



# Guided FlexRay Design with System

**FlexRay is on the verge of entering series cars. However, the complexity of this new communication technology brings along new challenges, especially for developers and their design tools. How a new software generation copes with these challenges and delivers systematically substantial support, is the topic of the following article.**

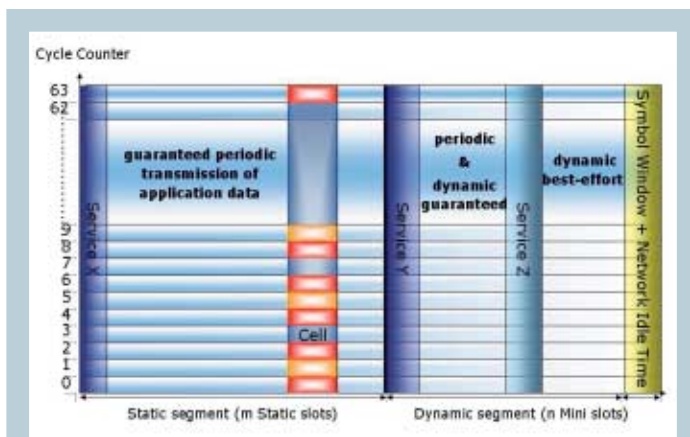
The most important automotive electronics applications, such as ABS, ESP, etc., depend on CAN, an event-triggered communication system. The imminent introduction of FlexRay series production leads to a change of paradigms. From now on, development engineers have to deal with a deterministic, time-triggered system. This new paradigm needs e.g., a schedule, which is like a timetable telling what information is sent resp. received via the bus at what time. In setting up such a schedule, a development engineer has to configure a so called configuration space – 2048 time slots in 64 cycles each – (Figure 1) in a technically correct way. Although this scheduling task is fundamental to the success of a FlexRay project, it is not at the center of the developers' attention and efforts – the focus is totally on the design of the application. FlexRay is only one important prerequisite, nothing more.

A design tool has to support and guide application developers – on the OEM side and the supplier side – to easily handle this huge amount of FlexRay configuration data. One approach how to deal with this challenging task will be discussed using following example: For the first time, developers of a supplier had to introduce FlexRay into a series

car. We will accompany them step by step through the design process of an ECU1 (ECU workflow). The used software: DECOMSYS::DESIGNER PRO.

### Easy design through guided workflow

For the development of the ECU in the series project, our engineer needs global configuration data concerning the



**Figure 1: FlexRay's huge configuration space - 2048 x 64 (=217 resp. ~130.000) cells - mandates careful planning from the beginning.**



Figure 2: The operation windows (on the left side of the screenshot) acts as a guide through the design process.

© automotive

FlexRay network from the OEM. This data comes in a FIBEX file, providing all the information on hardware architecture, i.e., ECUs with controller and their connections to the channels of the FlexRay network. Included is furthermore the communication schedule such as signals, frames, frame-cell assignment etc. (Sometimes

defines applications tasks as well as system tasks and assigns these tasks to the MCU. Now, the developer takes care of the "Schedule" defined by the OEM. Which signals are sent in which frames at what time? Since any change of this schedule has a global effect for any ECU on the entire network, this data should not be edited.

Following the Operation Window, the developer now enters the driver configuration – and a new challenging task: The detailed configuration of all required services and communication layers has to be performed in order to generate configuration code. These services and layers are contained in separate plug-ins and can thus be easily extended according to customer demands. Services and layers may include:

**COMMSTACK2:** The developer uses the COMMSTACK plug-in in order to assign — automatically or manually — frames to buffers of the communication controller, as you can see in detail down below.

**FLEXCOM2:** To allow access to the signals of the frames, so-called communication tasks are specified which pack and unpack signals into/from FlexRay frames. The exact timing, i.e., period and offset, of these tasks and the handled frames have to be defined.

**Transport Protocol (TP):** The TP configuration is easy for the supplier.



Figure 3: The Tabs provide a good overview.

© automotive

functions e.g., application tasks, can be part of the FIBEX file as well.) The MCUs (Micro Controller Units) within the ECU are not included.

The developer imports this data and adds the configuration data for his ECU step by step. The first step, the ECU Hardware Refinement, covers the service range of the ECU. This includes the specification of the MCU(s) within the ECU, the definition of specific parameters of the operations system, and the selection of the FlexRay Controller, e.g. MFR 4200 or E-Ray.

In the next step, the so called ECU Software Refinement, the engineer



Qualität ist zeitlos!



