

# ZPD Fitter Design and Implementation<sup>1</sup>

4 April 2003

Masahiro Morii

This document describes the final design of the Z Fitter algorithm and its implementation in the ZPD module for the BABAR Level-1 Trigger upgrade.

## Table of Contents

1	Overview.....	1
2	I/O Definition.....	2
2.1	Inputs.....	2
2.2	Outputs.....	3
3	Algorithm Outline.....	3
3.1	Input Conversion.....	4
3.2	R- $\phi$ Fitting.....	4
3.3	Z <sub>0</sub> Fitting.....	4
4	Software Emulation.....	5
5	Implementation.....	5
5.1	Engineering Constraints.....	5
5.2	Data Flow Model.....	5
5.3	Timing and Latency.....	9
5.4	Resources.....	9
Appendix A	Algorithm.....	10
A.1	Inputs.....	10
A.2	R- $\phi$ Fitting.....	10
A.3	Z <sub>0</sub> Fitting.....	11
Appendix B	C++ Implementation.....	14
Appendix C	Look-Up Tables.....	17
Appendix D	Full Block Diagram.....	18

## 1 Overview

The Z Fitter is a part of the ZPD modules being developed for the BABAR Level-1 Trigger upgrade. It will be implemented in the Algorithm Engine (AE) FPGAs. The primary purpose of the Z Fitter is to fit a *seed track* (a list of TSF segments that may comprise a charged track) to a helix so that its  $z_0$  (the  $z$  coordinate of the closest approach to the beam axis) can be determined. The proximity of  $z_0$  to the nominal interaction point will be used to discriminate physics events from beam-related background.

For the overall design of the ZPD modules and other background information, see *ZPD Overview*<sup>2</sup>.

---

<sup>1</sup> <http://www.slac.stanford.edu/BFROOT/www/Detector/Trigger/upgrade/fdr/zpd/Fitter.pdf>

<sup>2</sup> <http://www.slac.stanford.edu/BFROOT/www/Detector/Trigger/upgrade/fdr/zpd/ZpdOverview.pdf>

The ZPD modules look for seed tracks starting from the segments (*seed segments*) in superlayers (SLs) 7 or 10. The TSF sends at most 3 segments per SL in each  $22.5^\circ \phi$  sector. This results in at most 12 seed segments (6 each from SL7 and SL10 respectively) in the  $45^\circ \phi$  octant covered by each ZPD module. Each ZPD module uses 6 Algorithm Engines (AEs), 3 for SL7 and 3 for SL10 seed segments. This allows each AE to be programmed for only SL10 or only SL7. Each AE processes 2 seed segments, one from each  $\phi$  sector, in sequence. When the Decision Module receives the results of the fit, it knows the seed SL by the AE that found the track, and the  $\phi$  sector by the relative timing to the CLK4.

## 2 I/O Definition

The Z Fitter receives inputs from the Seed Track Finder and sends outputs to the Decision Module. Although a new set of TSF segments arrive every CLK4 (268.9 ns), the Algorithm Engine is expected to process 2 seed tracks in series within this period. The Seed Track Finder will therefore present 2 seed tracks/CLK4 to the Z Fitter. The latency between the input data and the corresponding output will be constant.

### 2.1 Inputs

The inputs to the Z Fitter come from the Seed Track Finder in the same Algorithm Engine. All the signals are therefore internal to an FPGA.

Input signals	Name	# of bits
TSF segments	segphi	$10 \times 11^3$
	segerr	$10 \times 4$
	hitmap	$10 \times 1$
$p_T$ bin	ptbin	6
Dip angle	dipin	6

#### 2.1.1 TSF Segments

A seed track consists of up to 10 TSF segments, one from each superlayer (SL) of the DCH. Each segment is represented by 16 bits:

- 11-bit (signed)  $\phi$  coordinate relative to the seed segment in the fine- $\phi$  unit ( $1/32^{\text{nd}}$  of a drift cell width).
- 4-bit  $\phi$  error. The baseline algorithm does not use this information.
- 1-bit hit flag indicating the presence of a segment.

The  $\phi$  coordinate for the seed segment is by definition 0. The hit flag for the seed segment is also redundant because no seed track can be found without a seed segment. The redundant information is nevertheless transported to the Z Fitter. Since these signals are internal, the cost is minimal.

#### 2.1.2 Track $p_T$ Bin

The Seed Track Finder provides an estimate of the track transverse momentum  $p_T$ , which is used by the Z Fitter as the starting point. The incoming value, ptbin, is a 6-bit integer. Its correspondence to the actual  $p_T$  value depends on the binning of the track search matrix in the Finder, and has not been fixed. Internally, the Z Fitter converts ptbin into an 8-bit variable  $\rho$ , the track curvature, using a LUT.

<sup>3</sup> The TSF segment phi values (segphi) has been changed from signed 10 bit to signed 11 bit.

### 2.1.3 Track Dip Angle

The Seed Track Finder also provides an estimate of the track dip angle  $\tan\lambda$ . The baseline Z Fitter algorithm does not use this information. We assign 6 bits for potential future use.

## 2.2 Outputs

The outputs from the Z Fitter are sent to the Decision Module. The amount of output data is 38 bits per track. These data are transferred over point-to-point connections between the AE and the DM. Since each AE processes 2 tracks per CLK4, a set of 38-bit outputs is produced every CLK8 (134.5 ns). It is therefore natural to time-multiplex the outputs on a bus that is narrower than 38 bits. A 12-bit bus<sup>4</sup> operating at 30 MHz is sufficient for this purpose, with small increase in the overall latency.

Output signals	Name	# of bits
$z_0$	z0	8
$z_0$ error	z0err	4
Curvature	rhoout	8
Dip angle	dipout	8
Hit map	hitmap	10

### 2.2.1 Fitted $Z_0$

The value of  $z_0$  obtained from the fit is expressed in cm using signed 8 bits.

