



*Level 1 Pixel Trigger Data Flow Analysis*

**BTeV Document 00??**

**19 September 2001**

Gustavo Cancelo

## Table of Contents

1	Some System definitions.....	1
1.1	Introduction .....	1
1.2	The Pixel Detector structure.....	1
1.3	The Input data file: .....	2
1.3.1	Some file statistics .....	2
1.3.2	Front-end bandwidth .....	3
1.4	Pixel Preprocessor and Segment Tracker .....	3
1.5	The Pixel Front end .....	4
2	Data flow analysis in the Pixel Preprocessor and Segment Tracker .....	5
2.1	The Pixel Preprocessor Architecture.....	5
2.2	The Pixel Preprocessor queuing analysis and simulation .....	6
2.2.2	The TS-ordering queues .....	8
2.2.2.1	Time Stamp distribution and inefficiency in the TS-ordering queues .....	11
2.3	The Segment Tracker Architecture .....	17
2.4	Analysis and simulations of the BB33 dataflow .....	18
2.4.1	Analysis of the BB33 queues as events from the buffer manager .....	18
2.4.2	Analysis of the BB33 queues as events from the buffer manager .....	20
2.4.3	Latency and Processing Times.....	23
3	Conclusions.....	23

***Disclaimer:***

***This document is still in preliminary stage. Some sections are incomplete or need more work. Many calculations and simulation results depend on input calculations, which are also preliminary. Use with care.***

## Pixel Trigger Queuing Analysis and Behavioral Simulations

### 1 Some System definitions

#### 1.1 Introduction

The following document summarizes the results obtained by modeling and simulating part of the Level 1 Pixel Trigger Processor for BTeV. The portion modeled and simulated corresponds to the Trigger section that process data from a Pixel Detector Triplet. A Pixel Detector Triplet is depicted in Figure 2.

The Level 1 Pixel Trigger architecture has been described elsewhere [ref]. In the current analysis it is assumed that the Level 1 Pixel Trigger Processor is subdivided in a number of parallel branches called *highways* (see Figure 3). The mapping of the Pixel data onto the highways is based on the data's Time Stamp (TS). For instance, if the number of highways is  $N$ , highway1 will receive data with Time Stamps  $1, N+1, 2N+1, \dots$ , highway2 will receive  $2, N+2, 2N+2, \dots$  and so forth. It is assumed that data acquired at any TS is uncorrelated with other TS, hence can be processed independently.

#### 1.2 The Pixel Detector structure

The data flow analysis and simulations make extensive use of Pixel Simulation Files. These files are Geant simulations of the BTeV detector [ref Penny]. The particular file use for the Trigger data flow simulations provides information of 3 complete Pixel Stations ( $N^{\circ}$ : 15, 16, and 17). The stations are laid out as shown in Figure 2.

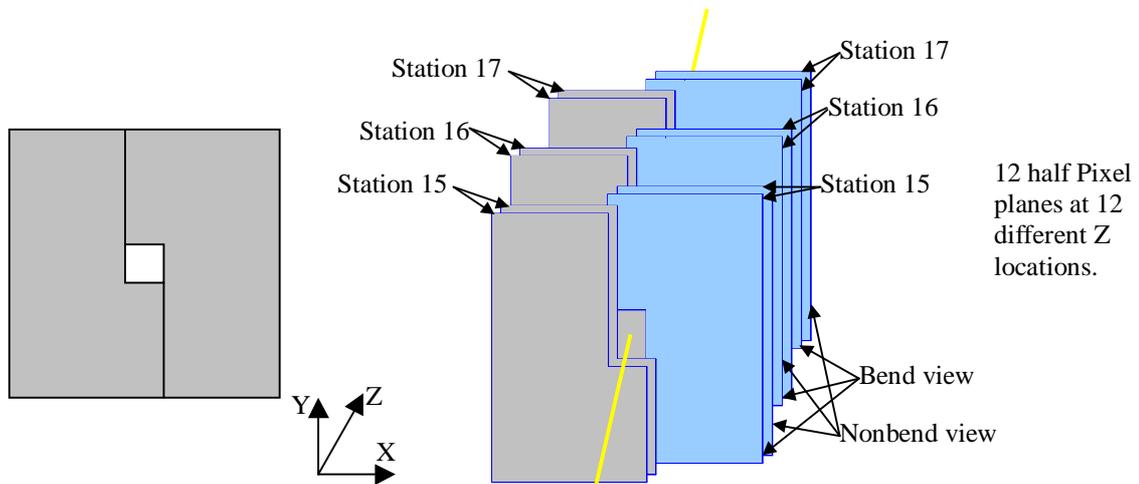


Figure 1 Pixel Detector Triplet

A Pixel Station is composed of two Half Pixel Stations. Each Half Station has one half of a bend view detector side and one half of a non-bend view detector side mounted on the same mechanical substrate. They make a Pixel Half Station. This two half detector planes, the bend and the non bend views, are at about 0.57cm apart. The other Half Station that completes the Station is shifted about 2.25cm in Z. For instance, the right side Half Station 15 is centered in Z between the left side Half Stations 15 and 16, and so forth. Right and left halves of the Pixel Stations keep its separation through the Pixel Preprocessor and Segment Tracker modules of the L1 Pixel Trigger Processor as detailed below.

### 1.3 The Input data file:

The Pixel data was generated using the following parameters [ref]:

- Pixel size: 50 x 400 microns,
- Chip size: 22 columns, 128 rows
- Magnetic field: 1.6T
- Threshold: 2000 e-
- Total N° of Bunch Crossings (BCO): 745
- Luminosity:  $2 \cdot 10^{32} \text{ cm}^{-2} \text{ s}^{-1}$  (4 interactions per BCO on average)
- No of stations simulated: 3, corresponding to the central Triplet of the Pixel Detector (i.e. stations 15, 16, and 17). Each Station is double-sided with one bend-view plane and one non-bend-view plane.

#### 1.3.1 Some file statistics

Total N° of Hits (6 planes): 260,855

Avg. No of tracks per BCO (1 plane): 25.14

Avg. No hits per BCO (1 plane): 58.35

Avg. No of hits generated by a track crossing a single sided plane: 2.32

Figure 2 shows the hit distribution in one Half Plane.

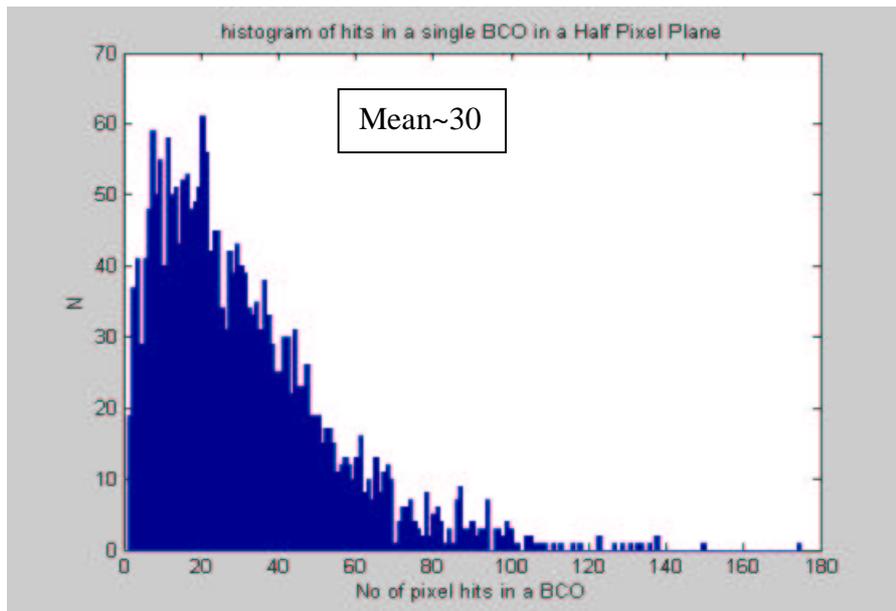


Figure 2 Pixel hit distribution in one Half Plane

The Pixel Preprocessor and Segment Tracker process pixel data coming from a Half Station. The Pixel Data goes from the Pixel Detector planes through the Data Combiner boards and into the L1 Trigger. The Pixel Detector Data Combiners split the data into a number of highways. As a consequence, the average data rate into the L1 Trigger Pixel Preprocessor equals the total average data rate of a Half Pixel plane divided by the number of Highways in the system. In the following example we consider that the Pixel Front-ends will split the data into 8 highways. The Pixel Preprocessors process one Highway from one Half Plane. The Segment Trackers process 6 Highways, one from each Half Pixel plane which make a Half Station Triplet (as shown in Figure 3).

