Timing measurements of some tracking algorithms and suitability of FPGA's to improve the execution speed

C. Hinkelbein, A.Khomich, A. Kugel, R. Männer, M. Müller
University of Mannheim, Germany

J. Baines
RAL, Chilton, Didcot, United Kingdom

ATL-DAQ-2003-026
12 September 2003

Abstract

Some of track reconstruction algorithms which are common to all *B*-physics channels and standard RoI processing have been tested for execution time and assessed for suitability for speed-up by using FPGA coprocessor. The studies presented in this note were performed in the C/C++ framework, CTrig, which was the fullest set of algorithms available at the time of study

For investigation of possible speed-up of algorithms most time consuming parts of TRT-LUT was implemented in VHDL for running in FPGA coprocessor board MPRACE. MPRACE (Reconfigurable Accelerator / Computing Engine) is an FPGA-Coprocessor based on Xilinx Virtex-2 FPGA and made as 64Bit/66MHz PCI card developed at the University of Mannheim. Timing measurements results for a TRT Full Scan algorithm executed on the MPRACE are presented here as well. The measurement results show a speed-up factor of ~2 for this algorithm.

1 Introduction

The ATLAS trigger strategy foresees a reduction of the event rate at three levels: LVL1, LVL2 and Event Filter (EF) [1]. The first level trigger (LVL1) uses information from the calorimeter and muon systems to reject uninteresting events and to identify Regions of Interest (RoI) within potentially interesting events. At LVL2 each RoI is examined in the detector system from which it originated, *i.e.* in the muon or calorimeter system, to see if it is confirmed as a valid object. After the confirmation of the LVL1 RoI, additional features associated with it may be searched for in other detectors, such as the SCT/Pixel and TRT.

Processing of *B*-hadron events is different from standard RoI processing [12]. *B*-hadron events are triggered by a low-*p*T single muon at LVL1. This muon is confirmed at LVL2, firstly in the muon detector and then in association with a track from Inner Detector (ID). The track search can be initiated from the outside of the ID using TRT information or from the inside using Pixel data. A full-scan is performed to search the entire volume of the TRT or Pixel Detector respectively. The result of the full-scan is a set of tracks which are used as track seeds for a Kalman filter (or Hough transform) algorithm in the semiconductor tracker (SCT).

Some of inner detector track reconstruction algorithms which are common to all *B*-physics channels and standard RoI processing have been tested for execution time and assessed for suitability for speed-up by using FPGA coprocessor.

The FPGA coprocessor – MPRACE (Reconfigurable Accelerator / Computing Engine) developed at the University of Mannheim can be used for analysis of a capability and benefit of the using of FPGA for speed-up of pattern recognition tasks.

MPRACE is an FPGA-Coprocessor based on Xilinx Virtex-2 FPGA and made as 64Bit/66MHz PCI card. The main features of this board are:

- Dedicated PCI-to-LocalBus bridge PLX9656. 64Bit/66MHz on PCI, 32Bit/66MHz on local bus.
- High capacity FPGA, Xilinx Virtex-2. First version uses XC2V3000-4BF957C (14336 slices (= 28672 LUTs/FFs), 12 DCMs, 96 Multipliers, 96 RAM blocks @18kBit (total 1.728Mbit), 684 I/Os)
- High bandwidth memory system: 4 banks ZBT, each 512k*36 bits (8 Mbytes in total).

Usage of the FPGA coprocessor can give some reasonable speedup as contrasted to general purpose processor only for those algorithms (or parts of algorithms), for which one there is a possibility to fulfil calculations with a major degree of parallelism.

Measurements presented in this note were performed in the C/C++ framework, CTrig (CTrig-01-15-12) [7], which was the fullest set of algorithms available at the time of study, running on computer equipped with dual Intel Xeon 2.4 GHz, 64bit/66MHz PCI bus, 1 Gb DDR RAM main memories with CERN Red Hat 7.1. All measurements were made using datasets Y00347 (file 1: Y00347_1.atrdmp.data) containing approx. 156 events ($B \rightarrow \mu X$) with pile-up at luminosity 10^33.

2. Detector geometry

The geometrical layout of the Inner Detector used for this work is described below. More details can be found in Inner Detector TDR [5].

The pixel detector provides 3-4 measurements along a track, and the silicon micro-strip detectors 4-5 measured points (3 in a small transition region from barrel to end-cap geometry). Individual detectors are arranged in concentric cylinders in the barrel, and in disks in the end-cap regions. There are overlaps between adjacent detectors in z (barrel) or R (end-cap) and in R-φ. It is therefore possible for a single track to give two measured points in a single plane in small regions of the detector.

The inner “B-layer” of the pixel detector is at a radius of 4 cm and provides coverage over the entire pseudo-rapidity range $|\eta|<2.5$. At least two additional measurements on a track are provided by barrel layers at radii of 11 cm and 14.2 cm and by disks, of which there are four in each end-cap in the ID-TDR design. The individual sensitive element in the barrel (end-cap) is a pixel 50 mm in R-φ and 300 mm in z (R).

