如何使用Vivado HLS视频库加速Zynq-7000 All Programmable SoC OpenCV应用

2013年9月11日

OpenCV简介

- 开源计算机视觉 (OpenCV) 被广泛用于开发计算机视觉应用
  - 包含2500多个优化的视频函数的函数库
  - 专门针对台式机处理器和GPU进行优化
  - 用户成千上万
  - 无需修改即可在 Zynq器件的ARM处理器上运行

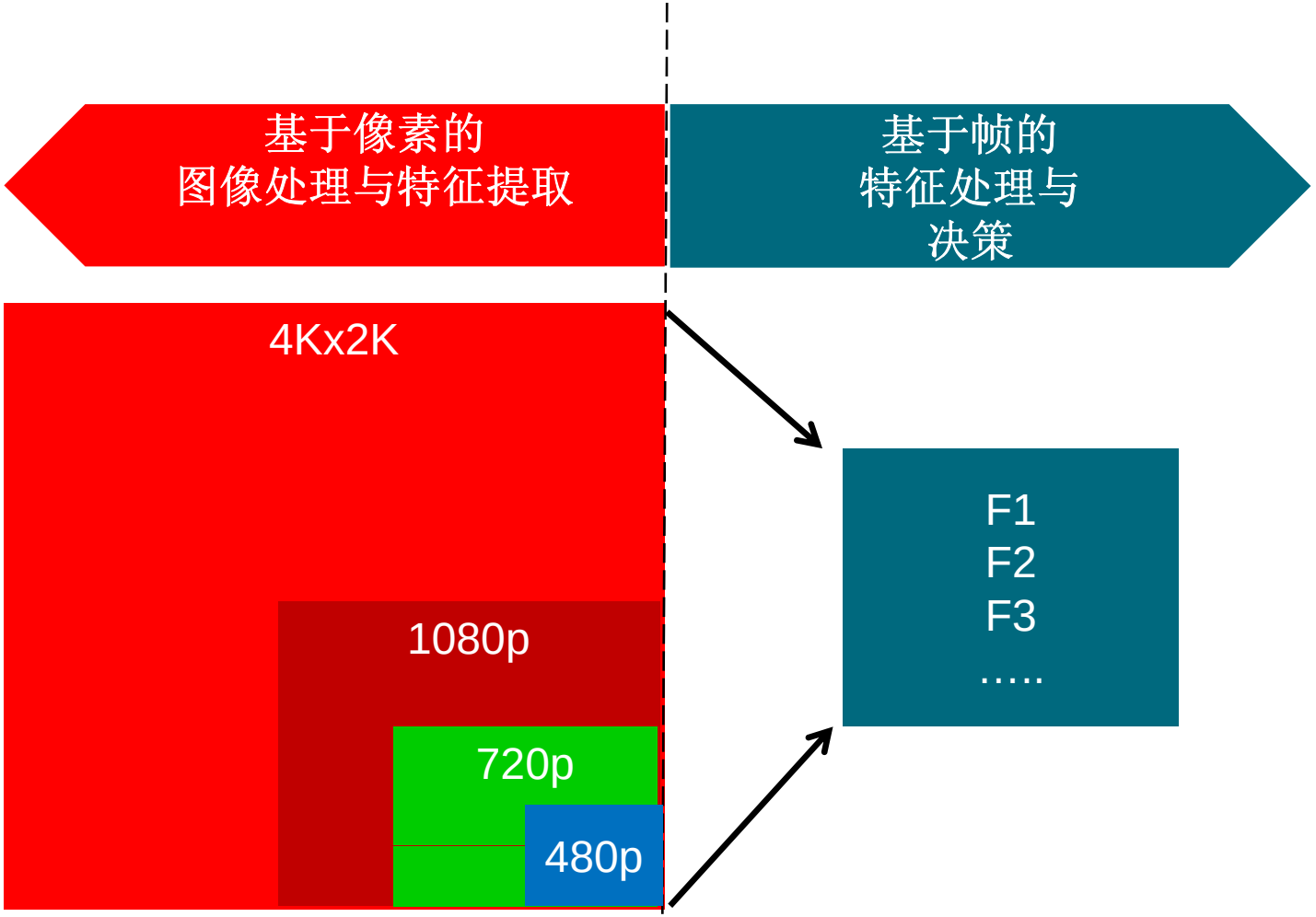
- 但是
  - 利用OpenCV实现的高清处理经常受外部存储器的限制
  - 存储带宽会成为性能瓶颈
  - 存储访问会限制功耗效率

- **Zynq All-programmable SOC是实现嵌入式计算机视觉应用的极好方法**
  - 性能高、功耗低

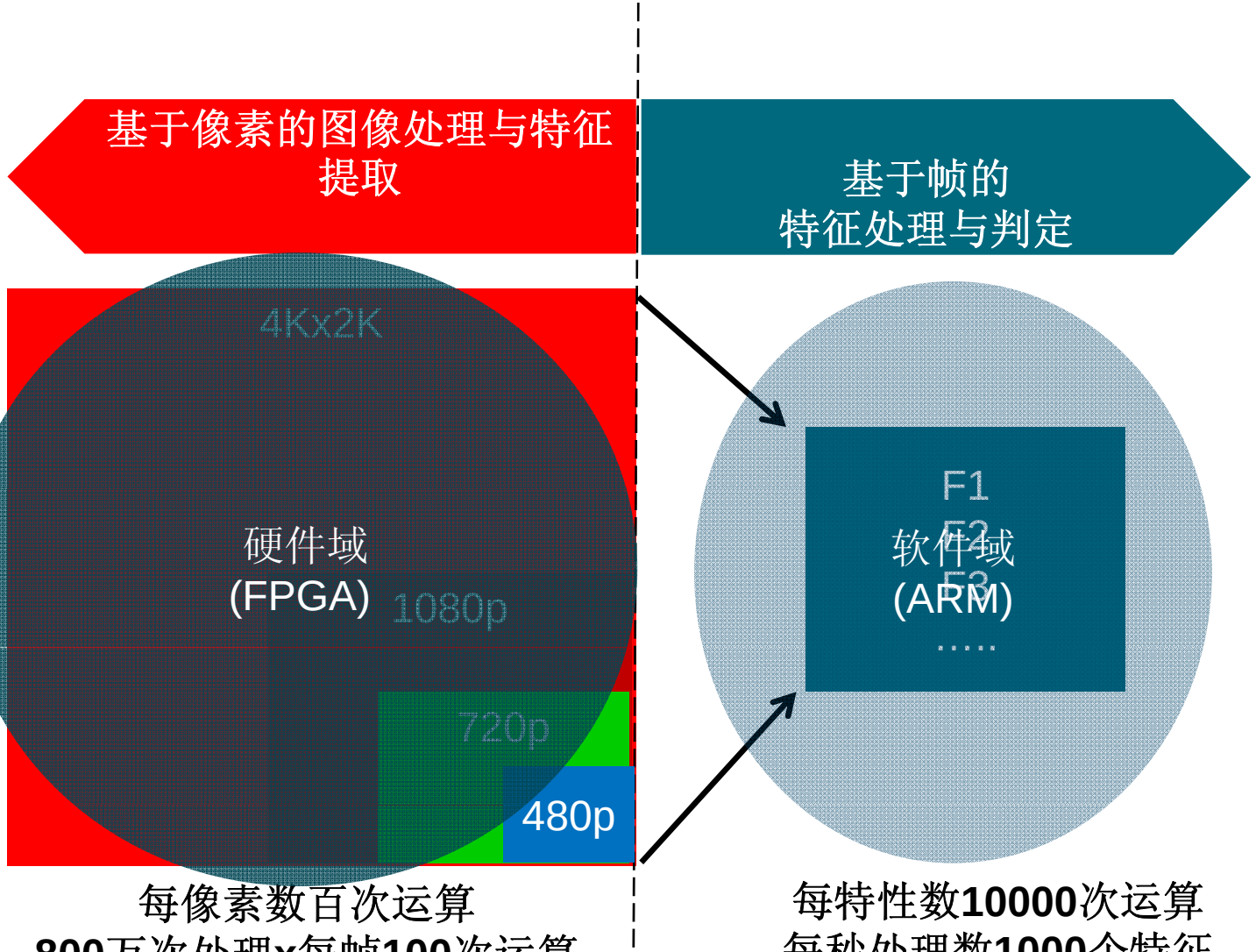
实时计算机视觉应用

计算机视觉应用		实时分析功能
高级驾驶员安全辅助		车道或行人检测
安防监视		敌我识别
用于工厂自动化的机器视觉		高速物体检测
非侵入式医疗成像技术		肿瘤检测

