embeddedworld2017
Exhibition&Conference
... it´s a smarter world

A Monitoring Based Safety Architecture for Multicore Microcontrollers

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*— Separation in the data-, resource- and time-domain is a big challenge on multicore microcontrollers as, depending on the architecture, resources like peripherals or memory are shared between the cores. In the resulting software architecture – which often becomes very complex and fragile – changes are hard to be incorporated. Together with an industrial partner, an innovative runtime environment, which is based on the ideas of Adaptive AUTOSAR has been developed and implemented on an AURIX TC29x multicore controller. It combines high performance with good usability and a strict separation of signals in the data- and time domain. In order to ensure the integrity of signals, this concept has been extended by implementing a safety harness, which consists of four monitoring blocks, supervising sensor-data-input, actuator-output, logic-function-calculation and system health. The developed architecture supports a clear traceability between safety requirements and monitoring code. The execution of safety functions is clearly separated from the application code. The structure of the monitoring logic is easily maintainable, including defining flexible escalation strategies in case of system errors.**

Keywords*— *Multicore; runtime environment; Monitoring; functional safety; engineering standars; AURIX;

I. INTRODUCTION

Multicore microcontrollers can be an interesting choice for safety critical applications. While on single core microcontrollers one core is responsible for executing safety and non-safety code, multicore microcontrollers allow assignment of specific tasks to dedicated cores, as illustrated in Fig. 1. In the developed architecture, only one core communicates with safety critical peripherals and performs safety monitoring, while the application code executes on another core.

This approach eases requalification of application code, because a change within the application does not directly affect safety critical code on other cores. Consequently, the freedom from interference improves. Even a fatal error on the application core can be detected by the safety core by supervising the timing behavior of the application core.

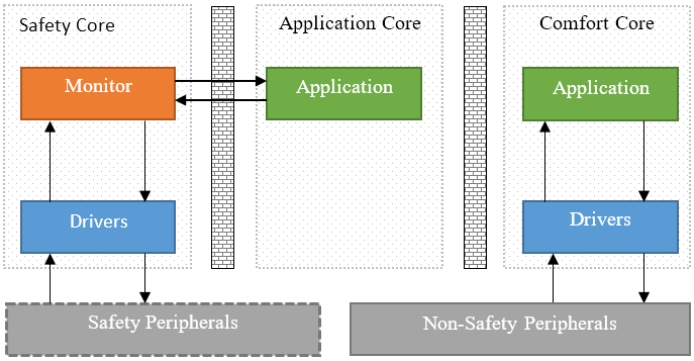

Fig. 1 multicore control/monitoring architecture [3]

Although multicore architectures seem to offer many benefits, the overall complexity of the system increases. It is estimated that multicore technology will increase development expenses significantly. The costs are expected to be increased by a factor of 4.5, while the personal need is expected to be increased by the factor 3. [1]

Advanced tools and methods are required to counteract those challenges. The developed runtime environment aims to ease development by standardizing signals, APIs and signal flow, while providing monitoring interfaces to ensure signal integrity. As application engineers work with dedicated signals instead of hardware interfaces, the overall system complexity remains hidden. Application artifacts have no dependencies to specific architectures and are portable to single core.

The monitoring interface not only ensures integrity of signals, but also allows implementation of more complex safety requirements like system health or engineering standards.

II. MULTICORE MICROCONTROLLERS FOR SAFETY CRITICAL APPLICATIONS

Safety critical architectures need to minimize the probability of failure. The definition of a failure is highly application specific, though certain patterns like detection of hardware failures are required by most safety related standards. Engineering standards like the EN ISO 13849 moreover define safety measures, which have to be applied based on the severity of a safety function failure.

Systems which e.g. have to apply with EN ISO 13849 PL D (can be compared with EN ISO26262 ASIL A [2]) either require a redundant signal path or a non-redundant signal path with at least a diagnostic coverage (DC) of >90%, in order to reach the required probability of failure. The diagnostic coverage describes the fraction of detectable failures, which may not lead to dangerous behavior of the system. Because the freedom from interference between multicore-cores seems not to be achievable [3] in terms of the standard, the focus is to reach a high DC with a non-redundant signal path. While the required DC is tackled in hardware by using lockstep cores and other hardware features, monitoring algorithms can support the DC in the software signal path.

Multicore microcontrollers allow separation of application and monitoring algorithms by assigning the algorithms to separate cores. Hardware separation mechanisms like the CPU-MPU and others, divide the system in safety domains [3]. While the core executing the monitoring algorithms needs a high degree of safety, the application core has lower requirements regarding safety. Errors in application code will be detected by the monitoring algorithm and therefore cannot lead to potentially dangerous behavior. The separation therefore can ease development and qualification of the system. In addition, requalification might be simplified, as changes on the application core will not affect the core executing the monitoring algorithms.