### 2.2.2 Error on the Fitted $Z_0$

The error  $\sigma_z$  of the fitted  $z_0$  is expressed in cm using unsigned 4 bits. Values between 1 and 10 represent the  $\sigma_z$  value, while 15 indicates that the Fitter could not find a solution. This occurs only when the TSF segments in the seed track do not provide sufficient measurement points to constrain a helix.

### 2.2.3 Fitted Curvature

The value of  $\rho$  obtained from the fit is expressed in  $2^{-12}/\text{cm}$  using signed 8 bits.

### 2.2.4 Fitted Dip Angle

The value of  $\tan\lambda$  obtained from the fit is expressed in  $2^{-5}$  using signed 8 bits

### 2.2.5 Hit Map

The 10-bit hit map information is copied from the input and sent to the Decision Module. This information is intended for diagnostic purposes.

## 3 Algorithm Outline

This section describes the outline of the fitting algorithm. A more formal definition is given in Appendix A.

It should be noted that the Z Fitter algorithm is tightly coupled to the actual geometry of the BABAR Drift Chamber. Although most of the knowledge is encoded in the look-up tables (LUTs), there are assumptions—such as the order of the stereo layers—that are hard-coded in the structure of the algorithm.

---

<sup>4</sup> The width if this bus has been changed from 10 bit to 12 bit to simplify the firmware implementation. The extra bandwidth may be used for transporting more information in the future.

### 3.1 Input Conversion

The  $\phi$ -coordinate data are serialized as they enter the Z Fitter, so that the Z Fitter algorithm processes one segment at a time. This reduces the resource requirement while maintaining the bandwidth required for processing 2 seed tracks per CLK4.

The superlayer order in which the segment data are processed depends on the seed superlayer (the superlayer from which the seed segment was taken).

- A10 seed track: 1, 4, 7, 2, 3, 5, 6, 8, 9.
- A7 seed track: 1, 4, 10, 2, 3, 5, 6, 8, 9.

This ordering scheme places the 3 axial layers first, followed by the 6 stereo layers. The second half of the algorithm requires only the stereo layers, and processing them later reduces the time the data must wait between the two stages.

The input  $\phi$  angle of each segment is expressed in the layer-dependent fine- $\phi$  unit. The angle is converted into the unit of  $2^{-11}$  radians inside the Fitter. This is achieved by a simple multiplication with a layer-dependent constant. The value of  $\phi$  after the conversion fits in a 14-bit signed integer.

The input track  $p_T$  is a 6-bit number that corresponds to the  $p_T$  bin number in the Seed Finder. A look-up table converts this number into the unit of  $2^{-12}$ /cm inside the Fitter. For the momentum acceptance of  $|p_T| > 0.2$  GeV/c, the value of  $p$  after the conversion fits in an 8-bit signed integer.

### 3.2 $R$ - $\phi$ Fitting

Two corrections are applied to the  $\phi$  values of the 9 segments:

- The twist between  $z = 0$  and the rear end plate of the DCH (where the readout electronics are located).
- The shift between each superlayer and the seed superlayer due to the track curvature, using the input curvature measured by the Seed Track Finder.

Each pair of the neighboring stereo superlayers are combined in such a way that the stereo effects cancel with each other to the first order. This provides up to 3 virtually axial measurements. Combining these with the 3 measurements from the axial superlayers, we have 6 points in the  $r$ - $\phi$  plane that should line up at  $\phi = 0$ . The residuals are due to the imperfect knowledge of the curvature,  $\rho$ , and of the location of the seed segment,  $\phi$ . Expressing the residuals in terms of  $\Delta\rho$  and  $\Delta\phi$  to the first order, one can calculate the best values of  $\Delta\rho$  and  $\Delta\phi$  analytically.

### 3.3 $Z_0$ Fitting

Only the stereo segment data proceed to this stage. Two more corrections are applied to the  $\phi$  values of the 6 stereo segments:

- The correction to the seed subtraction using  $\Delta\phi$ .
- The first-order correction to the track curvature correction using  $\Delta\rho$ .

The remaining difference from  $\phi = 0$  is converted to the  $z$  coordinate of the segment using the stereo angle. This provides up to 6 points in the  $z$ - $d$  space, where  $d$  is the 2-d path

length from the beam line to the segment along the track helix. A least-square fit of these points to a straight line gives measurements of  $z_0$  and  $\tan\lambda$ .

## 4 Software Emulation

The Z Fitter algorithm has been implemented in C++. The code<sup>5</sup> is intended to be a bit-wise emulation of the final implementation, and has been used for algorithm validation, resource optimization, and performance studies. The most relevant part of the code has been extracted in a single function in Appendix B.

## 5 Implementation

The Z Fitter is implemented in the Algorithm Engine FPGAs, which also contain the Seed Track Finder. The resource usage is significantly smaller than the Finder, whose resource requirement determines the size of the FPGAs. The design calls for 6 Xilinx Vertex-II<sup>6</sup> 4000 FPGAs (XC2V4000) per ZPD module. Each FPGA contains one Seed Track Finder/Z Fitter pair, which processes 2 seed tracks in every CLK4.

### 5.1 Engineering Constraints

The Finder/Fitter algorithms are implemented using three types of logic resources in the Vertex-II:

- Configurable Logic Blocks (CLBs). Each CLB contains 4 slices, each of which consists of 2 16-bit look-up tables (LUTs), 2 D-type flip-flops, plus some glue logic.
- 18-Kbyte SelectRAM Blocks. Each RAM block can be configured as either single- or dual-port memory. The aspect ratio can be chosen between 16K x 1 bit and 512 x 36 bits.
- 18-bit x 18-bit multipliers.

