

Calendar Time

LOOPTIME addition

Tue, Apr 23, 2002

For support of MiniBooNE, it is important to time stamp data with calendar time, because the experiment will use it to time-correlate with accelerator data. It is desired to do this more accurately than is done for traditional support of time-of-day in the IRM systems. In the latter case, the time-of-day is maintained by assuming that each node's 15 Hz interrupt occurs at 15 Hz, so that time is in units down to a 15 Hz cycle. But accelerator 15 Hz is not synchronized to calendar time, because it is synchronized to the power line frequency, which wanders somewhat during the day. This time-of-day is updated periodically by the response to a Network Time Protocol query. The current server used is `chablis.fnal.gov`, which exhibits a reasonably consistent, prompt response time. With queries performed once per minute, seldom is more than a one-cycle adjustment needed to align accelerator time-of-day with calendar time.

Calendar time update

The MiniBooNE experiment will work with calendar time in GMT units, without consideration of daylight/standard time adjustments. When the NTP server is queried via the standard UDP port 123, the returned 8-byte time value is in units of seconds, with the most significant 4 bytes representing seconds since January 1, 1900. The least significant 4 bytes is a 32-bit fraction of a second, implying that the units of precision are $10^{12}/2^{32} = 232$ picoseconds, although the absolute accuracy is presumably somewhat less.

The Tevatron clock event 0x8F occurs at 1 second intervals. It is synchronized to calendar time via a radio receiver in the MCR that delivers a 1 Hz output. When this event occurs, one can expect that the NTP time is close to an even second.

The clock event 0x0C occurs at 15 Hz and, with a programmed delay, is the interrupt trigger for IRMs that announces the start of a new 15 Hz cycle. This causes the Update task to run to refresh the data pool, execute any active local applications, and fulfill any active requests for which replies are due on the current cycle.

Additions are made to LOOPTIME to provide the 8-byte time adjusted to the current cycle's 0x0C clock event. Every cycle, LOOPTIME updates this time by adding the delta time between the two previous 0x0C clock events. This elapsed time is measured in an IRM by using the microsecond timer on the CPU board. To improve the accuracy of this elapsed time value, an average of delta times between successive 0x8F events is calculated. This may compensate for any variation in CPU board crystal that is the source of the microsecond timer's counting.

A special local application written to support MiniBooNE data collection uses this 8-byte GMT time along with accessing the clock event table, in which for each clock event are kept the values of the microsecond counter at the time of the event interrupt and the delta time between the last two such event interrupts. The 0x0C event occurs at the same time as the current Booster reset clock event. Given GMT(0C), one can easily compute the GMT time for any other event, such as the MiniBooNE extraction event.

Having just received a suitably prompt reply GMT(now) to a periodic query for NTP time, after possibly adjusting the value for half the measured response time, LOOPTIME sets GMT(0C) = GMT(now) - (t(now) - t(0C)), where t(now) is the current value of the microsecond counter. The value t(0C) comes from the clock event table via the function `GetEvtT`.

To get the GMT(1F), for example, simply calculate $\text{GMT}(0C) + (t(1F) - t(0C))$. Note that the difference $(t(1F) - t(0C))$ should be positive, as Booster extraction occurs about 35 ms after the 0x0C clock event, and MiniBooNE nodes are timed to begin cycle activities at a time following Booster extraction.

To get more accurate values for differences such as $t(x) - t(y)$, it may be useful to maintain a correction factor derived from an average of delta times associated with the 0x8F event, counting on the fact that the 0x8F event occurs at a 1 Hz rate to high accuracy.

If it is desired to have LOOPTIME running in only one MiniBooNE front-end, the latest received value of GMT(0C) can be forwarded to the other nodes, targeting a set of 8 bytes from the Generally Interesting Data area. System code updates a value of GMT(0C) for use within that node until the next LOOPTIME-determined GMT(0C).

