A Case Study of Clock Synchronization in Flexray

Alexander Hanzlik
Vienna University of Technology, Real-Time Systems Group
Treitlstr. 1-3/182-1, 1040 Vienna, Austria
hanzlik@vmars.tuwien.ac.at

Abstract

This paper presents a case study on the performance of a distributed clock synchronization algorithm used in Flexray, a communication protocol designed to meet the requirements of dependable, fault-tolerant real-time applications. The Flexray industry consortium drives forward the standardization of a fault-tolerant communication system for advanced automotive applications. In this case study we will analyze two different configurations for typical automotive applications by means of simulation. The focus of the simulation experiments is the assessment of performance and stability of the Flexray clock synchronization algorithm in the presence of varying clock drift rates. For the analysis we will use SIDERA, a simulation model for time-triggered distributed systems.

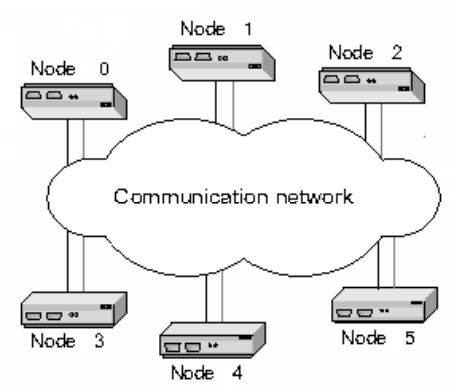

Figure 1. System structure

1. Prerequisites and Assumptions

1.1. System Structure

For the considerations in this paper, we assume that a distributed real-time system can be built by repetitive use of the following components:

Node. A computational unit that executes a part of a distributed application. Each node maintains a local clock.

Communication network. A shared communication resource connecting nodes.

Cluster. A set of spatially separated nodes that exchange messages via a communication network and that execute a distributed application in a cooperative manner.

1.2. Local Clocks

A *local clock* is a device for time measurement that contains a counter and a physical oscillation mechanism (e.g. a quartz oscillator) that periodically generates an event to increase the counter. This event is called the *microtick* of the local clock. The counter can be modified such that the speed of the local clock can be increased or decreased by application of an *adjustment value*.

1.3. Global Time

In time-triggered systems, all actions (i.e. the issuing of control signals to objects in the environment) are derived from a global notion of time which is established among all nodes by periodical execution of a distributed *clock synchronization algorithm* ([9], [6], [13], [11], [3], [1]). Flexray utilizes the concept of *microticks* and *macroticks* [8]. Microticks correspond to the local oscillator ticks at each node, while macroticks represent the global notion of time used to trigger actions and to order events. Each node generates a macrotick by selecting a number of microticks and synchronizes its macrotick by dynamically increasing or decreasing the number of microticks per macrotick, according to the clock state correction term that is delivered periodically by a clock synchronization algorithm. All nodes adjust their local clocks at the same point in global

time. The internally synchronized global time proceeds in units of macroticks. The macrotick counter at each node represents the node's view of global time (cluster time).

1.4. Communication

In distributed systems with a shared communication resource the access to the communication medium has to be regulated such that all nodes are able to deliver messages within an upper bound in time. In Flexray the access to the communication medium is controlled by a collision-free, Time Division Multiple Access strategy (TDMA). Each node traverses a (locally stored) global *communication schedule* in a cyclic manner. In this communication schedule, realtime is divided into *slots*. Each node is assigned one or more slots during which it is allowed to send a message or *frame*. The send time of the frame (in macroticks) is referred to as the *action point* of the node.

Figure 2 shows the principle of operation.

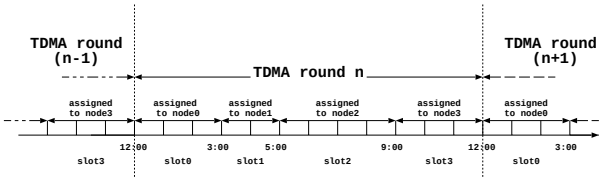

Figure 2. TDMA scheme

1.5. Clock Synchronization

Synchronization is the action of making different processes in a computer network or different parts of a circuit or different clocks to agree on a same time reading. In the context of multiprocessor and distributed systems, synchronization ensures that operations occur in the logically correct order, and it allows the establishment of causal implications between events in different computational units [12].

Clock synchronization is the periodic activity of determination and application of a *clock correction term* for each local clock to achieve agreement among an ensemble of clocks.

Every node periodically performs the following steps:

Step 1: Read the values of a well-defined ensemble of other clocks (*remote clock reading*). Due to the presence of possibly varying communication delays (*jitter*) and due to the existence of clock drifts, getting an exact knowledge of a remote clock value is not feasible. Thus, only estimates of the remote clock values can be acquired [1].

Step 2: Determine a clock correction term according to the remote clock readings obtained in Step 1.

Step 3: Apply the clock correction term obtained in Step 2 to the local clock (*clock adjustment*). The clock can be corrected by discrete adjustment (*state correction*), continous adjustment (*rate correction*) or a combination of the two. Flexray uses a combination of clock state and clock rate correction.

