# Hardware Acceleration of Feature Detection and Description Algorithms on Low-Power Embedded Platforms

Onur Ulusel, Christopher Picardo, Christopher Harris,
Sherief Reda, R. Iris Bahar,
School of Engineering, Brown University

BROWN

## Image Processing in Mobile Systems

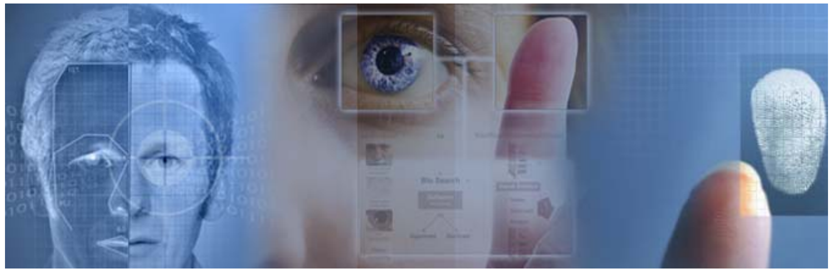
- Image processing is everywhere!
  - Input data has changed from words/numbers to images
  - Sensors have improved dramatically
- Image processing is a major driving factor in technological advancement
  - Autonomization relies on image processing
- Mobile/Embedded platforms??
  - Real-time computing + limited data bandwidth ➔ prefer local computing to offloading to cloud
  - ***BUT*** image processing can be very computationally intensive and power hungry

www.guardiantv.com

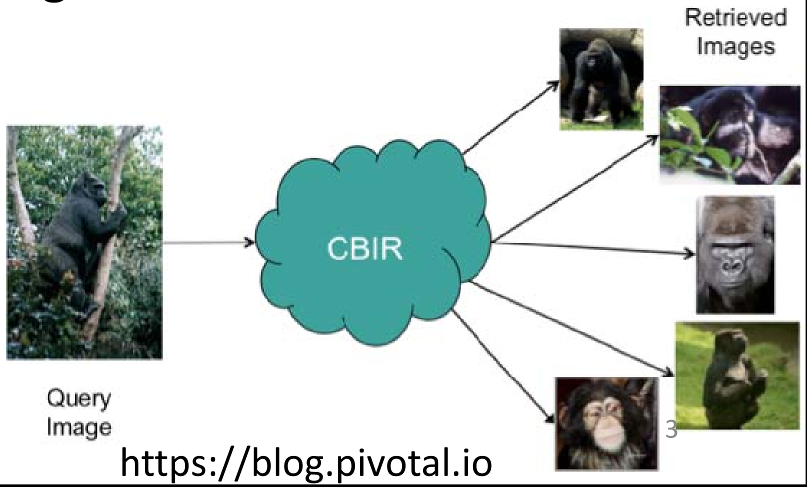
www.google.com

# Accelerating Image Processing on Low Power Embedded Platforms

- Meeting real time image processing requirements for many of these applications requires HW assisted acceleration
- Which algorithms do we accelerate?
  - *Feature detection* and *feature description* are key building blocks for image retrieval, biometric identification, visual odometry, etc.
  - Computational efficient detection and analysis of image features is critical for *performance* and *energy-efficiency*


http://www.sybernautix.com/

https://vision.in.tum.de



https://blog.pivotal.io

# Hardware Acceleration for Energy Constrained Image Processing

- Low power embedded platforms
  - Field Programmable Gate Arrays (FPGAs)
  - Graphical Processing Units (GPUs)
  - Low power general processors

**Which platform is best?**

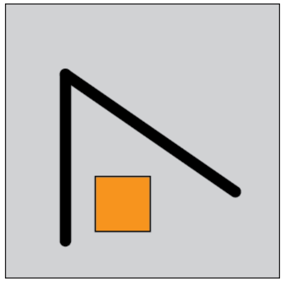
| | FPGA | | GPUs | |
|---|---|---|---|---|
| | Xilinx | 020 | 1532 core NVIDIA GeForce GTX 680 | 192 core NVIDIA Jetson TK1 |
| Pow | | <5W | 195 W | <12W |

## Our Contributions

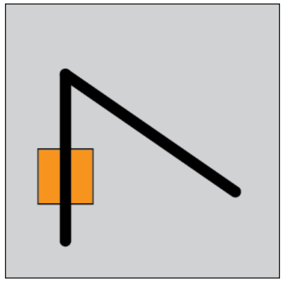
- Comparative study of feature detection and description algorithms
  - *What are their computation kernel characteristics?*
- Comparative study of platforms for *embedded* applications
  - *Advantages/disadvantages of each platform?*
- Accelerating algorithms on different platforms
  - *How can algorithms be modified to better exploit available hardware of each platform?*
  - *How does performance compare in terms of **run time** and **energy consumption**?*

## Feature Detection

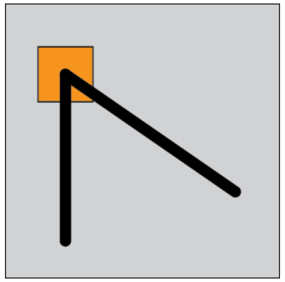
- What is a ‘*feature*’?
  - An “interesting” part of an image that can be used to identify objects
- Examples: Edges, corners, ridges, blobs



