MATLAB EXPO 2018

将算法部署到 FPGA/ASIC 硬件

单博

提纲

- 何时考虑使用 FPGA 、 ASIC 或片上系统 (SoC) 硬件
- 算法的硬件实现，所需考虑的问题
- 从系统 / 算法到 FPGA/ASIC 硬件的流程
- 案例：视觉处理算法部署到 FPGA 硬件
- 结论

为什么将算法部署到 FPGA/ASIC 硬件上？

Speed

“Real-time image processing for an aircraft head’s up display”

“Evaluate the algorithm in field testing to analyze system performance”

“Optimal performance @ Piezo resonance frequency”

Power

“11 year device with a 1 A*hr battery”

Latency

“Be able to stop the robot with millimeter accuracy in less than 0.5 seconds without causing damage to the robot”

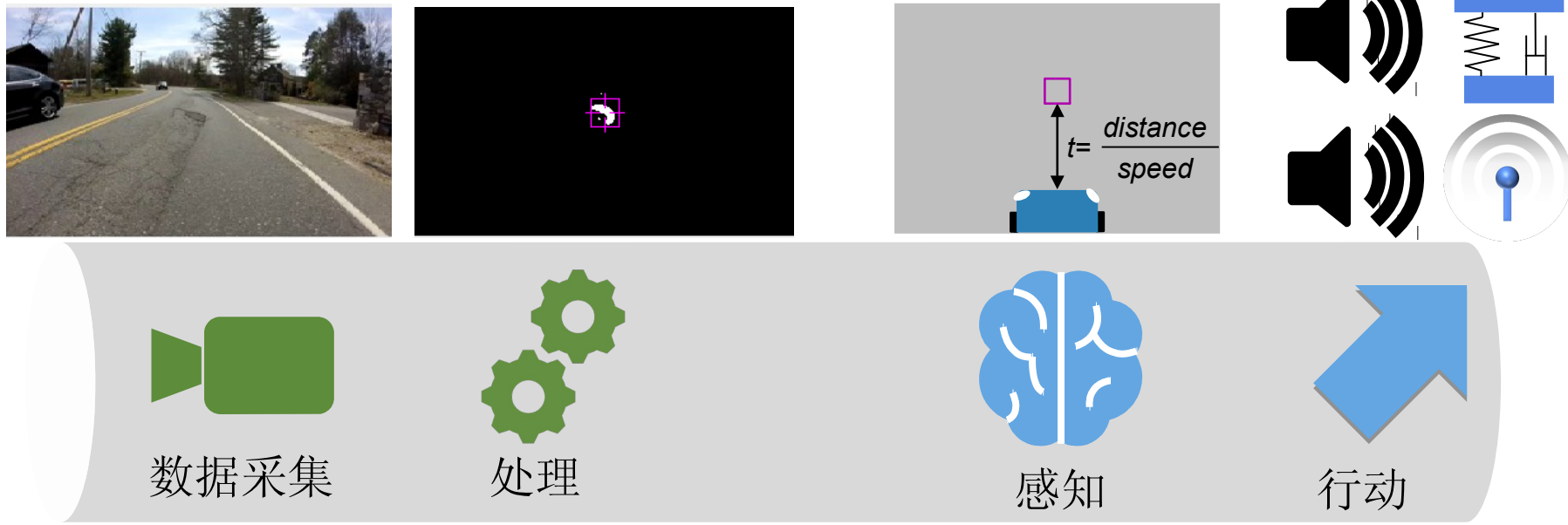
“Audio transducer prototypes must run in real time with low latencies”

“Motor control latency < 1us”

We need to get to market quickly, but we have no experience designing FPGAs!

现代应用通常需要定制化硬件

ADAS 应用案例：



1M+ 像素 / 帧

高速，明确定义的
- 循环往复处理
- 海量数 Large amount of data

FPGA 硬件

少量坐标结果

复杂，更加灵活
- 少量数据计算
- 执行控制

Embedded 软件

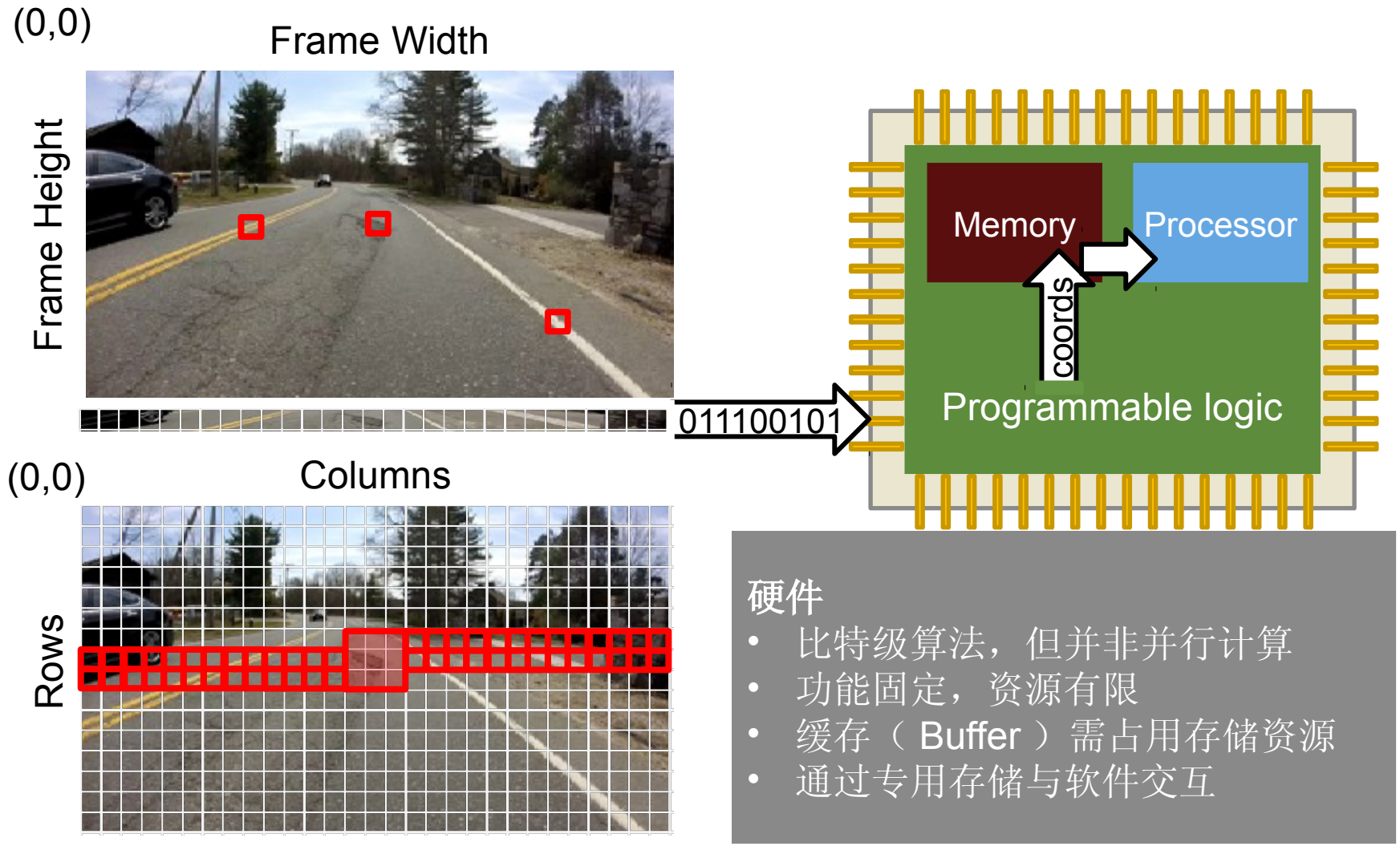
提纲

- 何时考虑使用 FPGA, ASIC, or System-on-Chip (SoC) 硬件
- 算法的硬件实现需考虑的问题
- 从系统 / 算法到 FPGA/ASIC 硬件的流程
- 演示：视觉处理算法部署到 FPGA 硬件
- 结论