2. Flexray protocol mechanisms

2.1. Communication

Flexray uses a TDMA access control scheme to arbitrate the communication medium. In TDMA protocols each node is permitted to periodically utilize the full transmission capacity of the medium for some fixed amount of time called TDMA slot which is the interval from one transmission start to the next one.

Communication cycle. The Flexray communication cycle consists of a mandatory *static segment*, an optional *dynamic segment*, an optional *symbol window* and a mandatory *network idle time (NIT)* (Figure 3).

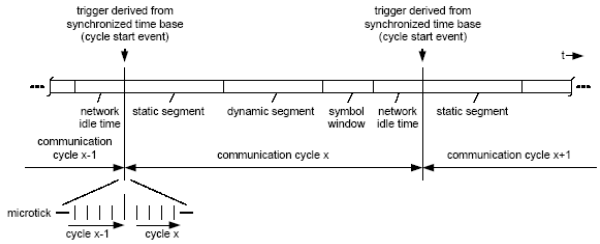

Figure 3. Flexray communication cycle

The static segment contains a configurable number of static slots, each slot assigned to one node via a *frame ID*. All static slots have the same duration (in macroticks) (Figure 4). Within the static segment a static TDMA scheme is applied to coordinate transmissions. The network idle time contains the remaining number of macroticks within the communication cycle not allocated to the static segment, dynamic segment, or symbol window and serves as a phase during which the node calculates and applies clock correction terms.

Within the dynamic segment a dynamic mini-slotting based scheme is used to arbitrate transmissions; the duration of communication slots may vary in order to accommodate frames of varying length. Within the symbol window a single symbol may be sent; the duration of the symbol window

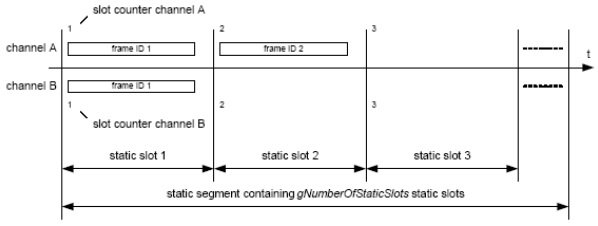

Figure 4. Flexray static segment

is a global constant for a given cluster. Both the dynamic segment and the symbol window are not used for clock synchronization and are therefore not regarded for the following considerations. For more information regarding these segments refer to [5].

2.2. Remote clock reading

Flexray uses the *a-priori* known action points of nodes (Section 1.4) for remote clock estimation. The communication schedule contains dedicated slots used for clock synchronization. Frames sent in these slots are referred to as *sync frames*. During each slot, the time difference between the expected arrival time (the sender's action point from the communication schedule) and the actual arrival time is measured in locally used microticks at the receivers [1]. Time differences obtained from sync frames are used for calculation of the clock correction terms.

Sync frames are sent during the static segment of the communication cycle. Every node measures and stores up to 16 time differences (in microticks) between the expected and the observed arrival times of all sync frames received. Only valid sync frames are used for clock correction term calculation.

2.3. Calculation of the clock correction terms

Flexray uses a combination of clock state and clock rate correction (Section 1.5) by periodical determination and application of a *state correction term* and a *rate correction term*.

The Fault-Tolerant Midpoint (FTM) algorithm is used for clock state correction. The valid time differences are sorted, the k largest and the k smallest values are discarded. The largest and the smallest of the remaining values are averaged for the calculation of the midpoint value which serves as the state correction term *vOffsetCorrection* that indicates by how many microticks the node's communication cycle length should be changed [5]. The term k is dependant on the number of values used for calculation (k=0 for up to 2 values, k=1 for up to 7 values, k=2 for more than 7 values).

[1]The measured deviation may be corrected by a known minimum propagation delay of the communication network.

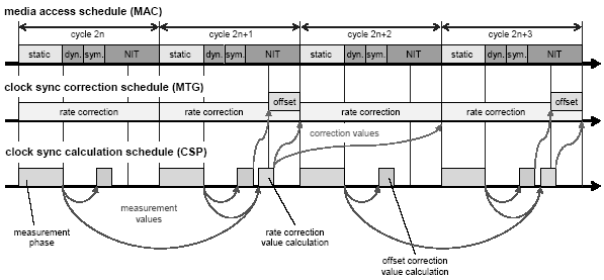

Figure 5. Flexray - clock state and clock rate correction

The rate correction term is determined by comparing the corresponding measured time differences from two successive communication cycles. A set of values is built such that pairs of previously stored deviation values are selected and the difference between the values within a pair is calculated (two values belong to a pair if both values indicate valid time differences that stem from sync frames received in the same slot on consecutive communication cycles). The FTM algorithm (see above) is executed on this set of values. The result is a clock rate correction term *vRateCorrection* that indicates by how many microticks the node's communication cycle length should be changed.

2.4. Clock adjustment

Flexray uses the clock correction values to dynamically adjust the number of microticks in each macrotick.

Clock state correction takes place during the network idle time NIT (Section 2.1) in odd communication cycles (i.e. every second communication cycle, Figure 5). Clock rate correction takes place during the entire communication cycle.

