

ZPD Seed Track Finder Algorithm Description¹

4 April 2003

S. Bailey, N. Sinev

This note describes the implementation of the ZPD Seed Track Finder for the BaBar Level 1 Trigger Upgrade.

| | | |
|-----|---|---|
| 1 | Seed Track Finder Overview..... | 2 |
| 1.1 | Block Diagram Description..... | 2 |
| 2 | Algorithm Implementation..... | 4 |
| 2.1 | Megabus Decoder..... | 4 |
| 2.2 | Seed phi correction LUT..... | 5 |
| 2.3 | Subtracting Seed Phi..... | 6 |
| 2.4 | Pattern Recognition Matrix (PRM)..... | 6 |
| 2.5 | Expected Phi Position LUT..... | 8 |
| 2.6 | Best Segment Finder..... | 8 |
| 3 | A7 and A10 Seed Track Finder Differences..... | 8 |
| 4 | Constants Generation..... | 8 |
| 5 | FPGA Resource Usage..... | 8 |
| 6 | Latency..... | 9 |
| 7 | Testing..... | 9 |

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

1 Seed Track Finder Overview

1.1 Block Diagram Description

Figure 1 shows a block diagram of the ZPD Seed Track Finder. The basic algorithm steps are:

- Decode input segments arriving on the Megabus and store them in memory.
- Convert the seed segment phi value into phi units for each superlayer.
- Subtract the seed phi value from every segment phi to get relative phi values.
- Fill the Pattern Recognition Matrix (PRM), counting how many segments are consistent with each curvature and tan λ combination.
- Find the best curvature and tan λ combination from the peak of the PRM.
- Determine the expected phi values for a track with that curvature and tan λ .
- Find the closest segments to those expected phi values.
- Output to the Fitter:
 - curvature bin (6 bits)
 - tan λ bin (6 bits)
 - mask of which superlayers have good segments on this track (10 bits)
 - segment relative phi values (10 segments of 11 bits each)

Each Finder processes two seed segments serially, first a seed segment in sector 2 followed by a seed segment in sector 3.² Each Finder has an ID 0-5 which determines which seed segments are processed.

| Finder Number | Seed Segment | | |
|---------------|--------------|--------|-------|
| | SL | Sector | Order |
| 0 | 10 | 2 & 3 | 0 |
| 1 | 10 | 2 & 3 | 1 |
| 2 | 10 | 2 & 3 | 2 |
| 3 | 7 | 2 & 3 | 0 |
| 4 | 7 | 2 & 3 | 1 |
| 5 | 7 | 2 & 3 | 2 |

The implementation presented here is based upon using seed segments in superlayer A10. The details of the Pattern Recognition Matrix and look up table (LUT) constants will be different for seed segments in superlayer A7 but the overall design and resources used are very similar.

² A sector is $2\pi/16$ radians in azimuth. Each ZPD receives segments from 6 sectors numbered 0 to 5. Each ZPD processes tracks which have seed segments in its central two sectors – sectors 2 and 3.