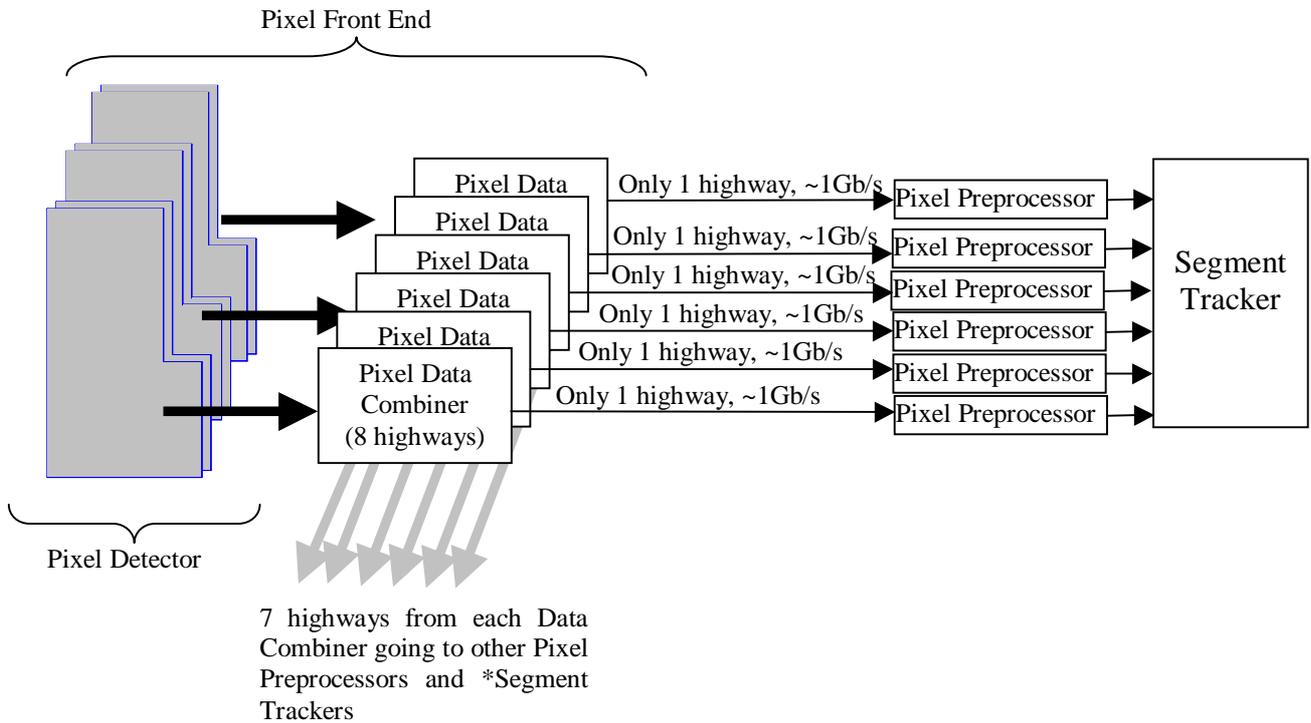


Figure 3 Pixel Front End Highways

### 1.3.2 Front-end bandwidth

The bandwidths are calculated using Half Planes and Half Stations as units. Based on the current file, the Half Pixel Plane generates an average of about 30 hits per BCO. If a pixel hit is represented by a 4 byte binary word, the total bandwidth per Half Plane is very close to 1Gbyte/s or 8Gb/s.

Since the Pixel Data Combiner boards split the data in 8 highways, the Pixel Preprocessor and Segment Trackers receive 6 x 1Gb/s links from the 6 Half Plane which form a Half Station Triplet.

### 1.4 Pixel Preprocessor and Segment Tracker

The Pixel Preprocessor and Segment Tracker have a functional block diagram as shown in Figure 4. A Segment Tracker process the data from a Triplet of two-sided Pixel Half-Stations. A Pixel Preprocessor module process a single-sided Pixel Half Plane (i.e. the bend view or the non bend view). The block diagram in Figure 4 shows 6 Pixel Preprocessor modules and 1 Segment Tracker. Even if the hardware can accommodate 6 Pixel Preprocessors and 1 Segment Tracker it still needs to send Pixel Preprocessor data to the neighboring Segment Tracker processors because there are up to 3 Segment Tracker processors using the same Pixel Preprocessor data. The function of the Segment Preprocessor Interconnection is to distribute the data to up to three Segment Tracker stations.

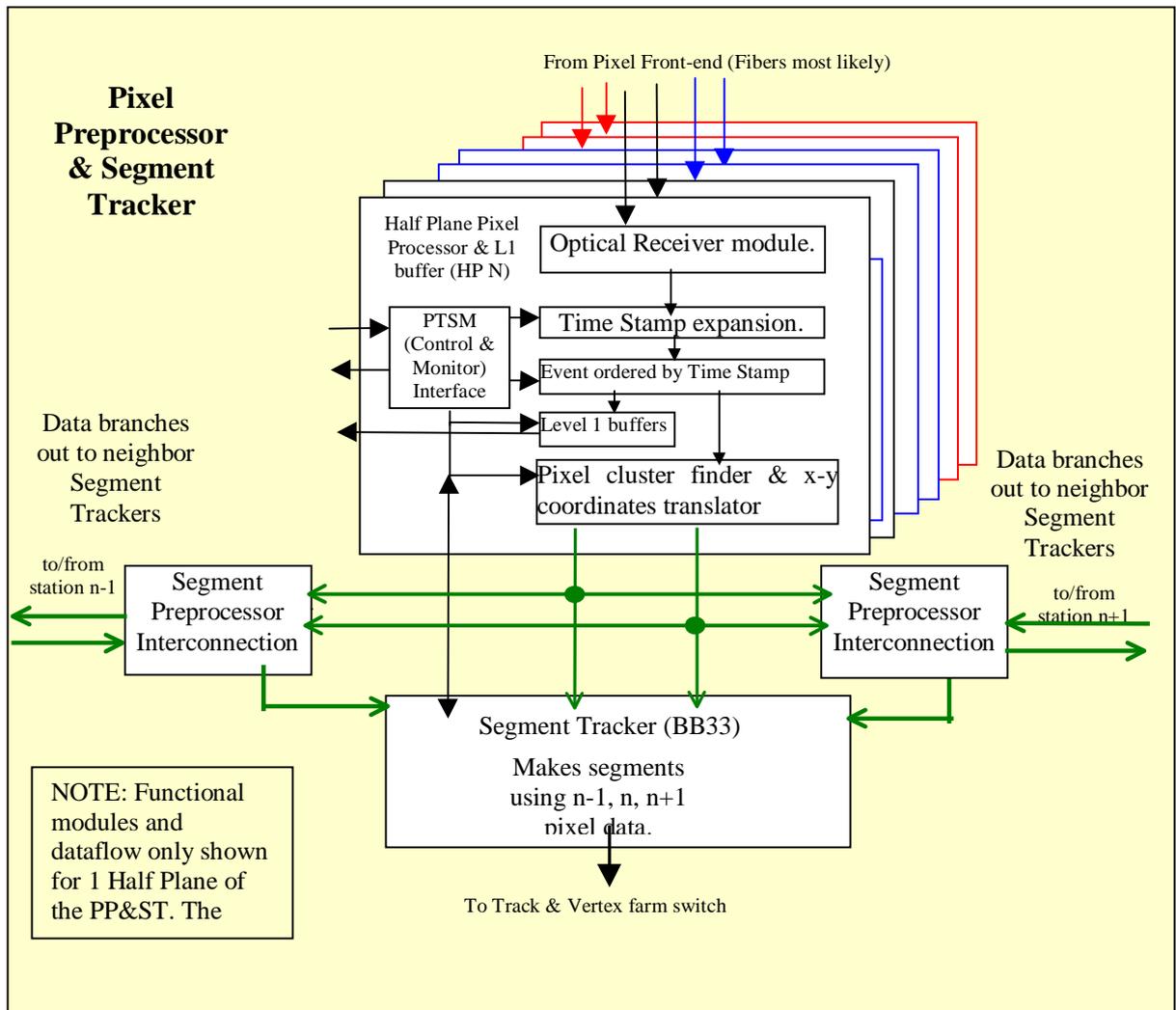


Figure 4 Pixel Preprocessor and Segment Tracker functional block diagram

### 1.5 The Pixel Front end

The purpose of the current document is to report on the data flow in the Pixel Preprocessor and Segment Tracker of L1 Pixel Trigger. However, some modeling and simulation of the Pixel Front End was necessary to obtain realistic input data to the L1 Pixel Trigger. The Pixel simulation files provide a set of chronologically organized events. A typical line of those files is:

BCO	Plane No	bend/nonbend	Xcoord	Ycoord	Zcoord	No pix hit
-----	----------	--------------	--------	--------	--------	------------

However, the data does not arrive chronologically sorted to the Pixel Trigger. The process of data readout in the FPIX chips of the Pixel Detector and the process of data readout and highway sorting in the Pixel Data Concentrator boards scramble the data.

A more realistic input data stream to the Pixel Trigger is needed in order to have better estimations of timing, bandwidths, and queue sizes.

The model used for the Pixel Front End is detailed in the Appendix. The Pixel Detector model considers the fact that the hit density in the FPIX chips is not uniformly distributed across the Pixel Plane. The hit distribution follows an inverse relation of the radial distance to the beam. As a consequence a FPIX chip closer to the beam needs more serial communication channels to the Pixel Data Concentrators.

The Pixel Data Concentrator model includes a two-layer switch to route the data from about 84 inputs to 8 output highways.

## *2 Data flow analysis in the Pixel Preprocessor and Segment Tracker*

The purpose of the data flow analysis in the L1 Trigger is to estimate the processing and storage requirements, to create a timing and queuing map and to optimize hardware resources. The tools used in the data flow analysis are two: queuing theory and behavioral simulations. The validity of the results depends on the assumptions made in the modeling and the limitations of the input files used during simulation runs. Some safety margins will be used in the design to account for all the unmodeled dynamics.

### *2.1 The Pixel Preprocessor Architecture*

The Trigger Processor system must provide one trigger accept/reject per BCO, on average. This is achieved by deeply pipelining the event processing. In order to optimize the throughput a number of buffers (queues) are needed. The buffers smooth out data rate fluctuations and diminish processor's idle times. An advantageous feature of the Trigger Processor System is that the data events are independent (i.e. uncorrelated) BCO wise. This characteristic facilitates the pipelining of the Trigger Processor by introducing many processing units, which are allowed to work, asynchronously, on uncorrelated events. Using queues between each two of those parallel processors allows pipelining by decoupling among data flows between processors. Figure 5 shows the proposed queuing model of the Pixel Preprocessor.

The first queue in the Pixel Preprocessor is generated by Input Link Receivers. The serial input data from the optical links are unserialized and placed in the Input Link buffers.

The Time Stamp (TS) field of the input data is expanded to the full length needed to match the maximum trigger latency. Latency here is defined as the time it takes the Trigger System to make a decision on weather to accept or reject an event. The Segment Tracker and the Level 1 Buffers need the data sorted by TS. Since the data from the Pixel Detector Front Ends come TS unsorted, they are sorted by the TS-ordering module. The TS-ordering module transfers the input data from the Input Link Buffers to separate queues where the data is ordered by TS. The TS ordering queues are the second set of queues in the Pixel Preprocessor. The number of open queues varies according to the TS distribution in the data stream.

The time each TS-ordering queue is open to receive data must be set deterministically based on data distribution analysis. Since the input data is chronologically unsorted and the event size is variable, the end-of-event time is unknown. We could wait a "long time" and still not be sure that an event corresponding to a certain TS is complete. Hence, the most logical approach is to make the departure time from the TS Ordering queues deterministic with respect to its arrival time. The time every data queue must be kept open for queuing (i.e. buffering input data of a certain TS) will affect the latency of the Trigger.