The clock correction terms are applied such that they are uniformly spread over a whole *relevant time range*. The relevant time range for *vOffsetCorrection* is the time between *gOffsetCorrectionStart* in the NIT and the start of the next communication cycle. For *vRateCorrection*, the relevant time range is the whole communication cycle.

3. Simulation experiments

We investigate the performance of the Flexray clock synchronization mechanism by means of simulation experiments using SIDERA, a simulation model for time-triggered distributed systems. SIDERA allows the simulation of single- and multi-cluster time-triggered systems

including real-time protocol services like system startup, communication, clock synchronization, membership service and protocol error detection and handling. A failure simulation module allows to investigate the stability of the systems under investigation in the presence of node failures. More information about SIDERA can be found in [7].

3.1. System configuration

We use two different system configurations for typical automotive applications. Each configuration consists of one cluster with 15 nodes.

3.1.1 Node configurations

The node configurations are equal for both clusters (Table 1). The nodes are numbered from 0 to 14.

Macrotick duration	1 μsec ($10^{-6} sec$)
Microticks per macrotick	40
Nominal Frequency F_s	40Mhz
Frequency Tolerance $\Delta f(0)$	10 ppm (10^{-5})
Frequency Stability $\Delta z_f(\Delta t)$	200 ppm (2×10^{-4})
Frequency Margin $\Delta f(0) + \Delta z_f(\Delta t)$	210 ppm ($2,1 \times 10^{-4}$)
$\rho_i(t)$	Drift rate node i at time t

Table 1. Node parameters and oscillator characteristics

Local clock characteristics. The macrotick has a duration of 1 microsecond. The number of microticks per macrotick is 40. It is derived from the nominal frequency of the physical oscillation mechanism of the local clock.

Oscillator characteristics. Table 1 shows the characteristics of the physical oscillation mechanism (Section 1.2) for all nodes. The *nominal frequency* F_s is the intended number of oscillations per second (40Mhz). The *frequency tolerance* $\Delta f(0)$ [2] is a measure for the deviation from the nominal frequency F_s at startup time [3]; it is a symmetric interval around F_s (i.e. the maximum deviation from the nominal frequency at startup time $t = 0$ is $\frac{\Delta f(0)}{2}$).

Drift rate. The drift rate ρ_i defines the deviation of the frequency of node i from the nominal frequency F_s in parts per million (ppm). This is equal to the deviation of clock i from a perfect clock in seconds per second. Different clock drift rates mean different microtick durations.

For the simulation experiments, the drift rate ρ_i of node i at startup time is equal to

$$\rho_i(0) = \frac{\Delta f(0)}{2} - i\frac{\Delta f(0)}{n-1} \quad (1)$$

Equation 1 ensures that that the node drift rates are equally spread over the whole frequency tolerance interval $\Delta f(0)$ ($\rho_0 = +5e-6, \rho_1 = +4,286e-7, ..., \rho_{14} = -5e-6$).

The drift rate of node i at time t is equal to

$$\rho_i(t) = \rho_i(0) + z_{f_i}(t), z_{f_i}(t) <= \Delta z_f(\Delta t) \quad (2)$$

The relation between frequency f_i and drift rate ρ_i of node i at time t is

$$f_i(t) = F_s(1 + \rho_i(t)) \quad (3)$$

The microtick duration at node i is $\frac{1}{f_i(t)}$. At time t, the local clock of node i is *fast* if $\rho_i(t) > 0$ and *slow* if $\rho_i(t) < 0$.

It can be seen from Equation 1 and from Equation 3 that at startup time node 0 has the fastest clock and that node 14 has the slowest clock.

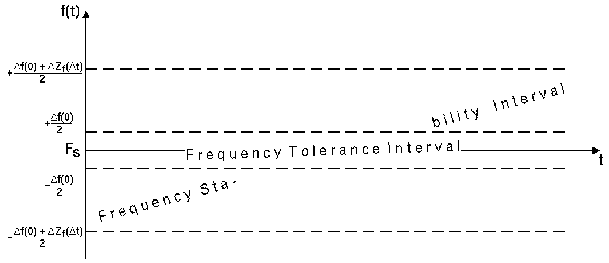

Figure 6. Oscillator tolerance intervals

There are several influences on oscillator frequency. A major influence is that of operating over variations in temperature [2] besides other influences like gravity, vibration, electromagnetic interference or aging. The function $z_f(t)$ (Equation 2) corresponds to the sum of all these influences on oscillator frequency at time t. The *frequency stability* $\Delta z_f(\Delta t)$ denotes the maximum deviation from the frequency at startup time during operation.

Figure 6 shows the relation between nominal frequency F_s, frequency tolerance $\Delta f(0)$ and frequency stability $\Delta z_f(\Delta t)$.

3.1.2 Cluster configurations

Table 2 summarizes the parameters for cluster 1 and cluster 2. Static slots are numbered from 0 to $m-1$ (m being the

[2] In crystal oscillator data sheets, $\Delta f(0)$ is also known as *calibration tolerance* at a given temperature (e.g. 10 ppm at 25 °C).

[3] The startup time is the point in time from which on the oscillator is used for time measurement.

number of static slots for each cluster configuration). Clock sync slots are slots during which sync frames are sent.

