THRESHOLD MINIMIZATION

Detailed instructions to use the procedure described in this talk
https://indico.cern.ch/event/304181/contribution/3/material/slides/0.pdf
are summarized in this document.

The procedure starts from the Vana settings obtained from the Vana calibration (see the POS user guide for details). *Before starting the threshold minimization, make sure you are working with proper settings (Baseline, AOH bias and gain, TBMUB, etc…) and that you have a good AddressLevel calibration.*

The procedure consists of the following steps:

1) do a VcThrCalDelFIFO3 calibration to get the initial VcThr,CalDel working point
2) do a VcThrCalibration targeting 50 VCal to go closer to the noise limit for each ROC, but still have threshold sufficiently high to be efficient
3) do a series of PixelAlive to get an initial guess of thresholds
4) do a PixelAlive with all pixels enabled and check for failing ROCs
5) if needed, refine the thresholds obtained from step 3) doing few more iterations with PixelAlive with all pixels enabled on the ROCs failing step 4)
6) do a SCurve to measure the thresholds with the final VcThr settings
7) check the quality of the SCurve results

Details on each step are given below.

1) VcThrCalDelFIFO3

The calib.dat file is:

```
Mode: ThresholdCalDelay
Rows:
10 | 20
Cols:
10 | 20
VcalLow
Scan: VcThr 0 255 8
Scan: CalDel 0 255 8
Set: Vcal 150
Repeat: 10
ToCalibrate:
all
```

Note: we inject 150 VCal (VcalLow) as we want to work with the same charge inject for PixelAlive

2) **VcThrCalibration with target 50 VCal**

The calib.dat file is:

```
Mode: VcThrCalibration
Rows:
10 | 20
Cols:
10 | 20
VcalLow
Scan: WBC 154 156 1
Scan: VcThr 0 200 2
Set: Vcal 50
Repeat: 25
ToCalibrate:
+ all
```

3) Perform various **PixelAlive** runs, changing VcThr at each iteration depending on if the ROC pass or fail the PixelAlive. Details on how this is implemented are shown in this talk:
https://indico.cern.ch/event/304181/contribution/3/material/slides/0.pdf
In this first round of iterations, the PixelAlive runs are done with one pixel enabled at a time. A ROC is considered failing PixelAlive if the number of pixels with efficiency < 100% is above 10 (the threshold of 10 pixels can be changed).

The calib.dat file for this PixelAlive is:

```
Mode: PixelAlive
Parameters:
ROCReset yes
Rows:
0 | 16 | 32 | 48 | 64 |
1 | 17 | 33 | 49 | 65 |
2 | 18 | 34 | 50 | 66 |
3 | 19 | 35 | 51 | 67 |
4 | 20 | 36 | 52 | 68 |
5 | 21 | 37 | 53 | 69 |
6 | 22 | 38 | 54 | 70 |
7 | 23 | 39 | 55 | 71 |
8 | 24 | 40 | 56 | 72 |
9 | 25 | 41 | 57 | 73 |
10 | 26 | 42 | 58 | 74 |
11 | 27 | 43 | 59 | 75 |
12 | 28 | 44 | 60 | 76 |
13 | 29 | 45 | 61 | 77 |
14 | 30 | 46 | 62 | 78 |
15 | 31 | 47 | 63 | 79
Cols:
0 | 13 | 26 | 39 |
1 | 14 | 27 | 40 |
2 | 15 | 28 | 41 |
3 | 16 | 29 | 42 |
4 | 17 | 30 | 43 |
5 | 18 | 31 | 44 |
6 | 19 | 32 | 45 |
7 | 20 | 33 | 46 |
```

```
8 | 21 | 34 | 47 |
9 | 22 | 35 | 48 |
10 | 23 | 36 | 49 |
11 | 24 | 37 | 50 |
12 | 25 | 38 | 51
VcalLow
Set: Vcal 150
SetRelative: VcThr 0
Repeat:
10
ToCalibrate:
all
```

Note: 150 VCal is the suggested value in the POS user guide in order to be above threshold for all Pixels but not too large to add unnecessary cross talk.

4) Once all the ROCs pass step 3) and we have an initial estimate of the thresholds, *take a PixelAlive run with ALL pixels enabled (use the Default mask).*

