McLANE® RESEARCH LABORATORIES, INC.

Technical Support Bulletin 2020-03	MMP and ITP v5.40 Technical Details	February 13, 2020

Alert Level: CRITICAL
Instruments: MMP and ITP
Software Versions: MMP/ITP firmware v5.40
Hardware Version: CF2
Subject: Technical Details of firmware v5.40
Summary: As summarized in Release Note MMP/ITP v5.40, MMP/ITP v5.40 includes the following:

- Critical fix that resolves the Real Time Clock (RTC) bug by using the correct time base of 2000.
- Implemented real time clock maintenance routines explained in this document.
- A Profile Independent Telemetry Session (PITS) was added to the sleep loop.

Technical Details: A copy of Programmer Technical Details is provided below.

MMP 5.40 Programmers Notes

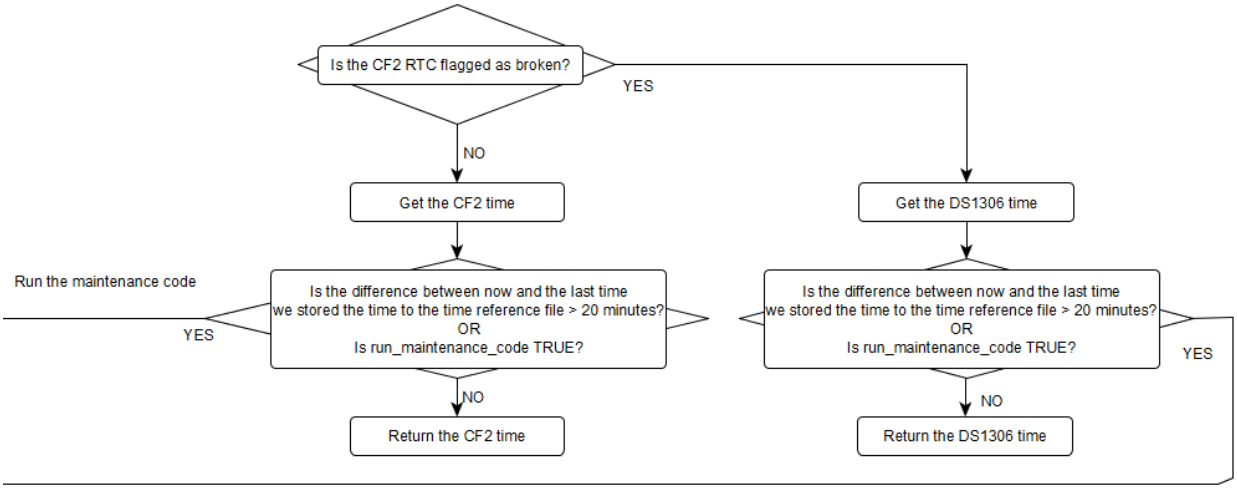
The RTC_Handler

The following walks the reader through the flowchart that accompanies this document.

MMP 5.40 includes an "RTC Handler" written on top of the existing RTC code in an attempt to detect, correct, and recover from unexpected RTC problems.

The RTC_Handler uses status flags that represent the current state of each RTC. A RTC can be CORRECT or WRONG, BROKEN, and UNREPAIRABLE. These flags are used throughout the RTC maintenance routines.

When Get_Time() is called, if the CF2 RTC is flagged as CORRECT, the CF2 RTC value is used. If the CF2 RTC is flagged as anything other than CORRECT, we fall back on the DS1306.

