The Time-Triggered System-on-a-Chip Architecture

Roman Obermaisser, Christian El Salloum, Bernhard Huber, Hermann Kopetz
Real-Time Systems Group
Vienna University of Technology, Austria
Email: romano@vmars.tuwien.ac.at

Abstract— **It is the objective of the presented System-on-a-Chip (SoC) architecture to provide a predictable integrated execution environment for the component-based design of many different types of embedded applications (e.g., automotive, avionics, consumer electronics). At the core of this architecture is a time-triggered network-on-a-chip for the predictable interconnection of heterogeneous components. A component can be a self-contained computer, including system and application software, an FPGA, or a custom hardware unit. By providing a single uniform interface to all types of components for the exchange of messages, the architecture supports the component-based design of large applications and enables the massive reuse of components. The time-triggered network-on-a-chip offers inherent fault isolation to facilitate the seamless integration of independently developed components, possibly with different criticality levels. Furthermore, mechanisms for integrated resource management support dynamically changing resource requirements (e.g., different operational modes of an application), fault-tolerance, a power-aware system behavior, and the implementation of fault-handling by reconfiguration.**

I. Introduction

During the past forty years, the semiconductor industry has developed chips of breathtaking complexity, increasing the number of transistors per chip to close to a billion. Fueled by these spectacular improvements in the functionality of chips and the cost-reduction of semiconductor devices, the field of embedded computing has grown to become the most important segment of the computer industry. However, the management of the increasing complexity is becoming a key challenge in the domain of embedded systems. The 2005 semiconductor industry roadmap [1] considers system design complexity and designer's productivity as key challenges on the way to giga-scale SoCs. This challenge can only be tackled if we lift the design process to a higher level of abstraction.

Also, it is amazing that the basic computational model has not changed significantly over the past forty years. According to Pollack's rule [2], the increase in performance of a sequential computer is only about the square root of the increase in the number of devices, which implies that doubling the transistor count will lead to a performance improvement of about 40%. Fortunately, the inherent concurrency in a typical embedded application (e.g., automotive electronics, avionics) offers the potential to circumvent Pollack's rule. If an application can be partitioned into a set of nearly autonomous concurrent functions, then a nearly linear performance improvement could be achieved by assigning a dedicated processing element to each of these concurrent functions. This architectural approach is followed in a number of embedded SoCs: to partition the SoC into a set of nearly autonomous possibly heterogeneous Intellectual Property (IP)-blocks or micro components that interact via an appropriate Network-on-a-Chip (NoC) [3].

It is the purpose of the paper to address these challenges by introducing a novel system architecture for SoCs, which offers a component-based design methodology for managing the complexity of billion-of-transistors SoCs through the consequent decoupling of the computational components from the communication infrastructure. The introduced system architecture provides an architectural framework that supports *composability* [4], [5], i.e., the side-effect-free composition of component services (based solely on interface specifications) to form larger systems-of-systems. For this purpose, the computational components are interconnected through a predictable and deterministic time-triggered NoC with inherent fault isolation.

The contributions and key properties of the presented SoC architecture are as follows:

a) ***Elevation of the level of design abstractions****:* In order to manage the complexity of an evolving design at a higher level of abstraction, we must conceptualize components that form stable intermediate forms and exhibit aggregate properties. If we can describe and specify these aggregate properties on their own by an appropriate interface model, then it is not required to understand the structure and the interactions within the components in order to reason about the interactions among components and the emerging system properties. Furthermore, it is then possible to change and enhance the implementation of the components in response to technological developments without a redesign of the system at this higher level of abstraction.

For this reason, we introduce in the proposed SoC architecture the notion of a micro component, which can be considered as a unit of abstraction that provides its functionality at a well-defined message-based network interface to its partners [6]. The clear separation of the processing within a micro component from the interactions among the micro components leads to a communication-centric model that is highly appropriate for many applications.