The calib.dat for PixelAlive with all pixels enabled is:

```
Mode: PixelAlive
Parameters:
ROCReset yes
ScanMode default
Rows:
0 | 16 | 32 | 48 | 64 |
1 | 17 | 33 | 49 | 65 |
2 | 18 | 34 | 50 | 66 |
3 | 19 | 35 | 51 | 67 |
4 | 20 | 36 | 52 | 68 |
5 | 21 | 37 | 53 | 69 |
6 | 22 | 38 | 54 | 70 |
7 | 23 | 39 | 55 | 71 |
8 | 24 | 40 | 56 | 72 |
9 | 25 | 41 | 57 | 73 |
10 | 26 | 42 | 58 | 74 |
11 | 27 | 43 | 59 | 75 |
12 | 28 | 44 | 60 | 76 |
13 | 29 | 45 | 61 | 77 |
14 | 30 | 46 | 62 | 78 |
15 | 31 | 47 | 63 | 79
Cols:
0 | 13 | 26 | 39 |
1 | 14 | 27 | 40 |
2 | 15 | 28 | 41 |
3 | 16 | 29 | 42 |
4 | 17 | 30 | 43 |
5 | 18 | 31 | 44 |
6 | 19 | 32 | 45 |
7 | 20 | 33 | 46 |
8 | 21 | 34 | 47 |
9 | 22 | 35 | 48 |
10 | 23 | 36 | 49 |
11 | 24 | 37 | 50 |
12 | 25 | 38 | 51
VcalLow
Set: Vcal 150
```

```
SetRelative: VcThr 0
Repeat:
10
ToCalibrate:
all
```

If ant ROC fails, repeat the procedure that increases the threshold with this PixelAlive configuration only on the failing ROCs (i.e the VcThr of all the ROCs that pass PixelAlive with all pixels enabled are left unchanged).

5) SCurve

Once you get the final VcThr settings for all ROCs after steps 3) and 4), take a SCurve run to measure the thresholds.
The calib.dat file for the SCurve run is:

Finally, check the quality of the SCurve run by looking at:

- Distribution of the thresholds: low or high tails or bumps are signs of possible problems
- RMS of the thresholds: low or high tails are signs of possible problems
- Number of pixels for each ROC: the number of expected pixels is 81, if Npixels <79 that ROC must be investigated.

Tools

To make the various iterations of PixelAlive more automatic, a script is available here:

https://github.com/martinamalberti/PIXEL/blob/master/Calibration/scripts/FastPixelAliveAnalysisForThr.py

It must be run from the directory AnalysisTools/test.

After you have taken a PixelAlive run, this script runs the analysis of the run, checks the efficiency in each ROC and makes a list of ROCs for which the number of pixels with efficiency <100% is lower than maximum number of bad pixels (defaults maxDeadPixels=10). It then creates the new dac settings containing updated values of VcThr for each ROC, which can be used at the next iteration.

The script requires minimum 3 options: the run number, the run key and the number of iteration (start counting from 0). So for example, to run it at your first iteration, you would do:

```
python FastPixelAliveAnalysisForThr.py –r <run> -k <key> -i 0
```

Other options allow you to chose the maximum number of bad pixels per ROC, the

output file names, the steps for lowering/increasing thresholds, if run on BPix only, FPix only, etc… Use:

```
python FastPixelAliveAnalysisForThr.py --help
```

to see all of them.

At each iteration N, a file called failed_N.txt containing the list of ROCs that fail the PixelAlive and a file called delta_N.txt which keeps track of the shifts in VcThr for each ROC are created.

After the scripts has finished to run, kill all the Supervisors to make sure that the new dacs are loaded and used in the new run.

Comments/tips:

- You expect that at iteration 0, all ROCs pass PixelAlive as the threshold should be sufficiently high. It may happen that a few ROCs fail. *You should understand why before going on*. A possible way to investigate them is to take one PixelAlive run with higher threshold using the SetRelative option (e.g. SetRelative VcThr -10 or -20). If it was a problem of too low threshold, it should go away when using higher thresholds. If it was a defect in the chip, it will not. In this case you can exclude the corresponding ROCs from the procedure by using the --exclude option. You must list the ROCs you want to exclude in a file (let's say, called tobeexcluded.txt) and pass it to the script, like:

  ```
  Python FastPixelAliveAnalysisForThr.py –r <run> -k <key> -i 0 –e tobeexcluded.txt
  ```

  When you run the PixelAlive run with VcThr-10 or -20, don't consider it part of the iteration procedure (i.e. don't update the dac settings after analyzing it!), thus analyze it simply by doing :

  ```
  ./bin/linux/x86_64_slc5/PixelAnalysis.exe configuration/PixelAliveAnalysis.xml <runNumber>
  ```

  and compare the efficiency maps of the concerned ROCs with those of the previous run.

  You can then re-start the iterative procedure (thus analyze the runs with the python script and updating the VcThr at each step).
  At the end, when you are done with the majority of the ROCs, you may consider to repeat the procedure of threshold minimization only on the ROCs that you excluded allowing a larger number of bad pixels.

- to check how many ROCs at iteration N have reached a stable VcThr type :
  `grep " 0" delta_N.txt | wc –l`

- The procedure should converge within about 10 iterations. You will stop when

there are no more ROCs failing (except those that you possibly excluded from the procedure via the - - exclude option). It may happen that in the last iterations you find few ROCs failing even if they were stable in all previous iterations. Usually it's a matter of few more pixels (>~ 10) and you should not worry about them.