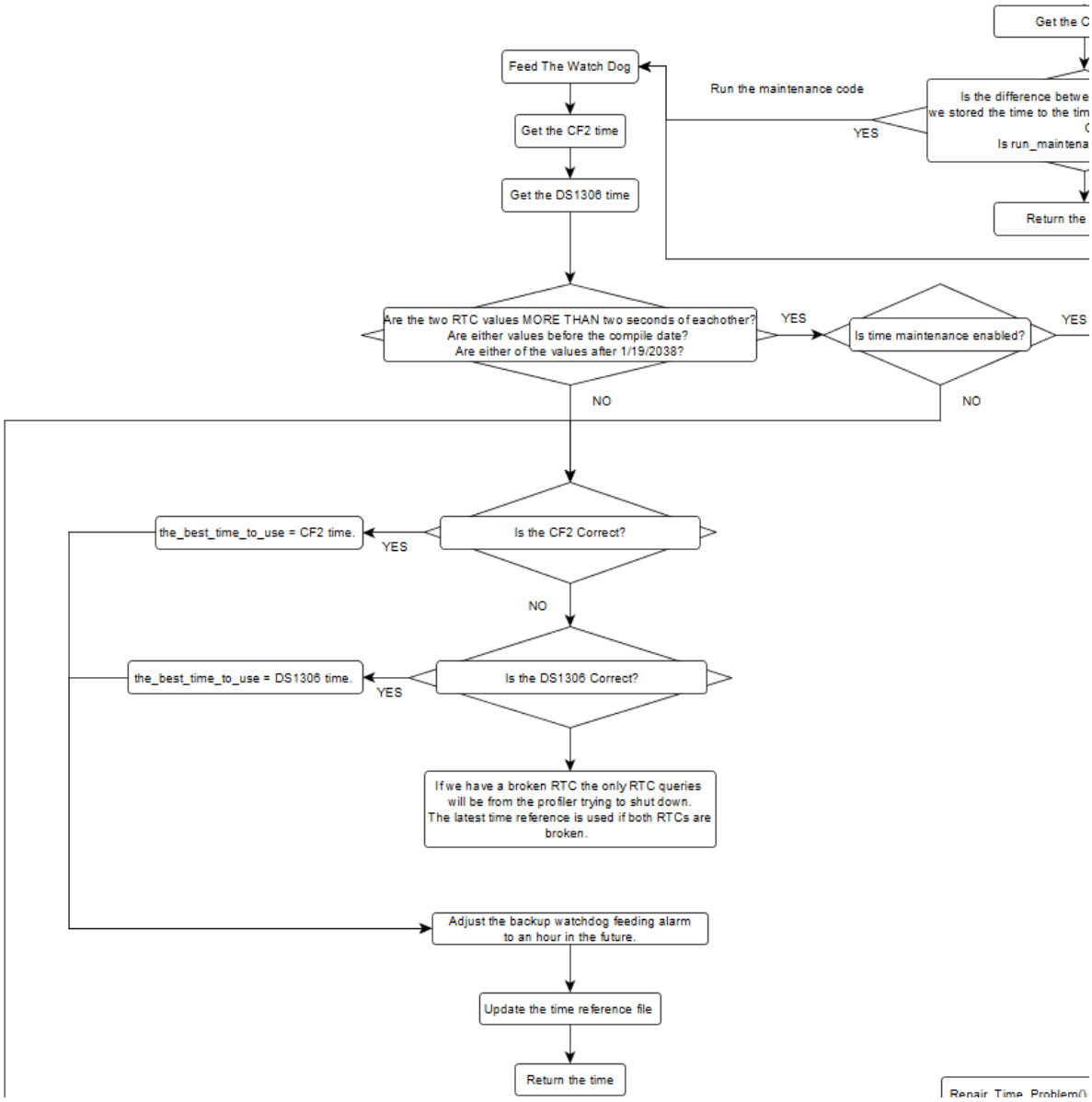


The DS1306 is the more accurate RTC between the two, but communicating with it takes longer than retrieving the time from the CF2 RTC. While profiling and retrieving sensor data the RTC is called so often, that if you read the DS1306 every time, the Profiler's ability to process sensor data is limited.