The third queue in the Pixel Preprocessor model (Figure 5) is the input to the Pixel cluster finder and x-y coordinate translator (XYPC). This module reads data from an input buffer and writes grouped pixel clusters into an output buffer. The input Pixel data is in row column form, that means the hits are represented by the physical row and column address of the Pixel Detector chip which detected those hits. A single track may generate more than one hit in the detector chip. The XYPC processor translate a whole group of row column hits in a single x-y pair, where x and y are in metric units with respect to the origin of the coordinate system. The XYPC reduces the event size by a factor proportional to the average pixel cluster size.

The fourth queue level (Figure 5) is the XYPC output buffer. It holds the x-y cluster data until is ready to be processed by the Segment Tracker. The Segment Tracker needs x-y cluster data from the two neighbor stations. The three queues generated by stations N-1, N, and N+1 are independent and work asynchronously.

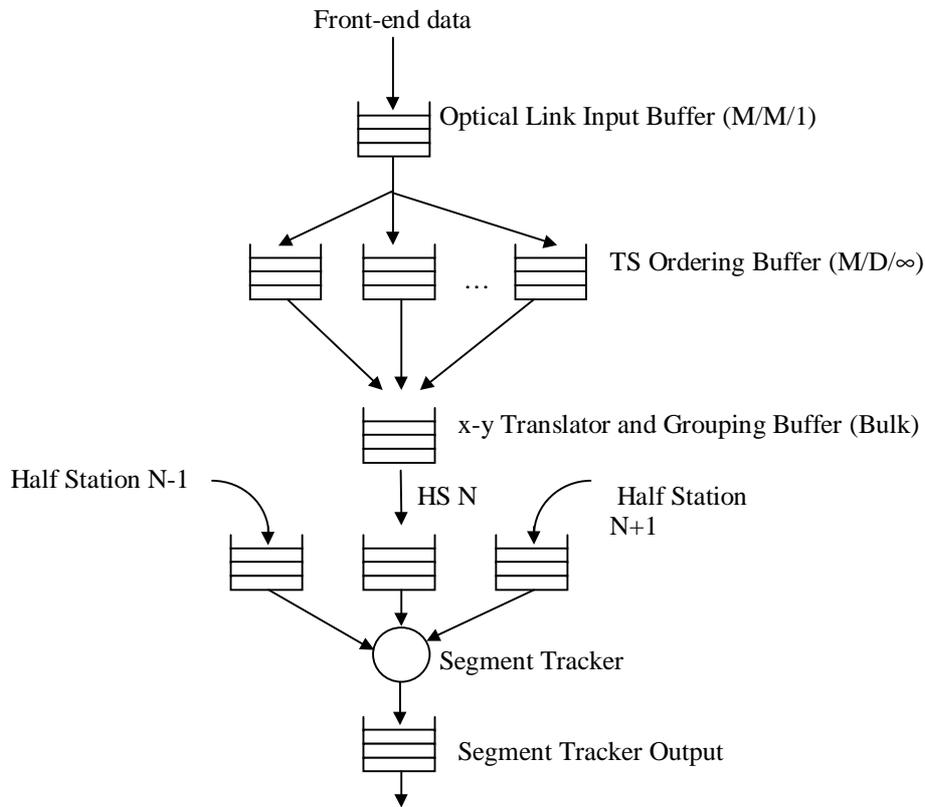


Figure 5 Queuing model for data flow analysis

## 2.2 The Pixel Preprocessor queuing analysis and simulation

### 2.2.1 The Input Link Buffer

The Input Link Buffer queue is fed by the Optical Receiver electronics. The expected maximum input bandwidth of the optical channel is 2 Gb/s. This is equivalent to 250Mby/s or 125 Mega-16bit words/s, which is close to the maximum frequency that an FPGA can handle. However, the analysis of the Input simulation file shows that the average bandwidth is about 1Gb/s Figure 3.

The processing time on the Input Link Buffer data is deterministic. The algorithm will do the following:

- Add an expansion field for the data TS.
- Create an output queue (unless it already exists) and place the data onto that queue based on the data's TS.

Algorithm:

```

if queue with data's TS already exist
    enqueue data in existing queue with its TS expanded
else
    enqueue data in a new queue with its TS expanded
end

```

Since the processing time is deterministic, the mean Input Link Buffer output rate,  $\mu$ , is constant and its variance is 0. If  $\mu$  is greater than the maximum input bandwidth of the optical channel (62.5 Mw/s), the Input Link Buffer size needed is just 1 word deep. Note that  $\mu$  must be, at least, greater than the average input bandwidth of the optical channel  $\lambda$  to avoid queue instability. The value of  $\lambda$  is directly proportional

to the clock frequency of the input receiver and the Input Link Buffer's utilization factor. In the later case, the Input Link Buffer behaves as a M/D/1 queue. The average queue size can be calculated by

$$E(N_q) = \frac{\rho}{1-\rho} - \frac{\rho^2}{2(1-\rho)} \quad \text{where} \quad \rho = \frac{\lambda}{\mu}$$

For the current simulation the input is distributed as shown in Figure 6.

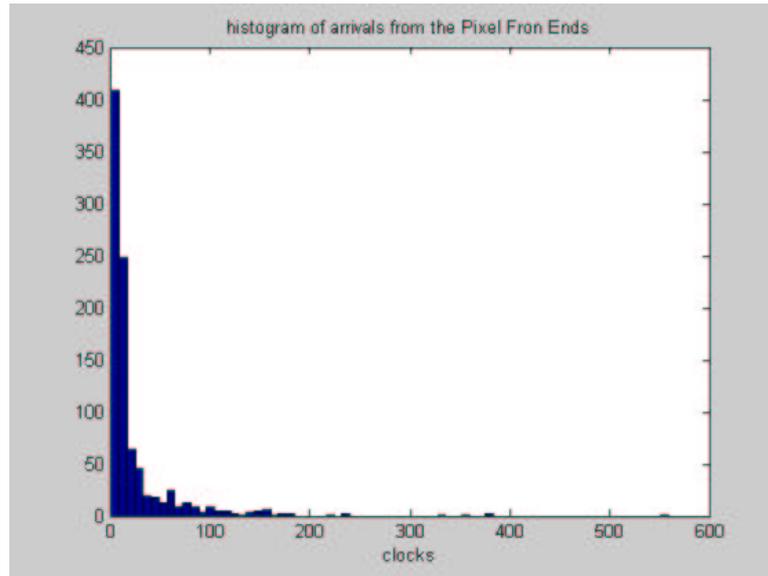


Figure 6 Pixel Hit arrival distribution in the receiver's queue

The arrivals at the receiver queue are exponentially distributed with  $\lambda=0.26$  hits/clock and  $\mu=0.5$  hits/clock. Hence,

$$\rho = \frac{\lambda}{\mu} = \frac{0.26}{0.5} = 0.53$$

The mean queue size becomes,  $E(N_q) = 0.82$ .

The simulations show that the Input Link Buffer queue does not exceed 1 word deep.

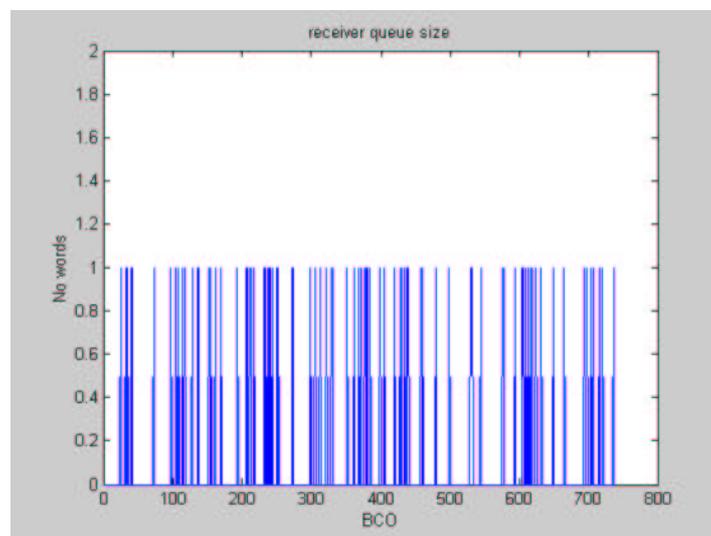


Figure 7 Receiver queue size

### 2.2.2 The TS-ordering queues

During the TS-ordering process queues are born and also die. A new queue is born when the TS event ordering process receives data with a TS different to all the ones in the existing queues. A queue dies when the data reception for that event is complete. As said above this time must be chosen deterministically. For the current example this time will be equal to a complete revolution of the TS clock, that is  $\sim 21\mu\text{s}$  if we roll over the TS counter at 159 BCOs, or  $33.8\mu\text{s}$  if we roll over at 256 BCOs. This number is, probably, too conservative and adds an unnecessary latency to the data flow. However, it represents a worst case bound. The simulations show data inefficiency in the TS-ordering queue as function of the lifetime of the TS queue as it is shown below.

The full queueing analysis of the TS event ordering is fairly complex because the process must not only consider the queue birth-death distribution but, also, the size distribution of each individual queue. At least, we want to find the first moments of a probability distribution function, which defines the existence of each specific queue and its size. If we look at individual queues this is a non-stationary problem. However, some simplifications can be made. We can define a new process looking only at the number of queues in the TS event ordering system, regardless of their sizes. This new process is a well-defined birth-death Markov chain. Each state represents the number of existing queues in the system (Figure 8). The process can be modeled as a M/D/ $\infty$  process. The birth time of the queues are generated by random queue arrivals. The interarrival times can be considered exponentially distributed. Queue deaths are caused by complete events leaving the system. The interdeparture times are deterministic.

