MATLAB EXPO 2018

Deploying Your Algorithm to FPGA or ASIC Hardware

Outline

- When FPGA, ASIC, or System-on-Chip (SoC) hardware is needed
- Hardware implementation considerations
- Workflow from system/algorithm to FPGA/ASIC hardware
- Demonstration: Vision processing algorithm deployed to FPGA
- Conclusion

Why Are Our Customers Deploying to FPGA/ASIC Hardware?

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"

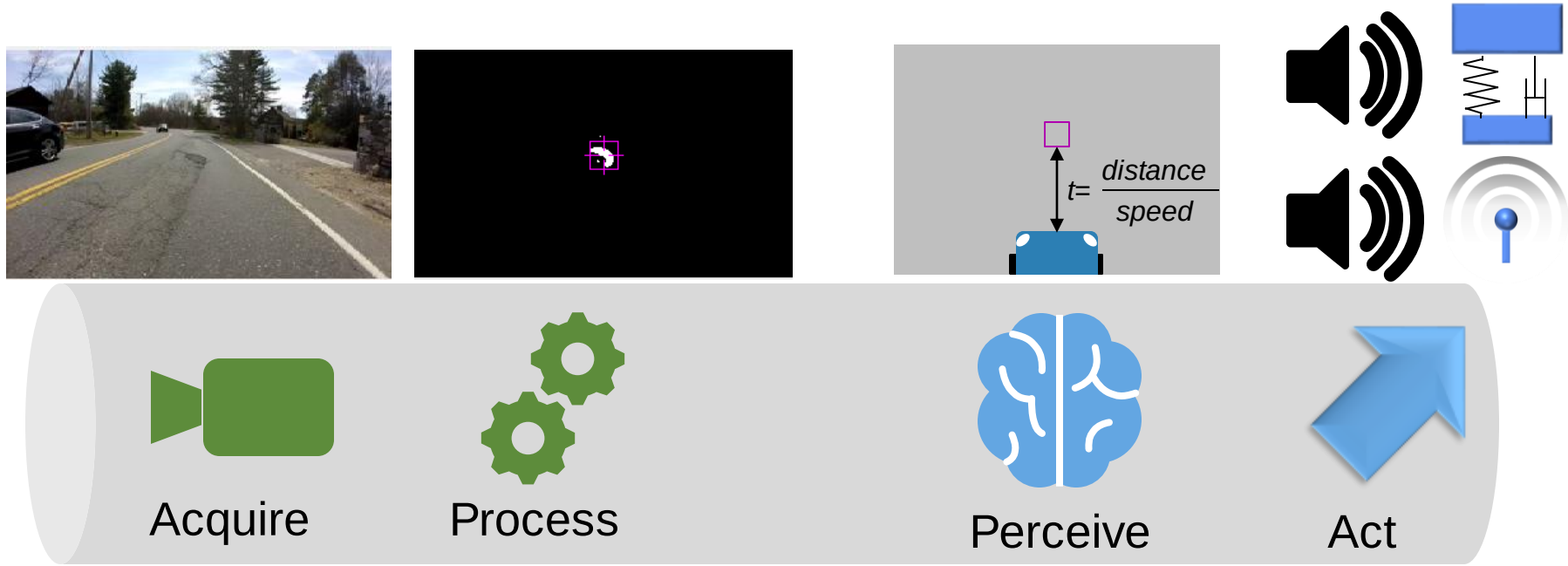
"Audio transducer prototypes must run in real time with low latencies"

"Motor control latency < 1us"

