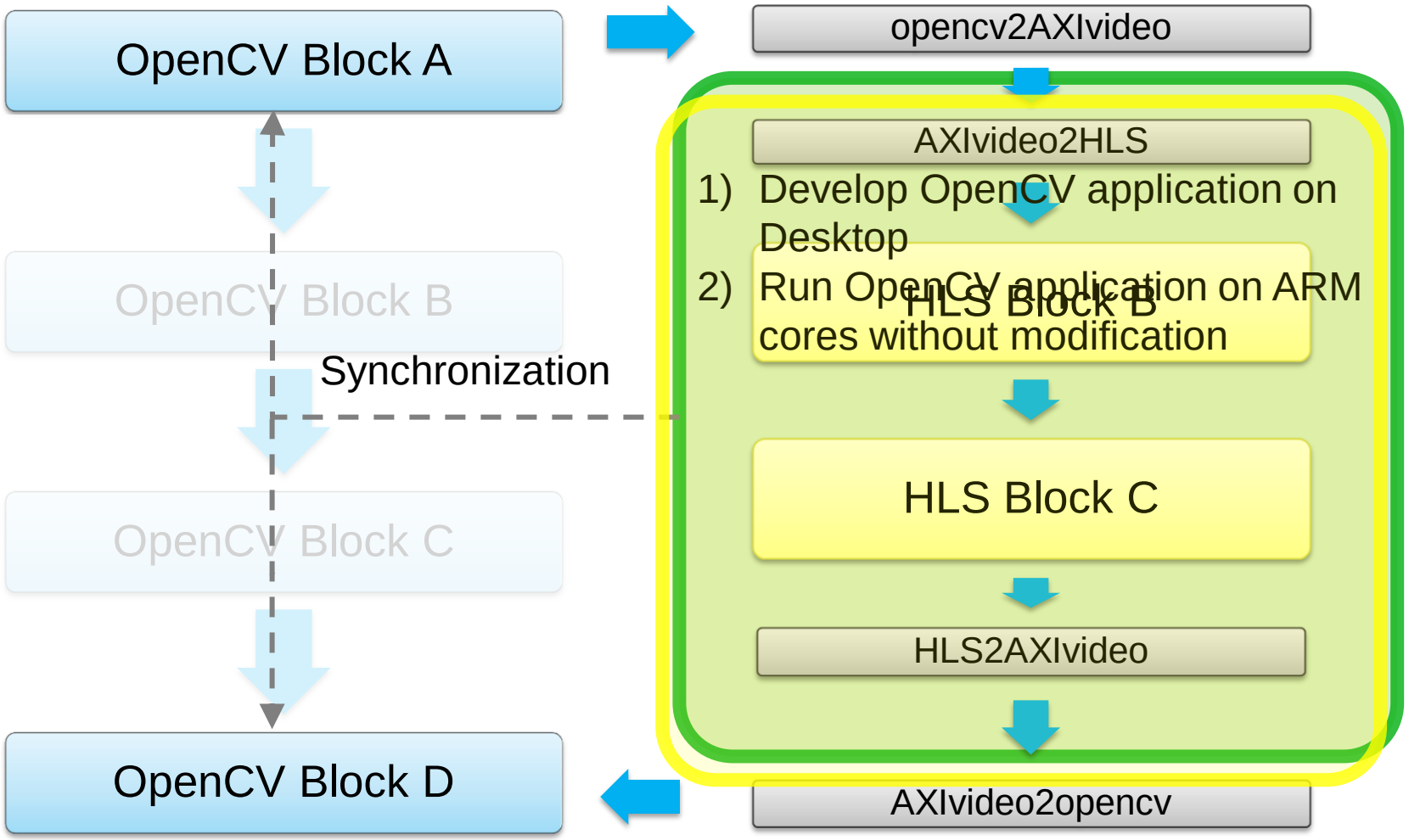
OpenCV Interface Functions (OpenCV data type <-> AXI-4 Stream)			
cvMat2AXIvideo	AXIvideo2cvMat	cvMat2hlsMat	hlsMat2cvMat
IplImage2AXIvideo	AXIvideo2IplImage	IplImage2hlsMat	hlsMat2IplImage
CvMat2AXIvideo	AXIvideo2CvMat	CvMat2hlsMat	hlsMat2CvMat

Video Functions			
AbsDiff	Duplicate	MaxS	Remap
AddS	EqualizeHist	Mean	Resize
AddWeighted	Erode	Merge	Scale
And	FASTX	Min	Set
Avg	Filter2D	MinMaxLoc	Sobel
AvgSdv	GaussianBlur	MinS	Split
Cmp	Harris	Mul	SubRS
CmpS	HoughLines2	Not	SubS
CornerHarris	Integral	PaintMask	Sum
CvtColor	InitUndistortRectifyMap	Range	Threshold
Dilate	Max	Reduce	Zero

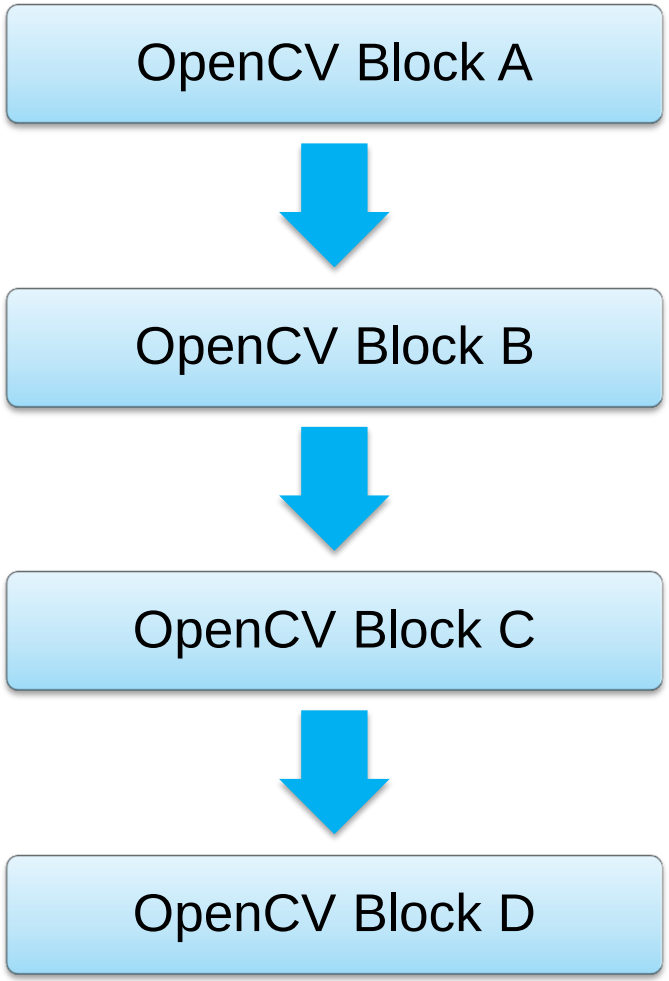
- For function signatures and descriptions, see the HLS user guide UG902

