

Booster BLM Requirements

Notes from David Herrup document

Apr 13, 2000

55–70 beam loss detectors around the Booster ring

Assuming 10KHz digitizers,
350 digitization/detector can cover 35 ms
 $350 \times 70 = 24500$ digitization total

Waveforms to be collected and stored based upon reset type

Want system to compare waveforms from past and present

Freeze data for access by console until console is done with it.

Can this be a problem for supporting multiple users of this data? It should be better to deliver a consistent waveform to the console.

Dose monitoring:

Compute a dose for each BLM and compare against some limit. Generate alarm if exceed limit. The dose is accumulated (for all resets) over a sliding time interval. Does the reading reflect an integrator's output? This question arises from "the BLM reading at the end of each pulse is added to the running sum"

Running sums are maintained for each Booster reset, although only the total dose generates an alarm.

Data logging snapshots

Specify up to 10 times during each pulse for dose computation, called average dose. This has to be done separately for each Booster reset.

Booster BLM Scenario

Suppose one IRM connected to 8 BLM waveforms using Swift digitizers, which are available now. They can operate at 800KHz, or slower rates by factors of two all the way to 12.5KHz and finally 6.25KHz. Choose one of these rates for BLM waveform digitizing.

Recognizing that the console cannot keep up with 55–70 waveforms at 15Hz, there must be some buffering in the front end. If 12.5KHz is used, about 440 points will be stored every cycle for each waveform. This covers 35 ms of time.

A local application can do the required linearization for each waveform, prepare a data stream record and write it into a data stream queue. Eight waveforms would occupy about 8K bytes.

A separate data stream could be defined for each Booster reset. If necessary, separate data streams could be defined for each waveform. If there were 8 different Booster resets, there might be a total of 64 data streams defined per node. In any single 15Hz cycle, only 8 waveforms would be processed and written into data stream queue(s). Assume for this discussion that one data stream queue is used for each Booster reset to hold all 8 waveforms. Then 32K bytes could hold the last four copies of the waveforms for a single Booster reset.

A console that needed to access such waveform data can do it via Acnet devices that target

the appropriate data stream. The number of bytes specified in the data request should be at least what can be recorded in one 15Hz cycle. If this were 8K altogether, there would have to be at least two separate requests, since there is a size limit to one Acnet message of 3982 bytes. One Acnet message can carry up to 4 waveforms.

A RETDAT data request can specify a return based upon a clock event. If a separate data stream is used for each Booster reset, then the request that specified returns on a given reset would send a reply only on those cycles for which that clock event occurred. This is ok, in terms of time scheduling, because the node will be synchronized to start its 15Hz operation after Booster extraction time, so that the waveforms will be immediately available in the Swift hardware memory for linearization and storing into the relevant data streams, after which active data requests will be fulfilled.

The dose calculations should be easy once the linearized waveforms are available.

Alarms based upon the total dose as measured and averaged over a time interval will mean that we may need to support an alarm based upon an upper limit, rather than a nominal and tolerance value. Without adding this generic support, one could have the local application that deals with this waveform data generate a bit pattern, say, that has a bit set for each waveform for which the total dose exceeds some limit. The limits can be specified via a set of dummy channels. This would eliminate the hysteresis logic that is a built-in part of analog alarm scanning. Another possibility would be to add an option in alarm flags to allow skipping the hysteresis logic.

Correlated data problem

It will be important to relate waveforms measured from different nodes. One way that can do this is via a time-of-day stamp. All of these nodes can mark the time-of-day together, all being occasionally synchronized with a NTP server all at once. In the data stream record will be some identifying information that can include the time-of-day as well as the waveform and the Booster reset type.

Booster reset types are recognized by monitoring the clock event bit array that is updated every 15Hz cycle with the clock events that have occurred since the last cycle. It appears that all of the required support for the BLMs can be provided via a local application.

Snapshot protocol

The Acnet snapshot protocol is unsuited for acquiring and plotting 15Hz waveforms. But RETDAT is capable of keeping up with 15Hz. That is why RETDAT is used to collect waveform data in the scenario above. Assuming that the digitize rates can remain the same for all users, there is no built-in limit on the number of users that can share access to this waveform data. The SWFT local application would not be used, which means that FTPMAN could not access these waveforms. But the special application that is to be written to display these waveforms can produce whatever forms of plots are required.

As for data logging waveforms, this can also be done via RETDAT, which would access the data streams to collect the most recent set. Does the data logger support storing and retrieval of waveform data?

It would be possible to add support to FTPMAN snapshot protocol for access to waveforms of the above sort that are accessed via a data stream queue. The listype would be different, and the header structure would need to be known. It would mean that the data stream would be accessed, then the waveform portion copied into a holding buffer for access by the console.

The FTPMAN snapshot request parameters include an event, a delay, a rate, and the number of points. The event parameter could be used to find the appropriate record in the data stream for the specified Booster reset event. If it were not required to await the given event, the request could be fulfilled immediately. But it is more likely that one would have to first wait for the event, then find the appropriate waveform just recorded into the data stream. The delay parameter would be obtained from the hardware register. The rate would also be known from the hardware. The number of points could be whatever the requester wants, up to the size of the waveform record. A new CINFO entry type would have to be designed to cope with this new form of access.