基于帧的算法 vs 基于数据流的算法

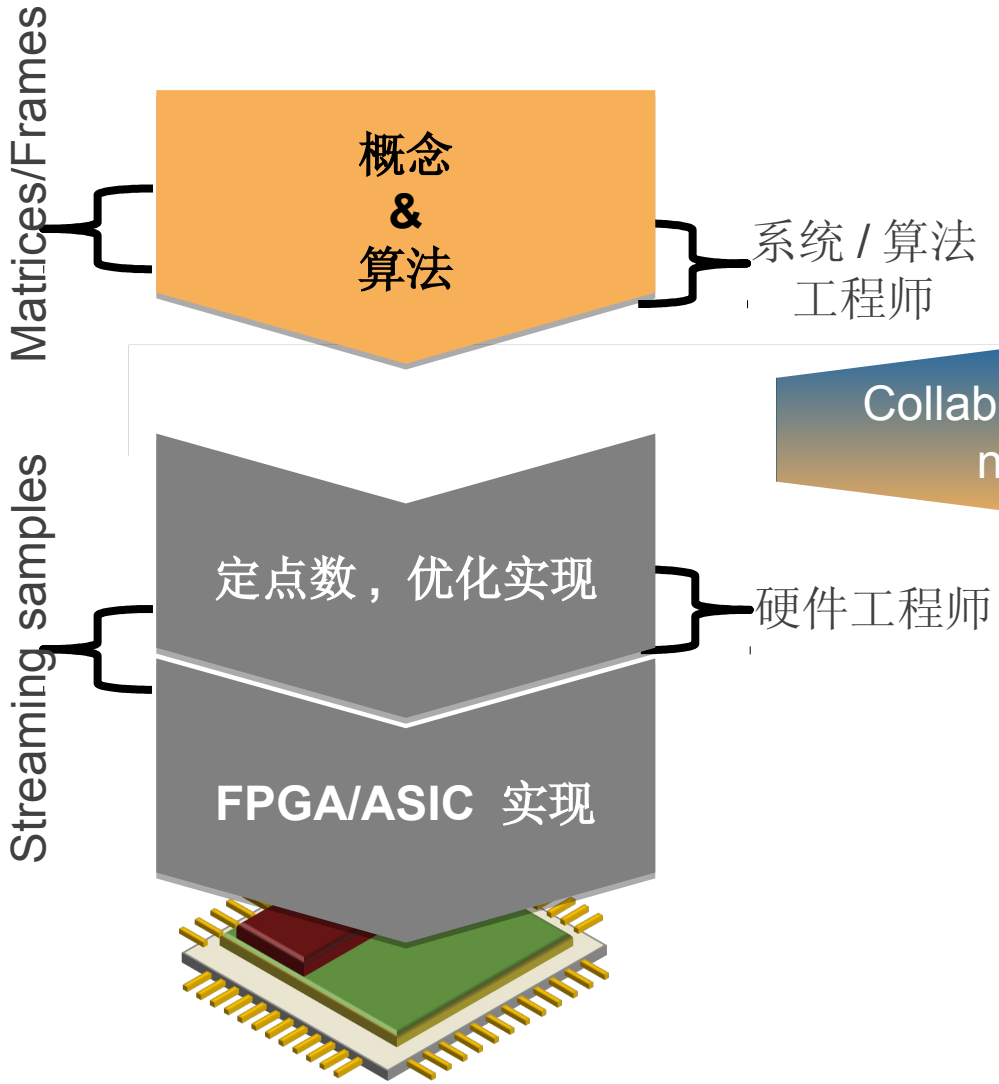
基于帧的算法
- 每次一整帧
- 可随机访问任意 [x,y] 坐标的像素

基于流的算法
- 逐个采样值，从上至下逐行
- 感兴趣区域 (ROI) 存储在多个行缓存里



硬件
- 比特级算法，但并非并行计算
- 功能固定，资源有限
- 缓存（Buffer）需占用存储资源
- 通过专用存储与软件交互

弥合从算法到实现的鸿沟



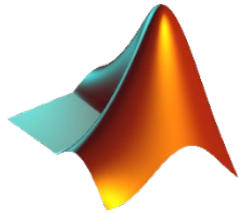
概念 & 算法
- 开发系统级算法
- 仿真、分析、修改
- 划分软 / 硬件边界

算法 到 微观架构
- 转化为流算法
- 添加硬件微观结构
- 转化为定点数据类型

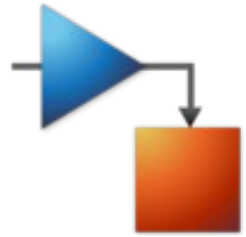
微观架构 到 实现
- 速度和面积优化
- HDL 代码生成
- 验证
- FPGA/ASIC 实现

提纲

- 何时考虑使用 FPGA, ASIC, or System-on-Chip (SoC) 硬件
- 算法的硬件实现，所需考虑的问题
- 从系统 / 算法到 FPGA/ASIC 硬件的流程
- 演示：视觉处理算法部署到 FPGA 硬件
- 结论



MATLAB



Simulink

- ✓ 支持大数据集
- ✓ 探索算法架构
- ✓ 数据可视化

- ✓ 并发式架构
- ✓ 具有严格时序
- ✓ 数据类型自动传输

算法
(Golden Reference)

算法软 / 硬件实现

定点数，优化实现

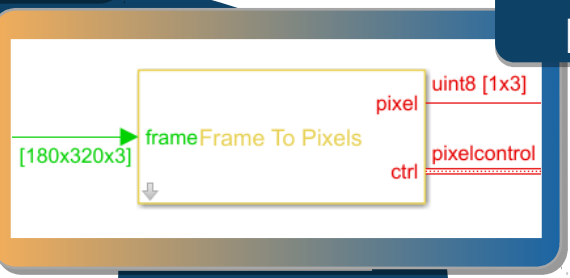
FPGA/ASIC 实现

MATLAB

Simulink

```
%% Frame pre-processing

    % Convert to intensity
    frmGray = rgb2gray(frmIn);

    % Bilateral filter
    frmBiFilt = imbilatfilt(frmGray, 'NeighborhoodSize', 9);

    % Edge detection
    frmEdge = edge(frmBiFilt,'Sobel', .05);

%% Trapezoidal mask
```