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

Modern Applications Often Require Custom Hardware

ADAS Application Example



1M+ pixels per frame

High-speed, well-defined
- Repetitive processing
- Large amount of data

FPGA Hardware

Few coords

Complex, more flexible
- Small data calculations
- Executive control

Embedded Software

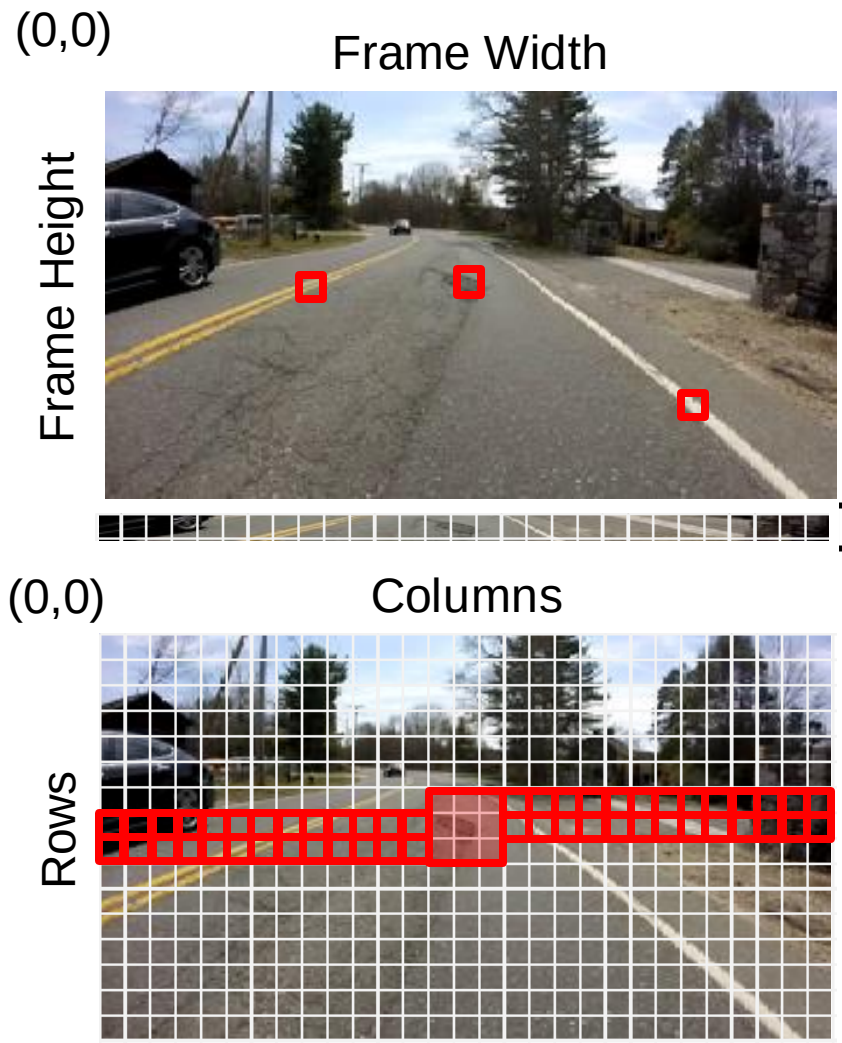
Outline

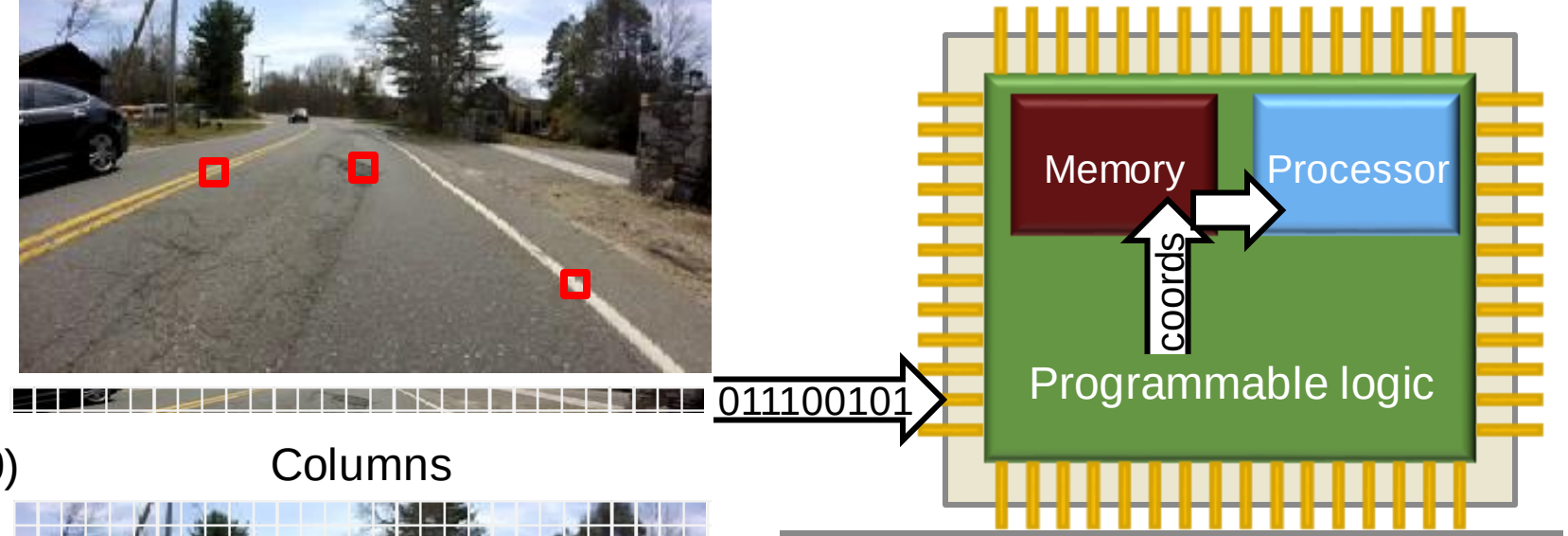
- When FPGA, ASIC, or System-on-Chip (SoC) hardware is needed
- Hardware implementation considerations
- Workflow from system/algorithm to FPGA/ASIC hardware
- Demonstration: Vision processing algorithm deployed to FPGA
- Conclusion

Frame-Based vs Streaming Algorithms

Frame-Based
- Whole frame at a time
- Random access to any pixel via [x,y] coordinate

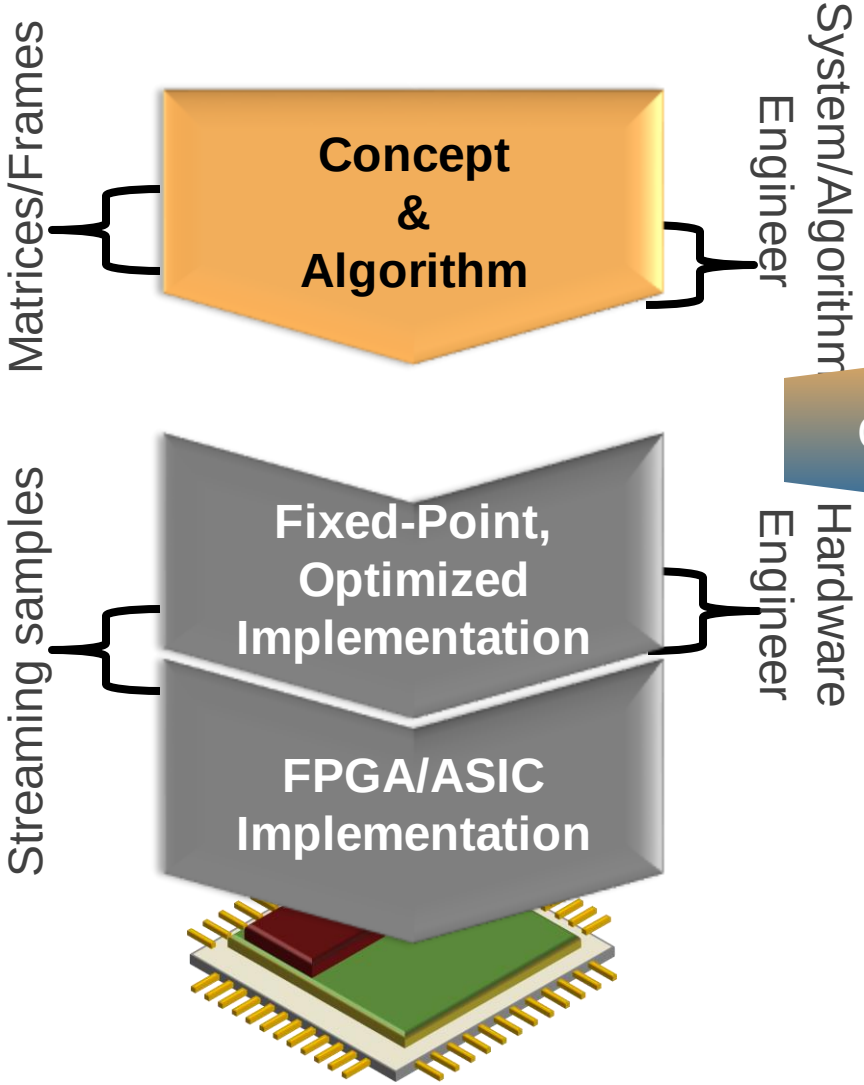
Streaming
- Sample-by-sample, row-by-row
- Region of interest (ROI) stored in a multi-line buffer





Hardware
- Bit-by-bit, but parallel computation
- Fixed and finite resources
- Buffers require memory storage
- Communications with software go through dedicated memory

Bridging the Gap from Algorithm to Implementation



Collaboration

Concept & Algorithm
- Develop system-level algorithms
- Simulate, analyze, modify
- Partition hardware vs software implementation

Algorithm to Micro-architecture
- Convert to streaming algorithms
- Add hardware micro-architecture
- Convert data types to fixed-point

Micro-architecture to implementation
- Speed and area optimization
- HDL code generation
- Verification
- FPGA/ASIC implementation

