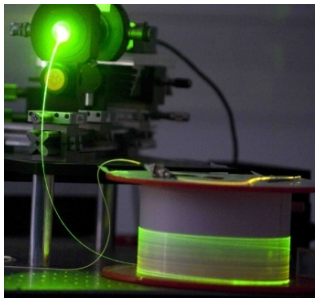
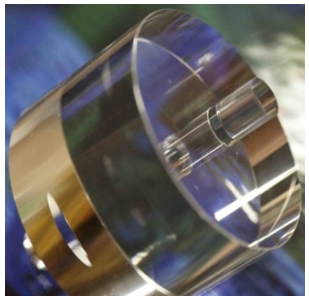
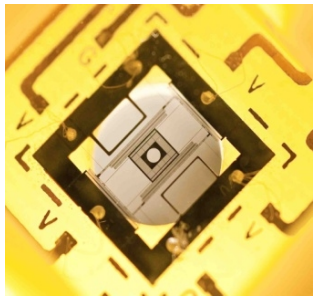
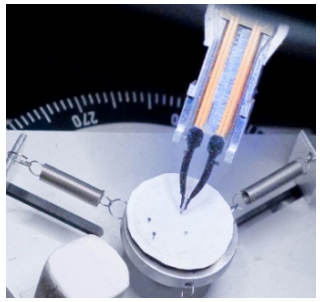
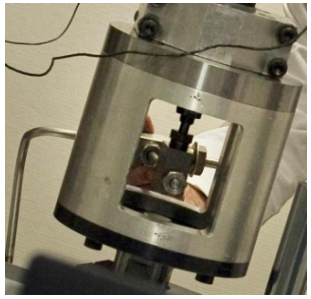
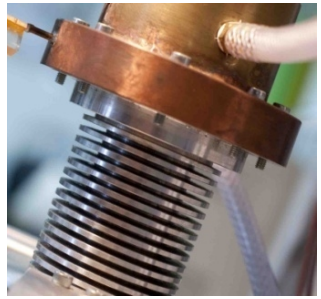
INSTITUT DE RECHERCHE
FEMTO-ST

femto-st
sciences & technologies

A new approach based on a least square method for real-time estimation of cantilever array deflections with an FPGA

R. Couturier, S. Domas, G. Goavec, M. Lenczner, M. Favre, A. Meister

FEMTO-ST DISC FEMTO-ST Time-Frequency CSEM

dMEMS 2012, April 2nd

Interreg FRANCE-SUISSE

UFC UNIVERSITÉ DE FRANCHE-COMTÉ cnrs ensmm utbm

Outline

Architecture and goals

- main goals,
- experimental setup,
- interferometry for deflection estimation.

Our solution

- two methods (splines, least squares) used for computations,
- why FPGA ?
- VHDL implementation and tests.

Main goals

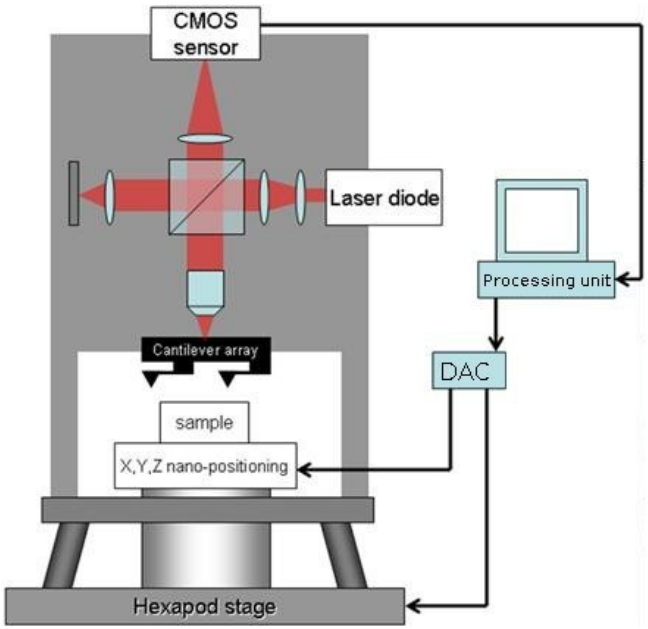
"Simplicity"

- no sensors (e.g. piezoresistors) on cantilevers,
- no complex acquisition hardware,
- easy setup, calibration and acquisition.

Performance and quality

- large arrays : up to 100 cantilevers,
- fast acquisition : 1000 Hz,
- smallest computation time to process a single acquisition to do an efficient control,
- errors on results close to 0.3nm.

