# Functional Safety on Multicore Microcontrollers for Industrial Applications

Thomas Barth
Department of Electrical Engineering
Hochschule Darmstadt – University of Applied Sciences
Darmstadt, Germany
thomas.barth@h-da.de

Prof. Dr.-Ing. Peter Fromm
Department of Electrical Engineering
Hochschule Darmstadt – University of Applied Sciences
Darmstadt, Germany
peter.fromm@h-da.de

***Abstract*— Besides the gain in performance, a strong motivation for the introduction of multicore microcontrollers is the realization of safety architectures. Together with an industrial partner it was investigated if safety critical applications, which require a PL d according to ISO 13849, running until now on redundant discrete microcontrollers can be replaced with an architecture running on a single AURIX multicore controller. In this paper, we compare a state of the art multicore architecture with the traditional solution of using redundant controllers. The focus is put on the question, how we can achieve a safe separation of the cores, memories and peripherals? Besides the separation in the data and resource-domain, detection and escalation of errors are crucial components to achieve the required performance level. The investigations have been performed on an AURIX TC27x multicore microcontroller utilizing the safe-RTOS PXROS-HR.**

***Keywords*— *Multicore; functional Safety; ISO 13849; AURIX;***

## I. Introduction

Safety critical architectures need to be designed in such a way, that failures become unlikely. In the given scenario, the system is controlling safety critical functions of a forklift truck. Functions like lifting of loads or steering need to be reliable in order to allow safe operation of the machine. Redundant architectures are a common approach to reduce the probability of failure. If one signal-path becomes erroneous, the output signals of both paths do not match and the error can be detected. The same can be applied for input signals. If e.g. a position is measured with 2 redundant sensors, the signals can be checked for plausibility. In either case, if there is a discrepancy between two signals and therefore a potential error, a safe state of the function can be triggered. In the current control system architecture, two redundant microcontrollers have been used to process data and monitor the corresponding microcontroller. The investigation focused on the question, if those two redundant microcontrollers can be replaced with a single AURIX multicore microcontroller in the future. The AURIX multicore microcontroller was chosen as it is an automotive grade microcontroller which is certified according to ISO 26262 and provides a wide set of safety mechanisms.

## II. Comparison Discrete Microcontrollers vs. Multicore

The current architecture utilizes two discrete microcontrollers, which are linked to each other using Infineon’s Multiprocessor Link Interface (MLI). One of the microcontrollers runs the control algorithm, while the other runs the monitoring algorithm, which checks for plausibility between the input and output signals as well as for plausibility of the control algorithm. Safety critical input signals are read redundantly and are ideally diverse to increase the reliability. Only if the monitoring algorithm verifies the plausibility, the output signal of the control algorithm is transmitted. If the output is disconnected from the actor, a safe state for this actor is reached automatically. A simplified schematic of this architecture is pictured in Figure 1.

Each control algorithm has its own monitoring algorithm. The assignment of control and monitoring algorithms is not fixed to certain physical controllers. In consequence, one physical controller can run both, control and monitoring algorithms. Examples for such control/monitoring pairs are the lift and drive functions of the forklift.

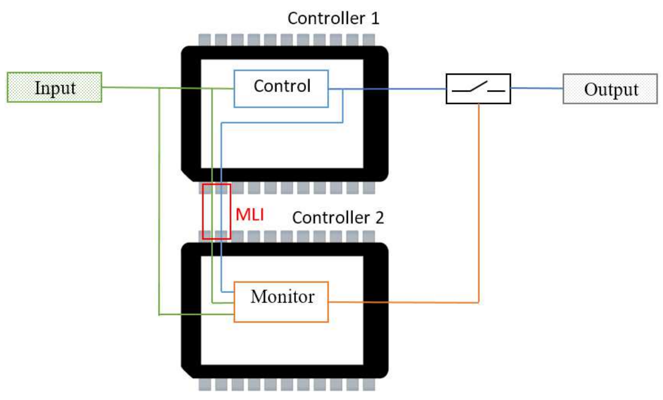

Fig. 1. simplified schematic of the discrete control/monitoring architecture

In order to reduce production cost and to improve the latency by removing the MLI, the current architecture shall be ported to a multicore controller. Instead of assigning the applications to multiple discrete singlecore microcontrollers, they now shall be assigned to separate cores on the same microcontroller. The diversity in acquisition of input signals will now be realized by using varied technologies such as DS ADC vs. SAR ADC for analog signals.

