

The Optimal Operating System for FlexRay-Based Applications

FlexRay is either introduced for its deterministic behavior or for its quick data transfer, depending on the application. Currently, its use in safety-related applications only plays a subordinate role. Criteria to be considered in the decision of which operating system to use together with FlexRay, besides deterministic behavior and performance, include memory protection and time monitoring. This article explains what is important in selecting the operating system and presents specific solutions offered by Vector Informatik in the context of AUTOSAR.

As a scalable high-speed communication system with deterministic behavior, FlexRay is the right solution for use as a data backbone, for distributed control systems or for safety-relevant applications in the automotive area. In contrast to event-triggered CAN communication, all messages are assigned to fixed communication intervals. This offers to each participating ECU a clearly timed availability of its data. The design of a FlexRay network requires that certain fundamental parameters be defined for all participating network nodes in a very early phase of development. These parameters include baudrate, cycle length, number and length of the slots in the static and dynamic segments or macrotick duration. This scheduling of communication processes is mapped in the communication-specific software components, but it can also influence the time structure of the application software.

The operating system (OS) coordinates the interplay of all participating software components. Both event-triggered and time-triggered operating systems are commercially available, each one having different services. Another available option is operating systems with memory protection.

Which operating system is best suited to a FlexRay-based application, and how should it be configured? In particular the question arises of whether a time-triggered operating system is absolutely necessary for synchronized FlexRay communication.

Fundamentals of event-triggered and time-triggered operating systems

Until now, event-triggered operating systems have usually been used in the automotive area. The broadest acceptance is enjoyed by the OSEK/VDX [1] OS, which in the future will also be available in the form of an ISO standard. The goal of an operating system is to provide, under optimal utilization of the hardware used, a supplemental run-time environment for managing functional units. Defined operating system services offer this functionality. In designing an application, at first time-independent, concurrent subtasks are defined. The outcome of this are tasks or interrupts compete for run-time assignment according to the operating system's scheduling algorithm:

- Tasks are triggered by alarms or events. A distinction is made between Extended Tasks and Basic Tasks. Extended Tasks are distinguished by their ability to wait for events.

- Interrupt Services Routines (ISRs) are hardware-specific and are triggered by the hardware periphery. They have higher priority than tasks and should therefore only be reserved for subtasks requiring the fastest possible reaction time.

In a time-triggered operating system all processes and actions under the control of the operating system are solely depending on the time. For the application this results in strictly deterministic time behavior.

AUTOSAR OS

AUTOSAR has taken the OSEK-OS as a basis and extended it to enable support of time-triggered functionalities. The specific properties of the AUTOSAR OS [2] are provided in four expansion stages, the so-called Scalability Classes (SC). Table 1 shows a summary of the relationship between these properties and the Scalability Classes. Only properties relevant to a FlexRay-based application are listed. These properties are explained in the following.

Scalability Class	Schedule Tables	Time Monitoring	Global Time/ Synchronization	Memory Protection	Comment
SC1	X				Event-triggered
SC2	X	X	X		Advanced development of OSEKtime
SC3	X			X	Advanced development of HIS-Protected OS
SC4	X	X	X	X	

[Table 1: FlexRay-relevant properties of the AUTOSAR OS]

The currently available specifications of BSW (Basic Software) and RTE (Runtime Environment) do not cover memory protection yet. This will be addressed in future versions of the BSW and RTE specifications.

Schedule Tables

The operating system manages multiple schedule tables. Each schedule table consists of a list of defined time-based actions that either activate tasks or send events to tasks.

