TIME-TRIGGERED SERVICES FOR SPACEWIRE

Session: Networks and Protocols

Long Paper

Wilfried Steiner, Reinhard Maier
TTTech Computertechnik AG, Vienna, Austria
David Jameux
European Space Agency ESTEC, Noordwijk, The Netherlands
Astrit Ademaj
Vienna University of Technology, Vienna, Austria
E-mail: wilfried.steiner@tttech.com, reinhard.maier@tttech.com, david.jameux@esa.int, ademaj@vmars.tuwien.ac.at

ABSTRACT
In a real-time network Time-Triggered Services allow a set of individual components to work as a coordinated whole with two main resulting benefits: firstly, a strong system-wide determinism is established and, secondly, the given physical resources can be high-efficiently utilized. The clock-synchronization service is a core time-triggered service that brings the local clocks of the individual components into agreement. The synchronized local clocks can then be used to trigger system-wide coordinated actions, such as the transmission of messages, which are then said to be time-triggered. In addition to time-triggered communication only, the synchronized local clocks can also define intervals in which event-triggered communication is allowed, which even enables mixed real-time non-real-time communication on a single physical network.

1 INTRODUCTION

Modern computer network architectures introduce dedicated network components like routers to reduce the number of communication links in the system. Nodes will then connect to a router, for example, instead of connecting directly to each other with individual point-to-point communication links. Besides the obvious weight and cost reduction of this architectural approach, additional media access logic has to be realized in order to establish a mutually exclusive access of the nodes to the communication links, which become a shared network resource.

In the simplest form, the media access logic implements an event-triggered principle, in which a node is free to access the network at arbitrary points in time and in which the nodes are serviced on a first-come first-served basis. An immediate drawback of this event-triggered principle is the cumulative transmission delay and jitter, when several nodes need to communicate onto the same shared communication link. The time-triggered principle constitutes a media access logic that uses a system-wide synchronized time-base to provide coordination between nodes in a distributed computer system, such that transmission delay and jitter are kept within low bounds.

This paper presents the outcome of an ESA-funded study on investigating the general applicability of time-triggered services for the SpaceWire protocol as well as identifying resulting constraints on SpaceWire Nodes and Routers. As a general outcome of this study we conclude that time-triggered services seamlessly integrate with the SpaceWire protocol which already provides synchronization primitives, so

called Time Codes that can be leveraged to establish a system-wide synchronized time-base.
A communication network consists of end systems that are connected via communication channels. Communication channels usually consist of passive wires and network components. In the case of SpaceWire the communication network consists of SpaceWire end systems (also called communication nodes) and SpaceWire Routers. SpaceWire is a communication protocol that defines low-level communication paradigms. The objective of this paper is to conceptually discuss how SpaceWire could be extended via time-triggered communication services and to identify possible constraints and restrictions in the specification of SpaceWire Links and SpaceWire Routers.

2 TIME-TRIGGERED SERVICE CLASSES

The number of time-triggered communication protocols is increasing and while the time-triggered protocols differ significantly in the algorithms they implement to realize time-triggered communication, there is a common set of problems that has to be solved. We call this common set of problems the Time-Triggered Service Classes.

Scheduled Dispatch Service Class: This class specifies methods for time-triggered dispatch of messages according to an off-line specified schedule table. This includes the representation of the schedule in the components, e.g. how the schedule is stored in local memory.
Clock Synchronization Service Class: This service class represents services that ensure that the local clocks of the components in the communication infrastructure stay synchronized to each other once synchronization is established.
Startup Service Class: The startup service class covers methods and services to initially synchronize the components in the communication infrastructure. This can be a coldstart procedure or an integration/reintegration procedure.
Clique Detection and Resolution Service Class: This service class defines measures that detect clique scenarios. These are unintended scenarios where disjoint subsets of components are synchronized within the subset but not over subset boundaries. Clique Resolution services define methods that re-establish synchronization when cliques have been formed and detected
Membership Service Class: Membership services are low-level diagnosis services that continually monitor the system's health state. In particular such services could reflect which end systems are present in the systems and which are not – for example because of transient/permanent failures.
External Synchronization Service Class: This service class specifies methods that allow the communication infrastructure to synchronize to an external time source.
Configuration and Maintenance Service Class: This service class defines services on how a communication infrastructure can be configured and maintained. Such services include for example configuration download procedures.
Dataflow-Integration Service Class: This service class defines measures on how message classes with different characteristics can be integrated such that all those message classes can use the same physical medium. In particular the integration of event-triggered and time-triggered messages classes is of interest in this service class.
Legacy Service Class: Existing protocols have interoperability requirements. This service class aims to identify these requirements and provide glue functionality to allow interoperability.