## III. Standards

The architecture of safety related control systems is specified by standards. While in the automotive-world this usually is the ISO 26262, for industrial machinery the ISO 13849 is applied. Within the ISO 13849, so called Performance Levels (PL) have to be reached, dependent upon the possible effects of a failure. The higher the probability of damage/injury/death which can be caused by a failure of a function, the higher the required PL (a-e) of the system (collection of functions). In the given scenario, a PL d has to be reached. A certain PL can be reached by architectural choices (category) and other measures like the detection of failures (Diagnose Coverage, DC) and the reliability of hardware (Mean Time to Failure dangerous, MTTFd). Figure 2 shows the achievable PLs in dependency from the DC, category and MTTFd.

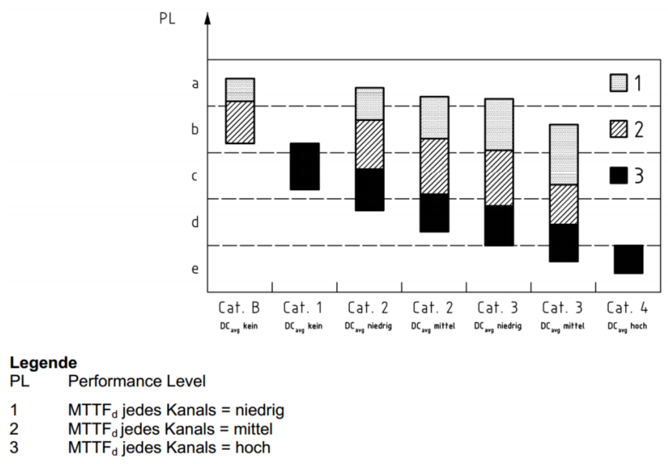

Fig. 2. achievable PL dependent from DC and MTTFd according to ISO13849 [1]

A category 2 architecture as displayed in Figure 3, is a single-channel structure with monitoring (m) of the input (I), logic (L) and output (O). If the test-unit (TE) is detecting an error, the output (O) can be deactivated with the output of the test-unit (OTE).

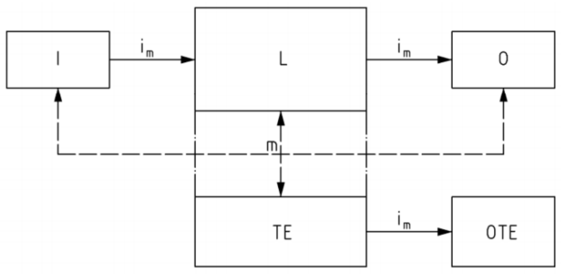

Fig. 3. Category 2 architecture according to ISO 13849 [1]

A category 3 architecture as displayed in Figure 4, is a redundant dual-channel structure with mutual monitoring of the channels. Both channels have the ability to disable the output of the other channel respectively.

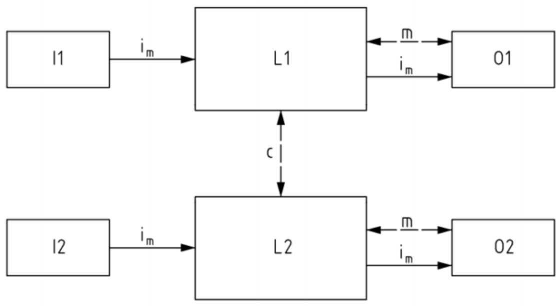

Fig. 4. Category 3 architecture according to ISO 13849 [1]

In order to implement a Cat. 3 Architecture, it must be guaranteed that the redundant channels (e.g. Core 1 vs. Core 2) not influence each other (Freedom from Interference). The freedom from interference must be achieved in the data-, resource- and time-domain [2].

Data-domain:
It has to be ensured, that different cores do not corrupt common data or worse, corrupt the local data of another core.

Resource-domain:
As multiple cores share common peripherals, it is absolutely essential that cores do not corrupt the configuration, inputs or outputs of those peripherals which are utilized by multiple cores. Moreover, global peripherals like the PLL need to be protected against unintentional reconfiguration.