The Semi-Conductor Tracker (SCT) is composed of four layers of modules in the barrel and nine discs in each end-cap. The transition region from barrel to end-cap geometry occurs in the pseudo-rapidity range $1.16<|\eta|<1.64$. An SCT module consists of two back-to-back detectors, one with strips parallel to the beam-pipe in the barrel and radial in the end-caps, the other with strips rotated by ±40 mRad to give a stereo measurement.

The Transition Radiation Tracker (TRT) consists of a central barrel part and two end-cap sections. The sensitive elements are straws of internal diameter 4 mm with a single sense wire running down the centre. The barrel comprises 52544 straws parallel to the beam axis in 73 layers; the end-caps comprise 319488 radial straws in 18 multi-plane wheels. The barrel has an inner radius of 56 cm and outer radius of 107 cm. However, for the inner layers up to a radius of 62 cm, the active part of the straw is limited to $|z|>40$cm. Outside this radius the straws have an electrical break at z=0. In order to reduce the occupancy and improve the measurement, each half of the barrel is read out separately.

The TRT straws provide a two-dimensional position measurement, r-φ in the barrel and z-φ in the end-caps. Typically the TRT provides 36 measurements on every track in the entire covered η range.

In the mechanical layout for the detector, the barrel consists of three concentric cylindrical rings. Each ring is formed from 32 independent identical mechanical modules. The modules are not projective and have a different shape from ring to ring [6]. Layers of straws are grouped into three types of modules. In each layer, straws are at approximately constant distance (about 6.8 mm), the $\Delta\varphi$ covered by each module is the same ($2\pi/32$), and the number of straws per layer in each module varies from layer to layer (from 15 to 29 straws per layer). The total number of straws is 1642 per 3 modules, or 52544×2 for the two halves of the detector.

In the simulation of the detector the geometry is different for the barrel. In the simulation the straws in each layer are arranged in a concentric cylinder. Within a layer all straws have equal distance in φ. For all layers the straw distance is approximately rφ = 6.8 mm. From layer to layer the first straws have small φ offsets. The distance between two layers is 6.8 mm, there are 75 layers, 2 of which are empty [5]. This geometry, used for simulation and reconstruction, does not contain any symmetry.

3 Algorithms

This section briefly describes some of the track search algorithms and presents execution time measurements results. The trigger algorithms and their performance in terms of efficiency are described more fully elsewhere. We try to analyse which of these algorithms can get some benefits from FPGA coprocessor as well. Algorithms and their performance are described more fully elsewhere.

3-1 Pixel-scan

"Pixel-scan" algorithm [8] uses only information from the Pixel Detector. The input data are the pixel cluster positions produced by a separate data preparation step [9]. In the first step of the full-scan algorithm, clusters are combined in pairs. All combinations are tested of one cluster per layer in the inner two planes of the barrel and in two of the end-cap planes. A pair is retained if the trajectory extrapolation inwards passes within a given distance of the interaction point. A track is formed if a hit is found close to the extrapolated track into the third layer barrel layer or in one of the outer pixel end-cap layers. In both cases the extrapolation is linear.

Therefore the track finding proceeds independently for every triplet of layers. For every triplet lines connecting every couple of points in layer 1 and layer 2 (so called "links") are created. This part of the algorithm may be done in parallel. For example:

Step 1: Store in coprocessor memory all points from layer 1

Step 2: Transfer into the coprocessor points from layer 2 one by one and calculate "links" parameters for this point and every point from layer 1 in parallel.

But for calculating only 1 "link" we need 6 floating point numbers (3 coordinates for every point in "link") and 3 of them should be read from memory. If 32 bit floating point numbers are used we should read 96 bits of data from memory for calculation only 1 "link" even without any parallelism. According to [8] we have ~50-75 clusters per layer and for calculating all links for 1 point ~150-225 floating point numbers should be read from memory.

Therefore even easily parallelized algorithms (or their parts) are not good candidates for an FPGA implementation if floating point calculations are in use.

But one could work with a reduced precision instead of 32 bit floats ([13] is one example of using floating point arithmetic on FPGA) or we can use fixed point calculations. In this case, if we will

use for example MPRACE board with XILINX Virtex-2 FPGA, we have 96 RAM blocks @18kBit were can store at least 96 numbers. Therefore we can calculate 96 “links” in parallel.

The execution time measurements results for this algorithm shown in table 1.

Algorithm part	**Execution time** (ms) p_T>0.5
Prepare cluster table	52.669
Build tracks	4.775
Solve ambiguities	3.451
Create structure	0.076
Delete list of clusters	2.243
Total	**60.975**

Table 1: Execution times on a dual Xeon 2.4GHz PC for the Pixel Scan. (The total time is not the sum of the average times shown in the table, but rather is the average of the total times per event.)

