FPGA implementations of Histograms of Oriented Gradients in FPGA

C. Bourrasset[1] , L. Maggiani[2,3], C. Salvadori[2,3], J. Sérot[1], P. Pagano[2,3] and F. Berry[1]

[1] Institut Pascal- D.R.E.A.M - Aubière, France

[2] TeCIP Institute - Scuola Superiore Sant'Anna - Pisa

[3] CNIT – National Laboratory of Photonic Networks - Pisa

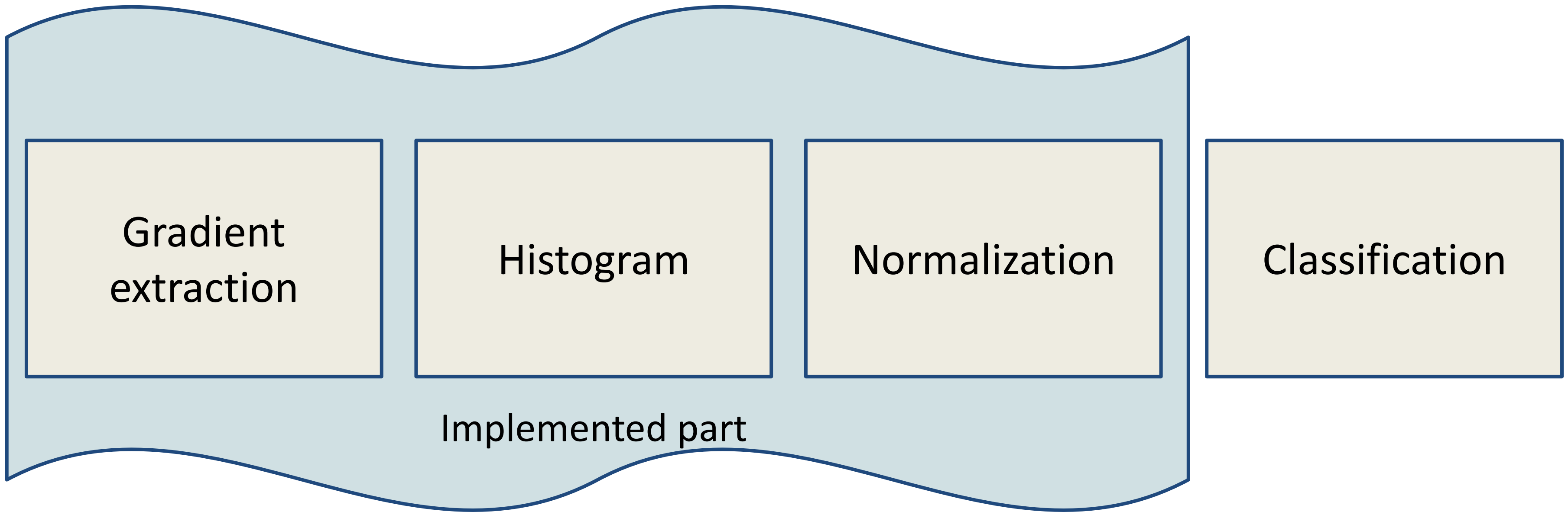
Outline

- Introduction
  - Histogram of Oriented Gradients (HOG) pipeline
  - Hardware vs. Software implementation
- Hardware-Software Codesign approach
  - Gradient extraction
  - Histogram generation
  - Normalization
  - Implementation results
- Domain Specific Language approach
  - CAPH language
  - Implementation results

Introduction

HOG pipeline[1]

In 2005 Dalal presents the HOG pipeline



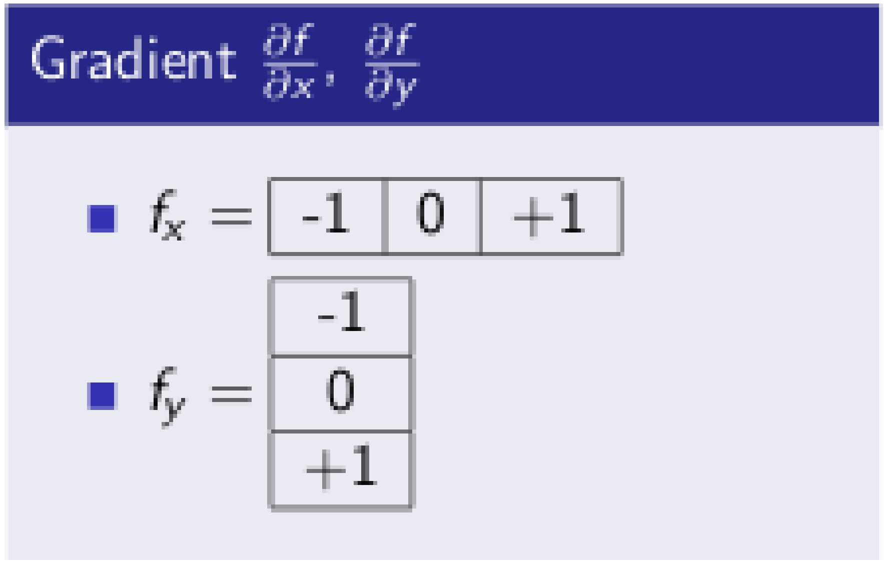
[1] *Histograms of Oriented Gradients for Human Detection*, Dalal and Triggs, INRIA, 2005

Gradient Extraction

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$

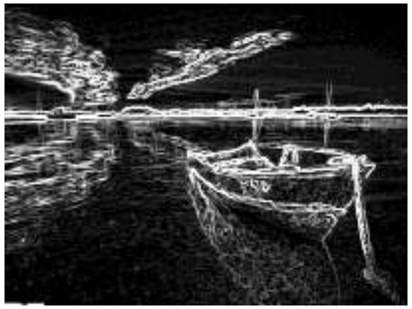
$$\frac{\partial I}{\partial x} \equiv I(x+1,y) - I(x-1,y)$$

$$\frac{\partial I}{\partial y} \equiv I(x,y+1) - I(x,y-1)$$



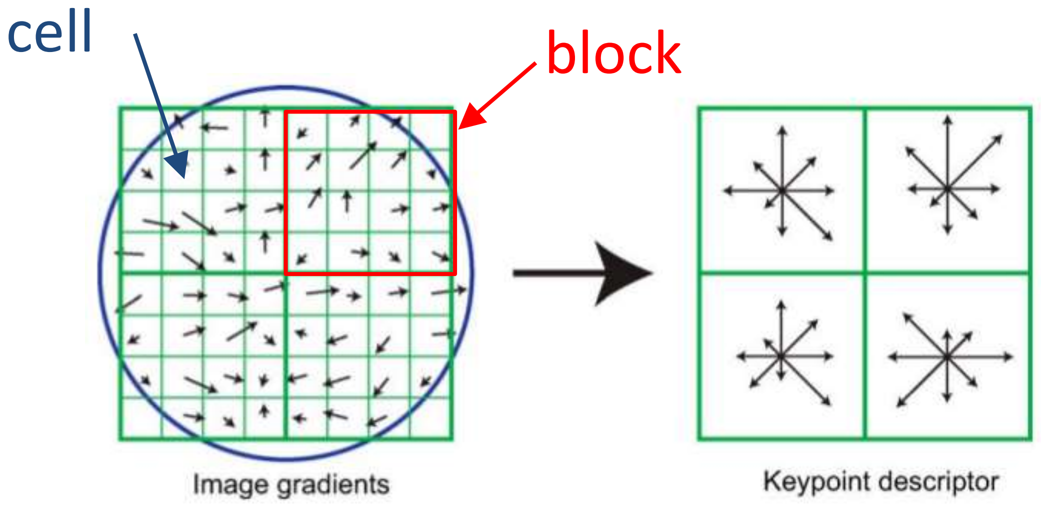
Gradient $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$

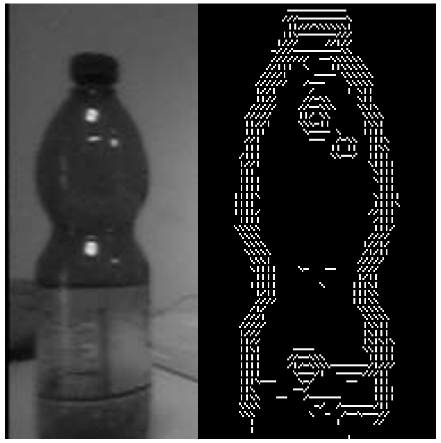
- $f_x = \boxed{-1}\boxed{0}\boxed{+1}$
- $f_y = \begin{array}{|c|}\hline -1 \\ \hline 0 \\ \hline +1 \\ \hline \end{array}$



Histogram and normalisation

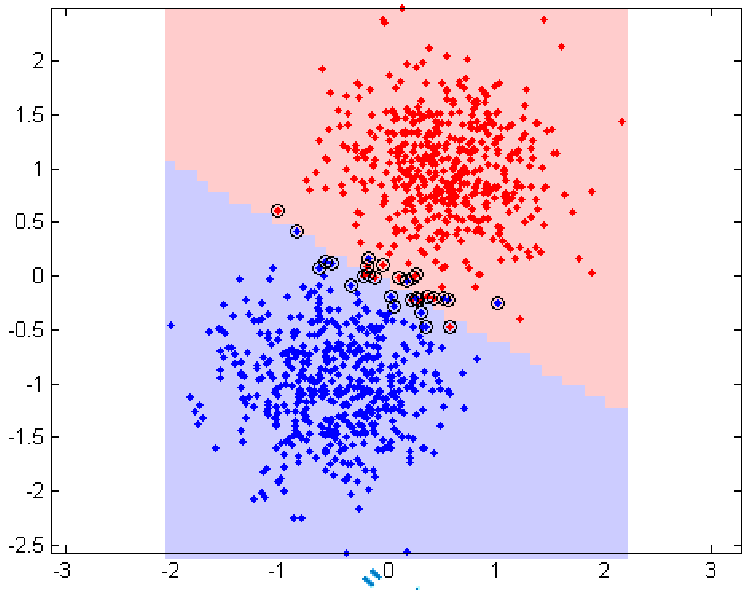
- The matrix of gradient is divided in cells
- Histogram of gradient orientations for each cell...
  - ... weighted by the gradient magnitude
- The adjacent cells are grouped in blocks
- The cell histograms inside are normalised...
  - .. in order to equalise the luminance among close cells





Support vector machine (SVM)

- SVM is supervised learning binary classifier
- Given a set of training examples, an SVM training algorithm builds a model that assigns new examples into one category or the other
- New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on[2]



[2] *Wikipiedia*

FPGA implementations of Histograms of Oriented Gradients for vehicle detection

The Hardware-Software codesign approach

Workshop on Architecture of Smart Camera, Sevilla, June 2013

Luca Maggiani(*), Claudio Salvadori(*), Paolo Pagano(**)

(*) Tecip Institute - Scuola Superiore Sant'Anna - Pisa

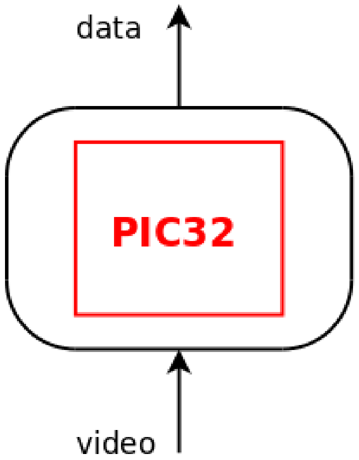
(**) CNIT – National Laboratory of Photonic Networks - Pisa

Why we need an optimized solution?