Time-domain:
All cores are independent but as they exchange data, there might be deadlocks or race-conditions. It has to be ensured that the timing-behavior of all cores remains consistent.

The focus of the investigation was mainly, how to ensure freedom from interference in the data- and resource-domain by utilizing the features of the AURIX and developing an appropriate software architecture.

## IV. Software Architecture

The basic concept of the developed architecture restricts access by defining a “safety core”, which has exclusive access rights to safety critical memory and peripherals [3]. This safety core ensures that the requirements by the standard are met (e.g. steering left might not result in steering right) but does not run control-applications. Instead, acquired data is dispatched to others core(s) (“application core”) which process the data and send the result back to the safety core. The safety core checks the result for plausibility, and accesses the hardware. Plausibility is checked for input signals, output signals and the logic of the safety-application itself.

Non-safety-related comfort functions (like a cabin light) can be controlled by another core which has access to certain non-safety memory and peripherals. This core is running the application on its own (“comfort core”). The interaction between the several cores is pictured in Figure 5.

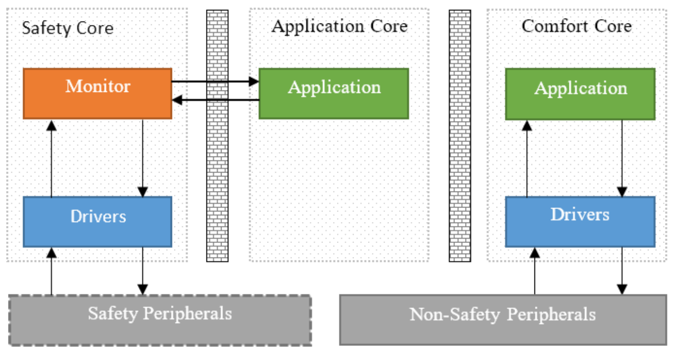

Fig. 5. simplified schematic of the multicore control/monitoring architecture

This architecture has the benefit that application-code can be easily replaced without the need to requalify the complete system, as the monitoring-code remains untouched. It is possible to flash independent parts of the firmware, which makes it even easier to update certain components.

The primary hardware feature of the AURIX to ensure freedom from interference in the data- and resource-domain, is the CPU-MPU. Each core is equipped with its own CPU-MPU, which is monitoring outgoing communication to the bus at address-level (covering memory and peripherals). Moreover the CPU-MPU is able to distinguish between read/write/execute accesses.

The CPU-MPUs not only control access on core- but also on task-level i.e. that each task has to reconfigure the CPU-MPU on the core where it is executed. In order to simplify this configuration, the RTOS PxROS from HighTec was chosen. After startup, every memory and peripheral is accessible by every core. With PxROS running, no memory and no peripherals are available and have to be unlocked explicitly. This is a big advantage in terms of freedom from interference in the data- and resource-domain.

PxROS is a certified multicore RTOS which utilizes the hardware features of the ARUIX in an optimal way [3]:

- Tasks can dynamically be assigned to different cores.
- Each Task has an own set of access rights for memory and peripherals.
- Tasks can exchange data in a non-blocking way, even between cores. I.e. the assignment of tasks can easily be reconfigured without the change of application-code.
- There is no limitation to the number of possible MPU-entries

It was required that developers can develop code regardless of the final partition. I.e. each application should only know which data is required and which data will be provided. The actual transfer of data shall be handled by an RTE which will be qualified and configured centrally.

A custom AUTOSAR-like RTE with the following features was developed:

- Assignment of runnables to a task and assignment of tasks to a core
- Extension for messaging between cores
- Support for access rights (RO, RW, etc.)
- Direct coupling of drivers to the RTE
- Scaling of raw-data in application-data in the RTE

## V. Memory/Peripheral Protection

On the AURIX, each core has its own RAM as well as shared RAM and ROM. The primary feature for protecting memory and peripherals from unwanted access is the CPU-MPU. However, the CPU-MPUs don’t control incoming accesses. Due to potential incorrect configuration (or in order to attack the system), cores could illegally access peripherals or other cores (memory, as well as CPU-configuration).