HDL-ready
IP blocks

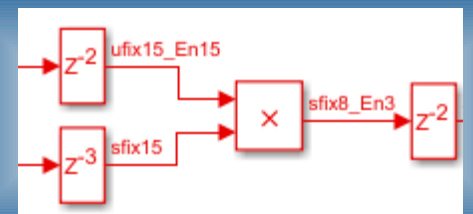


验证

HDL Coder
原型

Fixed Point
Designer

HDL Coder



HDL Coder
产品

Core
interface

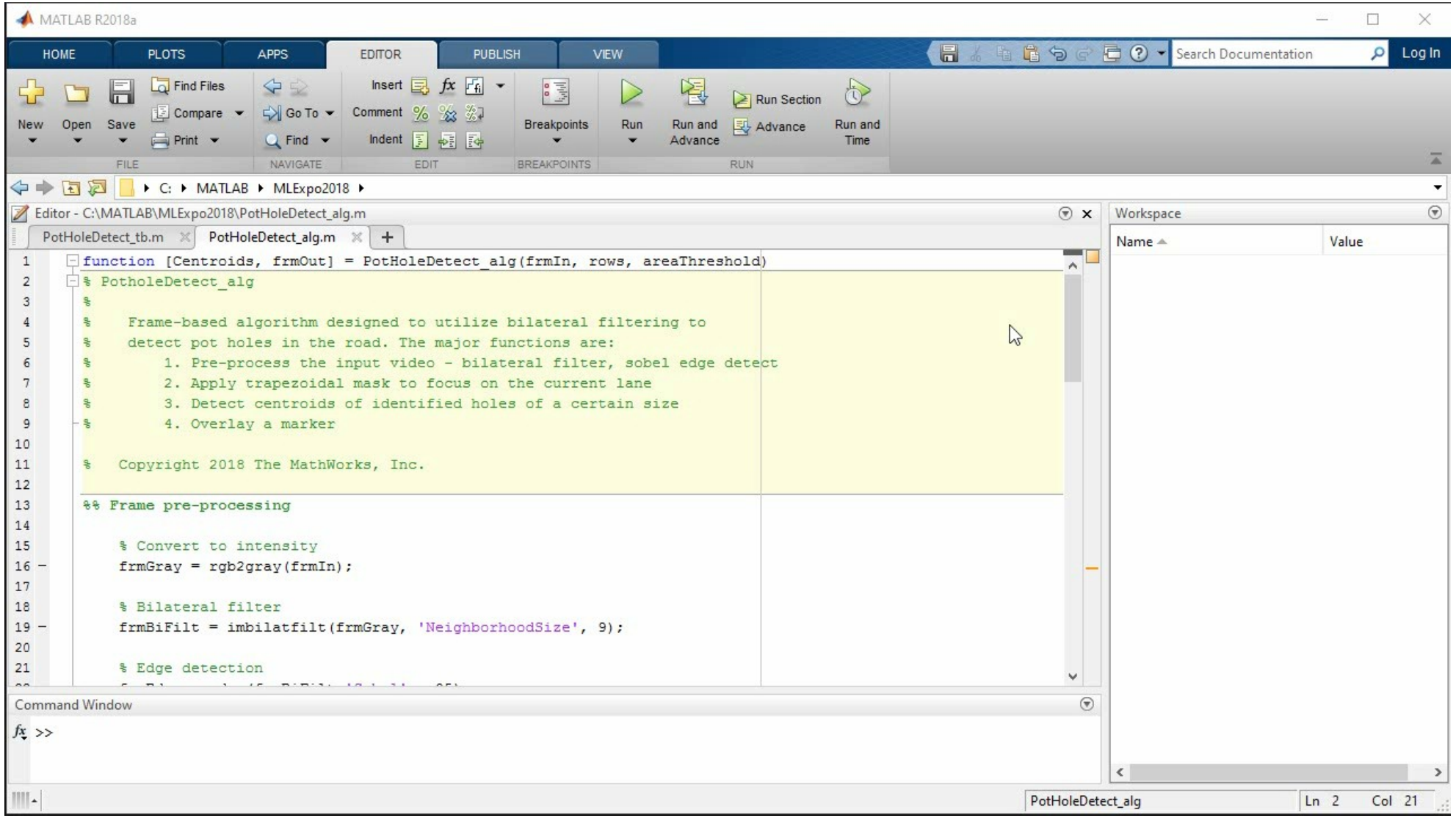
VHDL /
Verilog

Constraints

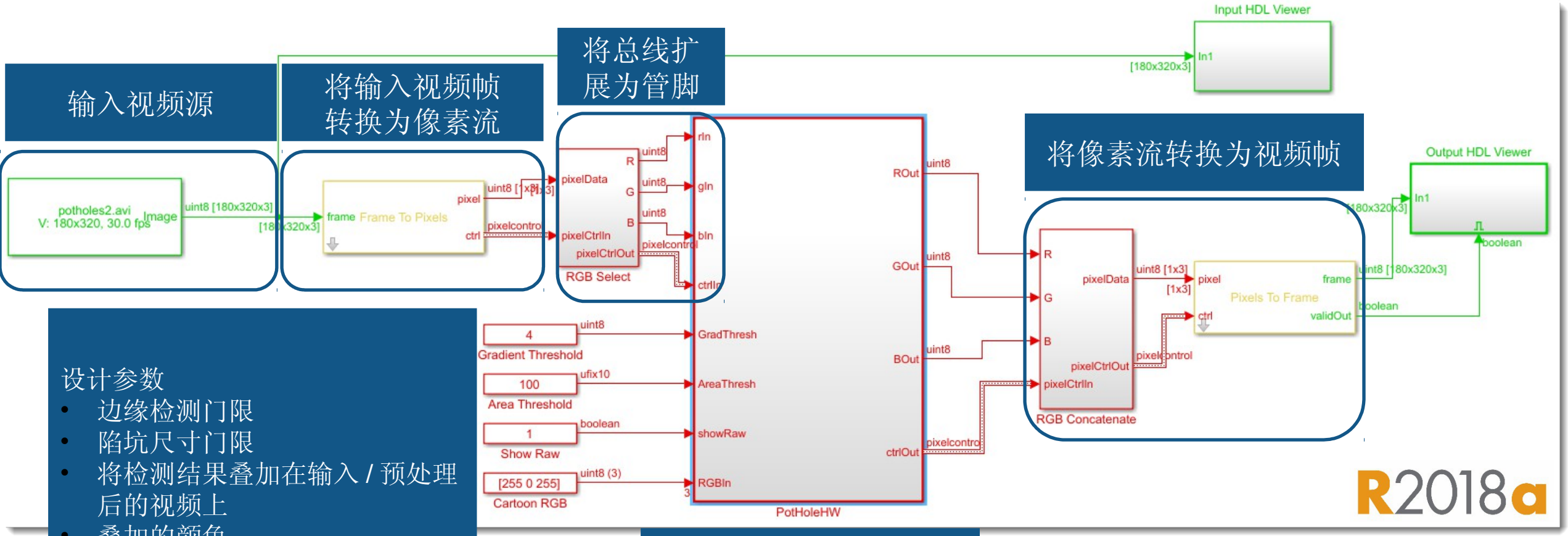
提纲

- 何时考虑使用 FPGA, ASIC, or System-on-Chip (SoC) 硬件
- 算法的硬件实现，所需考虑的问题
- 从系统 / 算法到 FPGA/ASIC 硬件的流程
- 演示：视觉处理算法部署到 FPGA 硬件
- 结论

算法简介


MATLAB R2018a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Log In
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
Run
Run and Advance
Run Section
Advance
Run and Time
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
C: ▸ MATLAB ▸ MLExpo2018 ▸
Editor - C:\MATLAB\MLExpo2018\PotHoleDetect_alg.m
PotHoleDetect_tb.m
PotHoleDetect_alg.m
function [Centroids, frmOut] = PotHoleDetect_alg(frmIn, rows, areaThreshold)
% PotholeDetect_alg
%
% Frame-based algorithm designed to utilize bilateral filtering to
% detect pot holes in the road. The major functions are:
% 1. Pre-process the input video - bilateral filter, sobel edge detect
% 2. Apply trapezoidal mask to focus on the current lane
% 3. Detect centroids of identified holes of a certain size
% 4. Overlay a marker
% Copyright 2018 The MathWorks, Inc.
%% Frame pre-processing
% Convert to intensity
frmGray = rgb2gray(frmIn);
% Bilateral filter
frmBiFilt = imbilatfilt(frmGray, 'NeighborhoodSize', 9);
% Edge detection
Workspace
Name
Value
Command Window
>>
PotHoleDetect_alg
Ln 2
Col 21

