Using SpaceWire Time Codes for Spacecraft Time Synchronization

SpaceWire Missions and Applications, Short Paper

Susan C. Clancy, Mazen M. Shihabi, Krisjani S. Angkasa
Flight Communications Systems Section
Jet Propulsion Laboratory
Pasadena, California 91109 USA

Susan.Clancy@jpl.nasa.gov, Mazen.M.Shihabi@jpl.nasa.gov, Krisjani.S.Angkasa@jpl.nasa.gov

Abstract—This paper describes how SpaceWire Time Codes can be used for synchronizing time within various subsystems of a spacecraft as well as, maintaining a common time reference needed for coordinating operations within a spacecraft. The algorithms to account for inaccuracies in the time distribution method were based on the NASA-4009 Space Telecommunication Radio System (STRS) standard [1], which defined an interface for synchronizing clocks running at different tick rates and tick resolutions.

Index Terms—Relevant indexing terms: SpaceWire, SpaceWire Time Codes, SpaceWire Time Distribution Protocol, CCSDS Unsegmented Time (CUC), Space Telecommunications Radio System (STRS).

I. INTRODUCTION

Spacecraft systems are typically comprised of many subsystems, each with their own clock running at different tick rates and with varying performance, which can degrade over time. Clock synchronization becomes very important in cases where commands and activities need to be correlated with a common time reference and for attitude determination based on current time or predicted position propagated over a period of time.

Subsystems needs to know what time it is in order to perform synchronized activities, or to time-tag telemetry that can be correlated with operations in other subsystems. One subsystem equipped with a Ground Navigation Satellite System (GNSS) receiver can maintain an accurate reference of time and can act as the time “master” to distribute the time to other nodes connected via SpaceWire.

II. SPACECRAFT TIME SYNCHRONIZATION METHODS

There are two common methods used for synchronizing time on a spacecraft: (1) a periodic “message” based method performed in software and (2) a periodic “hardware tick” based method performed in hardware or firmware.

The “message” based method uses a “master” to generate a “tick” message at specific intervals and sends a time message to the “slaves” at a specific “tick”. The “slaves” update their time at a time boundary after the time message is received. In the example below, the “tick” message is sent 100 times per second, and the time message is sent once per second prior to the one second time boundary.

![Fig. 1. Time Synchronization “Message” Based Method](image1)

The “hardware tick” method uses a “master” to send a “tick” signal to all the “slaves”, who will then increment their own slave clock. The hardware clock oscillator used to generate the clock tick signal is usually a Temperature Compensated Crystal Oscillator (TCXO) or Ovenized Crystal Oscillator (OCXO) with accuracy better than 1 part per million.

![Fig. 2. Time Synchronization “Hardware Tick” Method](image2)
III. SPACECRAFT TIME SYNCHRONIZATION CHALLENGES

The challenges in synchronizing spacecraft time are similar to those in ground-based systems:

A. Latency – the time it takes to transfer and respond to a time update. Each spacecraft subsystem must account for latency and be tolerant within a measured minimum and maximum range. A technique for measuring latency is described in the SpaceWire Time Distribution Protocol [2].

B. Jitter – the intermittent delay in the path between the master sending the time and the slave receiving and updating their time. Each spacecraft subsystem must tolerate a measured maximum jitter.

C. Drift – the variation in the clock tick rate due to oscillator performance, which typically degrades over time and varies with temperature. The time “master” clock must be calibrated periodically to account for the drift in the time conversion. The drift can be accounted for as a clock rate correction [2] to mimic the actual clock rate changes.

D. Time conversion – the different clocks may tick at different rates and a conversion from the hardware clock value to the time representation unit (usually in seconds) is applied using the clock tick rate, clock hardware value, and an offset, which typically includes drift. The conversion algorithm needs to account for latency, varying jitter, and clock degradation.

A further complication is that the performance of the clock oscillators in various parts of the system may be orders of magnitude different: a spacecraft computer may have a clock with 10 ppm performance, while spacecraft radios and GNSS receivers may be accurate to parts per billion (ppb). The system design, however, may be that all systems need to follow the time kept by the spacecraft computer, so the time distribution method must allow a better clock to follow a poorer clock, which is different than the typical Network Time Protocol (NTP) architecture, where clocks at a lower stratum follow more accurate clocks at a higher stratum.