3-2 Precision tracker data preparation

Precision tracker data preparation [9] algorithm is divided into four different steps: data pre-selection, clustering, space-point formation and post-selection (optional). This set of algorithms process the raw data from the SCT and pixel detectors into a format suitable for the precision tracker LVL2 Feature EXtraction (FEX) stage. The raw data consists of hits in the format produced by the Read Out Drivers (RODs). Data preparation algorithms may be implemented in the ROD or in the LVL2 processor, EF processor or in processors dedicated to data preparation (e.g. FPGA). All this algorithms a fast enough and usage of FPGA coprocessor can give some reasonable speed up only if coprocessor will be on the raw data path between ROD and LVL2 processor i.e. if raw data from ROD first came to the coprocessor and then, after preparation, come to the LVL2 processor. Otherwise time for data transfer from the LVL2 processors memory to coprocessor can remove possible speed up.

3-3 IDSCAN

IDSCAN algorithm [10] consists of four steps: z-finder, Hit filter, Group cleaner and Track fitting.

During the first step the z position of the event is found. For this the RoI is divided into many φ bins. In each φ bin, all pairs of hits from different layers are made. For each pair the z position of the event is found by linear extrapolation and fill a 1D-histogram. After finding the z of the physics event, a 2D-histogram in (η,φ) is made. Each bin in that histogram corresponds to a small solid angle. A track (above certain p_T) from the physics event will be fully contained in one such bin, while a pile-up track from a different z will cross many bins. Therefore, each bin counts how many different layers have been hit. If bin higher than threshold - accept all hits in this bin, else reject all hits in this bin. This step put hits from neighbouring bins into groups and returns these groups as result. Next step rejects garbage groups and garbage hits in a group and splits groups that contain many tracks to individual track candidates. For this a 2D-histogram in (φ_0, $1/p_T$) is filled with hit triplets. It is a bit like a Hough transform but for one triplet only one bin is incremented. All three steps have histogramming phase but in every case only one histogram bin is incremented for one input value. Therefore something like pipelining technique should be used for these steps. As far as data transfer from host memory to coprocessor board can be fast enough (DMA transfer for example), one can transfer several input values to board and run bin incrementing for this input values in parallel. But on the fast host PC (fast processor with big amount of cash memory and fast main memory) transferring can take more time then histogramming on the host processor.

The execution time measurements results for this algorithm shown in table 2.

Algorithm part	**Execution time** (ms)
ZFinder	2.072
Hit Filter	9.693
Kalman Filter	0.725
Last Loop	0.742
Total	**13.234**

Table 2: Execution times on a dual Xeon 2.4GHz PC for the IDScan. (The total time is not the sum of the average times shown in the table, but rather is the average of the total times per event.)

3-4 SctKalman

This FEX is based on a Kalman filter algorithm [14]. A track search may be initiated by either an external or internal seed. For the purposes of the *B*-physics trigger, the Si-Kalman FEX uses either pixel or TRT tracks as external seeds in pixel-seeded or TRT-seeded mode respectively. The track segment is extended by adding nearby hits layer by layer. The algorithm takes into account multiple scattering. More details can be found in [15].

The algorithm starts from a small track segment or from a fitted track of a neighboring detector, and then extends the candidate tracks by adding measured points one by one. The fitted parameters and weight matrix of the candidate track are updated when adding a point, and give an increasing precision on prediction of the next point. Thus, pattern recognition and track fitting can be accomplished simultaneously.

Therefore it is consecutive algorithm and it can not benefit from FPGA coprocessor.

The execution time measurements results for this algorithm shown in tables 3-5.

Algorithm part	**Execution time** (ms) p_T>0.5
Data preparation	0.495
SctKalman FEX – track finding	11.187
Fill output track structures	3.603
Total	**15.286**

Table 3: Execution times on a dual Xeon 2.4GHz PC for the pixel-seeded SctKalman. (The total time is not the sum of the average times shown in the table, but rather is the average of the total times per event.)

Algorithm part	**Execution time** (ms) p_T>0.5
Data preparation	3.510
SctKalman FEX – track finding	79.447
Fill output track structures	14.439
Total	**97.396**

Table 4: Execution times on a dual Xeon 2.4GHz PC for the TRT-LUT-seeded SctKalman. (The total time is not the sum of the average times shown in the table, but rather is the average of the total times per event.)

Algorithm part	**Execution time** (ms) p_T>0.5
Data preparation	1.948
SctKalman FEX – track finding	53.348
Fill output track structures	9.128
Total	**64.423**

Table 5: Execution times on a dual Xeon 2.4GHz PC for the TrtKalman-seeded SctKalman. (The total time is not the sum of the average times shown in the table, but rather is the average of the total times per event.)

3-5 SiTree

This algorithm associates locally related hits in the SCT using “link-and tree” method [17][18].
The basic element used in this algorithm is not a single space point, but a "link". The link is a segment of a potential track with enough hits to determine the track parameters of the segment. If the tracks being sought are straight lines or circles coming from the interaction point, only two hits are required to define a link. If the tracks are circles originating at arbitrary points, at least three hits are necessary to define a link since three points define a circle. In the current implementations, pixel-guided FEX and TRT-guided FEX with tree algorithm, only the two hit link is used assuming the origin of the track is very close to the interaction point.

This looks very similar to Pixel-scan algorithm from FPGA coprocessor point of view (see 3-1) and same comments can be applied. We can build “links” in parallel, but for these we need floating point arithmetic. We can work with a reduced precision or with fixed point numbers.

