Journal of
Low Power Electronics and Applications

MDPI

Article

Energy Efficiency Due to a Common Global Timebase—Synchronizing FlexRay to 802.1AS Networks as a Foundation

Paul Milbredt *, Efim Schick and Michael Hübner

Embedded Systems for Information Technology, Ruhr-University Bochum, 44780 Bochum, Germany; efim.schick@gmail.com (E.S.); michael.huebner@ruhr-uni-bochum.de (M.H.)
* Correspondence: paul.milbredt.btu@geneda.de; Tel.: +49-152-2151-2050

Received: 12 July 2018; Accepted: 9 August 2018; Published: 17 August 2018

Abstract: Modern automotive control applications require a holistic time-sensitive development. Nowadays, this is achieved by technologies specifically designed for the automotive domain, like FlexRay, which offer a fault-tolerant time synchronization mechanism built into the protocol. Currently, the automotive industry adopts the Ethernet within the car, not only for embedding consumer electronics, but also as a fast and reliable backbone for control applications. Still, low-cost but highly reliable sensors connected over the traditional Controller Area Network (CAN) deliver data needed for autonomous driving. To fusion the data efficiently among all, a common timebase is required. The alternative would be oversampling, which uses more time and energy, e.g., at least double the perception rates of sensors. Ethernet and CAN do require the latter by default. Hence, a global synchronization mechanism eases tremendously the design of a low power automotive network and is the foundation of a transparent global clock. In this article, we present the first step: Synchronizing legacy FlexRay networks to the upcoming Ethernet backbone, which will contain a precise clock over the generalized Precision Time Protocol (gPTP) defined in IEEE 802.1AS. FlexRay then could still drive its strengths with deterministic transmission behavior and possibly also serve as a redundant technology for fail-operational system design.

Keywords: Ethernet; FlexRay; 802.1AS; real-time

1. Introduction

A modern car is a highly-distributed system of more than a hundred embedded control units (ECUs) and even more sensors and actuators. To save costs and energy, sensors, ECUs, and actuators are connected by buses and sensors are only applied once in a car, distributing their data digitally.

The car therefore looks very much like the nervous system in the human body. Information is processed distributively like reflexes at lower levels, only where it is really needed. In the end, where low bandwidth is sufficient, low-cost connections like the LIN (Local Interconnect Network) or CAN bus are used. The higher the demand of bandwidth gets, the more expensive technologies like FlexRay are used. A highly simplified architecture is depicted in Figure 1.

The trend of big data in the IT industry takes advantage of the tremendous amount of available information and builds new applications by processing them. Within the embedded car domain, a similar trend applies. Since processing power still grows exponentially, more and more centralized functions can be implemented in the nervous system, e.g., autonomous driving. This requires a fast and reliable data channel to this central brain.

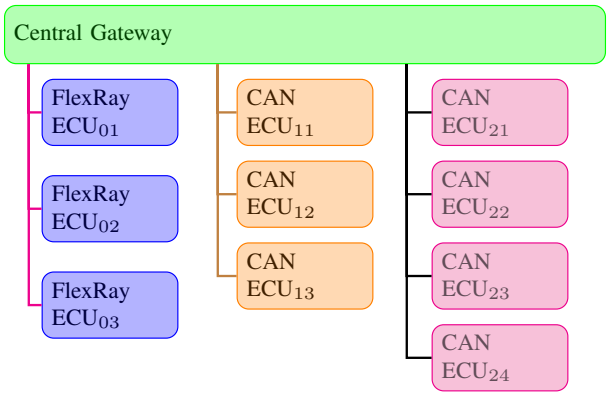

Figure 1. Simplified electronic architecture of a current car.

A car will still be a very fast moving system. If a car moves with 250 km h^{-1}, it moves by 70 cm in a typical calculation cycle of 10 ms. Other moving objects around, like a car upfront changing the lane, also add up to this change of the environment. Physical reaction times for, e.g., steering have to lie in the range of just a few milliseconds. The engineers today have a very specific tool set to solve the time-sensitive design. FlexRay, as the current dominant system, was designed specifically from and for the automotive industry. It offers not only a time-triggered communication scheme, but also serves as a common clock for the operating systems to adjust their computing in time to achieve guaranteed time-bound results.

For the new high-bandwidth needs, the automotive industry is adopting Ethernet [1]. The main reasons are its wide acceptance through consumer market and industry and the high bandwidths. However, Ethernet by default does not guarantee transmission times or is its physical layer suitable for the automotive domain. It requires additional protocols like 802.1AS and physical layers like 100BASE-T1 to adapt to the automotive domain.