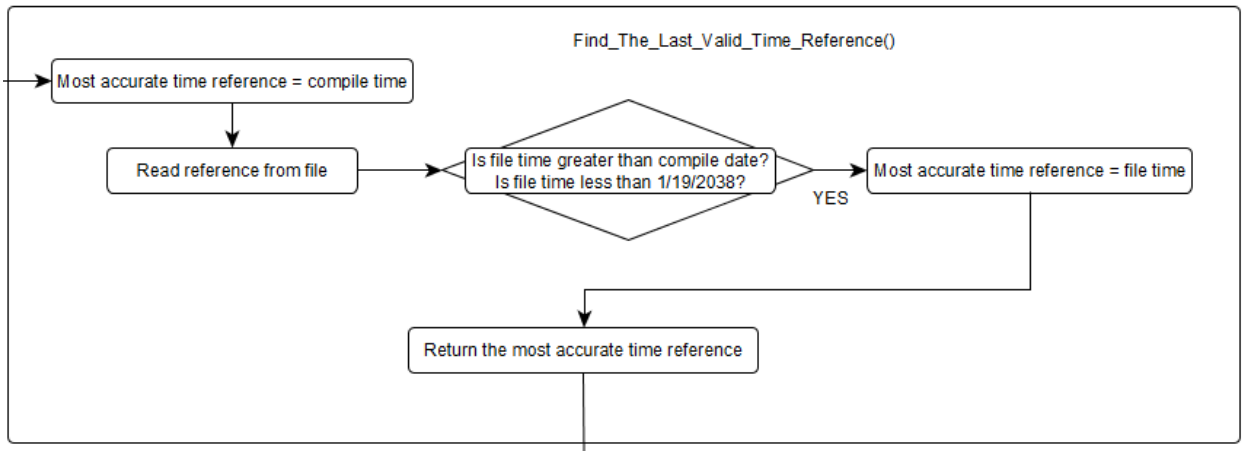
Real Time Clock Maintenance

The Profiler has a file where a validated timestamp is stored every 20 minutes. When Get_Time() is called, the RTC value is compared to the last time stored to the file. If the difference between the values is less than 20 minutes, Get_Time() immediately returns the RTC value. If the difference is more than 20 minutes, RTC maintenance routines are executed.

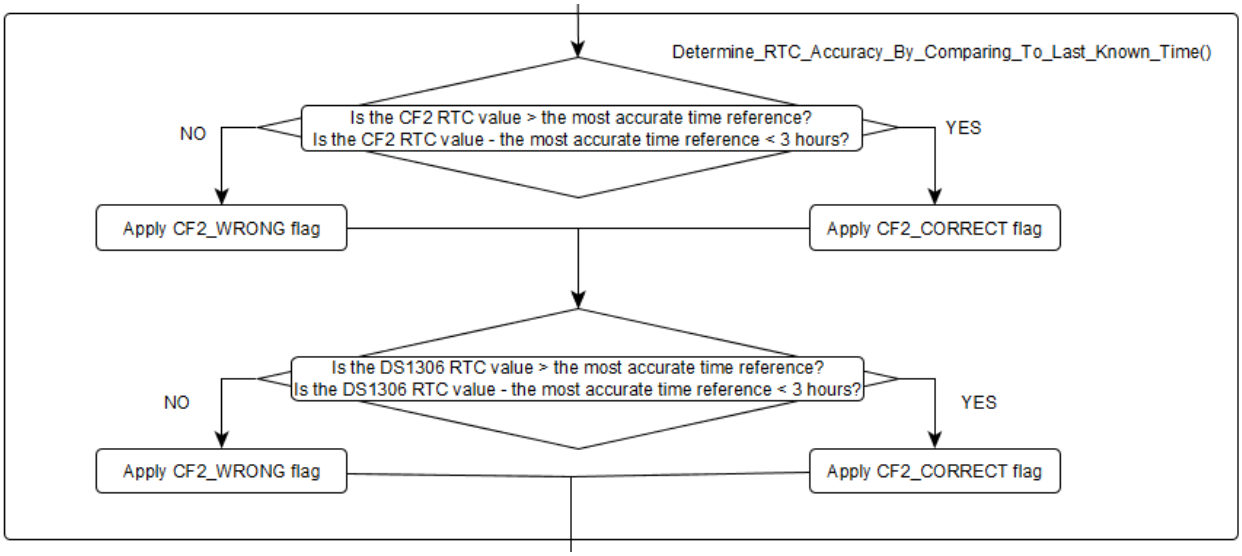
RTC Maintenance is executed at least every ~20 minutes. The watchdog is fed, and RTC values are compared against each other. If the values are within 2 seconds from each other, less than 1/19/2038, and greater than the compile timestamp, then we assume that the RTCs are working properly. The backup watchdog alarm is updated, and the time reference file is updated.