The XC2V4000 contains 5760 CLBs, 120 RAM blocks, and 120 multipliers. The Z Fitter should occupy as small fraction of these resources as possible. This is particularly important for the number of CLBs, which is the limiting factor for the implementation of the Seed Track Finder.

All operations inside the FPGA occur synchronously to a 120 MHz internal clock (CLK120). There are 32 CLK120 ticks in a CLK4, which is the frequency at which TSF segment data arrive to the ZPD module. As we plan to time-multiplex 2 seed tracks in each Algorithm Engine, the input rate to the Z Fitter is 1 seed track per 16 CLK120 ticks. The Z Fitter must pipeline the data in such a way that none of its logic blocks are occupied by a seed track for more than 16 ticks.

### 5.2 Data Flow Model<sup>7</sup>

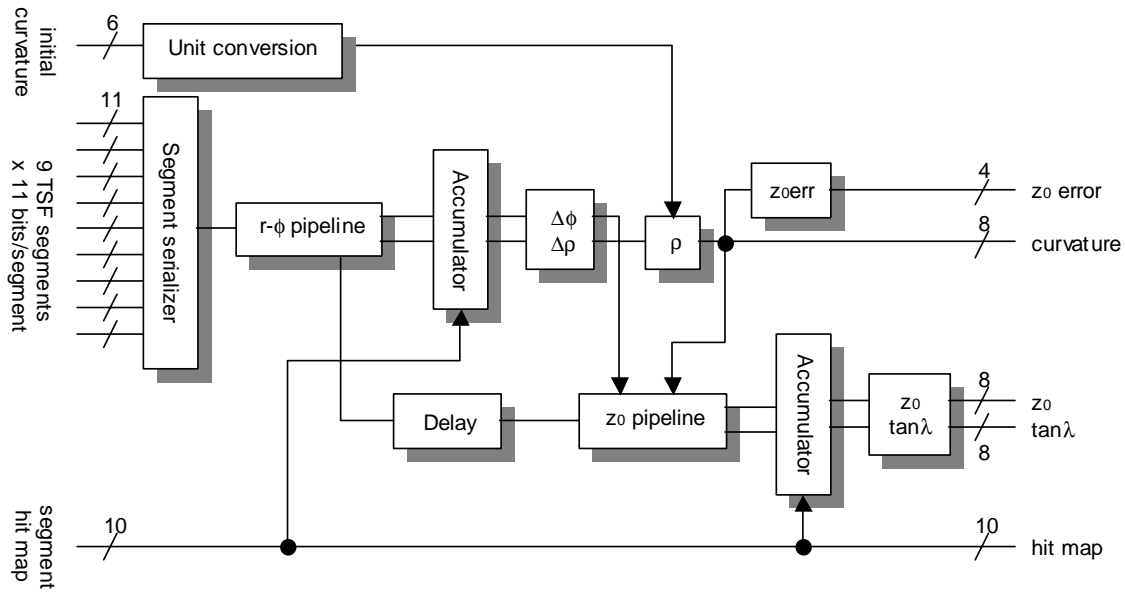
The implementation of the Z Fitter algorithm can best be described by the way the segment  $\phi$  data are processed and transferred through it. The following top-level block diagram shows the general data flow. A complete block diagram is found in Appendix D.

---

<sup>5</sup> L1DczNIFitter.cc in the BABAR software package L1DczTest.

<sup>6</sup> <http://www.xilinx.com/partinfo/ds031.pdf>

<sup>7</sup> The model presented in this section is intended as a guide for the implementation. The actual firmware under development has slightly different data flow.



Each 9-segment set of data representing a seed track flows through the following steps:

1. **Serialization.** A 9-to-1 multiplexer takes 1 out of 9 segments at a time and sends it to the next stage. The segments are separated from each other by 1 CLK120 tick.
2.  **$r$ - $\phi$  fit pipeline.** Each segment goes through a 4-stage pipeline until two quantities,  $r^2\phi$  and  $r^2\phi d\phi/d\rho$ , are calculated.
3.  **$r$ - $\phi$  fit accumulators.** Outputs from the pipeline are summed up in 2 accumulators.
4.  **$\Delta\phi$  and  $\Delta\rho$  calculation.** The sums obtained in the accumulators are used to calculate  $\Delta\phi$  and  $\Delta\rho$ . The latter is then used to calculate the fitted curvature  $\rho$ .
5. **Dual-port memory.** The  $\phi$  data for the 6 stereo segments are transferred from the  $r$ - $\phi$  fit pipeline to the  $z_0$  fit pipeline via a dual-port memory.
6.  **$z_0$  fit pipeline.** Each stereo segment goes through a 4-stage pipeline until two quantities,  $z/\sigma^2$  and  $zd/\sigma^2$ , are calculated<sup>8</sup>.
7.  **$z_0$  fit accumulators.** Outputs from the pipeline are summed up in 2 accumulators.
8.  **$z_0$  and  $\tan\lambda$  calculation.** The sums obtained in the accumulators are used to calculate  $z_0$  and  $\tan\lambda$ .

The operations performed in each step are discussed in the following subsections. A timing chart is found in Section 5.3.

### 5.2.1 Serialization

The input segments are serialized so that the segments flow serially into the  $r$ - $\phi$  pipeline. Each segment is represented by a 11-bit  $\phi$  value. This is effectively a 9:1 multiplexer reducing 99 bits into 11 bits.

### 5.2.2 $r$ - $\phi$ Fit Pipeline

Each segment goes through the following 4 stages.

<sup>8</sup> See Appendix A.3 for the definition of the calculated quantities.

### 5.2.2.1 Unit Conversion

The unit for the  $\phi$  data is converted from the fine- $\phi$  unit to  $2^{-11}$  radians. This is a multiplication by a layer-dependent constant. The output is a 14-bit signed integer.

