L1 Tracklet-Based Track Finding and Fitting

Anders Ryd (Anders.Ryd@cornell.edu)
Peter Wittich (wittich@cornell.edu)
Charles Strohman (crs5@cornell.edu)
Jorge Chaves (jec429@cornell.edu)
Louise Skinnari (louise.skinnari@cern.ch)
September 26, 2014

Abstract

This note describes the use of tracklets for track finding in the barrel+endcap geometry. The algorithm is described and results from simulations of the performance are presented. A description of the system architecture and the data volumes from the simulation is presented. Then a detailed description of the firmware implementation is given.

Contents

1 Introduction

In this white paper we describe the 'tracklet' approach to the Phase 2 L1 track trigger. The advantage of the tracklet approach is its reliance on commercial FPGA technology, rather than custom ASICs. This will allow us to trivially incorporate advances in FPGA development between now and the time where the final system needs to be constructed. The design is also fully pipelined for an efficient use of resources and is expected to result in an acceptable latency between the bunch-crossing time and the availability of L1 tracks.

The tracklet approach relies on *local reconstruction. In CMS the term local reconstruction has a specific meaning, for the tracker is means turing hits into clusters and hits. We should probably avoid this term in order to avoid confusions?* Tracklets are pairs of stubs from neighboring layers of the tracker that are linked to form candidate tracks and are the seeds of the tracking algorithm. These seeds are propagated to adjacent layers (both outside-in and inside-out), where hits consistent with the trajectory of a high-p_T (¿2 GeV) track are added. Efficiency is kept high by using seeds from all pairs of adjacent layers and by propagating both towards and away from the collision point. A linearized track fit provides the final 3-D track parameters (p_T, η, ϕ_0, z_0 and optionally d_0). A *ghost busting* phase eliminates duplicate tracks.

In this paper we describe the tracklet algorithm in detail and how this algorithm is implemented on an FPGA. We describe a system architecture that shows the data flow and building blocks needed for implementation and simulation of data volumes. The algorithm is designed for the five-disk barrel+endcap geometry.

The tracklet approach provides several advantages

- Algorithm can be implemented using FPGAs and does not require the development of specialized ASICs.
- We can take advantage of the rapid improvement of FPGAs.
- The implementation using FPGAs allows naturally for a pipelined implementation. This means that less time multiplexing and less processing units (boards or FPGAs) are required.
- The algorithm is simple to emulate as it involves calculations that are easily run in software and only small lookup tables.
- Demonstration of the algorithm can be accomplished with hardware that exists today.

2 Barrel+endcap geometry

The layout of the five disk barrel+endcap geometry is shown in Fig. 1. Tracklets are formed from pairs of stubs in different layers; the challenge with the tracklet based approach in this geometry is the relatively large separation between the layers in the barrel. This gives

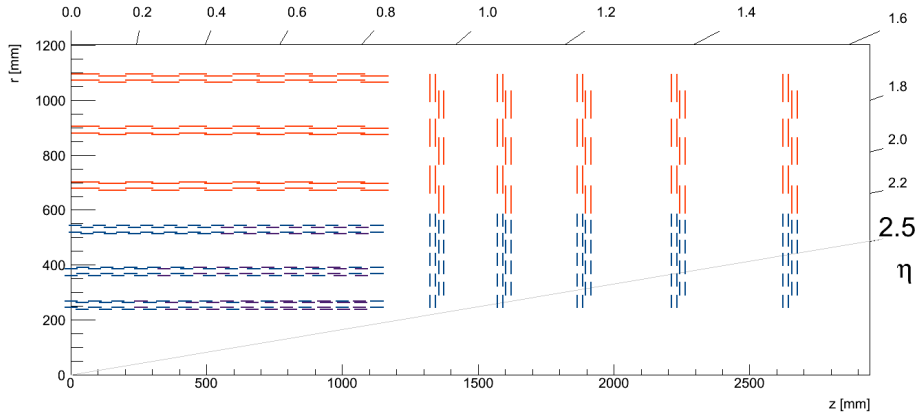


Figure 1: The geometry of the five disk barrel+endcap geometry. The modules shown in blue are the pixel-strip modules and the red modules are the strip-strip modules. (Note that the studies done uses the version of the BE5D geometry where the outer disks are not extending radially inward as far as indicated in this figure. This will be updated as soon as the new geometry is available.)

raise to combinatorics that will challenge the hardware implementation. With randomly distributed stubs in two layers the probability for two stubs to accidentally form a tracklet grows quadratically with the layer separation.

3 Algorithm

The track finding and track fitting described here consists of four major steps:

1. Tracklet finding - form pairs of stubs consistent with coming from a track with $|p_{\mathrm{T}}| > 2$ GeV and $|z_0| < 15$ cm.

2. Stub matching - project tracklets to other layers and match with stubs.

3. Track fitting - perform linearized track fit.

4. Duplicate removal - remove tracks that are found multiple times.

Each of these steps will be described in detail in this paper; here a brief description is given.

In the first step, pairs of stubs, one stub from each of two neighboring layers or disks, are combined to form tracklets. The two stubs and the luminous region constraint in the x-y plane is used to form a trajectory. If this trajectory is consistent with being from a particle with $p_{\mathrm{T}} > 2$ GeV and $|z_0| < 15$ cm the two stubs are said to form a tracklet.

The trajectory determined by the stubs in the tracklet is then projected to the other layers and disks in the detector and a search of matching stubs is performed. The list of stubs that are matched to a trajectory are then fit to obtain the optimal set of track parameters. Since

the same track can be found multiple times from seeding in more than one pair of layers, the last step involves removing duplicates.

3.0.1 Tracklet parameter calculation

We will describe the L1 tracks using four or five track paramteres. In the track finding step the tracks are constrained to come from the IP in the x-y plane such that the impact parameter, d_0 is fixed to zero. However, for the track fit step we can also fit for this parameter. In addition to d_0 the other parameters are ρ^{-1}, the inverse of the radius of curvature; ϕ_0, the track direction at the point of closest approach to the IP; t, the tangent of the 'dip' angel which is related to η by $t = \frac{1}{2}(e^{\eta} - e^{-\eta})) = \sinh\eta$; and z_0, which is the z position of the track at the point of closest approach to the IP.

Having found two stubs that are candidates for forming a tracklet the track parameters needs to be calculated. The assumption here is that each stub is given by its cylindrical coordinates, r, ϕ, and z. From Eq. 1 we have that the inverse of the radius of curvature is given by

$$\rho^{-1} = \frac{2\sin\Delta\phi}{\Delta}$$

where $\Delta\phi = \phi_2 - \phi_1$ is the difference of azimuthal angle between the two hits and Δ is the distance between the two hits in the x-y plane. We can write

$$\Delta^2 = r_1^2 + r_2^2 - 2r_1r_2\cos\Delta\phi = (r_2 - r_1)^2 + 2r_1r_2(1 - \cos\Delta\phi)$$

this then gives

$$\Delta = (r_2 - r_1)\sqrt{1 + \frac{2r_1r_2}{(r_2 - r_1)^2}(1 - \cos\Delta\phi)}.$$

Putting this together we get

$$\rho^{-1} = \frac{2\sin\Delta\phi}{r_2 - r_1}\left(1 + \frac{2r_1r_2}{(r_2 - r_1)^2}(1 - \cos\Delta\phi)\right)^{-1/2}$$

3.1 Tracklet finding

Pairs of stubs are tried to see if they satisfy the criteria for forming tracklets. First the $r - \phi$ view is examined. The two stubs and the origin are used to determine the inverse of the radius of curvature, ρ^{-1}, using Eq. 1. The inverse of the radius of curvature is required to satisfy

$$|\rho^{-1}| < 0.0057 \text{ cm}^{-1},$$

this corresponds to $p_T > 2.0$ GeV. Next the $r - z$ view is examined, and if the track projects to have

$$|z_0| < 15 \text{ cm}$$

the tracklet is accepted. The p_{T} of the tracklet is required to be consistent with p_{T} of the two stubs.

3.2 Tracklet parameter calculation

This section will describe how the track parameters for the tracklets are calculated in the hardware implementation. The expressions, like Eq. 3.0.1, involves evaluation of trigonometric functions and square roots. As we need to implement this on FPGAs we need to rewrite these equations in ways so that we can express this in terms of integer operations such as additions and multiplications. We do this in two steps. First we write the expressin in a form where we can taylor expand them and then we translate these expansions to integer calculations.

To calculate the tracklet track parameteres we start from Eq. 3.0.1 which is an exact expression and will be our starting point for making approximations that allow for a simpler implementation. First we introduce

$$\delta \equiv \frac{r_1 r_2}{(r_2 - r_1)^2}(1 - \cos\Delta\phi)$$

and Taylor expand the square root

$$\rho^{-1} = \frac{2\sin\Delta\phi}{r_2 - r_1}\left(1 - \delta + \frac{3}{2}\delta^2 - \frac{5}{2}\delta^3 + \frac{35}{32}\delta^4 + ...\right)$$

To understand how many terms we need to keep here we need to estimate the magnitude of δ. First we note that $1 - \cos\Delta\phi \approx \frac{1}{2}\Delta\phi^2$, but we also note that $\Delta\phi/\Delta r \approx 1/2\rho$ such that

$$\delta \approx \frac{r_1 r_2}{8\rho^2}.$$

Since we want to track particles with momenta as low as 2 GeV, with $\rho \approx 1.7$ m, we can estimate that $\delta < 0.8 \times 1/(8 \times 1.7^2) = 0.035$. This gives a maximum correction for the first order correction of 3.5%. The second and third order terms are 0.18% and 0.01%. For now we will keep two correction terms, though we can probably reduce this to one. Next we look at the approximation of

$$\sin\Delta\phi = \Delta\phi(1 - \frac{\Delta\phi^2}{6} + \frac{\Delta\phi^4}{120} + ...)$$

To estimate the size of $\Delta\phi$ we use $\Delta\phi \approx \frac{\Delta r}{2\rho} = 0.045$. This means that the first correction term is 0.03%, and we can drop this. This similarly holds for

$$\delta = \frac{r_1 r_2}{(r_2 - r_1)^2}(1 - \cos\Delta\phi) = \frac{r_1 r_2}{2(r_2 - r_1)^2}\Delta\phi^2(1 + \frac{\Delta\phi^2}{12} + ...)$$

where again we can retain only the leading term. Putting this together we have

$$\delta \approx \frac{r_1 r_2}{2(r_2 - r_1)^2}\Delta\phi^2$$

and

$$\rho^{-1} = \frac{2\Delta\phi}{r_2 - r_2} \times \left(1 + \delta + \frac{3}{2}\delta^2\right)$$

or

$$\Delta^{-1} = \frac{1}{r_2 - r_2} \times \left(1 + \delta + \frac{3}{2}\delta^2\right)$$

The angle ϕ_0 is obtained from

$$\phi_0 = \phi_1 + \arcsin\frac{r_1}{2\rho} = \phi_1 + \frac{r_1}{2\rho}\left(1 + \frac{1}{24}\frac{r_1^2}{\rho^2} + \frac{3}{1240}\frac{r_1^4}{\rho^4} + ...\right)$$

Need to keep one correction term?

To determine the track parameters in the r-z projection we use

$$z_1 - z_2 = t\rho(\psi_1 - \psi_2)$$

where $\rho(\psi_1 - \psi_2)$ is the length of the trajectory in the r-ϕ plane between the two hits. This length is given by

$$2\rho\arcsin\frac{\Delta}{2\rho} = 2\rho(\frac{\Delta}{2\rho} + \frac{\Delta^3}{48\rho^3} + ...) = \Delta(1 + \frac{\Delta^2}{24\rho^2}..).$$

The leading correction is less than 0.03% and is neglected. This then gives

$$t = \frac{z_1 - z_2}{\Delta} \times \left(1 + \frac{\Delta^2}{24\rho^2}\right)^{-1} = \frac{z_1 - z_2}{\Delta} \times \left(1 - \frac{\Delta\phi^2}{6}\right)$$

where $1/\Delta$ has been calculated earlier. Next we have

$$z_0 = z_1 - t\rho\arcsin\frac{r_1}{2\rho}$$

Though we have evaluated this arcsin above, it does not really help since we only have ρ^{-1}. The most efficient way to evaluate this is probably by Taylor expansion of the arcsin

$$z_0 = z_1 - tr_1\left(1 + \frac{1}{48}\frac{r_1^2}{\rho^2} + \frac{3}{1240}\frac{r_1^4}{\rho^4} + ...\right)$$

For the worst case when $r_1 = 0.8$ m, the first correction is about 0.5%, and the second 0.016%.

Now we can put all of this together as a step-by-step algorithm:

1. $\Delta\phi = \phi_2 - \phi_1$
2. $\Delta r = r_2 - r_1$
3. $\Delta z = z_2 - z_1$

4. $\Delta r^{-1} = 1/\Delta r$
5. $t_1 = r_1 r_2$
6. $t_2 = \Delta\phi\Delta r^{-1}$
7. $t_3 = t_2 t_2$
8. $\delta = t_1 t_3/2$
9. $t_4 = 1 + 1.5\delta$
10. $t_5 = 1 + \delta t_4$
11. $\Delta^{-1} = t_5 \Delta r^{-1}$
12. $\rho^{-1} = 2\Delta\phi\Delta^{-1}$
13. $t_6 = \Delta z \Delta^{-1}$
14. $t_7 = r_1\rho^{-1}/2$
15. $t_8 = t_7 t_7$
16. $t_9 = 1 + \frac{1}{12} t_8$
17. $t_{10} = t_7 t_9$
18. $\phi_0 = \phi_1 + t_{10}$
19. $t_{11} = t r_1$
20. $t_{12} = t_{11} t_9$
21. $z_0 = z_1 - u_2$

In Fig. 2 is a comparison between the exact calculation of the track parameters and the approximations above.

The only step here that is not trivial to implement on an FPGA is the calculation of Δr^{-1}. Since the range of Δr is limited this can be implemented efficiently as a memory lookup. Compared to the LB algorithm these calculations require approximately the same number of steps, but it is simpler in the sense that it only makes use of one memory lookup.

We also note here that this calculation works independent of whether we have hits in barrel layers or disks - or a mixture of the two. The only step where I think care has to be taken is in the calculation of Δr^{-1}, where the radial difference will be much smaller in the disks.

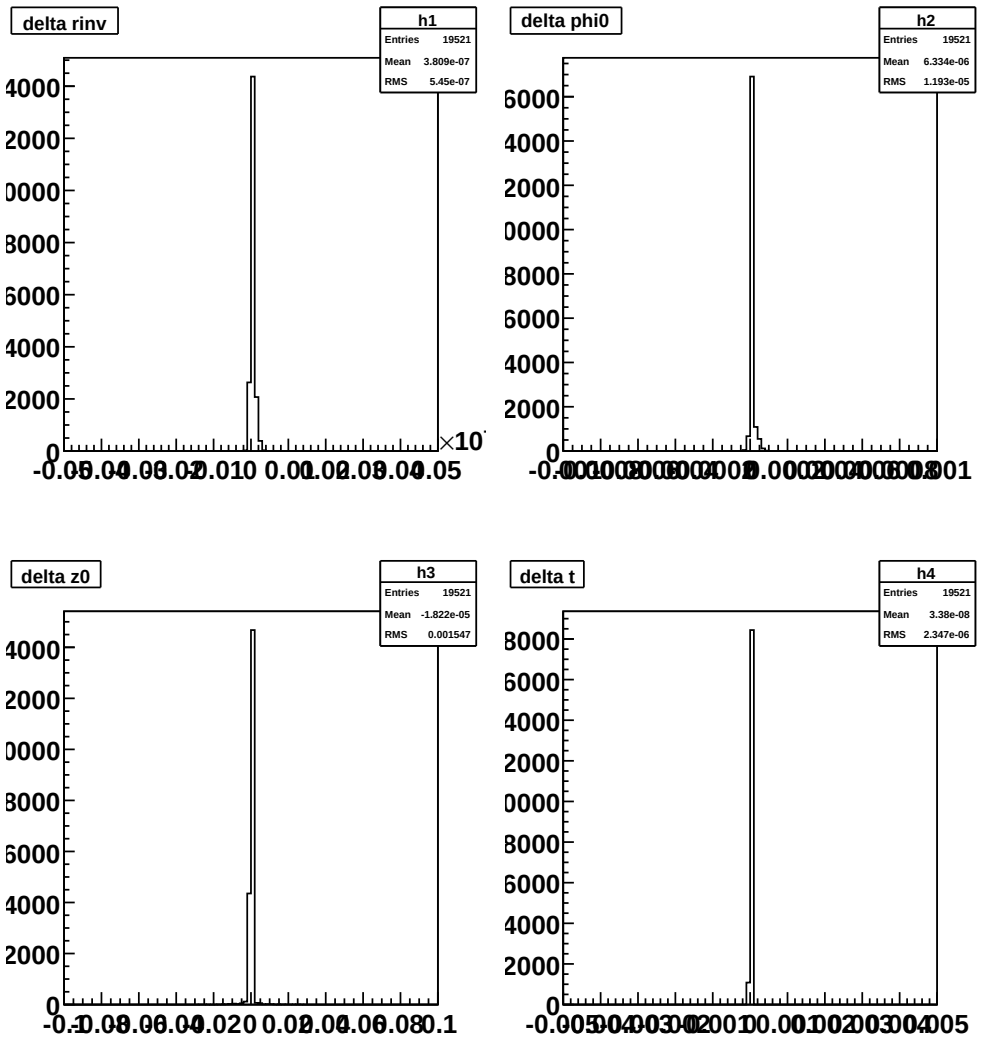


Figure 2: The difference between the exact track parameters for the tracklet and the approximations used in the integer based calculation.

3.3 Tracklet projection

After tracklets have been found and the track parameters calculated as described in the previous sections we need to project the trajectory to the other layers or disks. The calculations involved are described below. In the projection we use an average radius for projections to barrel layers or an average z position for the disks. In order to calculate a more precise projection we also evaluate the deriviatives of the projections with respect to the radius or z-position for barrel and disk hits respectively.

First we will discuss projections to a barrel layer, given by the radius. Here we need to calculate the ϕ and z position of the track.

$$\phi = \phi_0 - \arcsin\frac{r}{2\rho} = \phi_0 - \frac{r}{2\rho}\left(1 + \frac{1}{6}\frac{r^2}{(2\rho)^2} + \frac{3}{40}\frac{r^4}{(2\rho)^4} + ...\right)$$

Keeping one correction term here should be be sufficient.

$$z = z_0 + 2t\rho\arcsin\frac{r}{2\rho} = z_0 + tr\left(1 + \frac{1}{24}\frac{r^2}{\rho^2} + \frac{3}{640}\frac{r^4}{\rho^4} + ...\right)$$

The derivatives with respect to the projection radius, r, are given by

$$\begin{aligned}\frac{\partial\phi}{\partial r} &= \frac{1}{2\rho}\frac{1}{\sqrt{1-\frac{r^2}{(2\rho)^2}}} = \frac{1}{2\rho}\left(1 + \frac{1}{2}\frac{r^2}{(2\rho)^2} + ...\right)\\ \frac{\partial z}{\partial r} &= t\frac{1}{\sqrt{1-\frac{r^2}{(2\rho)^2}}} = t\left(1 + \frac{1}{2}\frac{r^2}{(2\rho)^2} + ...\right)\end{aligned}$$

Expressing this as a series of operations to evaluates we have

1. $t_1 = r\rho^{-1}/2$
2. $t_2 = t_1 t_1$
3. $t_3 = 1 + t_2/6$
4. $t_4 = t_1 t_3$
5. $\phi = \phi_0 - t_4$
6. $t_5 = tr$
7. $t_6 = t_5 t_3$
8. $z = z_0 + t_6$

Next we consider the case when the track is projected to a disk at a fixed z position. Now we need to calculate the ϕ and r position. From

$$z = z_0 + 2t\rho \arcsin \frac{r}{2\rho}$$

we write

$$r = 2\rho \sin \frac{z - z_0}{2t\rho} = \frac{z - z_0}{t}\left(1 - \frac{(z - z_0)^2}{6(t\rho)^2} + \frac{(z - z_0)^4}{120(t\rho)^4} + ...\right)$$

We also have

$$\phi = \phi_0 - \arcsin \frac{r}{2\rho} = \phi_0 - \frac{z - z_0}{2t\rho}.$$

The derivatives with respect to the projection position, z, are given by

$$\begin{aligned} \frac{\partial r}{\partial z} &= \frac{1}{t}\cos \frac{z - z_0}{2t\rho} = t\left(1 - \frac{(z - z_0)^2}{6(2t\rho)^2} + ...\right) \\ \frac{\partial \phi}{\partial z} &= -\frac{1}{2t\rho} \end{aligned}$$

All of these expression are straight forward to evaluate on an FPGA except that we need to inverse of t. We note here that we will only need the inverse of t for the forward disks where $|t| > 1$. Again, the best way to calculate the inverse if probably by using a lookup table.

Expressing this as a series of operations to evaluate we have

1. $t_1 = z - z_0$
2. $t_2 = 1/t$
3. $t_3 = t_1 t_2$
4. $t_4 = t_3 \rho^{-1}$
5. $\phi = \phi_0 - t_4$
6. $t_5 = t_4 t_4$
7. $t_6 = 1 - t_5/6$
8. $r = 2t_3 t_6$

3.4 Hit matching of projected tracklets

Based on the expressions in the previous section we can calculate the projected position in a different layer. Hits are searched for in the other layers and if a match is found it is selected as stub that belongs to the track. A hit is selected if it is close enough to the projected trajectory. In Fig. 3 is shown the residuals between the projected tracklets and hits in different layers. Based on plots like these search windows are established.

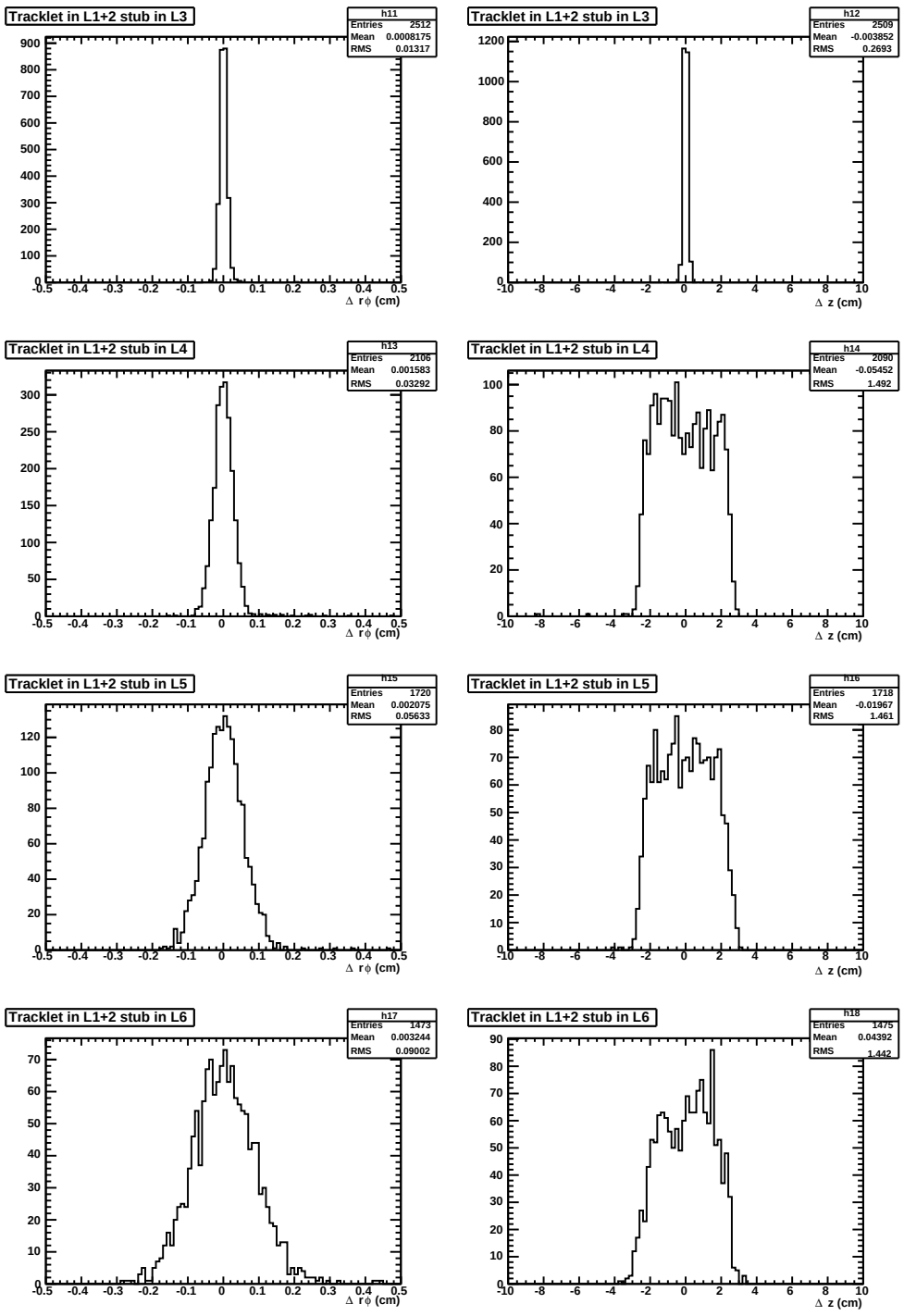


Figure 3: Seeds that are found in barrel layers 1+2 are propagated to the other layers and the residuals of the projected trajectory with respect to the hits are shown. The sample used is single muons with a $p_{\mathrm{T}} = 10$ GeV.

3.5 Linearized track fitting

The track fit implementation used to fit the seed tracks makes optimal use of the information calculated during the track finding step. The method uses precomputed derivatives to very quickly obtain an improved estimate of the track parameters.

Let's first review in a generic form the χ^2 fit employed here. We assume that we have a set of measurements f_i^{m}, where i labels the different measurements. In our fits this is ϕ or z positions of the track on the different layers. Next we have the projections of the track, $f_i(\vec{\eta})$, where $\vec{\eta}$ is the set of parameters that parameterize the track. First we expand the function f_i linearly around a set of seed parameters $\bar{\eta}$

$$f_i(\vec{\eta}) = f_i(\bar{\eta} + \delta\vec{\eta}) = f_i(\bar{\eta}) + \delta\vec{\eta}\frac{\partial f_i}{\partial \vec{\eta}} + \mathcal{O}(\delta\vec{\eta}^2)$$

where we will take $\bar{\eta}$ to be the track parameters from the tracklet. Next we write down the χ^2 as

$$\begin{aligned}
\chi^2 &= \sum_i (f_i(\vec{\eta}) - f_i^{\mathrm{m}})^2 \\
&= \sum_i (f_i(\bar{\eta}) - f_i^{\mathrm{m}} + \delta\vec{\eta}\frac{\partial f_i}{\partial \vec{\eta}})^2 \\
&= \sum_i (\delta f_i^{\mathrm{m}} + \delta\vec{\eta}\frac{\partial f_i}{\partial \vec{\eta}})^2
\end{aligned}$$

where in the last step we have used $\delta f_i^{\mathrm{m}} = f_i(\bar{\eta}) - f_i^{\mathrm{m}}$ which are the residuals between the hits and the seed track. We note that these residuals were already calculated in the track finding step.

Next we want to minimize the χ^2 in order to find $\delta\vec{\eta}$

$$0 = \frac{\partial \chi^2}{\partial \delta\vec{\eta}} = 2\sum_i \frac{\partial f_i}{\partial \vec{\eta}}(\delta f_i^{\mathrm{m}} + \delta\vec{\eta}\frac{\partial f_i}{\partial \vec{\eta}})$$

Introducing the matrices $D_{ij} = \frac{\partial f_i}{\partial \eta_j}$ and $M = D^{\mathrm{t}}D$ we can write

$$0 = D^{\mathrm{t}}\delta f^{\mathrm{m}} + M\delta\eta$$

which we can solve for $\delta\eta$ as

$$\delta\eta = -M^{-1}D^{\mathrm{t}}\delta f^{\mathrm{m}}$$

This expression gives us a simple linear form for how the track parameters should be update given a set of residuals with respect to the seed track. We note that in our fits the seed track is constructed to go through two measurement points so that δf^{m} are zero for the two hits that formed the stub. This means that to evaluate the updated track parameters we only need to consider the hits that were matched to the tracklet in the other layers. Obviously

the computation of $M^{-1}D^{\rm t}$ is nontrivial, but we can precompute these so that the new track parameters can be obtained by simply performing a few multiplications and additions.

We can also compute the χ^2 of the fit fairly efficiently.

$$\begin{aligned}
\chi^2 &= (\delta\vec{f}^{\rm m} + D\delta\vec{\eta})^{\rm t}(\delta\vec{f}^{\rm m} + D\delta\vec{\eta}) \\
&= (\delta\vec{f}^{\rm m} - DM^{-1}D^{\rm t}\delta\vec{f}^{\rm m})^{\rm t}(\delta\vec{f}^{\rm m} - DM^{-1}D^{\rm t}\delta\vec{f}^{\rm m}) \\
&= \delta\vec{f}^{\rm m\,t}(1 - DM^{-1}D^{\rm t})(1 - DM^{-1}D^{\rm t})\delta\vec{f}^{\rm m} \\
&= \delta\vec{f}^{\rm m\,t}(1 - DM^{-1}D^{\rm t})\delta\vec{f}^{\rm m} \\
&= \delta\vec{f}^{\rm m\,t}\delta\vec{f}^{\rm m} - \delta\vec{f}^{\rm m\,t}DM^{-1}D^{\rm t}\delta\vec{f}^{\rm m} \\
&= \chi^2_{\rm seed} + \delta\vec{f}^{\rm m\,t}D\delta\vec{\eta}
\end{aligned}$$

3.6 Adding pixel hits

This section describes how pixel hits could be added to the L1 tracks found in the outer tracker to improve the vertex position. None of this is used in the L1 tracking implementation for the outer tracker.

The assumption here is that we have found a L1 track with track parameters given by $\vec{\eta}_{\rm L1}$ where the components are $(\rho^{-1}, \phi_0, d_0, t, z_0)$. The uncertainties on these parameters are given by the variance matrix V_η. We then also have a set of hits from the pixel detector. These can be either on barrel or disk modules. We denote these hits by m_i (each physical hit in the pixel detector provides two measurements in each of the two dimensions on the sensor, we will treat these as two separate measurements). Each of these measurements have an associated uncertainty σ_i. Further we assume that for a track with parameters $\vec{\eta}$ we would predict that we would measure $f_i(\eta)$ corresponding to each of the obeservatons m_i.

