A System–on–a–Chip Platform for Mixed–Criticality Applications

Armin Wasicek, Christian El–Salloum and Hermann Kopetz
Institute for Computer Engineering
Vienna University of Technology
Vienna, Austria
Email: {armin, salloum, hk}@vmars.tuwien.ac.at

Abstract—High–integrity systems are deployed in order to realize safety–critical applications. To meet the rigorous requirements in this domain, these systems require a sophisticated approach to design, verification, and certification. Not only safety considerations have an impact on a product’s overall dependability, but also security has to be taken into account. In this paper we analyze the Time–Triggered System–on–Chip (TTSoC) architecture, which is a novel architecture for Multi–Processor System–on–Chip (MPSoC) devices, regarding its security properties. We discuss essential compliance criteria to the Multiple Independent Layers of Security (MILS) architecture, which is an industry–ready architecture for embedded high–integrity systems. We found that both architectures share intrinsic properties and we are able to show that the TTSoC architecture implements the core requirements of a MILS Separation Kernel and thus realizes its elementary security policies by design.

Index Terms—Mixed–criticality systems, multi–level security architecture, MILS, TTSoC

I. INTRODUCTION

Security in safety-critical embedded systems is a major topic since system failures can threaten human life, cause ecological catastrophes or result in financial loss. Examples are digital control systems in the automotive and avionics domain, process control systems for any kinds of industrial plants (e.g., nuclear power plants) as well as devices employed for personal health care (e.g., heart pacemakers, drug delivery and patient monitoring systems).

In many systems security is implemented in an ad–hoc manner which is very error–prone. Therefore it’s commonly agreed that security should be ideally implemented by design. This relieves application developers as much as possible from dealing with implementation specific details of the employed security mechanisms thereby effectively reducing the number of vulnerabilities. Security–by–design can be enforced by an architectural framework that guides and restricts designers in a way that resulting systems achieve certain non–functional properties. Usually, such a framework is accompanied by a platform and a set of verified architectural services that can be easily instantiated and that enable the designer to focus in the actual application.

The state–of–the–art approach to incorporate security services into a platform (e.g., an operating system) is to provide a Trusted Computing Base (TCB). According to Lampson a TCB is a small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security [1]. The services of the TCB should guarantee that the damage that any misbehaving component outside the TCB (e.g., a compromised device driver) can do, is limited according to the privileges that where granted to the component in the first place in accordance to the security policy. The TCB can be considered as a single partition, which means that it does not contain any mechanisms for protecting its subcomponents from each other. A single bug or vulnerability in one of the TCB’s subcomponents can corrupt the correct operation of the entire TCB. Consequently, the TCB has to be verified, and in the case of a safety–critical system certified as a whole. Due to the complexity involved with the exhaustive examination of large pieces of code and hardware, the size and thus also the functionality of the TCB is limited.

An approach to manage the complexity involved in verifying a large set of security functions is the Multiple Independant Layers of Security (MILS) architecture, which is based on the concept of separation of subsystems as introduced by Rushby [2], [3]. The basic idea of the MILS architecture is to provide a small and verifiable Separation Kernel that isolates processes in separate partitions on a shared processor. The availability of separated partitions enables the design of systems where application and security functions are developed in manageable units. The individual units can be analyzed separately which reduces dramatically the complexity compared to conventional designs without separation.

The approach of using software to enforce separation is employed in many modern operating systems and hypervisors (e.g., virtual memory). Since such software is the heart of the system, there will always be attackers who try to find weaknesses like buffer overflow vulnerabilities in operating systems or hypervisors. By exploiting such vulnerabilities, an attacker can potentially take control over the operating system and thus gain the highest privileges in the system. From that point on, all other protection mechanisms can be circumvented. The same argument holds for hypervisors. The nature of this problem comes from the fact, that the isolation mechanisms rely on software and that software is prone to be manipulated.