Outline

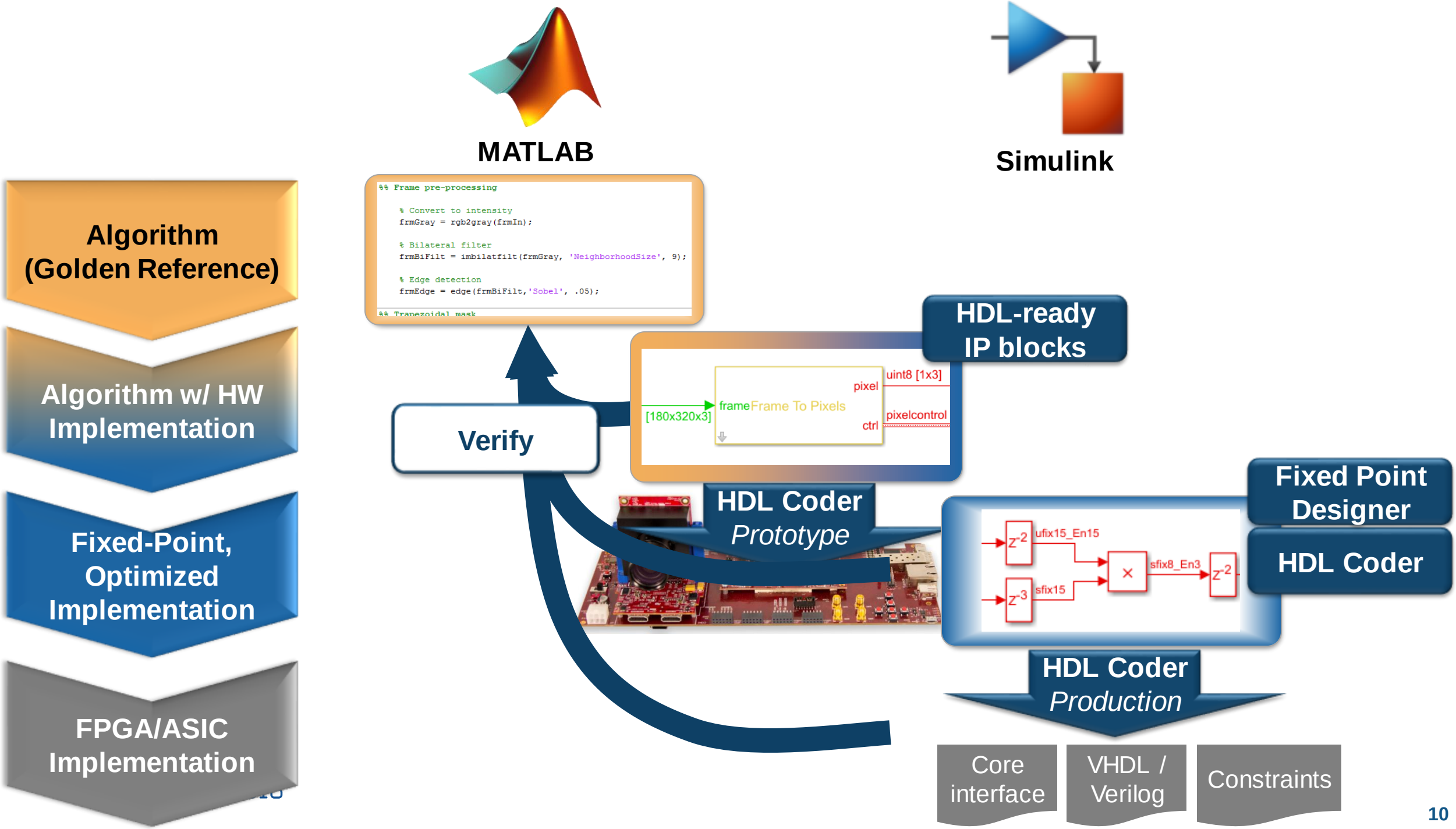
- When FPGA, ASIC, or System-on-Chip (SoC) hardware is needed
- Hardware implementation considerations
- Workflow from system/algorithm to FPGA/ASIC hardware
- Demonstration: Vision processing algorithm deployed to FPGA
- Conclusion

MATLAB

- ✓ Large data sets
- ✓ Explore mathematics
- ✓ Data visualization

Simulink

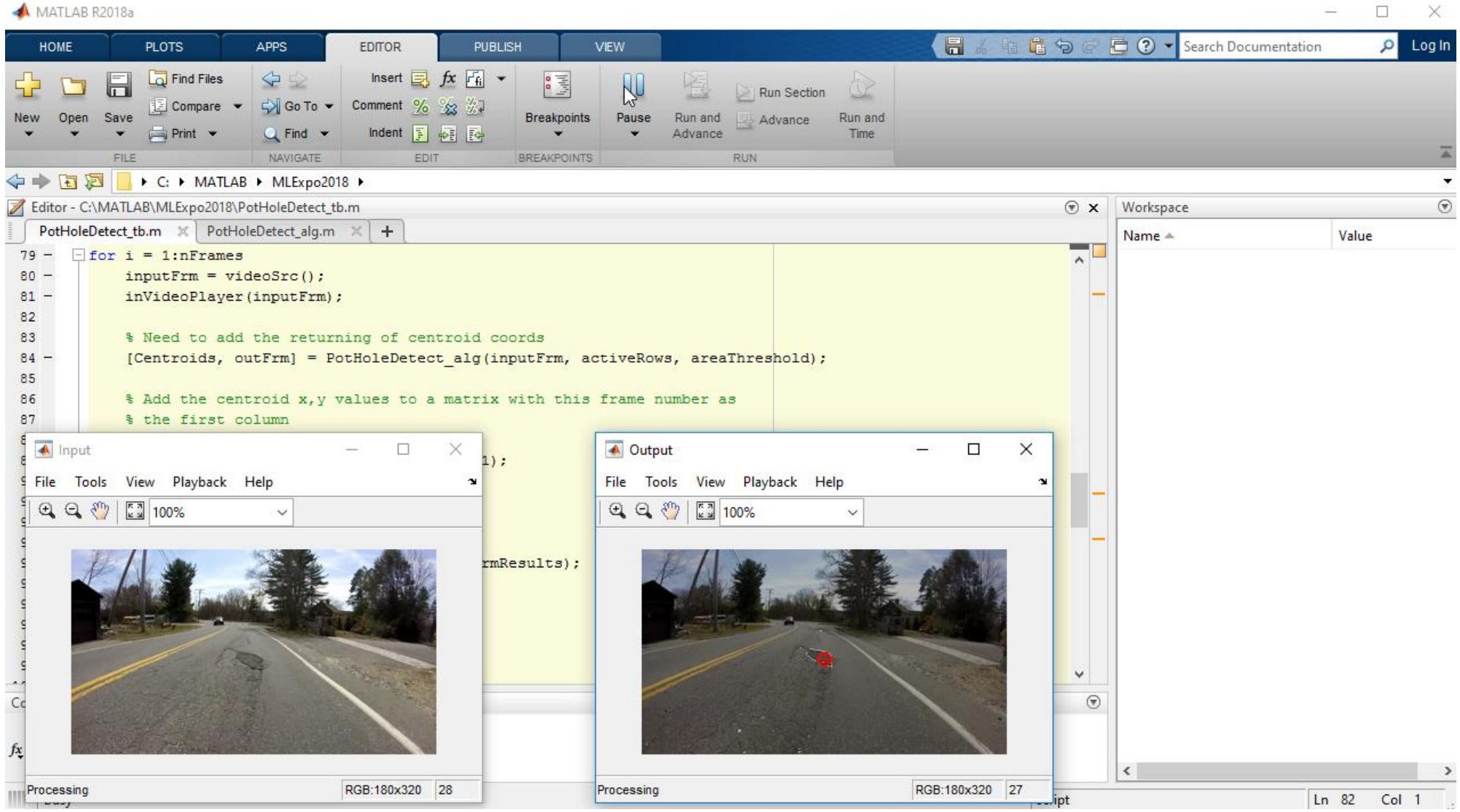
- ✓ Parallel architectures
- ✓ Timing
- ✓ Data type propagation


MATLAB
Simulink
Algorithm (Golden Reference)
Algorithm w/ HW Implementation
Fixed-Point, Optimized Implementation
FPGA/ASIC Implementation
%% 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
[180x320x3]
frame
Frame To Pixels
pixel
ctrl
uint8 [1x3]
pixelcontrol
Verify
HDL Coder
Prototype
Fixed Point Designer
HDL Coder
ufix15_En15
sfix15
sfix8_En3
HDL Coder
Production
Core interface
VHDL / Verilog
Constraints