Additionally, the legacy systems are still cheaper by an order of magnitude and, due to their specific design, do their job very well. Therefore, we are confident that, in contrast to [2], FlexRay will not be replaced by Ethernet, though this would, of course, technically be possible. We present in this paper how a holistic automotive design with the focus on Ethernet and FlexRay will most likely look. In addition, high bandwidth availability tends towards wasting it by oversampling sensors and data. This also leads to higher energy consumption, due to more data acquisition and distribution. A fully time-triggered system can lower the power to the absolute necessary minimum because data has only to be sent when required and no more often.

The article is organized as follows: in Section 2, we will first explain the benefit of a holistic global time. This will also be investigated by related work and the section closes by a short introduction of the timing concepts of FlexRay and Ethernet. Section 3 is to discuss the problem of drifting clocks, followed by Section 4, which gives an overview of the architecture of upcoming cars. The global timing concept is presented in Section 5, first for Ethernet, then FlexRay, and then the combination of both.

The theoretical precision is calculated in Section 6. That this matches the real industry application is experimentally proven in Section 7, before Section 8 concludes this paper and gives an outlook of future work.

2. State of the Art

For a holistic timing hierarchy, we present a short overview of the different notions of time that different technologies offer. Imagine that you own an analogue quartz driven watch. Even if you could set it perfectly, i.e., it would have no offset to the official time, it would afterwards deviate from it over time. It also would overflow every 12 h and always have an offset to clocks in different time zones.

A practical example should illustrate the demand of synchronization. For the autonomous driving, the best possible complete digital image of the environment has to be computed. To classify objects

and to use the best sensor combination possible, it is crucial to compare their data. As not only the car, but also the surrounding objects move, the point in time when data is acquired needs to be known [3]. In that case, predictions of movement make objects comparable that were acquired by different sensors. It would be even better to synchronize the acquisition times. Then, object positions match by definition, which eases the classification and makes predictions of their movement unnecessary. Thus, single faulty sensors can be compensated by others, and different quality of data leads to better classification of the environment.

Given the diverse architecture of a car, clocks need to be synchronized over all buses in the car. In addition, a single FlexRay gives a synchronized clocks to the application developer, but it is running independently from any other FlexRay that might exist. The whole car should get a common timebase to ease the time driven development and security features.

In [4], a similar thought is mentioned. However, it not only leaves all the calculations and proofs open, but also the real synchronization is only mentioned as future work.

The authors of [5] also say that they synchronize FlexRay to Ethernet. However, in their work, they need a tremendous amount of time (minutes) in comparison to our approach, which we will also prove (a few seconds). Their work also lacks all details on the implementation of the timing part. It remains unclear why they constrain their application.

Before we present how synchronization can be achieved, we will look into the timing of FlexRay and Ethernet.

2.1. FlexRay Clock Synchronization

The FlexRay bus is based upon a time-triggered communication scheme (TDMA). Any node may send its frame only within dedicated slots, which must be configured in advance. Therefore, a global common time base has to be established.

Simplified, it is done as follows: The node local clock is driven by its quartz. All known implementations derive a local *microtick* (µT) clock of 40 MHz out of it. The smallest synchronized global time unit is the *macrotick* (MT). It is always an integer multiple of the microtick in the range of 1 to 6 ms. Its length might vary slightly due to configuration and the clock synchronization.

It serves as a basis for all other units in the system, like slot sizes and cycle length. Every node measures for any received sync frame the deviation of the actual from the expected arrival time. With this information, it calculates an offset and a rate correction value. The offset correction is applied once during the *network idle time* to shift the local clock once. The rate correction value is applied distributed over the cycle to the macrotick generation, so that a deviation of the clock source is compensated.

The operating system on the microcontroller synchronizes its schedule now to the FlexRay time. Carefully designed, the transmitting tasks, the slots on the bus and the receiving tasks are well aligned.

This way, a time critical application can in theory be designed in a way that the overall reaction time from sensor to actuator time is minimized.

2.2. Ethernet Time Synchronization IEEE 802.1AS-rev

Historically, Ethernet is based on a CSMA/CD (Carrier-sense multiple access with collision detection) media access scheme. Nowadays, no collisions can occur anymore because they only consist of point-to-point links, which are actively switched to build networks.

Though a node always has exclusive access to its sending port, the exact physical transmission time is not known in advance, due to internal buffering in the MAC layer or no knowledge about the traffic on the other ports of the next switch. This also leads to different timing of the same message on different links within the network.

