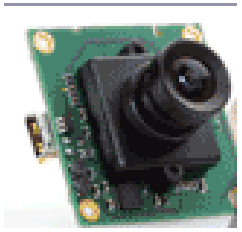
"Object Recognition and Tracking for Defense and Industrial Applications"

Presented by Michael Fawcett, CTO – iVeia
Dan Isaacs Director Marketing – Xilinx

Embedded Vision Image Processing Applications



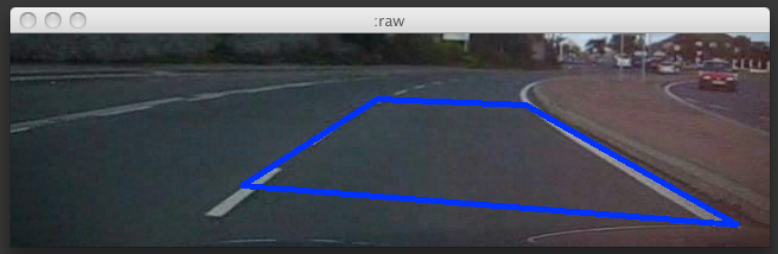
- **Trend towards "at-the-edge" image processing**
  - Applicable to wide range of applications
  - e.g., Smart cameras, in vehicle, gimbal, or telescoping robotic arm

- **Specific requirement for porting customer algorithm**
  - from OpenCV algorithm to embedded platform
- **Processes video from a miniature GigE Vision camera**
- **High-performance required for low-latency fluid refresh**

Example Application: Lane Detection

Illustrating OpenCV Algorithm

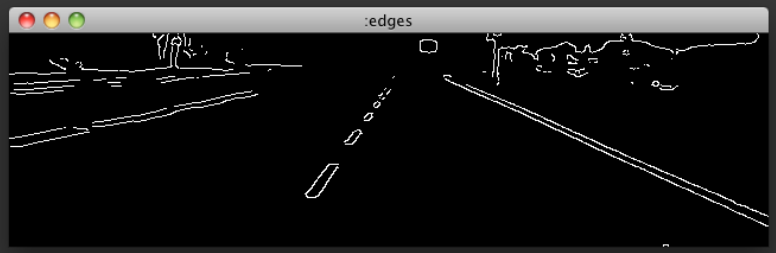
Following snippet demonstrates a quick and dirty way to do lane detection.




```
(ns lane-detection
  (:use vision.core :reload-all))

(defn detect-edges [i]
  (--> (convert-color i :bgr-gray)
       (smooth :gaussian 7 7 0 0)
       (canny 90 90 3)))
```

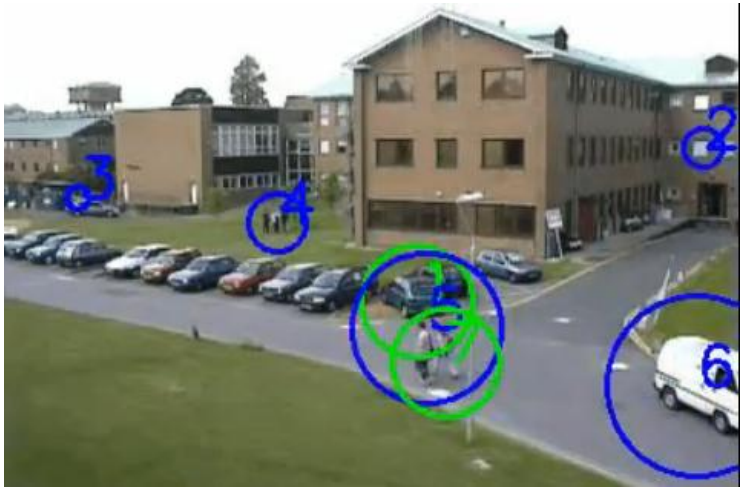
For every frame, we convert it to gray scale, smooth it, and mark the edges on the scene using the Canny algorithm. —> in the code is not a typo, none of the calls in the library modify the original image, instead they all return a brand new image which needs to be released when done, —> works just like the -> in Clojure with the only difference being, it will call release for each intermediate image. After edge detection we end up with the following,