The AURIX provides a set of features that can help to protect memory and peripherals from unwanted access and complement the CPU-MPU. Besides the blocking of illegal access, the reaction of the system to such an access right violation can be freely configured. While the CPU-MPUs are reconfigured frequently, the configuration of those additional features is more static.

### *A. Bus-MPU*

All RAM’s of the TC27x come with a Bus-MPU, which is able to limit write-access to certain memory ranges for certain cores. It is possible to assign up to 8 memory ranges to specific bus-masters (CPUs but also e.g. DMA) [4]. The difference to the CPU-MPUs is that they control outgoing traffic, while the Bus-MPU is controlling incoming traffic. If a core violates the Bus-MPU configuration, different escalation strategies can be freely configured.

## *B. Register Access Protection (RAP)*

Every Bus-Slave (peripherals, as well as memory) is equipped with a RAP (Register access protection) which is able to limit the write-access to the bus-slave for certain bus-masters. It is therefore possible to limit the write access to e.g. GPIO-Ports for certain cores. In the developed architecture, only the safety-core with the qualified safety functions is allowed to write to safety critical peripherals. Moreover, the reconfiguration of system-peripherals (clocks etc.) can be limited or locked after the initial configuration.

All safety related registers of global peripherals on the AURIX are SENDINIT protected. In order to write to them, the writing core needs to have access to the global safety watchdog. The access to this safety watchdog and therefore the permission to write safety related global registers, can be restricted using the RAP. A core could gain write-access to e.g. memory, by reconfiguring the RAP or the Bus-MPU. However without the access to the safety watchdog, it is not able to do so. In a static configuration, no core should have the right to change safety related registers of global peripherals after the initial configuration.

## VI. ERROR HANDLING

The required PL d with a Cat. 2 Architecture requires at least a medium DC, which means that at least 90% of all possible dangerous errors have to be detected [1]. The TC27x provides a big set of various features that are able to detect errors. The maximum achievable DC on the TC27x is >99% [4].

The primary feature for handling errors on the TriCore (TC) 1.6 cores are trap-handlers. Errors like illegal OP-codes and addressing-errors etc., that have been caused and detected by one of the cores, trigger a so called trap. Whenever a trap is triggered, the core calls user-defined trap-handlers. On the TC 1.6 cores of the TC27x, there are 8 different classes of traps of which all have their own trap-handlers.

While traps primarily handle errors that occur in the scope of a core, there are features that detect errors that occur in a more global scope, such as the bit-flips in memory or overheating of the die. Those error-detection features have a direct connection to the SMU (Safety Management Unit) of the AURIX and are called alarms in the context of the SMU. However, there are selected errors like access right violations in the CPU-MPU that cause a trap as well as triggering an SMU alarm. The different error paths are pictured in Figure 6 as an example of one core. Beside alarms from hardware (HW), the user also can trigger dedicated software (SW) alarms.

The SMU allows the individual configuration for the reaction of the system to an alarm. The system architect needs to decide how to handle the 138 different alarms for the TC27x, though they can be grouped.

An alarm can be escalated in different ways:

- No reaction
- Fire Interrupt to certain cores
- Fire NMI (non maskable interrupt) to certain cores. NMIs are handled as traps on the TC1.6
- Reset multicore controller
- Send certain cores to idle-mode (cut clock signal)

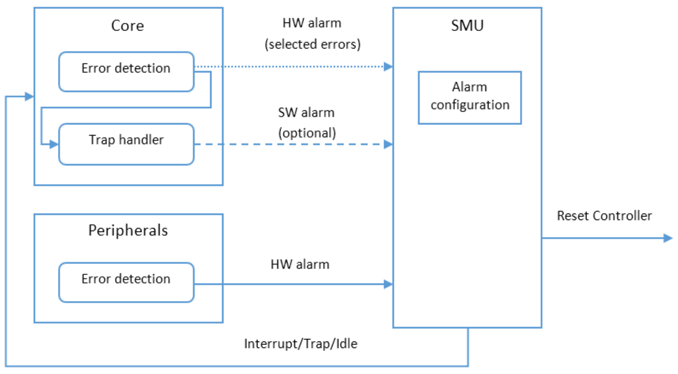

Fig. 6. simplified error handling paths

This flexibility in error handling makes it possible to create a division in different stages of error-handling which is pictured in Figure 7 (top-down). Software alarms allow for escalation of errors to the SMU and as such allow for iterative error handling based on the detected severity.

