# Functional safety in multicore heterogeneous systems for automotive surround view applications

Vibha Pant, Piyali Goswami, Sujith Shivalingappa
*Embedded Processing,*
*Texas Instruments*
*Bangalore, India*
Email: *v-pant@ti.com, piyali_g@ti.com, sujith.s@ti.com*

***Abstract*—Driverless cars are the next step in the evolution of the automotive industry. Advanced Driver Assistance Systems (ADAS) are the stepping stones to achieve and enhance the driving experience and more importantly safety in automotive systems. Safety standards such as ISO 26262 define Automotive Safety Integration Levels (ASIL) depending on the severity and probability of an error which may lead to loss/harm to life. Automotive electronic control units (ECUs) are made up of multi-core heterogeneous SoCs. Making all components of an automotive ECU to the highest ASIL level is not feasible due to significant cost and effort. This paper looks at software and hardware methods to have mixed safety levels co-exist in a heterogeneous SoC environment targeting surround view ADAS system while incorporating AUTOSAR requirements.**

***Keywords—ADAS, Multi-Core Heterogeneous SoCs, AUTOSAR, functional safety, FFI***

## INTRODUCTION

Automotive embedded systems are safety critical systems wherein errors or failures could result in injury or loss of human life, loss or severe damage to equipment, or environmental harm. Functional safety relates to the correct operation of a system while preventing life-threatening hazards with safe management of likely errors and failures. This involves identifying safety functions and assessing the risks. ISO26262 is an international standard for functional safety in the automotive domain. ISO26262 defines different Automotive Safety Integrity Levels (ASIL) for classifying the risk. A system can be classified based on the failure rate and the effectiveness of failure detection. The integrity requirements range from Quality Managed (QM), ASIL A to ASIL D. A system classified as ASIL D, the highest integrity requirement, would be extremely robust in preventing the occurrence of failures which would cause life-threatening consequences. QM systems, on the other hand, do not dictate any safety requirements in accordance with ISO26262 [1][15].

With autonomous vehicles gaining momentum, automotive systems gain complexity day by day. Heterogeneous multi-core systems are deployed to achieve the specific processing requirements of such advanced signal processing applications [2-5]. This makes achieving functional safety in such systems even more challenging. The ideal scenario is to have all components at the highest safety integrity level. However, this is not feasible as it makes the system complex and incurs significant development costs.

The automotive surround view (SRV) camera system is an automotive ADAS (Advanced Driver Assistance System) technology that's rapidly gaining importance. It allows the driver to see a top to down view of the 360-degree surroundings of the vehicle. A typical SRV system normally comprises of four to six wide-angle (fish-eye lens) cameras mounted around the vehicle, each facing a different direction [6][7][13]. A composite view of the surroundings of the vehicle is synthesized from these inputs from the camera and shown to the driver in real-time during parking. In order to generate a precise image, many components are required such as capture unit, display unit, synthesis creation, geometric alignment and photometric alignment. Additionally, such systems are equipped with advanced algorithms for pedestrian and vehicle detection. These tasks are distributed over different cores in a SoC for optimal utilization and performance [13][21]. A typical surround view system is shown in Figure 1.

The interaction with the vehicle CAN network is the highest critical task in such systems. This component monitors the execution of the rest of the system and notifies and contains system failures. Typically this component would be executing the AUTOSAR software stack. AUTOSAR (Automotive Open System Architecture) is a standard for the development of software for embedded devices, primarily created for the automotive domain. It specifies a software architecture which provides a common ground for building applications at a certain level of safety [8][16].

Capture and display units, on the other hand, need to detect sensor, display failures and need to operate at a fixed performance rate to ensure real-time and latency requirements are met. Similarly, the control sequence unit is responsible to make sure all the processing blocks work in tandem and monitor the execution of the algorithms. Failures in these blocks can lead to blind spots and complete lack of visibility.

