Getting More Out of Your FlexRay Controller – Indispensable Optimizations for Real-World Series Applications

Thomas M. Galla & Markus Eggenbauer
Elektrobit Austria GmbH
Kaiserstrasse 45 / Stiege 2, A-1070 Vienna, Austria
Email: thomas.galla@elektrobit.com
Email: markus.eggenbauer@elektrobit.com

Abstract

With the first FlexRay-based series application on the road, FlexRay has completed its transition from prototyping stage technology to a technology that has proven to be suitable for deployment in real-world automotive series applications. Due to the high bandwidth compared to CAN one of the main use-cases for the deployment of FlexRay at the time being is the use of FlexRay as a replacement for multiple CAN networks to reduce the costs for the wiring harness. Together with the AUTOSAR idea to integrate multiple logically independent application functions into the same electronic control unit (ECU) this results in a very demanding setup with respect to resource requirements like RAM, ROM, computation time, and last but not least available hardware buffers or message RAM of the FlexRay communication controller.

The fact that the number of available FlexRay hardware buffers or the amount of message RAM is rather limited compared to the other resources (e.g., ROM, computation time) on today's micro controllers makes the hardware buffers of a FlexRay communication controller a scarce resource. Real-world application existing today quickly attain a complexity where this scarce resource does no longer suffice introducing the need to apply clever optimization strategies to overcome this resource bottleneck.

In this paper we motivate the problem of FlexRay hardware buffers as a scarce resource for any real-world FlexRay application. We point out the optimization problem that needs to be addressed, we provide strategies and means to tackle this optimization problem in a structured way, and we show how these strategies can be deployed in an AUTOSAR environment without jeopardizing the conformance to the relevant AUTOSAR specifications.

We demonstrate the feasibility of this approach by laying out how these strategies are incorporated in the EB tresos product family, namely in the EB tresos Autocore FlexRay driver and interface basic software and in the EB tresos Studio FlexRay wizard as a tooling counterpart. Finally we evaluate the presented approach by applying the different optimizations steps to an existing real-world automotive ECU that is currently deployed in a premium class vehicle.

1 Motivation

At the time being FlexRay is deployed as a high-bandwidth replacement for multiple CAN networks to reduce the costs for the wiring harness. Together with the AUTOSAR idea to integrate multiple logically independent application functions into the same electronic control unit (ECU) this use case imposes significant resource demands on the FlexRay hardware. When these demands are examined in detail it becomes obvious that the available hardware buffer or the available message RAM of the FlexRay communication controller is the limiting resource that needs to be optimized for.

The different flavors of FlexRay communication controllers available today range from low-end controllers with less than 32 hardware buffers to high-end controllers with up to 128 hardware buffers. Due to cost reasons however low- and medium-end controllers are used in today's FlexRay ECUs increasing the need to cope with a rather limited amount of hardware buffers even more. In addition to the buffer requirements of the application itself special communication protocols used during development and integration time to facilitate the in-system debugging, re-programming, and the calibration of the ECU introduce additional demands for dedicated hardware buffers thus limiting the number of available buffers even further.

In the following sections of this paper we point out the optimization problem that needs to be addressed, we provide strategies and means to cope this optimization problem in a structured way, we show how these strategies can be deployed in an AUTOSAR environment without losing conformance to the relevant AUTOSAR specifications, we demonstrate the feasibility of our approach by laying out how these strategies are incorporated in the EB tresos product family, namely in the EB tresos Autocore FlexRay driver and interface basic software and in the EB tresos Studio FlexRay wizard as a tooling counterpart, and finally we evaluate the presented approach by applying the different optimizations steps to an existing real-world automotive ECU that is currently deployed in a premium class vehicle.

2 Optimization Concepts

The optimization concepts presented in this paper can be distinguished into two different categories. On the one hand the reuse or sharing of FlexRay hardware buffers is applicable as a general concept. On the other hand additional specific optimizations for higher-level communication protocols are possible. Both categories will now be discussed in further detail.