High precision time synchronization can be achieved, e.g., according to the generalized precision time protocol (gPTP) defined in 802.1AS-rev [6,7].

A so called grand-master is selected (most likely statically defined during design time of the network in the automotive domain), which owns the local clock with the best precision. It sends out

its time with so called sync messages, which are specific Ethernet frames in layer two of the Open Systems Interconnect (OSI) model. These packets contain a time stamp, which will be updated while they are really transmitted on the wire with the actual time. Thus, even if there was still an ongoing transmission, which would delay the sync message, it would be sent as soon as possible. While it is transmitted, the current time stamp would be included in its payload.

Since Ethernet is a point-to-point system, to connect more than two nodes, switches (which are always called bridges in the specification) are necessary. They also need time for reception, processing and transmission. Special frames are used to determine the link delay because a frame needs time until it is received and processed by the client MAC. With this information, the receiver can use the grandmaster's timestamp. If the node is also a bridge, it acts as a master on all the other links. Since it is synchronized to the master, clients that are synchronized to the bridge are also indirectly synchronized to the grandmaster.

From the reception of more than one message with a timestamp, the slave can also calculate its clock rate deviation.

3. Clock Drifts

If the clocks of two distinct systems run unsynchronized, they deviate from one another. In general, assuming you have n systems, let the deviations of all systems be $d_1, \ldots, d_n$ and the set of the absolute maximum deviations D be defined as $\{|d_1|, \ldots, |d_n|\}$.

Since any deviation is defined from the absolute physical time, the maximum overall deviation D is hence

$$d_{\max} = \max(D) + \max(D \setminus \max(D)). \quad (1)$$

The FlexRay specification allows a quartz deviation of ± 1500 ppm [8], Ethernet specifies ± 100 ppm [6]. Therefore, the combined worst case deviation is 1500 ppm + 100 ppm = 1600 ppm. For illustration, the maximum is shown in Figure 2. A typical FlexRay implementation has a cycle time of 5 ms. This would lead to an maximum relative deviation of FlexRay from Ethernet by $\Delta t_{\max} = 0.008$ ms or 1.6 ms every second.

The communication of FlexRay allows transmission only at specific points in time. This could then also lead to a scenario in which a gateway node receives a message from the Ethernet, but, because its FlexRay transmission slot just occurred, the message needs to be stored until the next possible timeslot.

It is also common practice to schedule a message only every 2^n-th cycle, with $n = 0, \ldots, 6$.

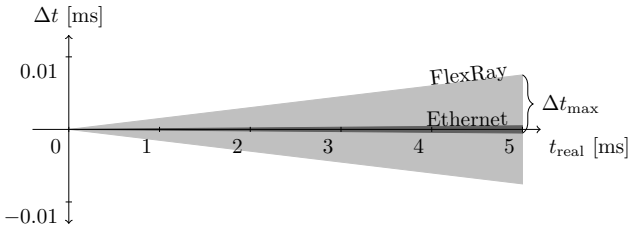

Figure 2. Possible clock deviations of FlexRay and Ethernet.

On the other hand, Ethernet communication could also be time-triggered. The new *time sensitive networking* (TSN) specifications, which 802.1AS belongs to, also allows very detailed time triggered scheduling and forwarding of traffic. Thus, also when routing a message from FlexRay to Ethernet, the problem of a missed timeslot can occur.

4. Future In-Car Network Architecture

The current typical electronic architecture of a car usually consists of a central gateway ECU as shown in Figure 1. This is limited in its computing power but (at least within Audi and the whole

Volkswagen group [9]) still executes the functions, which require information from almost all over the car. For instance, the *Drive Select* function, which puts the car in different driving modes, distributes its output to almost any ECU or power control functions, which also processes data from almost every ECU. It is like the center point of a star topology, with different buses as the branches.

4.1. Integrating New Internet-Based Applications

The car will be more and more integrated with internet applications. For instance, live maps, containing road conditions, traffic information, and blocking will be not only used by navigation, but also chassis systems, adjusting the car for more comfort and safety [10]. These applications require short transmission delays and, because they have central servers on the internet [11], they are built upon internet technologies like TCP/IP (Transmission Control Protocol/Internet Protocol), MQTT (Message Queue Telemetry Transport) and so on. These protocols are optimized for Ethernet, but not for classic automotive bus systems like CAN or FlexRay, their short frame lengths being only one cause.

