Fibex gateway configuration tool chain

Tobias Lorenz[1,2], Jan Taube[1,3], Markus Ihle[1], Otto Manck[2], Helmut Beikirch[3]

[1]Robert Bosch GmbH Reutlingen, [2]Technical University of Berlin, [3]University of Rostock

Usually gateway configurations have been based on non-standardized description formats. Unfortunately data exchange between different formats is inherently error prone and time consuming.

A common description format was necessary and is found in the Field Bus Exchange Format. FIBEX is an XML-based file format, the upcoming standard for network configurations, which combines information about every aspect of a complete in-car network including controllers, channels, frames and signals. FIBEX is also the first standard to describe gateway configurations.

Gateway implementations have proprietary internal formats in which they store their configuration data. Some routing information are stored as linked lists, while others are stored as executable code.

An optimal gateway tool chain should be based on the FIBEX configuration data of all connected networks and translate the routing information directly into the internal format of the target implementation. The development of the tool chain at Bosch followed the development of a hardware-accelerated multi-protocol gateway. Currently the gateway connects FlexRay with CAN networks. Support for other protocols, like LIN and MOST, are planned.

This paper describes the gateway hardware, gateway related enhancements to the FIBEX standard and the implementation of the tool chain to generate configuration images for the developed gateway.

1 Introduction

The need for data transparency and information exchange within the overall in-car network has increased with the continuous improvement of electronic systems. Just one of the numerous examples is the electronic stability program (ESP), which monitors the drive dynamics of the vehicle and takes control over engine management and brake systems, if the vehicle is in danger of tipping over or skidding. A low latency gateway system is needed to connect the networks of both systems.

Nowadays most gateways are software gateways, based on standard communication controllers and an high performance CPU running appropriate software. This is a flexible concept, but on the other hand the performance is affected, since the hardware structure of a usual micro-controller is not optimized for gateway operations and the ECU often has to do other tasks as well. As the number of interfaces and the total bandwidth of a gateway increase continuously, software gateways will soon become bottlenecks, too slow to handle all incoming traffic in the required time. Currently these gateways are using CPUs with a clock frequency of about 150 MHz or even more [5] [6] [7], causing other problems, like high power consumption and high electro-magnetic emission.

The major part of the configuration of such a gateway system is defining the routing information for frames and signals between the channels. Before the emergence of standardized gateway description formats, the configuration data was necessarily specific for a gateway product and prevented easy data exchange with other applications. Checking configurations was time consuming and inherently error prone.

FIBEX is the new upcoming XML-based data exchange standard with the capability to describe complete networks, composed of different communication protocols.

Currently full support for CAN, LIN and FlexRay networks is provided. The support of further protocols is presently under development.

A tool chain was developed to generate code for a special gateway. It is based on a set of protocol communication controllers and an event handler with a processing unit in hardware.

The tool chain reads FIBEX files and produces configuration images for the gateway and communication controllers. The generated code sizes and gateway execution times are presented.

2 FIBEX

FIBEX describes an XML format for data exchange between tools [2], which deal with message-oriented bus communication systems. It is based on an initiative of BMW and was developed in cooperation with automobile manufacturers, suppliers and tool producers. FIBEX is now being maintained at the Association for Standardisation of Automation- and Measuring Systems (ASAM e.V.).

The exchange format covers the functional network, the system topology and the communication level. FIBEX tries to be widely independent from all communication controller implementations and protocols. Currently it is usable for CAN, LIN and FlexRay networks. A draft for a MOST extension is already available upon request and will be published after a short evaluation period. A standardization for TTCAN is still pending.