### 5.2.2.2 Twist Correction

The  $\phi$  data are corrected for the stereo twist between the rear end plate and  $z = 0$ . This is an addition with a layer-dependent constant.

### 5.2.2.3 Curvature Correction

The  $\phi$  data are corrected for the track curvature. This is an addition with a constant from a LUT.

### 5.2.2.4 Calculation of $r^2\phi$ and $r^2\phi d\phi/d\rho$

Two quantities  $r^2\phi$  and  $r^2\phi d\phi/d\rho$  are calculated simultaneously from the  $\phi$  data. Both are done with a multiplication with constants ( $r^2$  and  $r^2 d\phi/d\rho$ ) from LUTs.

### 5.2.3 $r$ - $\phi$ Fit Accumulators

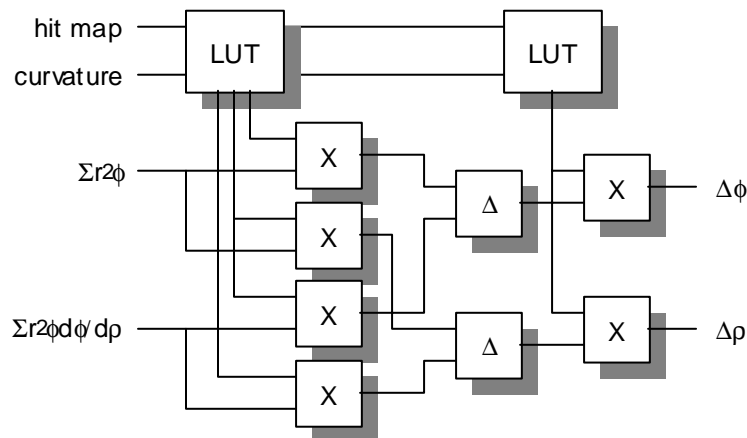
The final products from the pipelines are accumulated to calculate  $\Sigma r^2\phi$  and  $\Sigma r^2\phi d\phi/d\rho$ . For each quantity, the accumulator must add values from the pipelines arriving at every CLK180 tick.

### 5.2.4 Calculation of $\Delta\phi$ and $\Delta\rho$

Two quantities  $\Delta\phi$  and  $\Delta\rho$  are calculated in parallel from the outputs of the accumulators. Each of them require

1. Multiplying the accumulator outputs with constants from LUTs simultaneously using 2 multipliers.
2. Taking the difference between the two values obtained in 1.
3. Multiplying the result by a constant from a LUT.

The following diagram shows an example implementation.



From  $\Delta\rho$ , the fitted track curvature is obtained by an addition with the initial curvature.

### 5.2.5 Delay Line

The  $\phi$  values for the 6 stereo segments must be transported from the intermediate stage (after the curvature correction) of the  $r$ - $\phi$  fit pipeline to the input stage of the  $z_0$  fit pipeline. A register chain is needed for this purpose.

### 5.2.6 $z_0$ Fit Pipeline

Each segment goes through the following 4 stages.

#### 5.2.6.1 $\Delta\phi$ Correction

The  $\phi$  data are corrected for  $\Delta\phi$ . This is an addition.

#### 5.2.6.2 $\Delta\rho$ Correction

The  $\phi$  data are corrected for  $\Delta\rho$  to the first order. This is done by

1. Multiplying  $\Delta\rho$  with  $d\phi/d\rho$  from a LUT.
2. Adding the result to  $\phi$ .

#### 5.2.6.3 Calculation of $z$

The  $\phi$  data are converted to  $z$ . This is a multiplication with a layer-dependent constant.

#### 5.2.6.4 Calculation of $z/\sigma^2$ and $zd/\sigma^2$

Two quantities  $z/\sigma^2$  and  $zd/\sigma^2$  are calculated simultaneously from  $z$ . Both are done with a multiplication with constants ( $1/\sigma^2$  and  $d/\sigma^2$ ) from LUTs.