实时视频分析处理



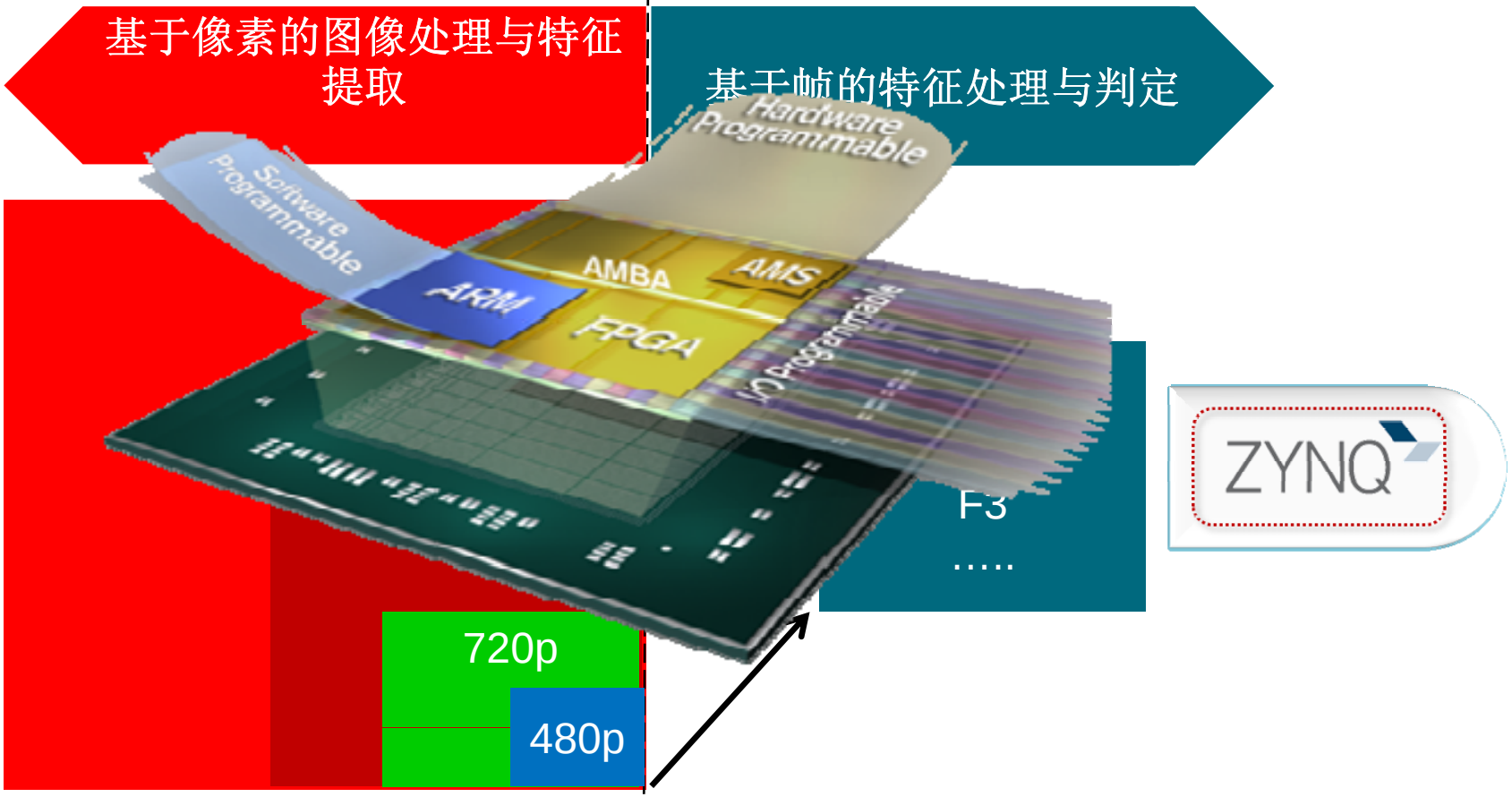
实时视频分析的异构实现



每像素数百次运算
800万次处理x每帧100次运算
= 数百亿次运算

每特性数10000次运算
每秒处理数1000个特征
= 数百万次运算

赛灵思实时图像分析的实现：Zynq All Programmable SoC



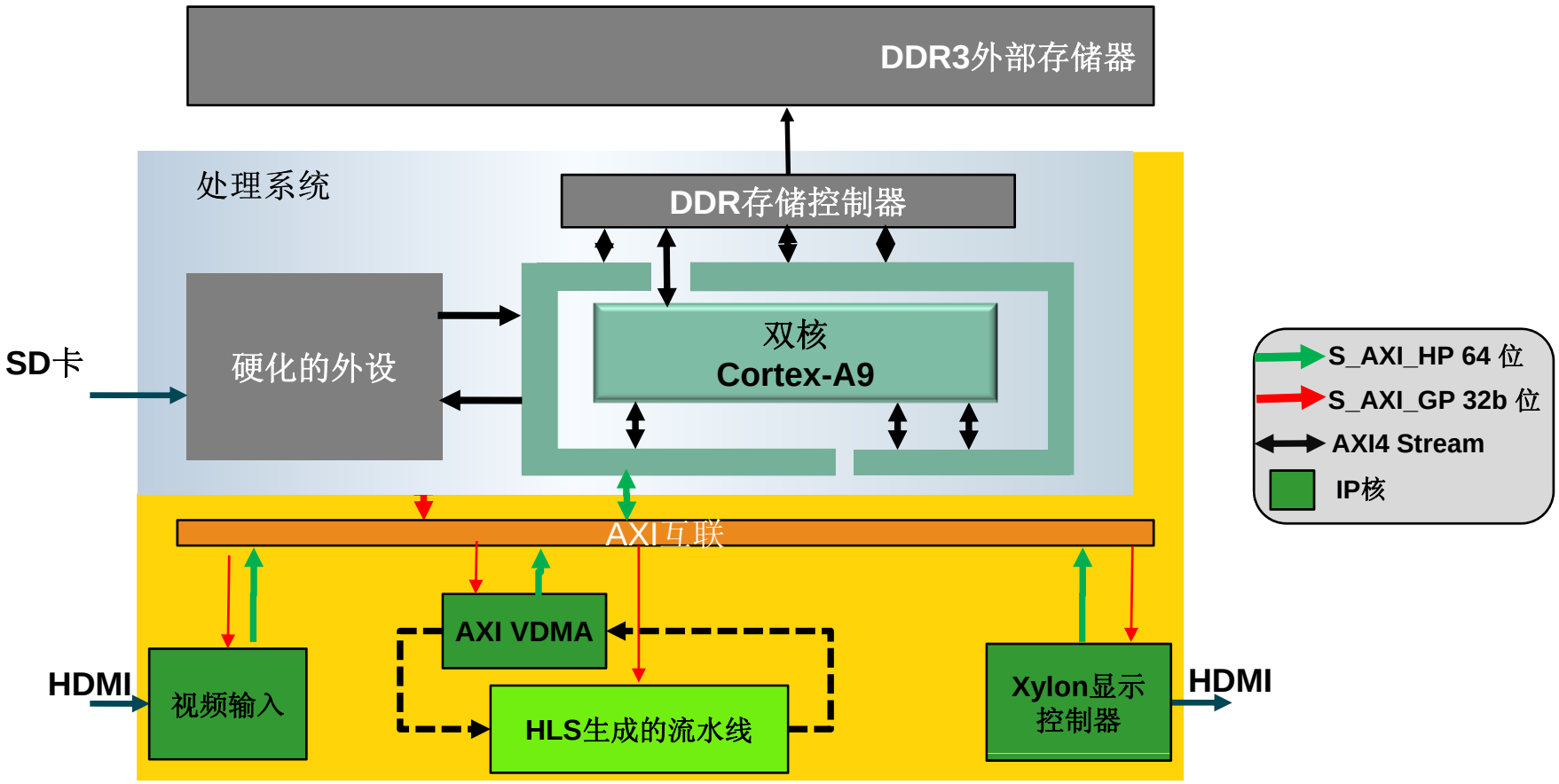
每像素数百次运算
800万次处理x每帧100次运算
= 数百亿次运算

每特性数**10000**次运算
每秒处理数**1000**个特征
= 数百万次运算

Vivado：提高OpenCV应用的效率

- 高清视频算法（每秒约**1**帧）的**C**语言仿真
  - 高清视频（每小时1 帧）的RTL仿真
- 实时**FPGA**实现方案高达**60fps**

Zynq视频参考设计架构



- 使用64位高性能端口实现对外部存储器的视频访问
- 使用32位通用端口实现控制寄存器访问
- 使用AXI4-Stream实现的视频流

以IP为中心的设计流程

更快速的IP生成与集成

基于C语言的IP创建

C、C++ 或 SystemC

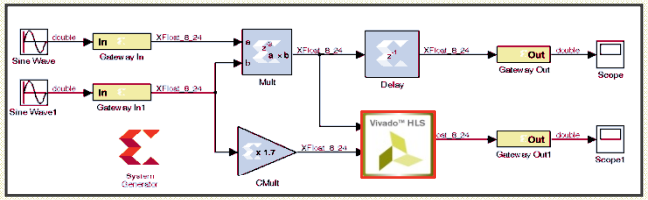
C 函数库
- 浮点math.h
- 定点
- 视频

Vivado™ HLS

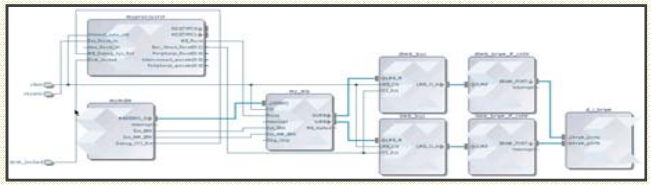
VHDL 或Verilog 以及软件驱动

IP子系统
赛灵思IP
第三方IP
用户IP

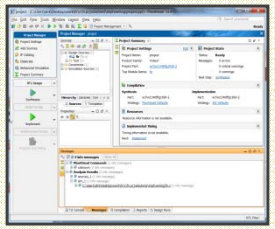
用户首选的系统集成环境



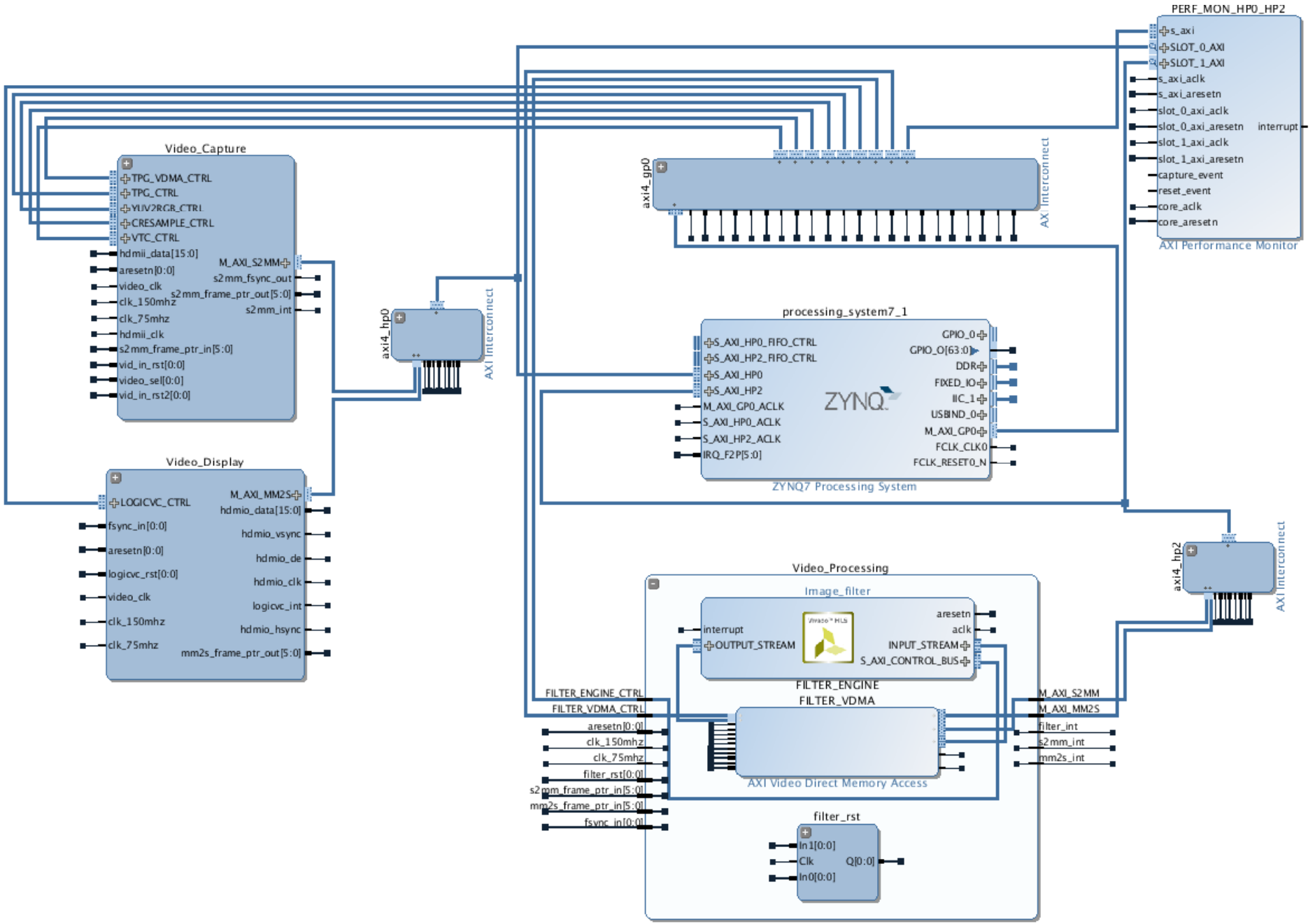
DSP系统生成器 (System Generator)



