Preparation of the FLAME based readout and DAQ system for the LumiCal test beam

Jakub Moroń, Sz. Bugiel, M. Firlej, T. Fiutowski, M. Idzik, A. Skoczeń, K. Świentek

AGH University of Science and Technology, Krakow, Poland

This work was supported by the European Union Horizon 2020 Research and Innovation programme under Grant Agreement no.654168 (AIDA-2020) and by the Polish Ministry of Science and Higher Education under Contract No. 3501/H2020/2016/2

34nd FCAL Collaboration Workshop
26-27 March 2019, CERN, Geneva

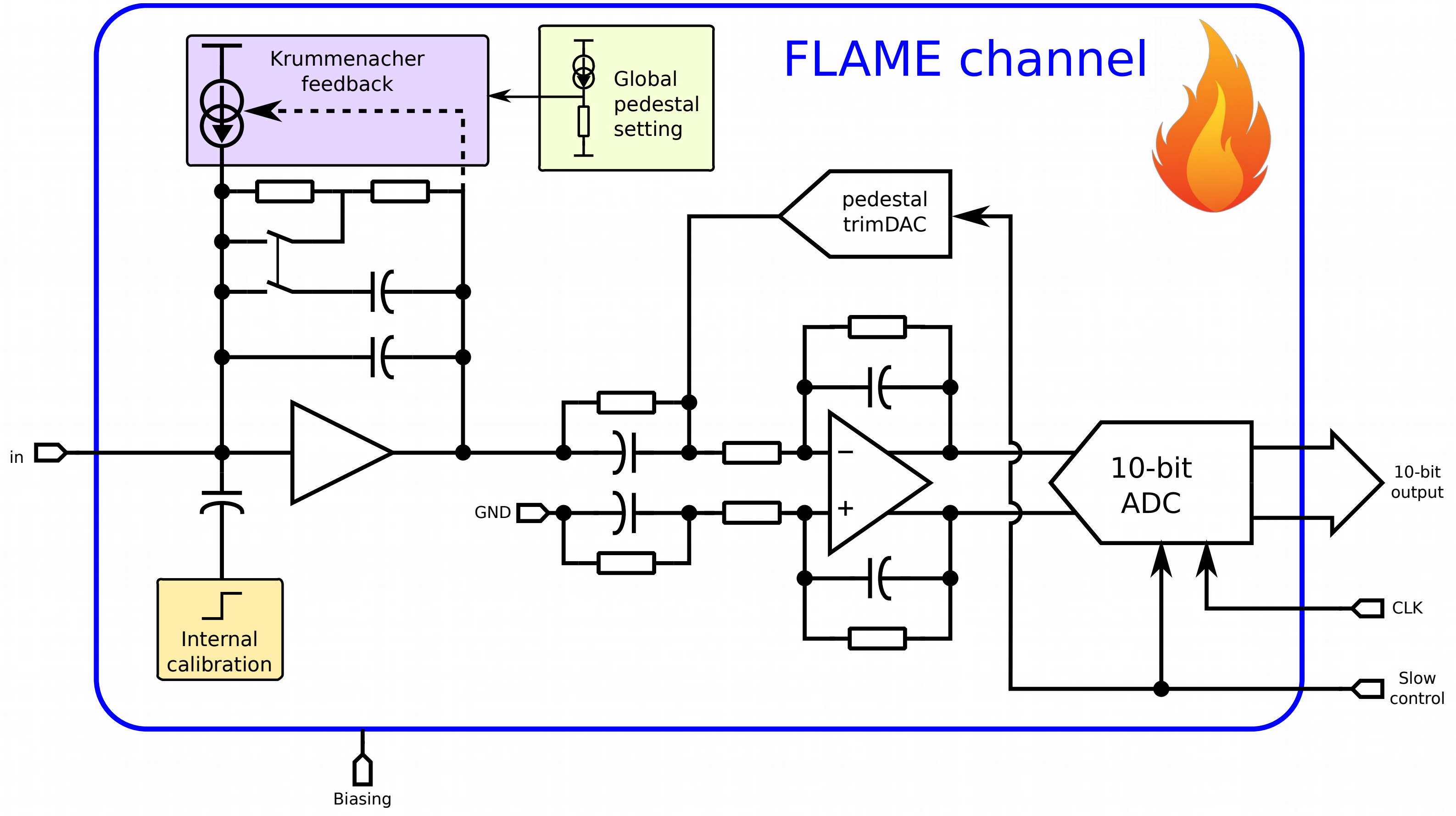
Outline

- FLAME ASIC status
- DAQ scheme and verification of DSP
- Hardware and firmware status

Outline

- ➔ FLAME ASIC status
- DAQ scheme and verification of DSP
- Hardware and firmware status

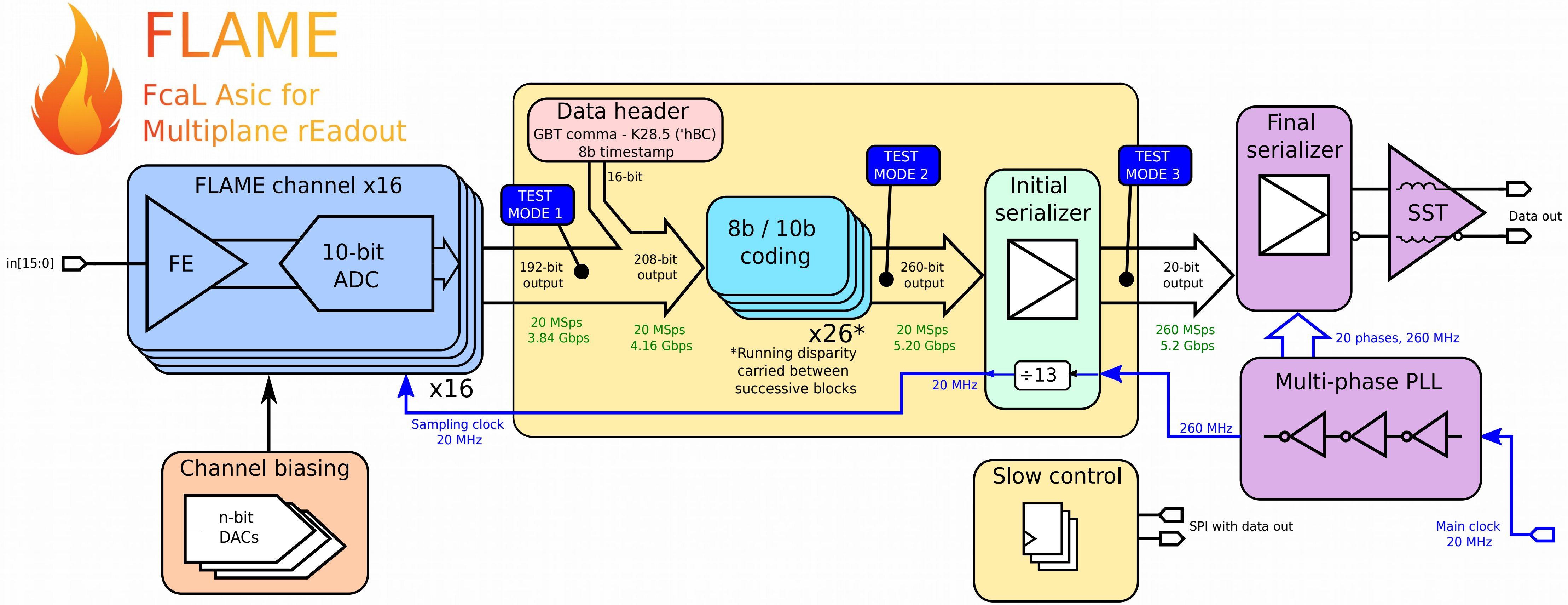
FLAME – Channel architecture



- Analogue front-end comprising:
  - Charge sensitive preamplifier with variable gain:
    - High gain – for testbeam - up to 200 fC with MIP sensitivity
    - Low gain – for shower development (up to 6 pC)
  - Differential CR-RC shaper with 50ns peaking time
  - Krummenacher feedback and pedestal trim DAC
  - Internal calibration circuit
- 10-bit multichannel SAR ADC
  - Sampling rate up to 50 MSps
  - DNL, INL < 0.5 LSB
  - ENOB > 9.5
  - Ultra low power consumption (below 1 mW per channel at 40 MSps)

FLAME – "Half-ASIC" architecture

Not the full ASIC shown!

