CCSDS Time Distribution over SpaceWire

Sandi Habinc, Marko Isomäki, Daniel Hellström

Aeroflex Gaisler AB
Kungsgatan 12, SE-411 19 Göteborg, Sweden

sandi@gaisler.com

www.Aeroflex.com/Gaisler

Introduction

Time synchronization in spacecraft has always been important. Previously it has been done via dedicated signals or deterministic on-board buses (e.g. MIL-STD-1553 or OBDH).

With the advent of SpaceWire point-to-point links and routing switches being used for critical control functions the need for accurate time synchronization via this network has arisen.

With SpaceWire-D (Deterministic) time synchronization has become a necessity, requiring a resolution of at least 10 us, and an accuracy of around 1 us (10% window for a kill-zone).

This presentation will focus on various issues related to time synchronization in SpaceWire networks, specifically latency, jitter and drift, but also on time distribution protocol features.

Time support in SpaceWire

- SpaceWire is an asynchronous network i.e. there is no common clock signal being distributed for the communication meaning that each node is responsible for its own clock
- No means for handling drift caused by unstable oscillators or crystals
- No support for automatic time message and pulse distribution
- Rudimentary time-code transmission
- No means for handling delays and jitter caused by routing

Features of a time protocol

A time distribution protocol can have the following features:

- Time message: carries the overall time information

- Time synchronization: synchronizes the time information (also called a synchronization pulse)

The SpaceWire standard already has a mechanism for carrying time information – the Time-codes. The Time-Codes provide 64 unique incrementing values, which do not provide a sufficient range for on-board time (normally in the range of 2**32 seconds).

The Time-codes can however be used for carrying part of the time information and be used for time synchronization.

Other features are also desirable, e.g. time initialization.

CCSDS Unsegmented Code (CUC)

CCSDS Usegmented Code (CUC) is the most commonly used format supported by CCSDS.

The time is counted as seconds ad sub-seconds from an époque, with seconds being represented as positive powers of two, and sub-seconds as negative powers of two.

The code includes a pre-amble field that defines the number of bytes being used for representing seconds and sub-seconds, respectively. The standard support 32 and 24 bits, respectively. There is an ongoing revision to support additional bits.

CCSDS Unsegmented Code														
P-Field		T-Field												
1st	2nd	Coarse Time								Fine Time				
		2^{31} 2^{24}	2^{23} 2^{16}	2^{15} 2^{8}	2^{7} 2^{0}	2^{-1} 2^{-8}	2^{-9} 2^{-16}	2^{-17} 2^{-24}						
0 7	8 15	0 7	8 15	16 23	24 31	32 39	40 47	48 55						
8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits						

CCSDS Unsegmented Code Transfer Protocol (CUCTP)

CUCTP is a experimental protocol for maintaining synchronization within a SpaceWire network developed in cooperation with ESA, SciSys and Astrium.