b) ***Predictability and determinism through encapsulation****:* The SoC architecture offers a predictable on-chip interconnect that is free of interference. Each micro component is assigned dedicated slots in a time-triggered communication schedule, which are protected from other micro components through the communication system. Encapsulation results in a complexity reduction, because the behavior of interfering

subsystems is more difficult to understand and to reason about than the behavior of cleanly encapsulated subsystems. If we want to reduce the cognitive complexity of a design, we have to avoid system mechanisms that increase the cognitive load required for understanding [7]. Also, the test and validation effort for an encapsulated subsystem is smaller than the test effort for interfering systems [8]. Finally, a deterministic behavior of the components is required for the transparent masking of hardware errors by Triple Modular Redundancy (TMR) [9].

c) ***Global time base****:* In general, an SoC cannot be assumed to provide a single clock signal for the entire chip. The reasons why designers introduce multiple clock domains include the handling of clock skew, the clocking down of individual IP blocks as part of power management, or the support for heterogeneous IP blocks with different speeds (e.g., high-clocked special purpose hardware and a slower general purpose CPU). Despite the existence of multiple clock domains, the presented SoC architecture supports a global time base through internal clock synchronization (i.e., within the SoC) and external clock synchronization w.r.t. an SoC-external reference time.

The resulting system-wide global time base allows the temporal coordination of actions on the distributed micro components within an SoC and in an ensemble of different SoCs. Consequently, timestamps assigned at different micro components can be related to each other. Due to the global time base, timestamps are also meaningful outside the micro component where the event has been observed.

d) ***Integrated resource management****:* The proposed SoC architecture supports integrated resource management taking into account requirements w.r.t. communication resources (e.g., bandwidth, latency, latency jitter), computational resources (e.g., dynamic allocation of micro components to application subsystems), and power (e.g., power limiter). Dynamic reconfiguration allows an efficient utilization of mutually exclusive resource demands, if it is a priori known that the worst-case resource consumption in different subsystems cannot occur simultaneously. Furthermore, in case a permanent fault affects only individual micro components, dynamic reconfiguration can relocate the application functionality to spare micro components in order to preserve the specified service of the SoC. Additionally, it builds an important cornerstone for the realization of power-aware systems [10]. Power management is identified as one of the grand challenges in the SIA's semiconductor road map [1].

The paper is structured as follows. Section II is devoted to related work on SoC architectures. The section focuses on related work that also employs NoCs with a Time-Division Multiple Access (TDMA) scheme in analogy to the SoC architecture presented in this paper. Section III gives an overview of the SoC architecture and identifies its constituting elements: the time-triggered NoC, the micro components, the architectural elements for dynamic reconfiguration, and the gateways. Section IV focuses on the encapsulated communication system, which prevents unintended interference between the micro components. The paper finishes with a conclusion in Section V.

II. Related Work

This section describes related work on SoC architectures and the contributions of the proposed architecture compared to existing solutions. We focus on related work that also employs NoCs with a TDMA scheme in analogy to the SoC architecture presented in this paper.

The *Æthereal* architecture [11] and the *Sonics SiliconBackplane μNetwork* [12] provide communication channels with guarantees on minimum bandwidth and maximum latency by using a TDMA scheme that is based on a table with a given number of time slots (e.g., 128 slots). The main difference between the TT-SoC architecture and the above mentioned networks is the level of abstraction at the network interface. Æthereal and the Sonics SiliconBackplane μNetwork provide a shared memory abstraction to the attached IP cores via a transaction based master/slave protocol as it used in OCP [13] or AXI [14]. These protocols define low-level signals like address signals, data signals, interrupt signals, reset signals or clock signals which are typically found at the interfaces of processors, memory subsystems, or bus bridges. In the TT-SoC architecture we have raised the level of abstraction by introducing the notion of a *micro component* which is a self contained computational unit (e.g., a processor or an FPGA with local memory) that provides its functionality over a well defined *message-based* interface. Another important difference lies in the way how, and for which purpose the TDMA scheme is used. The major objective of Æthereal and the Sonics SiliconBackplane μNetwork is to establish resource guarantees with respect to *bandwidth* and *latency*. In addition to guaranteeing bandwidth and bounded latency, the TT-SoC architecture uses the TDMA scheme to schedule periodic send instances of entire *application-level* messages. The supported periods are in the range from a few nanoseconds up to milliseconds or seconds. This allows us to perfectly align the phase of the periodic send instances of application-level messages to the phase of the periodic activation instances of a time-triggered application (e.g., periodic dissemination of a sensor value in a process controll application).