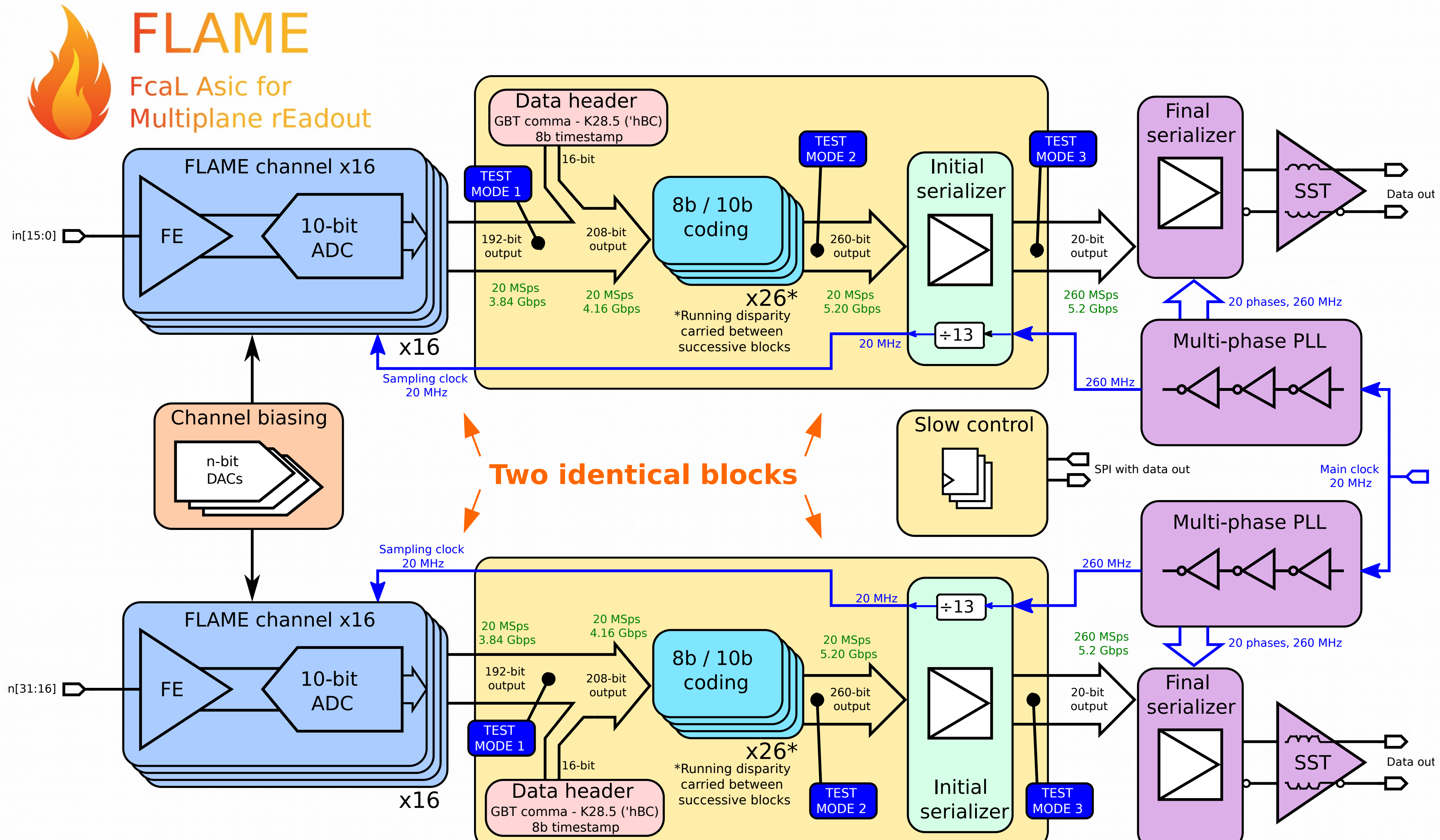


- Complete readout ASIC integrating whole functionality (biasing, calibration, etc.)
- 32 mix-mode channels comprising:
  - Variable gain front-end
  - 10-bit SAR ADC

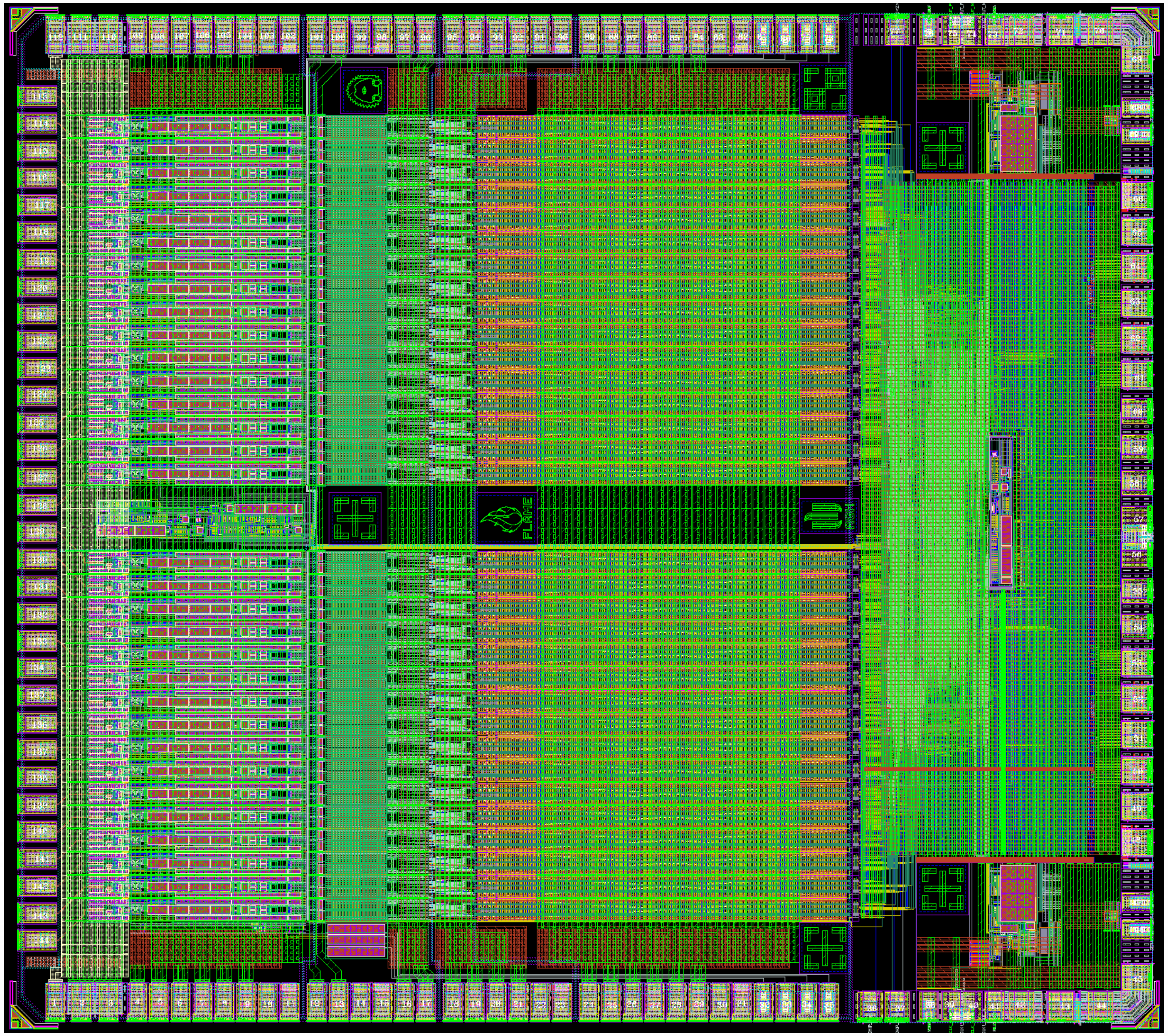
- Data encapsulation and 8b/10b coding (according to the Xilinx MGT specification)
- Multi-phase PLL based fast serializer (up to 8 Gbps)
- Fast SST driver (up to 8 Gbps)

FLAME – ASIC architecture

•Two 16-channel, fully functional blocks → two “ASICs” in one padring to save the PCB area and maximize the instrumented sensor area



