

IRM Digitizer Hook

Keeping up with 1KHz

Thu, Oct 14, 2004

The IRM 1KHz digitizer collects 64 channel readings every millisecond and deposits them into a 64K circular buffer that holds 512 such sets of data. An interrupt occurs at the end of each set that allows for time stamping the set. This note describes a means of allowing additional software to be performed during this 1KHz interrupt.

In the ECool system, there is a need to monitor some 30 analog signals at a 1KHz rate. If any signal is found to be out of bounds, a digital control line should be asserted. Consider allowing a local application to supply a routine that can be invoked at this rate by the interrupt code. The LA would set up the appropriate environment, then plant a pointer to such a routine in low memory by invoking a library function. If the local application terminates, another function can remove the pointer.

It will be important to make the routine that is invoked from the 1KHz interrupt as efficient as possible, even allowing it to be written in assembly code. To that end, consider an example of such code that would scan through the latest readings of some set of signals and determine whether any is out of bounds.

What sort of inner loop might accomplish the objective?

```
D1= loop count - 1
A0= ptr to consecutive pairs of (lower, upper) limits
A1= ptr to latest set of 64 channel readings

LOOP  MOVE (A1)+,D0
      CMP  (A0)+,D0
      BLT  OUT
      CMP  (A0)+,D0
      BGT  OUT
      DBRA D1,LOOP
IN    NOP          ;all signals within range
OUT   NOP          ;at least one signal out of range
```

The instructions include accessing a reading, two comparisons, and looping code. For the case of all signals within range, only one branch instruction is taken at the end of the loop. To keep it simple, a channel in the range allowed for this scan that is not to be so checked might have an artificially wide range, say 0x8000 and 0x7FFF. All upper and lower limits must be changeable online.

It will be important for the above simple logic that the readings be dependable, and that the limits be sensible. Any single deviation will immediately operate the control line to kill the electron beam. Of course, this would likely also be the case for any hardware solution.

The current A/D interrupt routine typically executes in 10 μ s, during which it stores the current value of the system 1MHz counter into an array of such times, one for each set of data. It also computes an average time between successive interrupts, normally 1000 μ s. Over a period of some minutes of time of observation, the maximum execution time was measured at 25 μ s, perhaps because some other interrupt occurred at a higher priority. These times were gleaned from the interrupt timing diagnostic that is included in the system code. (See the note, *Interrupt Timing Diagnostic*, dated Aug 23, 2002, for more details.) The point is that there is a lot of time available in the interrupt routine without crowding the next interrupt.

Details

Suppose there is a low memory global called `KHZHEAD` that can house a pointer to a block that includes the entry point of a routine to be called during the 1KHz interrupt routine. The LA can place the pointer there to point to its own code. The argument passed to the routine can be the address of this same KHz block. All the code needs to know can be found within the KHz block. So that more than one routine can be invoked from the same KHz interrupt, allow the KHz block to be an element of a linked list of such blocks, the first of which is pointed to by `KHZHEAD`.

Imagine this structure for the allocated block:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
<code>mBlkSize</code>	2	Size of allocated block
<code>linkNext</code>	4	Ptr to next such block, if any
<code>mBlkType</code>	2	Memory block type (?)
<code>entryP</code>	4	Ptr to executing code to be called
<code>capIndex</code>	2	Return value from routine signalling good-to-bad
<code>kHzSlot</code>	2	Slot# (0-511) of latest set of A/D readings in circular buffer
(other)	(any)	Arbitrary remainder of block

When a LA wants to install its routine to be called at 1 KHz, it calls an install routine to do it, passing the address of its allocated block of the above format. The install routine examines the low memory global, and if it is non-NULL, it looks at the address it points to and verifies it is the appropriate type. It checks the `linkNext` field, and if it is non-NULL, follows the linked list until it reaches one with a next that is NULL. Then it plants the pointer to the block to be installed in the linked list in that `linkNext` field that was NULL. In this way, it appends the new routine to the linked list. In order to later on remove such a linked list item, a search must be done to find a match on the one to be removed, then unlink it. All this logic must be done with care, as an interrupt can come at any time to make use of the linked list, so it must always be valid.

A special memory block type should be used, which can be checked for verification by the install and remove code. In the IRM, the A5 register should be saved and reloaded from the global used for that purpose, if it is not already done for this 1KHz interrupt code.

In the 1KHz interrupt routine called `IPDSCINT` in module `RdAD`, the logic accesses the memory pointer, which should be either `xx7E` or `xxFE`, since it is a byte offset, within 64K, of the memory word just stored. The logic then adds 2, which nets `xx80` or `xx00`, then subtracts 128. This gives a ptr to the start of the set of 64 readings just stored. (It may actually be slightly beyond this if another interrupt has delayed the start of `IPDSCINT`.) By right-shifting this value by 7 bits, one has the appropriate index for storing the current time, which was captured on entry to the interrupt routine. This value will be passed, via the `kHzSlot` field in the KHz block, to any KHz routine to be called using the scheme described here.

Approach with LA

The LA should call `KHZIns` to install a routine to be invoked at 1 KHz. The call will occur just after the time stamp has been stored for the current slot.

The routine called needs the ptr to the current set of 64 readings, which depends on the slot index ($\text{base} + \text{slot} * 128$). Then it performs its scanning loop, looking for any readings that are out of limits. The array of limits can be constructed by the LA from the current values of the nominal and tolerance fields of the corresponding `ADATA` table entries, updated occasionally.