	Cluster 1	Cluster 2
Number of nodes n	15	15
Communication cycle length	5000 MT	5000 MT
Static Segment	3026 MT	3479 MT
Dynamic Segment	1960 MT	1498 MT
NIT	14 MT	23 MT
Static Slots	89	71
Static Slot length	34 MT	49 MT
Clock sync nodes	0,5,10,14	0,5,10,14
Clock sync slots	61, 71, 81, 89	0,23,47,70

Table 2. Cluster parameters

Simulation time T_{sim}. All simulation experiments have a duration T_{sim} of 100 communication cycles.

3.1.3 Prerequisites and Assumptions

Notation. The following notational conventions will be used in the course of the following considerations:

- **t** always denotes a point in simulation time: $t \in [0, T_{sim}]$.
- t_{c_x} denotes the start time of communication cycle x (e.g. $t_{c_{10}}$).
- t_{NIT_y} denotes the start of network idle time of communication cycle y (e.g. $t_{NIT_{10}}$).
- **i, j** always denote node numbers (e.g. ρ_i is the drift rate of node i).
- The drift rate ρ_i is given in s/s.
- $e \pm y$ will be used to signify $\times 10^{\pm y}$ (e.g. 7e-4 means 7×10^{-4}).

Offset. The $offset_C$ is the maximum deviation of the microtick counters mt of two nodes in cluster C at time t.

$$offset_C(t) = max(mt_i(t)) - min(mt_i(t)) \quad (4)$$

Precision. The *precision* is the maximum offset of any two nodes in cluster C observed during simulation time.

$$\Pi_C = max(offset_C(t)) \quad (5)$$

Performance. Precision Π_C is used as a performance measure for the clock synchronization algorithm: performance improves as the precision values decrease.

Stability. The clock synchronization algorithm is *stable* with regard to a given cluster configuration if precision Π_C is less than the nominal duration of a macrotick (40 microticks for the configurations used in the course of the simulation experiments):

$$\Pi_C < 40 \quad (6)$$

The chosen stability criterion is based on the *reasonableness condition* [8] which demands that all local implementations of the global time satisfy the condition

$$g > \Pi_C \quad (7)$$

for the granularity g of the global time base. This condition ensures that the synchronization error is bounded to less than the duration between two macroticks.

Generally, the stability criterion has to be chosen according to the demands of the distributed application. In applications with less stringent synchronization requirements a softer stability criterion than the reasonableness condition may be sufficient.

Initial synchronization. Initial synchronization among an ensemble of clocks is usually established using a startup algorithm [14]. In Flexray, the initial clock rate correction term *vRateCorrection* at a node is determined from the reception times of two consecutive *startup frames* during the cluster startup phase before the node becomes operational (and starts execution of the clock synchronization algorithm). That means that node i becomes operational with an initial value of *vRateCorrection* that compensates for the drift rate of the local clock, ρ_i (Table 1). In the simulation experiments, all nodes start at the same time with a microtick counter value of 0 (i.e. the states of the local clocks are in perfect agreement) and with an initial clock rate correction term *vRateCorrection* of 0. The clock synchronization algorithm is used to determine the initial value of *vRateCorrection*.

Assumptions. The following assumptions are made in the course of the following considerations:

- **A1** The duration of the network idle time **NIT** is considered to be negligible compared to the duration of the communication cycle.

3.2. Experiment 1: Stable clock drift rates

In Experiment 1, we determine the performance of the clock synchronization algorithm in case of stable clock drift rates. Each node i starts with a clock drift rate $\rho_i(0)$ according to Equation 1. All nodes maintain their initial drift rates during simulation time.

$$z_{f_i}(t) = 0 \rightarrow \rho_i(t) = \rho_i(0) \quad (8)$$

The drift rates of all nodes remain constant and within the shaded interval in Figure 7.

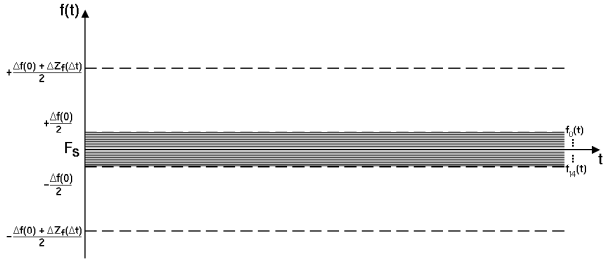

Figure 7. Experiment 1: Stable clock drift rates

Figure 8 shows the results for Experiment 1.

After start of simulation the clocks are running free until the first clock state correction (leftmost peak in Figure 8). All nodes apply new clock state correction values during the NIT of odd communication cycles and new clock rate correction values at the start of even communication cycles. Both Cluster 1 and Cluster 2 reach a precision of 5 microticks (0,125 microseconds).