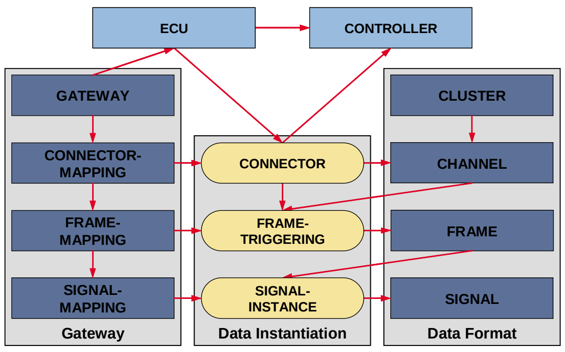


Figure 1: FIBEX object model (simplified)

In FIBEX the descriptions of all elements are split into different objects. The most important objects are shown in figure 1. The cluster, channel, frame and signal objects describe the format and configuration of data, whereas the connector, frame-triggering and signal-instance objects instantiate these data descriptions by providing time and position information. In FIBEX, the definition of gateway configurations is done by defining one-way mappings between these data instance objects. Timing attributes can be defined for each mapping, e.g. message timeout, debounce time, or cyclic sending. Trigger-conditions define the immediate reaction on frame receives, like send *immediate* or *on-change* of the received data.

Unfortunately, the description in FIBEX is very complex. The results are files, which evade easy comprehension. Therefore many tool manufacturers developed FIBEX editors with assistance functions and support for specific configuration aspects, e.g. FlexRay parameters and frame schedules.

Although FIBEX already allows comprehensive descriptions, manufacturer extensions may expand the capability even further. Manufacturer extensions have been defined to clarify certain protocol-specific aspects, like the CAN frame format (distinction between 11 bit or 29 bit identifiers) and to allow the definition of gateway-specific features.

Aside from FIBEX, there is another, similar standard, which can describe the communication layer. This standard, belonging to the Automotive Open System Architecture (AUTOSAR), is also XML-based. The AUTOSAR definition of System Constraint Templates [3] is in fact based on FIBEX. In the future, the tool chain may easily be expanded to support AUTOSAR by an appropriate input filter or converter.

3 Gateway

As described above, FIBEX contains all information to configure a gateway. The target gateway for the tool chain is a dedicated hardware structure optimized for gateway operations. It can be described in four layers as shown in figure 2:

- Processing Layer

- Bus Layer
- Wrapper Layer
- Communication Controller Layer

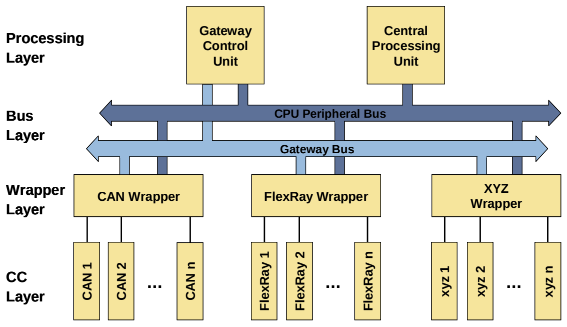


Figure 2: Gateway Layer Architecture

The Gateway Unit in the processing layer contains two configuration RAMs as shown in figure 3. The Vector RAM (VRAM) and Instruction RAM (IRAM) contents are processed by a Finite State Machine (FSM).

The configuration is minimalist and very efficient. The CPU is only needed for the configuration of the GU and to handle exceptional transfers that are too complex to be processed in the Gateway Unit, e.g. advanced transport protocols or extensive arithmetic functions. It has been estimated that with an average gateway configuration less than 20% of the traffic must be handled in software running on the CPU.

The bus layer contains the usual CPU system and peripheral bus with all CPU-accessible peripherals used in the micro controller. An additional gateway bus is connected to the GU and to every wrapper for a set of equal communication controllers. Both buses are used simultaneously and without interfering with each other.

The wrapper layer checks for incoming messages and combines information for groups of 32 message buffers. A selection can be made by masking message buffers meaningless for the gateway. The wrapper is necessary to provide an abstraction of the different types of communication controllers and their signaling of receive events to the GU. Additional features could be implemented here, for example direct data paths between communication controllers of the same type, already done in a CAN-CAN-gateway [1].

The communication controller layer contains slightly modified variants of already widespread CC IP modules.

All communication controllers can be accessed by the CPU and the gateway buses in parallel. Depending on the implementation of the CC different adoptions can be made to optimize the gateway capabilities.