硬件实现算法：顶层

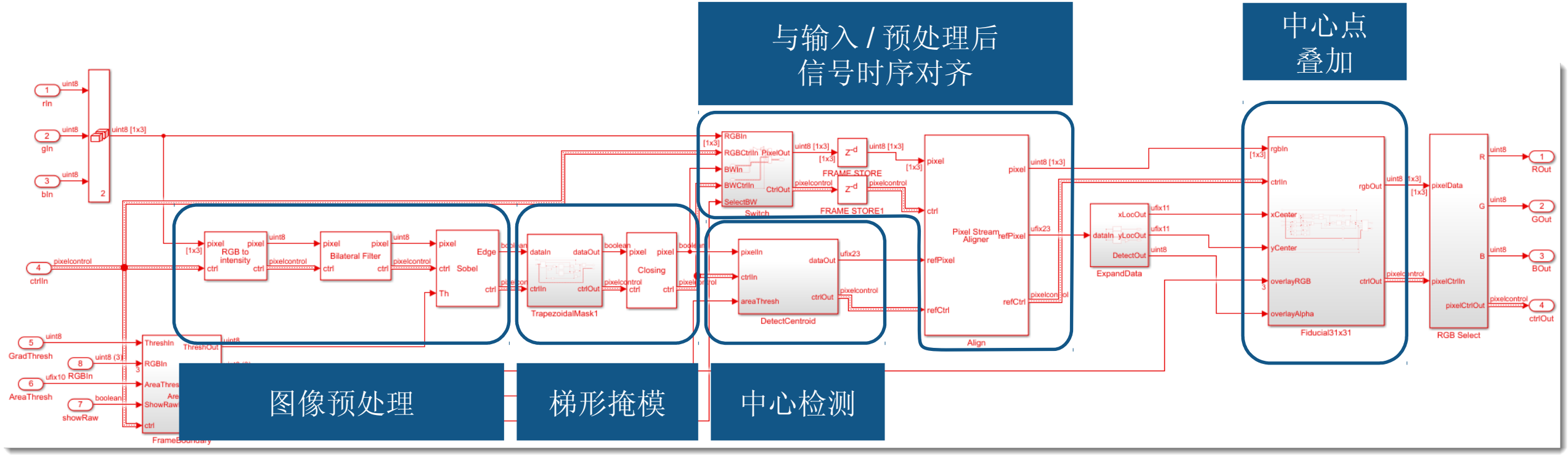


设计参数

- 边缘检测门限
- 陷坑尺寸门限
- 将检测结果叠加在输入 / 预处理后的视频上
- 叠加的颜色

硬件子系统

硬件实现算法：硬件子系统



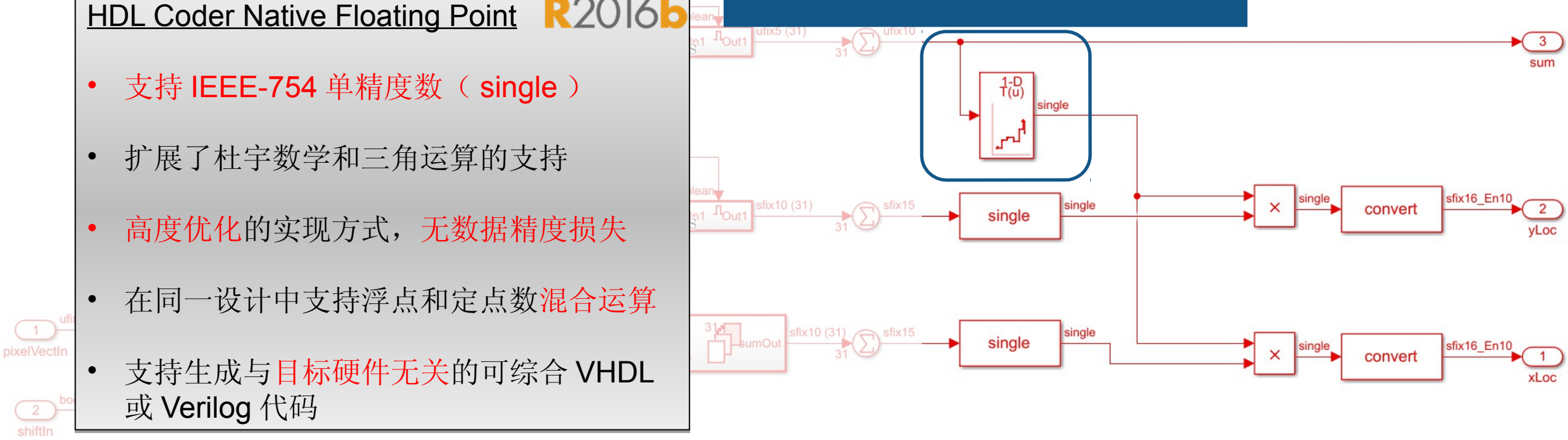
用自带的浮点算法进行流算法原型验证

Centroid Kernel

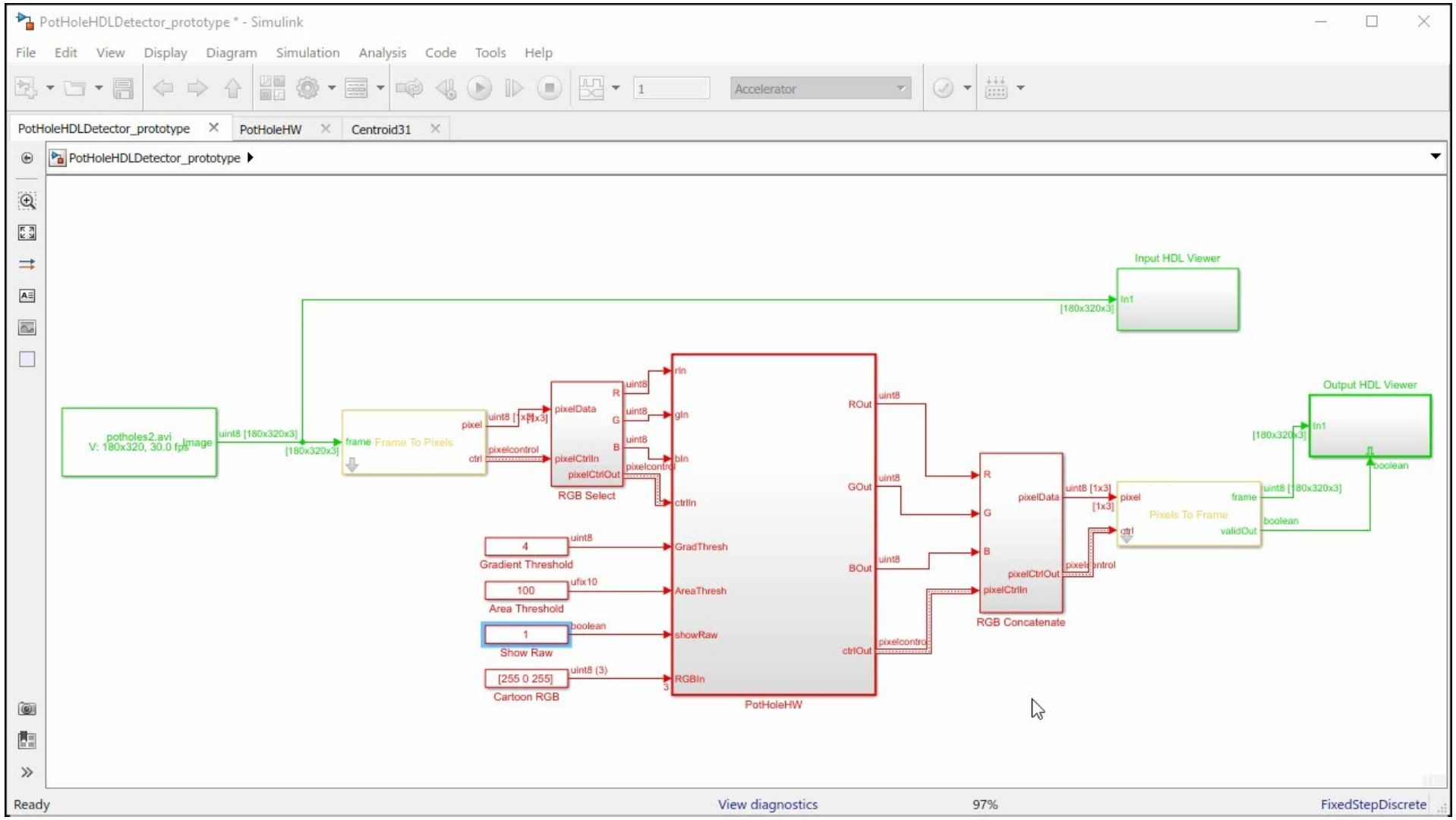
ROM 存储权重： `1./[1,1:1023]`
精度未知时需采用
自带浮点进行原型验证！