flat region:
no change
within block



edge:
no change
along the edge

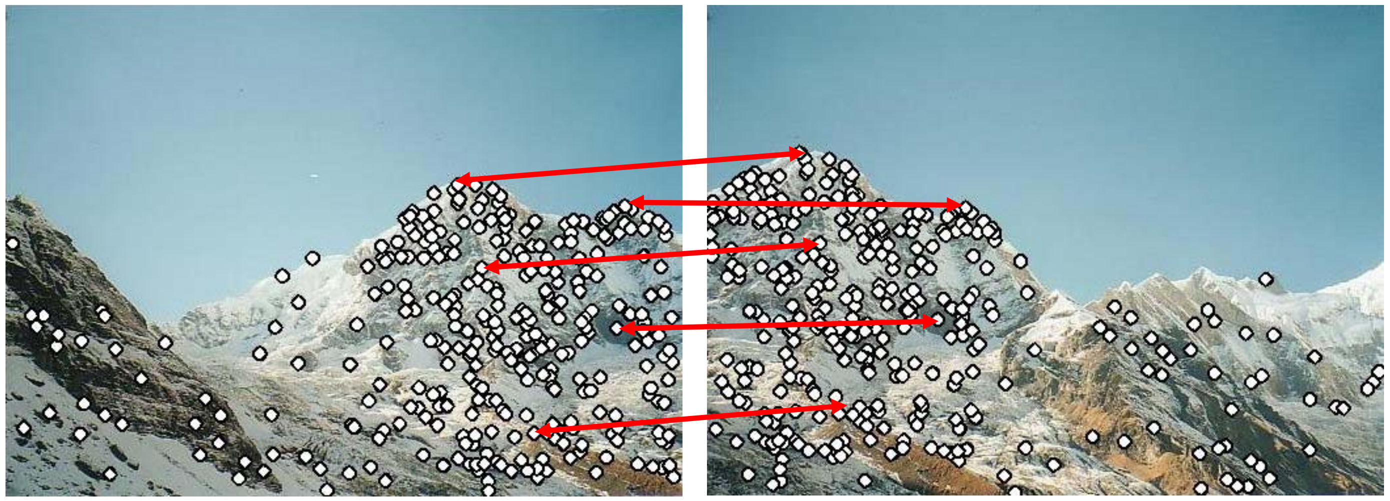


corner:
change in
all directions

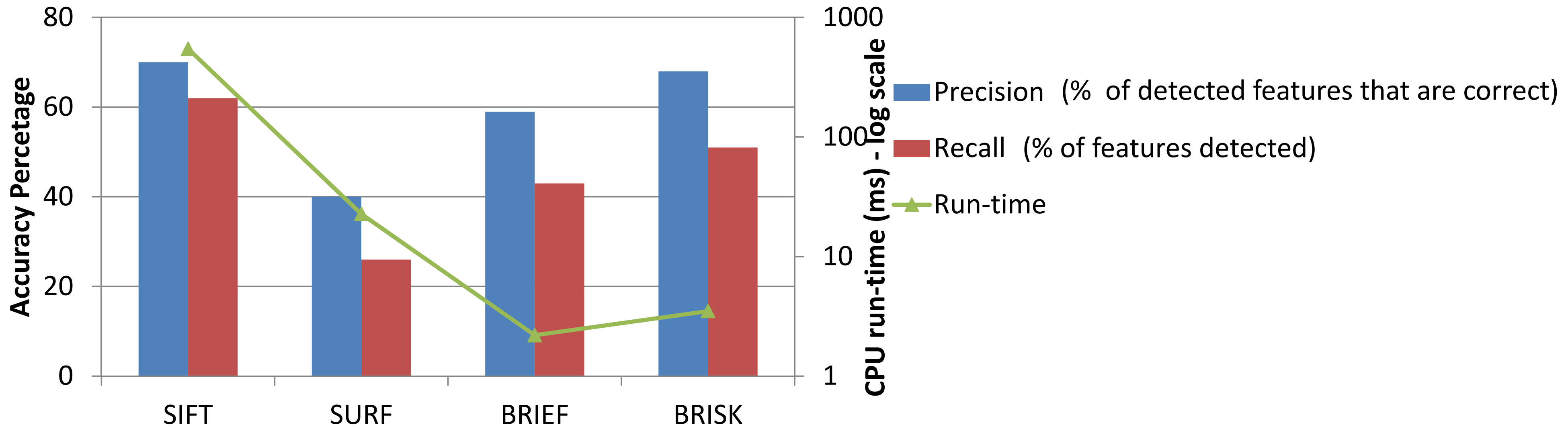
Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute

# Feature Description

- Given the features, *uniquely describe* them so they can be matched in other images
- Descriptors summarize characteristics of the features
  - E.g., intensity, orientation
- Descriptors should be distinctive and insensitive to local image deformations.

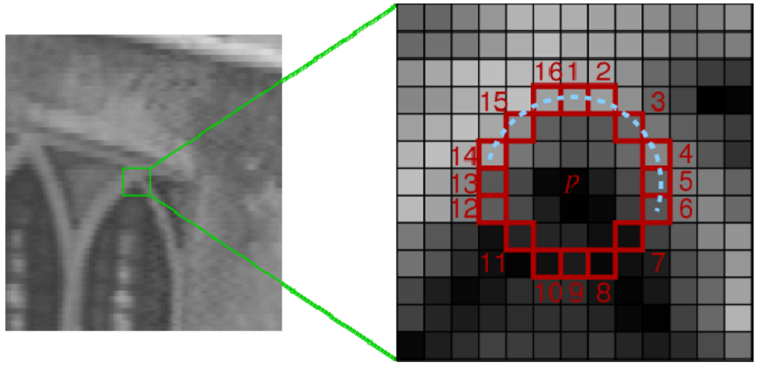

Images from: R. Szeliski, *Computer Vision: Algorithms and Applications*

# Accuracy and Run-time Comparisons

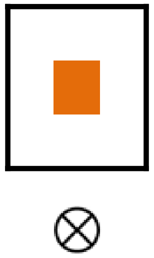


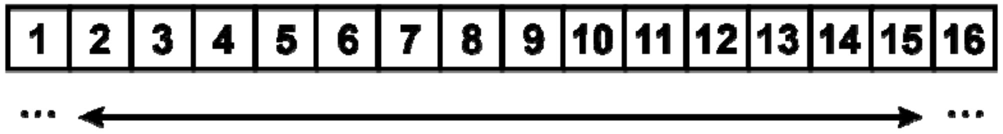
- HoG (Histogram of Gradient) based Descriptors
  - SIFT: Scale-Invariant Feature Transform
  - SURF: Speeded Up Robust Features
- Binary Feature Descriptors
  - BRIEF: Binary Robust Independent Elementary Features
  - BRISK: Binary Robust Invariant Scalable Keypoints

# FAST: Features from Accelerated Segment Test



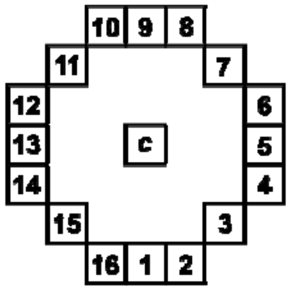
Rosten and Drummond, ECCV'06



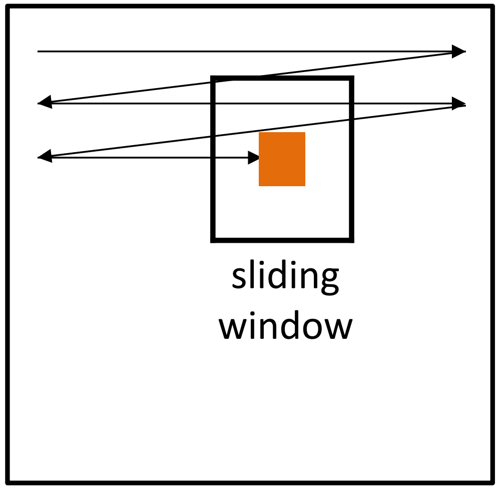


12-pixel continuity? → If so then feature

Pre-compare pixels 1, 5, 9, and 13 to determine possibility for continuity



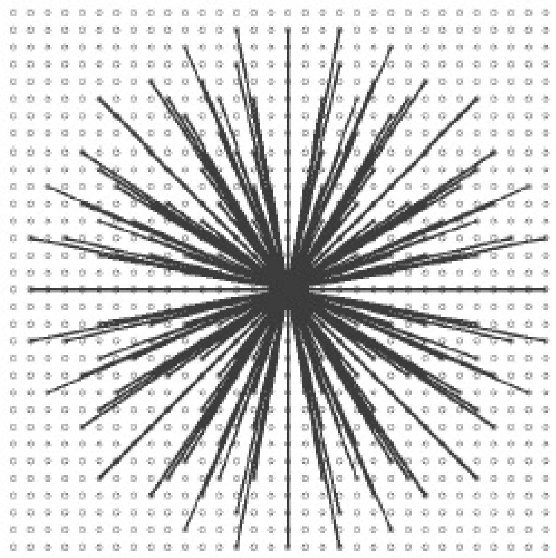
Bresenham Circle



On average 98.5% of the comparisons fail the continuity test at the pre-compare stage