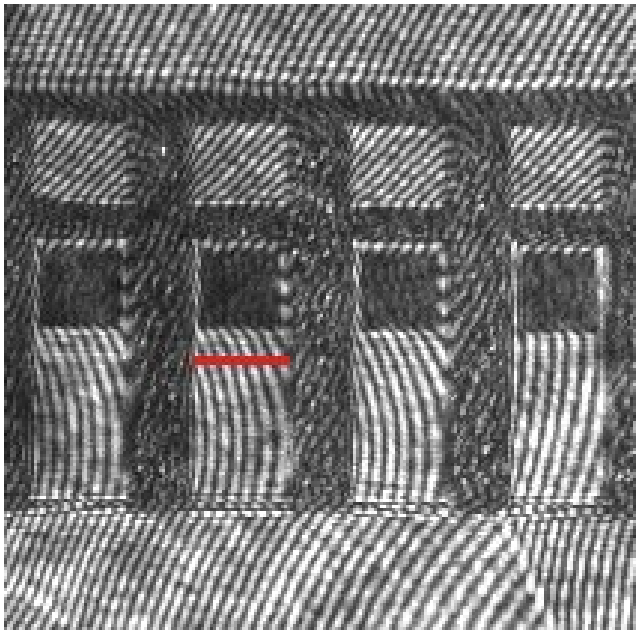
Experimental setup



Elements

- cantilever array,
- Linnick interferometer,
- sample placed on an hexapod stage,
- CMOS high-speed camera,
- "light" processing unit (laptop + embedded board)

Interferometry for deflection estimation (1/2)



For a given profile (= segment) perpendicular to levers

- the light intensity can be modeled by

$$I(x) = A\ \cos(2\pi f x + \theta) + ax + b$$

- frequency f of fringes is constant,
- phase θ of fringes varies within $[-\pi, \pi]$ when the lever moves.

$\Rightarrow$ computing the phase at the tip of levers can give their deflection.

Interferometry for deflection estimation (2/2)

Problems

- phase of the tip profile is cyclic into $[-\pi, \pi]$
  $\rightarrow$ same value corresponds to different positions.
- displacements produce vibrations and distortions.
- image taken by the camera is a discretized version of the physical domain.

Constraints

- tip phase must be "unwrapped",
- a reference profile on the base must be used to estimate and reduce/suppress the noise,
- modeling coupling between levers deflections and base distortions,
- using methods that approximate at best the "real" signal.

Frequency & phase computation (1/5)