Code oriented approach

NIOSII CPU, 6-stage pipeline, 50MHz

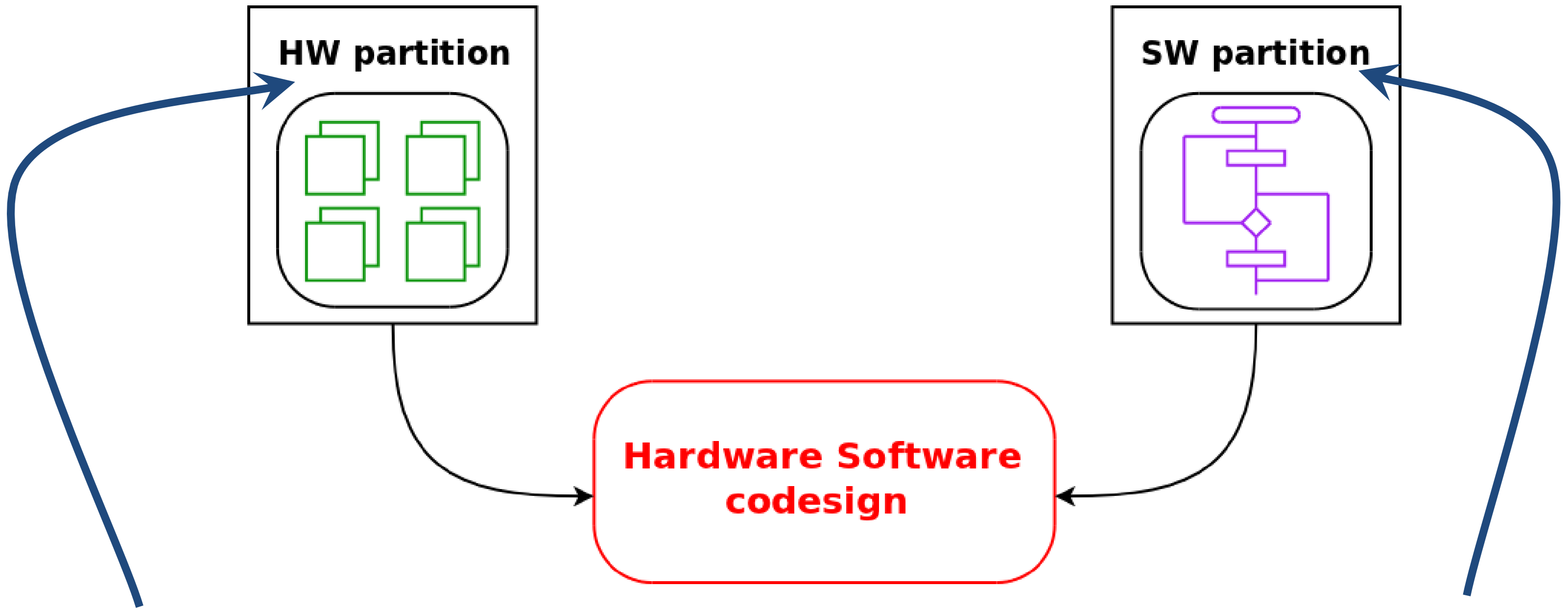
320x240, YUV422, CMOS Camera OV7670 @**15fps**



Platform	Function	nclk * 10^6	T (ms)	fps
SW elaboration NiosII/f	Camera + LCD	3.965	79.3	12.6 over 15
SW elaboration NiosII/f	Edge detector	37.700	754	**1.32 over 15**
SW elaboration NiosII/e	Edge detector	77.900	1550	**0.64 over 15**
SW elaboration NiosII/f	Backgrd Sub.	13.951	279	**3.58 over 15**

Hardware software codesign

Hardware-Software codesign: Hardware and Software joint development technique to exploit both the HDL optimisation and the code flexibility

