

HRM Support

Ruminations

Tue, Oct 8, 2002

Development of the new HRM (HotLink Rack Monitor) hardware is nearing completion, and software support implementation must be designed. This note reflects developmental plans.

Hardware

The HRM hardware is broken into two parts, one a PMC module that is connected to the host system, say, a PowerPC MVME-2401 module, and the other a remote I/O interface box, where the analog and digital I/O signals are connected. The link between these two parts is a HotLink hardware serial protocol that operates at either 320 Mbps or 160 Mbps. To the host computer, there is a large area of memory (6 MB) that holds all relevant data and registers. During operation, the remote box sends messages via HotLink that are interpreted by the PMC board and the data placed into this memory. Every 100 μ s, 64 A/D channels are delivered (referred to as slow data), as well as 64 bytes of remote register data. In addition, an 8-channel snapshot of waveforms sampled at rates up to 10 MHz, referred to as snapshot data, is also delivered into memory. There is a potential addition that could support data measured every 10 μ s also delivered into memory. The layout of the memory is as follows:

<i>Offset</i>	<i>Size</i>	<i>Meaning</i>
000000	2 MB	Slow data, 64 channels at 100 μ s/point, 16384 sets
200000	2 MB	Fast data, 64 channels at 10 μ s/point
400000	256 KB	Snapshot data, 8 channels at up to 10 MHz sample rate
440000	128 KB	unused, except for last 4 bytes, Snapshot timestamp
460000	64 KB	Slow data timestamps (16K x 4)
470000	64 KB	Fast data timestamps
480000	8 KB	Remote register cache (64 bytes used)
482000	56 KB	unused
490000	448 KB	unused
500000	1 MB	Error count and status registers (16 bytes used)
600000	2 MB	not used

This hardware has a lot in common with the 1 KHz digitizer used by IRMs. The HRM software support is likely to share much of that common code, or at least, common logic. Instead of 512 slots, or sets of data, there are 16384 slots in the HRM. This means that the size of the circular buffer is large enough for holding 1.6384 seconds of data in the HRM, not 0.512 seconds as for the 1 KHz digitizer. The HRM time per slot is 100 μ s, not 1000 μ s.

The CINFO entry for the 1 KHz digitizer looks like this:

```
1003 0100 FFF5 8300 Size=16, type=3, regAddr=FFF58300
7300 0000 0040 D800 memAddr=73000000, slotTimes=0040D800
```

Will another similar entry be needed for this new 10 KHz digitizer? In terms of a type 3 entry, it would have the following analogous form:

```
1003 0100 5048 0000 Size=16, type=3, regAddr=50480000
5000 0000 5046 0000 memAddr=50000000, slotTimes=50460000
```

Or, using a new type, the format might be simply:

```
0805 0100 5000 0000 Size=8, type=5, pmcAddr=50000000
```

Everything needed is within the PMC memory space, so one base address is enough.

It is likely that the 1 KHz digitizer will be supported only in IRMs, and the 10 KHz digitizer will be supported only in the PowerPC systems. This is because the IRMs have IP module support, but not PMC, and the PowerPCs have PMC support, but not IP support. Sure, one could use an IP carrier board in a PowerPC system, or one could use a PMC carrier board in an IRM, but why would one do that? This opens up the possibility of converting the 1 KHz support code in the PowerPC systems to work at 10 KHz, rather than try to support both digitizers in both systems.

On the other hand, it may be useful to consider installing a PowerPC CPU in an IRM crate. The system would then be expected to support the 1 KHz digitizer via an IP carrier board.

The listype 82 support that is designated for collecting “moderately fast” digitized data was modified during the PET era so that the slot time can be obtained from low memory, rather than merely assuming it was 1000 μ s. (This was done because PET wanted this digitizer to operate at 360 Hz, via external trigger, not 1000 Hz.) This may make it easier to support the 10 KHz case, as that low memory value can be set to 100—or it can be measured, by merely computing an average of the differences in successive time stamps for the various slots.

Digital I/O

The interface specifies two sets of registers, one for 8 input bytes and one for 8 output bytes. But accesses to these bytes are by words only; only the low byte of each word is used. When one writes to an output register, the PMC board sends a special message to the remote box. The next time the digital input data shows up via the normal 100 μ s update, it is overwritten with the latest values, which includes a readback of the output data for those bytes that are configured as outputs.

When testing the word-access-only characteristic, it seems that byte accesses to digital I/O do work, after all. As for accesses to other registers, such as the slow data block counter, one may not want to make byte accesses, since it can change at any time.

But if we set the BADDR table entries to point to the digital output register addresses, we will see the digital data last written to those registers. The same data shows up in the digital input register. This is ok for digital output, but we will not see digital input this way.

If we set the BADDR table entries to point to the digital input register addresses, it will be fine for input bytes, but not for those configured as output bytes. Do we need to edit the BADDR table entries according to whether the bytes are configured for input or output?

When setting a digital output bit, and the digital output register byte is addressed, one must first read the byte, modify its value in a register, then write the byte. But it is possible that a new value may be overwritten before the write takes place. Even if one had just set another bit but had not yet received the reflected value, this is ok, because the output register byte value will already have that bit set.