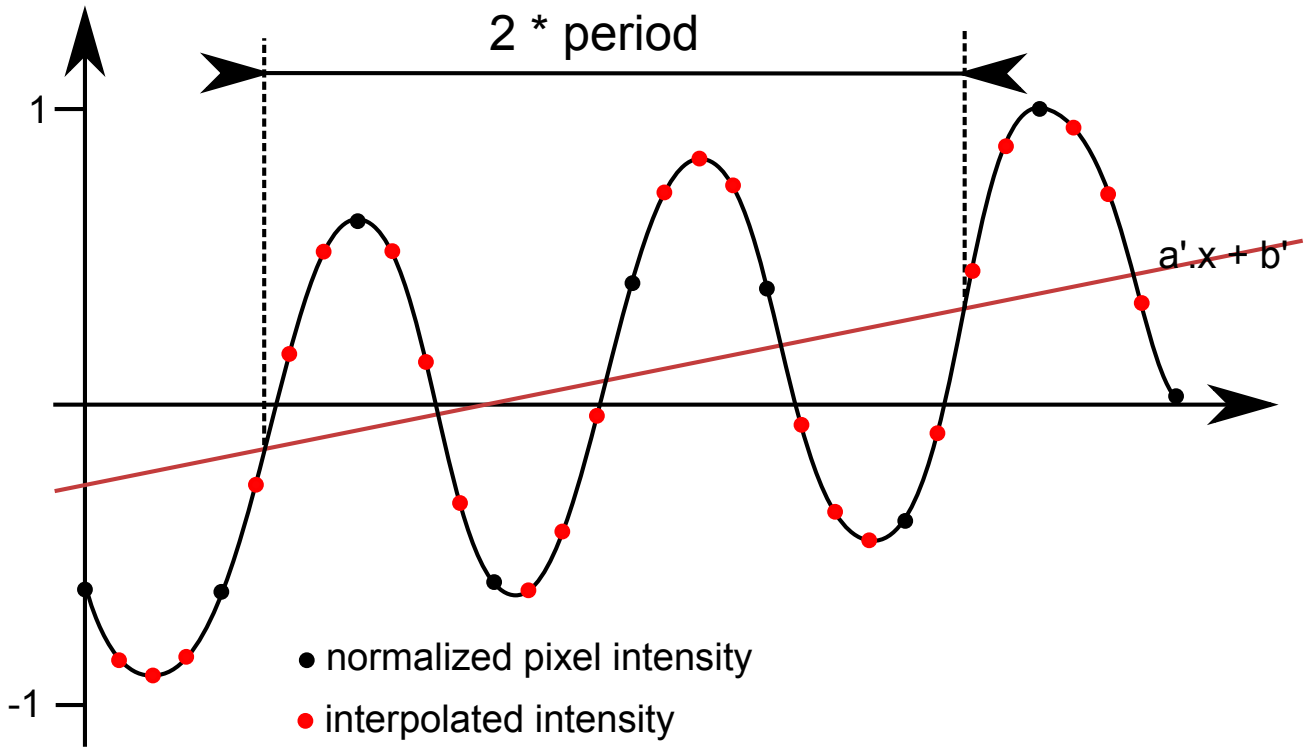
For each profile *p*

- light intensity : $I^p(x) = A\ \cos(2\pi f^p x + \theta^p) + a^p x + b^p$
- corresponds to a series of M pixels, extracted from an image,
- pixels are equally spaced, centered in position $i \in 0, 1, \ldots, M-1$
- pixel in i has a gray level, which is $I^p(i)$.

Frequency & phase computation (2/5)

Frequency computation

1. $I^p_{norm} \leftarrow$ pre-normalize $I^p(i)$ values into $[-1, 1]$,
2. $I^p_s \leftarrow$ **splines interpolation** of I^p_{norm},
3. evaluate I^p_s in points $j \in 0, \frac{1}{k}, \frac{2}{k}, \ldots, M-1$ (e.g. k = 4)
4. find intersections of I^p_s with the mean line $a'.x + b'$,
5. deduce frequency from intersections.



Frequency & phase computation (3/5)

Phase computation

- profile p has constant frequency f^p, i.e. period $T^p = \frac{1}{f^p}$,
- original algorithm : splines (SPL),
- novel method : least squares (LSQ).

SPL

1. $I^p_{norm} \leftarrow$ pre-normalize $I^p(i)$ values onto $[-1, 1]$,
2. $I^p_s \leftarrow$ **splines interpolation** of I^p_{norm},
3. evaluate I^p_s in points $j \in 0, \frac{1}{k}, \frac{2}{k}, \ldots, M-1$ (e.g. k = 4),
4. nb. of periods : $n_T = \left\lfloor \frac{M-1}{T^p} \right\rfloor$, $L = \lfloor k \times T^p \times n_T \rfloor$
5. $$\theta = arctan \left[\frac{\sum_{j=0}^{L-1} \sin(2\pi f^p . j) \times I^p_s(j)}{\sum_{j=0}^{L-1} \cos(2\pi f^p . j) \times I^p_s(j)} \right]$$

Frequency & phase computation (4/5)

LSQ

1. find mean line $a.i + b$ of $I^p(i)$,
2. "flatten" I^p : $I^p_{corr} \leftarrow I^p(i) - a.i - b$,
3. A^p supposed to be close to $\frac{\max(I^p_{corr}) - \min(I^p_{corr})(i)}{2}$
4. searching : $\min_{\theta \in [-\pi,\pi]} \sum_{i=0}^{M-1} \left[\cos(2\pi f^p.i + \theta) - \frac{I^p_{corr}(i)}{A^p}\right]^2$

Equivalent of searching θ^*, for which :

$$2\left[\cos\theta^* \sum_{i=0}^{M-1} I^p_{corr}(i).\sin(2\pi f^p.i) + \sin\theta^* \sum_{i=0}^{M-1} I^p_{corr}(i).\cos(2\pi f^p.i)\right] -$$

$$A\left[\cos 2\theta^* \sum_{i=0}^{M-1} \sin(4\pi f^p.i) + \sin 2\theta^* \sum_{i=0}^{M-1} \cos(4\pi f^p.i)\right] = 0$$

Frequency & phase computation (5/5)

LSQ & SPL comparison

- profiles generated with varying periods, phases, slopes,
- random perturbations (= noise) on each pair of pixels,
- error expressed as : $err = 100 \times \frac{|\theta_{real} - \theta_{comp}|}{2\pi}$.

noise (N)	SPL		LSQ	
	max. err.	aver. err.	max. err.	aver. err.
0	2.46	0.58	0.49	0.1
5	3.77	0.72	2.47	0.41
10	5.62	1.03	4.29	0.81
15	7.96	1.38	6.35	1.21
30	17.06	2.6	13.94	2.45

TABLE: Error (in %) for generated profiles, with noise.

