Digitization and hit reconstruction for silicon tracker in MarlinReco

S.Shulga and T.Ilicheva [*]

Joint Institute for Nuclear Research
141980 Dubna, Moscow reg., Russia

Fr.Scorina Gomel State University
245280 Gomel, Republic of Belarus

The program *SiliconDigi*, implementing a new *Marlin* processors for digitization and clustering in silicon tracker of LDC, is presented. Processors include member of class *Detector* which contains digitizer and clusterizer. Vector of samples of class *DetUnit* (simplest detector unit) are initialized by using *GEAR* interface. Digitizer and clusterizer take detector units to transform collection of simulated hits to collection of raw data and reconstructed hits for each detector unit. Current code contains barrel subdetectors. Codes of classes *Digitizer* and *Clusterizer* are taken from CMS software.

1 Design of package *SiliconDigi*

Simplest object of digitization is readout unit. The detector is a set of identical readout units. The detector should comprise process of digitization and clustering, receiving from the outside and making for external use corresponding hit information. It means that codes of digitizer and clusterizer should be parts of class describing detector.

To have flexible package it is useful to separate persistent part of the program (*MARLIN* [2] processors) from developing part. Developing part of codes describes detector including vector of detector units, interface to *GEAR* [2] initializing detector units, digitizer and clusterizer. Detector unit contains *LCIO* [2] collections (*SimTrackerHit*, *TrackerRawData* and *TrackerHit*) which are initialized or produced by processors.

That is why package *SiliconDigi* [3] consist of three sub packages (sub package in sense of *MarlinReco* [2]): *SiDetector* (includes classes *Detector*, *DetUnit*, *SiPixelDetUnitDigitizer* and *SiTrkDetUnitClusterizer*), *SiDigi* (class *SiTrkDigiProcessor*) and *SiClustering* (class *SiTrkClusterProcessor*).

Class *Detector* is container of layers and samples of *DetUnit*. Main method of *Detector* performs initialization of *DetUnit* by using *GEAR* xml-file. Abstract base class *DetUnit* is container of simulated/raw/reconstructed and temporary hits. *DetUnit* can read/write standard *LCIO* collection of hits. *Detector* contains digitizer and clusterizer. The object to be digitized/clusterized is a sample of *DetUnit*. Codes of digitizer and clusterizer are taken from CMS software [4, 5, 6, 7, 8, 9].

First function of digitizer is finding of ionization points in detector unit. Interval between hit entry and exit points is divided in N segments by using parameter of length of segment (0.01 mm by default). This function creates ionization points which contain information about positions of ionization points and energy loss in units of number of electrons. Energy losses are defined by using Landau distribution in thin silicon layer.

[*]This work is partially supported by BMBF(Germany).

Next step of digitizer is transformation of ionization points to collection points defined at charge collection plane where sensor pixels are placed.

Number of collection points is equal number of ionization points. Two physical phenomena are simulated at this step: Lorenz drift with the fixed Lorenz angle and diffusion around the drift direction. The collection point contains the calculated value of Gaussian charge diffusions along X and Y directions ($\sigma_X \times \sigma_Y$). Each collection point is mapped by pixel map to find low and upper bounds of fired pixels. Cluster of fired pixels is defined with sizes $3\sigma_X \times 3\sigma_Y$. In each fired pixel 2-dimensional integral is calculated according to Gaussian distribution of charge for each collection point separately. Charge fractions are summarized over the collection points for given fired pixel. For all simulated hits charge fractions are summarized over the simulated hits. All fired pixels are collected in *map<int channel, double charge>* where *channel* is packed 2-dimensional pixel number, *charge* is full charge from all simulated hits in the event. Map of signals is a last output result of digitizer. Digitizer translates this map to the sample of *DetUnit*. *DetUnit* modifies signal map adding noises and killing some channels according to the inefficiency. Method *DetUnit::add_noise* adds two types of noises. First one calculates noises in each hit pixels around zero by Gaussian distribution with σ_{noise} given by user parameter. After that noise charge is added to the hit pixel. Secondly, it calculates noise in not-hit pixels by so called noiser. The noises are ruled by two parameters: noise RMS in units of electrons and threshold in terms of σ_{noise}.

Method of pixel inefficiency kills some pixels, double columns of pixels or full readout chips. Two parameters are used to find readout chip inefficiency: sizes of readout chips along X and Y direction in units of number of pixels. One can introduce different inefficiencies for different layers.

The clustering is performed on a matrix with size which is equal the size of the pixel detector. Each cell contains the ADC count of the corresponding pixel. The search starts from seed pixels, i.e. pixels with sufficiently large amplitudes. Clusters are set of neighbor pixels including pixels which touched by corners.