IV. STRS TIME SYNCHRONIZATION METHOD

The NASA-STD-4009 Space Telecommunications Radio System (STRS) architecture standard [1] defines some time related functions and corresponding Application Programming Interfaces (APIs) for getting, setting, and synchronizing time. These functions are used by applications to maintain and coordinate time derived from different clocks that may have different tick rates and resolutions.

Note that the reference clock may or may not have a higher performance and stability than the monitored clock. The purpose is to synchronize the clocks and not to maintain the correct time. The reference clock and managed clock can exist on the same local host or on different hosts but can be synchronized to report the same time.

The core concept of the STRS clock model is that the underlying clock is allowed to run unhampered, and the relationship between the raw clock and “time” is encapsulated in the API which provides a standardized way of getting and setting time based on calling API functions that can account for latency, jitter, and drift using conversion data. This conversion data is set to values that initially synchronize the reference clock with the managed clock. The conversion data can be updated periodically to continuously account for drift.

The linear conversion algorithm commonly used to compute time, converts hardware clock ticks to time in seconds using the oscillator clock rate and hardware clock ticks as follows:

\[ time = (\text{clock rate} \times \text{clock ticks}) + \text{offset} \]

The STRS time conversion algorithms include additional adjustments to the rate and offset to account for the difference between two clocks plus the latency, drift, and even jitter as follows:

\[ \text{STRS time} = ((\text{clock rate} + \text{adjust rate}) \times \text{clock ticks}) + (\text{offset} + \text{adjust offset}) \]

Figure 3 below shows an implementation of an STRS time interface that synchronizes a local reference clock and a local managed clock. The conversion data is applied when getting the time via the STRS_GetTime API function which converts the clock value to a time in seconds and sub seconds.

![Fig. 3. STRS Time Synchronization Method](image-url)
V. **SPACEWIRE SPECIFICATION FOR TIME CODES**

The SpaceWire Protocol Standard [3] includes the definition of the time interface with Time Codes and the TickIn and TickOut signals. The key features in any implementation are:

A. Time Code generation or receipt can be enabled or disabled.

B. The Time Code rate is generated by a “master” and can be configured to send Time Codes at a specific rate.

C. The Time Code is a specific type of SpaceWire message containing a Time Code identifier and Time Code counter. The Time Code counter is an incrementing 0 to 63 integer value and any missing Time Code can be detected and reported by firmware using this counter.

The Time Code TickIn / TickOut signals can support an interface to a software interrupt line and/or hardware signal going to a hardware clock. The time “master” (aka initiator) can generate a software interrupt for each tick using the TickIn signal. Using this TickIn interrupt, a “slave” (aka target) can implement a SpaceWire “derived clock” to align the tick generation with the time message.

VI. **CCSDS TIME MESSAGE FORMAT**

The CCSDS Unsegmented Code (CUC) Time Specification [4] is a proposed standard for specifying time as a number of seconds and sub-seconds.

<table>
<thead>
<tr>
<th>CCSDS Unsegmented Code</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P-Field</td>
<td>T-Field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st</td>
<td>2nd</td>
<td>CoarseTime</td>
<td>FineTime</td>
</tr>
</tbody>
</table>

Fig. 4. CCSDS Unsegmented Code (CUC) Format

The fields in the time announced message are as follows:

<table>
<thead>
<tr>
<th>Unsegmented Code</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P-Field</td>
<td>T-Field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addr</td>
<td>0xF1</td>
<td>32-bit time [sec]</td>
<td>16-bit [not used]</td>
</tr>
</tbody>
</table>

Fig. 5. Time Announced Message Format

VII. **TIME SYNCHRONIZATION DEMO**

The first goal was to demonstrate the ability to compensate for time distribution inaccuracies due to latency, jitter, and drift using the STRS time API. The second goal was to demonstrate time distribution using SpaceWire Time Codes and the CCSDS CUC formatted time message.

In the first test, the time synchronization was performed on the SDR using the Clock Calibration waveform component (CLKCAL) to synchronize two different clock “kinds” on the SDR. The CLKCAL waveform (1) computes the delta between the reference clock time and managed clock time, (2) computes the drift detection value for each clock, (3) reports any time delta or drift detection, and (4) synchronizes the managed clock to report the same time as the reference clock. The STRS time API is used by CLKCAL for getting, setting, and synchronizing the time.

