Readout Architecture without MacroPixel: first simulation results

04/04/2009

Giulia Casarosa, Fabio Morsani,
Eugenio Paoloni, Giuliana Rizzo

outline

- motivations and general features
- pixel
- readout architectures
  - ARCH_V0
  - ARCH_V2
- simulation
  - hit generation
  - output & debugging tools
- results

Proposal for an alternative readout for Vertical Scale Integration: why?

- MP arrangement has been adopted to reduce at minimum in-pixel logic (competitive NWELL) and the digital switching lines running above pixel columns but it increases the local routing congestion: switching digital lines are most local to MP
- 3 reasons to eliminate the MP:
  - With MP the routing of private lines (FastOR, Latch Enable) scales with matrix column dimension
  - Inefficiency due to dead time (MP freezed) depends on MP dimensions
  - The proposed readout without MP is more efficient: at each $T_{RDclock}$ only data from a column fired are sent to the readout.
- We are looking forward bigger matrix and we are realizing the MP approach is an obstacle instead of an help
- We need any matrix readout speed increase expecially if it carries some readout logic simplification

general features

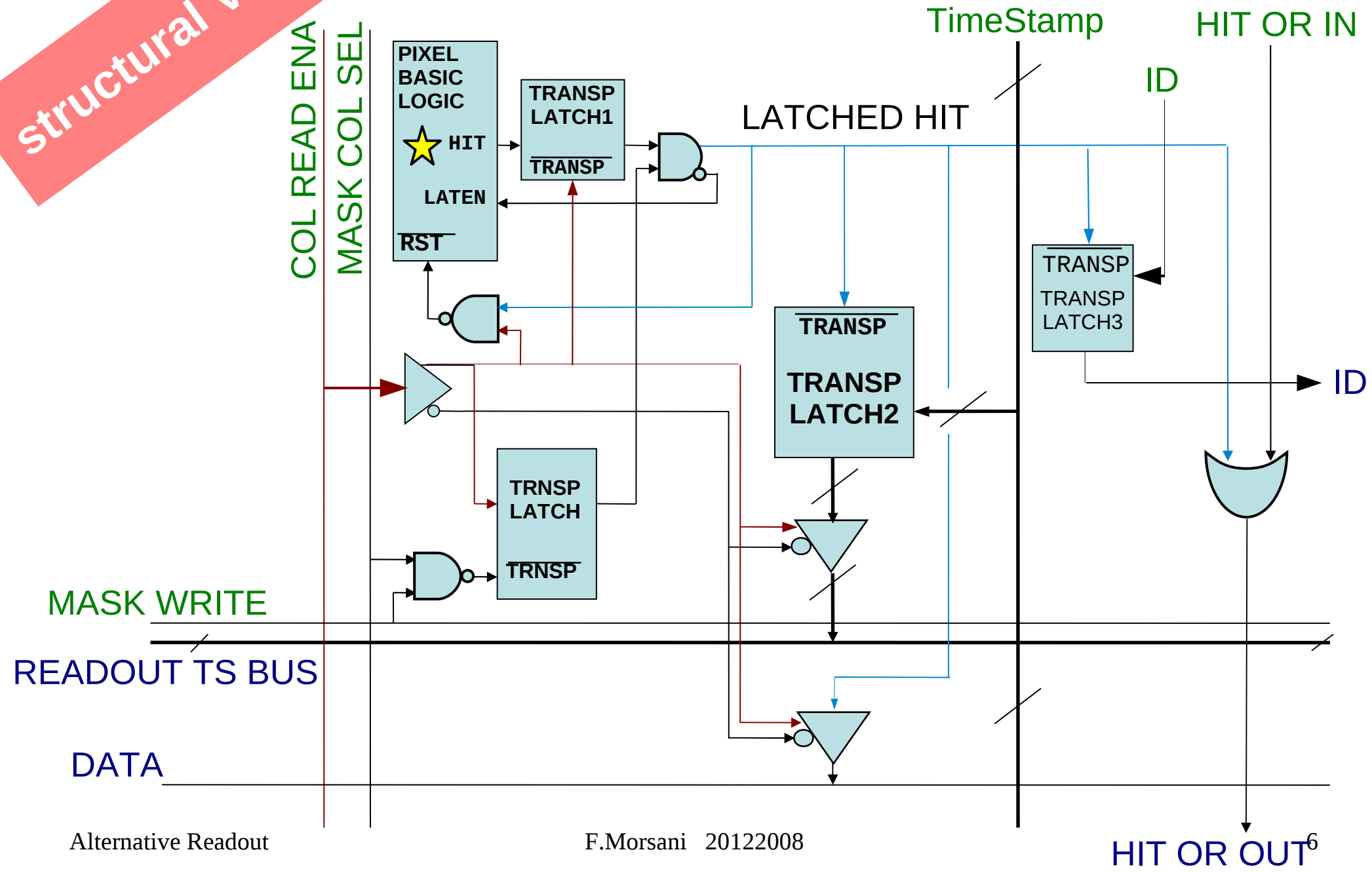
- no macropixel nor extra dead time associated with its freezing
- pixel level TS
  - TS at hit generation time (vs BCO time)
- per column readout
- pixel level masking