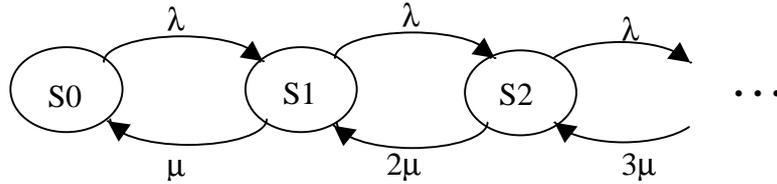


Figure 8 TS-ordering state transition model

When the TS ordering process receives data with a new TS, it opens a new queue immediately. That is, it starts processing the incoming event without queueing it. The response time of the server increases linearly. We can define:

$\lambda$ : queue birth rate

$\mu_k = k\mu$  : queue death rate

$\lambda$  represents the rate at which new queues are generated. From simulations the total Pixel Detector Half Station data rate is shown to be 0.9 events/BCO for 4int/BCO and 0.71 events/BCO for 2int/BCO. This rate is reduced by the fact that the events are separated along K parallel highways based on TS. Considering K=8 and that all TS are equally probable, the data rate in each branch (which is the interesting number here) is:

$$\lambda = 0.1125 \text{ events/BCO for a luminosity of 4int/BCO.}$$

$$\lambda = 0.0875 \text{ events/BCO for a luminosity of 2int/BCO.}$$

In other words, the interarrival time  $T_\lambda$  (i.e the average time between two new queue arrivals) is:

$$T_\lambda = 1/\lambda = 8.88 \text{ BCOs for a luminosity of 4int/BCO.}$$

$$T_\lambda = 1/\lambda = 11.4 \text{ BCOs for a luminosity of 2int/BCO.}$$

$\mu$ , the service rate, is deterministic and equal to the time we want to wait before considering that the event is complete. In this example we set  $\mu$  to  $1/(159 \text{ BCOs})$  or  $0.006289 \text{ BCO}^{-1}$ .

The M/D/ $\infty$  process is always stable. The probability distribution function of this system is given by

$$p_k = \frac{(\lambda/\mu)^k}{k!} e^{-\lambda/\mu} \quad k = 0, 1, 2, \dots$$

The average number of queues in the system is given by:

$$E(N_q) = \frac{\lambda}{\mu} = \frac{0.1125}{0.006289} = 17.89 \text{ queues}$$

The average response time of the system to a job, using Little's formula, is  $T = \frac{E(N_q)}{\lambda} = \frac{1}{\mu} = 159 \text{ BCOs}$ , which is obvious because the system's service time is deterministic.

The simulation of about 750 BCOs shows a similar result (Figure 9). The number of TS queues open increases linearly at the beginning and stabilizes at around 18 queues. If we discard the transitory (the first 200 BCOs) the average number of queues from simulation is 18.38.

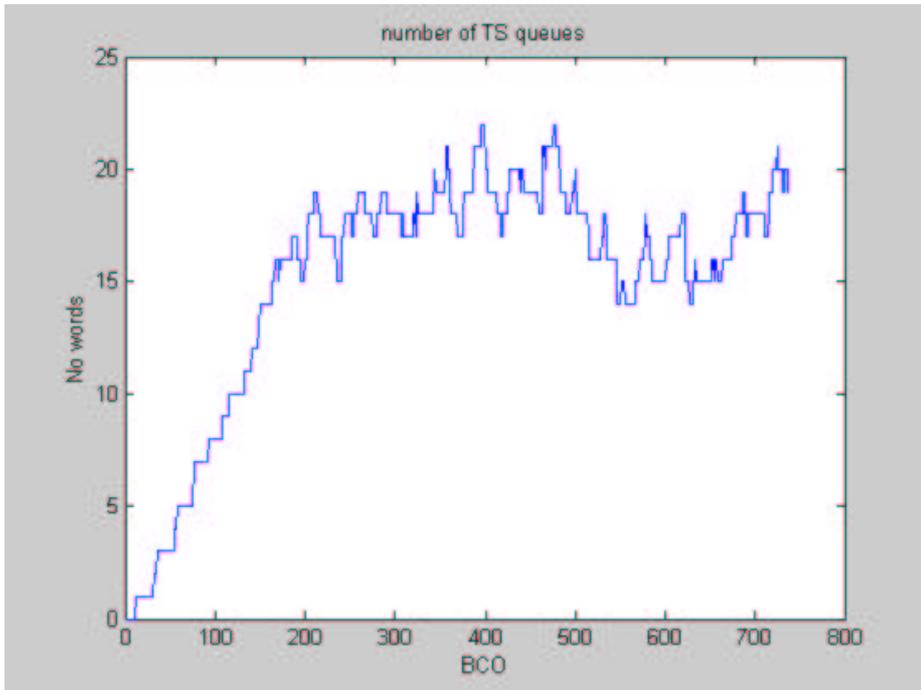


Figure 9 Simulation of the Number of TS queues

Before modeling the individual queues some data bounds can be calculated using the same system model. If we take into account the average number of queues and the average event size we can expect an Avg. Number of data words in all the queues of about:

Avg. N data =  $E(N_q) * \text{Avg. event size} = 17.89 \text{ queues} * 30 \text{ hits} = 536 \text{ hits}$  for a luminosity of 4int/BCO.

Avg. N data =  $E(N_q) * \text{Avg. event size} = 17.89 \text{ queues} * 20 \text{ hits} = 357 \text{ hits}$  for a luminosity of 2int/BCO.

This number is too pessimistic because if we take the averages as deterministic parameters (i.e. the system has always 18 queues open and the event size is constant at 30hits/event) it implies that all the queues are at maximum data capacity. The dynamics of the process tell us that this is not true and the total number of words in the queues must be smaller than that.

The analysis of the individual queues can be performed as follow: We can calculate the conditional probability distribution function of queue occupancies given that there are  $n$  queues and the total sum of data words in the queues is  $m$ . The selection of data in the queues can be modeled as a generalized binomial distribution:

$$P(M_1(t) = m_1, M_2(t) = m_2, \dots, M_n(t) = m_n | M(t) = m, N(q) = n) = \frac{m!}{m_1! m_2! \dots m_n!} p_1^{m_1} p_2^{m_2} \dots p_n^{m_n}$$

where:  $\sum_{i=1}^n p_i = 1$  and  $\sum_{i=1}^n m_i = m$

Since the input data-stream which generates the queues with individual TS is a Poisson process,

$$P(M(t) = m) = e^{-\lambda t} \frac{(\lambda t)^m}{m!}$$

Then, we can take away the conditionality on the total number of words  $m$

$$P(M_1(t) = m_1, M_2(t) = m_2, \dots, M_n(t) = m_n | N(q) = n) = \frac{m!}{m_1! m_2! \dots m_n!} p_1^{m_1} p_2^{m_2} \dots p_n^{m_n} \cdot e^{-\lambda t} \frac{(\lambda t)^m}{m!}$$

the last equation can be written as

$$P(M_1(t) = m_1, M_2(t) = m_2, \dots, M_n(t) = m_n | N(q) = n) = \prod_{i=1}^n e^{-p_i \lambda t} \frac{(p_i \lambda t)^{m_i}}{m_i!} \quad (1)$$

since all the TS are equally probable  $p_1 = p_2 = \dots = p_n = 1/n$

$$P(M_1(t) = m_1, M_2(t) = m_2, \dots, M_n(t) = m_n | N(q) = n) = e^{-(\lambda t/n)} \prod_{i=1}^n \frac{(\lambda t/n)^{m_i}}{m_i!} \quad (2)$$

Equation (2) is still conditioned by a fixed number of queues in the system. However, it let us study the distribution of data in the queues for a certain number of key values. For instance we can let  $n$  be the average number of queues or some upper bound.

What equation (2) shows is that for a given  $n$  the distribution of  $M_1(t) \dots M_n(t)$  are independent Poisson processes with data rate  $\lambda t/n$ . It is also known that as well as the interarrival times in a Poisson Process are exponentially distributed, the k-iterated interarrival of an event in (1) follows a k-stage Earlang distribution. In our case the distribution is conditioned for  $n$  fixed.

We can further simplify the job if we are only interested in the average total number of words in all the TS-ordering queues. The average number of hits in the TS-ordering queues can be calculated using the average number of TS-queues and the average number of hits per event.

$$E(N_q) = \frac{\rho}{1-\rho} \quad \text{where} \quad \rho = \frac{\lambda}{\mu}$$

$$\lambda = \text{hit\_input\_rate} = 0.241124$$

$$\mu = \frac{\text{Avg\_NoTS\_ordering\_queues}}{\text{Deterministic\_queuing\_time} \times 14 \text{clk/BCO}} = 0.242587$$

$$\rho = 0.993966$$

$$E(N_q) = \frac{\rho}{1-\rho} = 165 \text{hits}$$

The simulation of about 750 BCOs shows an average number of words of 202.8 after the transitory.

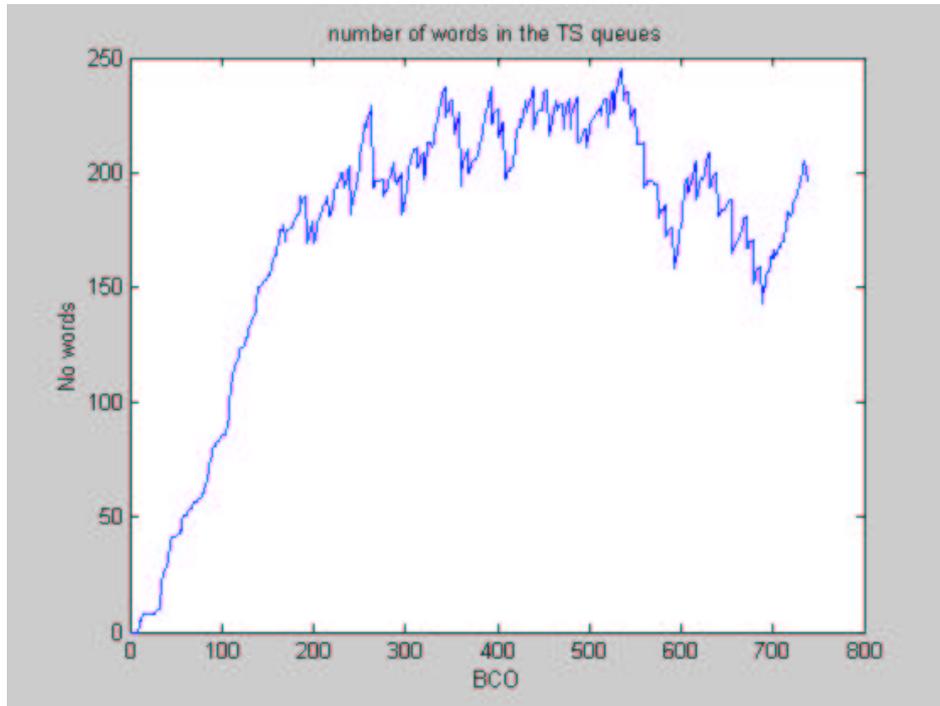


Figure 10 Simulation of the Number of words in the TS queues

The number of simulated BCOs does not allow us to determine whether the function stabilizes at around 200 hits. A longer run is on request.

