

Hardware format pattern banks for the Associative memory boards in the ATLAS Fast Tracker Trigger System

Clay James Grewcoe

Department of Physics

Faculty of Science, University of Zagreb
Zagreb, Croatia

Supervisor: Lucian Ancu

ATLAS FTK group

August 29, 2014

Abstract

The aim of this project is to streamline and update the process of encoding the pattern bank to hardware format in the Associative memory board (AM) of the Fast Tracker (FTK) for the ATLAS detector. The encoding is also adapted to Gray code to eliminate possible misreadings in high frequency devices such as this one, ROOT files are used to store the pattern banks because of the compression utilized in ROOT.

1 Introduction

1.1 ATLAS detector

At 14 TeV there are 40 million proton-bunch crossings per second in the LHC, with each bunch collision producing approximately 20 events. This means that there are 1 billion events occurring at any given second, out of these there are only a small handful of those interesting to current experiments. We need to discard all uninteresting events to stay within computational and storage costs, this is done by the 3-level Trigger system (level-1: hardware, level-2 and 3: software) which reduces the event rate to about 200 per second.

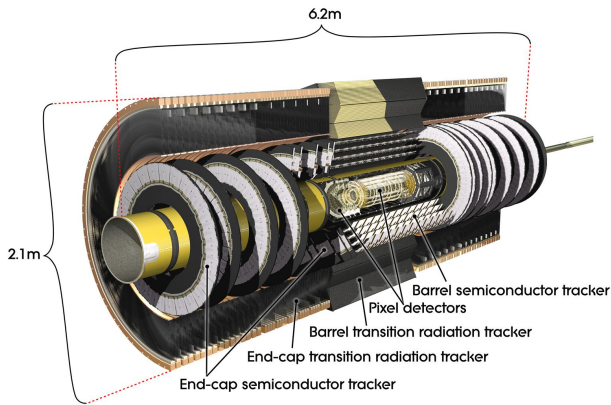


Figure 1: ATLAS inner detector with the barrel and end-cap regions of the sub-detectors. [4]

1.2 FTK

FTK is an electronics system that rapidly finds and reconstructs tracks in the inner-detector pixel and SCT layers for every event that passes the level-1 trigger. To be able to perform global tracking at the 100 kHz level-1 trigger rate, the FTK is made highly parallel, that is,

the system is segmented into 64 $\eta - \phi$ towers (16 ϕ and 4 η). [2]

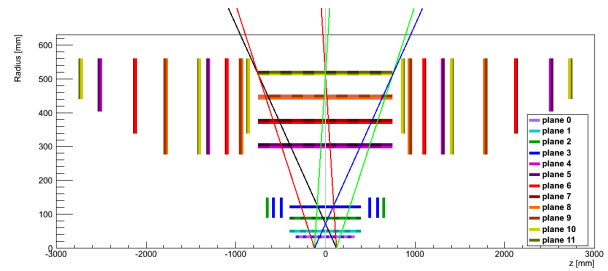


Figure 2: 4 η regions and 12 layers of the barrel and end-caps of the inner detector. End-cap regions left of the black line and right of the purple line, barrel regions between the red and green lines. Note the overlap between regions. [4]

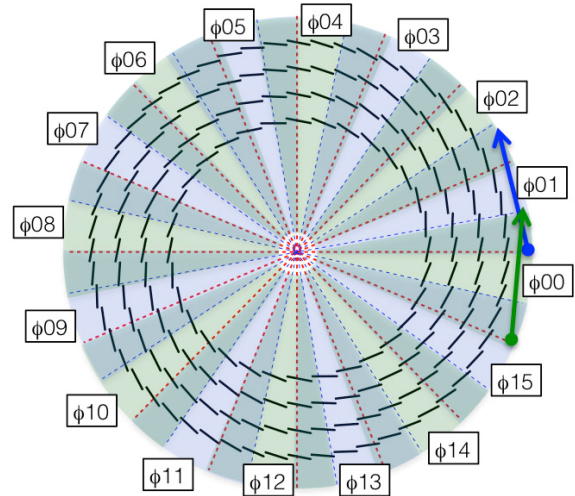


Figure 3: 16 ϕ regions. Regions 00 and 01 explicitly shown with green and blue arrows, note the overlap between the arrows. [4]

These correspond to different geometrical regions in the detector, there is also an overlap implemented between the regions to eliminate the problem of a particle traversing from one region to another. In the first stage only 8 (3 pixel and 5 SCT) of the 12 layers are used. Dataflow through the FTK is shown in Fig. 4 and is as follows.