pixel - features

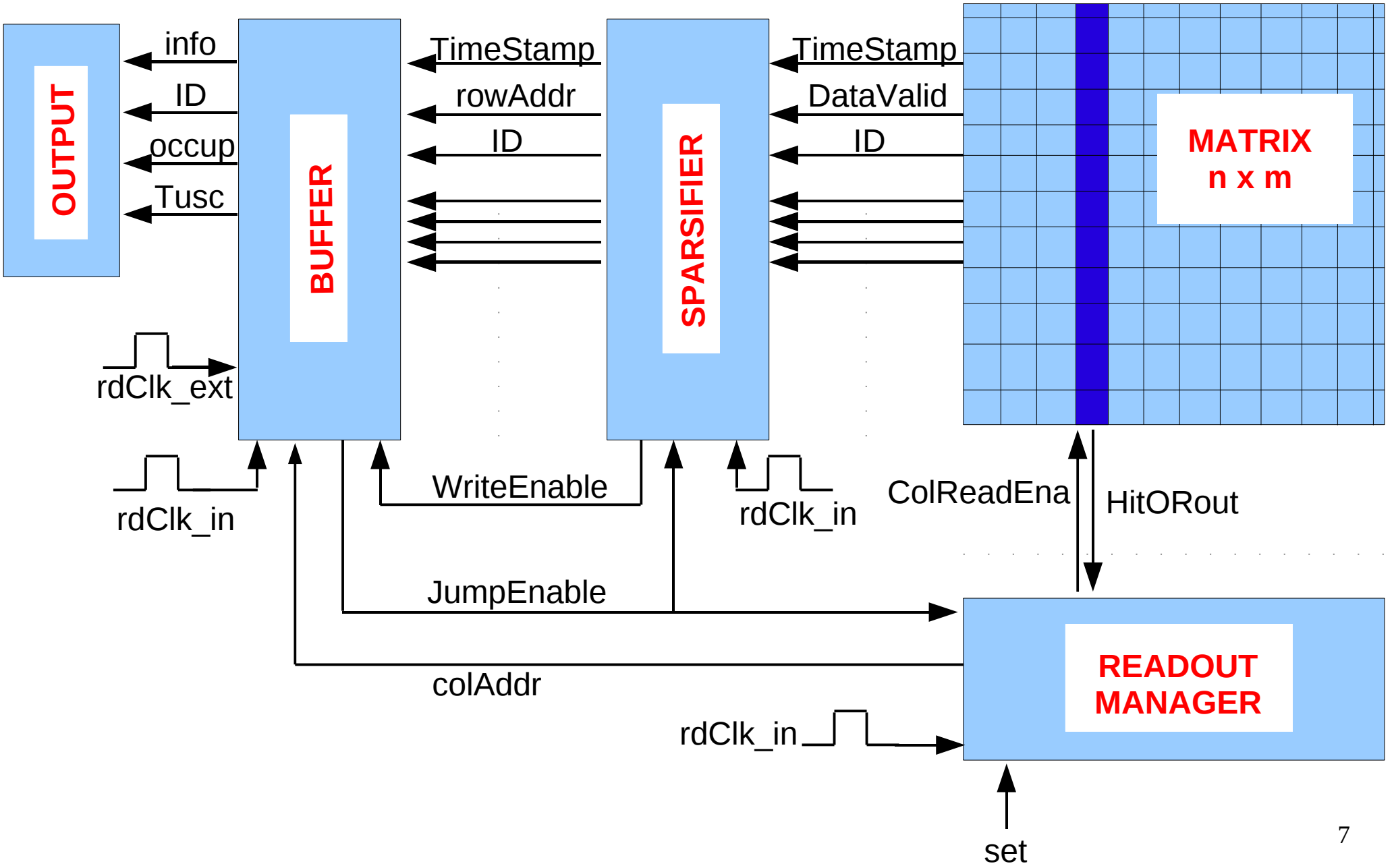
- no macropixels
  - freezing is only at pixel level
- reset
  - done when the column is read
- Time Stamp
  - latched locally into each pixel at hit occurrence
  - not related to the BCO clock
- masking
  - pixel basic logic is disabled

pixel - architecture

Structural VHDL



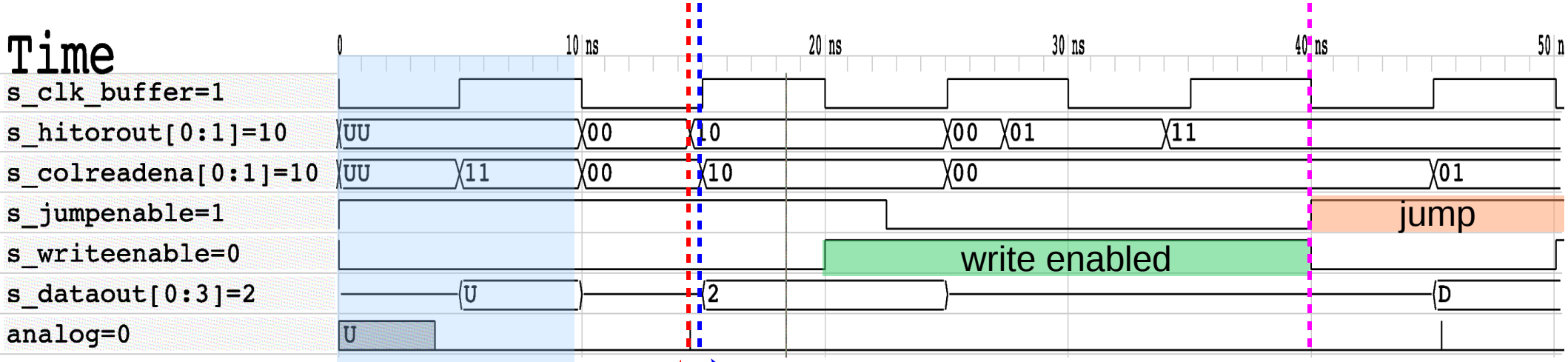
readout - ARCH_v0



readout - features

- readout manager (rdClk_in):
  - @ rising edge(clk): if readout is ready (if it's possible to receive new data), columns having their HIT OR = 1 are enabled for reading setting COL READ ENA = 1
- sparsifier & buffer (rdClk_in):
  - every pixel of the enabled column put on the output of the matrix DataValid (0 or 1) & TS
  - the sparsfier knows when data from a new column are coming
  - when the buffer has finished writing data of the column, enable the jump to the next column
- output (rdClk_ext):
  - @ rising edge(clk): buffer send the bits relative to one hit outside.