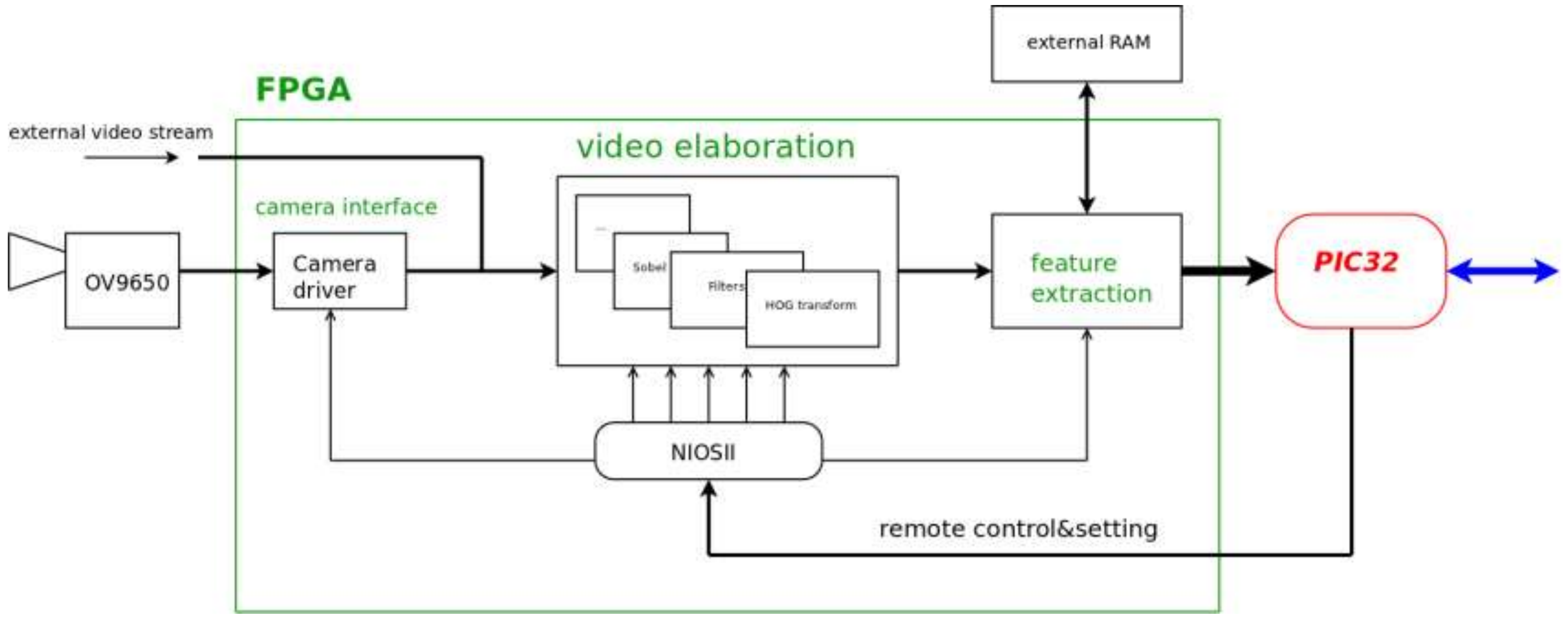


- Optimised solutions
- Dedicated architecture
- Power efficiency

- Code flexibility
- Dynamic configuration
- Sequential operations

HOG pipeline: FPGA implementation

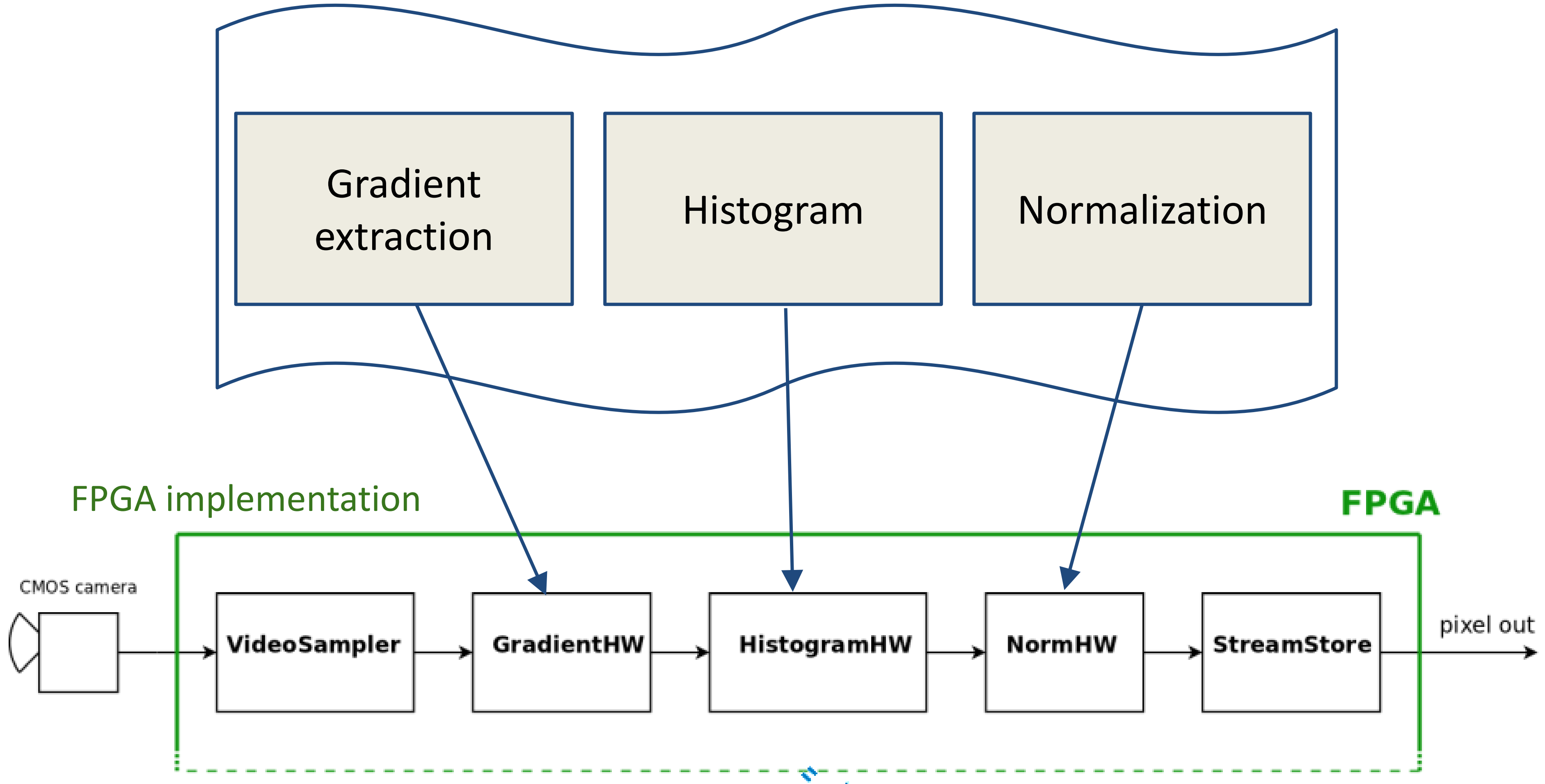
HOG algorithm implementation



Each hardware block is implemented using the **streaming paradigm**:

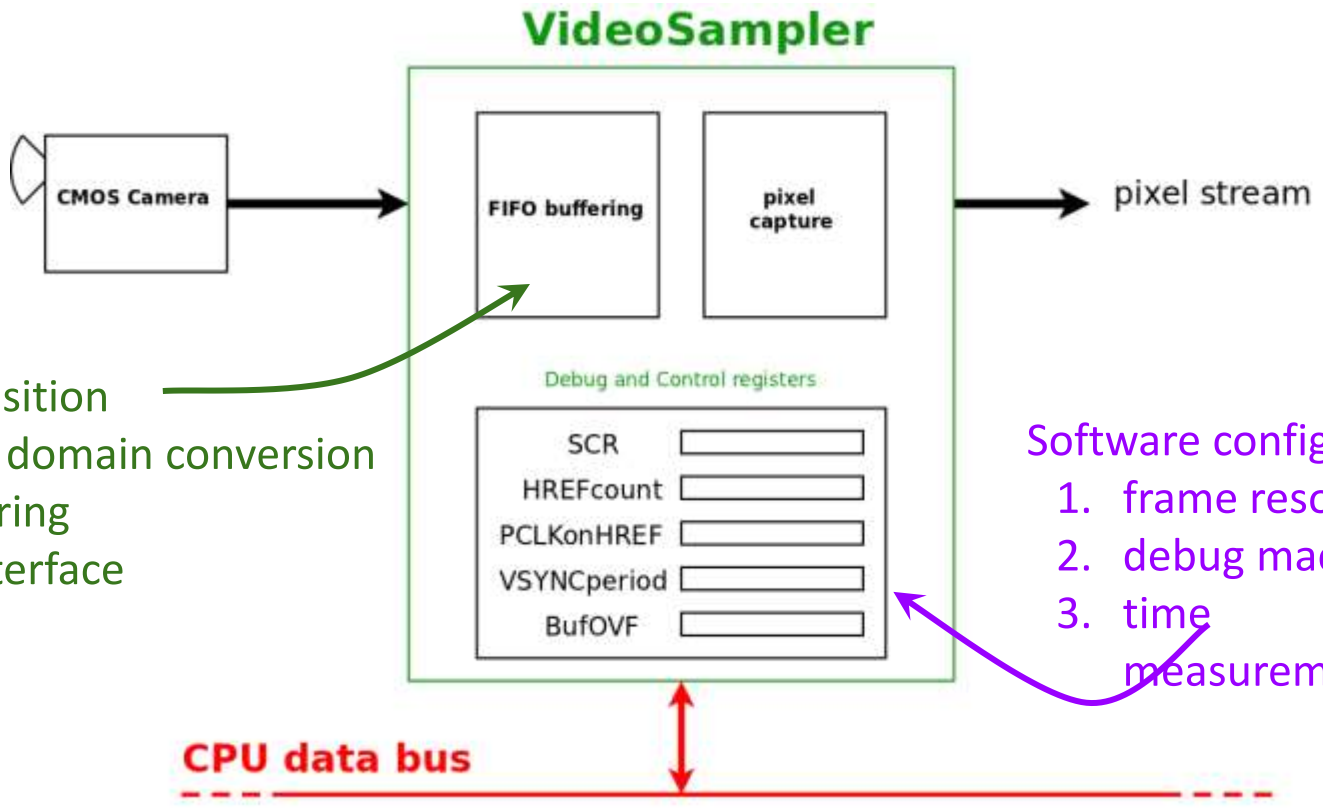
- data are processed when appear in input

FPGA implementation of the HOG pipeline



Video Sampler

Performs the video acquisition from a CMOS Camera



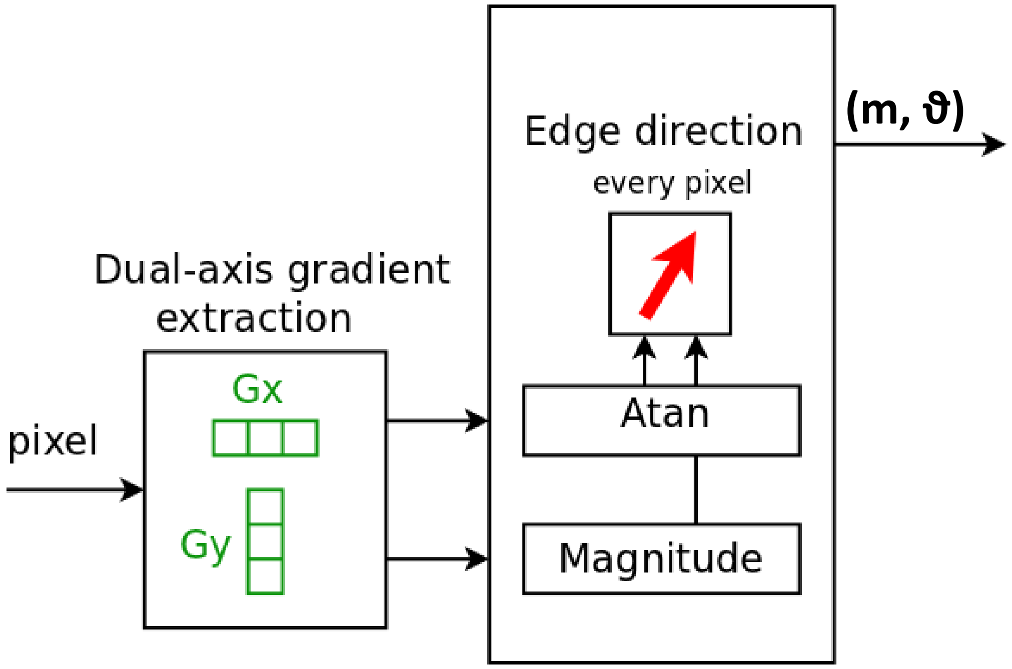
Hardware acquisition

1. Pixel-clock domain conversion
2. Pixel buffering
3. Camera interface

Software configuration

1. frame resolution
2. debug machine
3. time measurement

Gradient extraction



Standard method:

1. Horizontal and vertical kernel
2. Arctangent to compute the slope **ϑ**
3. Euclidean norm to compute the magnitude **m**

To implement on HW both "atan" and "sqrt", the usage of a Look-Up Table (LUT) is needed:

- massive memory access (the processing latency is increased)

Hardware gradient extraction (1/3)

Mathematical point of view:

1. The corner domain is sampled in N bins called ϑ_k
   a. The interval [0, π] is considered, as in the original HOG algorithm
2. The idea is to approximate the complete set of versors using a subset of N elements:

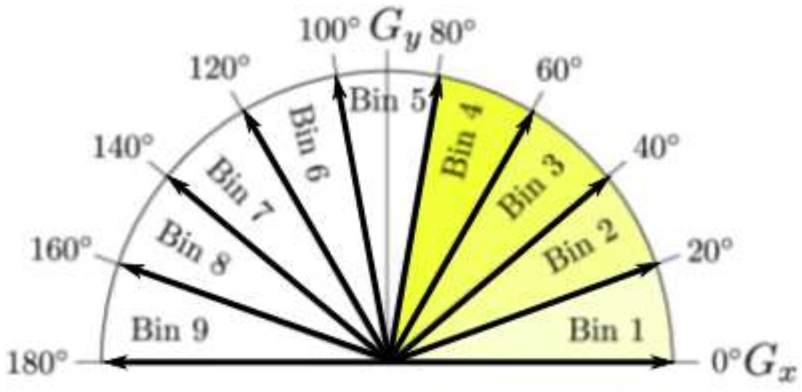
$$\hat{I} = \{\hat{i} = \hat{x}cos\theta_k + \hat{y}sen\theta_k | k = 0 \cdots N-1\}$$

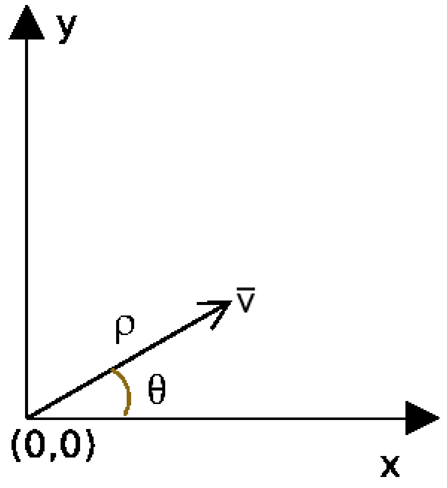
1. ρ and ϑ are computed as:

$$r = arg\{MAX\{\bar{v} \cdot \hat{i}_k | \hat{i}_k \in \hat{I}\}\}$$

$$\theta \simeq \theta_r$$

$$\rho \simeq \bar{v} \cdot \hat{i}_r$$





Hardware gradient extraction (2/3)

Practically:

1. starting from the **spatial gradient definition**...

