

Recent 15 Hz Samples

From data pool

Tue, Nov 16, 2004

Each front end node maintains a 15 Hz data pool that includes all data updated at 15 Hz. From this data, requests are fulfilled, and alarm scanning makes reference. Requests can be made for 15 Hz data replies, but some clients have trouble reliably collecting such reply data; they especially have problems correlating it with data obtained elsewhere. This note develops a method for returning 15 Hz data at slower rates, including a cycle counter time stamp.

A similar scheme was done for collecting Booster waveform data, in which an Acnet RETDAT protocol request made for 7.5 Hz replies was interpreted as a request for two cycles of reply data plus a 4-byte header that included a cycle counter time stamp. This allowed such waveforms to be returned reliably to an application that watched for these replies at an asynchronous rough 15 Hz rate. About half the time, no new data is found, but at least all data will be seen; in addition, all data is tagged with a cycle counter to serve as a correlation key across replies from different nodes. Not only waveforms, but 15 Hz readings can be so collected. The client only has to accept 7.5 Hz replies.

Another idea is to keep multiple copies of the 15 Hz data pool of analog readings in a separate circular buffer. For example, an array of 32 successive copies of each analog channel can be stored somewhere. A new listype can support access to this table via the usual channel number ident. Then one can make a request for 2 second replies and still get all the data. If we include a 2-byte header for the cycle counter, every piece of data can be correlated with every other piece of data similarly obtained. The client has a data management job to design and implement, but at least it has access to all the data.

The client that desires to see all the data should make a sensible data request with the appropriate number of bytes requested. To request data to be returned every 2 seconds, or 30 cycles, the client would ask for 62 bytes, including the 2-byte counter and 30 readings. To request data to be returned at 1 Hz, the appropriate number of bytes to ask for is 32. Note that all data in the 15 Hz data pool can be obtained in this way.

A one-shot request for such data would return the most recent values sufficient to fill the reply buffer. The same is true for the first reply to a periodic request. This means that one can look backwards at data that was measured before the request arrived.

The idea is that the last reading included in the buffer is the most recent one. If one asked for 32 bytes every 7 cycles, then one would get a sequence of the last 15 readings every 7 cycles, meaning that there would be much duplication. One would not be expected to do this, but the time stamp would help sort it out.

Maintain a Cycle table

In order to maintain a Cycle table that includes the last 32 values of channel readings, we need to copy current readings into that table. But there may be many channels that are unused out of the total allotment of 1K or 2K. One idea is to limit the copying to those channels that are in use. A way to automatically notice which channels are in use is via the Name Table. When a name is inserted into the table, usually during system initialization but also as new named channels are added to the system, a bit map can be built that has one bit representing, say, 16 consecutive channels. For 2K channels allotment, this would be 128 bits, or 16 bytes. The space for this can be found in the Name Table header. Then, an RDATA table entry could prompt this copying, or it can be automatically included in the system code. For the IRM, the copy code could include an inner loop as follows:

Assume the following register assignments:

```

D0    inner loop counter
D1    size of ADATA table entry
D2    bit# for sampling bit map within 32-bit portion
D3    32-bit portion of bit map
D4    32*size of ADATA table entry (=512)
D5    outer loop counter of 32
NWCOPY    #channels in block indicated by bit map bit
NWCYCL    #sets of data maintained in cycle table

CPYT      MOVEQ    #31,D5    ;enough to handle 32*NWCOPY channels
          BTST     D2,D3
          IF#      NE THEN.S ;at least one channel used in this series
          MOVEQ    #NWCOPY-1,D0
CPYL      MOVE     (A0),(A1) ;copy data into Cycle table entries
          ADD      D1,A0
          ADD      #NWCYCL*2,A1
          DBRA    D0,CPYL
          ELSE#.S
          ADD      D4,A0
          ADD      #NWCOPY*NWCYCL*2,A1
          ENDF#
          ADD      #1,D2
          DBRA    D5,CPYT

```

If NWCOPY is 16, this logic can handle only 512 channels, so it would have to be repeated 2 times for 1K channels, or 4 times for 2K channels. The main idea here is to reduce the time taken to maintain these copies in systems not completely full.