In addition, modern wireless technologies like LTE (Long-Term Evolution, 4G) or 5G (marketing term for the 5th generation of the wireless telecommunication standard) offer multiple gigabits of bandwidth, which can not be used by CAN or FlexRay.

4.2. Implementing Automotive Ethernet

With increasing processing power and by implementing Ethernet, distributed processing can also be realized in the car. Ethernet becomes a back-bone of the in-car network and, by this, no single central gateway needs to exist (see Figure 3). This enables a distributed, domain-based electronic architecture. Each of the main domain ECUs now has its own domain network. This could (and most likely will) be, for instance, FlexRay, which perfectly supports time-critical chassis control applications, CAN for standard applications in the comfort domain and Ethernet for consumer electronics and distributed applications support. We also disagree with the result of [12], which states that FlexRay is more complex, less efficient and the average cost is higher. Typical automotive applications cannot use the maximum frame size of 1500 bytes of Ethernet, making it with the CRC-32 and the larger header overhead more inefficient. Additionally, the automotive industry, though having static architectures, applies not only the Internet Protocol but also the Transmission Control Protocol or the User Datagram Protocol, which cause even more overhead. Additionally, FlexRay is an order of magnitude cheaper than Ethernet, due to its way simpler physical layer. It also does only require active stars and no switches, with a design of priorities and traffic analysis, making it less complex than Ethernet.

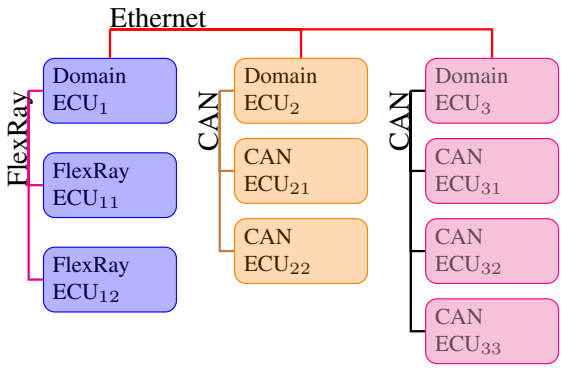

Figure 3. Simplified electronic architecture of a future car.

To perform computed decisions in autonomous driving, the more data a function can rely on, the more confidence is within the decisions or more faults can be tolerated. For example, a radar sensor spots objects, which then can be classified with image processing from a camera. The shape of the object might be derived from a laser scanner. Additionally, all sensors check one another's

plausibility [13]. This is crucial to guarantee safety requirements and the operation of the system in case of a failure.

5. Global Timing

Since the surrounding of the car contains numerous moving objects and the car is moving by itself, sensor data can only be compared, if the acquisition time is at least known or, even better, synchronized. In the latter case, no predictions of the relative motion of acquired objects have to be made.

Numerous more applications are possible, if a common timebase is established. If the speed and the road conditions are precisely known, the dampers can be controlled smoother in advance. Other actuators like airbags can be fired at that point in time, at which the least possible injuries and/or damages occur.

5.1. Ethernet Precision Time Protocol

With the presented architecture in the previous chapter, it is obvious to use the Ethernet as the master for the global time because every domain controller is connected to it, and, therefore, it is its task to serve as a time gateway between the Ethernet and the local domain network.

With 802.1AS [6], the IEEE offers a standard for precise synchronization of Ethernet nodes with a precision of less than 1 µs. The precise part of the timestamp offers 48 bits of 2^{-16} ns. It also offers a timestamp of 48 Bits for the seconds, which lets it overflow only every 8.9 million years.

Within the Autosar, very similar methods are proposed to distribute the time information over CAN or FlexRay [14]. However, just distributing time information is not enough. As we mentioned, one of the goals of system design would be to achieve minimal end-to-end transmission times. On the Ethernet, this could be done by also adjusting traffic to the synchronized time. As mentioned in Section 3, the TSN standards offer possible solutions. On top of that, the whole operating system task schedule should get synchronized to the network schedule.

5.2. FlexRay Timing

Once a foreign global timebase has been established, due to the TDMA access scheme, a FlexRay based system can only behave optimally, when the network timing is synchronized to the global time. Once this is achieved, now different FlexRay buses would also run synchronously, allowing all tasks to operate in an optimal manner.

The duration of a FlexRay macrotick, which is the smallest time unit being synchronized between all FlexRay nodes, limits the best possible synchronization. Although a small value would allow more precise results, it is proven in [15] that larger values lead to a more efficient use of the available bandwidth. Therefore, at Audi, the best possible synchronization is approximately 1.8 µs.