Given this we can write down the χ^2 that we want to minimize for the track including the pixel hits. This χ^2 has two components, the first is from the pixel hits, and the second from the outer track. First we consider the contribution from the pixels which is given by

$$\chi^2_{\rm pixel} = \sum_i \frac{(f_i(\vec{\eta}) - m_i)^2}{\sigma_i^2}$$

Next we write $\vec{\eta} = \vec{\eta}_{\rm L1} + \vec{\Delta}\eta$, i.e. we write the new track parameters as a correction with respect to the track that we found in the outer tracker. Then we linearize $f_i(\vec{\eta})$ as

$$f_i(\vec{\eta}) = f_i(\vec{\eta}_{\rm L1}) + \vec{\Delta}\eta\frac{\partial f_i}{\partial\vec{\eta}}$$

Now we note that

$$f_i(\vec{\eta}) - m_i = f_i(\vec{\eta}_{\rm L1}) - m_i + \vec{\Delta}\eta\frac{\partial f_i}{\partial\vec{\eta}} = \Delta_i^{\rm m} + \vec{\Delta}\eta\frac{\partial f_i}{\partial\vec{\eta}}$$

where $\Delta_i^{\rm m} = f_i(\vec{\eta}_{\rm L1}) - m_i$ is just the residual between the pixel hit and the trajectory of the outer tracker track. Putting this together we have

$$\chi^2_{\rm pixel} = \sum_i \frac{\left(\Delta_i^{\rm m} + \vec{\Delta}\eta \frac{\partial f_i}{\partial \vec{\eta}}\right)^2}{\sigma_i^2}$$

Further defining the matrix D by

$$D_{ij} = \frac{1}{\sigma_i}\frac{\partial f_i}{\partial \eta_j}$$

and

$$\tilde{\Delta}_i^{\rm m} = \Delta_i^{\rm m}/\sigma_i$$

we can write

$$\chi^2_{\rm pixel} = \left(\tilde{\Delta}^{\rm m} + D\Delta\eta\right)^{\rm t}\left(\tilde{\Delta}^{\rm m} + D\Delta\eta\right).$$

Next we consider the contribution to the χ^2 from the outer tracker. This information is contained in the variance matrix V_η and the contribution to the χ^2 is simply given by

$$\chi^2_{\rm outer} = \Delta\eta^{\rm t} V_\eta^{-1} \Delta\eta$$

Putting this together we can now write the total χ^2 as

$$\chi^2 = \chi^2_{\rm pixel} + \chi^2_{\rm outer} = \left(\tilde{\Delta}^{\rm m} + D\Delta\eta\right)^{\rm t}\left(\tilde{\Delta}^{\rm m} + D\Delta\eta\right) + \Delta\eta^{\rm t} V_\eta^{-1}\Delta\eta.$$

As usual we now find the minimum of the χ^2.

$$0 = \frac{\partial \chi^2}{\partial \vec{\Delta}\eta} = 2\left(\tilde{\Delta}^{\rm m^t} D + \Delta\eta^{\rm t} D^{\rm t} D + \Delta\eta^{\rm t} V_\eta^{-1}\right)$$

Solving this we get

$$\Delta\eta = \left(V_\eta^{-1} + D^{\rm t} D\right)^{-1} D^{\rm t}\tilde{\Delta}^{\rm m}$$

Like for the linearized track fit described above for the track fit in the outer detector we have here found a linear relation between the residuals and the corrections to the track parameters. The difference here is that the expression also includes the information from the outer track fit.

The implementation of this can also be done efficiently in hardware as we can precompute the matrix combination. To realize this we note a few things.

- The matrix D for barrel hits is independent (to a very good approximation) of the track parameters. For hits on disks it dependes on t, but can be tabulated for a few different values of t. (Needs more detailed study.)

- The matrix V_η depends on which layers were hit in the outer layer. But note that since we are primarily interested in the impact parameters z_o and d_0 in the fit to the pixel hits, and since the pixels will competely drive this measurement we are not going to be sensitive to the details of the V_η matrix.

Hence we can precompute the matrix combination

$$\left(V_\eta^{-1} + D^{\mathrm{t}} D\right)^{-1} D^{\mathrm{t}}$$

and the track fit has then been reduced to just a few multiplications and additions. The hard part will be to identify the correct hits in the pixel detector.

3.7 Duplicate removal

Explain how we will remove tracks that are found multiple times. Stay tuned - will soon come.

4 Performance of Algorithm

The track finding efficiency and track parameter resolutions are estimated using Monte Carlo simulation. Samples with single-muon events (45,000 μ^- and 46,500 μ^+ events) with a pileup of 140 were produced. The muons are uniformly distributed in the transverse momentum range $0.2 < p_{\mathrm{T}} < 100$ GeV, covering the pseudorapidity range $|\eta| < 3.0$. The samples used a preliminary version of the five-disk barrel+endcap geometry. The tracking efficiency and the track parameter resolutions are studied for tracks with $\chi^2 < 100$, $|\eta| < 2.5$, and a minimum of 4 stubs associated to each track.

4.1 Seeding performance

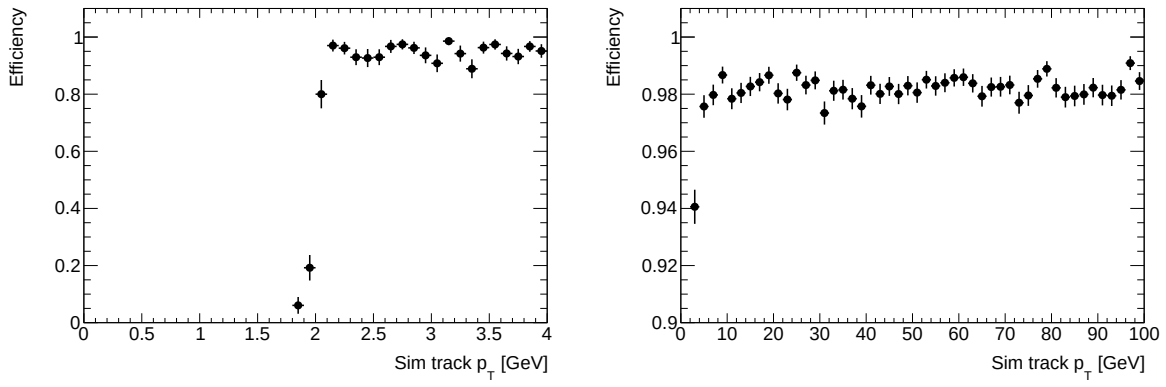
The performance of the seeding as a function of layer and η is discussed in this section.

4.2 Tracking efficiency

First, the track finding efficiency for single muons in the presence of high pileup is studied. The efficiency is measured with respect to the *sim track*, corresponding to the generated muon. Figure 4 shows the tracking efficiency as function of the sim track p_{T} for the low-p_{T} turn-on region as well as for the full p_{T} range. As illustrated by the figure, the efficiency has a sharp turn-on at 2 GeV, the minimum track p_{T} considered for the L1 track finding. Above the turn-on, the efficiency is 98–99%.

Figure 5 shows the efficiency as function of sim track η and ϕ for muons with $p_{\mathrm{T}} > 2$ GeV. The efficiency is observed to be more or less independent of ϕ, while an η dependence is observed. The efficiency at very high η values ($\eta > 2.3$) is decreased as a result of the geometric acceptance of the simulated five-disk geometry. In updated versions of this geometry, the efficiency loss in this region is expected to be largely recovered. In the central pseudorapidity region, $|\eta| < 1.0$, the efficiency is very high at >99%. In the barrel to endcap transition regions, around $1.0 < |\eta| < 1.5$, the efficiency is reduced to about 96%. The transition region is more complicated in terms of the tracking finding as forming tracklets

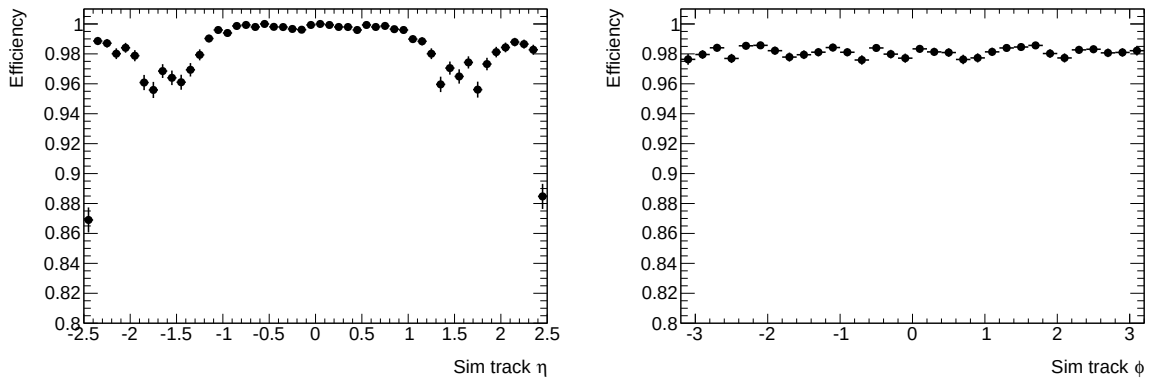
between stubs in the barrel layers and the endcap disks becomes necessary. Optimizing the tracking performance in the barrel-endcap transition region is currently under study[1].



(a) (b)

Figure 4: Tracking efficiency as function of sim track p_{T} for (a) $p_{\mathrm{T}} < 4$ GeV and for (b) $p_{\mathrm{T}} < 100$ GeV.

[1]The observed efficiency loss can be recovered by loosening the track selection criteria to include 3-stub tracks, however, this somewhat decreases the track parameter resolution in this region due to inclusion of poorly reconstructed tracks.



(a) (b)

Figure 5: Tracking efficiency as function of (a) sim track η and (b) sim track ϕ for muons with $p_{\mathrm{T}} > 2$ GeV.

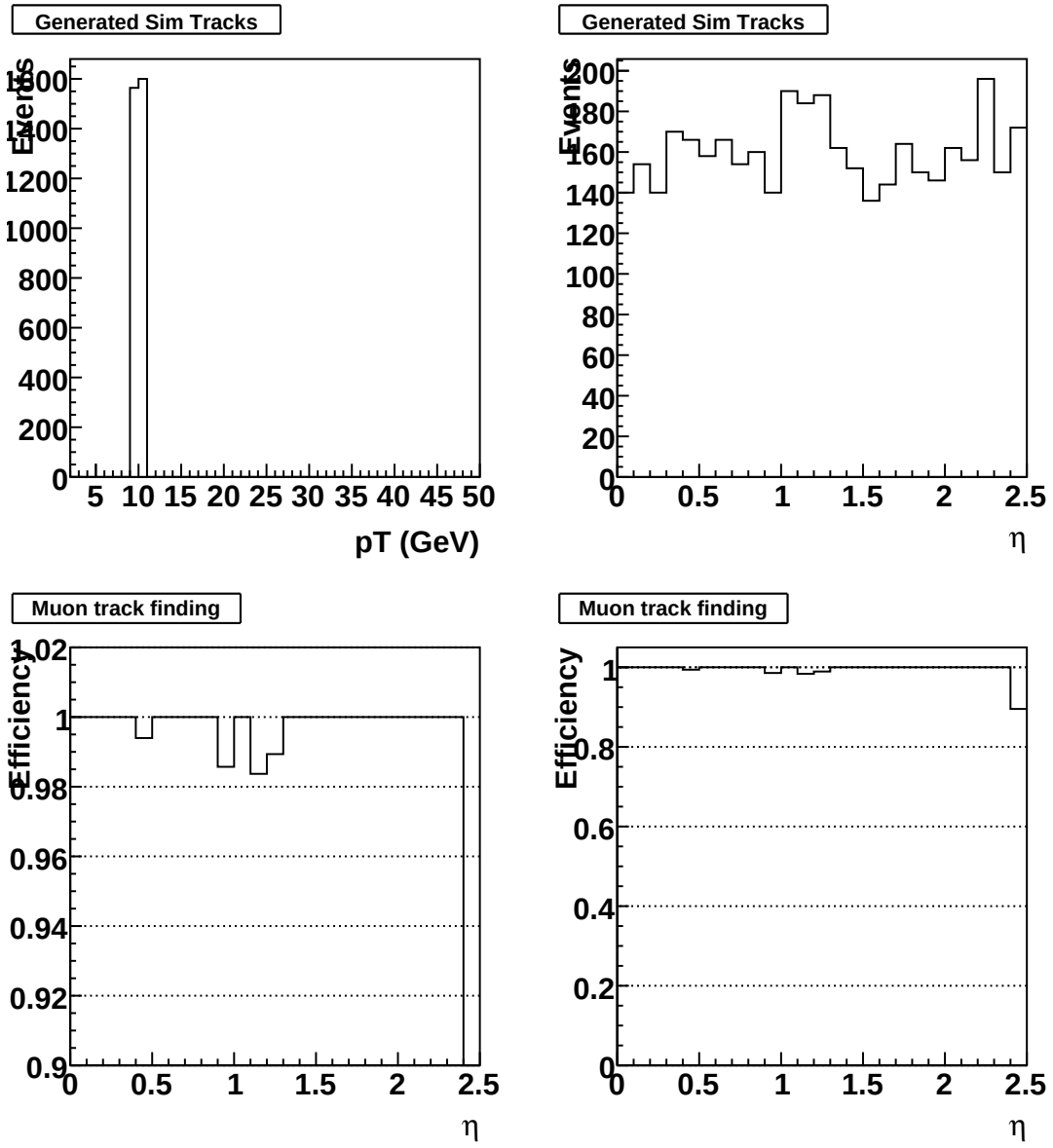


Figure 6: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. All layers and disks are used in the track finding.

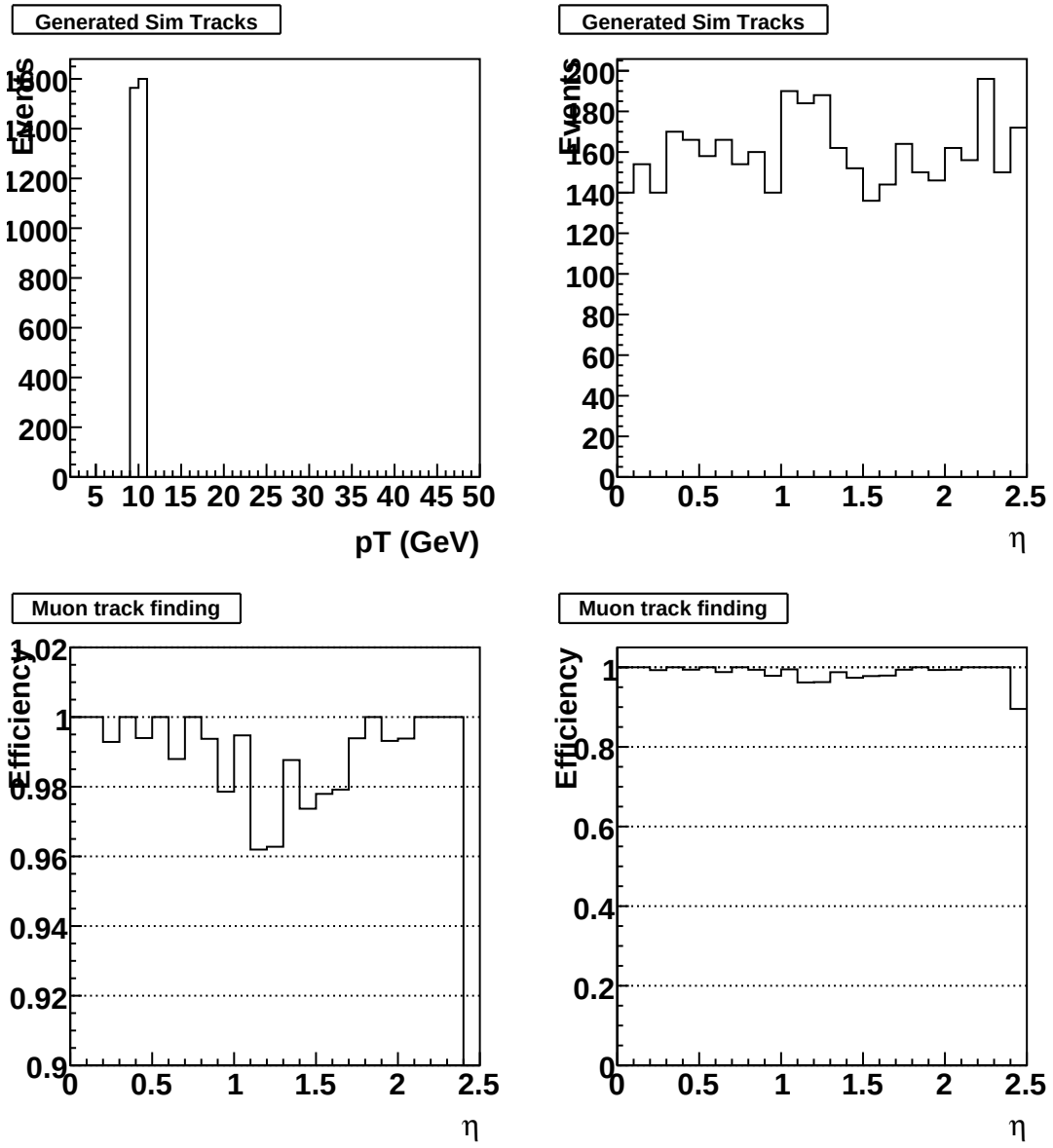


Figure 7: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses all layers *except* the innermost barrel layer.

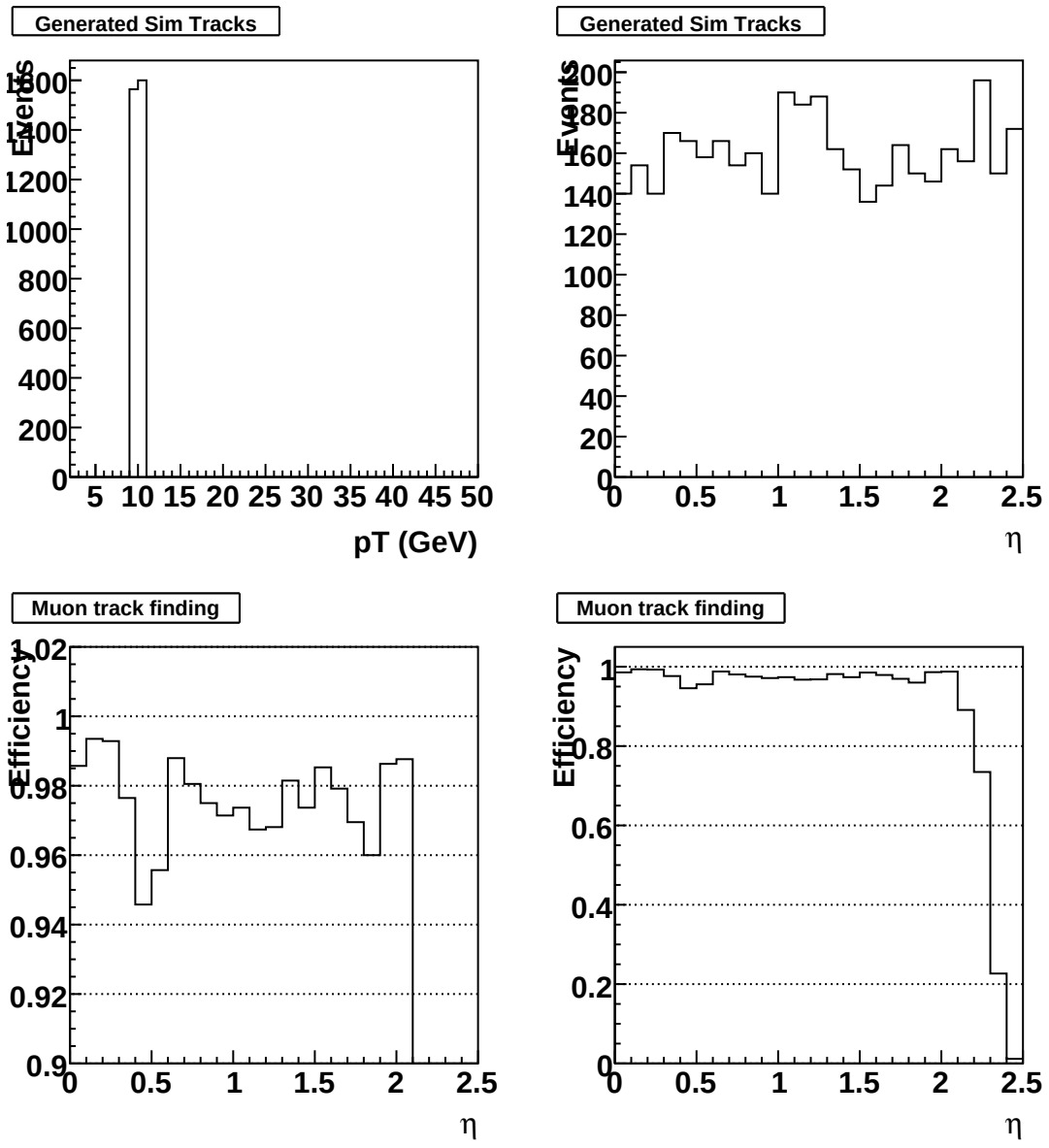


Figure 8: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses the innermost barrel layer combined with either the second barrel layer or the first disk.

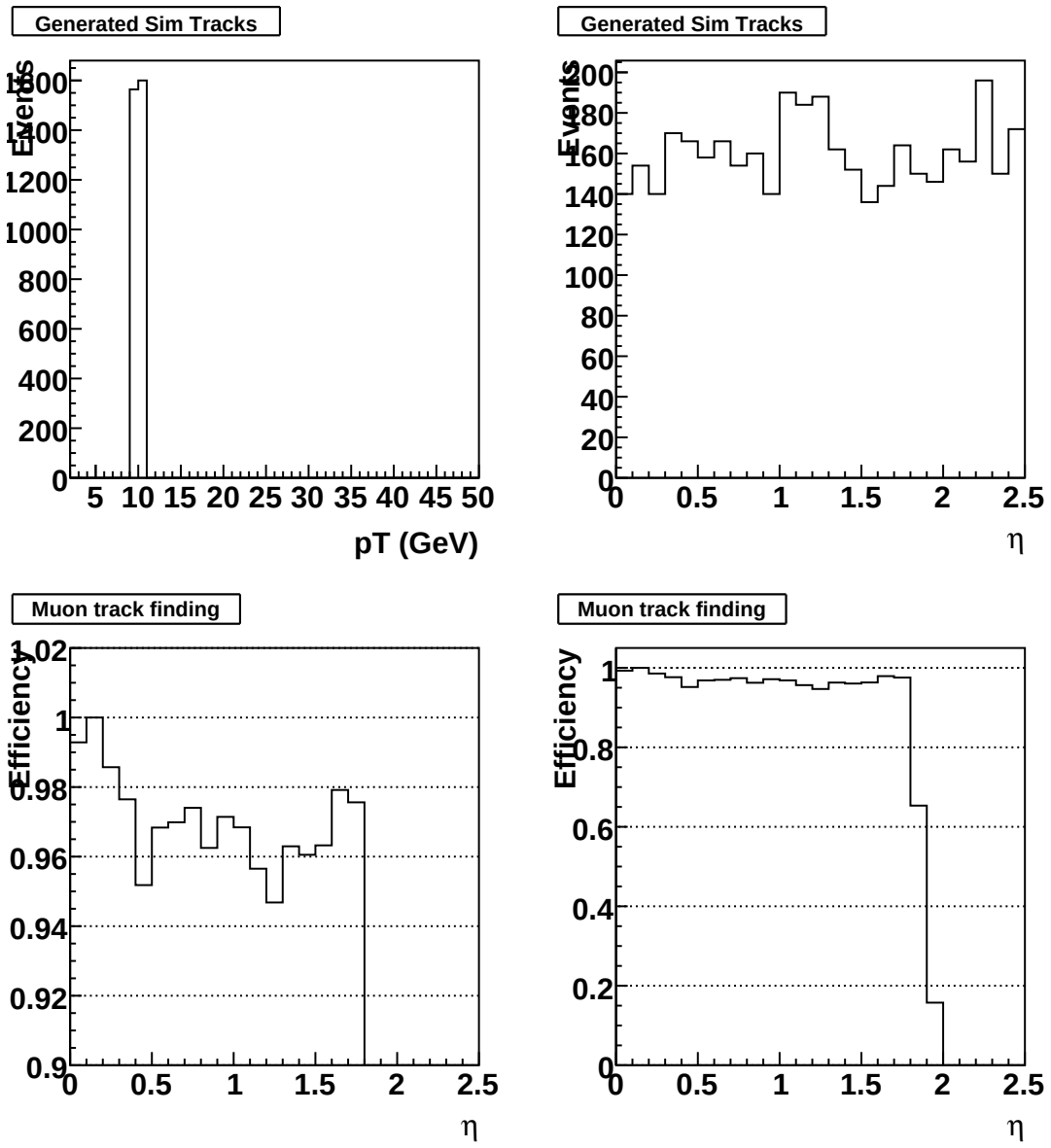


Figure 9: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses the second innermost barrel layer combined with either the third barrel layer or the first disk.

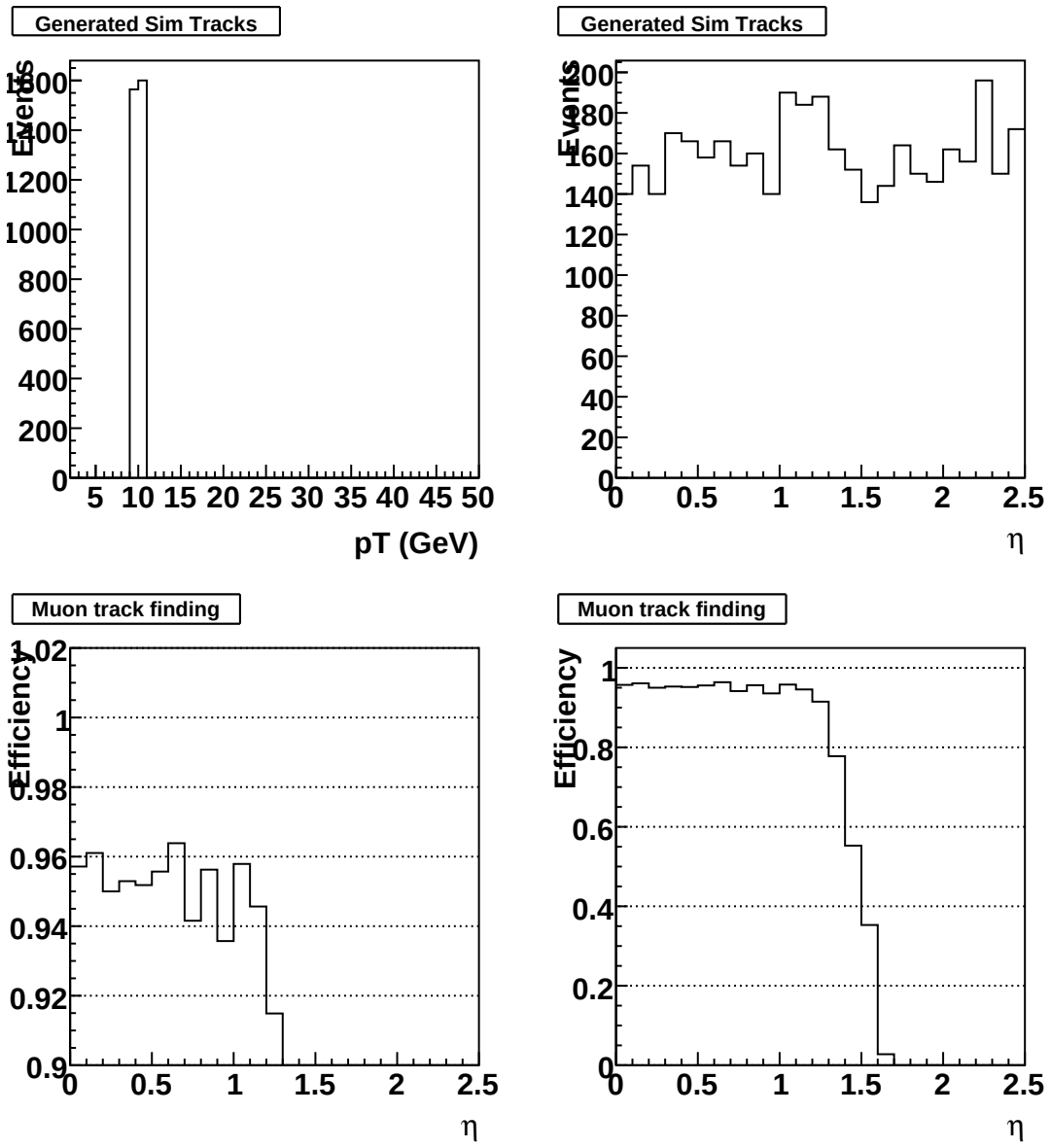


Figure 10: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses the third innermost barrel layer combined with either the fourth barrel layer or the first disk.