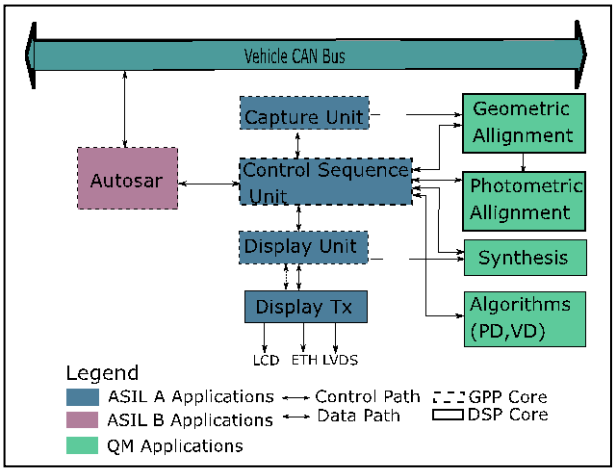


Fig 1: Surround View System Block Diagram

Algorithms can have varying levels of criticality depending upon how their outputs are being consumed. In this system, we assume the output of a detected pedestrian/vehicle are used to provide information/warning to the driver and are not influencing the vehicle operation. Hence they are assumed to

be the lowest safety components of the system. This leads to a system with mixed criticality levels and imposes each level to safely communicate with the other. The co-existence of various safety levels is a major challenge in heterogeneous multi-core systems as we need to ensure that the integrity levels of higher ASIL components are not compromised because of their interaction with the lower safety level components[6][15][18][20].

In this paper, we discuss the proposed solution to the issue of mixed safety levels in a heterogeneous environment in section I and the implementation and results in section II.

## I. PROPOSED SOLUTION

### A) *System Architecture and modules to implement freedom from interference(FFI)*

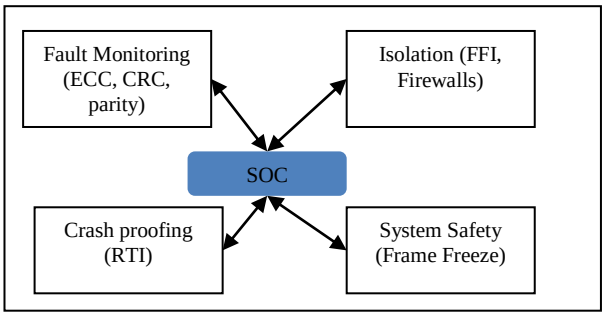


Fig 2: Enabling Safety

There are multiple focus areas to achieve a required safety level in an automotive ECU. They can be classified as follows:

- Fault Monitoring: Error checking and correcting (ECC), parity, dual clock comparators (DCC), boot and runtime diagnostics, and Cyclic Redundancy Checks (CRC) are mainly used in this context on memory and communication channels.
- Crash Proofing: Crash proofing in a safety critical component is essential, such that when a component with lower safety integrity crashes, the higher safety integrity component would indicate a warning to check the failed component but would still continue to function. This can be implemented through Real-Time Interrupts (RTI) and watchdog timers. The RTI is responsible for starting tasks and keeping track of how much minimum and maximum time it is supposed to take for completion. If the scheduling constraint is violated, then generate an interrupt
- System Specific Safety goals: In the SRV system, one key requirement is to be able to detect if any one of the video input channels is not frozen and showing the same frame always. Here CRC checksums can be compared across frames to make sure the same CRC is not getting received for multiple consecutive frames.
- Isolation: ISO26262 mandates that in a system with components of different levels of safety integrity, freedom from interference (FFI) needs to be guaranteed [1]. FFI in a system ensures that errors in a component do not propagate to other components which would lead to the violation of safety integrity or lead to a system break down [16]. The higher ASIL components can be safeguarded from lower safety level components through memory and task isolation [17][19][21].

Memory isolation can be achieved by restricting the access to specific memory locations. Firewalls are commonly used for memory isolation wherein, access to specific memory regions is restricted by using identifiers. Identifiers consist of the CPU Index (master Id) and the mode in which the CPU is operating. The DMA channels can have their own unique indexes and could potentially inherit the mode of the CPU from which the DMA is triggered. Memory protection units and memory management units are commonly used memory isolation hardware units which based on the CPU mode provide different access permissions to different regions of memory.