Integrity Service Class: This service class defines services that enhance the integrity of the communication infrastructure. In particular we are interested in two types of integrity measures: a guardian measure that can be central, local, or both, and end-to-end arguments, such as sequence numbers and timestamps.
Availability Service Class: This service class defines services that enhance the availability of a communication infrastructure. Such services include redundancy management of communication channels and redundancy management in case of fault-tolerant computation entities such as TMR configurations.

The complexity of the actual services that are realized for the service classes above, heavily depends on the system requirements. A master-based system, for example, will allow the realization of very simple services, and a single function may be sufficient to address multiple service classes at the same point in time. A master-less system will require services to be realized in form of distributed algorithms, which are inherently more complex. On the other hand, master-less systems provide higher system reliability as the failure of a single device will typically not result in an overall system loss.

This paper discusses the Clock Synchronization Service Class, the Scheduled Dispatch Service Class, and the Dataflow-Integration Service Class in particular for compliance with SpaceWire. More service classes are discussed in the final report of the study "Time-Triggered Techniques for Quality of Service over SpaceWire" [1].

3 CLOCK SYNCHRONIZATION SERVICE CLASS

This service class represents services that ensure that the local clocks of the components in the communication infrastructure are synchronized to each other once synchronization is established.
Each oscillator, as a physical component, has slightly different characteristics. One of these characteristics is the *Drift Rate*, which is defined as the difference to an oscillator perfectly aligned with real-time. Note that in this context of real-time data communication and distributed control, relativistic effects in time are not considered. According to Kopetz [2], typically Drift Rates are in the range of 10^{-2} to 10^{-7} sec/sec. It is the aim of the clock synchronization service to compensate for this inherent drift of local clocks. One straight forward clock synchronization services is Master-Slave, which off-line declares a node as Master which is used as reference clock in the network. SpaceWire inherently supports Master-Slave via SpaceWire *Time-Codes*.
SpaceWire specifies Time-Codes at the character level. A Time-Code is formed by: an Escape Character (ESC), consisting of 1 parity bit and 3 control bits, and a single Data character, consisting of 1 parity bit, 1 data-control flag, and 8 data bits. The structure of the SpaceWire Time-Codes is depicted in Figure 1.

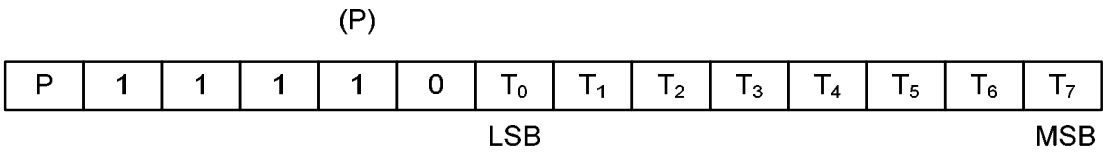

Figure 1: SpaceWire time-code

In the SpaceWire specification [3], Section 7.3(d), the Time-Code is further specified as "Six bits of time information shall be held in the least significant six bits of the

Time-Code (T0 – T5) and the two most significant bits (T6, T7) shall contain control flags that are distributed isochronously with the Time-Code."

Hence, six bits allow for sixty-four different Time-Codes. However, as a minimum only a single Time-Code is needed for time-triggered communication. In general, the number of different Time-Codes required is a function of

- the size of the communication schedule and the number of required integration points in the schedule: as the communication schedule may temporarily become long it may be required that a node can integrate at specified points inside the communication schedule instead of only at the communication schedule start, and
- the required precision in the system: again, as the communication schedule may temporarily become long, it may be necessary to schedule multiple Time-Codes for re-synchronization of the local clocks.