# BRIEF: Binary Robust Independent Elementary Features

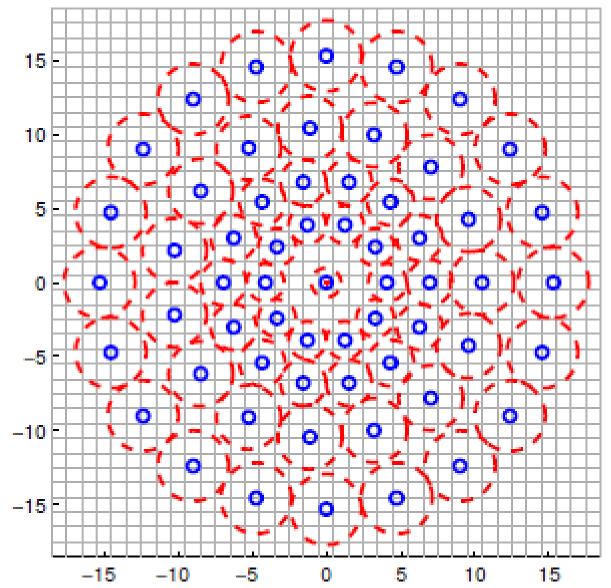
- Compare intensities of pairs of points using Hamming distance
- BRIEF Sampling pattern
  - 512 sampling pairs
  - For each pair, $X_i$ is at (0,0) and $Y_i$ takes all possible values from coarse polar grid
  - Sampling pairs are generated from a 31×31 region around center pixel



Chosen sampling pattern results in a 512-bit characterization array

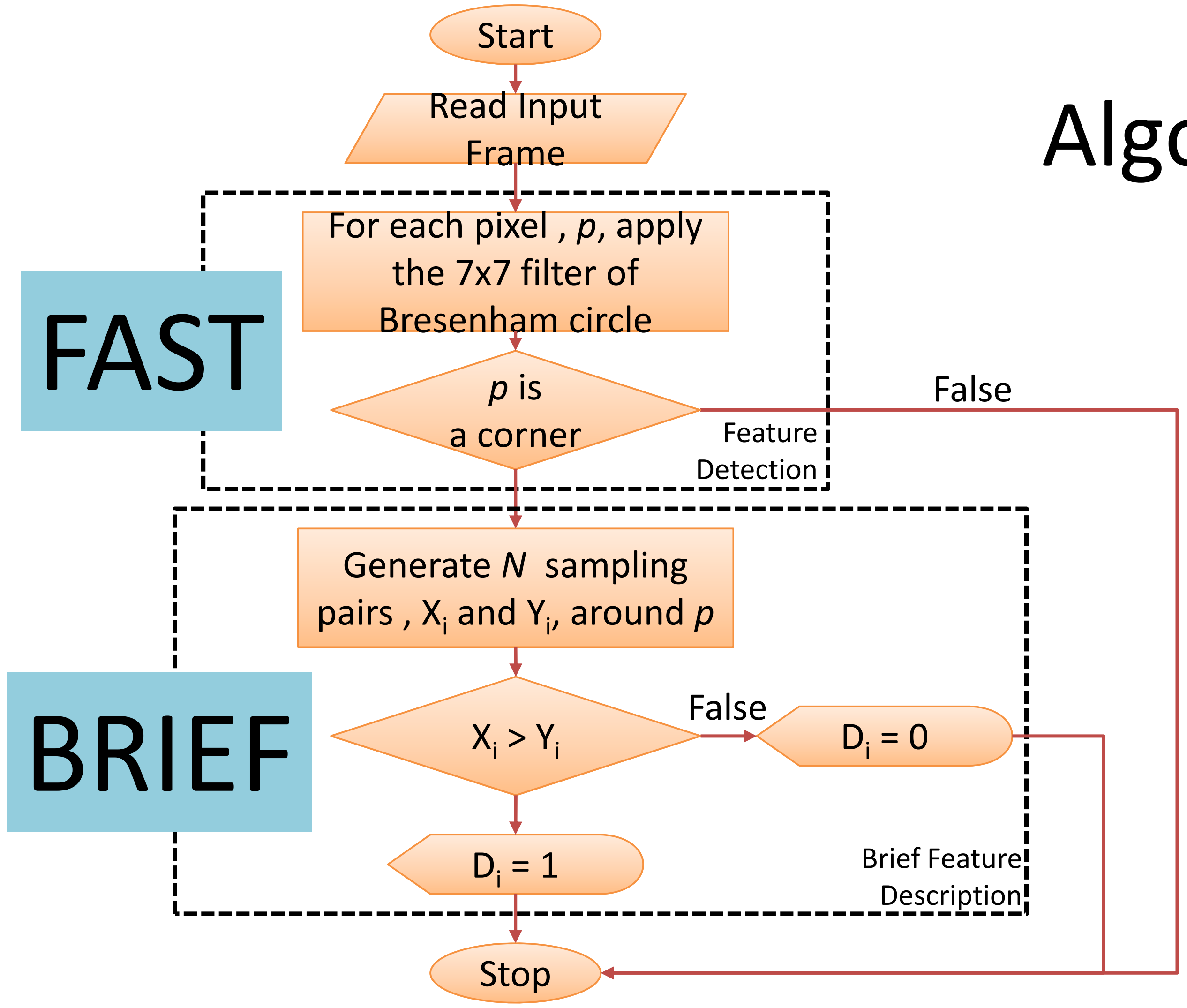
# BRISK: Binary Robust Invariant Scalable Keypoints

- BRISK uses custom sampling pattern
- 512 sampling pairs generated from a 31×31 region (like BRIEF)
- Distinguishes between *short/long pairs*
  - Short pairs used similar to BRIEF to generate descriptor vectors based on *intensity comparisons*
  - Long pairs used for *orientation computation* by rotating sampling pattern



BRISK sampling pattern
Red circles represent standard deviation of Gaussian smoothing