The execution time measurements results for this algorithm shown in table 6.

Algorithm part	**Execution time** (ms) p_T>0.5
Data preparation	0.409
SiTree FEX – track finding	9.241
Fill output track structures	0.049
Total	**9.699**

Table 6: Execution times on a dual Xeon 2.4GHz PC for the pixel-seeded SiTree. (The total time is not the sum of the average times shown in the table, but rather is the average of the total times per event.)

3-6 TrtKalman

The FEX algorithm [16] consists of four stages:

- An initial track search using a histogramming method
- Fine tuning
- Track fitting
- Track candidate selection.

The initial search looks for hits on a straight line in the appropriate projection. The trajectory of a charged particle is linear in the z - φ plane and approximates to a straight line in the R - φ plane for particles with high p_T produced close to the origin. The trajectory can thus be described as:

$$\varphi \approx \varphi_0 + C_t \bullet R \text{ (Barrel)}$$

$$\varphi = \varphi'_0 + C_z \bullet R \text{ (End-Cap)}$$

where $\varphi'_0 = \varphi_0 - z_0 \bullet C_Z$, $C_t \approx 0.003Bq/p_T$ $C_Z = C_t \tan\theta$ and θ is polar angle of the track. Thus all points on track lie on a line characterized by slope C and intercept φ_0. For each hit in a detector, within the RoI, the value of φ_0 is calculated for each value of slope between $-C_{max}$ and $+C_{max}$. The value of C_{max} is determined by the lowest p_T track that is to be sought. Each hit will populate many cells of the histogram, but for each track in the RoI there will be a bin where all hits on the track have an entry (provided an appropriate bin size has been chosen). Thus the track candidates can be identified from peaks in the histogram. If the RoI contains more than one part of the TRT (-z end-cap, -z barrel, +z barrel and +z end-cap), the initial search is performed independently in the two parts. The bin size in φ_0 is not fixed but is determined dynamically for each RoI by the straws with hits. The low-φ and high-φ boundaries of all the straws with hits in the RoI, sorted in order of increasing φ, determine the division of histogram. For speed, some quantities are calculated in an

initialisation phase and stored in look-up tables. For each bin with more than 8 (this is a parameter) hits (out of a maximum of typically 40 straws), the procedure continues to the fine tuning stage.

The execution time measurements results for this algorithm shown in table 7.

Algorithm part	**Execution time** (ms) p_T>0.5
Detector Initialization	49.964
Space point production	3.359
Histogramming	92.273
Fitting	24.134
Total	**9.699**

Table 7: Execution times on a dual Xeon 2.4GHz PC for the TRTKalman. (The total time is not the sum of the average times shown in the table, but rather is the average of the total times per event.)

An initial track search which utilizes a Hough transform with look-up table (most time consuming part of this algorithm) can be done in parallel inside FPGA like it is implemented for TRT-LUT algorithm.

3-7 SCT-Hough and TRT-LUT

SCT-Hough [11] and TRT-LUT [2] are using similar approach to track finding. Algorithms consist of a track candidate search followed by track-fits performed to select the best candidate and to determine the track parameters. The purpose of the candidate search is to find sets of points that lie within roads thereby reducing the number of point combinations to be investigated at the track fitting stage. A histogramming method based on a Hough transform is used. The roads correspond to bins of the histogram in transform space. This method can be easily parallelized and it is a good candidate for FPGA implementation. TRT scan will be described in more details below. SCT-Hough can be implemented by the same way.

Since all particle trajectories to search for can be calculated in advance a histogramming method based on the Hough Transform is well suited for the initial track search in TRT [2]. The Hough Transform is a standard method in image analysis that allows recognition of global patterns in an image space by recognition of local patterns in a transformed parameter space. It is based on the idea that every hit in the three-dimensional detector image can belong to a number of possible (predefined) tracks characterized by different parameters. All such tracks (or roads) are stored in Look-Up table (LUT). Thus every hit increases the “probability” for the existence of these tracks by 1 (histogramming). The histogram for a single track consists of a “bow-tie” shaped region of bins with entries with a peak at the centre of the region. The bin at the peak of the histogram will, in the ideal case, contain all the hits from the track. The roads corresponding to the other filled bins share straws with the peak bin, and so contain sub-sets of the hits from the track. The histogram for a more complex event consists of a superposition of the entries from the individual tracks. The bins containing the complete set of points from each track can be identified as local maxima in the histogram. After a clean-up step followed by a fit the final tracks are selected.

LUT based Hough Transform algorithm for TRT was implemented in C++. It was integrated and investigated in ATLAS Level-2 reference software [2]. This algorithm was implemented in VHDL for MPRACE board as well. This implementation was integrated in to ATLAS Level-2 reference software for performance study and comparison with C++ implementation.