Fig 3 describes such a multi-core automotive ECU where general purpose cores, signal processing cores and DMAs coexist to initiate memory requests to internal/external memory and I/O peripherals. Each system master is capable of running tasks with mixed criticality and the techniques to achieve FFI ensure tasks within or across cores are isolated. We deep dive into details regarding the FFI implementation in further details in the following sections.

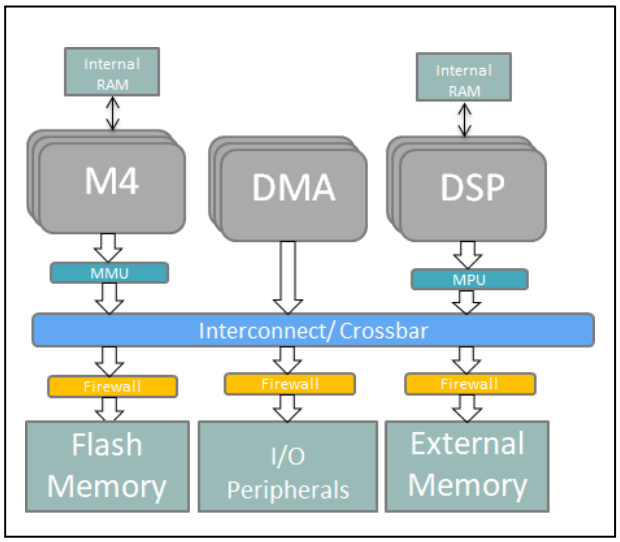


Fig 3: System block diagram to achieve Isolation

Isolation also requires components/tasks of different criticality to communicate with one another in a way that if either task crashes, the other continues operation and can report failures gracefully. This mandates the need for a non-locking IPC between tasks of mixed-criticality.

### B) *IPC between AUTOSAR and non Autosar world*

Inter-processor Communication (IPC) is essential for communication in a heterogeneous environment. In a mixed criticality system, the communication needs to be deadlock free, mainly relying on the hardware to transport the arguments. Efficient IPC requires non shared memory between the sub-systems of different

integrity levels. Software imposed timeouts help in ensuring that deadlocks do not occur

A typical scenario is illustrated in Fig 4. IPC between components in the SRV system is performed via mailboxes. Communication between the tasks uses a queued mailbox-interrupt mechanism. The queued mailbox-interrupt mechanism allows the software to establish a communication channel between two processes through a set of registers and associated interrupt signals by sending and receiving messages (mailboxes). Mailboxes allow many-to-many interaction.

Both the sender and receiver components start by setting up the individual core specific parameters (Step 1). The Sender and receiver then synchronize to make sure both are ready to transmit and receive messages (Step 2). Based on the tasks, the different event handlers are registered on either core (Step 3). Each component can send multiple events (Step 4). The corresponding recipient's callback is invoked upon reception of a new event (Step 5 & 6). A table is maintained to keep track of the entire received request. The receiver can acknowledge the apt request (Step 7). The sender component also maintains a callback table and delivers the message/payload to right callback upon receiving acknowledgment interrupt (Step 8 & 9).

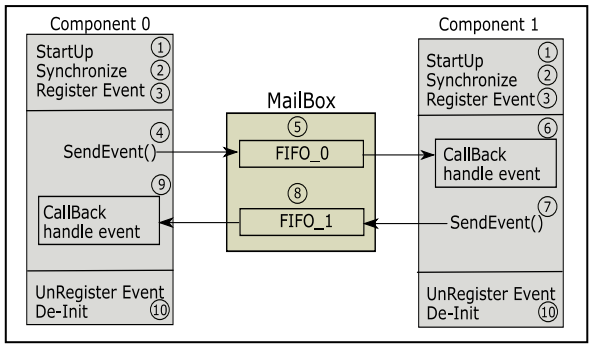


Fig 4: Mechanism for IPC

### *C) Freedom from Interference (Task and Memory Isolation)*

When mixed ASIL components co-exist in a system, it is mandated by ISO26262 to have "Freedom From Interference (FFI)" of lower ASIL component towards higher ASIL component [1]. The freedom or non-interference should be on below points [8]

a. Memory accesses
b. Timing behavior and execution order
c. Exchange of information

