Manipulating Time in a Logix Environment

Purpose of this Document: This document outlines techniques to manipulate time on a Logix processor. This includes adding/subtracting or interpreting time on these systems

Application Description: There are many uses for time on a Logix processor.

- first fault detection
- fault logging
- processes monitoring
- coordination with a PC
- output scheduling

Many of these applications require some form of time manipulation or time interpretation.

Challenges of Time Manipulation in the Logix platform:
A key differentiator in the controller market is the ControlLogix implementation of time. Time is actually built into a ControlLogix system as an integral part of its operation. The CST clock (coordinated system time) in its simplest form is a backplane clock propagated between all modules on the ControlLogix backplane. This clock is an integral part of coordination between different modules, for example coordination of multiple motion cards, as well as time stamping and scheduling of events on I/O cards.

The processor also keeps a WallClock time. The WallClock is the current real world time.

Real world time (or WallClock time) is stored in the controller and battery backed up. The CST backplane time is not battery backed up. CST time starts counting in microseconds when the controller is powered up from a base date of January 1, 1972 (Version 15 and lower) or January 1, 1970 (Version 16 and higher) . WallClock can then be derived from CST by adding an offset value. This offset is calculated by the controller or can be manipulated manually.

The two main challenges of dealing with time on a Logix controller are:

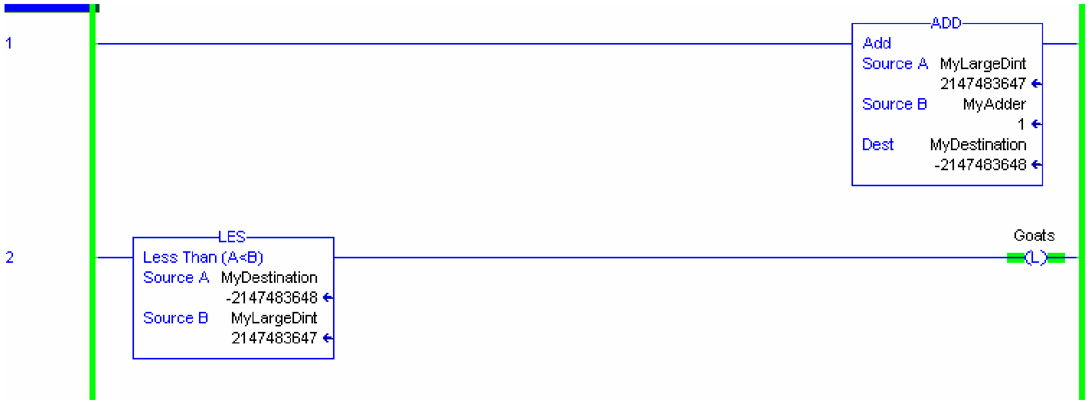
- CST clock and the WallClock are both 64 Bit words. Logix is a 32 bit processor. Storage of the CST and WallClock are in an array of two 32 bit DINTs. This means special consideration must be taken to add or subtract these numbers to compensate for the lower 32 bits rolling up into the upper 32 bits. Logix does not do 32 bit unsigned math. That means you need to compensate for negative numbers that are in reality positive time represented in the 32nd bit.
- CST and WallClock, when returned from I/O modules or the controller, are not always in a man readable format. They are typically returned as two 32 bit DINTs in microseconds from January 1, 1972 (Version 15 or lower) or January 1,1970 (Version 16 or higher). The challenge here is interpreting this time in a man readable format, I.E. Month, Day, Year, Hour, Second, Millisecond.

Some applications may require time stamps to be sorted in order of their occurrence. Sample programs for bubble sort and insertion sort are included on the RSLogix5000/Project/Samples directory when you install Logix 5000.

Math functions on 64 Bit Numbers:
The most common math operation on time is simple addition and subtraction. This is typically used for applications such as:

- Add the WallClock offset to CST to provide WallClock time of the event.
- Add an offset to the current CST to provide a time for a scheduled output card
- Subtract time stamps to get the time deference between events

To get an understanding of the challenge here lets examine the following two simple 32 bit math rungs. The first rung adds 1 to a dint equal to 2147483647 (the largest number you can store in 31 bits). The next rungs compare the product of this addition with the large dint. After adding one you would expect this value to be larger then the original DINT. Note the condition below tests it as if adding one to the number makes it smaller. This is because you have moved into Bit 32 of the DINT which is now a sign bit. Notice the product of this addition is a large negative number, instead of 2147483648.