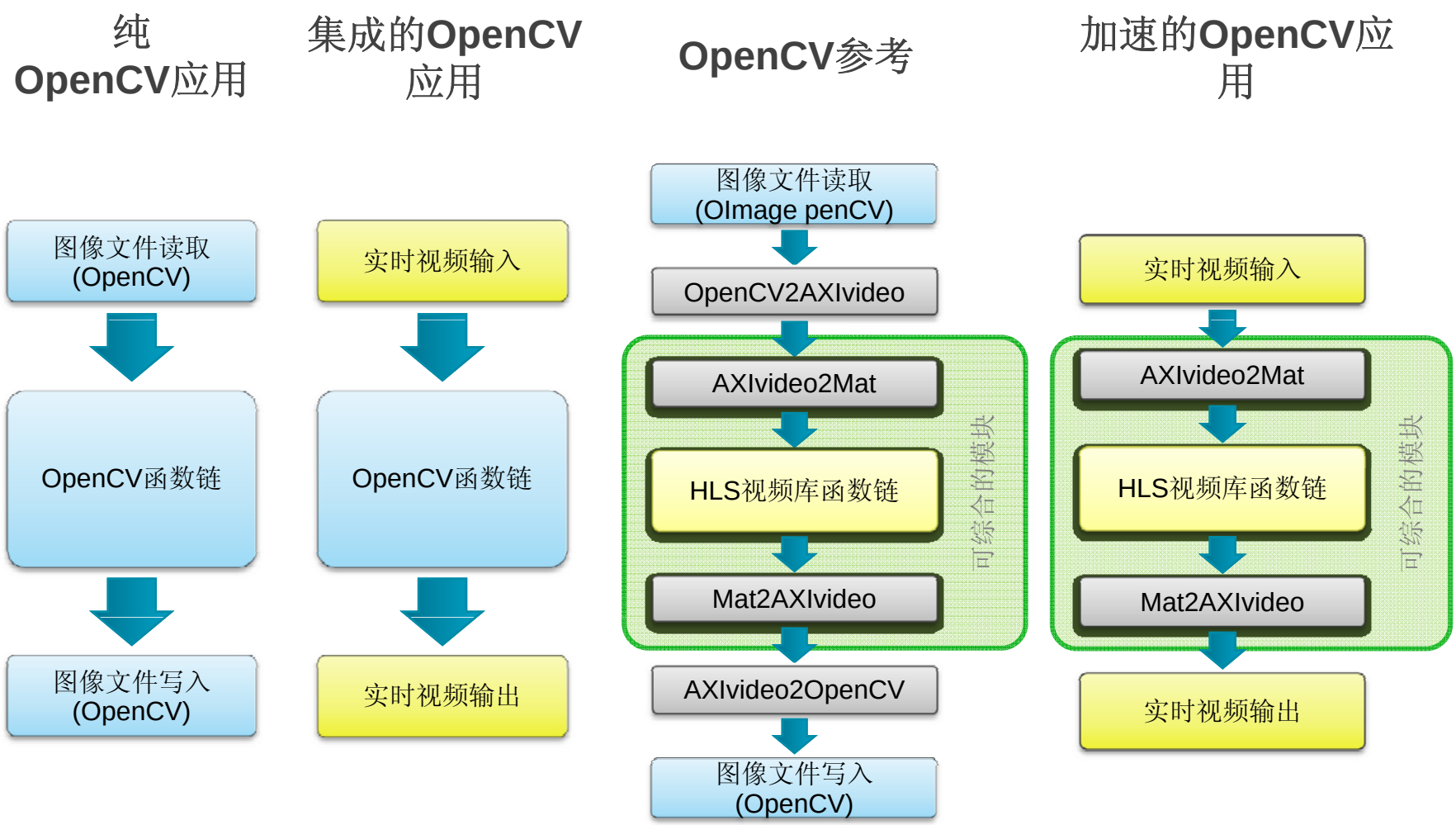
Vivado IP集成器



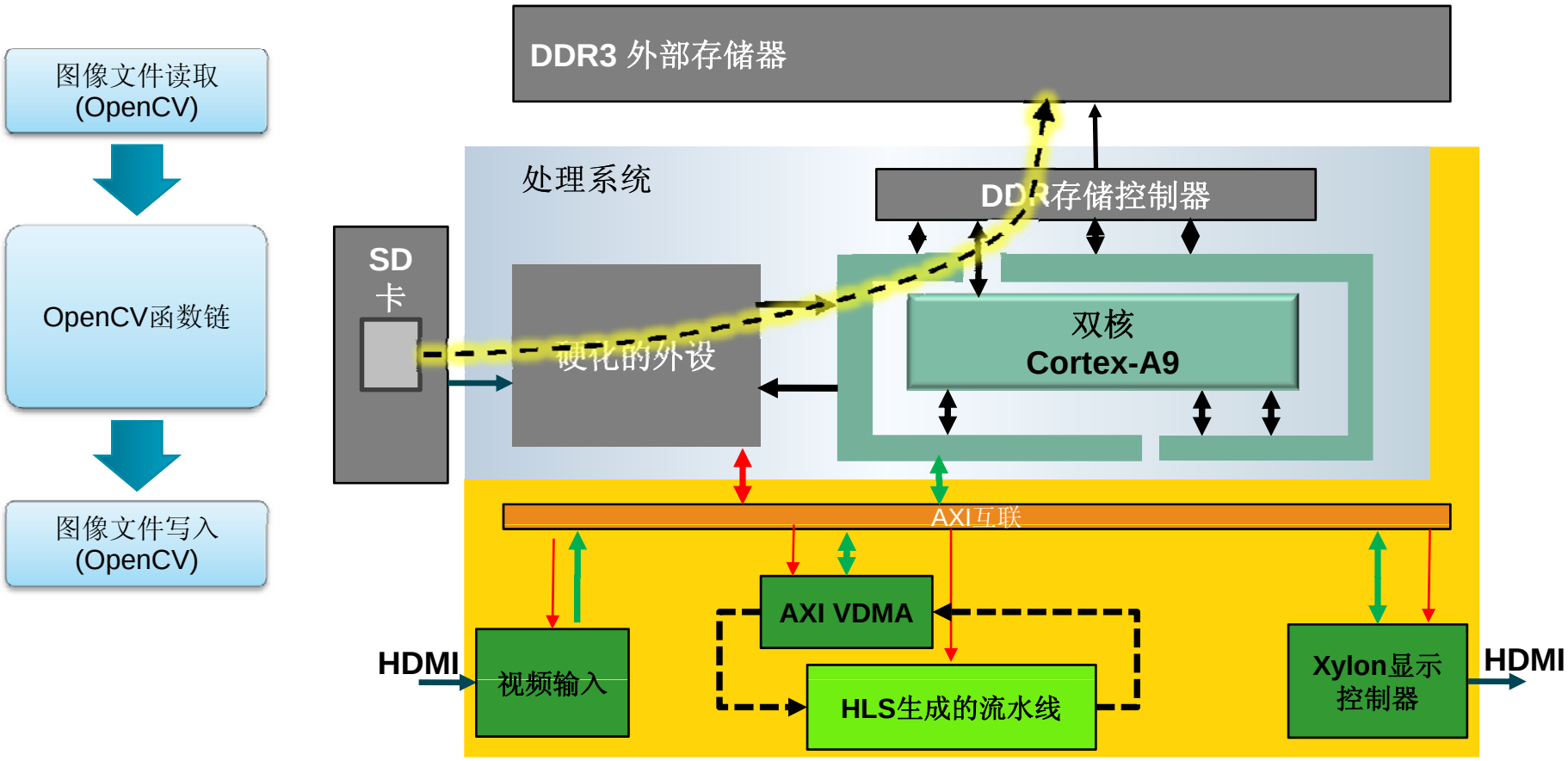
Vivado RTL集成



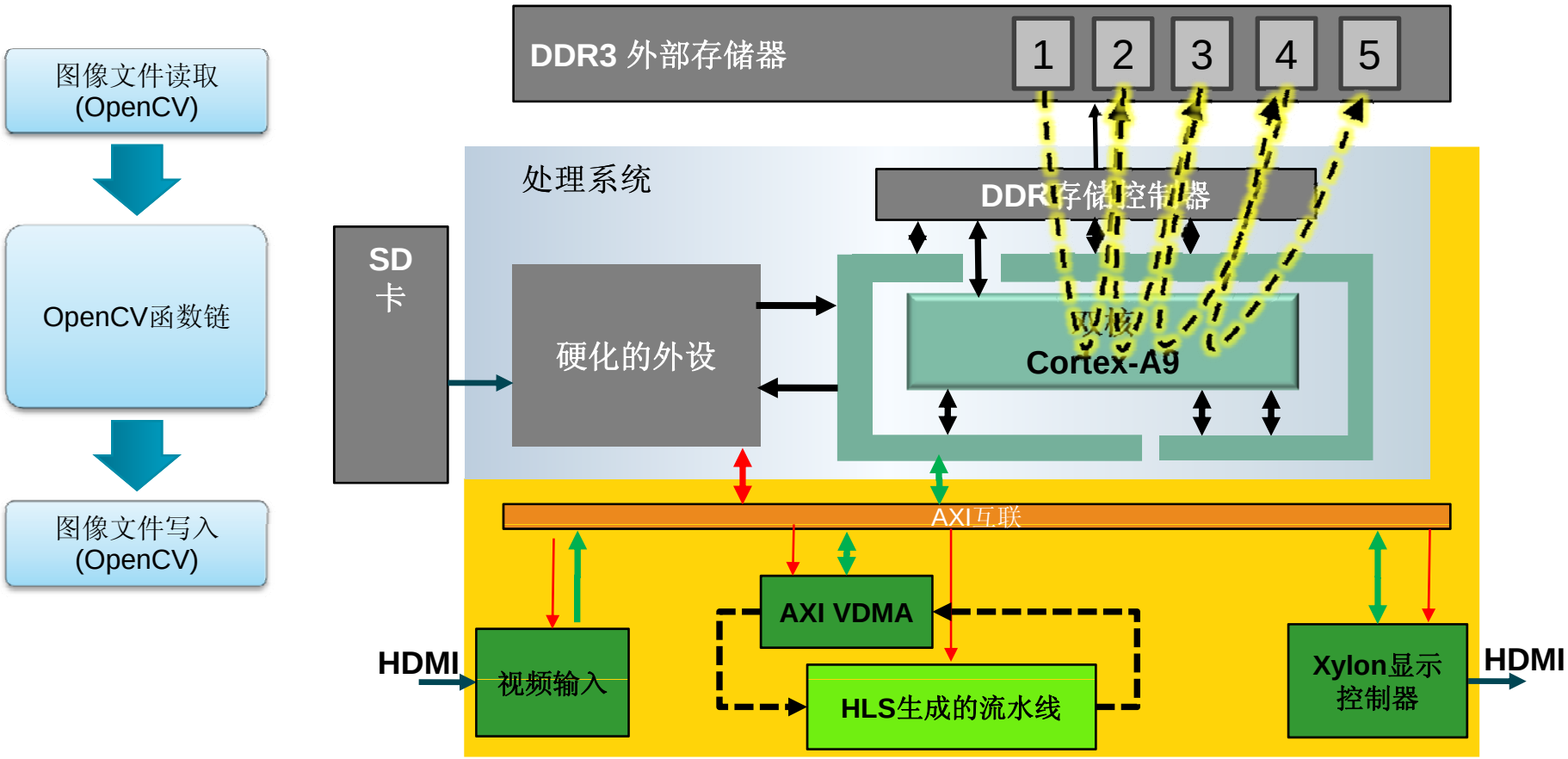
在FPGA设计中使用OpenCV



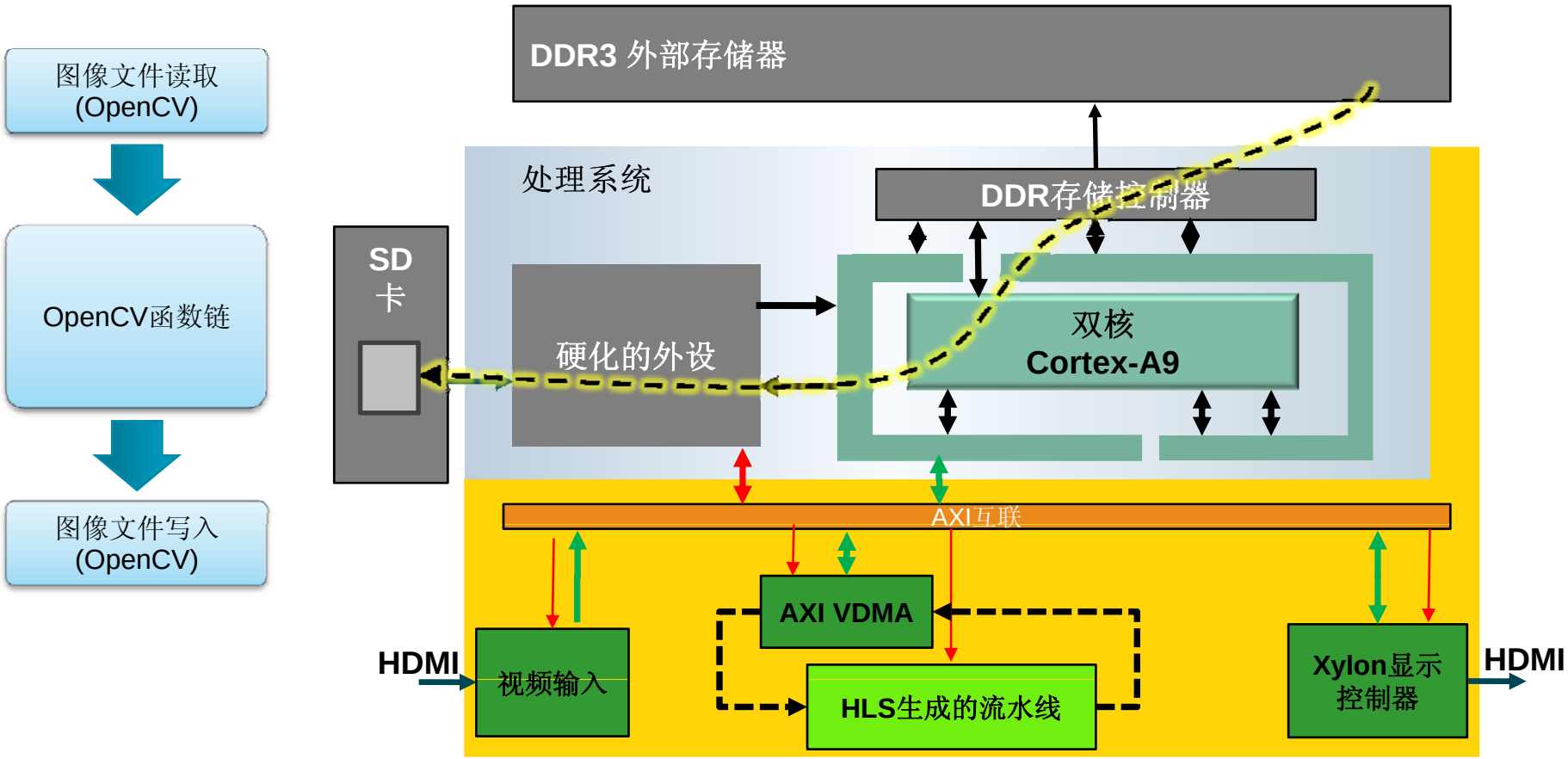
纯OpenCV应用



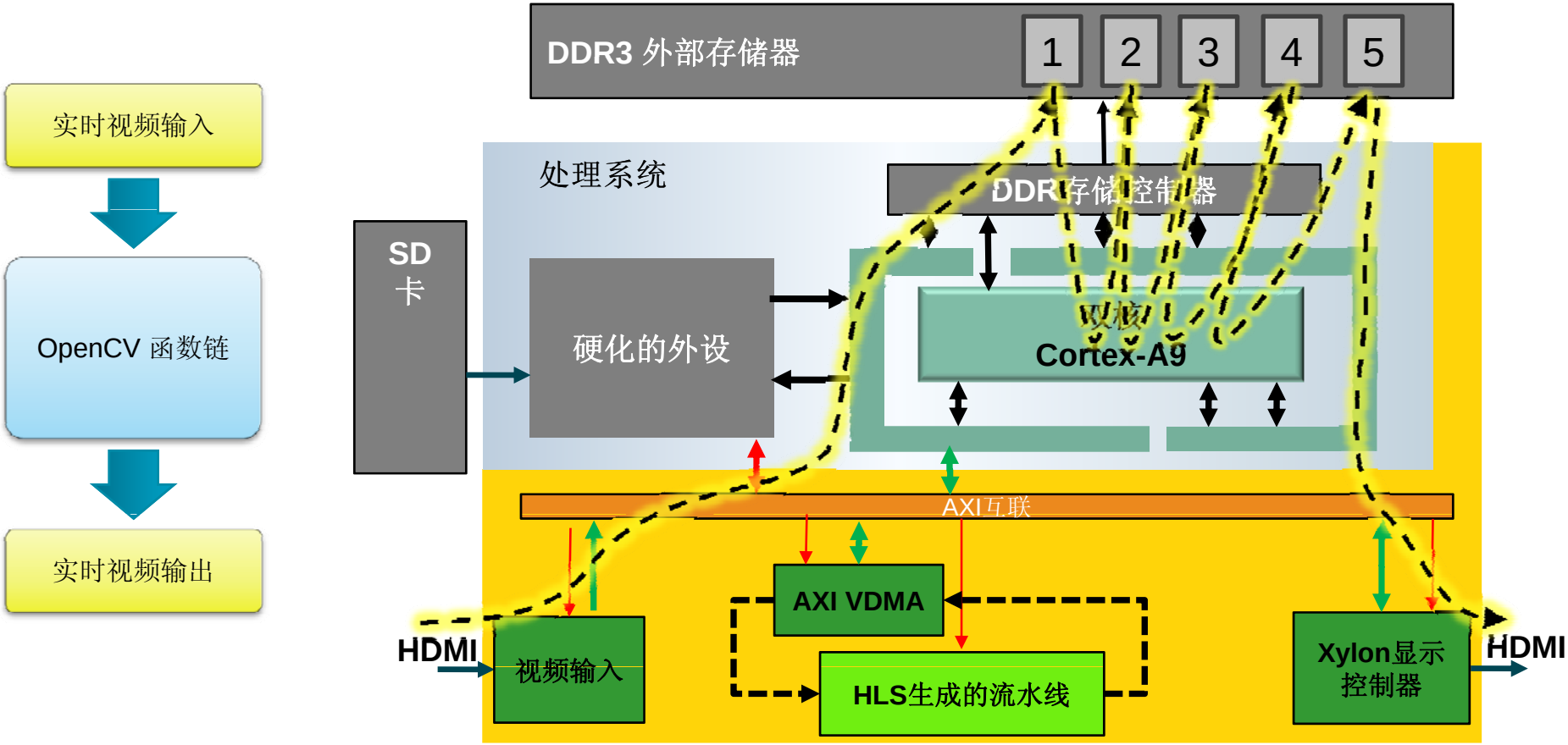
纯OpenCV应用



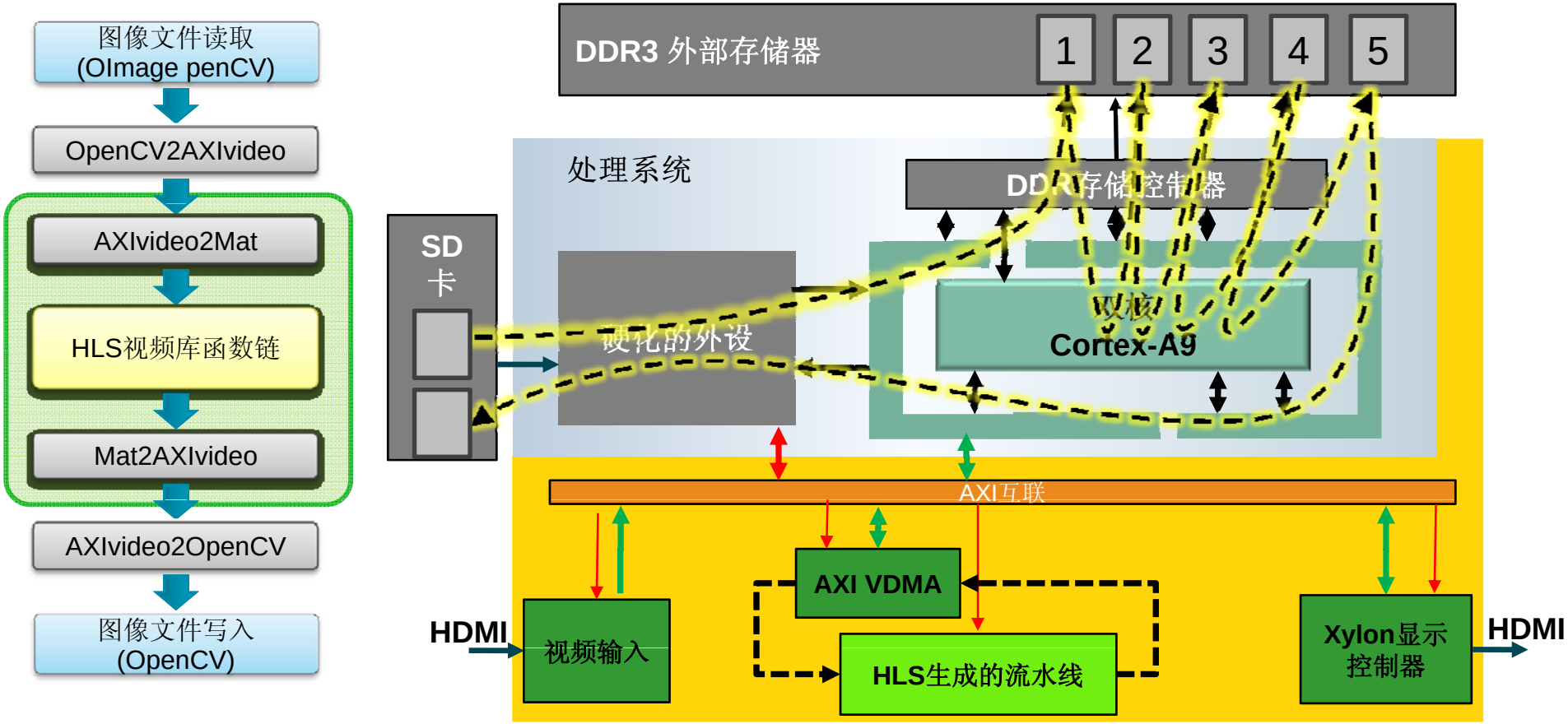
纯OpenCV应用



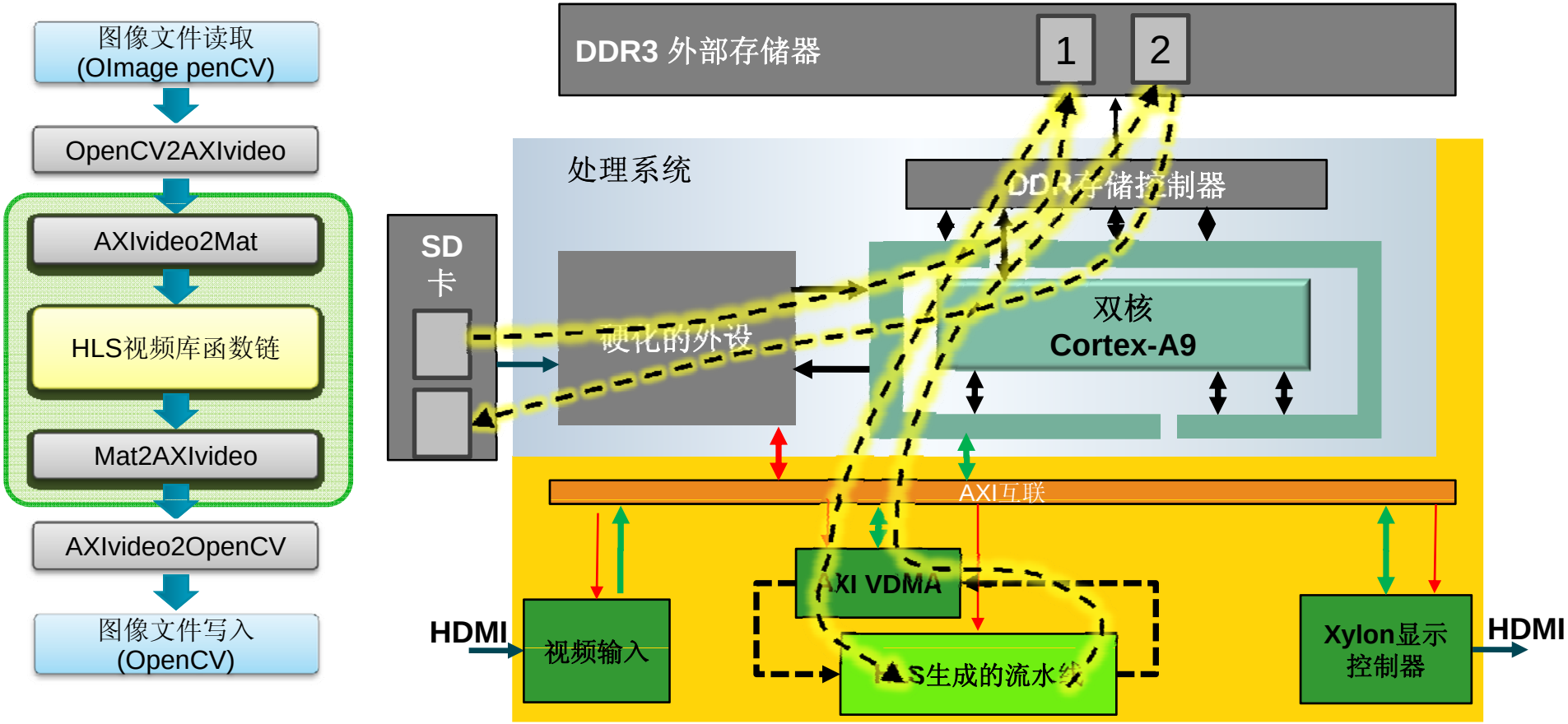
集成的OpenCV应用



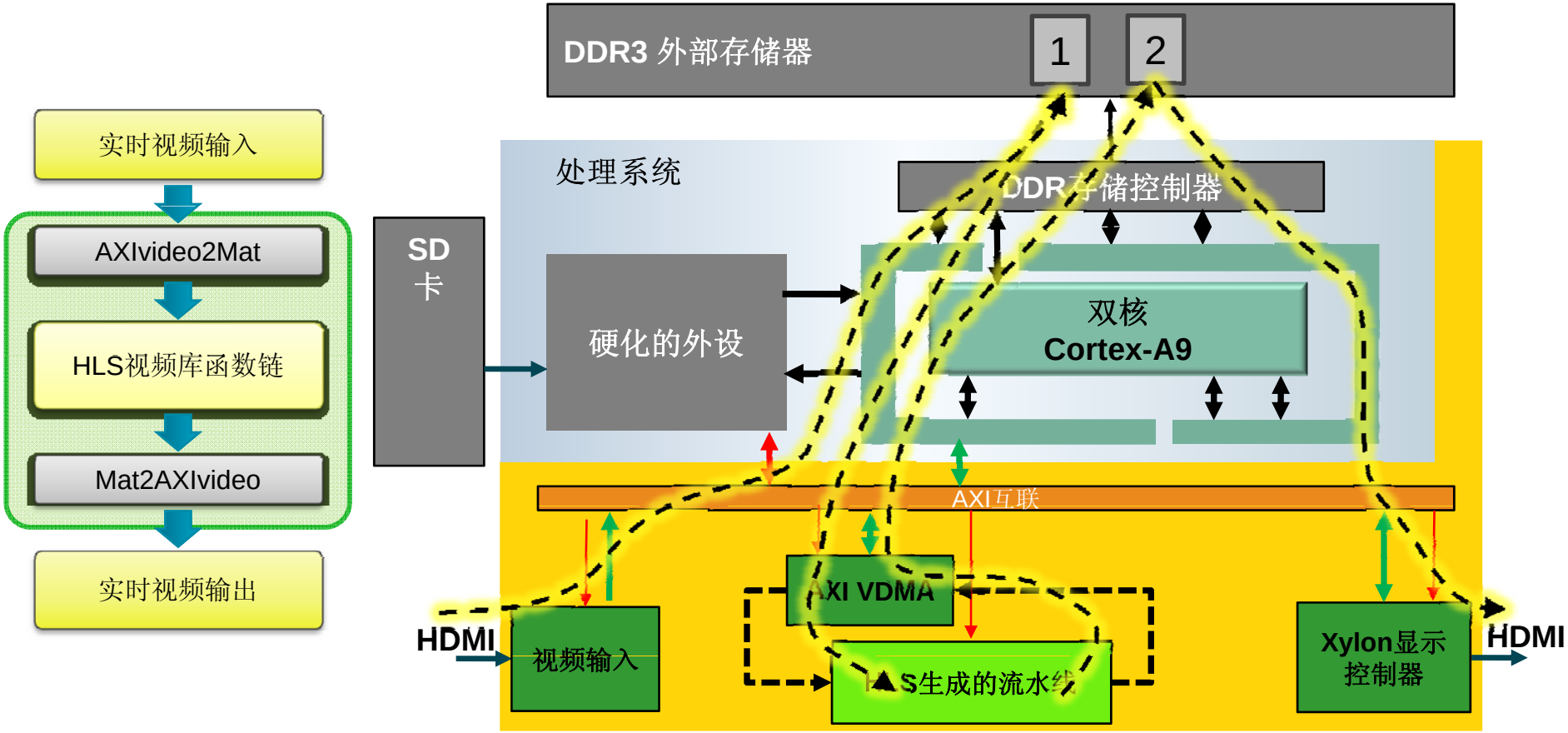
OpenCV参考/软件执行



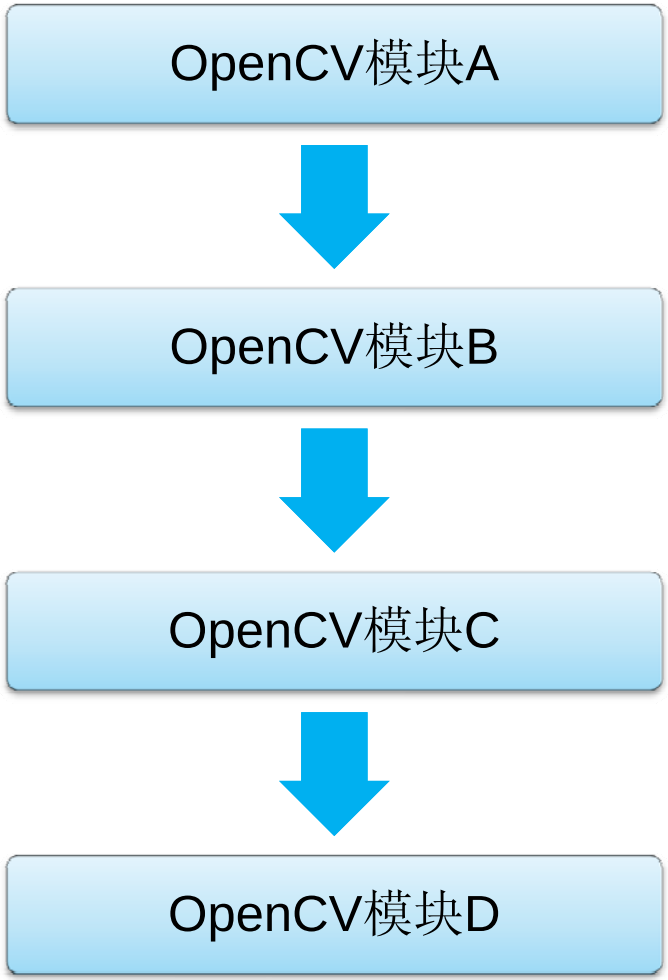
OpenCV参考/系统测试



加速的OpenCV应用

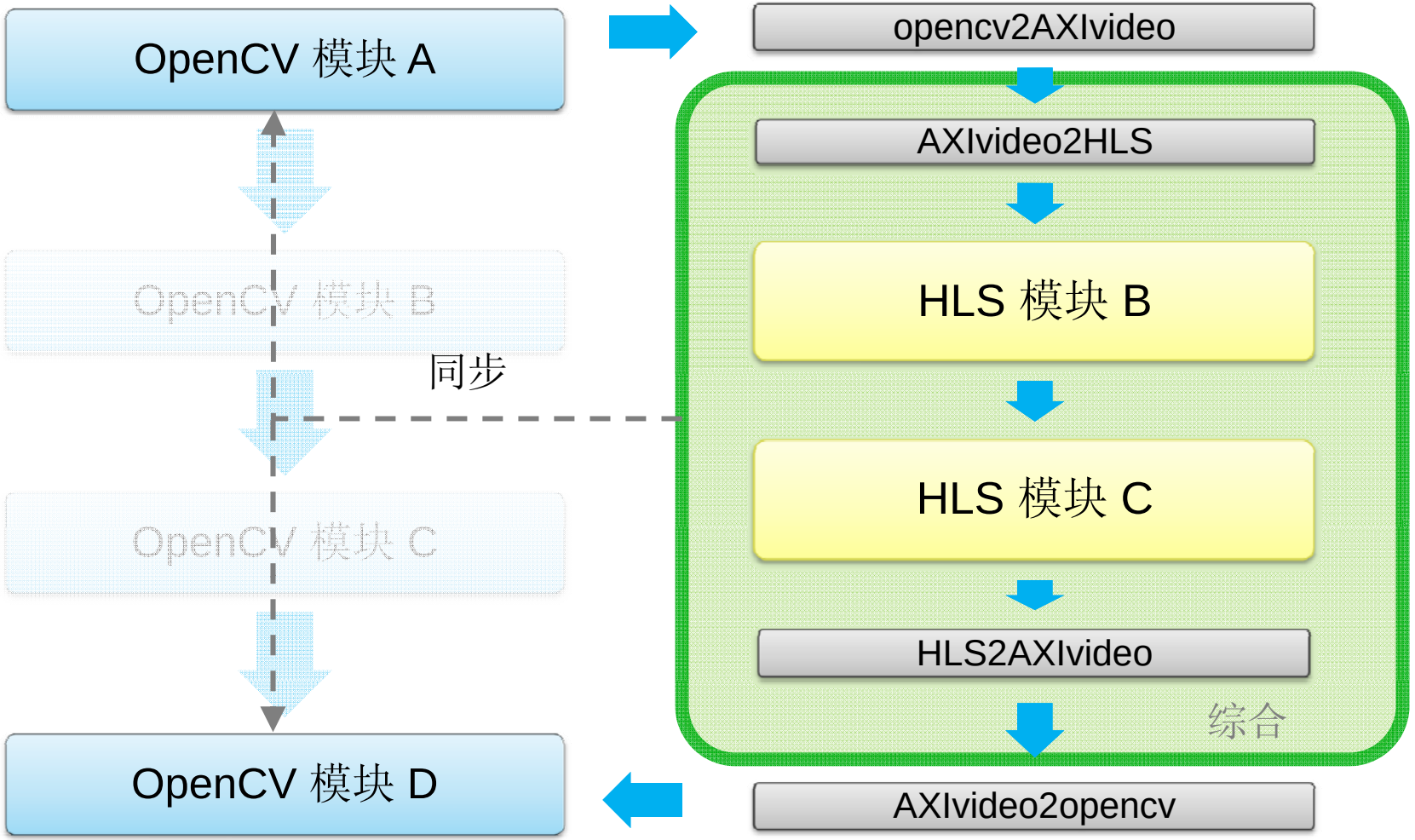


OpenCV设计流程



1) 在台式机上开发OpenCV应用
2) 无需修改即可在 ARM内核上运行OpenCV应用