The Cell Broadband Engine Architecture (CBEA) [15] is a microprocessor architecture that has been jointly developed by Sony, Toshiba, and IBM for game, media and broadband systems. It encompasses a general-purpose processing unit and eight specialized and identical processing units. A major difference of the proposed time-tiggered SoC architecture in comparison with the CBEA is the temporal predictability resulting from the time-triggered control of the NoC. The shared communication bus of the CBEA exhibits a variability of the transmission latencies between 79.5 cycles and 939.5 cycles [16]. A further difference is the support for heterogeneous applications. The CBEA encompasses eight identical synergistic processor units, which are optimized for applications based on Single Instruction, Multiple Data (SIMD)

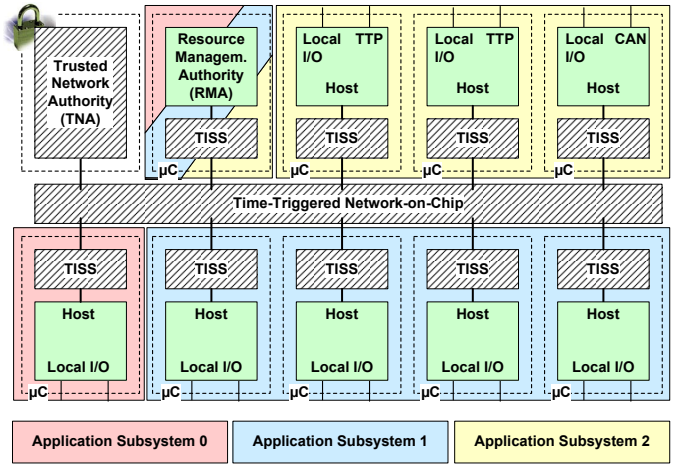

Fig. 1. Structure of Time-Triggered SoC Architecture: *trusted subsystem (shaded) and non-trusted subsystem (hosts of micro components)*

capability. The proposed time-triggered SoC architecture, on the other hand, supports the integration of heterogeneous micro components. For example, a micro component can be a general purpose processor, an FPGA, or special purpose hardware. Individual micro components can also reside within different clock domains, whereas all synergistic processor units of the CBEA belong to the same clock domain.

III. Architecture Overview

The central element of the presented SoC architecture is a time-triggered NoC that interconnects multiple, possibly heterogeneous IP blocks called micro components (see Figure 1). The SoC introduces a *trusted subsystem*, which ensures that a fault (e.g., a software fault) within the host of a micro component cannot lead to a violation of the micro component's temporal interface specification in a way that the communication between other micro components would be disrupted. For this reason, the trusted subsystem prevents a faulty micro component from sending messages during the sending slots of any other micro component.

Furthermore, the time-triggered SoC architecture supports integrated resource management. For this purpose, dedicated architectural elements called the Trusted Network Authority (TNA) and the Resource Management Authority (RMA) accept resource allocation requests from the micro components and reconfigure the SoC, e.g., by dynamically updating the time-triggered communication schedule of the NoC and switching between power modes.

A. Micro Component

The introduced SoC can host multiple application subsystems (possibly of different criticality levels), each of which provides a part of the service of the overall system. An example of an application subsystem in the automotive domain would be a braking subsystem. A nearly autonomous and possibly heterogeneous IP-block, which is used by a particular application subsystem is denoted as a micro component. A micro component is a self-contained computing element, e.g., implemented as a general purpose processor or as special purpose hardware. An application subsystem can be realized on a single micro component or by using a group of possibly heterogeneous micro components (either on one or multiple interconnected SoCs).

The interaction between the micro components of an application subsystem occurs solely through the exchange of messages on the time-triggered NoC. Each micro component is encapsulated, i.e., the behavior of a micro component can neither disrupt the computations nor the communication performed by other micro components. Encapsulation prevents by design temporal interference (e.g., delaying messages or computations in another micro component) and spatial interference (e.g., overwriting a message produced by another micro component). The only manner, in which a faulty micro component can affect other micro components, is by providing faulty input to other micro components of the application subsystem via the sent messages.