Using Canny Edge Detection
Common Element Object Recogntion

Object Recognition Approach

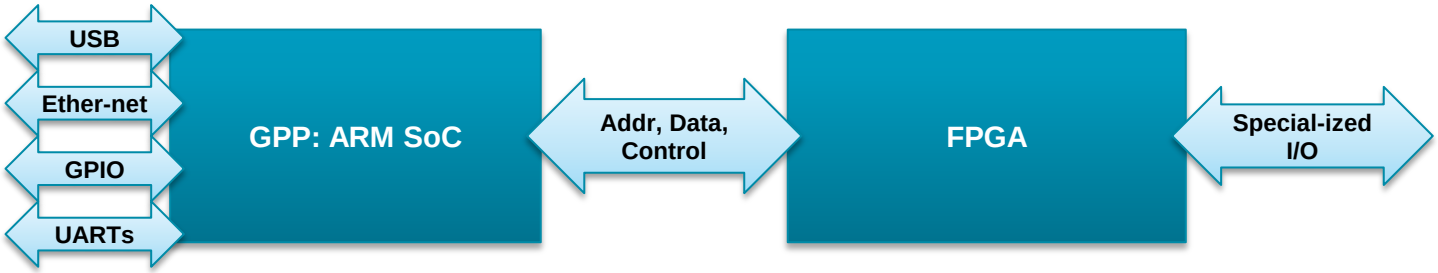
- **Object Recognition**
  - Open CV algorithm
  - Generate co-processing accelerator
  - Test and validate



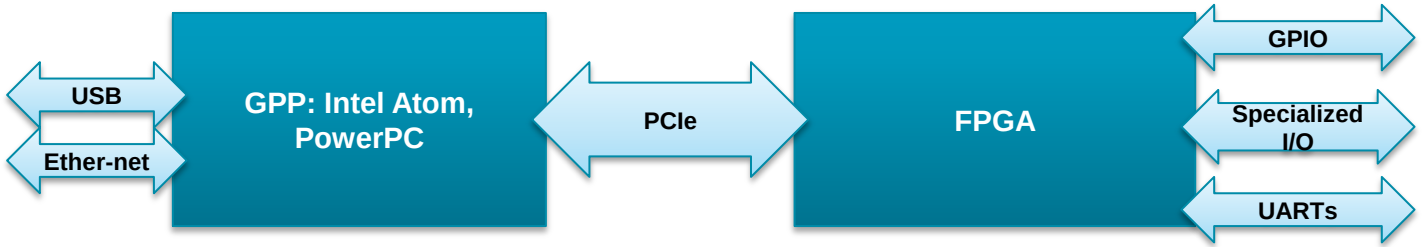
- **Example Image Processing**
  - Utliize the Open CV library
  - Canny algorithm most compute Intensive function

- **Case study focus: Accelerating Canny for Object Recognition Utilizing a Hybrid Processing Platform**

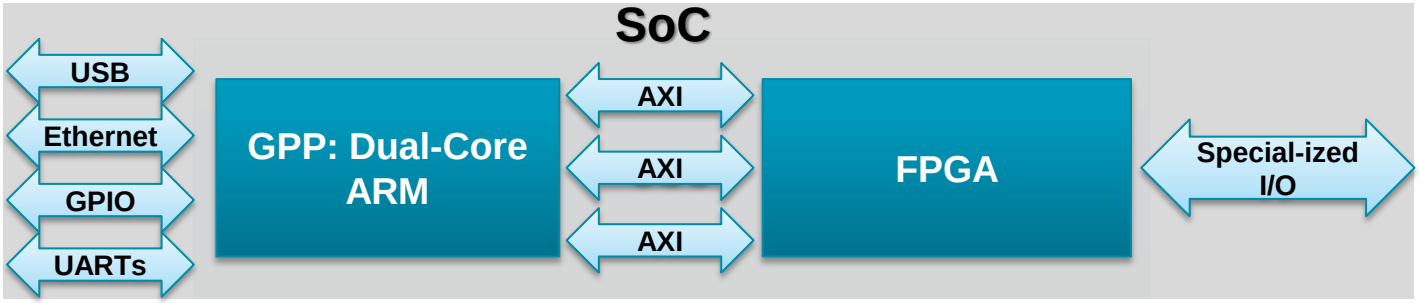
Hybrid Architecture Examples



- Discrete devices connected via general-purpose External memory bus



- Discrete devices connected via External Peripheral: PCI Express



- Processor cores and FPGA fabric integrated on a single device

Hybrid Processing Platform

Definition

- A hybrid processing platform consists of two different types of computational units, one typically being a general-purpose processor (GPP).