5.3. External FlexRay Clock Synchronization

Though version 3 of the FlexRay specification is available, the only standard available in silicon is Version 2.1A [8]. It allows an external clock correction through very strict boundaries.

Already at design time, both an offset and rate correction value in the range from none to seven microticks have to be chosen and cannot be changed later. This only allows a maximum external correction of just a few microseconds per double cycle.

During runtime, the host application can only apply a factor of 1, -1 or 0 to each of the two configured values. The result is then added to the calculated rate and offset correction values.

The offset correction is always reset to zero at the next cycle. However, because it is likely that the rate deviation of a quartz only slightly changes over time, the rate correction keeps its value. Therefore, any applied value will be additional to the previous total correction value. Only to prevent a drift of the cluster, a damping value is selected at design time, which reduces the rate correction towards zero. At Audi, the value is two microticks.

As a result, the rate correction value could grow more and more. This allows fast adjustment of the system clock. The offset correction allows single but precise movements.

6. Theoretical Analysis

The nodes of a FlexRay system could deviate by a maximum of ±1500 ppm. Though the fault tolerant midpoint algorithm should remove the extreme values from the calculation, we assume for a worst case analysis that there are more nodes at the maximum deviation.

6.1. Worst Case Calculation

In a 5 ms cycle, a fast node N_F has a real cycle duration $c_F = 4992.5$ µs, whereas a slow node N_S might have 5007.5 µs. As for both with a nominal microtick m duration of 25 ns, the cycle would last 200,000 µT. The m_S of a slow node lasts

$$m_S = \frac{5007.5 \text{ ns}}{200{,}000} = 25.0375 \text{ ns}, \tag{2}$$

whereas the m_F of a fast node lasts

$$m_F = \frac{4992.5 \text{ ns}}{200{,}000} = 24.9625 \text{ ns}. \tag{3}$$

When the fast node expects the same sync frame of the next cycle from the slow node, only $\frac{c_F}{m_S}$ = 199,400 µT passed for the slow node. Therefore, the fast node would store one rate calculation value of 600 µT, whereas the slow node would store −600 µT.

Values greater than ±601 are not possible by the specification because higher clock drifts are forbidden. However, additionally, the cluster drift damping reduces the applied value in our case by a maximum of 2 µT towards zero. If there was only one fast node, the slow nodes would measure −600 but eliminate this value due to the fault tolerant midpoint algorithm. This would apply vice versa if there was only one slow node.

If there was a distribution of slow and fast nodes, the FTM algorithm would then use both values for calculating the midpoint. Therefore, the result would be $\frac{1}{2} \cdot (600 + 0) = 300$ in an extreme case because the offset of the slow nodes relative to one another is zero.

Assuming the system is in an extreme state, i.e., starting from a correction of 300 µT, we then would apply our external correction of $e \in \{2, 3, \ldots 7\}$, and the cluster drift damping might reduce 2 µT per cycle. The actual applied rate correction values for a rate correction of 7 would then be $305, 310, 315, \ldots$, until we reach the maximum of 601. This acceleration phase would last $\frac{600-300}{5} = 60$ double cycles.

The deceleration could go quicker, as the cluster drift damping would help. Assuming we started from a value of 600, the following values would be $591, 582, \ldots, 303$. In total, it would take $\frac{591-303}{9} = 32$ double cycles.

Then, we would need to adjust the FlexRay cycle, so that its length is equal to the length of the Ethernet timebase. As Ethernet allows only a deviation of 100 ppm, the theoretical maximum offset is 320 µT.

If we correct by 601 µT, we move the FlexRay cluster by 281 µT relatively to the Ethernet every cycle. Assuming we only want to adjust the FlexRay cycle start, in the worst case, we need to correct it by 2.5 ms. This means that it would take 8.8 s in the worst case for the correction, without acceleration and deceleration phase. These take 92 double cycles, i.e., 920 ms, but within this time also correct on average half the distance after we reached the maximum. The total worst case correction time T is therefore

$$T = 8.8 \text{ s} + \frac{920 \text{ ms}}{2} \approx 9.3 \text{ s}. \tag{4}$$

6.2. Algorithm Proposal

As described in the previous chapter, we exploit the rate correction, in order to achieve a fast adjustment of the cycle. Only after this is achieved will we use it to slightly adjust the FlexRay cycle towards the Ethernet time.

In addition, because we can only apply a factor f of $\{-1, 0, 1\}$, we need to find out in which direction it would be more efficient to correct.