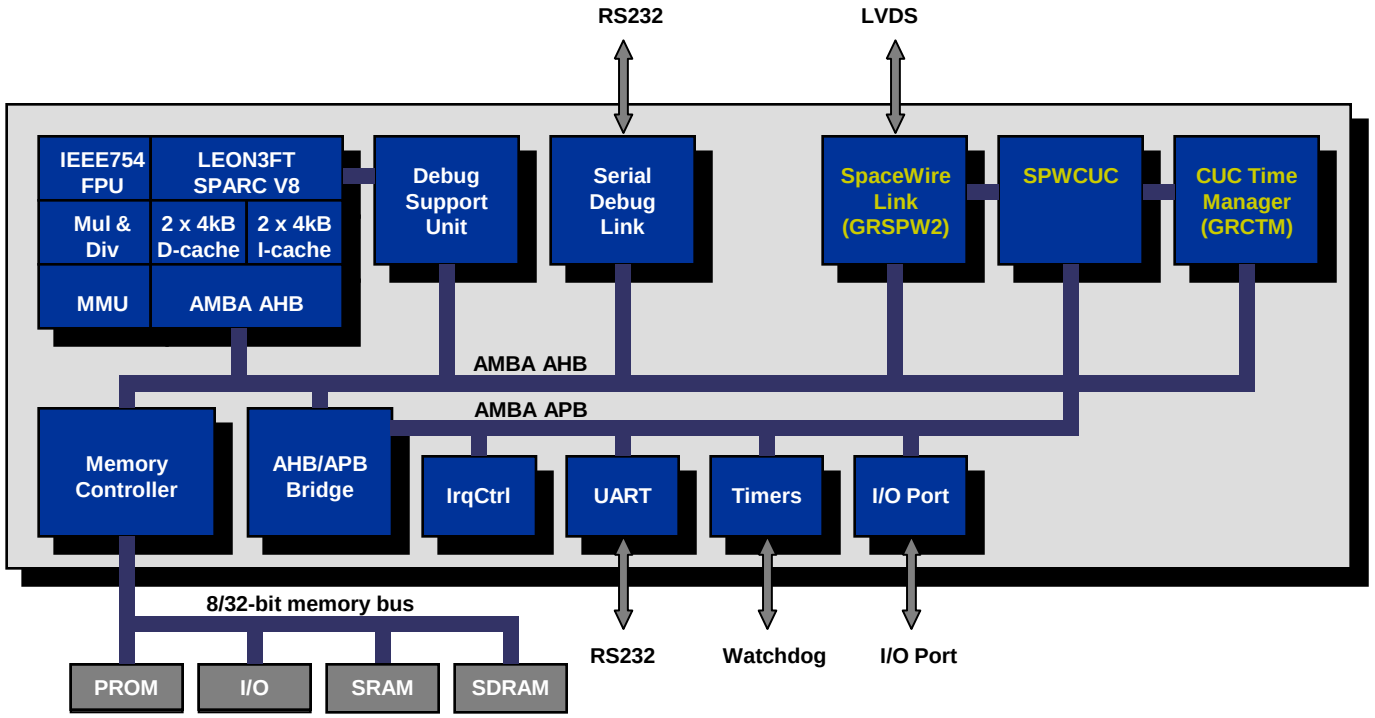
Uses SpaceWire packets for high-level synchronization, based on CCSDS Unsegmented Code (CUC). Uses SpaceWire Time-Code time-information for low-level synchronization, which is coupled to CUC. A new Protocol Identifier (PID) based transfer protocol is defined for SpaceWire packets carrying CUC.

Destination Logical Address	Protocol Identifier	CCSDS Unsegmented Code										CRC	EOP
		P-Field		T-Field									
		Default	Extended	Coarse Time				Fine Time					
				2^{31} 2^{24}	2^{23} 2^{16}	2^{15} 2^{8}	2^{7} 2^{0}	2^{-1} 2^{-8}	2^{-9} 2^{-16}	2^{-17} 2^{-24}			*weight*
		0 7	8 15	0 7	8 15	16 23	24 31	32 39	40 47	48 55			*index*
8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits		8 bits	*no. of bits*

CUCTP in LEON3 Systems

The CUCTP has been implemented as an IP core and has been used in LEON3 systems as a mean for exploring the possibility of time distribution over SpaceWire.

The experiment has helped us to identify and understand the pitfalls with time distribution: latency, jitter and drift.