Time monitoring

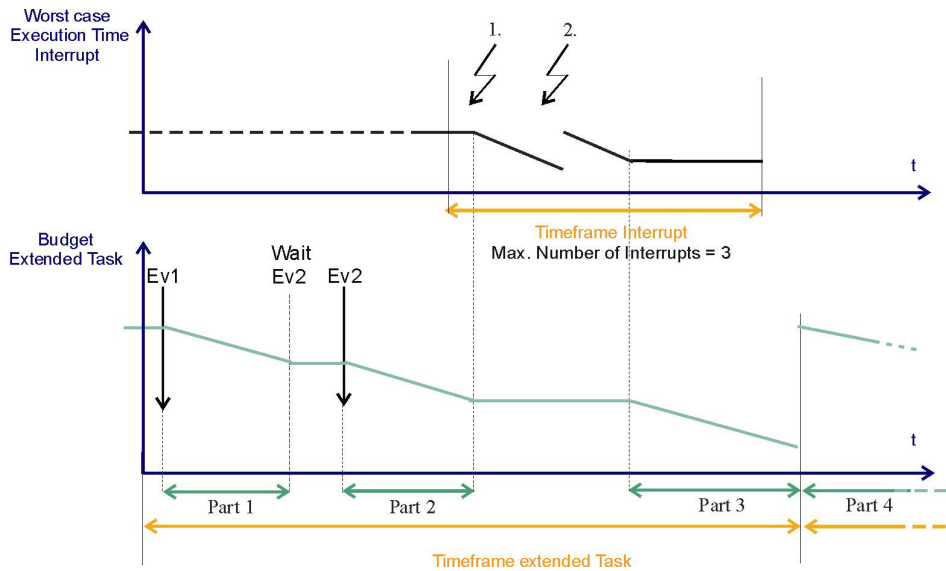
In a real-time system what is important is to process all tasks at the right times. In the design phase sub-tasks are allocated fixed time windows. To avoid delays at runtime a task may not tie up the CPU longer than specified in the design. Therefore the operating system monitors the execution time of each individual task and each interrupt. OSEKtime [3] provides classic deadline monitoring for this purpose: Tasks must be completed before their assigned deadlines. If a violation occurs, the overall system triggers a reset. This type of monitoring is inadequate for extended tasks that can wait for events. That is why the AUTOSAR-OS working group has defined execution time monitoring for each individual task and each interrupt. This monitoring is defined by various parameters in the configurator.

For each task:	For each interrupt:
<ul style="list-style-type: none"> • Timeframe: Monitoring time period (greater than the execution time; it is not absolutely essential that the task be completed within the monitoring time period; cf. Figure 1 Case A). • Execution time budget: Worst case execution time per monitoring time period. 	<ul style="list-style-type: none"> • Timeframe: See Task • Worst case execution time per execution • Max. number of interrupts per Timeframe
<p>If multiple activation of a Basic Task should be allowed in the operating system's configurator, the execution budget contains the total execution time of all activations of the affected task. In principle multiple activations are not possible for extended tasks.</p>	<p>Multiple interrupts may be allowed per Timeframe. However the worst case execution time only refers to a single loop pass. The effect of an additional interrupt per Timeframe is shown in cases B and C of figures 2 and 3.</p>

When a monitoring parameter is violated, depending on the configuration the operating system initiates a "Reset_OS", "Kill_Application", "Kill_Task" or "Kill_Interrupt" as a reaction to the error.

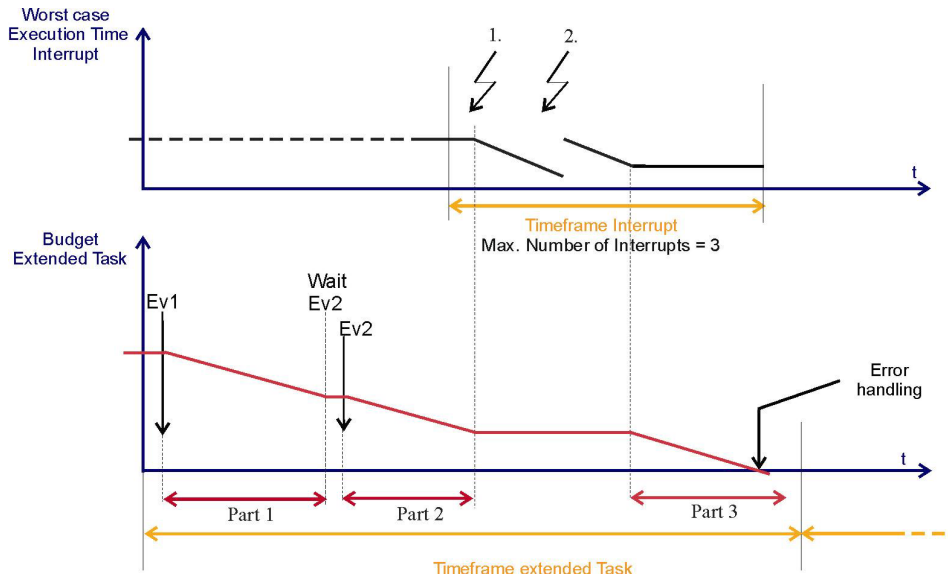
Figures 1-3 represent an AUTOSAR OS Scalability Class 2 with time monitoring tasks and interrupts.

