

Front-ends at 15 Hz

Performance analysis

Robert Goodwin

Fri, Mar 1, 2002

A key characteristic of the Linac front-end design since its inception has been reliable 15 Hz performance consistent with the operation of the Fermilab Linac. The software is designed to “keep up” with the process. If an important accelerator device reading is found by the alarm scan to be in a “bad” state, accelerator beam is inhibited on the very next 15 Hz cycle. In order to achieve this objective, care is taken not to write software that usurps too much CPU time. Focusing on the installation of the PowerPC/vxWorks-based front end software used in 22 Linac nodes, this note analyzes its performance against the “SRM status” alarm messages, which result from imperfect 15 Hz data collection. This note results from recent efforts to eliminate the causes of such alarms.

SRM status alarms

Linac front ends collect a large part of their data from I/O interfaces known as Smart Rack Monitors on each and every 15 Hz cycle. Early in the cycle, a request message is sent to all (from 1–6) arcnet-connected SRMs with the meaning, “read all your data and return it.” Upon receipt of this request, all SRMs in parallel acquire new readings of all their analog and digital data. As each finishes, it returns a single frame to the front end that contains all of this data. The front end anticipates the arrival of such replies as a major step in updating its data pool each cycle. But a time-out is imposed so that missing responses can be detected without hanging up the logic. This time-out is specified in terms of a deadline of time within the current 15 Hz cycle, which is defined as beginning with the 15 Hz interrupt signal that initiates task-level processing of cyclic activities. This deadline is currently set at 24 ms. Typically, all SRM replies are received by about 12 ms. During this waiting time, there is nothing else of significance that can be done in order to maintain the façade of a data pool that is updated instantly on every 15 Hz cycle; an outside data requester is not allowed to “see” a partially updated data pool. Local applications cannot be run until the data pool is completely refreshed with new SRM data, since many of them need access to fresh device readings.

In order to monitor the continued success of arcnet communication with SRMs, a status bit is set for each SRM upon successful reception of its cyclic reply message. In each front end, a collection of such status bits is monitored for alarms in the usual manner. Missing replies thus produce an SRM status alarm message. An analysis of the causes of such alarm messages is one method of assessing successful 15 Hz operation of the Linac front end software.

How goes the analysis?

Although the installation of the upgraded Linac system was completed last year, occasional SRM status alarms occurred. Although they were not enough to impede Linac operations, they were a nuisance for generating transient alarm messages. A serious effort was mounted to analyze the causes of these messages. Was something wrong with the arcnet hardware? Were there weaknesses in the new software? Were unusually heavy requests arriving from console computers that placed a heavy load on the front ends?

Various software diagnostics were used to help answer such questions. One diagnostic is provided by the alarm messages themselves, which are multicast via Classic protocol, in which each message is time-stamped to the accelerator cycle on which the alarm condition was detected by the Alarms task. This is possible only because complete alarm scanning is performed on each and every 15 Hz cycle. (Currently, such alarm scanning is accomplished in less than 1 ms.) All such alarm messages are captured into files for later analysis. Extracting the SRM status alarms provides a “bottom line” for the present analysis. The goal is to eliminate the occurrence of such messages as much as possible.

Since most of these alarm messages occurred on single 15 Hz cycles only, followed by a “good” message on the very next cycle, we could have simply selected “tries needed = 2” for such alarm messages, with the result that nearly all such alarm messages would cease. But that would be like hiding dirt under a rug, as there is no reason in principle why there should be such messages at all during stable operation. Over the 10 years that the former Linac front ends had been in operation, we had a lot of experience with reliable SRM data collection. For the latest upgrade, no changes were made

to the SRM software. With nearly 100 million transactions per day—24 hours of 15 Hz operation with each of 75 SRMs—we are used to expecting high reliability.

Diagnostic tools

A number of diagnostic tools are built into Linac system software. Some of these are built around the “data stream” paradigm, a kind of formalized circular buffer scheme that is supported by the system. One example is the “network frames” diagnostic, in which all network messages, either transmitted or received, are logged and time-stamped to a half-millisecond within a 15 Hz cycle. In order to support this tool under vxWorks, all received datagrams pass through a single Socket Reader task via the `select()` feature. This diagnostic helps to see any unusual network traffic that might delay normal 15 Hz task execution.

Another useful diagnostic is the task activity data stream. This was made to work under vxWorks by specifying a `taskSwitchHook` function, along with utilizing a special user-accessible field in the task control block that helps identify which task is being switched in or out. The usual time-to-the-half-millisecond of the start of task execution is included along with task execution time in microseconds. By examining detailed task execution within every 15 Hz cycle, we can gain a measure of confidence that we know what is happening inside these front ends. For example, one can easily make measurements to show what happens when receiving certain network data requests and how much CPU time is required to build a reply.