  Also referred to as heterogeneous computing

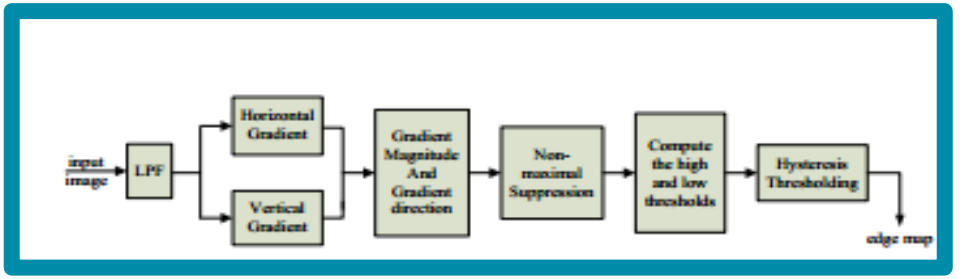
- In the context of this presentation: consists of GPP and field-programmable gate array (FPGA) computational units on an SoC

Benefits

- Computing that can be specialized to target a specific set of applications
  - Massive parallelism - signal processing
  - Extends the peripheral and I/O capability of the GPP
  - Provides a low power, high performance reconfigurable SoC processing platform

Edge Detection Fundamentals: Canny Algorithm

1. **Gaussian Blur to perform Noise Filtering**
   - Utilize 2D Filters to smooth the image
2. **Sobel Filter to determine Intensity Gradient of Image**
   - Convolution masks
3. **Gradient Magnitude and Direction**
   - Determine gradient magnitude & angle at each pixel location
   - Use CORDIC algorithm



4. **Candidate Edge Selection**
   - Suppress pixels not considered as part of an edge

     NMS: Non-maximum suppression
   - Keep thin line candidate edges
5. **Statistical Analysis - Hysteresis**
   - Computing the hysteresis thresholds based on the histogram of the magnitudes of the gradients of the entire image.
   - Performing hysteresis thresholding to determine the edge map

Accelerating using Coprocessing Offload

Primary Edge Detect Functions Offloaded to FPGA

1. Gaussian Blur to perform Noise Filtering
2. Sobel Filter to determine Intensity Gradient of Image
3. Gradient Magnitude and Direction

