https://doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-017

Enabling Functional Safety ASIL Compliance for Autonomous Driving Software Systems

Kedar Chitnis, Mihir Mody, Pramod Swami, Sivaraj R, Chaitanya Ghone, M G Biju, Badri Narayanan, Yashwant Dutt, Aish Dubey*
{kedarc, mihir, pramods, sivaraj, csghone, biju, badri, yashwant.dutt, aish}@ti.com
Automotive Processors, Texas Instruments (India and *USA)

Abstract

With cars driving autonomously on roads, functional safety assumes critical importance to avoid hazardous situations for humans in the car and on the road. ISO 26262 defines Automotive Safety Integration Level (ASIL) with level QM (Least) to ASIL-D (Highest) based on severity and probability of defect causing harm to human life. This paper explores functional safety requirements and solutions for software systems in autonomous cars in four broad aspects. The first aspect covers usage of redundancy at various levels to ensure the failure of one system does not affect the overall operation of the car. It explores the usage of redundancy via multiple sensors and diverse processing of data to arrive at functionally safe results. Based on the redundancy requirements, in the second aspect, an HW (SoC) and SW architecture is proposed which can help meet these requirements. It explores the definition of SW framework, task scheduling, and tools usage to ensure systematic faults are prevented at the development stage. Autonomous driving systems will be complex and expecting all software modules comply with the highest functional safety level may not be feasible. The third aspect explores the usage of freedom from interference (FFI) via HW and SW mechanisms like Firewalls, MMU to allow safe and non-safe sub-systems to co-exist and operate according to their specification. The final aspect covers usage of SW and HW diagnostics to monitor, detect, and correct random faults found at run-time in HW modules. It explores the usage of diagnostics features like ECC, CRC, and BIST to help detect and avoid run-time failures.

Introduction

With increased focus on autonomous cars, the question of autonomous cars being a reality has shifted from "if" to "when". In the rush to get autonomous cars on the roads one must not forget functional safety aspects of the HW/SW systems so that humans, in the car and outside the car, are not put into hazardous situations due to the operation of the car. In this paper, we explore various aspects that need to be covered by software systems to ensure an autonomous car is safe on the road.

We first argue the need for redundancy in sensors and processing in order to make autonomous driving decisions. We then explore HW and SW architectures to implement these redundancies. SW best practices to follow in order to avoid systematic faults and frameworks to use for these are explored. Software system in autonomous cars can get very complex. The cost for making a software sub-system compliant to ASIL specification is huge. Hence it is not feasible to make all sub-systems to adhere to the highest ASIL level. Freedom from interference concept is explored to allow safe and non-safe tasks to co-exist within the same system. Finally, even with all the logic to implement redundancies, reduce systematic faults, isolate safety-critical systems, it important to ensure that all systems are operational and working as expected at run-time. The last section explores the run-time monitoring and diagnostics that can be done for this.

System Redundancy via Multi-Sensor Fusion

In order to let a car run autonomously, first it has to **sense** the external environment/surroundings; **process** the data and **act** by making meaningful decisions. In this sense, process and act chain, the sensing part of the external environment is taken care by sensors like camera, radar, LIDAR and referred as surround sensors in rest of the paper. Apart from surround sensors, other sensors like vehicle odometry sensors and actuators are also important to feed the information to decision-making block. For example, the steering wheel angle and wheel speed is important data for a car to make the right decision along with surrounding information. So broadly we would divide sensors in be below three categories

- **Surround sensors**: These are mounted on the external/internal surface of the car and useful to provide surrounding information. Example: Camera, radar, Lidar, ultrasonic, infrared camera, IMU, GPS and digital map etc.
- **Vehicle odometry sensors**: These sensors capture the information about vehicle motion. Example: wheel speed, acceleration, yaw rate, steering wheel angle etc.
- **Actuators**: These are the sensors which translate the human/machine actions. Example: Break Torque, Engine Torque, restraint actuators, wheel spring etc.