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$

1. It is possible to compute its component along the previously introduced versor $\hat{i_k}$

$$\frac{\partial I}{\partial \hat{i_k}} = \nabla I \cdot \hat{i_k}$$

1. Then, expliciting the spatial gradient and carrying out the **dot-product**, we obtain the matrix coefficients (**edge detector kernel**)

$$\frac{\partial I}{\partial x} \equiv I(x+1,y) - I(x-1,y)$$
$$\frac{\partial I}{\partial y} \equiv I(x,y+1) - I(x,y-1)$$

$$\boxed{\frac{\partial I}{\partial \hat{i_k}} = cos\theta\Big[I(x+1,y) - I(x-1,y)\Big] + sin\theta\Big[I(x,y+1) - I(x,y-1)\Big]}$$

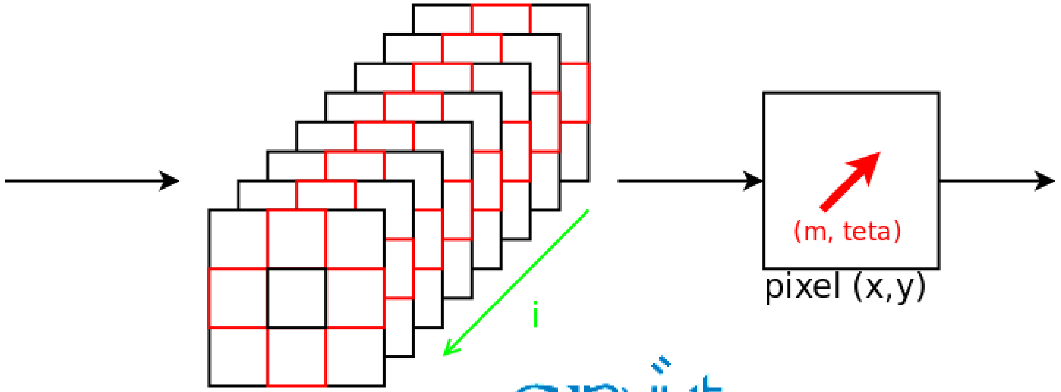
Hardware gradient extraction (3/3)

Edge detector kernel matrix:

every $\hat{i}_k$ vector generates a 3x3 matrix, which is used as a **kernels** above the image

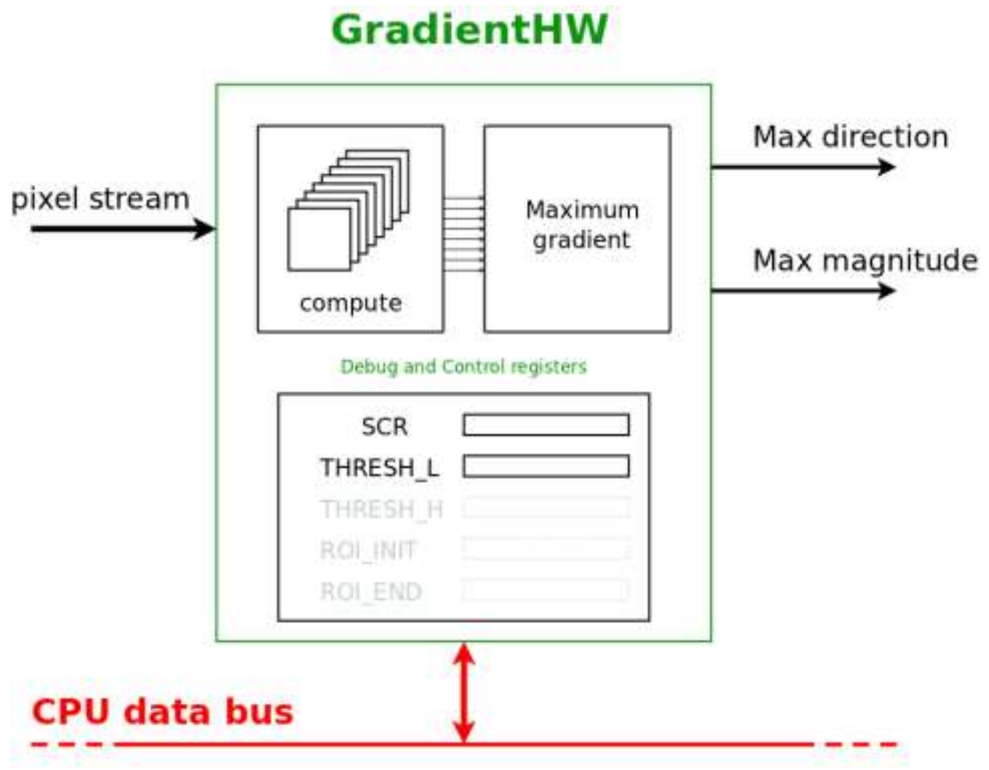
0	sinϑ_k	0
- cosϑ_k	0	cosϑ_k
0	- sinϑ_k	0

Exploiting the **hardware parallelism**, we can process a generic N kernels and obtain a Gradient with a resolut $2\pi/N$



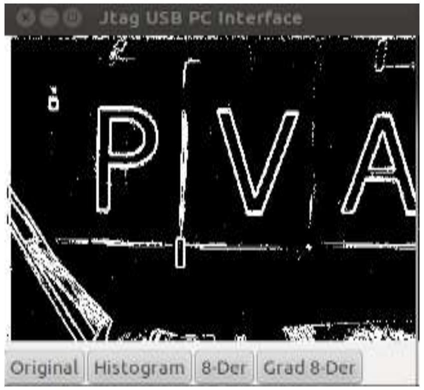
GradientHW

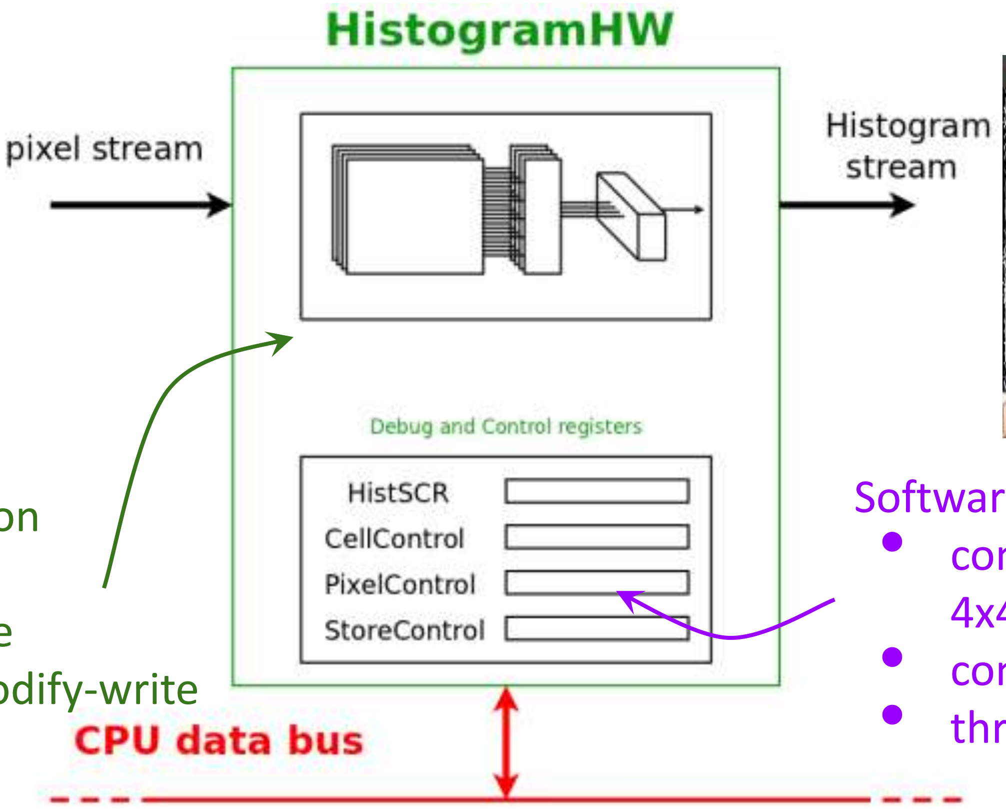
Performs a spatial gradient extraction,
with a **fixed result latency** (2 clock cycle)

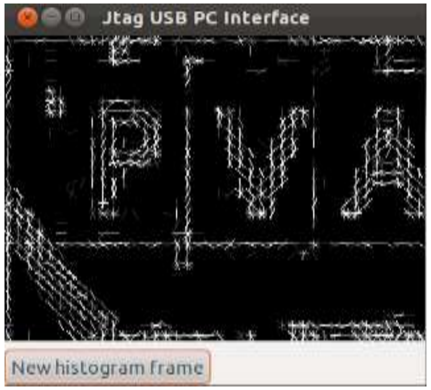


HistogramHW (1/3)

Performs the histogram extraction over a **8x8 pixel** cell from the previously extracted Gradient-frame