example – ARCH_V0



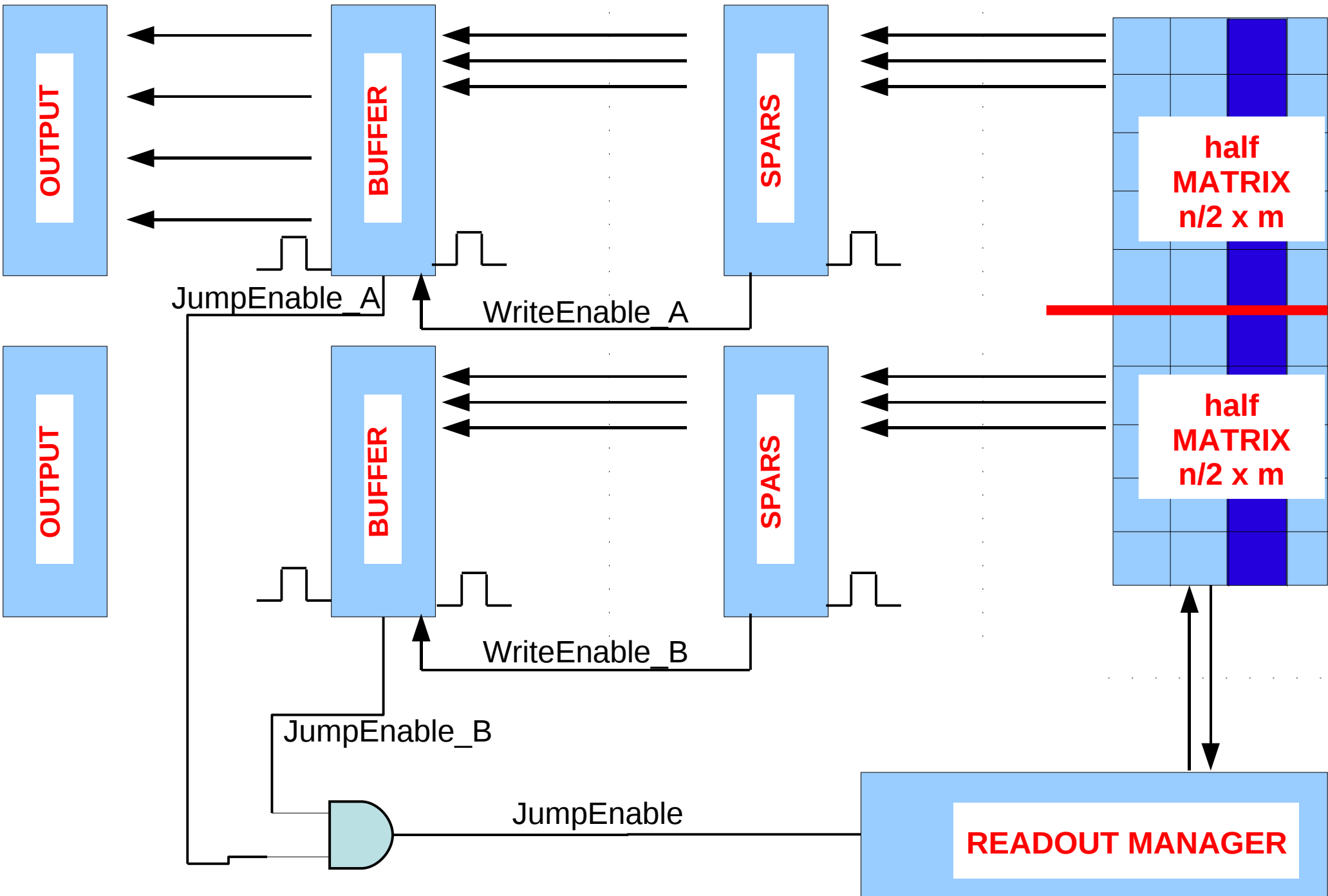
reset of the matrix

@ rising edge(clk)
COL READ ENA =1

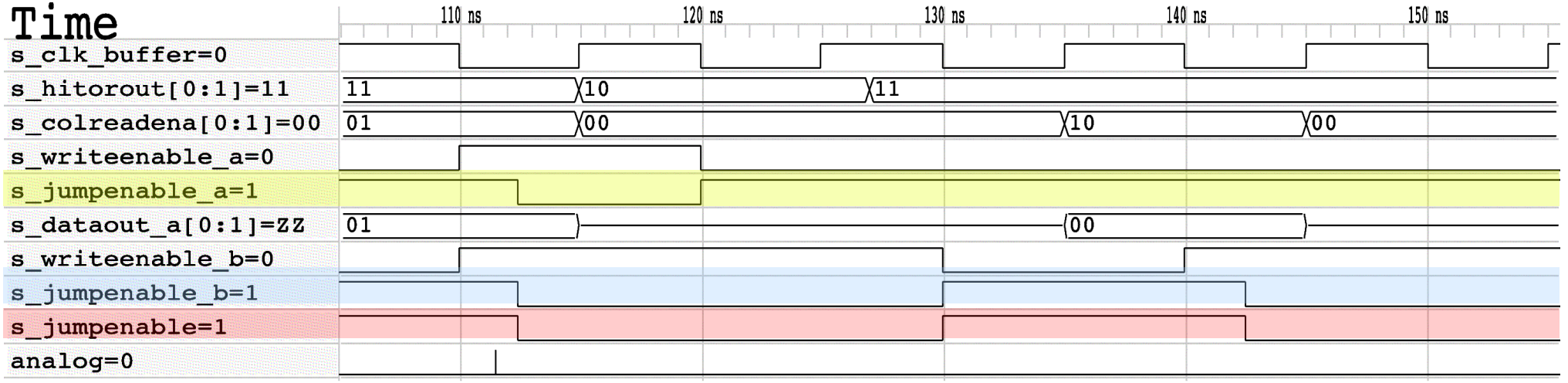
The buffer finished to write infos from the column, enable the jump to the next column