Time-Codes have highest priority and are transmitted interleaved with the regular dataflow. This means a Time-Code is sent immediately or immediately after the transmission of an ongoing Character is finished. Hence, on a per SpaceWire link basis, the latency jitter of a Time-Code is bounded by the maximum character in transit, which is the length of one data code (10 Bits). For the Master-Slave clock synchronization process the precision is a simple function of the latency jitter and the drift offset (where R.int is the re-synchronization interval, i.e. the duration in between two consecutive resynchronization attempts):

*Precision = Latency Jitter + Drift Offset = Latency Jitter + 2 * Drift Rate * R.int*

The end-to-end latency jitter is calculated from the link latency jitter and is discussed in the SpaceWire specification [3] under 8.12 (p) Note 2: ST.jitter = 10 N/R, where, N is the number of Links traversed and R is the average link operating rate.
The best theoretically achievable precision in the system is therefore:

*Precision = Latency Jitter + Drift Offset = (10 * N / R) + (2 * Drift Rate * R.int)*

Note, that this does not include additional Latency Jitter imposed by a SpaceWire Router as this additional Latency Jitter is implementation dependent.

A Master-Slave clock synchronization algorithm is attractive due to its simplicity and the resulting low overhead in specification, implementation, testing, and certification. On the negative side, a pure Master-Slave clock synchronization algorithm does not provide fault-masking. This means that, if the master fails (a) no time may be generated at all or (b) a malicious timeline may be generated or, (c) an interrupted timeline may be generated. Hence, if fault masking is not required the Master-Slave approach is a good solution. However, if fault masking is a requirement, Master-Slave has to be enhanced via distributed algorithms.

One straight-forward fault-masking extension is a backup clock synchronization master. This means that instead of a single node, a second node is configured with an active TICK_IN signal. Note that this is already a violation of the guideline in the SpaceWire specification that suggests assigning only one node an active TICK_IN signal. This backup master could run in warm standby or hot standby:

- Warm Standby: the backup master continuously checks the status of the primary clock synchronization master, potentially via checking if it receives valid Time-Codes. If the backup master does not receive valid Time-Codes for a specified duration it starts sending Time-Codes itself. This approach is very simple and could be argued to be in-line with the guideline suggesting only one master as there actually is only one master at any point in time.
- Hot Standby: both the primary and the backup master can send Time-Codes with an offset to each other. The identity of the primary and the backup master can be encoded via a static distribution of name space via the six least significant bits of the Time-Code (for example Time-Codes with T0 = 0 are sent by the primary, Time-Codes with T0 = 1 are sent by the backup master). As there are two sources of Time-Codes present the fail-silence failure of one of the masters is immediately masked. In order to mitigate a drift in the timeline of the primary and the backup master, these timelines have to be synchronized to each other. A simple synchronization procedure would be that the backup master always follows the primary, if the primary is present.

It has to be mentioned that the failure models that can be covered with these types of standby only cover benign failure modes like fail-silent failures of the clock synchronization masters. Failure models like babbling idiots, that are faulty nodes or faulty routers that do not fail silently, but send at arbitrary times demand the implementation of guardian instances. Guardian instances can be for example local at the node/router or centralized at a router to protect against babbling nodes.

4 SCHEDULED DISPATCH SERVICE CLASS

This class specifies methods for time-triggered dispatch of messages according an off-line specified schedule table. This includes the representation of the schedule in the components, e.g. how the schedule is stored in local memory. In a Schedule Master-based system, the Schedule Master will execute a request-response protocol: the Schedule Master sends a request to a slave, which in turn will respond with the requested information. The MIL-BUS 1553 is an example of a Schedule Master-based system. On the other hand, the schedule information can be distributed in the components and time-triggered services can be used instead of a master-driven request-response protocol. The time-triggered services ensure that the distributed schedule tables are synchronously executed. While, the time-triggered services reduce the communication overhead introduced by the request-response protocol, the Schedule Master-based approach is more flexible, as schedule changes can be done locally, without a dedicated mode change protocol.

In both approaches, Schedule Master-based, and Distributed Schedule-based, it is required that the dataflow will not be dependent on the receiver of a message. Hence, even in case of the failure of a receiver (or in case of a faulty destination address), there must be some (HW) mechanism ensuring that the messages will be delivered such that the bus/network will not be deadlocked. Therefore, non end-to-end flow control is required.

Figure 2 presents a communication network consisting of four nodes and a single router on the left hand side. On the right hand side an example of a communication schedule is depicted.

