Algorithms for Threshold Dispersion Minimization of the CHIPIX65 Asynchronous Front-End

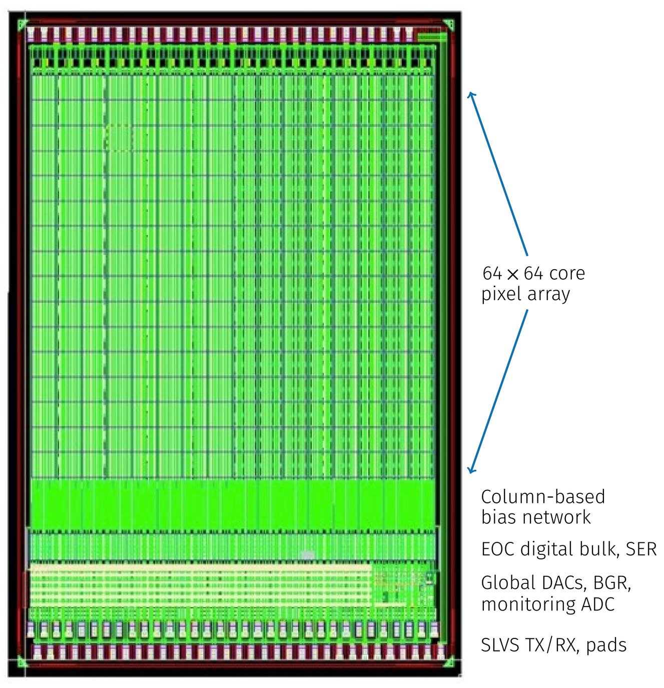
M. Sonzogni[1,2], F. De Canio[1,2], L. Gaioni[1,2], M. Manghisoni[1,2], L. Ratti[2,3], V. Re[1,2], G. Traversi[1,2]

University of Bergamo[1], INFN Pavia[2] & University of Pavia[3]

HL-LHC Features & RD53 Requirements

- Extraordinarily high levels of radiation and particle rates will be attained in the high-luminosity upgrades of the Large Hadron Collider experiments. The **instantaneous luminosity will reach $5 \cdot 10^{34}$ cm^{-2}s^{-1}**, while **the expected level of ionizing radiation** for the innermost layers of the tracking pixel detectors **is 1 Grad in 10 years**.
- **The CERN RD53** collaboration was founded in 2013 to investigate new technologies and architectures for future readout chips. The Italian Institute for Nuclear Physics (INFN), through **CHIPIX65 Project**, has been actively contributing to RD53, which served the purpose of testing design solutions by means of a small-scale demonstrator called CHIPIX65_FE0.
- One of the main requirements of the new HL-LHC readout chip is to **guarantee sub-1000 electrons stable threshold operation**. In order to satisfy this requirement, **minimization of threshold dispersion** is fundamental. In this work we present four threshold tuning algorithms and discuss their performance in terms of threshold dispersion and operation time together with some of the characterization results of the CHIPIX65 asynchronous front-end.