Due to encapsulation, the SoC architecture supports the detection and masking of such a failure of a micro component using TMR. Encapsulation is necessary for ensuring the independence of the replicas. Otherwise, a faulty micro component could disrupt communication or communication of the replicas, thus causing common mode failures.

Encapsulation is also a key mechanism for the correctness-by-construction of application subsystems on an SoC. The SoC architecture ensures that upon the incremental integration of micro components, the prior services of the already existing micro components are not invalidated by the new micro components. This property, which is denoted as *composability* [5], is required for the seamless integration of independently developed application subsystems and micro components.

Also, encapsulation is of particular importance for the implementation of SoCs encompassing application subsystems of different criticality levels. In such a mixed criticality system, a failure of micro components of a non safety-critical application subsystem must not cause the failure of application subsystems of higher criticality.

For the purpose of encapsulation, a micro component comprises two parts: a *host* and a so-called *Trusted Interface Subsystem (TISS)*. The host implements the application services. Using the TISS, the time-triggered SoC architecture provides a dedicated architectural element that protects the access to the time-triggered NoC. Each TISS contains a table which stores a priori knowledge concerning the global points in time of all message receptions and transmissions of the respective micro component. Since the table cannot be modified by the host, a design fault or a hardware fault restricted to the host of a micro component cannot affect the exchange of messages by other micro components.

B. Requirements for the Time-Triggered Network-on-a-Chip

The time-triggered NoC interconnects the micro components of an SoC. The purposes of the time-triggered NoC encompass clock synchronization for the establishment of

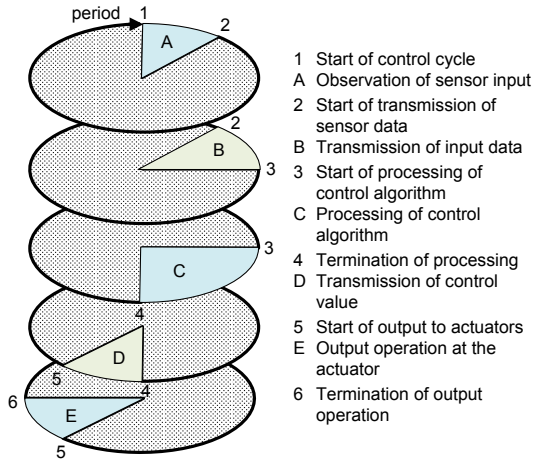

Fig. 2. Temporal Alignment in Control Loops. In this cyclic model of time, the perimeter represents the period of the control application.

a global time base, as well as the predictable transport of periodic and sporadic messages.

*a) **Clock Synchronization**:* The time-triggered NoC performs clock synchronization in order to provide a global time base for all micro components despite the existence of multiple clock domains. The resulting system-wide global time base allows the temporal coordination of actions on the distributed micro components within an SoC and in an ensemble of different SoCs. It is based on a 64 bit wide binary time format, which has been standardized by the OMG in the smart transducer interface standard [17].

*b) **Predictable Transport of Messages**:* Using TDMA, the available bandwidth of the NoC is divided into periodic conflict-free sending slots. We distinguish between two utilizations of a periodic time-triggered sending slot by a micro component. A sending slot can be used for the *periodic transmission of messages* or the *sporadic transmission of messages*. In the latter case, a message is only sent if the sender must transmit a new event to the receiver.

The allocation of sending slots of the time-triggered NoC to micro components occurs using a communication primitive called *pulsed data stream* [18]. A pulsed data stream is a *time-triggered periodic unidirectional data stream* that transports data in pulses with a defined length from *one* sender to *n* a priori identified receivers at a specified phase of every cycle of a periodic control system.

The pulsed behavior of the communication network enables the efficient transmission of large data in applications requiring a temporal alignment of sender and receiver. Temporal alignment of sender and receiver is required in applications where a short latency between sender and receiver is demanded. This is typical for many real-time systems. For example, consider a control loop realized by three micro components performing sensor data acquisition (A), processing of the control algorithm (C), and actuator operating (E) as it is schematically depicted in Figure 2. In this application, temporal alignment between sensor data transmission (B) and the start of the processing of the control algorithm (cf. instant 3 in Figure 2) as well as between the transmission of the control value (D) and the start of actuator output (cf. instant 5) is vital to reduce the end-to-end latency of the control loop, which is an important quality characteristic of many real-time systems. By specifying two pulsed data streams corresponding to (B) and (D) in Figure 2, efficient, temporally aligned data transmission can be achieved.

