

LATBL Scanning

Efficiency improvement
Sat, Aug 3, 2002

Local applications are designed to add features to the system operation without having to be installed as part of the linked system code. This flexibility permits use of a common version of system code in the front-ends of different projects even though particular projects have specific needs that can be accommodated via this local application mechanism. The nonvolatile Local Applications Table (LATBL) is the catalog of local applications that are invoked at least every cycle. In the PowerPC systems, access to nonvolatile memory is relatively slow, so special attention might be paid to improving the operational use of this table. This note describes a scheme for doing this in a way that can save many references to nonvolatile memory.

The basic idea behind the scheme is to keep a volatile (fast) memory copy of the nonvolatile table. The wrinkle is, of course, that we need to permit modification of the nonvolatile table contents during system operation, so that such modifications are effective not only immediately but also used the next time the system is reset.

The basic means of modification of the LATBL is via "Page E" on the little consoles or via emulation of same. The name of the page application is `PAGELAPP`, referring to local application parameters, since a common need for modifications of LATBL entries is to change parameter values that are housed therein. Such modifications of LATBL entries are done using settings to listype 74, which currently targets the nonvolatile entries via generic memory access.

Suppose a copy of LATBL is created at reset time to be used during system operation. Suppose further that in order to preserve the ability to modify the table contents online, read access to this listype references this online copy of the LATBL, but setting access to this listype references both the nonvolatile table and the volatile table. Viewing the situation as presented on the Page E display, one would see the volatile (active) table entries shown, but changes to those entries would not only be recorded in the active table entries viewed, but they would also be stored in the nonvolatile entries.

Let us review the 32-byte LATBL entry layout:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
<code>codesx</code>	1	CODES table index of related LA
<code>lastAct</code>	1	Record of current LA enable bit status
<code>elapsed</code>	1	Elapsed execution time in 0.5 ms units
<code>callCnt</code>	1	Counter for cycle calls to show activity
<code>cName</code>	4	Program name, with <code>LOOP</code> , found in CODES table
<code>smPtr</code>	4	Ptr to static memory used by program while enabled
<code>enable</code>	2	Enabling Bit number
<code>params</code>	18	Nine parameters for uses definable by the LA

When performing a setting, some of these fields would be better ignored; for example, it would not be a good idea to target the `smPtr` field, which is a pointer to allocated memory written to when the LA is initialized. We are fortunate in the present system of getting away with this, because the Page E logic always has the latest copy of the entry

available when making a setting, so that a setting does not destroy the `smPtr` field; it merely overwrites it with the same value that it had. For a setting that is not made in such an interactive environment, however, this kind of setting could be dangerous.

The only fields of the online LATBL entry that can be changed are the `cName` and the 10 parameter values, including the enable Bit number. These are also the only fields that need be written to the nonvolatile LATBL entry. The other fields are used during operations only. The new setting type routine that implements this logic would target both the online and nonvolatile copies of the LATBL entries affected.

There is a potential pitfall in modifying the `cName` field, which would change the program that is related to the LATBL entry. Such a modification should not be made while the entry is active. It may be better to refuse a setting that would change the program name while the enable bit is set. This is another benefit of the new scheme. In a similar vein, one may also disallow modifying the enable Bit number while the program is enabled. But this is occasionally useful during configuration if the state of the Bit number currently in use is not modifiable, so it may be as well to expect the user exercise due care here. Altering the other parameters should be ok, even if it may also require some care.

In order to implement the scheme sketched here, the system code that scans the entries in the LATBL should be reviewed, so that it references the online copy of the table entries, except for the initialization code of `InzLOOP`, of course, which will need to create the online copy of LATBL. There are only a small number of places in the system code where the LATBL is referenced, so this should not be difficult. There may have to be a new global variable that houses the pointer to the online copy of LATBL, or some particular memory could be set aside for such use, or both. There is not the same need for access to the online copy for printing out a record across all nodes in a project, say. Saving the LATBL contents for use in later system restoration can be done via access to the (nonvolatile) memory that is found in the system table directory. (The online copy need not be saved for such a purpose.)

The special listype used to access LATBL entries will need a read type routine that accesses the online copy of LATBL. The associated set type routine will have to target both the nonvolatile and online copies of LATBL, but with care. Not only will the scheme save time by reducing the number of accesses to nonvolatile memory, it may also improve its robustness *vis-a-vis* setting modifications to LATBL entries.

Implementation details

There are two listypes that refer to LATBL entries. The first one provides simple access to LATBL entries. This listype needs separate read-type and set-type routines to support the behavior described above. The read-type accesses the online copy of LATBL; the set-type routine writes to both the online and nonvolatile copies of the LATBL, with special care.