### 2.2.2.1 Time Stamp distribution and inefficiency in the TS-ordering queues

As said, the TS-ordering process opens one individual queue for each TS in the data stream. These queues are open for data collection during a deterministic time. When that time is over, the queue is closed and loaded into the XYPC input queue for data grouping. All data having a TS field corresponding to a queue that is closed is lost and contributes to inefficiency in the Trigger. This problem can be solved by increasing the time the queues are open for data collection, but that, of course, increases the latency of the Trigger. In other words, it increases the time an event in the entire BTeV detector must be stored waiting for a trigger accept or reject.

The TS scrambling in the data stream is generated by the scattered and asynchronous way in which Pixel data is collected and routed to the Trigger system. The analysis of the Pixel Detector's readout is outside the scope of this document. However, we here present a crude simulation to illustrate the problem. In order to study the Trigger's Pixel Preprocessor we have generated a simplified model of the Pixel Detector and Data Concentrator's readout. A detailed model can be found in the Appendix.

Figure 11a and b shows the distribution of TS spread (i.e. the distribution of times between the first and the last event word with a certain TS).

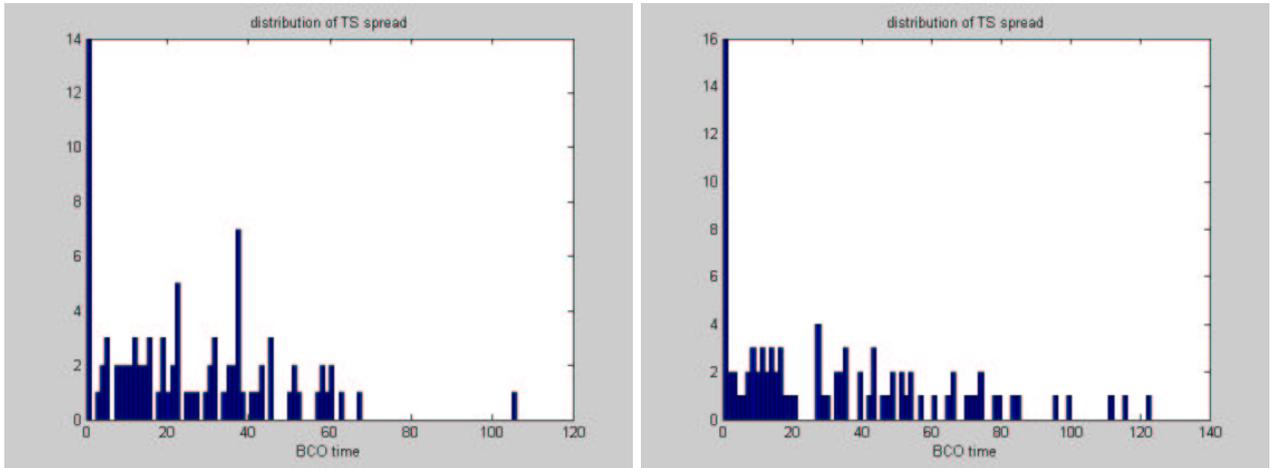


Figure 11 Distribution of TS spread in the data

The Figure 12 a and b show the data inefficiency as a function of the time the TS-ordering queues are open for data collection. The distributions in Figure 11 a and b correspond to “highwayed” data from 2 of the 6 planes that feed a Segment Tracker Triplet. It can be noticed that even when the amount of data generated for each plane is similar, the distributions are quite different and the minimum time for data collection in the TS-ordering queues varies a lot.

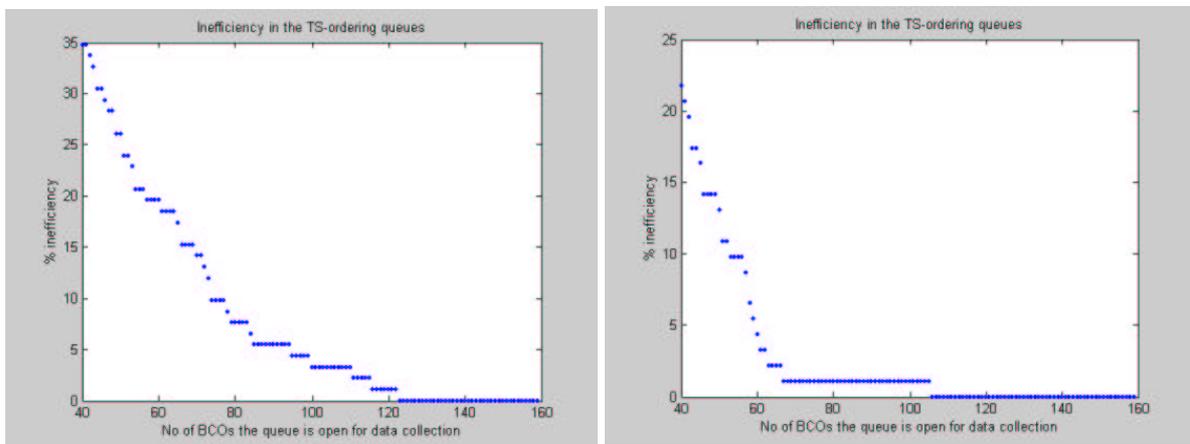


Figure 12 Inefficiency in the TS-ordering queues

### 2.2.3 The x-y pixel cluster (XYPC) queue

The x-y pixel cluster (XYPC) queue can be modeled as a “bulk” M/M/1 process. In such a process the data arrives at the input queue in “bulks”. The x-y translator buffer receives “bulk” arrivals from the output of the TS ordering process. Every time the TS ordering process closes a queue, that entire queue is placed in the x-y translator buffer. This queue is of variable size and equal to the size of the event that generates it. In other words, the x-y translator’s queue is composed by a number of queued customers, which are in turn of variable length. This problem is a generalization of the system with an r-stage Erlangian service, in this case using variable r. The bulk arrival state-transition diagram can be represented as in Figure 13.

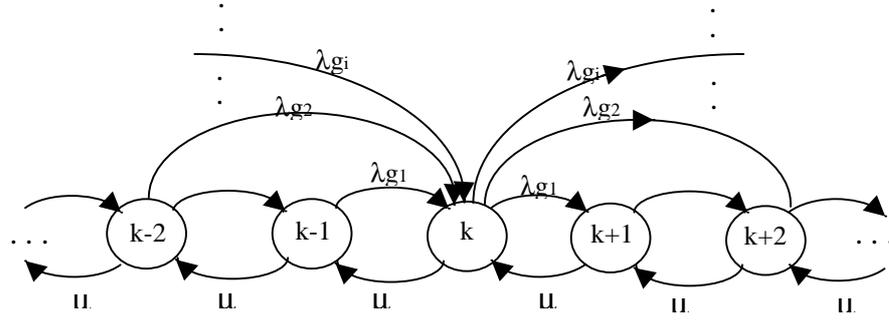


Figure 13 XYPC state transition model

A good idea of the bulk size distribution  $g$  is given by the event size histogram provided by the simulations (Figure 2).

Let  $g_i = \text{Prob}[\text{bulk size is } i]$ , then  $\sum_{i=1}^{\infty} g_i = 1$

The equilibrium equations for the bulk arrival system are:

$$(\lambda + \mu) p_k = \mu p_{k+1} + \sum_{i=1}^{k-1} p_i \lambda g_{k-i} \quad k > 1 \quad (1)$$

$$\lambda p_0 = \mu p_1$$

The numbers we are looking for are the size of the x-y translator queue and the average service time. The solution of the equilibrium equations involves z-transform methods. The bulk M/M/1 queue size in equilibrium suffers a “modulation” effect caused by the changing size of the events (bulks). This modulation is reflected in the discrete convolution shown in equation (1). As we know, discrete convolutions are much easily handled in the z-transformed plane because they turn into the product of the z-transforms. The z-transform of the probability distribution is

$$P(z) = \frac{\mu(1-\rho)(1-z)}{\mu(1-z) - \lambda z[1-G(z)]} \quad (2)$$

Here  $P(z)$  represents the z-transform of the probability distribution of the x-y transform queue size and  $G(z)$  is the z-transform of the probability distribution of the bulk size. The utilization factor  $\rho$  is defined, as usually,  $\rho = 1 - p_0$ . The value of  $\rho$  can, also, be obtained from (2) taking into account that  $P(1)=1$ .

Then,  $\rho = \frac{\lambda G'(1)}{\mu}$ . This result is not surprising because  $G'(1)$  is the average bulk size, hence  $\lambda G'(1)$  is the average arrival rate and  $1/\mu$  is the average service rate.

The average queue size can be directly calculated from (2) using the method of moments.

$$E(N) = \left. \frac{dP(z)}{dz} \right|_{z=1}$$

After some algebra,  $E(N) = \frac{2\lambda G'(1) + \lambda G''(1)}{2(\mu - \lambda G'(1))}$ . Of course, this equation depends on the  $g_k$  distribution.

If we assume that  $g_k$  follows a Poisson distribution then

$$G(z) = e^{(z-1)\alpha}, \text{ where } \alpha \text{ is the spread in the event size distribution.}$$

$$G'(z) = \alpha e^{(z-1)\alpha}$$

$$G''(z) = \alpha^2 e^{(z-1)\alpha}$$

Then expected number of queues in the bulk M/M/1 process is

$$E(N) = \frac{2\lambda\alpha + \lambda\alpha^2}{2(\mu - \lambda\alpha)}. \text{ It can also be expressed in terms of } \rho, E(N) = \frac{2\rho + \rho\alpha}{2(1-\rho)}$$

using  $\lambda=0.0083$ ,  $\mu=0.1072$ ,  $\alpha=25$ ,  $E(N)=0.103$

In fact, as it can be appreciated in Figure 2, the hit distribution is not Poisson. We can approach it much better using a Rayleigh or a Landau distribution.