System additions

It is useful to include support for GMT(0C) in the system code, rather than put it into LOOPTIME. Maintained in the Generally Interesting Data area, it is updated from any node that runs LOOPTIME and multicasts the GMT(0C) when it obtains a fresh reply from the Network Time server. Using a copy of this last updated GMT(0C) as a starting point for calculating a working GMT(0C), until the value in GID changes, a better corrected value of GMT(0C) can be obtained. Delta times between successive 0x0C events are accumulated since the last update of GMT(0C) via GID, and the correction factor is applied to the entire accumulation. This builds a working GMT(0C) to be used by applications, updated each cycle by system code. This copy must be updated whenever a new GMT(0C) base arrives to be placed into the GID. In the case that too much time elapses since the last update from the GID, meaning that the accumulation becomes inconveniently large, copy the latest GMT(0C) into the GMT(0C) base in the GID area, then reset the accumulation. This GMT(0C) update logic should be accomplished by the Update task before the Data Access Table is interpreted, so that local applications will have access to the GMT(0C) that refers to the current 15 Hz cycle.

In order to support this new calendar time feature, changes must be made in several different program modules:

- System reset initialization
 - Initialize variables used
- Update task
 - Compute new correction factor, save in nonvolatile memory
 - Compute new GMT(0C) using latest correction factor, capture t(0C)
- Setting handler for GID
 - Copy to GMT0C also, clear accumulations, capture t(0C)
- LOOPTIME
 - Capture GMT(now), compute local GMT(0C)
 - Multicast to GID area in other nodes

Adjust microseconds for local crystal

Compute an average value of delta times between successive 0x8F events, which occur at accurate 1 second intervals. This average value is used to normalize other microsecond timing within the same node. A function called `MicroAdj` can perform this computation, referencing the local correction factor in low memory.

```
FUNCTION MicroAdj(us: Longint): Longint;
```

The low memory correction factor is a 32-bit unsigned value scaled at 2^{31} ; hence, its expected

value is approximately 2^{31} . This function simply multiplies by the correction factor and divides by 2^{31} . The reason for using 2^{31} scaling rather than 2^{32} is, of course, that the correction factor value is very near 2^{31} , more or less.

The correction factor is derived from the average measurement of N delta times between successive 0x8F clock events. (To avoid bad influences from missing 0x8F events, insure that each value averaged is quite close to 1 second.) To calculate the correction factor, multiply 1000000 by 2^{31} and divide by the average value. The correction factor is updated every N seconds. The correction factor is initialized from a value maintained in nonvolatile memory at reset time. Using N = 64 results in a new average about every minute. In this case, a more accurate correction value would be

$$1000000 * 2^{31} / \text{sum}$$

where sum is the accumulation of the 64 measured delta times between 0x8F events.

Variables to be used

Variable	Size	Location	Meaning
GMT0CR	8	GID+16	GMT(0C) received as GID setting
GMT0C	8	Low+0x680	Working value of GMT(0C)
USCOR	4	Low+0x688	Working μ s correction factor
ACCUS	4	Low+0x68C	Accumulated 'microsec' since GMT0CR
GMT0C_T	4	Low+0x690	t(0C) corresponding to last update of GMT0C
USCORSV	4	PAGEM+0x70	Saved value of USCOR in NV memory

Note that USCORSV is in the PAGEM table along with other system-level variables. Low refers to "low memory," where various system global variables are housed. The value of Low is 0 for IRM/68K-based systems that use pSOS, but it is a higher value for PowerPC/vxWorks-based systems, since vxWorks uses much of the real low memory for its own purposes.

The format of GMT0C is a long word of seconds since 1900 followed by a long word of microseconds. This means that LOOPTIME will convert what it receives from the NTP server into this format and adjust it to the time of the most recent 0x0C event before multicasting it to other nodes. The GMT0C_T holds the microsecond time of the 0x0C event that is associated with the current value of GMT0C. It is needed to resolve the problem of multicasting to nodes that operate with different offsets relative to the 0x0C event. This is explained later.

At system reset time, copy the nonvolatile USCORSV into the working version USCOR, and clear the ACCUS and its associated counter. It is impossible to have a correct GMT value until receipt of the next value from LOOPTIME, so clear GMT0CR at reset time, thereby disabling the logic that updates GMT0C. (GMT0C is also cleared at reset time.)

Logic performed at 15 Hz

During the Update task, prior to Data Access Table processing, the following logic is used to update GMT0C:

If GMT0CR is nonzero, it means the system has received a correct GMT0C from the LOOPTIME local application, either in the local node or in another node that forwarded the value to this node. If GMT0CR is zero, we don't need to update GMT0C at all.