A communication controller with multiple interface buffers for reading/writing message objects is essential to avoid access conflicts between CPU and Gateway Unit. If the CC does neither support multiple buffers or a modification is not possible, both CPU and GU have to arbitrate their buffer accesses using mutexes.

For even better performance it is valuable to have two different and logically independent interface ports for both CPU peripheral bus and gateway bus, called a dual bus interface. If not possible, arbitration has to be used on a per-access base.

Whereas Message RAM based communication controllers have message buffers to differ incoming messages, FIFO based communication controllers have to evaluate the incoming messages by instructions in the IRAM. Specialized commands have been implemented to do the filtering between messages intended for the Gateway Unit and the CPU in an efficient way.

4 FIBEX gateway configuration

Whereas the tool chain is provided with FIBEX files as input, it outputs information for the hard- and software parts of the gateway as shown in figure 3.

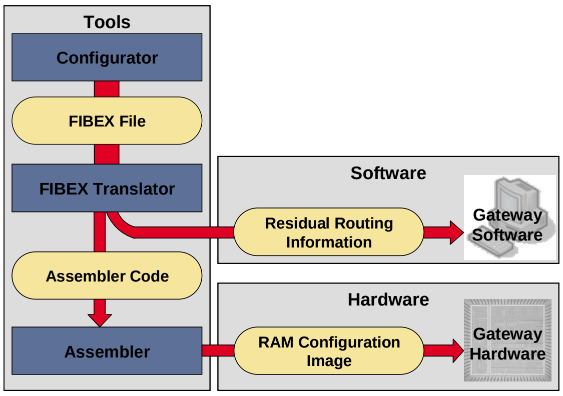


Figure 3: Gateway Tool Chain - Overview

The hardware information is provided by RAM configuration images for the Gateway Unit. Therefore the development was based on the specific RAM layout of the GU. Residual routing information is generated for the software part.

4.1 RAM description

The VRAM configuration selects the communication controller buffers to be used by the Gateway Unit. It also contains additional processing information, like a vector to the event handling functions in the IRAM. The Gateway Unit checks for received messages in the communication controllers and for time events like timeouts, bouncing messages and transmit cycles. Three partitions constitute the VRAM:

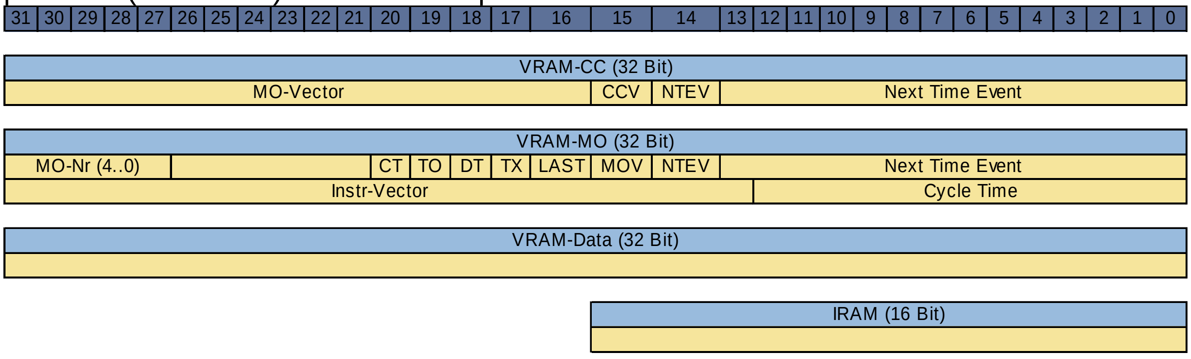
The VRAM Communication Controller partition (VRAM-CC) contains a specific entry for each group of 32 message buffers. When a receive event or a time event occurred in a group, the FSM is triggered to process a table in the VRAM partition addressed by the MO-Vector.