CHIPIX65 Demonstrator



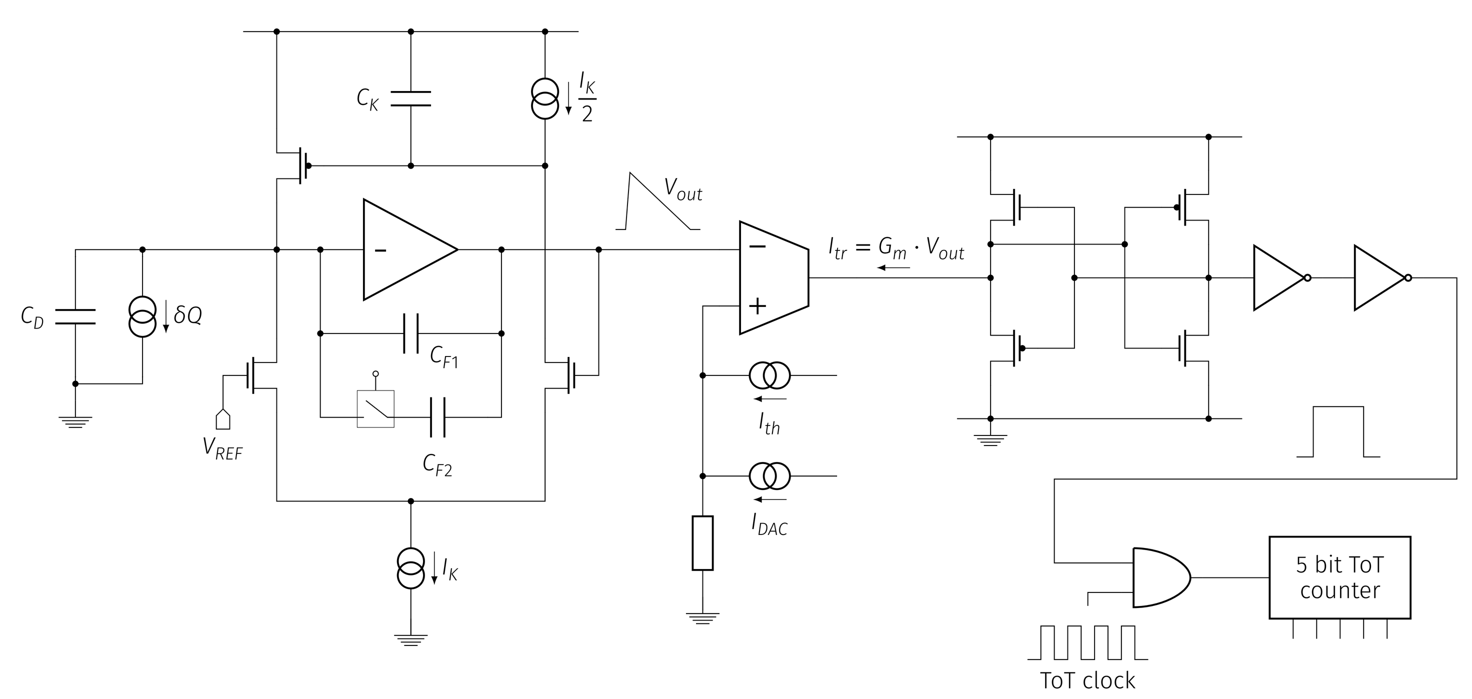
Features:

- Designed in **65 nm CMOS technology** (chosen by RD53 collaboration for HL-LHC upgrades, which allows the designer to integrate a large number of in-pixel analog and digital functions)
- **64 × 64 pixels matrix**, divided into two 32 × 64 submatrices with 50 μm pitch, embodying two analog front-end architectures based on synchronous and asynchronous hit discriminator interfacing with a common digital readout and configuration scheme
- On-chip bias network and monitoring
- Chip configuration based on SPI protocol
- Usage of CERN I/O library
- Dimensions: 3.5 mm × 5.5 mm

Design flow:

- Digital-On-Top chip assembly and Top-down hierarchical flow
- Pixel matrix composed of 16 × 16 pixel regions
- A pixel region (4 × 4 pixels) contains the digital architecture and the analog FEs

Asynchronous Analog Front-end

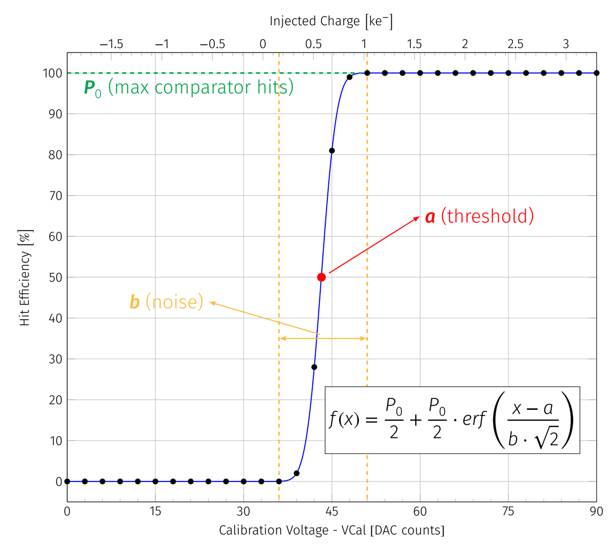


This front-end is designed according to RD53A requirements: capability of operating at a **minimum threshold lower than 1000 e$^-$**, **power consumption not exceeding 5 μW** in a silicon area close to 1000 μm^2 and a mean equivalent noise charge (ENC) lower than 126 e$^-$. Some of its features:

- Single amplification stage for minimum power dissipation
- Charge sensitive amplifier with a gain stage featuring a folded cascode architecture, with two local feedback networks designed to increase the output small signal resistance
- Krummenacher feedback, providing a constant current discharge of the feedback capacitor $C_{F1,2}$, to comply with the expected large increase in the detector leakage current after irradiation
- High speed, low power current comparator
- In-pixel threshold trimming DAC, based on a 4-bit binary weighted architecture, generating the current I_{DAC} which adds to the current I_{th} (equal for each pixel in the matrix) in order to obtain the same threshold in all matrix pixels (**threshold dispersion minimization**)
- Relatively slow time-over-threshold (ToT) clock – 40 MHz
- 5 bit, dual edge ToT counter – 400 ns maximum ToT
- 30 ke$^-$ maximum input charge, 450 mV preamplifier output dynamic range
- Selectable gain and recovery current
- Overall current consumption: 4 μA

Algorithms for threshold dispersion minimization

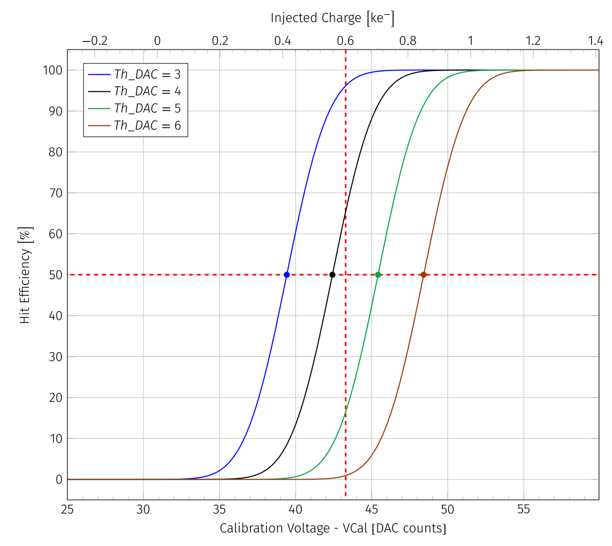
Introduction



In this section 4 algorithms for threshold minimization are described. Goal: all matrix pixels must have a threshold as close as possible to a threshold selected by the user. In the figure, the error function curve used to fit hit efficiency data is reported: **a** represents the pixel threshold. Variables used in the algorithms:

- **nb**: Number of DAC bits
- **np**: Number of pixels
- `sth`: Pixels threshold, selected by the user
- `ncomb`: Number of DAC combinations (2^{nb})
- `th_dac`: Array of DAC combinations to obtain `sth`. Each element of the array corresponds to one pixel.
- `nscan`: Number of charge scans