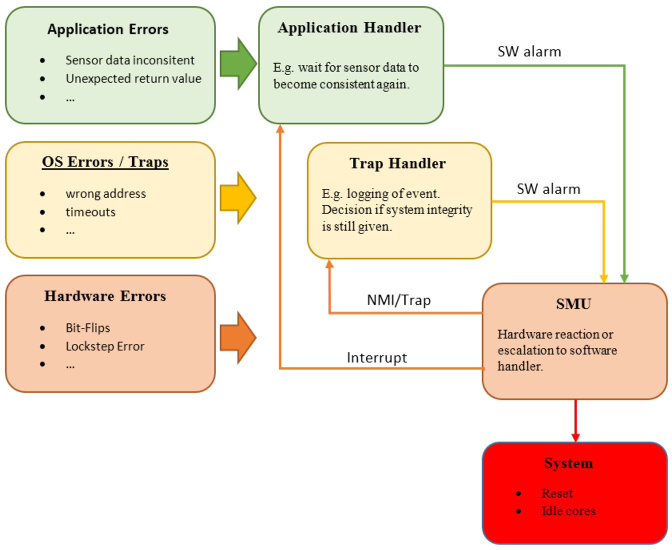

Fig. 7. error escalation possibilities

Moreover, the TC27x features an emergency-stop feature. If an alarm, associated with this feature, is escalated to the SMU, it is capable of switching pins into a predefined state automatically without software.

## VII. Conclusion

The project showed that multicore architectures seem to be an interesting option for safety-related applications. Freedom from interference in the data- and resource-domain seems to be primarily achievable with the CPU-MPUs and a hardware optimized OS.

To limit the risk of wrong CPU-MPU configuration or possible attacks, additional features are available on the TC27x:

- The Bus-MPUs allow for static assignment of memory ranges to specific cores on the bus level. A clear partition of memory can be achieved.
- The Register Access Protection (RAP) allows static assignment of peripherals to specific cores on bus-level (GPIO ports are seen as separate peripherals). Moreover, by restricting the access to certain peripherals, cores can lose the ability to gain access rights by reconfiguring the control mechanisms.

The detection of hardware faults and logical errors is crucial to meet the high requirements of the Diagnostic Coverage given by the ISO 13849. The utilization of the SMU seems to be the correct strategy to reach a high DC of >99% [4]. Moreover, the SMU allows for flexible configuration of the reaction to a fault and therefore opens the door for advanced escalation and logging strategies.

The usage of HighTec PxROS-HR along with the Aurix has been proven to be suitable. It was possible to build up a powerful multicore runtime-environment with manageable effort, which already is in use in initial projects.

In comparison to a singlecore architecture, multicore architectures are a big challenge. There are many different safety-mechanisms available which complement each other but need to be kept consistent. A change in one part of the overall architecture might affect several safety-mechanisms and therefore configurations. Tools have to be developed to provide a safe and economical configuration of safety-mechanisms.

Another interesting question for the future is the advancement from a fail-safe towards a fail-operational solution. In such a fail-operational solution, tasks could be executed by other cores which would make a reconfiguration of the safety/separation-mechanisms necessary.

## List of abbreviations

| | | |
|---|---|---|
| DC | - | Diagnostic Coverage |
| MPU | - | Memory Protection Unit |
| PL | - | Performance Level |
| RAP | - | Register Access Protection |
| RTE | - | Runtime Environment |
| RTOS | - | Real-Time Operating System |
| SMU | - | Safety Management Unit |
| TC | - | TriCore |

## References

[1] DIN EN ISO 13849, Safety of machinery – Safety-related parts of control system, 2006.

[2] Infineon, AURIX Safety Manual AP32224 V1.2, Infineon Munich, 2015.

[3] Prof. Dr.-Ing. Peter Fromm, Thomas Barth, and Mario Cupelli, “Sicherheit auf allen Kernen, Entwicklung einer Safety Architektur auf dem AURIX TC27x”, Tagungsband Embedded Software Engineering Kongress, 2015, pp. 243-251.

[4] Infineon, TC27x C-Step User’s Manual V2.0 2014-07 , Infineon Munich, 2014.