

# ECool Loss Monitors

## *Averaging scheme*

Thu, Jun 16, 2005

Loss monitor signals are monitored in ECool to inhibit the electron beam if losses are too high. But these signals are influenced by losses in the Main Injector that are unrelated to ECool operation. It is thought that averaging the readings from the loss monitors over one second may remove this unwelcome interdependence. This note discusses how this can be done as a local application in the front end.

Since the averaging is to be done over a one second interval, it should not be necessary to perform the averaging calculation more often than 15 Hz. A local application could generate the average data without being involved in the KHz interrupt activity. But if it does this, how can the results be monitored by the KHz monitoring already in place? The answer is to use wide-open limits for those loss monitor channels that are to be averaged, and instead use the normal 15 Hz alarms scanning to perform the check desired.

What is entailed by the averaging? For each LM, sample the KHz data points in the hardware buffer that were written during the last cycle. This will bring us "up to date." Accumulate a sum of these data, which can be called a partial sum. Also keep a partial count of the number of points represented in this partial sum. Referencing an array of 15 such partial sums, access the oldest element. Subtract the (oldest) partial count from the total count, and subtract the (oldest) partial sum from the total sum. Then overwrite the oldest partial sum with the newest partial sum, and add this same newest partial sum to the diminished total sum, and add the newest partial count to the diminished total count. Compute a revised average value by using the updated total sum and count.

If the average results are stored in dummy analog channel readings, they can be accessed in the normal way, and they could be monitored for alarms in the usual way and even inhibit the electron beam via the "beam inhibit" control line. If the relay-based hardware is not installed, the control line could be brought into a spare analog channel to be scrutinized at 1 KHz by the `KHZM` logic. This would be the means of turning off the electron beam. The indication would only identify the channel to which the digital control line is attached. The client application would then monitor the dummy channels for being in the "bad" alarm state in order to identify the LM that "got us."

The new LA can be called `KHZA`, for KHz data averaging. It operates independently from the logic of `KHZM`. As for parameters, we need to know which channels must be so averaged, and we need to know the target dummy channels. We might specify a range of LM channels and a range of target dummy channels. Multiple instances of `KHZA` can provide for more flexibility in channel assignments.

The period of averaging may be one second, but for greater flexibility, we can allow the period to be specified as a number of 15 Hz cycles, up to a maximum of, say, 30.

Let the parameters be as follows:

<code>ENABLE</code>	B	Usual LA enable Bit#
<code>PERIOD</code>		Averaging period in 15 Hz cycles
<code>FIRST</code>	C	First channel whose KHz readings are to be averaged
<code>NCHAN</code>		Number of channels to average
<code>TARGET</code>	C	First target channel to receive averages

If there are two ranges of channels to be so averaged, then two instances can cover them.

As for variable names, let `MAXCHANS` be the maximum number of channels to be handled with one instance, which might be 16. Let `MAXPARTS` be the maximum number of partial sums to be handled, which may be 30. Each element of array `partSum` can house the partial sums for each channel. Thirty sets of these partial sums make up the array. In parallel to these sets is an array of `partCnt` partial counts. A separate variable holds the `totCnt` total count for the total sums array `totSum`. One Memory Dump page (64 bytes) can thus show the total sums for all channels, or it can show any of the sets of partial sums for all channels.

Each 15 Hz cycle, the LA must determine the current KHz digitizer slot#, so it can find all the data points that have been measured and stored within the last cycle. The variable `prevSlot` can hold the previous such slot#. It would need to be initialized to the current (valid) slot#, and all sums and counts initialized to zero. Each cycle, it reduces the total sums by the oldest partial sums, the set to which `partX` refers. Then this set of partial sums is cleared, and, starting with the slot following the `prevSlot`, it collects all the readings (from that slot to the present completed slot) for each channel, accumulating them into the cleared partial sums, modifying the total sums accordingly. Then the total sums are divided by the total count to yield the average readings that are deposited as the dummy channel readings to be monitored by the usual alarm scanning task.

### Post implementation notes

The above plan was implemented as `KHZA` and testing in `node0593`, where it showed that 1.7 ms per 15 Hz cycle is required to perform the averaging of KHz data for the maximum of 16 consecutive channels. The maximum specifyable averaging period is 32 cycles, or just over 2 seconds. (The minimum is 1 cycle.) The arrangement of the static memory allocated by the LA is as follows:

```
typedef struct Globals {
    MBHdrType    newHdr;           /* generic memory block header */
    Integer      localNode;       /* local node# */
    Integer      timeInit;        /* initialization processing time */
    Integer      timeEx;          /* execution time in usec */
    Integer      prevSlot;        /* previous slot in A/D memory buffer */
    Integer      nParts;          /* #partial sums */
    Integer      bChan;           /* base input chan# */
    Integer      nChans;          /* #chans in sequence */
    Integer      tChan;           /* target result chan# */
    Integer      partX;           /* partial sum index */
    Integer      totCnt;          /* total count */
    Longint      regBase;         /* register base address of A/D IP board */
    Longint      memBase;         /* base address of A/D memory */
    Integer      fill[14];
    /*-----0x0040-----*/
    Integer      average[MAXCHANS]; /* average results */
    Integer      fillAvg[16];
    /*-----0x0080-----*/
    Integer      partCnt[MAXPARTS]; /* partial sum counts */
    /*-----0x00C0-----*/
    Longint      totSum[MAXCHANS]; /* total sums for all channels */
    /*-----0x0100-----*/
    PartBlk      partSum[MAXPARTS]; /* partial sums for all channels */
    /*-----0x0900-----*/
} Globals; /* static memory globals */
```