Contrary to the on-chip communication system of the introduced SoC architecture, many existing NoCs provide only a guaranteed bandwidth to the individual senders without support for temporal alignment. The resulting consequences are: (i) either the short latency cannot be guaranteed, (ii) a high bandwidth has to be granted to the sender throughout the entire period of the control cycle, although it is only required for a short interval, or (iii) the communication system has to be periodically reconfigured in order to free and re-allocate the non-used communication resources.

A pulsed data stream is specified using the 3-tuple $\langle$pulse period, pulse phase, duration$\rangle$ (cf. Figure 3). These three parameters determine the allocation of TDMA slots to the micro component that sends the pulsed data stream. A pulsed data stream consists of periodic *pulses* with a defined *pulse period* and a defined *pulse phase*. Our design restricts the pulse periods of a pulsed data stream to 32 different periods corresponding to negative powers of two of the second, i.e., a period can be 1 second, 1/2 second, 1/4 second, 1/8 second and so forth. This restriction is introduced in order to reduce the complexity of the NoC and the computation of the time-triggered schedule significantly. The pulse phase, denotes the offset to the start instant of the pulse from the start of the pulse's period.

A pulse consists of at least one *fragment* of variable size. Successive fragments of one pulse are not required to be transmitted in a dense sequence on the NoC, i.e., fragments of one pulse can be interleaved by fragments of other pulses. The time between the instant of transmission of the first fragment and the instant of transmission of the last fragment of a pulse is denoted as the *duration* of the pulse. Each fragment is further decomposed into a set of atomic (fix-sized) *flits*. A flit is the basic entity that can be transmitted over the time-triggered NoC and occupies one TMDA slot. However, the fragmentation of pulses into fragments, as well as the further fragmentation of fragments into flits is abstracted from the host within the TISS. This reduces the complexity of the application design, as it must only be dealt with periodic pulses of pulsed data streams.

For an exemplary pulsed data stream, Figure 3 depicts the allocation of TDMA slots, which are occupied by the flits of variable sized fragments of a pulse.

C. Architectural Elements for Resource Management

The purpose of the integrated resource management in the SoC architecture is to dynamically assign computational resources (i.e., micro components) to application subsystems

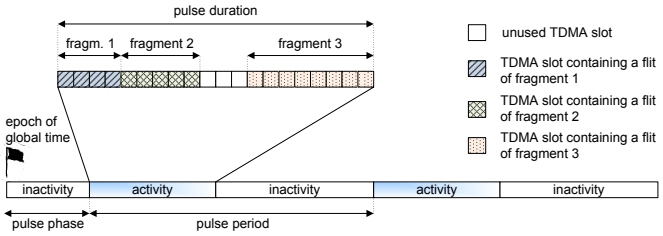

Fig. 3. Pulsed Data Stream

and to grant communication resources and power to individual micro components.

We distinguish two fundamentally different types of application subsystems: *safety-critical* and *non safety-critical* application subsystems. While safety-critical application subsystems need to be certified to the highest criticality classes (e.g., class A according to DO-178B), non safety-critical applications subsystems, on the other hand, do not have such stringent certification requirements. In general, these two types of applications subsystems will involve fundamentally different design paradigms. The focus of safety-critical applications lies on simplicity and determinism in order to facilitate thorough verification and validation. In contrast, non safety-critical applications can provide more complex application services (e.g., need to deal with insufficient a priori knowledge about the environment) and dynamism to handle the challenges of evolving application scenarios and changing environments.

The TTSoC architecture follows this bivalent distinction of application subsystems and provides two distinct architectural elements for enabling integrated resource management, namely the Trusted Network Authority (TNA) and the Resource Management Authority (RMA). The RMA computes new resource allocations for the non safety-critical application subsystems, while the TNA ensures that the new resource allocations have no adverse effect on the behavior of the safety-critical application subsystems. As depicted in Figure 1 the TNA is part of the trusted subsystem of the SoC, whereas the RMA is not. By splitting the entire resource management into two separate parts, where only one is part of the trusted subsystem, the certification of the time-triggered SoC is significantly simplified, since the checking of the correctness of a resource allocation through the TNA is significantly simpler than its generation at the RMA.