In this paper, we will discuss more on surround-view sensors and importance of multi-sensor fusion for an autonomous car. Car makers have been using different sensors mainly Lidar, radar, camera and ultrasonic for safety features like ACC (Automatic Cruise Control), LKA (Lane Keep Assist), blind spot detections, forward collision warning, and very recently for active safety features like AEB (Auto-Emergency Brake) as well. In recent past, industry has seen the usage of more sensor/information like satellite information, vehicle and infrastructure (V2V and V2x) and Lidar to improve the robustness of these safety features. There is significant overlap of the information provided by these sensors. At the same time, their degree of reliability varies. For example, radar and camera both can identify the distance of an object but the reliability of information from a radar sensor is higher as compared to a camera. Autonomous driving systems need to provide the highest degree of reliability and would require a good overlap of information from different sensors to make a confident decision.

Table 1 provides a good comparison of these sensors by listing their pros and cons and their applicability to different ADAS features which are critical for autonomous driving. In areas where

a sensor lacks, an alternative sensor name is mentioned in square bracket []. For example, a vision sensor is bad for speed detection whereas Radar/LIDAR can help there.

Sensor	Pros	Cons	Suitability to Features
Radar	Accurate distance and speed detection	No environmental knowledge of the scene, eg: during turn the object is not in line of sight [Vision]	Adaptive Cruise Control (ACC), Blind spot Detection (BSD), Lane Change Assist, Park Assist, Forward/Rear Collision Warning
	Can detect objects at very short range with good range resolution	Non-metallic objects, such rocks/pedestrians produce no/weaker reflections so poor detection [Vision, Lidar]	
	Aesthetically good, as it can hide behind bumpers	Poor spatial localization in absense of many antennas [Vision, Lidar]	
	Can detect objects at very long range detection	Cannot detect visual information's - traffic light, traffic signs, lane markings, debris on road [Vision]	
	Agnostic to light and weather condition	Beam blockage	
Vision	Only suitable sensor for visual appearance based object detections - traffic light, traffic signs, lane markings, debris on road	Complex processing elements for data processing and significant software development cost	Adaptive Light Control, Adaptive Cruise control, Automated emergency braking, Forward/Rear Collision Warning, Lane Keep support, road user or obstacle (pedestrian, vehicles, animals, motorist, bicyclist, debris) detections, Traffic sign and Traffic Light detection, Park Assist, Free space detection
	Good detection of road users (ped, bicyclist, vehicles, animals)	Bad (or costly using stereo camera) distance deetction	
	Easy retrieval of angular information of objects	Sensitive to weather conditions such as rain, fog [Radar, Lidar]	
	Provides good environmental information for heuristic based complex decisions	Sensitive to lighting conditions mainly unreliable in low light during night [Radar, Infrared Camera]	
	Small sizes	Bad speed detection [Radar, Lidar]	
LIDAR	Can detect very small obstacles	Cannot detect visual information's - traffic light, traffic signs, lane markings, debris on road [Vision]	Adaptive Cruise Control (ACC), Blind spot Detection (BSD), Lane Change Assist, Park Assist, Forward/Rear Collision Warning, Free space detection and path planning
	Accurate distance/speed detection	Insufficient angular resolution at long ranges [Radar]	
	Good obstacle detection	Commercially not viable yet because of big size, higher cost	
	Can provide surrounded environment model	Bad vehicle, pedestrian detection [Vision]	
		Smaller range than Radar	
Ultrasonic	Very Short range detection (< 3 meter)	Easily distorted by reflections of the road [Vision]	Park Assist during back maneuvers
	High angular range	No angular position [Vision]	
	Low cost (both hardware & software development)	No echo cancellation	
Infrared Camera	Usable under night conditions	Traffic signs show very poor contrast	Night Vision
GPS / Digital Map	Good contextual information about routes, traffic situations, road guidelines (eg. speed limit)	use of outdated databases particularly in urban environment, which changes often, or in roadwork [Vision]	Path planning, Curve and Speed Limit information

Table 1 Different surround sensors their Pros/Cons

It is quite evident from this table that there is no single sensor which can handle all cases. Therefore, for autonomous driving systems, sensor fusion is a must. Most of the existing work in this field have highlighted this fact. The first urban autonomous vehicle demonstration in DARPA 2007 had 18 sensors (9 Lidar, 5 Radar, 2 vision and 2 GPS/IMU) having redundant information for vehicle path planning [1]. Puthon, A.S. et al highlighted the importance of sensor fusion of vision and GPS for speed sign recognition [2]. With redundant information measurement precision can be enhanced and in addition, the fault tolerance of the overall system increases, as the failure of one sensor does not necessarily result in the failure of the system as a whole [3].