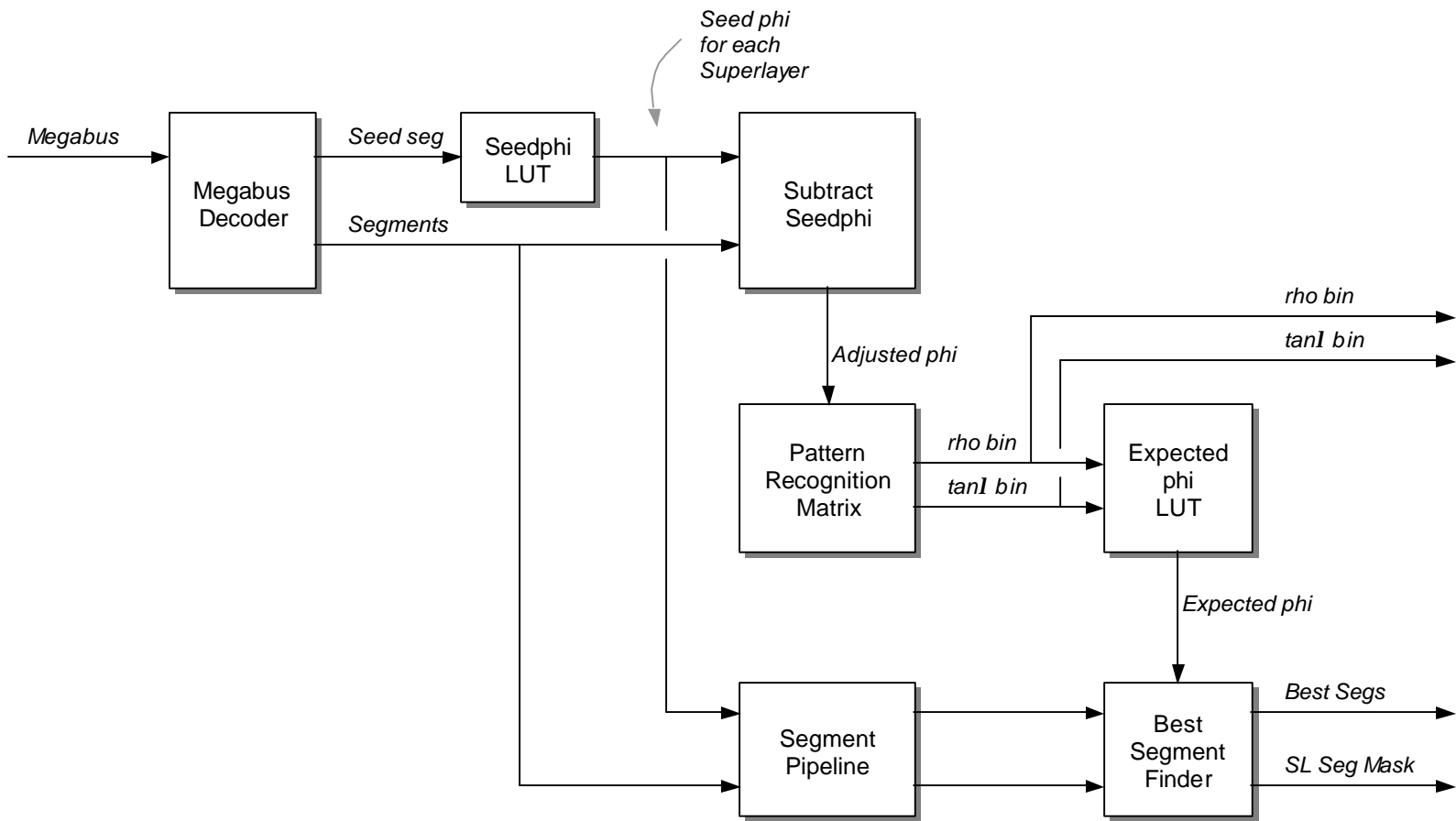


Figure 1: Seed Track Finder block diagram.

2 Algorithm Implementation

2.1 Megabus Decoder

2.1.1 Megabus Input

The segments arrive on the MegaBus on 75 LVDS pairs clocked at CLK120. These are decoded into 144 segments and stored in dual port memory for algorithm processing. The segments are saved using one port while the second port is used to read out segments from the previous event for algorithm processing.

The segment format from the TSF and on the Megabus is:

| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----------|----|----|----------|---|---|---|---|-----------|---|---|---|---|
| | M | loc[3:0] | | | phi[5:0] | | | | | dphi[2:0] | | | | |

- mask bit: 1 or 0 to indicate whether this is a valid segment or not.
- loc: location of the TSF pivot cell within the sector
- phi: bin bin with respect to the center of the cell
- dphi: error on phi
- The order in which the segments arrive indicate their superlayer and sector location

5 segments arrive in parallel every clk120 tick in the following superlayer:sector order. Each superlayer:sector combination has 3 segments which arrive sequentially. Seed segment locations are highlighted in yellow.

| clk120 Tick | MB Segment | | | | |
|-------------|------------|-----|-----|------|------|
| | 4 | 3 | 2 | 1 | 0 |
| 0 – 2 | 1:1 | 1:3 | 1:5 | 10:1 | 10:3 |
| 3 – 5 | 1:2 | 1:4 | 1:0 | 10:2 | 10:4 |
| 6 – 8 | 2:1 | 2:3 | 2:5 | 7:1 | 7:3 |
| 9 – B | 2:2 | 2:4 | 2:0 | 7:2 | 7:4 |
| C – E | 3:1 | 3:3 | 3:5 | 5:1 | 5:3 |
| F – 11 | 3:2 | 3:4 | 5:0 | 5:2 | 5:4 |
| 12 – 14 | 9:1 | 9:3 | 9:5 | 4:1 | 4:3 |
| 15 – 17 | 9:2 | 9:4 | | 4:2 | 4:4 |
| 18 – 1A | 6:1 | 6:3 | 6:5 | 8:1 | 8:3 |
| 1B – 1D | 6:2 | 6:4 | | 8:2 | 8:4 |
| 1E – 1F | | | | | |