III. RUNTIME ENVIROMENT

In a previous project, an innovative runtime environment has been developed together with an industrial partner. The runtime environment consists out of two main components:

- The signal layer provides separated data classes and communication objects.
- The runtime engine realizes the timing behavior.

A. *Signal layer*

Core element of the signal layer are data classes, which define a uniform API to signal objects, see Fig. 2. Data classes consist of three aggregates:

- Application data provided towards the application in a user-friendly format (e.g. SI units)
- Driver interface definition towards the system in a driver specific format. A driver can be a hardware interface but also an OS service or a communication stack.
- Meta data of the signal object such as size or age.

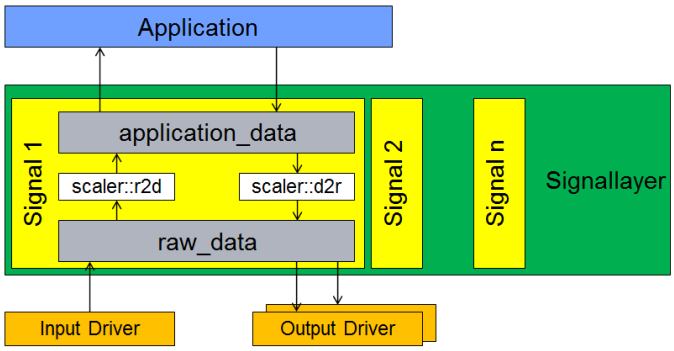

Fig. 2 Signal layer

B. *Timing behavior*

The runtime environment organizes software components in tables. The software components are called sequentially and access signals in the signal layer. A typical sequence is acquisition of input data, applicative processing and output of the processed data.

IV. EXEMPLARY SIGNAL FLOW

An exemplary signal flow shall be discussed on the example of a driving automation platform, e.g. forklift truck or semi-autonomous transportation system. The input value for a target speed signal is received from two complementary sensors. The target speed signal is processed by the application, before it is send to a communication stack, which is addressing the actuators of the system. The general signal flow is visualized in Fig. 3

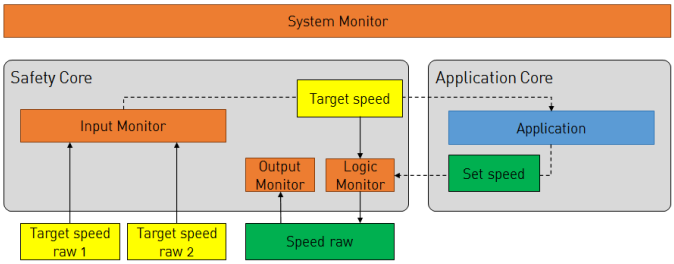

Fig. 3 exemplary signal flow

The input monitor checks the plausibility of the input signals and creates a target speed signal, which is send towards the application and the logic monitor. The application is processing the target speed signal and writes the result into a "set-speed" signal". The "set-speed" signal is checked for plausibility by a logic monitor, before it is send to a communication stack in form of a "speed raw" signal. The "speed raw" signal is continuously monitored by the output monitor.

Safety runnables attached to the monitors not only allow checks for application specific plausibility but also provide an easy to use interface for safety requirements e.g. by engineering standards. In the given example, the maximum speed, which is send to the actuators, shall be limited by a system-safety-FSM. The system-safety-FSM is derived from a global system monitor, which checks the system health by e.g. processing e.g. hardware errors. See Fig. 5.