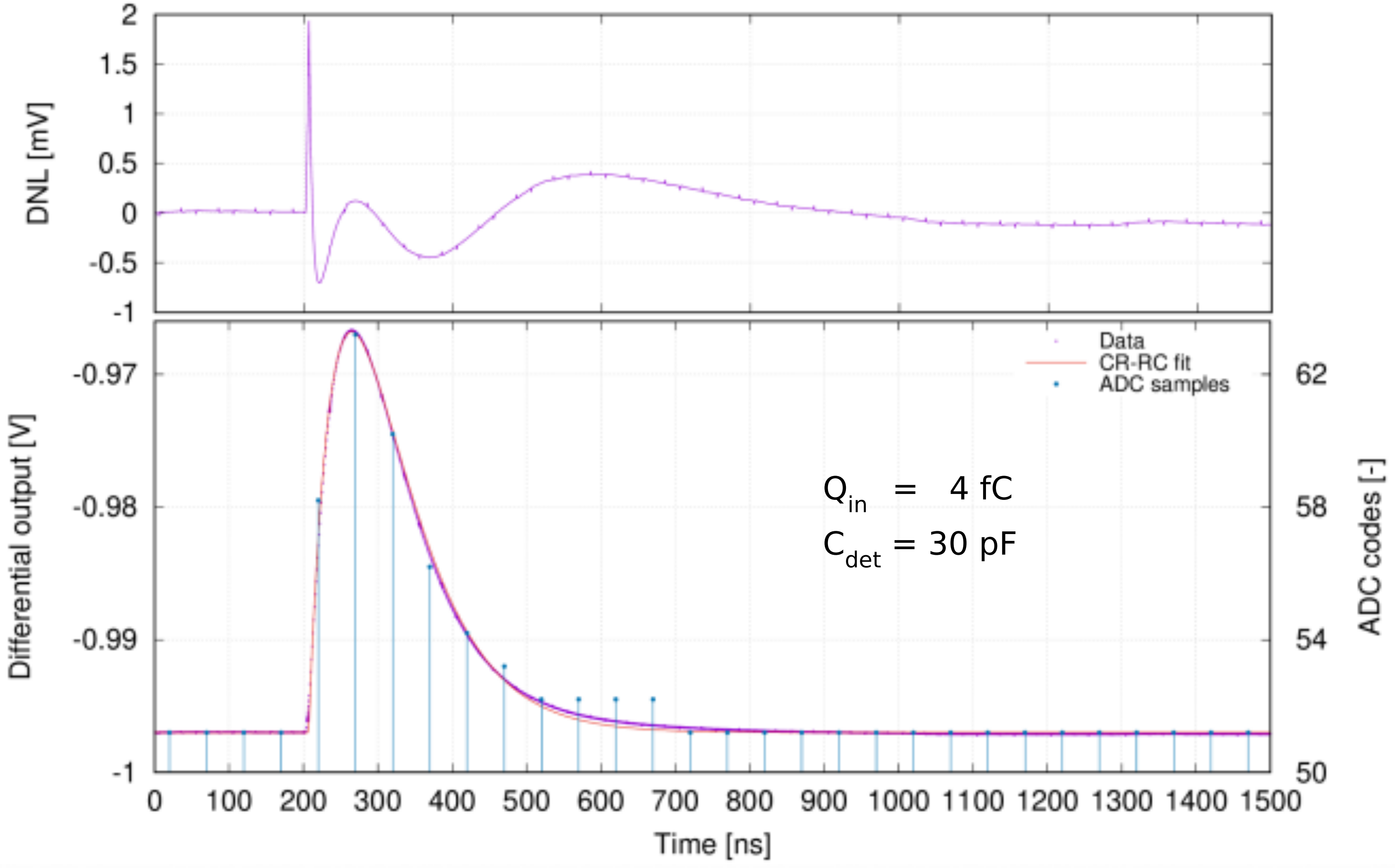
FLAME – Current status



ASIC submitted for fabrication on 16.02.2019

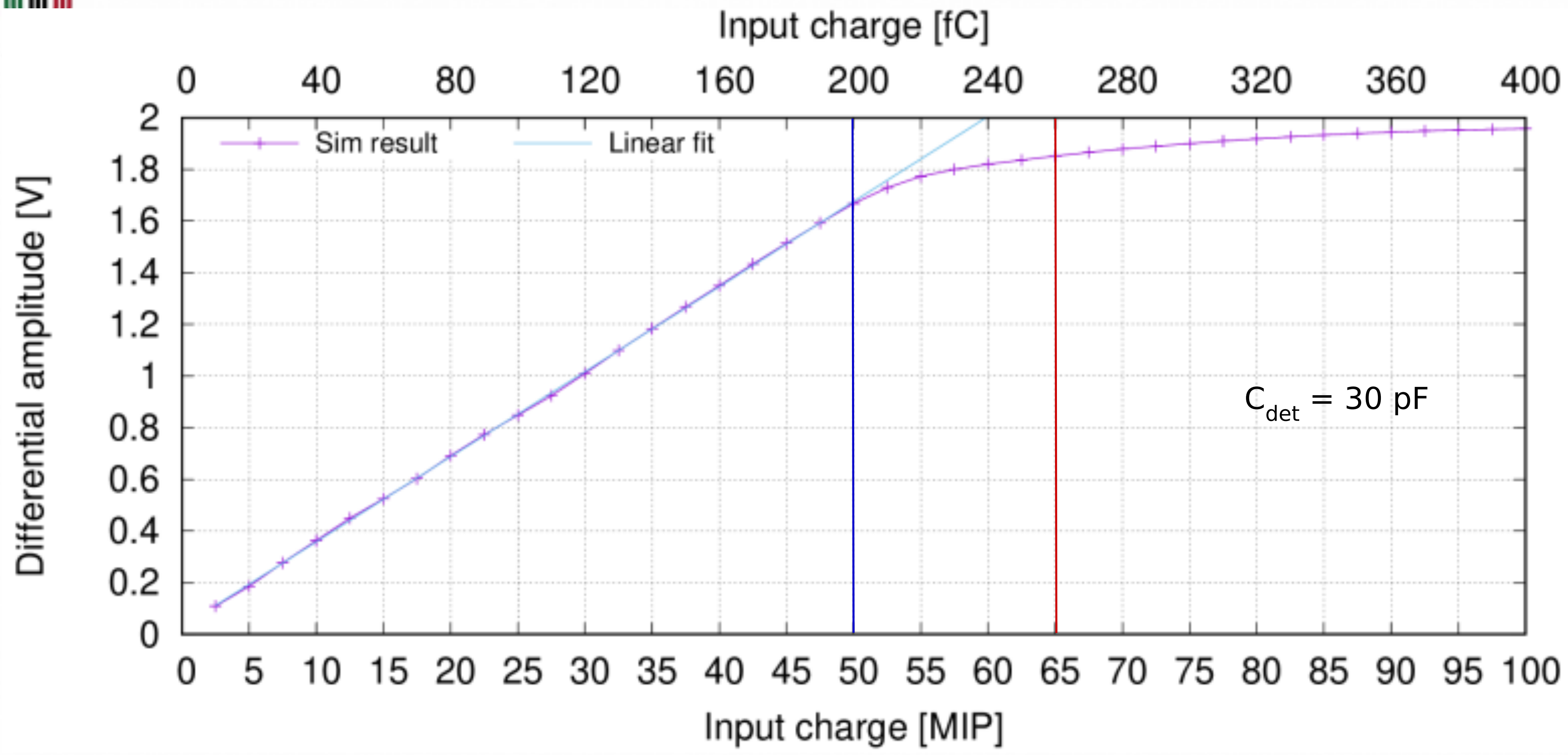
- ASICs expected at ~**beginning of June**
- 3 wafers × 80 ASICs / wafer – we should expect ~**240 ASICs**
- Die size: **3.7 × 4.3 mm^2**

FLAME – MIP response and pulse shape



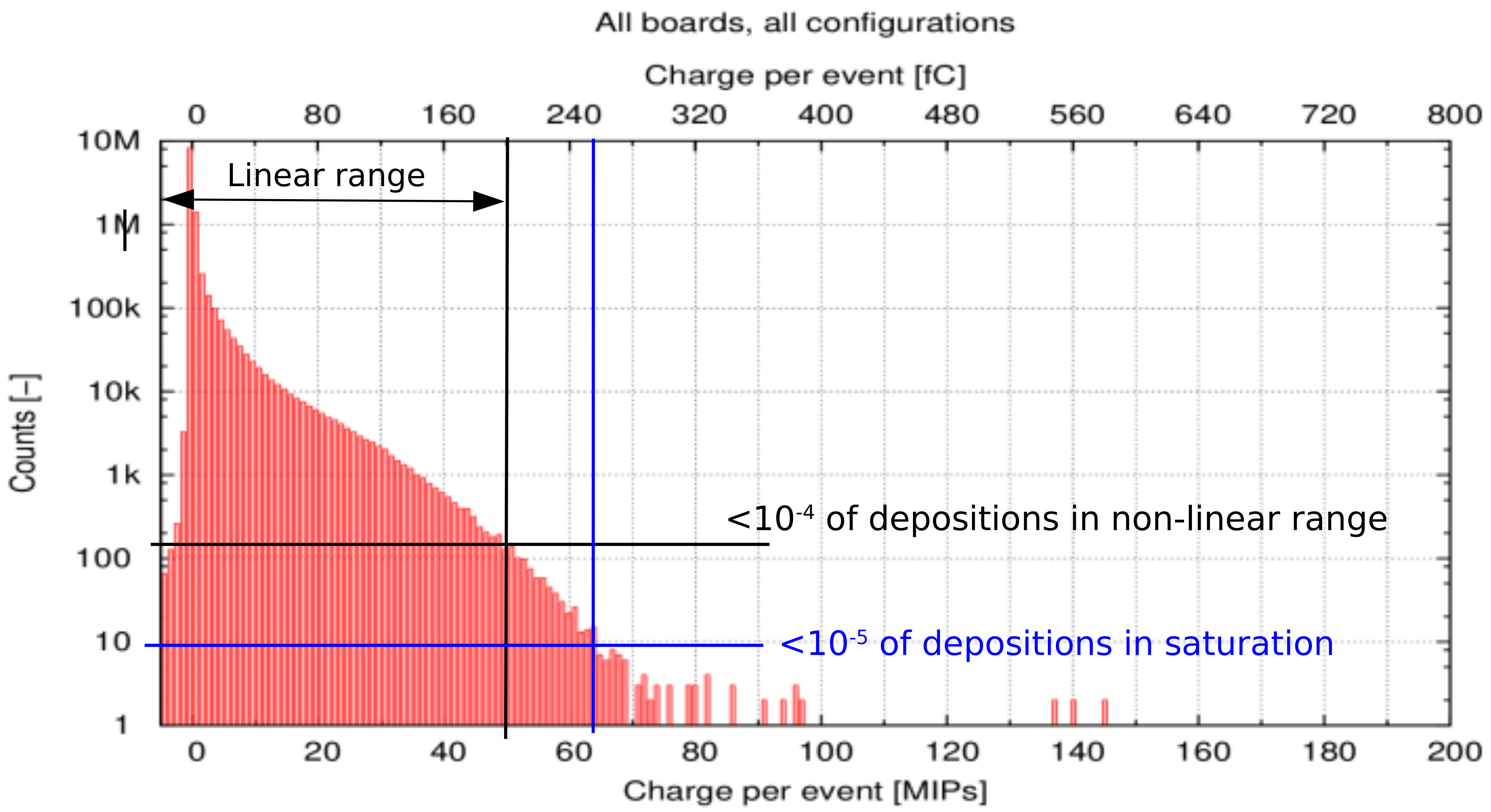
- Simulated analogue response for MIP (4fC) in high gain
- Amplitude ≃ 30.4 mV (13 LSB)
- Very good agreement with CR-RC pulse – difference (DNL) between data and fitted CR-RC below ±0.5 mV (±0.25 LSB) – excluding the narrow peak at pulse beginning

FLAME – Linearity



- High gain → testbeam gain:
  - Linear range up to 50 MIPs,
  - Dynamic range (without pulse shape deterioration)up to ~65 MIPs
  - Gain ≃ 32.9 mV/MIP ≃ 8.2 mV/fC ≃ 3.7 LSB / fC
  - SNR for MIP @ 30 pF detector capacitance ~ 30

Testbeam 2014 – max. deposition per pad



- Results from 2014 testbeam (5 GeV electrons) – deposition per pad per event
  - Around 10M events reconstructed:
  - <1k events ($<10^{-4}$) with depositions > 200 fC (outside linear range)
  - <100 events ($<10^{-5}$) with depositions > 260 fC (in saturation range)