The VRAM Message Object partition (VRAM-MO) has a variable length and contains in tabular form detailed information for each message buffer/object in the corresponding group of the VRAM-CC partition, like rx/tx configuration, timing conditions and the instruction vector to the event handling function in the IRAM. The partitioning between the VRAM-CC and the VRAM-MO reduces the memory usage, as only the message buffers used by the gateway need to have an entry for the message object. It also reduces the time needed to look up a message object.

The remaining VRAM can be used as data storage and is therefore called VRAM-Data.

The state machine processes the events by executing the procedures in the Instruction RAM. The instruction set of this processing unit contains specific functions to

- access the communication controllers
- transfer data
- handle transport protocols
- interact with the host CPU



CCV	= CC Configuration Valid	TX	= Transmit Buffer
NTEV	= Next Time Event Valid	BO	= Dead Time
MOV	= Message Object Valid	TO	= Timeout Buffer
LAST	= Last configured MO	CT	= Cyclic Timeout Message

Figure 4: Gateway vector and instruction RAM

4.2 Tool chain

A tool chain has been developed to generate configuration images for the new gateway architecture. Being able to use FIBEX files as an input, it provides a standardized interface to most multi-network configuration tools, e.g. the DeComSys Designer Pro [4].

Small enhancements to the FIBEX standard have been added. They allow the description of gateway-specific features, like routing signals depending on certain conditions. Other extensions are defined to enable the interaction with a software gateway on top of the hardware gateway, providing coordination between hardware and software by predefining buffer configurations and partitioning of the gateway tasks.

The processing stages are comparable to that of a typical compiler, apart from the different input data and optimization options as shown in figure 5. Figure 6 shows the Assembler stage in more detail.

The first processing step is to generate configuration register sets and Message RAM contents for all involved communication controllers. This can be done for any communication controller and runs independently from the rest of the gateway tool chain. The output of this module is presented as C-style header file.

The gateway object in FIBEX contains all connector, frame-triggering and signal-instance mappings. All required information is read, sorted and stored in an internal gateway table data structure. At this state the gateway table only contains symbolic references to the message buffers of the communication controllers. By searching the generated Message RAM contents, the symbolic references can be resolved and the correct locations added to the gateway table. Aside from using the gateway table as a debug point, it also provides an abstraction of different input formats in the future, e.g. AUTOSAR.

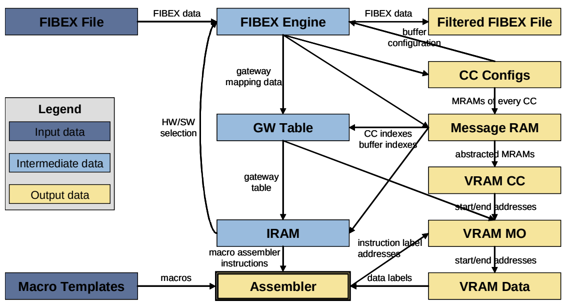


Figure 5: Gateway processing and data flow

The gateway table also contains the timing- and trigger-conditions of every mapping. The frequency of the global FSM timer can be computed from this data.

The gateway table and timer configuration allows generating the Vector RAM contents. However, the address vectors to the actual event handling functions in the Instruction RAM are still symbolic and need to be resolved after the generation of the Instruction RAM contents.

The Instruction RAM contains the functions to handle the receive and time events processed by the FSM. If a message cannot be processed completely in hardware, a special function is generated that can be triggered by the CPU. This information is stored in the filtered FIBEX output by a manufacturer extension of the FIBEX controller object. A software layer has to use this information to interact with the Gateway Unit.

Depending on the timing- and trigger-conditions a high-level assembler code is generated. It contains a lot of macro functions and is therefore almost independent of the target gateway.

By evaluating the macros in the assembler code, it becomes more specific toward the communication controllers and their base addresses in the gateway as shown in figure 6.

