

BLM Correlated Data

Capture waveforms on successive cycles

Fri, Nov 1, 2002

Booster BLM waveform access can represent a heavy load when collected at 15 Hz. There are 70-odd BLM waveforms, divided up among 8 IRM front ends with at most 12 waveforms coming from a single front end. Each waveform includes 500 data points covering 40 ms of time, enough to include the entire acceleration cycle. A front end sending 12 waveforms, amounting to 12000 bytes, requires about 10 ms of 10 Mbps ethernet bandwidth. With 6 times as much data as that, we are up to 60 ms of ethernet bandwidth, which is a serious piece of a 66 ms cycle. And these times say nothing about processing time on either the server side or the client side.

Now in real life, collecting such data at 15 Hz causes problems, because the Acnet VAX-based console system does not operate synchronously with the accelerator 15 Hz, which means that an application might not notice some waveforms, because two of them may arrive between successive times that it could check. Recognizing this as a problem, a “time-stamped data” support was added, so that the front end would deliver data from two successive cycles at 7.5 Hz, or every other 15 Hz cycle. The application would still have to look for new data to be available at 15 Hz, but would only find new data about half the time. This scheme was designed to support 15 Hz data collection reliably to an Acnet client. But it may be considered overkill, when one wants to monitor beam-related 15 Hz cycle, but not those one which no beam is accelerated. What would be desirable is a scheme for collecting such time-stamped data, but have it collected only on cycles of a selected event. The current scheme replies with one or two sets of data—one for the first-time reply, two thereafter. Limiting the production of such data to those cycles of a selected event could mean that replies containing one set of data may appear when only one of the two sets is a cycle of that selected event. But how would such data collection be specified? It would seem that the client side would have to support some flag bit in the FTD that could signal such replies.

Right now, the time-stamped data “trigger” is in part the specification of 7.5 Hz. But that leaves out the possibility of specifying a clock event. If one uses an event-format FTD, one cannot say data is to be returned at 7.5 Hz, too.

Assuming that nothing can be done on the client side, possibly because all client-side effort is devoted to supporting the new Java system, what can one do? First relax the requirement for reliable data acquisition of 15 Hz data—events can of course occur in bursts at 15 Hz—but retain the need for correlated data, in which data coming from different front ends can be identified as to the 15 Hz pulse on which it was collected.

If the client makes a request for event-based data, no time-stamp support is available, which just means that one will get replies whenever the event occurs. When the events occur at 15 Hz, it is possible that some of these replies, say the second consecutive one, may be missed. But we may get enough to make it possible to make useful measurements, as long as we don't lose the ability to process correlated data.

If, along with each request, a device is included besides the waveform devices that is the copy of the cycle number that is the key to recognizing correlated data across front ends. The client can then receive a set of waveform data and know what cycle number “time stamp” is associated with it.

In addition, the client program should make a 7.5 Hz request for the B:BREVNT device, which delivers the Booster reset event number appropriate for each cycle in a time-stamped fashion. This allows the client application to monitor the reset events and know what cycle number is the correlation key to each event cycle.

As an example, suppose the application wants to select out the waveforms corresponding to the first two 0x1D events in a series. It will monitor the waveform data returns at 15 Hz, capturing each waveform into a buffer along with the associated time stamp cycle number. At the same time, it will monitor the reliable B:BREVNT values to note when it has seen the first two events in a series. When it sees the first two 0x1D events following at least one non-0x1D cycle, it knows which two consecutive cycle numbers are of interest, and it can see how many of them it has collected. If it has most of them, it may be able to learn something useful from the data. But it will know which ones it does not have, those that “fell through the cracks.” And it will know that all the waveforms it has that have the same time stamp were measured on the same 15 Hz beam cycle. The disadvantage is that it is not perfect. The advantage is that it will know exactly in what way it is imperfect.

As the application monitors the replies of waveform data at 15 Hz, as best it can, it may find that most of the time, there is no new data to be had. Let’s say it first asks for the cycle number. If it receives status that says there is no new cycle number value, it can omit trying to “get” the waveform data, which may save some time.

So, no heavy data will arrive at the console unless there is a selected event cycle, and the application should not have to capture uninteresting data into its buffers. This can reduce the load on the client side greatly compared to continuous 15 Hz delivery of waveform data, even if it actually arrives in double-size bunches at 7.5 Hz. Isolated event cycles would have no problem at all being collected completely. The problem occurs only for sequential cycles of the selected event.

Different approach

Suppose the objective is to monitor beam losses so as to see where the weaknesses are; *i.e.*, where are the unusual losses occurring within the cycle? If the client requested 7.5 Hz replies containing the 40 sets of double precision accumulations from all the BLMs, one could use this data as a coarse reading of where unusual rises in the BLM integrator signals occur. That could pin down a time within the cycle which it is interesting to examine in more detail. (Assume here that one is studying a stable situation, not an isolated unusual occurrence.) Then request that part of the BLM waveforms that covers the time of interest in order to determine how the unusual losses are distributed around the accelerator ring.

Let’s look at the numbers. Forty sets of 8-byte doubles are 320 bytes. If we take 72 BLMs, this amounts to a total of 23K bytes. And one would receive this at 7.5 Hz, so 46K bytes arrives at the same time. (Actually it may be more like 23K on each cycle, since the 7.5 Hz requests may not have been initiated on the same 15 Hz cycle, so that the replies from different front ends may be out of phase.) In 10 Mbps ethernet bandwidth terms, we are looking at 18 ms per cycle, on the average. Maybe this is doable. Now look at the detailed examination problem. If one chooses to examine 3 ms, say, this means about 40 data points per BLM, which is only 80 bytes. This is only one-quarter of the load of monitoring the 40 doubles.

This approach is a two step process: first determine the time of interest within the cycle, then get a picture of the losses everywhere during that time. If one tunes to reduces the losses

during that time, one could imagine making them worse at some other time, although it would seem more likely that the losses would perhaps move to another location in the ring, but at the same time. (Maybe not; I am not an expert.) But an iterative approach might make some improvements. And in this approach, no data should be missing. It's just a scheme for reducing the load for the client compared to that which results from collecting all BLM data all the time, taking advantage of the millisecond breakdown provided by the double-precision accumulations of the integrator signals.