It would be a good idea not to keep the data in a new nonvolatile memory table, as it's a bit large, maybe 128K bytes. It should be kept in some fixed area of dynamic memory. One suggestion is to use 128K bytes starting at 0x260000. (Ethernet receive buffers currently use 512K bytes starting at 0x280000. This Cycle table would immediately precede that area.)

It would also be better to make this a permanent part of the system code, so we do not have to install a separate entry in the Data Access Table, or install an LA, to get the job done.

Fulfilling requests

The code for fulfilling a request for such data can use an "Internal ptr" format that is merely a pointer to the analog channel's Cycle table entry. Additional information needed to update the user's buffer is an indication of the index used for storing the present cycle's data, plus the current cycle counter time stamp. With the number of bytes requested, including 2 bytes for the time stamp, this is enough to know how to copy the code into the user's buffer.

Acnet interface

Since this is a special listype, one cannot use the same device names as one uses to put a channel's value up on a parameter page. Also, only analog channel readings can be requested in this new way; however, digital status words could be accessed this way, since they are housed within the same 15 Hz data table. And a generic Acnet device could access any channel in a single node, via the offset, assuming the client knows the channel number.

Event-based requests

One might wish to acquire 15 Hz data for a string of consecutive beam events, such as

the 0x1D events that announce MiniBooNE cycles. By making an event-based request using the above scheme, we would get a string of replies at 15 Hz, each of which might contain recent history. There would be a lot of overlapping data, making it not particularly useful.

An alternative would be to “invent” an event within the front end. This we can do, since the occurrence of an event is merely known via a bit set in a bit map table. This invented event could mean that a cycle was detected *following* an event 0x1D cycle. It would not be repeated until there was another sequence of 0x1D cycles. By making a request that is based upon this invented event, one could get the set of 15 Hz data that existed for the last number of cycles, up to 2 seconds’ worth according to the above scenario, ending with the present cycle that was not a 0x1D event. One could then compare data across multiple, correlated 15 Hz beam cycles without worrying that data was missed. And during the time that no 0x1D events were occurring, there would be no replies.

Acnet waveform access

A means of accessing waveforms at 15 Hz from asynchronous clients was done a few years ago for the Booster BLM upgrade project. It is similar to what has been described here, but the client must receive data at 7.5 Hz, where each reply includes two sets of data. The new scheme described here does not permit access to waveforms, but only access to analog reading values. Its advantage is that a client copes with (much) slower replies than 7.5 Hz.

Summary

What does the new scheme accomplish? The Fermilab accelerator operates at 15 Hz, except for the Main Injector and points beyond. To achieve the highest performance of the lower energy components, an increasing level of attention is paid to detailed measurements of accelerator operation. Client computers do not operate in real time; they are asynchronous to accelerator timing. The scheme tries to bridge the gap for ordinary analog readings, anything that shows up in the front end data pool that is updated at 15 Hz. Any client can grab all such data, each of which is tagged in a way to allow correlation across other devices, either in the same front end or in other front end(s). This may allow for precision studies of accelerator performance during a sequence of 15 Hz beam cycles.

Post-implementation note

The above scheme was implemented in the IRM system code. The copying code needs about 1 ms to copy a full 1K-channel data pool. With one third of the channels in use, only 300 μ s is needed. The bit map of channels in use uses one bit per 16 channels. The copying code examines one byte of the bit map at a time, rather than 32 bits. Porting this to the PowerPC is likely to have similar timing due to the 1 μ s access time for each reading word.

The “invented” clock event idea was done earlier for MiniBooNE. Specifically, a fake event 0xFD was invented on any cycle following a 0x1D cycle that is not itself a 0x1D cycle. This logic is provided by a local application called MBEV. To be useful, it should be installed in each front end node that supplies such event-based data. Additional instances of MBEV can be used to mark the end of other sequences of specific clock events, as needed.

Listype 88 provides access to the data in the Cycle table, which is located at 0x00260000. There are 32 words kept for each channel, giving a bit more than 2 seconds of recent history.

After a node is booted, one must wait up to a minute until the first multicast is received, before the cycle counter time stamp matches those found in other nodes. It is node06C3 that delivers this cycle counter data every 256 cycles, or 17 seconds at 15 Hz.