2.1 Buffer Reuse/Sharing as General Optimization Concepts

According to the FIBEX specification [8] the temporal aspects of a FlexRay frame transmitted on the FlexRay network is modeled by a so-called *frame triggering*. The frame triggerings for FlexRay are described by a 2-tuple consisting of the ID of the TDMA slot in the static or the dynamic segment of a FlexRay communication cycle and a so-called *cycle set (CS)* written as $\langle ID, CS \rangle$. The cycle set itself is a 2-tuple consisting of the first communication cycle where the frame is transmitted (the so-called *base cycle (BC)*) and the number of communication cycles between two consecutive transmissions of the frame (the so-called *cycle repetition (CR)*) written as $\langle BC, CR \rangle$.

A naive but commonly widely applied allocation scheme for a FlexRay communication controller's hardware buffers would use a 1:1 mapping between frame triggerings and hardware buffers. This allocation scheme however is highly suboptimal with respect to hardware buffer consumption.

2.1.1 Buffer Sharing without Reconfiguration

One optimization that can be applied here to reduce the buffer consumption is to merge multiple frame triggerings into a single one and allocate a single hardware buffer for all of them. The requirement for this optimization is that the frame triggerings to be merged have to be *compatible*. This means that the frame triggerings have to use the same slot ID and that the cycle sets of these frame triggerings are compatible, i.e., that the combination of all the cycle sets can again be described as a new cycle set. The cycle sets $\langle 0, 4 \rangle$, $\langle 2, 4 \rangle$, and $\langle 1, 2 \rangle$ for example are compatible since they can be merged into the cycle set $\langle 0, 1 \rangle$, while the cycle sets $\langle 0, 4 \rangle$, $\langle 2, 4 \rangle$, and $\langle 1, 4 \rangle$ are not compatible, since the cycle set $\langle 3, 4 \rangle$ is missing to be able to denote the combination as a single cycle set.

When allocating multiple compatible frame triggerings (i.e., frame triggerings with equal slot IDs and compatible cycles sets) to a single hardware buffer the scheduling of the FlexRay interface's communication operations however has to be done in a way that the data is provided/retrieved at the correct point in time in order to prevent that a transmission uses the data intended for a previous frame triggering and to prevent a reception from overwriting the data of a previous frame triggering with the data of the next one.

Figure 1 depicts an example with two compatible TX frame triggerings (A, B). During the time interval I_A^{Tx}, the data for the transmission of frame triggering A has to be written to the shared hardware buffer

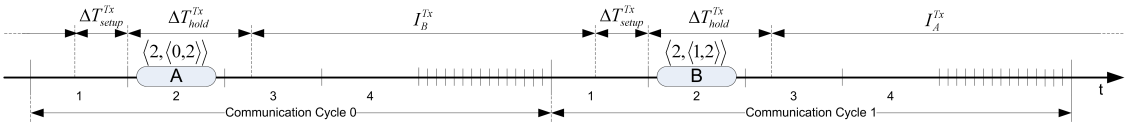

Figure 1: Buffer Sharing without Reconfiguration – Transmission

of the FlexRay communication controller by the respective function of the FlexRay driver. I_B^{Tx} denotes the corresponding interval for the transmission of frame triggering B. Hereby transmission setup time (ΔT^{Tx}_{setup}) denotes the time difference between the point in time the FlexRay communication controller starts reading the first data byte of the frame triggering to be transmitted from the hardware buffer till the start of the TDMA slot, where the transmission shall take place. The transmission hold time (ΔT^{Tx}_{hold}) denotes the time difference between the start of the TDMA slot, where the transmission shall take place, till the point in time when the FlexRay communication controller finished accessing the respective hardware buffer for the transmission. Note that ΔT^{Tx}_{setup} and ΔT^{Tx}_{hold} are heavily hardware specific and thus vary from FlexRay communication controller implementation to FlexRay communication controller implementation.

