Approaches to functional safety in automotive design

Abhinav Jain and Atulesh Kansal, NXP - April 15, 2019

In basic layman’s terms, automotive safety is intended to enable an entire automotive electronic system to continue functioning in the event of a component malfunction. The goal of automotive safety is to prevent a complete system failure that can lead to catastrophic consequences, including loss of life and property. The past decade, however, has witnessed a substantial increase in the number of parts in an automobile, primarily due to an increase in comfort levels and providing driving assistance. Further, the share attributed to electronics continues to increase as the concepts of driverless cars and advanced driver assistance systems (ADAS) take shape in reality. Hence, it becomes a fundamental requirement that the overall automotive system be foolproof, ensuring the safety of its riders.

Unfortunately, faulty system states cannot be prevented. System states can fail due to natural factors like component aging, environment, noise, or fabrication process defects. For automobiles, design goals instead must focus on functional safety. The formal definition, "Absence of unreasonable risk due to hazards caused by malfunctioning behavior of electrical/electronic systems" classifies functional safety in the globally accepted automotive standard ISO 26262 [1].

Such risks for failure are the responsibility of the entire value chain: from microcontroller (MCU) suppliers to original equipment manufacturers (OEMs), and to car assemblers, who all together must deliver a highly reliable system. In turn, a highly reliable system requires continuous monitoring of safety-critical elements, including reporting by every entity in the supply chain of any misbehavior and by initiating recovery mechanisms, either through software or in hardware, to return to the correct path [2]. The paradigm shift from a mechanical system to an electronically controlled system has lead to an exponential increase of electronic control units (a.k.a. MCU in modern automobiles). This increase in the share of electronics is pushing safety standards back into the semiconductor design flow, allowing MCUs to be designed so that reporting and recovery from any undesirable state occurs, potentially preventing fatal consequences [1].

The importance of functional safety pertains to numerous systems. Driver assistance, radar, and vision systems cause a modern automobile to continuously process data coming from multiple sensors. A failure in any of the sensors can mislead data processing systems and cause major life-threatening accidents if not addressed immediately. Therefore, sensors and the flow of data become safety-critical. Although not straightforward, even power windows require safety. Consider a case of a car accident resulting in a car fire. A person inside the car cannot quickly escape if the power windows fail. The person can neither open the door nor roll down the window glass to save himself. Another obvious candidate for safety is a braking application since any failure in the braking domain will lead to immediate and well-known consequences.
In this way, functional safety plays a major role in the modern automotive domain. Many failures can’t be avoided, but the after-effects can be controlled or avoided. In this paper, we identify a set of problems that can affect safety, present possible solutions with or without reporting mechanisms, and offer a system design approach for implementing functional safety.

**System states affecting safety and design strategies to tackle them**

There are a number of typical system problems that can result in failures that affect automotive functional safety. For each of them we provide the problem statement followed by solutions that can be implemented through the design process.

**Stuck in reset** -- Stuck in reset (SiR) is a condition where an MCU does not come out of reset to start its intended functions. An MCU SiR possibly gates the functionality of the connecting components as well, resulting in a system failure. Several possibilities can cause a state of SiR, stemming from both hardware and software sources. A few examples (from a hardware perspective) include clock failures, clock glitches, and environmental noise. On the software front, the software may be incapable of handling certain states like core lock up. Core lock up occurs when the core is stuck inside nested exceptions and cannot execute the functional code. This type of event may take the system into SiR.

Watchdog timers offer one proposed solution, and are often the best way to countermand the state
of SiR. Watchdog timers run in the background and are periodically reset by the software running on the MCU. In a SiR event, the watchdog timer count will not get reset and will thus time out. A proper design can use this time-out event to trigger an MCU, thus bringing it out of the SiR condition and aiding system recovery. Software watchdog timers are like a watchman that keeps on monitoring the activity of MCU in the background.

**Bit flips in memory** -- Memory, especially static RAM, is an integral part of a digital system and has a prominent share of the area in most MCUs. During read and write operations on the memory, which occur often, it is possible that a digital one (1) flips to a digital zero (0) or vice versa. Factors that cause bit flips can be component aging, external attacks, cosmic rays, etc.

Bit flips that happen in a safety-critical data path could be very dangerous. Hence, safety-critical memory should support error correctable codes (ECCs). The errors can be correctable or not correctable, depending on the number of bit flips that occur and the ECC algorithm used. A popular error-correctable code is the Hamming code, which can detect at most two errors, can correct only single-bit errors, and is quite easy to implement. Depending on the application and critical nature of the problem, more enhanced ECC algorithms may be used that are capable of correcting multiple bits as well.

**Clock variations** -- A continuous and clean system clock is an essential requirement for a system to work properly. Clock shutdown or “glitchy” clocks may cause the system to malfunction, thereby affecting system safety. Continuous monitoring of the clocks offers a viable solution.

The design of such clock monitoring units may vary: one design might measure the clock frequency; another may indicate if the clock is within a range. A very basic clock monitor design uses a separate reference clock to establish a user-configurable time period, and counts the clock cycles of the clock being monitored that occur during that period. The final count of the monitored clock is compared against upper and lower thresholds to conclude if the clock is within the safe range.