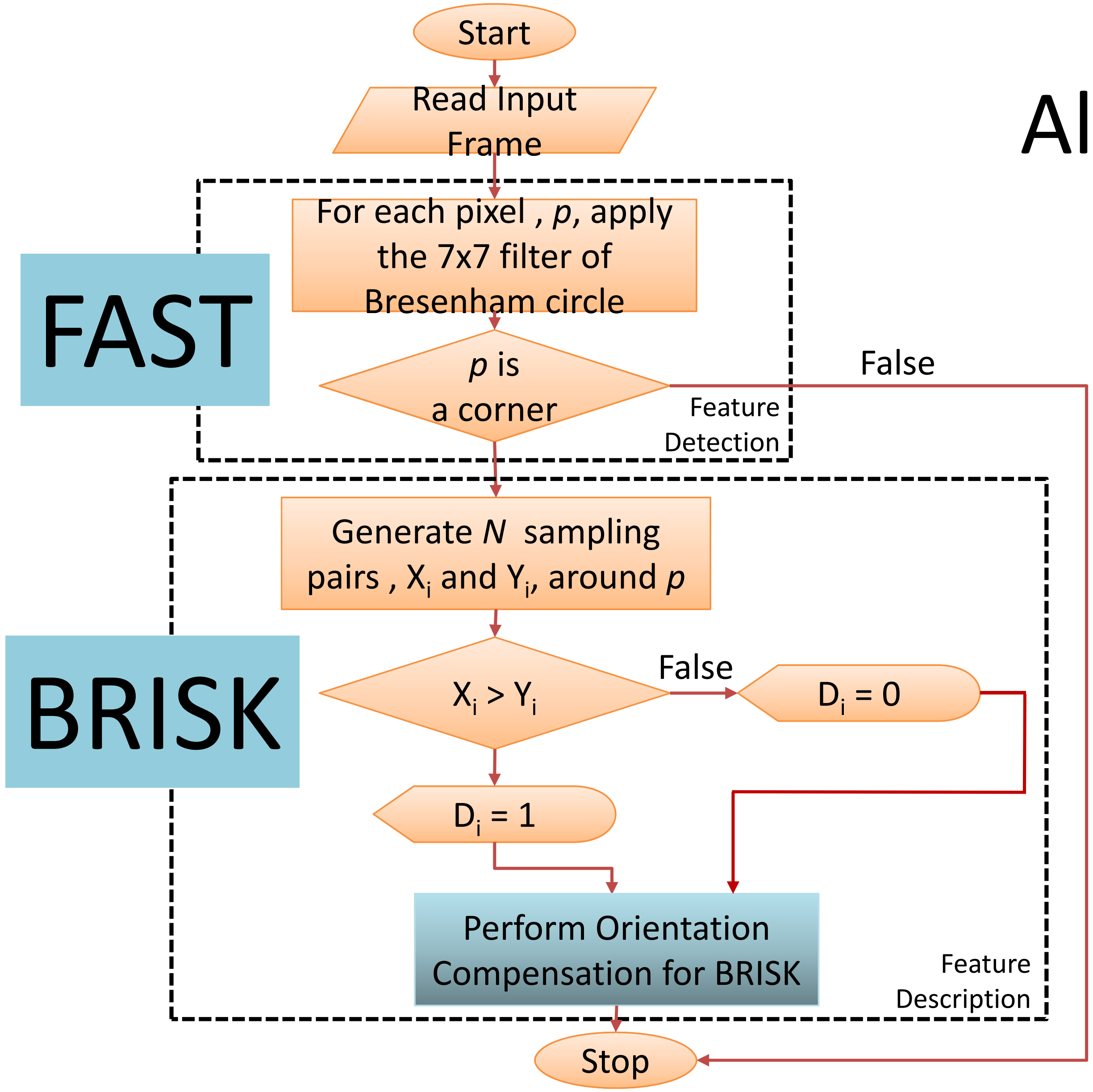
# Algorithm Flowchart



Start
Read Input Frame
For each pixel , *p*, apply the 7x7 filter of Bresenham circle
*p* is a corner
False
Feature Detection
FAST
Generate *N* sampling pairs , $X_i$ and $Y_i$, around *p*
$X_i > Y_i$
False
$D_i = 0$
$D_i = 1$
Brief Feature Description
BRIEF
Stop

- FAST feature detection + BRIEF feature description
- Obtaining sampling window for feature description requires irregular access pattern

# Algorithm Flowchart

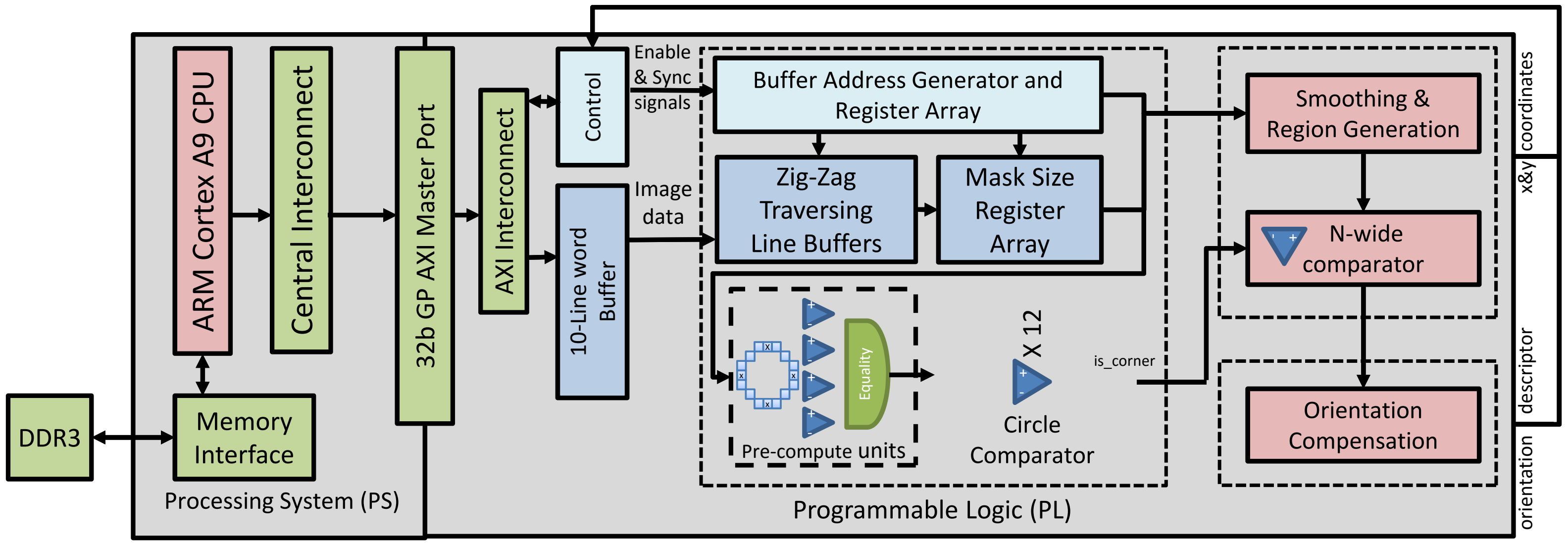

Start
Read Input Frame
For each pixel , $p$, apply the 7x7 filter of Bresenham circle
$p$ is a corner
False
Feature Detection
FAST
Generate $N$ sampling pairs , $X_i$ and $Y_i$, around $p$
$X_i > Y_i$
False
$D_i = 0$
$D_i = 1$
BRISK
Perform Orientation Compensation for BRISK
Feature Description
Stop

- FAST feature detection + BRISK feature description
- BRISK requires an extra step for orientation compensation
  - A significant amount of extra hardware resources for this step