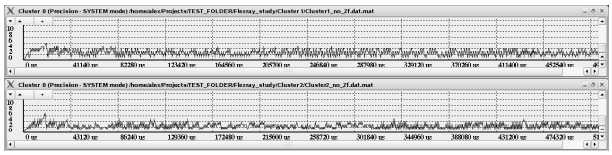

Figure 8. Experiment 1 - Precision

3.3. Experiment 2: Immediate clock drift rate change

In Experiment 2, we determine the performance of the clock synchronization algorithm in case of an immediate clock drift rate change at one node. Each node i starts with a clock drift rate $\rho_i(0)$ according to Equation 1. At time $t_{c_{10}}$ (the start time of communication cycle 10), node 0 immediately changes its clock drift rate such that its frequency f_0 reaches the frequency margin (Figure 9). Node 0 maintains this new drift rate till the end of simulation time.

$$z_{f_0}(t) = 0 \quad t < t_{c_{10}} \quad (9)$$

$$z_{f_0}(t) = +1e-4 \quad t \geq t_{c_{10}} \quad (10)$$

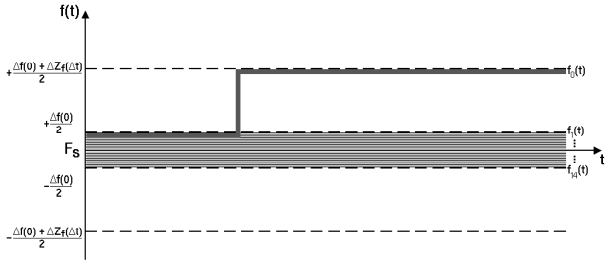

Figure 9. Experiment 2 - Immediate clock drift rate change at Node 0

We know from Section 2.4 that clock state correction is performed during the NIT of odd communication cycles (i.e. immediately before the start of even communication cycles). All nodes have performed clock state correction during the NIT of communication cycle 9 (starting at t_{NIT_9} immediately before the start of communication cycle 10 $t_{c_{10}}$).

Till $t_{c_{10}}$, the clock drift rates $\rho_i(t)$ at all nodes have been constant and have been compensated for by the clock rate correction algorithm. A clock state correction has taken place immediately before $t_{c_{10}}$. That means, that at time $t_{c_{10}}$, the deviation of all clocks shows a local minimum [4]. This minimum is non-zero: the Impossibility Result in [10] shows that it is not possible to internally synchronize the clocks of an ensemble perfectly due to the non-zero communication delay jitter in distributed systems. Even if all local clocks are driven by perfect oscillators with zero drift rate, this communication delay jitter causes an inaccuracy in remote clock reading and consequently in the determination of the clock state correction term.

$$offset_C(t_{c_{10}}) = min > 0 \quad (11)$$

The next clock state correction takes place during the NIT of communication cycle 11 starting at $t_{NIT_{11}}$ which is 2 communication cycles (10000 MT = 0,01s) apart from $t_{c_{10}}$ (Assumption A1).

The offset at $t_{NIT_{11}}$ (the time of the next clock state correction) is equal to

$$offset_C(t_{NIT_{11}}) = offset_C(t_{c_{10}}) + z_{f_0}(t) \times 0,01 \times F_s \quad (12)$$

The clock synchronization algorithm becomes unstable ($\Pi_C > 40$) because

$$z_{f_0}(t) \times 0,01 \times F_s = 40 \rightarrow offset_C(t_{NIT_{11}}) > 40 \quad (13)$$

[4] The offset immediately after clock state correction is less or equal to the offset immediately before clock state correction.

Figure 10 shows the results for Experiment 2.

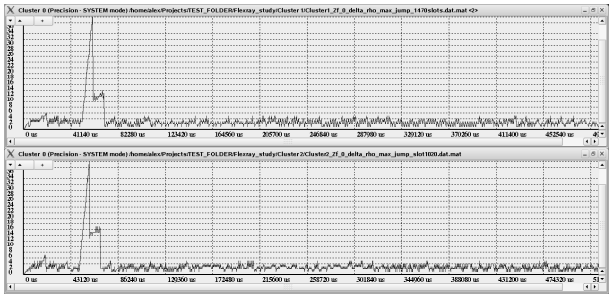

Figure 10. Experiment 2 - Precision

The immediate drift rate change at node 0 is indicated by the leftmost rising edge in the precision diagrams in Figure 10. Within a duration of 2 communication cycles, the offset exceeds 40 microticks. The clock state correction at $t_{c_{12}}$ reduces the offset (leftmost falling edge in Figure 10). However, the clock state correction leaves a remarkable offset of more than 10 microticks. This is because the clock drift rate change of node 0 has not yet been compensated by the clock rate correction algorithm (the clock rate correction terms applied during communication cycle 10 and 11 are based on clock readings from node 0 obtained during communication cycle 8 and 9, i.e. before the clock drift rate change of node 0). New rate correction values obtained from clock readings in communication cycle 10 and communication cycle 11 are applied at $t_{c_{12}}$. The rate correction values account for the clock drift rate change of node 0 causing the offset to remain stable between $t_{c_{12}}$ and $t_{c_{14}}$, the time of the next clock state correction (rightmost falling edge in Figure 10). At $t_{c_{14}}$, the clock state correction brings the offset to a minimum. The clock rate correction algorithm is also provided with accurate clock readings because the clock drift rates of node 0 and the other nodes are stable. From $t_{c_{14}}$, the clock synchronization algorithm reaches a precision of 5 microticks. This is equal to the performance of the clock synchronization algorithm in case of stable clock drift rates presented in Section 3.2.