*c) **Resource Management Authority (RMA)**:* The RMA is responsible for scheduling the available resources and creating the allocation of the micro components. For this purpose the RMA exploits application-specific knowledge (e.g., communication topology) and system knowledge (e.g., temporal properties of the time-triggered NoC). However, the RMA is not able to change the configuration of the SoC directly, i.e., to update the configuration of the affected TISSs.

*d) **Trusted Network Authority (TNA)**:* The TNA acts as a guard for the reconfiguration activities performed by the RMA. Therefore, the TNA detects potential collisions on the time-triggered NoC or violations on resource reservations. If an erroneous resource schedule is detected, the current configuration remains unchanged and the new schedule is rejected. In case the schedule is correct, the TNA updates the configuration of the micro components. Since the TNA is part of the trusted subsystem it is permitted to reconfigure the micro components via the TISSs.

As a first step, we are focusing on dynamic resource management of communication resources, where the topology and Quality of Service (QoS) parameters (e.g., latency and bandwidth) of pulsed data streams are subject of reconfiguration. The topology of a particular pulsed data stream is determined by the set of TISSs, which processes the pulsed data stream. The QoS parameters of a pulsed data stream are affected by its pulse period, phase and duration.

D. Gateways

The proposed SoC architecture supports gateways for accessing chip-external networks (e.g., TTP [19], TTE [20] and CAN [21] in Figure 1). The benefits of gateways include the interoperability with public networks, such as the Internet, and the ability to interconnect multiple SoCs to a distributed system. The realization of a distributed system enables applications based on the SoC architecture for ultra-dependable systems. In ultra-dependable systems, a maximum failure rate of 10^{-9} critical failures per hour is demanded [22]. Today's technology does not support the manufacturing of chips with failure rates low enough to meet these reliability requirements. Since components failure rates are usually in the order of 10^{-5} to 10^{-6} (e.g., [23] uses a large statistical basis and reports 100 to 500 failures out of 1 Million Electronic Control Units (ECUs) in 10 years), ultra-dependable applications require the system as a whole to be more reliable than any one of its components. This can only be achieved by utilizing fault-tolerant strategies that enable the continued operation of the system in the presence of component failures.

In case the chip-external network is also time-triggered (e.g., TTP [19], TTE [20]), the TDMA scheme of the NoC can be synchronized with the TDMA scheme of the chip-external network. The periods and phases of the relayed pulsed data streams on the NoC can be aligned with the transmission start instants of the messages on the time-triggered chip-external network. Consequently, a message that is sent on the chip-external network is delivered to the micro components within a bounded delay with minimum jitter (only depending on the granularity of the global time base). The alignment between pulsed data streams and messages on time-triggered networks ensures that replicated SoCs perceive a message at the same time, i.e., within the same inactivity interval of the global sparse time base [24]. This property is significant for achieving replica determinism [9] as required for active redundancy based on exact voting. Without synchronization between the NoC and the chip-external network, there could always occur a scenario in which one SoC forwards the message to the micro components in one period of the pulsed data stream, while another SoC would forward the message in the next period.

Furthermore, the introduced gateways provide the SoC with an externally synchronized time base. For example, the global time base of the SoC can be synchronized to GPS. Consequently, a timestamp assigned to an event is also meaningful outside the SoC. Furthermore, the global time base enables a global coordination of activities spanning multiple SoCs (e.g., output to actuators at same global point in time).

IV. Encapsulated Communication Service

As mentioned already in the introduction, one major objective of the time-triggered SoC architecture is to facilitate independent development of subsystems by the use of encapsulation mechanisms that prevent any unindented interference between these subsystems. On micro component level, encapsulation is naturally achieved by the physical separation of the individual micro components. On the next higher level where subsystems are formed by multiple micro components interacting which each other via the exchange of messages, encapsulation is required with respect to the communication infrastructure that interconnects these micro components. For this purpose, the time-triggered SoC architecture provides so-called *encapsulated communication channels*.