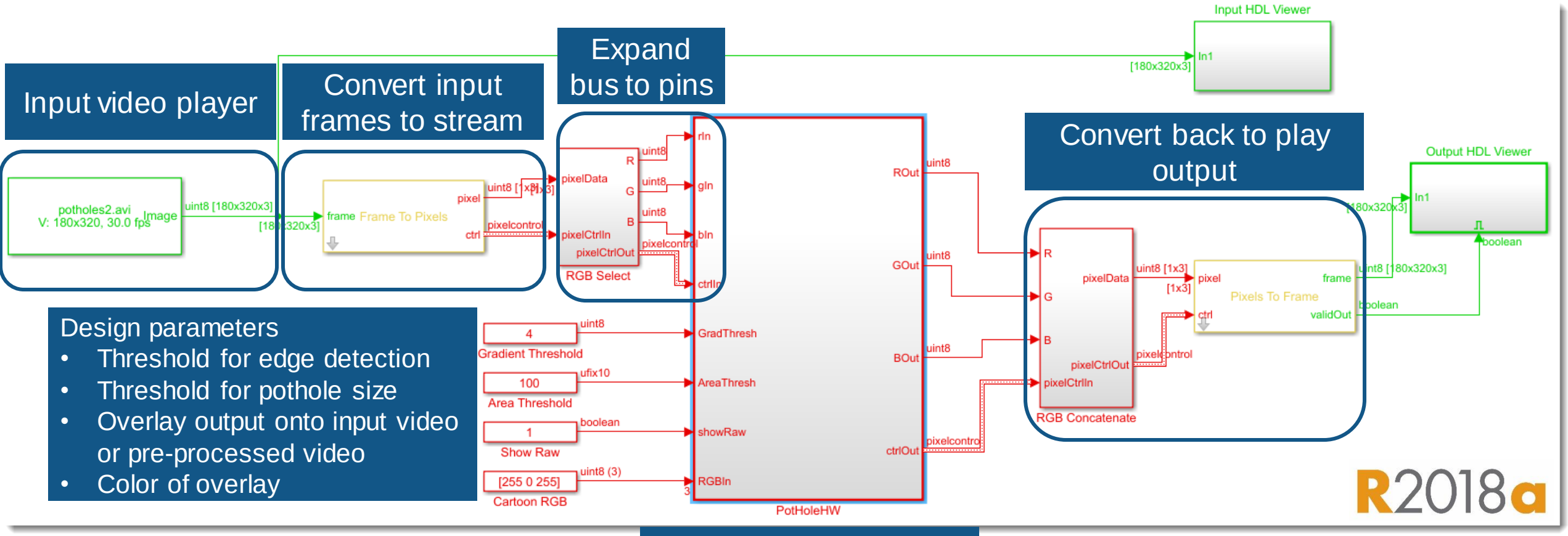
Outline

- When FPGA, ASIC, or System-on-Chip (SoC) hardware is needed
- Hardware implementation considerations
- Workflow from system/algorithm to FPGA/ASIC hardware
- Demonstration: Vision processing algorithm deployed to FPGA
- Conclusion

Algorithm Overview



Algorithm with Hardware Implementation: Top-Level

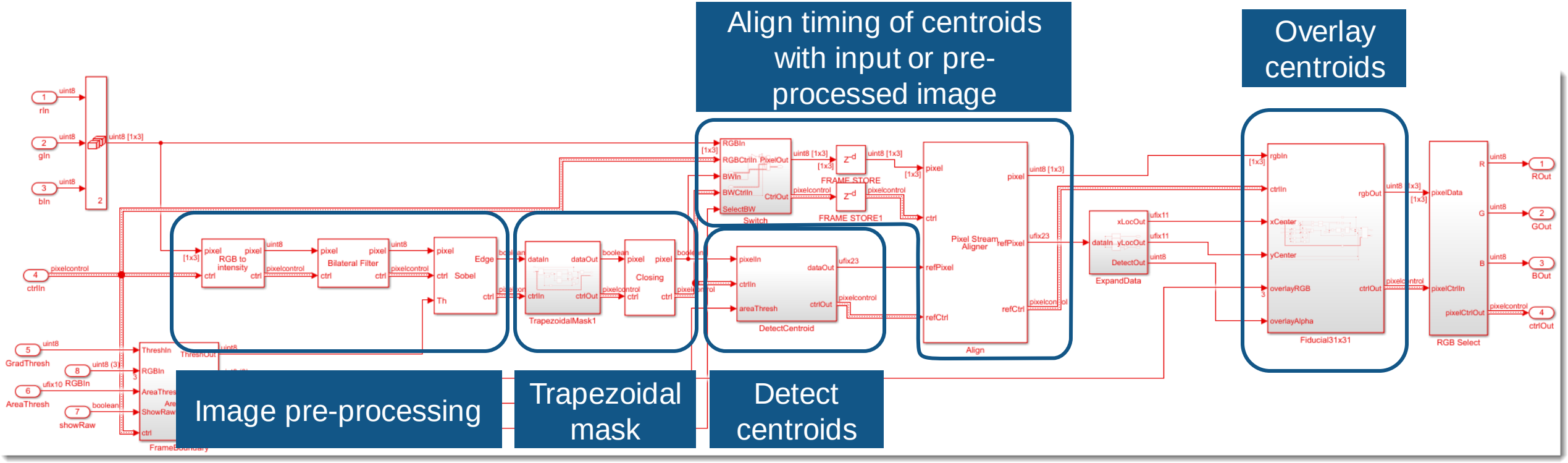


Design parameters

- Threshold for edge detection
- Threshold for pothole size
- Overlay output onto input video or pre-processed video
- Color of overlay

Hardware subsystem

Algorithm with Hardware Implementation: Hardware Subsystem



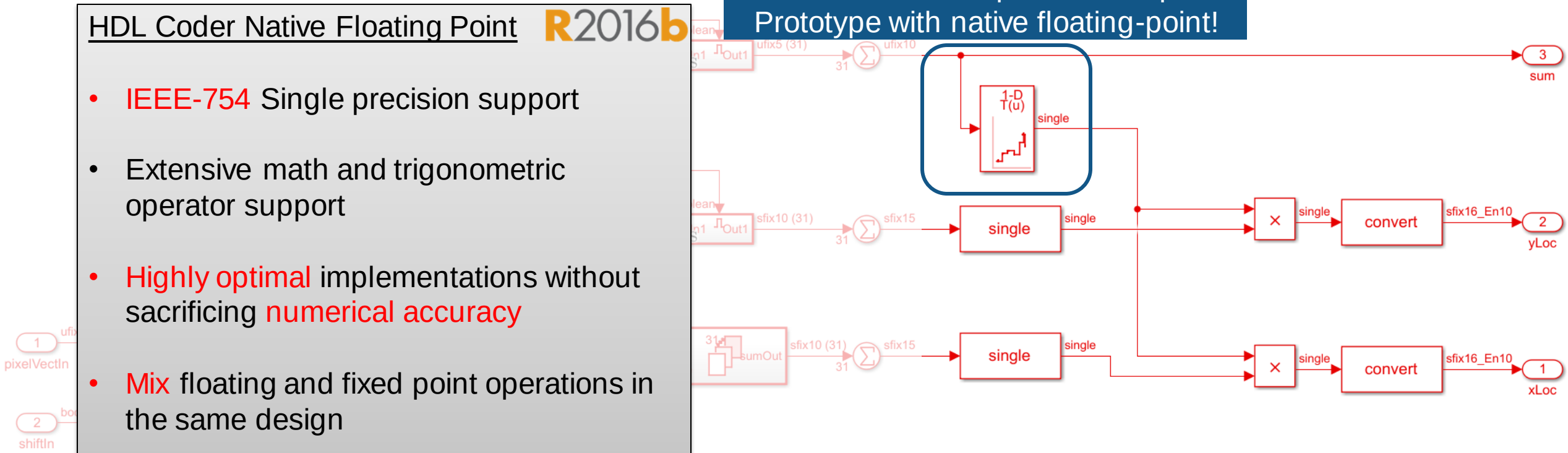
Streaming Math with Native Floating-Point for Prototyping