Partitioned OpenCV Application

3) Replace selected OpenCV functions with HLS IP accelerator

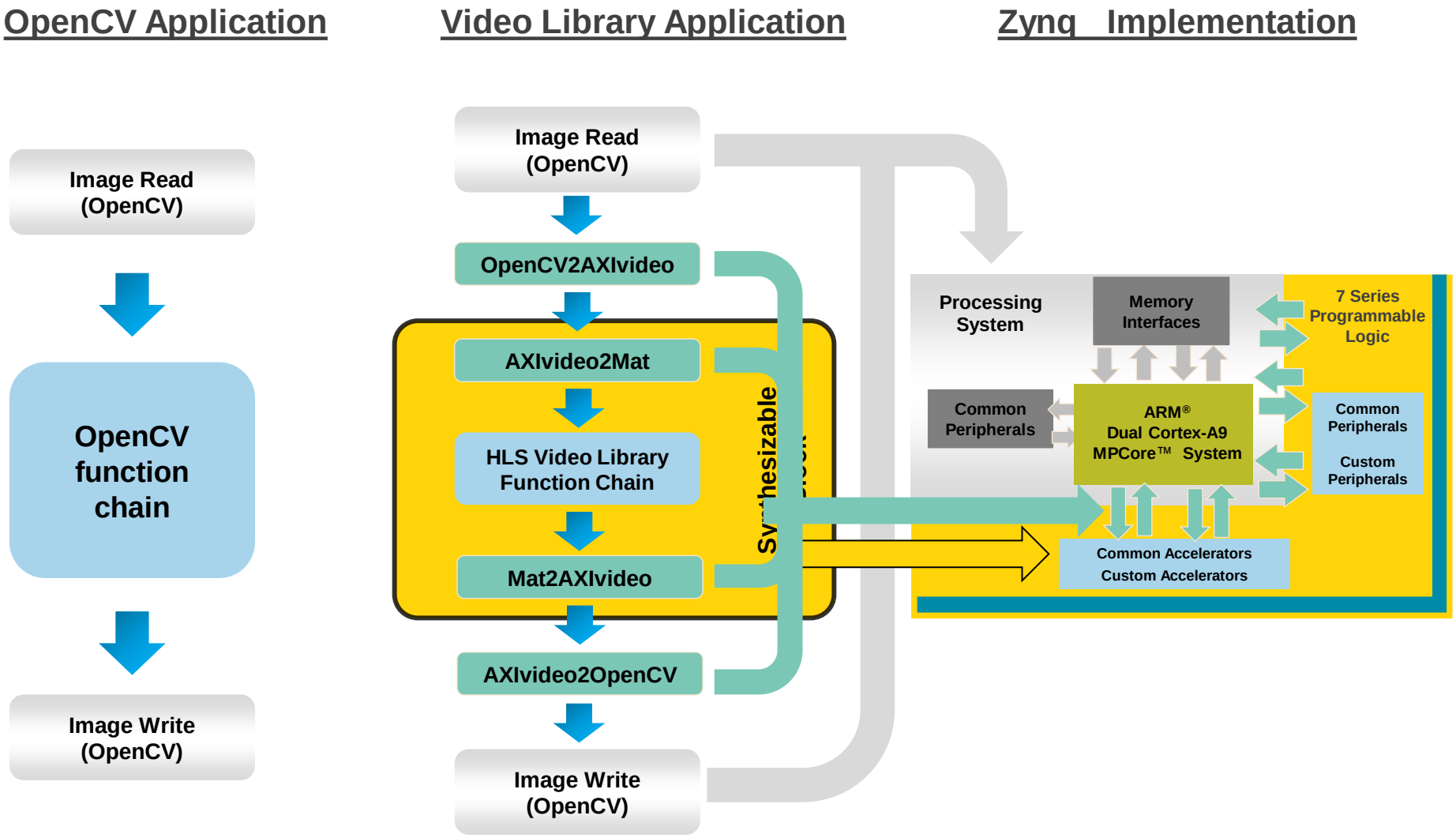


OpenCV design flow



1) Develop OpenCV application on Desktop
2) Run OpenCV application on ARM cores without modification
3) Abstract FPGA portion using I/O functions
4) Replace OpenCV function calls with synthesizable code
5) Run HLS to generate FPGA accelerator
6) Replace call to synthesizable code with call to FPGA accelerator

Implementing an OpenCV System on Zynq



OpenCV Design Tradeoffs

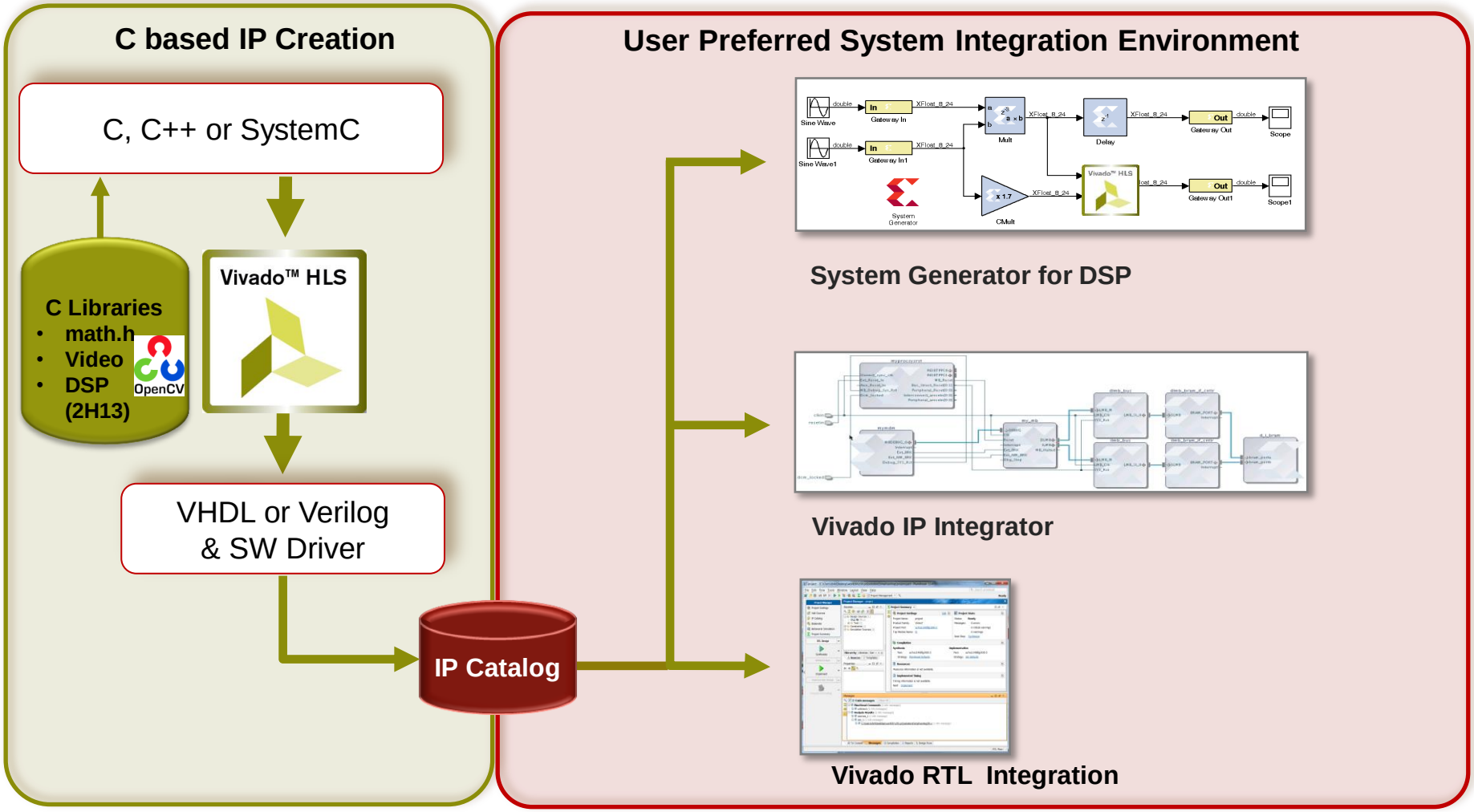
- **OpenCV-based image processing is built around memory frame buffers**
  - Poor access locality -> small caches perform poorly
  - Complex architectures for performance -> higher power
  - Likely ‘good enough’ for many applications
    - Low resolution or framerate
    - Processing of features or regions of interest in a larger image
- **Streaming architectures give high performance and low power**
  - Chaining image processing functions reduces external memory accesses
  - Video-optimized line buffers and window buffers simpler than processor caches
  - Can be implemented with streaming optimizations in HLS
  - Requires conversion of code to be synthesizable