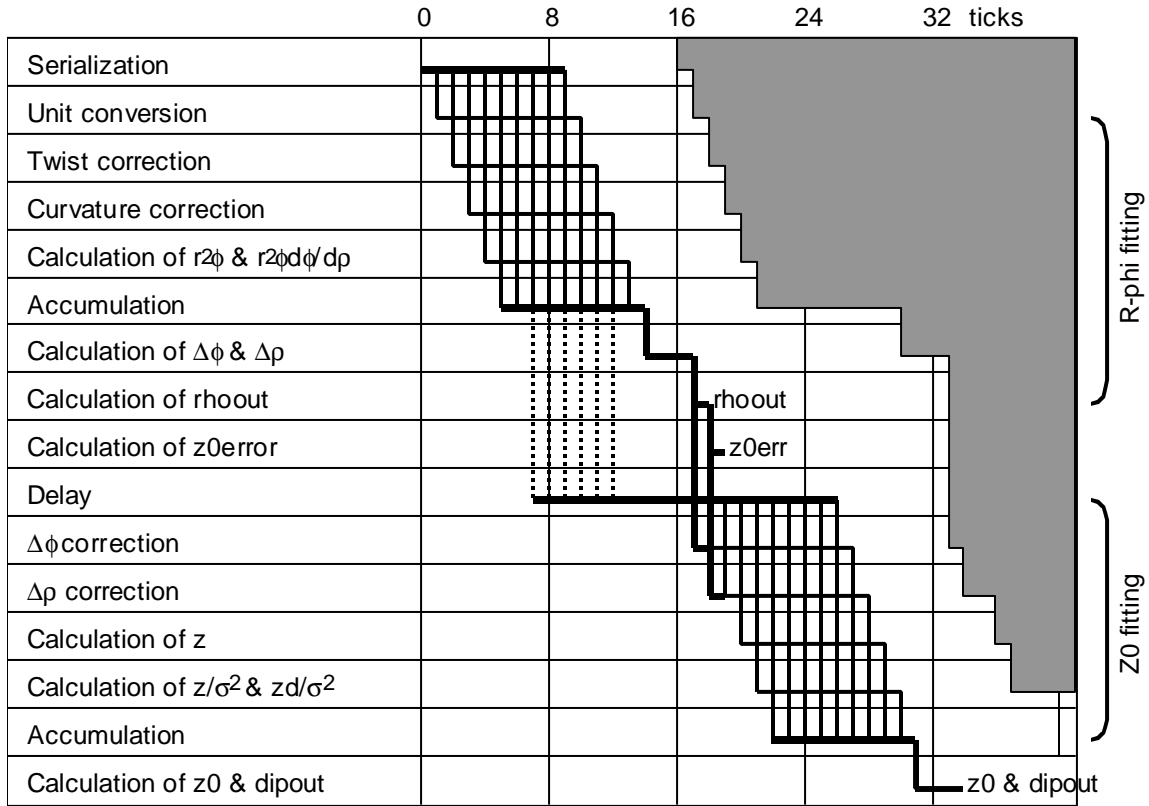
### 5.2.7 $z_0$ Fit Accumulators

The final products from the pipelines are accumulated to calculate  $\Sigma z/\sigma^2$  and  $\Sigma zd/\sigma^2$ .

### 5.2.8 Calculation of $z_0$ and $\tan\lambda$

Two quantities  $z_0$  and  $\tan\lambda$  are calculated in parallel from the outputs of the accumulators. This step is very similar to 5.2.4 above.

### 5.3 Timing and Latency



The above chart shows the timing of the data flow through the Z Fitter algorithm according to this data flow model. The horizontal axis is the number of internal clock (120 MHz) ticks from the arrival of the input data. The vertical lines show data transfers from one block to another. The process time of each block was calculated assuming that each addition or multiplication takes 1 tick, and the result is ‘clocked’ into a register for synchronization.

The two sets of parallel staircase-like lines represent segment  $\phi$  information being processed in sequence. The delay line in the middle carries the stereo segment information from the  $r-\phi$  fitting stage to the  $z_0$  fitting stage. The length of the delay corresponds to 10 clock ticks.

The shaded area indicates the arrival of the data for the next seed track, which is later by 16 ticks. The closest approach between the two successive seed track is 6 ticks. The Z Fitter can therefore accept, if necessary, seed tracks at every 10 ticks.

The  $z_0$  and  $\tan\lambda$  outputs become available 34 ticks after the arrival of the input data. With 120 MHz internal clock, this amounts to 1.06 CLK4.

### 5.4 Resources

The Z Fitter algorithm requires rather small hardware resources. The current design is smaller than that was presented at the CDR, which was already negligibly small compared with the Seed Finder.

## Appendix A Algorithm

### A.1 Inputs

The primary inputs to the Z Fitter algorithm are the  $\phi$  coordinates of the TSF segments that comprise a track candidate. Let us call them  $\{\phi_i\}$ , where  $i = 1 \dots 10$  is the superlayer (SL) number. One of the segments, either  $i = 7$  or  $10$ , is called the seed segment. Since each Seed Track Finder uses either only SL7 or only SL10 as the seed SL, the Z Fitter knows the seed SL a priori. In the following discussion, I assume that the seed segment is in SL10. The case with SL7 seed is very similar.

The seed track does not necessarily have all 10 segments. A 10-bit hit map is needed for the Z Fitter to know which  $\phi_i$  exist. Absence of the seed segment implies that the Seed Track Finder did not find any seed track for this event.

At this point, it is convenient to redefine the origin of the  $\phi$  coordinates so that  $\phi_{10} = 0$ . All the other segments are converted so that  $\phi_i \rightarrow \phi_i - \phi_{10}$ . This serves two practical purposes: first, the range of  $\phi_i$  becomes limited so that fewer bits are needed to represent the values; second, one can drop  $\phi_{10}$  and only deal with the remaining 9 segments.

The last input the Z Fitter requires is an initial estimate of the track curvature. Let us call it  $\rho_{in}$ . The Seed Track Finder provides this information in 4 to 5-bit resolution.

### A.2 R- $\phi$ Fitting

In this stage,  $\{\phi_i\}$  are fitted to a circle in the  $r$ - $\phi$  plane in order to derive a better estimates of the  $\phi$  coordinate of the track at the seed SL, and  $\rho$ . Since we have subtracted  $\phi_{10}$  from all  $\phi$  coordinates, the former should be close to 0. Let us call it  $\Delta\phi$ . We also call the difference between the latter and its initial estimate as  $\Delta\rho = \rho - \rho_{in}$ .

In order to make the fit in the  $r$ - $\phi$  plane, the effect of the stereo angles must be eliminated. This can be approximately achieved by the following process. For each stereo segment  $\phi_i$ , we first apply correction for the twist angle between the DCH end plate and the beam interaction point ( $z = 0$ ). The remaining stereo effect is proportional to the  $z$  coordinate of the hit, which is in turn proportional to the radius of the SL if the track originated from  $z = 0$ . If we take a pair of neighboring stereo SLs,  $i$  and  $j$ , their stereo angles are similar in size and opposite in direction. By averaging  $\phi_i$  and  $\phi_j$  with weights that are inversely proportional to the SL radii  $r_i$  and  $r_j$ , the result

$$\phi_{ij} = \frac{r_j \phi_i + r_i \phi_j}{r_i + r_j}$$

becomes nearly free from the stereo effects. One can consider  $\phi_{ij}$  as a measurement obtained from a virtual axial SL located at the radius

$$r_{ij} = \frac{2r_i r_j}{r_i + r_j}.$$

Applying this method to all the stereo SLs leaves us with 3 real ( $\phi_1, \phi_4, \phi_7$ ) and 3 virtual ( $\phi_{23}, \phi_{56}, \phi_{89}$ ) measurements in the  $r$ - $\phi$  plane.