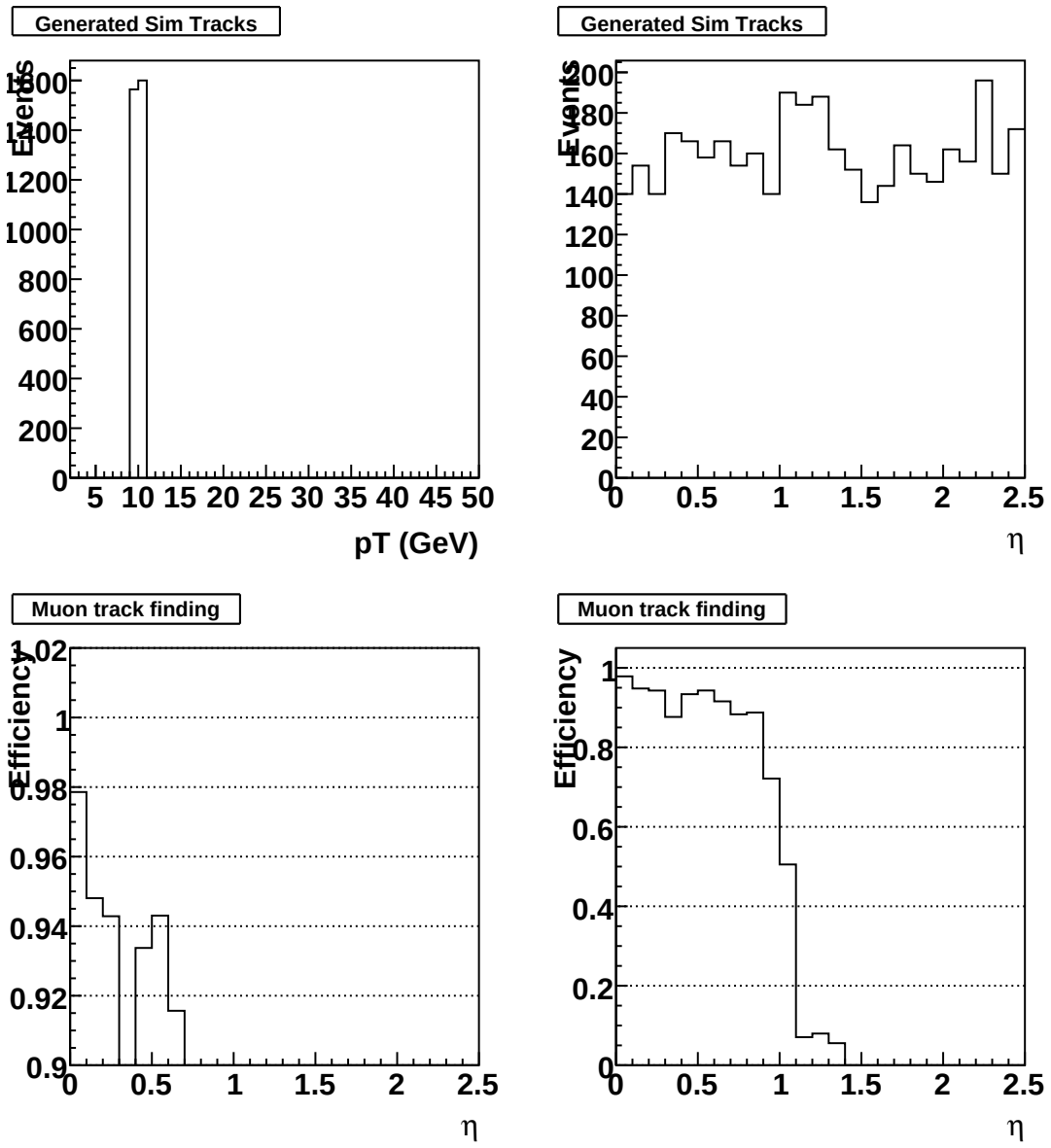


Figure 11: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses the fourth innermost barrel layer combined with either the fifth barrel layer or the first disk.

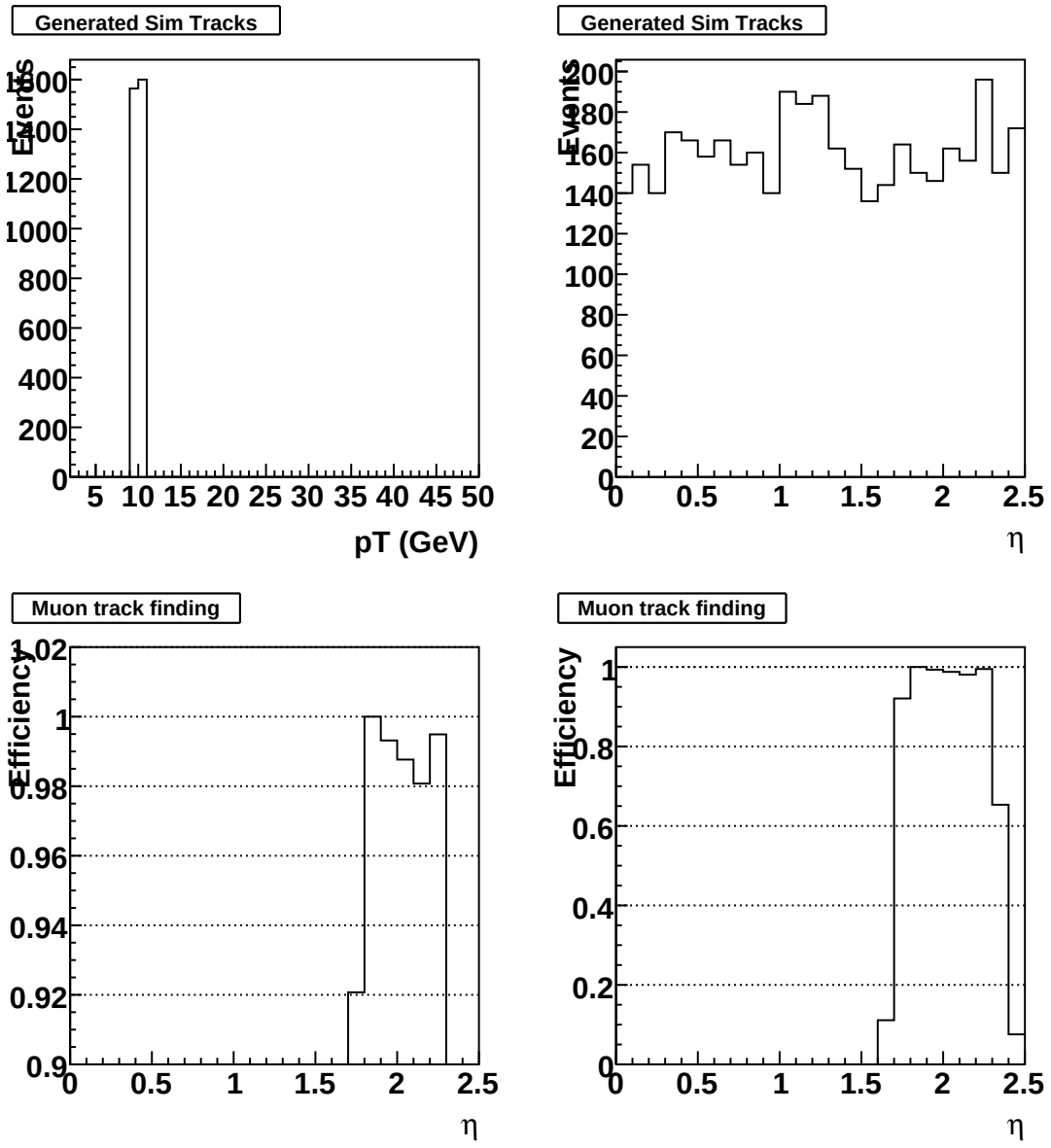


Figure 12: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses the innermost, either forward of backward, disks combined with the second disk.

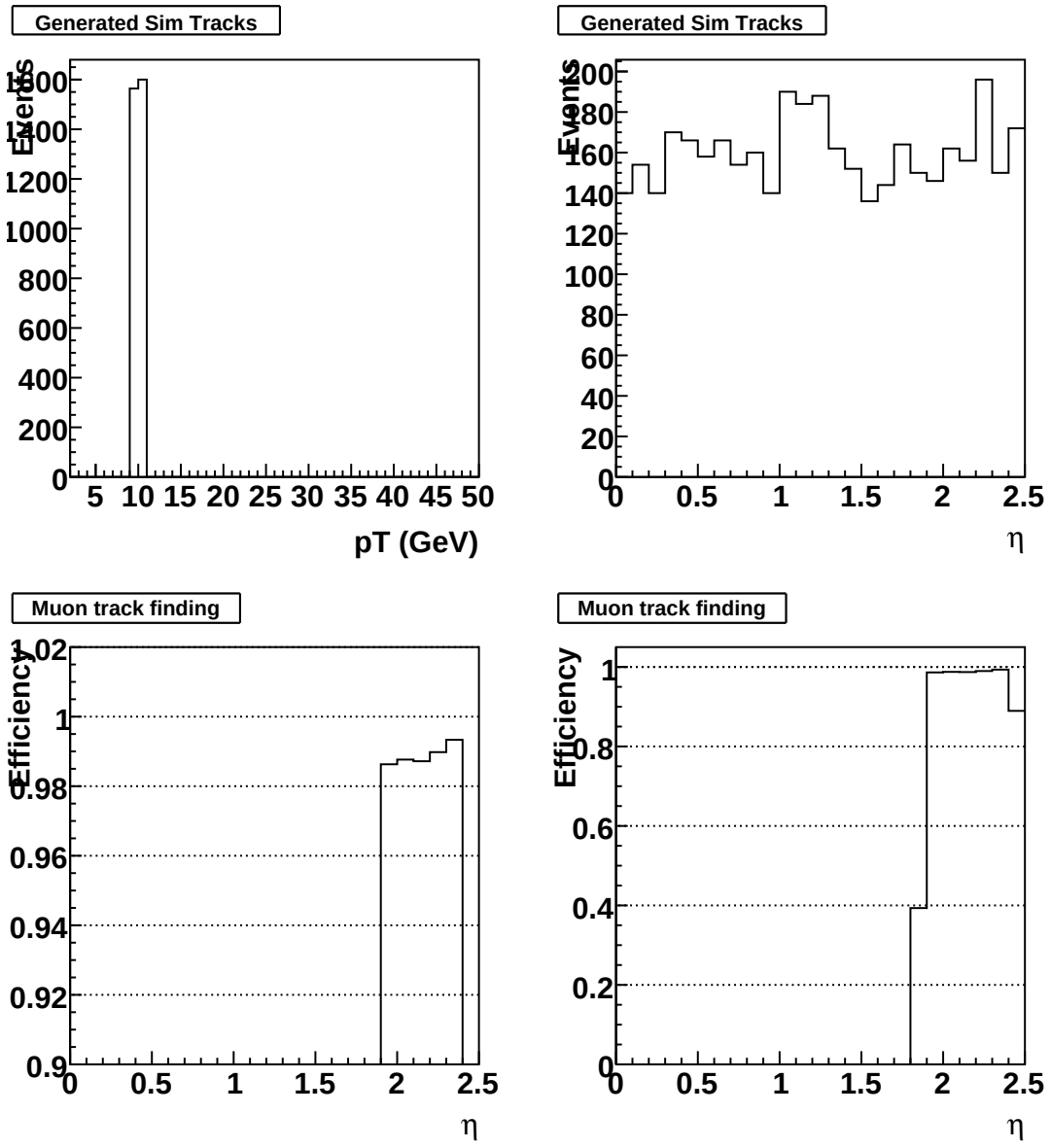


Figure 13: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses the second innermost, either forward of backward, disks combined with the third disk.

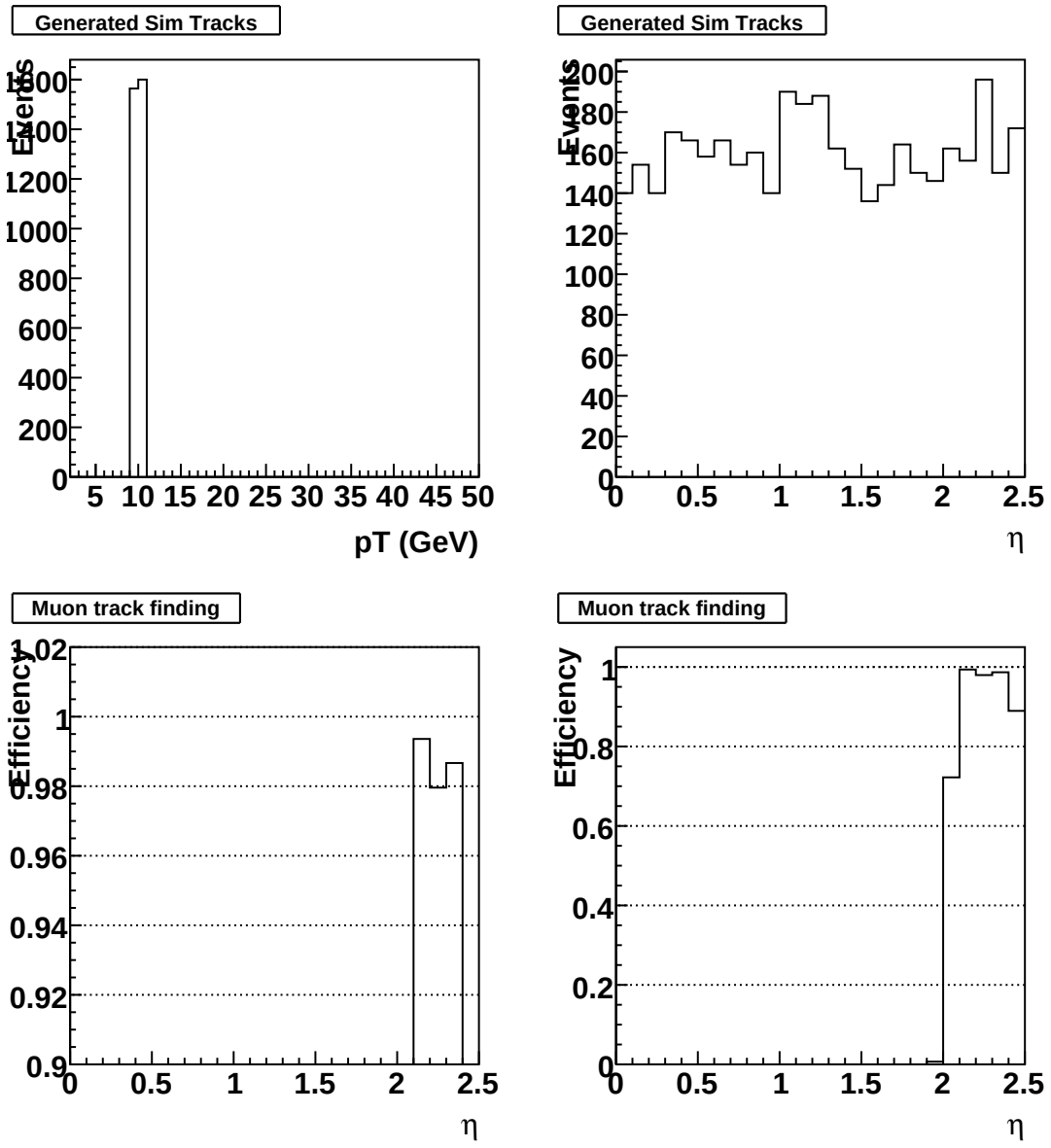


Figure 14: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses the third innermost, either forward of backward, disks combined with the fourth disk.

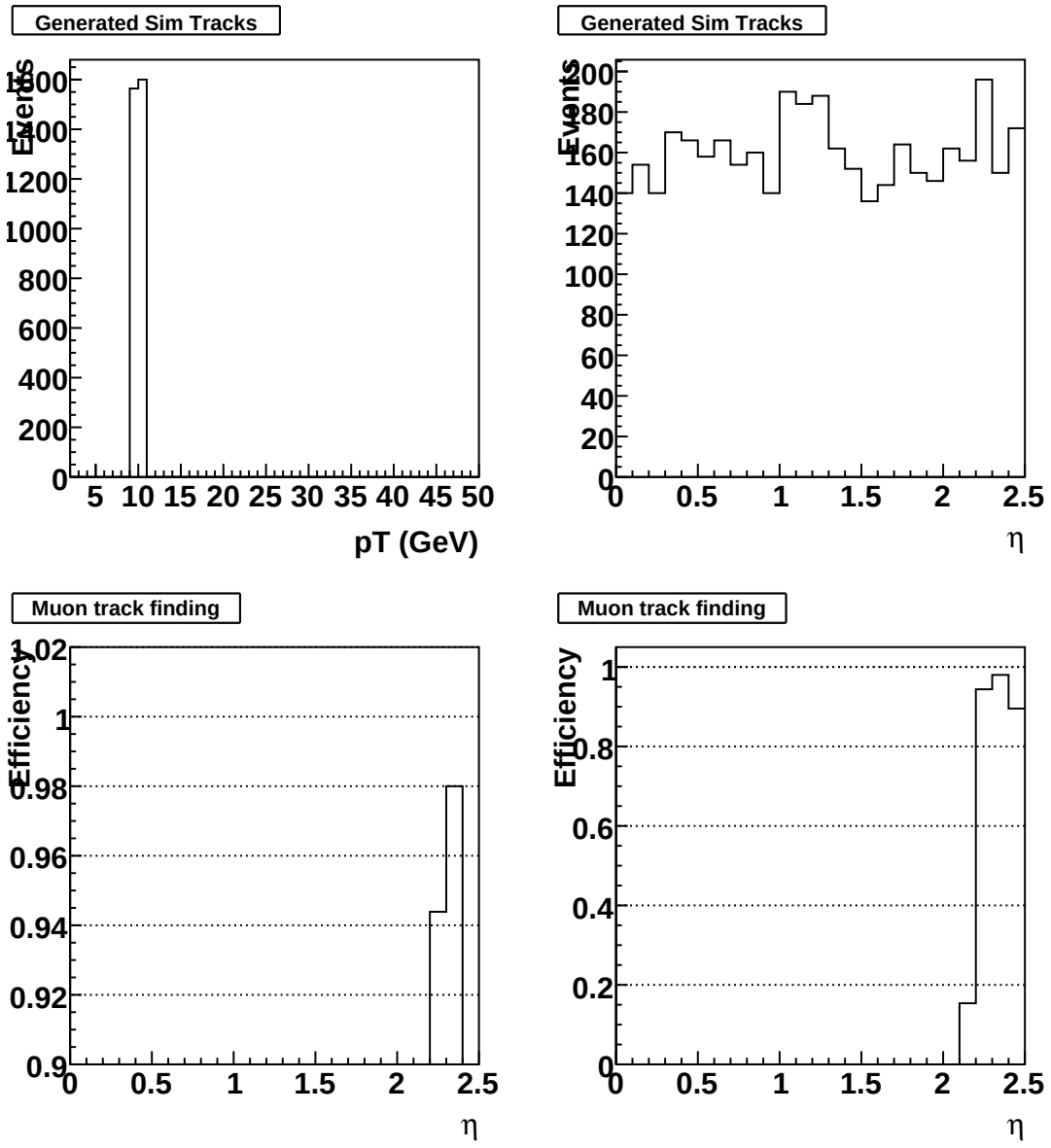


Figure 15: Tracking efficiency as function of sim track η for muons with generated $p_{\mathrm{T}} = 10$ GeV. Seeding for the track finding uses the fourth innermost, either forward of backward, disks combined with the fifth disk.

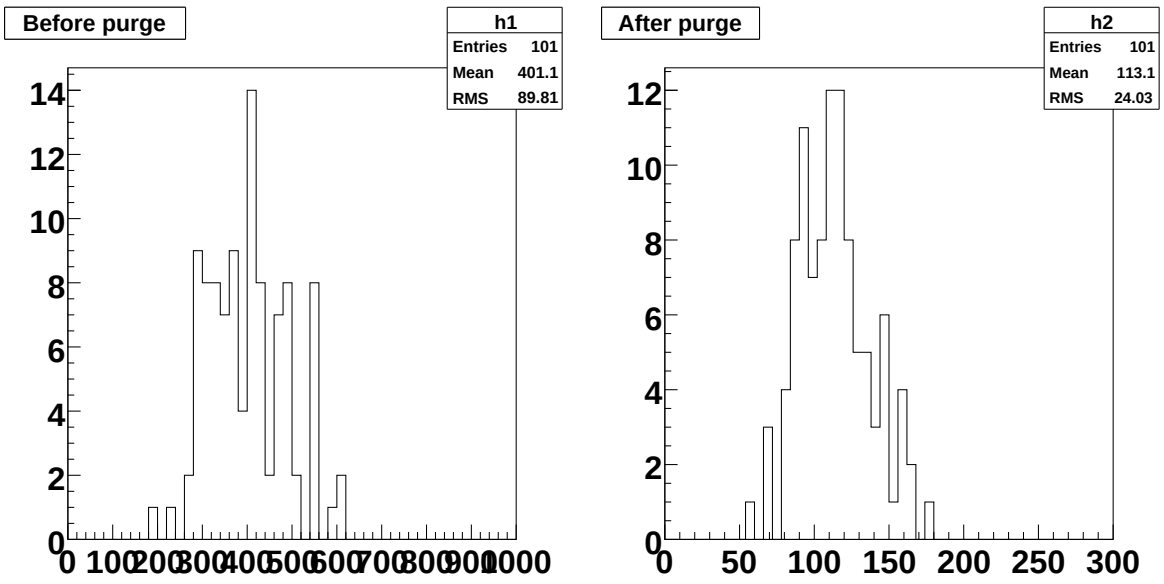


Figure 16: The track multiplicity in 100 events with 140 PU. The plot to the left shows the track multiplicity before the removal of duplicates and the plot to the right shows the multiplicity after the duplicate removal.

4.3 Track parameter resolutions

The track parameter resolutions of the fitted L1 tracks are here studied using the single muon events with high pileup. The η, ϕ, z_0, and relative p_{T} resolutions are evaluated as function of sim track η, shown in Figure 17. The η resolution ranges between 1–2.5 mrad but is improved for intermediate values of η. The ϕ resolution is about 2 mrad in the barrel region but is increased to up to 4 mrad in the endcap region. The z_0 resolution is about 1 mm at central η but is worsened at higher η. For higher η values, this is partly attributed to failing to match tracklets formed in the endcaps to stubs in the innermost barrel layer, which has the highest impact on the z_0 measurement. The p_{T} resolution is about 1% at central η but is significantly worsened in the transition and outer endcap regions.

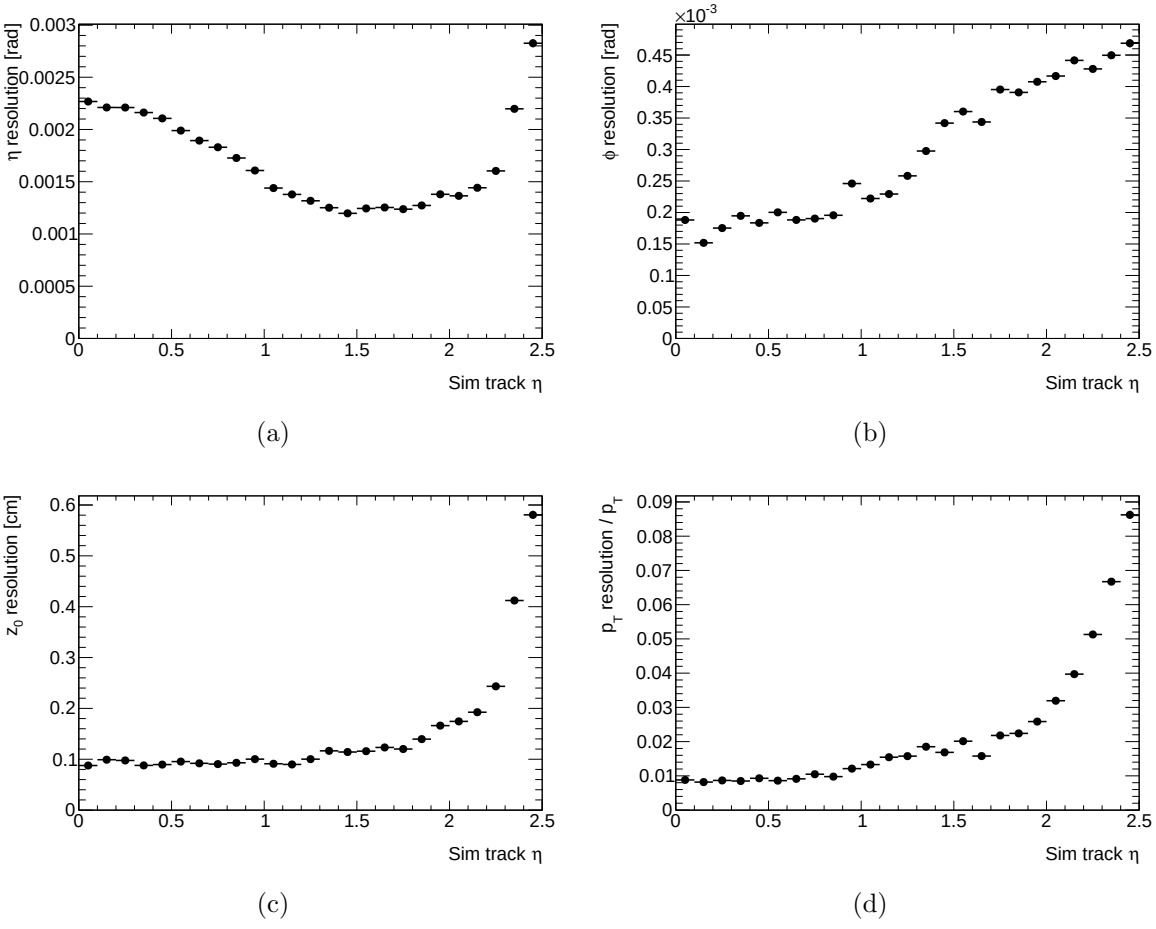


Figure 17: Track (a) η, (b) ϕ, (c) z_0, and (d) p_{T} resolution as function of sim track η.

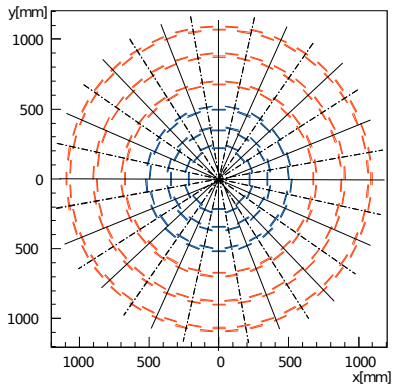


Figure 18: ϕ view of proposed sector layout. There are 28 sectors in ϕ. The sectors are split into two regions in $\pm\eta$.

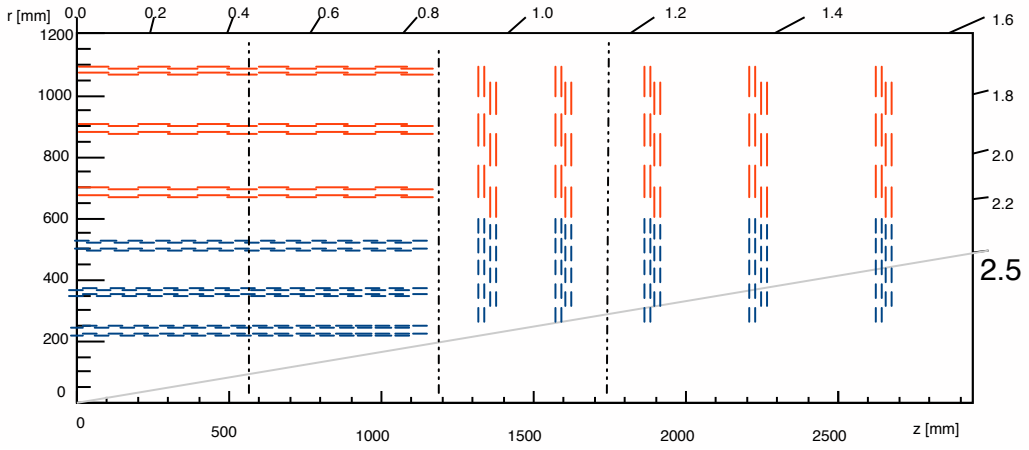


Figure 19: $r - z$ view of tracker with proposed FED layout. Four divisions in z - two in the barrel, two in the endcap. The η divisions are solely for the purpose of assigning and counting FEDs; except for the $\eta = 0$ boundary the sectors do not have η divisions.

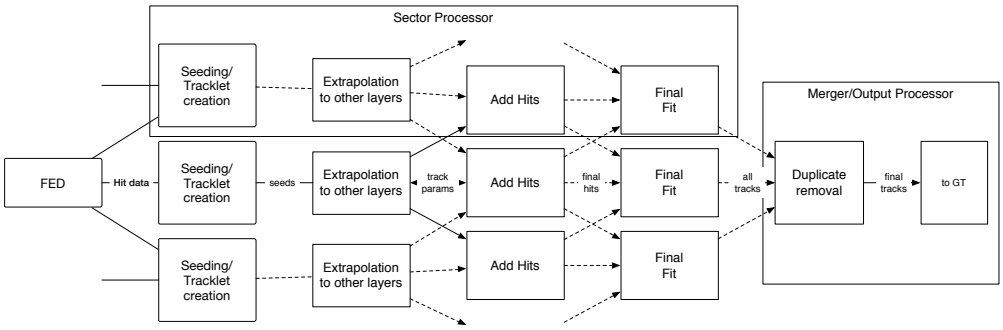


Figure 20: Block diagram. Data from the FEDs is sent to sector processors, which first take pairs of stubs from adjacent layers to make tracklets. The tracklets are extrapolated to all other layers and a request to add hits on those layers to the track is sent. The matching hits are returned for a final fit. Duplicates are removed in a merging step. Finally, the track list is sent to the global trigger. More details in the text.

5 Architecture

For the purpose of the L1 trigger, the tracker is divided in $\eta - \phi$ slices called sectors. There are 16 sectors in total in ϕ and two in η. See Fig. 18 & 19 for an approximate representation. The ϕ layout is driven by the symmetries of the detector. The FED mapping matches the sectors. The $r - z$ view shows a suggested layout for FEDs in z. With the proposed sector arrangement, a 2 GeV track can be contained within one sector, and each FED transmits data to at most three adjacent sector processors, except for the overlap region around $\eta = 0$.

5.1 System overview

A block diagram is presented in Fig. 20. Data from the FEDs is sent to sector processors, which first take pairs of stubs from adjacent layers to make tracklets. The sector processors receive data from 12 FEDs in ϕ (3 in phi times 4 in eta) The tracklets are extrapolated to all other layers (towards and away from the beam spot) and a request for matching hits on those layers is sent to all other layers. These requests can be directed within the same FPGA, on a different FPGA on the same processor, or to other processors, but the sector size is chosen data is only needed in the adjacent sectors (± 1 in ϕ). The information that is sent to the other layers are the preliminary track parameters from the tracklet, plus addressing information. The information sent back from the other layers are the matched stubs plus addressing information). The matching hits on all layers are returned to the originating seed tracklet for a final fit. Duplicates are removed in a merging step. Finally, the track list is sent to the global trigger. Each track is

As previously mentioned, except for the $\eta = 0$ transition region, each FED sends data to three sectors. In turn, each sector processor receives data from $3 \times 4 = 12$ FEDs. (If we

Data	Size in bits
Stub	46
Tracklet	100

Table 1: Approximate data sizes of various elements. Not sure we want to show this. Most are internal to Verilog code and but do include addressing data. Missing BC (8 bits?) to identify time slice.