The entire algorithm as included in the ATLAS Level-2 reference software consists of the following steps:

Initial Track Finding: Utilizes an LUT-based Hough Transform using histogramming to find potential track candidates. In the barrel, the Hough Transform performed is from (R,φ) space to (φ,1/p_T) space. The LUT consist of 81920 (φ × 1/p_T = 1024 × 80) predefined roads. All predefined roads point to the origin. The assumption of straight lines in the R-φ projection is not sufficiently accurate for low-p_T tracks in magnetic field. Therefore predefined overlapping roads are computed as exact circles in the x-y projection. The road width increases linearly from 4.5 mm at layer 1 (numbering from the innermost layer outwards) to 6.8 mm at layer 42 and is then constant and equal to 6.8 mm from layer 42 to layer 73. With this definition, ~65 straws are assigned to each road. The predefined roads overlap by 30% - 50% in 1 p_T and φ. This overlapping of the roads prevents the loss of hits from a track with a trajectory which could otherwise pass between two predefined roads, but can lead to multiply reconstructed tracks, which have to be eliminated in subsequent steps. Each straw is assigned to ~120 roads (max. 130).

Threshold Application and Local Maximum Finding: Selects potential track candidates and eliminates multiple reconstructed tracks.

Track Splitting: Removes hits incorrectly assigned to a track, and splits tracks that have been erroneously merged.

Track Fitting and Final Selection: Performs a fit in the r-ϕ (barrel) or z-ϕ (end caps) plane using a third order polynomial to improve the measurement of ϕ and p_T . The algorithm uses only the straw position (i.e. the drift time information is not used). The final track candidates are selected during this step.

Profiling of a C++ implementation of the TRT full scan algorithm shows that most of the computing time is spent with access of LUTs, incrementing of 8-bit numbers, and a non-linear 2-D filter operation (local maximum finding) [3]. CPU-only implementation of Histogramming (or Initial Track Finding) and Thresholding/Local Maximum Finding require 89% of the total processing time. These two steps are good candidates for an FPGA implementation because of both steps can exploit an 80-fold parallelism (80 $1/p_T$ blocks) and make use of the fast internal dual port block RAM. The main difference between an FPGA implementation and a CPU implementation of the Hough transformation is that the loop over all predefined roads for one straw (one row in LUT), which is executed once per active straw and increments ~130 histogram counters, is executed sequentially in the CPU and in parallel in the FPGA.

A schematic view of the VHDL implementation of the initial track finding step of the algorithm is shown in Figure 1. This implementation takes advantage of both the external SRAM and the entire available internal RAM blocks.