Figure 1 shows the the monitoring of an application consisting of an extended task. The extended task is disrupted suspended several times by an interrupt.



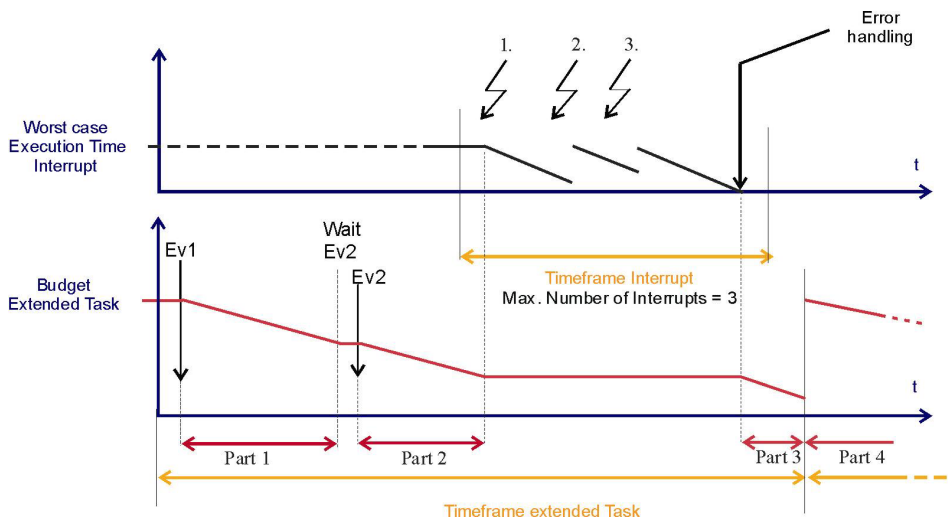
[Figure 1: Case A - Without error handling]

Due to a longer execution time of the extended task, e.g. for handling Event 1 (Ev1), this may result in error handling, see Figure 2. This exclusively affects the originator. According to the budget, time monitoring guarantees lower-priority tasks or interrupts access to the processor to execute their tasks.



[Figure 2: Case B - With error handling caused by excessively long execution time of a task]

If an interrupt lasts too long, as in Case C of Figure 3, it is cancelled so that the extended task can continue to execute.



[Figure 3: Case C - With error handling caused by excessively long execution time of an interrupt]

Global Time / Synchronization Support

Distributed control systems requiring synchronization of tasks beyond ECU boundaries need the support of a global

time provided to the operating system on a regular basis.

An application may synchronize to the global time either:

- Stepwise, as soon as the global time is made available and according to the maximum adaptation step from the configuration ("smooth"), or
- By complete adaptation in one step ("hard").

Memory Protection

Mechanisms for memory protection are necessary for early detection and prevention of disturbances by other modules. When software modules from different producers are integrated in an ECU the different software packets can be separated at runtime in terms of their memory areas. A processor with hardware support is also necessary. In addition, it should have a large memory and high processing speed, since memory protection mechanisms slow down operating system services by a factor of 1.5 to 2 on average.

Process of data exchange between FlexRay bus and application

Any of the four AUTOSAR-OS properties presented may be selected, and their costs and benefits must be weighed against one another from a production implementation point of view. Thereby consideration must be given to the OS for optimal support of the synchronization of the data exchange between FlexRay and the application.

In the FlexRay protocol up to 64 base cycles are combined to form a continuously repeating round. Each base cycle contains frames with different process data or signals for

different application tasks. With its global time the FlexRay protocol provides the foundation for synchronized data exchange. As soon as the data network has been synchronized this global time is available to every Communication Controller (CC) in the network.