A. Encapsulated Communication Channels

The term *encapsulated communication channel* denotes an unidirectional channel that transports messages at predefined points in time from a single source to one or more destinations. A message corresponds to one pulse in a pulsed data stream.

The endpoints of an encapsulated communication channel are called *ports*. We distinguish between output ports which are located at the source - where the messages are produced - and input ports which are located at the destinations where the messages are consumed. A single micro component can be attached to multiple encapsulated communication channels, and thus can have multiple input and output ports. The topology of an encapsulated communication channel is defined by the number of destinations (i.e. the number of input ports) and by the assignment of the source and the destinations to specific micro components. Since the number of destinations of an encapsulated communication channel is variable, *single-cast*, *multi-cast* and *broad-cast* topologies are supported.

In order to prevent any unintended interference between subsystems, the time-triggered SoC architecture ensures *temporal and spatial partitioning* [25] with respect to encapsulated communication channels. Communication activities in a given encapsulated communication channel are neither visible nor have any effect (e.g. performance penalty) on the exchange of messages in any other encapsulated communication channel. It is guaranteed, that the only micro component that can send messages over a given encapsulated communication channel is the micro component that is defined as the source of that encapsulated communication channel (i.e. the micro component where the output port of the encapsulated communication channel is located).

Temporal and spatial partitioning of encapsulated communication channels is enforced by the TISS which is located between the host of a micro component and the time-triggered NoC (see Figure 1). The TISS acts as a guardian for the shared time-triggered NoC by accessing the time-triggered NoC exclusively at a priori known points in time according to the TDMA scheme. Therefore, a specific pulsed data stream, which defines a set of TDMA send slots, is assigned to each encapsulated communication channel. The TISS of each micro component incorporates its own dedicated time-triggered message schedule - the so-called *Message Descriptor List (MEDL)* - which holds the information about the periodic points in time when a fragment of a given pulsed data stream should be sent or received by the TISS.

In order to guarantee that the encapsulation properties of the communication service are not violated in the presence of a design fault or a hardware fault within the host, the implementation of the TISS ensures, that the host cannot alter the internal time-triggered schedule of the TISS. As already mentioned in Section III the TISS itself is part of the trusted subsystem and is considered to by free of design faults.

V. Conclusion

In order to manage the complexity of an evolving design at a higher level of abstraction, we must conceptualize components that form stable intermediate forms and exhibit aggregate properties. If we can describe and specify these aggregate properties on their own by an appropriate interface model, then it is not required to understand the structure and the interactions within the components in order to reason about the interactions among components and the emerging system properties. Furthermore it is then possible to change and enhance the implementation of the components in response to technological developments without a redesign of the system at this higher level of abstraction.

For this reason, we have introduced in the presented SoC architecture the notion of a micro component, which can be considered a unit of abstraction that provides its functionality at a well-defined message-based network interface to its partners [6]. The clear separation of the processing within a micro component from the interactions among the micro components leads to a communication-centric model that is highly appropriate for many applications and thus can lead to a substantial reduction of the complexity of a design at the system level.

The contributions of the proposed time-triggered SoC architecture include the realization of a predictable and deterministic on-chip network with inherent fault isolation and a global time base. The proposed solution is fundamentally different to the prevalent trend of asynchronous on-chip interconnects. Also, the proposed Time-Triggered System-on-a-Chip (TTSoC) architecture provides significant contributions compared to existing synchronous on-chip interconnects (e.g., Æthereal, Sonics). The TTSoC architecture uses a TDMA scheme to schedule periodic send instances of entire application-level messages. The supported periods are in the range from a few nanoseconds up to milliseconds or seconds. This allows us to perfectly align the phase of the

periodic send instances of application-level messages to the phase of the periodic activation instances of a time-triggered application (e.g., periodic dissemination of a sensor value in a process controll application). In addition, a global time base is provided at the application level in order to facilitate the temporal coordination of subsystems distributed across multiple micro components. The global time base, which is internally synchronized (between the micro components of an SoC with different clock domains) and externally synchronized (with the SoC environment).

