

New additions to trgStructures.h

John Nelson

9 October 2007

Why do we need changes to trgStructures.h?

1. The trigger rate is dependent on the frequency which Level 2 can deliver ACCEPT commands to the detectors. The ACCEPT decision is made when all fast detectors have transferred their data to L2. In the absence of data from any detector, L2 will timeout and generally abort the event. The algorithms which interrogate the data in order to provide the ACCEPT/ABORT decision depend, in particular, on data from Level 1 and data from BTOW and ETOW.

At present, data transfer time from Level 1 is about 20% of that from BTOW/ETOW. A project has been in progress over the summer to re-route the data from EMC direct to Level 2 via a high-speed link which will significantly reduce the BTOW/ETOW transfer time, and hence the time taken to make the ACCEPT/ABORT decision, leading therefore to a significant potential increase in trigger rate.

But this means that BTOW/ETOW data will now arrive in DAQ (via the Global Broker) from Level 2 and not from the VME BTOW/ETOW nodes.

2. The data transfer from Level 2 to GB will be a single block containing all the usual trigger data as well as EMC data. It is therefore necessary to make additions to trgStructures.h

A new struct is defined:

```
#define MAX_OFFSET 25
typedef struct {
    int byteCount_Version; /* Transfer count in MS 24 bits; Version in LS 8 bits */
    TrgOffflen OffsetBlock[MAX_OFFSET]; /* Offset/length into transferData */
    int transferData[1]; /* Place holder array for trigger, BTOW and ETOW */
} TrgTowerTrnfer;
```

The transferData block will contain trigger data, and tower data. The (byte) offsets from the beginning of the TrgTowerTrnfer block to the various elements of the data, and the length of each data element, are contained in the OffsetBlock. The offflen struct is defined as:

```
struct TrgOffflen {
    int offset; /* Offset (in bytes) from the start of TrgTowerTrnfer to data */
    int length; /* Length (in bytes) of data. */
};
```

The indices into the OffsetBlock are defined as:

```
#define TRG_INDEX 0 /* Offset/length of trigger data: TrgDataType */
#define BTOW_INDEX 1 /* Offset/length of BTOW data: */
#define ETOW_INDEX 2 /* Offset/length of ETOW data: */
#define RAW_TRG_INDEX 3 /* Block of 11 contiguous offsets to rawTriggerDet[] */
```

The trigger data begin, as defined in TrgDataType, with the Event descriptor. The remaining elements of the trigger data follow after the Event Descriptor as usual. Please note that each element of the rawTriggerDet array in TrgDataType is not guaranteed to have the same length. Consequently, access to the data via direct indexation into the rawTriggerDet array will give erroneous results.

There is one exception to this: the first element of the array are data for the main crossing and can be accessed as rawTriggerDet[0].<data>

The block of 11 contiguous offset/lengths provide access to the elements of rawTriggerDet.

3. How do you access the rawTriggerDet data in the TrgTowerTrnfer block?

This code snippet shows how the offsets in the OffsetBlock could be used.

```
TrgTowerTrnfer  trgTowerData;
RawTrgDet      *rawDet[11];      /* Array of pointers */

for (ix=0; ix<pre+post+1; ix++) {
    rawDet[ix] = (RawTrgDet *) ((int) &trgTowerData +
                               trgTowerData.OffsetBlock[RAW_TRG_INDEX+ix].offset);
}
/* Now for the analysis loop */
for (ix=0; ix<pre+post+1; ix++) {
    rawDet[ix]->bbc[14]; /* Etc to get trigger data for a particular crossing */
}

```

In this case, ix=0 gets access to the main crossing data.

4. How do you access tower data in the TrgTowerTrnfer block?

```
TrgTowerTrnfer  trgTowerData;
int *towerPtr;
int towerLength;

towerPtr = (int *) ((int) &trgTowerData +
                   trgTowerData.OffsetBlock[BTOW_INDEX].offset);
towerLength = trgTowerData.OffsetBlock[BTOW_INDEX].length;

```

The exact format of the towerData will be provided by Gerard Visser. Other mechanisms to reach the tower data in a conventional way will be provided by Jeff Landgraf and Jan Balewski.

5. Data written to trigger disks will remain unchanged and will continue to be written in the same format as has been done in the past.