The offset correction only has a minor impact on the performance. Therefore, we propose the following procedure:

1. determine offset to Ethernet;
2. if it is very high, caculate acceleration factor;
3. if we cannot accelerate, stay at high value;
4. if we are getting close, decelerate;
5. distribute the factor over the network, so that all sync nodes apply it simultaneously;
6. continuously measure and apply clock correction value once in a while.

This way, like a leap second keeps the earth time towards sun time, we keep in the same way the FlexRay time close the Ethernet time.

7. Prototype Implementation

Since we only have a very limited set of values to choose from, we built up a system to derive the values empirically.

To derive comparable results, we adjust the FlexRay cycle to a 5 ms tick of the gPTP clock. Of course, it would be possible to divide a second into 64 cycles and adjust the cycle zero to the full second tick. The only difference would be the longer synchronization time. Therefore, in the setup, we will only adjust to a full 5 ms tick. The worst case in this scenario would be an offset of the clocks of 2.5 ms. For every experiment, we let the clocks drift until they reach this time difference exactly. Then, we start the synchronization process. Hence, the results of every measurement are comparable.

7.1. Test System

The test setup is shown in Figure 4. The automotive Ethernet backbone is represented by one Ethernet only and the Gateway node, which perform clock synchronization according to 802.1AS. The gateway ECU is also connected to a FlexRay system and therefore performs the calculation of our algorithm and distributes the factor for the external clock synchronization among the other FlexRay nodes.

To be as close to Audi's existing FlexRay implementation, we also have three additional sync nodes. This setup is proven to be stable and avoids cliques.

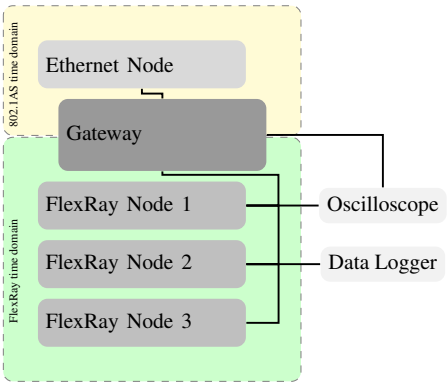

Figure 4. Experimental setup to prove the concept.

According to the FlexRay specification, all nodes must be configured to the same FlexRay parameters for the external clock correction (being *pExternOffsetCorrection* and *pExternRateCorrection*). These values cannot be modified on the fly. To change the value, the node must be halted, reconfigured with the new value, and the FlexRay startup process must be started again.

First, the two Ethernet nodes will synchronize with gPTP. Afterwards, the gateway node will calculate the factor for offset and rate correction and send it on the FlexRay bus to all other nodes. All nodes will apply that value at the end of the FlexRay cycle, where the clock correction is performed.

Every node transmits its calculated clock correction value. This value is used to check the implementation and efficiency of our algorithm. To check the precision, we generate a digital pulse every 5 ms. Additionally, all set an output pin upon start of the FlexRay cycle. With an oscilloscope, we can measure the time offset between both pulses and let it derive statistical data.

7.2. Results

The first experiment is just to compare the precision of the freely running gPTP T_p nodes and the FlexRay T_F system itself, without any external interference. The results in Table 1 show the high precision of our gPTP network. The FlexRay network runs slightly slower and with a higher standard deviation.

Table 1. Accuracy of independent clocks.

Value	Min	Max	Mean	σ
T_F	5.0001 ms	5.0004 ms	5.0002 ms	47.202 ns
T_p	5.0000 ms	5.0000 ms	5.0000 ms	7.9921 ns

7.2.1. Maximum Possible Rate Correction

In the first experiment, we allow the maximum rate correction of seven microticks. After starting the correction at the maximum offset of 2.5 ms (that is approximately 1400 MT), the applied rate correction values ramps up in a linear fashion up to the allowed maximum of around 500 MT. More would lead to instabilities and therefore the FlexRay node would shut down. One can see that we apply a positive factor, which means that we lengthen the FlexRay cycle slightly.

As one can see in Figure 5, the overall synchronization needs 2.5 s. Due to the high correction value, if corrections are needed, they lead to a high deviation, which needs higher frequency in correction towards the gPTP time.

