To appear in"Proceedings of the 17th IEEE Computer Society Symposium on Object/Component/Service-oriented Real-time Distributed Technology (ISORC), Reno, Nevada, 2014

Virtual CAN Lines in an Integrated MPSoC Architecture

Armin Wasicek[1 2],
University of California, Berkeley[1]
EECS Department
Email: arminw@berkeley.edu

Oliver Höftberger[2], Martin Elshuber[2], Haris Isakovic[2], Andreas Fleck[3]
Vienna University of Technology
Institute for Computer Engineering[2]
Institute for Mechanical Engineering and Mechatronics[3]
Email: {oliver,martine,haris}@vmars.tuwien.ac.at

***Abstract*—The standard solution for automotive control networks is the Control Area Network (CAN) bus. Almost any vehicular computer system comprehends at least one CAN line. For the past two decades, software development for control system has been strongly connected to the properties and interfaces of the CAN bus. Currently, the automotive industry is in the middle of a technology leap towards an information-based industry. New technologies are getting ready to fulfill newly emerging requirements for innovative products such as hybrid engine control, intelligent energy management, and advanced driver assistance systems. Integrated Multi-Processor-on-a-Chips (MPSoCs) will be one part of the solution to provide an adequate computing infrastructure for these newly emerging systems. The established technologies like the CAN bus will have to be reconsidered. In this work, we propose a virtual CAN overlay that abstracts the communication interfaces of an MPSoC to provide the Application Programmer Interface (API) of CAN to programmers. The overlay provides the standard behavior of a CAN line and works transparently over chip boundaries. The major implications is that the programmers can continue their used software development approaches and tools when introducing a new computing infrastructure. The main benefit is that the productivity can be maintained during this critical phase. In summary, our solution helps to mitigate the effects from a technology shift to integrated MPSoCs. Our approach is fully compliant with new automotive software development approaches like AUTOSAR.**

I. Introduction

The automotive industry is undergoing a time of substantial change: current technologies and business models are are getting obsolete and new ones have to be researched. The newly emerging automotive marketplace will be fueled by innovation. It is expected that electronics and software will make up 75% of all future innovations [1]. The percentage of production cost of electronics (including software) increased from 19% in 2004 to 40% in 2010 and it will eventually reach 50% in 2020 [2]. Cars will be future software platforms, much like telephones became nowadays.

Traditional, federate automotive system architectures have been recognized as a bottleneck: in a federated architecture each system function is implemented as a separate computer system (i.e., an Electrical Control Unit (ECU) in automotive terms). A subsystem is then formed by several loosely coupled ECUs (e.g., through a CAN bus). Clearly, the obedience to physical units restricts automotive system architectures greatly. For instance, current vehicles already employ up to 100 ECUs, which is the upper limit to the number of devices a car can harness [3]. Moreover, adding more and more ECUs is a significant cost driver. The solution to overcome these limits is to integrate several system functions in single ECUs, thus forming an *integrated architecture* [4]. Integrated architectures that implement the required partitioning mechanisms can be realized either in software (e.g., as a hypervisor) or in hardware (e.g., as an MPSoC). In this work we focus on MPSoC-based integrated architectures, because we are convinced that they are appropriate to solve the partitioning challenge for hard real-time systems appropriately [5].

Keeping pace with the rapid technological development is challenging, time-to-market is a critical success factor for any future product based on software. An automotive software development process is a complex process that requires teams of engineers with different domain expertise to collaborate, it comprehends a myriad of different tools, diverging standards, and varying legal regulations for each of the global six key markets[1], etc. Introducing new technologies in this tightly organized process and changes happen slowly and incremental. For instance, one of the recent milestones in automotive software engineering is the introduction of the AUTomotive Open System ARchitecture (AUTOSAR)[2], whose project plan was released in May 2003 and the first cars with AUTOSAR technology inside were launched in the market in 2008. Setting up appropriate tool chains and development methodologies simply takes time.

The main contribution of this paper is a virtual CAN overlay to speed up the integration of new computer platforms leveraging MPSoC technologies. The presented virtual CAN overlay facilitates similar properties and behavior to a real CAN bus. Implementing automotive applications against this virtual CAN interface leverages existing software development methodologies and tool chains while introducing a new computational MPSoC platform.

The paper is organized as follows: Section II elaborates on integrated MPSoC architectures which form the basis for our research. Section III contains a technical description of the proposed virtual CAN overlay. In Section IV we present a case study of a hybrid engine controller which has been implemented and tested in an automotive Hardware in the Loop (HiL) testbed. Finally, we discuss some related work in Section V and draw a conclusion in Section VI.

[1] I.e., China, India, Western Europe, Japan, Korea, and the United States
[2] http://www.autosar.org/

II. An Integrated MPSoC Architecture

In this section we present briefly the ACROSS MPSoC architecture which is a research platform for hard real-time embedded systems. We the following sections we discuss implementation and validation of the virtual CAN overlay on this platform. Our results, however, are transferable between similar system architectures with predictable interconnect and according spatial and temporal partitioning properties.

A. Benefits of Integrated Architectures

In a federated architecture, the provision of dedicated resources for each subsystem sums up to a high amount of redundant hardware. Integrated architectures aim at reducing this overhead by providing common computing resources to several system functions and thus *virtualizing* the required resources for executing a single system function. These architectures have the potential to enable massive cost savings, reliability improvements, and to overcome limitations for spare components and redundancy management [4].