The Rayleigh distribution can be expressed as:

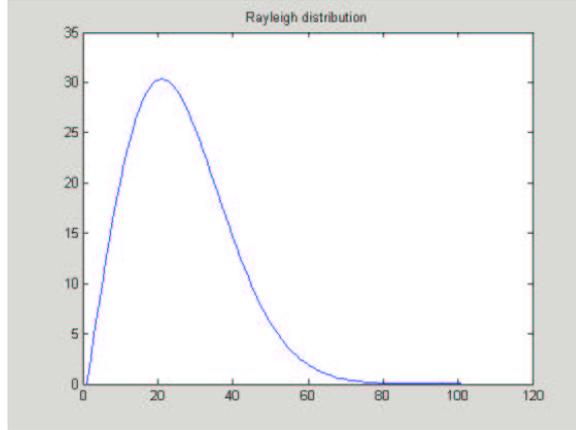


Figure 14 Rayleigh distribution

$$f_X(x) = \frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

The Rayleigh distribution is a continuous pdf. Its Fourier transform can be calculated as

$$F(w) = \int_{-\infty}^{\infty} f(x) e^{-jwx} dx = \int_{-\infty}^{\infty} \frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}} e^{-jwx} dx$$

after solving this we get

$$F(w) = jw\sigma\sqrt{\frac{\pi}{2}} e^{-\frac{w^2\sigma^2}{2}}$$

and its counterpart z-transform is (using  $z = e^{jw}$ ):

$$G(z) = \sigma\sqrt{\frac{\pi}{2}} \ln(z) z^{-\frac{\sigma^2}{2}}$$

the n-iterated derivatives of G(z) are:

$$G'(z) = \sigma\sqrt{\frac{\pi}{2}} \left[ z^{-\left(\frac{\sigma^2}{2}+1\right)} \left( 1 - \frac{\sigma^2}{2} \ln(z) \right) \right]$$

$$G''(z) = \sigma\sqrt{\frac{\pi}{2}} z^{-\left(\frac{\sigma^2}{2}+2\right)} \left[ (-1) \left( \frac{\sigma^2}{2} + 1 \right) \left( 1 - \frac{\sigma^2}{2} \ln(z) \right) - \frac{\sigma^2}{2} \right]$$

The z-transform derivatives calculated at z=1 are

$$G'(1) = \sigma \sqrt{\frac{\pi}{2}}$$

$$G''(1) = -\sigma \sqrt{\frac{\pi}{2}} (1 + \sigma^2)$$

Then expected number of queues in the bulk M/M/1 process is

$$E(N) = \frac{2\sqrt{\frac{\pi}{2}} \frac{\lambda\sigma}{\mu} - \sqrt{\frac{\pi}{2}} \frac{\lambda\sigma}{\mu} (1 + \sigma^2)}{2\left(\mu + \lambda\sigma\sqrt{\frac{\pi}{2}}\right)}$$

Using  $\rho = \frac{\lambda G'(1)}{\mu}$ , equation (4) can be written as

$$E(N) = \frac{\rho(\sigma^2 - 1)}{2(1 - \rho)}$$

The Rayleigh distribution fits much better the data distribution of Figure 9. The parameter  $\sigma$  can be calculated using Maximum Likelihood Estimation (MLE) over the data sample. MLE estimation is straightforward using Matlab. Table 1 shows the MLE values of  $\sigma$  and the mean queue size for the 6 Half Pixel Planes in the current example,

Table 1

Half Pixel Plane	$\hat{\sigma}$	E(N)
N-1 bend	31.15	4.02
N-1 non bend	21.64	1.94
N bend	31.74	4.18
N non bend	23.50	2.29
N+1 bend	31.87	4.21
N+1 non bend	21.97	2.0

A 750 BCO simulation shows that the XYPC queue is empty half of the time and peeks suddenly every time a bulk fills it up. Since the bulk interdeparture time is fairly smaller than the bulk interarrival time, the queue shows to return to 0 most of the time. The BB33 input queue shows a similar behavior. Figures 15 and 16 show the simulation of the XYPC and BB33 input queues for plane N-1 non bend. The mean queue sizes are,

x-y input queue: E(N)= 2.26

BB33 input queue: E(N)=2.83

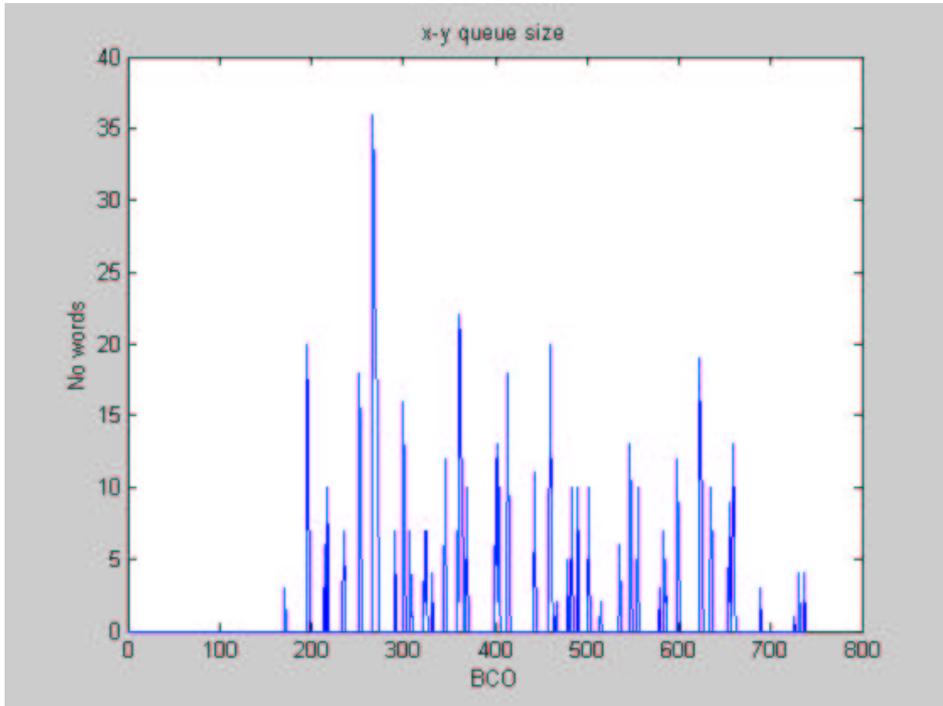


Figure 15 XYPC queue size

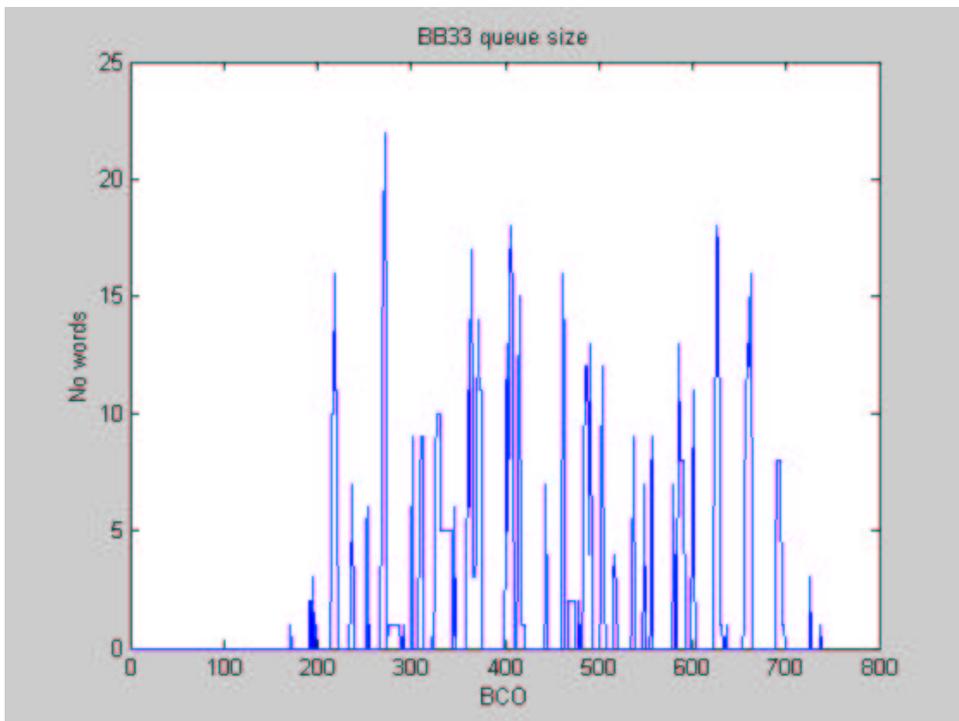


Figure 16 BB33 Input queue size

### 2.3 The Segment Tracker Architecture

As said in section XX, the Segment Tracker finds 3-station long inner and outer triplets. The current analysis is based on the proposed BB33 algorithm. A detailed description of the BB33 can be found in [Ref1] [Ref2]. The Segment Tracker receives input from 6 Half Planes corresponding to the bend and non-bend views of three consecutive stations in the Pixel Detector. There is a queue associated to each input to store the incoming data. We have, also, defined other 7 internal queues for temporary data storage, which allows pipelining through the processing modules.