Quality

- in our experimentations : 1 rad. $\approx$ 50nm,
- error of 1% $\Rightarrow$ error of 0.5nm : very close to our goals.

Why FPGA ? (1/3)

Evaluating constraints

- 100 levers $\approx$ 200 profiles,
- 1000 Hz $\rightarrow \frac{1000}{200} = 5\mu s$ to compute a phase,

Benchmark

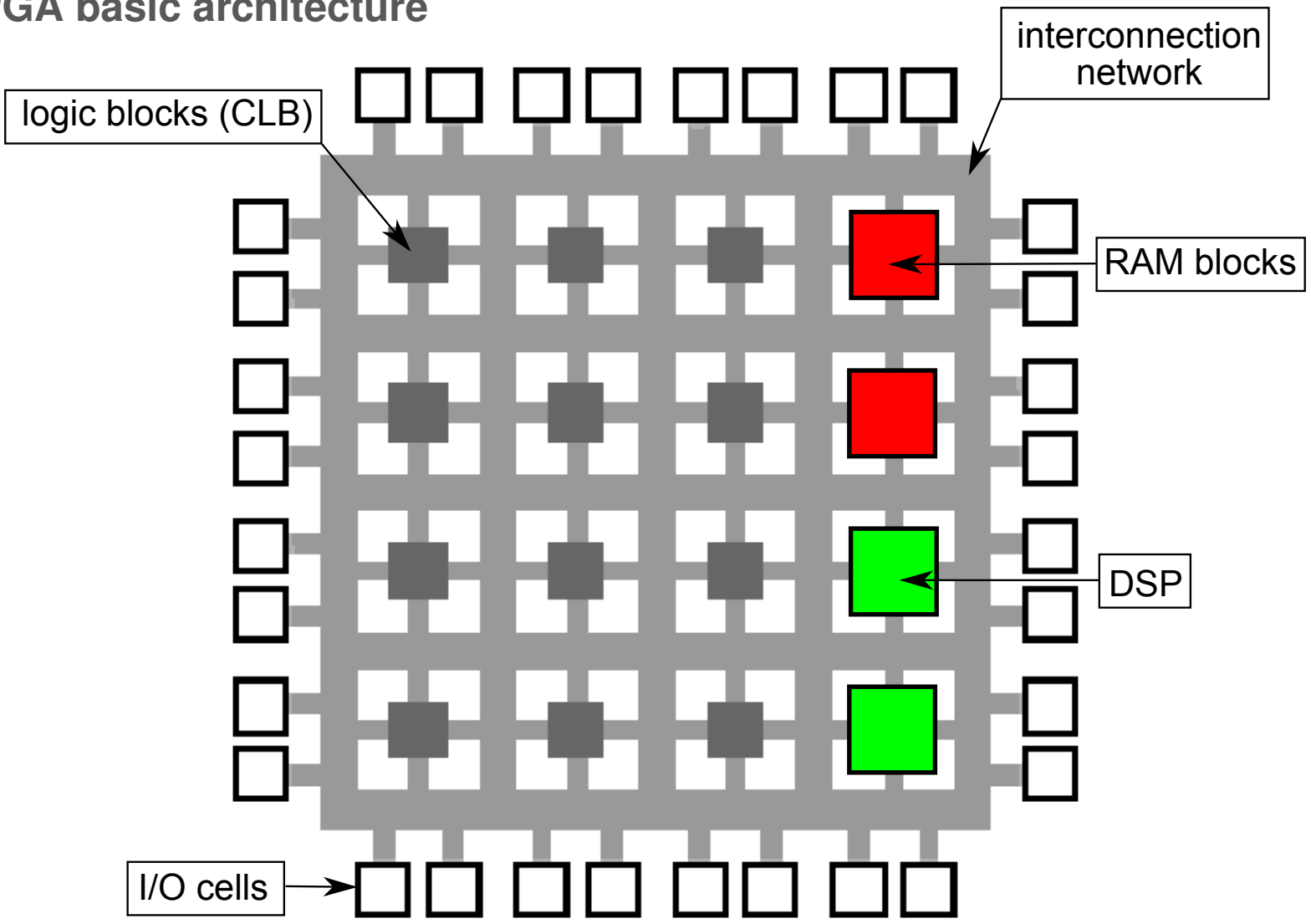
- implementation in C (with -O3 compilation),
- file of 1MB containing 200 profiles of 20 pixels,
- execution on Intel Core 2 Duo at 2.33GHz, under Linux,
  - $\rightarrow$ SPL : 7.7 μs per phase, too much.
  - $\rightarrow$ LSQ : 2.2 μs per phase, should be OK.