3) 使用I/O函数抽象FPGA部分
4) 用可综合代码代替OpenCV函数调用
5) 运行HLS以生成 FPGA加速器
6) 用FPGA加速器调用代替可综合代码调用

OpenCV应用的软硬划分



OpenCV设计中的权衡

- **OpenCV图像处理是基于存储器帧缓存而构建的**
  - 访问局部性较差 -> 小容量高速缓存性能不足
  - 架构比较复杂（出于性能考虑） -> 功耗更高
  - 似乎足以满足很多应用的要求
    - 分辨率或帧速率低
    - 在更大的图像中对需要的特征或区域进行处理
- **基于视频流的架构能提供高性能和低功耗**
  - 链条化的图像处理函数能减少外部存储器访问
  - 针对视频优化的行缓存和窗口缓存比处理器高速缓存更简单
  - 可使用HLS中的数据流优化来实现
  - 需要将代码转换为可综合的代码

HLS视频库

- **OpenCV函数不能直接通过HLS进行综合**
  - 动态内存分配
  - 浮点
  - 假设图像在外部存储器中修改
- **HLS视频库用于替换很多基本的 OpenCV函数**
  - 与OpenCV具有相似的接口和算法
  - 主要针对在FPGA架构中实现的图像处理函数
  - 包含专门面向FPGA的优化
    - 定点运算而非浮点运算
    - 片上的行缓存和窗口缓存
  - 不必精确到比特位

赛灵思HLS视频库2013.2

视频数据建模	