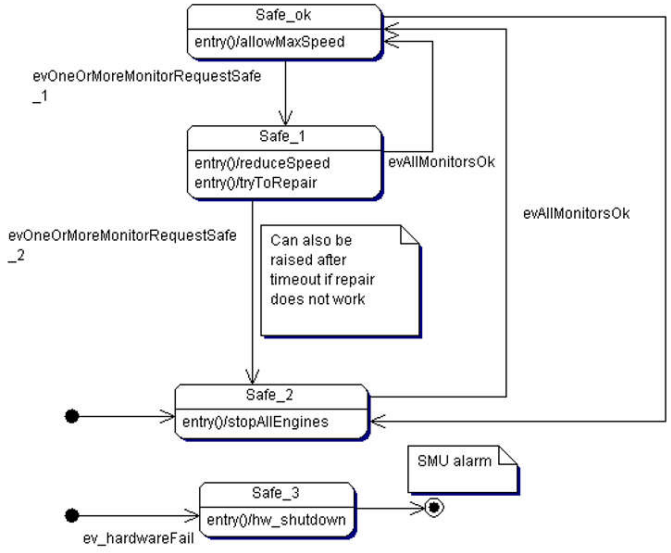

Fig. 5 exemplary system-safety-FSM

By limiting the output speed in case of an unsafe system state, the application cannot overwrite this restriction and the system behavior remains defined. In a more complex scenario, requirements by engineering standards can be checked within the monitor. By checking safety requirements within the monitors, (re)qualification processes can be simplified, as the application cannot violate the constraints. Moreover, by assigning safety runnables to the monitoring tasks, the portability and readability of software artifacts is drastically improved. A system can easily be checked against application specific safety requirements.

V. ASSIGNING SAFETY REQUIREMENTS

Safety runnables are assigned like application runnables, by inserting pointers to the runnable into a table, which is executed by the monitoring tasks (Fig. 4).

```
/*
 * Cyclic Runnables Table
 * Entry to the table shall be made as
 * runnable ptr,        cyclic Time,        relative Offset Time
 */
const RT_cyclicSafetyTable_t RT_C0_safetyInput_cyclicRunTable[] = {

      //runnable ptr,
      {(RT_runnableSafetyPtr_t)(REMOTE_run_readAndCheckProtocol),
      {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkJoystickData),
      {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkJoystickZeroLeavingSafe2State
      {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkRemoteTransmissionProtocol),
      {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkZigbeeSignalStrength),
      {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkEngineControlHeartBeat),
      {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkCollissionSensor),
```

Fig. 4 Safety runnable assignment

Each runnable within the table returns a state which tells if the safety requirements had been met. The result can be evaluated to control system-health monitors. In the example given in exemplary signal flowIV, the target speed signal is only generated if two complementary inputs are valid. The input monitor not only can suppress sending the target speed to the application, but also can influence the system state in order to stop all engines, as an invalid target speed signal implies a severe system error.

VI. CONCLUSION

Multicore microcontrollers remain a big challenge in embedded software engineering. Although questions regarding safety on multicore microcontrollers are not answered completely yet, they seem to offer a high degree of separation between safety domains and might ease qualification processes. Besides the benefits, multicore microcontrollers introduce a new era of system complexity. A clear and safe system architecture is very important and can be supported with tools like the developed runtime environment. In order to ease the development process further, tools for model-based base system generation and timing analysis are under development.

While the system complexity increased, the runtime environment introduces an easy to use interface for application developers, integrators and system architects. The first version of the runtime environment had successfully been introduced to the field on forklift trucks. In research projects, it was possible to integrate five sub-projects from five teams with 30 students and no multicore experience in only three month. Because of the stringent monitoring of signals, students cannot violate safety constraints and the target system remains safe even in case of erroneous application code. This approach accelerates application development by separating safety from application code and reducing the risk of unintended behavior. The complexity from the point of view of a developer is reduced, allowing application development even with no multicore experience.

An omnidirectional driving robot (Fig. 6) is currently under development, in order to evaluate the monitoring architecture, introduce students to multicore microcontrollers and develop the architecture further.

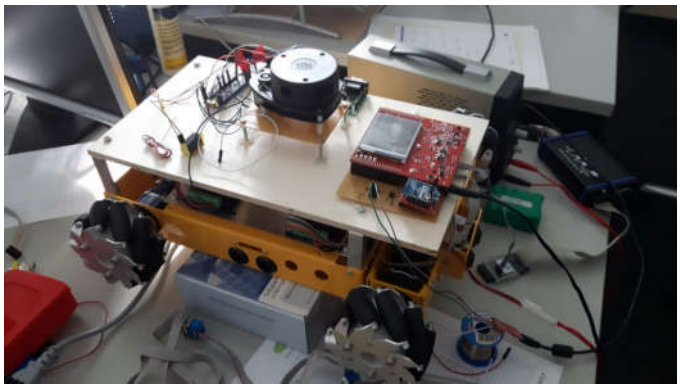

Fig. 6 Demonstrator

LIST OF ABBREVIATIONS

API	-	Application Programming Interface
DC	-	Diagnostic Coverage
FSM	-	Finite State Machine
PL	-	Performance Level

REFERENCES

[1] S. Balacco and C. Rommel, "Next Generation Embedded Hardware Architectures," in *VDC Research*, 2010.

[2] M. Kieviet, *ISO13849 and ISO26262 for the same Domain,* innotec GmbH.

[3] T. Barth, "Functional Safety on Multicore Microcontrolles for Industrial Applications," in *Embedded World Conference*, Nürnberg, 2016.