1 - Complete Charge Scan

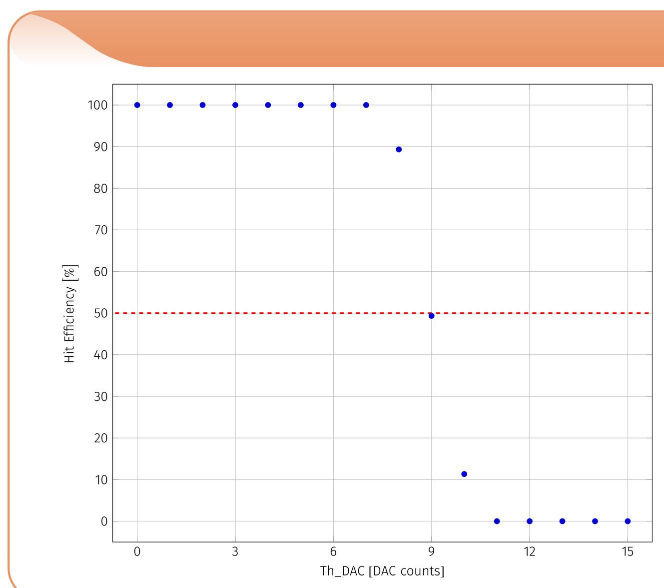


Given one pixel, this algorithm executes a complete charge scan for each DAC code. Injected charges ranges from 0 e$^-$ to 1500 e$^-$. Data obtained from these scans, must be fitted with the function described in the Introduction in order to obtain the threshold for all pixels and DAC code combinations. For each pixel, the DAC code which sets the threshold as close as possible to `sth` is selected and saved in the array `th_dac`.

```
def complete_charge_scan(sth):
    th = zeros(np, ncomb)
    for p in 0..np:
        for c in 0..ncomb:
            P0, a, b = fit(f, SCAN[p, c, :])
            th[p, c] = a
        th_dac[p] = argmin(th[p, :] - sth)
```

- Complexity: $O(np \cdot ncomb \cdot nscan)$, $\Omega(np \cdot ncomb \cdot nscan)$
- Pros: Pixels which don't work can be easily detected; complete scans
- Cons: Slowest algorithm; fit function computationally expensive

2 - Fixed Charge Scan



For each DAC code, a fixed charge equivalent to the threshold `sth`, is injected in each pixel n times. The DAC code which allows to obtain a hit occupancy closer to 50 % is selected for that pixel.

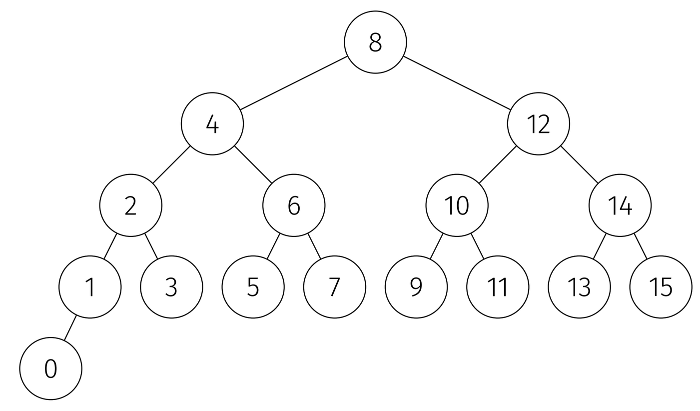
```
def fixed_charge_scan(sth):
    th = zeros(np, ncomb)
    for p in 0..np:
        for c in 0..ncomb:
            hit = SCAN[p, c, sth]
            th[p, c] = hit
        th_dac[p] = argmin(th[p, :] - sth)
```

- Complexity: $O(np \cdot ncomb)$, $\Omega(np \cdot ncomb)$
- Pros: Faster than algorithm 1, faulty pixels can be detected, easy implementation, optimal threshold dispersion minimization
- Cons: Slower than algorithms 3 and 4

3 - Root to Leaf Binary Tree Scan

This algorithm is based on the binary search tree shown in figure below. Starting from a DAC code equal to **ncomb**/2, a fixed charge `sth` is injected in each pixel. At each step, the number of comparator hits is stored as an array element: this number defines the direction of the next branch, either up (≥ 50%) or down (< 50%). Lastly, the DAC code associated to a hit occupancy closest to 50 % is selected.

```
def binary_tree_scan(sth, sel_hit):
    for p in 0..np:
        min_comb, max_comb, comb = 0, ncomb, ncomb / 2
        combs = list(), hits = list()
        while True:
            P0, a, b = SCAN[p, comb, sth]
            hits.append(a), combs.append(comb)
            if comb != 1:
                if a > sel_hit:
                    min_comb = comb
                else:
                    max_comb = comb
                comb = floor((max_comb + min_comb) / 2)
                if comb == min_comb or comb == max_comb:
                    break
            else:
                max_comb, comb = 0, 0
        th_dac[p] = combs[argmin(hits[:] - sel_hit)]
```