FFI for memory accesses in a mixed ASIL system is about providing selective memory access restriction to QM tasks so that it does not contaminate ASIL memory [9][16]. This means:

d. QM tasks and ASIL tasks need to co-exist on multiple CPUs sharing a common memory
e. ASIL tasks, therefore, can be provided R-W access for all memory
f. QM tasks need read-access to ASIL regions as they may share data with ASIL tasks
g. QM tasks should not be given write permission to ASIL regions
h. All tasks will have R-W access to QM regions

The components in the SRV system can be allowed to access different memory regions based on the permissions assigned. These permissions are recommended to be clearly defined based on the safety integrity levels. The memory is divided into multiple regions having different access restrictions. The access is restricted based on a master ID and the CPU mode. The CPU mode can either be supervisor or user mode. Protection policies are implemented to raise exceptions to avoid illegal memory accesses between different safety integrity level memories.

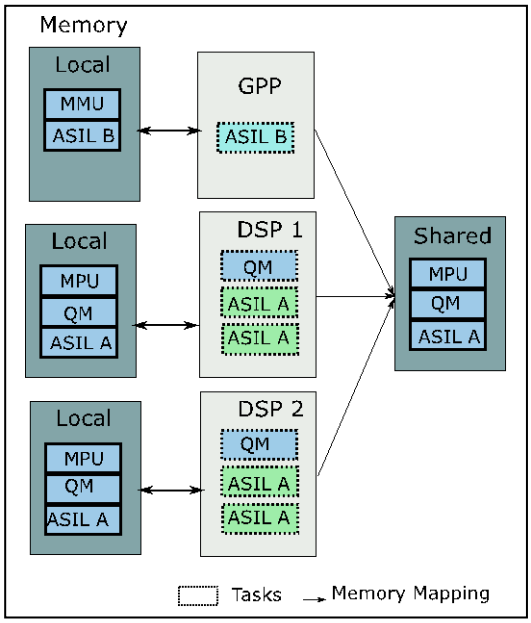


Fig 5: Safety Partitions and Memory Mapping

Task isolation can be achieved by creating safety partitions based on the task context. The commonly used tasks can be shared between the safety partitions with these common tasks running at the highest safety level [17].

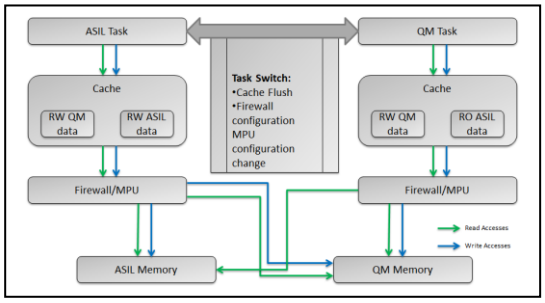


Fig 6: Task Switching

TABLE I. SWITCHING OVERHEADS FOR TASKS BETWEEN DIFFERENT SAFTEY LEVELS

| CPU | Operation | Frequency of Operation (MHz) | Cache Size (KB) | Switch Per Sec (# Switch per frame * #Frames per second) | Context switch time / frame (ms) | Task Switch (ms) | Interrupt Switch (ms) |
|---|---|---|---|---|---|---|---|
| **M4** | Capture & Sync of frames | 212 | 32 | 4 * 30 | 0.015 | 0.015 | 0.0072 |
| | Display | | | 1 * 30 | 0.015 | | |
| **DSP** | 3 Algorithms: Geometric Align, Photometric Align & Synthesis | 600 | 256 | 3 * 1 * 30 | 0.366 | 0.003 | 0.004 |

The key considerations, illustrated in Fig 6, to achieve task isolation for multi-criticality tasks running on a single CPU core are as follows. We assume the core is executing an RTOS which helps schedule QM and ASIL tasks on the same CPU core.