Data stream requests

There is more than one listype used to access data streams. It is desirable to use one that will return the most recent waveform for a one-shot request, but will return ongoing records as they are written for a repetitive request. For event-based requests, the forward-looking listype 50 should be used. But if a one-shot request is used, nothing of interest would be returned the first time. Perhaps this support should be changed to return the most recent records that fit in the requester's buffer.

If listype 51 is used, a one-shot request will return the most recent waveform already in the queue. An event-based request will not return anything until an event occurs, when it will return the previous event. If the buffer is large enough to hold two waveforms, it will return both the most recent that was measured before the request started, then the one that was measured on the indicated cycle. Perhaps an event-based request, since it does not return an immediate reply, should act the same as listype 50, so that it await the event, then return the fresh waveform just recorded in the data stream.

Another scenario

Request waveforms to be returned on an event, so that data is not flowing to the console at 15Hz. This requires, however, that all data measured on a single cycle be returned on that cycle, as anything extra would not be returned on the following cycle. The amount of data measured on one cycle is, assuming 8 signals per node, $880 \times 8 = 7040$ bytes, per node. If there are about 10 nodes, this may be more than the console can handle. But it is not too much for the front end to collect. Do we need to start/stop the collection of data?

Suppose there is a separate data stream for each waveform, and all Booster resets are recorded in each one. If each waveform is about 1K bytes, then a 32K-size data stream could hold 32 Booster beam events. This could be about 4 seconds of time, if we assume Booster is operating at a 7.5Hz average rate.

If we kept a separate waveform for each Booster reset and waveform combination, we could keep a longer history before overwriting. One could more easily focus study on a single reset type. Of course, there would have to be many Acnet devices defined for this case, not only many data streams.

Console problems with 15Hz

Consoles cannot reliably collect 15Hz data. They run on a nominal 15Hz cycle, but they are asynchronous with any front end. They actually schedule a 60 ms cycle, followed by two 70 ms cycles in an attempt to average out to 15Hz. This means that it is possible to miss some 15Hz data, in the case that two replies arrive between successive application executions. If consoles provided a callback facility, in which the application could be invoked when fresh data is available, one could get reliable 15Hz data. But they don't.

What can be done about this limitation? The console can collect reliable 7.5Hz data. If the application calls for the data at its approximate 15Hz rate, it will be able to ask the data acquisition support software whether the data it collected is new. So, how can we queue waveform data in the front end so it can be sampled at 7.5Hz by the console application?

Requesting data to be returned on an event would only result in reply data when there is something real to provide. But there is no opportunity for queuing this way. Assume that the request rate is specified to be 7.5Hz. The application should call for the data every 15Hz cycle as best it can. About half the time, there will be no new data there. But at least, the data won't be missing. For this to really work, one must assume that the data is not being filled more often than 7.5Hz, on the average, within the wrap period of the data stream queue.

Single detector

Take a case of displaying only waveforms from a single detector at rates up to 15Hz. We need only 440 words of data, or less than 1KB. If we make a request for 2KB of data at 7.5Hz, the application will be able to see all of the data, even if it occurs in a burst at 15Hz. If we make a request for 1KB of data, it can see all the data, as long as new data doesn't occur more often than 7.5Hz on the average, or as long as we don't get too far behind so that the data stream, which is really a circular buffer, wraps. In this scheme, a data stream might contain waveforms for all Booster resets, so that we need only an Acnet device per detector. Each record should include a time-stamp as well as a Booster reset event number.

A reasonable plot might show a series of successive traces with the vertical offset depending upon the cycle number in the series. Perhaps a pre-pulse might be the start of the series of pulses, such that it would be plotted with the smallest offset. In order to know which cycle of data there is, one would have to interpret the time-stamp accordingly. If the application could plot an entire waveform in less than one cycle, one could see the data shown in nearly real time.

More detectors

Viewing more detectors requires collecting much more data. How many detectors can the console system handle? The application might request 1KB from each device. If there are 8 devices, there may be 8KB to collect on a given 15Hz cycle. It may get behind, but the use of the data stream queues to provide buffering should help. Once the data has been collected from all detectors for the number of cycles of interest, the 8 device data requests can be canceled.

Worse yet is collecting such data from many nodes. This can make the application run even further behind. But it cannot run too far behind lest it begin to miss data.

Dose calculations

By reducing the data to 10–16 values that represent the dose at a certain point in the cycle, one can get an overall feel for beam loss without processing an exorbitant amount of data. One may want to have the distribution of beam losses plotted for each such time around the ring. The Y-axis can indicate beam loss and the X-axis detector position. Do this for 10–16 times, and one gets a feel for what part of the cycle produces the most beam loss.

Ring-wide snapshots of the dose measured at one point (out of 10–16) in the cycle can be plotted with vertical offsets that depend upon the time slot. This will show how the loss distribution varies over the 33 ms acceleration cycle.

A dose calculation requires first deriving the loss during a time slot. This should be the difference of two integrated doses that bracket the time slot. Although the difference in later time slots may be significant, the values may have poor accuracy, because the two log values may have been very close. For example, a difference of 5 lsb's of log readings may indicate a large loss difference in the higher range, but its accuracy will not be good.