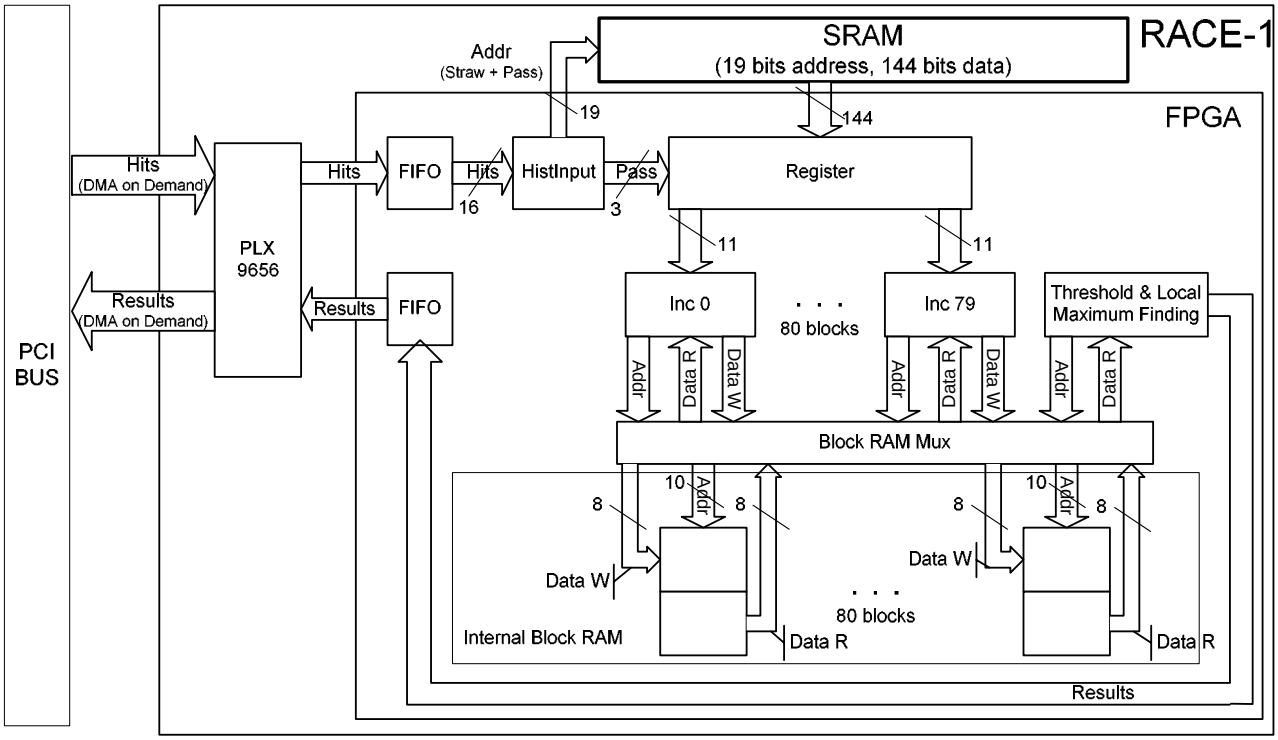

Figure 1: FPGA initial track finding

The initial track finding works as follows.
The array of active straws (hits) prepared in host memory and transferred over PCI bus to MPRACE by DMA. For each hit straw, counters are incremented for all roads containing that straw. The histogram counters for all TRT straws which have to be incremented are stored in the “straw-ordered” LUT. The construction of the LUT with overlapping roads in φ and $1/p_T$ guarantees that each active straw contributes to exactly one or two pre-defined φ values of a certain pre-defined $1/p_T$. For the barrel, there are 80 '$1/p_T$ -blocks' with 1024 possible φ each, thus 81920 patterns are pre-defined. Therefore the 1-2 φ values out of 1024 possible values, which have to be incremented, are stored in 11 bits (10+1), because the (possibly) 2 values are always directly neighboured.

The “straw-ordered” LUT is stored in the SRAM. An address space of 16 bits for the straws in the "straw-ordered" LUT is required (if symmetry of the detector not in use). The SRAM itself has 19 bits addresses, of which 16 are used to identify the current straw. The LUT stores for each straw address 880 bits – histogram counters addresses (80×11 bits). The word length of the SRAM is only 144 bits; therefore seven steps (passes) with the same straw address (16 bits) and changing pass addresses (3 bits) are needed for transfer from SRAM (LUT) all information corresponding to one straw to FPGA. Using of the detector symmetry can significantly reduce requirements for size and word length of the memory used for LUT. The histogram is stored in the internal RAM blocks in the FPGA. The implementation profits from a true Dual Port RAM (internal RAM Blocks) to allow a fast read-modify-write cycle during one external SRAM cycle. Therefore histogramming for each straw is executed in seven passes with up to three external clock cycles each, whereas a CPU implementation requires looping sequentially over 130 histogram bins.

After the histogramming step the maximum finding step first applies a threshold filter and then a local maximum finding. The maximum finding has to be applied in two dimensions (φ and $1/p_T$). This is done in parallel for 80 $1/p_T$ - blocks and sequentially for 1024 φ - blocks. Resulting are the histogram counters with values above the threshold and which are local maximum in both φ and $1/p_T$. The output information contains the pre-defined φ and $1/p_T$ values of the track (pre-defined from the LUT) and the number of active straws corresponding to the track. These results stored in output FIFO and transferred over PCI bus to host memory by DMA.

If the track splitting step is also done in the FPGA, the results of the local maximum finding step are stored internally in the FPGA for the following track splitting.

In the track splitting step the pattern of hits associated to a track candidate is analysed. If potential track candidate contains ≥9 consecutive layers without a hit, the track is split into two separate candidates either side of the gap. If one of the candidates contains more than N_{thr} (14) hits, it is retained. If both candidates pass this threshold, the track segment which starts at the lowest radius is retained. The result of the track splitting step is a candidate that consists of a sub-set of the straws in a road. For this step "bin-ordered" LUT is constructed (each bin correspond to road). The list of straws lying within the road is stored in the LUT.

A schematic view of the FPGA implementation of the track splitting step looks similar to initial track finding and is shown in Figure 2.

The track splitting step uses the 75 internal RAM blocks (each block for one layer/plane) for a hash-table, which stores for each straw in this layer/plane if it is active or not ("active straws" hash-table), and the threshold information (and optionally the drift-time). The SRAM is used to address the bin-ordered LUT with histogram bin numbers above the threshold which were a local maximum. The LUT outputs all straws which belong to the pattern definition layer by layer. The bin-ordered LUT stored 825 bits (75 layers × 11 bits for straw number in layer). The word length of the SRAM is 144 bits; therefore six steps (passes) with the same bin address and changing pass addresses are needed for transfer from SRAM (LUT) all information corresponding to one pattern to FPGA.

Straw numbers from LUT are used as address for "active straws" hash-table. We have two 74 bit words from this hash-table: one of them give us information about active straws, second – about high threshold hits. This information is used for perform track splitting and the result is a 28 bit word which contain number of hits, number of high threshold hits, fist hit layer and last hit layer for track candidate. This result is passed to the host CPU over PCI bus by DMA for the track fit.