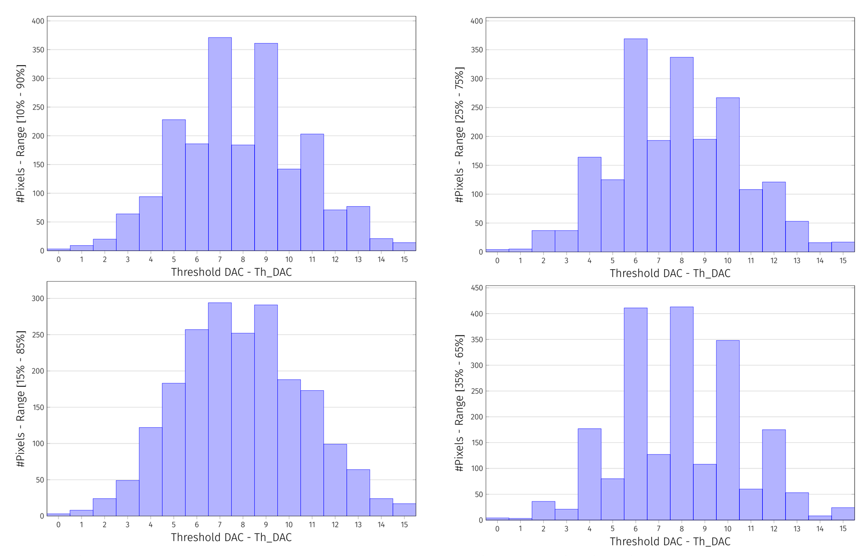


- Complexity: $O(np \cdot (\lfloor \log_2(ncomb) \rfloor + 1))$, $\Omega(np \cdot \lfloor \log_2(ncomb) \rfloor)$
- Pros: Fast algorithm; if all pixels work correctly, results like in algorithm 1 or 2 are obtained
- Cons: More difficult to implement; faulty pixels are hard to be detected

4 - Bounded Binary Tree Scan

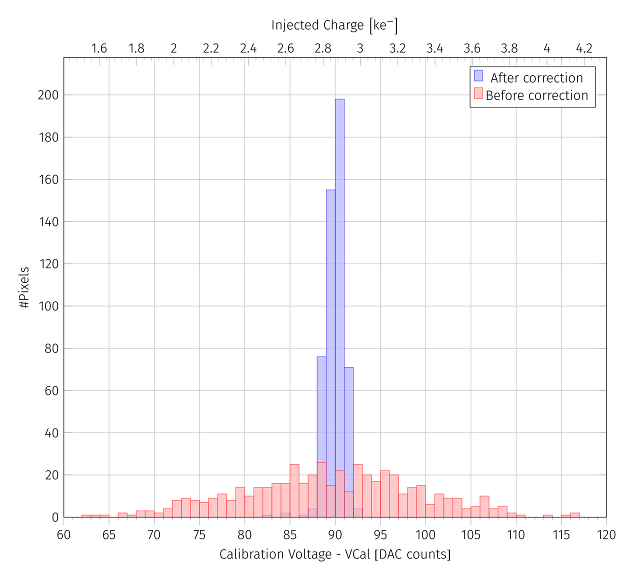
This algorithm is based on a binary search tree like algorithm 3. The user defines two values: `smin` and `smax`. Like in the last algorithm it start from a DAC code equivalent to **ncomb**/2 and a fixed charge `sth` is injected in each pixel. At each step, if the comparator hits are more than `smax%` the DAC code has to be greater (up in the binary tree), if they are less than `smin%` smaller (down in the binary tree), while if they are between `smin%` and `smax%` that DAC code is selected and the scan of next pixel starts.

```
def range_tree_scan(smin, smax, sth):
    for p in 0..np:
        max_comb = ncomb
        min_comb = 0
        th_dac[p] = ncomb / 2
        while max_comb - min_comb > 2 or th_dac[p] == 1:
            P0, a, b = SCAN[p, th_dac[p], sth]
            if smin <= a and a <= smax:
                break
            else if a > smax:
                min_comb = th_dac[p]
            else:
                max_comb = th_dac[p]
            th_dac[p] = floor((max_comb + min_comb) / 2)
            if th_dac[p] == min_comb:
                break
```