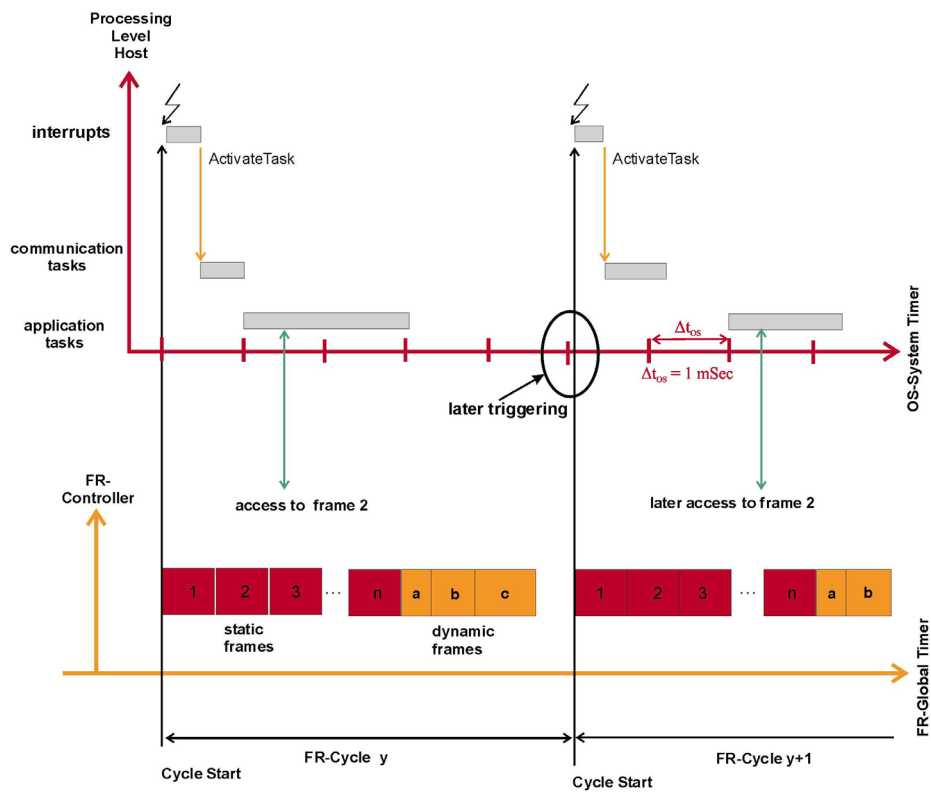
In a distributed control system a function is distributed among multiple ECUs in a network. The dead time caused by the signal propagation time from the sending application to the receiving application can have a decisive influence on the quality of control. In principle, a small dead time has a beneficial effect.

In event-triggered communication systems the exchange of data between CC and Host is essentially interrupt-driven, and accessing to the bus may result under some conditions in wait times. The signal propagation time cannot be predicted precisely; therefore a worst case estimate is typically made. Not until a network-global time of the FlexRay protocol is made available can the operating system offer services for synchronization to this time. Via the TDMA (Time Division Multiple Access) method all participating ECUs are aware of the precise point in time each message is sent or received in the static area of the FlexRay cycle. These two properties minimize imprecision with regard to the signal propagation time of a FlexRay-based application.

For synchronizing the exchange of data in a FlexRay-based application various methods of resolution are possible, depending on the application needs and the availability of the operating system properties. In principle, communication tasks should be triggered by the CC's FlexRay timer. Activation of application tasks, on the other hand,

could be triggered by any desired timers. Four solution approaches are explained in the following:

Solution A - Alarms attached to the OS System Timer trigger the application tasks.