With GMT0CR nonzero, add the delta time for event 0x0C from the clock events table to advance ACCUS, which is maintained in uncorrected microseconds. Use this accumulated microseconds value to multiply by the correction factor USCOR. Divide the result by 1000000 to obtain a

quotient of seconds and a remainder of microseconds. This GMT0C computation uses GMT0CR as a base, to which is added this corrected time obtained from ACCUS since the last time GMT0CR was updated.

In case there is no updating of GMT0CR for too long a time, say 2^{31} μ s, or about 35 minutes, take the GMT0C result obtained above, overwrite GMT0CR, and clear ACCUS. (This avoids problems with overflowing ACCUS. Wherever the GMT feature is used, there will always be periodic (say, once per minute) updating of GMT0CR, so that this adjustment will not usually be needed.

When a new value of GMT0CR arrives via the special listype used for accessing the GID area, it is accepted as a new GMT(0C) for the current cycle, so that it is copied into the working GMT0C, and ACCUS is cleared. The current t(0C) is also saved in GMT0C_T.

The logic that calculates a new correction factor USCOR operates independently of the above GMT0C updating logic. It watches for 0x8F events and builds a sum of elapsed times in microseconds. After 64 sums have been accumulated, each one checked for having a value near 1000000, it computes the correction factor as noted above:

$$\text{USCOR} = 1000000 * 2^{37} / \text{sum}$$

If 1024 accumulations are used, taking about 16 minutes, the formula is

$$\text{USCOR} = 1000000 * 2^{41} / \text{sum}$$

This correction factor is scaled at 2^{31} , so use the correction factor as follows:

$$\text{corrected} = \text{microsec} * \text{USCOR} / 2^{31}$$

In the LOOPTIME local application, a few changes must be added to support the above additions to the system code. When a response comes back from the Network Time server, it must be adjusted to the GMT0C format from the format that was received. Assuming that the 4-byte seconds component is ok as is, the last unsigned 4-byte fraction must be adjusted for the time of the last 0x0C event. The unsigned computation that does this is simply

$$\text{frac} * 1000000 / 2^{32} - (\text{t(now)} - \text{t(0C)})$$

If this turns out to be negative, decrement the 4-byte seconds value by 1 and add 1000000 to the negative value. Together, these 8 bytes should be forwarded to a target node number, which probably refers to a multicast IP address. Because of the convention that all Classic protocol messages are ignored if they come from the local node, a separate setting must be made to the local node to update its own GMT0CR in the local GID. It may be unnecessary to apply the microsecond correction factor to the value for $(\text{t(now)} - \text{t(0C)})$ in the above formula, since the amount of the correction would likely be less than one microsecond.

Post-implementation

The above changes were installed in a new version of the system code and a new version of the local application LOOPTIME. The system code includes a new source module called CalTime, within which are the following routines, described as Pascal code:

```

TYPE
  GMTType=    { GMT time structure }
  RECORD
    secs: Longint; { unsigned seconds since January 1, 1900 }
    mics: Longint; { microseconds within second, range 0-999999 }
  END;
```

```

MicroSum(VAR usSum: Longint; VAR usCor: Longint; VAR usCnt: Integer): Integer;
GMTUpdt(VAR gmt0CR: GMTType; VAR gmt0C: GMTType; VAR accus: Longint; usCor: Long:
MicroAdj(us: Longint): Longint;

```

The Update task calls a new local routine `GMTTIME` prior to executing the instructions in the Data Access Table that update the data pool and invoke all active local applications. The routine `GMTTIME` invokes `MicroSum`, which builds a sum to produce the correction factor. If a nonzero result is returned, the new value of the correction factor is copied into nonvolatile memory. Then, if `GMT0CR` is nonzero, and the current `t(0C)` is different from the last-recorded `GMT0C_T`, `GMTUpdt` is called to update `GMT0C`. The function `MicroAdj` is included to perform the correction factor adjustment to microsecond units. Although not needed here, it may be added to `LASYSLib` for use by local applications.

Besides the new `CalTime` module and the changes in the Update task, a change was made to the `SetGID` routine in the `SetType` module. When a new setting is made to `GMT0CR`, using the offset value 16, the new `GMT0CR` is copied into `GMT0C`, and `ACCUS` is cleared. The newest variable `GMT0C_T` is also set to `t(0C)`, the microsecond time of the most recent `0x0C` event.