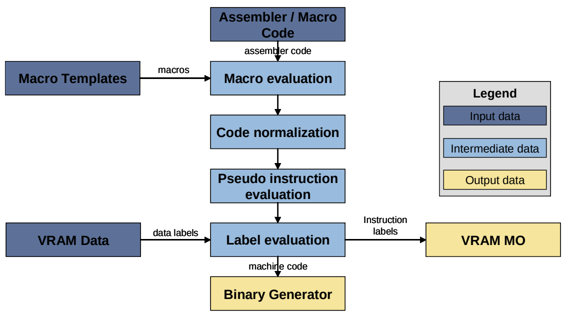


Figure 6: Assembler processing flow

During the normalization, the code is cleaned of any comments, all labels put to extra lines and a unified indentation is made.

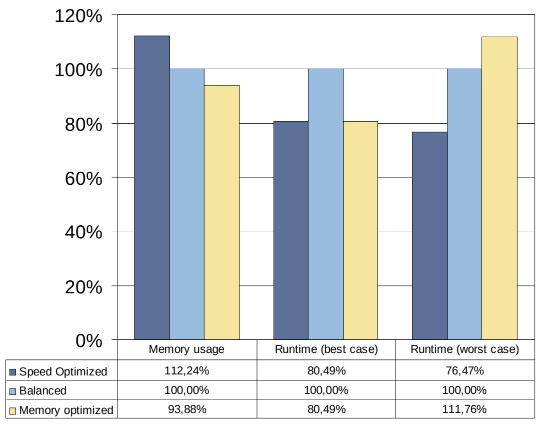
Some pseudo instructions are symbols representing other instructions and need to be translated in the pseudo instruction evaluation stage.

The data and instruction labels are evaluated in the label evaluation stage. They reference entries in the VRAM-Data partition and some locations in the generated assembler code.

The actual address of every function in the IRAM is reported back to the VRAM-MO entries to resolve the symbolic references.

Different output filters can generate C-style header files or binary images containing the contents of the VRAM and IRAM.

The input and output data of every module in the tool chain can be loaded from and saved to external text files. This allows intermediate testing of every module and manual editing and viewing. There exist different test environments around the tool chain and single modules.



	Memory usage	Runtime (best case)	Runtime (worst case)
▪ Speed Optimized	112,24%	80,49%	76,47%
▪ Balanced	100,00%	100,00%	100,00%
▪ Memory optimized	93,88%	80,49%	111,76%

Figure 7: Runtime and Memory Optimization

By choosing different optimization settings of the instruction generator a more runtime- or memory-optimized codes can be generated. The results are shown in figure 7.

4.3 Frontends

There are multiple frontends build on top of the tool chain modules.

One frontend is a graphical wizard for the generation of the gateway configuration, based on a selected FIBEX file. After the generation the data is presented in multiple views to the user and can be directly saved as static configuration images to the software gateway.

An assembler frontend can directly read assembler code from a text file and generate different output formats, like C-style header files or binary images. This allows assembling manually edited files. Another frontend provides a graphical FIBEX viewer with a unique browsing and limited editing capabilities.

5 Results

The combination of tool chain, hardware and software gateway provides one of the first solutions, that is fully configurable by a FIBEX file.

The hardware gateway is able to process about 80% of all transfers of an average gateway configuration. Complex transport protocols and arithmetic operations are processed in cooperation with the software layer of the gateway.

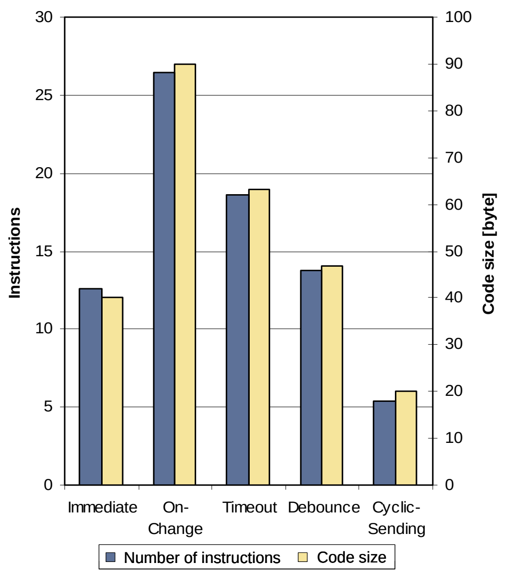


Figure 8: Instruction and code size

A configuration was implemented to demonstrate the available trigger- and timing-conditions. The Gateway Unit was synthesized at 50 MHz on an Altera StratixII EP2S60 FPGA. One FlexRay and three CAN controllers are part of the gateway.