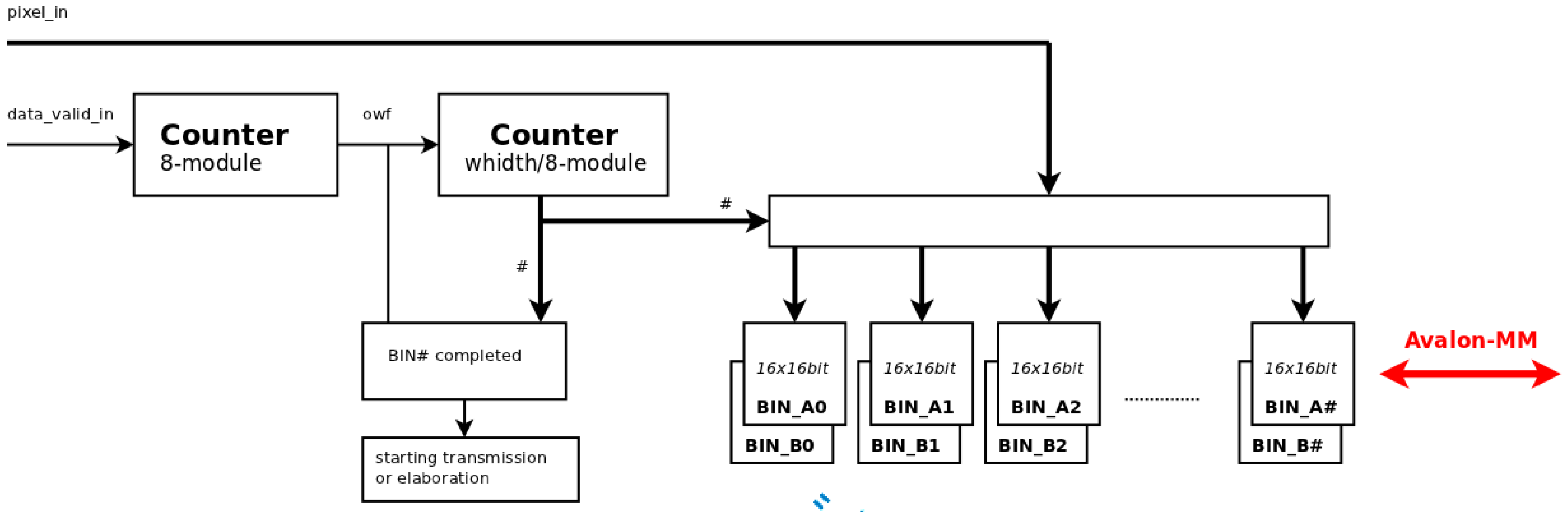
Hardware optimisation
- video stream
- low memory use
- parallel read-modify-write operations

Software flexibility
- configurable cell width 4x4, 5x5, 6x6, 8x8
- configurable bins n°
- threshold

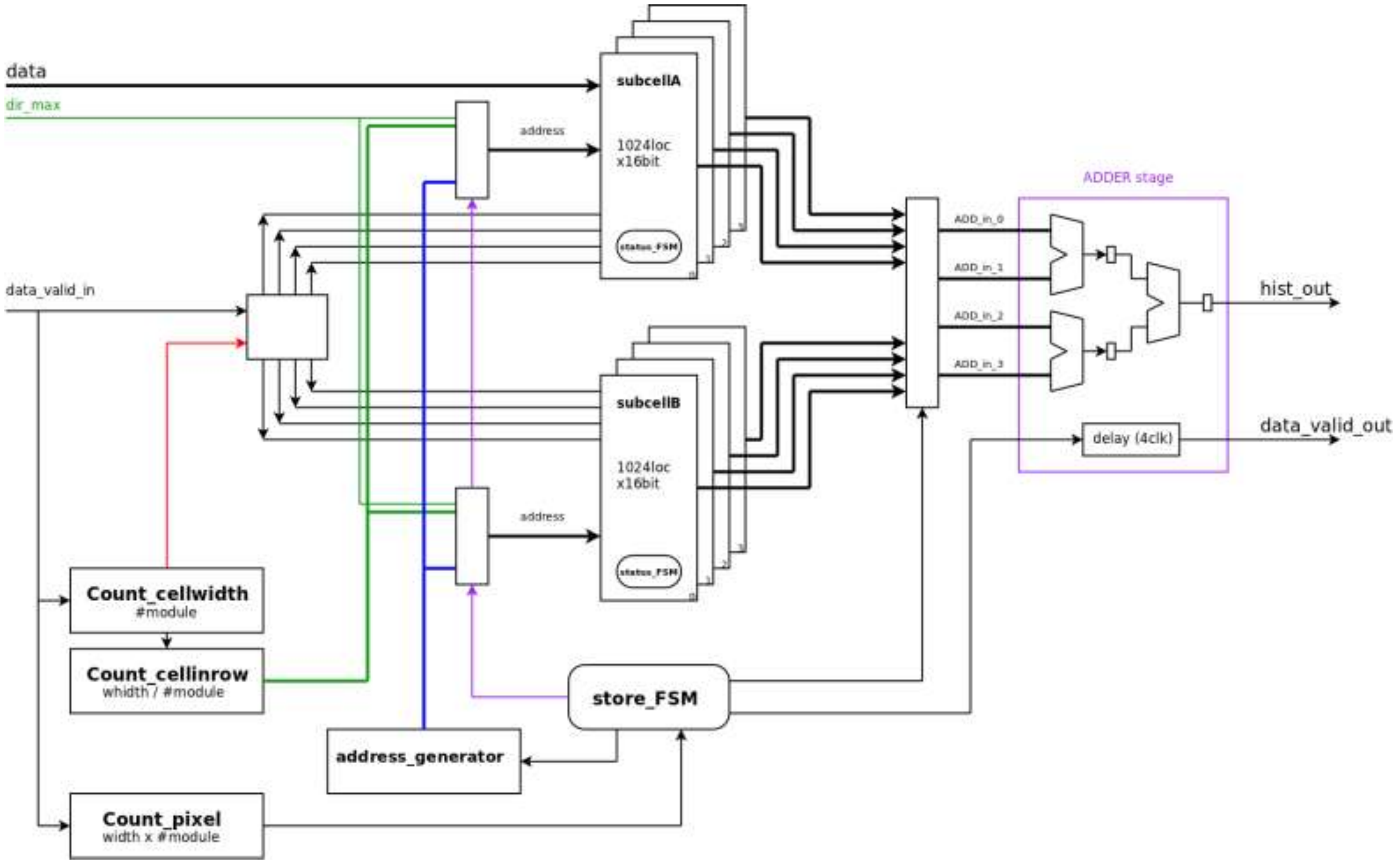
HistogramHW (2/3)

The HistogramHW is developed as a FSM able to manage a continuous stream of pixels

- performs a **single clock** read-modify-write operation (using parallel module)
- **constant data output latency** (depends on both the cell size and the image size)
    - about 2560 clock cycles @ Q-VGA and 8x8 cells

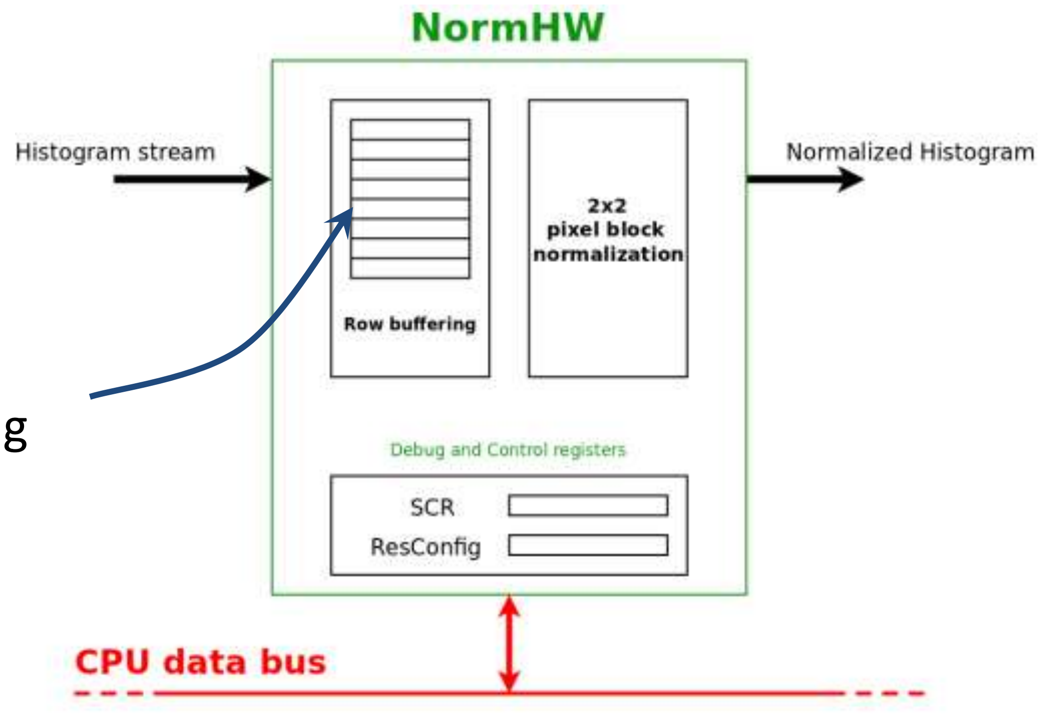


HistogramHW (3/3)



NormHW

Equalisation the luminance among close cells
(composed by a 2x2 cell)



Row cell buffering

Module still in development

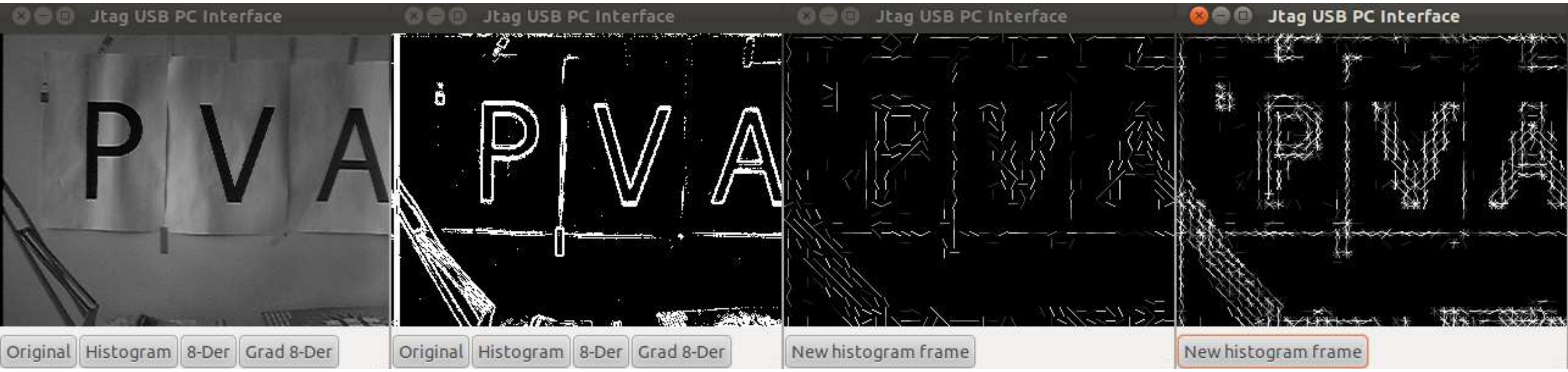
Implementation results

FPGA resource use

VideoSampler	**200 LUT** 512 Byte (FIFO buffering)
GradientHW	**1200 LUT** 960 Byte (Row buffering) 32 DSP module 9x9
HistogramHW	**850 LUT** 16 kByte (Cell buffering)
NormHW	**400 LUT** 2 kByte (Block buffering) *module still in development*