The argument passed to the routine is the address of the KHz block structure. Parameters needed for this job include the address of the byte containing the control line to be written to, and the bit# within the byte of the control line. It also needs to derive the base address of the most recent set of readings, as mentioned above. It needs the address of the structure prepared by the LA containing the limits against which to check the latest readings.

(Note that the nominal and tolerance words in the `ADATA` entries can be used as lower and upper limits, respectively, if desired; otherwise, a separate set of limits can be maintained in the static memory block allocated by the LA.)

It also needs the number of readings to scan. For the case of signals that are not to be scanned, have the routine that prepares the structure used in the scan install `0x8000` and `0x7FFF`, respectively, for the limit values. Any reading always falls within these two values.

The `CINFOEntry` routine is available within the `LASysLib`. Given a channel number, it can return a ptr to the `CINFO` table entry and thereby the base address of the 64K circular buffer.

The address of the control line byte can be gotten from a `Bit#` parameter of the LA. It would look up the `BADDR` entry for the byte, assuming it is a memory address of the byte.

LA parameters

The Page E display list for the LA parameters may be as follows:

ENABLE	B	Enable Bit#
FIRST	C	Initial channel
NCHANS		#channels to scan
CONTROL	B	Control Bit#
ACTIVE	C	Channel# set to the #calls made to KHz routine

The 24-character descriptive text for `KHZM` can be "KHZ DIGITIZER MONITOR".

LA Plan

Initialize usual SM static memory block. Also create and initialize KHz block, including the field that points to the assembly routine that will be called by the KHz interrupt routine. (At first, try this LA before the new code is added to the system code, in order to be sure everything is set up correctly.) Use `KHZIns` to install the KHz block into the linked list. Call `CINFOEntry` to get the base address of the 64K hardware memory buffer.

Handshake

There should be a client manager program to arrange for setting the limits and enabling the KHz monitoring logic. Suppose this involves setting a bit. In order for the client to have some assurance that the logic is enabled, there should be a separate bit that the LA sets in response to indicate when the LA is enabled; it should not merely be the readback of the bit that the client set.

This means that we have a separate control line for this enabling compared to a status line that shows whether the logic is currently operational. When the LA is disabled, this status and control line should both be off. Ideally, if the node resets, one would wish that the status line would drop, but maybe we cannot do this.

If an out-of-limits condition is detected, further checking should be suspended until some sort of reset signal is granted. Or, the checking could continue, but the control action should

not. Or, the checking and operation of the control line can continue, but further capture should not. Capture should be initiated when the output control line goes from good to bad. If it goes good to bad again within the next 8 cycles, it should just start over with the capturing. Viewing the captured data is always subject to an update, but one might think that aborting the electron beam may just reinforce the abort condition, so that once it goes to the bad condition, it stays that way.

The size of the limits array is fixed at `MAXCHANS` (=64), but `firstC` is used to determine the pointer into the limits array. This means that a given channel's limits are always found in the same place. A client application may be able to target this array by using the appropriate offset into the SM structure.

The first channel number parameter `firstC` and `nChans` should also determine how much data gets captured. But the location of the captured data should be fixed; i.e., some 64K bytes can be used for it no matter how many channels are actually captured. Doing it this way means that a given channel's captured waveform resides at a location that does not depend upon the range of channels of interest as given by the `firstC` and `nChans` values.

Changing the `firstC` and `nChans` parameters should probably require restarting the local application instance. There may be more than one instance for handling separate ranges of channels, but the ranges should not overlap, and each instance should operate a separate control line.

Initial testing

A local application called `KHZM` was written in C for the IRM. A version of the system code was prepared that includes the option of invoking a KHz routine from the 1KHz interrupt code. Assembly language code was written for the two routines `KHzIns` and `KHzDel` and linked with the LA. Assembly code for `KHZMON`, the actual code that is invoked at 1KHz, was also written and linked with the LA. Using a default limits array with values of `0x8000` and `0x7FFF` for (lower, upper) limits, so that any values checked are always within limits, the Interrupt Timing diagnostic showed that its time increased from 10 μ s to 100 μ s when all 64 channel readings are scanned. This implies that the scanning loop time is about 1.5 μ s per channel checked. This therefore adds only a 9% load on the system operation overall. Since a typical IRM system is only 10% busy, the additional load to do the KHz scanning is small.

Code was then added to the LA to capture the data from the circular buffer around the time of the detection of a good-to-bad transition. The objective is to capture as many as 64 channels of 1KHz data, 512 points per channel. When the anomalous condition is detected, the KHz routine deposits the `kHzSlot` number in the KHz block so that the LA can discover it the next time it executes its 15 Hz thread. At that time, it backs up 400 points in the circular buffer and copies 64 points for each of the 64 channels into the target capture buffer, whose address is an additional parameter of the LA. This amounts to one-eighth of the job, and it is nearly the amount of data that is placed into the circular buffer every 15 Hz cycle (66 points). On each of the next 7 cycles, the rest of the job is completed in like fashion. This results in 512 points of waveform for each channel being captured and stored as 64 waveforms. Each waveform includes 400 ms of data before the triggering anomaly and 112 points after. Of course, the value of 400 is not magic, but it should not be more than about 440, since up to 66 points may have already been digitized before the LA has a chance to notice that an anomaly was detected. (The idea here is that the capture should not be done by the KHz routine that is called, but should be done by the normal task level LA code.) The result of the testing showed that 4.6 ms was required on each of the 8 consecutive 15 Hz cycles to accomplish the capture for all 64 channels. This is not a serious impact on the rest of the system operation.