2.1.2 Segment Memory

The segments from each event are stored in memory in the order:

| | SL 1 | | | SL2 | | | SL3 | | | SL4 | | | SL5 | | | SL6 | | | SL7 | | | SL8 | | | SL9 | | | SL10 | | |
|----------|------|----|----|-----|----|----|-----|----|----|-----|----|----|-----|----|----|-----|----|----|-----|----|----|-----|----|----|-----|----|----|------|----|----|
| 0 | 1a | 3a | 5a | 1a | 3a | 5a | 1a | 3a | 5a | 1a | 3a | 1a | 3a | 5a | 1a | 3a | 5a | 1a | 3a | 5a | 1a | 3a | 1a | 3a | 5a | 1a | 3a | 5a | 1a | 3a |
| 1 | 1b | 3b | 5b | 1b | 3b | 5b | 1b | 3b | 5b | 1b | 3b | 1b | 3b | 5b | 1b | 3b | 5b | 1b | 3b | 5b | 1b | 3b | 1b | 3b | 5b | 1b | 3b | 5b | 1b | 3b |
| 2 | 1c | 3c | 5c | 1c | 3c | 5c | 1c | 3c | 5c | 1c | 3c | 1c | 3c | 5c | 1c | 3c | 5c | 1c | 3c | 5c | 1c | 3c | 1c | 3c | 5c | 1c | 3c | 5c | 1c | 3c |
| 3 | 2a | 4a | 0a | 2a | 4a | 0a | 2a | 4a | 0a | 2a | 4a | 2a | 4a | 0a | 2a | 4a | 0a | 2a | 4a | 0a | 2a | 4a | 2a | 4a | 0a | 2a | 4a | 0a | 2a | 4a |
| 4 | 2b | 4b | 0b | 2b | 4b | 0b | 2b | 4b | 0b | 2b | 4b | 2b | 4b | 0b | 2b | 4b | 0b | 2b | 4b | 0b | 2b | 4b | 2b | 4b | 0b | 2b | 4b | 0b | 2b | 4b |
| 5 | 2c | 4c | 0c | 2c | 4c | 0c | 2c | 4c | 0c | 2c | 4c | 2c | 4c | 0c | 2c | 4c | 0c | 2c | 4c | 0c | 2c | 4c | 2c | 4c | 0c | 2c | 4c | 0c | 2c | 4c |

The numbers in the table refer to the segment sector, with a, b, c referring to the 3 possible segments for each superlayer:sector combination. The segments in each row are read out simultaneously by the Finder algorithm so that all superlayers are processed in parallel, odd sectors first followed by even sectors.

2.1.3 Sector + loc + phi flattening

The incoming segments have a 4 bit cell location (loc) and 6 bit phi bin (phi) with respect to that cell. The Megabus decoder flattens this information into a signed phi value measured with respect to the middle of the ZPD coverage.

$$\text{signedPhi} = \text{sector offset} + 32 * \text{loc} + \text{phi}$$

For a given segment, the sector offset is a constant which is hard coded into the firmware:

$$\text{sector offset} = (\text{sec} - 3) \times \text{phiBinsPerSec}$$

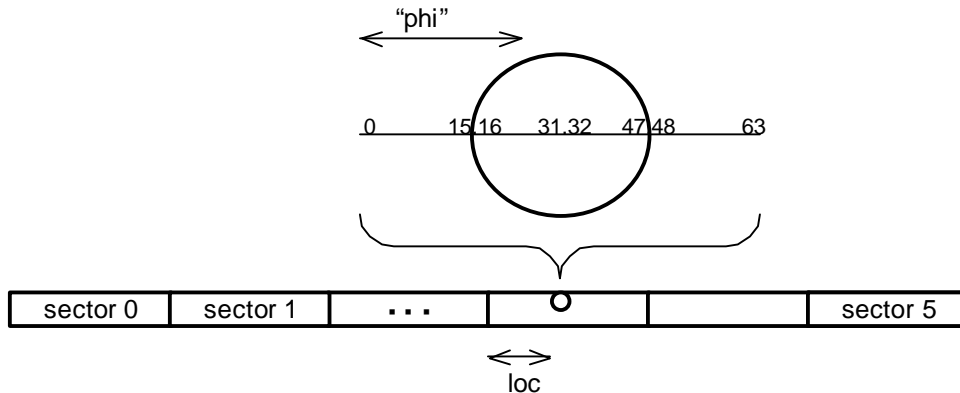


Figure 2: Segment phi location information.