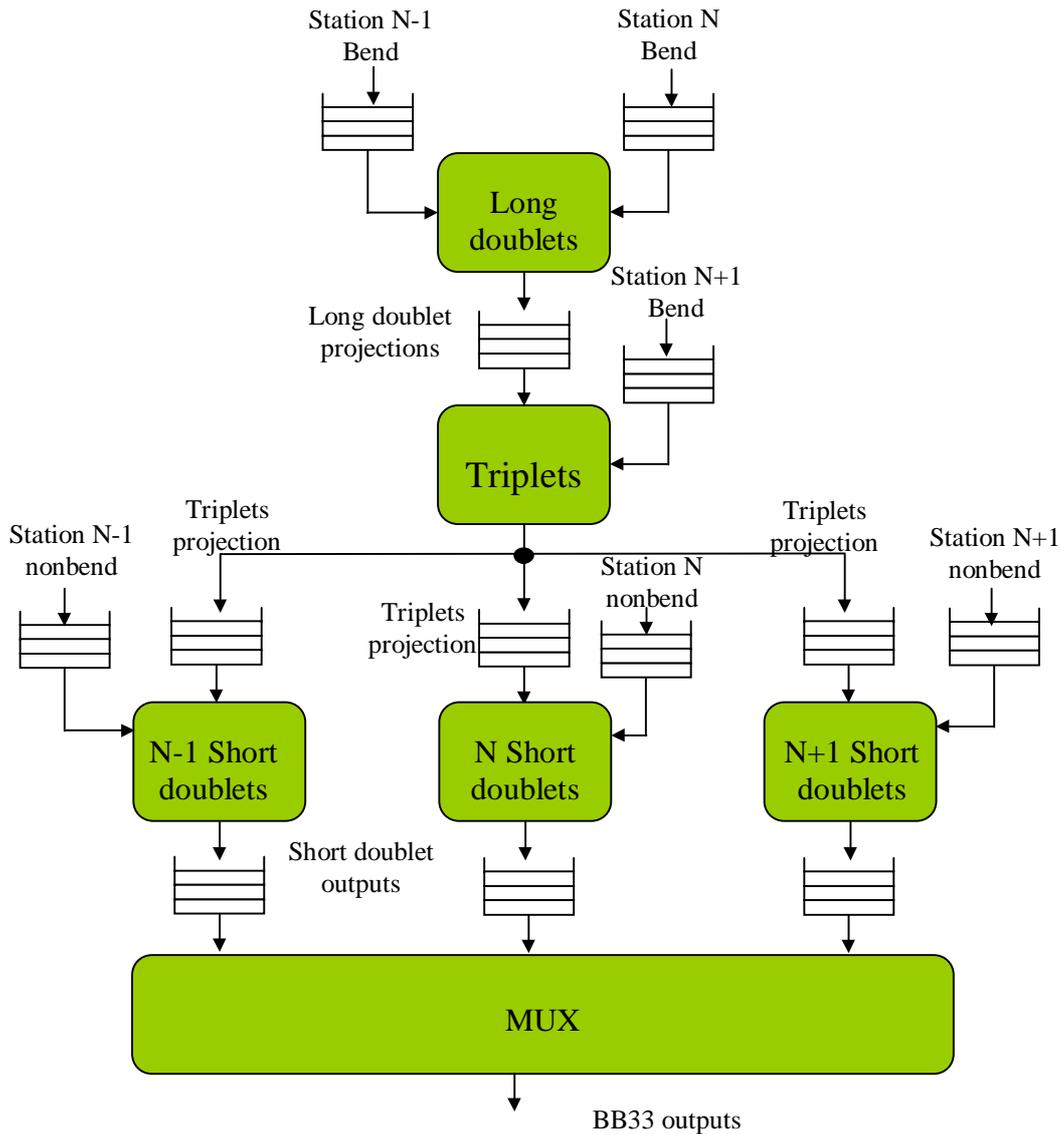


Figure 17: The Segment Tracker Architecture

Each of the first five modules in the BB33 algorithm process entire events of data coming from two sources. Here, we associate the word *event* with all the data generated by a particular section of the Pixel Detector (i.e one Half Plane) during one BCO time. As shown in Figure 17, the pixel hits preprocessed by the Pixel Preprocessor accumulate in the input queues of the BB33 processing modules. An event is processed when the buffer manager of a processing module detects that one event in each of the two input queues are complete. The buffer manager of each processing module synchronizes the data streams. The buffer managers are not explicitly shown in the block diagram above but are the first function in each

processing module. Each processing module produces pixel doublets and projections as results, which are used as input for the next processing module.

## 2.4 Analysis and simulations of the BB33 dataflow

### 2.4.1 Analysis of the BB33 queues as events from the buffer manager

The data flow of the BB33 algorithm can be analyzed in several ways. We can start with the simplest analysis, disregarding the individual pixel hits that accumulate in the input queues and only looking at the output of the buffer managers. As said, the buffer managers output a random sequence, which can be represented by a Poisson process. The buffer managers store data in the two input queues that they control, until they detect that a complete event is in the queue. At that time they issue a “complete event” primitive that is used by the processing module to start the event processing. This “complete event” sequence can be modeled by a Poisson process. The BB33 algorithm is seen as an open network of queues, where inputs are Poisson. The simulation shows that the 5 data mean arrival rates and mean processing times are as specified by the following table:

Table 2

Pixel Half Plane	Event arrival rate ( $\lambda_i$ ) (hits/clock)	Mean Processing Time (clocks)
Long Doublet	0.0089	39.34
Triplet	0. (not final results yet, need double checking from Erik)	0.
N-1 short doublet	0.	0.
N short doublet	0.	0.
N+1 short doublet	0.	0.

This means that mean number of queued events in the Long Doublet process is

$$E(R_q) = \frac{\rho}{1-\rho} = 0.538 \quad \text{where} \quad \rho = \frac{\lambda}{\mu} = 0.35$$

where  $\rho$  is the utilization factor.

We can estimate the average number of hits in the N-1 bend and N bend queues by multiplying the Average event size to the result above.

$$E(N_q) = E(E_v) * E(R_q) = 20.82 * 0.538 = 11.02$$

Figure 18 shows a simulation run of the queue sizes of N-1 bend and N bend queues

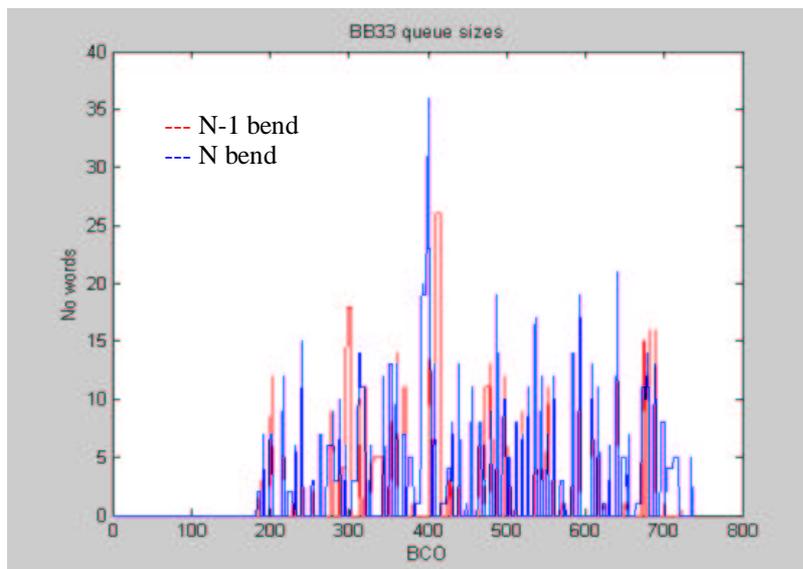


Figure 18 BB33 queue sizes

The average queue sizes out of the simulations are:

N-1 bend: 13.57

N bend: 11.64

These values are reasonable close to the ones calculated by using queueing analysis. I think a longer simulation will get them closer to the calculated value. Some discrepancy may come from the fact that the process is not 100% Poisson. The simulation shows that the distribution of event arrival times is not strictly exponential, and has a bias.

Note that during the first 159 BCO the queues are empty. This is caused by the transitory in the TS-ordering queues, which has a deterministic delay of 159 BCOs.

The analysis of the other queues is fairly similar. Figure 19 shows all the queue sizes

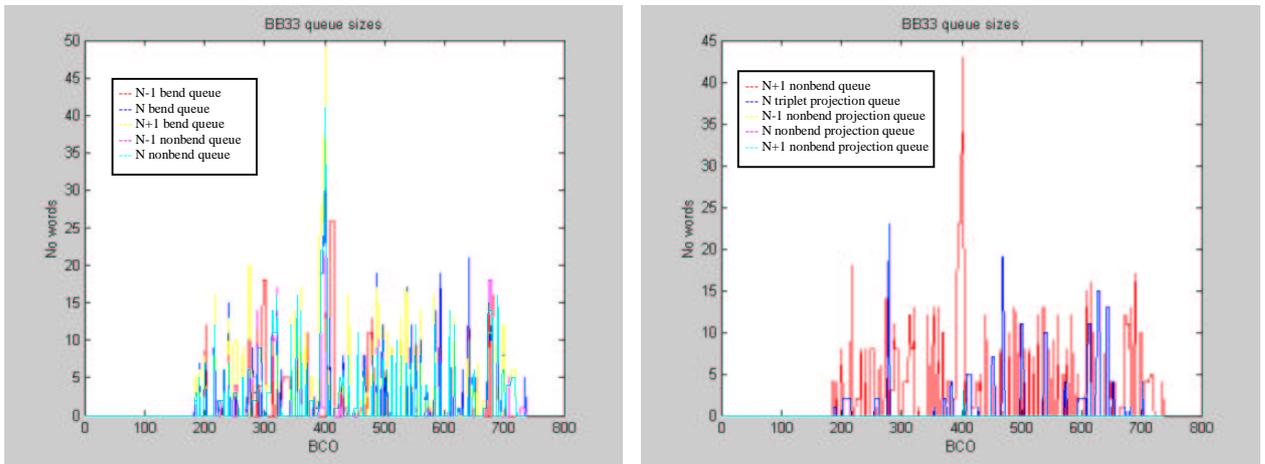


Figure 19: BB33 queue sizes

The average queue sizes are summarized in the following table

Table 3

Queue size Mean and $\sigma$	Calculated		Simulated	
	Mean	$\sigma$	Mean	$\sigma$
N-1 bend	11.02	?	13.5796	8.8559
N-1 non bend	?	?	11.6466	9.0579
N bend	11.75	?	15.9897	8.0108
N non bend	?	?	12.1828	9.2420
N+1 bend	?	?	14.8655	9.6172
N+1 non bend	?	?	12.9621	9.9339
N triplet projection bend	?	?	2.4667	5.0369
N-1 projection non bend	?	?	0.1315	0.6177
N projection non bend	?	?	0.0259	0.2845
N+1 projection non bend	?	?	0.0259	0.2845

The BB33 input queues during this simulation run show a high peak at about BCO 400. This is caused by 3 consecutive large events of about 25 tracks each. Since the utilization factor of the modules is low the Segment Tracker recovers very quickly.