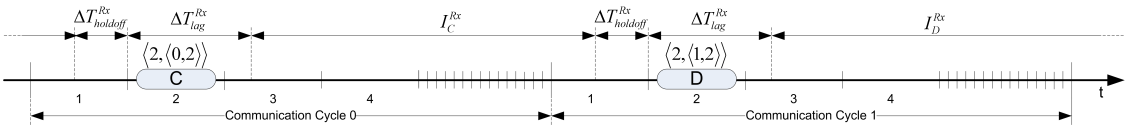

Figure 2: Buffer Sharing without Reconfiguration – Reception

Figure 2 shows a similar scenario for two compatible RX frame triggerings (C, D). During the time interval I_C^{Rx}, the data obtained through the reception of frame triggering C has to be retrieved from the shared hardware buffer of the FlexRay communication controller by the respective function of the FlexRay driver. I_D^{Tx} denotes the corresponding interval for the reception of the data of frame triggering D. Hereby reception holdoff time ($\Delta T^{Rx}_{holdoff}$) denotes the time difference between the point in time the FlexRay communication controller starts modifying the contents of the the hardware buffer till the start of the TDMA slot, where the reception shall take place. – Usually $\Delta T^{Rx}_{holdoff}$ is zero. The reception lag time (ΔT^{Rx}_{lag}) denotes the time difference between the start of the TDMA slot, where the reception shall take place, till the point in time when the FlexRay communication controller finished writing data to the respective hardware buffer due to the reception. Just like ΔT^{Tx}_{setup} and ΔT^{Tx}_{hold} the parameters $\Delta T^{Rx}_{holdoff}$ and ΔT^{Rx}_{lag} are heavily hardware specific and thus vary from FlexRay communication controller implementation to FlexRay communication controller implementation as well.

2.1.2 Buffer Sharing with Reconfiguration

In case the merging of multiple frame tiggerings as described in Section 2.1.1 cannot be applied, since the involved frame triggerings are incompatible (i.e., exhibit either different slot IDs or incompatible cycle sets), an enhanced strategy has to be applied. Here a reconfiguration of the FlexRay communication controller's hardware buffer has to take place during run-time in order to use the buffer for the transmission/reception of the new frame triggering. Similar to the use-cases described in Section 2.1.1, this reconfiguration action has to take place within a restricted time interval.

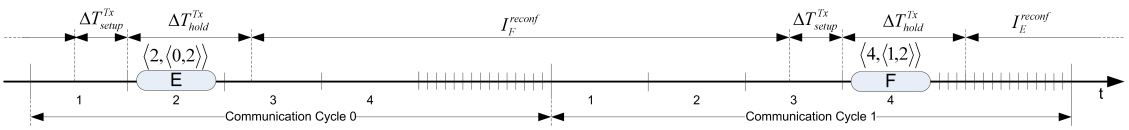

Figure 3: Buffer Sharing with Reconfiguration – Transmission

Figure 3 depicts an example with two incompatible TX frame triggerings (E, F). During the time interval I_E^{reconf}, the FlexRay communication controller's hardware buffer has to be configured properly for

the transmission of frame triggering E by calling the respective function of the FlexRay driver. I_F^{reconf} denotes the corresponding interval for the reconfiguration of the buffer for frame triggering F. Hereby transmission setup time (ΔT_{setup}^{Tx}) and the transmission hold time (ΔT_{hold}^{Tx}) have the same semantics as described in Section 2.1.1.

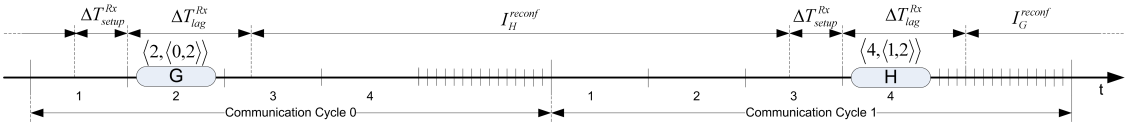

Figure 4: Buffer Sharing with Reconfiguration – Reception

Figure 4 depicts an example with two incompatible RX frame triggerings (G, H). During the time interval I_G^{reconf}, the FlexRay communication controller's hardware buffer has to be configured properly for the reception of frame triggering G by calling the respective function of the FlexRay driver. I_H^{reconf} denotes the corresponding interval for the reconfiguration of the buffer for frame triggering H. Hereby the reception lag time (ΔT_{lag}^{Rx}) has the very same semantics as described in Section 2.1.1. The reception setup time (ΔT_{setup}^{Rx}) denotes the time difference between the point in time when the FlexRay communication controller starts its preparations for the reception of the respective frame triggering, till the start of the TDMA slot, where the reception shall take place. Again ΔT_{lag}^{Rx} is very hardware dependent.