Consider a scenario when ACC detects that the distance to a car in front is too small. Here, ACC decides to reduce the speed. However, this decision will not be valid in the case where the driver intends to overtake the car in front. In fact, the decision to slow down increases the probability of a rear-end collision [4]. So such situations require more holistic knowledge of environment using multiple sensors

Figure 1, provides a pictorial view of how multiple sensors can help to cover different fields of view and basic functions for autonomous driving [4].

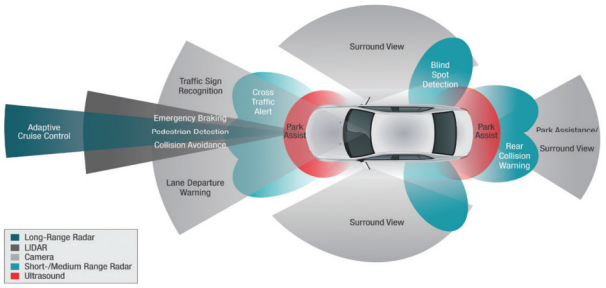

Figure 1 Surround Sensors, coverage area and applications

Multiple sensor fusion architectures are described [5] and are shown in below Figure 2. High-Level fusion architecture seems to be more practical from a safety point of view as it avoids single point failure as well more viable with more and more new sensors being added to existing systems.

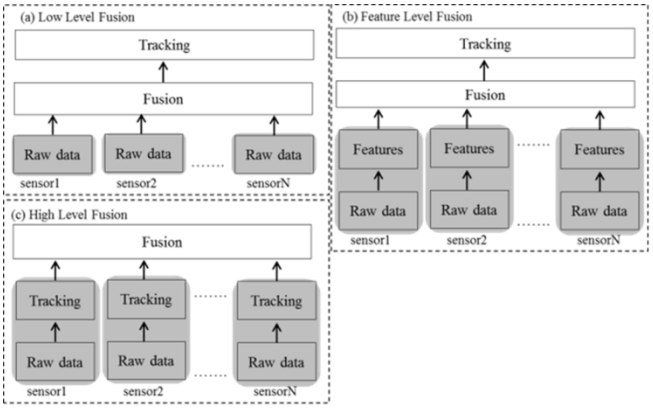

Figure 2 Different architectures of fusion systems

Irrespective of sensor fusion architecture, the importance of multiple sensors having redundancy remains high to allow a better decision from a fusion algorithm.

HW and SW Architectures

Based on Figure 2, different HW architectures are possible. Low-level fusion, Figure 2(a), implies a more centralized HW processing based system. Figure 2(b), Figure 2(c), implies a distributed HW processing based system.
From a safety point of view, the fusion SoCs shown in Figure 3 and Figure 4 represent a single point of failure and redundancy at fusion SoC would be required for functional safety. The amount of processing done in fusion SoC for centralized HW architecture would be large and redundancy of fusion SoC, in this case, would be expensive. In distributed HW architecture, the processing is

spread across multiple SoC so the processing requirements at fusion SoC would be modest and thus implementing redundancy would be relatively inexpensive.