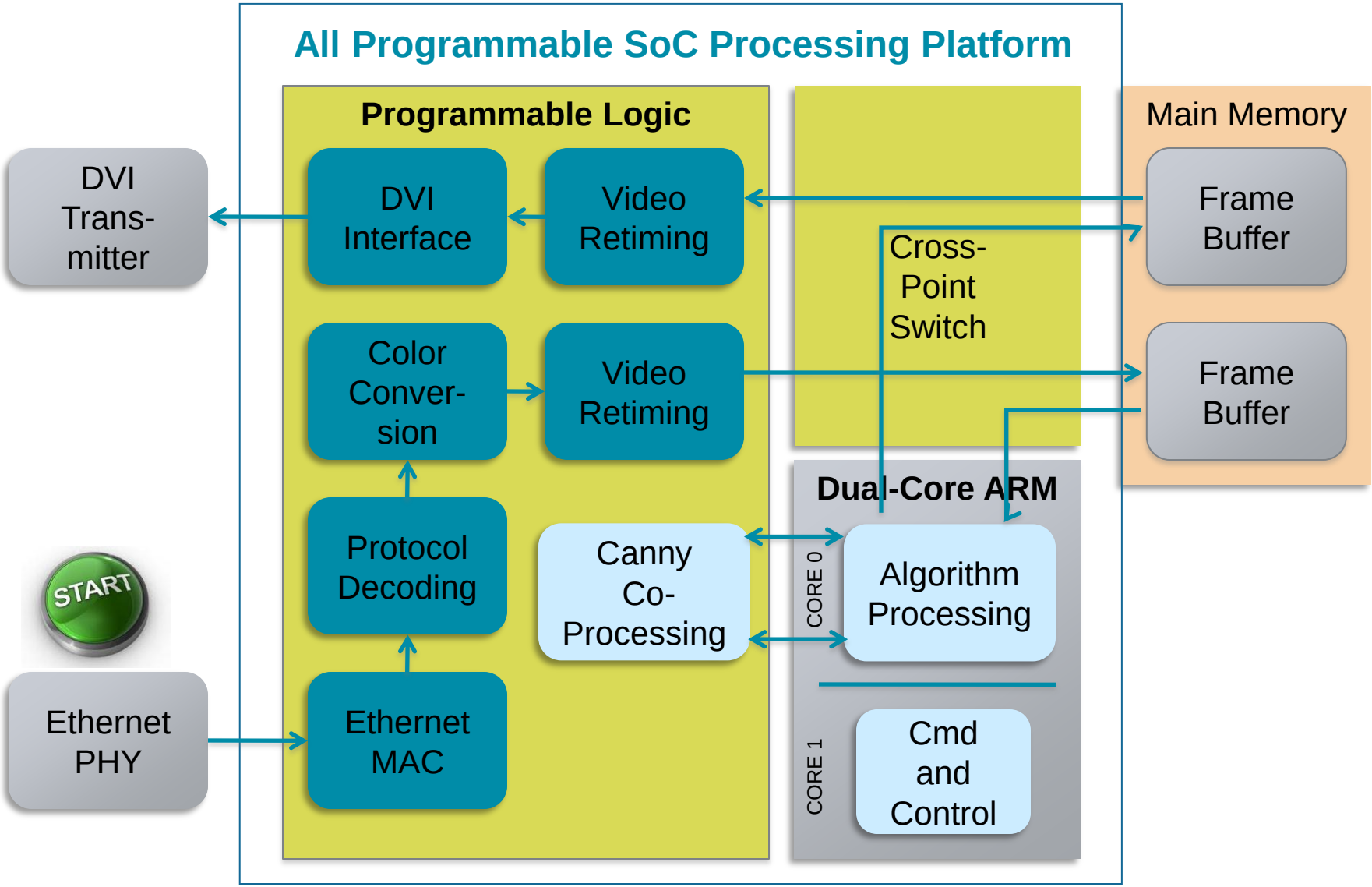
Statistical Functions running on the ARM Processor