2.2 Protocol Specific Optimization Concepts

Additional buffer optimizations are possible when focusing on higher-level communication protocols like network management (NM) [6], transport protocols (TP) [5], or calibration protocols like XCP [7]. These higher-level communication protocols encode protocol information like sender information and payload semantics in a dedicated protocol control information (PCI) field that is part of the payload from FlexRay's point of view. Thus this PCI field must be provided by the sending instance of the protocol layer and must be examined by the receiving instance of the protocol layer. Therefore there is no need to use different hardware buffers or to use temporal multiplexing (e.g., sending of data with different semantics in different communication cycles) to implicitly encode this information. Depending on the relationship between transmission period of the frames used for the specific higher-level protocol and the execution period of the software instance of the higher-level protocol state machine, a single buffer (in case both periods a equal – NM use case) or a hardware FIFO[1] (in case the transmission consists of short bursts with a period smaller than the execution period – TP use case) can be used. The use of a hardware FIFO hereby reduces the the processing load on the ECU.

2.3 Compliance to AUTOSAR

It is important to point out that especially the concepts addressed in Section 2.1 can be implemented with the mechanisms currently defined by AUTOSAR [3, 9]. The payload data can be written to a FlexRay controller's hardware buffer by using the AUTOSAR FlexRay driver API `Fr_TransmitTxLPdu()`. Received data can be retrieved from the reception hardware buffer of the FlexRay communication controller by calling AUTOSAR FlexRay driver API `Fr_ReceiveRxLPdu()`. Last but not least a hardware buffer can be reconfigured via the API call `Fr_PrepareLPdu()`

The concepts addressed in Section 2.2 require minor vendor specific enhancements to AUTOSAR at the time being though.

3 Implementation and Evaluation

According to AUTOSAR the ECU configuration is performed in two steps. For each module

[1] Such a hardware FIFO is provided by many of the currently available FlexRay communication controllers.

1. a standardized *AUTOSAR ECU configuration* in XML [10] format [2, 4] is created

2. an implementation specific *module configuration generator (MCG)* is run that transforms this AUTOSAR ECU configuration into implementation specific configuration ISO C 90 [1] source file

3.1 Providing the Basis for Optimizations – The FlexRay Interface Wizard

The ECU configuration of the FlexRay driven and the FlexRay interface module contains the complete FlexRay communication matrix related to a single ECU. For FlexRay the configuration is highly complex. Beside the assignment of frames to frame triggerings the configuration contains the so called FlexRay interface *job list*. This job list describes the temporal interaction of FlexRay driver software and FlexRay controller hardware. It contains pre-scheduled points in (global FlexRay) time at which specific operations (*communication operations*) are performed on the FlexRay controller's hardware buffer. Those communication operations contain:

Transmit: write payload in a FlexRay controller's hardware buffer

Receive: read payload from a FlexRay controller's hardware buffer

Prepare: reconfigure the hardware buffer to frame triggering assignment

Those communication operations shall be properly placed in order to minimize communication delays and enable hardware buffer optimization potential. According to Figure 1 the transmit communication operation for transmission of frame triggering B must be located in interval I_B^{Tx}. For reception the receive communication operation for frame triggering C must be located in interval I_C^{Rx}, see Figure 2. If hardware buffer reconfiguration potential shall be used, additional prepare communication operations must be introduced. Those are located, e.g., in Intervals $I_{[F-H]}^{reconf}$, see Figure 3 and Figure 4. The FlexRay interface's configuration, however, only statically schedules the communication operations, whether there is need or room for optimization is not the FlexRay interface's decision.