At this point, we can draw a circle that passes the nominal beam line and the seed segment using the curvature  $\rho_{\text{in}}$  and calculate the distance between this curve and the 6 measurements above. This residual is given by

$$r_i \left( \phi_i - \sin^{-1} \frac{r_i \rho_{\text{in}}}{2} + \sin^{-1} \frac{r_{10} \rho_{\text{in}}}{2} \right).$$

We redefine the terms in the parenthesis as  $\phi_i$ . Assuming that all the measurements have a similar resolution, we want to minimize the sum of the squares

$$\sum (r_i \phi_i)^2$$

where the sum is taken over the valid (existing) measurements.

Assuming that  $\Delta\phi$  and  $\Delta\rho$  are small, one can take the first derivatives of  $\phi_i$  as

$$\frac{\partial \phi_i}{\partial \phi_{10}} = -1 \quad \text{and} \quad \frac{\partial \phi_i}{\partial \rho_{\text{in}}} = \frac{r_{10} \rho_{\text{in}}}{2\sqrt{1 - \left(\frac{r_{10} \rho_{\text{in}}}{2}\right)^2}} - \frac{r_i \rho_{\text{in}}}{2\sqrt{1 - \left(\frac{r_i \rho_{\text{in}}}{2}\right)^2}},$$

and expand  $\phi_i$  in terms of  $\Delta\phi$  and  $\Delta\rho$ . The square sum becomes

$$\sum r_i^2 \left( \phi_i + \frac{\partial \phi_i}{\partial \phi_{10}} \Delta\phi + \frac{\partial \phi_i}{\partial \rho_{\text{in}}} \Delta\rho \right)^2 = \sum r_i^2 \left( \phi_i - \Delta\phi + \frac{\partial \phi_i}{\partial \rho_{\text{in}}} \Delta\rho \right)^2.$$

This is minimized by

$$\Delta\phi = \frac{\sum r_i^2 \phi_i \sum r_i^2 \left(\frac{\partial \phi_i}{\partial \rho_{\text{in}}}\right)^2 - \sum r_i^2 \phi_i \frac{\partial \phi_i}{\partial \rho_{\text{in}}} \sum r_i^2 \frac{\partial \phi_i}{\partial \rho_{\text{in}}}}{\sum r_i^2 \sum r_i^2 \left(\frac{\partial \phi_i}{\partial \rho_{\text{in}}}\right)^2 - \left(\sum r_i^2 \frac{\partial \phi_i}{\partial \rho_{\text{in}}}\right)^2},$$

$$\Delta\rho = \frac{\sum r_i^2 \phi_i \sum r_i^2 \frac{\partial \phi_i}{\partial \rho_{\text{in}}} - \sum r_i^2 \phi_i \frac{\partial \phi_i}{\partial \rho_{\text{in}}} \sum r_i^2}{\sum r_i^2 \sum r_i^2 \left(\frac{\partial \phi_i}{\partial \rho_{\text{in}}}\right)^2 - \left(\sum r_i^2 \frac{\partial \phi_i}{\partial \rho_{\text{in}}}\right)^2}.$$

We have obtained better estimates of the track parameters in the  $r$ - $\phi$  plane. In particular, the fitted curvature is

$$\rho_{\text{fit}} = \rho_{\text{in}} + \Delta\rho.$$

It is useful to note that many terms that appear in the above equations do not depend on  $\{\phi_i\}$ . Although the term  $\partial\phi_i/\partial\rho_{\text{in}}$  depends on  $\rho_{\text{in}}$ , the limited accuracy of  $\rho_{\text{in}}$  allows us to use relatively small look-up tables to hold precomputed values of these terms. The denominator is the same for both  $\Delta\phi$  and  $\Delta\rho$ , and its inverse can be stored in a look-up table to eliminate the needs for divisions. The two sums that do depend on  $\{\phi_i\}$  are both linear in  $\phi_i$ . This allows us to merge the weighted-averaging of the stereo measurements with the accumulation of these sums to simplify the flow of the algorithm.

### A.3 $Z_0$ Fitting

In this stage,  $\{\phi_i\}$  for the stereo SLs are converted into  $z$  measurements and fitted to find the value of  $z_0$ . First, the values of  $\{\phi_i\}$  must be corrected for the  $\Delta\phi$  and  $\Delta\rho$  that have been calculated in the previous stage.

$$\phi_i \rightarrow \phi_i - \Delta\phi + \frac{\partial\phi_i}{\partial\rho} \Delta\rho.$$

The remaining  $\phi_i$  is due to the stereo angle  $\theta_{\text{stereo}}$  times the  $z$  coordinate of the segment location. The actual relationship is

$$r_i \tan \phi_i = z \tan \theta_{\text{stereo}}.$$

(This equation is not exact due to the fact that  $z = 0$  is not at the geometrical center of the DCH. The difference is small and can be neglected.) For small  $\phi_i$ , one can approximate  $\tan\phi_i$  with  $\phi_i$  and obtain

$$z_i = \frac{r_i}{\tan \theta_{\text{stereo}}} \phi_i.$$

The 2-dimensional distance between this segment and the beam axis measured along the track (circle) is given by

$$d_i = \frac{2}{\rho_{\text{fit}}} \sin^{-1} \frac{r_i \rho_{\text{fit}}}{2}.$$

The measurements  $(z_i, d_i)$  should make a straight line that satisfy

$$z = z_0 + d \tan \lambda.$$

One can construct a  $\chi^2$  as

$$\chi^2 = \sum \frac{(z - z_0 - d \tan \lambda)^2}{\sigma_i^2},$$

where the sum is taken over all existing stereo segments. The error  $\sigma_i$  is calculated by

$$\sigma_i = \frac{r_i \times (1 \text{ mm})}{\tan \theta_{\text{stereo}}}$$

assuming that the error in the  $r$ - $\phi$  plane is approximately 1 mm.