2.2 Seed phi correction LUT

Since each superlayer has a different number of phi bins, the phi bin coordinates of one superlayer must be translated into the phi bin coordinates of another superlayer if they are to be compared. The Seed phi correction LUT translates the seedphi bin in the seed layer into the phi bin units for each of the other superlayers.

2.3 Subtracting Seed Phi

The seed track finding algorithm works with the difference in phi between a seed segment and every other segment in order to remove any dependence upon an absolute phi value.

2.4 Pattern Recognition Matrix (PRM)

The Pattern Recognition Matrix (PRM) is a binned grid of track curvature and $\tan\lambda$ values which records how many segments are consistent with each curvature and $\tan\lambda$ hypothesis. The outputs of each bin are 3 signals, indicating whether the matrix bin has 9, 8, or 7 segments consistent with that curvature and $\tan\lambda$ value in addition to the seed segment (all 0's indicates 6 or fewer consistent segments).

2.4.1 Hit Registers

As an intermediate step between subtracting the seed phi value and filling the PRM, a set of hit registers are filled. After subtracting the seed phi, each segment has a 10 bit relative phi (relPhi) value, representing its distance in fine-phi units from the seed segment phi location. This is flattened into a Hit Register which contains one bit for every four possible relPhi values. The outputs of these registers are mapped to the input of the Pattern Recognition Matrix bins.

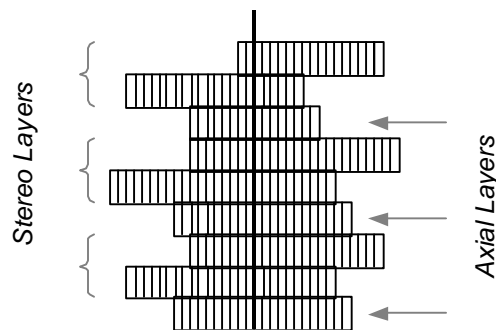


Figure 3: Ranges of the relPhi registers for an A10 seed. Ranges are not to scale.

2.4.2 Hit Registers to Pattern Recognition Matrix Mapping

Each curvature and $\tan\lambda$ bin of the PRM has relative phi ranges for each superlayer which are consistent with a track having those curvature and $\tan\lambda$ values. This mapping from relPhi values to PRM bins is accomplished by ORing together the appropriate outputs from the Hit Registers. Each PRM bin then counts how many superlayers have hits in the Hit Registers which are within its range.

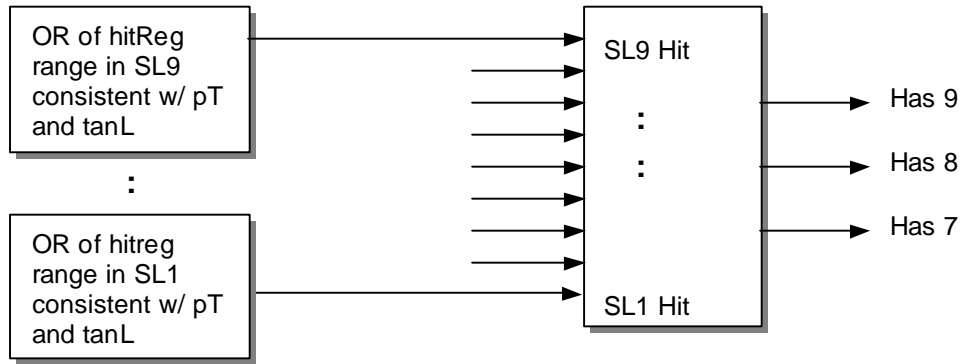


Figure 4: One bin of the Pattern Recognition Matrix

2.4.3 Rho and tanλ Registers

After the Pattern Recognition Matrix has been filled, the curvature (rho) and $\tan\lambda$ Registers select out the values of rho and $\tan\lambda$ which have maxima in the Pattern Recognition Matrix. I.e., if the maximum of the Pattern Recognition Matrix is 7, the rho register contains a 1 for every rho value which has a 7 for some value of $\tan\lambda$, and 0 otherwise.