HDL Coder Native Floating Point R2016b

- 支持 IEEE-754 单精度数（ single ）
- 扩展了杜宇数学和三角运算的支持
- 高度优化的实现方式，无数据精度损失
- 在同一设计中支持浮点和定点数混合运算
- 支持生成与目标硬件无关的可综合 VHDL 或 Verilog 代码



目标板原型算法（Prototype Target）



定点化，硬件架构优化实现流程

1 了解你的硬件

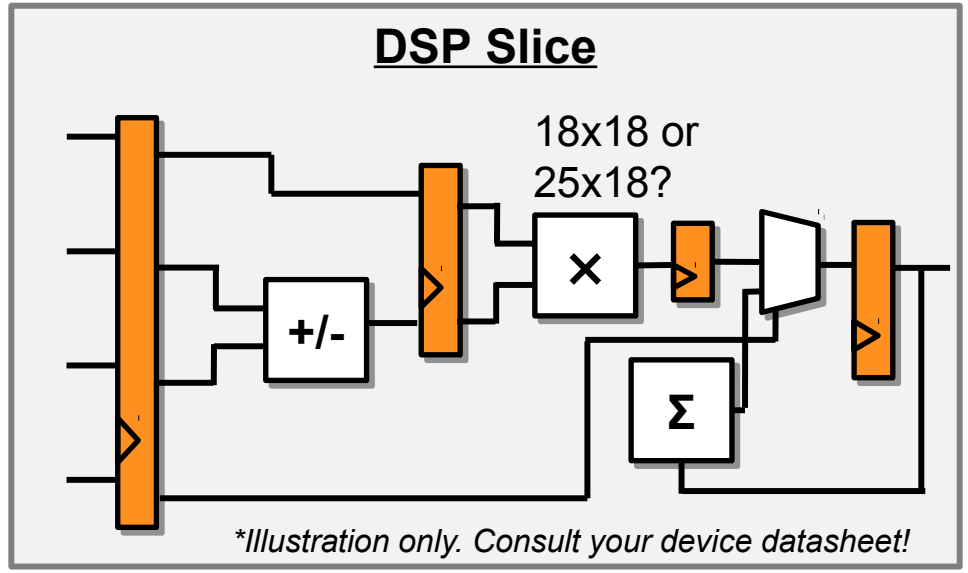
- How much on-chip RAM?
- Typical achievable frequency
- Available I/O
- FPGA: How many DSP slices?

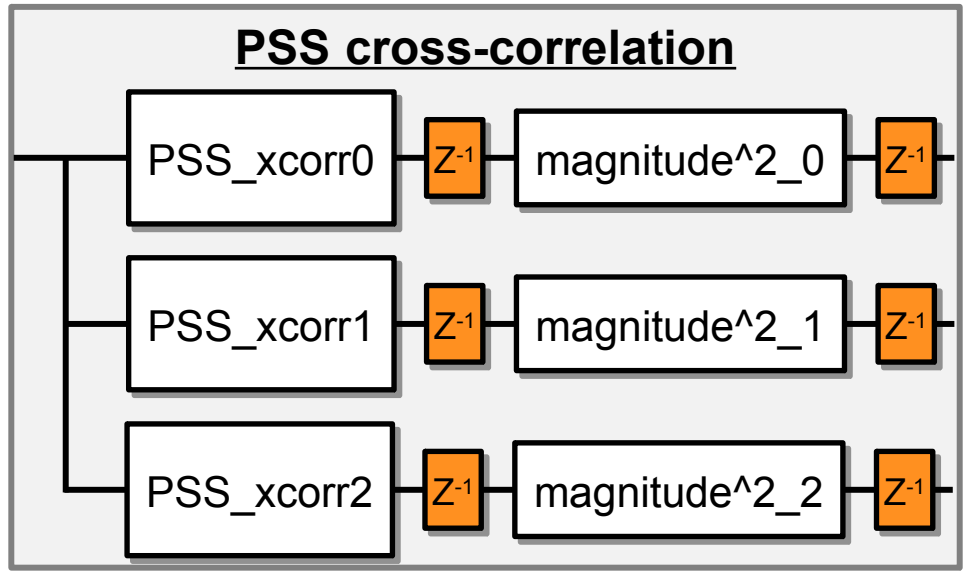
2 了解性能要求

- Control system latency
- Comms system throughput
- Video frame size & rate

3 简单的步骤，而后优化瓶颈

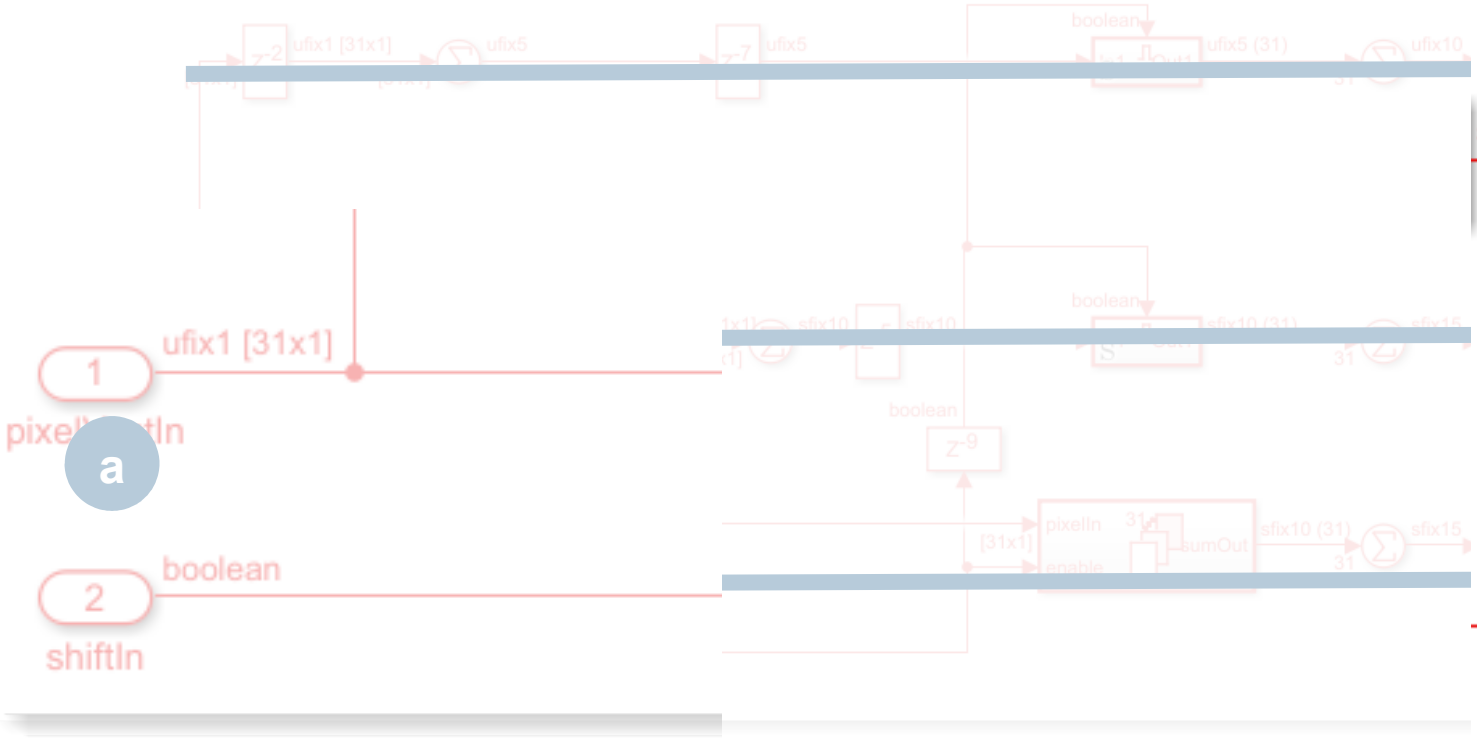
- Fixed-point quantization – esp. multipliers!
- Minimize/avoid use of off-chip RAM
- Parallelize operations for speed
- Use HDL Coder optimizations & reports