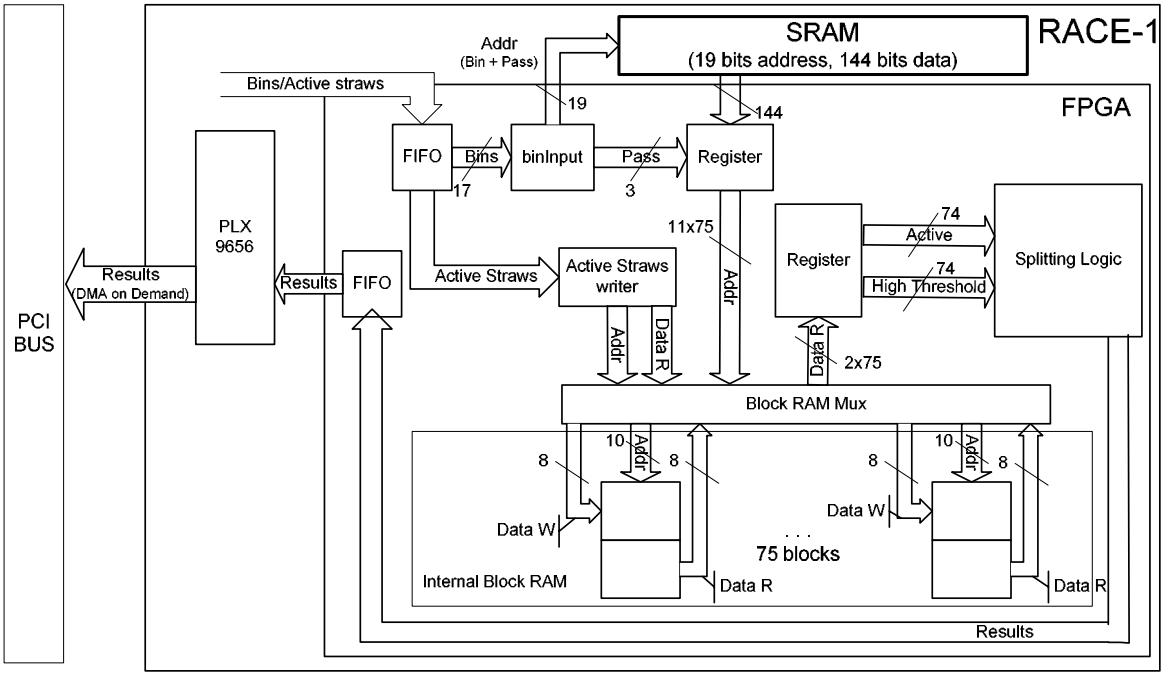

Figure 2: FPGA Track Splitting

It is possible to put both LUTs (straw-ordered and bin-ordered) in to SRAM memory of the MPRACE board if information about detector symmetry will be in use. For barrel the 32-fold symmetry allows to store only 32 $\varphi \times 80$ $1/P_T$ = 2560 bins. Therefore we need 20 bit address at the

SRAM: 16 bits for “straw-ordered” LUT (52544 straws), 3 bits for passes and 1 bit for selecting one of LUTs (“bin-ordered” or “straw-ordered”). Without symmetry we need 21 bit address: 17 bits for “bin-ordered” LUT, 3 bits for pass and 1 bit for LUT selecting.

3-7-1 Execution time with FPGA coprocessor measurement results

VHDL implementation of the initial track finding step of the TRT-LUT algorithm is ready and integrated with CTrig-01-15-12.

Measurements have been made on a computer equipped with dual Xeon 2,4 GHz CPU, 64Bit/66MHz PCI bus, 1024 Mb DDR RAM main memories with CERN Red Hat Linux 7.1 running CTrig-01-15-12 and MPRACE board. A data file (Y00347_1.atrdmp.data) containing approx. 156 events was used.

Identical results are obtained for the initial track finding step for both the CPU and FPGA implementations.

The TRT consists of two subdetector types, the barrel and the end caps. Both measure essentially in two dimensions, the barrel in r-ϕ and the end caps in z-ϕ. The algorithms for barrel and end caps differ, but the principles are identical. For the performance measurements presented in this paper only the barrel algorithm was used. The TRT barrel is composed of two identical (in the scope of this document) parts, the left and right barrel half. The following numbers refer to one half barrel.

The total number of straws is 52500. The resolution required for the Initial Track Finding is defined by 1024 bins in φ space and 80 bins in $1/p_T$ space leading to total search space 81920 patterns. Information about barrel symmetry is not used.

Measurements results are shown in following table:

Number of events: 156
Average number of hits in event: 2167.413
CPU: Pentium-IV XEON/2400; FPGA board: MPRACE(VIRTEX II) 64 MHz; PCI 64Bits/66MHz
Pattern-ordered LUT with 80 P_T and 1024 φ values predefined
Threshold: 12; Isolation: 9
Average number of track candidates per event: 57

Task:	Platform:	Time(μs)
CPU-only Implementation		
Fill event	CPU	107.77
Histogramming:	CPU	2314.03
Threshold/Max Find	CPU	376.76
Histo/Thresh/Max	**CPU**	**2694.72**
Track Splitting	CPU	1246.18
Track Fitting	CPU	151.91
Final Selection	CPU	25.05
Copy tracks	CPU	145.62
Extract time:	**CPU**	**4371.24**
Hybrid Implementation		
Fill event	CPU	115.23
Data converting	CPU	24.62
prepare write buffer	CPU	81.55
DMA write/Histogramming	PCI/FPGA	433.04
prepare read buffer	CPU	23.44
Max Finding/DMA read	FPGA/PCI	22.42
output formating	CPU	17.71
Histo/Thresh/Max	**FPGA/CPU**	**612.77**

Track Splitting	CPU	1274.87
Track Fitting	CPU	155.54
Final Selection	CPU	24.80
Copy tracks	CPU	149.71
Extract time:	**FPGA/CPU**	**2332.92**

Table 7: Execution times on a dual Xeon 2.4GHz PC for the TRT-LUT CPU only and hybrid implementation. (The total time is not the sum of the average times shown in the table, but rather is the average of the total times per event.)