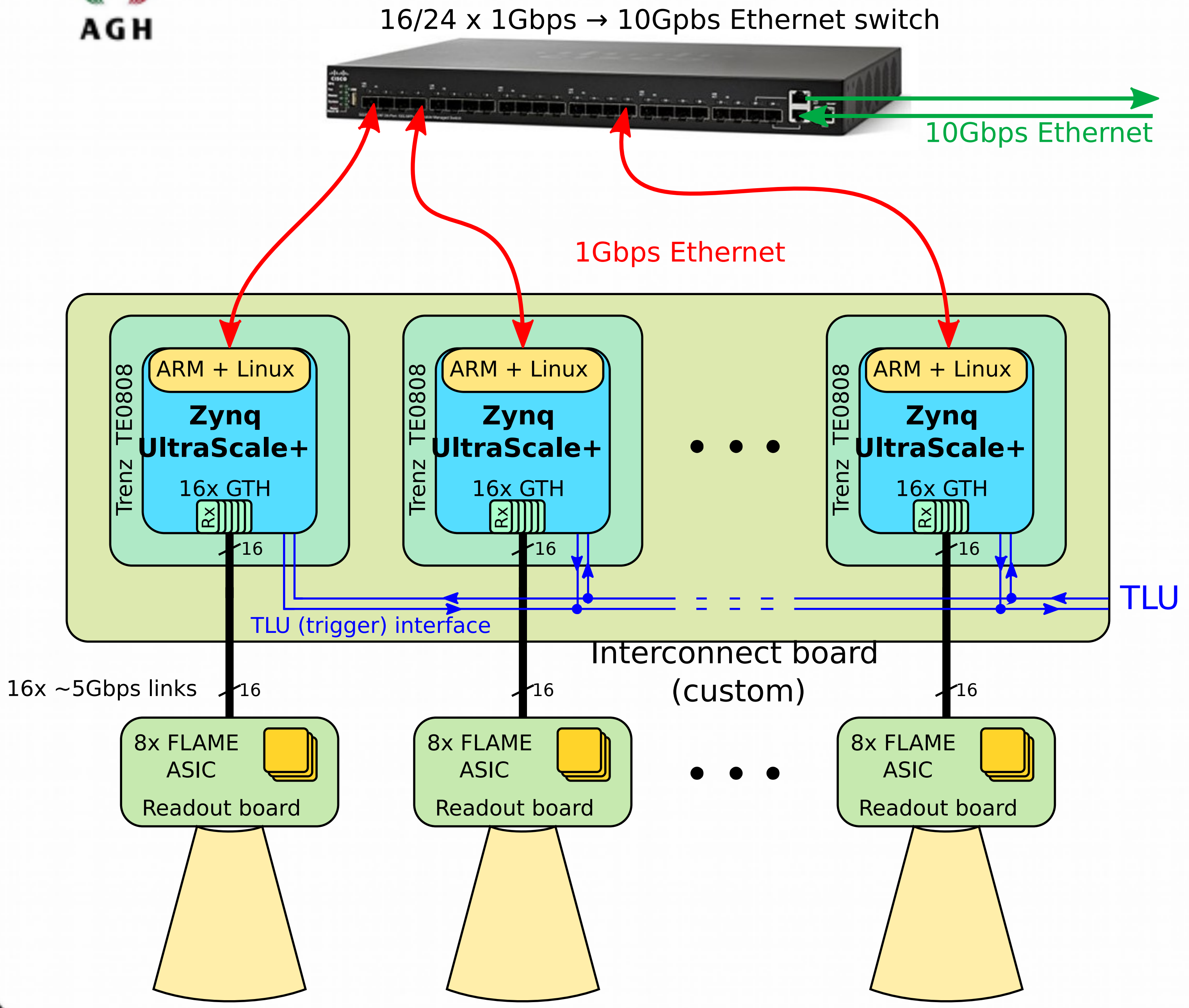
Outline

- FLAME ASIC status

➔ DAQ scheme and verification of DSP

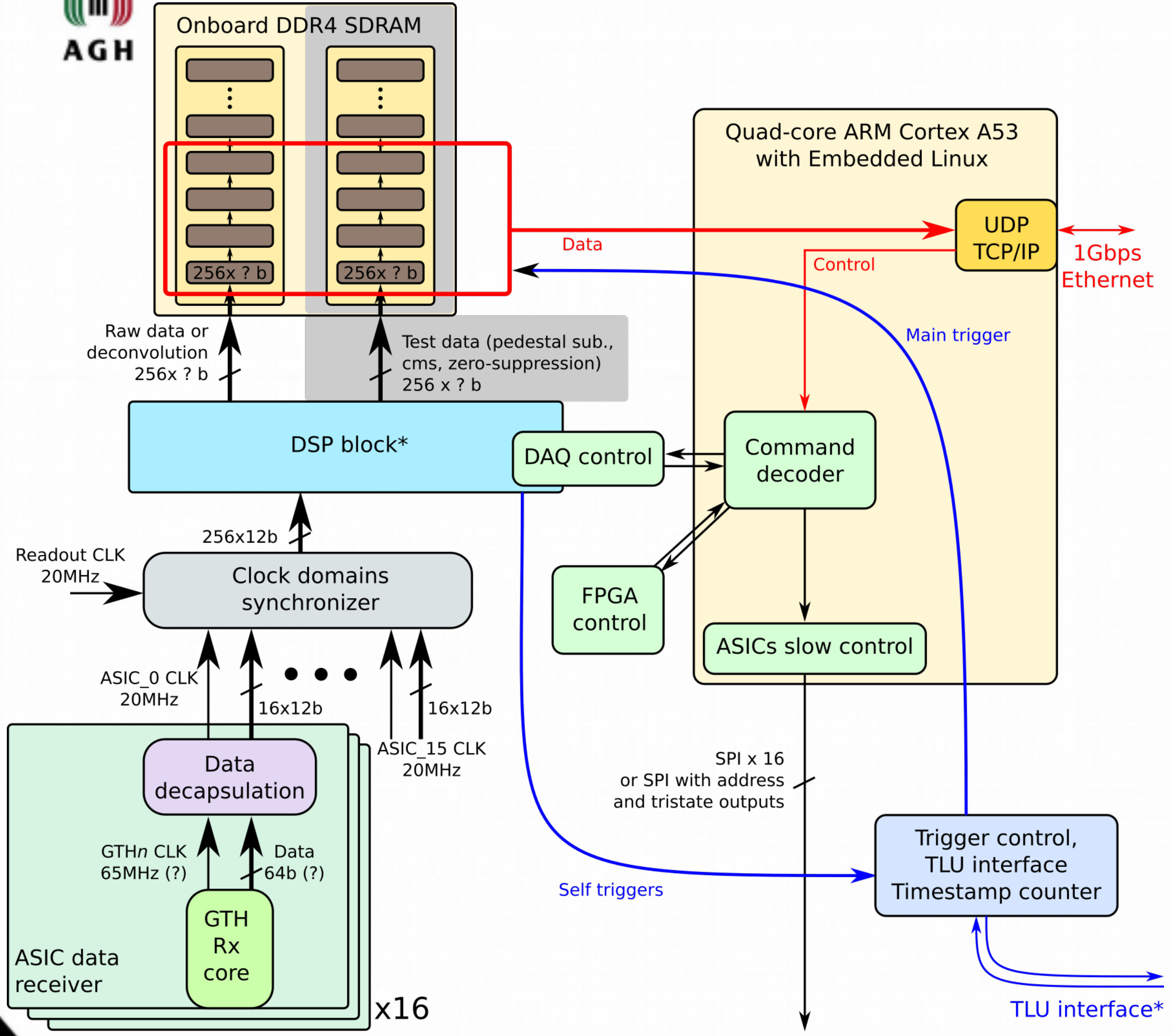
- Hardware and firmware status

Proposed DAQ architecture



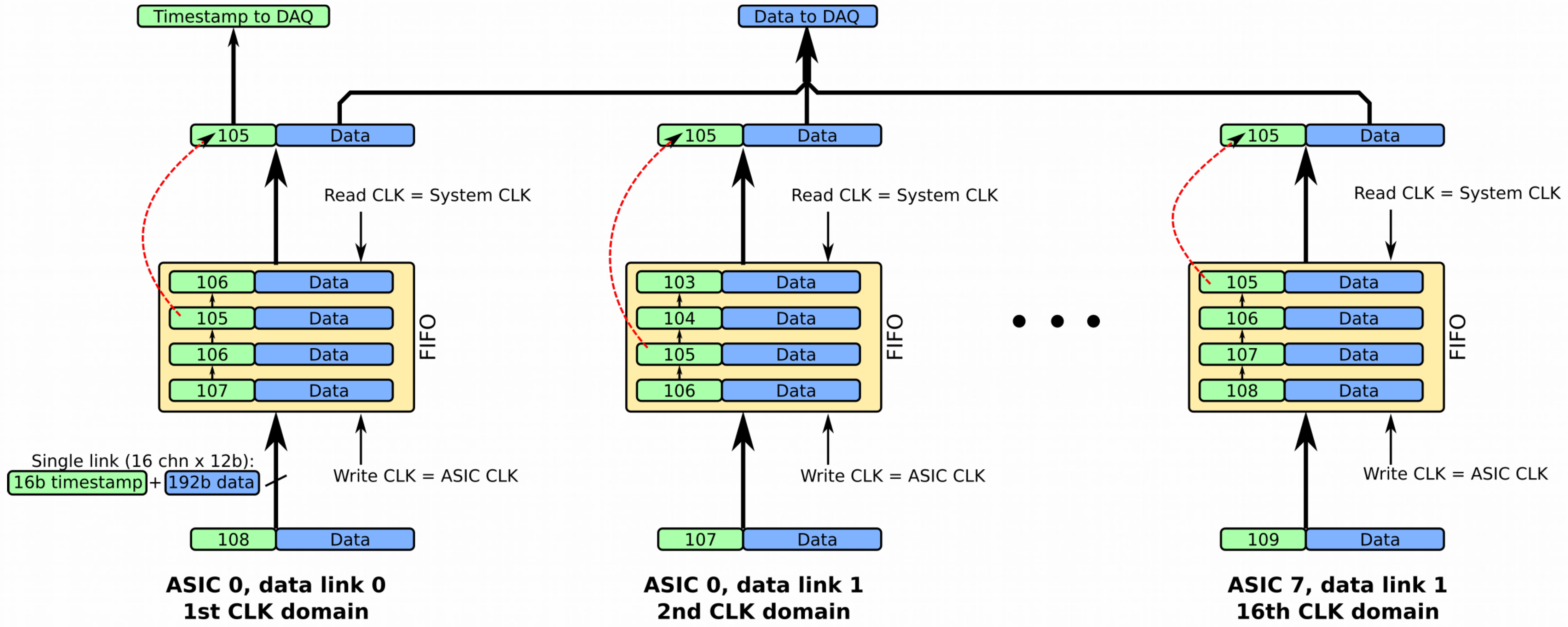
- 8 FLAME ASICs / plane = 256 channels = 16 data links (2 links per ASIC)
- New Trenz Electronic modules with Zynq UltraScale+ FPGAs available from the end of this year.
- 16 GTH transceivers / FPGA = 1 FPGA / plane
- Integrated ARM + embedded linux = 1Gbps Ethernet “for free”
- Simple Ethernet switch used as data concentrator
- One drawback – TLU (trigger) interface and timestamp synchronization not so straightforward...