For instance, integrated architectures can help to reduce the wiring of an automotive system by replacing physical cabling with virtual channels. This directly leads to lower manufacturing costs since fewer connectors, cables, electronic parts, and assembly steps on the production line are required. Furthermore, a reduction of the number of connectors can lead to improved reliability of the entire system, because, if fewer connectors are used, one major source (i.e., 30 %) of electrical failures in cars gets eliminated.

B. The rise of MPSoCs

A MPSoC incorporates multiple, potentially heterogeneous processing cores and other functional units on a single silicon die. Compared to general-purpose single core processors, MPSoCs can provide enormous computational capacity in an energy-efficient and cost-efficient way. The roadmaps of the semiconductor industry [5] show a very clear trend towards multi-core technology, and we can safely assume that the majority of future high- end processors will be MPSoCs. Today, MPSoCs are typically applied in personal computers or consumer electronic devices like smart phones or tablets.

Safety-critical systems are systems whose failure could result in loss of life, significant property damage, or damage to the environment. Examples are flight control systems for aircraft, automotive control systems, medical devices, industrial control systems or nuclear power plants. MPSoCs could bring many benefits (e.g., energy and area efficiency, increased computational performance, specialized cores, reduction of physical units) to safety-critical applications. However, the currently existing MPSoC architectures were not designed with a strong focus on safety and certification and thus have serious drawbacks and limitations for hard real-time applications.

C. The ACROSS MPSoC in a nutshell

The ACROSS MPSoC is a computer system architecture specifically targeted at safety-critical applications, i.e., systems including safety functions that have to fulfill the highest certification requirements. The ACROSS project[3] has been started in order to overcome the limitations of MPSoCs for hard real-time systems. This is achieved by implementing spatial and temporal partitioning (also called segregation, or non-interference) between single components.

[3]http:www.across-project.eu

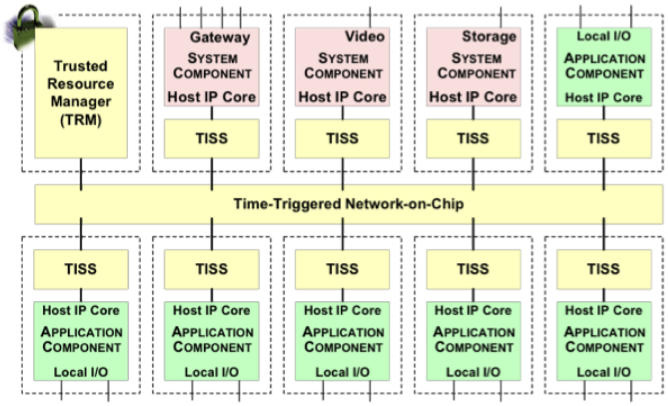

Fig. 1. ACROSS MPSoC architecture

An ACROSS MPSoC (see Figure 1) consists of two subsystems. The application-specific subsystem (green and red) encompasses a set of potentially heterogeneous components and is targeted to implement the actual function of an application. The trusted subsystem (yellow) provides an interconnect to integrate application components into a system. The core innovation of the ACROSS MPSoC is the Time-Triggered Network-on-a-Chip (TTNoC) that enables the dependable interconnection of IP cores (i.e., components) [6]. Each core in the MPSoC has access to a consistent chip-wide notion of time (i.e., the macro tick).

1) Structure of the Interconnect: The trusted subsystem provides a set of core services towards the application-specific subsystem. These core services manifest themselves in the Trusted Interface Subsystem (TISS) which essentially implements a periodic and sporadic message transfer scheme, as well as some services to control the attached component. The trusted subsystem acts autonomously and transfers messages according to an a priory defined communication schedule between the sender and the set of receivers (multi-cast is supported) at a message's defined send instance with reference to the macro tick. An immediate benefit for the system engineer is that the time-triggered schedule is free of conflicts, so no online arbitration required and temporal behavior is 'designed', so no wearisome analysis after the implementation has to be done. Moreover, the trusted subsystem will ensure fault-containment [7] by exploiting information about the permitted temporal behavior of the host IP cores in order to detect and contain an arbitrary temporal failure of a host (e.g., babbling idiot, masquerading faults). For this purpose, each TISS acts as a guardian of the hosted component preventing temporal and spatial interference.

2) Anatomy of a Component: The application-specific subsystem is further enhanced by deploying a combination of middleware and system components to customize the generic core services to a particular application domain. System components (red) usually are used to implement a specific gateway functionality that can be shared by several application components (green). Middleware layers refine single

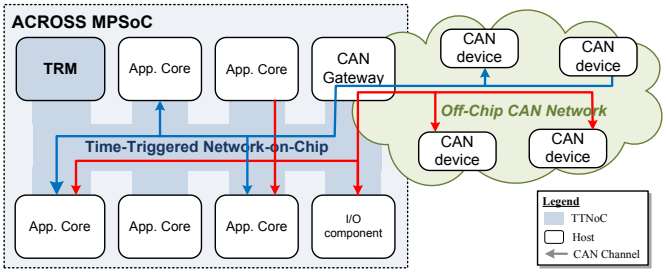

Fig. 2. VCAN overlay system structure

application components. For instance, a port of the PikeOS microkernel [8] is available to provide typical Real-Time Operating System (RTOS) services (e.g., real-time scheduling) to application components. Single application component run distinct instances of PikeOS and communicate only using the core services. A refinement of the core services towards the automotive domain is the virtual CAN overlay which is also implemented as a Middleware according to ACROSS terminology.