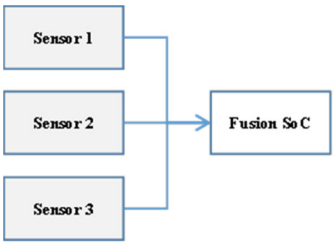

Figure 3: Centralized HW architecture

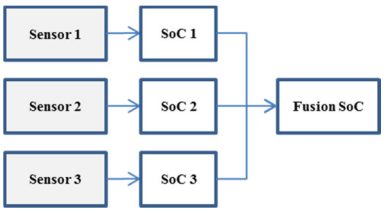

Figure 4: Distributed HW architecture

Whether centralized or distributed HW architecture is used, the processing or compute elements within a SoC would typically be heterogeneous in nature. One example of such a heterogeneous processing element SoC is the TDA2x SoC [6] shown in figure 5. It includes

- Dual-core ARM® Cortex®-A15 running at 750MHz
- Up to two dual-core ARM® Cortex®-M4 subsystems running at 212MHz
- Two cores of latest generation of fixed/floating C66x DSPs running up to 750MHz
- Up to four cores of Embedded Vision Engine (EVE) for vector processing.

It also integrates hardware for camera capture and Display subsystem resulting in better video quality at lower power. The TDA2x SoC also includes TI's IVA-HD technology which enables full HD video encode and decode, as well as dual SGX544 3D graphic cores capable of rendering 170 Mpoly/s at 500 MHz. It contains large on-chip RAM, rich set of input/output (I/O) peripherals for connectivity, and safety mechanism for the automotive market and offers lower system cost.

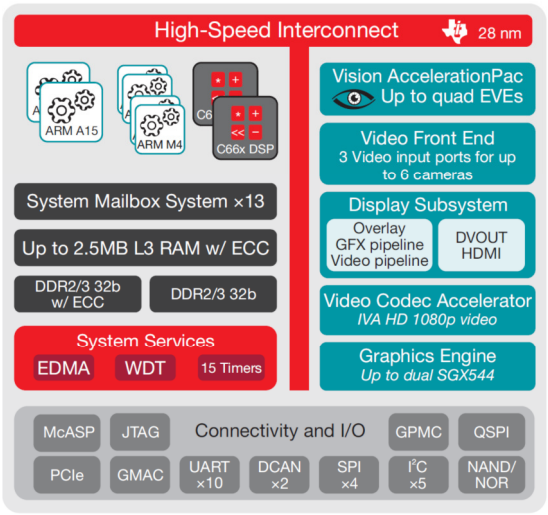

Figure 5: Block Diagram of TDA2x SoC

From a SW architecture point of view, two possible data flows are possible. Figure 6 shows centralized SW data processing. Here a "master" CPU submits work to worker "threads" on same or different CPUs. Characteristics of this approach include more finer control on data flow at master CPU, more dynamic data flow, more complex SW logic at the master CPU. Such a SW architecture is suited for fusion like processing where a fixed data flow is typically not followed and interactions between CPUs could be different depending on different sensor inputs and results at a given moment.

Figure 7 shows distributed SW data processing. Here different threads on same on different CPUs communicate directly with each other to make a data flow. Characteristics of this approach, lower SW overhead at a given core, reduced latency, simplified SW design, relatively static or fixed data flows. This is suited for an individual sensor processing data flow where the sequence of steps is typically well known for given sensor modality (Figure 4). HWA below refers to HW accelerator or similar module on a SoC.