Linebuffer Class	Window Class

AXI4-Stream IO 函数	
AXIvideo2Mat	Mat2AXIvideo

OpenCV 接口函数			
cvMat2AXIvideo	AXIvideo2cvMat	cvMat2hlsMat	hlsMat2cvMat
IplImage2AXIvideo	AXIvideo2IplImage	IplImage2hlsMat	hlsMat2IplImage
CvMat2AXIvideo	AXIvideo2CvMat	CvMat2hlsMat	hlsMat2CvMat

视频函数			
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

- 如需了解函数的详情，敬请阅读HLS用户指南 UG 902

视频库函数

- 包含在**hls**命名空间内的**C++**代码。**#include “hls_video.h”**
- 与**OpenCV**等具有相似的接口和等效的行为，例如
  - OpenCV库：
  - HLS视频库：

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

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

- 一些构造函数具有类似的或替代性的模板参数，例如
  - OpenCV库：
  - HLS视频库：

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

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

- **ROWS**和**COLS**指定处理的最大图像尺寸

视频库的核心结构

OpenCV	HLS 视频库
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>

局限性

- 必须用视频库函数代替OpenCV调用
- 不支持通过指针访问帧缓存
  - 使用VDMA和 AXI Stream adpater函数
- 不支持随机访问
  - 读取超过一次的数据必须进行复制
  - 请见**hls::Duplicate()**
- 不支持**In-place**更新
  - 例如 **cvRectangle (img, point1, point2)**

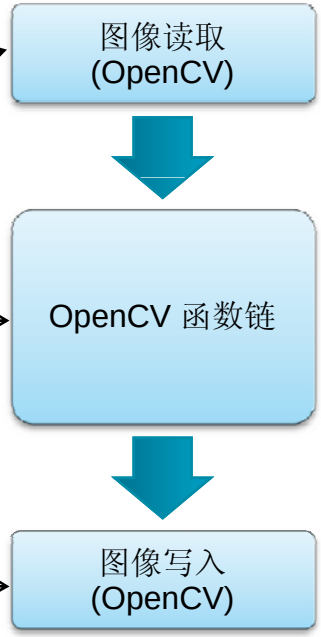
	OpenCV	HLS视频库
读操作	`pix = cv_mat.at<T>(i,j)` `pix = cvGet2D(cv_img,i,j)`	`hls_img >> pix`
写操作	`cv_mat.at<T>(i,j) = pix` `cvSet2D(cv_img,i,j,pix)`	`hls_img << pix`