Limitations

- **Must replace OpenCV calls with video library functions**
- **Frame buffer access not supported through pointers**
  - use VDMA and AXI Stream adapter functions
- **Random access not supported**
  - data read more than once must be duplicated
  - see **`hls::Duplicate()`**
- **In-place update not supported**
  - e.g. **`cvRectangle (img, point1, point2)`**

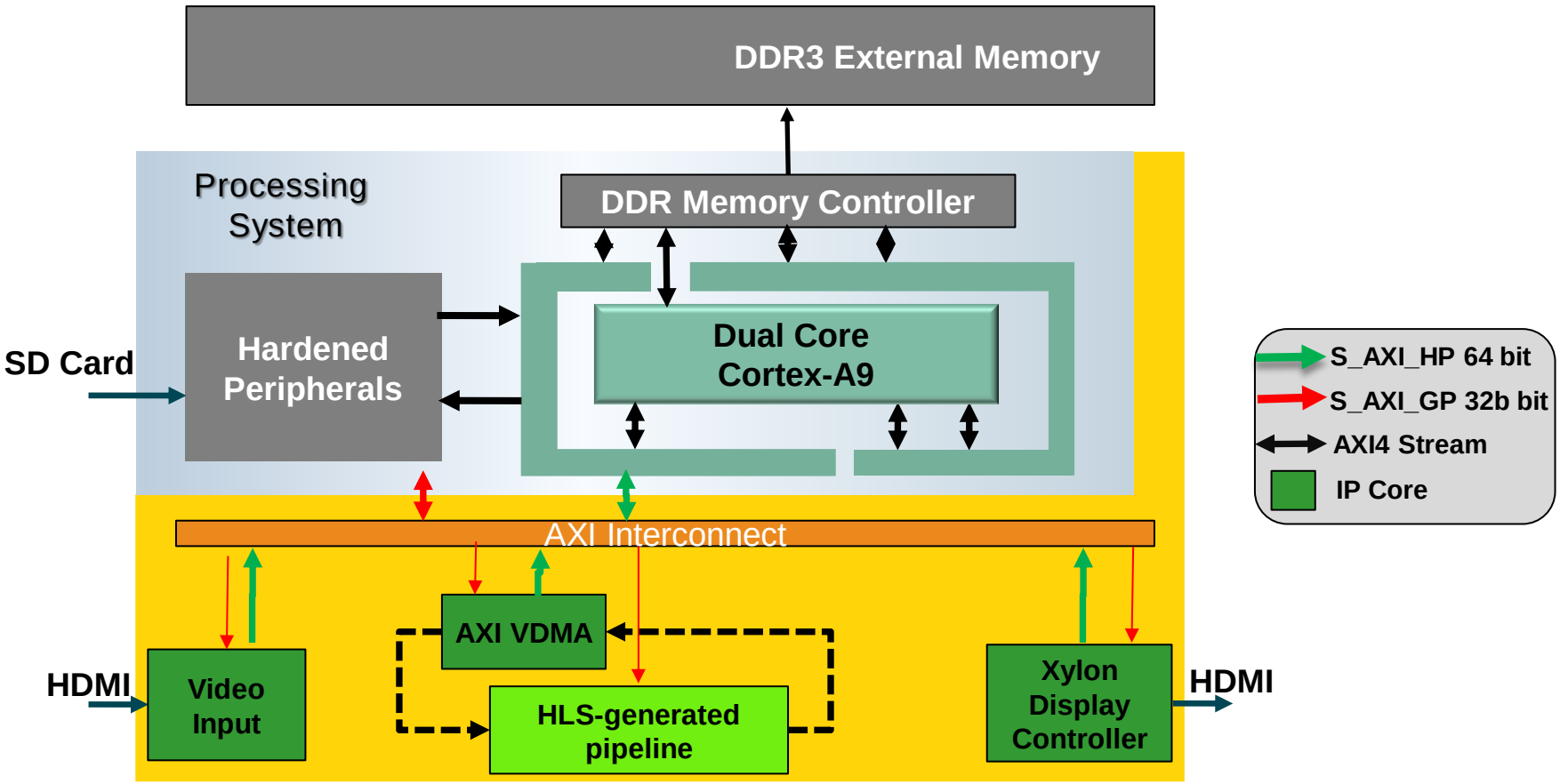
	OpenCV	HLS Video Library
Read operation	`pix = cv_mat.at<T>(i,j)` `pix = cvGet2D(cv_img,i,j)`	`hls_img >> pix`
Write operation	`cv_mat.at<T>(i,j) = pix` `cvSet2D(cv_img,i,j,pix)`	`hls_img << pix`