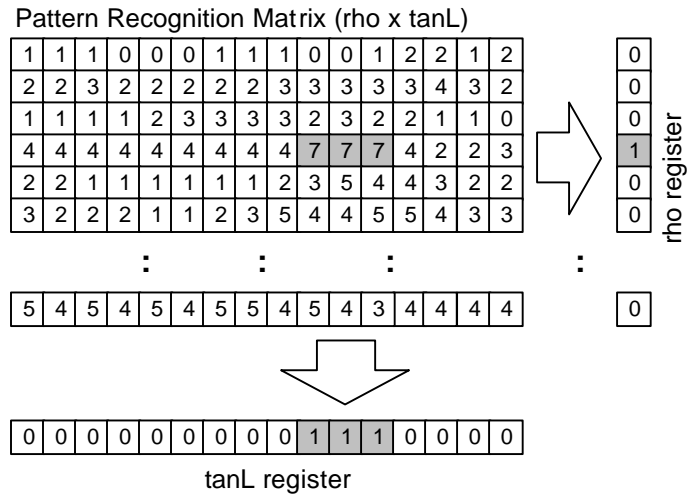


Figure 5: Example of filling the rho and tanλ Registers from the Pattern Recognition Matrix.

The average of the tanλ and rho register peaks forms the estimated tanλ and rho values for the seed track.

2.4.4 Pattern Recognition Matrix Size

The default size of the PRM is 20 curvature bins by 13 tanλ bins, covering $-1.75 < \tan\lambda < 3.0$ and $p_T > 200$ MeV. More bins may be used to slightly improve the algorithm performance at the cost of FPGA resources. Changing the binning requires regenerating the VHDL constants (see section 4), changing the values in one Fitter LUT, and re-implementing the Finder/Fitter.

2.5 Expected Phi Position LUT

After a rho and $\tan\lambda$ estimate as been made, the best segments for that hypothesis must be found. The expected phi LUT returns the expected phi position for a segment in each superlayer for the rho and $\tan\lambda$ hypothesis determined by the Pattern Recognition Matrix.

2.6 Best Segment Finder

Once the expected segment positions are known, the Finder algorithm loops over all segments and finds the closest match for each superlayer.

3 A7 and A10 Seed Track Finder Differences

The firmware has been written to minimize the code differences between the Finders for A10 seeds and for A7 seeds. The differences are:

- Seedphi conversion LUT values
- Relative phi to Pattern Recognition Matrix bin mapping
- Which 9 out of 10 superlayers are counted for finding the PRM maximum
- Expected segment phi position LUT values

These differences appear in only 3 VHDL source code files: the package which contains the LUT constants and two components of the PRM logic. The Finder version can be selected using a generic constant on the top level diagram. This design organization is easily maintainable for the two versions since they are identical for the majority of the source code.

4 Constants Generation

The VHDL constants for the PRM relative phi to PRM mapping and the LUT default values are automatically generated by the C++ code which simulates the algorithm. This ensures that the same mapping is used in the hardware and software and makes it easy to generate new constants when algorithm updates are available.

5 FPGA Resource Usage

The FPGA resource usage of the Finder algorithm is dominated by the relative phi registers and the Pattern Recognition Matrix. Together with the Fitter and diagnostic memories, the total resources used are:

| Xilinx Virtex-II | 4000 | 3000 |
|------------------|------|------|
| Logic Blocks | 44% | 72% |
| RAM Blocks | 74% | 93% |
| Multipliers | 20% | 26% |

The Finder/Fitters will use the Xilinx Virtex-II 4000 FPGAs. The next smaller chip in this series, the 3000, is included for comparison.

6 Latency

The Finder requires one clk4 to receive the data on the Megabus, one clk4 to fill the Pattern Recognition Matrix and find its peak, and one clk4 to find the best segments on the track. For the full ZPD latency, see the ZPD Design Overview.

7 Testing

The Finder has been tested with hundreds of MC tracks and thousands of perfect toy tracks, comparing the results to the C++ software used to study the algorithm performance. 98% of the tracks have a perfect bitwise match with the C++ software simulation results. Of the tracks which fail to have an exact match, about half of them differ by only 1 or 2 segments while the other half have more significant differences. The source of these differences with the simulation is still being studied.