[Figure 4: Solution A - Data exchange between FlexRay hardware and an application task with OSEK-OS]

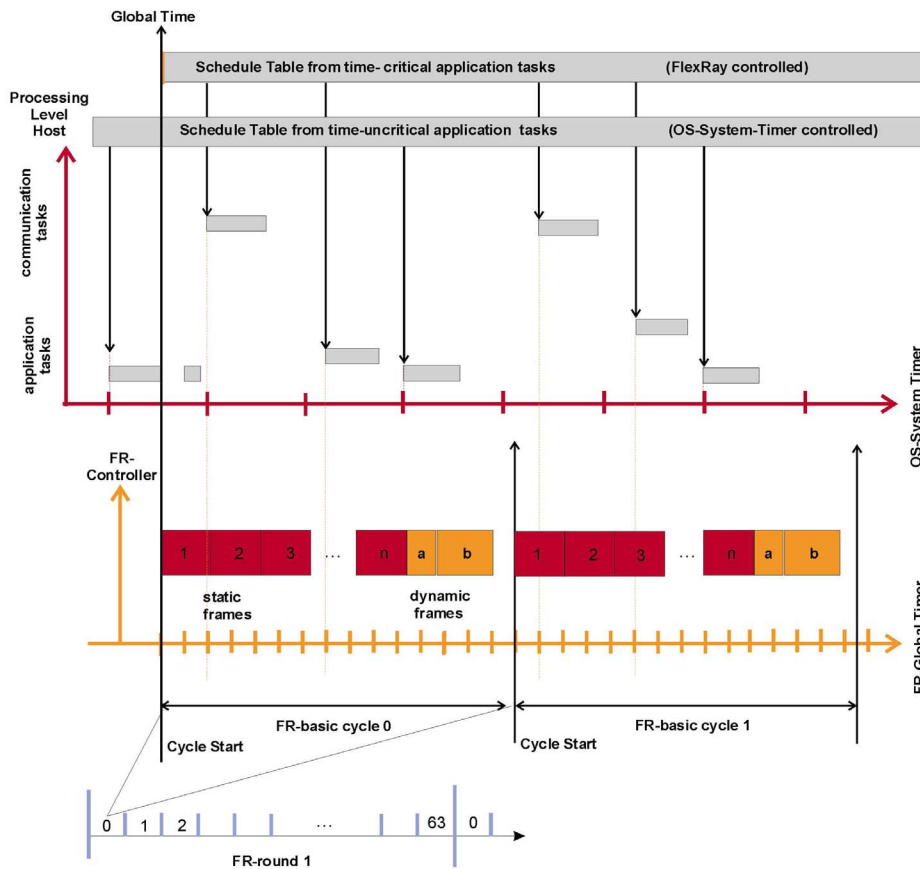
At the beginning of each base cycle the "FlexRay Cycle Start Interrupt" is triggered by the FlexRay controller. The communication task is activated immediately. Independently, the operating system timer initiates the sequence of application tasks. Figure 4 shows a simplified form of this solution approach with just one application and one communication task. It should be noted that later triggering of a Cycle Start Interrupt, e.g. caused by drift between the CC oscillator and the Host oscillator, results

in delayed processing of the transported FlexRay data (e.g. from frame 2) of max. $\Delta t_{OS} = 1$ ms. If the application does not expect a quicker reaction time, an OSEK-OS is sufficient.

Solution B - If the control system requires a shorter response time (less than 1 ms), triggering of the application task requires a High Resolution Timer (resolution approx. 10 microseconds) with the OSEK-OS. A suitable timer is then needed.

Solution C - If the ECU does not have a High Resolution Timer, the FlexRay timer can be additionally used to activate time-critical application tasks. FlexRay-independent tasks can continue to be attached to the OS System Timer.

In startup behavior it should be noted that the FlexRay timer, and consequently also the tasks that it activates, are unavailable until the FlexRay controller has synchronized to the network for the first time. If synchronization is lost afterwards, the FlexRay timer can continue to operate based on its local time, provided that the FlexRay controller was configured for this. An AUTOSAR OS of Scalability Class 1 with schedule tables is sufficient for this purpose. This solution enables a control system distributed over multiple ECUs, since the FlexRay global time assures synchronization between all ECUs. This solution approach is depicted in Figure 5.



[Figure 5: Solution C - Data exchange between FlexRay hardware and an application task with AUTOSAR SC1 OS]

Solution D - If in the above cases it is also necessary to monitor the execution time of tasks and interrupts, an AUTOSAR OS SC2 or SC4 is required. This prevents excessive execution times and achieves time deterministic behavior.