D. Connection to AUTOSAR

The PikeOS microkernel has the capability to implement different APIs, so-called personalities. For instance, a personality for AUTOSAR is available. This personality gives the the application programmer the possibility to implement against the domain-specific Runtime Environment (RTE) interface rather that towards the native system call interface. On the bottom of the middleware stack, however, there is a missing link between the microkernel and the TISS. This link can be established by either directly integrating the core services as a new Complex Device Driver which requires additional integration effort on the application programmer's side. Or AUTOSAR middleware and TISS can be linked via the CAN overlay. This solution does not require any changes to the application programmer's programming model or tool chain[4].

III. Virtual CAN lines

The Virtual Control Area Network (VCAN) overlay provides VCAN lines on top of a MPSoC. It enables the seamless integration of physical and VCAN lines. The application programmer just has to deal with nodes in a CAN network. The location (i.e., either on-chip or off-chip) of the nodes is transparent to the programmer. Figure 2 depicts this concept. Two CAN lines (red and blue) connect an arbitrary number of on-chip and off-chip nodes. On the boundary between on-chip and off-chip sits a VCAN gateway that implements a physical CAN connectivity. Each node deploys a VCAN Provider which implements the CAN API.

A. CAN System Model

A CAN system [9] is a multi master network which uses a CSMA/CD (Carrier Sense Multiple Access/Collision Detection) variant with arbitration on message priority. Each message is assigned a fixed priority/ID and a sequence of messages forms a signal. Before sending a message the CAN controller checks, if the bus is busy, then starts transmission. If two nodes are simultaneously starting to send, the one with the higher priority will win the arbitration and the other will back-off.

In order to support a wide variety of legacy CAN applications the VCAN gateway service supports the BasicCAN (receive queues) and the FullCAN (mailboxes) behavior. BasicCAN and FullCAN are not defined in the specification of the CAN standard, but describe two different principles in the architecture of a physical CAN controller, considering mainly the way how incoming messages are filtered. BasicCAN is usually used in cheaper standalone CAN controllers or in smaller microcontrollers with an integrated CAN controller. A BasicCAN controller has a single FIFO receive-queue with a global acceptance filter. FullCAN is used in high performance CAN controllers and microcontrollers. FullCAN controllers have a set of buffers called mailboxes that are assigned an identifier and are set to work as transmit, receive or remote buffers. When the CAN controller receives a data message it checks the mailboxes in order to see whether there is a mailbox configured as a receive buffer that has the same identifier as the incoming message. If such a mailbox exists, the message is stored in the mailbox and the host is notified. Otherwise the message is discarded. When a remote message is received, the controller checks the message identifier against the mailboxes configured as remote buffers. If a match is found, the controller automatically sends a message with the identifier and data contained in that mailbox.

B. CAN Hardware

Physically, the VCAN Gateway consists of some hardware logic and resources (i.e., TISS, NIOS2 processor, some on-chip memory, local IO, pins) on the FPGA and an expansion board. The expansion board uses an M51–Quadruple CAN Interface, which provides four distinct SJA1000 CAN controllers[5]. Thus, we support with this design up to four different physical CAN lines. Each CAN controller can be operated in one of two modes: BasicCAN mode or PeliCAN mode. For the VCAN overlay only PeliCAN mode is used, because this mode also covers the BasicCAN mode. Moreover, the communication speed for each controller can be configured individually from 62.5 kbits/s up to 1 Mbits/s.

For outgoing messages each CAN controller provides a transmit buffer that is able to hold one message. This message is autonomously transmitted after a successful bus arbitration. A receive buffer with 64 bytes allows to receive between 4 and 21 messages, depending on the frame format and number of data bytes per message. When the receive buffer is full, messages have to be read from the buffer before further messages can be received. Otherwise, messages can get lost. In order to guarantee the same message order for components inside the MPSoC as well as receivers outside, the CAN controllers are configured to also receive messages that have been sent by the CAN controller itself. Thus, virtual, on-chip CAN messages are not treated differently to messages on the

[4]Note that the CAN overlay is fully functional without AUTOSAR. We point out this connection only because our case will make use of this potential integration.

[5]The actual number of transceivers is arbitrary and not constrained by our design. Four simply seemed to satisfy our demonstrator's requirements.

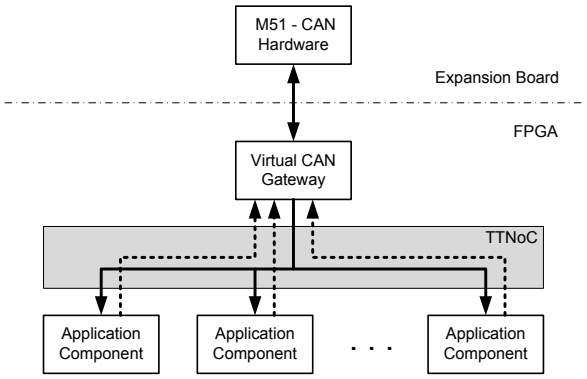

Fig. 3. On-chip and off-chip parts of the VCAN overlay

off-chip, physical CAN hardware. Only messages that won the bus arbitration and that have successfully been transmitted on the MPSoC external CAN network can be received by components inside the MPSoC.