Creating this job list requires a lot of expertise and patience, if made manually. Even for simple ECUs it is required to setup approximately 128 sub containers in the FlexRay interface's ECU configuration, each containing multiple configuration parameters. In order to optimize this subtle task EB tresos Studio is equipped with a FlexRay interface wizard that allows to automatically generate this job list and provides a domain specific user interface to manually edit the job list in an efficient way, see Figure 5.

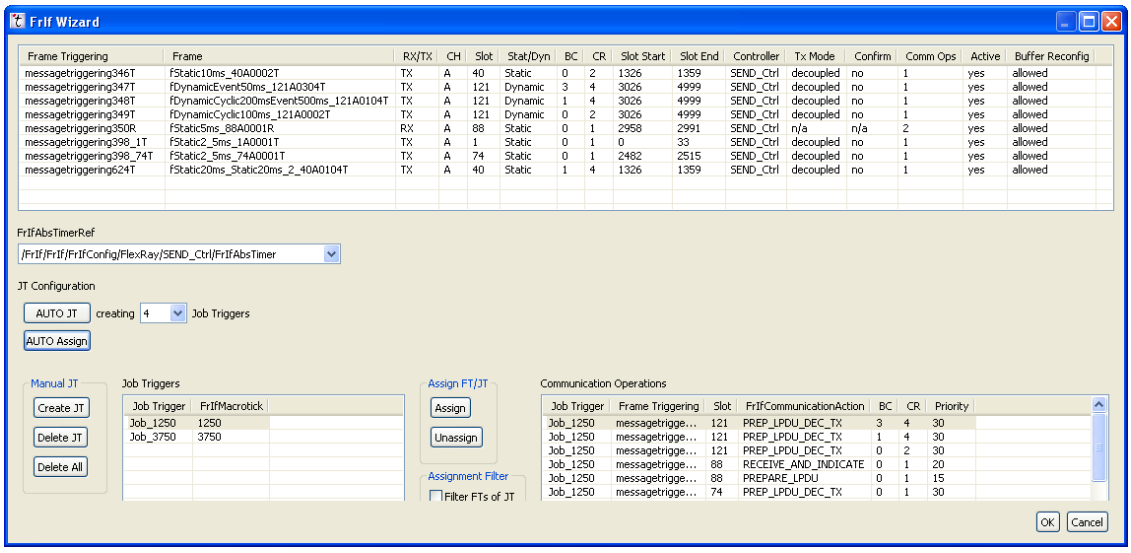

Figure 5: FlexRay Interface Wizard – Screenshot

The FlexRay interface wizard provides advanced options that allow to

- activate buffer reconfiguration on a per-frame-triggering basis (column `Buffer Reconfig` set to `allowed` in Figure 5)
- completely enable/disable frame triggerings for debugging purposes (column `Active` set to `yes` in Figure 5)

Even a hybrid approach, automatically creation of the job list (buttons `AUTO JT` and `AUTO Assign` in Figure 5)) and afterwards a manual fine tuning of the generated job list, is supported. With this FlexRay Interface wizard the AUTOSAR ECU configuration for FlexRay turns into an easy job.

3.2 Performing the Hardware Specific Optimizations – The FlexRay Driver Generator

The MCG's job is to read the AUTOSAR ECU configuration and generate an implementation specific configuration that controls the behavior of the basic software module. In case of the FlexRay driver this configuration contains all the FlexRay communication parameters as well as the FlexRay controller's hardware buffer assignment to frame triggerings. As hardware buffers are a scarce resource the hardware buffer assignment is a critical process. In order to achieve the optimizations described in Section 2.1.1 and Section 2.1.2 the MCG analyses the complete job list to find frame triggerings that can share a single hardware buffer. For all possible optimizations the MCG considers the device specific hardware buffer setup- and hold-times.

Optimization is performed only in case the MCG runs out of hardware resources. The optimizer's priority policy ensures that non-runtime consuming optimizations (buffer sharing without reconfiguration) are preferred over optimizations introducing an additional runtime overhead (buffer sharing with reconfiguration). Additionally the MCG prefers optimizations which have more relaxed temporal constraints (i.e., optimizations where I_*^{reconf}, I_*^{Tx}, or I_*^{Rx} is smaller) over optimizations that have tighter temporal constraints (i.e., optimizations where I_*^{reconf}, I_*^{Tx}, or I_*^{Rx} is larger).