In addition, separating processes on a shared processor faces challenges that have to be solved regarding potential
side-effects. For instance, one process might interfere with the cache of the subsequently scheduled process. This will cause additional cache misses for the subsequently scheduled process, which has a negative effect on its execution time. Another example of such a side-effect is a process that heats up the core by frequently using instructions that consume a high amount of energy. This behavior can cause a modern core with built-in energy management mechanisms to degrade its performance (e.g., dynamic frequency scaling or CPU throttling). Due to the degraded performance, the subsequently scheduled processes will have a longer execution time. In general, such effects are very difficult to prevent or estimate.

In this paper we describe a System-on-Chip (SoC) architecture, namely the Time-Triggered System-on-Chip (TTSoC) architecture, which provides process isolation mechanisms realized in hardware. It is guaranteed by design, that the software executed on the cores of the SoC cannot override the isolation mechanisms. A major innovation of our approach is the one-to-one mapping of separated partitions to the IP cores of an SoC. Each partition is mapped to exactly one IP core and each IP core hosts at most one partition. Compared to shared single-core solutions, the allocation of partitions to physically separated IP cores eases the implementation of inter-partition isolation mechanisms by eliminating the typical side-effects of shared-core solutions as mentioned above.

It is the main contribution of this paper to show, that the TTSoC architecture supports the four fundamental security functionalities identified by the MILS architecture (see Section III). In essence, we will argue that the TTSoC architecture can be viewed as a realization of the MILS Separation Kernel in hardware. The realization of security-relevant mechanisms in hardware will harden a system against many kinds of software-based attacks.

The remainder of the paper is structured as follows: Section II gives an overview over the TTSoC architecture and Section III describes the concept of MILS. In Section IV we show how the properties of the TTSoC architecture map to the fundamental security functionalities of the MILS architecture, and Section IV-B depicts an exemplary mixed-criticality application based on our approach. Section VI concludes the paper.

II. TIME-TRIGGERED SYSTEM-ON-A-CHIP (TTSoC) ARCHITECTURE

This section gives an overview of the TTSoC Architecture [4] and its generic architectural services. The TTSoC architecture follows the architectural style of the FP7 project Generic Embedded Systems (GENESYS\(^1\)) which developed a blueprint of an European cross-domain reference architecture for embedded systems satisfying the requirements collected in the ARTEMIS\(^2\) Strategic Research Agenda. There exist several running prototype implementation of the TTSoC architecture where the latest realized on a Stratix III FPGA [5], [6], [7].

---

\(^1\)http://www.genesys-platform.eu/
\(^2\)https://www.artemis-ju.eu/

---

A. General Overview

The TTSoC Architecture is the realization of a predictable and deterministic SoC architecture for embedded systems with inherent fault isolation and a system-wide global time base. Single Intellectual Property (IP) cores called micro components are assembled around a dedicated time-triggered Network-on-a-Chip (NoC) [6] which establishes a global time base and enables a temporal firewall interface [8], [9] for read and write accesses to the network. In addition to micro components for the application the architecture employs dedicated infrastructural IP cores for reconfiguration (Trusted Network Authority (TNA)), resource management (Resource Management Authority (RMA)), diagnosis (Diagnostic Unit (DU)), and for the interconnection of the NoC to off-chip networks (gateways) as depicted in Figure 1.

---

Fig. 1. Time-Triggered System-on-a-Chip Architecture

A key objective of the TTSoC architecture is to facilitate independent development of application subsystems and the integration of subsystems with mixed criticality levels. This is accomplished by the use of encapsulation mechanisms that prevent any unintended interference between these subsystems.

B. Encapsulation in the Host

Since the time-triggered NoC is a shared resource, it has to be prevented that a design fault (e.g. a software fault) within a given micro component can lead to a violation of the micro component’s temporal interface specification in a way that the communication between other micro components can be disrupted. In order to prevent any temporal interference (e.g., delaying messages of other micro components) or spatial interference (e.g., overwriting a message produced by another micro component), a micro component is structured into two architectural elements, the host and the Trusted Interface Subsystem (TISS).