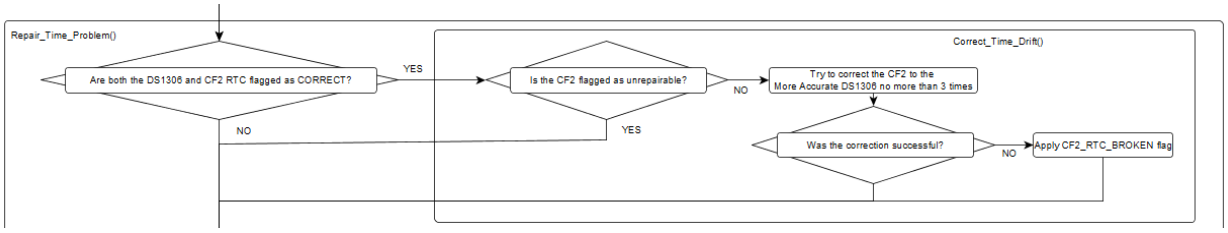
If the RTC values are more than 2 seconds from each other, a problem is suspected. The first step to repairing the problem is finding the most accurate time reference. If the value stored to the time reference file is greater than the compile time, and less than Tuesday, January 19, 2038, then the most accurate time reference is the time stored in the time reference file.



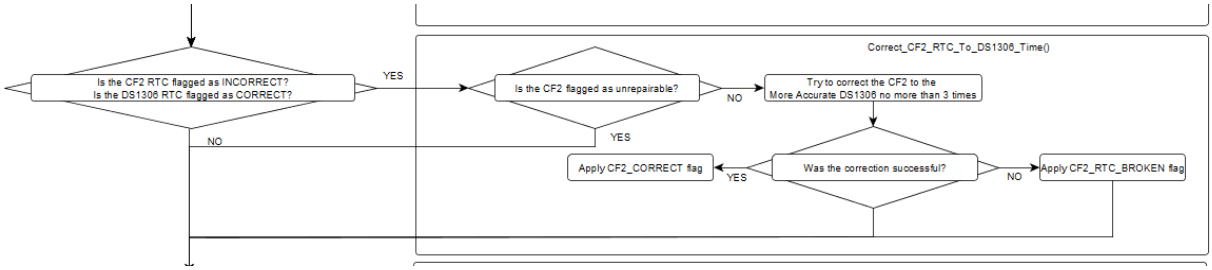
Once the best time reference is found, it is used to determine which RTC is more accurate.



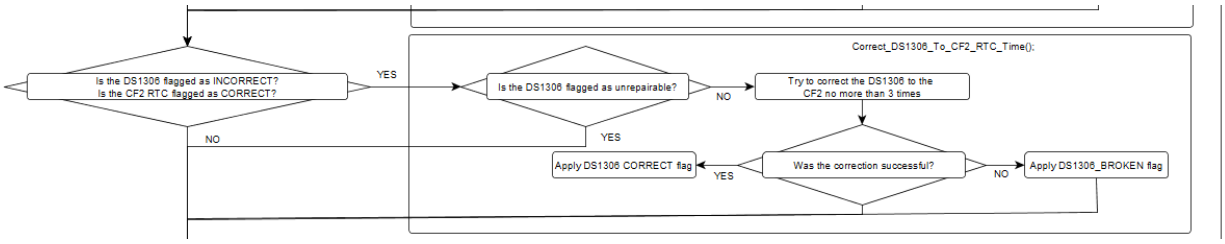
An RTC value should never be more than three hours in the future from the value in the time reference file. If the RTC value is greater than the most accurate time reference found in the last step, and less than three hours in the future from the most accurate time reference, the Profiler considers the RTC "correct". Otherwise, the RTC is flagged as incorrect. Next the Profiler attempts to correct any problems that it found in the time verification process.