The CAN hardware is connected to a gateway (marked as *IO Component* in Figure 3) via a memory-mapped interface that allows accessing the hardware by simply reading or writing memory locations. On this interface data has to be transferred in bytes. Each component within the MPSoC that uses the VCAN overlay is connected to the gateway through a periodic, unidirectional communication channel. Therefore, each component implements one single-cast channel to the gateway and the gateway implements a multi-cast channel to all registered nodes.

C. Virtual CAN Gateway

The VCAN Gateway acts as an intermediary between the endpoints of a CAN line. Endpoints can be either VCAN Providers (see Section III-D) on the application components, or CAN nodes connected to the off-chip network. The purpose of the VCAN Gateway is to implement virtual bus arbitration between CAN messages in order to privilege messages with a lower ID (i.e., higher priority). Furthermore, incoming messages have to be distributed to all components that are interested in a specific CAN line. As at the gateway implements no message filtering, all messages are forwarded to the VCAN Providers.

The VCAN Gateway Application consists of two tasks, one task is responsible for reading and writing to the CAN hardware, and the other task manages the communication with the VCAN Providers connected via the TTNoC. In order to synchronize the event-triggered hardware access task and the time-triggered virtual CAN communication task, both tasks implement receive queues and send buffers. Figure 4 depicts the structure of the VCAN Gateway.

A receive queue is an array of messages for an individual CAN channel that stores incoming messages until the time-triggered process collects them and sends them via the TTNoC. Send buffers hold one message per CAN line for each component. After a message has been sent successfully, the according send buffer is freed and the component is informed in order to allow further messages to be sent. Due to the semantics of

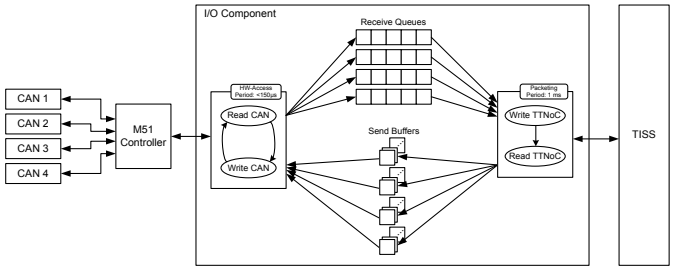

Fig. 4. VCAN gateway structure

CAN messages the following update strategies for send buffers are possible:

Send buffer empty: a new message can be placed in the send buffer without restriction

Send buffer occupied: if new message has

- *Higher message ID (i.e., lower priority):* new message cannot be copied to the send buffer
- *Same message ID:* new message replaces old
- *Lower message ID:* new message replaces old

1) Hardware access task: The hardware access task is responsible to write messages into the transmit buffers of the individual CAN controllers and to read messages from each of the receive buffers of the CAN controllers. Therefore, it checks each CAN controller whether the transmit buffer is ready for a new message to be sent. If yes, it searches the highest priority message (i.e., the message with the lowest message ID) from the send buffers that has to be sent on the according CAN channel. The send buffers contain at most one message per CAN line per component.

After the transmit buffer of all CAN controllers have been written, the task checks, if the receive buffer of the CAN controller contains messages. In case new messages are available, the controller copies them to the receive queues of the VCAN Gateway and the hardware buffers of the CAN controller are freed.

2) Periodic communication task: The periodic communication task manages the TTNoC communication with the VCAN Providers at the application components. It is activated with a frequency of 1kHz – this frequency is necessary to attain the requirements for the automotive case study (see Section IV). The task's activation instant is synchronized to the send and receive instants of virtual CAN messages by means of a task-trigger event emitted by the TISS.

At first, CAN messages are read from the receive queues and appended to the VCAN message sent on the TTNoC. This message has a static size and is designed to guarantee that no message will be lost at a CAN transfer rate of 1 Mbits/s. It additionally contains acknowledgment information about successfully sent messages for each individual component (for details on the virtual CAN message format see Section III-E).

After the virtual CAN message has been written to the TISS, incoming virtual CAN messages from the application components are read. These virtual CAN messages contain at most one CAN message per CAN line. The content of the CAN message is stored in the according send buffer. From

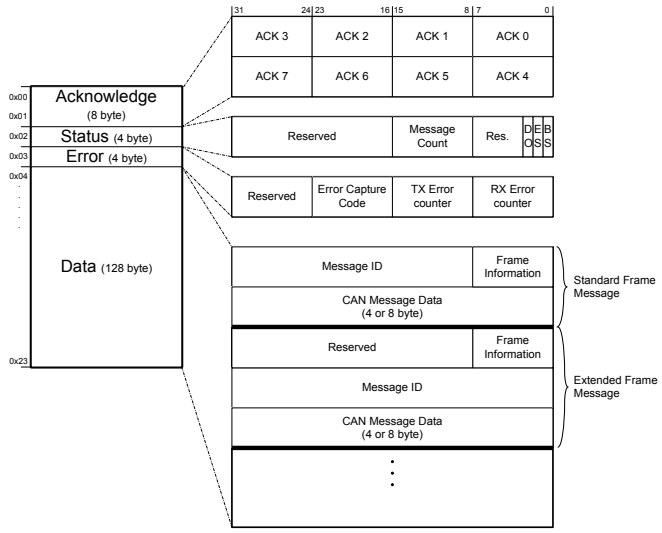

Fig. 5. VCAN message segment

there, the hardware access task constantly selects the message with the highest priority to be sent. Additionally, the virtual CAN message contains abort request information which tells the VCAN Gateway to cancel the transmission of a specific CAN message.