The host performs the computations that are required to deliver the intended service of a micro component. It can belong to different criticality levels, can be developed by different suppliers, and is generally not assumed to be free of design faults.

The TISS is certified to the highest criticality level of any application within the SoC. The certification of the TISS is facilitated (a) by the simplicity of the TISSs’ design and (b)
by the fact that a TISS can be certified once and reused for all micro components in different designs. It provides a stable and validated set of core services to the host via a Uniform Network Interface (UNI), and acts as a guardian for the shared time-triggered NoC by accessing it exclusively at a priori known points in time according to the Time Division Multiple Access (TDMA) scheme. The time-triggered schedule cannot be changed by the host, but exclusively programmed by the TNA. The physical separation of the micro components and the guardian functionality of TISS ensure, that a host constitutes a Fault Containment Region (FCR) for accidental and malicious faults. An FCR is a collection of components that operates correctly regardless of any arbitrary logical or electrical fault outside the region [10].

C. Encapsulated Communication Channels

In the TTSoC architecture micro components can interact exclusively via encapsulated communication channels. The term encapsulated communication channel denotes a unidirectional channel that transports messages at predefined points in time from a single source to one or more destinations. The endpoints of an encapsulated communication channel are called ports. We distinguish between output ports which are located at the source—where messages are produced—and input ports which are located at the destinations where 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, singlecast, multicast, and broadcast topologies are supported. Figure 2 depicts three exemplary encapsulated communication channels, each having a different type of topology.

The TTSoC architecture ensures temporal and spatial partitioning with respect to encapsulated communication channels in order to prevent any unintended interference between application subsystems. 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).

Encapsulation is established by the TISS, which acts as a guardian for the shared time-triggered NoC by accessing it exclusively at a priori known points in time according to the TDMA scheme. The implementation of the TISS ensures that the host cannot alter the internal time-triggered schedule of 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 time-triggered schedule in the TISS can only be modified by the TNA.

The TNA accepts configurations only from an authorized source, either statically by the system integrator (e.g., one time programmable memory) or dynamically via digitally signed configuration messages.

In [11] we proposed a mechanism to realize bulk encryption to add the property of confidentiality to an encapsulated communication channel. In [12] we present the design for a gateway that extends the concept of encapsulated communication channels over NoC boundaries. In this manner, a System-of-Systems (SoS) [13] spanning multiple SoCs can be implemented.

III. THE MULTIPLE LAYERS OF SECURITY (MILS) ARCHITECTURE

In this section we give an overview of the MILS architecture [14] [15] which was developed to resolve the difficulty of certification of high integrity systems with respect to security. Its mechanisms are closely related to the 'robust partitioning' facilitated in the Integrated Modular Avionics (IMA) [16] and to the sandboxing [17] technique implemented in some secure systems. In contrast to the IMA certification process, the MILS architecture follows a compositional approach to assurance, evaluation, and certification of a MILS-based system: The overall security of a system can be determined largely based on the evaluations of the single components. This is a very ambitious goal.

The MILS security architecture (see Figure 3) isolates processes in separate partitions. A partition is a collection of
data objects, code, and system resources. In order to enforce this partitioning, the MILS architecture is divided into three layers, the Partitioner Layer, the Middleware Layer and the Application Layer.

A. Partitioner Layer

The Partitioner Layer is realized by the separation kernel which is a relatively small piece of code which is rigorously checked and trusted. It guarantees the separation of temporal and spatial partitioning. In particular, the separation kernel provides the four fundamental security functionalities:

1) **Data Isolation** ensures that the memory address spaces of a partition are completely independent of any other partition. Data isolation prevents *exfiltration* which means that any access to a memory location in a given partition must not affect the state of any other partition, and *infiltration* which means that the execution in a given partition must not be affected by the state of any other partition.