HOG implementation test

- Image captured from an OmniVision CMOS Camera OV9650
- Elaboration performed through the configurable HOG pipeline
- PC-interface use Altera *JtagAtlantic* interface to receive the datastream (USB link)



Processing latency

All the blocks inside the pipeline are implemented using the **streaming paradigm**:

- constant *latency*: the maximum value between all the block latencies
    - Maximum latency = Histogram latency (2560 clock cycles)
- works at the same fps as input
    - the maximum manageable frame rate depends on technological constraints
    - contingent case: 12 fps (~83ms)

Conclusions

- The HOG pipeline is implemented on FPGA using Hardware-Software codesign approach
- A new edge detector kernel is proposed in order to compute directly both the magnitude and the orientation of a vector, exploiting the hardware parallelism capability
- Hardware blocks implemented using the streaming paradigm:
  - edge detector: 2 clock cycles of latency
  - histogram: the latency depends on image size and cell size
    - Contingent case: Q-VGA and 8x8 cells, about 2560 clock cycles
  - total latency: the maximum value between the blocks latencies
    - Contingent case: histogram latency, about 2560 clock cycles

FPGA implementations of Histograms of Oriented Gradients for vehicle detection

Domain Specific Language Implementation

C. Bourrasset, J. Sérot and F. Berry

Institut Pascal- D.R.E.A.M - Aubière, France

Domain Specific Language approach

- High level abstraction programming
- Rapid prototyping
- FPGA programming becomes easy !

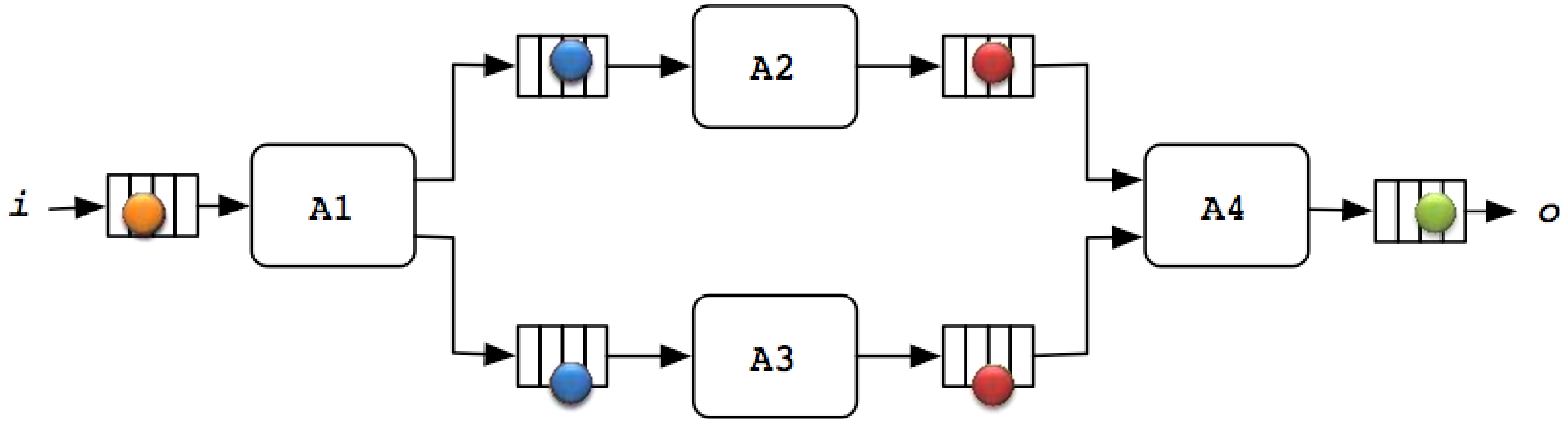
- HDL generated code is less efficiently than handwritten code

The CAPH programming language

- Based upon the dataflow model of computation
  - offers an bridging MoC between high-level, programmers oriented specification and low-level, HDL-based descriptions
- Well suited to stream-processing applications (operating *on-the-fly* on continuous streams of data coming directly from sensors)

Dataflow model

- An application = a collection of computing units (*actors)* exchanging tokens through unidirectional, *buffered* links (*FIFOs*)
- For each actor, a set of firing rules specifies when it consumes input tokens and produces output tokens



CAPH language

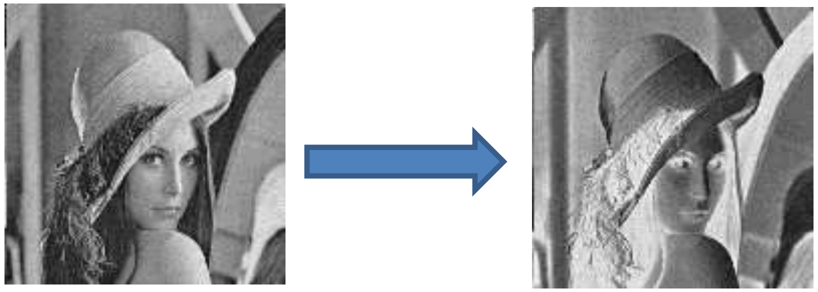
- Dataflow model need **actors** and **connection** between actors :
  - One sub-language for actor behavior description
  - One sub-language for network description

Actor description

```
type pixel= unsigned<8> ;

actor inv ()
  in (a: pixel dc)
  out (c: pixel dc )
rules
| a: '<  -> c: '<
| a: 'p  -> c: '255-p
| a: '>  -> c: '>
;
```

Network description

```
stream i: pixel dc from  ‘’dev:cam0 ’’;
stream o: pixel dc to ‘’dev:mon1’;

net o = inv i;
```



CAPH tokens

- A token in CAPH may represent
  - Datas (integers, boolean, floats)
  - Control signals ‘ < ‘, ‘ > ’
- Example image

Gray level representation

103	45	50	…
60	34	24	…
150	42	210	…
…	…	…	…

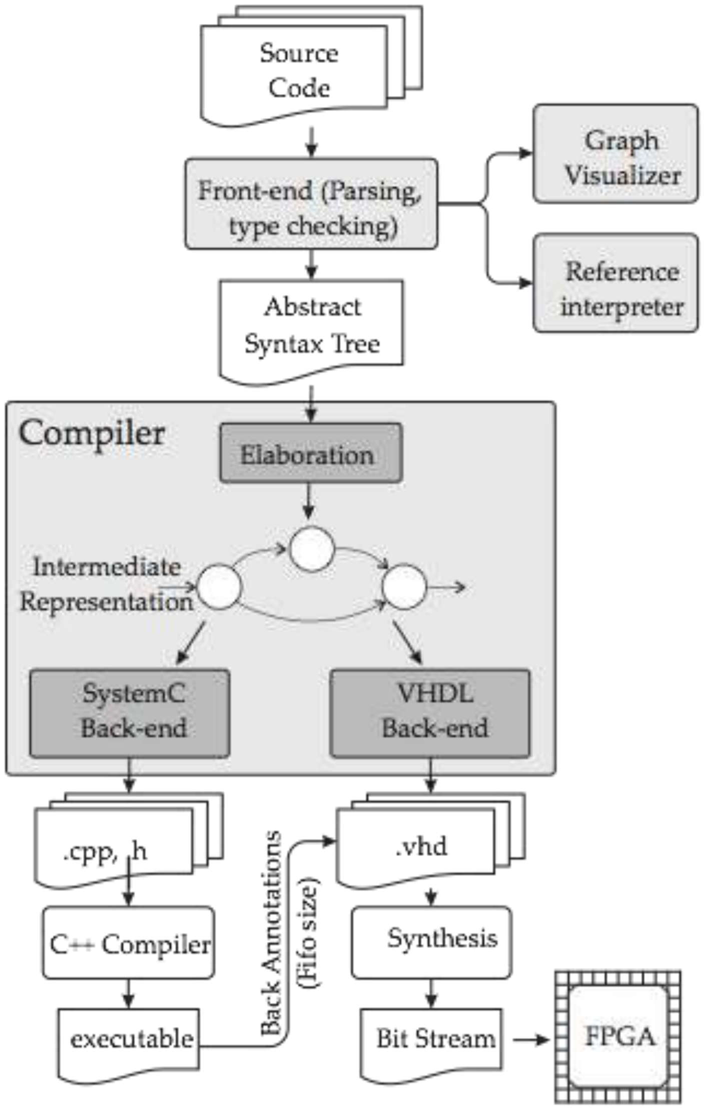
- CAPH tokens transcription

< < 103 45 50 … > < 60 34 32 … > < 150 40 210 .. > < … > >

Start of Frame Start of Line End of Line End of Frame

Caph toolset

- **Graph visualizer** : .dot format
- **Reference interpreter** :
  - based on the fully formalized semantics
  - tracing, profiling and debugging
- **Compiler** :
  - elaboration of a target-independant IR
  - specialized backends (SystemC, VHDL)



HOG

Window size 320 x 240

$$\begin{tikzcd}
\boxed{\text{Detection window}} \arrow[r, "I"'] & \boxed{\text{Gradient / Orientation}} \arrow[r, "|G|"'] & \boxed{\text{Histogram Generation}} \arrow[r] & \boxed{\text{Normalization}} \arrow[r] & \boxed{\text{HOG Descriptor}}
\end{tikzcd}$$