D. Virtual CAN Provider

The VCAN Provider is middleware running on an application component on top of a PikeOS instance. It enables applications to use the VCAN overlay. The VCAN Provider creates a virtual CAN controller locally. The Virtual CAN Provider is implemented as a file provider in the PikeOS and uses PikeOS' system extensions mechanism. Each VCAN line corresponds to one file provider. Each file provider contains all necessary information for the configuration of a CAN line, and it also provides memory containers for storing incoming and outgoing CAN messages. The VCAN Provider communicates with the TISS system extension on one side and the application software on the other side. The VCAN Provider is connected with the VCAN Gateway over two sporadic TTNoC ports.

E. Virtual CAN Message Format

VCAN messages are used to distribute CAN messages, status information regarding the hardware, control information, and acknowledgment data about sent CAN messages between VCAN Providers and VCAN Gateway. There is one message type for each direction (corresponding to solid and dotted lines in Figure 3). Both types of VCAN messages reserve bandwidth for each of the four CAN channels. Thus, VCAN messages consist of four segments having the same structure.

The benefits of this setup is that (a) no CAN line is influenced by the traffic load of another CAN channel and no message gets lost even in full load scenarios, because of no shared resources, and (b) it enables direct access to the required CAN segment, since the address offset within the virtual CAN message is known.

1) Virtual CAN Gateway message: The Virtual CAN Gateway message is a single message that is transmitted to all VCAN Providers simultaneously. Figure 5 depicts the structure of one segment of a VCAN message. Each segment starts with an acknowledgment block that contains an **ACK** field for each component. This field is an 8 bit sequence number, which indicates to the VCAN Provider that a CAN message has been sent successfully on the CAN line. If the *ACK* field is 0, then no message from the according VCAN Provider has been sent on the CAN line in the last period. Otherwise, the sequence number that has been sent by the Virtual CAN Provider in the transmit request message is returned. The next element in the message structure is the **status word** which is divided into Bus Status (BS), Error Status (ES), Data Overrun (DO), and Message Count record fields.

The BS flag is read from the CAN controller and signals whether the controller is involved in bus activities, or not. When the **ES** flag is set, an error occurred, which cause can be read from the error capture code. Is the **DO** flag set it means, that an incoming CAN message has been discarded as the receive buffer was full. **Message count** contains the number of CAN messages that are transmitted in the Data section of the virtual CAN message.

In the **error status word** a copy of the content of different hardware registers of the CAN controller is provided. This includes the **RX and TX Error counters**, as well as the **Error Capture Code**, that may indicate the cause of a communication or controller error.

The **data section** of the VCAN Gateway message is reserved for the actual CAN messages that have been received by the CAN controller. The size of this section (i.e., 128 byte) is designed to guarantee no message loss at a CAN communication speed of 1 Mbits/s and a virtual CAN message period of 1 ms. In order to save bandwidth, this section is organized as one continuous data block where a CAN message only consumes as many data words as needed. In this data section, CAN messages are further distinguished according to their frame format. For standard frame messages the Frame Information and the Message ID fields share one 32 bit data word. Extended frame messages use separate data words for the frame information and the message ID. Depending on the number of bytes in the CAN message data section, one (i.e., for $1-4$ data bytes) or two (i.e., for $5-8$ data bytes) data words in the virtual CAN message have to be used. Thus, for each CAN message between 1 and 4 data words with 32 bit are consumed.

2) Virtual CAN Provider message: The Virtual CAN Provider message is an individual message from each of the Virtual CAN Providers to the Virtual CAN Gateway Application. It is used to communicate transmit requests, which require that a CAN message that has to be sent is added, or abort requests (or both). In Figure 6 the structure of one segment of the VCAN Provider message is shown. Beside the CAN message content (i.e., Frame Information, Message ID and Data) this message type contains a **Transmit Request Sequence Number** and an **Abort Request Sequence Number**. Only if the sequence number is not equal to zero, the according command is valid.

The transmit request sequence number is incremented

whenever a new CAN message has to be sent. If no CAN message is available, it is reset to 0. This number will be returned by the VCAN Gateway when the CAN message has been sent successfully on the CAN line. When the transmission of a CAN message should be aborted, the abort request sequence number is set to the sequence number of the transmit request to be deleted. If the CAN message has not been sent, the VCAN Gateway will delete the referenced CAN message from the send buffers and CAN controller registers.

In contrast to VCAN Gateway messages, CAN messages that are contained in transmit requests always have the same structure and reserved message size.