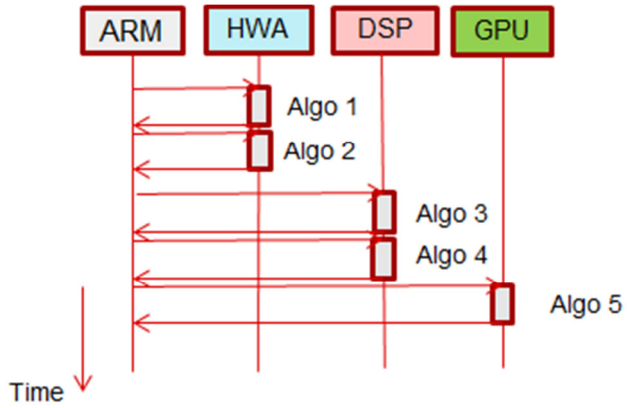

Figure 6: Centralized SW data processing

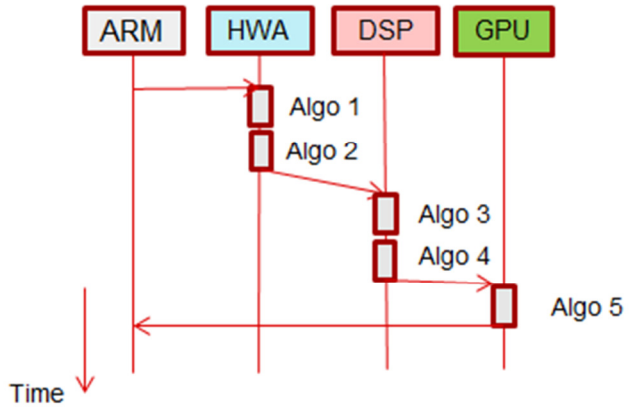

Figure 7: Distributed SW data processing

In autonomous driving, SW architecture would involve both centralized and distributed data flows. In this paper, we propose usage of OpenVX framework for the realization of system level SW data flows. OpenVX provides a graph-based definition of a data flow. Distributed SW data flows can be represented as graphs with multiple nodes. Nodes are connected to each other via data objects. Centralized data flows can be implemented by having the master CPU submit works as single node graphs to "worker" CPUs.

Further, OpenVX allows users to operate at a higher level of abstraction by hiding the lower level SoC details, this would allow the SW to scale to different SoC types and system configurations.

From a safety perspective, additional requirements need to be met by the software architecture. Static predictable systems are desired from a safety point of view, ex, no dynamic resource allocation, predictable control flow. In OpenVX, the data flow graph and required resources can be specified upfront during system initialization. Any system level parameter verification, scheduling choices can be done during graph verify stage. This allows optimization for a given SoC. At the same time, it makes the resource allocation static and execution predictable on the given SoC. Future specifications of OpenVX will support graph "import", "export" capabilities which will allow this verification step to be done offline instead of during system initialization in the final system. Unit level SW testing and verification is an important aspect towards achieving safe SW systems. Test harness can be developed around the graph API of OpenVX in order to test OpenVX nodes at a unit level before they are integrated at a system level. SW/HW in loop mechanisms can be used to ensure correctness of data processing nodes.

Freedom from Interference

Automotive software needs to be qualified as per ISO 26262 "Road Vehicles - Functional Safety" standard. This standard provides processes to identify and assess safety hazards in a system and establish, manage and track safety requirements to reduce risks to acceptable levels. ISO 26262 defines Automotive Safety Integrity Level (ASIL) - a risk classification system. It defines four ASILs – ASIL A through ASIL D - in increasing order of safety requirements. It also provides a classification level called QM (Quality Managed) for modules whose safety requirements are not critical.

In contemporary solutions, a single ECU is responsible for a variety of operations. A mix of software modules – a minority of which will be safety-critical – co-exist on a system. A majority of modules will be classified as QM. Such a co-existence is a major risk in a safety-critical system. A naïve solution is to make all software comply with the highest safety standard required by the system. However, this is not practical due to the very high complexity and large development costs. To address this, ISO 26262 allows a system with mixed criticality as long as it ensures "Freedom from Interference (FFI)" between different software modules so that errors in one module do not propagate into other modules. ISO-26262 requires a system to address three types of interference

- Memory usage
- Timing and execution order
- Exchange of information

A typical solution for interference on timing and execution order involves the use of task monitors and watchdog timer. Interference on information exchange is typically handled in software by using redundancies in messaging and using features like checksums to ensure the integrity of a communication channel. However, the problem of ensuring freedom from interference on memory usage does not have a simple solution on a heterogeneous multi-core SoCs. Solutions available in market rely mainly on MMUs and CPU modes and don't address interference across different cores in a system. In this section using TDAx shown in figure 5 as a case study we show how FFI can be implemented.

Consider a system that consists of a mix of QM and ASIL tasks. These tasks can exist on one or multiple cores. They share common memories. Memory space can be divided into two regions – QM and ASIL. ASIL tasks typically have read-write permissions to all memories. QM tasks will have write permissions to QM memories only. QM tasks will have read-only permissions for ASIL memories. Thus, the problem of preventing interference across tasks can be solved by providing means of defining a combination of read-only and read-write memory sections.