Vivado HLS System IP Integration Flow



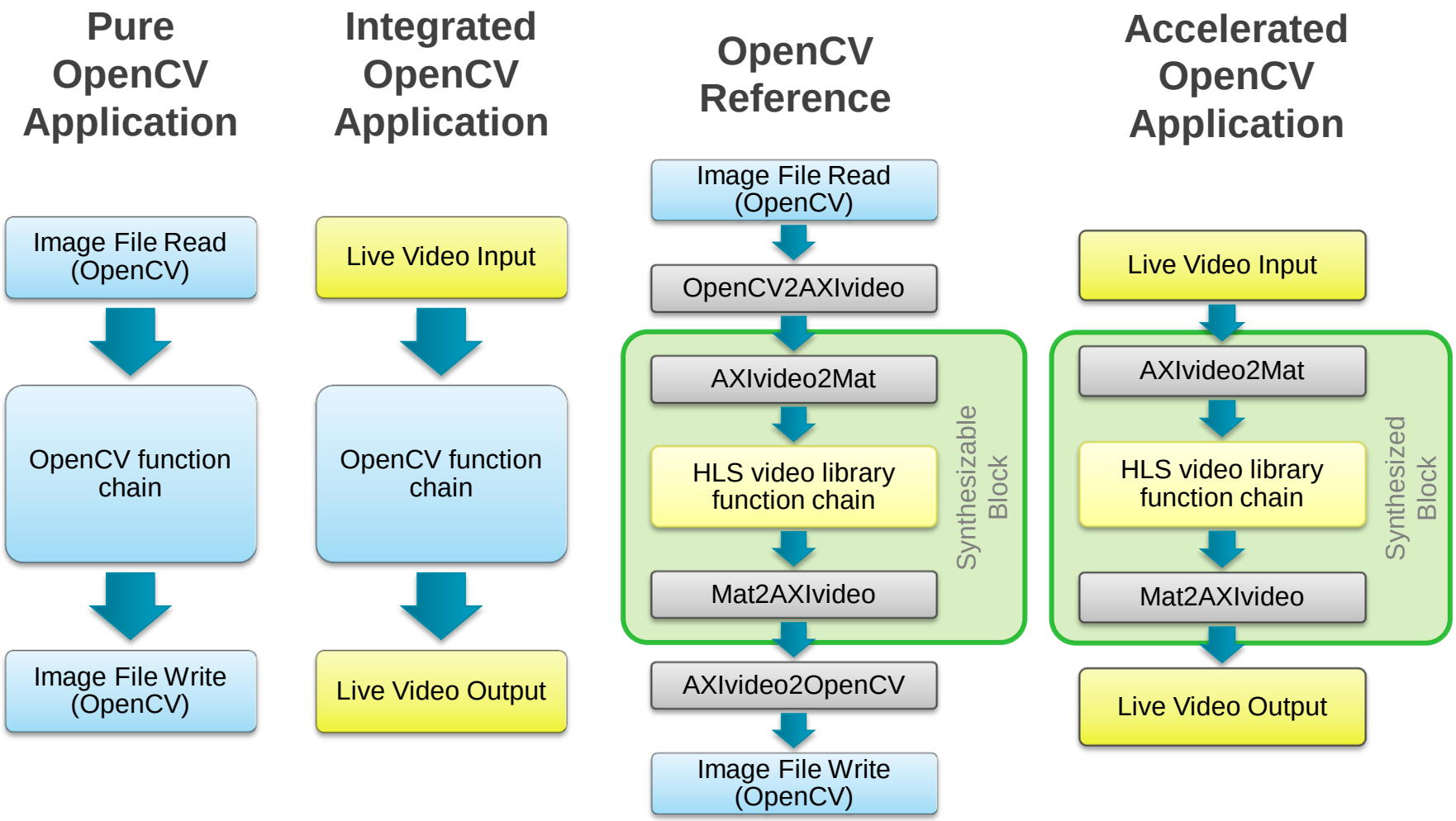
7 Series FPGA and Zynq SoC Vivado Implementation

Zynq Video TRD architecture

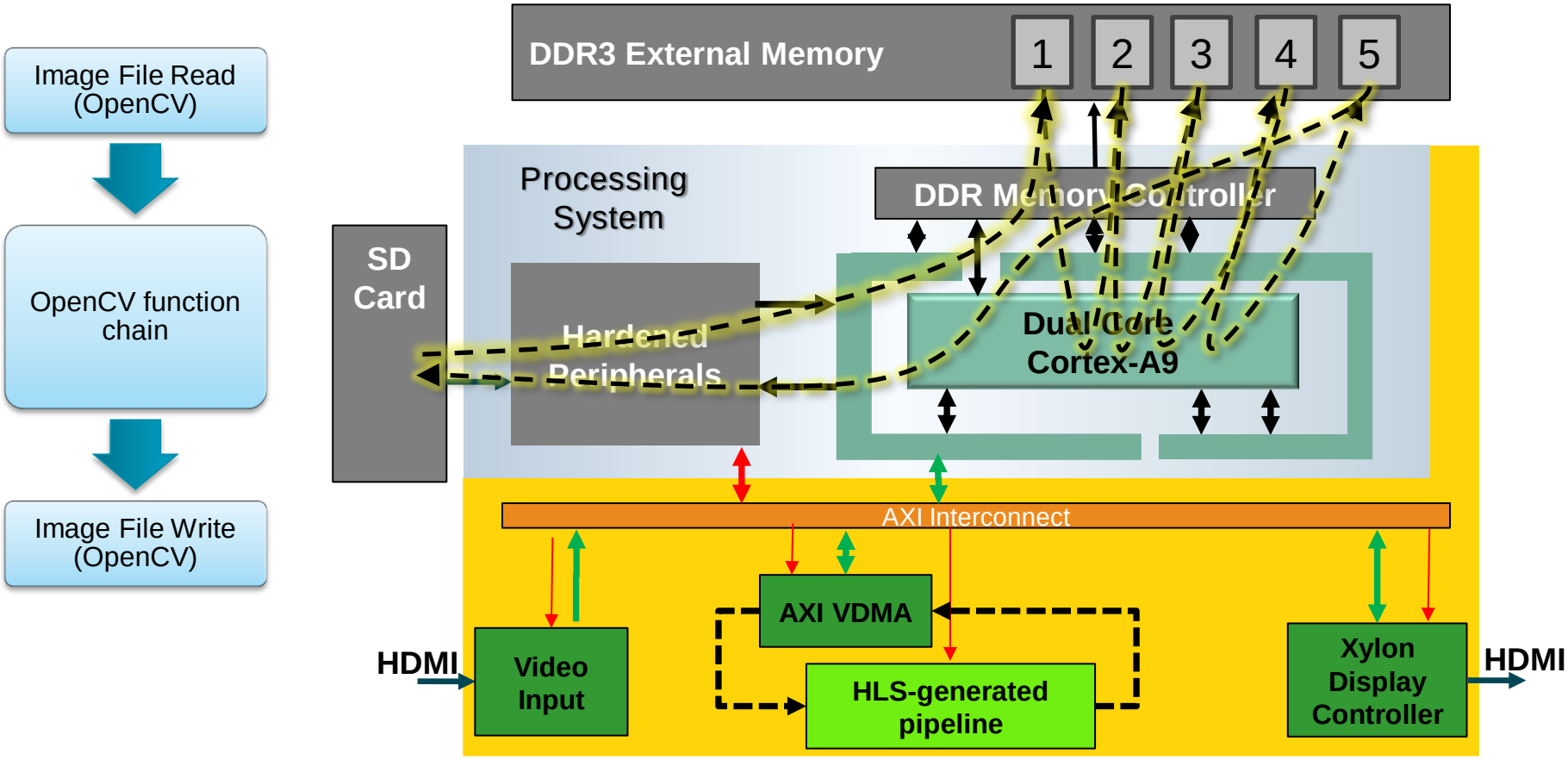


- Video access to external memory using 64-bit High Performance ports
- Control register access using 32-bit General Purpose ports
- Video streams implemented using AXI4-Stream