Tools and engineering for modern times



www.canway.de

ID	Type	Slot	CC	Pool	#Frame Triggering
000	BCP	0	0	1	
001	BCP	0	0	1	
002	BCP	0	0	1	

Figure 4: All the Buffer Information at a glance - a vital help for the configuration work

© automotive

The OEM usually configures the TP; the information is passed on to our engineer via the FIBEX file. No further change is required to generate configuration code.

**Network Management (NM):** The NM configuration is also rather simple for the supplier. The configuration of the NM is contained in the FIBEX file as well. The engineer only adds the Node ID (Identifier) and the Message Timeout.

**OS Config:** To use a time-driven operating system, so called OIL (OSEK Implementation Language) entries have to be generated covering application tasks, system tasks, and communication tasks.

**Quick grouping, fast sorting, great overview**

This grid-view not only enables the user to perform various selections according to ECU, MCU, and CC. It also offers a multiple sort function – and thus clear advantages: It only takes a few mouse clicks for a compact view of the numerous configuration data and the input or change of data is also very simple.

The configuration of the single driver layers is done by means of so called “tabs”: TP, NM, FLEXCOM, OSCONFIG, and COMMSTACK. These tabs consist of sub-tabs with more specific configuration information. This concept gives the developer easy and quick access to all configuration data. In addition, the tab/sub-tab concept again acts as a guiding system supporting the developing engineer in the tedious task of configuring the single layers. Working the tabs from the left to the right, all design steps are completed. Of course, it is also possible to go directly to a specific tab and to configure the data right away. In order to keep this article as concise as possible, only one driver layer is exemplified in detail.

**Example: COMMSTACK Configuration**

For the configuration of this layer, the developer uses the plug-in “COMMSTACK”. Here the tab/sub-tab concept provides one tab with four sub-tabs (Figure 3). The configuration work starts with the sub-tab on the very left, the so called CC Registers. The developer can

easily access all register values of the Communication Controller (CC). If necessary, the values can be changed. The next two tabs take care of the buffer assignment (manual or automatic). Here, the Buffer Information Grid plays a vital part. This grid facilitates not only the access to e.g., the Buffer Index (Idx) of the corresponding CC or to the Buffer Type. It also enables the user to get a quick overview by means of multiple sorting, or to change the values – a simple double-click on the selected column is all that it takes in order to edit a value.

Although it is possible to assign buffers automatically (Auto BA), the developer of our series project wants to do this job manually. Frame triggerings can be selected and assigned to single buffers or to buffer pools. The result of this operation is immediately visible and thus, possible problems can be realized and solved immediately. And again, the multiple sorting feature comes in handy, e.g. sorting according to slots, buffer types, pool names etc.

The last step of the driver configuration is the code generation, which can be done either layer for layer separately or for all layers at once. The developer links the generated configuration data (in .c and .h-files) with his application code. The application code itself can be generated manually or automatically based on models, e.g. in MATLAB/Simulink. This step concludes the design of the system.

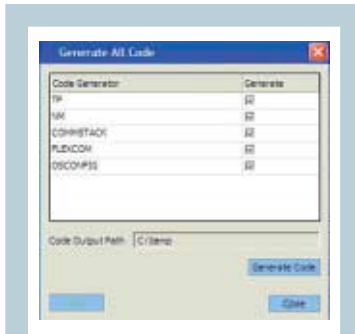


Figure 5: It is possible to generate code for all driver layers at one scratch.

© automotive

**Special feature: CSM – Controller Support Module**

Many semiconductor companies are about to offer FlexRay in their portfolio.

Therefore, it is vital for the user that the design tool supports the relevant controllers. This challenge is solved by the CSM concept: The user integrates the needed Controller Support Module (CSM) as a plug-in into the design tool with all the technical information. CSMs also take care of the important step of mapping the parameters of the FlexRay specification to the parameter of the controller.

**Conclusion**

This article reveals the workflow in a FlexRay series project. For the first time, the development steps of an ECU (ECU workflow) are described using the DECOMSYS::DESIGNER PRO. Hereby, the huge amount of configuration data is displayed in both a compact and concise manner. Smart sorting features enable the developer to find and group the required information fast and easy. Guidance support through all development steps is given. Furthermore, the plug-in concept ensures that the required controller support modules needed CSMs and additional layers and service extensions will also be available in the future.

The DECOMSYS::DESIGNER PRO was released in October 2005 and later-on offer a MATLAB/Simulink extension.

Nicht zur Verwendung in Intranet- und Internet-Angeboten sowie elektronischen Verteilern

© 2005 Carl Hanser Verlag, München www.hanser-automotive.de



**Dr. Roman Pallierer**  
is Manager of the Tools Development Team at DECOMSYS.



**Dr. Arnold Zimmermann**  
is responsible for Marketing Communications.