2) **Information Flow** This requirement modifies the data separation mentioned above, by allowing partitions to communicate with each other in a controlled way by authorized communication channels. The architecture ensures, that these authorized communication channels are the only way how pure data separation can be violated.

3) **Sanitization** The processor itself will not be a *covert channel* to leak classified data to unclassified processes as the processor moves from partition to partition. This is usually accomplished by cleaning any shared resources (e.g., registers) before a process in a new partition can use them.

4) **Damage Limitation** A failure in one partition will be contained within one partition and it will not propagate to another partition.

B. Middleware and Application Layer

The Middleware layer comprehends all the required device drivers, operating system functionality, and a communication system. An important requirement for the communication system is to provide separate channels in order to extend the protected environment established by the Separation Kernel to multiple nodes. The third layer is the Application layer. The applications are themselves responsible for enforcing application layer security policies.

IV. THE TTSoC ARCHITECTURE AS A HARDWARE PLATFORM FOR MILS

In this section we will show how the MILS architecture is mapped to the TTSoC architecture, and in particular how the basic architectural services of the TTSoC architecture support the four fundamental security functionalities of the MILS separation kernel (see Section III).

A. Partitioner Layer

We use the TTSoC architecture as a hardware platform for the MILS architecture by employing a one-to-one mapping of MILS partitions to TTSoC micro components. Each partition is mapped to exactly one micro component and each micro component hosts at most one partition. The mapping of partitions to physically separated micro components and the encapsulated communication channels of the TTSoC realize the four fundamental security functionalities of the MILS separation kernel in hardware:

1) **Data Isolation** Data isolation ensures that each partition has an independent address space. This property is inherently supported by the TTSoC architecture since the micro components are physically separated and each micro component has its own local memory. In our current approach we assume that the local memory of the micro components will be sufficient to execute the applications and middleware services running in their partition. In many embedded applications this assumption is valid (e.g., control algorithms with a small memory footprint), but we are aware that there are applications where providing the required amount of on-chip memory would not be feasible. Therefore, a Memory Management Unit (MMU) that enables secure and deterministic access to off-chip memory via the time-triggered NoC is a current research topic at our department.

2) **Information Flow** The TTSoC architecture supports this property via the encapsulated communication channels, which can be used to enforce a strict policy regulating which parts of an SoC can communicate with the other parts in a given direction of information flow. The encapsulated communication channels are established by the TNA. From an abstract point of view, the TNA acts like an operator in a traditional telephone exchange system, where the local terminations of telephone lines where connected by patch chords to establish a connection. In the case of the TTSoC architecture, the TNA connects the output ports of the sending micro components to the input ports of the receiving micro components. Thereby it ensures —as a trusted component—that the encapsulated communication channels are routed only in such a way that security violations do not occur. The routing information of the encapsulated communication channels is stored within the TISSs of the micro components and can be exclusively (re-)configured by the TNA (The host of a micro component has neither access to the time-triggered message schedule stored in the TISS nor to the routing information). Thus, no other—potentially malicious—component in the system can interfere with the routes that have been set up by the TNA.
a micro component. Each input and output port of an encapsulated channel is mapped to a dedicated memory region within the micro component. This mapping can be exclusively configured by the micro component’s host. Thus, the host can decide which memory region should be accessible for a given port and the associated encapsulated communication channel. Memory regions that are associated with an input port can only be written by the source of the corresponding encapsulated communication channel, while memory regions that are associated with an output port can only be read by every destination of the corresponding encapsulated communication channel.

To summarize the TTSoC architecture supports regulation of information flow on two levels: (i) The TNA arranges the encapsulated communication channels between the micro components, and (ii) the individual hosts of the micro components decide to which memory locations (and thus to which information) the ports of the communication channel should be mapped to.