★ HIT

readout - ARCH_v2



example – ARCH_V2

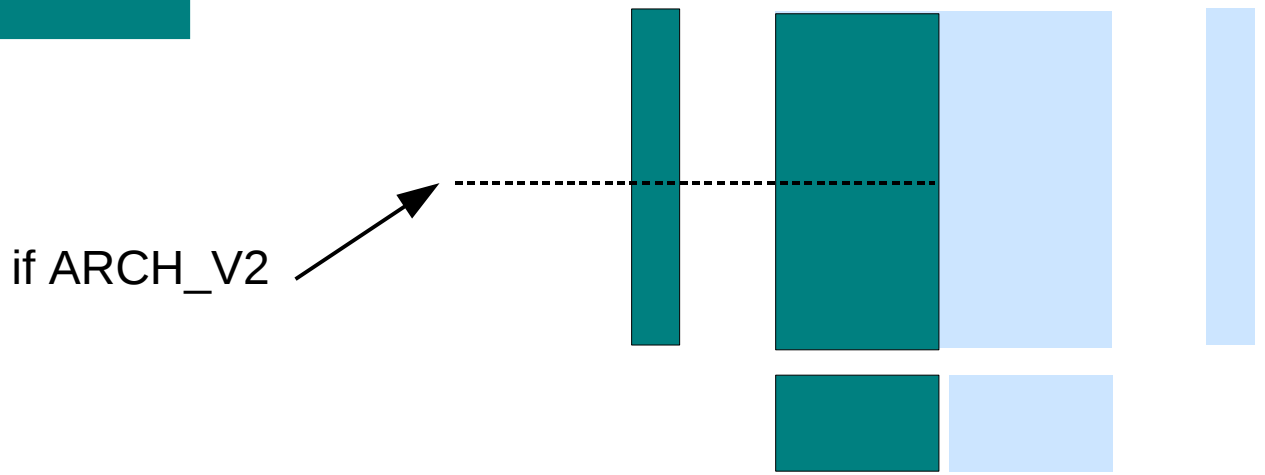


the simulation (1)

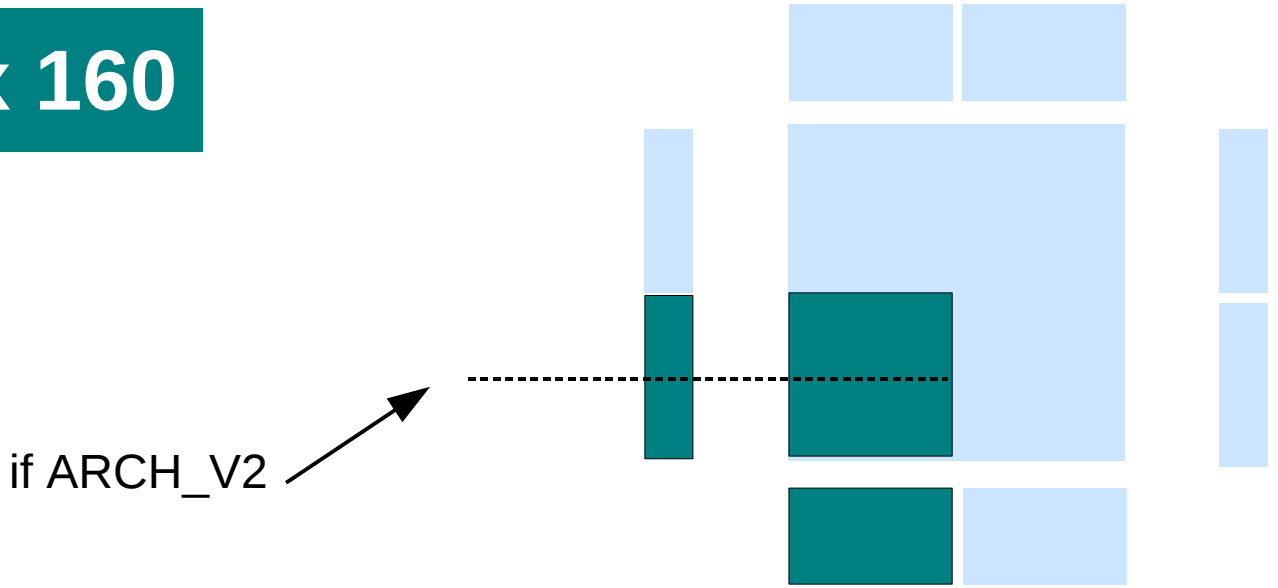
- set of the parameters used for simulations:
  - Time Stamp (8 bits) period = 10 ns **NOT correlated to readout efficiency**
  - rdClk_in period = 10 ns (→ 100 MHz)
  - rdClk_out period = 10 ns (→ 100 MHz)
  - n_rows = 256
  - n_columns = 320
  - matrix pitch = 40 x 40 μm^2
- ....not only, we'll show results on readout where we changed:
  - architecture (V0 – V2)
  - rdClk_out 100 MHz or 75 MHz
  - n_rows x n_columns = 256x320; 256x160; 128x160 (see next slide)
- Rate = 100 MHz/cm^2 (also 150 and 200 MHz/cm^2)

geometries simulated

256 x 160



128 x 160



the generation

- language used VHDL
  - pixel architecture is structural, readout is purely behavioral
  - clock periods, n_rows, n_columns, rate are customizable
- generate a hit in a random pixel of the matrix after a Δt which has a exponential lifetime distribution
  - row = u * (n_rows-1) where u ~ U(0,1)
  - col = u * (n_columns-1) where u ~ U(0,1)
  - $\Delta t = - (1/\mu) * \ln(u)$ where u ~ U(0,1) and μ = integrated rate
- every hit has a unique ID useful for debugging and other studies
- informations from the simulation: some files are produced (see next slide)

output files for debugging & studies

- MC_truth.dat

#	ID	row	col	pix	post	Ttrue	runCol	prevCol
	0	41	4	0	1	20.879452 ns	-1	-2147483648
	1	61	43	0	1	46.731374 ns	-1	4
	2	26	40	0	1	54.435123 ns	-1	4

  - “pix” and “post” = LATCHED HIT signal before and after the hit
  - “runCol” = column enabled @ Ttrue, -1 if no column is enabled
  - “prevCol” = last column enabled

- results.dat

#	ID	row	col	TS	Tusc	Mem occup
	0	41	4	2	3	1
	2	26	40	5	6	1

  - “Tusc” = # rdClk_ext when hit is sent outside buffer
  - “Mem occup” = # events stored in the buffer

- sparsifier.dat (rdClk_in)