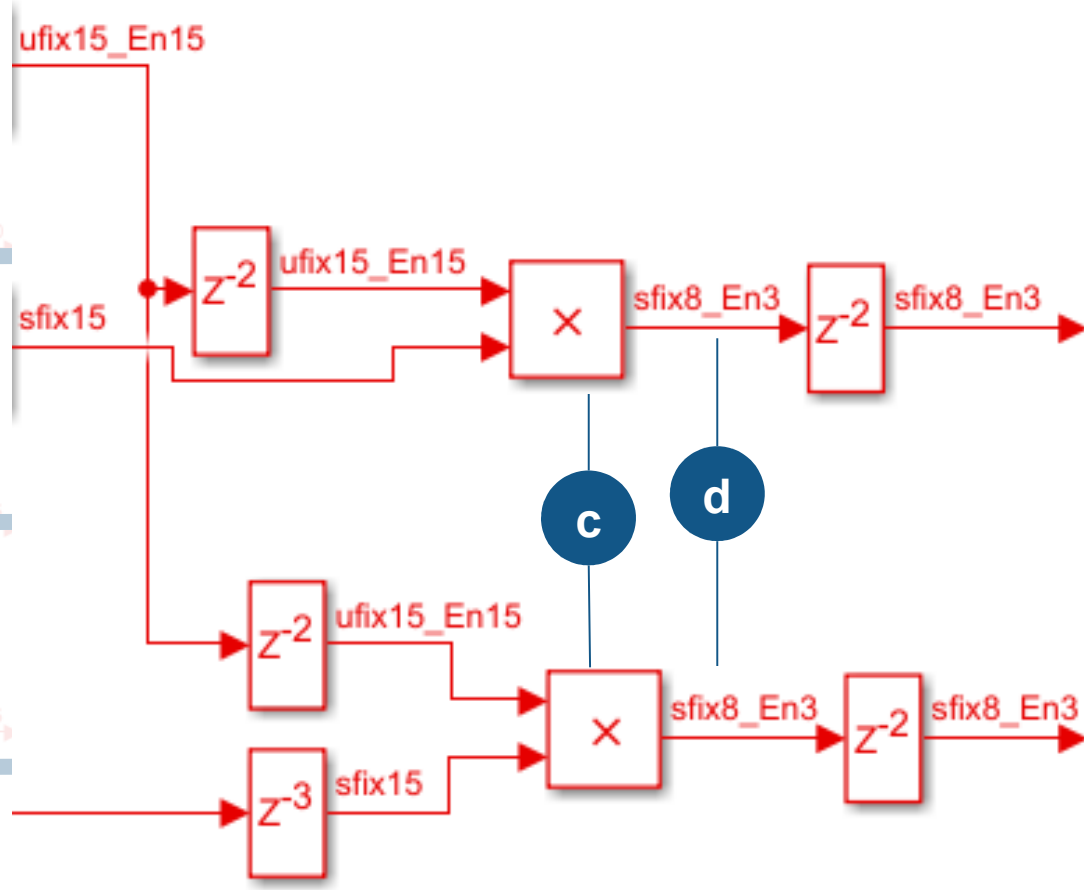




定点化，优化实现实例

Centroid Kernel





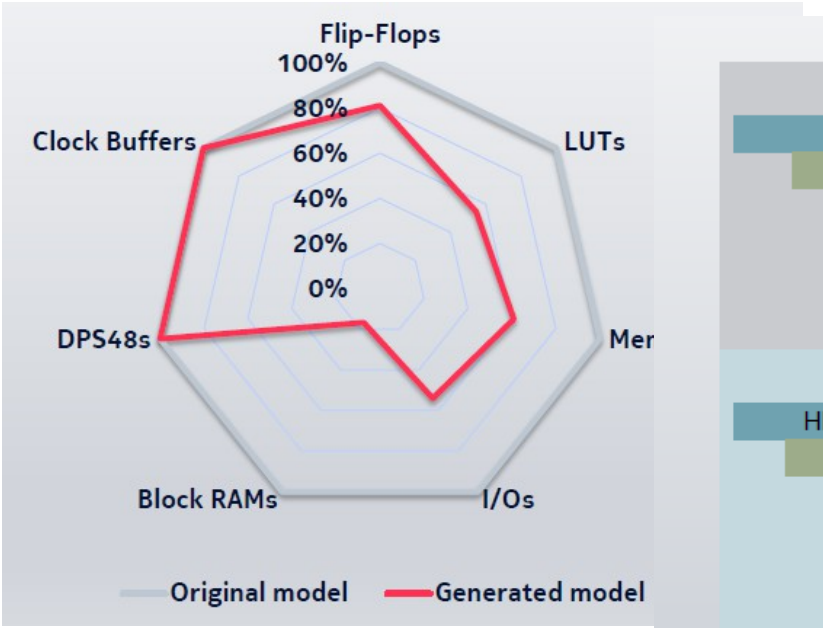
a 采用 31*31ROI 区域模板对视频帧处理，以适应片上存储容量

b 采用 3 条并行数据路径处理像素向量

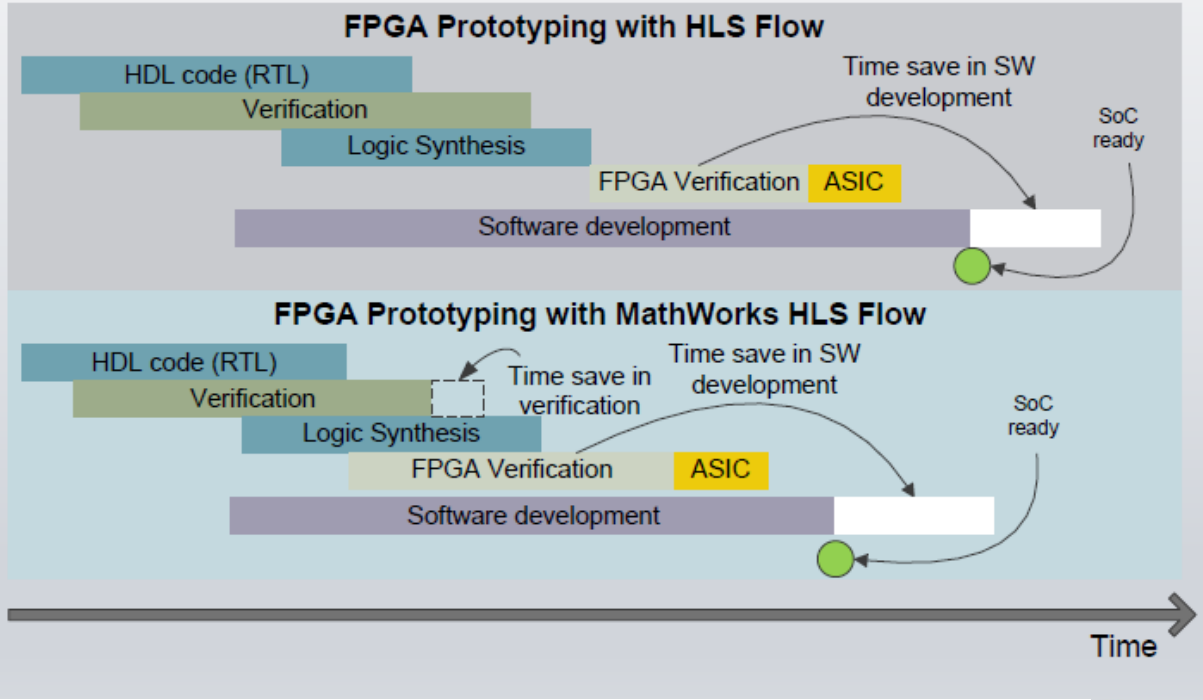
c 乘法器输入小于 18 bits ，并在处理前后各有两级 delay，确保 map 到 DSP Block 资源

