BLM Comfort Display Data

Sketches

Thu, Oct 19, 2000

The BLMS local application currently computes long term beam losses for each of about 70 BLMs for each Booster reset event. With 10 reset events, this is a total of about 700 "dose" accumulations. Each accumulation is assigned an analog channel that returns a single precision floating point value. The accumulation is summed over a period of time, updated several times during that period. (For example, the sums might reflect the last hour, updated every 5 minutes.) We must add to this a breakdown of the beam losses throughout each Booster acceleration cycle in steps of one millisecond. This implies a total of 70*35*10 (= 24500) summed values. Within a single node, there may be typically 12 BLM signals, so that such a node has to deal with 4200 sums If the sums are 32-bit floating point values, 16800 bytes would be needed for a node to support 12 BLMs. But how can these values be most easily dealt with? This note is an attempt to reach a conclusion on some of the details.

Each BLM will have 35*10 values, with 35 points for each BLM for each reset event. This data should not have to be accessed frequently. Each set of millisecond single precision fl;oating point sums occupies 140 bytes; a complete set of sums for all clock events therefore requires 1400 bytes. Imagine that we have a set of Acnet array devices. Shall there be one Acnet array device per BLM? The offset value might be used to select any set of the 350 sum values.

In addition to maintaining sums of losses, we also need to keep a set of 35 values for the beam charge signal that can be used for normalization purposes.

Finally, sums of the number of reset events should be maintained. This is only 10 sums, but they should be in floating point so that range is not a problem. Single precision floating point only maintains 6–7 digits of precision. At 15 Hz, one can count to a million in 66666 seconds, or somewhat less than a day. Do we need to maintain double precision floating point sums? If we don't, after about a week's time, the sums may no longer increment.

Single precision floating point is supported as a raw data type, but there is no current support for double precision floating point. Still, it may be possible to build a table of such data in memory, so that it could be accessed occasionally by a client application.

Every 15 Hz cycle, what does BLMS have to do to fulfill these requirements? For each BLM in the node, up to a maximum of 16, evaluate 35 differential loss values across the 35 ms acceleration cycle. Accumulate these values into the sums for that BLM for the current reset event. For the single signal of beam charge, accumulate its values (not differential) into sums according to the current reset event. Finally increment the counter for the current reset event.

A client that reads such sums will likely want to compute differences, in order to show what accumulations occurred during the previous minutes, say, or for any time interval.

Conclusions:

Let the sums be maintained in double precision floating point. This means that there can be no overflow, such as might be the case with 32-bit integer accumulations, and there cannot be a loss of significance, as might be the case with single precision floating point. Each double occupies 8 bytes, so that using 10 possible events with 35 points, we need 2800 bytes per BLM. In a single node, we should not have to support more than 16 BLMs, and it will more likely be only 12 BLMs. The total space required would therefore be 2800*12 = 33600 bytes. Even if we

allow for a full 16 BLMs, and an additional 2 clock events, and one more point (for a total loss), the total memory required would be 36*12*16 = 55296 bytes, or 2048 bytes shy of 56K bytes.

Since differences of these sums will always be used for interpretation of the beam loss data, it would be better if the sums are truly long term sums that are not normally reset at all. The sums may be archived using the Acnet datalogger, and one may want to compute sums over very long times; therefore, it would be convenient not to reset the sums. This raises the question of how to prevent it if the power goes off, or even if the IRM resets. The solution to this is to maintain the sums in nonvolatile memory. When BLMS starts, it knows that the sums are already active. When it exits, for whatever reason, perhaps to update the program, it knows that the sums will be retained.

Looking for a place to house these beam loss sums in nonvolatile memory, the `ADESC` table in an IRM is located at `420000–42FFFF`, when configured for 1024 analog channels, as it always is. The `TRUNK` table is located at `43E000–43FFFF`, allowing for the maximum of 8 trunks from 9–16. The loss accumulations can therefore be housed in the 56K bytes at `430000–43DFFF`.

The extra 2048 bytes can be used for keeping the date and time of clearing the sums, and perhaps the sums of the various clock events for which losses were accumulated during the same period of time. For support of sums of 12 clock events, we need 12 doubles. These occupy 12*8 = 96 bytes. The sums of the beam charge signal can be housed in the same way that an extra BLM can be supported. The only difference is that the beam charge readings should not be converted into loss units nor differenced for each millisecond. The beam charge information is presumably only available in one node, so that the beam loss summations in all nodes should be reset at the same time, along with the beam charge sums and the clock event sums.

For 36 doubles, we need 36*8 = 288 bytes. This is 0x120, a fairly even number. Using 12 clock events per BLM, we have 288*12 = 3456 bytes, or 0xD80. An Acnet device for each BLM would point to one such two-dimensioned array of 36*12 = 432 doubles. Using memory access to these values, the SSDN will have to use multiples of 0xD80 bytes offset from 0x430000.

BLM#	*Address*
0	0x430000
1	0x430D80
2	0x431B00
3	0x432880
4	0x433600
5	0x434380
6	0x435100
7	0x435E80
8	0x436C00
9	0x437980
10	0x438700
11	0x439480
12	0x43A200
13	0x43AF80
14	0x43BD00
15	0x43CA80
end	0x43D800

As an example, an SSDN for the BLM #7 in node06C3 would be:

```
1D12 06C3 0043 5E80
```

Listype 29 (0x1D) specifies memory word access via memory address. The offset flag is set to enable the use of the offset word to modify the acual address referenced. This is needed by an application to read a part of the 3456-byte structure that allows for 12 sets—one per clock event—of 36 double precision floating point sums.

The end leaves 2K bytes before `0x43E000`, where the `TRUNK` table currently resides. In order to reach a specific clock event's millisecond loss measurements, use the following offsets.

Event	*Offset*	*Offset/hex*			
11	0	0x000			
12	288	0x120			
13	576	0x240			
14	864	0x360			
15	1152	0x480			
16	1440	0x5A0			
17	1728	0x6C0			
19	2016	0x7E0			
1C	2304	0x900			
1D	2592	0xA20			
xx	2880	0xB40	yy	3168	0xC60
end	3456	0xD80			

Using the above offsets with the SSDNs that apply for each BLM allows access to the long term double precisions beam loss sums for a given clock event. Note that an array element cannot be addressed on the parameter page because Acnet does not support 8-byte values.

When accumulating the beam losses for every millisecond, given that the digitization rate is 12.5KHz, or 80 μs between each point, we must skip along every 12 or 13 digitized values, althernately, in order to average out to 1 ms sampling. If the loss difference between successive samplings is negative, assume zero loss for that millisecond.