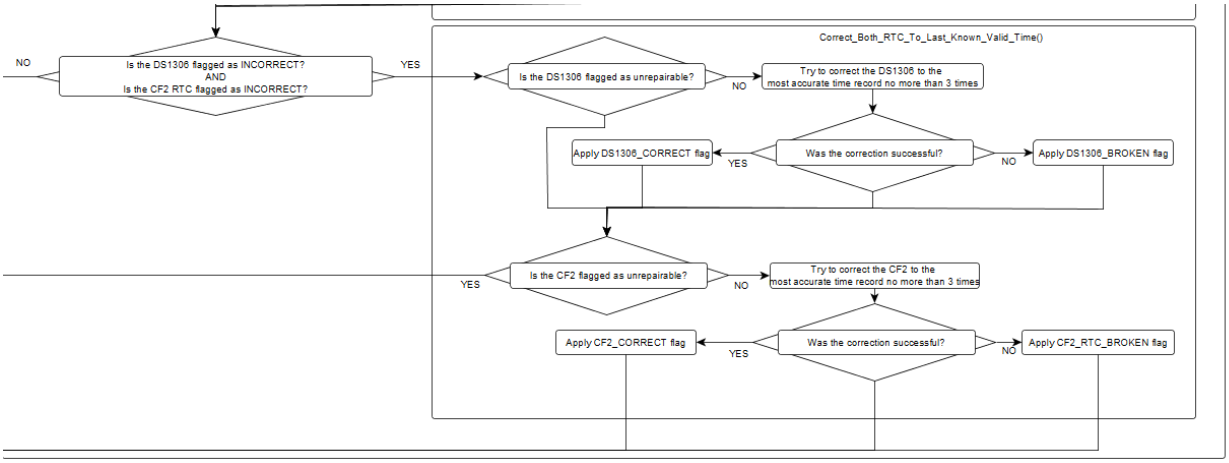
If the Profiler entered the time verification routine and determined that both RTCs were past the last time reference value, but less than three hours in the future, then both RTCs are flagged as “correct”. If the profiler enters the Repair_Time_Problem() routine with two “correct” RTCs, then we are probably dealing with time drift between the two RTCs. The Profiler attempts to correct the CF2 RTC to the more accurate DS1306 value.



If the Profiler enters Repair_Time_Problem() with a correct CF2 RTC and an incorrect DS1306, it attempts to correct the CF2 to the DS1306.



If the Profiler enters Repair_Time_Problem() with a correct CF2 and an incorrect DS1306, it attempts to correct the DS1306 to the CF2.



If both RTCs are flagged as incorrect, the profiler attempts to correct them both to the most accurate time record.