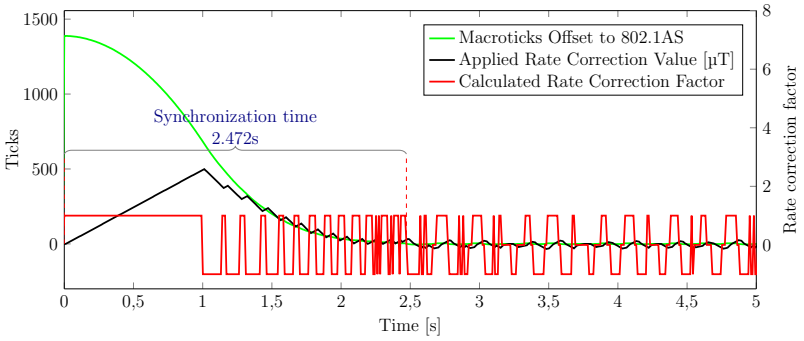

Figure 5. Maximum of 7 for pExternRateCorrection.

7.2.2. Rate Correction Equals Cluster Drift Damping

The second extreme case is when the rate correction equals our parameter of cluster drift damping, which is 2. As a result, in this specific case, only the natural drift and our seven ticks per cycle offset correction move the FlexRay cluster close to gPTP timebase. Afterwards, due to natural rate correction

of the cluster in the negative range, we can apply a negative factor, which accumulates with a negative calculated rare correction value. In Figure 6, one can see the linear approach phase and afterwards slow, nonlinear corrections. The highest measured applied value was 1, whereas the lowest was −9.

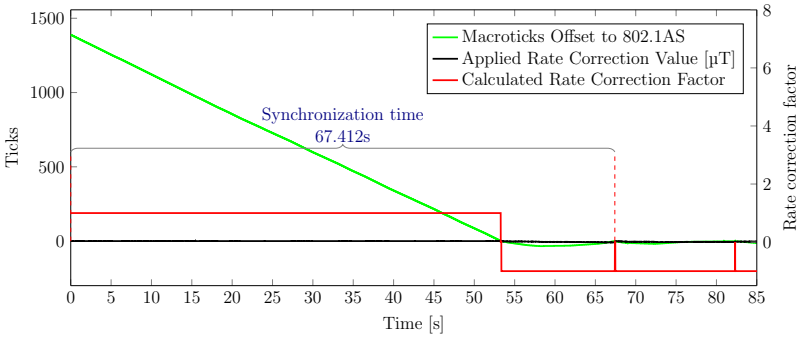

Figure 6. pExternRateCorrection equals pClusterDriftDamping.

7.2.3. Rate Correction Values

The remaining rate correction values are of no surprise. They take more time than the value 7 but stay more precise once the clusters are in sync.

Table 2 contains all the results, including the previously mentioned. Additionally, we give the standard deviation after the sync was completed. This represents the accuracy of the combined clock.

Table 2. Rate correction values.

Value	Sync Duration	σ
2	67.4 s	n/a
3	4.12 s	119.24 ns
4	3.11 s	163.76 ns
5	2.94 s	190.14 ns
6	2.57 s	211.82 ns
7	2.47 s	233.29 ns

8. Conclusions

In this paper, we presented the synchronization of the well established automotive bus system FlexRay to the Ethernet with 802.1AS based clock synchronization. Even over the boundaries of different technologies, we gave the possibility to establish a strict time-triggered communication. This allows the system designer to minimize end-to-end transmission latency wherever needed. This then also minimizes power consumption. Energy that would be needed to over-acquire data, because neither transmission time nor age of the data was known, is no longer needed. The optimization of the network with a focus on the timing always also optimizes the power consumption.

Additionally, the already used FlexRay timestamp (cycle counter and macrotick counter) can be used to get a fraction of a common timebase, which is synchronized over the whole car. This could easily be extended with the remainder of the timestamp as a payload signal. Then, the FlexRay network is included in the common global timebase of the car, allowing its sensors and actuators to distribute precise timestamps in their data.

In the future, we plan to investigate the TSN standards even further. We then would design an automotive system completely time triggered. In first real-world applications, only the FlexRay traffic is predictable, whereas the Ethernet traffic is simulated to check worst case end-to-end latencies.

Further work will be spent on including the CAN and LIN networks. Though Autosar offers solutions, no safety is guaranteed, but crucial for the autonomous driving. For a holistic timebase, these must be included.

To compare sensor data among cars and with external measurement equipment, it would be helpful to synchronize the time of the car to the physical time. Car-to-car communication would then allow also sensor fusion beyond the boundaries of the car. In addition, the data acquired by the car can be used in other applications.