3.4. Experiment 3: Linear clock drift rate change

In Experiment 3a, we determine the performance of the clock synchronization algorithm in case of linear clock drift changes at one node. Each node i starts with a clock drift rate $\rho_i(0)$ according to Equation 1. At the start of communication cycle 10, node 0 changes its clock drift rate from $+5e-6$ to $+1,05e-4$ within a duration of 10 communication cycles. Node 0 maintains the changed drift rate of $+1,05e-4$ till the end of simulation.

$$z_{f_0}(t) = 0 \quad t < t_{c_{10}} \tag{14}$$

$$z_{f_0}(t) = +2e-3t \quad t_{c_{10}} \leq t < t_{c_{20}} \tag{15}$$

$$z_{f_0}(t) = +1e-4t \quad t \geq t_{c_{20}} \tag{16}$$

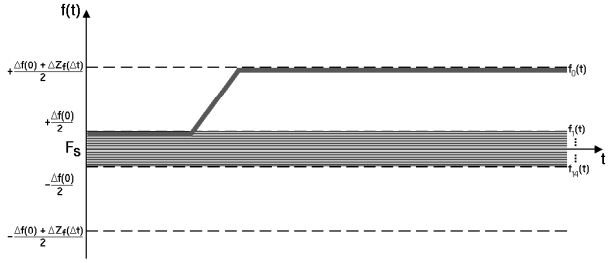

Figure 11. Experiment 3a - Linear clock drift rate change at Node 0

Figure 12 shows the results for Experiment 3a.

Although the final drift rate at node 0 is the same as in Experiment 2 (where the clock synchronization algorithm became unstable), the algorithm remains stable for both configurations. This is because the clock drift rate at node 0 changes slowly enough such that the rate correction part of the clock synchronization algorithm is able to keep the cluster stable during the clock drift rate change.

In both configurations, the cluster precision deteriorates during the drift rate change at node 0, starting at $t_{c_{10}}$. Figure 12 shows five peaks, one for each clock state correction during the drift rate change at node 0 (which takes 10 communication cycles). The clock rate correction part of the algorithm ensures that the precision is bounded and better than 16 microticks for both clusters. At time $t_{c_{20}}$, the start time of communication cycle 20, the clock drift rate change is complete. From this point in time, the clock synchronization algorithm is provided with accurate clock readings, causing the cluster precision to improve to 5 microticks. This is equal to the performance of the clock synchronization algorithm in case of stable clock drift rates presented in Section 3.2.

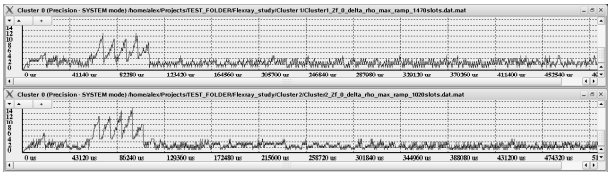

Figure 12. Experiment 3a - Precision

In Experiment 3b, we determine the performance of the

clock synchronization algorithm in case of linear clock drift rate changes at two nodes. Each node i starts with a clock drift rate $\rho_i(0)$ according to Equation 1. Like in Experiment 3a, at the start of communication cycle 10, node 0 changes its clock drift rate from $+5e-6$ to $+1,05e-4$ within a duration of 10 communication cycles (Equation 16). Node 14 also changes its clock drift rate from $-5e-6$ to $-1,05e-4$, starting at $t_{c_{10}}$, within a duration of 10 communication cycles.

$$z_{f_{14}}(t) = 0 \quad t < t_{c_{10}} \tag{17}$$

$$z_{f_{14}}(t) = -2e-3t \quad t_{c_{10}} \leq t < t_{c_{20}} \tag{18}$$

$$z_{f_{14}}(t) = -1e-4t \quad t \geq t_{c_{20}} \tag{19}$$

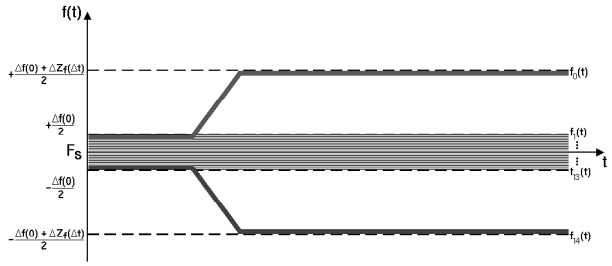

Figure 13. Experiment 3b - Linear clock drift rate change at Node 0 and at Node 14

Figure 14 shows the results for Experiment 3b, which are very similar to the results of Experiment 3a. The achieved cluster precision during clock drift rate change at node 0 and node 14 is double the precision achieved in Experiment 3a. This is not surprising and due to the fact that the deviation between node 0 and node 14 during the drift rate change is twice the deviation observed in Experiment 3a because node 14 also changes its clock drift rate. This selection of drift rate changes for node 0 and node 14 cause the worst possible impact on cluster precision: node 0, the fastest node, speeds up until it reaches one end of the frequency stability interval. Node 14, the slowest node, slows down until it reaches the other end of the frequency stability interval. Both nodes touch the frequency margin at the same time, which means that the offset introduced by the drift rate change at both nodes is the maximum possible one.