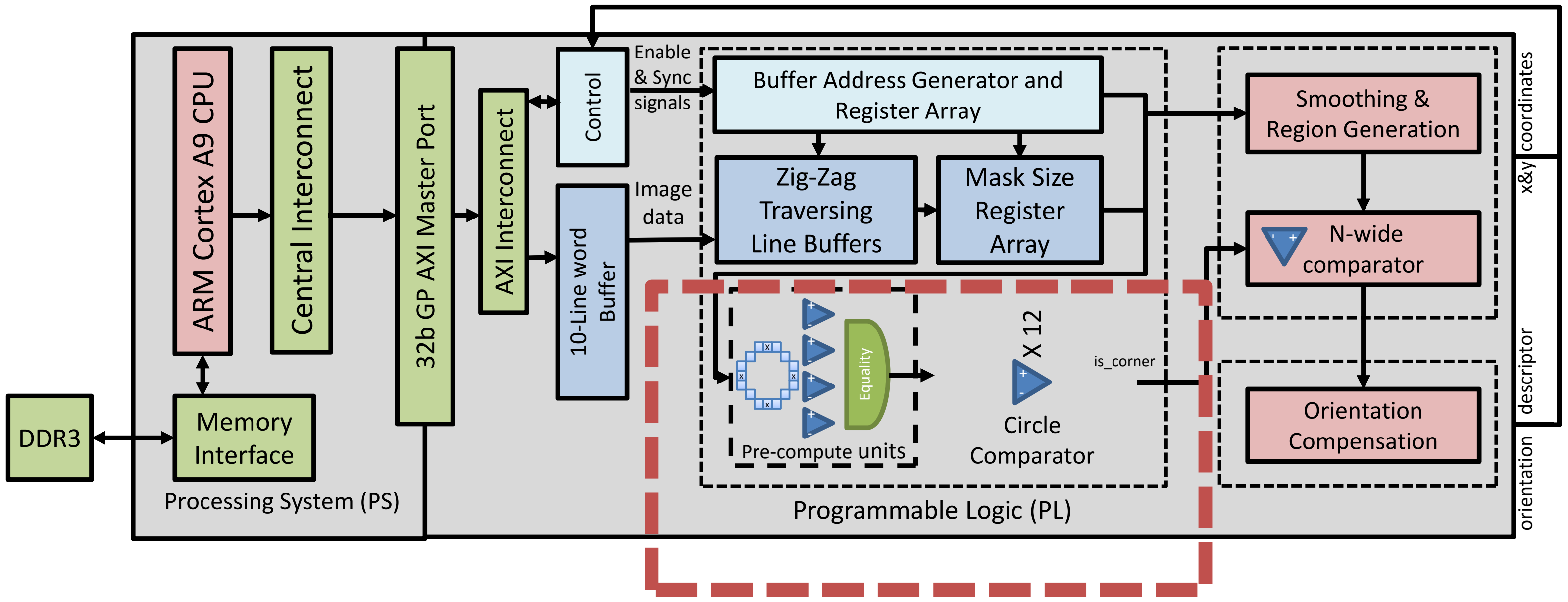
## Experimental Embedded Platforms

- FPGA: MicroZED development board:
  - 28nm Zynq 7020 SoC
  - Artix-7 FPGA + 1GB DDR3
  - dual-core Arm Cortex A9 CPU (for debug and init. only)
- GPU & CPU: Jetson TK1 development kit
  - 28nm Tegra K1 SoC
  - Kepler GPU with 192 CUDA cores @ 950MHz
  - Quadcore ARM Cortex A15 CPU @ 2.5GHz (single core activated)
  - 2GB Memory
  - Running OpenCV versions of FAST, BRIEF, BRISK