- Define memory regions in external memory using firewalls
- Define memory regions in internal memories using MPU/MMU.
- Firewalls and MPU are reconfigured to QM mode in the following cases:
  - QM tasks entry
  - After RTOS API execution in QM task
- Firewalls are reconfigured to ASIL mode in the following cases
  - QM tasks exit
- MPU is reconfigured to ASIL mode in the following cases
  - QM tasks exit
  - Before RTOS API execution in QM task
- Cache flush is executed in following scenarios
  - QM tasks exit
- Cache flush is required for following cases
  - Doing a switch between ASIL and QM mode
  - Executing BIOS APIs
- System integrators can choose to:
  - Ensure all ASIL tasks run consecutively followed by all QM tasks running consecutively to ensure minimal execution of cache-flush APIs
  - Ensure all RTOS data structures are in internal memory – this avoids the need of cache-flush when executing RTOS APIs

## II. RESULTS

The proposed task isolation and safe IPC mechanisms were implemented on Texas Instruments' TDA2/3 family of automotive devices. A typical 2D SRV setup on Texas Instruments' TDA2x is shown in Fig 7. The techniques for functional safety such as memory and task partitioning and FFI have been incorporated to achieve FFI in a mixed safety integrity environment. In a typical application scenario, the sequence of events is as follows:

1. The frames from the camera units are captured, duplicated and synchronized on a general purpose core (M4).
2. The algorithms for geometric alignment, synthesis and photometric alignment are executed from the DSP.
3. The synthesized outputs from the DSP is then processed for display on the GPP core and forwarded to the display.

The frequency of operation and cache sizes for the GPP (M4) and DSP cores are elaborated further in Table I. The number of switches per second is calculated by multiplying the number of frames per second and the number of switches required per second. The latency for the context switches, task switches and interrupt switches on the M4 and DSP are measured and recorded in Table I.

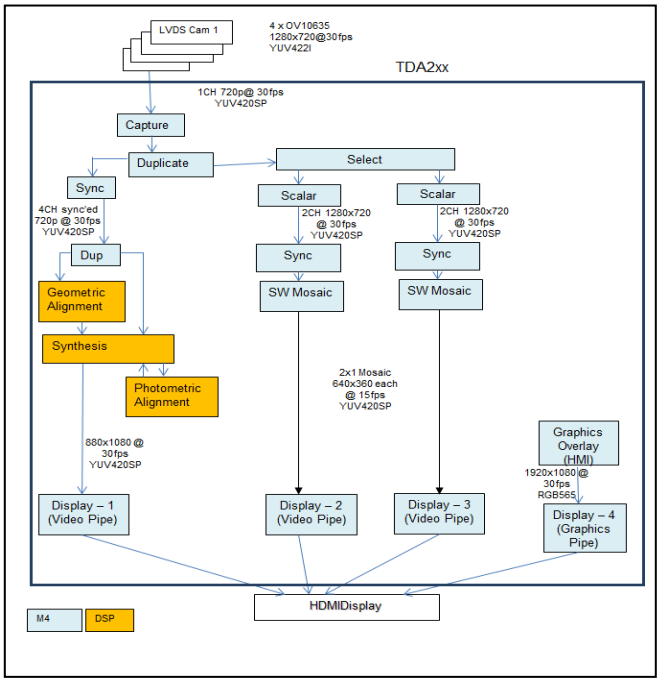


Fig 7: Experiment Setup

The CPU and memory overheads after incorporating the functional safety requirements have been measured for this setup and are recorded in Table II. The results indicate that after implementing the safety mechanisms, CPU load and memory utilization increase marginally, while still maintaining the real-time behavior required for the SRV automotive system.

TABLE II. CPU AND MEMORY OVERHEADS FOR INCORPORATING FUNCTIONAL SAFETY IN A 2D SRV AUTOMOTIVE SYSTEM

| | **M4 CPU Load %** | **DSP CPU Load %** | **Total Memory Bandwidth (MBps)** |
|---|---|---|---|
| **Non-Safety** | 33.500 | 41.700 | 1320.000 |
| **Safety** | 33.506 | 41.737 | 1348.508 |

## III. CONCLUSION

Functional safety in a heterogeneous system with mixed ASIL/QM safety levels is a critical and complex issue. Especially in autonomous driving systems, the safety goals are even more challenging and are imperative to achieve. In this paper, we have elaborated on techniques to implement functional safety between components of different safety-criticality. This has been achieved through FFI mechanisms such as memory isolation, task partitioning and safe methods to communicate through IPC. The main challenge is to minimize the overheads incurred by implementing the safety mechanisms so that the real-time behavior of the system is not compromised, as a failure can even lead to loss of human life. Finally, to ensure that the automotive system behaves as required, monitoring and diagnostics should be implemented.

