ATLAS : Monitoring the AMBoard for Fast Tracking (FTK)

Mathilde GUILLAUD
CERN - Switzerland

2019
August

Abstract

The purpose of this project is to make the status information of the AMBoard on ATLAS's FTK available to the remote monitoring application of the board. The Fast TracKer is designed to reconstruct rapidly the tracks of an event occurring in the ATLAS detector. The AMBoard is in charge of matching the detector hits with the corresponding charged particle paths. To check the data and status in the board, software to access those data in the board is being developed. In and Out data of the AMBoard is monitored by an application through the EMon service, and the status information is currently being sent to the IS service. The merging between IS and EMon is possible but will imply several modifications of the current software such as the creation of new tools and classes as well as changes in the main file and the monitoring applications. The function which connects IS and EMon information has been prepared in this project. Some issues still remain, such as defining a way to cope with the difference in the type of the variables.

Supervisors : Paolo Mastrandrea, Alberto Annovi and Tomoya Iizawa

Introduction

With a diameter of 25 m and a length of 46 m, ATLAS is the largest volume particle detector in the world. With its six detecting subsystems, it measures the paths, momentum and energy of the particles created in the collisions in its center. Collisions happen every 25 ns, which corresponds to a rate of 40 MHz. In order to be able to process the data from each collision, ATLAS uses a two-level trigger system. The Level-1 Trigger (L1) is a custom hardware that reduces the trigger rate to 100 kHz. The second level is called High-Level Trigger (HLT). The HLT is a software running on a server farm, receiving the events accepted by L1 and reducing the rate down to 1 kHz. For Run 3 which will start in 2021, the luminosity of the LHC is going to increase, which will result in a dramatic increase in the trigger rate. Currently, track information of the particles is one of the important information for the HLT to decide whether a L1 accepted event is interesting or not. However, it is very much time consuming to calculate track information. The FTK (Fast TracKing system) will be installed on ATLAS for Run 3. FTK will be placed between L1 and HLT. FTK is composed of custom hardware used to reconstruct the tracks of the particles which are provided to HLT at the start of its processing. The system is based on a pattern-recognition system. All possible tracks are simulated prior to the collisions and stored in a library. When a collision occurs, the data from the tracker devices are matched to the possible tracks and sent to HLT for trigger.

ATLAS' Fast TracKer (FTK)

When a collision occurs, the hit information from the detector is reformatted to be sent to the AMBoard, which will be in charge of matching the hits with the pattern and returning the index of the corresponding road.

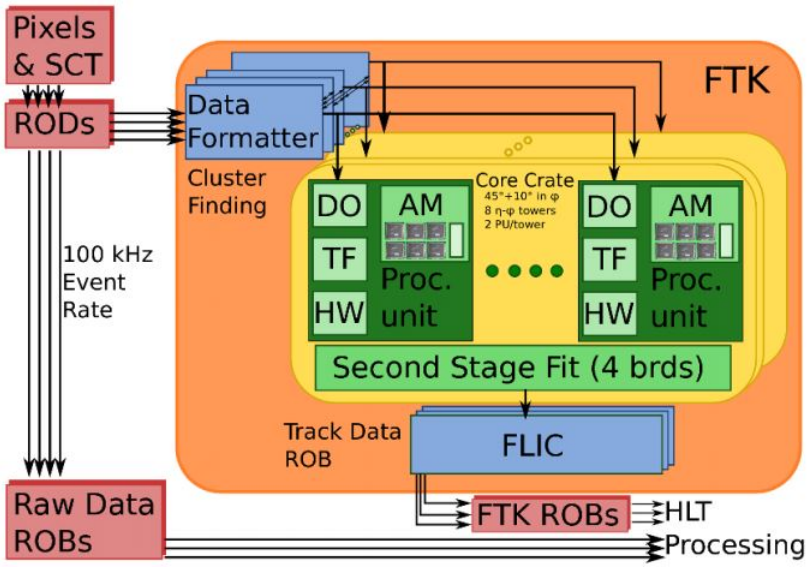

Figure 1: Functional diagram of the FTK components

The Data Formatter (DF) gets the hits from the inner tracking detector Read-Out Drivers and reorganizes them to send them to the Data Organizer (DO) on the AUX. The DO converts the hits to coarser resolution hits (Super Strips) for pattern matching and sends it to the AMBoard. The AMBoards contain almost 1 billion patterns , each correspond to paths constituted by possible combinations of hits in the tracker layers. When the hits are matched with seven or eight layers inside a pattern, the pattern is considered as matched. The matched patterns are called "roads" and the road identifiers are sent to the DO that retrieves the full-resolution tracks and sends it to the Track Fitter (TF). TF calculates tracks parameters for the received roads and the Hit Warrior removes the duplicate tracks. The Second Stage Fit then extrapolates the tracks to twelve layers instead of eight.
The FTK uses a service called EMon to transfer data for monitoring to a remote application. EMon can freeze and readout contents in the buffer for monitoring (SpyBuffers) without interrupting the data flow. The monitoring is used to compare the data in and out of the boards and check the relevance of the tracks matched by the FPGA. FTK also uses a service called IS to monitor the status information of the FPGA, such as the temperature, the states of the buffers or the number of events processed.