# Feature Detection & Description: Block Diagram

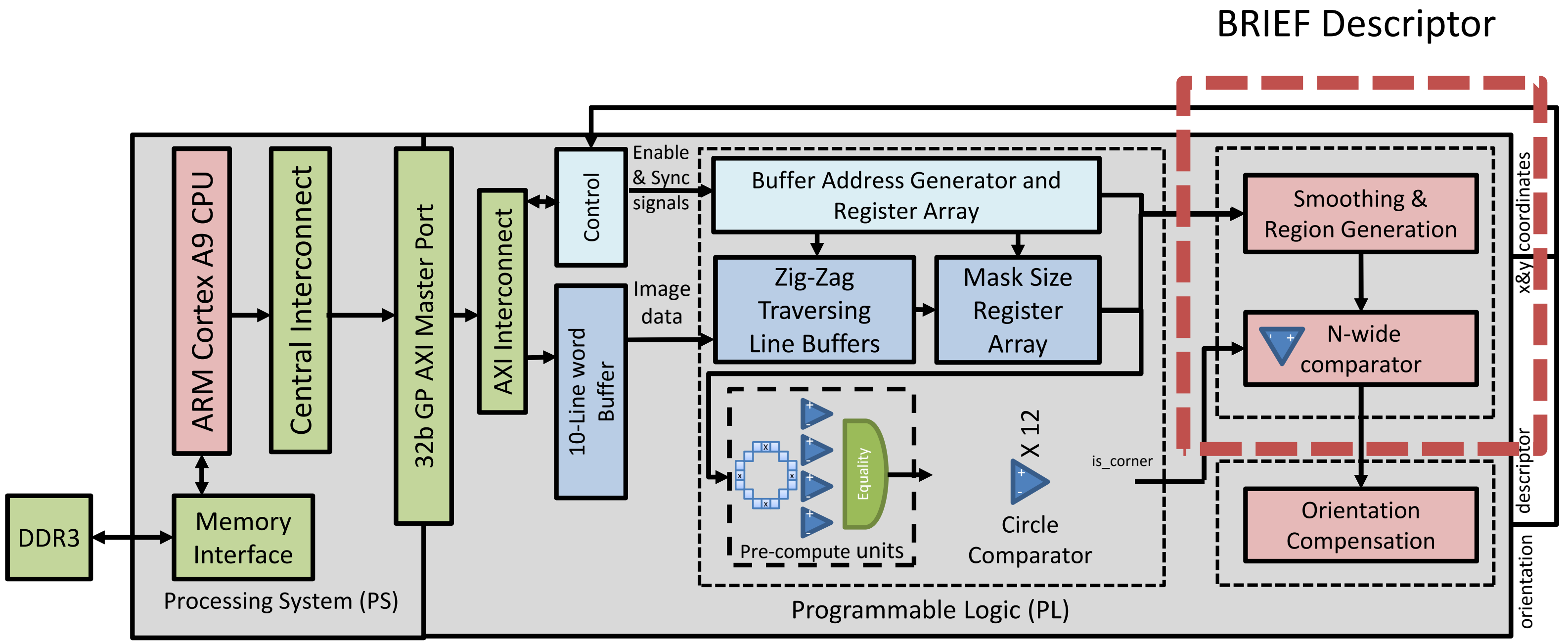


# Feature Detection & Description: Block Diagram

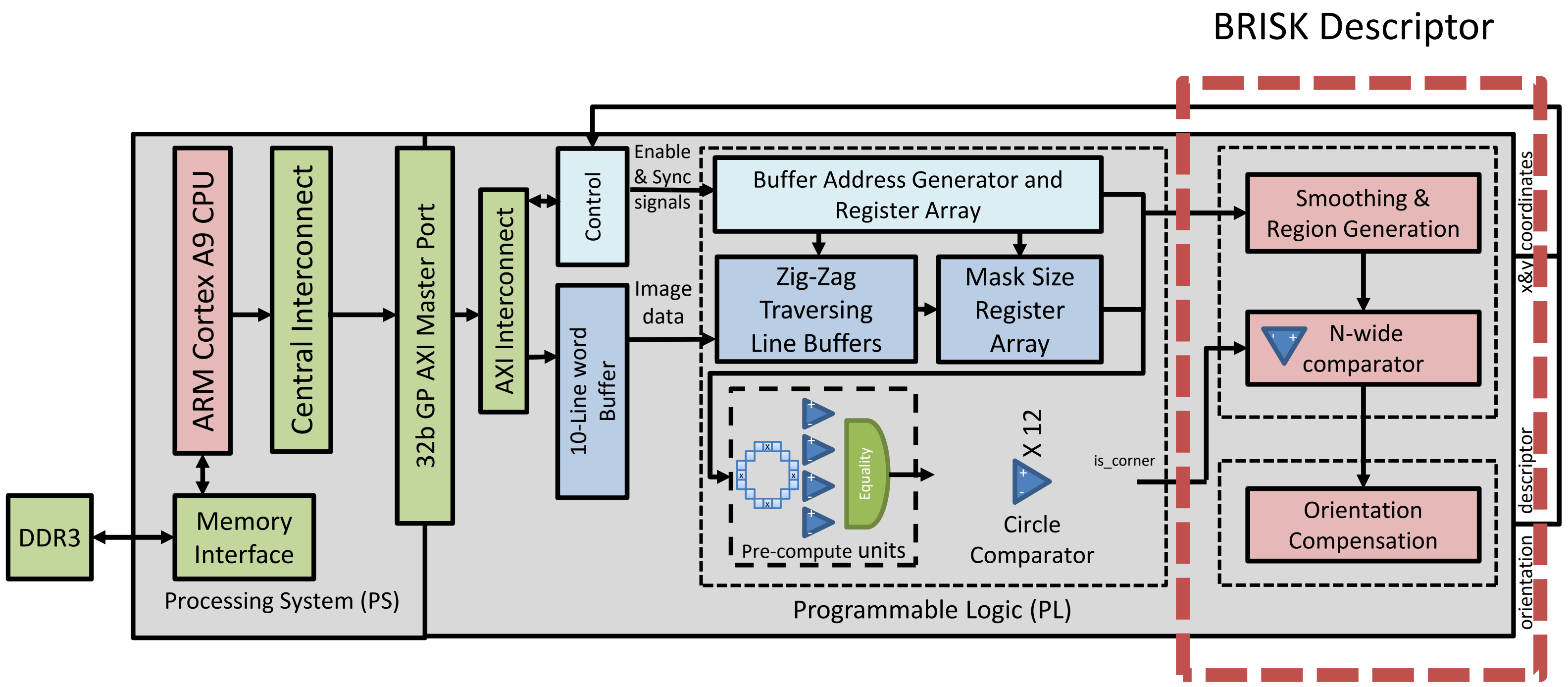


FAST Feature Detection

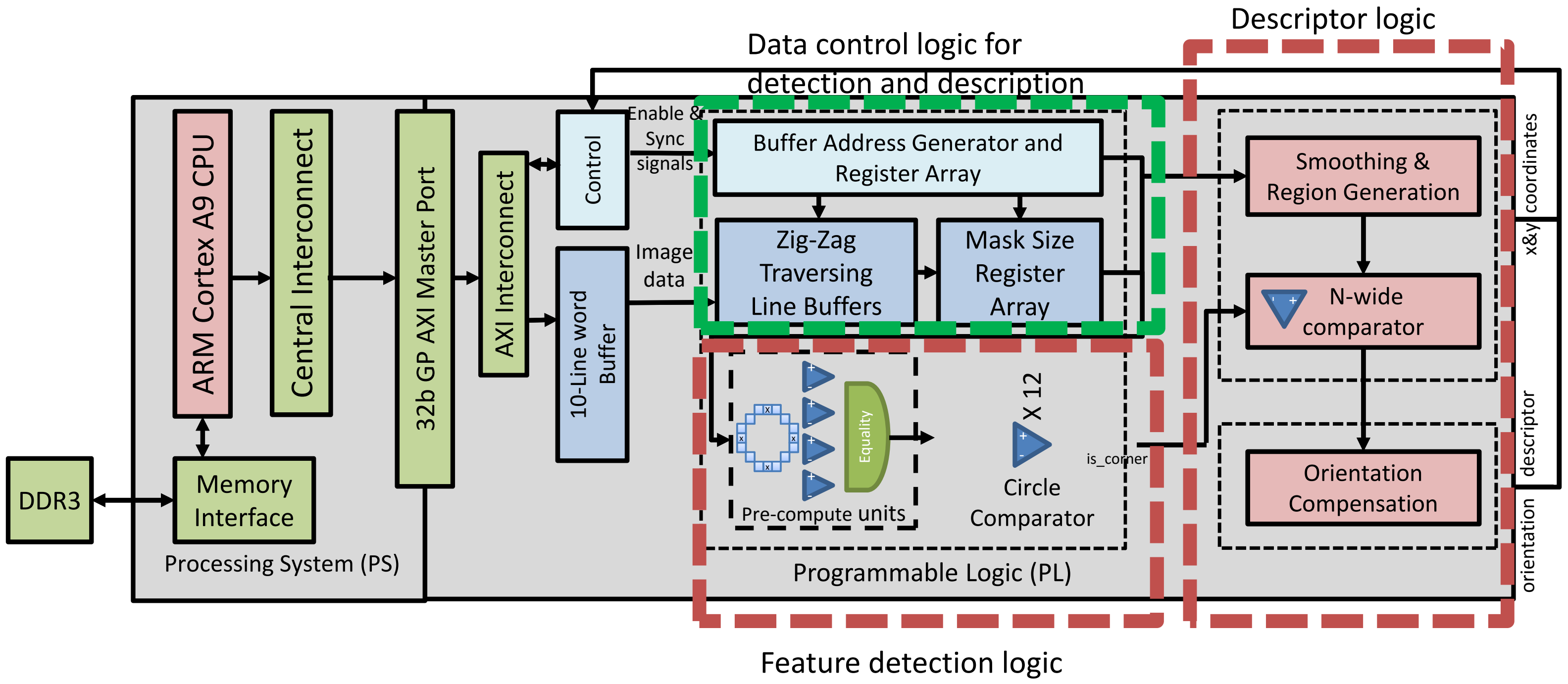
# Feature Detection & Description: Block Diagram