Figure 8 shows the number of instructions and the corresponding code sizes for every mapping. The results show an average number of 15 instructions and an average code size of 50 bytes.

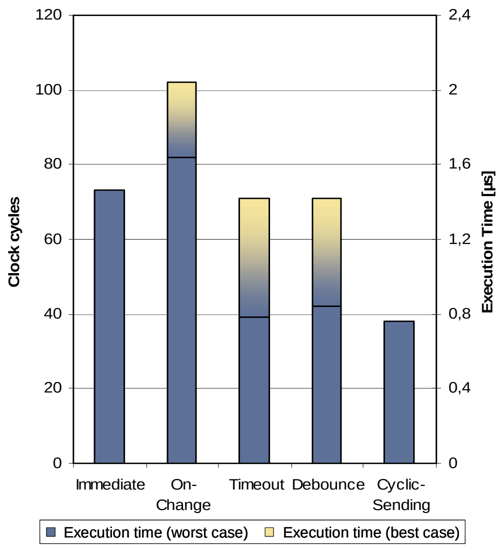


Figure 9: Execution time

The best and worst case execution times of the Gateway Unit are shown in figure 9. Additionally the FSM generates an overhead of 0.4 us for the preparation and post processing of the instruction execution.

6 Conclusion and outlook

A complete tool chain has been developed for generating gateway configurations based on FIBEX files. The configurations include data for the specialized gateway and several communication controllers. The results have shown that most transfers can be completely processed in hardware.

The hardware gateway as coprocessor can reduce the load on the CPU significantly and provides faster data transfers.

The assumptions of very less code size and high execution speed have been verified by a demonstration environment.

More extensive test cases will evaluate the performance in different simulated environments. Support of additional protocols, like MOST/MLB, is planned for the future.

References

[1] Taube, J.; Hartwich, F.; Beikirch, H.: *C_CAN Gateway Module - a new approach for CAN gateways*; Proceedings Embedded World 2005; Nuremberg (Germany); 2005; pp. 80-87

[2] ASAM e.V.: *FIBEX - Field Bus Exchange Format*; Date 26.09.2005; http://www.asam.net/03_standards_06.php

[3] AUTOSAR: *Specification of System Template*, Version 1.0.0; Last Access 31.05.2006; https://svn.autosar.org/repos/10Releases/ AUTOSAR_SpecificationOfSystemTemplates.pdf

[4] DeComSys: *DECOMSYS::DESIGNER PRO*; Date 31.05.2006; http://www.decomsys.com/flyer/ DESIGNER_PRO.pdf

[5] Freescale Semiconductor: *MPC5567*; http://www.freescale.com/webapp/sps/site/ prod_summary.jsp?code=MPC5567; Last Access 17.07.2006;

[6] NEC Electronics GmbH: *NEC Electronics' FlexRay Solutions*; http://www.eu.necel.com/applications/ automotive/040_flexray/index.html; Last Access 17.07.2006

[7] Infineon Technologies: *TC1130 Product Brief*; http://www.infineon.com/upload/Document/ cmc_upload/documents/098/690/ tc1130-pb.pdf; Last Access 17.07.2006

Definitions, Acronyms, Abbreviations

ASAM	Association for Standardisation of Automation- and Measuring Systems
AUTOSAR	Automotive Open System Architecture
CAN	Controller Area Network
CC	Communication Controller
CPU	Central Processing Unit
ECU	Electronic Control Unit
FIBEX	Field Bus Exchange Format
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GU	Gateway Unit
IP	Intellectual Property
IRAM	Instruction RAM
LIN	Local Interconnection Network
MAC	Media Access Controller
MLB	Media Local Bus
MO	Message Object/Buffer
MOST	Media Oriented System Transport
TTCAN	Time-Triggered CAN
VRAM	Vector RAM