Listype support for Fast Time Data

Studying the support used by the former 1 KHz digitizer, it appears that the “internal ptr” has a rather complex format. The 3-word ident format for listype 82 is as follows:
node#, chan#, event#

In the event# word, there is the possibility of up to 3 flag bits in the upper bits of the event# word, as the event# needs only 8 bits.

The corresponding internal ptr looks like this:

<i>Bits</i>	<i>Size</i>	<i>Meaning</i>
31	1	Ext answers bit. Remaining 31 bits are ptr to answers already collected.
30–28	3	Flags. Only ms bit is used, which enables group event logic
27–24	4	Bits 9–6 of original channel#. Only the ls bit is used to identify which of two possible 64-channel 1KHz digitizer interfaces is referenced.
23–16	8	Event#
15–0	16	Byte offset into 64KB circular buffer. Modified as answers updated.

In order to support the new HRM digitizer, we need more bits for the offset into the circular buffer. We can pick up one bit since this byte offset is always even. The HRM digitizer memory is 2 MB, or 1 MW, so we need 20 bits to specify this word offset. So a new plan for the internal ptr is as follows:

<i>Bits</i>	<i>Size</i>	<i>Meaning</i>
31	1	Ext answers bit.
30	1	Event group flag
29–28	2	Indicator of which 64-channel interface is referenced.
27–20	8	Event#
19–0	20	Word offset into 2 MW circular buffer. Modified as answers updated.

It's a tight fit, but this allows for only one flag bit, the event group bit, plus 2 bits for supporting up to 4 different HRMs. If it should turn out that another flag bit is needed, we could reduce the support capabilities to only 2 HRMs in one front end. It is unusual to find too many analog channels physically available at one location. Almost no IRM has been installed with 128 channels. What is typical is that when another block of 64 channels is supported, it is done with another IRM. But the situation with HRMs is quite different, since each HRM may be in a different physical location, yet all connect to the same front end. This is why the idea of using two bits to specify which IRM seems appropriate, and thereby limiting the flag bits to the only known flag bit, that which enables group event logic.

Ok, suppose we have a 2-bit field to indicate which HRM we are talking about. Given a 2-bit value, how do we get the relevant parameters? The 2-bit value relates to the 2 bits of channel number above the 6-bit channel# within the 64-channel set. If we have 4 HRMs, say, we might use base channel numbers such as 0100, 0140, 0180, and 01C0. The IRM software makes an implicit assumption about the channel#s in the range 0100–017F, relating this range to the two possible 64-channel blocks supported by IndustryPack slots d and c, respectively, on the CPU board. We can get a bit more flexibility by requiring that bits 7,6 of the channel number identify each connected HRM, without restricting the higher-order bits. But this still begs the question of how we get the relevant parameters given a 2-bit value. We can set up a volatile memory copy of CINFO information to facilitate looking it up. This means we must require a CINFO entry for each HRM, which is probably a good idea. There would be no default addressing, say, about the first HRM. The logic that copies CINFO entries into volatile memory can also check for these HRM-style entries and cache them into the 4-entry table for convenient reference by the listype 82 read routine. The other way we learn about an HRM might be a type 0x28 entry in the Data Access Table. This is the entry that specifies copying data from the circular buffer memory to help update the data pool. In the IRM, this entry is necessary to causing initialization and calibration of the 1 KHz interface.

It is probably desirable to define a new `CINFO` entry type for the HRM. The listype 82 support could check for a `CINFO` entry for the given channel range to determine whether it should support the 1 KHz or 10 KHz digitizer. For a user making a listype 82 request, it should not matter what rate the digitizer works. The user defines the sampling rate implicitly via the choice of buffer size and reply period.

A 4-entry table, `HRMTable`, can be defined to hold the base addresses of the existing PMC modules that connect to each possible HRM. These addresses will be found in the `CINFO` table entries, so the `HRMTable` can be automatically populated according to what is noticed during `CINFO` copying logic. Alternatively, this could be done via routine scanning of the fast memory copy of `CINFO`. In any case, it is a good idea to have this table populated properly as soon as possible following system reset, rather than at some leisurely time later.

If listype 82 is to support either 1 KHz or 10 KHz digitizers, how can the single read routine know which it is? It only has the internal ptr to access. And it needs to know, because the format of the internal ptr is different for the two cases. Do we need to restrict support in one front end to 4 sets of 64 channels, be they 1 KHz or 10 KHz or some combination? In that case, the HRM table could have an entry that specifies 1 KHz, too.

A related document is called *Larger Internal Ptrs*, and it explores how to support them. With this kind of support, we could use 8 bytes per internal ptr for this case and thus include more information. This would allow support of more flag bits or more HRMs. It would also allow for an easy way to support both 1 KHz and 10 KHz digitizers with listype 82. The second longword could hold a ptr to the last word used in the circular buffer. We could also support any number of high order bits for the channel number. Covering a possible range of 2048 allocated channels, we would need 128 entries in the `HRMTable`.