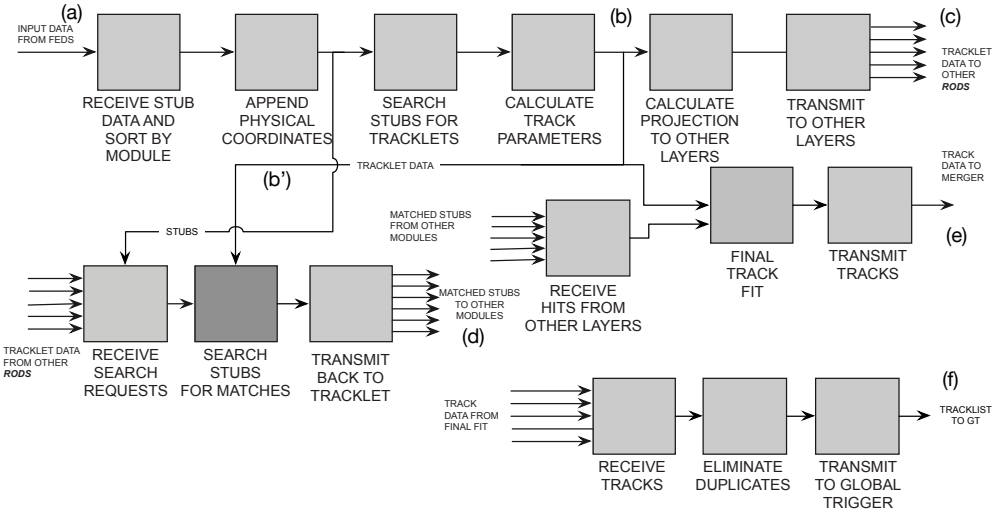


Figure 21: Algorithm Block Diagram. The locations labeled (a) through (f) are the rates described in Sec. 5.2. For label (c), data can be transmitted to processors within the same FPGA, to other FPGAs on the same board, or to other boards, when sector boundaries are crossed. Similarly for step (d).

can use a scheme were each sector board receives data from two sets of FEDs each providing half of the data to the sector board we can reduce the number of links to 2×4. This also has the advantage of producing links that are having approximately the same data volume.)

In Fig. 21, the algorithmic steps are broken down in more detail. Data is received from the FEDs (label a on the Figure). Geographic location data is appended to the data from the FEDs, and the data is used for seed tracklet generation and for hit matching. First, we search stubs in adjacent layers for candidate tracklets, which seed later steps in the algorithm. The label b will be used below to show a rate of tracklets. We calculate an initial set of track parameters (p_t, η, ϕ_0, z_0). This initial track helix is used to calculate the intersection point of this tracklet candidate with an average radius at layers going towards and away from the beam spot, and will also be used in the final fit. We determine which modules the track intersects with, and send a request for hits to these modules (label b'). A separate state machine receives the request for hits and compares the stored hits with the incident tracklets. We make a refined track parameter calculation now with the precise radius of the module and return hit residuals within a matching window to the originating sector board (label d).

Here, a final fit is performed with all matching hits from all layers. The final track is sent to a merging stage (label e), where duplicates are removed, and the data is then transmitted to the global trigger (label f).

5.2 Data rates

For the algorithm described in the previous Section we estimate data volumes. In the architecture described above we can study the data volumes based on simulation of minimum bias events and higher occupancy events.

5.2.1 Stub rates

With 28 sectors we first show the number of stubs that each sector needs to process in Fig. 22. (All plots in this section are for average PU of 140.) On average there are about 300 stubs in each sector. (This includes both barrel and the disks.)

This data will be sent to the sector boards from the FEDs. We first consider a scheme where we divide the detector into 8 'FED regions' in z. The barrel part is divided into 4 region, the innermost two disks on each side forms two regions, and the outermost two disks form the last 2 regions. The data volume for each of these regions is plotted in Fig. 23. The top two and bottom two corresponds to the disks. As can be seen there are much fewer stubs in these disks regions than there are in the barrel region. (Need to check that this is really true, seems like the occupancy is really low.)

To balance the load we can combine all the disks on each side into a FED region and have a total of 6 fed regions. This combination is shown in Fig. 24. In this configuration the data volume for each FED region is fairly similar with about 50 stubs per BX on average, but with tails over 100 stubs.

The data delivered to each sector board would come from two FEDs. If we further split the data in one sector into two halfs, each half coming from one FED, we get a load per link as shown in Fig. 25.

If we consider that each stubs requires 40 bits and we use a 100 Gbits/s linke we can transfer 250 stubs in 100 ns. This should provide more than sufficient safety margins.

5.2.2 Tracklet rates

From the stubs we form tracklets in pairs of layers. This corresponds to label (b) in Fig. 21. With the large separation between layers, about 15 cm, we have a fairly large rate of fake tracklets. In Fig. 26 is shown a distribution of tracklets found in a half sector. We see that the number of tracklets is by far the largest in the inner layers, for example there are on average around 26 tracklets found in the innermost two layers per half sector. However, the tail of this distribution reaches close to 100 tracklets.

Some of these tracklets that are found will project to the neighboring sectors and hence to perform the stub matching we need to send the tracklet information to nearby sector

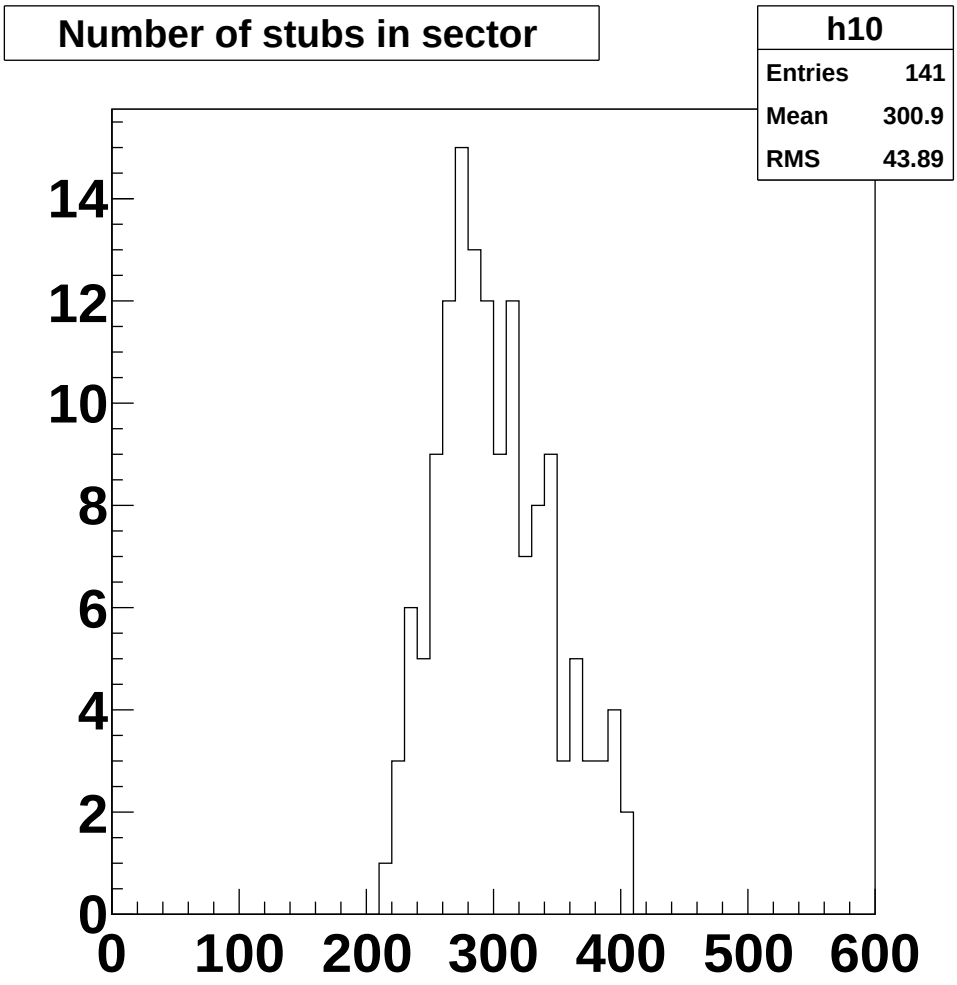


Figure 22: This plot shows a distribution of the number of stubs that each sector recieves for each bunch crossing.

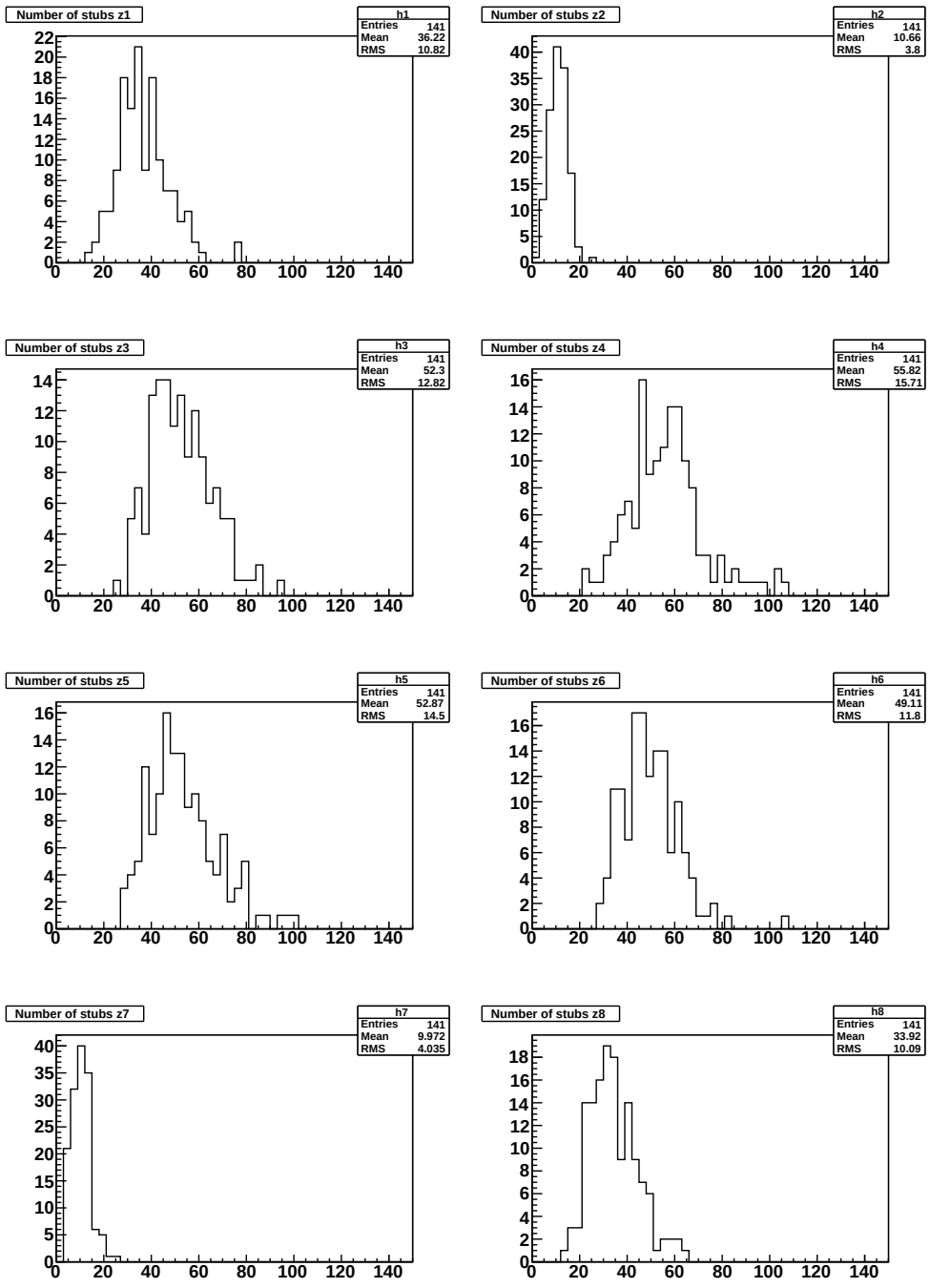


Figure 23: This plot shows a distribution of the number of stubs that are sent to a sector in each of 8 fed regions.

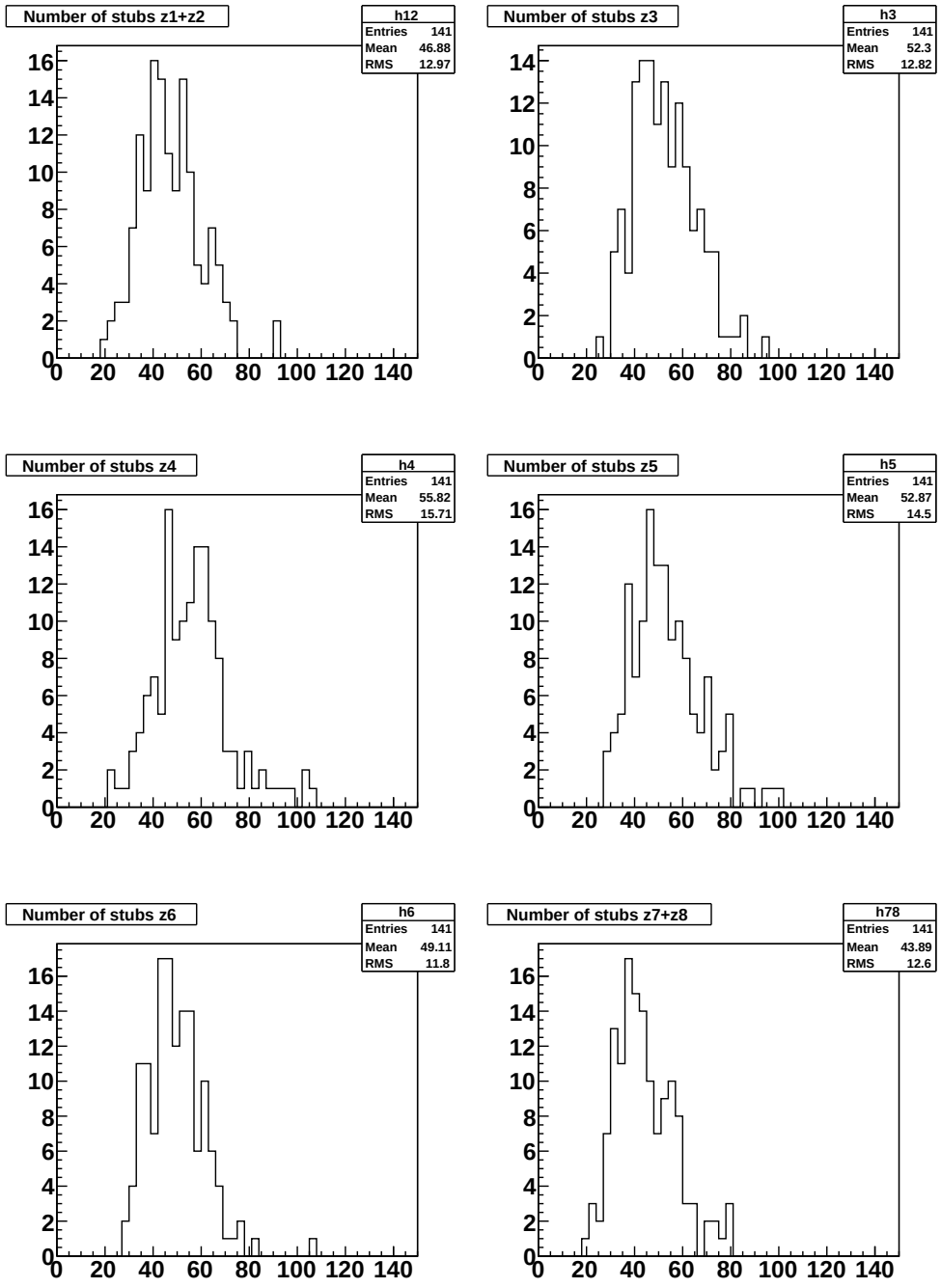


Figure 24: This plot shows a distribution of the number of stubs that are sent to a sector in each of 6 fed regions.

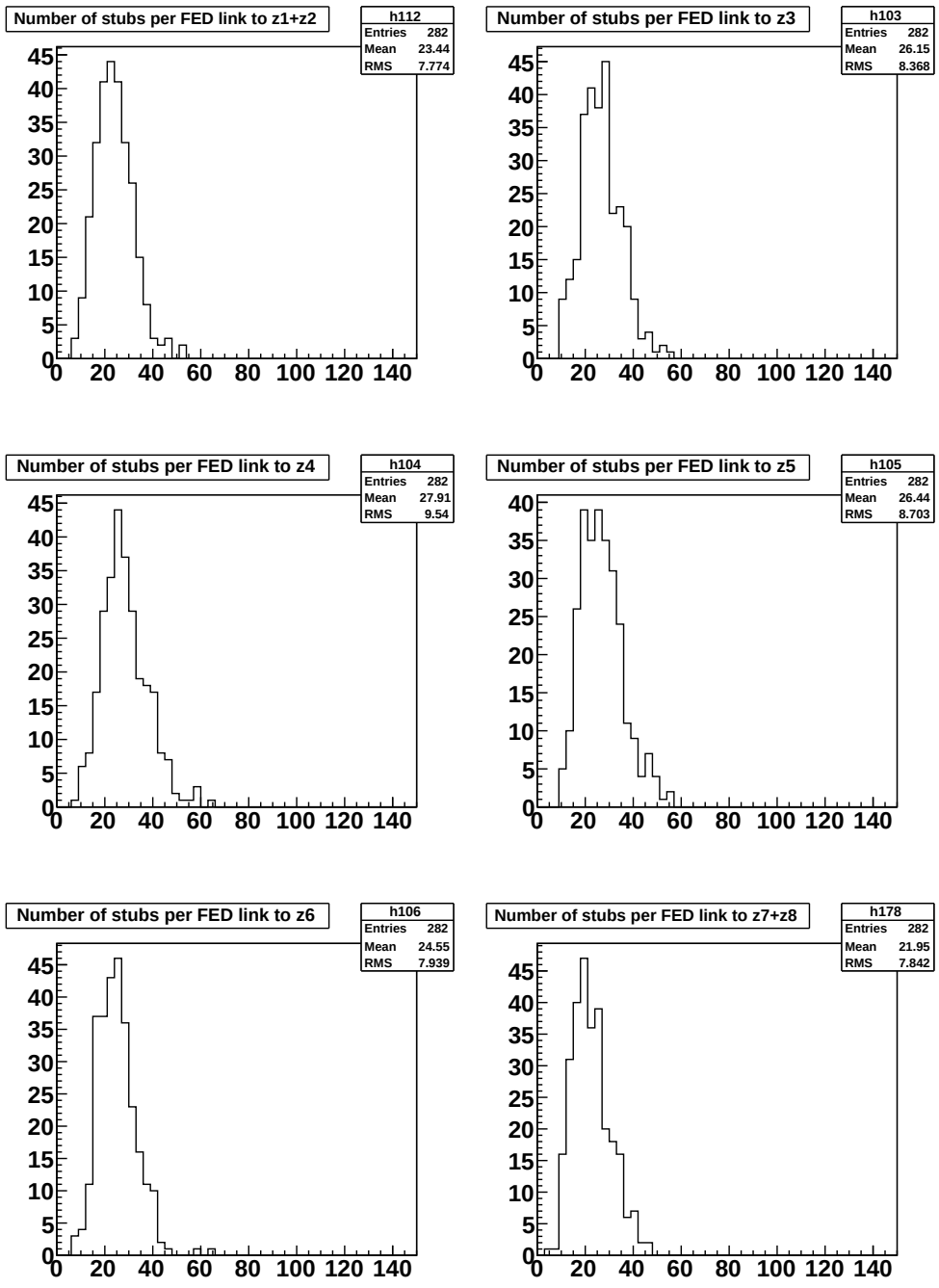


Figure 25: This plot shows a distribution of the number of stubs that are sent from a FED to a sector board.

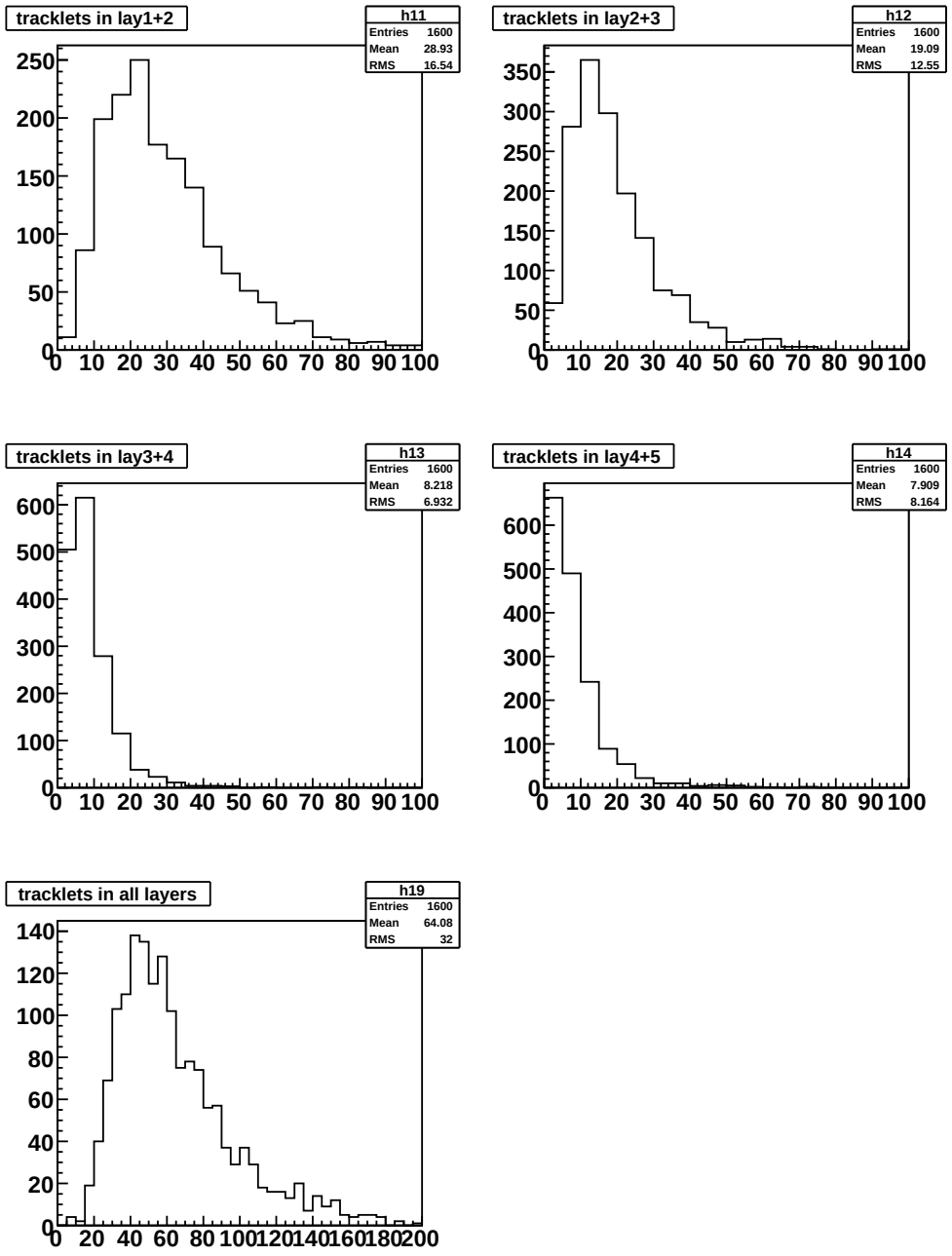


Figure 26: The number of tracklets found in the barrel for different combination of layers. The lowest histogram shows the sum of all stubs found in the sector.

Table 2: Number of DSP operations required for each track.

Step	Number of calculations	Number of objects	Number Calculations
Tracklet parameters	21	130	2,520
Tracklet projections	10×4	65	2,600
Residual calculation	4×4	20	320
Track fit	50	20	1,000
Total			6,440

(label c). In Fig. 27 are show the distribution of the number of tracklets per half sector and bunch crossing that needs to be propagated to a nearby sector.

The number of tracklets found is fairly large, but most of these are combinatoric, i.e. from stubs not belonging to the same track. In Fig. 28 is shown the number of stubs matched to each tracklet. The majority of tracklets are matched to zero or one additional stub and this shows that most of the fake tracklets will be rejected as they are not matched to any stubs.

5.2.3 Matching occupancy

5.2.4 Resource estimate

Based on the estimate of the number of stubs, tracklets, and tracks per sector from the earlier sections we will estimate the computing resorces required to implement this algorithm. First we focus on the number calculations required by the DSPs. In Tab. 2 is shown a summary of the number of calculations we estimate are required to perform the L1 track finding and fitting in one half sector. Note that the DSP optimizes the multiply and add such that this is just considered as one operation. The assumptions that went into building Tab. 2 are explained below.

The first step is to calculate the physical coordinates r, ϕ, and z for each stub. It is assumed that this can be done by three operations for each of the coordinates, giving a total of 9 operations for each of the 300 stubs in a half sector. We form on average 65 tracklets per sector, but we will probably try about twices as many combinations and reject some as they will not satisfy the stub p_{T} and z_0 criteria. As described earlier it takes 21 calculations to find the track parameters. To project the tracklets to the other 4 layers or disks we need 10 operations. If matches are found we need to evaluate the residuals, we estimate this to be 4 operations for each of the 4 layers. These residuals are estimated for each of about 20 tracks. In the last step we find the optimal track parameters and evaluate the χ^2. This is estimated to take 20 operations for each track.

If we operate the FPGA at 1 GHz we get 25 clock cycles per BX. This means that with 3000 DSPs we would have a capacity of 75,000 DSP operations per BX. The estimate above of a total of 9,140 operations means that we are using 12% of the FPGAs capacity. Another way of saying this is that for each task we can allocate a factor of 8 more FPGAs than what is needed on average.

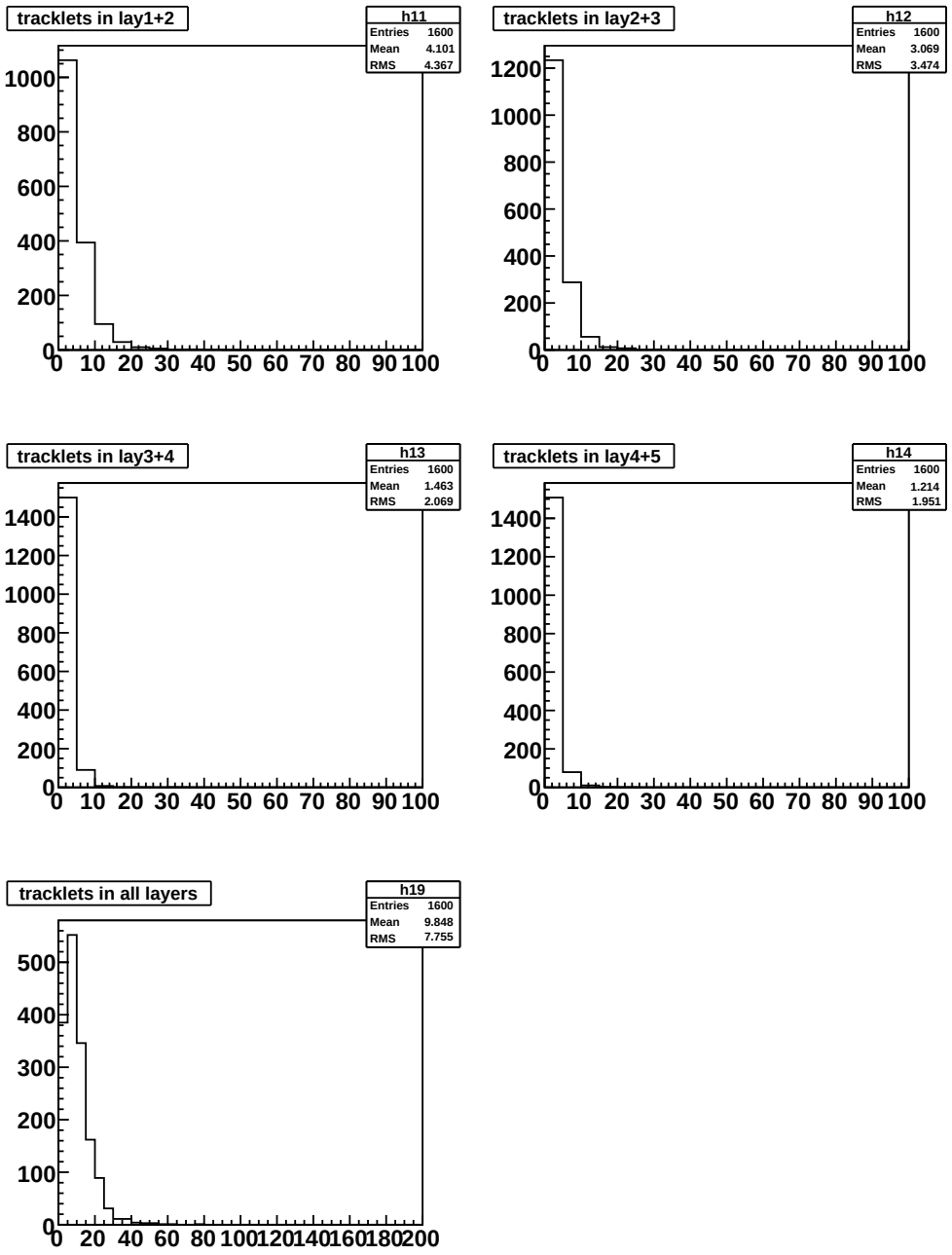


Figure 27: The number of tracklets that projects to one of the neighboring sectors.