OpenCV 代码

- 输入一个图像，输出一个图像
  - 按函数链顺序处理

```
…
IplImage* src=cvLoadImage("test_1080p.bmp");
IplImage* dst=cvCreateImage(cvGetSize(src),
                    src->depth, src->nChannels);


cvSobel(src, dst, 1, 0);
cvSubS(dst, cvScalar(100,100,100), src);
cvScale(src, dst, 2, 0);
cvErode(dst, src);
cvDilate(src, dst);


cvSaveImage("result_1080p.bmp", dst);
cvReleaseImage(&src);
cvReleaseImage(&dst);
…
```

test_opencv.cpp

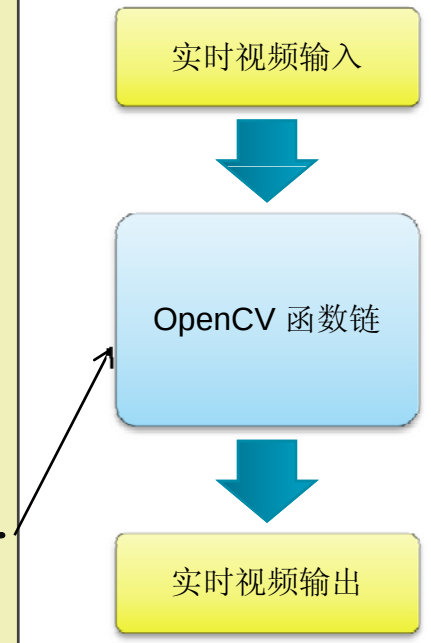


集成的OpenCV应用

- 系统提供指向帧缓存的指针
- 可综合代码也可在**ARM**上运行

```
void img_process(ZNQ_S32 *rgb_data_in, ZNQ_S32 *rgb_data_out, int
height, int width, int stride, int flag_OpenCV) {
  // constructing OpenCV interface
  IplImage* src_dma =
    cvCreateImageHeader(cvSize(width, height), IPL_DEPTH_8U, 4);
  IplImage* dst_dma =
    cvCreateImageHeader(cvSize(width, height), IPL_DEPTH_8U, 4);
  src_dma->imageData = (char*)rgb_data_in;
  dst_dma->imageData = (char*)rgb_data_out;
  src_dma->widthStep = 4 * stride;
  dst_dma->widthStep = 4 * stride;

  if (flag_OpenCV) {
    opencv_image_filter(src_dma, dst_dma);
  } else {
    sw_image_filter(src_dma, dst_dma);
  }

  cvReleaseImageHeader(&src_dma);
  cvReleaseImageHeader(&dst_dma);
}
```

img_filters.c



使用Vivado HLS视频库加速

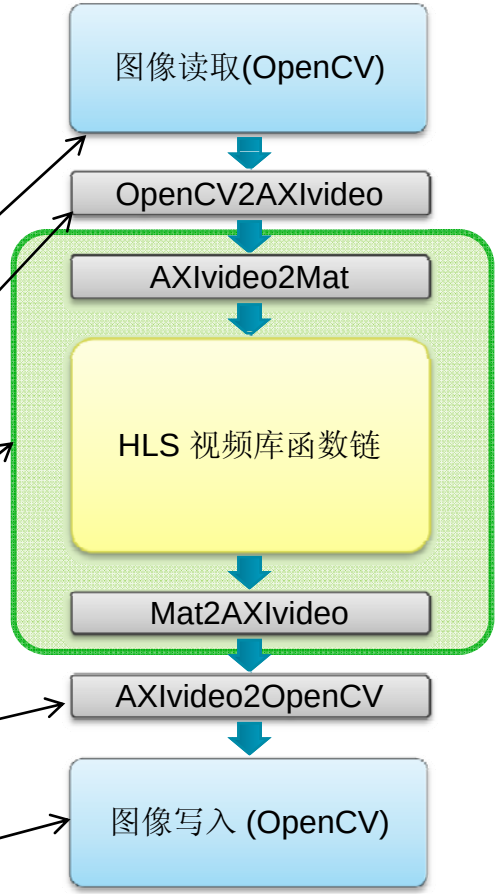
- 抽取顶层函数进行硬件加速

```
#include “hls_video.h”  // header file of HLS video library
#include “hls_opencv.h” // header file of OpenCV I/O

// typedef video library core structures
typedef hls::stream<ap_axiu<32,1,1,1> >   AXI_STREAM;
typedef hls::Scalar<3, uchar>             RGB_PIXEL;
typedef hls::Mat<1080,1920,HLS_8UC3>      RGB_IMAGE;

void image_filter(AXI_STREAM& src_axi, AXI_STREAM& dst_axi,
         int rows, int cols);
```

top.h

```
#include “top.h”
…

IplImage* src=cvLoadImage("test_1080p.bmp");
IplImage* dst=cvCreateImage(cvGetSize(src),
                  src->depth, src->nChannels);

AXI_STREAM src_axi, dst_axi;
IplImage2AXIvideo(src, src_axi);

image_filter(src_axi, dst_axi, src->height, src->width);

AXIvideo2IplImage(dst_axi, dst);

cvSaveImage("result_1080p.bmp", dst);
cvReleaseImage(&src);
cvReleaseImage(&dst);
```

test.cpp

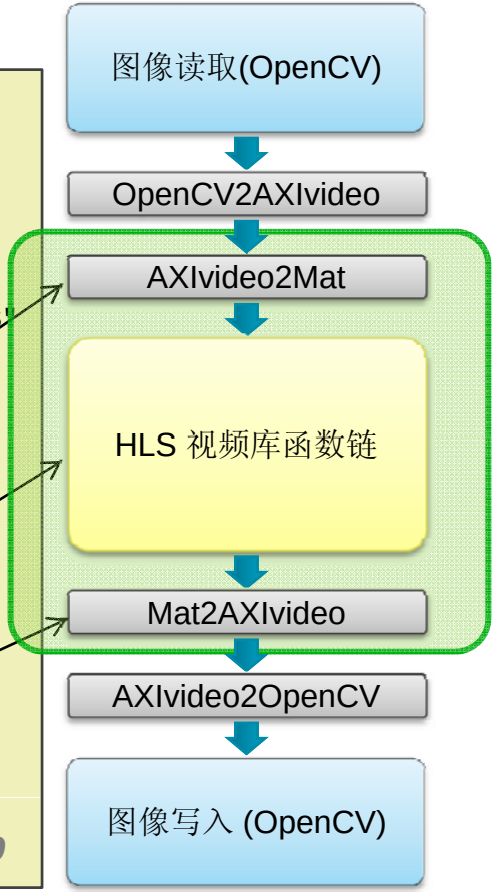


使用Vivado HLS视频库加速

- 用于FPGA加速的硬件可综合模块
  - 由视频库函数与接口组成
  - 用hls命名空间中的相似函数代替OpenCV函数

```
void image_filter(AXI_STREAM& input, AXI_STREAM& output, int rows, int cols) {
    //Create AXI streaming interfaces for the core

#pragma HLS RESOURCE variable=input core=AXIS metadata="-bus_bundle INPUT_STREAM"
#pragma HLS RESOURCE variable=output core=AXIS metadata="-bus_bundle OUTPUT_STREAM"
#pragma HLS RESOURCE variable=rows core=AXI_SLAVE metadata="-bus_bundle CONTROL_BUS"
#pragma HLS RESOURCE variable=cols core=AXI_SLAVE metadata="-bus_bundle CONTROL_BUS"
#pragma HLS RESOURCE variable=return core=AXI_SLAVE metadata="-bus_bundle CONTROL_BUS"
#pragma HLS INTERFACE ap_stable port=rows
#pragma HLS INTERFACE ap_stable port=cols

    RGB_IMAGE img_0(rows, cols), img_1(rows, cols), img_2(rows, cols);
    RGB_IMAGE img_3(rows, cols), img_4(rows, cols), img_5(rows, cols);
    RGB_PIXEL pix(50, 50, 50);
#pragma HLS dataflow
    hls::AXIvideo2Mat(input, img_0);
    hls::Sobel<1,0,3>(img_0, img_1);
    hls::SubS(img_1, pix, img_2);
    hls::Scale(img_2, img_3, 2, 0);
    hls::Erode(img_3, img_4);
    hls::Dilate(img_4, img_5);
    hls::Mat2AXIvideo(img_5, output);
}
```

top.cpp



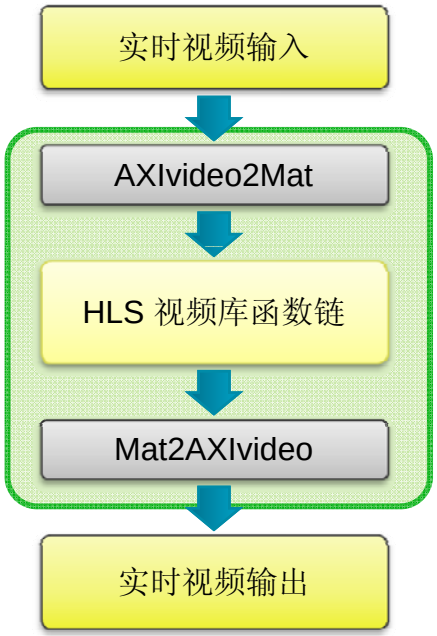
使用 Linux 用户空间的API

- 修改器件树，使其包含寄存器映射

```
FILTER@0x400D0000 {
        compatible = "xlnx,generic-hls";
        reg = <0x400d0000 0xffff>;
        interrupts = <0x0 0x37 0x4>;
        interrupt-parent = <0x1>;
};
```

- 执行mmap()函数后从用户空间调用

```
Ximage_filter xsfilter;
int fd_uio = 0;

if ((fd_uio = open("/dev/uio0", O_RDWR)) < 0) {
        printf("UIO: Cannot open device node\n");
}

xsfilter.Control_bus_BaseAddress =
        (u32)mmap(NULL, XSOBEL_FILTER_CONTROL_BUS_SIZE,
                  PROT_READ|PROT_WRITE, MAP_SHARED, fd_uio, 0);
xsfilter.IsReady = XIL_COMPONENT_IS_READY;

// init the configuration for image filter
XImage_filter_SetRows(&xsfilter, sobel_configuration.height);
XImage_filter_SetCols(&xsfilter, sobel_configuration.width);
XImage_filter_EnableAutoRestart(&xsfilter);
XImage_filter_Start(&xsfilter);
```



HLS视频处理指令

- 将“input”指定为以 “INPUT_STREAM” 命名的AXI4 Stream

```
#pragma HLS RESOURCE variable=input core=AXIS metadata="-bus_bundle INPUT_STREAM"
```

- 将控制接口分配到AXI4-Lite接口

```
#pragma HLS RESOURCE variable=return core=AXI_SLAVE metadata="-bus_bundle CONTROL_BUS"
```

- 指定“rows”可通过AXI4-Lite接口进行访问

```
#pragma HLS RESOURCE variable=rows core=AXI_SLAVE metadata="-bus_bundle CONTROL_BUS"
```