Author Contributions: Conceptualization, P.M.; Data curation, E.S.; Investigation, P.M.; Methodology, P.M.; Project administration, P.M.; Resources, M.H.; Software, E.S.; Supervision, P.M. and M.H.; Visualization, E.S.; Writing-original draft, P.M.; Writing-review & editing, E.S. and M.H.

Funding: This research received no external funding.

Acknowledgments: This research was done, while Paul Milbredt was working for AUDI AG, whom we thank not only for all the resources but also for the permission to publish.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Malaguti, G.; Dian, M.; Ferraresi, C.; Ruggeri, M. Comparison on technological opportunities for in-vehicle Ethernet networks. In Proceedings of the 2013 11th IEEE International Conference on Industrial Informatics (INDIN), Bochum, Germany, 29–31 July 2013; pp. 108–115.
2. Steinbach, T.; Korf, F.; Schmidt, T.C. Comparing time-triggered Ethernet with FlexRay: An evaluation of competing approaches to real-time for in-vehicle networks. In Proceedings of the 8th IEEE International Workshop on Factory Communication Systems, Nancy, France, 18–21 May 2010; pp. 199–202.
3. Hu, M.; Luo, J.; Wang, Y.; Veeravalli, B. Scheduling periodic task graphs for safety-critical time-triggered avionic systems. *IEEE Trans. Aerospace Electron. Syst.* **2015**, *51*, 2294–2304. [CrossRef]
4. Zinner, H.; Noebauer, J.; Seitz, J.; Waas, T. A comparison of time synchronization in AVB and FlexRay in-vehicle networks. In Proceedings of the Ninth International Workshop on Intelligent Solutions in Embedded Systems, Regensburg, Germany, 7–8 July 2011; pp. 67–72.
5. Lee, Y.S.; Kim, J.H.; Jeon, J.W. FlexRay and Ethernet AVB Synchronization for High QoS Automotive Gateway. *IEEE Trans. Veh. Technol.* **2017**, *66*, 5737–5751. [CrossRef]
6. 802.1AS-2011—IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. Available online: https://ieeexplore.ieee.org/document/5741898/ (accessed on 10 August 2018).
7. Stanton, K.B. Distributing Deterministic, Accurate Time for Tightly Coordinated Network and Software Applications: IEEE 802.1AS, the TSN profile of PTP. *IEEE Commun. Stand. Mag.* **2018**, *2*, 34–40. [CrossRef]
8. FlexRay Consortium, FlexRay Communications System Protocol Specification Version 2.1 Revision A. Available online: https://svn.ipd.kit.edu/nlrp/public/FlexRay/FlexRay%E2%84%A2%20Protocol%20Specification%20V2.1%20Rev.A.pdf (accessed on 10 August 2018).
9. Krieger, O. How Ethernet helps building a scalable network architecture. In Proceedings of the IEEE-SA Ethernet & IP @ Automotive Technology Day 2016, Paris, France, 20–21 September 2016.
10. Barton-Zeipert, S. Fahrbahnprofilerfassung für ein aktives Fahrwerk. Ph.D. Thesis, Helmut-Schmidt-Universität/Universität der Bundeswehr Hamburg, Hamburg, Germany, 7 July 2014.
11. Hudi, R. Die Automobilindustrie im (radikalen) Umbruch—Chancen, Risiken, Trends, Herausforderungen. In Proceedings of the Internationaler Fachkongress Fortschritte in der Automobil-Elektronik, Ludwigsburg, Germany, 14–15 June 2016.
12. Zeng, W.; Khalid, M.; Chowdhury, S. A qualitative comparison of FlexRay and Ethernet in vehicle networks. In Proceedings of the 28th Canadian Conference on Electrical and Computer Engineering (CCECE), Halifax, NS, Canada, 3–5 May 2015; pp. 571–576.
13. Mehmed, A.; Punnekkat, S.; Steiner, W. Deterministic Ethernet: Addressing the Challenges of Asynchronous Sensing in Sensor Fusion Systems. In Proceedings of the 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Denver, CO, USA, 26–29 June 2017; pp. 22–28.

14. Specification of Time Synchronization over FlexRay, AUTOSAR CP Release 4.3.0, Document ID 675 AUTOSAR_SWS_TimeSyncOverFlexRay, 30 December 2016.
15. Milbredt, P.; Glaß, M.; Lukasiewycz, M.; Steininger, A.; Teich, J. Designing FlexRay-based automotive architectures: A holistic OEM approach. In Proceedings of the Conference on Design, Automation and Test in Europe, Dresden, Germany, 12–16 March 2012; pp. 276–279.