F. Virtual CAN Protocol

The VCAN protocol defines the sequence of messages exchanged on the TTNoC and the off-chip CAN network. For incoming CAN messages (i.e., originating from an off-chip node), this sequence is straightforward:

1) M51 CAN controller receives CAN message
2) VCAN Gateway reads CAN message from controller
3) VCAN Gateway appends CAN message to VCAN Gateway message
4) VCAN Provider receives and processes the VCAN message

The sequence for outgoing CAN messages (i.e., originating from some application component) is as follows:

1) VCAN Provider sends CAN message with Transmit Request Sequence Number (TRSN) to VCAN Gateway
2) VCAN Gateway tries to send the CAN message on the CAN network
3) If CAN message can be transmitted on the CAN network, the message is simultaneously read by the CAN controller
4) CAN message is read back (i.e., read from the receive buffer of the CAN controller) by the VCAN Gateway and treated like a normal incoming CAN message
5) CAN message is packed into the VCAN Gateway message and the TRSN is copied into the ACK field of the application component from which the CAN message originates.
6) VCAN Provider reads the VCAN Gateway Message and uses the acknowledgment information to free the send buffer

The VCAN Provider has to wait at least one complete period until it receives the acknowledgment for a transmit request. But due to the send buffer update strategy (see Section 1.2), the VCAN Provider may send a new transmit request (with incremented TRSN) to the VCAN Gateway Application even if the old transmit request has not yet been acknowledged. This may only happen if the message ID of the new CAN message is lower or equal to the previously sent transmit request. CAN message with higher message ID can only be transmitted when the old transmit request was acknowledged. Because of the TRSN and the ACK field, the VCAN Provider always knows which of the transmit requests is acknowledged.

In Figure 6 an exemplary protocol sequence is shown. The VCAN Provider sends a transmit request ($TR = 27$) for a CAN message with $ID = 48$. As no CAN messages have been receive on the CAN network until the first VCAN Gateway Message is generated, this message does not contain any CAN message and an empty ACK field (i.e., $ACK = 0$). After an (unrelated) incoming CAN message with $ID = 35$ was received, the VCAN Gateway could send the CAN message from the transmit request 27 on the CAN network. Simultaneously, the CAN message is read by the CAN controller.

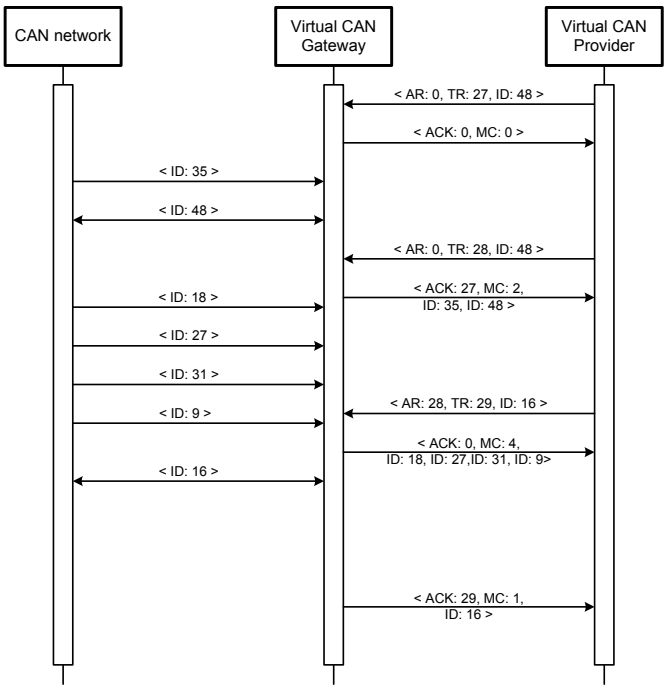

Fig. 6. Exemplary VCAN protocol execution

The VCAN Provider sends another transmit request ($TR = 28$), even if the old request was not yet acknowledged. Thereafter, the VCAN Gateway transmits the CAN messages that have been received since the last VCAN Gateway Message (i.e., messages 35 and 48). As the CAN message of transmit request 27 has been successfully sent, the ACK field returns the sequence number of this request ($ACK = 27$).

During the next period the VCAN Gateway is not able to send its CAN message, as all incoming messages have higher priority on the CAN network. When the VCAN Provider sends its next transmit request ($TR = 29$), the message also includes an abort request command ($AR = 28$). This tells the VCAN Gateway to remove the CAN message from the previous request from the send buffers. The next VCAN Gateway Message does not acknowledge any transmit request, but contains all CAN messages that have been received in the last period.

Finally, the VCAN Gateway send the CAN message from the last transmit request on the CAN network and sends the acknowledgment at the end of the period.

IV. Case Study: Hybrid Vehicle Control

The automotive case study demonstrate how the ACROSS MPSoC can be used to integrate several ECUs on a single chip.

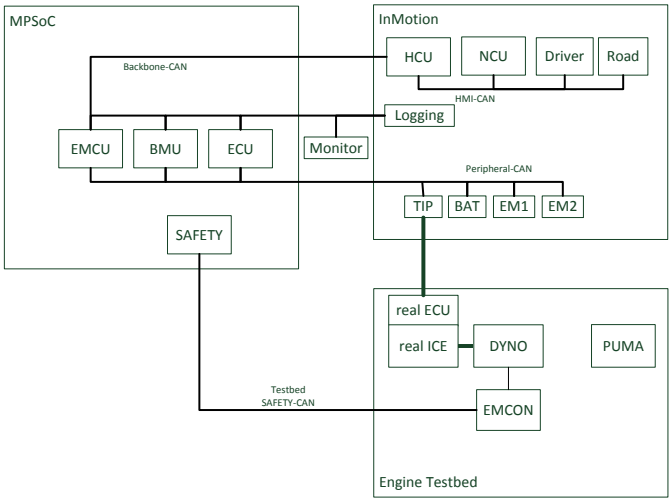

Fig. 7. System model of the case study

It shows how the VCAN overlay is used to connect the single ECUs and how it integrates with a sophisticated automotive tool chain. The goal of the used hybrid control application is to minimize fuel consumption.