The above  $\chi^2$  is minimized by

$$z_0 = \frac{\sum \frac{z_i}{\sigma_i^2} \sum \frac{d_i^2}{\sigma_i^2} - \sum \frac{z_i d_i}{\sigma_i^2} \sum \frac{d_i}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{d_i^2}{\sigma_i^2} - \left( \sum \frac{d_i}{\sigma_i^2} \right)^2},$$

$$\tan \lambda = \frac{\sum \frac{z_i d_i}{\sigma_i^2} \sum \frac{1}{\sigma_i^2} - \sum \frac{z_i}{\sigma_i^2} \sum \frac{d_i}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{d_i^2}{\sigma_i^2} - \left( \sum \frac{d_i}{\sigma_i^2} \right)^2}.$$

As it was in the  $r$ - $\phi$  fitting, many of the terms that appear in the solutions do not depend on  $z_i$  and can be precomputed. The distance  $d_i$  depends weakly on the curvature  $\rho$ , and only 3-bit resolution is required for  $\rho$  to provide enough accuracy. The denominator is common, and its inverse can be calculated and stored in a look-up table.

The uncertainty on  $z_0$  is given by

$$\sigma_z = \frac{\sum \frac{d_i^2}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{d_i^2}{\sigma_i^2} - \left( \sum \frac{d_i}{\sigma_i} \right)^2}.$$

This can also be precomputed and stored in a look-up table.

## Appendix B C++ Implementation

The Z Fitter algorithm has been implemented in a C++ function `L1DczNIFitter::zFitter()` shown below. The code makes extensive use of look-up tables, which are held in an object `table`. Each occurrence of `table->xxx()` in the code represent a look-up of a table named `xxx`. A full list of the look-up tables is found in Appendix C.

```
bool
L1DczNIFitter::zFitter(const L1DczNIFtable* table, int hitmap,
                      const int* segphi, int ptbin,
                      int &z0, int &z0err, int &rhoout, int &dipout) const
//
// Real job is done in this function.
// Everything is done in integer.
// Returns true if the fit succeeds, i.e. there are enough segments
// to constrain a helix track in 3 dimensions.
//
// Inputs:  table      = constant table
//          hitmap     = 9-bit map of which SL has a segment
//          segphi[9]  = segment fine phi for each SL
//          ptbin      = 1/pt bin number from the Finder
//
// Outputs: z0        = z0 in [cm]
//          z0err      = z0 error in [cm]
//          rhoout     = fitted rho0 in [1/2^12/cm]
//          dipout     = fitted tandip in [1/2^5]
//
// Note on the constant table:
//   All references to "table->function()" are simple table look-ups.
//   The tables absorb most of the computation that would otherwise
//   have to be done in this function.
//   See L1DczNIFtable for the implementation.
//
{
//
// Check if hit pattern is good.
// This is just counting the number of segments.
//
if (!table->fitok(hitmap)) {
// set default values for a fit failure
z0 = -128;           // beyond the DCH length
z0err = 15;         // largest error
rhoout = -128;     // -200 MeV/c (valid value)
dipout = -128;     // -4 (beyond the acceptance)
return false;
}
//
// Convert the 1/pt bin number into track curvature.
// Internal to the fitter, I use [2^(-12)/cm] as the unit.
// The value of rho0 is signed 8 bits
//
int rho0 = table->rhoconv(ptbin); // [2^(-12)/cm]
//
// R-phi Fitting Block
//
// Inputs:  segphi[9], hitmap, rho0
// Outputs: phi[9], dPhi, dRho, rhoout
//
int phi[9]; // seed-subtracted phi [2^(-11)rad]
int rho5 = table->rho5(rho0); // abs(rho0) in 5 bits
int hitax = table->hitax(hitmap); // for table lookups (6 bits)
int useax = table->useax(hitmap); // used in accumulation (9 bits)
//
// Accumulation section calculates the next two sums
//
int sumr2phi = 0; // sum(r^2*phi)
int sumr2phidpdr = 0; // sum(r^2*phi*d(phi)/d(rho))
```

```

//
int i;
for ( i = 0; i < 9; i++ ) {
    //
    // Conversion to internal unit.
    // Constant phiconv is in [2^(-13)], so I shift by -13 bits.
    // phi[i] is signed 12 bits in [2^(-11)rad]
    //
    phi[i] = (segphi[i]*table->phiconv(i))>>13;          //[2^(-11)rad]
    //
    // Correction for the twist angle between the end plate and z = 0.
    //
    phi[i] += table->twistzero(i);
    //
    // Correction for the curvature rho0.
    //
    phi[i] += table->curvcorr(i,rhoin);
    //
    // Accumulate only if this segment is "valid".
    //
    if (useax & (1<<i)) {
        //
        // Accumulate r^2*phi and r^2*phi*d(phi)/d(rho).
        // sumr2phi is signed 16 bits
        // sumr2phidpdr is signed 18 bits
        //
        sumr2phi      += table->wr2(i)*phi[i];          //[2^(-6)radcm^2]
        sumr2phidpdr += table->wr2dpdr(i,rho5)*phi[i];  //[2^(-3)rad^2cm^3]
    }
}
//
// Calculate the phi and rho corrections.
// These are a bunch of "look-up and multiply" operations that
// fit nicely to the SelectRAM + multiplier combinations in Xilinx.
//
// All calculation is done in signed 16 bits AFTER bit shifts.
// For example: dPhi1 is signed 28 bits before >>12 is applied.
// The only exception is sumr2phidpdr which is signed 18 bits.
//
// NB: dRho is calculated to 2^(-16)/cm, i.e. 4 extra bits compared
// with the final rhoout.
// This extra resolution is necessary to get the segment phi
// corrections accurate enough for SL10 seed tracks.
// Without this, z0 resolution suffers and the central value
// <z0> shifts relative to SL7 seed.
//
int dPhi1 = (sumr2phi*table->sumr2dpdr2(hitax,rho5))>>12;  //[2^17rad^3cm^6]
int dPhi2 = (sumr2phidpdr*table->sumr2dpdr(hitax,rho5))>>12;  //[2^17rad^3cm^6]
int dPhi  = ((dPhi1-dPhi2)*table->denomrp(hitax,rho5))>>15;  //[2^(-11)rad]
int dRho1 = (sumr2phi*table->sumr2dpdr(hitax,rho5))>>10;    //[2^12rad^2cm^5]
int dRho2 = (sumr2phidpdr*table->sumr2(hitax))>>10;         //[2^12rad^2cm^5]
int dRho  = ((dRho1-dRho2)*table->denomrp(hitax,rho5))>>15;  //[2^(-16)/cm]
//
// Calculate fitted rho0.
// Note that rhoout is 4 bits less accurate than dRho.
// The "+8" makes rounding instead of truncation.
//
rhoout = rhoin + ((dRho+8)>>4);
//
// Sometimes, rhoout goes beyond the physical limits of DCH.
// Abort if this happens, to avoid crashes due to uninitialized
// table values. (Hardware doesn't crash, though.)
//
if (abs(rhoout) > table->maxrho()) {
    // set default values for a fit failure
    z0 = -128;          // beyond the DCH length
    z0err = 15;        // largest error
    rhoout = -128;     // -200 MeV/c (valid value)
    dipout = -128;     // -4 (beyond the acceptance)
    return false;
}