Hits come from the **ROD** or silicon detector Read-Out-Driver into the **DF** or Data Formatter input mezzanine cards that perform the initial cluster finding. The DF then distributes the data to the corresponding towers. The **DO** or Data Organizer receives the full resolution hits and converts them to a coarser resolution (Super-Strips) appropriate for pattern recognition. Connected to the DO via high speed link is the **AM** or Associative Memory, a memory bank that compares the received hits with pre-calculated patterns stored in the memory bank (Pattern Bank). Patterns that have been hit in 7 or 8 layers (called roads) are passed on back to the DO and then the full resolution hits to the **TF** or Track Fitter which performs track fitting. **HW** or Hit Warrior removes duplicate tracks. The tracks are then sent to the **SSB** or Second-Stage Board that extrapolates to the 4 unused layers. **FLIC** or FTK-to-Level-2 Interface Crate reorganizes the data with the standard ATLAS protocol.

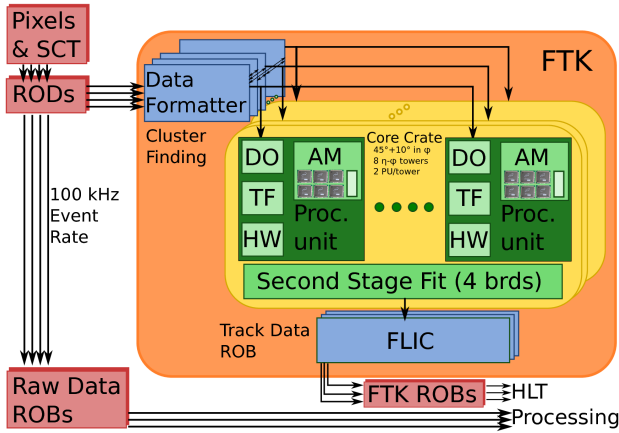


Figure 4: FTK functional diagram. [4]

2 Pattern bank

2.1 Fixed resolution problem [2]

Because of the finite width of a pattern, a road may contain silicon hits from different particles. This produces fake roads through which no single particle fully passes. Fake roads, whose track candidates will all be rejected when fit at full hit resolution, are produced at a rate that increases with increasing pattern width and increasing hit occupancy. The number of roads found can become much greater than the actual number of tracks at high detector occupancy. Since the number of tracks to be fit is determined by the number of roads

to be processed and the combinatorics of 1 hit per layer in a road, the fitting workload depends on the average ratio of the number of matching roads to the number of actual tracks in the event, $\langle \text{roads/track} \rangle$, as well as on the average number of hit combinations per road, $\langle \text{combinations/road} \rangle$. For this reason the quality of the pattern bank is characterized by both its coverage and by the rate of fake roads it generates. The coverage is defined as the fraction of tracks that match at least one pattern in the bank. Given a certain pattern size, the coverage of the bank can be increased by adding patterns to the bank. However, the number of fake roads is roughly proportional to the number of patterns in the bank. Moreover, as the luminosity increases, the fake rate increases rapidly because of the increased detector occupancy. To counter that, the pattern width must be reduced. If we decreased the pattern width using the standard AM technology, the number of patterns would increase sharply, making the overall system very large and extremely expensive.