Centroid Kernel

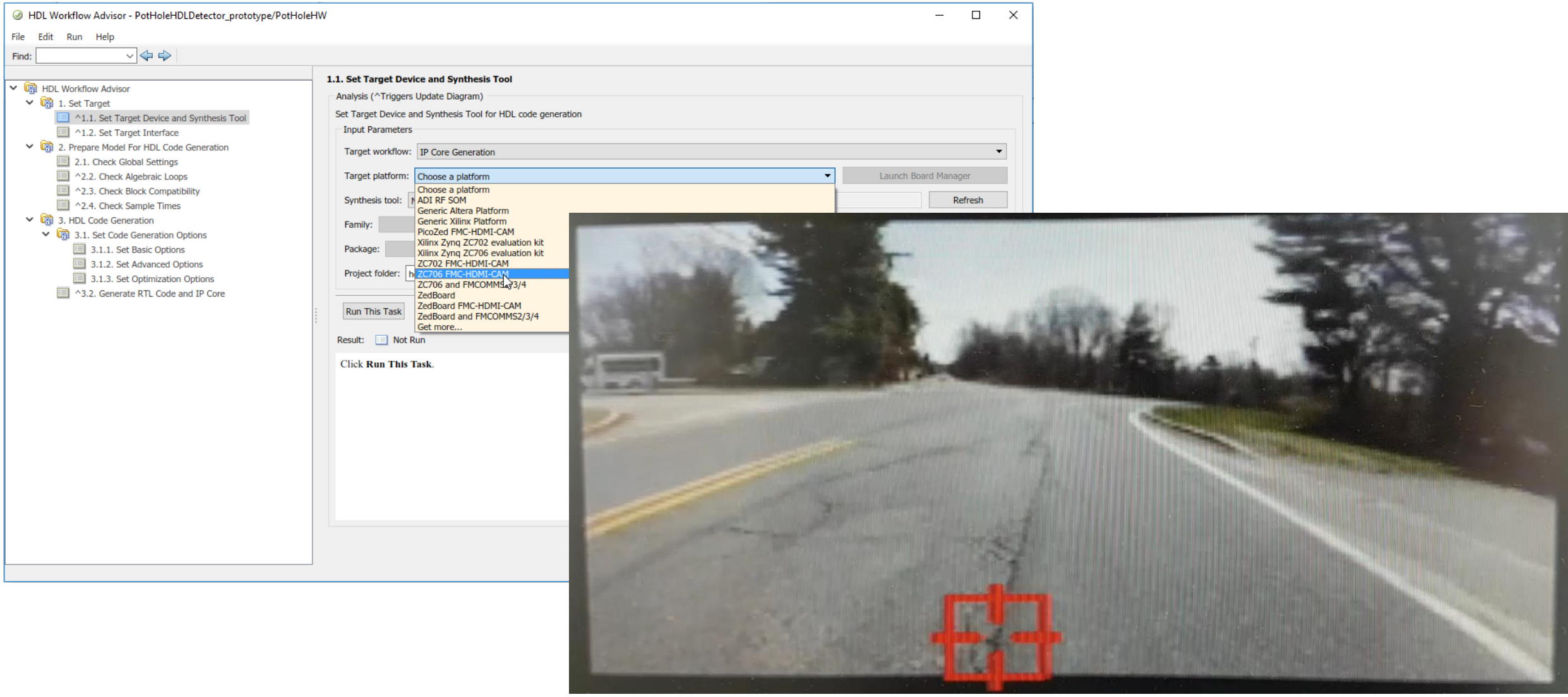
ROM stores weights: `1./[1,1:1023]`
Don't know level of precision required
Prototype with native floating-point!

HDL Coder Native Floating Point R2016b

- IEEE-754 Single precision support
- Extensive math and trigonometric operator support
- Highly optimal implementations without sacrificing numerical accuracy
- Mix floating and fixed point operations in the same design
- Generate target-independent synthesizable VHDL or Verilog



Prototype Target



Fixed-Point, Optimized Implementation: General Approach

1 Know your hardware

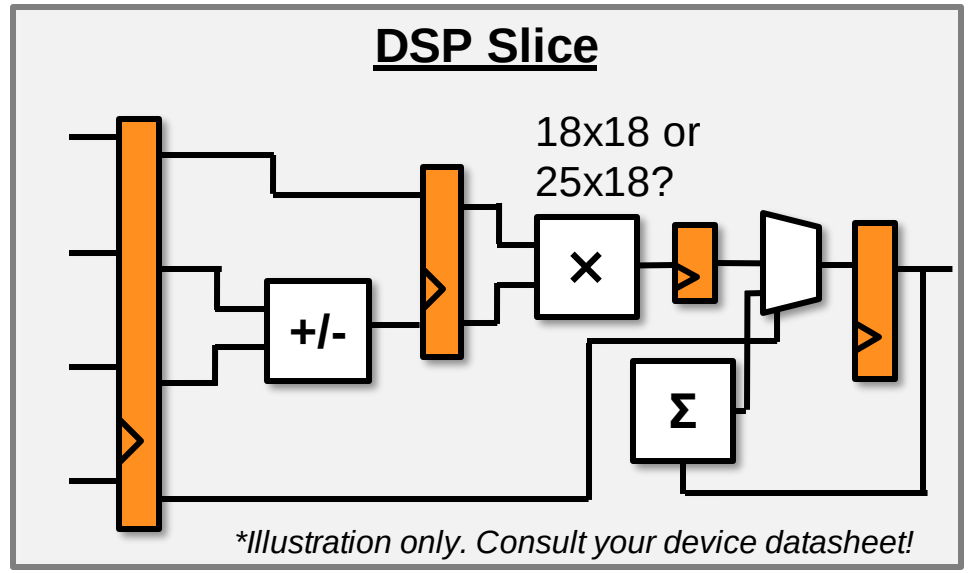
- How much on-chip RAM?
- Typical achievable frequency
- Available I/O
- FPGA: How many DSP slices?