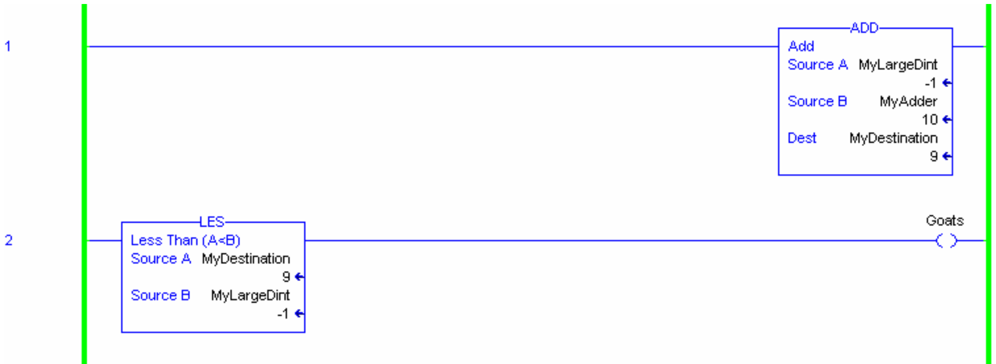
As strange as this may seem, the binary representation of this number is accurate, even though tests in Logix show this value as negative. Continuing to add to this number will continue to produce a less negative number. So the sequence goes:

2147483646
2147483647
-2147483648
-2147483647
-2147483646
…
-1 (Where negative 1 is the largest number represented by a 32 bit number).

Now let's apply this to a 64 bit unsigned time stamp. Time, while itself is unsigned, is represented in Logix as two 32 bit signed DINTs. The lower 32 bits (element [0] of the two DINT array) are the least significant bits of time. Typically addition is performed in this lower 32 bits. The complication arises when the value in the lower 32 bits will exceed a 32 bit DINT after the addition of some offset (number goes past -1).

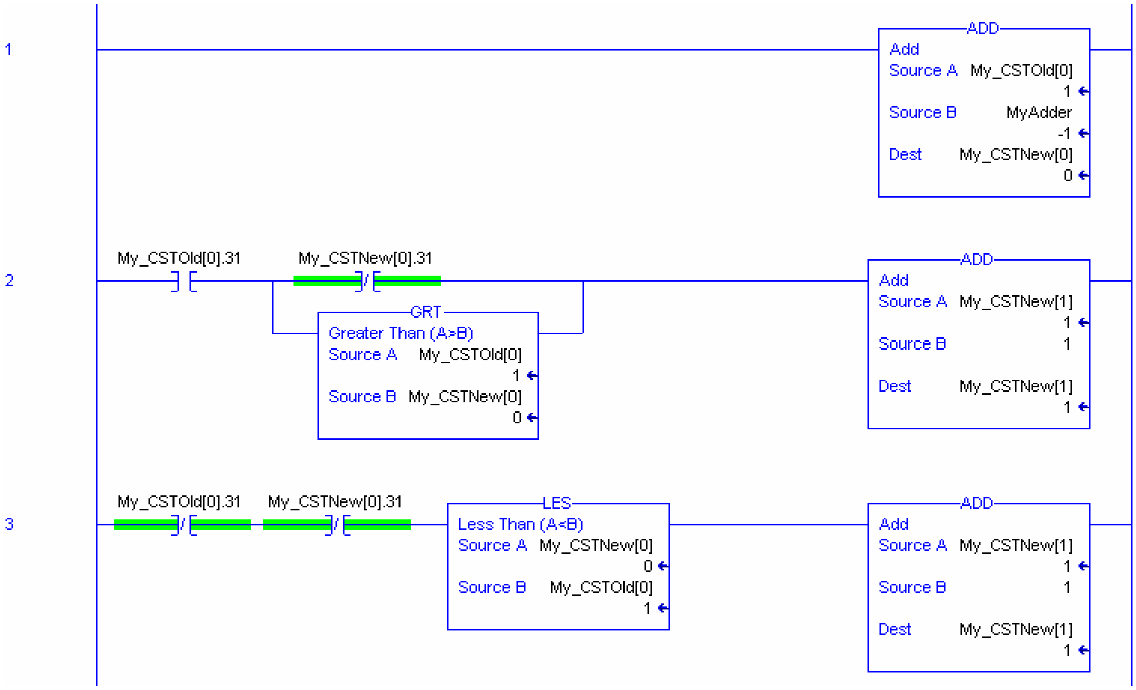
Scheduled output cards, like the 1756-OB16IS, only use the lower 24 bits of the CST time as a schedule. This means math on the upper 32 bits of the word is not required. In fact simple addition using an ADD instruction to the lower 32 bits of the CST will provide a number that can be passed directly to the output card. Any rollover into bits 25 or higher is taken into account by the lower 24 bits in time. Note 24 bits allow you to schedule up to 16777215 microseconds into the future (16.777215 seconds).