At time $t_{c_{20}}$, the start time of communication cycle 20, the clock drift rate change at both nodes is complete. From this point in time, the clock synchronization algorithm is provided with accurate clock readings, causing the cluster precision to improve to 5 microticks. Again, this is equal to the performance of the clock synchronization algorithm in case of stable clock drift rates presented in Section 3.2.

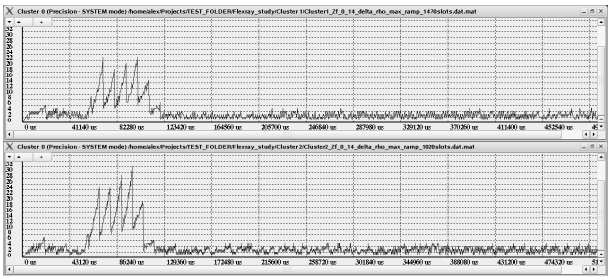

Figure 14. Experiment 3b - Precision

3.5. Experiment 4: Oscillating clock drift rate change

In Experiment 4, we determine the performance of the clock synchronization algorithm in case of oscillating clock drift rate changes at two nodes. Each node i starts with a clock drift $\rho_i(0)$ according to Equation 1. At the start of communication cycle 10,

- node 0
  - changes its clock drift rate from $+5e-6$ to $+1,05e-4$ within a duration of 10 communication cycles, then
  - changes its clock drift rate from $+1,05e-4$ to $-9,5e-5$ within a duration of 20 communication cycles, then
  - changes its clock drift rate from $-9,5e-5$ to $+5e-6$ within a duration of 10 communication cycles and
- node 14
  - changes its clock drift rate from $-5e-6$ to $-1,05e-4$ within a duration of 10 communication cycles, then
  - changes its clock drift rate from $-1,05e-4$ to $+9,5e-5$ within a duration of 20 communication cycles, then
  - changes its clock drift rate from $+9,5e-5$ to $-5e-6$ within a duration of 10 communication cycles.

Figure 16 shows the results for Experiment 4.

The selection of the node drift rate changes at node 0 and node 14 ensures that both nodes "visit" all allowed clock drift rates within the frequency margin of 210 ppm (Table 1). Like in Experiment 3b, this selection of drift rate changes also causes the worst possible impact on cluster precision.

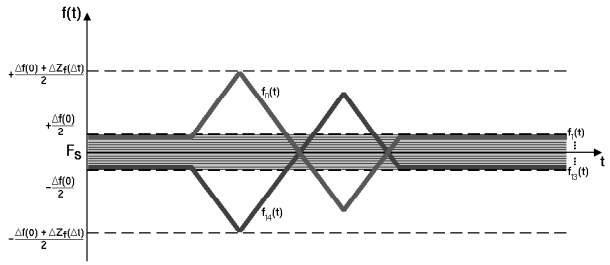

Figure 15. Experiment 4 - Oscillating clock drift rate change at Node 0 and at Node 14

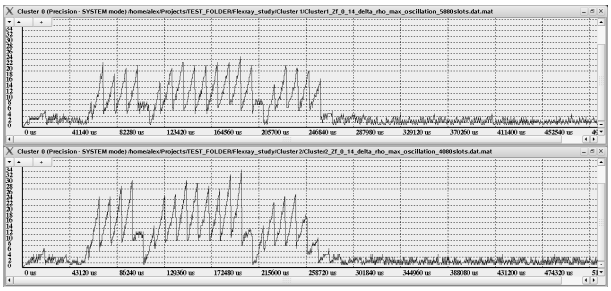

Figure 16. Experiment 4 - Precision

In both configurations, the cluster precision deteriorates during the drift rate change at node 0 and node 14, starting at $t_{c_{10}}$. Figure 16 shows 5 peaks followed by a local minimum at time $t_{c_{20}}$, the point in time where the clock drift rates stop to diverge. The next 10 peaks after the first local minimum denote the time between $t_{c_{20}}$ and $t_{c_{40}}$, where the clock drift rates of both nodes converge (Figure 15), become equal at $t_{c_{30}}$ and diverge again after $t_{c_{30}}$ till $t_{c_{40}}$. It is interesting to note that the precision between $t_{c_{20}}$ and $t_{c_{30}}$ (converging node drift rates) is not better than the precision between $t_{c_{30}}$ and $t_{c_{40}}$ (diverging node drift rates). The performance of the clock state correction part of the algorithm is constant during the whole time interval between $t_{c_{20}}$ and $t_{c_{40}}$ (the offset immediately after a clock state correction,indicated by the falling edges of the precision diagram in Figure 16). However, the clock rate correction part of the algorithm is provided with inaccurate clock rate estimations due to the node drift rate changes during this time interval between $t_{c_{20}}$ and $t_{c_{40}}$. The five last peaks denote the time between $t_{c_{40}}$ and $t_{c_{50}}$, where the clock drift rates of both nodes converge again (Figure 15) until they return to the values that they had at the start of simulation.