- Gradient computation

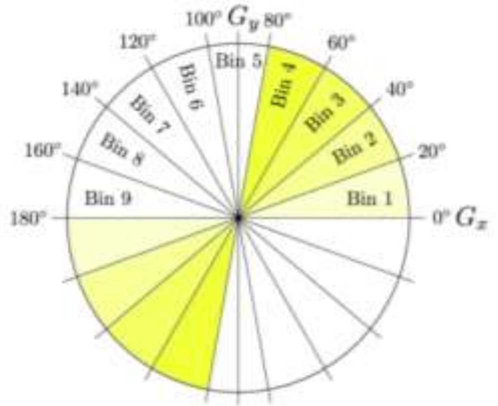
$$|G| = |G_x| + |G_y|$$

$$G_x = M_x * I \qquad M_x = [1 \quad 0 \quad 1]$$

$$G_y = M_y * I \qquad M_y = [1 \quad 0 \quad 1]^T$$

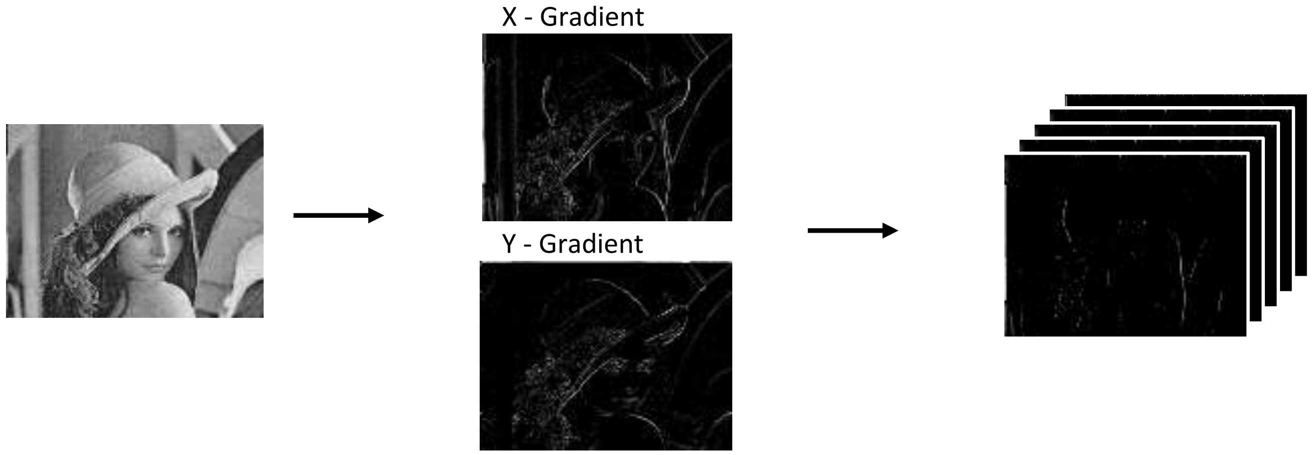
- Gradient orientation

$$G = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

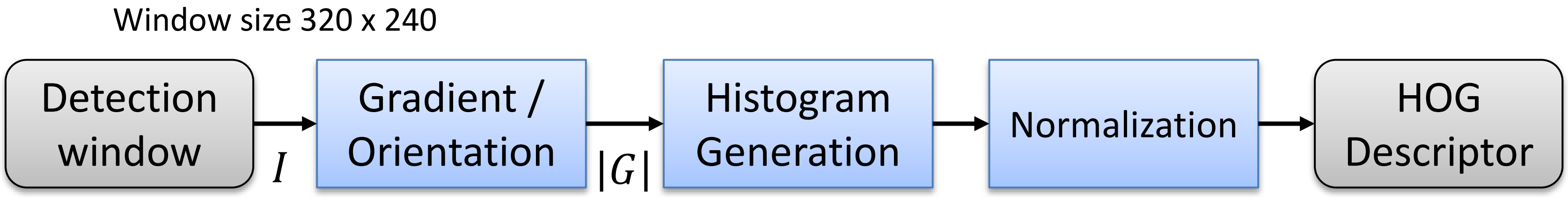


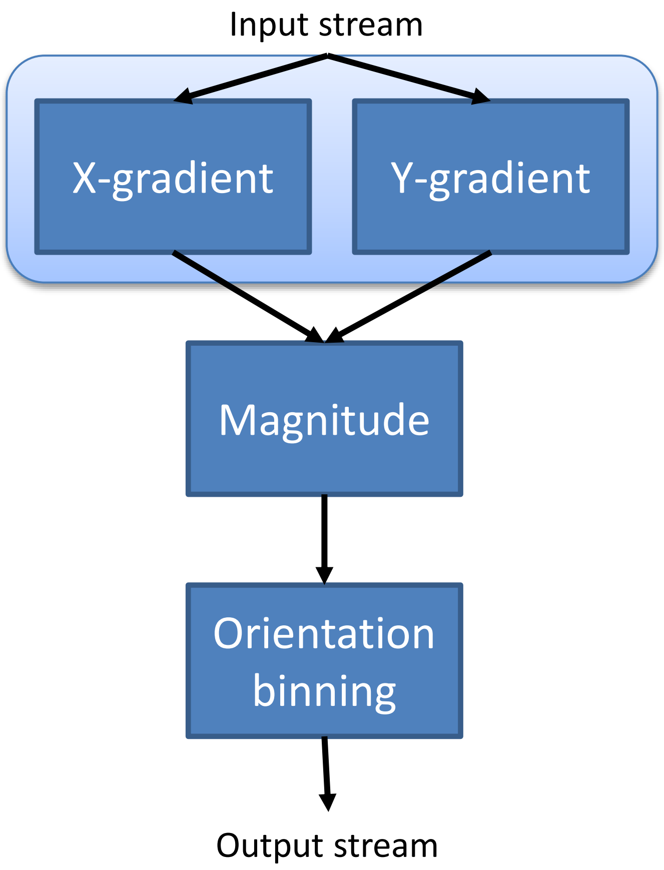
- Discretization into 9 spaced angular bin over 0°-180°

- Generate 9 magnitude-weighted bin images



HOG





Source Code

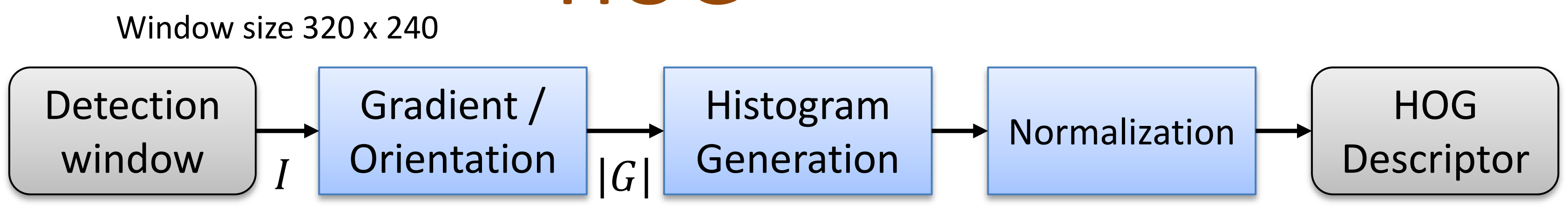
```
actor maddn(k:signed<9> array[3])
 in (a:unsigned<8> dc,b:unsigned<8> dc, c:unsigned<8> dc)
 out (o:signed<9>dc)
rules
 ( a,b,c) -> o
 |('<,'<,'<) -> '<
 |('zzp,'zp,'p) -> '( zzp*k[0] + zp*k[1] + p*k[2])
 |('>,'>,'>) -> '>;
net dx = maddn [[1,0,(-1)]] (neigh2u13 i);
net dy = maddn [[1,0,(-1)]] (neigh2u31r i);

net neigh2u13 x =
  let y2 = d1p x in
  let y3 = d1p y2 in (x, y2, y3);
```

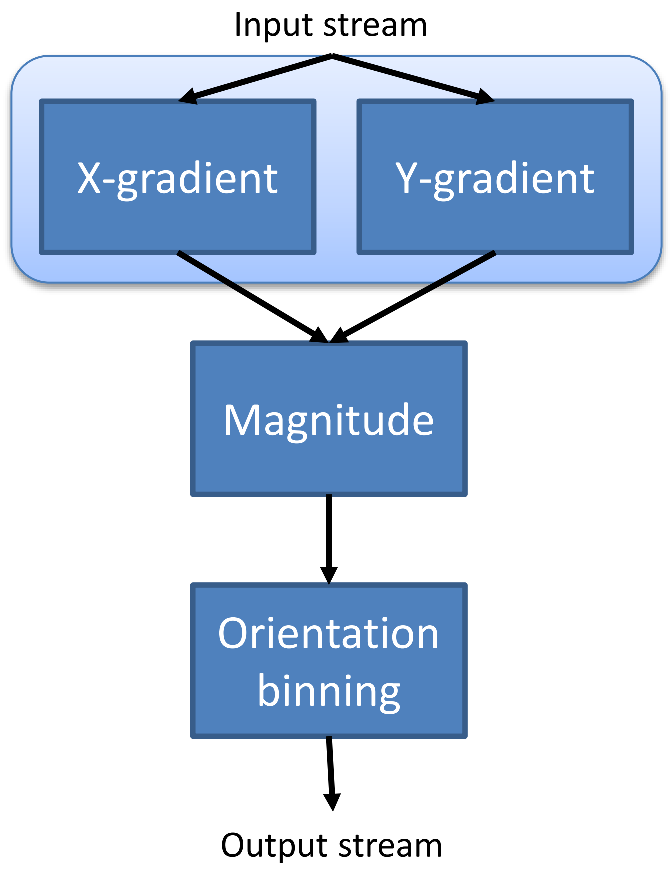
- d1p actor delays one pixel to the right

Example:

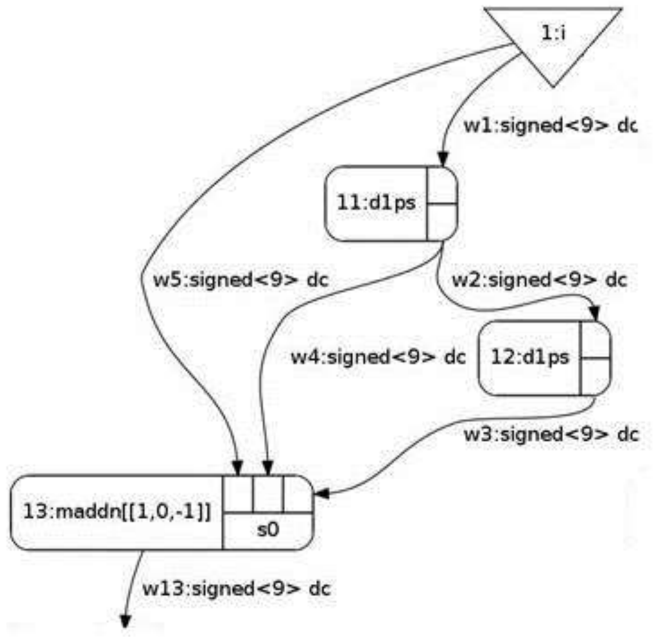
$$< 1\ 2\ 3\ 4 > \xrightarrow{d1p} < 0\ 1\ 2\ 3 >$$