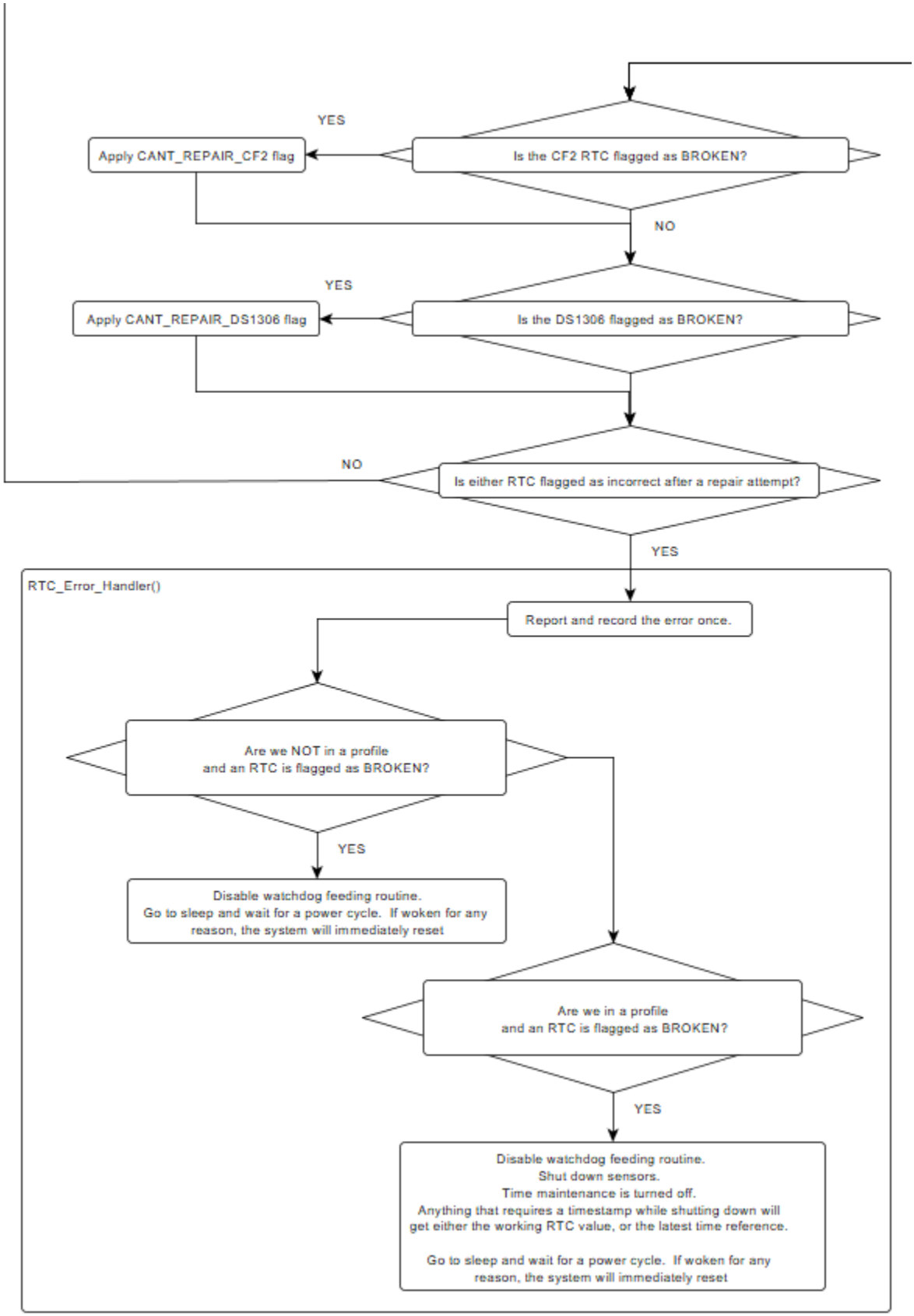
If Repair_Time_Problem() successfully corrects the incorrect RTC we will exit the function with two RTCs flagged as correct. If we had a problem correcting an RTC, it will be flagged as broken. If an RTC is flagged as broken after we exit Repair_Time_Problem(), it is immediately flagged as unrepairable so we won't continue to report errors and attempt to correct it. If either RTC is flagged as broken after an attempt to correct it, RTC_Error_Handler() is called.

The rare RTC problems we have seen on the Profiler (mentioned in the bug report) have been difficult to recreate, but a power cycle usually fixes the problem.