Drawbacks

- works on increasing resolution (profile of 50-60 pixels) : not OK.
- Linux OS = unpredictable latency to retrieve image and to compute deflections : no real-time control.
  - $\Rightarrow$ solution : computing unit on FPGA directly linked to the camera

Why FPGA ? (2/3)

FPGA basic architecture



Why FPGA ? (3/3)

FPGA drawbacks

- limited number of I/Os,
- dedicated units (RAMs, DSP, ...) limited in size,
- "low" clock rate (100-400 MHz),
- very low level of programmation (i.e. bit level)
  - $\rightarrow$ no native double precision operations,
  - $\rightarrow$ no native "complex" operations (cos, sin, divisions, ...).

Advantages

- very efficient with pipelined computations,
- "natural" parallelism,
- I/Os strictly cadenced $\rightarrow$ implicit "real-time".

Implementation in VHDL (1/3)

Problems

- only integers operations,

  $\rightarrow$ choosing a sufficient quantizations of double precisions values.

- only +, -, *,

  $\rightarrow$ transform divisions into multiplications (when possible),

  $\rightarrow$ pre-compute complex functions (cos, sin, ...) and store in look-up tables (LUT).

- slicing computation in very small tasks,
- tasks using dedicated units may take more than 1 clock cycle,
- VHDL language (i.e. mixing concurrent and sequential instructions).

Implementation in VHDL (2/3)

Noticeable points for LSQ

- for a profile of size M, $a.x + b$ can be found using only +, -, and *,
- since f^p is also known, we can pre-compute :

  $\sum_{i=0}^{M-1} \sin(4\pi f^p.i)$, $\sum_{i=0}^{M-1} \cos(4\pi f^p.i)$

  $\sin(2\pi f^p.i)$,$\cos(2\pi f^p.i)$ for each i.
- θ^* can be searched within a discretized space of $[-\pi, \pi]$.

  $\rightarrow$ 1024 values are sufficient to reach the chosen precision,

  $\rightarrow$ $\sin(\theta)$,$\cos(\theta)$,$\sin(2\theta)$,$\cos(2\theta)$ can be pre-computed too.

  $\rightarrow$ dichotomic search.
- all parts (except dichotomy) can reduce to a loop of size M, using the same pixels as inputs.

  $\rightarrow$ they can be pipelined.

$\Rightarrow$ No more hard problems ... let's code.