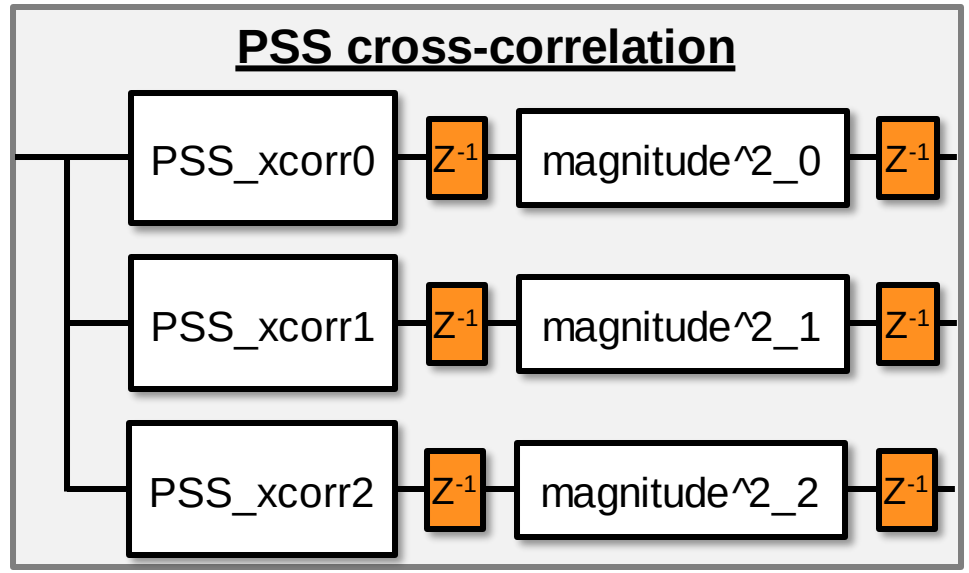
2 Know your performance requirements

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

3 Simple steps, then address bottlenecks

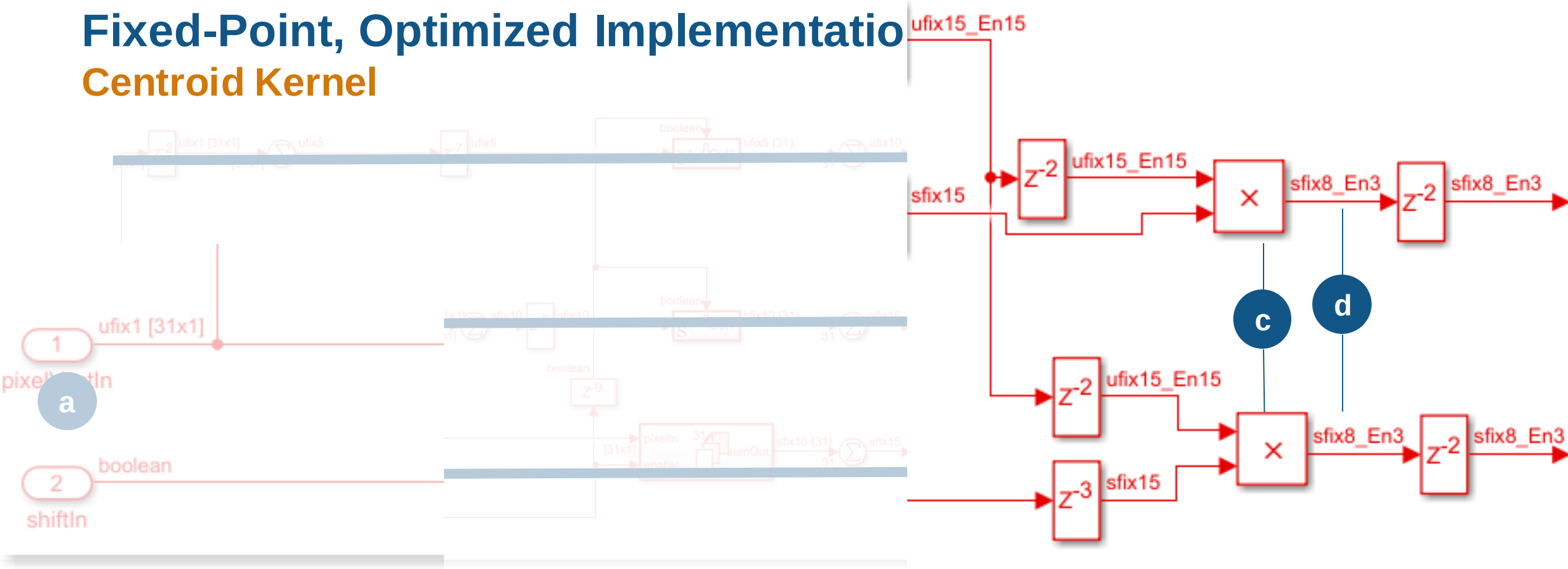
- Fixed-point quantization – esp. multipliers!
- Minimize/avoid use of off-chip RAM
- Parallelize operations for speed
- Use HDL Coder optimizations & reports





Fixed-Point, Optimized Implementatio

Centroid Kernel



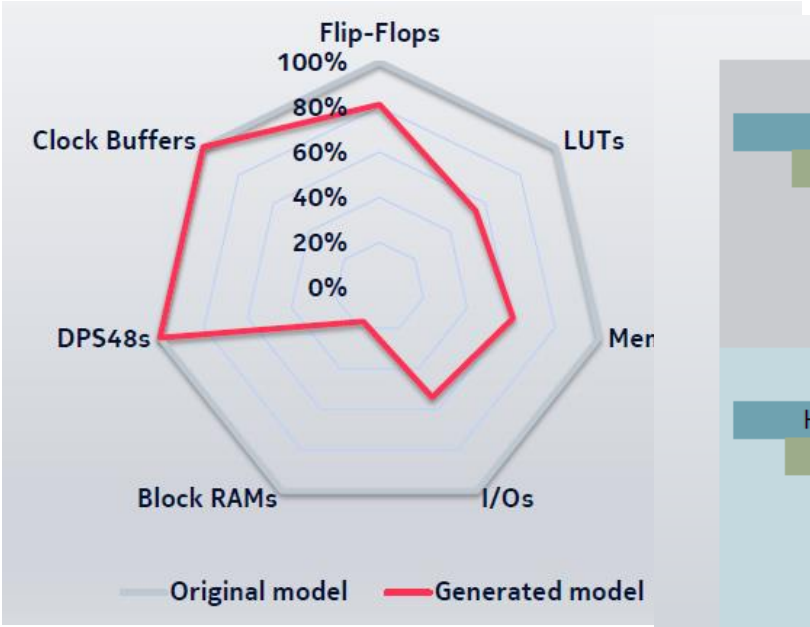
a Slide a 31x31 ROI across the frame to fit in on-chip RAM