3) **Sanitization** Due to the fact that each micro component hosts at most one partition, sanitization is not an issue in the TTSoC architecture, as long as the partitions are statically assigned to the micro components. If dynamic reconfiguration with respect to the partition-to-micro component mapping is required (e.g., this may be required in some scenarios to increase hardware efficiency and flexibility of the system) sanitization mechanisms would be required that are securely triggered by TNA.

4) **Damage Limitation** Damage limitation, was one of the key aspects in the design of the TTSoC architecture. As mentioned in Section II, the hosts of a micro component constitute FCRs for accidental as well as for malicious faults. Since the fault containment is based on the physical separation of the micro components, the verification of complex hardware and software mechanisms for fault isolation—as they are employed for example in shared single-core solutions—can be eliminated.

**B. Middleware and Application Layer**

In the TTSoC architecture third party suppliers should be unrestrictedly allowed to provide middleware security services. Of course these services can not always be assumed to be free of design faults (e.g., there can be vulnerabilities). Therefore, non-interference and fault containment with respect to middleware services is a key issue.

Our approach is to realize the middleware layer together with the application layer within the host. Since a host constitutes an FCR, a design fault or any compromised software within the host can only affect the local host but cannot disrupt the correct computation or communication of any other micro components. Thus, it is sufficient to certify a middleware according to the criticality level of the micro components in which it will be employed (the criticality of a micro component is derived by the criticality of the subsystem to which the partition hosted on the micro component belongs to). This property is of major significance for the integration of application subsystems with mixed criticality levels.

**V. EXEMPLARY MIXED-CRITICALITY APPLICATION**

In this section we give an example, how the TTSoC architecture can be used as a platform for mixed-criticality embedded applications. Figure 4 depicts a possible TTSoC instantiation with eight micro components. The system consists of three subsystems with different criticality levels: (i) an unclassified application, (ii) a secret application (S), and (iii) a top secret application (TS).

![Exemplary Mixed-Criticality Application](image)

In addition to the micro components that execute the actual applications, there are five special purpose micro components, namely the TNA which allocates the encapsulated communication channels on the time-triggered NoC, the crypto component which performs encryption of sensitive data before it leaves the chip, the network I/O component which connects the chip to an off-chip network, and two console components which interface to the terminals for the unclassified and the top secret application.

With respect to security, the TNA and the crypto component have to be verified—and if necessary certified—according to the criticality-level of the most critical subsystem on the chip, which is in our example the top secret application. Due to the encapsulation properties of the TTSoC architecture, this is not required for the other special purpose micro components.

The example depicts among other features, how the Bell–LaPadula Model (BLP) [18] can be easily implemented with the TTSoC architecture by means of encapsulated communication channels. The BLP forbids any read-up, which means that an object at a given security level may not read an object at a higher security level and forbids any write down which means that an object at a given security level must not write to any object at a lower security level.

Within the SoC, the top secret application adheres fully to the BLP. A single unidirectional channel leads from Console 2 to the top secret application. Thus, no information flow

---

3We consider the network I/O component as part of the off–chip network.
from the top secret application to the console (i.e., no read-up and no write-down) is possible. Furthermore, the data produced by the top secret channel can leave the chip only by passing the crypto component, because there is no other output channel for the top-secret application. The crypto component—which is trusted to the highest criticality level in the SoC—encrypts the data and forwards it to the network I/O component. Because the the network I/O component receives only encrypted or unclassified data, it must not be verified to the highest criticality level.

The secret application implements the BLP only partially. There is one unidirectional channel from the unclassified application to the secret application which means that regarding, these two applications in isolation there is also no read-up and no write-down. However, the BLP is violated since encapsulated communication channels are spanned in a way that allows bidirectional communication between the secret application and the Console 1 which has a lower criticality-level. In the example this violation is justified by a flow control middleware within the partition of the secret application which assures that only non-sensitive data is transmitted to the console. Since the flow control middleware resides in the partition (i.e. FCR) of the secret application, a design fault within the middleware can only cause vulnerabilities with respect to that partition (e.g., sensitive data of the secret application could be displayed on the console) and cannot interfere with the rest of the system (e.g., the top secret application).