Proposed DAQ architecture



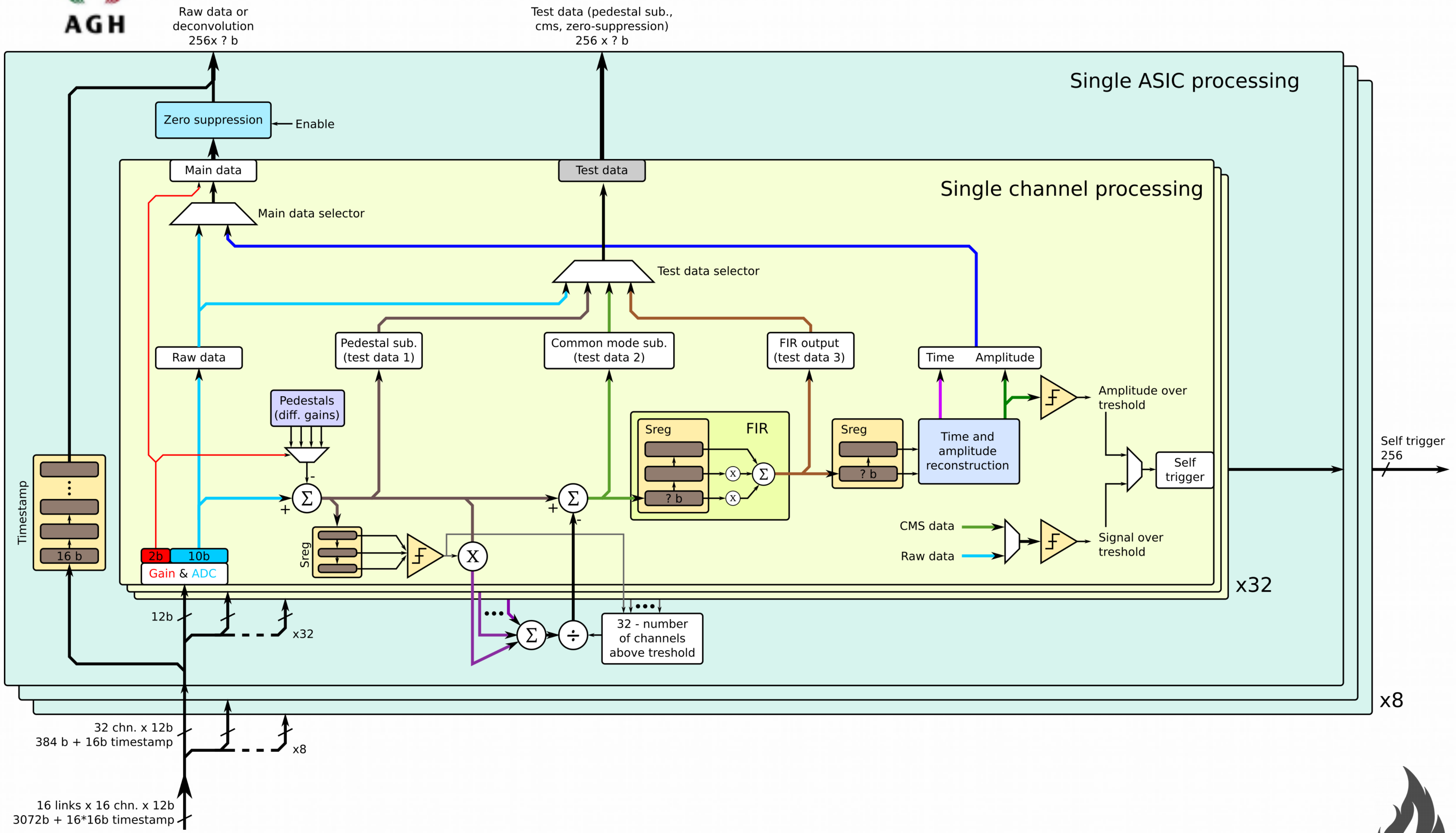
- Data from 8 ASICs (16 links) received by GTH transceivers and decapsulated
- Clock domains (16 receivers = 16 domains) synchronized with main CLK (see next)
- DSP (pedestal, cm subtraction, FIR, deconvolution, ZS)
- Data feed from FPGA logic into onboard RAM
- On trigger data read out by ARM and send out through 1 Gbps ethernet
- DAQ and ASICs slow control – by software on ARM (linux)

DAQ - Clock domains synchronizer



- Clock domains synchronizer combines samples with the same timestamp and synchronizes clock phases
- If one ASIC / data link is dead, the synchronizer should build incomplete sample and inform DAQ that one data channel is missing and should not be processed, especially in the CMS procedure

DAQ – DSP scheme



Raw data or deconvolution 256x ? b
Test data (pedestal sub., cms, zero-suppression) 256 x ? b
Single ASIC processing
Zero suppression
Enable
Main data
Test data
Single channel processing
Main data selector
Test data selector
Raw data
Pedestal sub. (test data 1)
Common mode sub. (test data 2)
FIR output (test data 3)
Time
Amplitude
Pedestals (diff. gains)
Sreg
FIR
? b
Sreg
? b
Time and amplitude reconstruction
Amplitude over treshold
Self trigger
Self trigger 256
CMS data
Raw data
Signal over treshold
Timestamp
16 b
2b
10b
Gain & ADC
Sreg
X
x32
12b
x32
32 - number of channels above treshold
x8
32 chn. x 12b 384 b + 16b timestamp
x8
16 links x 16 chn. x 12b 3072b + 16*16b timestamp

DAQ – DSP scheme

DSP scheme:

1) Pedestal subtraction

2) Common mode subtraction:
   - a) Signal detection (only channels without signal taken into CMS procedure):

     At least *three* consecutive samples > *cms_threshold*
   - b) Sum of channels without signal: – *from each ASIC* or *from whole plane*
   - c) Subtraction of CM (sum / no. of channel without signal) from all channels

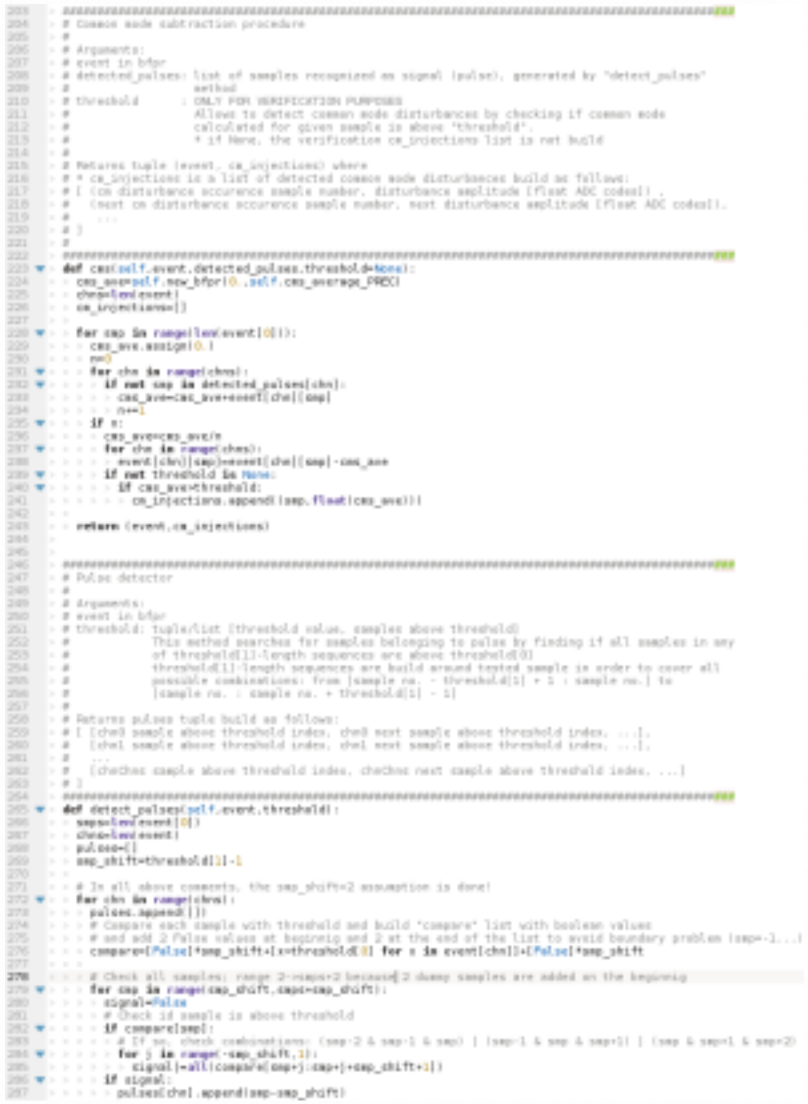
3) FIR (deconvolution filter)

4) Signal reconstruction:
   - a) Detection of signal FIR output samples – one or two samples > *fir_threshold*
   - b) *Combination the FIR signal detection with CMS signal detection – not implemented nor tested yet*
   - c) Reconstruction of amplitude and time – gives zero suppression for free