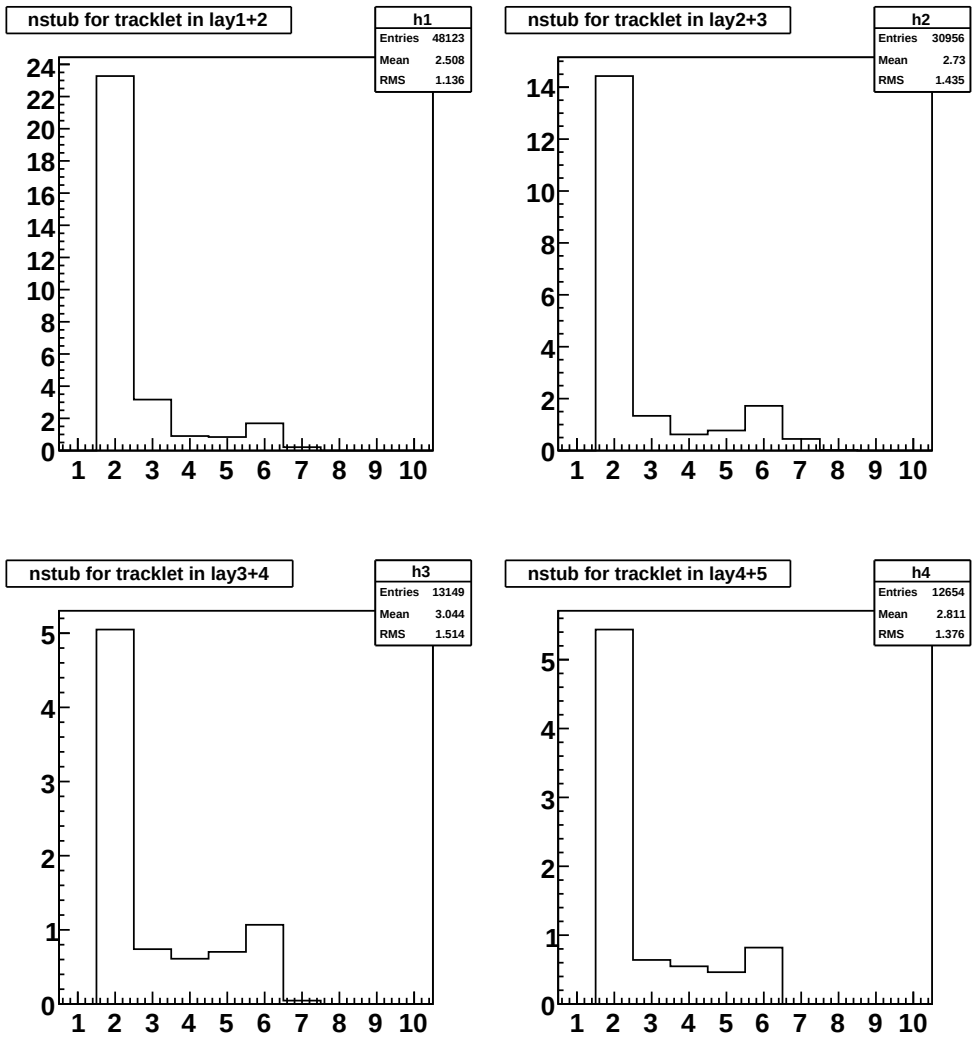


Figure 28: For tracklets found in different pairs of layers is shown how many stubs were matched to the tracklet when it is projected to the other layers. The plot includes also the stubs in the original tracklet such that the peak at 2 stubs means that when the tracklet was projected to the other layers no additional stubs were matched.

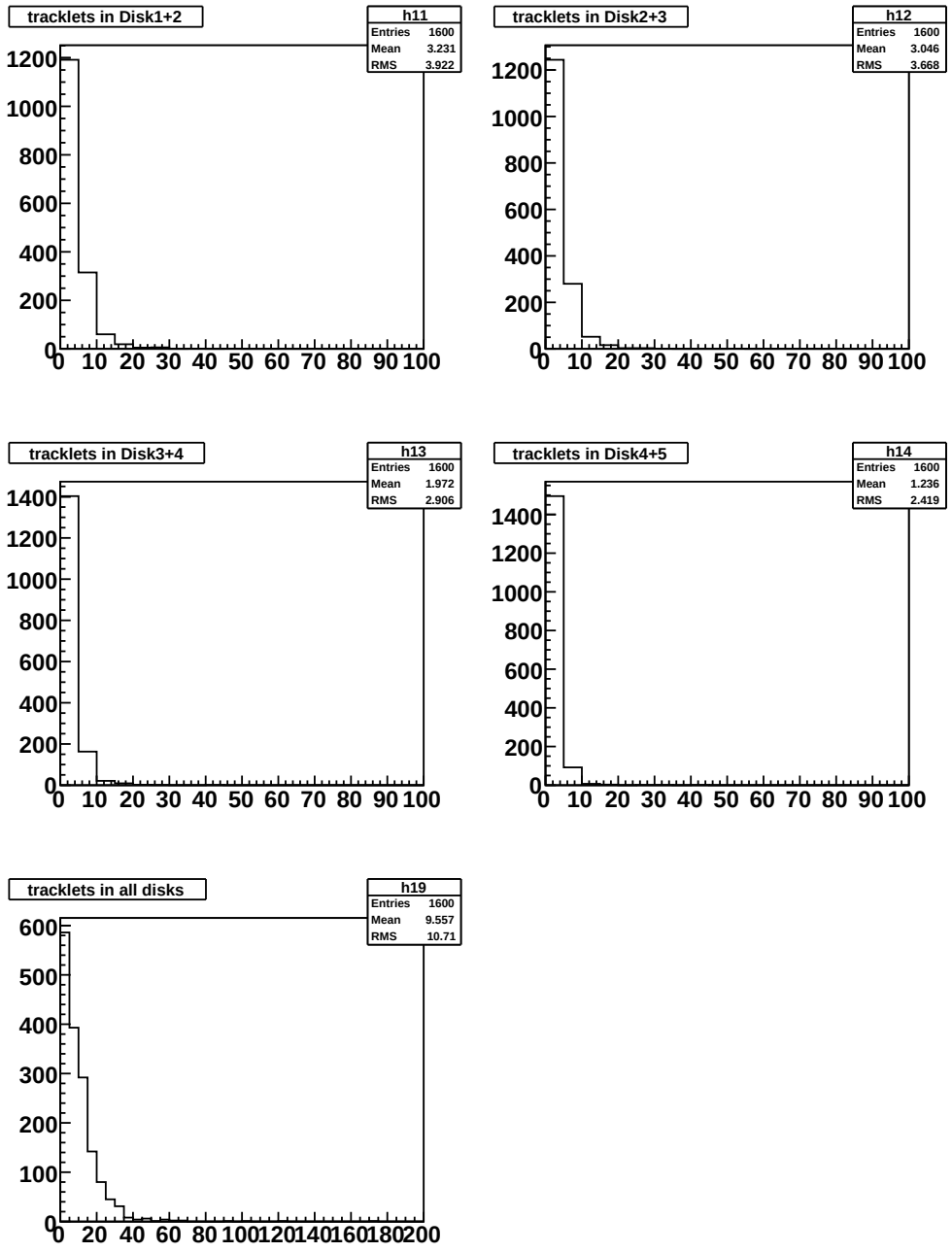


Figure 29: The number of tracklets found in the endcaps for different combinations of disks. The lowest histogram shows the sum of all stubs found in the sector.

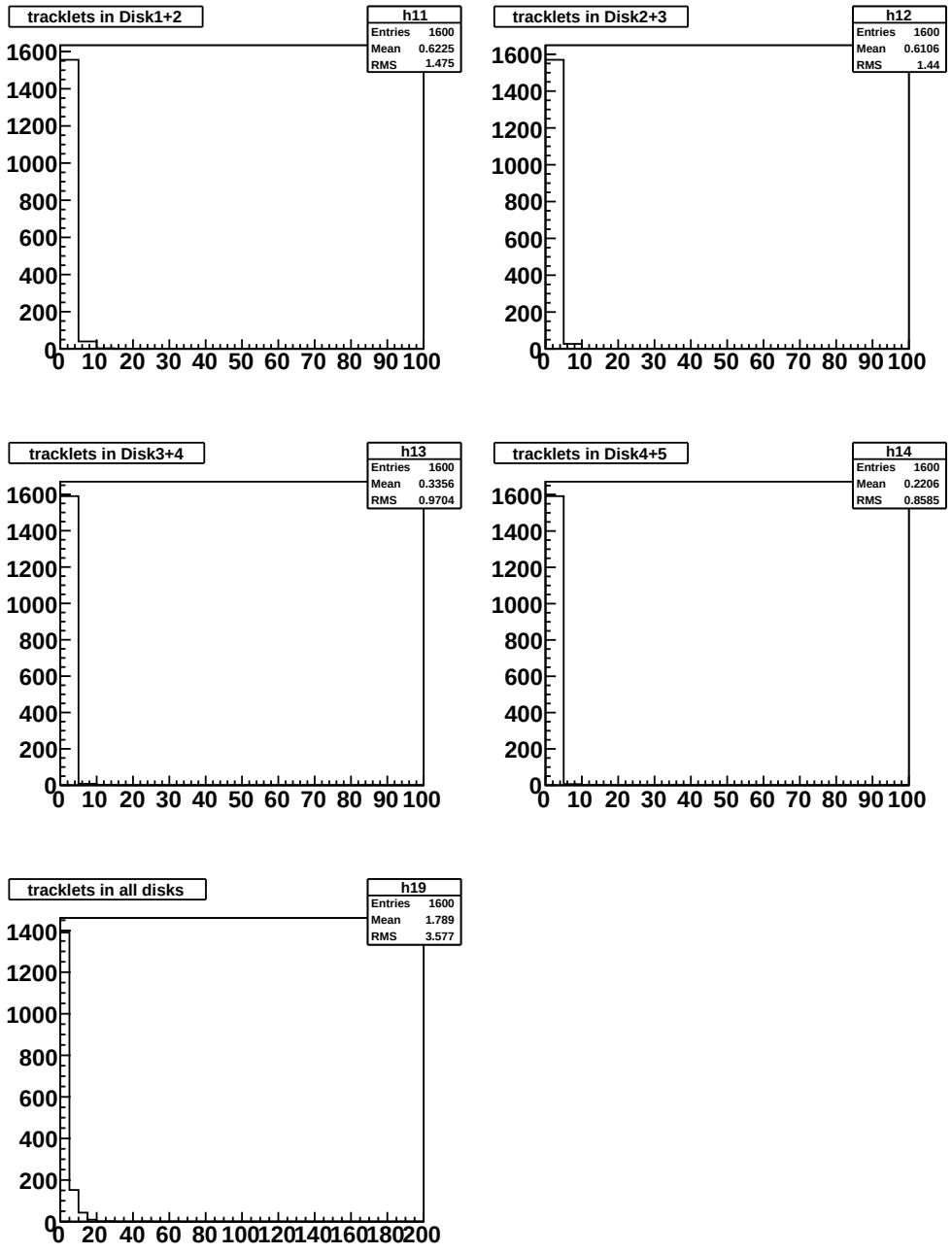


Figure 30: The number of tracklets that projects to one of the neighboring sectors for tracklets found in the endcaps, i.e. seeded by the disks.

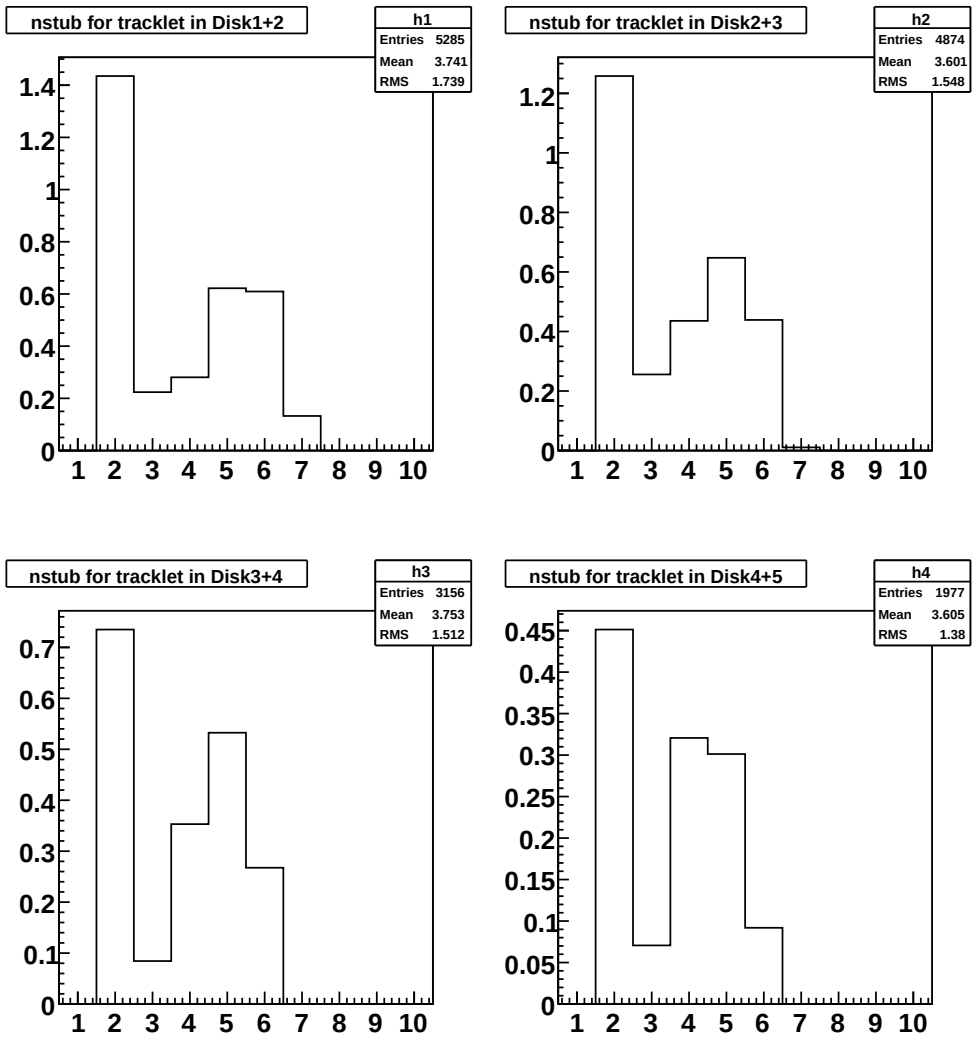


Figure 31: For tracklets found in different pairs of disks is shown how many stubs were matched to the tracklet when it is projected to the other layers and disks. The plot includes also the stubs in the original tracklet such that the peak at 2 stubs means that when the tracklet was projected to the other layers no additional stubs were matched.

Table 3: Estimate of latency of the track finding and fitting.

Task	Latency (25 ns BX)
Frontend readout and cables (From Francois)	42
FED (sorting data)	10
Sending data to sector boards	2
Receive data from FED and calculate global position ($r,\ \phi,\ z$)	3
Tracklet finding	2
Tracklet Projections	2
Mapping for matching	2
Transfer to neighboring sectors	4
Stub matching	6
Return matches	4
Trackfit	6
Local duplicate removal	6
Transfer overlap tracks	4
Final duplicate removal	6
Total	99

5.2.5 Latency

The latency of the trigger is a key performance parameter. In this section we will estimate the latency in the calculation on the sector board. Estimates of the latency for the different steps in the algorithm are shown in Table 3. Below each of these steps are discussed and the assumptions (or guesses) are motivated.

We first include 42 BXs for the frontend readout and fiber delays. This estimate is taken from Francois. Next we have the data sorting in the FED. The data will arrive avaraged over 8 BXs. We allow for 10 BXs in order to 'undo' this averaging. Then we allow for 2 BXs to send the data to the sector boards.

The first step in the actual track finding on the sector boards is to receive the data from the FEDs. The data from the FED basically consists of hardware addresses and pixel addresses for the stub. These hardware addresses will need to be translated to physical coordinates, ($r,\ \phi,\ z$), for use in the tracking algorithm. Here we will have lookup tables foreach module that gives us the 'alignment' for the module such that we can calculate the physical coorinates as a linear (or possibly) quadratic function of the local chip coordiates. Each coordinate can be calculated in parallel so only 3 DSP cycles should be needed for this. Memory lookup for the alignment constants will take a 2 FPGA cycles, so the coordinate calculation should be accomplished in one BX. Then, based on the calculated coordinates, the stubs will need to be 'sorted' into some bins of ϕ and z and possibly duplicated. The exact steps here are not yet worked out for the BE, but we allow two BXs for accomplishing this. Should be sufficient as it only involves some simple comparisons.

The second step is the tracklet finder. This actually is two steps, first finding the combinations that form tracklets and then performing the steps that calculate the track parameters.

These steps have been prototyped for the LB algorithm. But the algorithm needed for the BE is very similar and should take two BXs.

After the tracklets are found we need to project the track to other layers or disks. The calculations required for a projection to on given layer or disk is less than 1 BX and all the different projections can be done in parallel. In our estimate we assume that this will take 2 BXs.

After the projection has been done we need to sort the projected tracklets so that we can route them to where we will do the stub matching. Some tracklets project to other sectors. This sorting involves simple comparison to put the data on different FIFOs. We allow two BXs for this step.

The next step involves sending the track projections to where the mapping is done. If the mapping is done in the same sector, i.e. the same FPGA, there is practically no delay, but if the projection is to a nearby sector then we assume that this incures a 4 BXs delay.

For the actuall matching step we are just now working out this algorithm. For this estimate of the latency we assume this can be done in 6 BXs, but hopefully this will be faster.

After the matching has been done the information has to be sent back to the sector where the tracklet was found. Again we allow for 4 BXs for this data transfer.

Having the tracklet and the matched stub we then perform the trackfit. Each track parameter requires only 4 multiply-add operations and can be done in just a DSP cycles. This step will also need to select the best stub match in each layer in case of multiple candidate stubs. We allow for 4 BXs for this step.

Within the sector we will need to remove duplicate tracks. We allow 6 BXs for this step. (Details needs to be worked out for a more solid estimate.)

We also can have duplicates between tracks founds in different sectors to remove these duplicates we send the tracks that overlap with the neighboring sector to the nearby sector. We allow 4 BXs for this transfer.

Last we have a final step of duplicate removal, again we allow 6 BXs for this. (Again this needs more detailes to be worked out.)

The total of this is 99 BXs, or 2.5 μs.

6 FPGA Implementation of Algorithm

In this section we discuss some of the details of how to translate the algorithm described in Sec. 3 into something that can be run in an FPGA. This section contains a description of an implementation for the BE geometry.

6.1 Coordinate calculation

The raw stubs arrive at the DCTs and the physical coordinates needs to be evaluated. The exact ϕ position of stubs are used by the DCT to route stubs to the correct sector, and the

Table 4: Estimate of number of bits required for the constants when calculating the physical coordinates for a barrel module.

Constant	Number of bits
r_0	8
α_r	8
x_0	8
ϕ_0	12
α_ϕ	12
z_0	12
α_z	6
Total	66

global coordinates are then used by the sector processors in the track finding algorithm. The formats used for the stubs are described in Sect. 7.3.

The steps required to calculate the physical coordinates are described here. It is assumed that the data format is such that the local x (global ϕ for barrel modules and disks) and y (global z for barrel modules and r for disks) are given as integers. The x coordinate will be 10 bits and the y 5 bits. (This is for the more complicated case of the PS modules; for the 2S modules only two bits are used for the local y coordinate.)

First we consider the calculation of the positions for a barrel module

$$\begin{aligned} r &= r_0 + \alpha_r(x_0 - x_i)^2 \\ \phi &= \phi_0 + \alpha_\phi x_i \\ z &= z_0 + \alpha_z y_i \end{aligned}$$

In Table 4 is estimated the number of bits required on the alignment constants for calculating the physical coordinates. The exact number of bits is probably not very relevant. Originally (when we thought this was done on the sector boards) we had to store these constants in memory and look up the constants based on the module number. However, now that this is done in the DCT we would likely have one processing pipeline to calculate the global coordinates per input from a DCT. Hence we would not need to look them up in memory, but simple store them in registers for each imput link.

For disk modules we need to do something similar. We assume we can parameterize the physical coordinates using

$$\begin{aligned} r &= r_0 + \alpha_r y_i + \beta_r(x_i - x_0)^2 \\ \phi &= \phi_0 + \alpha_\phi(x_i - x_0)(\beta_\phi - y_i) \\ z &= z_0 \end{aligned}$$

Estimates of the number of bits required for the various constants are given in Table 5. There are relations between some of the parameters that could be explored in order to reduce the

Table 5: Estimate of number of bits required for the constants when calculating the physical coordinates for a disk module.

Constant	Number of bits
r_0	12
α_r	6
β_r	6
x_0	8
ϕ_0	12
α_ϕ	12
β_ϕ	5
z_0	8
Total	69

data volume, but it would come with a price of more complicated calculations. Since the data fits within 72 bits we use the paramterization that duplicates some data. Again, the number of bits needed is probably irrelevant.

The calculated global coordinates are on the format given in Sect. 7.1.2.

6.2 Example of tracklet formation combinatorics

This section shows example of plots for the combinatorics of stubs and tracklets in a sample sector. In this example the detector is divided into 28 ϕ sectors and the full barrel is considered. To break down the combinatorics each layer is divided into 'virtual' modules. Along z there layer is divided into 8 bins and in ϕ the odd layers are split into 6 bins. The even layers are divided into 7 bins.

In Fig. 32 is shown the occupancy in each sector as a function of the layer. In Fig. 33 is shown the occupancy in the virtual modules.

Tracklet candidates are then formed by combining all the tracklets in virtual modules that are consistend with having tracks with $p_T > 2$ GeV and $|z_0| < 15$ cm. In Fig. 34 is shown the number of tracklet candidates and in Fig. 35 is shown the total number of tracklet candidates in a layer.

The next step is to calculate the track parameters for the tracklet candidates. If you now require that the tracklet has $p_T > 2$ GeV and $|z_0| < 15$ cm and that the stubs p_T are consistent with the tracklet p_t we can significantly reduce the number of tracklets. This is illustrated in Figs. 36 and 37.

As can be seen the number of tracklet candidates are significantly reduced when the cuts (p_t, z_0 and stub consistency) are applied. To understand what the impact of each of these cuts are we remove one of them at the time. The plot without the p_t cut is shown in Fig. 38, without the z_0 cut in Fig. 39, and without the stub p_t consistency cut in Fig. 40. As can be seen from these figures the p_t is the most efficient cut in removing combinatorial background.

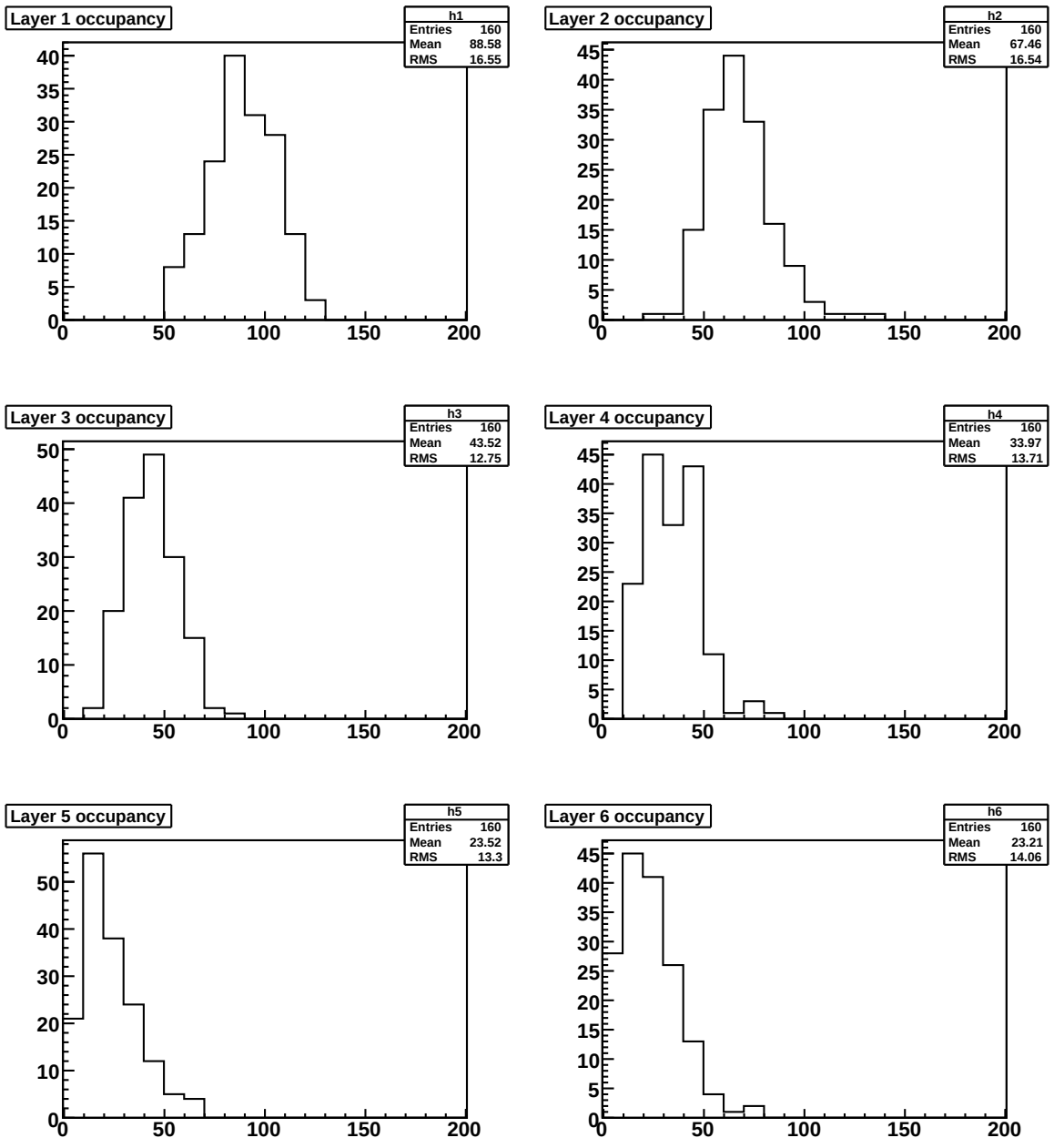


Figure 32: This plot shows, for each layer in the barrel, the stub occupancy in each of the 28 sectors.

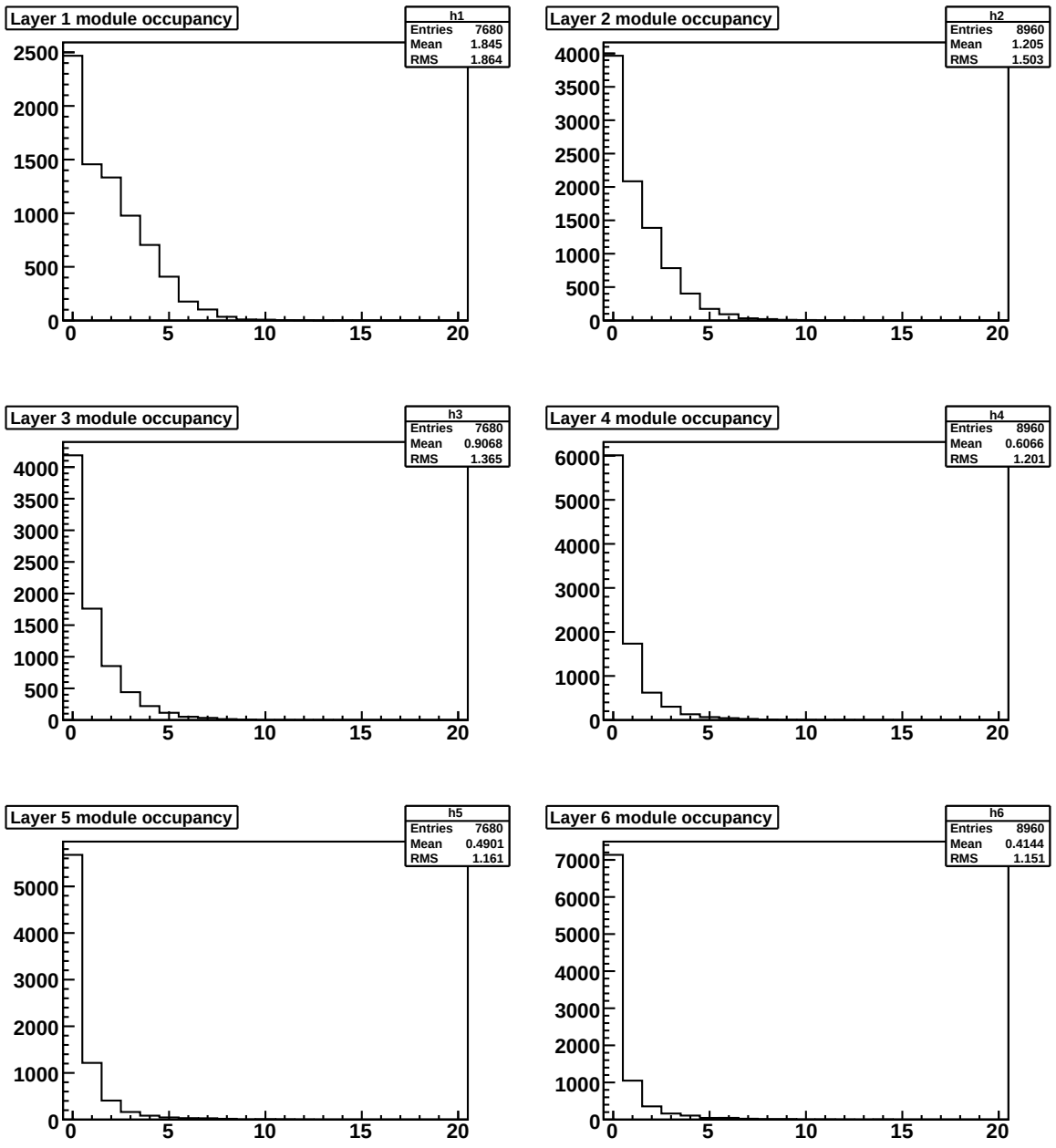


Figure 33: This plot shows the occupancy in each of the virtual modules, see description in the text.