Besides the changes to the system code, additional changes were made to `LOOPTIME`. The `GetTimeGMT` routine computes `GMT(0C)` from the NTP server reply, using the server's response time in the computation. It also uses an empirically-determined offset explained in the next paragraph. This `GMT(0C)` is stored in the local `GMT0CR` via `SetTimeGMT`, which uses a `GID` setting, and it is optionally forwarded to a multicast address to reach other nodes. (This is the same multicast address used to forward the usual BCD date-time format.)

The empirically-determined offset that is applied to the calculation of `GMT(0C)` is derived from comparing an average of `GMT(8F)` values during the one minute period between NTP server queries with the occurrences of `0x8F` clock events. This offset is used by `GetTimeGMT` to improve the accuracy of the `GMT(0C)` determination. The amount of this offset is confined to the -10 ms to $+10$ ms range. Such large deviations are expected only if something is wrong either with the `0x8F` events or with the NTP server.

Several diagnostic logs have been added to help characterize the performance of the new `GMT` feature. Each log uses a generic circular buffer scheme, built similarly to a data stream, that may be useful for programming other local application diagnostics. The logging scheme supports the recording of records that contain a primary 32-bit value and an optional secondary 32-bit value that are time-stamped with the usual BCD time. The minimum and maximum primary values are maintained in the header, along with a count of the total number of records written.

The diagnostic logs currently in use are as follows:

Log	Value	Value2	Meaning
1Hz	abs(usec)	usec	usec = delta(8F), for usec > 50 μ s
Cor	corFact	delta(corFact)	correction factor scaled at 2^{31}
Rpy	usec	—	response times of NTP server, for usec > 5000 μ s
GMT	delta(usec)	GMT.usec	deviation of GMT second, for delta(usec) > 400 μ s
Dev	avg(dev)	—	average GMT(8F) deviation, for avg(dev) > 500 μ s

Preliminary analysis of these statistics over one day's time are as follows:

The 1Hz log shows no `delta(8F)` values logged more than 50 μ s away from 1000000 μ s.

The Cor log shows correction values in the range 1.00000700–1.00000711. This corresponds with the fact that the test node0509 typically measures a time between successive 0x8F events of 999993 μ s (without applying the correction factor). The small variations in the computed correction factor may be explained by the jitter in software time-stamping of two 0x8F events that are 1024 0x8F events apart, not any measurable variation in CPU crystal frequency.

The Rpy log shows that response times of the NTP server measured to be > 5 ms actually ranged from 9 ms to 226 ms. It seems that the NTP server can at times be found to be quite busy. (Any response time > 5 ms is treated as an invalid time-of-day reference by LOOPTIME.) The typical NTP server response time is less than 2 ms.

The GMT log shows values of GMT(8F) that are significantly far off the exact second mark. Those logged values of deviations beyond 400 μ s range from –2.5 ms to +2.0 ms.

The Dev log shows average deviations more than 500 μ s that range from –2.5 ms to +2.0 ms, essentially that shown by the GMT log. Typically, after applying the empirically-derived adjustment, the GMT(8F) deviation from the calendar second mark is well under 100 μ s.

Plotting the amount of the empirically-derived adjustment shows a systematic variation that wanders 0.5–1.0 ms from the calendar second mark. It is not clear what causes this, but there is some ambiguity in the correct time to assign from the NTP server reply message. Some of this wandering may be imposed by the NTP server in adjusting to its source NTP server.

These diagnostics show that the GMT(0C) performance in an IRM will be more than adequate to let MiniBooNE identify unambiguously the 15 Hz accelerator beam cycle used for MiniBooNE.

Explanation of the need for GMT0C_T

Most IRM nodes operate at 15 Hz, triggered by a delay of 3 ms from clock event 0x0C. Linac beam is delivered to the Booster 2 ms after this event, so this timing places most nodes cycle operation at 1 ms after Linac beam/Booster injection. But IRMs that support Booster data are scheduled with a delay after the 0x0C event of 35–40 ms, so that their operating cycles begin after Booster beam extraction. One of the early nodes runs LOOPTIME, and its results are shared with all other nodes via periodic multicasting, say, once per minute. The problem is that a late node receiving this updated value of GMT0C will, in effect, see a glimpse of the future, since it will normally advance its own GMT0C only when its cycle activity begins later within the current cycle. By keeping a copy of t(0C) that corresponds with the current value of GMT0C, such a late node can avoid incorrectly updating its own GMT0C when it begins its own cycle.