

# HRM Time Stamp Management

*Interim solution*

Sat, Jan 4, 2003

As an interim solution to providing support for event-relative requests for Slow Data in the HRM, a proposed scheme maintains an average offset between the event time stamps that come from the Digital PMC board and the Slow Data time stamps. But the code to do this must be located somewhere and the related data structures also. This note discusses how this might be managed.

Measuring the offset between the two time stamps can be done by first reading the current DPMC time stamp register. Access to the latest time stamp value from the Slow Data hardware is done by monitoring the Slow Data block count register, then accessing the current time stamp that is stored into the time stamp array at the start of the 64-channel digitization sequence. The DPMC time stamp is good to 1  $\mu$ s, but the latest Slow Data time stamp is only good to 100  $\mu$ s; therefore, the offset difference is only good to 100  $\mu$ s. But since the 10 KHz timing that controls the digitizations is independent of the 15 Hz accelerator timing system, an improvement should be possible by averaging the above determined offsets. If 256 offset determinations are averaged, for example, we may be able to improve the accuracy by a factor of 16. If 1024 are used, we may improve accuracy by a factor of 32, which brings us to 3  $\mu$ s.

A new entry in the Data Access table is type 0x32, which invokes the HRMAD routine that is used to copy 15 Hz samples of the Slow Data channels into the local data pool. This may be a good place to put the code that computes a time stamp offset and performs the averaging computations. But in order to do this, there must be some persistent memory available. Also, bear in mind that there could be more than one Slow Data module for which such support is needed. In such a case, there will be more than one HRMAD entry. We can probably safely assume that whenever there is a Slow Data module, there is a corresponding HRMAD entry in the Data Access table.

The current design for the HRMAD entry allows for 3 spare words. A 32-bit field could be used for keeping a pointer to allocated memory that can house whatever context is appropriate to do this job. In order for this to work, part of the InzRDATA routine should make sure this 32-bit field is cleared when the system is initialized.

The context block should include the average time stamp offset value and the accumulation of a new average and the associated counter. It can also hold some diagnostics so we can get a good understanding for how much the computed average offset varies. This might include a circular buffer of the last 16–32 averages, for example.

But how does code such as RFTData gain access to the latest offset value? It needs to find the appropriate RDATA entry so it can obtain the context memory pointer and thereby get the average offset. With the current support for listype 82, an internal pointer structure is used that is 9 long words in size. The first of these long words is used for server requests, but it is not used for nonserver requests. One possibility is that the FTData routine, whose responsibility is to build the internal pointer structure during request initialization, could perform the appropriate search of the RDATA table to find the context memory pointer, then save it in the first long word of the internal pointer structure. The key for the search is the pointer to the Slow Data block number register.

The RFTData routine does not interpret the first long word as a pointer to “external data” unless a flag is set in the flags field. In the nonserver case, that flag bit is not set. So, this proposal makes use of that first long word that would otherwise not be used, in the nonserver case.

An alternative would be for FTData to store a pointer to the most recent average time stamp offset rather than a pointer to the entire context block used for the averaging computations. This would save RFTData from knowing about the structure of that context block, but it seems it is not asking too much for it to know about one more structure. (What’s one more include file, anyway?) When the new HRM design is in use, another method will exist for obtaining a better time stamp offset, and this will mean that the RFTData

code will need to be modified.

### *Post-implementation*

After writing the code, here is the present situation. There is an `SDRoot` array of entries located in low memory, each of which is filled automatically as `HRMAD`-type Data Access Entries are encountered. The key to the entries in this table is the address of the Slow Data block counter. Contained within each entry is also a pointer to an allocated structure that the `EvtHRMOffset` management routine, called by `HRMAD` processing, uses to come up with an average offset between the `DPMC` time stamps and the Slow Data time stamps. A copy of the latest such average offset is also contained within this `SDRoot` entry. Accordingly, the `FTData` routine calls `FindSDRoot` to install a pointer to this average offset field into the first long word of the internal `ptr` structure. This is used by `RFTData` whenever it needs to provide time stamps for replies to requests for which those time-stamps are to be delivered relative to a selected clock event. It will also be used for the `FTPMan` support code that needs to provide event-relative time stamps.

In summary, two new globally accessible routines are as follows:

#### **EvtHRMOffset**

Called by the `HRMAD` routine to manage the determination of the offset in microseconds between the clock event time stamps obtained from the `DPMC` and the `HRM` low Data time stamps. Note that there is a `HRMAD` entry for each `SD` module.

#### **FindSDRoot**

Called by any routine needing access to an `SDRoot` table entry. The `FTData` routine needs it to obtain a pointer to the `avgOffset` field that `EvtHRMOffset` keeps up-to-date. `EvtHRMOffset` itself also calls this routine. If a matching entry for the specified `SD` block count register is not found, a new entry is automatically created so that subsequent calls to `EvtHRMOffset` can properly manage the offset determination.

To install a new `SD` module in a front end, an `HRMAD` entry should be installed in the Data Access table. A `CINFO` table entry should also be added to identify the range of 64 channels that are mapped to the `SD` digitizer. The latter is needed by the listype 82 support for Fast Time Plot data used by the Macintosh Parameter Page, as well as the support for the `FTPMan` protocol.

### *Post-installation*

After running the code for the first time, we learned that the two time stamps for which an average offset is to be calculated are derived from different crystal sources. One could see that the drift between them is about 750 usec over 68 seconds. The scheme cannot work well in this case. (The real hardware will use the same crystal—from the Tevatron clock carrier—to derive the 1MHz counter time stamps.) But an improvement can be made by extrapolating from the previous measured difference between that last two average offset determinations. The average offset value kept in the `SDRoot` entry can be updated every cycle in order to keep up with the drift between the two time stamp crystal frequencies. In this way, the code in `RFTData` is not affected.