Conclusions

Some of track reconstruction algorithms which are common to all *B*-physics channels and standard RoI processing have been tested for execution time and assessed for suitability for speed-up by using FPGA coprocessor. Execution time measurements results are presented here. There is possibility of speed-up some algorithms with FPGA coprocessor board. Possible speed-up has been shown by the example of TRT-LUT algorithm.

The entire TRT barrel scan algorithm is implemented on the hybrid FPGA/CPU system with MPRACE coprocessor board. The two most time consuming steps of the algorithm are implemented on FPGA. It gives identical physics results as CPU only C++ implementation. We have speed-up by factor ~2 for hybrid FPGA/CPU realisation in comparison with CPU only implementation.

References:

[1] Atlas detector and physics performance TDR, ATLAS Collaboration
ATLAS TDR 14, CERN/LHCC 99-14

[2] Pattern Recognition in the TRT for the ATLAS B-Physics Trigger. C. Hinkelbein, A. Kugel, R. Männer, M. Müller, M. Sessler, H. Singpiel, J. Baines, R. Bock, M. Smizanska,
ATL-DAQ-99-012, CERN, September 1999

[3] LVL2 Full TRT Scan Feature Extraction Algorithm for B Physics Performed on the Hybrid FPGA/CPU Processor System ATLANTIS: Measurement Results. C. Hinkelbein, A. Kugel, R. Männer, M. Müller, M. Sessler, H. Simmler, H. Singpiel,
ATL-DAQ-2000-012, CERN, 21 Mar 2000

[4] Prospects of FPGAs for the ATLAS LVL2 Trigger. C. Hinkelbein, A. Kugel, L. Levinson, R. Männer, M. Müller, M. Sessler, H. Simmler, H. Singpiel,
ATL-DAQ-2000-006, CERN, 2000

[5] ATLAS Inner Detector Technical Design Report, ATLAS Inner Detector Community, CERN/LHCC 97-16, 30 April 1997.

[6] Detector and readout specifications, and buffer-RoI relations, for the level-2 studies, P.Clarke, S.Falciano, P.Le Du, J.B.Lane, M.Abolins, C Schwick, F.J.Wickens, ATL-DAQ-99-014, 18 Oct 1999.

[7] Algorithms in CTrig and the TrigSteerExample1 Athena-based prototype
http://hepunx.rl.ac.uk/atlasuk/simulation/level2/doc/ctrigCVS/algorithms.html

[8] PixTrig: a Level 2 track finding algorithm based on pixel detector. A. Baratella, P. Morettini, M. Dameri, F. Parodi, ATL-DAQ-2000-025

[9] A Data Preparation Algorithm for the Precision Tracker LVL2 FEX, R. Dankers, J. Baines, ATL-DAQ-99-001, 8 Feb 1999.

[10] Fast tracking in hadron collider experiments, N. Konstantinidis and H. Drevermann

[11] Performance of a LVL2 Trigger Feature Extraction Algorithm for the Precision Tracker, R. Dankers, J. Baines, S. Sivoklokov, ATL-DAQ-99-013

[12] B-Physics Event Selection for the ATLAS High Level Trigger, J. Baines, A. Baratella, B. Epp, S. George, V. M. Ghete, L. Guy, S. Gonzalez, D. Hutchcroft, W. Li, P. Morettini, A. Nairz, F. Parodi, S. Qian, F. Rizatdinova, D. Scannicchio, M. Sessler, P. Sherwood, S. Sivoklokov, M. Smizanska, ATL-DAQ-2000-031, 20 Jun 2000

[13] Using Floating Point Arithmetic on FPGAs for Accelerating Scientific N-Body Simulations. G. Lienhart, A. Kugel, and R. Männer. In Proc. FCCM'02, Symposium on Field-Programmable Custom Computing Machines, Napa Valley, California, USA, April 2002.

[14] Further Test For The Simultaneous Pattern Recognition and Track Fitting by the Kalman Filtering Method. P. Billoir, S. Qian (CERN & Frascati). Nucl.Instrum.Meth.A294:219-228,1990 & Nucl.Instrum.Meth.A295:492-500,1990

[15] An Object Oriented Model for the Track Reconstruction in High Energy Physics Experiments http://www-wisconsin.cern.ch/~sijinat/oo.html

[16] High p_T Level 2 Trigger Algorithm for the TRT Detector in ATRIG, S. Sivoklokov, ATL-DAQ-2000-043, 13 May 2000

[17] Pattern Recognition in Layered Track Chambers using a Tree Algorithm, D.G.Cassel and H.Kowalski Nucl.Instrum.Meth.185(1981)235.

[18] Track finding in SCT using a tree algorithm, W.Li
http://www.hep.ph.rhbnc.ac.uk/~li/ctrig/treeAlgorithm.htm