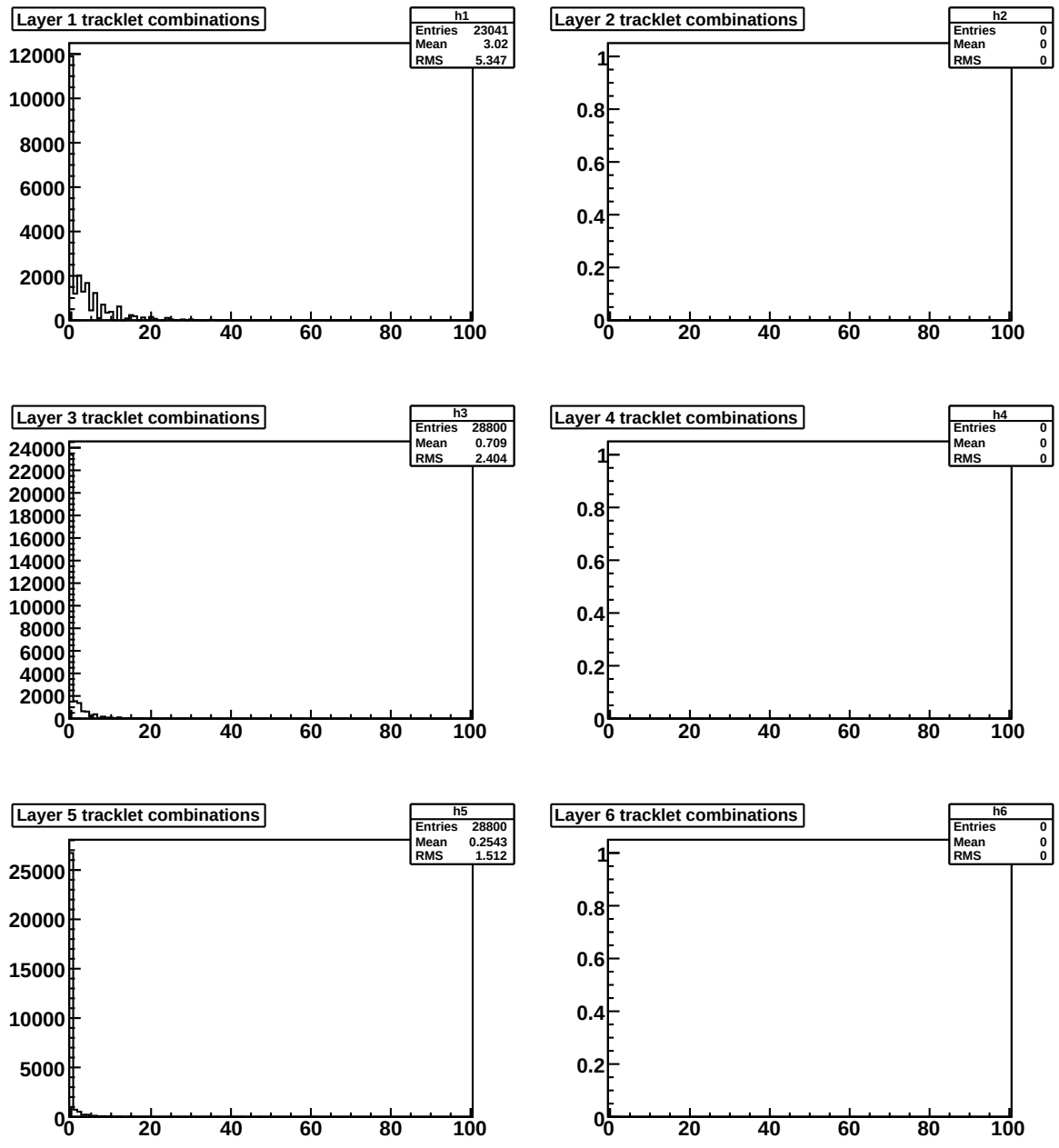


Figure 34: This plot shows the number of candidate tracklets between each pair of virtual modules that can produce a valid L1 track.

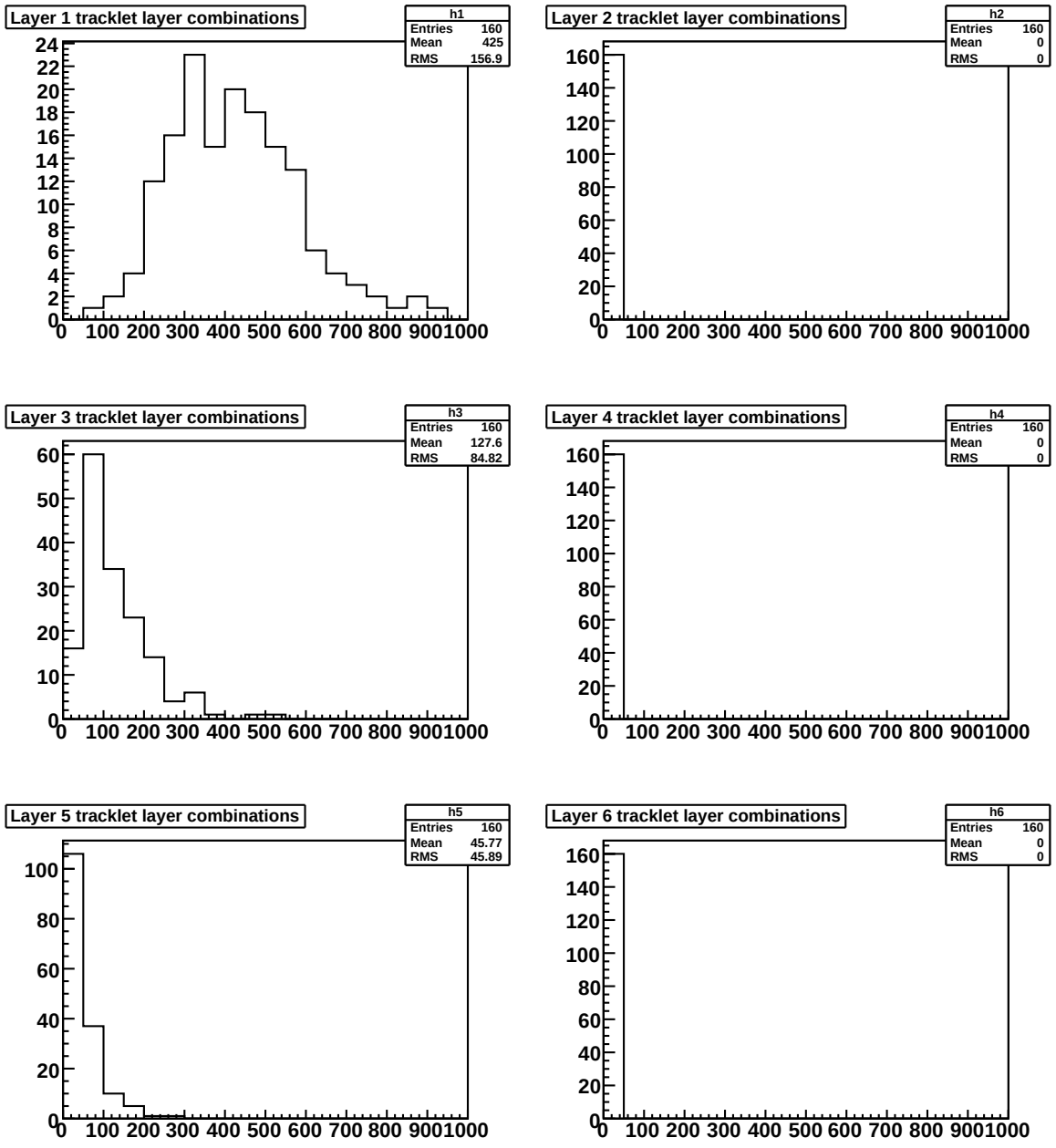


Figure 35: This plot shows the number of candidate tracklets for all virtual modules in the layer.

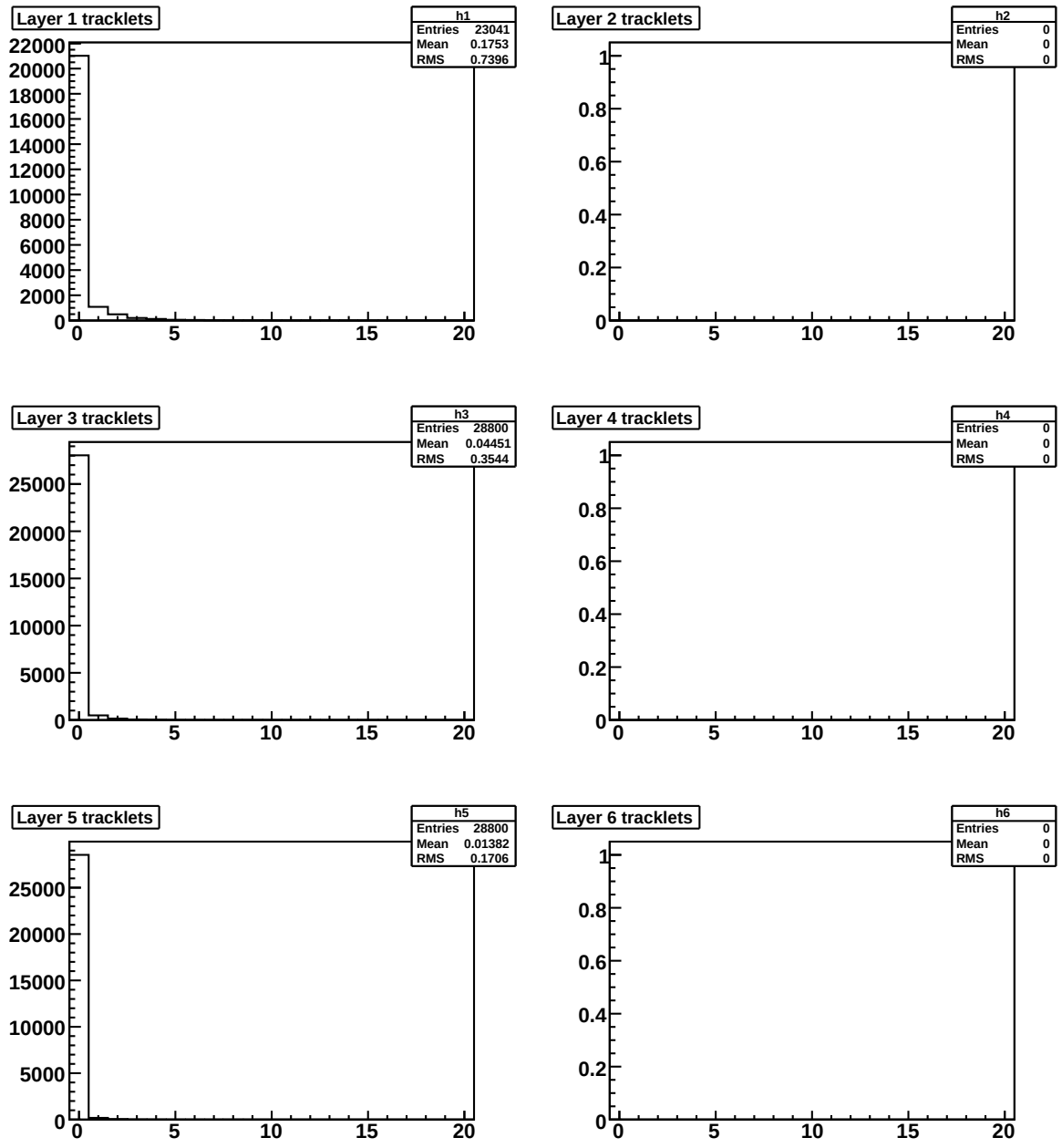


Figure 36: This plot shows the number of tracklets in each pair of virtual modules after the cuts (p_t, z_0 and stub consistency) on the tracklets have been applied.

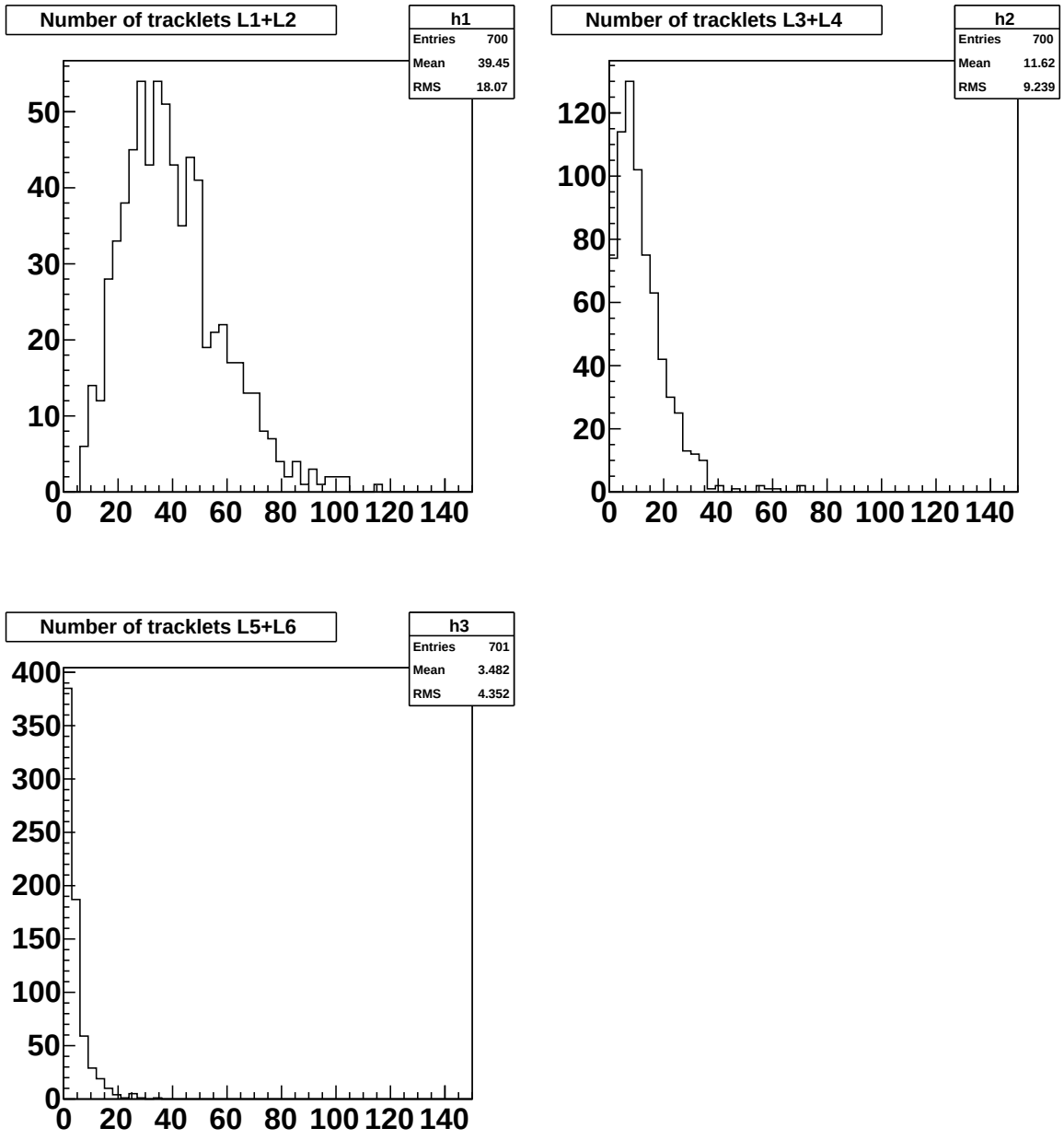


Figure 37: This plot shows the number of tracklets in all pairs of virtual modules after the cuts (p_t, z_0 and stub consistency) on the tracklets have been applied.

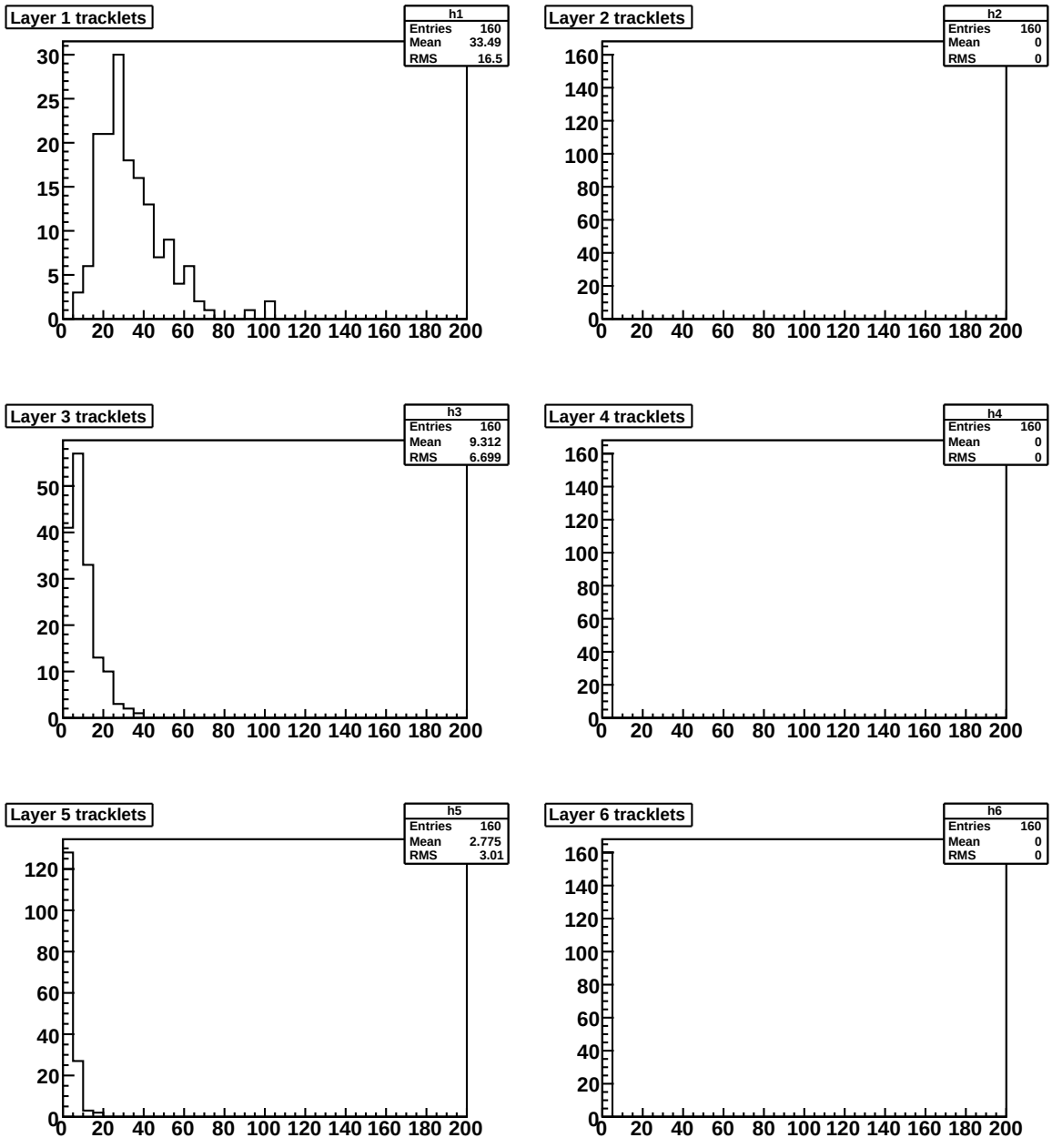


Figure 38: This plot shows the number of tracklets in all pairs of virtual modules without applying the p_t cut.

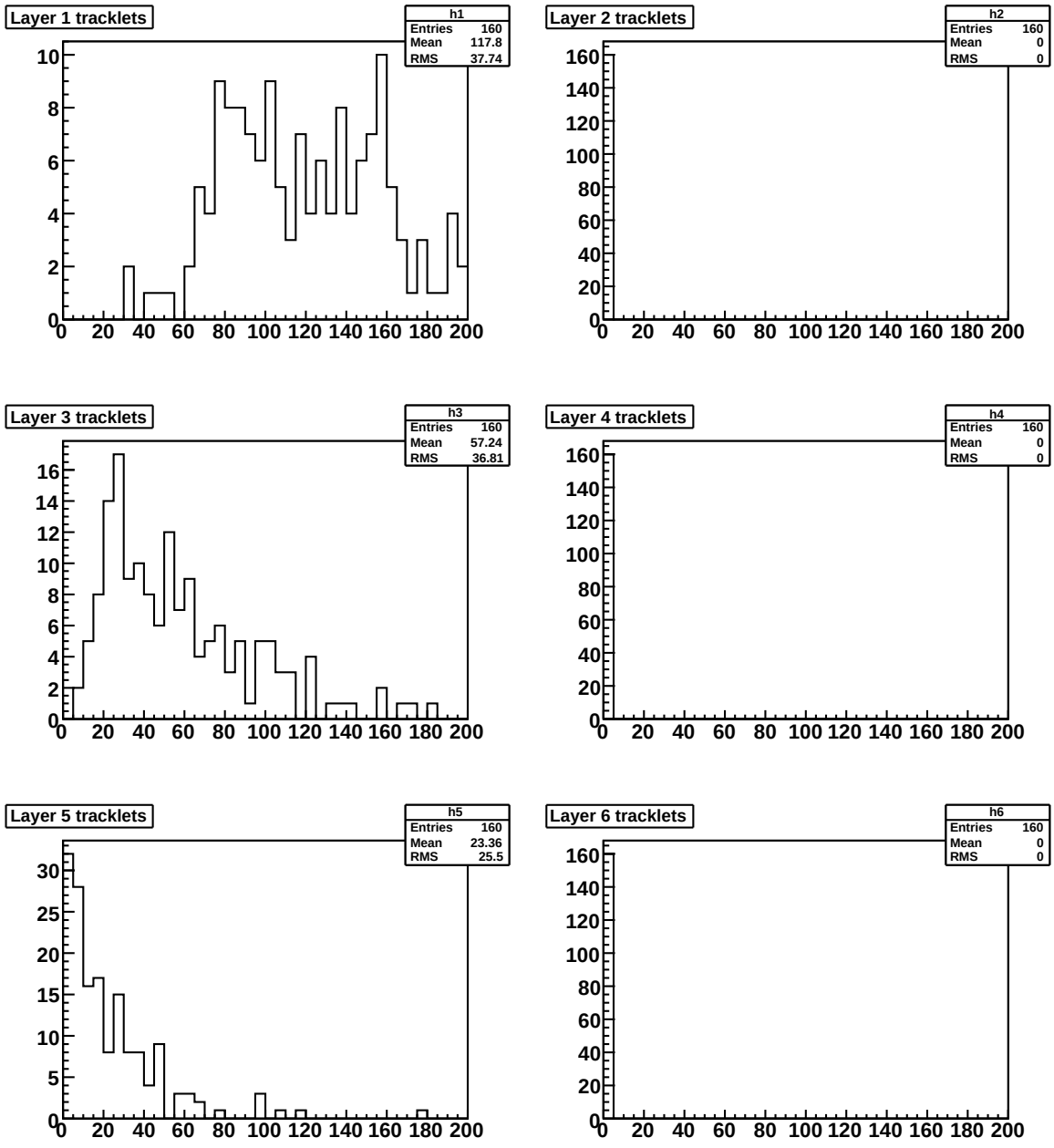


Figure 39: This plot shows the number of tracklets in all pairs of virtual modules without applying the z_0 cut.

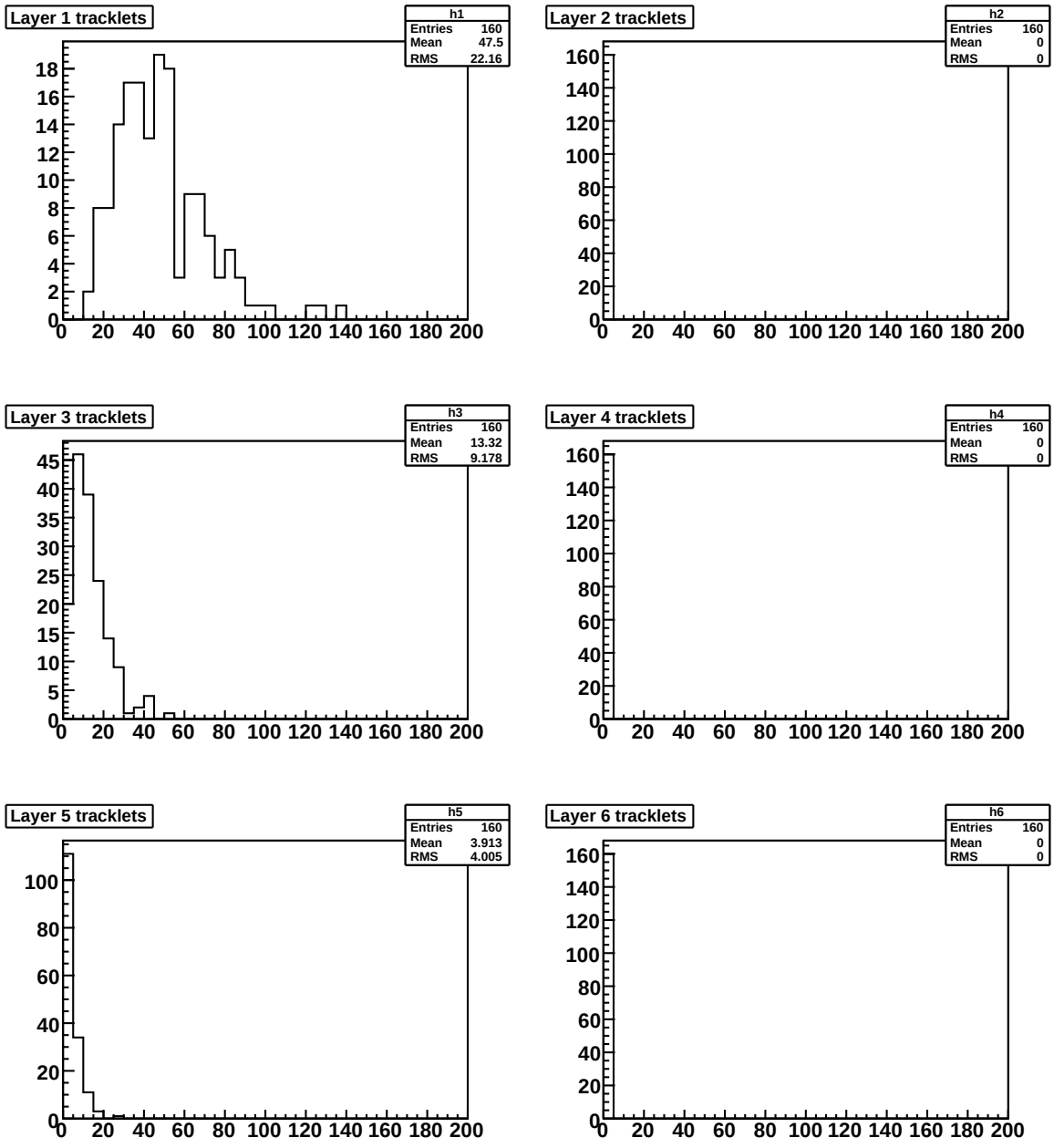


Figure 40: This plot shows the number of tracklets in all pairs of virtual modules without applying the stub p_t consistency cut.

7 Implementation of the track finding

This section describes the implemenation of the track finding and track fitting. A few assumptions are made here. First, we will only consider the barrel part of the detector. (We will add the forward disks later; most of the tools developed for the barrel will also work in the forward disks.) The detector is divided into 28 ϕ sectors, but each sector covers the full η range. We assume that the data are delivered from FEBs that are connected to the barrel sensors in 4 blocks of z as shown in Fig. 19. Further we will assume that the data are time multiplexed by a factor of 4. This means that the sector board we are looking at will process one out of 4 BXs. The implemenation so far does not address the duplicate/ghost removal.

The next section gives an overview of the processing steps in the implementation and then there is a section with some more detailed information about the buffer formats used. After this there is a section describing in more detail each of the processing steps.

7.1 Overview

The implemenation of the tracking is done in 8 processing steps; from the input where the stubs are delivered from the FEBs to the output where the final tracks are produced. Between the processing steps the data are written to memories where different blocks are used for different bunch crossings. In the figures below where the different processing steps are shows the memories that store the data between the processing steps are shown as rectangles and the processing steps are shown as squares.

In Fig. 41 is shown the first two processing steps where the data is received from the input and in step 1 sorted by the layer and in the second step sorted into the virtual modules. The next two steps are shown in Fig. 42 where in step 3 the tracklet engine forms the tracklet candidates and in step 4 the track parameters and projections of the tracklets are calculated. Figure 43 shows steps 5 and 6 where the projections are first sorted into the virtual modules and candidate matches are formed in the virtual modules. In the last two steps, illstrated in Fig. 44, the residuals are calculated and then used in the track fit.

7.1.1 Details about FIFOs and buffers

It is assumed that the data are arriving at the input with some form of timestamp. How this would be defined is not yet clear to me. The data will internally be tagged with a 3 bit 'time stamp'. As will be explained below this time stamp will be used to identify the event data in buffers.

The stubs arrive at the raw stubs FIFO with their time stamp. The sorting state machine that sorts stubs into layers process stubs with the current time stamp and write them into the 'raw stub layer' buffers. If data arrives in the raw stubs FIFO for the wrong BX the data will be ignored and an out-of-synch error raised. The data from the raw stub layer buffers are then collected into the 'combined raw stub layer' FIFOs. Then the data is piped through a fixed latency pipeline to calculate the global positions in the 'global stub' FIFOs.

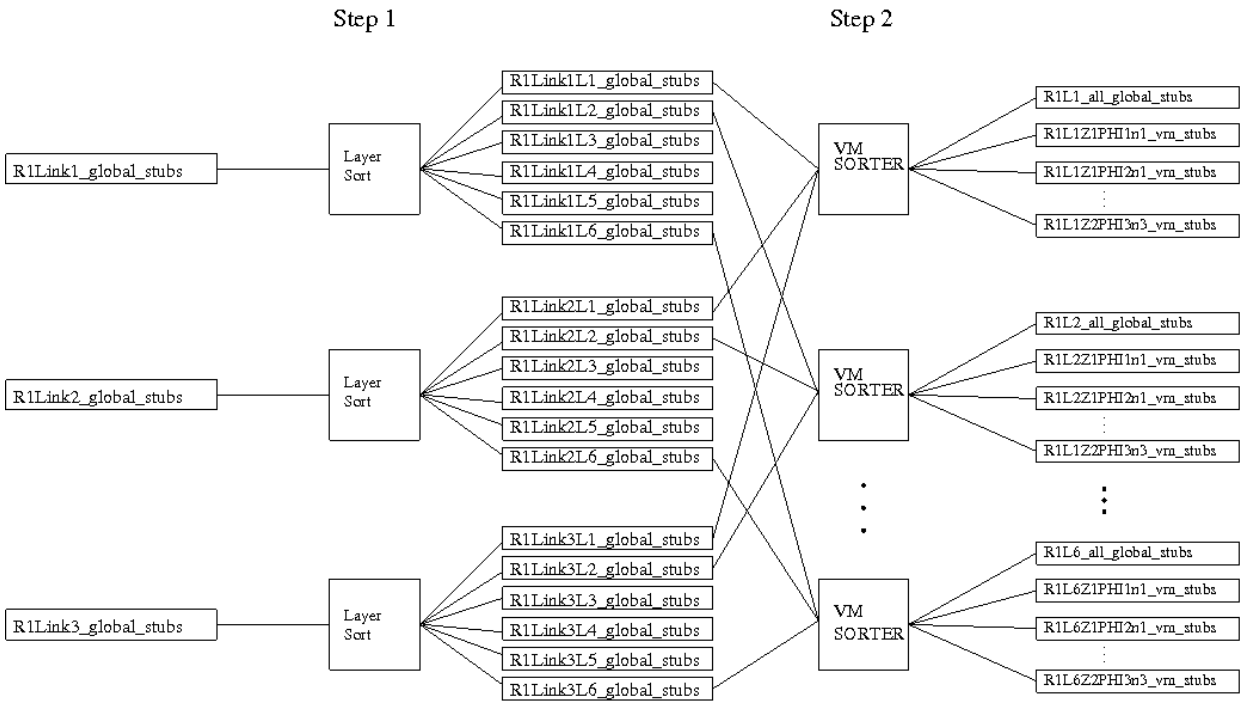


Figure 41: This figure illustrates the data arriving on three input links for one FED sector. The data on each linkt is sorted by layer number and in the second processing step the stubs are organized by the virtual module they belong to.