HOG

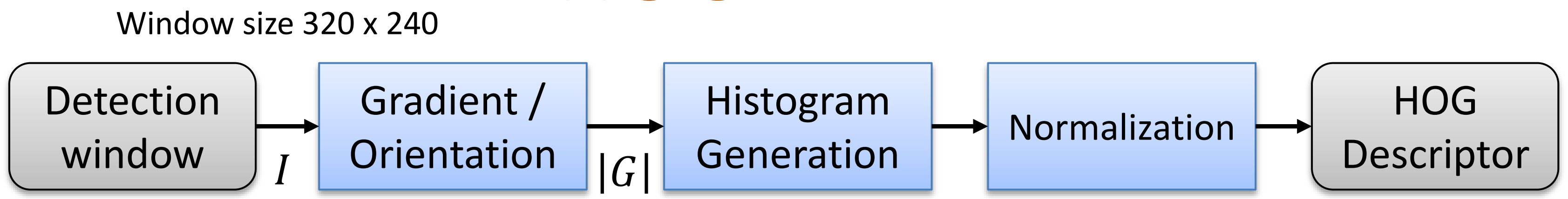


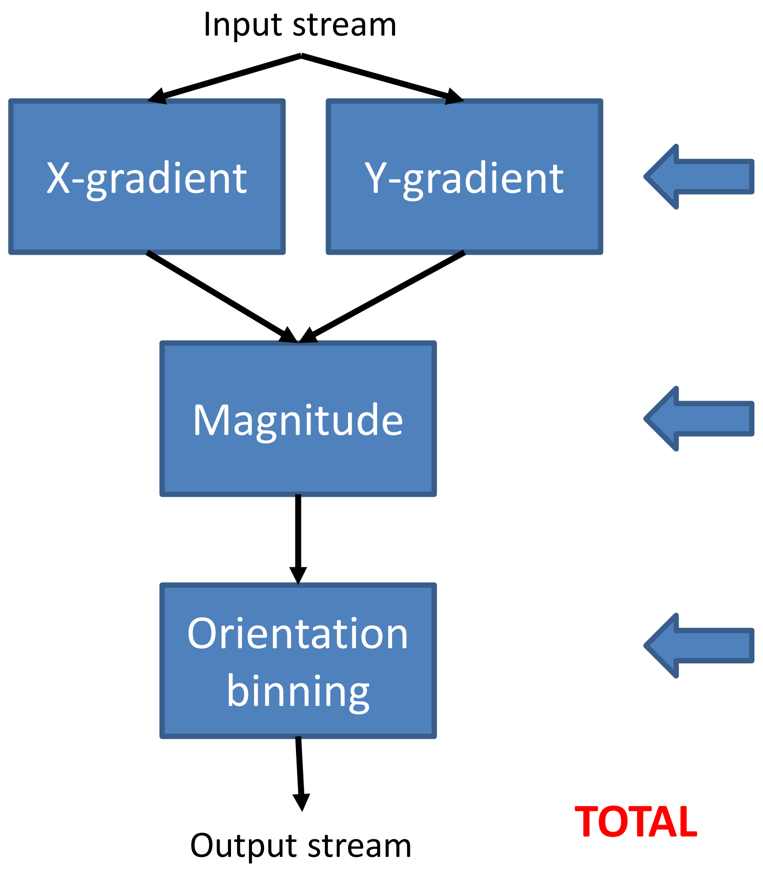


- Network of actors for x gradient



HOG

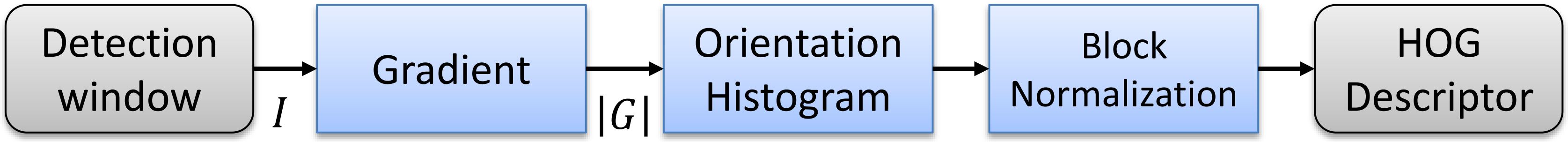




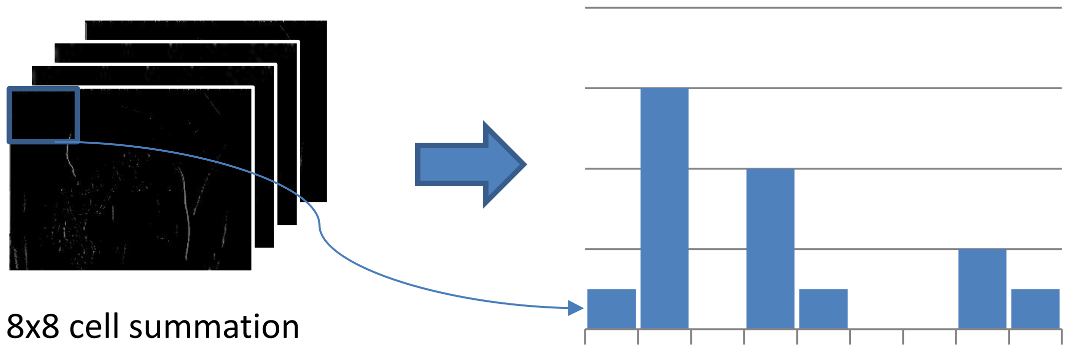
- Hardware Resources (Altera Cyclone III FPGA)

	939 LUT	5778 bits RAM
	101 LUT	0 bits RAM
	448 LUT	0 bits RAM
TOTAL	**1468 LUT**	**5778 bits RAM**

HOG



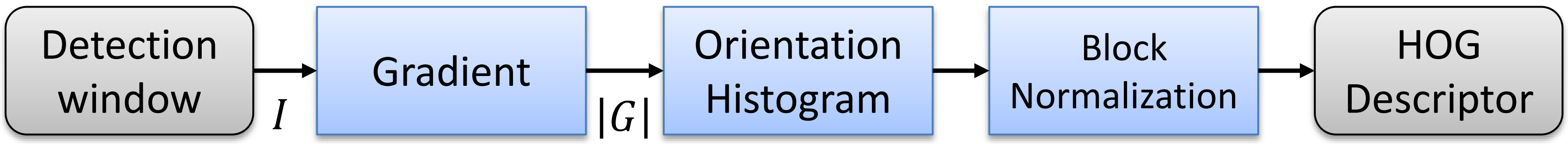
- Histogram extraction over a 8x8 pixel block (cell) from gradient frame



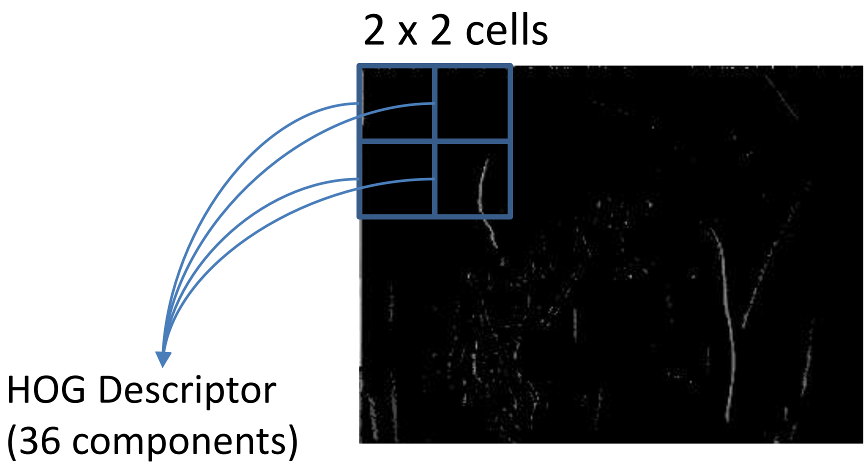
- 9 parallel streams computed by 9 actors in parallel
- Need to store 8 rows to compute histogram extraction (using FIFOs)

Hardware resources for histogram extraction
9500 LUT 7,8 kbits RAM

HOG



- Local contrast normalization
- Reduce the luminance variation
- L1-norm, $\boldsymbol{v} \rightarrow \boldsymbol{v}/(\|\boldsymbol{v}\|_1 + \epsilon)$



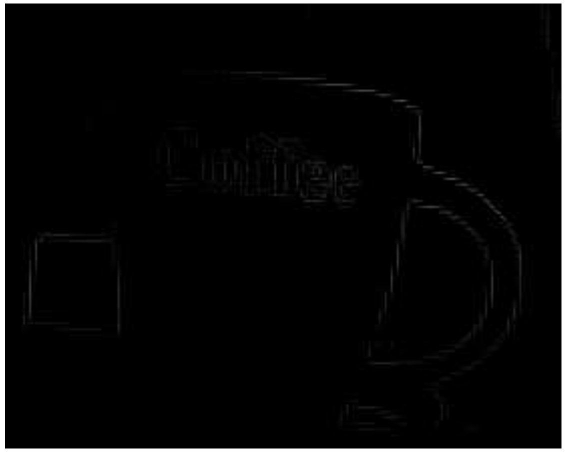
Hardware resources for normalization

900 LUT 2 kbits RAM

FPGA resource summary

Gradient	1468 LUT	5,7 Kbits
Histogram	9500 LUT	7,8 Kbits
Normalization	900 LUT	2 Kbits

- Histogram step can be optimized
- Generated HDL code synthetize LUT locks to memory job



CAPH Project

- Continuous development since 2009 (Major release soon)
- Substantial increase in abstraction level compared to classical HDLs
  - large gain in development times (x5-x10)
  - .... without significant performance penalty (< 30%)
- Several realistic applications implemented
  - motion detection, MPEG encoding, connected component labeling, Harris-Stephen POI detection, ...
- Toolset and manual freely available at

http://dream.univ-bpclermont.fr/index.php/softmenu/caph

Conclusion

	Co-design approach	DSL approach
VideoSampler	**200 LUT** 512 Byte (FIFO buffering)	
GradientHW	**1200 LUT** 960 Byte (Row buffering) 32 DSP module 9x9	**1468 LUT** 5.7 kbit
HistogramHW	**850 LUT** 16 kByte (Cell buffering)	**9500 LUT** 7,8 kByte
NormHW	**400 LUT** 2 kByte (Block buffering) *module still in development*	**600 LUT** 2 kByte
		210 Lines of Code

Thank you