b Process pixel vector in three parallel data paths

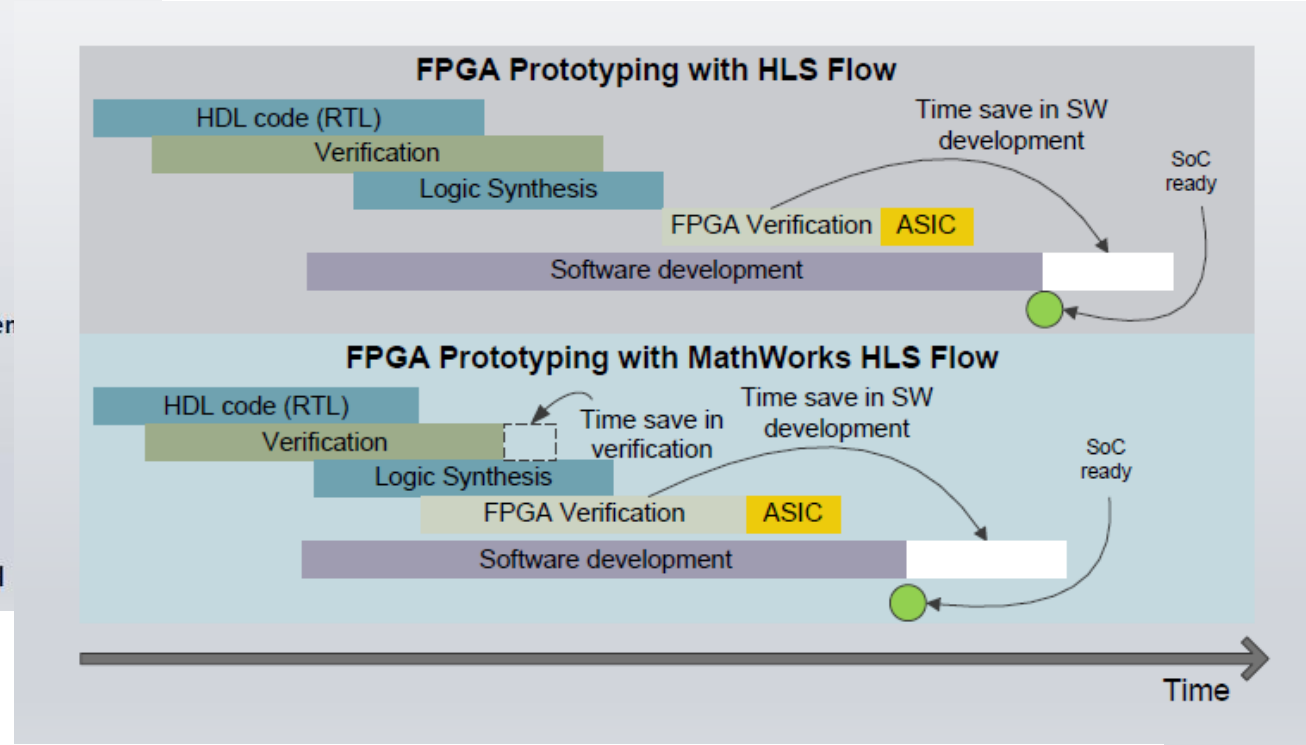
c Multiplier inputs <18 bits with 2 delays before and after, to ensure mapping to DSP resources

d Fixed-Point Designer helped determine smaller word lengths needed

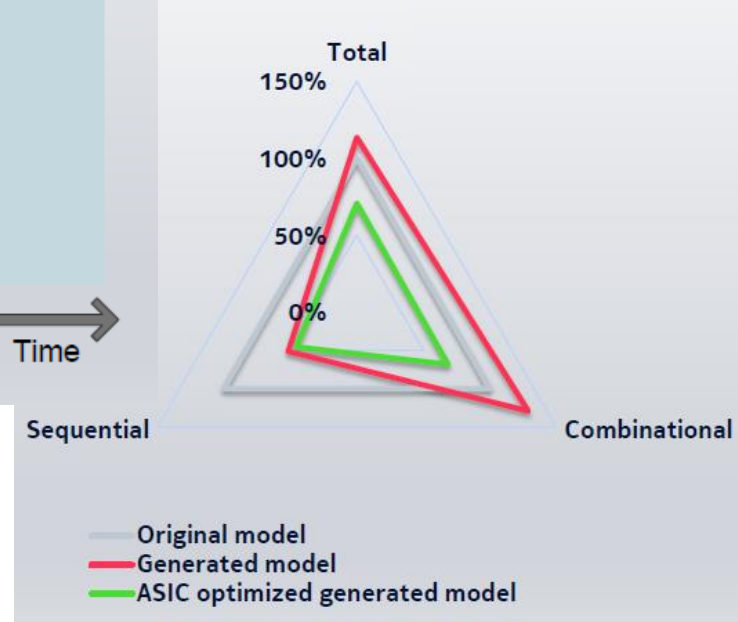
Nokia Speeds ASIC Prototyping and Improves Quality of Results with HDL Coder



FPGA Prototype resource usage



FPGA Prototype schedule improvement



ASIC implementation area

Production Target – IP Core Gen

HDL Workflow Advisor - PotHoleHDLDetector_production/PotHoleHW

File Edit Run Help

Find:

- HDL Workflow Advisor
  - 1. Set Target
    - ^1.1. Set Target Device and Synthesis Tool
    - ^1.2. Set Target Interface
  - 2. Prepare Model For HDL Code Generation
  - 3. HDL Code Generation

1.2. Set Target Interface

Analysis (^Triggers Update Diagram)

Set target interface for HDL code generation

Input Parameters

Processor/FPGA synchronization: Free running

Target platform interface table

Port Name	Port Type	Data Type	Target Platform Interfaces	Bit Range / Address / FPGA Pin
pixelIn	Inport	uint32	AXI4-Stream Video Slave	Pixel Data
ctrlIn	Inport	bus	AXI4-Stream Video Slave	Pixel Control Bus
GradThresh	Inport	uint8	AXI4-Lite	Map target platform interfaces to subsystem ports
AreaThresh	Inport	ufix10	AXI4-Lite	x"104"
showRaw	Inport	boolean	AXI4-Lite	x"108"
RGBIn	Inport	uint8 (3)	AXI4-Lite	x"110"
pixelOut	Outport	uint32	AXI4-Stream Video Master	Pixel Data
ctrlOut	Outport	bus	AXI4-Stream Video Master	Pixel Control Bus