2 Thresholds and efficiency of hit reconstruction

It is convenient to use σ_{noise} as an unit of collected charge. Then it is possible to define admissible thresholds, expressing them in terms of σ_{noise}. In the beginning we find a threshold defining a signal in the channel (pixel). Clearly, that this threshold should make few σ_{noise}, to avoid a plenty of false fired pixels. Simultaneously big threshold reduces efficiency of registration of hits.

To define a pixel threshold, we shall construct dependence of efficiency from value of pixel threshold, setting the threshold of seed pixels and a threshold of the cluster charge as equal to zero (seed pixel is a pixel from which the clustering is started). Noises also are switched off. From figure 1 (a) we see, that without essential decrease in efficiency of registration of a hit it is possible to choose pixel threshold not more than $4\sigma_{noise}$.

After that to build second plot (dependence of efficiency from seed threshold) the pixel threshold is fixed to $4\sigma_{noise}$, cluster threshold is set to zero. Seed threshold can be more than pixel threshold. Maximal value of seed threshold will be set $5\sigma_{noise}$.

Last plot is dependence of efficiency from cluster threshold with minimal pixel and seed thresholds which were found by previous pictures. Cluster threshold can be more then seed threshold. Maximal cluster threshold is restricted by decreased efficiency and will be set not more than $6\sigma_{noise}$.

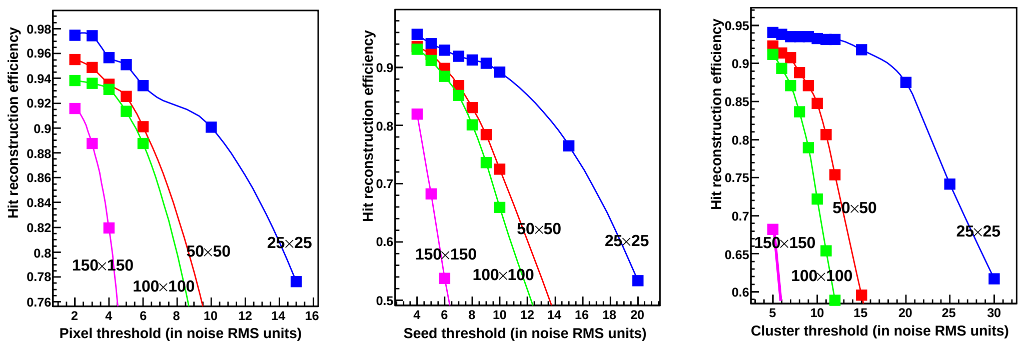

Figure 1: Efficiency of hit reconstruction as function of pixel, seed and cluster thresholds for pixel sizes 25×25, 50×50, 100×100 and $150 \times 150 \mu m^2$.

3 Conclusion

Pixel size, μm^2	Reconstructed hits, %	True hits, %
25×25	94.3	93.5
50×50	89.8	89.6
100×100	86.9	86.8
150×150	54.0	53.9

Table 1: Hit reconstruction efficiency for different sizes of pixels. Pixel, seed and cluster thresholds equal $4\sigma_{noise}$, $5\sigma_{noise}$ and $6\sigma_{noise}$ correspondingly.

Classes *Detector*, *DetUnit* and *BarrelDetUnit* are developed to use in digitization and clustering processors for silicon tracker in framework of *MarlinReco*. Class *Detector* includes pixel digitizer and pixel clusterizer for rectangular detector units. Pixel, seed and cluster thresholds are investigated. The table 1 contains hit reconstruction efficiencies for different sizes of pixels with thresholds which were found above.

References

[1] Slides: `http://ilcagenda.linearcollider.org/contributionDisplay.py?contribId=311&sessionId=74&confId=1296`

[2] `http://ilcsoft.desy.de/portal/software_packages`

[3] `http://www-zeuthen.desy.de/lc-cgi-bin/cvsweb.cgi/SiliconDigi/?cvsroot=marlinreco;only_with_tag=v00-04`

[4] S.Cucciarelli, D.Kotlinsky, T.Todorov, CMS Note 2002/049

[5] S.Cucciarelli, D.Kotlinsky, CMS IN 2004/014

[6] D.Kotlinski, Pixel Software Workshop, 11-15/01/07 (CMS)

[7] G.Giorgiu, Pixel Workshop, 01/12/07 (CMS)

[8] D.Kotlinski, Pixel Software meeting, 19/09/06 (CMS)

[9] S.Shulga, ILC software and Tools Workshop, LAL-Orsay, 2-4 May, 2007, `http://ilcagenda.linearcollider.org/conferenceDisplay.py?confId=1446`