

HRM Waveform Access

RETDAT support

Thu, Mar 6, 2003

The Booster BLM waveforms are supported by RETDAT access, in which the SSDN specifies the usual analog channel number, and the matching CINFO table entry leads to the array of 2-byte data points that are placed there by the Swift digitizer hardware, operating at 12.5 KHz. It is from this waveform array that a portion is copied to build the reply data. (The portion is specified by the length and offset parameters in the RETDAT request packet.) For the case of the HRM Slow Data digitizer, operating at 10 KHz, how could such support be provided via RETDAT?

What are the differences in the waveform data provided by the two digitizers? The Swift digitizers place a waveform in a fixed region of memory, where the trigger was some clock event plus delay known to the hardware. When it is time to build the RETDAT reply, all the data is there, by definition, so it is just a matter of copying the requested portion of the array. For the case of the Slow Data digitizer, the data is continuously recorded into a circular buffer that wraps about every 1.6 seconds. To capture the data, one needs to know a clock event plus delay, then find the first data point recorded after that time. The consecutive 10 kHz data words can then be copied out to fulfill the request.

Alternatively, one might suggest building up arrays of waveform data at fixed addresses routinely, every 15 Hz cycle. But this is a lot of copying that might be avoided, when there are no requests actively seeking such waveform data. On the other hand, if we assumed that the users would want all of this data at the same time, then that means we must have enough time in one 15 Hz cycle to do it anyway. In addition, once we have copied the data into fixed waveform arrays, they would be in fast memory, so that multiple requests would not cost additional slow memory accesses. We should tally up what this would actually cost, depending on the total number of waveforms to be maintained.

What kind of internal ptr structure could be used to facilitate this? Keeping a memory address would not work, since it would be changing all the time. At the time one is to return data, a clock event must somehow be already known, analogous to the clock event in the Swift digitizer hardware. Given that clock event, one can find the time stamp corresponding to its most recent occurrence. Then find the time stamp that most nearly exceeds that one in the Slow Data time stamp array. That identifies where the waveform logically starts. Add the offset, considering it as a byte offset into an array of words. Copy out consecutive words, separated by 128 bytes, into the reply buffer.

At first blush, this might seem like a big job, since the time stamp array has 16384 elements. But we can monitor the hardware to know where it is "now." Consider that the relevant clock event is 0x0C, the reliable 15 Hz event that occurs at Booster reset event time. And we can know how far back in time the 0x0C event occurred. Therefore, we can know pretty well where to look in the time stamp array that corresponds to the time the event occurred.

If we are copying many channels of data, how long can a straight copy take, assuming 1 μ s per access? Suppose, we copy 400 points at 10 KHz, which is 40 ms of waveform. If we have 64 channels, we must copy 25600 words, which might take 25.6 ms. If we can copy longwords, we can cut this time to 13 ms, approximately. The system could probably stand this, but another factor of two may not be suitable. All of this argument depends upon the slow access to this memory requiring 1 μ s per access.

Some measurements were made of the access times to the Slow Data memory, using the Data Access Table diagnostics in low memory at 0x4800. these measurements show that a word (16-bit) access required 1.1 μ s, while a longword (32-bit) access required 1.6 μ s. This implies that accessing this memory by longwords helps, but by not as much as a factor of two. If we use these results to revise the above numbers, we have 25600 words of data accessed in 28 ms if accessed by words, or in 20 ms if accessed by longwords. This would indicate a maximum of 64 channels might be supported, but 20 ms out of each 66 ms cycle would be required to copy the data into fast memory for processing.

Some other memory accesses were timed in the same way. For accesses to the Digital PMC registers, at 0x49100000, the times were 1.6 μ s/access. For the VMEbus-based Quick Digitizer memory, accesses were 1.1 μ s, the same as for the PMC board Slow Data memory. Accesses to nonvolatile memory were 0.9 μ s. For DRAM, accesses were about 0.06 μ s.

For comparison, the IRM system, when accessing either DRAM or SRAM, shows access times of 0.55 μ s. Again, the measurement includes the loop time in the support code for the 0x2A (memory copy by words) Data Access table entry type, which also includes writing to DRAM, so the numbers represent more work than simple access times. But the results do seem to indicate that there is no difference in access time between DRAM and SRAM.