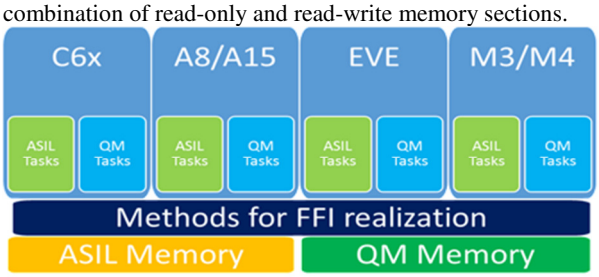

Figure 8: Typical TDAx SoC

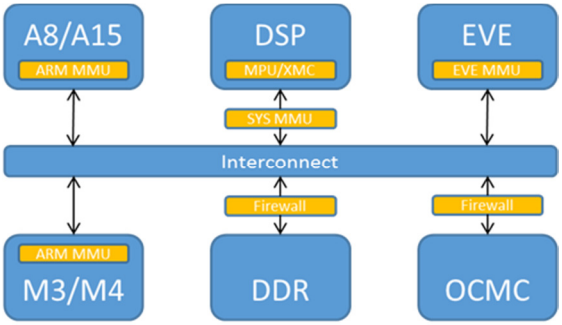

Figure 9: Summary of HW modules in TDAx for FFI

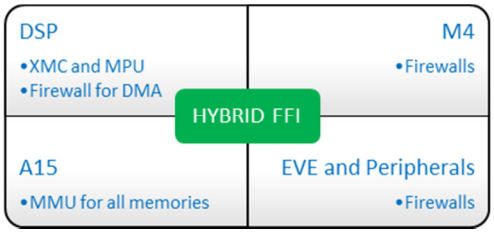

Figure 10: CPUs and HW modules used for FFI

The TDAx platforms provide three classes of hardware modules to control memory accesses:

- MMU (Memory Management Units) and XMC (Extended Memory Controller) for external memory accesses from a CPU subsystem
- MPU (Memory Protection Units) for internal memory accesses within a CPU subsystem
- Firewalls to limit accesses to memories to only a list of specified initiators

MMU and MPU

Cortex A15 and Cortex A8 provide a powerful MMU designed for High-Level Operating System (HLOS). This gives a straightforward solution for FFI by providing the ability to define read-only memory sections. It also provides a feature called Address Space Identifier (ASID) which allows different tasks to use different memory mappings and, therefore, allow seamless task switching.

DSP MPUs can provide isolation between QM and ASIL tasks for the internal memories based CPU mode (user vs. supervisor). DSP XMC provides the same protection mechanism for external memory accesses.

The M4 subsystem can use the CPU mode (user vs. supervisor) in conjunction with MMU and Firewalls to enable FFI.

Firewalls

Firewalls are a key means to support FFI on a heterogeneous multi-core system. They provide an ability to restrict access to memory regions based on master identification. This is a common use-case for non-compute HW units like DMA, Video Capture, and Ethernet in the SoC which may not use MMU.

EVE cores do not provide task identification via CPU mode (user vs. supervisor). Therefore, firewalls are used to enable FFI on EVE cores.

Run Time Application Monitoring

The ISO26262 functional safety standard [7] defines requirements which try to avoid or reduce the risk caused by malfunction of a Safety System. As the level of SW complexity increases, the standard emphasizes on test and verification of application SW components throughout the product lifecycle, including deployment in the field. Multiple techniques [8] have been proposed to catch systematic SW faults during development. But without a mechanism to monitor and report the state of the application, even when the application crashes or malfunctions, it is impossible to guarantee a fail-safe system.