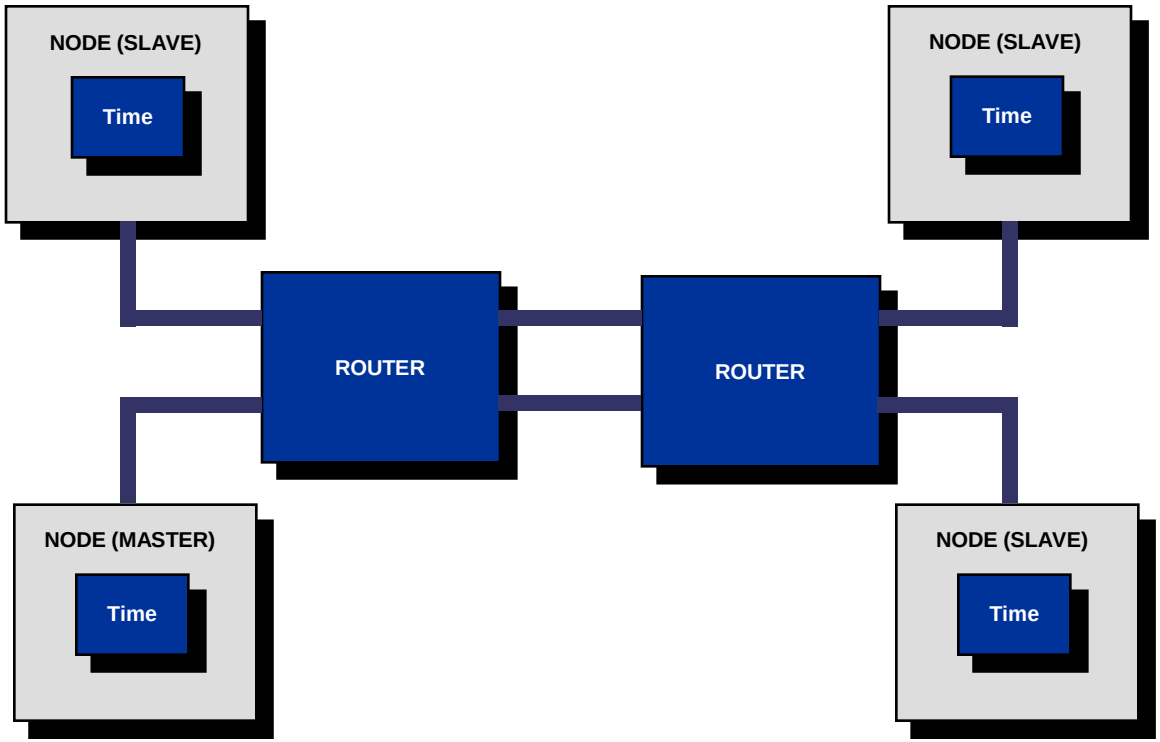
Time distribution pitfalls

- Latency: time it takes to transfer a synchronization pulse from source to destination
- Jitter: variation of the above time
- Drift: mismatch between the local time in source and destination

Let us assume that time synchronization is done via time-codes in a SpaceWire network, then:

- Latency: serialization/de-serialization, re-synchronization
- Jitter: time-code being blocked by other transfers
- Drift: no common clock in the network

Example of a network



Time distribution pitfalls - solutions

- Latency: can be characterized for each device
- Jitter: can be characterized for each device
- Drift: can be handled locally in source

Examples of implementations:

- Latency: can also be measured by of e.g. interrupts and interrupt acknowledge (future extension of time-codes), done once in a system with fixed topology
- Jitter: same approach as above, but should be measured with active data traffic, done once in a system with fixed topology
- Drift: done locally in destination, e.g. by measuring statistically the difference between synchronization pulse and local time

Time distribution protocol – trade-off

ESA activity started to standardize protocol

Time format: CUC

Time message: own PID, RMAP, RMAP-like protocol & new PID

Synchronization pulse:

- Via time-code wrap-around, possibility for low level synchronization with the other 63 values

Latency and jitter measurement:

- For SpaceWire rev D., utilized interrupt & acknowledge

Drift: not in protocol, implement locally, e.g.:

- averaging of mismatch error, brute force truncation, programmable frequency synthesizer

Conclusion

Accurate time synchronization is essential for upcoming SpaceWire-D, accuracy around 1 us required

Time-codes insufficient for over-all time keeping

ESA has initiated an activity for standardization of a time-distribution protocol, synchronization, and handling of latency, jitter and drift

The directions for the work have been presented today

The next update will be in December 2011 at the WG meeting

The goal is to have a specification ready in January 2012

The goal is to have a verified implementation ready in Q1 2012