d Fixed-Point Designer 帮助自动完成决定更小的所需字长

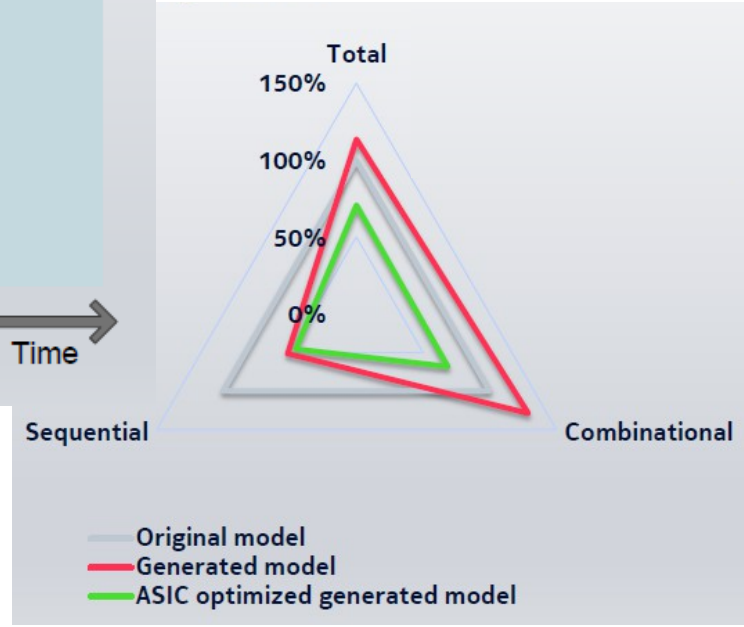
Nokia 采用 HDL Coder 加速 ASIC 原型验证并提高质量



FPGA 原型
资源使用量

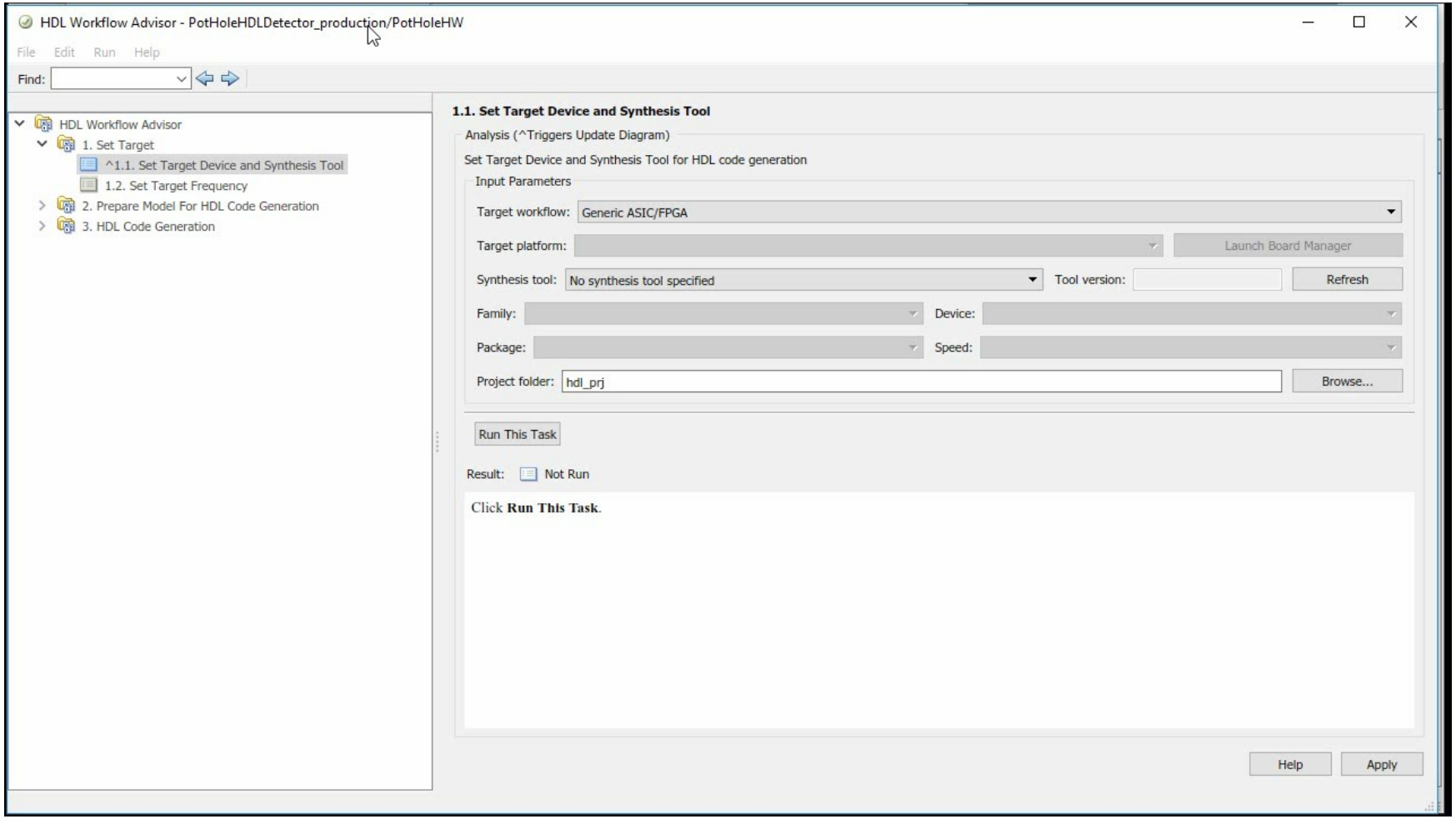


FPGA 原型
进度表改善



ASIC 实现资源消耗

目标产品—— IP Core 自动生成



算法
(Golden Reference)

算法软 / 硬件实现

定点数，优化实现

FPGA/ASIC 实现

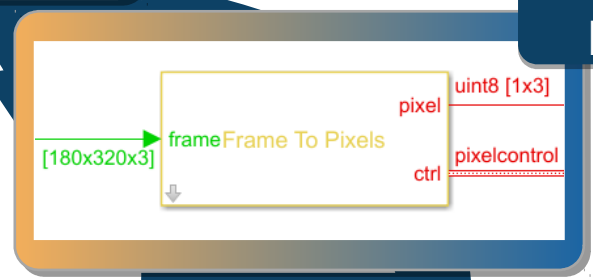
MATLAB

Simulink

```
%% Frame pre-processing

    % Convert to intensity
    frmGray = rgb2gray(frmIn);

    % Bilateral filter
    frmBiFilt = imbilatfilt(frmGray, 'NeighborhoodSize', 9);

    % Edge detection
    frmEdge = edge(frmBiFilt,'Sobel', .05);

%% Trapezoidal mask
```