- Complexity: $O(np \cdot \lfloor \log_2(ncomb) \rfloor)$, $\Omega(np)$
- Pros: Fastest algorithm
- Cons: Different couples `smin` and `smax` have to be tested in order to find the one which minimize the threshold dispersion (if the range is too wide tree leafs will never be reached, while if it's too narrow the algorithm will stop in the internal nodes, as shown in figures above); the obtained results could be not optimal.

Results & Conclusions



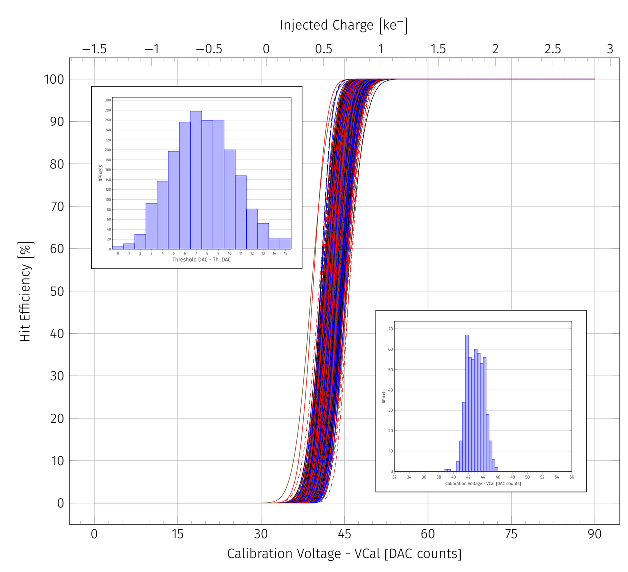
Different measurements, mainly concerned with threshold dispersion minimization have been performed on three chips. One of these has been tested after exposure to total ionizing doses (TID) up to 630 Mrad (SiO_2). Threshold dispersion at 600 e$^-$, obtained with these four tuning algorithms, is reported in the table below (expressed in electrons). The picture on the left represents Chip 1 threshold distribution at high threshold before and after tuning (with Algorithm 2).

Threshold dispersion	Alg 1	Alg 2	Alg 3	Alg 4
Chip 1	54.1	54.41	54.96	N/A
Chip 2	N/A	N/A	65.79	67.99
Irradiated Chip	81.1	N/A	N/A	73.56*

* Algorithm 4 tests have been carried out two months after irradiation with a final dose of 630 Mrad

Figures on the right show the threshold dispersion minimization case obtained with algorithm 2 on chip 1. All the s-curves for the tuned matrix are reported in the main plot. All pixels DAC codes (Gaussian Curve) and threshold histograms are shown in the inner plots. A threshold dispersion of 55 e$^-$ has been obtained with a medium threshold of 590 e$^-$, starting from a initial threshold dispersion of 450 e$^-$.

In conclusion, test results confirm that threshold dispersion performance of the CHIPIX65 continuous-time analog front-end is compatible with the specifications set by the RD53 consortium, in view of the development of 65 nm CMOS readout pixel chips for the innermost detector layers of ATLAS and CMS. The algorithms 1 and 4 are not recommended; the first is computationally too slow, while threshold dispersion minimization in the second one could be not optimal. The algorithm 2 can detect faulty pixels but is slower than 3, which, instead, could give optimal results faster if all pixels work correctly.



Topical Workshop on Electronics for Particle Physics, TWEPP 2018, September 17-21, Antwerp, Belgium

mauro.sonzogni@unibg.it