The clock rate correction part of the algorithm ensures that the precision is bounded and better than 25 microticks for cluster 1 and better than 36 microticks for cluster 2 during the time interval between $t_{c_{10}}$ and $t_{c_{50}}$, the time where node 0 and node 14 change their clock drift rates as described above. At time $t_{c_{50}}$, the clock drift rate changes at node 0 and node 14 are complete. From this point in time, the clock synchronization algorithm is provided with accurate clock readings, causing the cluster precision to improve to 5 microticks, which is is equal to the performance of the clock synchronization algorithm in case of stable clock drift rates presented in Section 3.2.

4. Conclusion

This paper presents a case study that aims at the assessment of performance and stability of the Flexray clock synchronization algorithm by means of simulation experiments.

The following lessons have been learned from the experiments:

- The achievable precision within a cluster is nearly independent from the clock drift rates of the different nodes if these clock drift rates are stable. Even if the clock drift rates of two nodes differ by the maximum possible value within the pre-defined frequency stability interval, the clock rate correction part of the algorithm compensates these clock rate differences and reaches a stable and minimum precision.

- The algorithm is quite robust in the presence of clock drift rate changes. In case of continous and linear clock drift rate changes that do not exceed the predefined frequency stability interval, the clock rate correction part of the algorithm compensates for these rate changes. However, cluster precision may deteriorate remarkably during the clock drift rate change. The degree of precision deterioration mainly depends on the speed of the clock drift rate change as shown in the experiments.

- The experiments have also shown that immediate clock drift rate changes may de-stabilize the clock synchronization algorithm, even if the clock drift rate differences are within the pre-defined frequency stability interval. Whether the algorithm remains stable or not with regard to the given cluster configuration depends on the degree of the clock drift rate change.

The Flexray clock synchronization algorithm is suited to solve the clock synchronization problem in the presence of changing clock drift rates with regard to the achievable precision within a single cluster. The focus of the investigations was on single-cluster systems with no synchronization

to an external time source (e.g. GPS [4] time). The increasing demand on fault-tolerant real-time applications in the automotive domain, like the emerging market for drive-by-wire systems, may also increase the complexity of the control systems deployed. One solution to cope with the increasing complexity of control systems is *clustering*, i.e. to build complex systems from single clusters into multi-cluster systems.

For this reason, further investigations of the algorithm are of interest:

- The cluster drift rate (i.e. the drift rate of the internally synchronized global time base within a cluster) depends on the average clock drift rate of the nodes that send synchronization frames. The clock drift rates of these nodes in turn depend on the frequency stability of the oscillators deployed. The impact of common mode effects on the crystal oscillators with regard to the cluster drift rate is an interesting topic of further research.

- The cluster drift rates of independent clusters may differ significantly. If synchronous operation is desired among an ensemble of clusters in a multi-cluster system, it is necessary to synchronize the global times at the different clusters by means of external clock synchronization, i.e. to synchronize the global time of a cluster to an external time source. The Flexray protocol provides means for external clock synchronization. The assessment of stability and performance of the Flexray clock synchronization algorithm in multi-cluster systems is another interesting topic of further research.

References

[1] E. Anceaume and I. Puaut. Performance Evaluation of Clock Synchronization Algorithms. Technical Report 3526, Institut de Recherche en Informatique et Systèmes Aléatoires, www.irisa.fr, October 1998.

[2] Hewlett-Packard Company. Fundamentals of Quartz Oscillators. HP application note 200-2, 1997.

[3] Flaviu Cristian, Houtan Aghili, and Ray Strong. Clock Synchronization in the Presence of Omission and Performance Failures, and Processor Joins. In Zhonghua Yang and T. Anthony Marsland, editors, *Global States and Time in Distributed Systems, IEEE Computer Society Press*. 1994.

[4] P.H. Dana. Global Posinitioning System (GPS) time dissemination for real-time applications. *Real-Time Systems*, 12:9–40, January 1997.

[5] Flexray. FlexRay Communications System Protocol Specification Version 2.1. Specification, FlexRay Consortium, 2005.

[6] J.Y. Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant Clock Synchronization. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 89–102, 1984.

[7] A. Hanzlik. SIDERA - A Simulation Model for Time-Triggered Distributed Systems. Research Report 62/2005, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2005.

[8] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[9] J. Lundelius and N. Lynch. A new Fault-tolerant Algorithm for Clock Synchronization. In *Proceedings of the 3rd annual ACM symposium on Principles of Distributed Computing*, pages 75–88. ACM, 1984.

[10] L. Lundelius and N. Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62:199–204, 1984.

[11] P. Ramanathan, K.G. Shin, and R.W. Butler. Fault-tolerant Clock Synchronization in Distributed Systems. *IEEE Computer*, 23(10):33–42, October 1990.

[12] F. A. Schreiber. Is Time a Real Time? An Overview of Time Ontology in Informatics. In W. A. Halang and A. D. Stoyenko, editors, *Real Time Computing*, pages 283–307. Springer, Berlin, Heidelberg, 1994.

[13] T.K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, 1987.

[14] Wilfried Steiner and Michael Paulitsch. The Transition from Asynchronous to Synchronous System Operation: An Approach for Distributed Fault-Tolerant Systems (Including Simulation). Research Report 26/2001, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2001.