Four of the BB33 processing modules (i.e. the triplet and the 3 short doublet processors) perform a very similar task to the Long Doublet processor. The main difference is that in each one of these four processing modules, one of the queues is the output of a previous processing module in the BB33 algorithm. For

instance the Triplet processor process data from two queues, the input of one of them is the output of the Long Doublet processor. So we need to find out the pdf of this input. We can extend the analysis to the Short Doublets as well. For that, we can take the processing modules in pairs. Each pair can be seen as a network of queues as shown in Figure 20.

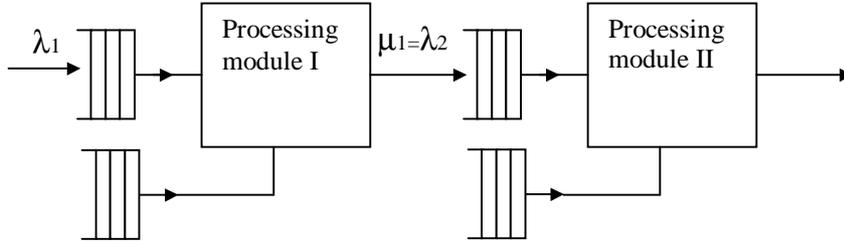


Figure 20 Network of queues

If the arrival and service distribution functions of the Processing module I are exponentially distributed with parameters  $\lambda_1$  and  $\mu_1$  respectively, it can be shown using the Laplace transform that the departure is exponentially distributed with parameter  $\lambda_1$ . This means that the input to the Processing module II is, also, Poisson distributed with parameter  $\lambda_1$ . This property can be applied to the Short Doublets as well. This is the justification to why the input to the Triplet and Short Doublet processes can be seen as Poisson. If the service distribution functions of those processes are exponential, then the queues are M/M/1 queues.

2.4.2 Analysis of the BB33 queues as a “bulk” service process

A more detailed analysis of the BB33 queues must look at the full dynamics of the number of hits in the queues. The processes can be modeled as “bulk” service. In the “bulk” service process the input queue receives single arrivals but allows “bulks” of variable size in the departure. The state transition diagram of the “bulk” service process only allows “birth” type of transitions to a neighbor state on the right. However, the “death” transitions (i.e. right to left) can be to non-neighboring states. As in the “bulk” arrival case we must define

$$g_i = \text{Prob}[\text{bulk size is } i], \text{ then } \sum_{i=1}^{\infty} g_i = 1$$

$g_i$  has a “modulation” effect over the distribution of the queue size  $p_i$ .

The equilibrium equations for the bulk arrival system are:

$$(\lambda + \mu) p_k = \lambda p_{k-1} + \mu \sum_{i=k+1}^{\infty} p_i g_{i-k} \quad k > 1 \quad (*)$$

$$\lambda p_0 = \mu \sum_{i=1}^{\infty} p_i g_i$$

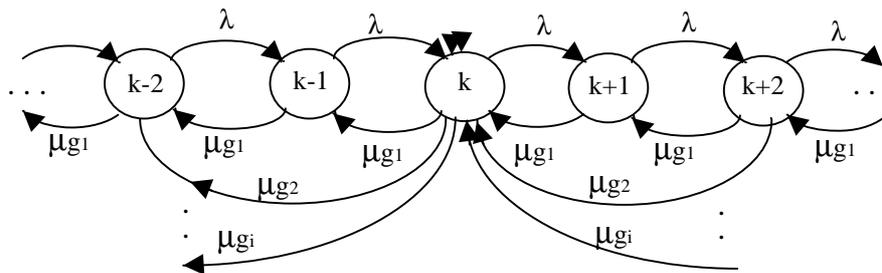


Figure 21 Bulk service queuing model

Using the Z Transform

$$P(Z) = \sum_{i=1}^{\infty} p_i Z^k$$

equation (\*) becomes

$$(\lambda + \mu)(P(Z) - P_o) = Z\lambda P(Z) + \mu \sum_{k=1}^{\infty} \sum_{i=k+1}^{\infty} p_i g_{i-k} Z^k \quad k > 1 \quad (**)$$

the last term of equation (\*\*) is a double summation. If we change variables in the inner summation

$$\mu \sum_{k=1}^{\infty} \sum_{j=1}^{\infty} p_{j+k} g_j Z^k \quad \text{where } j = i - k$$

We can work it out swapping the summations and momentarily fixing j. Then, this last term becomes

$$\mu \sum_{k=1}^{\infty} \sum_{j=1}^{\infty} p_{j+k} g_j Z^k = \frac{\mu}{Z} g_k \left[ P(z) - \sum_{k=0}^j p_k Z^k \right] \quad \text{where } j = \text{fixed} \quad (***)$$

combining (\*\*) and (\*\*\*) the equilibrium equations become

$$(\lambda + \mu)(P(Z) - P_o) = Z\lambda P(Z) + \mu \sum_{k=1}^{\infty} \sum_{i=k+1}^{\infty} p_i g_{i-k} Z^k \quad k > 1$$

$$(\lambda + \mu)(P(Z) - P_o) = Z\lambda P(Z) + \frac{\mu}{Z^k} g_k \left[ P(Z) - \sum_{k=0}^j p_k Z^k \right] \quad j = \text{fixed}$$

Equation (\*\*\*\*) can be solved for j=fixed by becomes analytically untractable if we try to solve for all j. j represents the size of the bulk departing from state k after processing. A good estimation can be achieved using the average bulk size and solving for a fixed j=Avg bulk size.

The solution to that is shown in the Appendix II. The final equation is,

$$P(Z) = \frac{1 - 1/Z_o}{1 - Z/Z_o}$$

We can obtain the distribution inverting the last equation

$$p_k = \left( 1 - \frac{1}{Z_o} \right) \left( \frac{1}{Z_o} \right)^k, \quad \text{where } Z_o \text{ is obtained from equation (****).}$$

$p_k$  is geometrically distributed. Its mean value is

$$E(N) = Z_o \left( \frac{1}{1 - \frac{1}{Z_o}} \right) = \frac{Z_o^2}{Z_o - 1}$$

---

The simulations show that the 6 data individual data streams are Poisson processes with rates as specified by the following table

Table 4

Pixel Half Plane	Queue arrival rate ( $\lambda_i$ ) (hits/clock)
N-1 non bend	0.1072
N-1 bend	0.1014
N non bend	0.1116
N bend	0.1014
N+1 non bend	0.1133
N+1 bend	0.1023

The superposition of two independent Poisson processes is also a Poisson process with arrival rate equal to the sum of the individual input rates. The combined arrival rate for stations N-1 bend and N bend at the input of Long Doublet processing module is 0.2028 hits/clock.

(write results here)

### 2.4.3 Latency and Processing Times

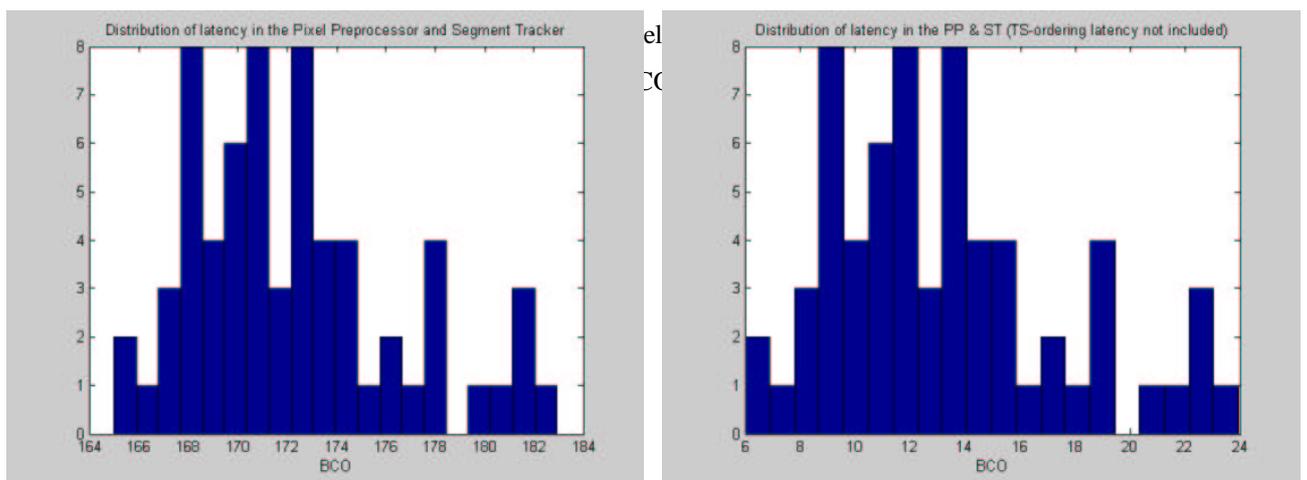
In this section we analyze Processing Times in each stage of the Pixel Preprocessor and Segment Tracker. The Processing Time of a piece of data or Service Distribution Time of a Processing module impacts the queue sizes. The

Table 5

Pixel Preprocessor			
	Processing Distribution	Average Processing Time	
Receiver interface	deterministic	2 clock/hit	
TS-ordering (queue)	deterministic	159 BCOs (4452 clocks!)	
TS-ordering (hits)	exponential	18.8 clocks/hit	
X-Y translation & grouping	Rayleigh	$1 + \text{Nohits}/\text{group} = 3.2 \text{ clk/hit}$	
Segment Tracker			
	Processing Distribution	Average Processing Time	
Long Doublet	Rayleigh: $1\text{clk} + \text{AvgNoQuery}/\text{BCO} * (2 + \text{AvgNomatches}/\text{query})$	29.17 clocks/event	
Triplet	Rayleigh: $1\text{clk} + \text{AvgNoQuery}/\text{BCO} * (2 + \text{AvgNomatches}/\text{query})$		
Short Doublets	Rayleigh: $1\text{clk} + \text{AvgNoQuery}/\text{BCO} * (2 + \text{AvgNomatches}/\text{query})$		

We define latency in a specific stage of the Pixel Preprocessor and Segment Tracker modules as the time between the arrival of the first hit of an event and the time when the event departs from that stage.

The latency in the Pixel Preprocessor and the Segment Tracker is dominated by the sorting time in the TS-ordering queues (i.e 159 BCOs). Figure 22a shows the latency distribution. ). Figure 22b is the same figure but suppressing the latency in the TS-ordering queues.



APENDIX I

(pixel front end model)

APENDIX II

(derivation of the “bulk” service equations)