The second listype used for LATBL access is listype 86. It allows a request to be made for the LATBL entries that use a given name, which is included in the specified ident when the request is made. The reply data to such a request identifies all of the LATBL entries that use that name. Recall that multiple instances can be defined for use with the

same local application program, resulting in the same program being invoked multiple times but with different parameters and different static memory context. The only change to the special read-type routine used here is to reference the online copy of the LATBL rather than the nonvolatile copy.

In the `LocApp1` module, the `LocApp1` routine is changed so that it references the online copy of LATBL. (This is the change that is designed to save some time in the PowerPC version of the system by reducing the number of accesses to nonvolatile memory.) One of the fields in the online LATBL entry holds the measured elapsed time of execution every cycle that it is invoked during processing of the Data Access Table. This has until now been done in units of 0.5 ms, which is rather coarse, as most LAs execute in much less time than that. To improve the resolution, we can change the units to 0.1 ms. Since the microsecond counter is used to derive this number, it will be much more stable as displayed on Page E than the previous value that was derived from the difference of two readings of an asynchronous 0.5 ms counter. One might argue for measuring elapsed time with even finer resolution than 0.1 ms, but the elapsed time result must fit in an 8-bit field. Using 0.1 ms, this places a limitation of 25.5 ms on the maximum value recordable, which is probably enough for even very heavy LA work. If a measured elapsed time turns out to be larger than this, limit the value stored to `0xFF`. To support this change in units for the elapsed time byte field, a change to the `PAGELAPP` program must be made, in which it somehow recognizes which is the appropriate units in use by the target node. (Not all nodes are updated to the latest system code at once.)

Although unrelated to this LATBL upgrade, another change can be made that will be helpful in the PowerPC version of the system code. In many diagnostic records maintained by the system, or by helpful local applications, there is a need to register the time-of-day at which something happened. This time stamp is in BCD, for convenience in viewing in hexadecimal, using 7 bytes to show the time in `yr-mo-da-hr-mn-sc-cy` to the 15 Hz cycle within the second, and using the final byte of the total 8-byte structure as the binary number of 0.5 ms units within the current 15 Hz (or 10 Hz) cycle. In the PowerPC, this value is obtained from the digital PMC board, which unfortunately has a rather slow access time. Since this is used very heavily by the system, when all uses are considered, it would be better to eliminate the need for obtaining this time from that hardware. By capturing the base reference time at the start of 15 Hz cycle execution in more accurate units, we will have a basis for providing a new version of the `TIMECYCL` routine that returns the value of time since the beginning of the current cycle (but still in 0.5 ms units so that its range can cover the entire cycle). All that is required is to capture some 4-byte high resolution time value, such as from the Time Base register, and to record it somewhere globally so it can be accessed by the `TIMECYCL` function.

Modules affected

`Defines`. Use the area of memory from `0x1800-0x27FF` to hold up to 4K bytes for the online copy of LATBL. The usual size of this table is 2K bytes, for 64 entries.

`InzSys`. Define two 4-byte global variables. `CYCLETB` (at offset `-96`) holds the microsecond counter at the start of the current cycle. `LATABLE` (at offset `-92`) holds the address of the online copy of LATBL. If zero, there is no online copy.

INTSFp. Capture the microsecond counter reading into the new global variable **CYCLETB** at the start of the **CYCLE** routine.

LTT. The entry for listype 74 is changed to specify read-type 28 and set-type 39.

ReadType. The read-type 86 is changed to access the online copy of **LATBL**. The read-type 28 routine is new and merely copies from the online copy of **LATBL**, rather than the nonvolatile **LATBL**. This is important for **PAGELAPP**, since it would not otherwise display signs of visible activity, such as changing counters.

SetType. Add a new reference to set-type 39 routine **SETLATBL** in **LocApp1**.

LocApp1. Change **InzLOOP** so that it creates the online copy of **LATBL**, recording its base address in the new global variable **LATABLE**. Change **LocApp1** so it uses the online copy of **LATBL** for its scan; also, measure the execution time of each **LA** in units of 0.1 ms, rather than 0.5 ms. Measure the total time for the **LATBL** processing in microsecond units to be recorded in the Data Access Table entry. (Currently this diagnostic uses 0.5 ms units.) The new set-type 39 routine carefully copies portions of the up-to-32-byte setting data into both the online and nonvolatile versions of the **LATBL**.

PAGELAPP. Obtain a copy of the **LATABLE** global variable. If it is nonzero, consider that the elapsed time units for **LAs** are 0.1 ms; otherwise, assume 0.5 ms. This only makes sense considering that these two changes are being made at the same time: support an online **LATBL**, and measure elapsed time in 0.1 ms units.

As a timing aid, if **LATBL** has an odd number of entries, do not use online copy.