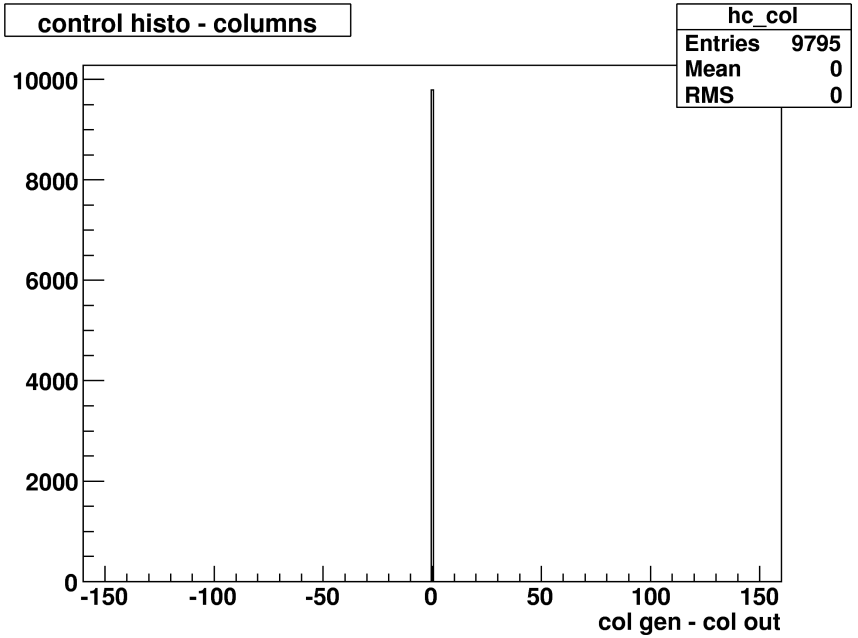
  # rdClk_in & {row, ID, TS} of the hits present on the sparsifier

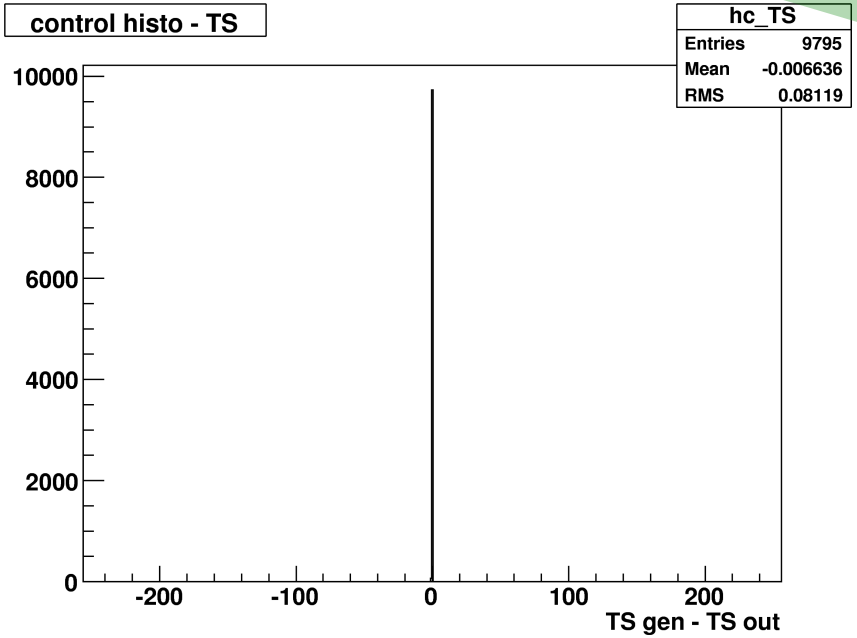
```
clock 9
 row = 26  ID = 2  TS = 00000101
-----------------------------------------------------
clock 10
```

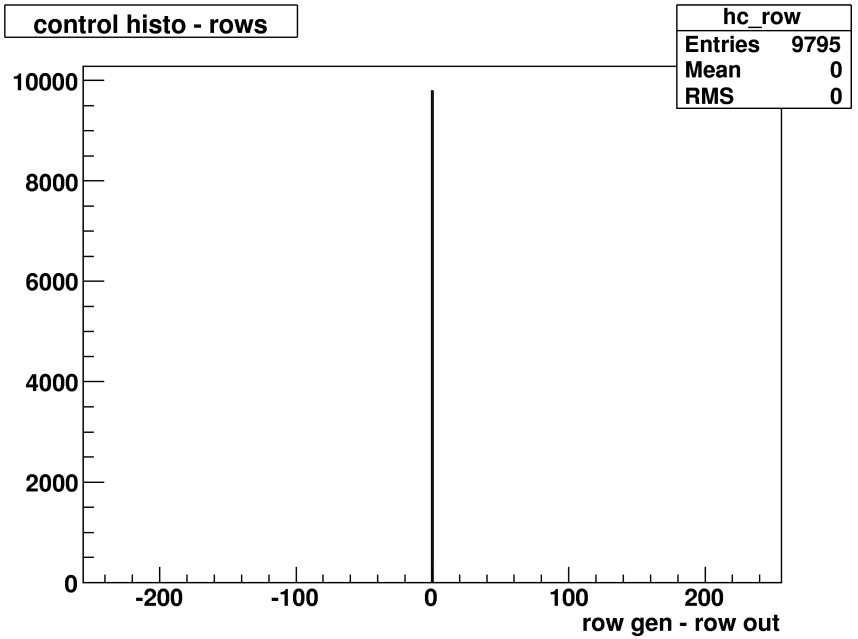
- [buffer.dat (rdClk_in)
  - all the content of the buffer]