2.2 Variable resolution [2]

Variable resolution solves this problem, each pattern and each detector layer within a pattern can have an optimal width that is implemented using a “Don’t-Care” (DC) feature to increase the width of a pattern for any detector layer when that is more appropriate. In other words we use patterns of variable shape. As a result we can reduce the number of fake roads while keeping the efficiency high and avoiding the excessive bank size that would occur from an overall reduction in pattern width (see Fig. 5). The choice of wider or narrower road width is made layer by layer using the simulation. For example, when we split the road in a layer into two parts, in some cases both halves are traversed by real tracks while in other cases just one of them contains real particle trajectories. For the former, the layer is used with the DC feature enabled to widen the road by a factor of 2. For the latter, the layer is used at full resolution, reducing by a factor of two the area in that layer that the road offers to uncorrelated hits from other tracks.

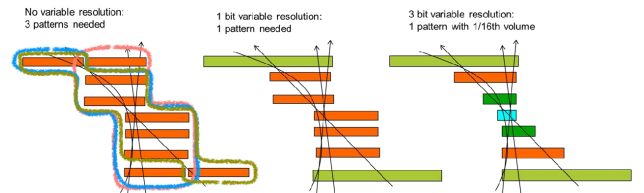


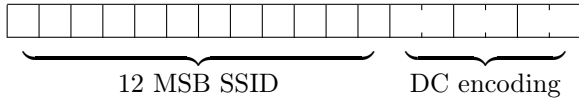
Figure 5: The sketches from left to right show the ability of variable resolution to reduce the number of Associative Memory patterns and the pattern volume. The fuzzy lines in the left sketch show the 3 patterns needed to accept the tracks. The color of a Super-Strip indicates its area. [4]

2.3 DC encoding

A pattern bank holds the unique pattern ID, the IDs of each super-strip in every layer that correspond to that pattern and also two variables called DC mask and HB (half bin) mask which contain the information for every SSID on whether the SS in that layer for that particular pattern should be single sized or double sized. Before this information can be loaded on the AM hardware it needs to be encoded to what is called hardware format, that is a format which contains the pattern ID (same as before) and the SSIDs encoded in such a way that they contain both the relevant information stored in the SSID and in the DC and HB masks. This is done with the DC encoding as follows:

- Start with 18-bit “software format” SSID
- Remove 6 least significant digits
- for PIXEL layers - 2 DC bits (ϕ and z)
- for SCT layers - 1 DC bit
- PIXELs: bits no. 0 and 1 encoded with 00 if DC-mask is true otherwise 10 if HBmask is 1 or 01 if HBmask is 0
- SCTs: bit no. 0 encoded with 00 if DCmask is true otherwise 10 if HBmask is 1 or 01 if HBmask is 0
- bits no. 1 (if SCT) and 2 encoded with 10 if bit is 1 or 01 if bit is 0.

The final format has the form shown below



2.4 Encoding scripts

Prior to my arrival there were two scripts to do the encoding of the pattern bank, `Bank_Class.C` and `main.cpp`, the first read all relevant information from the pattern bank ROOT file and dumped it into a text file, this file was then read by `main.cpp` encoded and the results dumped on screen. `Bank_Class.C` was made to work as a macro under ROOT while `main.cpp` was a standalone C++ program. These two scripts needed to be merged to streamline the process and also to avoid using the unnecessary intermediary text file. The encoding is schematically shown in Table 1. The output has been also moved from screen to both text and ROOT files.

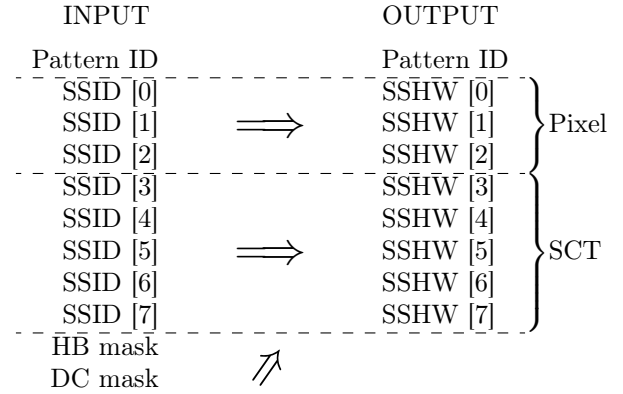


Table 1: Diagram of the information contained in the input file and the information to be contained by the output.