DSP simulation software

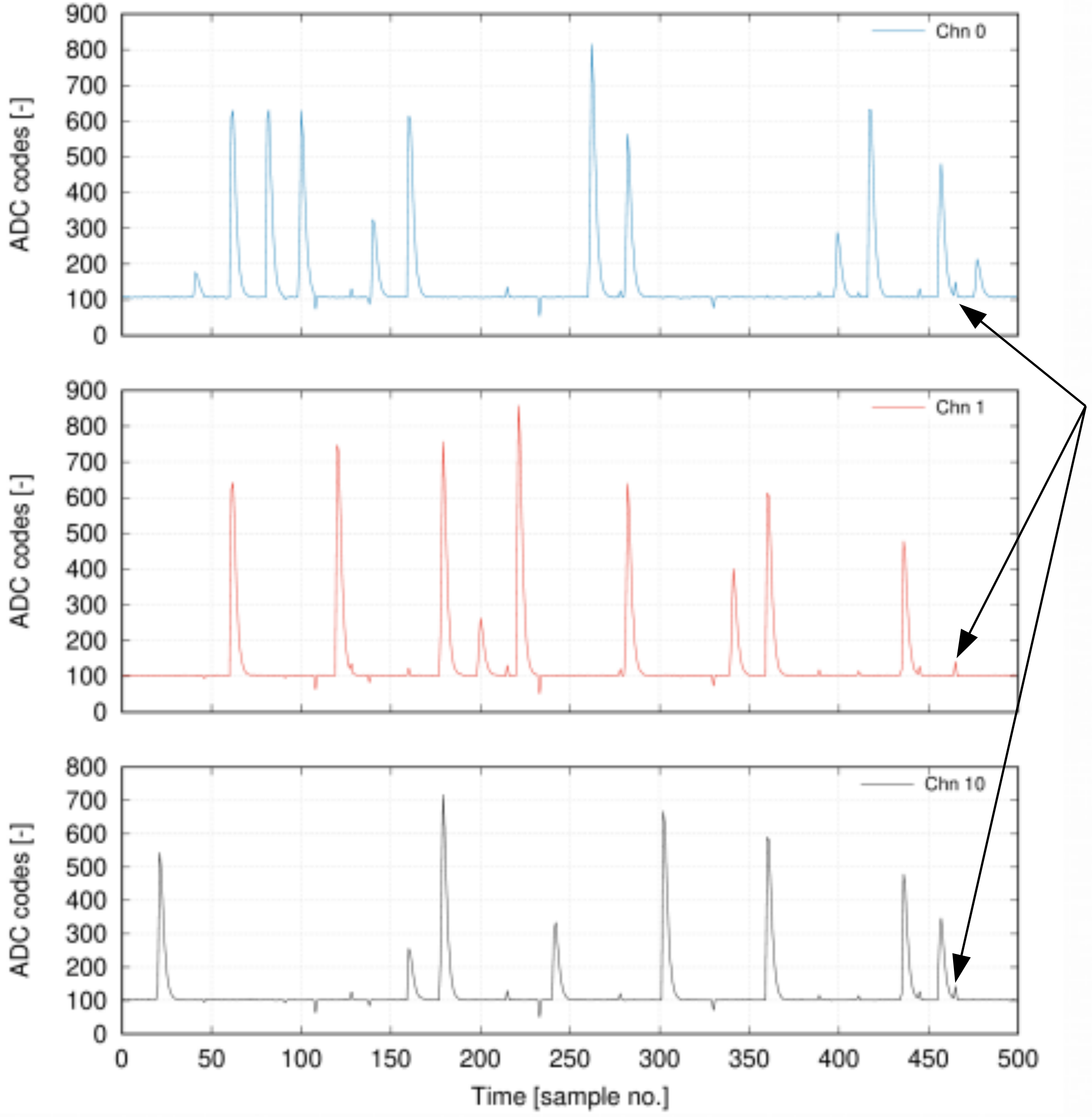
Python-based software simulating whole ASIC and DAQ chain is almost done:

- FLAME data generator is done
    - Generates “real” data – pedestal with noise, randomly distributed CM disturbances and randomly generated CR-RC pulses
- DAQ based on real binary fixed point arithmetic (with overflow supervision) is done
    - Pedestal subtraction
    - Signal detection for CMS (this is a little tricky…)
    - Common mode subtraction (CMS)
    - FIR (deconvolution filter)
    - Signal detection in FIR output samples for amplitude reconstruction (gives also zero suppression)
    - Amplitude and time reconstruction
- Verification class done
- Simulations of complete chain done
- Some improvements (signal detection procedure) still possible



```
################################################################################
# Common mode subtraction procedure
#
# Arguments:
# event in bfpr
# detected_pulses: list of samples recognized as signal (pulse), generated by "detect_pulses"
#                  method
# threshold      : ONLY FOR VERIFICATION PURPOSES
#                  Allows to detect common mode disturbances by checking if common mode
#                  calculated for given sample is above 'threshold'.
#                  * if None, the verification cm_injections list is not build
#
# Returns tuple (event, cm_injections) where
# * cm_injections is a list of detected common mode disturbances build as follows:
# [ (cm disturbance occurence sample number, disturbance amplitude [float ADC codes]) ,
#   (next cm disturbance occurence sample number, next disturbance amplitude [float ADC codes]),
#   ...
# ]
#
################################################################################
def cms(self,event,detected_pulses,threshold=None):
    cms_ave=self.new_bfpr(0.,self.cms_average_PREC)
    chns=len(event)
    cm_injections=[]

    for smp in range(len(event[0])):
        cms_ave.assign(0.)
        n=0
        for chn in range(chns):
            if not smp in detected_pulses[chn]:
                cms_ave=cms_ave+event[chn][smp]
                n+=1
        if n:
            cms_ave=cms_ave/n
            for chn in range(chns):
                event[chn][smp]=event[chn][smp]-cms_ave
            if not threshold is None:
                if cms_ave>threshold:
                    cm_injections.append((smp,float(cms_ave)))

    return (event,cm_injections)


################################################################################
# Pulse detector
#
# Arguments:
# event in bfpr
# threshold: tuple/list (threshold value, samples above threshold)
#            This method searches for samples belonging to pulse by finding if all samples in any
#            of threshold[1]-length sequences are above threshold[0]
#            threshold[1]-length sequences are build around tested sample in order to cover all
#            possible combinations: from [sample no. - threshold[1] + 1 : sample no.] to
#            [sample no. : sample no. + threshold[1] - 1]
#
# Returns pulses tuple build as follows:
# [ [ch0 sample above threshold index, ch0 next sample above threshold index, ...],
#   [ch1 sample above threshold index, ch1 next sample above threshold index, ...],
#   ...
#   [chnChns sample above threshold index, chnChns next sample above threshold index, ...]
# ]
################################################################################
def detect_pulses(self,event,threshold):
    smps=len(event[0])
    chns=len(event)
    pulses=[]
    smp_shift=threshold[1]-1

    # In all above comments, the smp_shift=2 assumption is done!
    for chn in range(chns):
        pulses.append([])
        # Compare each sample with threshold and build "compare" list with boolean values
        # and add 2 False values at beginnig and 2 at the end of the list to avoid boundary problem (smp=-1...)
        compare=[False]*smp_shift+[x>threshold[0] for x in event[chn]]+[False]*smp_shift

        # Check all samples; range 2->smps+2 because 2 dummy samples are added on the beginnig
        for smp in range(smp_shift,smps+smp_shift):
            signal=False
            # Check id sample is above threshold
            if compare[smp]:
                # If so, check combinations: (smp-2 & smp-1 & smp) | (smp-1 & smp & smp+1) | (smp & smp+1 & smp+2)
                for j in range(-smp_shift,1):
                    signal|=all(compare[smp+j:smp+j+smp_shift+1])
            if signal:
                pulses[chn].append(smp-smp_shift)
```

DSP simulation software

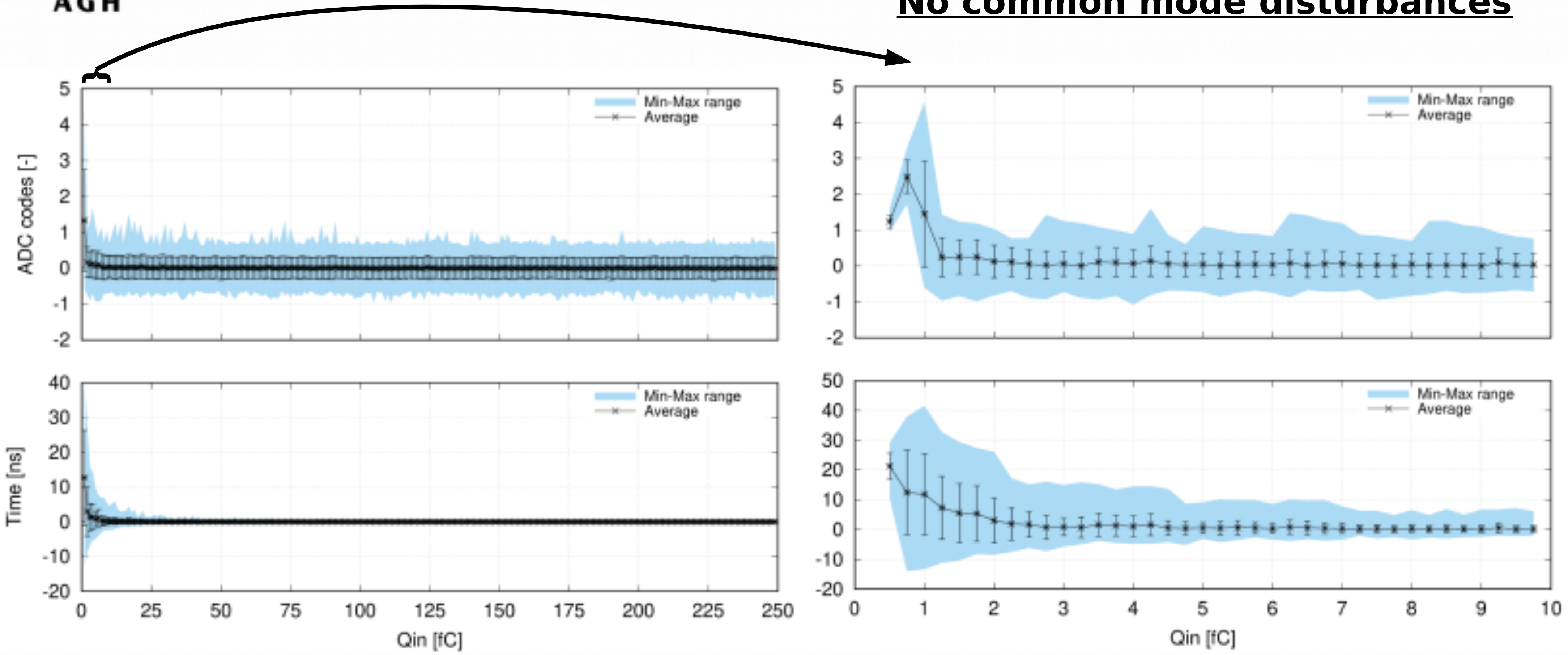


Example of FLAME data generator output with:

- Random charge injection *(Landau distribution not implemented)*
- Random CMS events (amplitude and time)
- Electronic noise (not seen in this scale)

DSP simulation results

No common mode disturbances

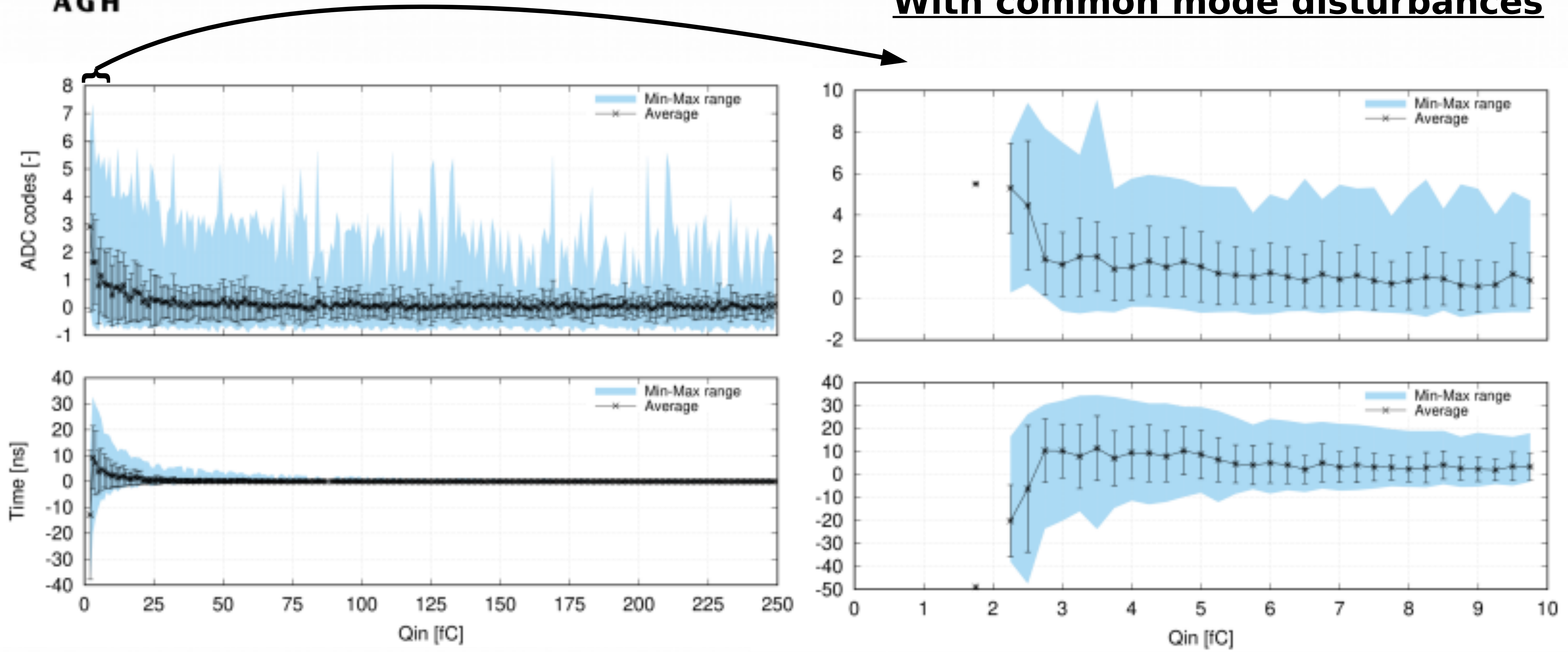


- Top plot: (generated pulse amplitude – reconstructed pulse amplitude)
- Bottom plot: (generated time of arrival – reconstructed time of arrival)

- Very good amplitude reconstruction for Q > 1 fC: **MPV ≃ 0 LSB, σ < 0.5 LSB**
- Time reconstruction with MPV ≃ **0.5 ns, σ < 1.5 ns**, but only for Q > 5 fC

DSP simulation results

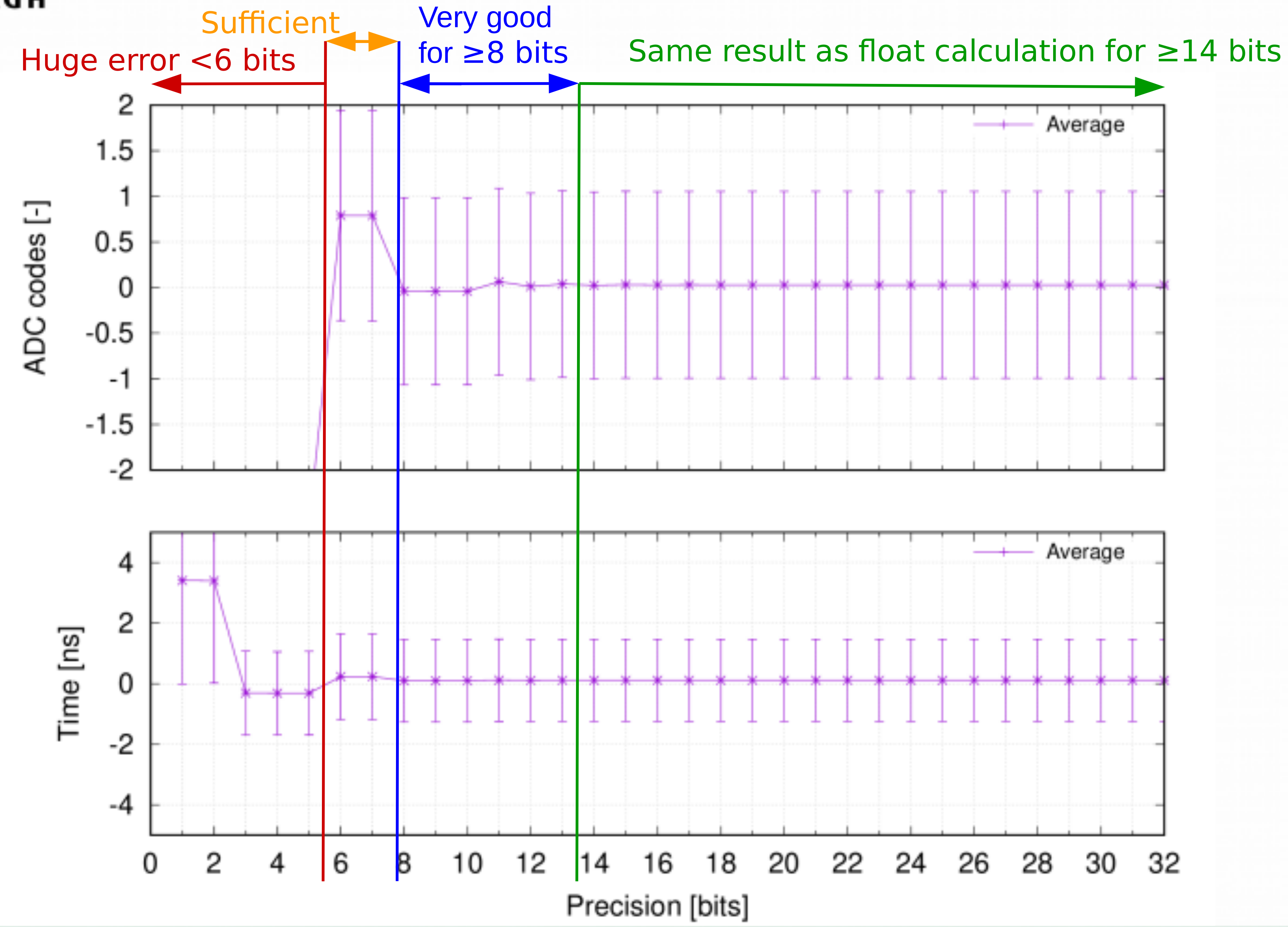
With common mode disturbances



- Still good amplitude reconstruction: **MPV ≃ -0.5 LSB, σ < 1.25 LSB**, but some pulses not reconstructed – pulse recognition logic needs improvements...
- Time reconstruction with **MPV ≃ 2.5 ns, σ < 5 ns** for smaller charges, improves for larger ones
- Reconstruction efficiency and accuracy strongly dependent on thresholds for cms and fir signal detection – needs to be tuned to real system CM disturbances

DSP simulation results

Example of fixed point precision simulation – fir coefficients



Outline

- FLAME ASIC status
- DAQ scheme and verification of DSP

➔ Hardware and firmware status

Hardware – FPGA status

FPGA modules

TE0808-04-09EG-1EE

- Two modules already delivered (thanks Hans!)
- Three more has been already ordered (by AGH-UST, we expect delivery ~next one/two weeks)
- We have/will have 5 modules → 5 sensor planes
- More can be ordered on ~May
- **How many more we want / need?**