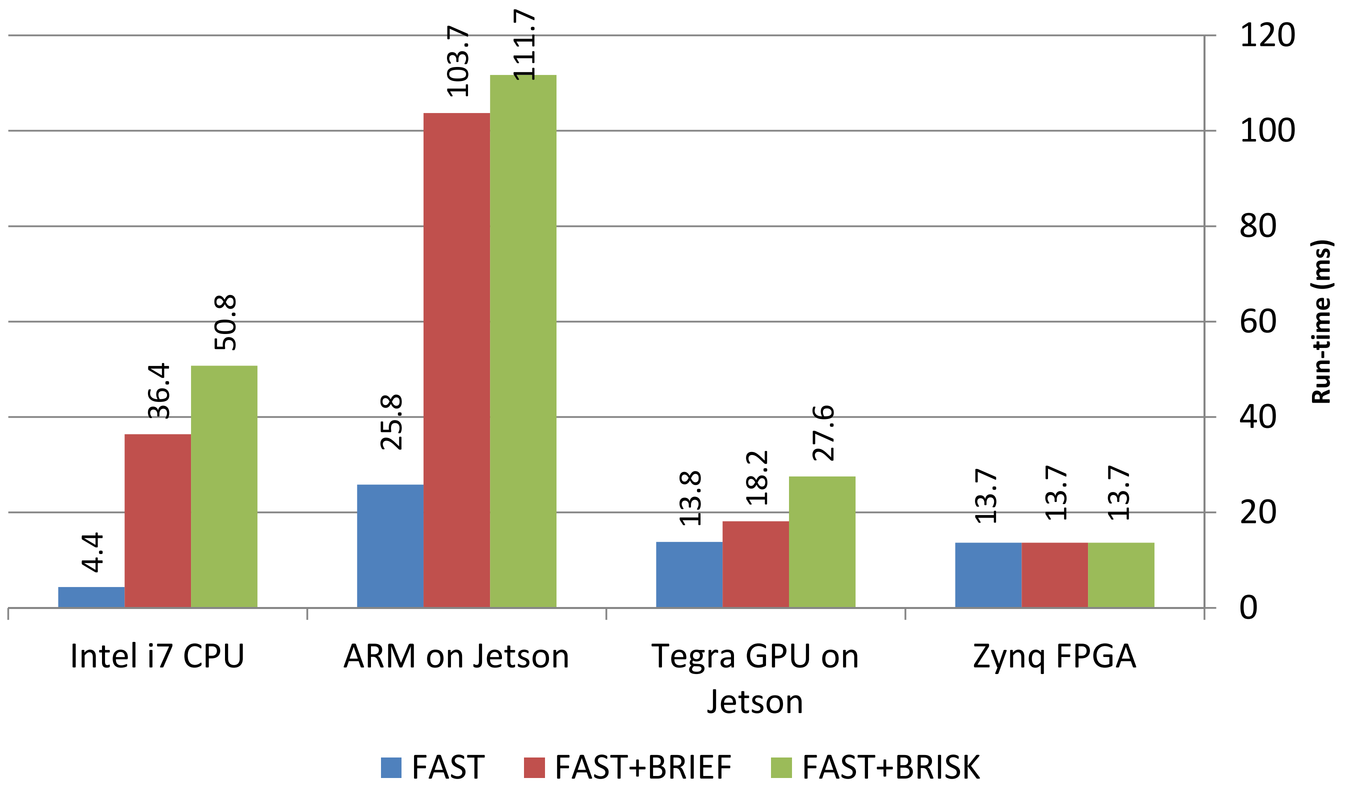
# Feature Detection & Description: Block Diagram



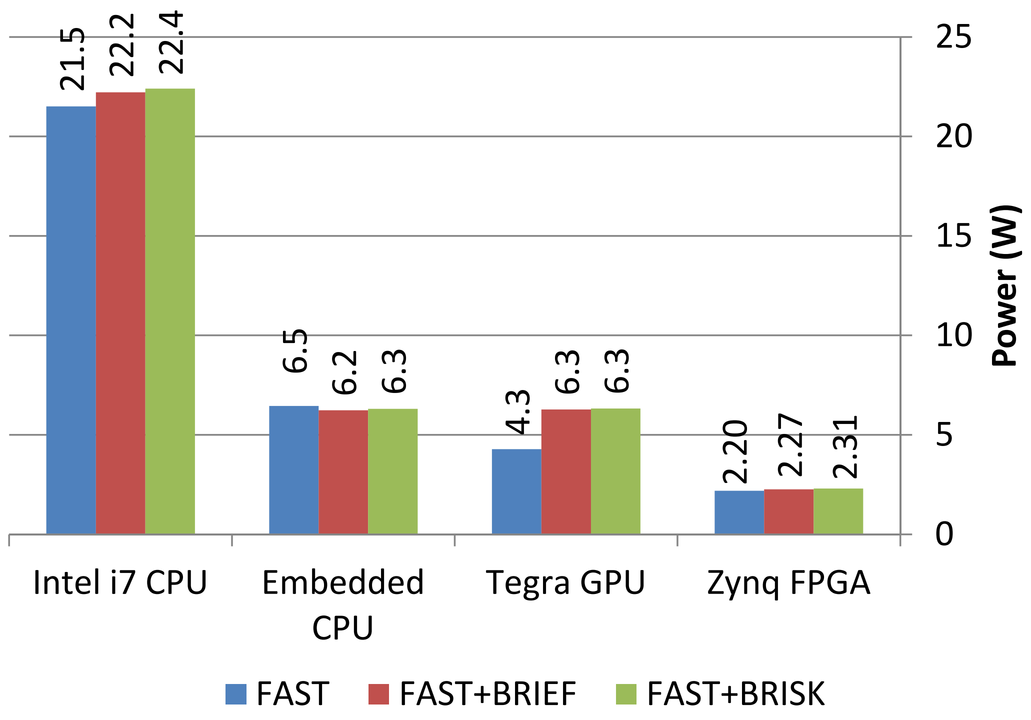
# Feature Detection & Description: Block Diagram