HDL-ready
IP blocks

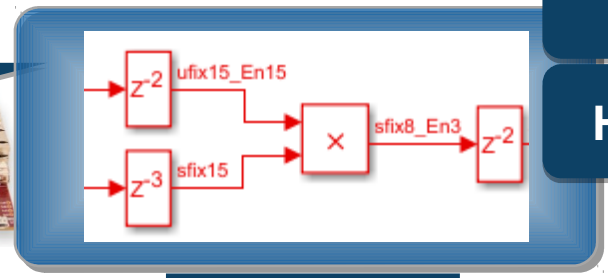
验证



HDL Coder
原型

Fixed Point
Designer

HDL Coder



HDL Coder
产品

Core
interface

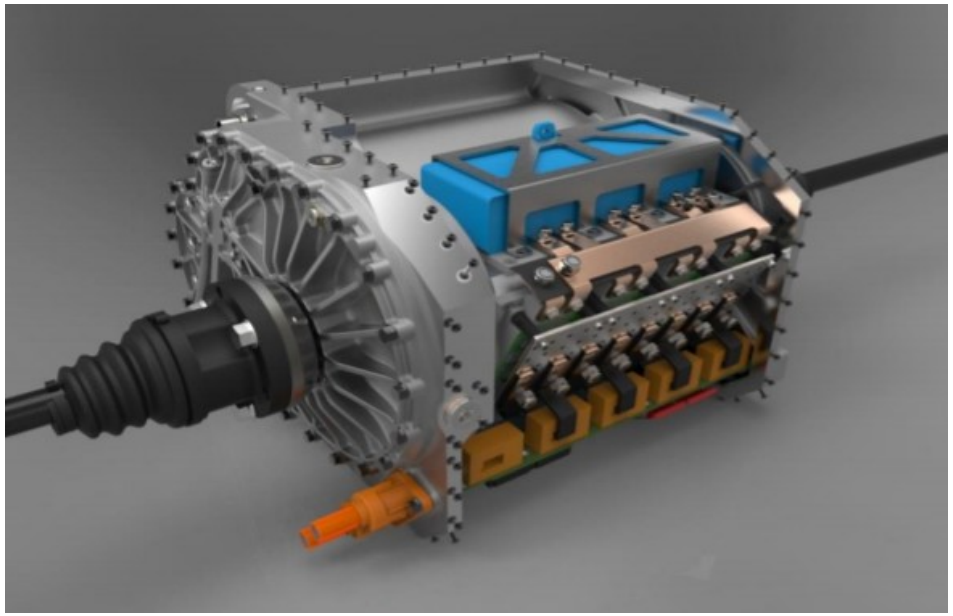
VHDL /
Verilog

Constraints

提纲

- 何时考虑使用 FPGA, ASIC, or System-on-Chip (SoC) 硬件
- 算法的硬件实现，所需考虑的问题
- 从系统 / 算法到 FPGA/ASIC 硬件的流程
- 演示：视觉处理算法部署到 FPGA 硬件
- 结论

用户案例：Punch Powertrain （邦奇自动变速箱）

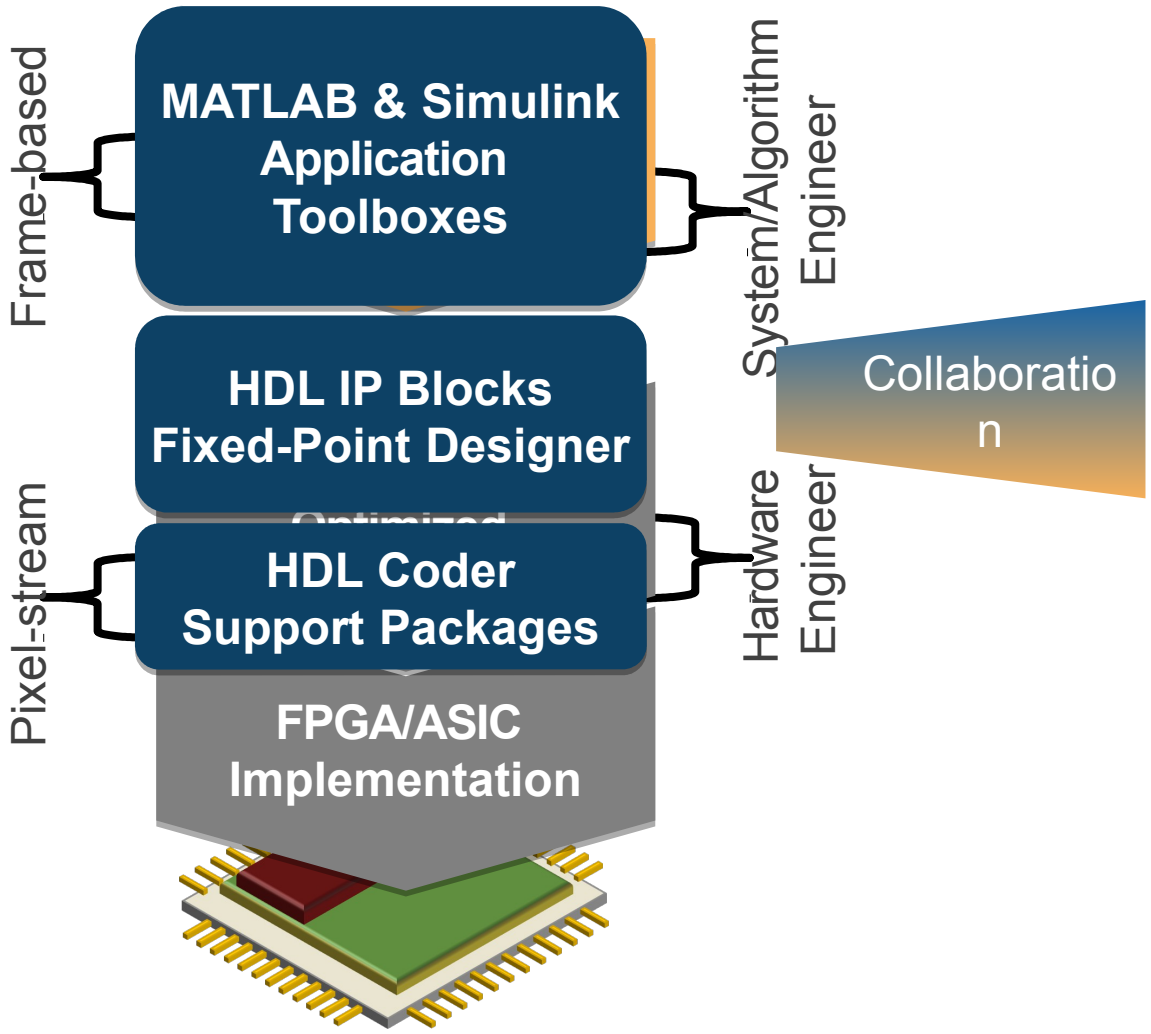


需求

- 新型开关磁阻电机 －——全新复杂控制策略
- 快速：速度 2 倍与已有电机
- 目标板卡采用 Xilinx® Zynq® SoC 7045
- 快速上市
- 无 FPGA 设计经验！

- ✓ 设计集成式电子驱动器：电机，电力电子和软件
- ✓ 采用了 4 种不同的控制策略
- ✓ 2 名工程师用 1.5 年完成全部开发
- ✓ 模型方便在产品中复用
- ✓ 顺利通过集成和测试——因生产前已通过验证平台进行全面验证

从帧到像素至硬件的流程



- 新应用的创新在系统级发生
  - 涵盖软件和硬件的实现
- 软硬件的成功实现需要紧密协作

- 流程延伸至 FPGA/ASIC/SoC 硬件实现：
  - 可快速尝试更多微架构
  - 敏捷完成修改、仿真、代码生成的迭代
  - 贯穿始终的验证

Thank You ！