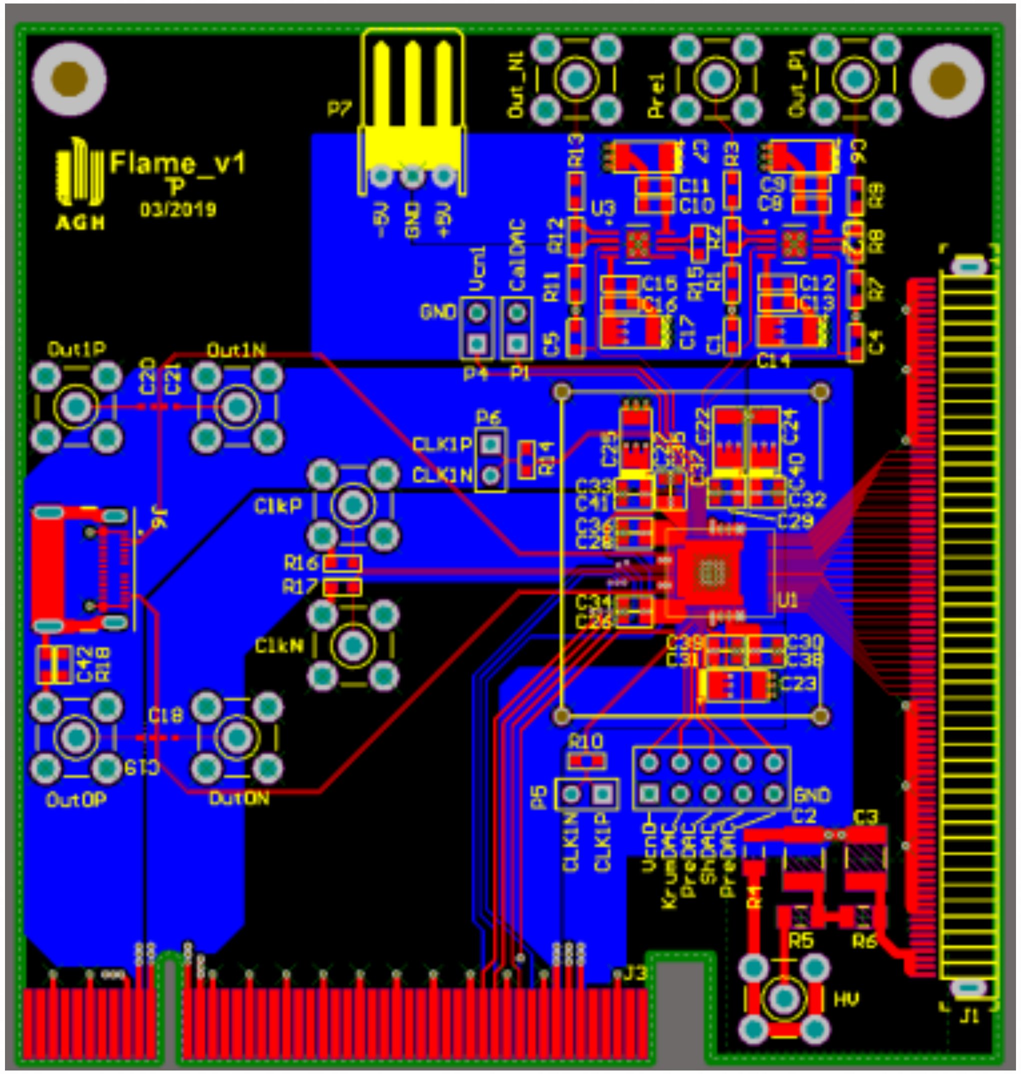
Baseboard

TEBF0808-04

- Baseboard only for firmware early development – we need only one
- It is already delivered (once again thanks to Hans!)

Hardware – PCB status

1) Test board for single FLAME ASIC – designed, but not yet sent for fabrication



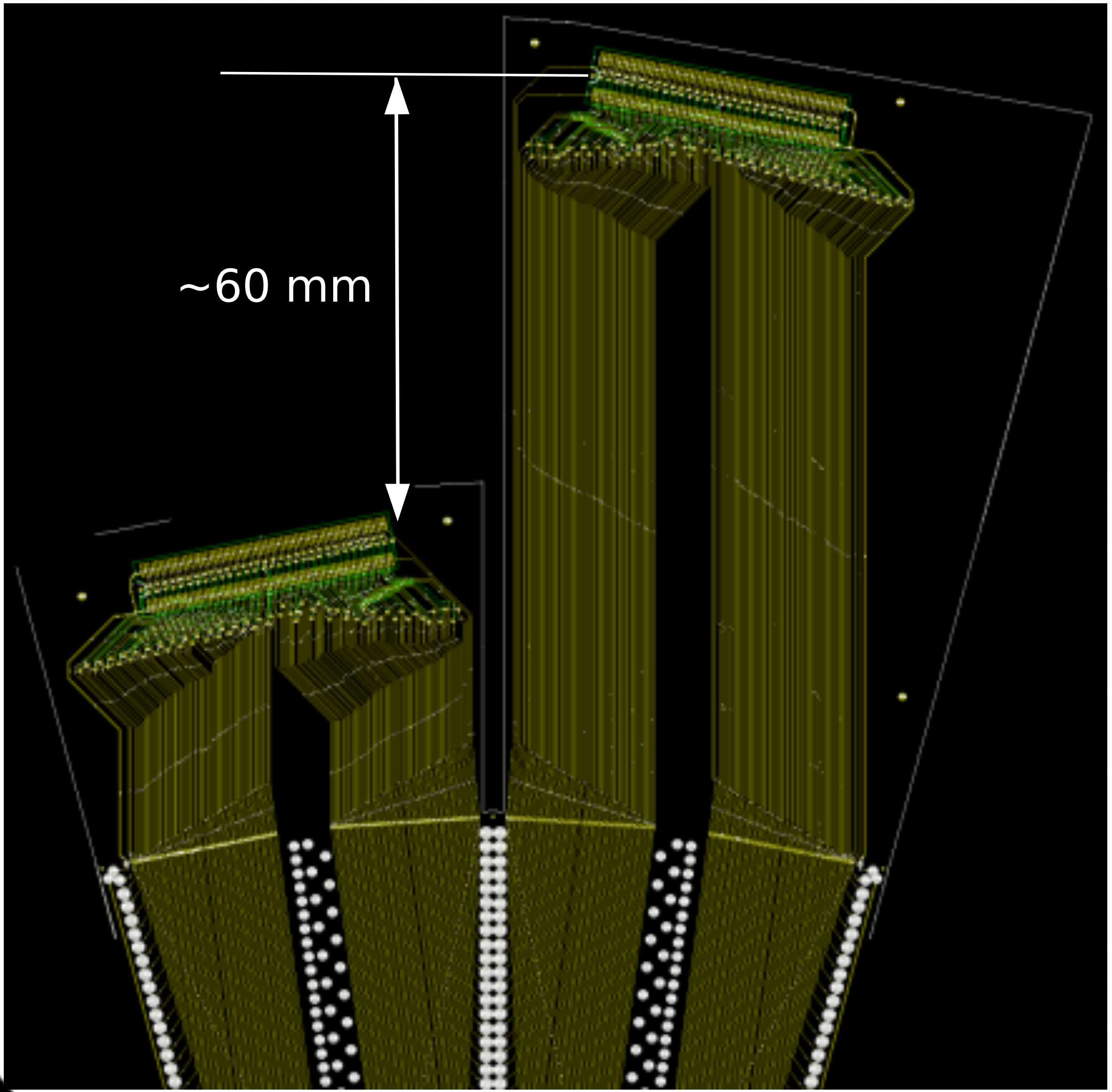
2) Readout board for testbeam – not started, some decisions needed – see next slide

This board should be designed soon, but fabricated ***after*** *FLAME verification on single ASIC board.*

3) Motherboard for FPGA farm – no started, should be designed, fabricated and tested as soon as possible (after short tests of FPGA module on Trenz basebord).
I expect some bugs here due to lack of experience...

Hardware – readout PCB

New sensor fanout



- Current fanout – 128 channels with shorter tail and 128 with longer
- Two possible solutions for readout PCB:

  1) Two boards / plane: 4 ASICs, (128 channels) each

  2) One L-shaped board with 8 ASICs, with short input tracks for right side and long input tracks for left side

- For: powering scheme (CM injections!), digital signal integrity, mechanical assembly, wiring – single 8-ASICs PCB is significantly better
- For sensor to front-end input routing two boards seems to be better

- Can we assume that longer tracks for left side on L-shaped 8-ASICs PCB will equalize the total capacitance?
- ***Which scheme should we used?***

Firmware status

1) FLAME data receiver – done by IFJ Krakow
   Different FPGA family (Artix-7) used, have to be ported to Zynq UltraScale+

   The design done by IFJ Krakow will be extremely useful during single ASIC commissioning and veryfication

2) Clock domain synchronizer – some work done by JINR Dubna, but status is unknown – assuming as not done…

3) DSP – not started yet (expect Python-based model, still requiring some tweaks)

4) TLU interface, trigger and timestamp synchronization, slow control, etc. – not started yet

5) Debian linux at Zynq – almost done, some small issues remains

6) Data transfer FPGA → Linux → PC (heart of the DAQ) – not started yet

7) Control and DAQ software – not started yet

There is a lot of work – any help is very welcome

Summary

1) FLAME ASIC sent for fabrication – we expect ~240 ASICs on beginning of June

2) PCB for single ASIC designed and ready for fabrication.
   Readout PCB need some input from the collaboration:
   a) One L-shaped 8-ASICs board or two smaller 4-ASICs boards per sensor?
   b) Input from Tel-Aviv really needed: schematic of the fanout, fanout connector model, HV connector type

   Motherboard for FPGA farm awaits on Trenz baseboard test results.

3) Python-based DSP model done, needs some tweaks but results are promising

4) DAQ firmware started (linux), but **lot** of work needed – can anyone contribute?

Thank you for the attention