Journal of Physics: Conference Series **119** (2008) 022032 doi:10.1088/1742-6596/119/2/022032

Cathode Strip Chamber (CSC) raw data unpacking and packing using bit field classes

Alex Tumanov[1]

Rice University, MS315, 6100 Main St., Houston, TX 77030

tumanov@rice.edu

Rick Wilkinson

California Institute of Technology, 1200 East California Blvd, Pasadena, CA 91125

rickw@caltech.edu

Abstract. Unprecedented data rates that are expected at the LHC put high demand on the speed of the detector data acquisition system. The CSC subdetector located in the Muon Endcaps of the CMS detector has a data readout system equivalent in size to that of a whole Tevatron detector (60 VME crates in the CSC DAQ equal to the whole D0 DAQ size). As a part of the High Level Trigger, the CSC data unpacking runs online and it needs to be able to cope with high data rates online. Early versions of the unpacking code used bit shifts and masks to unpack binary data. To reduce the unpacking time we decided to switch to bit field based data unpacking. The switch allowed us to gain an order of magnitude in speed. In this paper we explain how bit field data unpacking works and why it is dramatically faster compared to conventional bit shift and mask methods.

1. Introduction

One of the tasks of data acquisition software is to interpret the byte stream read out by detector electronics. This is commonly referred to as raw data unpacking and it is usually one of the first steps in the DAQ software chain and therefore it usually is being executed online, i.e. in the real time as the data come in from the detector. As a result one of the main criteria for the unpacking software must be its execution time.

The unpacking software most often is implemented in C/C++ using masks and bit shifts and assignment operators that designate certain bits to specific variables that are being unpacked. Here we would like to present an alternative way to unpack raw data using bit field classes which dramatically reduces the CPU time spent on the unpacking.

[1] To whom emails should be addressed.

© 2008 IOP Publishing Ltd

2. CSC detector overview

Cathode Strip Chambers, 468 multiwire proportional drift chambers with cathode strip and anode wire readouts, are a part of the Muon Endcaps of the CMS detector. The CSCs are read out via 60 VME crates that are situated on the detector and contain the bulk of the L1 filtering electronics. The L1 data is then sent into 36 Detector Dependent Units (DDUs) that concentrate the information further into 8 Detector Concentrator Cards (DCCs) which serve the purpose of the Front End Drives (FEDs). Each FED has a maximum data rate of about 200 Mb/s. The FEDs supply the data to the event builder and consequently the High Level Trigger (HLT) at 100 kHz. See Figure 1.

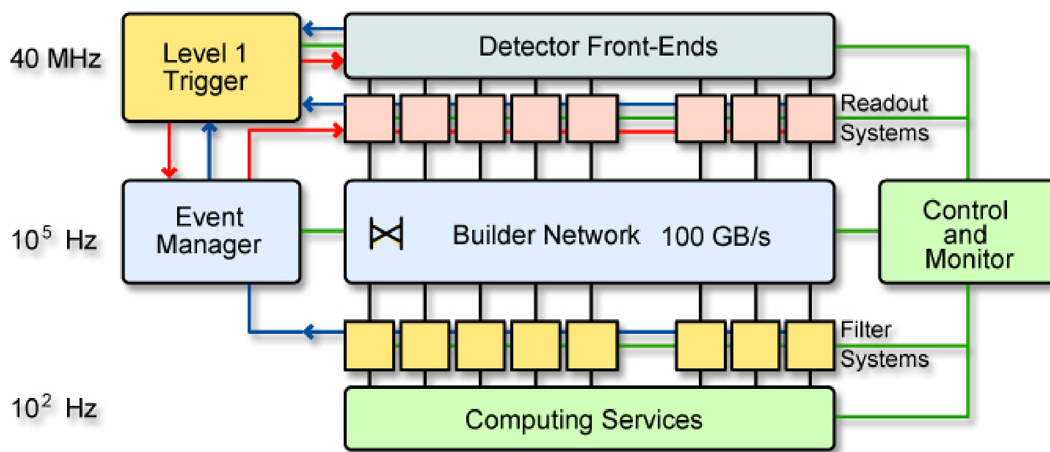

Figure 1. Schematic view of the CMS DAQ system.

The CMS HLT is fully software based. It is designed to reduce the event rate from 100 kHz as supplied by the FEDs to about 100 Hz – the limit dictated largely by the data archiving capabilities. The CSC unpacking software runs as one of the first steps in the event processing chain. The raw event data from the detector, coming in at 100 kHz, need to be interpreted into higher level C++ objects, digis, that later could be processed into tracks or other HLT objects. Based on which the HLT is then able to make a decision on whether or not to keep the event. The HLT time budget is about 40 ms per event. Being a part of the HLT the CSC unpacking should be able to process events in a fraction of the total HLT time budget, i.e. a few milliseconds.

3. CSC Unpacking

CSC Unpacking classes generally follow the data structure of the CSC Event. On-chamber and detector-mounted electronics boards such as ALCT, CFEB, DMB and TMB send their data to the 36 DDUs which are then funnelled via 8 DCCs. This results in the CSC Event having a structure of a Russian doll set with smaller data structures from on-chamber electronics wrapped in large DDUEventDatas and several DDUEventDatas encompassed by the DCCEventData, see Figure 2. The unpacked data is stored in a set of persistent C++ data structures called digis. Digis are the final product of the unpacking. The online event reconstruction uses the digis to reconstruct higher level objects such as tracks.