```

```

//
// Z-tan(lambda) Fitting Block
//
// Inputs: phi[6], hitmap, dPhi, dRho
// Outputs: z0, z0err
//
// Following 3 variables are needed for table lookups
//
hitmap >>= 3; // drop the axial part
int rho3 = table->rho3(rhoout); // abs(rhoout) in 3-bit accuracy
rho5 = table->rho5(rhoout); // abs(rhoout) in 5-bit accuracy
//
// Accumulation section calculates the next two sums
//
int sumzs2 = 0; // sum(z/sigma^2)
int sumzds2 = 0; // sum(z*d/sigma^2)
//
for (i = 0; i < 6; i++) {
//
// Correction for dPhi
// phi[i+3] has an offset because the first 3 axial segments have
// to be dropped.
//
int phist = phi[i+3] - dPhi; // [2^(-11)rad]
//
// Correction for dRho
//
phist += (table->dphidrho(i,rho5)*dRho)>>10; // [2^(-11)rad]
//
// Conversion of phi to z
// This should involve tan(phi), but the difference between
// phi and tan(phi) for |phi| < 0.2 is negligible.
// The resolution is 9 bits in [2cm].
//
int z = (table->rstereo(i)*phist)>>9;
//
// Accumulate only if this segment is "valid".
//
if (hitmap & (1<<i)) {
//
// Accumulate z/sigma^2 and z*d/sigma^2.
//
sumzs2 += z*table->sigma2z(i); // [2^(-2)/cm]
sumzds2 += z*table->dsigma2z(i,rho3); // [2^(-2)]
}
}
//
// Calculation of z0 and tandip.
//
int z01 = (sumzs2*table->sumd2s2(hitmap,rho3))>>13; // [2^8/cm]
int z02 = (sumzds2*table->sumds2(hitmap,rho3))>>13; // [2^8/cm]
z0 = (z01-z02)*table->denomzt(hitmap,rho3)>>8; // [cm]
int td1 = (sumzds2*table->sums2(hitmap))>>7; // [4cm^2]
int td2 = (sumzs2*table->sumds2(hitmap,rho3))>>7; // [4cm^2]
dipout = (td1-td2)*table->denomzt(hitmap,rho3)>>9; // [2^(-5)]
//
// z0 error is determined by the hit pattern.
//
z0err = table->z0err(hitmap,rho3);
//
return true;
}

```

## Appendix C Look-Up Tables

The table below summarizes the look-up tables used by the baseline Z Fitter algorithm. The “Resources” column indicates the number of CLBs or RAM block resources needed to implement each table. All memories are assumed to be dual-port. RAM blocks are preferred over CLBs except for very small tables. Some tables are merged into single RAM block if they are accessed together.

The values stored in the LUTs are available online<sup>9</sup>

Name	Dimension	Bits/word	Resources		Note
			CLBs	RAMs	
fitok	512	1		1	Accessed together
hitax	512	6			
rhoconv	64	8	8		
rho5	256	5		1	rho5 used twice
useax	64	9	9		
phiconv	9	16	4		
twistzero	9	10	3		
curvcorr	9 × 256	12		3	
wr2	9	8	2		Accessed together
wr2dpdr	9 × 32	12		1	
sumr2	64	9	5		Accessed together
sumr2dpdr	64 × 32	14		2	
sumr2dpdr2	64 × 32	15		2	
denomrp	64 × 32	15		2	
rho3	256	3		1	Accessed together. rho5 used twice.
rho5	256	5			
dphidrho	6 × 32	12		1	
rstereo	6	8	4		Accessed together
sigma2z	6	3	2		
dsigma2z	6 × 8	9	4		
sums2	64	5	5		Accessed together
sumds2	64 × 8	11		1	
sumd2s2	64 × 8	17			
denomzt	64 × 8	15		1	
z0err	64 × 8	4			
Total			29	14	

<sup>9</sup> <http://huhepl.harvard.edu/~masahiro/trigger/lut/>

## Appendix D Full Block Diagram

The diagram below shows the entire Z Fitter algorithm, except for the error handling in which the outputs are replaced by the “invalid” values described in Section 2.2.