Several other diagnostic tools are also useful. One is a log of unusual elapsed times measured between successive 15 Hz interrupts. There are occasions in which something strange happens with the Tevatron clock, and it is useful to know this so that one is not chasing down errant path of analysis. These front end systems are driven by an accelerator-synchronous 15 Hz interrupt, derived from the 0x0C clock event plus a delay, which is set at 3 ms for all Linac front-end nodes. In addition, there is an Acnet alarm log that shows communications with AEOLUS, and there is a RETDAT log that time-stamps all Acnet data requests. There are others that will not be described here.

Analysis progression

A new local application called TSKM was written for the purpose of monitoring a “4.5 ms anomaly” that was detected in the latest vxWorks kernel, as it turned out. It captured task activity data stream records when excessively long `tNetTask` activity occurred. This sometimes correlated with SRM status alarms. At this time, some of the nodes used a shorter (16 ms) deadline for SRM replies, when the last reply routinely arrived at about 12 ms. Because of the anomaly, when it occurred at just the wrong time, the logic was fooled into thinking that the SRM reply failed to arrive in time. As a result of analyzing this diagnostic, the deadline time was moved to be at least 24 ms for all nodes. We had been bitten by the 4.5 ms anomaly kernel bug.

Another local application, called BADM, was written to monitor SRM status alarms and capture task activities occurring at that same time. (The local application ran in a test node and monitored the SRM alarm status in a target node at 15 Hz. When it observed ‘bad’ alarm status, it sent one-shot requests for the most recent task activity and network activity, logging the results.) There had been anecdotal evidence that SRM status alarms often occurred around times of Acnet Big Save operations. So we looked for a correlation of the occurrences of SRM alarms with those times. The correlation was close, but not coincidental with the acquisition of device data from Linac nodes. (This was made more clear when new time stamping of such accesses was added to the Acnet Big Save statistics log available on the Web.) But in connection with Big Save data acquisition, there is also monitoring of what FTPMAN snapshot support is available for each device. To learn that, a special FTPMAN request is made for each device, apparently somewhat ahead of the data requests for the Big Save log.

The front end software support for answering the question, “what plotting capabilities are available for these 1–4 devices,” is not particularly efficient. A table search is performed by the FTPMAN logic, perhaps multiple times for each device, in order to derive the proper plot class code to return in answer to this query. Memory searches are normally fast in the IRM nodes, but for these PowerPC nodes, the table being searched is in nonvolatile memory, to which access is unusually slow, about one microsecond per access. (One microsecond might seem fairly short, but in the context of a 233 MHz

CPU, it amounts to 200-odd wait states!) Examining the detailed search logic, at least three accesses are made for each table entry scanned. Somewhat arbitrarily, the Linac node default allocation was 512 entries for this table, allowing for a lot of expansion. For most devices for which the question is asked, the table search ends in failure, so that all entries are scanned. One scan might take 1.5 ms, assuming 3 accesses per scan, but multiple scans are made, looking for different potential fast digitizer hardware related to the device in question. All in all, up to 16 ms might be required to perform the necessary scans to fulfill a request for 4 devices. (This was actually observed.) If such a request is received at the wrong time—say, near the end of the cycle just ahead of the 15 Hz interrupt—the system code may be busy dealing with that request, so that the start of its normal 15 Hz activity on the next cycle is significantly delayed. That means that the request for SRM data is delayed, so that the deadline within the next cycle is passed before the reply is received. The task activity diagnostics revealed this effect.

As it was late on a Friday afternoon, the immediate remedy for this discovery was to drastically reduce the size of the nonvolatile memory table to be searched. This was easy to do, since only three of the 22 nodes had any entries in this table at all. This cut the search time way down, and the requests are now handled much more quickly. At some point, this scan logic will be modified to better optimize the number of nonvolatile memory accesses needed.

This was the second time that slow access time to nonvolatile memory was a problem. The first time was in the Alarm task scanning of the Analog Data table for alarm conditions, which required as much as 4 ms to perform a complete scan. We modified the alarm scan logic, optimizing it to minimize the number of accesses to nonvolatile memory, with the result that the alarm scan in the worst-case node now requires about 0.8 ms.