Summary

A FlexRay-based application does not absolutely require a time-triggered operating system. The choice of a suitable operating system should be made individually for each ECU under consideration of the application and architecture. For this purpose an analysis pertaining to the synchronism among ECUs, safety requirements, response time and time monitoring is necessary. Vector Informatik offers the

developer the optimal operating system for all FlexRay applications: osCAN certified to the OSEK/VDX standard with or without High Resolution Timer or osCAN AUTOSAR that covers Scalability Classes SC1-SC4 according to AUTOSAR OS V2.0. The Vector FlexRay software components will operate together with any of these OS variants. The osCAN TimingAnalyzer analyzes the schedulability of tasks from a worst-case perspective.

Vector supports the user with software components and individualized service for universal development of FlexRay systems up to series production. Development is simplified by mature tools that are tuned to one another, like for instance DaVinci Network Designer for all FlexRay-typical design tasks or CANoe.FlexRay 6.0 for simulation and stimulation of a network, integration tests and rest-of-bus simulation as well as analysis of the finished FlexRay network. CANape 6.0 is used to access to all internal parameters of the FlexRay ECU via the standardized measurement and XCP-on-FlexRay calibration protocol. The FlexRay Evaluation Bundle provides for quick and flexible implementation of a FlexRay network. This integrated environment of software components and tools also includes a sample application for a FlexRay system with two nodes.

Revised: 11/2006

Word count: 2,227

Character count: 14,627

Table 1: FlexRay-relevant properties of AUTOSAR OS

Figure 1: Case A - Without error handling

Figure 2: Case B - With error handling caused by excessively long execution time of the task

Figure 3: Case C - With error handling caused by excessively long execution time of an interrupt

Figure 4: Solution A - Data exchange between FlexRay and an application task with OSEK-OS

Figure 5: Solution C - Data exchange between FlexRay and an application task with AUTOSAR SC1 OS

All figures: Vector Informatik GmbH

Literature references:

- [1] OSEK/VDX Operating System, Version 2.2.2, www.osek-vdx.org
- [2] AUTOSAR - Specification of Module Operating System V2.0, www.autosar.org
- [3] OSEKtime - OSEK/VDX Time-Triggered Operating System, Version 1.0, www.osek-vdx.org

Authors:



Pascale Morizur (Graduate Engineer) studied Physics/Electronics at the Grande Ecole ICPI in Lyon (France). After graduating in 1986 she worked for 10 years at MAN commercial vehicles in advanced development for CAN, J1939 and diagnostics. She is employed at Vector as a product management engineer in the area of FlexRay embedded software components.
Tel. +49-711/80670-2211, Fax +49-711/80670-111,
E-mail: pascale.morizur@vector-informatik.de



Dirk Grossmann (Graduate Engineer) studied Electrical Engineering at the University of Stuttgart. After graduating in 1997 he worked first in the development of operating systems at ETAS GmbH before assuming responsibility for operating systems as North American product manager for 2 years. Since 2003 he has been working at Vector as a team leader responsible for the development of FlexRay embedded software components.
Tel. +49-711/80670-223, Fax +49-711/80670-111,
E-mail: dirk.grossmann@vector-informatik.de



Winfried Janz (Graduate Engineer) studied Electrical Engineering at the University of Stuttgart. After working for 4 years developing software for embedded controllers in control and automation engineering he came to Vector in 1995. Since 1997 he has been team leader and product manager responsible for the development of OSEK real-time operating systems. Through his work in OSEK and AUTOSAR working committees he helped to give shape to the Operating System and Configuration specifications.
Tel. +49-711/80670-367, Fax +49-711/80670-111,
E-mail: winfried.janz@vector-informatik.de

Vector Informatik GmbH
Ingersheimer Str. 24
D-70499 Stuttgart
Germany
www.vector-informatik.com

Editorial contact person: Holger Heit
Tel. +49-711/80670-567, Fax +49-711/80670-555,
E-mail: holger.heit@vector-informatik.de