4. Edge Selection
5. Statistical functions run on ARM Processor

Benefits

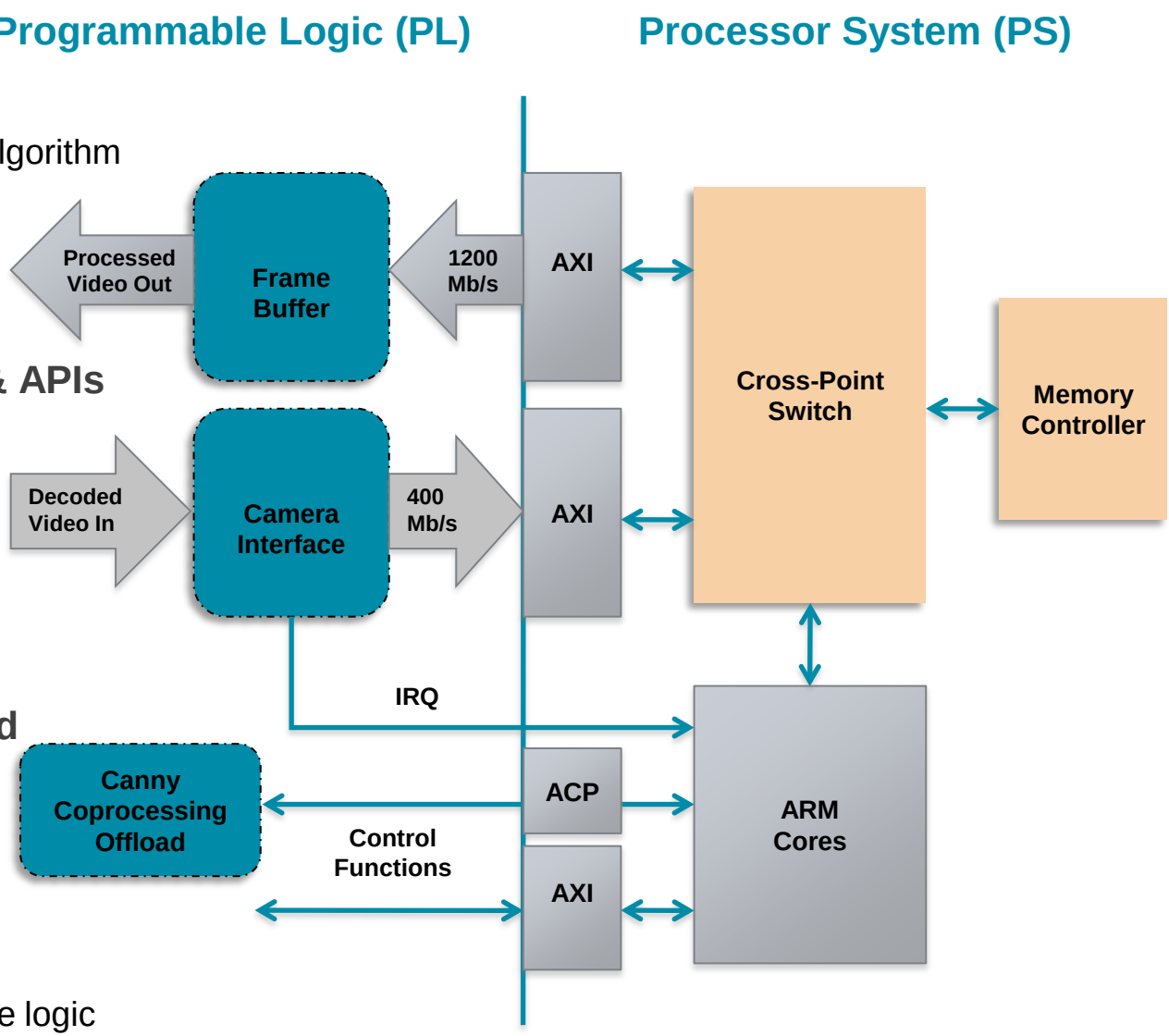
- Offloaded functions run in parallel with the ARM processor
- CPU offloaded freeing up aditional compute capability
- Optimized partitioning to easily add more filtering, image enhancement co-processing engines

Image Processing Application on Hybrid Processing Platform

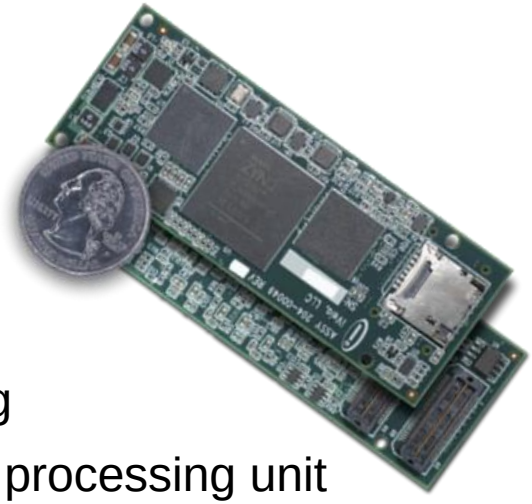


Algorithm Partitioning

- **Dual-core processors:**
  - Core 0: Object Recognition Algorithm
  - Core 1: Command & control: System management
- **Custom Linux AXI drivers & APIs**
  - move data from PL to PS
- **Algorithm re-complied**
  - from OpenCV to ARM
- **Portions of algorithm ported**
  - use NEON extensions for additional acceleration
- **Data flow management**
  - Handled by the programmable logic