In examining the SRM alarm-related task activity, there were occasional strange occurrences that were not related to FTPMAN data request fulfillment. Very seldom, there were cases in which the normal 15 Hz Update task execution failed to run! This was a big surprise, as this activity is crucial to Linac system 15 Hz operation. Looking at the network activity as well as the task activity occurring at the same time, and using the time stamps to correlate between the two, these unusual cases occurred when a network message was received very near the end of the cycle. It was not a difficult request message to support, but its precise timing was perhaps a problem. Still, one could find many cases when request messages were received near the end of a cycle in which nothing went wrong during the next cycle. Indeed, it should not matter at all, according to the planned system design. It was a “puzzlement.”

This led to a review of the detailed logic that emulates a “task events” feature that was supported by the older pSOS kernel but not supported by vxWorks. In order to emulate it, a binary semaphore was used for each task along with a word to hold task event bits. The review revealed a small window of opportunity for a problem. If the 15 Hz interrupt occurred at precisely the wrong moment, the attempt by the interrupt code to send the 15 Hz data pool update event to the Update task might fail. It had to do with the need to sample a word from memory and clear it before an interrupt might have a chance to set a new bit in that word. To be sure, the chance for failure was very small, but we were looking for causes of unusually rare failures.

The scenario would play out in this way. A data request is received late in the 15 Hz cycle. After initializing the request, the Update task is triggered (via another event) to build and transmit a reply to the request. During the exit of the Update task, the 15 Hz interrupt occurs, and an event is sent to the Update task to signal that 15 Hz data pool updating is needed. But the interrupt occurred just at the wrong moment, so that upon return to task level, this event bit was lost.

We had some discussion on how to close this small window. The PowerPC architecture provides a mechanism to support such things in its CPU design. But vxWorks does not provide sufficient routines to support the usual set of “atomics” that might use this mechanism. The alternative is to consider inhibiting interrupts around the critical section of code. This technique of inhibiting interrupts is avoided in many real time system designs, and it has not been used before in Linac front end designs. (The extent that it is used by the vxWorks kernel is unknown.) But `intLock()` and `intUnlock()` routines are each only about 5 instructions long, very short for a PowerPC running at 233 MHz. The method of locking interrupts was implemented to close these tiny windows. Extensive testing over a

few days of the modified code on a test node showed no SRM status errors, even when a client test node sent requests to that node at 15 Hz near the end of the cycle in an attempt to provoke them.

It was during this last testing that yet another activity was found that can cause adverse impact on reliable 15 Hz processing in the Linac front-ends. This is caused by telnet access to the shell running in each front-end. (It also applies to local console interaction using the same shell.) The shell is a very high priority task in a vxWorks system, and telnet communications is also a high priority task. The danger is that any such activity might steal CPU cycles for too long. Some informal timing measurements showed the following shell command interference:

<i>shell command</i>	<i>execution time (shell + telnet)</i>
d 0x114000,64	1.4 ms
lkup "atom"	7.0 ms
i	13 ms
ls "RAMDEV1"	2.5 ms
lkup "Show"	3.4 ms
ping "131.225.123.215"	1.0 ms to get started + 0.2 ms per ping reply
^C	7.2 ms to restart shell
lkAddr 0x111000	17 ms
l 0x111000,32	15 ms, or about 0.5 ms/disassembled instruction
(telnet login)	1 ms
inetstatShow	2 ms
logout	1 ms

One can see that telnet or console shell activity may cause undue impact on the real time performance of the front-ends. This feature should be used with care on Linac vxWorks nodes.

After testing a version of the system code that used `intLock()`, all PowerPC Linac nodes were upgraded to that version during a downtime about a week ago. A review of the SRM alarm messages occurring during that time showed none that was mysterious. A number of them occurred in the node0610 front end that supports the preaccelerator, but these are presumably 750 KV spark-induced. No other nodes showed SRM alarm errors of the sort to which we had become all too accustomed.

Conclusion

The Linac front-ends were designed and implemented to operate reliably in the Fermilab 15 Hz environment, but some diligence is required to ensure that level of performance. A number of diagnostic tools are included in the system design that can verify its detailed operational behavior. Use of these tools enabled us to find and correct several "latent software instabilities" in the new PowerPC-based front-end software implementation.

The Fermilab accelerator is inherently a real time process. The challenge is to provide an operational interface that enables the operators to view and control this real time process. A front end design that provides consistent and reliable 15 Hz real time performance is a step along a direction that can address that challenge.

Credits

The author acknowledges beneficial interactions with a number of people, including Bob Peters, Mike Sliczniak, Mike Shea, Bob Florian, Mike Kucera, Duane Voy, and Dinker Charak.