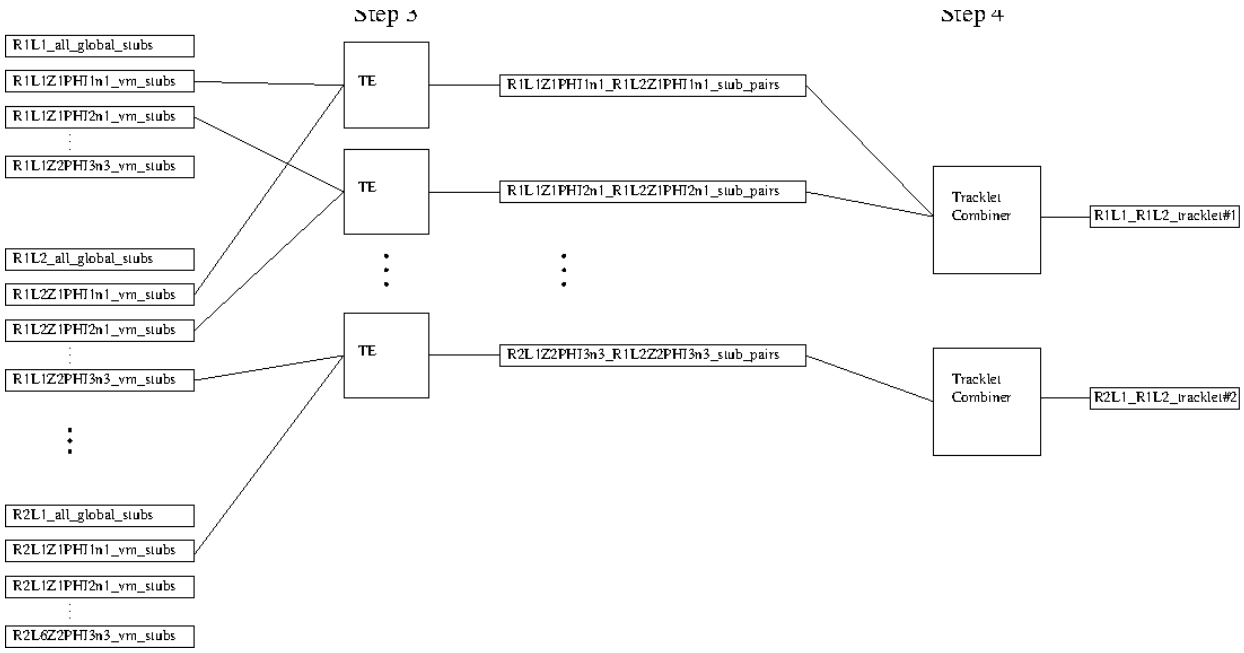


Figure 42: The stubs are combined to form candidate tracklets in the tracklet engine in step 3. The output is combined and the precise track paramaters and projections are calculated in step 4.

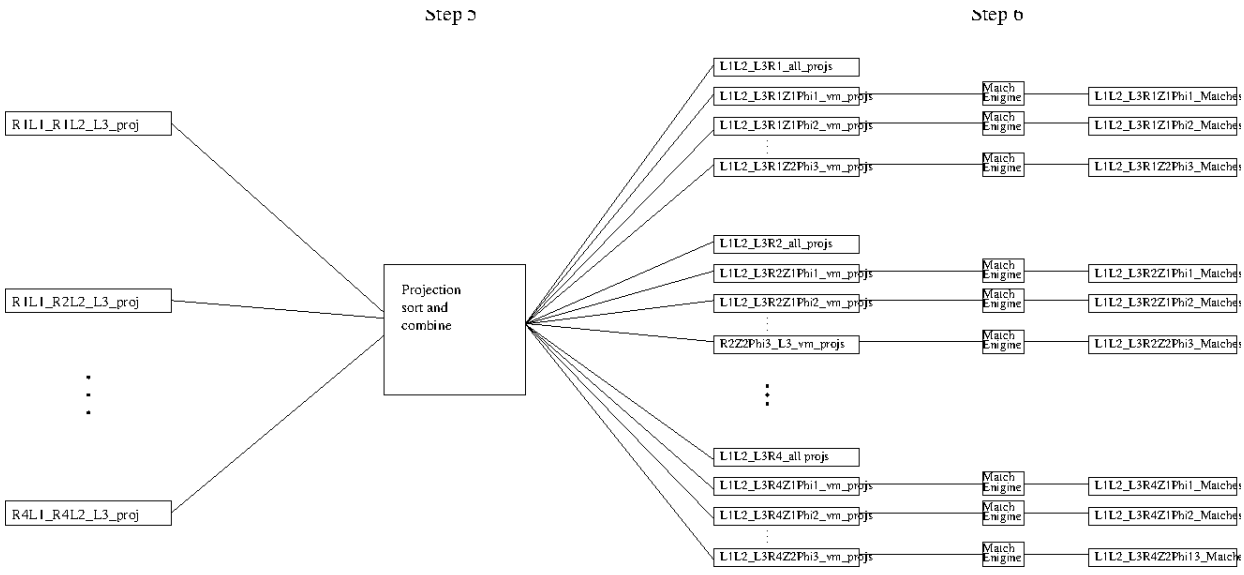


Figure 43: This figure illustrates how projections to a given layer are sorted in step 5 based on FED region and virtual module. The projections to a virtual module are processed in the match engine in step 6 to find approximate tracklet projection matches with stubs.

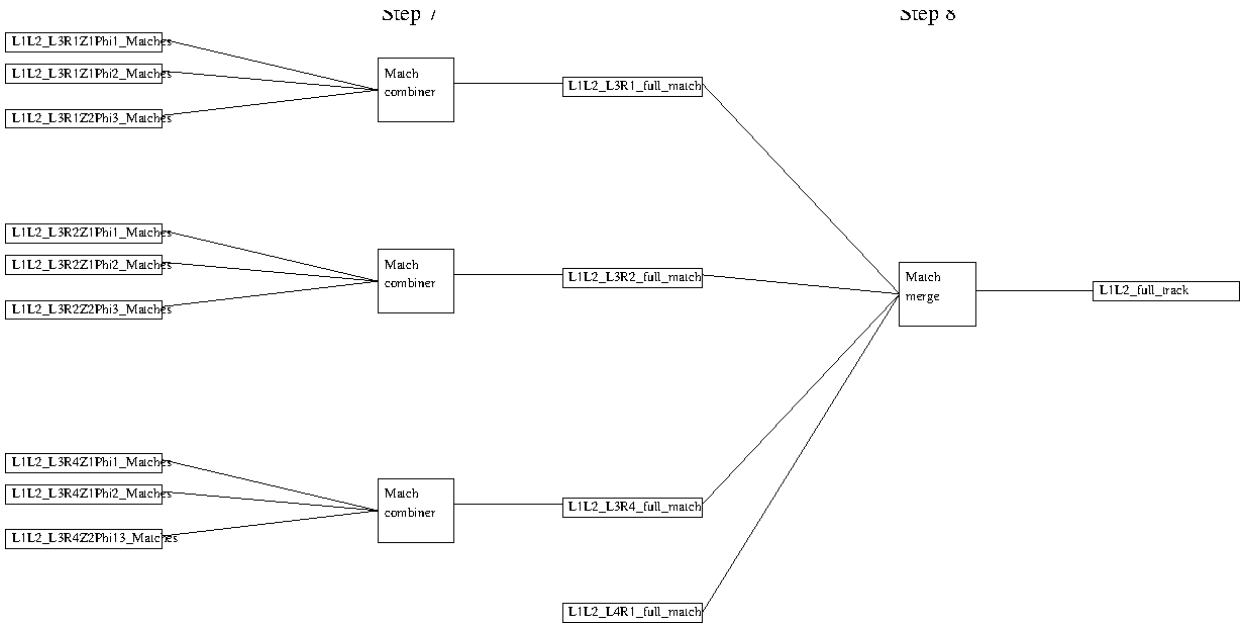


Figure 44: This figure illustrates how tracklet match candidates are combined and the detailed calculation of the residuals are performed in step 7. In step 8 the matches to stubs formed from L1+L2 are combined to form a track and this track is then fit to obtain the final track parameters.

Based on the global positions the stubs are first sorted based on the z into the 'stub z-split stub' FIFO and after this the go to the 'global stub z and ϕ split' buffers.

Here pairs of these buffers are processed to look for tracklets. Tracklets that pass the selection criteria are put into the 'tracklet' buffer.

Between each of these levels of buffers or FIFOs are state machines that makes the decisions or calculations to propagate to the next level. Each of these are described in the subsequent sections.

To simplify the data handling (and debugging) a common format is used for the different stages in the processing. First, recall that we are assuming a factor of 4 time multiplexing so that a new crossing needs to be processed every 100ns. This will correspond to somefixed number of clock cycles, but this is not known yet. It is assumed that the input data arrives into the FPGA in a FIFO. This FIFO contains header and trailer information that labels the crossing the data belongs to. However, for the subsequent processing steps we do not use FIFOs for the data storage, but rather memories that are used as a form of cicular buffers. All memories are assumed to be 512 slots deep and we partition the memory such that we use 64 memory slots for each bunch crossing. The addressing scheme is such that the highest 3 bits are taken from the current bunch crossing and the lower 6 bits from a counter that keeps track of the number of objects in the current bunch crossing. On the last clock for a given bunch crossing the counter value is written to the last memory address (0x3f) for the bunch crossing. In most cases this count will not be used, and could be optimized away. However, for debugging it is convenient to have this count. For most use cases, when the data in a memory is needed directly in the next processing step the counter is past on the last clock to the next processing step. Though there are cases, such as the stub-tracklet matching step when the number of objects in the list needs to be restored.

A detailed emulation of this algorithm is currently being implemented. The emulation of the algorithm produces the exact format of the data in the memories of each stage. In the description below examples of the data content is taken from the emulation code to illustrate the formats.

7.1.2 Step 0: Input data

The input data to the sector processing comes from the FEBs. The stub data has been formatted to correspond to the global coordinates used in the track finding. The data is received by the sector processor and stored in a FIFO which is read during event processing.

The data in the FIFO for one BX consists of the header, the payload, and the trailer. The header consists of two words, the first word identifies a new event and the contains the marker (six 1's in the highest bits of the word) and the bunch crossing number as described in Table 6. The next word contains the number of stubs in each layer. Six bits is used per layer and allows for a maximum of 63 stubs in a layer. (In reality the system will have other limitations that reduce the number of stubs.) The format of the second word is shown in Table 7 for barrel hits and in Table 8 for disk hits..

The actual payload, the stubs, then follows. Each stub is encoded in 36 bits and contains

Table 6: First word of the header for the input stubs.

Quantity	bits
Marker (0b111111)	35:30 (6 bits)
BX	29:22 (8 bits)
0's	21:0 (22 bits)

Table 7: Second word of the header for the input stubs in barrel.

Quantity	bits
Hits in L1	35:30 (6 bits)
Hits in L1+L2	29:24 (6 bits)
Hits in L1+L2+L3	23:18 (6 bits)
Hits in L1+L2+L3+L4	17:12 (6 bits)
Hits in L1+L2+L3+L4+L5	11:6 (6 bits)
Hits in L1+L2+L3+L4+L5+L6	5:0 (6 bits)

Table 8: Second word of the header for the input stubs in disks.

Quantity	bits
Hits in D1	29:24 (6 bits)
Hits in D1+D2	23:18 (6 bits)
Hits in D1+D2+D3	17:12 (6 bits)
Hits in D1+D2+D3+D4	11:6 (6 bits)
Hits in D1+D2+D3+D4+D5	5:0 (6 bits)

Table 9: The bit assignment for the input stubs in global coordinates for hits in the barrel.

Quantity	Layer 1-3	Layer 4-6
r	35:29 (7 bits)	35:28 (8 bits)
z	28:17 (12 bits)	27:20 (8 bits)
ϕ	16:3 (14 bits)	19:3 (17 bits)
Stub p_{T}	2:0 (3 bits)	2:0 (3 bits)

Table 10: The bit assignment for the input stubs in global coordinates for hits in the disks.

Quantity	PS modules	2S modules
r	33:24 (10 bits)	36:27 (10 bits)
z	23:17 (7 bits)	26:20 (7 bits)
ϕ	16:3 (14 bits)	19:3 (17 bits)
Stub p_{T}	2:0 (3 bits)	2:0 (3 bits)

the position, r, z, and ϕ and also the stub p_{T}. The format for each stub is described in Table 9 for barrel stubs and in Table 10 for hits on the disks. The stubs are sent ordered by layer; the stubs in layer 1 are first and layer 6 is last. This sorting has to be guaranteed by the the FEB.

Last we have the trailer. It is similar to the header; it repeats the marker bits and the bunch crossing and then follows with 1's to fill out the rest of the word as shown in Table 11.

An example of the data in the input FIFO is given in Table 12. This example contains two events for bunch crossings 0 and 1.

7.1.3 Step 1: Layer sorter

This step reads in the stubs from the input fifo and writes the to the output memories based on the layer of the stub.

When a new bunch crossing arrives the layer sorter reads the header and verifies the bunch crossing for the three least significant bits. After this it reads the next word that contains the counts of stubs in each layer. Then it proceeds to read the stubs, layer-by-layer, until all stubs have been processed. Then the trailer is read. The bunch crossing is again verified.

The output format contains the same information as the input stubs, but, largely for

Table 11: Trailer word for input stubs.

Quantity	bits
Marker (0b111111)	35:30 (6 bits)
BX	29:22(8 bits)
1's	21:0 (22 bits)

Table 12: Example data in input FIFO. The example below contains data for 2 events.

data here...

Table 13: The bit assignment for the global stubs

Quantity	Layer 1-3	Layer 4-6
Stub p_{T}	35:33 (3 bits)	35:33 (3 bits)
r	32:26 (7 bits)	32:25 (8 bits)
z	25:14 (12 bits)	24:17 (8 bits)
ϕ	13:0 (14 bits)	16:0 (17 bits)

historical reasons, we store the data in a different bit order. The bit format is explained in Table 13 and an example of the data is given in Table 14.

7.1.4 Step 2: Virtual module sorter

In this step we will combine the stubs within each FED region and layer into one memory and sort the stubs into virtual modules.

For each layer the virtual module sorter will read the input from three memories, corresponding to the different input links. All stubs are copied in the the 'All_Stubs' memory and a reduced format is written in the virtual module memories. The sorting into the virtual modules just requires looking at the two highest bits in ϕ for the even layers and the three highest bits for the odd layers and the third most significant bit of the z coordinate. See discussion in Sect. **??**. The reduced data format for the VM stubs uses 18 bits. First we use 6 bits to index into the 'All_Stubs' memory to have a link back to the full stub. Then we store 3 bits of ϕ and 4 bits of z and 2 bits of r position within the VM. Last 3 bits for the stub p_{T} is retained. The exact bit format is shown in Table 15. An example is given in Table 16.

Since the content of the vm_stubs will be used also much later in the tracklet-stub matching we need to store the number of stubs in each vm. We do this by storing in memory location 0x3F the stub count. (This is really only needed for the memories that are used for tracklet-stub matching...)

On the last clock in the BX pass the counts of objects in the vm_stubs memory to the tracklet engines.

7.1.5 Step 3: Tracklet engine

The Tracklet Engine (TE) forms pairs of stubs in two virtual module memories from two adjacent layers. After reading the stubs bits from the ϕ, z, and r positions of the stub, as well as the stub p_{T} are extracted and used to form two addresses used in two lookup tables to check if this pair of stubs are consistent with a track with $p_{\mathrm{T}} > 2$ GeV and $|z_0| < 15$ cm. If the pair of stubs pass the lookup test then the addresses (index) into the all stubs

Table 14: Examples of stubs in all global stubs buffers after step 2. The stubs are stored by the bunch crossing such that the three highest bits in the address are the BX. In memory location 0x3f we store the number of stubs after the processing step that writes to this memory is done.

BX=0x000 R1L1 0x00 000—0001000—000111101010—01001011100100
BX=0x000 R1L1 0x01 000—0001000—000001110010—01011001001000
BX=0x000 R1L1 0x02 000—0001000—001001101010—01000101100100
BX=0x000 R1L1 0x03 000—0010100—001111101010—00111010101000
BX=0x000 R1L1 0x04 001—0011000—000010001000—10001011011100
BX=0x000 R1L1 0x05 000—0001010—001010010110—01110111001100
BX=0x000 R1L1 0x06 000—0010111—001011001100—01111101000100
BX=0x000 R1L1 0x07 010—0010111—001111011010—01111100111000
BX=0x000 R1L1 0x08 000—0001111—001001011010—10111110100000
BX=0x000 R1L1 0x09 000—0010010—001101011010—11010100000100
BX=0x000 R1L1 0x0a 011—0001110—001100101000—10101100011100
BX=0x000 R1L1 0x0b 001—0011111—001100111100—11010101001100
BX=0x000 R1L1 0x0c 000—1000100—001011100010—11011000000000
BX=0x000 R1L1 0x0d xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
...
BX=0x000 R1L1 0x3e xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
BX=0x000 R1L1 0x3f 00000000000000000000000000000001101

Table 15: The bit assignment for the global stubs after we have sorted into z and ϕ bins.

Quantity	Bit assignment
Stub p_t	17:15 (3 bits)
Stub index	14:9 (6 bits)
z	8:5 (4 bits)
ϕ	4:2 (3 bits)
r	1:0 (2 bits)

Table 16: Examples of stubs in the virtual module stub buffers after step 2. (Needs to be updated to reflect new bit assignements.)

BX=0x000 R1L1Z1PHI1 0x00 000—000000—1111—010—00
BX=0x000 R1L1Z1PHI1 0x01 000—000001—0011—011—00
BX=0x000 R1L1Z1PHI1 0x02 xxxxxxxxxxxxxxxxxx
...
BX=0x000 R1L1Z1PHI1 0x3e xxxxxxxxxxxxxxxxxx
BX=0x000 R1L1Z1PHI1 0x3f 000000000000000010
BX=0x000 R1L1Z2PHI1 0x00 000—000010—0011—010—00
BX=0x000 R1L1Z2PHI1 0x01 000—000011—1111—001—00
BX=0x000 R1L1Z2PHI1 0x02 xxxxxxxxxxxxxxxxxx
...
BX=0x000 R1L1Z2PHI1 0x3e xxxxxxxxxxxxxxxxxx
BX=0x000 R1L1Z2PHI1 0x3f 000000000000000010

Table 17: The bit assignment for the stub pairs formed by the TEs.

Quantity	Bit assignment
Inner Layer Index	11:6 (6 bits)
Outer Layer Index	0:5 (6 bits)

memories are saved in the output. The format of these stub pairs is explained in Table 17. An example of this data is shown in Table 18.

On the last clock in the BX pass the counts of stub pairs to the tracklet combiner.

7.1.6 Step 4: Tracklet combiner

The stub pairs found by the TEs are combined in the Tracklet combiner and the precise tracklet track parameters are calculated using the full stub information. If the stub $p_{\mathrm{T}} < 2$ GeV and $|z_0| < 15$ cm then we also calculate the projections to the other layers and save the output to the track projections. The calculation of the tracklet track parameters are detailed in Sect. **??** and the calculation of the projections in Sect. **??**.

The format of the tracklet track parameters is detailed in Table 19 where the format of the projections are given in Table 20. The tracklet parameters contain the information to identify the stubs used to form the tracklet. This is done by specifying the FED region and the stub index into the All_Stubs memory. We use 3 bits for the FED region, this will allow for handling forward disks, and 6 bits for the stub index. The track parameters use a total of 54 bits, and each projection use 36 bits.

On the last clock in the BX pass the count of tracklets to the projection sort and combine.

Table 18: Examples of stubs pairs found in step 3.

BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000100000000xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000100000001xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000100000010xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000101000000xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000101000001xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000101000010xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000110000000xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000110000001xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x01 000110000010xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x09 xxxxxxxxxxxxxxxxxx
...
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x3e xxxxxxxxxxxxxxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z1PHI1 0x3f 000000000000001001
BX=0x000 R1L1Z2PHI1_R1L2Z2PHI1 0x01 000100000011xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z2PHI1 0x01 000101000011xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z2PHI1 0x01 000110000011xxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z2PHI1 0x03 xxxxxxxxxxxxxxxxxx
...
BX=0x000 R1L1Z2PHI1_R1L2Z2PHI1 0x3e xxxxxxxxxxxxxxxxxx
BX=0x000 R1L1Z2PHI1_R1L2Z2PHI1 0x3f 000000000000000011

Table 19: The bit assignement for the full tracklet including the projections to other layers.

Quantity	Bit assignment
Inner FED region	17:15 (3 bits)
Inner Layer Index	14:9 (6 bits)
Outer FED region	8:6 (3 bits)
Outer Layer Index	5:0 (6 bits)
Signed rinv	53:40 (14 bits)
ϕ_0	39:23 (17 bits)
z_0	22:13 (10 bits)
t	12:0 (13 bits)
Projection 1	35:0 (36 bits)
Projection 2	35:0 (36 bits)
Projection 3	35:0 (36 bits)
Projection 4	35:0 (36 bits)

Table 20: The bit assignement for tracklet projections

Quantity	Layers 1-3	Layers 4-6
ϕ position	44:31 (14 bits)	44:28 (17 bits)
z position	30:19 (12 bits)	27:20 (8 bits)
ϕ derivative	18:10 (9 bits)	19:9 (11 bits)
z derivative	9:0 (9 bits)	8:0 (8 bits)

Table 21: The bit assignment of the reduced virtual module projection.

Quantity	Bit assignment
Index	12:7 (6 bits)
ϕ position	6:4 (3 bits)
z position	3:0 (4 bits)

7.1.7 Step 5: Projection sort and combine

This step is similar to step 2 where the stubs were sorted into the virtual modules. In this step we sort the projection into the virtual modules the project to. Note that with the seeding done in L1+L2, L3+L4, and L5+L6 a projection to a given layer, e.g., L4 can come from either seeding in L1+L2 or L5+L6. This is true for any layer; the projection can come from one of two possible seeding layers. We will consider projections to a given layer from two layers where the tracklets were formed.

The sort and combine module read projections from several tracklet combiners and writes them to an All_Projections memory where all the information about the projection is retained. In addition a reduced format is written to the VM_Projections. The VM_Projection contains six bits for the index, 3 bits for ϕ, and 4 bits for z as described in Table 21

7.1.8 Step 6: Match engine

The Match Engines (MEs) loop over pairs of stubs and projections in a virtual module and compare if they are close. If they are close a stub-projection pair is formed and saved on the output. The output contains the indices of the stub and projection as shown in Table 22.

qwert

Table 22: The bit assignment for the tracklet stub matches

Quantity	Bit assignment
Tracklet Index	11:6 (6 bits)
Stub Index	5:0 (6 bits)

Table 23: The bit assignment for the tracklet stub matches with residuals

Quantity	Bit assignment
Tracklet Index	35:26 (10 bits)
Stub Index	25:17 (9 bits)
ϕ residual	16:9 (8 bits)
z residual	8:0 (9 bits)

Table 24: The bit assignment for the full track.

Quantity	Bit assignment
p_{T}	125:111 (15 bits)
ϕ_0	110:95 (16 bits)
d_0	94:85 (10 bits)
t	84:71 (14 bits)
z_0	70:59 (12 bits)
χ^2	58:54 (5 bits)
Stub Index L1	53:45 (9 bits)
Stub Index L2	44:36 (9 bits)
Stub Index L3	35:27 (9 bits)
Stub Index L4	26:18 (9 bits)
Stub Index L5	17:9 (9 bits)
Stub Index L6	8:0 (9 bits)

7.1.9 Step 7: Match combiner

In this step the precise stub position and tracklet projection are retreived and using the projection derivatives the projection to the precise radial position of the stub is calculated. This involves a simple multiply and add operation as described in Sect. **??**. The residuals in ϕ and z are calculated. Note that the loop over the stubs is the inner loop; and for each projection (or tracklet) we keep just the closest match. If the match is close enough that we want to keep it we store the identifiers of the tracklet and the stub.

7.1.10 Step 8: Match sorter

In this step we collect all tracklets that had 2 or more projections matched with stubs. For tracklets with 2 or more stubs we perform a linearized track fits where the seed parameters from the tracklet are updated using the residuals that were calculated in the previous step. The details of the track fit are discussed in Sect. **??**. The format of the result from the track fit is shown in Table 24, and an example of the data is shown in Table **??**.

Table 25: Examples of tracks found in step 8.

BX=0x000 L5 0x00 111000110001000‖0001011110100011‖xxxxxxxxxx‖1111110110000‖100000000011‖00000‖001000‖000111‖001000011‖001000000‖001000000‖001000000
BX=0x000 L5 0x01 111000011111011‖0001011101011101‖xxxxxxxxxx‖1111110111001‖011111111100‖00000‖001011‖000111‖001000011‖001000000‖001000000‖001000000
BX=0x000 L5 0x02 111000011010111‖0001011101000010‖xxxxxxxxxx‖1111110110011‖100000000000‖00000‖001100‖000111‖001000011‖001000000‖001000000‖001000000
BX=0x000 L5 0x03 111000100001001‖0001011101111001‖xxxxxxxxxx‖1111110001111‖100000100000‖00000‖000011‖000110‖001001011‖001000111‖001000000‖001000000
BX=0x000 L5 0x04 111000110101001‖0001100000110101‖xxxxxxxxxx‖1111110011101‖100000001101‖00000‖000100‖000110‖001001011‖001000111‖001000000‖001000000
BX=0x000 L5 0x05 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
...
BX=0x000 L5 0x3f xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx000101

7.2 DCT Region Project

As a step towards demonstrating the implementation of the L1 tracking on hardware we are implementing the tracking for one DCT region. The different modules that are involved in this project are shown in Fig. 45.

7.3 Sector coordinates

This section describes the coordinate system used in the sector. The basic geometry of one sector is shown in Fig. 46. The inner most layer is divided into three virtual modules and the next outer layer consists of four virtual modules that are staggered by half a virtual module such that we have one half virtual module acutally from the neighboring sector. (The data here is duplicated to allow for having the stubs in a tracklet contained to one sector during the tracklet finding step.)

We now introduce a coordinate system for the ϕ coordinates in the sector such that for the N bits used to represent ϕ, we have 0 corresponding to the angle of

$$-\frac{1}{6}\frac{2\pi}{28}$$

and $2^N - 1$ corresponding to

$$\frac{7}{6}\frac{2\pi}{28}$$

This means for example that if the three most significant bits of the angle are either 011 or 100 then a stub in the innermost layer belongs to the central virtual module. Which allows for convenient sorting of stubs into virtual modules. With $N = 14$ bits the granularity of the ϕ position is

$$\delta\phi = \frac{2\pi}{21} \times 2^{-14} = 18\mu\text{rad}.$$

At a radius of 22 cm this corresponds to 4 μm position accuracy, at a radius of 50 cm this corresponsds to 9 μm accuracy. For the outer layers were we use 17 bits for ϕ we have a position accuracy of 2.5 μm. *This precision is not actually needed should be optimized.*

For the z position we have stubs from $z = -115$ cm to $z = 115$ cm in the barrel. We store the z position as a signed integer over this range with 0 corresponding to $z = 0$ and the minimum and maximim correspnding to $z = \pm 115$ cm. Using 12 bits for the PS modules we have a z granularity of 0.56 mm. This is one third of the z pixel length in the PS modules. For the 2S modules we use 8 bits for the z position which gives a granularity of 9 mm. This is about 1/5 of the 2S module strip length.

For the radial position we use a relative radius with respect to the average, or typical, radius of a layer. We take these radii to be 23.0, 35.5, 50.5, 68.4, 88.4, and 107.8 cm respectively for layers 1 through 6. We use 7 bits for the barrel PS modules, and 8 bits for the barrel 2S modules, to encode the radial position over a range of ± 3 cm with respect to the average radial position. This gives us a radial position accuracy in the PS modules of 0.47 mm, and 0.23 mm for the 2S modules. *Should probably trade ϕ position resolution to*

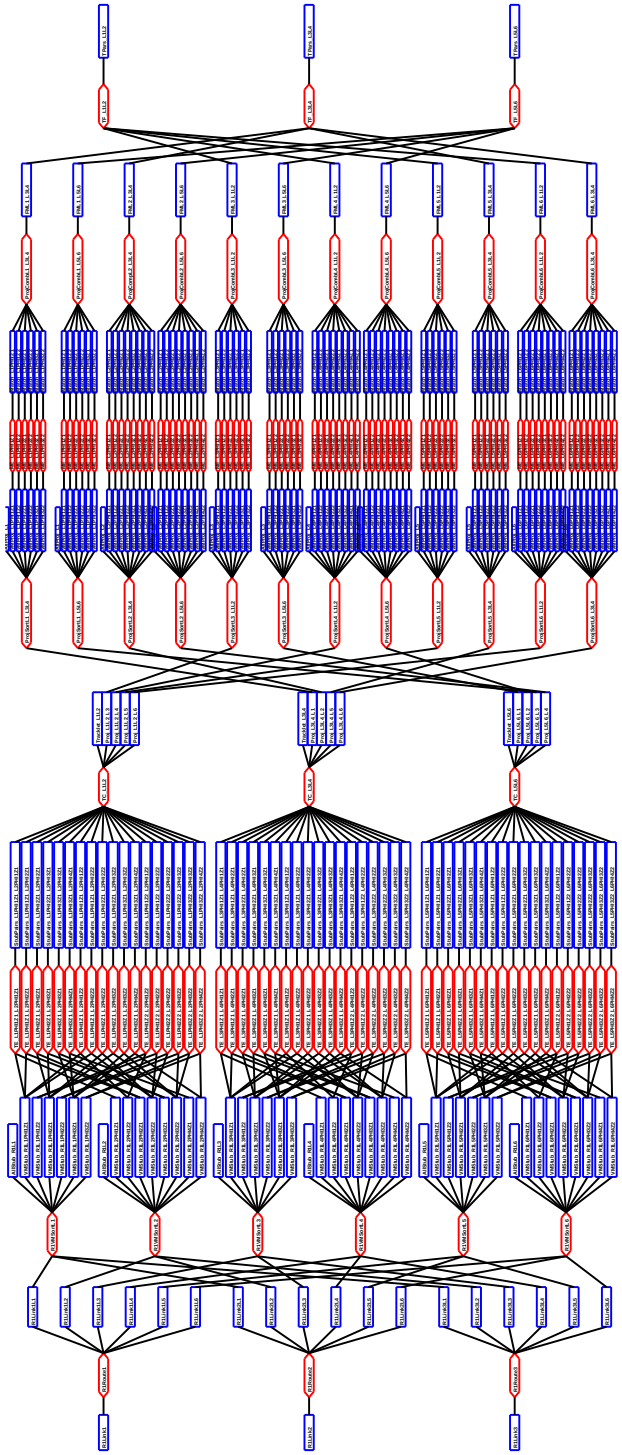


Figure 45: The different modules involved in the implementation of DCT region.