In this section we describe a SW framework for Autonomous Driving applications using heterogeneous multi-core complex SoCs with the aim to provide:

- A centralized application SW monitoring framework
- A fail-safe mechanism to communicate monitored results from different CPUs to the central monitor
- A consolidated view of the overall system workload and data bandwidth by extending the scope of CPU specific libraries which provide information local to a given processing core.
- Support multiple different monitors to achieve higher confidence for SW and hardware fault detection.
- Application state like processing time, latency, number of frames processed/dropped, CPU low power time, etc. from multiple CPUs and HWAs.

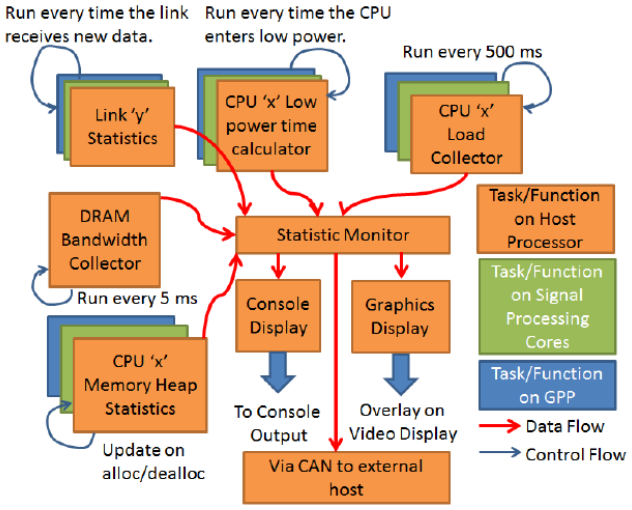

Figure 11: Framework support and flow for monitoring different system statistics

Centralized Monitoring of Multi-Processor ADAS SoCs

In order to describe the scheme for centralized monitoring, we consider a Multi-processor System on Chip (MPSoC) consisting of a host processor which acts as the application master, general purpose processors (GPP) and signal processing (SP) cores with associated vision/radar/compute accelerators. The overall SW infrastructure used to monitor system and CPU level statistics, key components, data and control flow between different monitoring tasks and functions executing on multiple CPUs is highlighted in Figure 11. Measurements include thread level statistics, CPU workload percentage, memory and heap status, DRAM bandwidth, CPU utilization and low power time, hardware errors like CRC, ECC failures.

A centralized statistics monitor task ($M_{Central}$) is created on the host processor. Individual statistic monitor probes, M^x_i, running locally on CPU_x register with $M_{Central}$. Depending on the kind of measurement made, M^x_i update data in their local buffers based on an application defined sampling interval (e.g. CPU workload, bandwidth measurement) or occurrence of an event (e.g. heap alloc/de-alloc, new frame received). In order to limit the amount of data transferred between local monitor tasks and $M_{Central}$, individual M^x_i pre-process the captured data by performing statistical operations like mean, maximum, variance and so on.

Based on the application defined logging interval, data is collected from different M^x_i sources and is collated on the host processor $M_{Central}$ task to forward the data to console output logger or the diagnostics framework on MCU side via CAN for further analysis.

Fail-Safe Communication of Monitored Results

Due to real time constraints placed on Autonomous applications, inter-CPU communication is often established using hardware assisted lowest latency interrupt based Inter-Processor Communication (IPC). Application crashes and hangs can often

cause this low latency communication channel to be out of service to send out state information for failure analysis. We recommend using a non-locking shared memory region queue for transferring commands and data between different monitor tasks.
The advantage of this approach is that the communication between monitor tasks (to detect and report errors) remains active even when the application runs slow or becomes dead. The non-locking nature of the communication ensures that $M_{Central}$ does not hang while trying to pull data from a monitor running on a crashed or hung CPU. Shared memory region based IPC also ensures minimal CPU overheads. The shared memory should be preferably allocated such that it does not share the same interconnect access path as the application data memories. For example, the developer can place the monitor IPC shared memory in internal SRAM while the application uses DRAM. This allows the monitor statistics to be reported to the MCU even in the case of a catastrophic hardware interconnect hang.