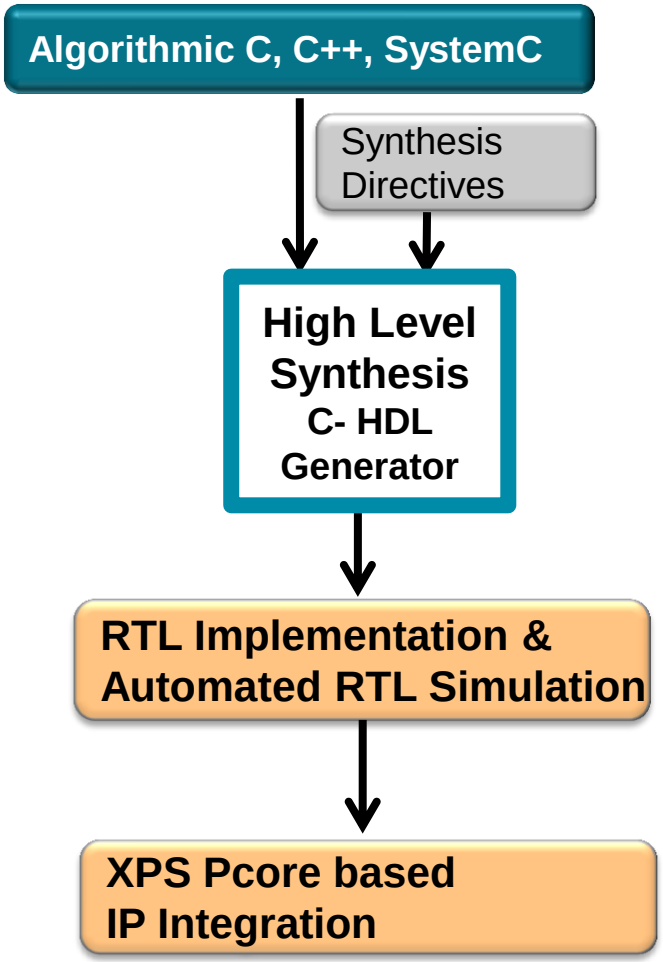
Demo Highlights



- **Embedded Vision Imaging "at the edge"**
  - GigE Vision for a smart camera application
  - Image sensor & processing in the camera housing
  - Transfer compressed video or statistics to central processing unit
- **Object Recognition Algorithms operating on Streaming Video**
  - Demonstrating integral Canny edge detection algorithm
  - Image coprocessor leveraging on-chip programmable logic DSP blocks
- **Showcases Co-processing acceleration**
  - Running Canny algorithm on the ARM compared to same algorithm running in the HW co-processing engines

Reference

Methodology to Generate Coprocessing Accelerators: C-Code to HDL



- **HLS abstracts**
  - Algorithmic description
  - Data types
  - Interfaces
- **Synthesis Directives for Micro-architecture exploration**
  - Loop and function pipelining
  - Extraction of task level parallelism
  - Scheduling and resource allocation
  - Interface generation
- **Accelerated implementation and verification**
  - RTL IP implementation
  - C/C++/SystemC simulation
  - Automatic RTL simulation: No RTL testbench required

ARM NEON Acceleration

Assembly Example

```
iv_neon_mono8_to_rgb16_565:

    . . .

loop_m8rgb16_565:

    # Load 8 of the 8-bit pixels into d0. Increment source address (r1)
    vld1.64         {d0}, [r1]!

    # Expand the vectors from 8 to 16-bits, mask R values
    vmov.i64        d1, #0
    vzip.8          d1, d0
    vand            d2, d1, d6

    # Shift R value right by 5 to get G value (put in d3)
    vshr.u16        d3, d2, #5

    # Shift R value right by 11 to get B value (put in d4)
    vshr.u16        d4, d2, #11

    # Or all 3 vectors to get 565 pixels
    vorr            d1, d2, d3
    vorr            d1, d1, d4

    # Put pixels into destination buffer and increment dest (r3)
    vst1.64         {d1}, [r3]!

    # Repeat process for high-order pixels in d0
    . . .

    # Decrement pixel count by 8 and loop
    subs            r2, r2, #8
    bgt             loop_m8rgb16_565
    . . .
```

- SIMD instruction set extensions for the ARM processor
- In our experience, 2-10x performance of DSP functions
- Auto-vectorization compilers, Intrinsics, or Assembly

Intrinsics

```
#include <arm_neon.h>

uint32x4_t double_elements(uint32x4_t input)
{
    return(vaddq_u32(input, input));
}
```

Resultant Image Processing Platform
Ultracompact form factor