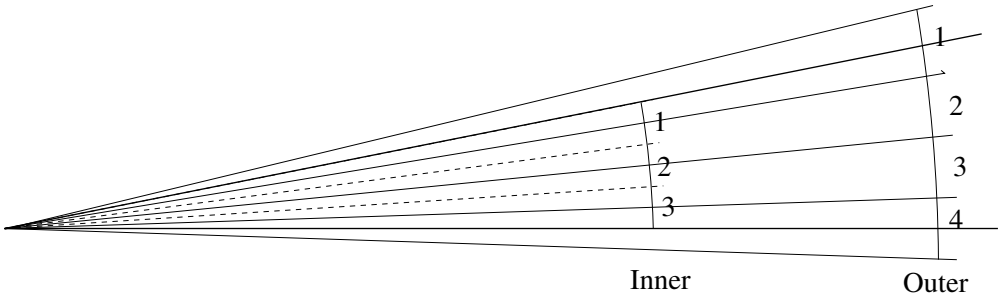


Figure 46: The division of the sector into three virtual modules.

improve the radial accuracy. However, this seems to be 'good enough'. We note here that the stub data format does not contain information about what layer the stub is in. This is not needed since stubs are sorted into different memories/fifos depending on what layer they are in.

The stubs in the disks are treated in a similar way. The format for the ϕ coordinates is exactly the same. While the roles of r and z are swapped, i.e., we measure the z position with respect to an average z position of the disk, but over a range of ± 4 cm. With 7 bits in the PS modules and 8 bits for the 2S modules we have an accuracy of 0.62 mm and 0.31 mm respectively in the PS and 2S modules. For the radial position we measure this in the range from 22 to 110 cm with 12 bits for the PS modules and 8 bits for the 2S modules, giving an accuracy of 0.21 and 3.4 mm respectively for the two types of modules. *This parameterization might not be the most convenient when we combine information from barrel and endcap and will likely be reconsidered. It is also likely that we need to have better precission for the 2S modules as the strips are not projecting to the IP at the edges of the module and to correct for this we need better radial precission. We also likely need to know the strip position on the module. Hence we might end up in the awkward situation that we will need more bits for the 2S modules than the PS modules to represent the stubs.*

7.4 Tracklet finding

The stub data received from the FED, which consists of a physical position and a local pixel index within the module, is routed to the corresponding sector board to find tracklets. The incoming data flow can be seen in Fig. 47. The stub data is stored in a memory of fixed-sized words such that every crossing (even if there are no stubs) corresponds to the address in the memory. The number of stubs expected per crossing and how much buffering is required to run the entire algorithm will determine the size of the memories and how many memory resources will be required in the FPGA.

In order to form the tracklets from pairs of stubs, the outer stub must fall inside a window determined by the curvature of the lowest energy tracks and the separation between the inner and outer modules. The locations of the window edges are pre-calculated and stored in lookup tables. These locations are compared to the outer stub data. Figure 48

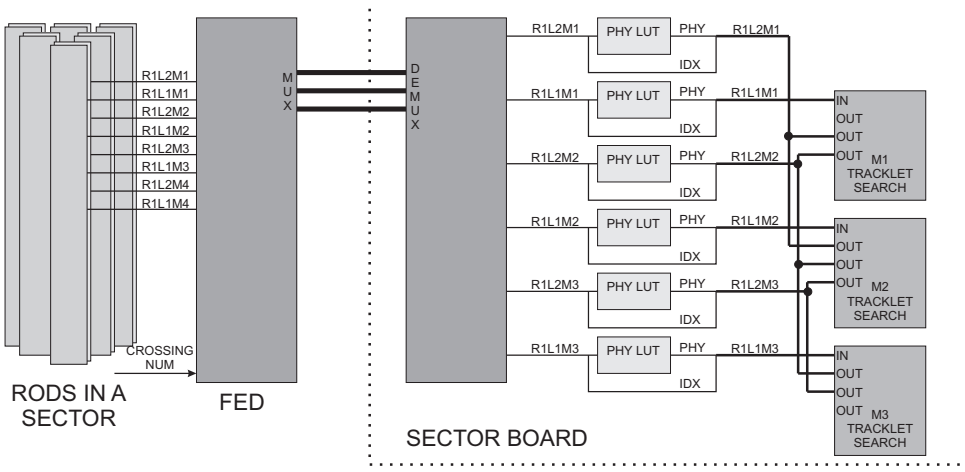


Figure 47: Data flow from RODs to sector processors.

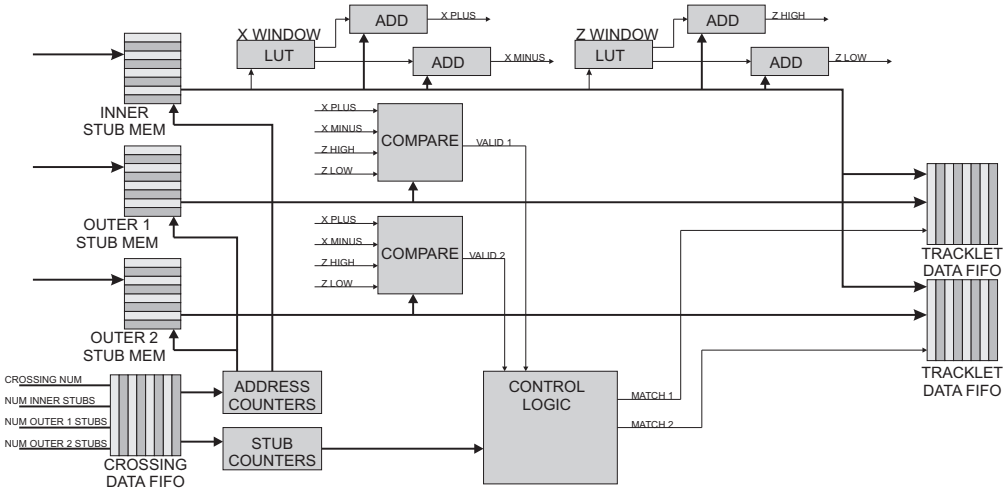


Figure 48: Tracklet finding block diagram. Pre-defined windows are used to compare pairs of stubs. The windows correspond to the lowest-momentum stubs. If a matching stub is found within the window, the resulting tracklet is stored in a FIFO.

shows the block diagram for the tracklet finding. If a stub is found within the window, the corresponding data is routed to the next step where the initial estimate of the track parameters are calculated. These parameters are used for projections into other layers in the hit-finding stage.

7.5 Tracklet projection

A FIFO contains the stub data for the tracklet while another one keeps count of how many tracklets were found in the previous step. This way we are able to pipeline the calculation of the track parameters as new data is constantly coming into the FIFOs. At this point, we implement the integer calculation outlined in Sec. 3.3. The parameter calculation requires a set of constants and lookup tables that are specific to the module. These can be loaded to the FPGA during the initial setup and can be optimized to minimize the use of resources. In Fig. 49 we can see the pipelined calculation for three simulated tracklets in a crossing and we can estimate the number of clock cycles required for the calculation.

After we have determined the track parameters $\rho^{-1}, \phi_0, t, z_0$, the track helix must be projected to other layers to look for matching hits. The projection calculation gives a value for the track ϕ and z positions at the average radii of the other tracker layers. This is only a rough estimate, and is used to determine where the calculated parameters should be sent. Since different ϕ sectors of the detector are handled by different boards, there will be a link that connects these boards in order to share data at the expected rate.

7.6 Hit matching of projected tracklets

Once the data is transferred to the appropriate destination, we can recalculate the ϕ and z positions using a more precise value for the radial distance of the module. With the recalculated position, we look for hits that are close enough to fall within a window around this point. In order to have the hits available for the matching stage, we construct a separate module memory with the same stub data as in the tracklet search stage. This is done in order to avoid collision while retrieving the data from the memory. The results from the matching can be sent back to the original module where the tracklet was found or to a separate area for processing the final track fit.

7.7 Track fit

The track fit has not yet been implemented on the FPGA, but it is currently being worked on.

8 Summary

This document has detailed the tracklet based algorithm and how it can be implemented on FPGAs.

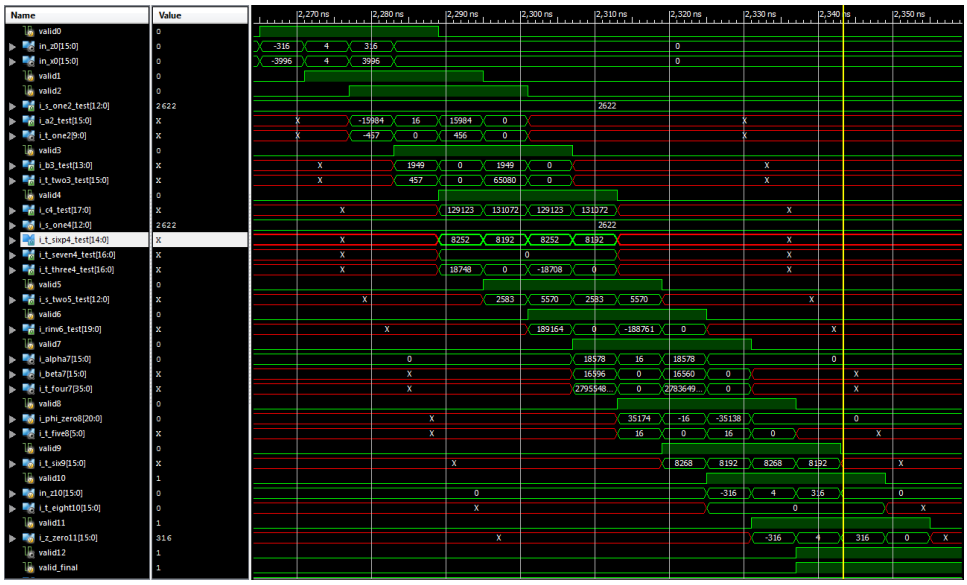


Figure 49: HDL simulation results for the initial tracklet track parameter estimation pipeline. We can see the pipelined calculation for three simulated tracklets in a crossing and we can estimate the number of clock cycles required for the calculation. At this time no optimization of the HW algorithm has been attempted.

A Virtual modules

This appendix collects some numbers for the virtual modules. With 28 sectors, we have $2\pi/28 \approx 0.2243$ radians per sector. Within each sector we have 3 virtual modules where each virtual module covers $2\pi/84 \approx 0.07480$ radians. Within the virtual module we us 3 bits for the ϕ positions so we then have a ϕ granularity of 0.0093480. Using

$$\rho^{-1} = 2\sin\Delta\phi/\Delta$$

Using the resolution of the ϕ position in the virtual modules and the separation of about 12 cm between the two innermost layers we have a resolution in ρ^{-1} of about 0.0016 cm^{-1}.

B Geometry and track parameters

The start of the algorithm is to combine two hits, from stubs, to form a tracklet using the origin as a constraint. The stubs provide two space points. This geometry is illustrated in Fig. 50. The trajectory is assumed to be a circle going through the IP. (We can still fit for an impact parameter in the track fit.)

$$\begin{aligned}
a &= \frac{r_1}{2\cos\Delta\phi} \\
b &= \frac{r_2}{2} - a = \frac{r_2}{2} - \frac{r_1}{2\cos\Delta\phi} \\
l &= \frac{b}{\tan\Delta\phi} = \frac{1}{2\sin\Delta\phi}(r_2\cos\Delta\phi - r_1) \\
\rho^2 &= l^2 + r_2^2/4 = \frac{1}{4\sin^2\Delta\phi}(r_2^2 + r_1^2 - 2r_1r_2\cos\Delta\phi) \\
\rho &= \frac{1}{2\sin\Delta\phi}\sqrt{r_2^2 + r_1^2 - 2r_1r_2\cos\Delta\phi} \\
\rho^{-1} &= \frac{2\sin\Delta\phi}{s}
\end{aligned}$$

where we have used that $s = \sqrt{r_2^2 + r_1^2 - 2r_1r_2\cos\Delta\phi}$ is just the distance between the two hits at r_1 and r_2.

From Fig. 51 we can derive what the ϕ position of the trajectory is at a given radius. The half angle $\theta/2$ is given by

$$\sin\frac{\theta}{2} = \frac{r}{2\rho}$$

From the figure we see that

$$\phi_0 = \phi + \alpha = \phi + \arcsin\frac{r}{2\rho} \tag{1}$$

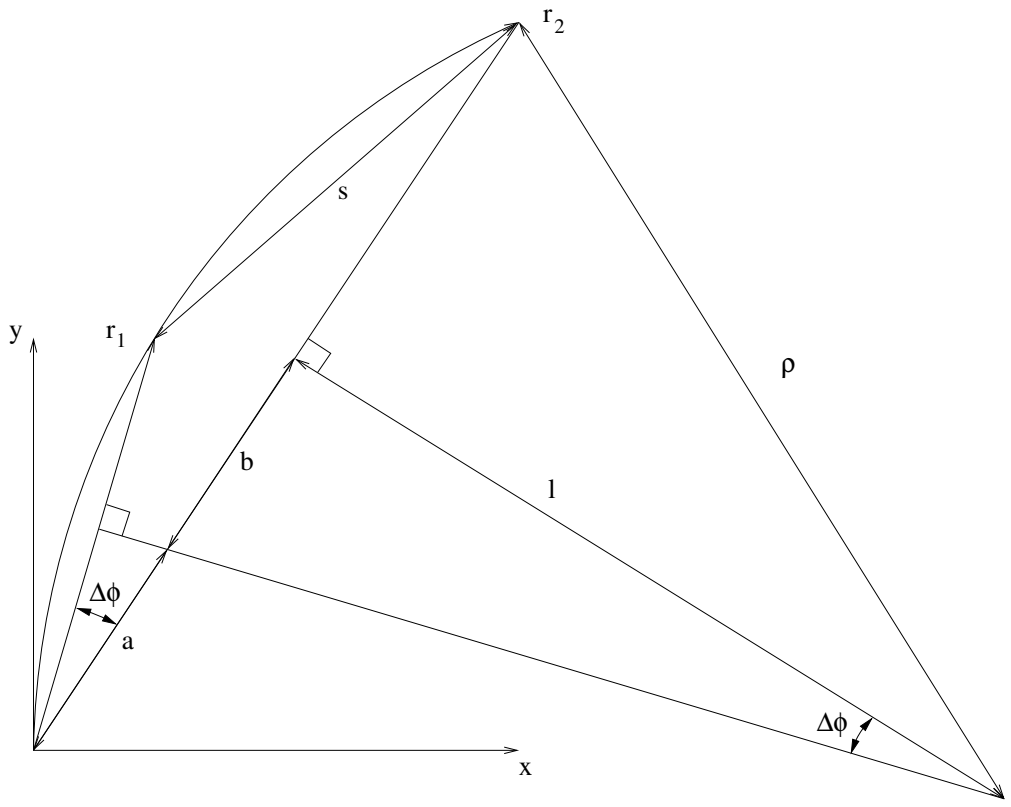


Figure 50: The geometry of a track trajectory in a uniform magnetic field given two hits at radii r_1 and r_2 with an angle between them in the x-y plane of $\Delta\phi$.

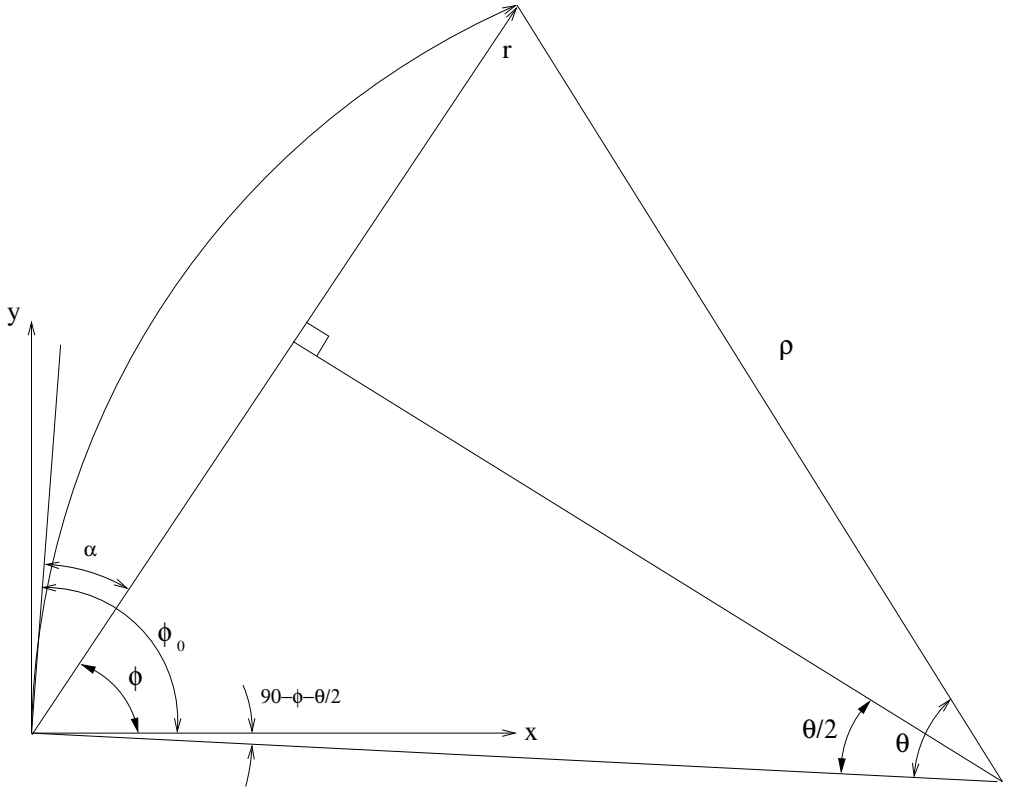


Figure 51: This figure shows the geometry for a trajectory in a uniform magnetic field and in particular relates the ϕ position of the trajectory at a given radius, r.

or

$$\phi = \phi_0 - \arcsin \frac{r}{2\rho} \tag{2}$$

This defines the helix track parameters in the $r-\phi$ plane. For the $r-z$ projection we use two additional parameters. The first is z_0 which is the z coordinate of the track at the point where the track intercepts the z-axis. (Remember that we are working here with a trajectory with no impact parameter, *i.e.*, $d_0 = 0$. But in general z_0 is the tracks z coordinate at the point of closest approach to the z-axis.) The other parameter is $t = tan\lambda$, where λ is the 'dip angle' which is defined by $\lambda = \pi/2 - \theta$, such that $\lambda = 0$ for a track with momentum direction perpendicular to the beam axis. With these parameters we can write the z position of the track as

$$z = z_0 + t\rho\psi$$

where ψ is the turning angle of the track with respect to the point at the origin. *Should have a figure to define this angle.* For a point on the trajectory at a radius r the turning angle is given by

$$\psi = 2 \arcsin \frac{r}{2\rho}$$

This allows us to write

$$z = z_0 + 2t\rho \arcsin \frac{r}{2\rho}$$

In the above we have assumed that the track has no transverse impact parameter. The transverse impact parameter d_0 is the distance the trajector has to the z-axis at the point of closest approach to the z-axis. The impact parameter is signed and the convention is that d_0 has the same sign as the curvature, ρ, *if* the z-axis is outside the circle in the $x - y$ plane formed by the track helix projection. This means that the ϕ position of a track, for small d_0 values, is given by

$$\phi_0 - \arcsin \frac{r}{2\rho} - \frac{d_0}{r}\mathrm{sgn}(\rho).$$

B.1 Curvature and sectors

The relation between P_t [GeV/c], B-field [T] and radius of curvature ρ [m] is given by

$$P_t = 0.3B\rho.$$

At CMS with a 3.8 T magnetic field a particle with a transverse momentum of 2 GeV has a radius of curvature $\rho = 1.754$ m.

The ϕ position for a track with zero impact parameter is given by

$$\phi = \phi_0 + \arcsin(r/2\rho).$$

Calculating the difference in ϕ between two radii we have

$$\Delta\phi = \arcsin(r_2/2\rho) - \arcsin(r_1/2\rho).$$

Taking $\rho = 1.754$ m, $r_1 = 0.32$ m, and $r_2 = 1.02$ m we find $\Delta\phi = 11.7°$. We can also turn this around and ask what P_t gives a 15° $\Delta\phi$ and find $P_t = 1.57$ GeV/c.

So with a 3.8 T magnetic field and sensors spaced between 0.32 m and 1.02 m we find that a 15° sector would allow you to find tracks down to a P_t of 1.57 GeV/c.

C Track parameter representation

This section describes how the track parameters are represented.

For the t parameter we use 14 bits for storing this parameter. This is a signed quantity and covering out to $\eta = 2.5$, corresponding to $t = 6.05$ we have a granularity in t of $\delta t = 7 \times 10^{-4}$.

D Track derivatives

This section describes how the 'track derivatives' are calculated. With a track derivative we mean by how much a measured point, e.g. a ϕ and z position, changes when any of the track parameters are changes.

D.1 Barrel hits

To apply the formalism developed in the previous section we need to evaluate the derivatives that forms the matrices D and M. With $f_i(\rho^{-1}, \phi_0) = r_i\phi_i = r_i\phi_0 - r_i \arcsin(r_i\rho^{-1}/2)$ it is easy to evaluate the derivative with respect to ϕ_0 and ρ^{-1}

$$\begin{aligned}\frac{\partial f_i}{\partial \rho^{-1}} &= -\frac{r_i^2}{2\sqrt{1-\frac{r_i^2\rho^{-2}}{4}}},\\ \frac{\partial f_i}{\partial \phi_0} &= r_i.\end{aligned}$$

We note here that these derivatives depend only weakly on the actual (seed) track parameters. This is what allows us to pre-calculate these derivatives. Note that we work here with $r \times \phi$ since this distance has essentially the same uncertainty in all layers.

Next we calculate the derivatives for the z position of the hit. This calculation follow closely what was done in the previous section except that we now have $f_i(t, z_0) = z_i = z_0 + 2t\rho \arcsin(\frac{r_i}{2\rho})$ which allows us to evaluate

$$\begin{aligned}\frac{\partial f_i}{\partial t} &= 2\rho \arcsin(\frac{r_i}{2\rho}),\\ \frac{\partial f_i}{\partial z_0} &= 1.\end{aligned}$$

Again these derivatives are to first order independent of the track parameters which allows precomputing these derivatives.

D.2 Disk hits

Some formulas related to the geometry of hits on disks are collected here. (Other content to this will be added later.)

As a reminder, as a function of r the ϕ and z positions of the trajectory are given by

$$\begin{aligned}\phi_{\text{proj}} &= \phi_0 - \arcsin\frac{r r_{\text{inv}}}{2}\\ z_{\text{proj}} &= z_0 + \frac{2t}{r_{\text{inv}}}\arcsin\frac{r r_{\text{inv}}}{2}\end{aligned}$$

If you have a disk placed at $z = z_{\text{disk}}$ we can solve

$$z_{\text{disk}} = z_0 + \frac{2t}{r_{\text{inv}}}\arcsin\frac{r r_{\text{inv}}}{2}$$

for r and obtain

$$r = \frac{2}{r_{\text{inv}}}\sin\frac{r_{\text{inv}}}{2t}(z_{\text{disk}} - z_0)$$

using this we can obtain the ϕ position of the track at the disk

$$\phi = \phi_0 - \frac{r_{\rm inv}}{2t}(z_{\rm disk} - z_0).$$

Next we can calculate the track derivatives

$$\begin{aligned}
\frac{\partial r}{\partial r_{\rm inv}} &= -\frac{2}{r_{\rm inv}^2}\sin\frac{r_{\rm inv}}{2t}(z_{\rm disk}-z_0) + \frac{z_{\rm disk}-z_0}{r_{\rm inv}t}\cos\frac{r_{\rm inv}}{2t}(z_{\rm disk}-z_0)\\
\frac{\partial r}{\partial t} &= -\frac{z_{\rm disk}-z_0}{t^2}\cos\frac{r_{\rm inv}}{2t}(z_{\rm disk}-z_0)\\
\frac{\partial r}{\partial z_0} &= -\frac{1}{t}\cos\frac{r_{\rm inv}}{2t}(z_{\rm disk}-z_0)\\
\frac{\partial r}{\partial \phi_0} &= 0\\
\frac{\partial \phi}{\partial \phi_0} &= 1\\
\frac{\partial \phi}{\partial r_{\rm inv}} &= -\frac{z_{\rm disk}-z_0}{2t}\\
\frac{\partial \phi}{\partial t} &= \frac{r_{\rm inv}}{2t^2}(z_{\rm disk}-z_0)\\
\frac{\partial \phi}{\partial z_0} &= \frac{r_{\rm inv}}{2t}
\end{aligned}$$

D.3 Data formatting

The data that arrives at the sector board needs to be formatted and sorted in order to allow for an efficient execution of the tracking algorithm.

The stubs as received from the FEDs will only contain physics address such as row and column numbers for a given module. The first step in receiving this data is to compute the physical coordinates of the hit, i.e., the r, ϕ, and z coordinates of the stubs. This information is used by the tracking algorithm described below. Also some sorting of the data will be required. This will be discussed in more detailed later.

E Possible algorithm improvements

We are collecting here ideas for how to improve the algorithm or the implementation.

- Instead of using lookup table for the stub consistency when we form the tracklet candidates we use combinatorial logic. This can be done either for the ϕ projection or the z projection, or for both. This will save a significan number of memory blocks at the expense of using more logic. Could it help routing?

- Currently we are filling the VM stub memories with all stubs that are in the VM. However, for the z projection we have areas in the VM that will never form valid tracklets. We can reduce the number of stub pairs that has be be tested by only writing to the VM the stubs that can actually be used. This is a simple cut on the z position that should not cost much in terms of resources to implement. But it makes the design more complicated because we will have customized the different VM memories with specific cuts on the z position of the stub.

- Currently we use one pair of memories for each tracklet engine. However, we could use only on memory in one of the two layers and then combine the stubs in this VM with all the other VM modules in the other layers. This would save a significant number of memories for some loss of efficiency.

- We can do something similar for the track matching where we loop over the stubs in a VM module and compare it to all the stubs that project to this layers. Again this will reduce the number of VM stubs memories used for the matching by a factor of 2 (or more if we do seeding in more than 3 layers) while we will loose some efficiency.

F FED functionality

This section describes the functionality we need to have in the FED to implement the tracklet based track finding. We will not be specific here about some number of input and output links, but just explain the functionality and the data formats.

The FED receives data from a number of input links, connected to 2S or PS modules. The data format of the input data is described elsewhere [**?**]. We just remember here that the data is formatted such that it comes in packages of 8 bunch crossings, there are two such packages for each module, corresdonding to each half of the module controlled by one concentrator chip. It is further assumed that the FEDs are connected to the frontend modules such that data from the modules needs to be sent to two, or three, sector processors. If we arrange the data such that it needs to be sent to two sector processors then we have the FEDs organized such that it sends about equal amounts of data to the two sector boards and the FED reads out a region that is centered between two sector boards. In the case of sending data to three sector processing boards the FED readout region is centered on one sector and most of the data will be sent to the central sector processor, with a smaller amount of data to the neighboring sector proceessors.

The FED board needs to do the following tasks:

1. Sort out stubs by the BX, we assume a factor of 4 time multiplexing so every 4th BX will go to the same destination.

2. From the raw stubs we need to calculate the global stub coordinate that are used in the track finding algorithm. (The reason we need to do this calculation in the FED

is so that we can use this information to sort the stubs into the correct output links.) The calculation of the global stub coordinates are explained in Sec. 6.1

3. Based on the ϕ position for the stubs they are routed to the right sector board.

4. Data are sent to the sector processing boards using the format described in Sect. 7.1.2. Note that there will be a maximum number of stubs that can be transfered over 4 BX to a given sector board. This limit has to be enforced by the FED.

5. The FED also needs to buffer the stubs until the L1 decision is made so that the full set of stubs sent from the frontends can be recorded with the event. Note that from the hits in the detector it is not possible to predict what stubs will actually be written out as the averaging is done over 8 BXs.

G Back-of-the-envelope resource estimates (DSPs)

It is always useful to be able to do a back-of-the-envelope estimate of the resources required. This gives you an intuition for the problem and allow you to understand the magnitude of the resources. Of course, if the back-of-the-envelope estimate looks OK, it does not mean necessarily that an implementation will work.

To understand the computational resources required for the tracklet based approach we will consider the combinatorics in one sector, 1/28 of the detector. From the simulations we know that we have about 5 tracks on average in a sector and we have about 350 stubs. (This includes barrel+endcap.) We form tracklets that correspond to $p_{\mathrm{T}} > 2$ Gev and $|z_0| < 15$ cm. From simulation studies we find that we have on average about 50 tracklets formed in L1+L2 and a total of about 100 tracklets in the full sector. For these tracklets we need to calculate the track parameters (requires 20 DSP operations) and project the tracklet to 4 other layers (each projection requires 10 DSP operations). This means that we have a total of 60 DSP operations per tracklets, or 6,000 DSP operations in total. The vast number of these tracklets are fakes - i.e. a combination of two uncorrelated stubs. So when projected to the other layers only a small number of tracklets actually will match to a stub. Let's assume that 25% of the tracklets match to a stub, then we need to calculate the residuals. This requires 4 DSP operations, so for each tracklet we will need 4 operations per tracklet, or 400 operations. The last step requiring DSPs is the track fit. Assuming that 20% of the tracklets has matching hits and needs to be fit we have 20 track fits, each requiring 40 DSP operations for a total of 800 DSP operations. We note that 90% of the computational resources are in the formation of the tracklets and calculating the projections to the other layers.

Next we consider what DSP resources are available in a modern FPGA. The Vertex 7 generation has about 3,000 DSP slices (Ultrascale FPGAs goes a factor of 2 higher). With a time multiplexing of a factor of 4 we have 100 ns per BX, and assuming a 500 MHz clock speed for the project each DSP can perform 50 operations during 100 ns, or 150,000 operations on all DSPs. If we compare this to the estimated number of operations 7,200=6,000+400+800

we see that we are using about 5% of the DSP resources. Of course the challenge is to design an algorithm such that the computing resources are effectively used. However, this estimate of the computational needs show that we are well within the reach of what could be implemented on an FPGA, i.e. this estimate does not present a show stopper.