Conclusion

Autonomous cars are coming and they need to be safe in order for wide acceptance of this technology. There is no one solution which will make autonomous cars safe. Multi-sensor fusion and distributed HW architectures can make sure there is no single point of failure which causes hazardous situations for humans. SW architectures using frameworks like OpenVX can help implement multi-sensor fusion on distributed HW architectures. Since it's infeasible to make all the SW systems adhere to highest ASIL level, mixed ASIL systems need to be implemented using Freedom from interference techniques using HW units like MMUs, firewalls. Finally, real-time HW/SW monitoring and diagnostics need to be deployed to make sure the system behaves according to specification.

References

[1] C. A. J. B. D. B. C. B. R. C. M. D. J. D. D. G. T. G. C. a. G. M. Urmson, "Autonomous driving in urban environments: Boss and the urban challenge.," *The DARPA Urban Challenge,* no. Springer Berlin Heidelberg, pp. 1-59, 2009.

[2] A. N. F. a. B. B. Puthon, "Improvement of multisensor fusion in speed limit determination by quantifying navigation reliability," *In Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference,* pp. 855-860, 2010, September.

[3] R. a. K. M. Luo, "Multisensor integration and fusion in intelligent systems," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 19, no. 5, pp. 901-931, 1989.

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

[5] M. a. K. N. Aeberhard, "High-level sensor data fusion architecture for vehicle surround environment perception," *Proc. 8th Int. Workshop Intell. Transp.,* 2011.

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

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

[8] R. E. V. K. P. M. M. a. S. P. Gulati, "Resolving ADAS imaging subsystem functional safety quagmire," in *IEEE International Conference on Consumer Electronics (ICCE)*, 2015.

[9] A. Bolles, "A flexible framework for multisensor data fusion using data stream management technologies," *Proceedings of the 2009 EDBT/ICDT Workshops,* pp. 193-200, 2009.

Authors' Biography

Kedar Chitnis is Principal Software Architect for Automotive ADAS business at Texas Instruments (TI) Incorporated. His domains of interest are imaging, video, vision system software and frameworks for embedded solutions. He received Bachelor of Engineering (BE) degree from the University of Mumbai in 2001

Mihir Mody is Senior Principal Architect for Automotive business at Texas Instruments (TI) Incorporated. His domains of interest are video coding, image processing, computer vision & machine/deep learning algorithms with a focus on embedded HW & SW solution. He received Master of Engineering (ME) degree from Indian Institute of Science (IISc) in 2000 and Bachelor of Engineering (BE) from GCOEP in 1998.

Pramod Swami is Principal Engineer with Automotive group at Texas Instruments (TI). His domains of interest are hardware and software architecture, design and implementation of signal processing for video compression and computer vision. He received Bachelor of Engineering degree from MNIT, Jaipur.

Sivaraj Rajamonickam is Principal Engineer for Automotive ADAS business at Texas Instruments (TI) Incorporated. His domains of interest are video and system software for embedded solutions. He received Bachelor of Engineering (BE) from College of Engineering Guindy, Chennai in 2003.

Chaitanya Ghone is Software Systems Engineer for Automotive business at Texas Instruments (TI) Incorporated. His domains of interest are imaging, video, vision system software and frameworks for embedded solutions. He received Bachelor of Technology (B. Tech.) degree from Indian Institute of Technology, Bombay in 2006.

Biju MG is Senior Engineering Manager for Automotive ADAS business at Texas Instruments (TI) Incorporated. His domains of interest are imaging, video and vision systems. He received MS degree from BITS, Pilani and B. Tech. from Calicut University.

Yashwant Dutt is Automotive ADAS SW Apps and AVV Manager at Texas Instruments (TI) Incorporated. His domains of interest are imaging, video and vision systems. He received Bachelor of Engineering degree from BITS, Ranchi in 2000.

Aish Dubey is product manager for TI ADAS portfolio. His interest areas are safe automotive systems and algorithms, machine vision, robust sensor fusion and silicon architecture for real time ADAS. He received Master of Technology degree from Indian Institute of Technology, Delhi in 2002

Badri Narayanan is Principal Engineer in Automotive ADAS in TI. His domains of interest are real-time safety critical systems, networking and video. He received Bachelor of Engineering (BE) degree in 2001