The unclassified application incorporates no flow control middleware and can write directly to the network I/O component, bypassing the crypto component.

The TNA instantiates the encapsulated channels interconnecting the micro components, and thus enforces the employed security policies. It is guaranteed by design, that the TNA accepts only authenticated configuration data that cannot by changed by any other micro component in the system. It is the responsibility of the system integrator, to configure the TNA in a way that the system specific security policies are enforced.

Table I analyzes the impact on the overall system security, if single architectural element would be compromised.

<table>
<thead>
<tr>
<th>Element</th>
<th>Impact of corruption</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNA or on-chip network</td>
<td>The attacker can reroute the channels. Secret and top secret information can be disclosed.</td>
</tr>
<tr>
<td>Top secret Application</td>
<td>Integrity of top secret information can be violated, but confidentiality is still maintained.</td>
</tr>
<tr>
<td>Secret Application</td>
<td>Integrity of secret information can be violated, but confidentiality is still maintained.</td>
</tr>
<tr>
<td>Flow Control Middleware</td>
<td>Integrity and confidentiality of secret information can be violated.</td>
</tr>
<tr>
<td>Unclassified Application</td>
<td>Has no impact on system security.</td>
</tr>
<tr>
<td>Console 1</td>
<td>Communication between terminal and console can be intercepted. Only unclassified information of the secret application (this is guaranteed by the flow control middleware) or of the unclassified application can leak to the outside.</td>
</tr>
<tr>
<td>Console 2</td>
<td>Communication between terminal and console and can be intercepted. No information produced by any other component can leak to the outside.</td>
</tr>
<tr>
<td>Crypto</td>
<td>Secret and top secret information can be disclosed.</td>
</tr>
<tr>
<td>Network I/O</td>
<td>Has no impact on system security.</td>
</tr>
</tbody>
</table>

The TTSoC architecture can be considered as a realization of the MILS Separation Kernel in hardware and is therefore less vulnerable to software-based attacks (e.g., exploitation of buffer overflows). Several running and documented prototype implementation of the TTSoC architecture exist, the latest one was realized on a Stratix III FPGA.

In an exemplary application, we have depicted how multiple applications with different criticality-levels can be integrated into a single SoC. In addition, we have shown how the rules of the BLP can be enforced by the isolation and communication mechanisms of the TTSoC architecture.

A major innovation of our approach is the one-to-one mapping of MILS partitions to the IP cores of an SoC. Each partition is mapped to exactly one IP core and each IP core hosts at most one partition. Compared to shared single-core solutions, the allocation of partitions to physically separated IP cores eases the implementation of inter-partition isolation mechanisms and eliminates the additional costs and verification efforts of complex mechanisms for process isolation realized in software. Moreover, our approach follows the trend in the semiconductor industry to increase the number of processor cores on a single die in order to improve performance, computational power to die area ratio and energy efficiency of their products.

The insights gained from this work about the TTSoC architecture will serve as an important input for the implementation of architectural services in the INDEXYS project, which establishes a common multi-domain architecture for industrial mixed-criticality systems.

ACKNOWLEDGMENT

This document is based on the INDEXYS project in the framework of the ARTEMIS programme. The work has been funded in part by the ARTEMIS Joint Undertaking and National Funding Agencies of Austria, Germany, Hungary.
and the Netherlands under the funding ID ARTEMIS-2008-1-100021. The responsibility for the content rests with the authors.

Furthermore, we thank Vaclav Mikolasek, Roman Obermaisser, Sandra Wögenstein, and Désirée Brand for the fruitful discussions and the teamwork.

REFERENCES