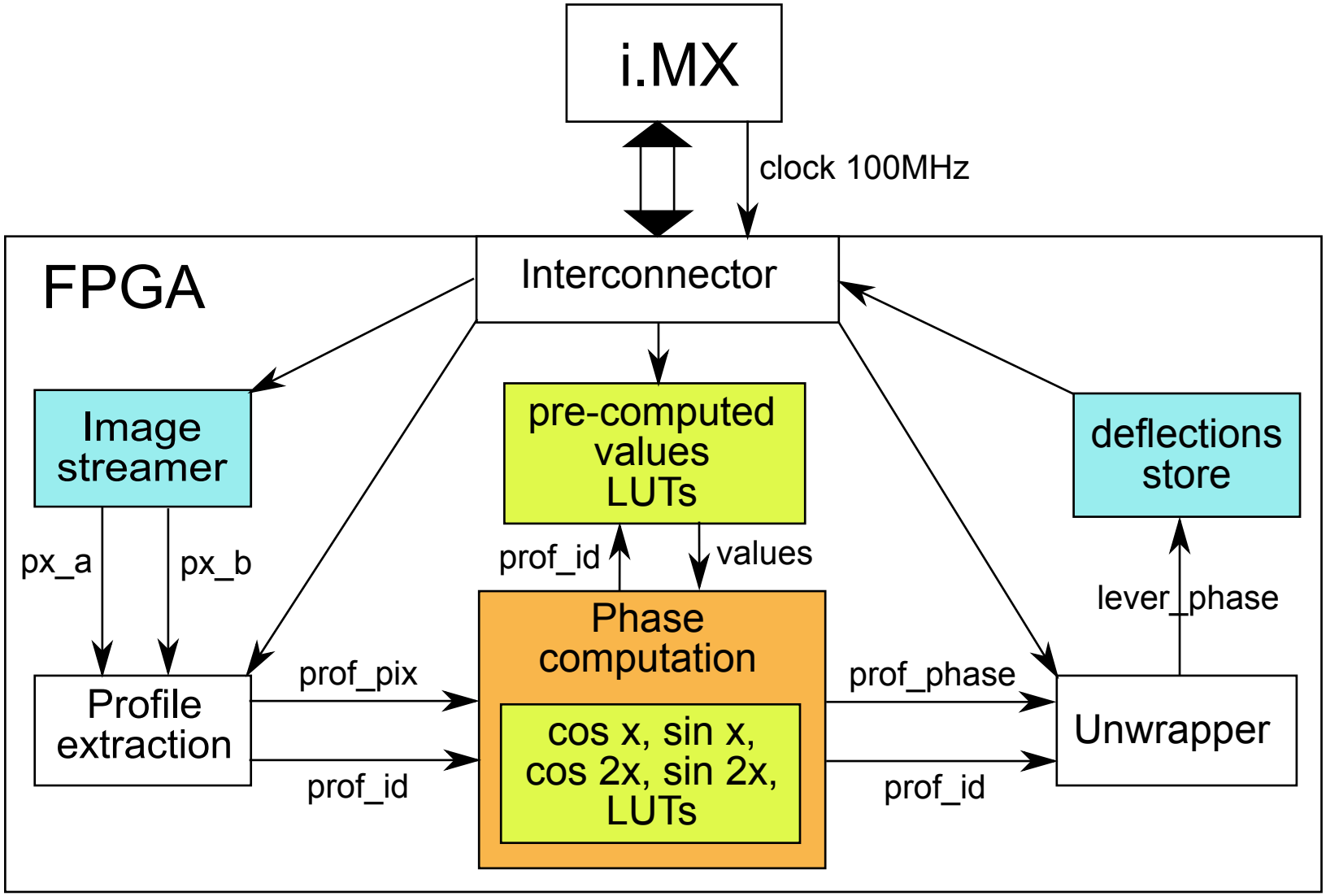
Implementation in VHDL (3/3)

Framework

- Armadeus APF27 + SP-Vision card :
  - → development board (USB, Ethernet, VGA, ...)
  - → Spartan 6 LX 100 ($\approx$ 100000 logic cells, 180 DSP, 976 Kb RAM),
  - → ARM processor (FreeScale i.MX 27), linked to FPGA,
  - → boot under Linux (stored in flash).
- emacs + ghdl + gtkwave :
  - → coding "at-hand" components and test-bench,
  - → debug and simulation.
- ISE v12 :
  - → CoreGen to generate VHDL source for dedicated units,
  - → synthesis, mapping, routing and bitstream generation.

Results (1/2)

Global design



Results (2/2)

Source code

- $\approx$ 9300 lines (without comments) in total,
- 2200 for deflection estimation, and 3800 generated by CoreGen.

FPGA utilization (as given by ISE)

- occupied slices : 914 (= 5%),
- RAM blocks : 28 (= 10%), DSPs : 15 (= 8%).

Timings

- maximum throughput (for a single phase) : 0.57μs ($\times$4 for CPU),
- latency (first pixel of profile $\rightarrow$ phase) : 1.28μs (unknown for CPU),
- 0.13μs to compute $\theta_{tip}^{unw.}$,

$\Rightarrow 1.28 + 200 \times 0.57 + 0.13 \approx 116\mu s$ for 100 levers.

$\Rightarrow$ maximum of 8620 images per second.

$\Rightarrow$ in real experiments : $\approx$ 1780 images per second.

Conclusion

Goals largely reached

- maximum computation rate : $\times 8$ on what expected,
- average precision equals to noise at rest (i.e. 0.3nm),

Works in progress

- component to drive the camera,
- component to output deflection efficiently,
- driving application :

  laptop part : define profiles locations, calibration, pre-computations,

  apf27 part : bitstream loading, components initialization, collecting deflections, ...

Future works

- taking account coupling between levers deflections,
- studying and modeling the dynamic behavior of the array,
- adding real-time control.