In the second test configuration, the CLKCAL waveform was integrated with the SpaceWire Time interface. The SpaceWire time interface on the SDR “slave” was implemented as a “waveform” component with counterparts running in both firmware on a Field Programmable Gate Array (FPGA) and software running on the SDR Sparc computer.

The SPW waveform continuously (1) receives the time codes, (2) maintains a Time Code tick counter, (3) captures the time sent in the SpaceWire time messages, (4) sends periodic notifications at synchronization intervals and (5) makes the time available to other waveforms.
minimum should not be 0 since there will always be some amount of jitter. The threshold minimum value can be determined by analyzing the delta values over a period of time.

Any delta above the threshold minimum but below the rate adjustment maximum will cause a rate adjustment update to synchronize the clocks. The rate adjustment is included in the conversion data used in the time conversion algorithm. This is the smoothest update method. Any delta between the minimum and the incremental adjustment maximum will use an incremental adjustment over a period of time. Incremental updates will be made until they add up to the desired delta. This adjustment period can be longer to make smaller incremental updates or shorter to make bigger incremental updates. Any delta above the incremental adjustment maximum will cause a time “jump”. A “jump” is not desired when the managed clock is used for time based computations or activities but is a common method used for updating or synchronizing time during initialization.

The clock drift is obtained by capturing a counter for each clock at specific intervals. This counter should remain constant unless the clock is drifting. Watermarks are used to track the range of drift for each of the clocks. A drift watermark reporting threshold maximum value is used to determine when to report drift. This reporting threshold can be 0 to always report any detected drift or a value that must be exceeded before the drift is reported.

VIII. TEST RESULTS

The initial tests run on the SDR show the STRS time interface successfully synchronizing two different clock “kinds” that exist on the same SDR. The data below (in red) shows the software detecting the delta above the threshold, and performing the synchronization.

![Fig. 8. Clock Synchronization](image)

The clock delta and drift reported by the CLKCAL waveform used inputs distorted by the jitter introduced by the software itself due to running in a multitasking environment on both the “master” and “slave”. This artificial input data was useful in developing and testing the clock synchronization thresholds and synchronization response. The use of an independently generated counter latched at fixed intervals as described in earlier work in [2] and a “distributed” interrupt generated via the TickOut signal as described in [6] are needed to account for the real inaccuracies introduced by latency, jitter, and drift.

The synchronization parameters that were tested included thresholds to control whether time was updated gradually or immediately in one-time jump.

The following test result shows the “threshold min.” should be set to 6 usec to avoid synchronization for changes smaller than the expected. 1 to 5 usec range. Based on this example, the changes above 5 usec would result in a clock synchronization.

![Clock Delta](image)

In earlier tests on the SDR, the CLKCAL waveform attempted to poll the received Time Code counter value to increment the Time Code virtual clock ticks. These tests intermittently failed when generating Time Codes at 100 per second. The “slave” reported a missed tick error when the Time Code value did not increment as expected, although this issue was not encountered when Time Codes were generated at once per second.

The TickOut interrupt interface and a latched counter interface have since been implemented in the SDR FPGA firmware to mitigate these issues. The TickOut interrupt unit tests showed that software increments the SpaceWire DCLK virtual ticks properly. However, tests using these mechanisms integrated with CLKCAL are planned for the future.

IX. CONCLUSIONS AND FUTURE WORK

The STRS time API does accommodate synchronizing various clock "kinds" using clock compensation data to mitigate inaccuracies (latency, jitter, drift) in a time distribution system.

Synchronization tolerance ranges (i.e. thresholds) can be used to determine which method to use for synchronizing clocks and when to correct for drift. Future work is needed to establish the tolerance ranges for synchronizing clocks using the SpaceWire Time Distribution Protocol such as those described in [2] and [6].

The SpaceWire Time Codes are useful for creating a virtual clock on hosts connected via SpaceWire. This SpaceWire virtual clock can be implemented on a "slave" host that may not have a clock.
ACKNOWLEDGEMENT

The authors wish to acknowledge the following individuals for their significant contributions: James P. Lux, Minh Lang and David E. Robison from the Jet Propulsion Laboratory, and David Chelmins and Larry M. Vincent from the NASA Glenn Research Center.

This work was carried out at the Jet Propulsion Laboratory in Pasadena (JPL), California, under contract with the National Aeronautics and Space Administration, for the SCaN Testbed Project. References herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the U.S. Government or the Jet Propulsion Laboratory.


REFERENCES