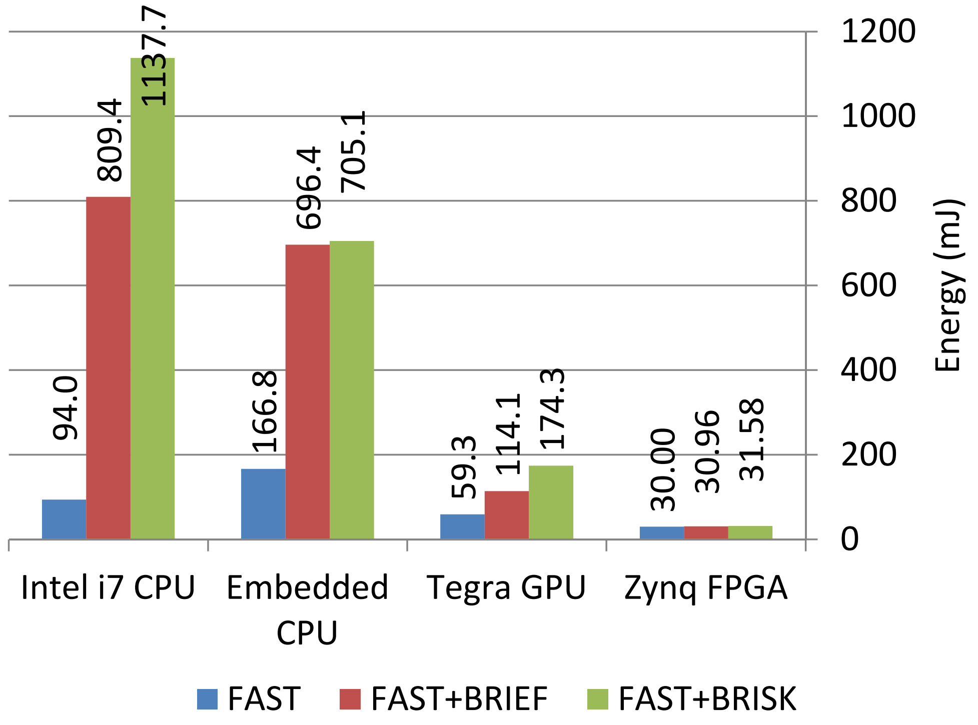


## Results: Run-time



# Results: Power & Energy

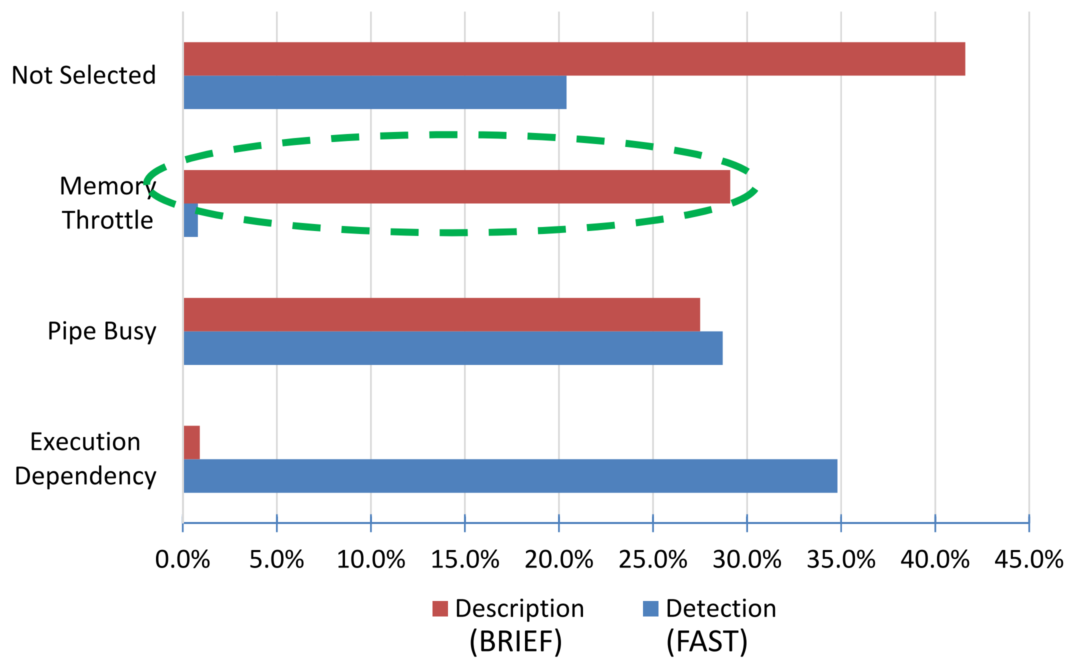




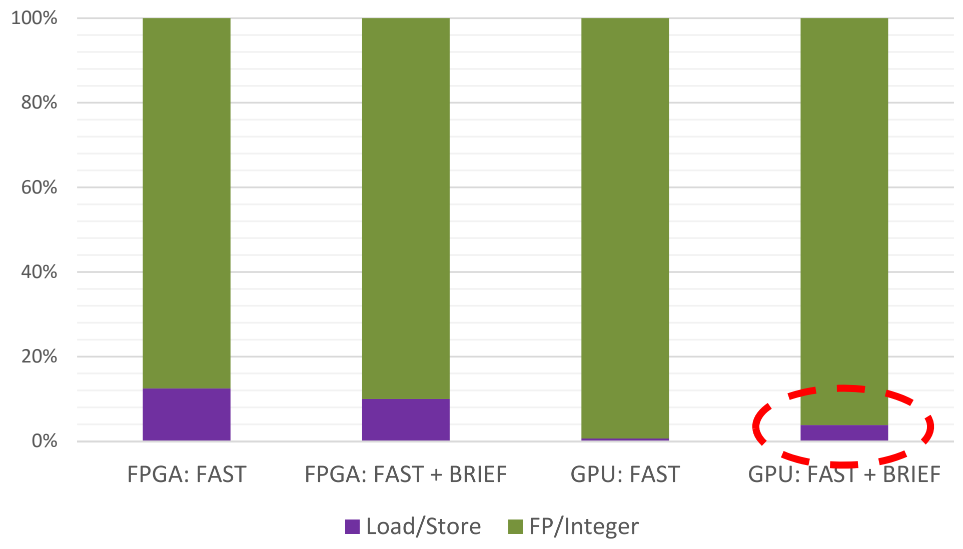
🟦 FAST (detection) 🟥 FAST+BRIEF 🟩 FAST+BRISK (detection + description)

# Results: Profiling

Stall Reasons during GPU Computation



Instruction Distribution



- Feature description stalled due to memory throttle
  - Needs better data management

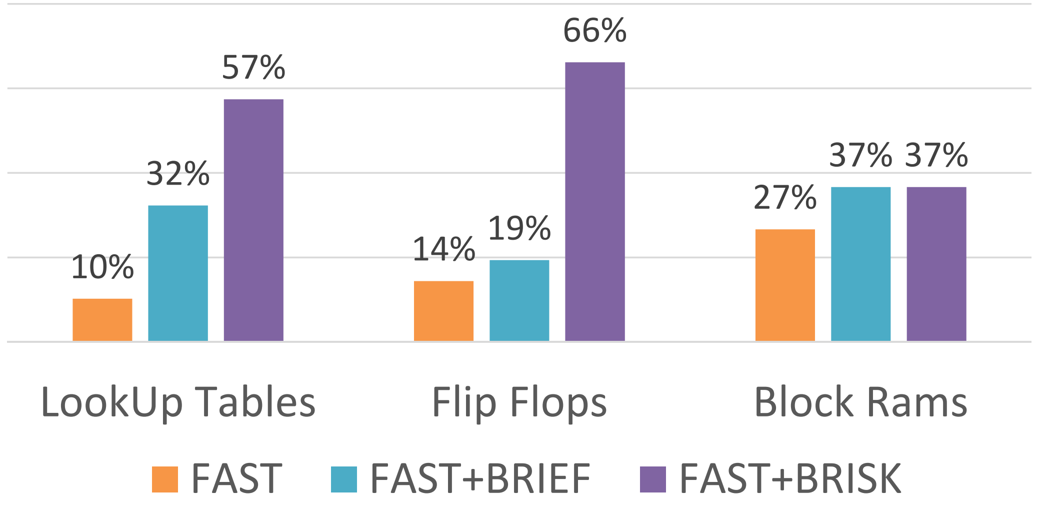
- Feature description for GPU implementation has bump in load/store ops
  - Almost 10X more than just FAST

# Results: FPGA Resource Utilization

Resource utilization

| | Lookup Tables | Flip Flops | Block RAMs |
|---|---|---|---|
| FAST | 4564 | 1551 | 8 |
| FAST + BRIEF | 14398 | 2093 | 11 |
| FAST + BRISK | 25575 | 7115 | 11 |

Distribution of resources



- BRISK requires significant amount of extra resources for smoothing and orienting
- Extra resources do not translate to much extra power

## Conclusions

- FPGA outperforms CPUs and GPUs in terms of power & performance
  - FAST + BRISK:  36 fps vs. 147 fps
  - FPGA amenable to various HW optimizations:
    - deep pipelining, optimized memory access, pre-computation
- FGPA implementations better for handling multiple kernels
  - For GPUs, multiple kernels highly bounded by kernel scheduler and memory bottlenecks
  - FPGA customization on layers better for tackling operations on multiple kernels.
- Use profiling on GPU implementation as first step to FPGA optimization
  - identify nature of bottlenecks
  - Customized FPGA HW can often better manage certain types of bottlenecks