## ACKNOWLEDGMENT

The authors would like to thank Yashwant Dutt, Mihir Mody, Sivaraj R, Kedar Chitnis, Shiju Sivasankaran, Brijesh Jadav, Biju MG, Jayant Thakur, Ankur, Vivek Dhande, Rishabh Garg and Prasad Jondhale from Texas Instruments, Bangalore for their valuable inputs and review comments which have given direction to the paper.

## REFERENCES

[1] "ISO 26262-1:2011," 2011. [Online]. Available: http://www.iso.org/iso/catalogue_detail?csnumber=43464.

[2] Urmson, "Autonomous driving in urban environments: Boss and the urban challenge.," *The DARPA Urban Challenge*, no. Springer Berlin Heidelberg, pp. 1-59, 2009.

[3] A. S. Puthon, F. Nashashibi and B. Bradai, "Improvement of multisensor fusion in speed limit determination by quantifying navigation reliability," *13th International IEEE Conference on Intelligent Transportation Systems*, Funchal, 2010, pp. 855-860.

[4] R. C. Luo and M. G. Kay, "Multisensor integration and fusion in intelligent systems," in *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 901-931, Sep/Oct 1989.

[5] T. I. Inc, "Advanced Driver Assistance (ADAS) Solutions Guide, SLYY044A," 2015.

[6] Texas Instruments,"TI Gives Sight to Vision-Enabled Automotive Technologies," [Online]. Available: http://www.ti.com/lit/wp/spry250/spry250.pdf.

[7] R. Gulati, V. Easwaran, P. Karandikar, M. Mody and P. Shankar, "Resolving ADAS imaging subsystem functional safety quagmire," *2015 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, 2015, pp. 291-294.

[8] AUTOSAR,"Overview of Functional Safety Measures in AUTOSAR" , [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_FunctionalSafetyMeasures.pdf

[9] A. Goebel, R. Mader and O. Tripon, "Performance and Freedom From Interference - a contradiction in embedded automotive multi-core applications," *ARCS 2017; 30th International Conference on Architecture of Computing Systems*, Vienna, Austria, 2017, pp. 1-9

[12] Vikram Vijayan, Sujith,et al,"Three Dimensional Rendering for Surround View Using Predetermined Viewpoint Lookup Tables", Patent Available: https://patents.google.com/patent/US20170195564A1

[13] Texas Instruments, "Surround view camera system for ADAS on TI's TDAx SoCs 2", [Online]. Available: http://www.ti.com/lit/wp/spry270a/spry270a.pdf

[14] M. Niklas, S. Voget, and J. Mottok, "Safety relevant development by adaptation of standardized safety concepts in autosar 4.0," Embedded Real Time Systems Conference , Toulouse, February, 2012

[15] Glas, Benjamin ,et.al, " Automotive safety and security integration challenges", Automotive - Safety & Security 2014

[16] Haworth, David et al. "Freedom from Interference for AUTOSAR-based ECUs: a partitioned AUTOSAR stack." Automotive - Safety & Security (2012).

[17] Ficek, Christoph and Nico Feiertag. "Applying the AUTOSAR timing protection to build safe and efficient ISO 26262 mixed-criticality systems." (2011).

[18] S. Faucou, "Mixed criticality in multicore automotive embedded systems," Mixed Criticality on Multicore/Manycore Platforms, 2015.

[19] T. Piper, S. Winter, O. Schwahn, S. Bidarahalli and N. Suri, "Mitigating Timing Error Propagation in Mixed-Criticality Automotive Systems," *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*, Auckland, 2015, pp. 102-109.

[20] G. Macher, A. Höller, E. Armengaud and C. Kreiner, "Automotive embedded software: Migration challenges to multi-core computing platforms," *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, Cambridge, 2015, pp. 1386-1393.

[21] F. Leitner-Fischer, S. Leue, and S. Liu, "Automated Freedom from Interference Analysis for Automotive Software," in CARS, 2016.