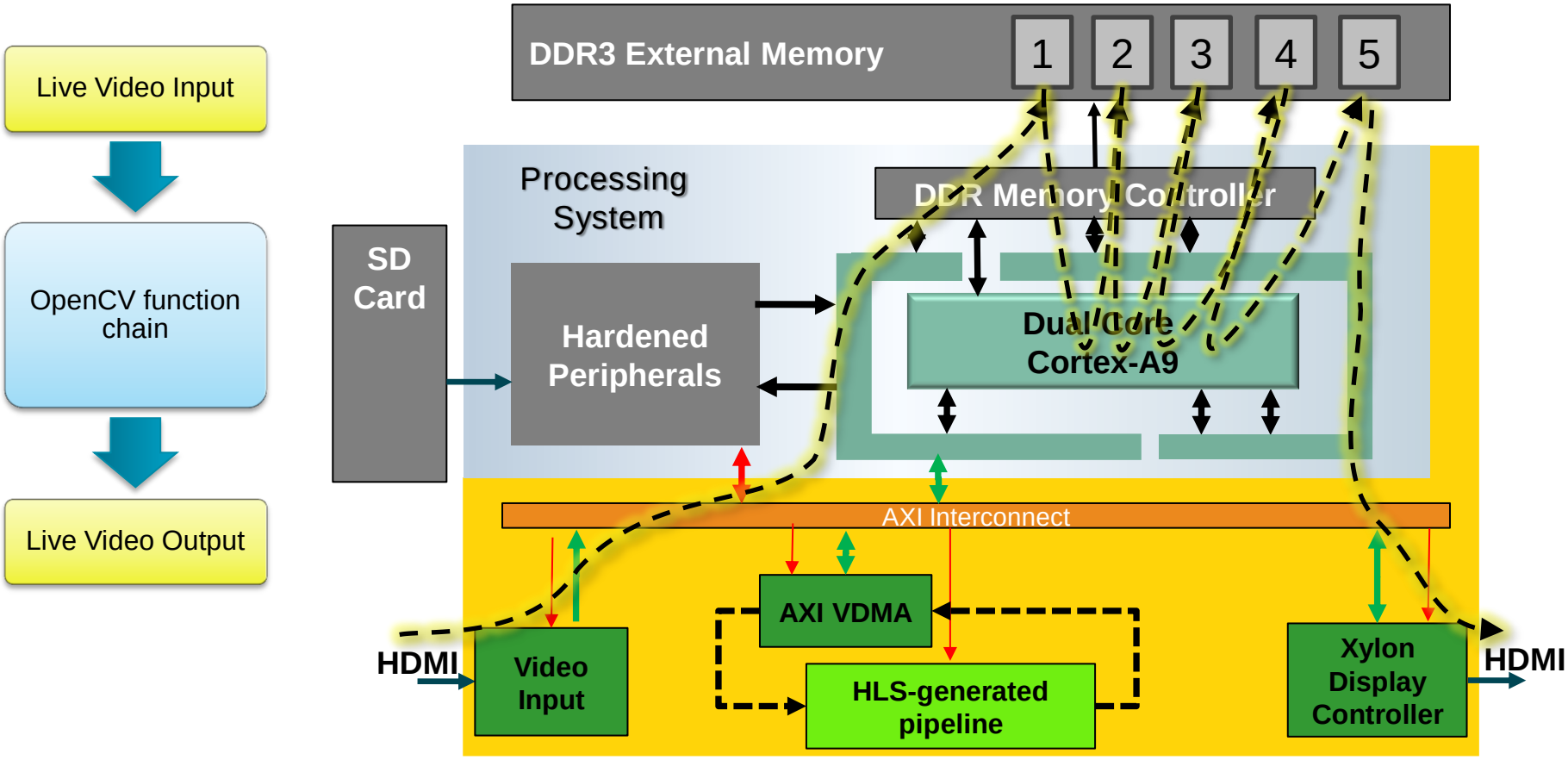
Using OpenCV in FPGA designs



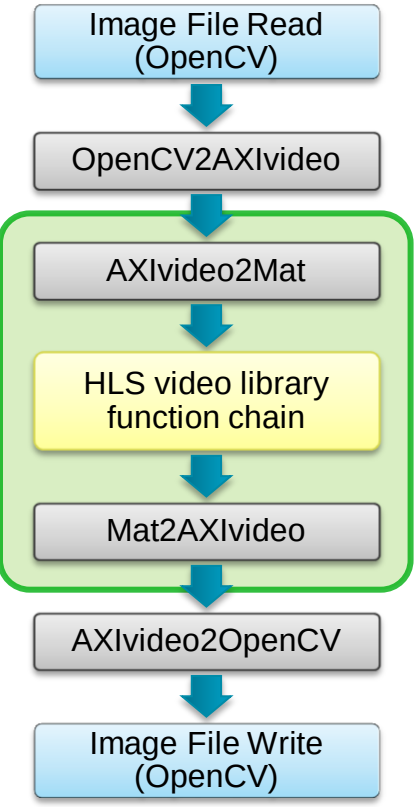
Pure OpenCV Application

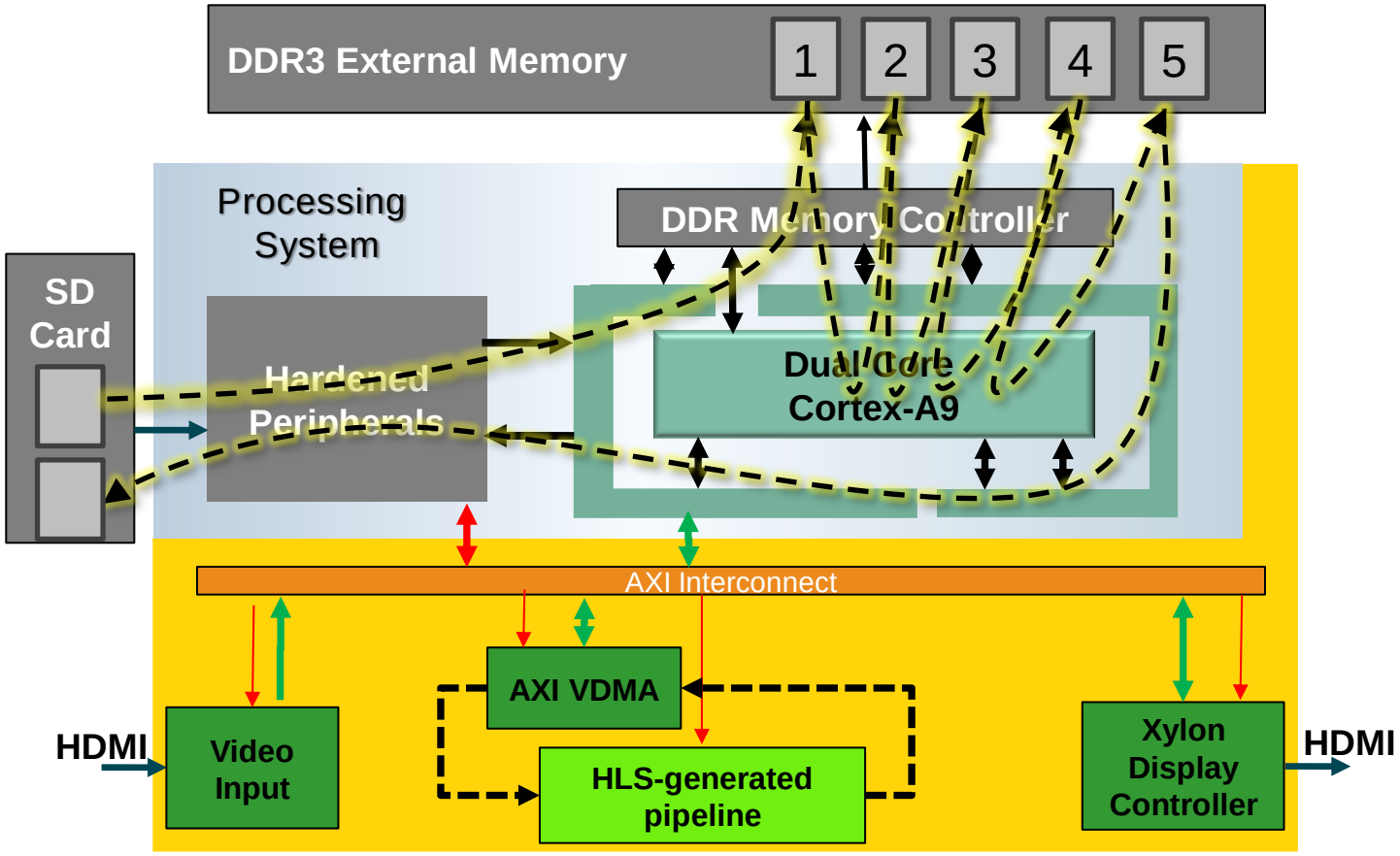


Integrated OpenCV Application

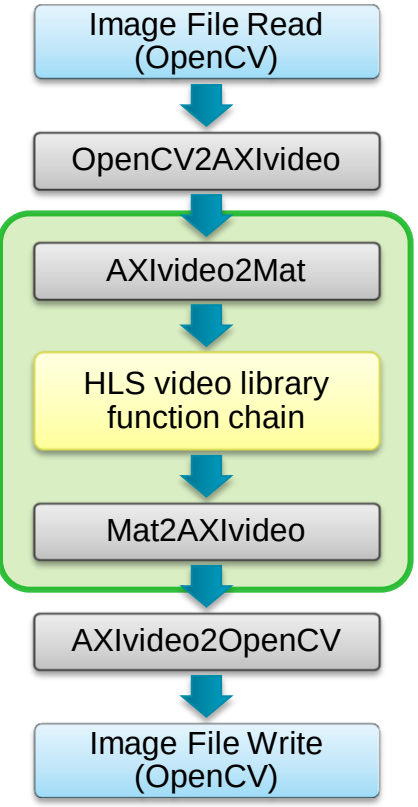


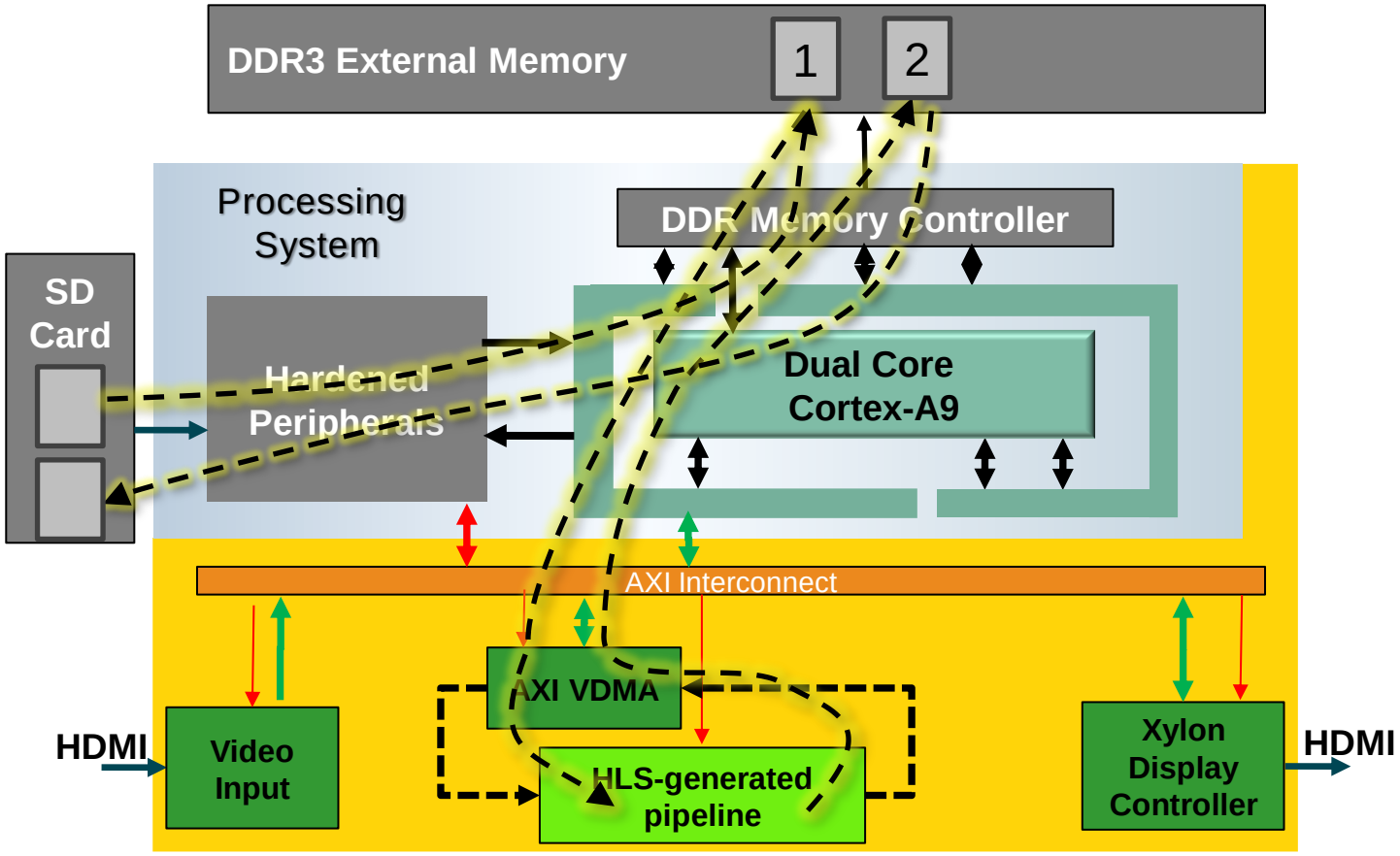
OpenCV Reference / Software Execution



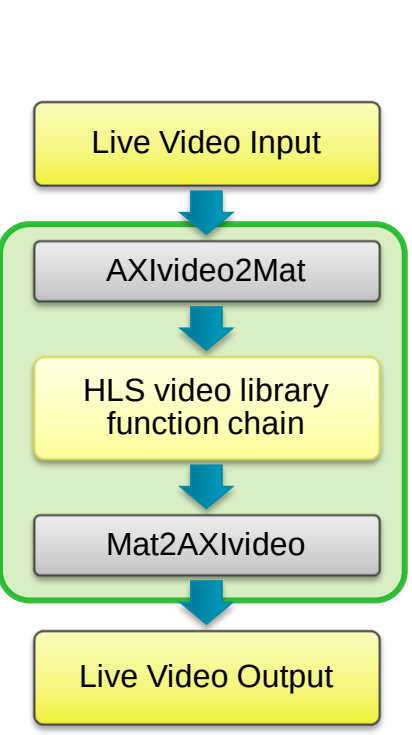


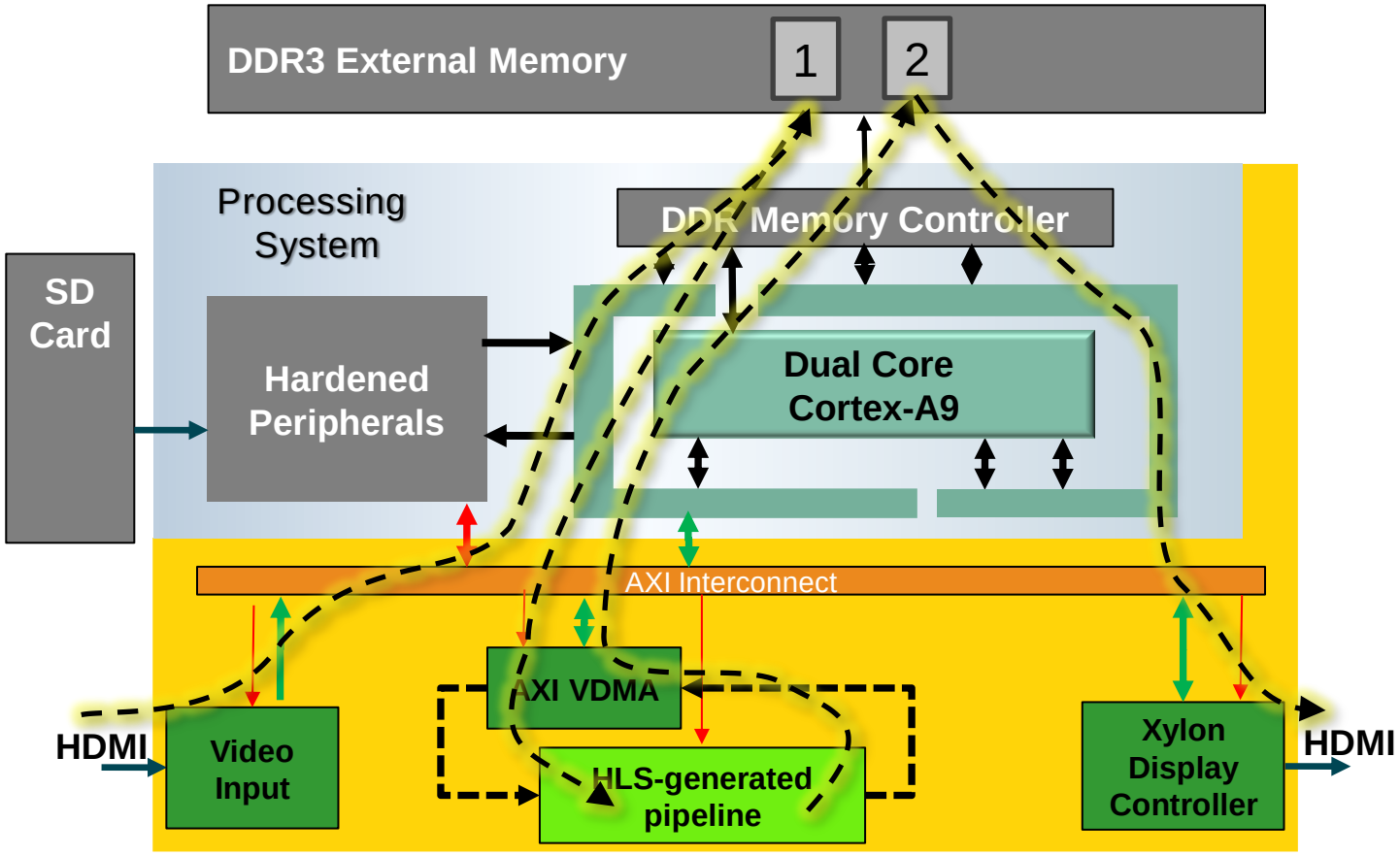
OpenCV Reference / In system Test

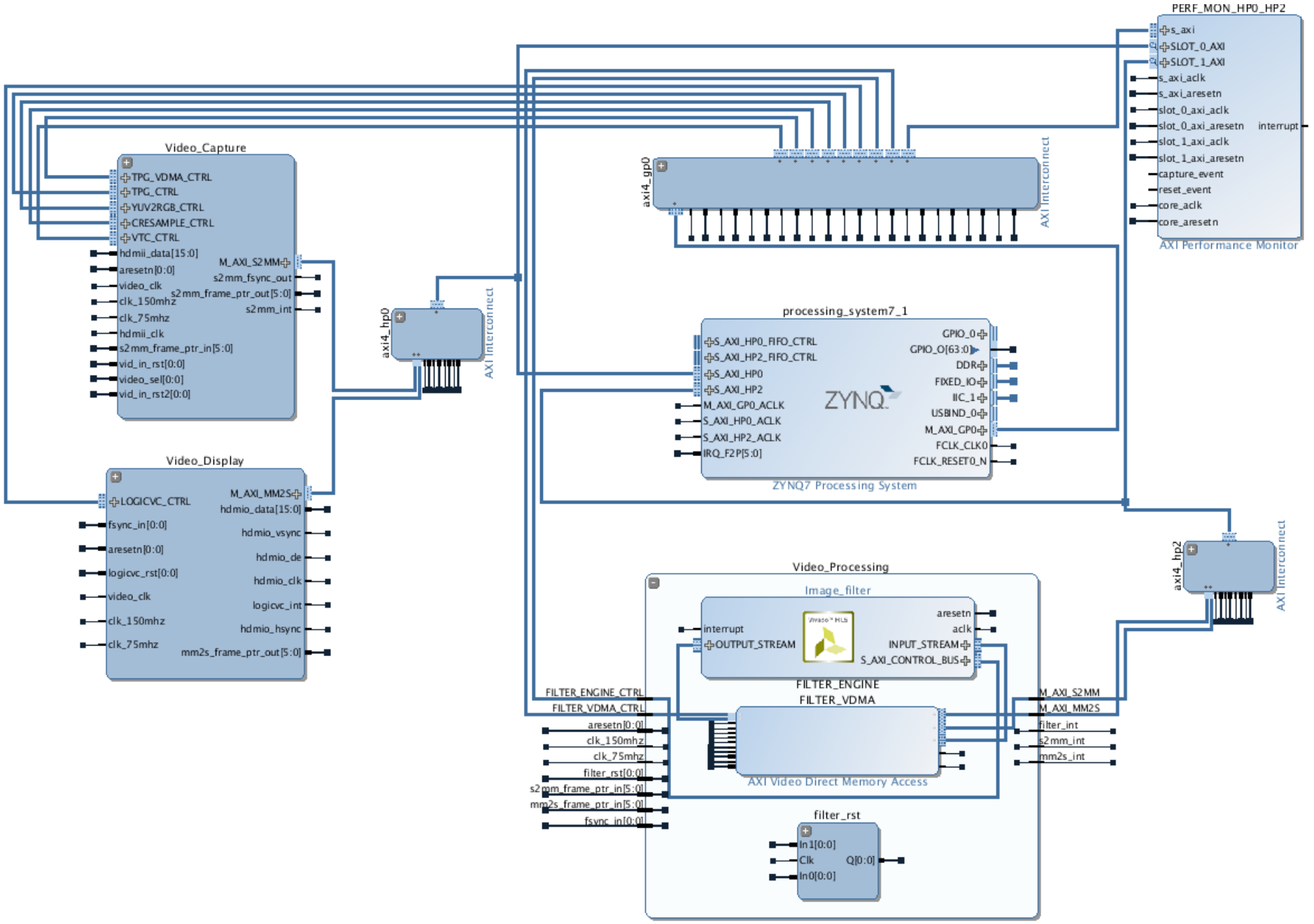




Accelerated OpenCV Application





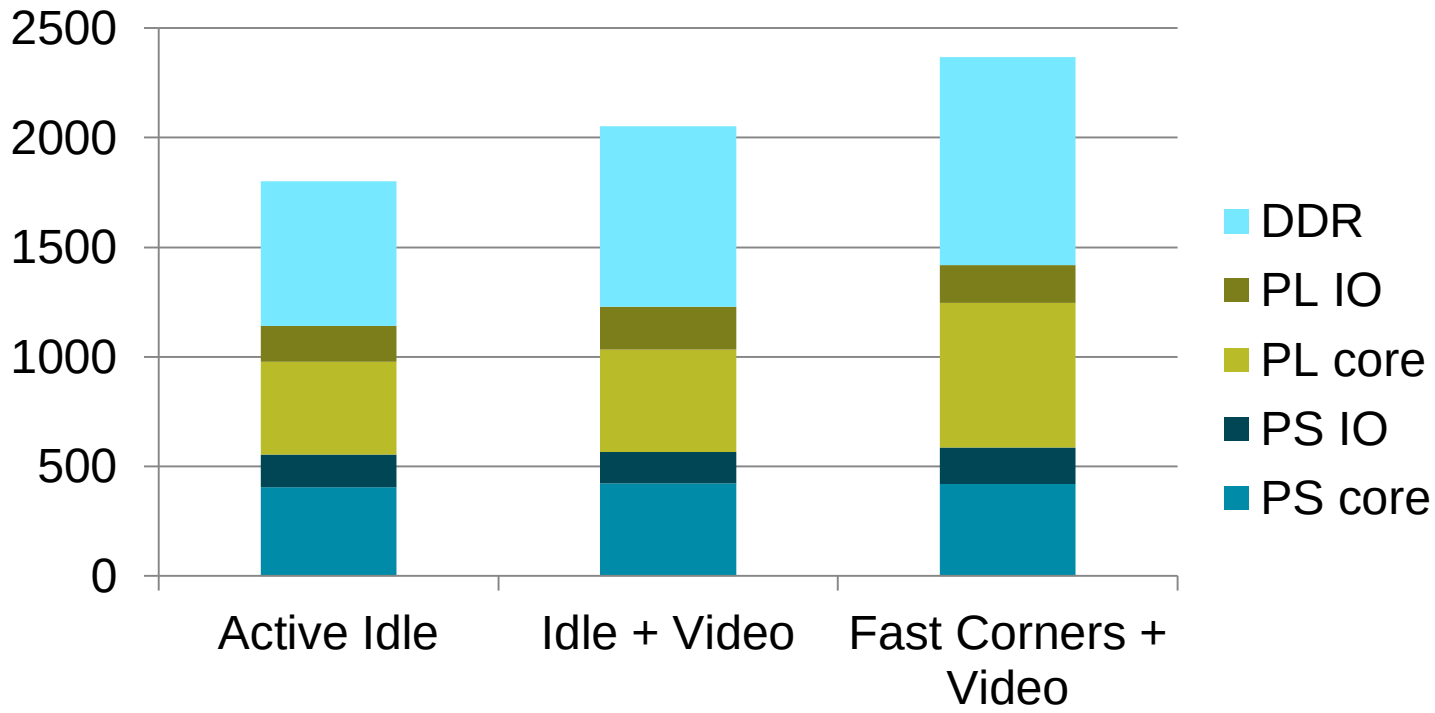


Performance Analysis

- **AXI Performance Monitor collects statistics on memory bandwidth**
  - see /mnt/AXI_PerfMon.log
- **Video + fast corners**
  - 1920*1080*60*32 = ~4 Gb/s per stream
  - **`HP0: Read 4.01 Gb/s, Write 4.01 Gb/s, Total 8.03 Gb/s`**
  - **`HP2: Read 4.01 Gb/s, Write 4.01 Gb/s, Total 8.03 Gb/s`**

Power Analysis

- **Voltage and Current can be read from the digital power regulators on the ZC702 board.**
- **Custom, realtime HD video processing in 2-3 Watts total system power**
  - FASTX is less than 200 mW incremental power

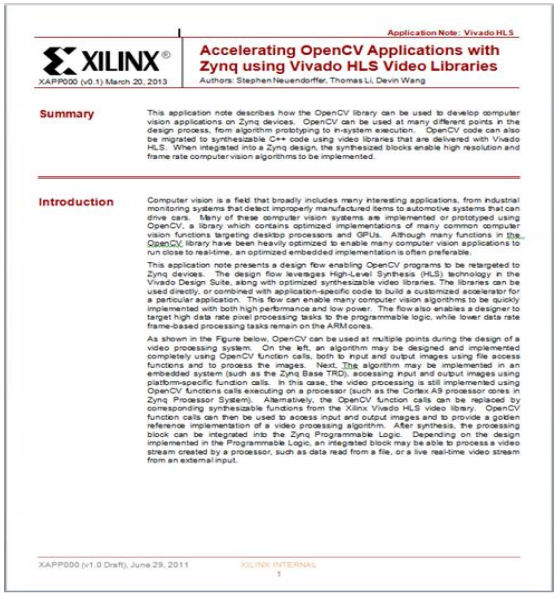


HLS and Zynq accelerates OpenCV apps

- OpenCV functions enable fast prototyping of Computer Vision algorithms