- 声明在函数执行过程中 “rows”不会改变

```
#pragma HLS INTERFACE ap_stable port=rows
```

- 启用数据流优化

```
#pragma HLS dataflow
```

更复杂的OpenCV实例：快速角点

- 代码不是“**streaming**”，必须重写
  - 随机访问以及 “dst” in-place运算

```
void opencv_image_filter(IplImage* img, IplImage* dst ) {
  IplImage* gray = cvCreateImage(cvSize(img->width,img->height), 8, 1 );
  cvCvtColor( img, gray, CV_BGR2GRAY );
  std::vector<cv::KeyPoint> keypoints;
  cv::Mat gray_mat(gray,0);
  cv::FAST(gray_mat, keypoints, 20,true );
  int rect=2;
  cvCopy(img,dst);
  for (int i=0; i<keypoints.size(); i++) {
    cvRectangle(dst,
      cvPoint(keypoints[i].pt.x,keypoints[i].pt.y),
      cvPoint(keypoints[i].pt.x+rect,keypoints[i].pt.y+rect),
    cvScalar(255,0,0),1);
  }
  cvReleaseImage( &gray );
}
```

opencv_top.cpp

更复杂的OpenCV实例：快速角点

- 代码是“Streaming”
  - 要注意函数不是1:1对应！

```
void opencv_image_filter(IplImage* src, IplImage* dst)
{
    IplImage* gray = cvCreateImage( cvGetSize(src), 8, 1 );
    IplImage* mask = cvCreateImage( cvGetSize(src), 8, 1 );
    IplImage* dmask = cvCreateImage( cvGetSize(src), 8, 1 );
    std::vector<cv::KeyPoint> keypoints;
    cv::Mat gray_mat(gray,0);

    cvCvtColor(src, gray, CV_BGR2GRAY );
    cv::FAST(gray_mat, keypoints, 20, true);
    GenMask(mask, keypoints);
    cvDilate(mask,dmask);
    cvCopy(src,dst);
    PrintMask(dst,dmask,cvScalar(255,0,0));

    cvReleaseImage( &mask );
    cvReleaseImage( &dmask );
    cvReleaseImage( &gray );
}
```

hls::FASTX

hls::PaintMask

opencv_top.cpp

更复杂的OpenCV实例：快速角点

- 可综合的代码
  - 注意“#pragma HLS stream”

```
    hls::Mat<MAX_HEIGHT,MAX_WIDTH,HLS_8UC3>      _src(rows,cols);
    hls::Mat<MAX_HEIGHT,MAX_WIDTH,HLS_8UC3>      _dst(rows,cols);
    hls::AXIvideo2Mat(input, _src);
    hls::Mat<MAX_HEIGHT,MAX_WIDTH,HLS_8UC3>      src0(rows,cols);
    hls::Mat<MAX_HEIGHT,MAX_WIDTH,HLS_8UC3>      src1(rows,cols);
#pragma HLS stream depth=20000 variable=src1.data_stream
    hls::Mat<MAX_HEIGHT,MAX_WIDTH,HLS_8UC1>      mask(rows,cols);
    hls::Mat<MAX_HEIGHT,MAX_WIDTH,HLS_8UC1>      dmask(rows,cols);
    hls::Scalar<3,unsigned char> color(255,0,0);
    hls::Duplicate(_src,src0,src1);
    hls::Mat<MAX_HEIGHT,MAX_WIDTH,HLS_8UC1>      gray(rows,cols);
    hls::CvtColor<HLS_BGR2GRAY>(src0,gray);
    hls::FASTX(gray,mask,20,true);
    hls::Dilate(mask,dmask);
    hls::PaintMask(src1,dmask,_dst,color);
    hls::Mat2AXIvideo(_dst, output);
```

top.cpp

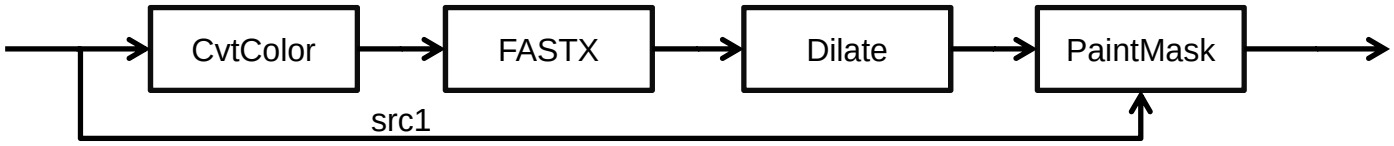
流与再收敛路径

- **hls::Mat**从概念上讲代表一个完整图像，但以像素流的形式执行

```
template<int ROWS, int COLS, int T> class Mat {
public:
    HLS_SIZE_T rows, cols;
    hls::stream<HLS_TNAME(T)> data_stream[HLS_MAT_CN(T)];
};
```

hls_video_core.h

- 快速角点包含一条*再收敛路径*
  - src1像素流必须包含足够的缓冲空间，用以匹配FASTX和Dilate的延迟（大约10个视频行 * 1920像素）



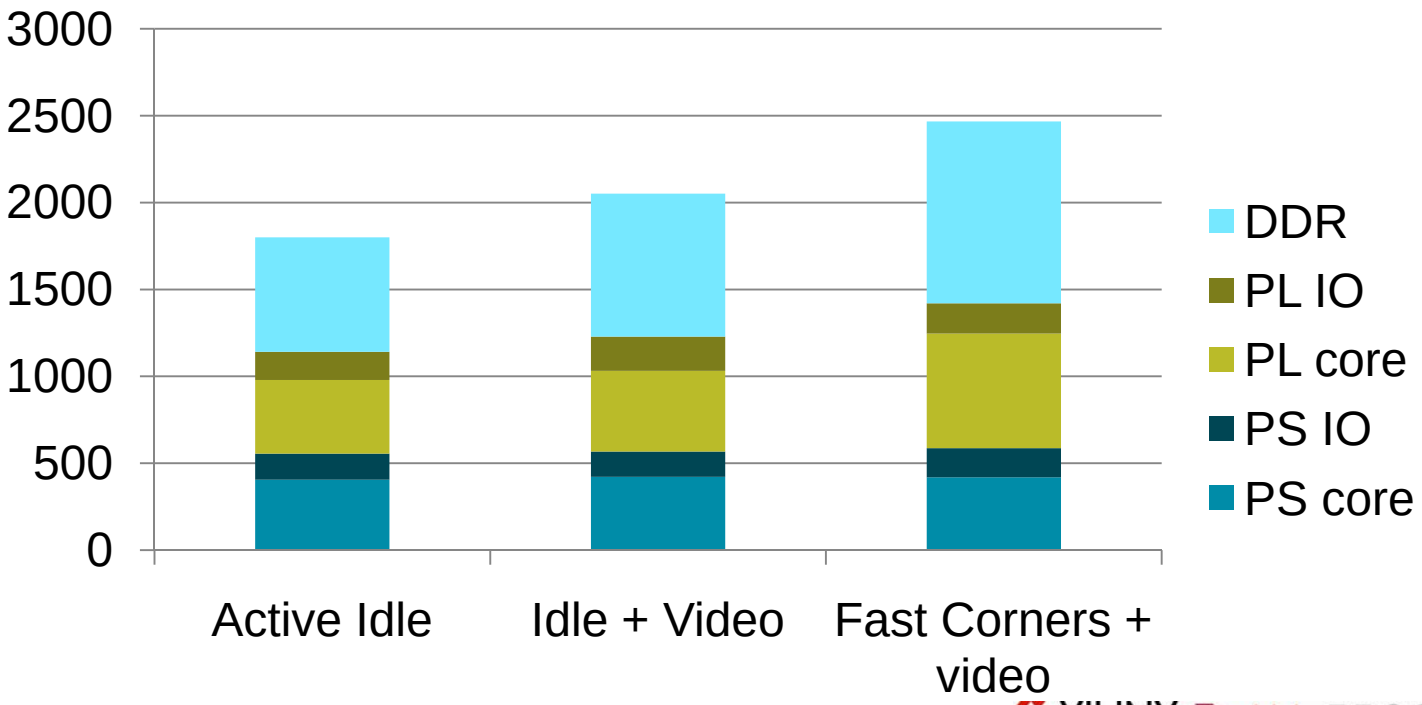

```
#pragma HLS stream depth=20000 variable=src1.data_stream
```

性能分析

- **AXI**性能监视器可以从存储器带宽收集统计数据
  - 请查看 /mnt/AXI_PerfMon.log
- 视频 + 快速角点
  - 1920*1080*60*32 = ~4 Gb/s per stream
  - **HP0: 读 4.01 Gb/s, 写 4.01 Gb/s, 总共 8.03 Gb/s**
  - **HP2: 读 4.01 Gb/s, 写 4.01 Gb/s, 总共 8.03 Gb/s**

功耗分析

- 可从ZC702开发板上的数字调功器读取电压和电流。
- 用2-3瓦的系统总功耗即可实现定制的实时高清视频处理
  - FASTX所增加的功耗不足 200mW

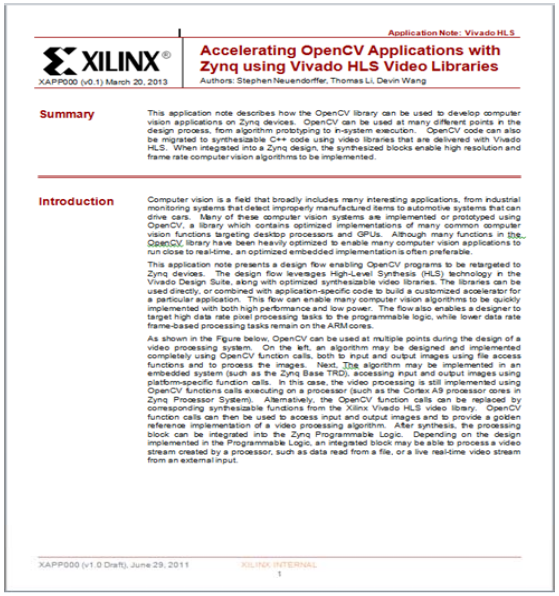


HLS与Zynq加速OpenCV应用

- OpenCV函数可实现计算机视觉算法的快速原型设计
- 计算机视觉应用与生俱来的异构特性，使其需要软硬件相结合的实现方案
- Vivado HLS视频库能加快OpenCV函数向FPGA可编程架构的映射
- Zynq采用高性能可编程逻辑和嵌入式ARM内核，是一款功耗优化的集成式解决方案

其他有关OpenCV的补充资料

下载 XAPP1167



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://china.xilinx.com/hls

http://china.xilinx.com/getlicense

QuickTake系列视频: 使用Vivado提升OpenCV和HLS