Project : Updating the AMB status information shipment for synchronisation

The aim of this project is to ship the status information to EMon instead of sending it to IS, in order to get synchronisation between this and the in and out data. The goal is then to find a protocol to re-format the information and ship it to EMon.
The data sent to EMon is stored in SpyBuffers, which are buffers able to copy the data from the real FPGA buffers without altering the flow of tracks. A special format for the status information couldn't be set, therefore, the idea was to store the status into SpyBuffers and send them to EMon.

SpyBuffers

The AMBoard class contains two AMBSpyBuffer attributes : m_InSpyBuffer and m_OutSpyBuffer. The AMBSpyBuffer class has an attribute m_buffer which is a vector of 32 bits integers in which the data is stored. AMBoard contains also the methods RC_publishFull() and shipSpyBuffer(...) to ship these AMBSpyBuffers to EMon.
The first step is then to implement an AMBSpyBuffer for the status and the methods to retrieve its elements and size.

- **AMBoard.h:**

```
1 std::vector< std::shared_ptr< AMBSpyBuffer >> m_StatusSpyBuffer;
2 std::shared_ptr< AMBSpyBuffer > getStatusSpyBuffer(unsigned int i) const
3 { return m_StatusSpyBuffer[i] };
4
5 int getStatusSpyBufferSize() const { return m_StatusSpyBuffer.size() };
```

The status information of the board is read differently from in and out data. The data is copied from the buffers of the board into AMBSpyBuffers whereas the status is retrieved via a DataFlowReaderAMB object. The class DataFlowReaderAMB possesses methods to access the different elements of the status information. We then need to implement a new method for the AMBSpyBuffer class to read the status information and fetch it to an AMBSpyBuffer.

- **AMBSpyBuffer.h:**

```
#include "ambslp/DataFlowReaderAMB.h"
void readStatusSpyBuffer(int n, DataFlowReaderAMB _dataFlowReader);
```

- **AMBSpyBuffer.cxx:**

```
void readStatusSpyBuffer(int n, DataFlowReaderAMB _dataFlowReader)
{...};
```

This method is only usable on an AMBSpyBuffer so we need to implement a function in AMBoard to call this previous function from the board.

- **AMBoard.h:**

```
void readStatusSpyBuffers(DataFlowReaderAMB _dataFlowReader) ;
```

- **AMBoard.cxx:**

```
void readStatusSpyBuffers(DataFlowReaderAMB _dataFlowReader)
{...};
```

With the previous functions, it is now possible to build a function equivalent to RC_publishFull() and the corresponding shipStatus (...) for the status information.

- **AMBoard.h:**

```
void AMBoard::RC_publishStatus();
void AMBoard::shipStatus(std::vector<std::shared_ptr<AMBSpyBuffer>>
                         & vec_spyBuffers);
```

- **AMBoard_procedures.cxx:**

```
void AMBoard::RC_publishStatus()
{...};
void AMBoard::shipStatus(std::vector<std::shared_ptr<AMBSpyBuffer>>
                         & vec_spyBuffers)
{...};
```

With all these functions, the status information could be made available to EMon. One of the issues which needs to be fixed is the format. The type used for the status is a long int (64 bits) whereas the AMBSpyBuffers only use int vectors (32 bits).
One solution could be to test the different status variables to check if they need to be on 64 bits or can be fitted in 32 bits. Still, some variables for the status are strings so this solution would remain incomplete.
Another solution could be to create a new class of SpyBuffer which would have a 64 bits int buffer and a string buffer. This solution demands more work since all the methods from the SpyBuffer class that are necessary for manipulation of the data needs to be adapted for the new class. Moreover, it would also require access to the main script and the remote application used for monitoring in order to add functions there to interpret the incoming new buffers.

Conclusion

The merging of IS and EMon should be possible but since there are several differences between the two, a lot of alterations and discussions are going to be necessary to obtain a complete solution. Indeed, it's necessary to solve how to cope with the difference in the type of variables. In order to implement such a solution, access to the main file and to the details of the remote monitoring application are needed to link all the elements together and make sure the information made available to EMon can be interpreted by the application and read correctly. The procedure is under discussion.

References

- FTK: The hardware Fast TracKer of the ATLAS experiment at CERN - Ioannis Maznas on behalf of the ATLAS Collaboration
- https://twiki.cern.ch/twiki/bin/view/Atlas/FtkEMonDataOut