With its inherent fault isolation capabilities, the time-triggered SoC is an effective solution for mixed criticality systems. A mixed criticality system is characterized by the coexistence of safety-critical micro components (e.g., X-by-wire functionality, active safety-functions in a car) and micro components with a lower level of criticality (e.g., multimedia, comfort functionality in a car). The trusted subsystem (i.e., TISSs, time-triggered NoC, TNA) ensures that a design fault (e.g., a software fault) within a given micro component cannot lead to a violation of the micro component's temporal interface specification in a way that the communication between other micro components would be disrupted. It is prevented by design that a faulty micro component sends messages during the sending slots of any other micro component.

ACKNOWLEDGMENT

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527 and the FIT-IT (Research Programme initiatiated by the Austrian Federal Ministry of Transport, Innovation, and Technology (BMVIT)) project TT-SoC under grant number 813299/7852.

REFERENCES

[1] Semiconductor Industry Association (SIA). International technology roadmap for semiconductors. Technical report, 2005.

[2] P. Gelsinger. Microprocessors for the new millenium, challenges,opportunities, and new frontiers. In *Proc. of the Solid State Circuit Conference*. IEEE Press, 2001.

[3] W. Wolf. The future of multiprocessors systems on chips. In *Proc. of the Design Automation conference*. IEEE Press, 2004.

[4] H. Kopetz and R. Obermaisser. Temporal composability. *Computing & Control Engineering Journal*, 13:156–162, August 2002.

[5] J. Sifakis. A framework for component-based construction. In *Proc. of 3rd IEEE Int. Conference on Software Engineering and Formal Methods (SEFM05)*, pages 293–300, September 2005.

[6] C. Jones et al. DSOS conceptual model. Technical Report Techn. Report CS-TR-782, University of Newcastle, 2003.

[7] P.J. Feltovic, R. Coulson, and R. Spiro. Learners' misunderstandidng of important and difficult concepts. *Smart Machines in Education*, pages 354–380, 2001. AAAI Press.

[8] J. Owens. GPUs: Engines for future high-performance computing. Technical report, Lincoln Labs, Boston, September 2004.

[9] S. Poledna. Replica determinism in distributed real-time systems: A brief survey. *Real-Time Systems*, 6:289–316, 1994.

[10] O.S. Unsal and I. Koren. System-level power-aware design techniques in real-time systems. *Proceedings of the IEEE*, 91(7):1055–1069, 2003.

[11] K. Goossens, J. Dielissen, and A. Radulescu. The aethereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):414–421, 2005.

[12] Sonics. Sonics unetwork technical overview (www.sonicsinc.com), 2002.

[13] OCP-IP Association. Open core protocol specification 2.1, 2005.

[14] ARM. Axi protocol specification, 2004.

[15] IBM, Sony, and Toshiba. Cell broadband engine architecture. Technical report, October 2006.

[16] T.W. Ainsworth amd T.M. Pinkston. On characterizing performance of the cell broadband engine element interconnect bus. In *Proc. of the First International Symposium on Networks-on-Chip*, pages 18–29, May 2007.

[17] OMG. Smart Transducers Interface. Specification ptc/2002-05-01, Object Management Group, May 2002. Available at `http://www.omg.org/`.

[18] H. Kopetz. Pulsed data streams. In *IFIP TC 10 Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006)*, pages 105–124, Braga, Portugal, October 2006. Springer.

[19] H. Kopetz and G. Grünsteidl. TTP – a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, January 1994. Vienna University of Technology, Real-Time Systems Group.

[20] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer. The Time-Triggered Ethernet (TTE) design. *Proc. of 8th IEEE Int. Symposium on Object-oriented Real-time distributed Computing (ISORC)*, May 2005.

[21] Robert Bosch Gmbh, Stuttgart, Germany. *CAN Specification, Version 2.0*, 1991.

[22] N. Suri, C.J. Walter, and M.M. Hugue. *Advances In Ultra-Dependable Distributed Systems*, chapter 1. IEEE Computer Society Press, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264, 1995.

[23] B. Pauli, A. Meyna, and P. Heitmann. Reliability of electronic components and control units in motor vehicle applications. In *VDI Berichte 1415, Electronic Systems for Vehicles*, pages 1009–1024. Verein Deutscher Ingenieure, 1998.

[24] H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *Proc. of 12th Int. Conference on Distributed Computing Systems*, Japan, June 1992.

[25] J. Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999.