![Basic clock monitor](image)

**Figure 2** Basic clock monitor

**Power supply failures** -- The environment inside an automobile is very harsh, which may create instances where the power supply path gets broken or weak. A dead or poor supply voltage could be due to pins or balls (e.g., in a ball grid array package) that get damaged or de-soldered due to overheating, physical damage on a track or trace of the printed circuit board, or functional failure of the supply itself. The system should be able to flag such instances to prevent further mishaps.

To correct for power failures, “presence detectors” for each supply voltage should be integrated
within the MCU. These detectors need not be very precise, but must be capable enough to get triggered in a worst-case supply level drop. The reaction to such an event could depend on how critical the supply is to the system. For example, if the core supply is adversely affected, then the recommended reaction could be power-on-reset. On the other hand, if the supply is powering an Ethernet network, then it may be enough to flag a fault. The degree of criticality would also depend on whether or not there’s a secondary mechanism that will detect an incorrect state of a module that’s being fed by the supply. For instance, the ethernet has CRC checks on packets, timeout mechanisms, and so forth. Designing a supply presence detector could be as simple as biasing a transistor by the supply that’s being monitored.

![Figure 3](image-url) Basic structure of supply presence detector.

**Stuck logic and memory** -- The MCU is a semiconductor; all the logic works based on the controlled movement of electrons. Due to the possibility of a high level of noise in the surrounding environment, one cannot ignore the possibility of incurring soft errors. Soft errors are faults that are caused by energetic charged particles hitting the semiconductor, which can flip the flops from zero (0) to one (1), and vice versa, possibly causing the operating hardware to misbehave.

A built-in self-test (BIST) is one likely candidate for detecting the digital logic faults [3]. Digital BISTs have two types: Logic BIST (LBIST) and Memory BIST (MBIST). In LBIST [4], the logic is exercised by predefined patterns for a fixed number of clocks, then a signature is calculated and compared to a “golden signature” corresponding to the predefined pattern and number of clock cycles. In MBIST, checker patterns are written to the memory and then read back and compared to what was written.

BIST circuits can be implemented by using a special flip-flop known as scan flop to replace the actual flip-flops in the logic. A scan flip-flop can select test input when in BIST mode and works as a normal flip flop in functional mode. Also, these scan flip-flops are connected to form a chain of flip-flops known as scan chains [3]. A BIST controller can be included in the MCU design to enhance safety by exercising the patterns and checking results when triggered by software at regular intervals.

**Instruction word flip in the instruction read path** -- Another probable source of reaching an erroneous state is when an instruction word gets corrupted due to noise or coupling in the read path, voltage spikes, or ionizing radiation, among other incidents. When the core executes a corrupted instruction, it may give an exception, or any incorrect logic would get exercised. Such an
event could be catastrophic in a safety-critical MCU.

To tolerate such instruction faults, it is recommended that there be duplication in the processing. In duplication, multiple cores execute the same instruction and their outputs get compared in parallel. The probability that the same instruction fault would occur on all of the cores is pretty small. Therefore, this "lockstep" arrangement aids fault-tolerant system design by helping ensure word corruption is detectable.

An arrangement where redundant cores execute the same instruction after a fixed number of clock cycles is termed a delayed lockstep. The ideology behind delayed lockstep is that voltage spikes are of very short duration; thus any corruption of the instruction will be caught when it is executed in the next clock cycle. Following the ideology, the lockstep can also be extended to other hardware to catch any random failures.

**Recovery strategies**

Continuous monitoring of the MCU’s critical components alone may not suffice to implement safety. Any out-of-order event that’s detected by the monitor needs to be reported, but reporting should also trigger an appropriate corrective action. An obvious and very simple approach would be to logically OR all these signals and connect them to the reset of the MCU. In this case, the recovery mechanism of any faults will be the same -- a reset -- which might be excessive. Another approach could be based on individual signal detection and triggering of its recovery mechanism rather than logical OR of all the signals. This approach would lead to a single recovery path for every instance of the fault occurrence, however, which might not be required under all circumstances.

Because each event on every instance may not require the same corrective path, some flexibility is often desirable. For instance, the SiR condition does not need be reported if the MCU is in reset, and only a higher reset can bring it out of the reset condition. On the other hand, an incorrectly executed instruction can be recovered from without resetting the MCU. To support the different paths of recovery, depending on how crucial the error-affected path is, an additional level of the centralized reporting scheme becomes important.

This centralized unit collects all the reported faults and generates different recovery paths or "reactions," implemented in hardware or software according to the configuration that’s been programmed. For example, the clock monitors keep track of different clocks in the system. For a system clock monitor, the reaction to a fault that the monitor reports must be to generate a reset to the system, since the system clock is the most crucial clock. In case a fault is generated by a monitor that’s minding the clock of a communication IP, however, the system may only need an interrupt triggered to the core directing the software to take appropriate corrective action – such as flushing the FIFOs and initiating retransmission. This centralized fault control unit, then, becomes the most critical safety IP in the entire MCU, providing flexibility in selecting the type of reaction needed for a fault.

_Atulesh Kansal is a lead design engineer, and Abhinav Jain is a design engineer, both at NXP Semiconductors._

**References**

1. Functional Safety Methodologies for Automotive Applications, Cadence
2. Understanding the Automotive Functional Safety Best-Practices with ISO 26262 standard, embitel

Related articles:

- Autonomous vehicles: The electronics road to making them safe
- Redundancy for safety-compliant automotive & other devices
- ISO 26262 certification first for automotive MCU
- Lighting technology for the modern automobile
- 10 steps to selecting a microcontroller