If we enter the error handler with a RTC flagged as broken:

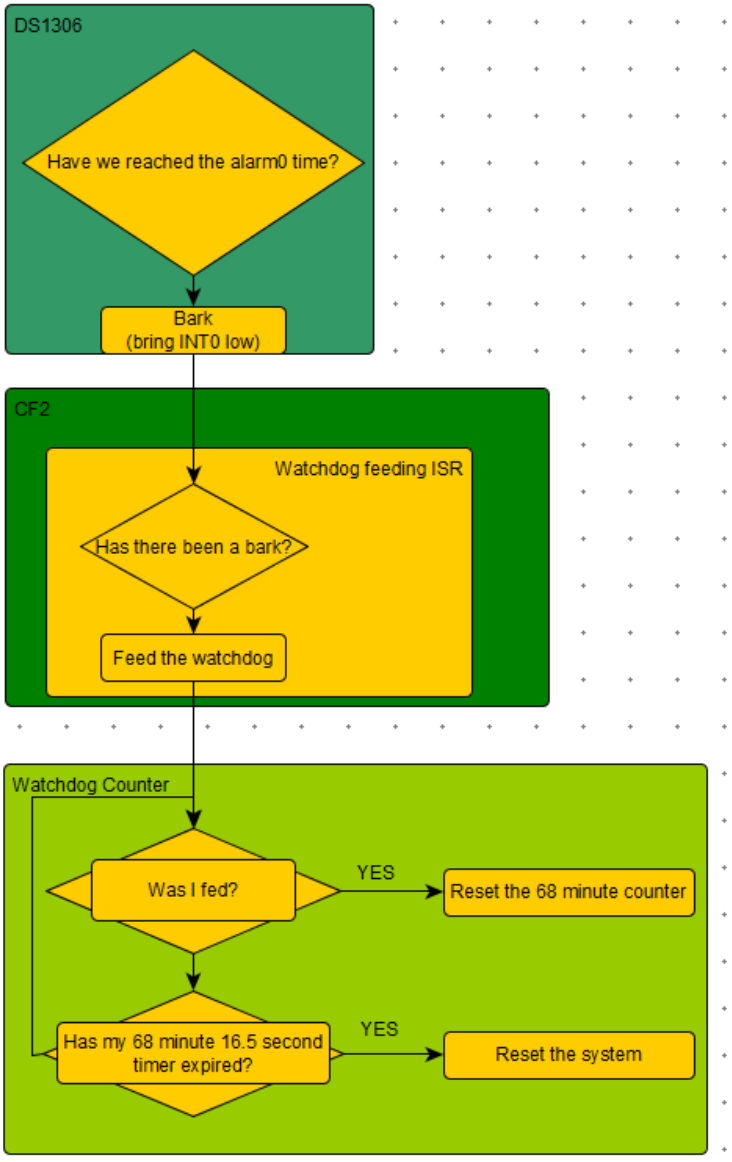
- If the Profiler is not in the middle of a profile, RTC_Error_Handler disables the watchdog feeding routine in order to force a watchdog system reset, and then enters sleep mode until a watchdog reset occurs.
- If the Profiler is profiling, the profile is terminated immediately. The watchdog feeding routine is disabled, and the system goes to sleep until a watchdog reset occurs.

McLANE®
RESEARCH LABORATORIES, INC.



Watchdog Changes

The watchdog will reset the system if a 4096 second (68 minute 16 second) timer isn't reset before it expires. The DS1306 has an alarm that is programmed to bark at regular intervals, that bark is wired to an IRQ pin that is attached to a "watchdog feeding" interrupt service routine that resets the watchdog timer.



This is how the watchdog has worked up until now. The watchdog is fed using a digital input / output expansion chip that relies on SPI transactions inside of an interrupt service routine. If possible, I prefer

to be in control of when the watchdog feeding routine is called if it is using a shared SPI bus.

RTC maintenance code feeds the watchdog every ~20 minutes. Every time we feed the watchdog in the maintenance code, we also schedule the DS1306 bark alarm an hour in the future. Testing up to this point has proven that this is enough to keep the watchdog satisfied without the interrupt-driven feeding routine being called. If there is a path somewhere in the program that leads to a loop where the RTC isn't checked for more than 4096 seconds, we are able to fall back on the watchdog feeding ISR to keep the watchdog timer from expiring.