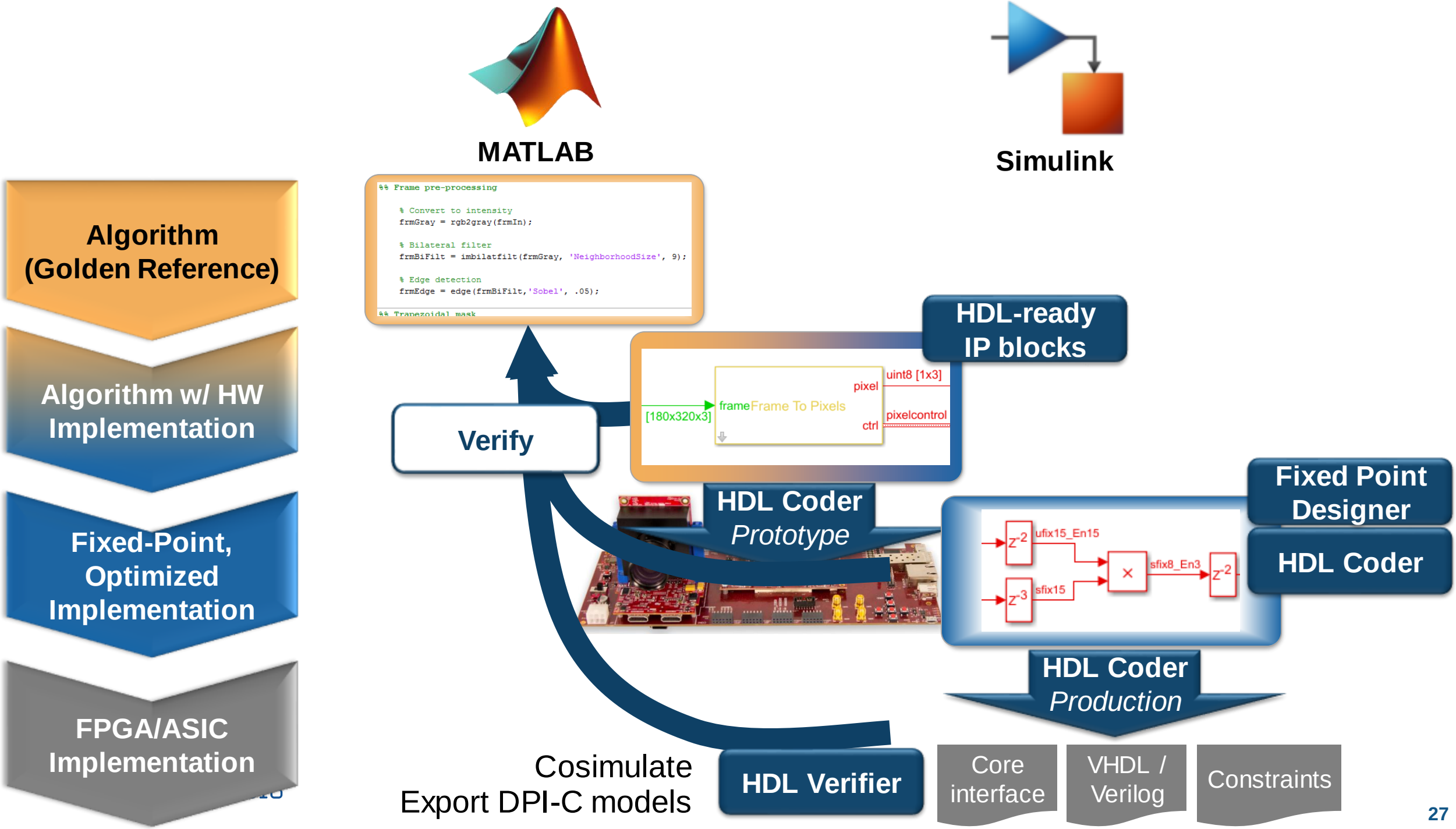
Run This Task

Result: Not Run

Click **Run This Task**.

Help Apply

Production device has a different target interface


MATLAB
Simulink
Algorithm (Golden Reference)
Algorithm w/ HW Implementation
Fixed-Point, Optimized Implementation
FPGA/ASIC Implementation
%% 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
[180x320x3]
frame
Frame To Pixels
pixel
ctrl
uint8 [1x3]
pixelcontrol
Verify
HDL Coder
Prototype
Fixed Point Designer
HDL Coder
ufix15_En15
sfix15
sfix8_En3
HDL Coder
Production
Cosimulate
Export DPI-C models
HDL Verifier
Core interface
VHDL / Verilog
Constraints

Outline

- When FPGA, ASIC, or System-on-Chip (SoC) hardware is needed
- Hardware implementation considerations
- Workflow from system/algorithm to FPGA/ASIC hardware
- Demonstration: Vision processing algorithm deployed to FPGA
- Conclusion

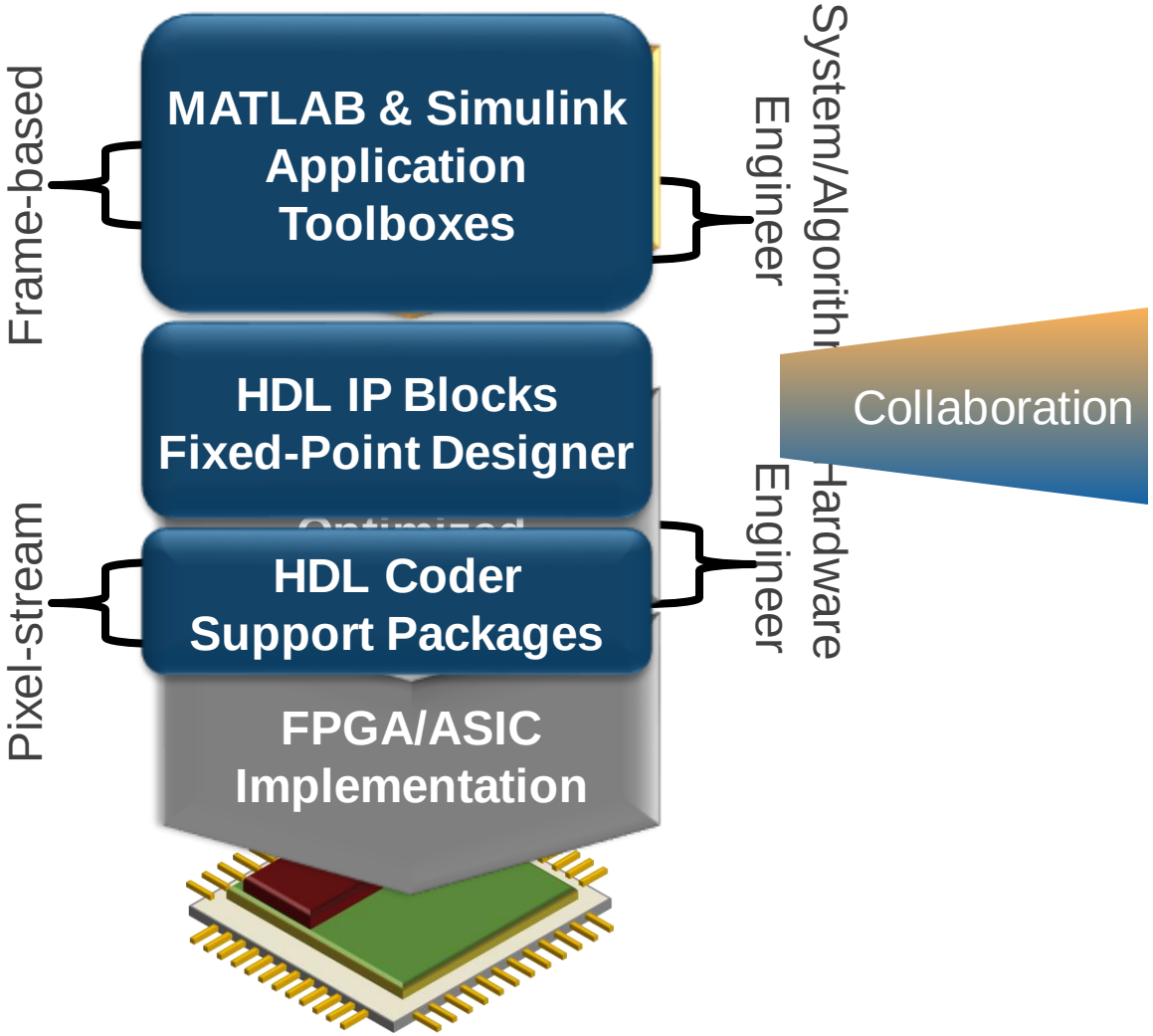
Customer Case Study: Punch Powertrain

Requirements

- New switched reluctance motor – new complex control strategies
- Fast: 2x the speed of their previous motor
- Target to a Xilinx® Zynq® SoC 7045 device
- Needed to get to market quickly
- No experience designing FPGAs!

- ✓ Designed integrated E-drive: Motor, power electronics and software
- ✓ 4 different control strategies implemented
- ✓ Done in 1.5 years with 2FTE's
- ✓ Models reusable for production
- ✓ Smooth integration and validation due to development process – thorough validation before electronics are produced and put in the testbench

Workflow From Frames to Pixels to Hardware



- New application innovation happens at the system-level
  - Implemented across software and hardware
- Successful implementation requires collaboration
- Connected workflow to FPGA/ASIC/SoC hardware delivers:
  - Broader micro-architecture exploration
  - Agility to make changes, simulate, generate code
  - Continuous verification

Learn More

White Paper: Deploying LTE Wireless Communications on FPGAs: A Complete MATLAB and Simulink Workflow

Webinar: Modeling HDL Components for FPGAs in Control Applications

Video Series: Vision Processing for FPGA (5 Videos)