2.5 Gray code

Gray code (also called reflected binary code) is a type of binary (base-2) code with the key property that two successive numbers differ only by one bit. For example all 3-bit Gray code numbers are shown in table 2 with their corresponding decimal and binary numbers.

Decimal	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Table 2: All 3-bit numbers in decimal, binary and Gray code.

Gray code, thus, eliminates the possibility of misreading a number in fast devices that deal with sequential numbers, because there are no intermediary numbers from differently timed bit flips that can be read instead of the final number.

3 Look-Up-Tables for the IM and DF

3.1 Brief introduction to LUTs [3]

The data from the pixel and SCT detectors is transmitted to the RODs on optical fibers (FELs), which carry the link number, to the IM. The IM combines the linknumber with the RODID to obtain the OnlineId which then gets converted via the Look-Up-Table to a hashID. Via the second LUT the DF can then propagate the information which plane the data is coming from, to the appropriate towers handling this RODID.

This is schematically shown in Fig. 6. More details in [3].

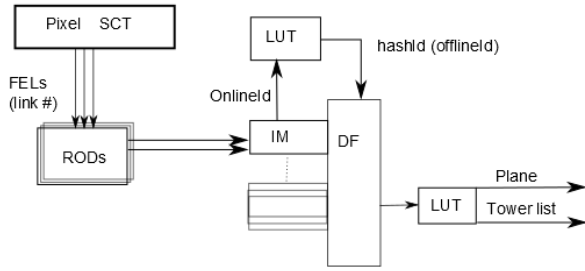


Figure 6: Dataflow diagram. [3]

3.2 Modification of output to ROOT file

Originally, the LUTs were text files, one file per RODID, however because of the better compression qualities of ROOT files and integration with other software it was needed to make the output a single ROOT file. This ROOT file has trees for each RODID instead of text files, the information in each text file is now sorted to specific branches within that tree. The information stored in the branches of a tree are shown in Table 3.

BRANCH	MEANING
Link-number	Integer (0-127)
OnlineID	hexadecimal (32-bit)
HashID	Integer (0,N), also known as OfflineID
Plane	Maps the HashID to a Plane
Tower List	List of towers processing the specific RODID
Barrel Endcap	0 if barrel, 1 if endcap
Layer Disk	Indicating the layer
Phi module, min, max	Region mapping number
Eta module, min, max	Region mapping number
Eta index, min, max (Pixel)	Region mapping number
Side (SCT)	Inner/outer part of Si wafer (0,1)
Strip (SCT)	Strip number (0,767)

Table 3: List of branches and their meaning. Table from [3].

4 Results and Conclusion

The pattern bank translation to hardware format is currently not used because of the input files not ready yet. The unified Bank_Class.C and main.cpp and modified Bank_Class.h are available in SVN at [5] and [6]. The LUTs are being used currently and the updated ROOT output version is available in SVN at [7]. In order to work on this project I had to learn ROOT and C++ both of which I have never used before, I've gained experience of working in a large collaboration and presenting my work.

References

- [1] ATLAS Fact Sheet http://www.atlas.ch/pdf/ATLAS_fact_sheets.pdf.
- [2] ATLAS Collaboration, CERN-LHCC-2013-007, ATLAS-TDR-021 CDS: <http://cdsweb.cern.ch/record/1552953/>.
- [3] Jean-Michael Poudroux, CERN-STUDENTS-Note-2014-019 CDS: <https://cds.cern.ch/record/1748468>.
- [4] Images from [2], <https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/UPGRADE/CERN-LHCC-2013-007/>
- [5] Bank_Class.C [SVN link not available yet]
- [6] Bank_Class.h [SVN link not available yet]
- [7] FTKRegionalWrapper.cxx [SVN link not available yet]