control checks

thanks to unique ID







CHECK that row, column & TS MATCH the GENERATED ones

some useful quantities

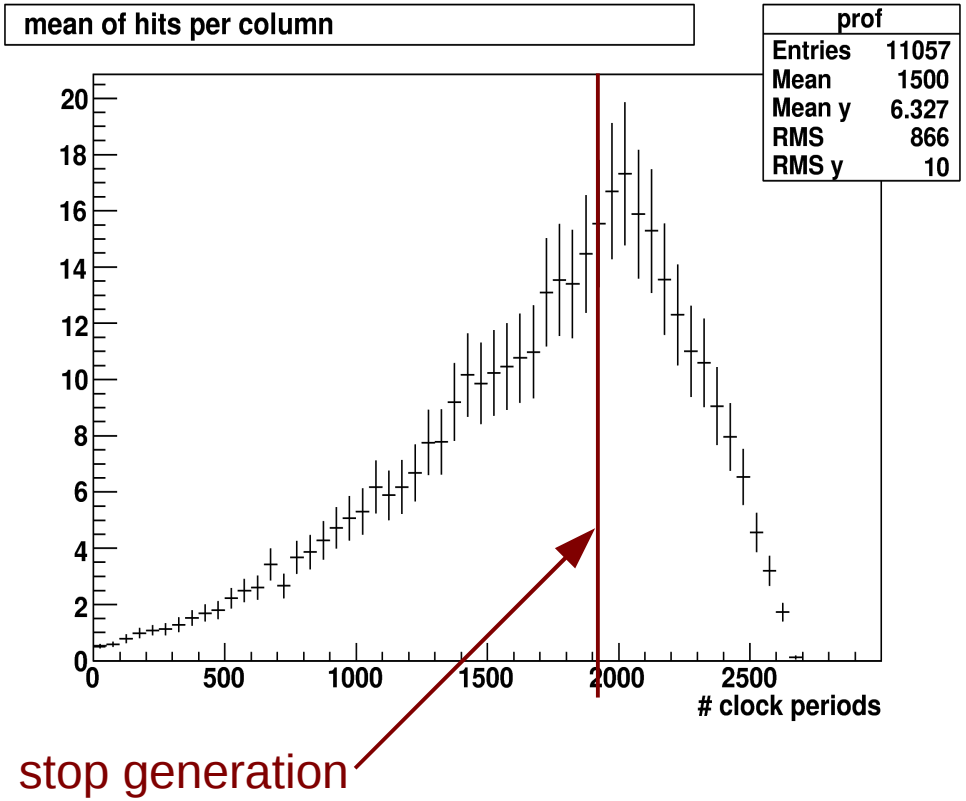
- Δt = time the hit needs to get out from the readout
- $m = \langle \#\ \text{hit / col} \rangle_{t\_clk}$ = mean number of hit on a column
- t_{LOOP} = time to complete the matrix readout loop
- ε = efficiency = N_{out}/N_{in}

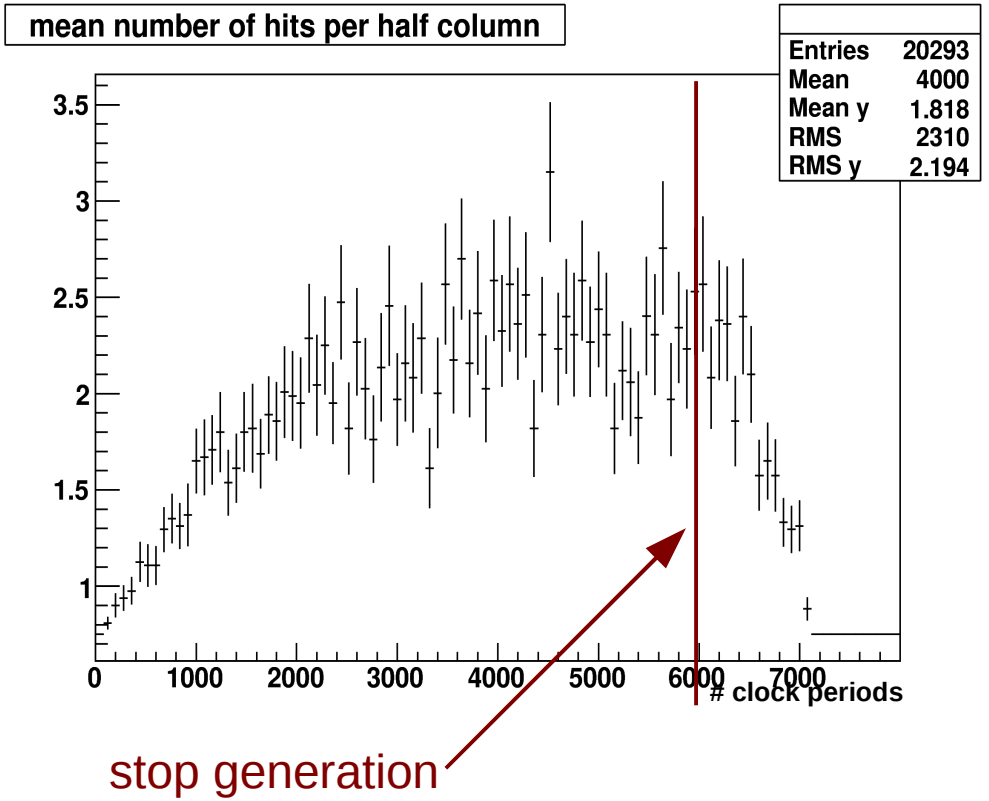
the readout **inefficiency** is due only to pixel which are hit when already hit

$$1 - \varepsilon \sim \frac{t_{LOOP} / 2}{\left(\text{n\_rows x n\_columns} \,/\, \text{rate} \right)}$$