Now for other applications, where it is necessary to actually interpret this time or add a value to it for some other reason, it is a simple matter of identifying that the rollover occurred. In the following example, let's look at what happens when we add 10 to the largest number available to Logix 32 Bit Dint -1 or #16 FFFF_FFFF (4294967295 decimal).



In the rung above, we took -1 and added 10 to that number. Now the result of 9 may look okay, however in 64 bit unsigned math, we just rolled over into the upper 32 bits. i.e. we were at (#16FFFF_FFFF) in hex and added ten. The result should be (#16 1_0000_0009) but instead we're at (#16 0000_0009). What this example shows us is that while the lower 32 bits are correct, we need a mechanism to identify that the rollover occurred, and track that change into the upper 32 bits of the 64 bit number. To represent this number as two 32 bit Dint's we should have [H] #16 0000_0001, and [L] #16 0000_0009.

The following sample shows one method for identifying the 32 bit crossover point.



Rung one just adds a number of microseconds (MyAdder) to a CST value (My_CSTOld[0]). The resulting CST is stored in My_CSTNew. Remember this is unsigned 64 bit math using singed DINTs.

Rung two checks for the following conditions if the sign bit is on in the lower original CST time:

- Is the sign bit off in the resultant (that means we've crossed over FFFF_FFFF) and need to add 1 to the upper 32 bits.
- Is the resultant more negative then the original. That means we've crossed FFFF_FFFF and our new number just happens to have the sign bit on. If this is the case, add 1 to the upper 32 bits.

Rung three checks for the following conditions if the sign bit is off in the lower original CST time:

- Is the sign bit of the resultant also off, if so is the resultant less then the original value. If this is the case add one to the upper 32 bits.

Examples:

Let's run thru some examples.

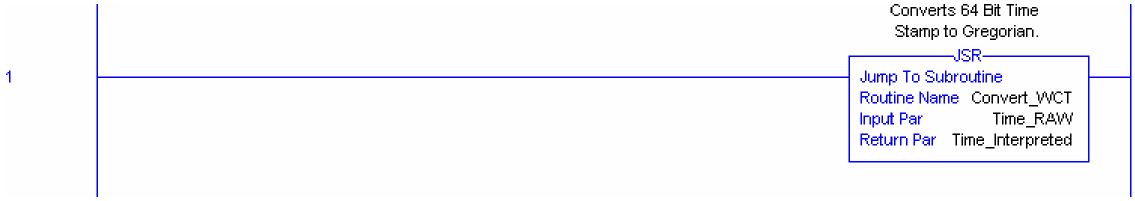
1 + 1 = 2 (in unsigned math = 1 + 1 = 2) no rollover in lower 32 bits.
1 + -1 = 0 (in unsigned math = 1 + FFFF_FFFF = 1_0000_0000) rollover into next 32 bits uses rung 3
-1 + 10 = 9 (in unsigned math = FFFF_FFFF + 0000_000A = 1_0000_0009) rollover into next 32 bits uses rung 2
-1 + -1 = -2 (in unsigned math = FFFF_FFFF + FFFF_FFFF = 1_FFFF_FFFE) rollover into next 32 bits uses rung 2

Note this is just a simple technique for adding numbers to time. It only covers adding less then 32 bits of microseconds to the CST (one rollover) In the event that you need to add more than 4294967295 microseconds to a number (FFFF_FFFF) first divide the number by FFFF_FFFF and add the integer resultant to the upper 32 bits. The remainder of this division is added to the lower 32 bits using the above technique.

When adding CST numbers to other CST numbers it requires both the upper and lower 32 bits be added together. First add the upper 32 bits together, and then add the lower 32 bits using the technique described above. The Logix system has no provisions for storing more then 31 bits in the upper 32 bits of time. The sign bit should always be ignored in the upper 32 bits. This bit is reserved to identify if the time stamp is synched with a CST master.

Interpreting 64 Bit Time Stamps as Day/Month/Year/Hour/Minute/Second/Microsecond:

Decoding a 64 bit time stamp is a complicated operation. It requires a significant amount of code to correctly interpret the time which is represented as the number of microseconds that have occurred since January 1, 1972 (**Version 15 and lower) or** January 1, 1970 (**Version 16 or higher**). A sample application is available on Logix5000 V16 Cd or higher that provides a routine that inputs a CST and outputs time in a Gregorian format (Time_64Bit_Interpreted.ACD).