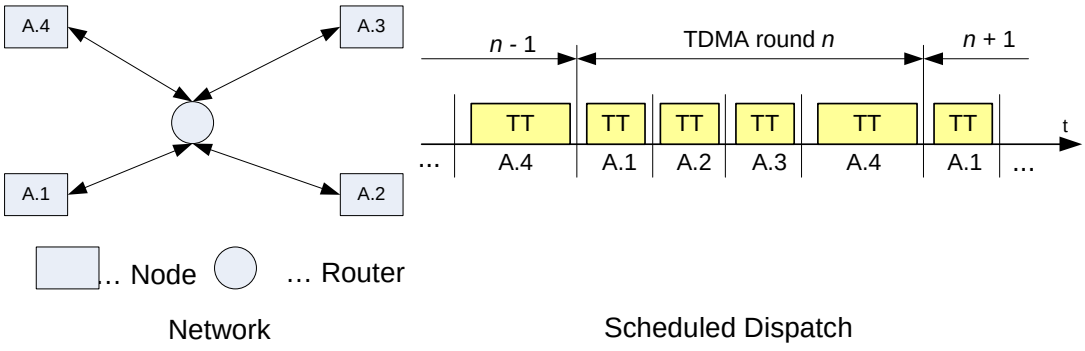

Figure 2: Communication network of four nodes, one router and communication schedule

4.1 Traditional Time-Division Multiple Access

In a protocol using Time-Division Multiple-Access, time is split up into pieces of not necessarily equal durations, which are called *slots*. These slots are grouped into sequences called TDMA rounds (see Figure 2), in which every node occupies one or more slots. The knowledge, which node occupies which slot in a TDMA round is static, available to all components a priori, and equal for all TDMA rounds. When the time of a node's slot is reached, the node is provided exclusive access to the communication medium for the duration of the slot. After the end of one TDMA round, the next TDMA round starts, that is, after the sending of the node in the last slot of a TDMA round, the node that is allowed to send in the first slot sends again. Consequently, the sending slots of each node are repeated with each TDMA round.

As any two local clocks in the network will always be slightly offset, a node may not use the full assigned timeslot for message transmission. This is a restriction that comes from bus-based networks, in which concurrent bus-accesses of different nodes must be avoided as they would result in physical signal interferences on the wire. Durations between two consecutive transmissions are called *inter-frame gaps*. The inter-frame gaps have to be chosen with respect to the precision in the system and the different propagation delays of the messages on the channels.

4.2 Novel Concepts in Time-Division Multiple-Access

The traditional TDMA as discussed in the previous paragraphs was designed for bus-based network protocols. With the movement from bus-based to network-based topologies, the rather static principle of time-triggered communication can be improved. Some of these improvements are as follows:

- Time-Triggered Multi-Cast Communication: in network-based communication topologies the switches, or routers, can be used to route messages only to subsets of nodes (or routers) instead of sending all messages in broadcast. Note that in SpaceWire, messages are only sent in uni-cast (except Time-Codes).
- Inter-Frame Gap Reduction: As discussed under traditional Time-Division Multiple-Access, in order to avoid communication conflicts, the inter-frame gap is a function of the precision (the quality of synchronization) in the communication network. In network-based networks the switches, or routers, can give priority to the transmission in progress, when a second node starts its transmission early. Hence, the router can intermediately buffer parts of the second node’s communication data, until the first node finishes its transmission.

- Dynamic Slot-Assignment: a schedule master can be implemented in the network that dynamically adjusts the communication schedule to optimize the bandwidth utilization when nodes enter or leave the network. However, in space applications, modification of the communication scheme of nodes happens rarely enough (e.g. at phase change or at stage separation) and is always predictable, so that the different configurations can be considered as “static” modes.
- Support of non-harmonic message periods: in traditional TDMA, harmonic message periods can be supported. For example in TTP this is achieved by defining a Cluster Cycle that consists of a configurable number of TDMA rounds. As the TDMA round is the shortest period, this is also the highest frequency of message transmission. Messages that have to be sent with a higher period can be sent in every other TDMA round instead of every TDMA round. New schedule dispatch services also allow contention-free communication schedules with non-harmonic periods. An example of a communication schedule is given below in Figure 5. The communication schedule depicts a message with a period of 2 ms and a message with a period of 3 ms.