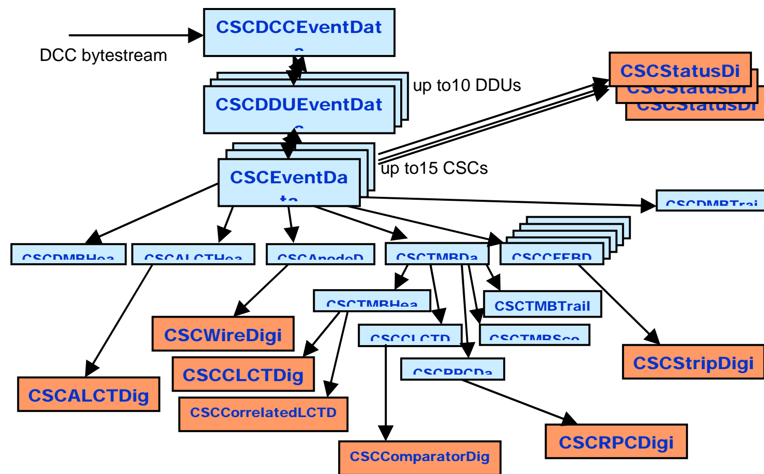

Figure 2. Schematic view of the CSC unpacking classes. Arrows indicate data flow rather than inheritance structure. Shown in brown are persistent data classes – digis.

To understand the details of the bit field based unpacking we will focus on one class – CSCDMBHeader. This class unpacks the header of the data from one of the detector-mounted electronics boards, the DAQ Mother Board (DMB). There are 29 variables in the DMB header such as L1A number, DMB Id, etc. The conventional way to unpack raw DMB header buffer is to use bit shifts and masks to fill each of the 29 variables. This results in at least 29 lines of code such as those shown in Figure 3. This was how the DMB header was unpacked in the early versions of the CSC unpacking code. Faced with the growing numbers of variables[2] and format versions[3] on one hand, and the limit on the execution time on the other, we have decided to adopt a new way to unpack the data using bit fields.

```
… … …
// the L1A number
int L1A = (theData[i] >> 12) & 0xF;
// CSC Id
int CSCId = (theData[i] >>6) & 0x3F;
//DMB Id
int DMBId = (theData[i+2] >> 8) & 3F;
… … …
```

Figure 3. This is a fragment of the old version of the DMB header unpacking class. It contained 29 lines of bit shifts and masks and assignments.

The fragment of the bit field implementation of the CSCDMBHeader class is given in Figure 4. In it all of the 29 variables are arranged in the same order as they appear in the data. Each variable, or a bit field, has a size of a fixed number of bits defined after colon. The size in bits for each variable is defined by the data format of the given electronics board, in this case the DMB. Thus, for example the

[2] Total number of variables being unpacked currently is 355.
[3] There are five different data format versions corresponding to different versions of the DMB and other boards.

L1A variable is 4 bit long and it appears in the DMB data 6 bits prior to the DMB Id. The 6 bits between the L1A and the DMBId carry the information for another variable – CFEBId. The length of the DMBId is also 6 bits. When all of the 29 variables in the DMBHeader class are designed as bit fields it can be unpacked with just one line of code -

*CSCDMBHeader (unsigned short * theData) { memcpy(this, theData, sizeInWords()*2); }*

In the bit field based CSCDMBHeader class the single memcpy replaces 29 lines of bitshifts, masks and assignments. As a result we were able to achieve an order of magnitude reduction in the average unpacking time, which was recently measured during the HLT exercise [1] to be 3ms/event.

```
class CSCDMBHeader {
   public:
     ... ... ...
   unsigned L1ANumber() const {return L1A_;}
   unsigned CSCId() const {return cscId_;}
   unsigned DMBId() const {return dmbId_;}
     ... ... ...
   private:
     ... ... . ..
unsigned L1A_      : 4;
unsigned cfebId_   : 6;
unsigned dmbId_    : 6;
    ... ... ...
```

Figure 4. Bit Field implementation of the CSCDMBHeader. The data format could easily be read since variables are arranged in the same order as they appear in the data.

4. Raw data packing

Data stored in persistent objects, digis, can be converted into a binary byte stream. This is referred to as Digi-to-Raw packing. There are several uses for data packing, for example, it is needed to realistically simulate HLT performance based on Monte Carlo data. In this case, simulated digis are converted into MC raw data which are then fed to the HLT at rates that realistically simulate real data taking.

One of the advantages of the bit fields is the fact that the data fields are already "packed" inside of the classes. And a data() method like this *unsigned short * data() {return (unsigned short *) this;}* returns the needed byte stream for the given class. We use *boost::dynamic_bitset*<> type to combine the byte streams for each of the data classes into a single continuous raw event.

5. Conclusions

An order of magnitude in data unpacking speed can be gained by using bit field based classes instead of conventionally used bit shifts, masks and assignment operators.

Another important advantage of the bit field based unpacking is the readability of the classes. Given the data format, bit field variables are orderly arranged as the class data members. The fact that the bit field classes are easier to read and write is a huge advantage because it helps to avoid probably the biggest source of unpacking problems – errors/typos in defining bits.

When designing bit field classes, particular attention should be paid to portability issues and the memory management. Different endian-ness and CPU architecture may affect the unpacking. We

have tested our code on gcc/Linux 32 bit CPU machines and recently successfully moved to 64 bit machines without any problems.

References

[1] CMS High Level Trigger, CMS Collaboration, CERN/LHCC 2007-021, LHCC-G-134