time between a hit and another on the same pixel

generation of hits

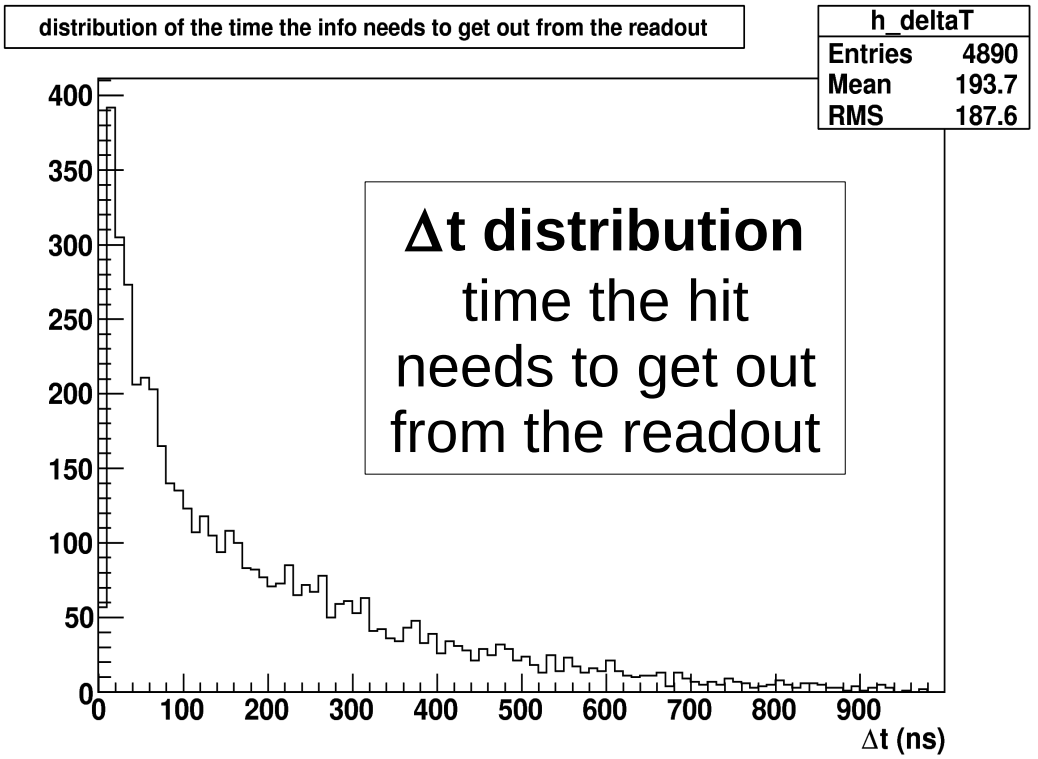


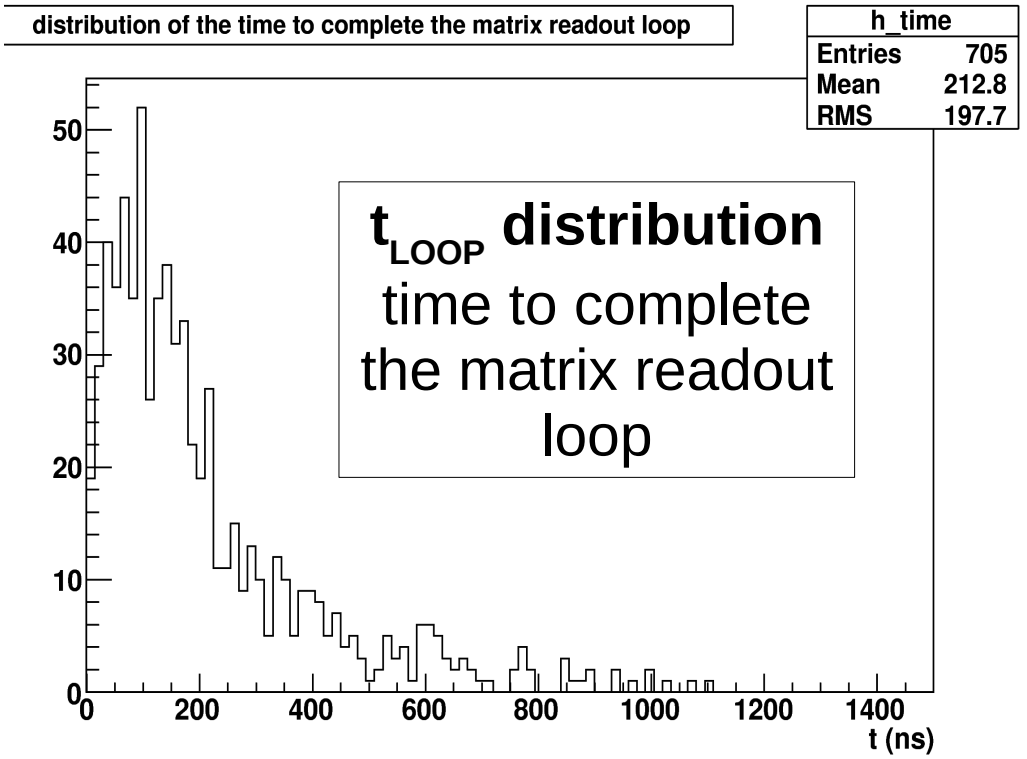


256 x 320
rate = **100 MHz/cm²**
(132 MHz)
ARCH_V0 (100 MHz)

256 x 320
rate = **100 MHz/cm²**
(132 MHz)
ARCH_V2 (100 MHz)