5 DATAFLOW-INTEGRATION SERVICE CLASS

In order to efficiently utilize the given physical resources the data-flow integration service class addresses methods that allow using the same physical network for both, scheduled time-triggered traffic (TT) and non-scheduled event-triggered traffic (ET). This service class, hence, enables payload data and command & control data on the same physical network. Similar as required for time-triggered traffic only, we assume that the delivery of a time-triggered message will never be blocked by a faulty or non-present receiver as the network would deadlock otherwise. For event-triggered messages, either readout at destination node is also ensured or end-to-end flow control must be implemented.

In this section we present three dataflow-integration services and their suitability to SpaceWire. We name these services according to the protocols that realize them, namely, the “DECOS-Approach”, the “FlexRay™-Approach” and the “TTEthernet-Approach”.

5.1 DECOS-APPROACH

The dataflow-integration service may specify a “tunnelling”-mechanism of ET data over TT communication: all transmissions on the network follow the scheduled-dispatch principle. ET messages are not directly sent to the network, but a middleware layer places the contents of ET messages inside a dedicated part of a TT message. We could call this mechanism “allocated transport of ET messages over a scheduled network”. As the DECOS-Approach appears at the network interface as regular scheduled dispatched traffic, it is suitable for SpaceWire.

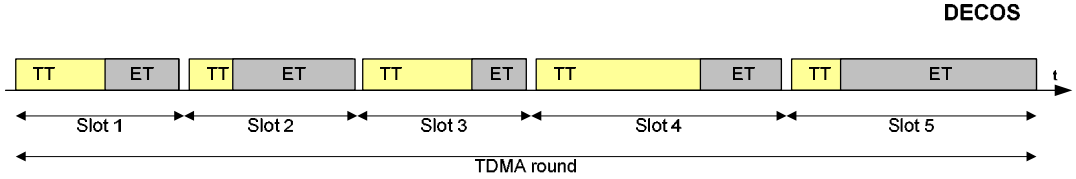

Figure 3: DECOS-Approach

The time-triggered slot may be dynamically split between time-triggered end event-triggered data or may even be fully used for event-triggered data. In the latter case we call the time-triggered slot “sporadic”, which will only be used for time-triggered data

if new data is present and free for node-local event-triggered communication otherwise. We could call this mechanism “opportunistic traffic”.

5.2 FLEXRAY™-APPROACH

The dataflow-integration service may specify a “meta” time-division multiplexing approach: the bandwidth is split into (a) a static segment in which only TT messages are communicated and (b) a dynamic segment, in which only ET messages are communicated. The static and the dynamic segments are alternately executed.

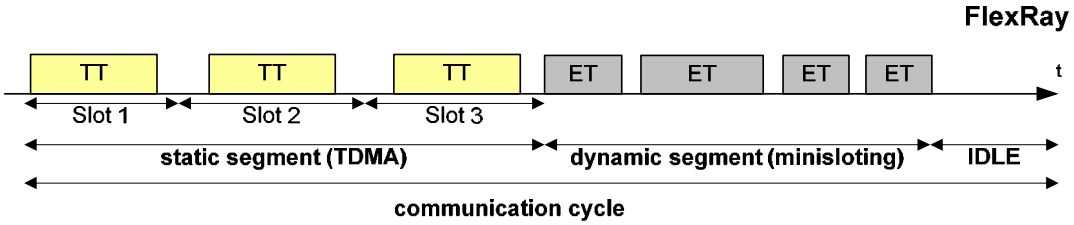

Figure 4: FlexRay-Approach

For SpaceWire, the FlexRay-Approach can be realized by offline-specifying the dynamic segment as a “long” slot in which ET communication is allowed. The start and end instant of this “event-triggered slot” can be derived from the synchronized local clocks or from dedicated Time-Codes. The latter approach is attractive, as SpaceWire nodes that communicate ET messages only do not have to be synchronized.

When a node identifies the end instant of the event-triggered slot, it may finish a pending or active transmission. However, as this increases the IDLE phase before the next static segment can be started, the node may also immediately remove pending transmissions and stop active transmissions by sending an Error end of packet (EEP) control code. In both cases, the necessary IDLE time between the end of the dynamic segment and the start of the next static segment is bounded, given the network MTU (both for TT and ET messages) and the maximum number of nodes allowed to send ET messages.