- Computer Vision applications are inherently heterogenous and require a mix HW and SW implementation
- Vivado HLS video library accelerates mapping of openCV functions to FPGA programmable fabric
- Zynq offers power-optimized integrated solution with high performance programmable logic and embedded ARM

Additional OpenCV Collateral at Xilinx.com

Download XAPP1167 from Xilinx.com



Application Note: Vivado HLS

XILINX®

XAPP000 (v0.1) March 20, 2013

Accelerating OpenCV Applications with Zynq using Vivado HLS Video Libraries

Authors: Stephen Neuendorffer, Thomas Li, Devin Wang

Summary

This application note describes how the OpenCV library can be used to develop computer vision applications on Zynq devices. OpenCV can be used at many different points in the design process, from algorithm prototyping to in-system execution. OpenCV code can also be migrated to synthesizable C++ code using video libraries that are delivered with Vivado HLS. When integrated into a Zynq design, the synthesized blocks enable high resolution and frame rate computer vision algorithms to be implemented.

Introduction

Computer vision is a field that broadly includes many interesting applications, from industrial monitoring systems that detect improperly manufactured items to automotive systems that can drive cars. Many of these computer vision systems are implemented or prototyped using OpenCV, a library which contains optimized implementations of many common computer vision functions targeting desktop processors and GPUs. Although many functions in the OpenCV library have been heavily optimized to enable many computer vision applications to run close to real-time, an optimized embedded implementation is often preferable.

This application note presents a design flow enabling OpenCV programs to be retargeted to Zynq devices. The design flow leverages High-Level Synthesis (HLS) technology in the Vivado Design Suite, along with optimized synthesizable video libraries. The libraries can be used directly, or combined with application-specific code to build a customized accelerator for a particular application. This flow can enable many computer vision algorithms to be quickly implemented with both high performance and low power. The flow also enables a designer to target high data rate pixel processing tasks to the programmable logic, while lower data rate frame-based processing tasks remain on the ARM cores.

As shown in the Figure below, OpenCV can be used at multiple points during the design of a video processing system. On the left, an algorithm may be designed and implemented completely using OpenCV function calls, both to input and output images using file access functions and to process the images. Next, The algorithm may be implemented in an embedded system (such as the Zynq Base TRD), accessing input and output images using platform-specific function calls. In this case, the video processing is still implemented using OpenCV functions calls executing on a processor (such as the Cortex A9 processor cores in Zynq Processor System). Alternatively, the OpenCV function calls can be replaced by corresponding synthesizable functions from the Xilinx Vivado HLS video library. OpenCV function calls can then be used to access input and output images and to provide a golden reference implementation of a video processing algorithm. After synthesis, the processing block can be integrated into the Zynq Programmable Logic. Depending on the design implemented in the Programmable Logic, an integrated block may be able to process a video stream created by a processor, such as data read from a file, or a live real-time video stream from an external input.

XAPP000 (v1.0 Draft), June 29, 2011 XILINX INTERNAL 1

http://www.xilinx.com/hls

http://www.xilinx.com/getlicense

QuickTake: Leveraging OpenCV and High-Level Synthesis with Vivado