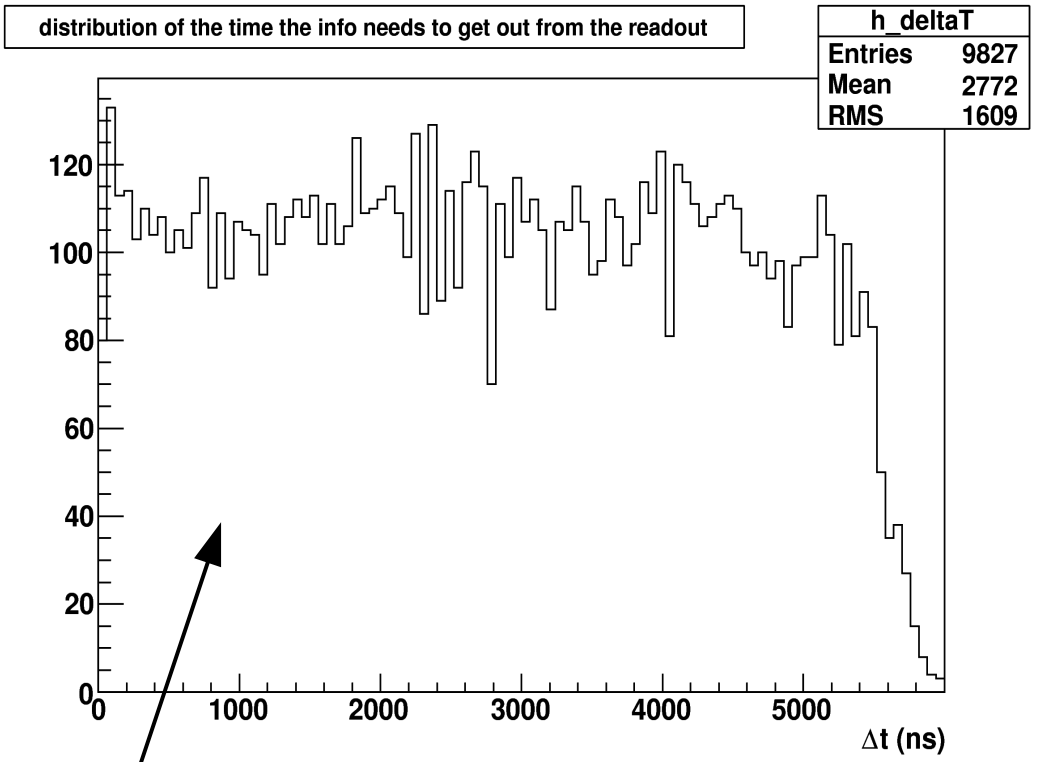
timing – some distributions

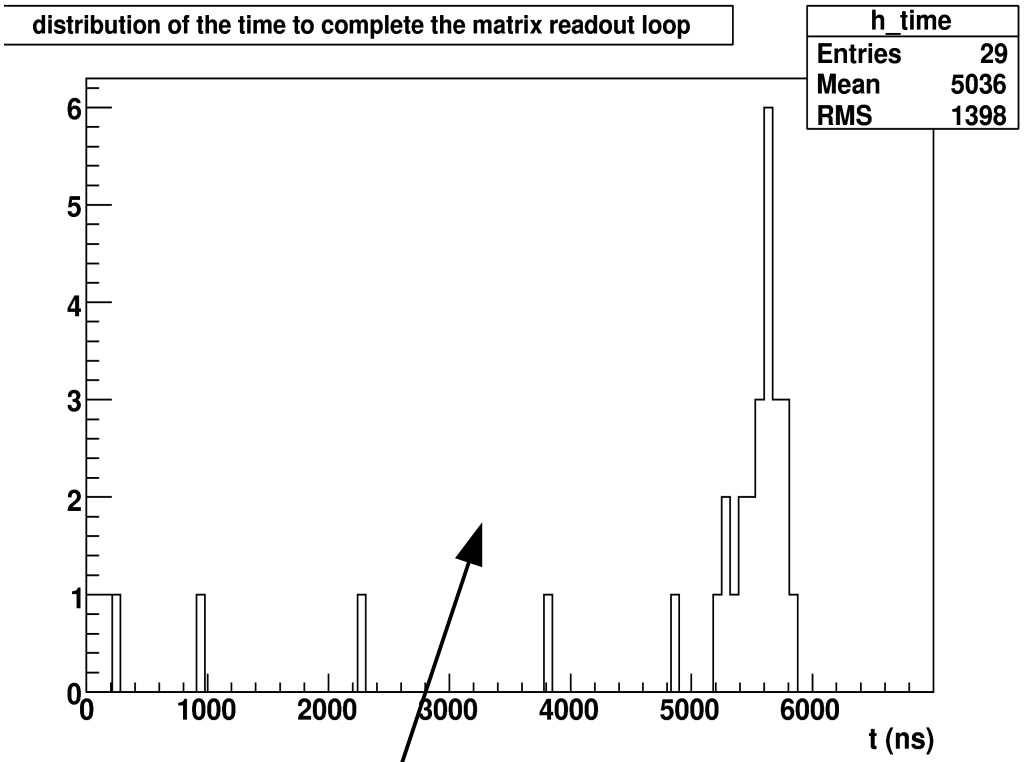




128 x 160
rate = **100 MHz/cm²**
(33 MHz)
ARCH_V2 (75 MHz)

timing – some distributions





Δt distribution
time the hit needs to get out from the readout

256 x 160
rate = **100 MHz/cm²**
(66 MHz)
ARCH_V2 (75 MHz)

t_{LOOP} distribution
time to complete the matrix readout loop

PRELIMINARY

results of simulations

RATE
100 MHz/cm²

	ARCH_V0 @ 100 MHz	ARCH_V2 @ 100 MHz	ARCH_V2 @ 75 MHz
256 x 320 (132 MHz)	ε = 95.82% m = 6 Δt = 24 µs	ε = 98.10% m = 3.6 Δt = 13 µs	ε = 98.10% m = 3.6 Δt = 13 µs
256 x 160 (66 MHz)	ε = 99.22% m = 1.6 Δt = 4.2 µs	ε = 99.54% m = 2 Δt = 2.8 µs	ε = 99.54% m = 1.8 Δt = 2.8 µs
128 x 160 (33 MHz)	ε = 99.94% m = 0.5 Δt = 209 ns	ε = 99.94% m = 1.5 Δt = 188 ns	ε = 99.94% m = 1.5 Δt = 193 ns

considering the matrix **256 x 320**

barrell out operates @ rate:		
100 M hit/s	200 M hit/s	150 M hit/s
200 M hit/s	400 M hit/s	300 M hit/s
400 M hit/s	800 M hit/s	600 M hit/s

PRELIMINARY

results of simulations

RATE
150 MHz/cm²

	ARCH_V0 @ 100 MHz	ARCH_V2 @ 100 MHz
256 x 320 (196 MHz)		ε = 95.[illegible]% m = [illegible].6 Δt = 27 µs
256 x 160 (98 MHz)	ε = 97.19% m = 3 Δt = 13 µs	ε = 99.00% m = 2 Δt = 4.4 µs
128 x 160 (49 MHz)	ε = 99.50% m = 0.8 Δt = 2.3 µs	

considering the matrix **256 x 320**

barrell out operates @ rate:	
100 M hit/s	200 M hit/s
200 M hit/s	400 M hit/s
400 M hit/s	800 M hit/s

PRELIMINARY

results of simulations

RATE
200 MHz/cm²

	ARCH_V0 @ 100 MHz	ARCH_V2 @ 100 MHz
256 x 160 (132 MHz)		ε = 97.90% m = 2.8 Δt = 6.9 µs
128 x 160 (66 MHz)	ε = 98.70% m = 1.8 Δt = 4.2 µs	ε = 99.19% m = 1.2 Δt = 2.7 µs

considering the matrix **256 x 320**

barrell out operates @ rate:	
200 M hit/s	400 M hit/s
400 M hit/s	800 M hit/s

BACKUP

readout - ARCH_V2