RTC Maintenance in the Profiler Program

At the beginning of program the RTC_Handler is initialized by providing Get_Time() with an argument that forces the RTC maintenance routines. This will correct the time to the most accurate time it can come up with at the start of the program.

If a user is present, the time can be adjusted. Whatever time is entered is recorded to the time reference file.

At the start of a deployment, the user is asked to verify the time. This value is stored to the time reference file. A deployment cannot be started without a user verifying the system time.

Before starting each profile, the RTC maintenance routines are forced. After the profile and telemetry session is completed, the "in a profile" flag is removed. After the "in a profile" flag is removed, the Get_Time() verification / correction code is executed before we schedule the next event. We can't schedule new events without an accurate RTC value. If there is a RTC problem that could not be corrected, the watchdog is disabled and we wait for a system reset.

We can only schedule and enter the sleep loop with an activated watchdog feeding routine if we have a valid time and time reference. The profiler wakes up every 20 minutes during the sleep loop to do some housekeeping. Every time we wake up the RTC maintenance code is executed.

Profile-Independent Telemetry Session

A profile-independent telemetry session was added to the Profiler sleep loop as an additional safeguard against unexpected problems leading to a Profiler getting stuck in the sleep loop. If a Profiler is scheduled to wake up every day, the user might want to program a profile-independent telemetry session to happen every 2 days. If the user wants a heartbeat from the MMP, they could program a profile-independent telemetry session to happen every 20 minutes.

Option <Y> of the Advanced Interface Menu allows the user to program an interval at which the Profiler will execute a telemetry session while inside of the sleep loop.

```
Config: MPP_IM                                                  CF2 V5.40

                    Pattern Profiler
                   Advanced Interface
                Thu Feb 13 00:29:32 2020

 <0> full Speed                 0.250 dbar/sec
 <1> pressure Rate threshold    0.045 dbar/sec
 <2> pressure rate Time limit      60 seconds
 <3> Sensor warmup                120 seconds
 <4> Sensor warmdown              120 seconds
 <7> Configure active sensors     ----------
 <8> Configure offload queue      ----------

 <A> Annunciate comm traffic      YES
 <B> Backtrack iterations           3
 <D> Display verbose messages      NO
 <F> IMM use Force capture        YES
 <G> Get pressure during ramp      NO
 <H> History reset
 <I> Infinite deployment      ENABLED
 <J> use Slower uart clock         NO
 <K> IM ACK|NAK|ETK reply timer   100 seconds
 <L> IM Listening loop timer       40 seconds
 <M> profiling Mode           PATTERN
 <N> adjust profile couNter        -1
 <P> caPture file enabled          NO
 <S> IMM configure Surface         NO
 <T> Terminate profile in ramp     NO
 <U> Ramp duration                 30
 <W> IMM send Wakeup tone         YES
 <Y> Profile-independent telemetry interval       2 Days 0 Hours 0 Minutes

 <X> Save changes  <R> Reset factory defaults  <^c> Cancel changes

 Selection [] ? y
Enable profile-independent telemetry session [Y] ?

Enter the emergency IMM session interval
Days: (0-14) [2] ?
Hours: (0-23) [0] ?
Minutes (0, 20, or 40) [0] ?

Accept values? [Y] ?
```

While inside the sleep loop, the profiler wakes up at the sleep loop housekeeping interval of 20 minutes. The user programs the profile independent telemetry interval in days, hours, and minutes. That time is broken down into 20 minute increments. This is done so the telemetry session isn't dependent on a valid RTC value. Every time the profiler wakes up inside of the sleep loop, a counter is incremented. If that counter value is greater than or equal to the number of 20 minute increments programmed by the user, the Profiler executes a telemetry session inside of the sleep loop.

Every time the Profiler enters the sleep loop, that counter is reset.