5.3 TTETHERNET-APPROACH

The dataflow-integration service is realized inside the switches in a TTEthernet network. To a TTEthernet switch, dedicated TTEthernet nodes, as well as, arbitrary standard Ethernet nodes (e.g. a laptop) can be attached. While the TTEthernet nodes execute a clock synchronization service and are therefore able to communicate TT messages, the standard Ethernet nodes will typically send ET messages only. The TTEthernet switch privileges the TT messages over the ET messages. For this a TTEthernet switch realizes two configurable modes: in case of a conflict of a TT and an ET message the switch either (a) pre-empts the ET message and relays the TT message with a constant latency, or (b) finishes the transmission of the ET message and relays the TT message immediately after the ET message (that means the TT message is treated with highest priority in case of other messages waiting for relay; as TT messages are scheduled, there will never be more than one TT message waiting for relay).

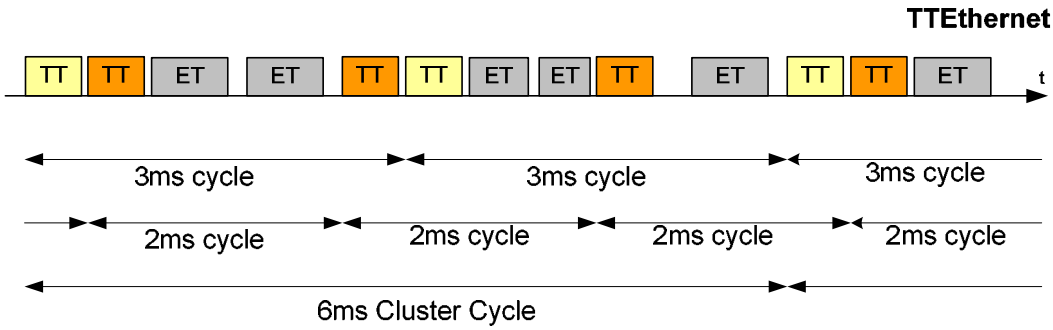

Figure 5: TTEthernet-Approach

For SpaceWire the first privilege mode will be difficult to achieve, as SpaceWire does not specify store-and-forward switching behaviour, but cut-through (also called "Wormhole Routing"). However, the second approach seems realizable, if the SpaceWire Router is able to distinguish a TT message from an ET message, which can be done by an appropriate higher layer identifier inside a SpaceWire packet.
In the TTEthernet-approach time-triggered slots can also be defined as sporadic slots in analogy to the DECOS-approach. The TTEthernet-approach is, however, more powerful, as the decision to free a time-triggered slot for event-triggered communication is done in the switch and not node-local. Hence, the sporadic time-triggered slot can be used also by event-triggered traffic coming from other nodes.

6 CONCLUSION

In this paper we discussed the feasibility of time-triggered services on top of SpaceWire. We conclude that the SpaceWire Time Codes inherently allow the realization of a Master-Slave clock-synchronization service. This clock-synchronization service is the enabler to synchronize the sending actions of the SpaceWire nodes and ensure that communication conflicts are avoided. We also discussed three approaches for SpaceWire to integrate time-triggered and event-triggered communication on a single physical SpaceWire network. We conclude that the FlexRay-approach is attractive for SpaceWire as it is simple to realize. However, it is not optimal as time-triggered slots cannot be made sporadic and, hence, be reclaimed for event-triggered. The TTEthernet-approach on the other hand would allow reclaiming bandwidth reserved for time-triggered traffic, but is more complex to realize.
From a fault-tolerance perspective, the Time-Code mechanism tolerates only simple failure modes of SpaceWire components. Fault-tolerance capabilities that go beyond simple fail-silent or detectably-faulty failure modes would cause a significant enhancement to SpaceWire as is (involving the concept of Guardian) and is not addressed in this paper.

7 REFERENCES

1. W. Steiner, R. Maier, A. Ademaj, "Time-Triggered Techniques for Quality of Service over SpaceWire", ESA Contract Number 21050/07/NL/LvH
2. H. Kopetz, "Real-Time Systems" Kluwer Academic Publishers, 1997, p.59
3. SpaceWire - Links, nodes, routers and networks, ECSS--E--50--12A, January 2003