The case study realizes a HiL setup. Figure 7 and Table I depict and describe the according system model. A HiL setup typically consists of a unit under test (the MPSoC), an environmental simulation and optionally a physical testbed. We used the AVL InMotion[6] HiL tool which offers an interface to different kinds of automotive testbeds. In the automotive testbed we used a custom hybrid engine.

For validation of the correct functioning of the VCAN infrastructure, we did some integration tests. Next, we considered the communication between InMotion and the MPSoC in the hybrid engine control scenario. This communication is done using two CAN lines (Backbone-CAN and Peripheral-CAN). Measurements were made with the AVL InMotion Software CAN-Analyzer.

A. Integration Tests

1) Message Completeness and Frequency Test:

Rationale Despite a high bus utilization, the completeness of messages and their respective frequencies must be guaranteed.

Procedure Both the MPSoC and AVL InMotion are connected to a CAN-Analyzer. The messages received on the CAN-Analyzer are checked against their reference values while frequency is increased. This is repeated for all CAN lines.

Result This test has been completed successfully.

2) Signal Calibration Test:

Rationale Signals (e.g., a speed signal) from the application must be packed into an according sequence of CAN messages by the sender and unpacked by the receiver.

Procedure Both the MPSoC and AVL InMotion are connected to a CAN-Analyzer. Each signal is brought to certain defined values (Using InMotion and the CAN-Analyzer). By checking the received signal for differences, this test indicates systematic and sporadic errors in definitions and implementation.

Result Sent and received signals were consistent.

3) Delay/Jitter – Tests:

Rationale Communication delays – or dead times – are of special interest in the area of control theory since they have a crucial impact on the control performance and even stability.

Procedure A CAN-Analyzer is used to introduce specially shaped (rectangle) signals on the CAN line, which are mirrored by different nodes and then read back by the CAN-Analyzer. By comparing the sent signal and the received signal, the transportation delay can be evaluated.

Result Delay times were nearly constant in all test cases. Delay values scatter around $500\mu s$. Thus, the VCAN overlay does not introduce significant delay or jitter.

[6]https://www.avl.com/

TABLE I. DESCRIPTION OF THE ELEMENTS OF THE CASE STUDY

Component	Description
HCU	The HCU (Hybrid Control Unit) implements the supervisory hybrid control strategy. It interfaces the Driver and commands EMCU and ECU.
NCU	The NCU (Navigation Control Unit) processes position information and estimates future loads based on the desired route.
Driver	The driver is a simulation that keeps the virtual vehicle on the virtual track in a desired manner.
Road	The road simulates external conditions and interacts with the virtual vehicle as well as with the Driver and the NCU.
EMCU	The EMCU (EMotor Control Unit) receives demands from the HCU and controls EM1 and EM2 considering constraints such as StateOfCharge.
BMU	The BMU (Battery Management System) monitors the vehicle's battery and calculates the StateOfCharge as well as other vital parameters.
ECU	The ECU (Engine Control Unit) receives demands from the HCU and controls the ICE (Internal Combustion Engine) of the vehicle.
SAFETY	This module implements a safety concept according to a machinery directive. It monitors the doors at the testbed, speed of engines, ...
DYNO	The Dyno is a powerful electrical drive controlled by EMCON. It is used to establish the simulated rotational speed at the shaft of the engine.
EMCON	Hardware and software for the control, manual and automatic operation of the combustion engine and dynamometer on an engine test bed.
PUMA	PUMA supports its users to execute testing tasks at the testbed.
TIP	TIP is the interface layer between AVL InMotion and the testbed.
BAT	The BAT is traction battery of the HEV. It is simulated within AVL InMotion and interfaces both EMotors and the respective Control Units.
EM1	EM (EMotor and Generator) is used to accomplish hybrid electric functionalities that lead to reduction of consumption and emissions.
EM2	See EM1
Monitor	In order to keep track of the communication between the ACROSS
Logging	MPSoC and AVL InMotion. External components (CAN-Analyzer) are used. Also, inside AVL InMotion, a Data-Logging unit is realized.

B. AUTOSAR Software Development

The workflow to derive the control application for hybrid vehicles is depicted in Figure 8. It conforms to the AUTOSAR specification and involves several tools from different suppliers. The input is basically the System Model, that describes the whole application running on the MPSoC, and the applications, which are the ECUs and the safety unit.

C. Hybrid Engine Control Use Case

1) Backbone CAN: The Backbone CAN line is used to send demand-values from the supervisory hybrid control unit (HCU) to the respective plant control units (ECU, EMCU, BMU) and to report measured values from the hybrid powertrain back. It uses a CAN speed of $1Mbit$ and Standard CAN (11bit) identifiers. In total there are 17 CAN messages defined that require a bandwidth of $15.04kb/s$.