Finally a report that exactly lists the performed optimizations is generated thus giving the human user the possibility to precisely examine the optimization operations performed and to verify their correctness.

3.3 Evaluation Based on a Real-World ECU

Table 1 illustrates the optimization impacts on a real-world FlexRay ECU with 201 frame triggerings currently deployed in a premium line vehicle.

Optimization	# Used HW Buffers	# Saved HW Buffers
No optimization	201 buffers	0 buffers
Buffer Sharing without Reconfiguration	173 buffers	28 buffers
Buffer Sharing with Reconfiguration	163 buffers	10 buffers
Protocol Specific Optimizations (NM, TP, XCP, ...)	down to 114 buffers	up to 49 buffers

Table 1: Optimized Buffer Consumption Example

4 Conclusion

In this paper we proposed several optimization strategies for the allocation of hardware buffers of the FlexRay communication controller with the aim to reduce the overall buffer consumption for the application running on a given ECU. Afterwards we showed how these optimizations can be integrated into an AUTOSAR environment. Finally we evaluated the effectiveness and the impact of these optimizations using a real-world FlexRay ECU deployed in a premium line vehicle as an example. This evaluation showed that by applying the presented approaches, a reduction of the buffer consumption of approximately 25% is achieveable.

We deem that these optimizations are essential when developing real-world applications in a cost driven market since without these optimizations the applications will run out of hardware buffers inducing the need for high-end FlexRay communication controllers with additional hardware buffer causing additional hardware costs per ECU as a consequence.

All of the presented optimization are part of the EB tresos product family and are thus readily available for a large number of todays FlexRay-equipped micro controllers.

References

[1] Information Technology – Programming Language C. Technical Report ISO/IEC 9899:1990, ISO (International Organization for Standardization), 1, rue de Varembe, Case postale 56, CH-1211 Geneva 20, Switzerland, 1990.

[2] AUTOSAR Consortium. AUTOSAR – ECU Configuration Parameter Definition. Technical Report Version 2.2.0, Release 3.1, Rev 0002, AUTOSAR Consortium, 2008.

[3] AUTOSAR Consortium. AUTOSAR – Layered Software Architecture. Technical Report Version 2.2.1, Release 3.0, Rev 0001, AUTOSAR Consortium, February 2008.

[4] AUTOSAR Consortium. AUTOSAR – Specification of ECU Configuration. Technical Report Version 2.1.0, Release 3.1, Rev 0002, AUTOSAR Consortium, 2008.

[5] AUTOSAR Consortium. AUTOSAR – Specification of FlexRay Transport Layer. Technical Report Version 2.2.2, Release 3.1, Rev 0001, AUTOSAR Consortium, August 2008.

[6] AUTOSAR Consortium. AUTOSAR – Specification of FlexRay Network Management. Technical Report Version 3.0.4, Release 3.1, Rev 0002, AUTOSAR Consortium, February 2009.

[7] Association for Standardization of Automation and Measuring Systems. XCP – The Universal Measurement and Calibration Protocol Family; Part 3 XCP on FlexRay - Transport Layer Specification. Technical Report Version 1.0, Association for Standardization of Automation and Measuring Systems, December 2005.

[8] Association for Standardization of Automation and Measuring Systems. ASAM MCD-2 NET: Data Model for ECU Network Systems (Field Bus Data Exchange Format). Technical Report Version 3.1.0, Association for Standardization of Automation and Measuring Systems, January 2009.

[9] H. Heinecke. AUTomotive Open System ARchitecture An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In *Proceedings of the Convergence International Congress & Exposition On Transportation Electronics*, Detroit, MI, USA, 2004.

[10] World Wide Web Consortium (W3C). Extensible Markup Language (XML). Technical Report Version 1.1 (Second Edition), World Wide Web Consortium (W3C), 2006.