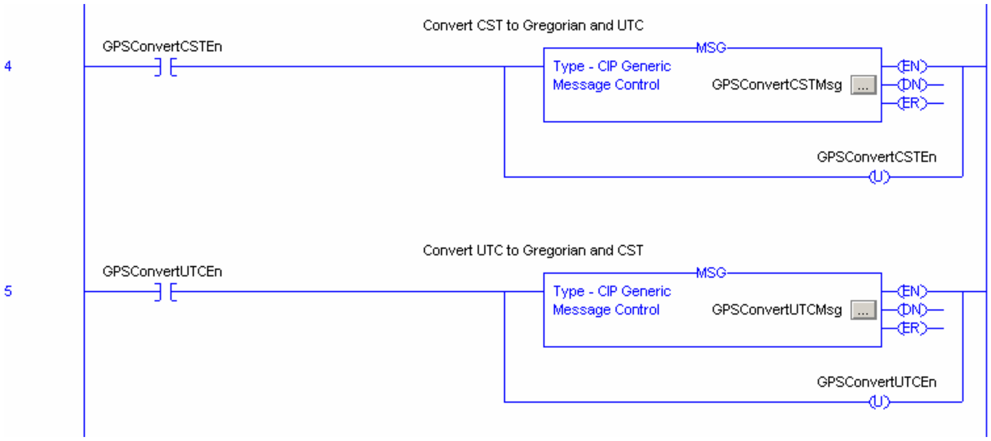
1

Converts 64 Bit Time Stamp to Gregorian.

JSR
Jump To Subroutine
Routine Name Convert_WCT
Input Par Time_RAW
Return Par Time_Interpreted

A couple alternative methods for deriving Gregorian Time:

- If your system contains a 1756HP-GPS module, that module is capable of receiving time stamps and returning them in Gregorian time.



4

GPSConvertCSTEn

Convert CST to Gregorian and UTC

MSG
Type - CIP Generic
Message Control GPSConvertCSTMsg

GPSConvertCSTEn

5

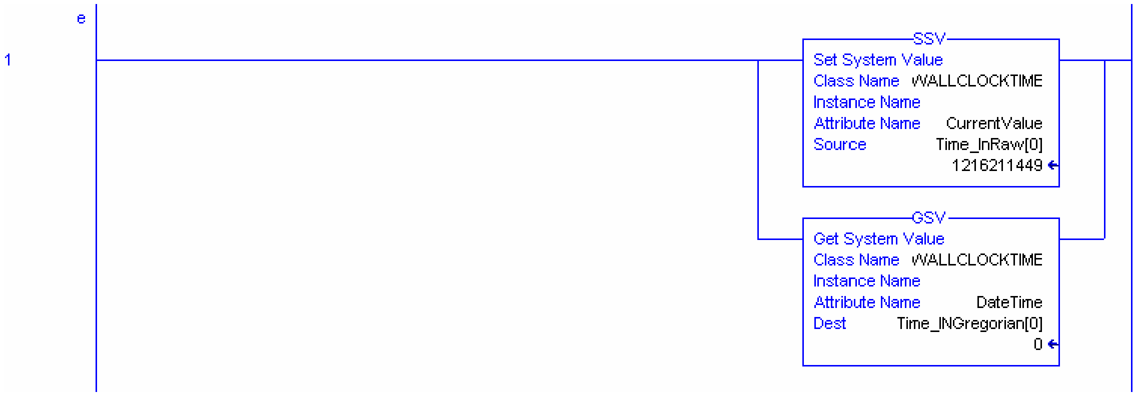
GPSConvertUTCEn

Convert UTC to Gregorian and CST

MSG
Type - CIP Generic
Message Control GPSConvertUTCMsg

GPSConvertUTCEn

- If you do not care about the current WallClock of the controller in use, you can write this 64 bit number to the controllers WallClock object using an SSV. Then immediately read back the WallClock time in Gregorian format. This will be inaccurate by the amount of time that the instructions take as the WallClock will begin counting in microseconds as soon as the SSV is complete. If your application can withstand a few milliseconds inaccuracy this technique can be valid.



e

1

SSV
Set System Value
Class Name WALLCLOCKTIME
Instance Name
Attribute Name CurrentValue
Source Time_InRaw[0]
1216211449

GSV
Get System Value
Class Name WALLCLOCKTIME
Instance Name
Attribute Name DateTime
Dest Time_INGregorian[0]
0

- Please indicate where in the control architecture this solution has been applied:

☐ From the HMI workstation to a higher-level computer system
☐ At the HMI level
☐ Between the HMI workstation and the controller(s)
☐ From controller to controller (peer to peer or peer to lesser controller)
☐ Controller code specific to the application or required for the application
☐ From controller(s) to field devices
☐ At the field device
☐ Other aspects of the application

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe/Middle East/Africa: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 6356-9077, Fax: (65) 6356-9011

Publication 000-0000-0000 – Month, 2003