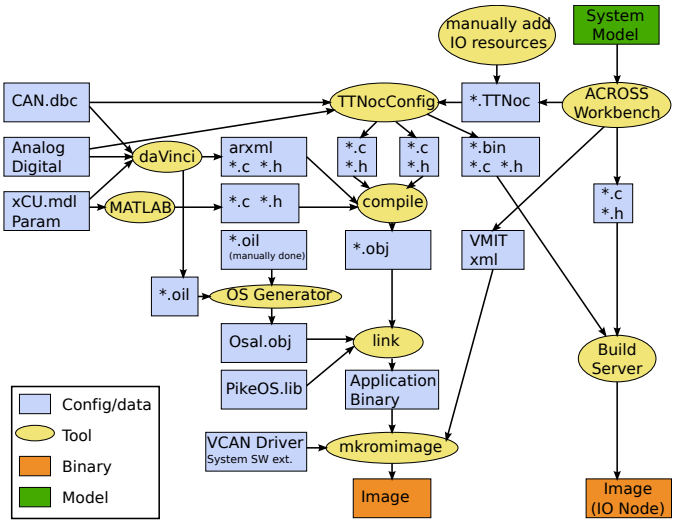

Fig. 8. Workflow and component view for automotive case study

2) Peripheral CAN: The Peripheral CAN line realizes the communication with powertrain-components such as ICE, EM and Battery and their according ECUs. In addition to this CAN line, several signals from sensors are fed into the system using analog and digital IO (and vice versa with actuator signals to powertrain components). In total there are 16 CAN messages defined that require a bandwidth of $13kb/s$.

A successful NEDC test [10] was conducted to assess the fuel economy of different hybrid control strategies.

V. Related Work

The CAN bus [9] was introduced by the Robert Bosch GmbH in 1986 and it became soon the de-facto standard for in-vehicle communication [11]. A multitude of works on the CAN bus is available. Many textbooks on real-time systems describe its working principle [12], [13]. Studies on its a capabilities and properties have been published, for instance, critical review on its real-time performance [14] or on its security [15].

The huge success of the CAN bus and the availability of low cost controllers extended its usage beyond the automotive industry and triggered new developments based on the original CAN protocol. Most notably, CANopen [16] and time-triggered CAN [17]. Also the applicability of the CAN bus as a serial communication protocol for the avionics has been considered [18]. Virtual CAN networks on top of a time-triggered network have been treated in [19]. A CAN gateway connecting physically distinct CAN lines is described in [20].

VI. Conclusion

In this work we presented a virtual network layer emulating a CAN bus. The resulting VCAN overlay facilitates the connection of purely virtual with physical CAN lines. Such a software layer is particularly useful when introducing new computation platforms like MPSoCs in existing production environments with established work flows and tool chains. We demonstrated our solution by implementing an automotive control application for hybrid vehicles on top of the VCAN overlay. The application was developed using the AUTOSAR design flow and tools.

Acknowledgment

The authors would like to thank Christian El-Salloum and Michael Kang for the fruitful cooperation. This research was in part supported by a Marie Curie IOF Action within the 7th Framework Programme under the funding ID PIOF-GA-2012-326604 (MODESEC). The responsibility for the content rests with the authors.

References

[1] M. Girardot and M. Schwarz, "Electronics-ization in automotive: Reinventing the industry model to boost profitable innovation," Cisco IBSG, Whitepaper, 2012.

[2] H. Wallentowitz, A. Freialdenhoven, and I. Olschewski, *Strategien in der Automobilindustrie: Technologietrends und Marktentwicklungen*. Teubner Verlag / GWV Fachverlage GmbH, 2009.

[3] M. Broy, I. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356–373, 2007.

[4] R. Obermaisser, C. El-Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 7, pp. 956–965, 2009.

[5] C. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek, "The across mpsoc – a new generation of multi-core processors designed for safety-critical embedded systems," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, 2012, pp. 105–113.

[6] C. Paukovits, "The Time-Triggered System-on-Chip Architecture," Ph.D. dissertation, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, Dec. 2008.

[7] R. Obermaisser and O. Hoftberger, "Fault containment in a reconfigurable multi-processor system-on-a-chip," in *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, 2011, pp. 1561–1568.

[8] R. Kaiser and S. Wagner, "The pikeos concept: History and design," SYSGO AG, Whitepaper, 2007.

[9] *Road vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication*, International Standardization Organization Std. ISO-11 898, 1993.

[10] T. J. Barlow, S. Latham, I. S. McCrae, and P. G. Boulter, "A reference book of driving cycles for use in the measurement of road vehicle emissions," TRL Limited, Published Project Report PPR354, June 2009.

[11] L.-B. Fredriksson, "Can for critical embedded automotive networks," *Micro, IEEE*, vol. 22, no. 4, pp. 28–35, 2002.

[12] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer, 2011.

[13] W. Voss, *A Comprehensible Guide to Controller Area Network*, 1st ed. Copperhill Media Corporation, 2005.

[14] H. Kopetz, "A comparison of can and ttp," in *Proc. 15th IFAC Workshop on Distributed Computer Control Systems (DCCS)*, 1998.

[15] R. Kammerer, B. Fromel, and A. Wasicek, "Enhancing security in can systems using a star coupling router," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, 2012, pp. 237–246.

[16] *Industrial Communication Subsystem based on ISO 11898 (CAN) for Controller-dDevice Interfaces - Part 4: CANopen*, CAN-in-Automation (CiA) Std. DS/EN 50 325-4.

[17] *Road vehicles – Controller area network (CAN) – Part 4: Time-triggered communication*, International Standardization Organization Std. ISO-11 898-4:2004.

[18] C. Lin and H. Yen, "Reliability and stability survey on can-based avionics network for small aircraft," *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, vol. 2, p. 8, 2005.

[19] R. Obermaisser, "Reuse of can-based legacy applications in time-triggered architectures," *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 4, pp. 255–268, 2006.

[20] Infineon, "Multican – CAN-gateway functionality witout cpu interaction," Infineon Technologies AG, Application Note AP29005, 2007.