Silicon evolution for the automotive revolution

By Andrew Hopkins,
Director of System Technology

White Paper

1. Automotive revolution

The convention of vehicle ownership and operation is ending. It started with ownership through finance and leasing, now ride and car sharing schemes increasingly remove the need for occasional use cars within cities. The next step is fully capable Mobility as a Service (MaaS) deployments. MaaS is revolutionary in several ways:

- Primary mobility shifts from car ownership to a service model
- Vehicles are operated by service providers rather than consumers
- Service providers are fully accountable for vehicle functionality, safety and security
- Consumers become disconnected from vehicle manufacturers unless their brand forms part of the MaaS
- Deployment of robotized taxis becomes practical

Taxi companies who have invested in smartphone apps and operate their own fleets will argue that MaaS is here already. They interface to the consumer to provide the service, they operate the vehicles and take responsibility for their safety. Consumers don't need to drive, and there is value add through a nice person to help with luggage and provide anecdotal information and support. For sure, it's a form of MaaS and provides a consistent platform for a quality taxi business, however it's not a trigger for revolution.

The mass deployment of robot taxis will be revolutionary and goes beyond the engineering masterpiece and breakthroughs in artificial intelligence. In economies where labor costs are high, robots lower the cost of MaaS to a level where owner operation of vehicles ceases to make financial sense. That's a seismic shift, where contract mobility expands greatly at the expense of declining car purchases by consumers.

As with any change, MaaS unleashes opportunities and threats. Demand for the humble taxi driver seems likely to decline progressively over the coming decades, allowing them time to retrain. Original Equipment Manufacturers (OEMs) are those with the most to lose and an established business to adapt.

Autonomous vehicles are forecast to create a Trillion USD market for ride services and media content delivery by 2030, in addition to traditional car platforms already valued around two Trillion USD [1]. With market growth focused on autonomy, the executives of large OEMs, who are charged with upholding brand legacy and generating shareholder value, must invest in automation and content delivery. Staying connected to consumers throughout the transition to MaaS is also pivotal to capturing value, and key to fending off the advances of the even larger tech giants.

2. Opportunity for semiconductors

Regardless of who succeeds in deploying robot taxis, there is a huge opportunity for the semiconductor industry [2]. The sensory and mental abilities of humans are sophisticated and set the benchmark for reliable driving. Provided the driver is in good health and paying attention, the likelihood of an accident for the car occupants is very low in most developed countries. For example, in the UK, 787 fatalities resulted from 254 billion miles driven in 2017 [3].

Driver occupancy monitoring can ensure drivers are alert, as well as keep an eye on their health, such as detecting a heart attack or intoxication. Arm's open source Arm NN software and advanced processing capabilities in Arm Cortex-A76AE, Arm Cortex-65AE and new Neural Processing Units (NPUs) are an ideal compute solution for such machine learning applications.

"Driver occupancy monitoring can ensure drivers are alert, as well as keep an eye on their health, such as detecting a heart attack or intoxication."

Taking humans out of the loop requires many sensors including cameras, LiDAR and RADAR. Each sensor requires front-end processing, and subsequently more processing to perceive the environment. Robot taxis will deploy multiple sensor technologies, their inputs will be fused with other data to form a model of the environment that enables planning of the path ahead. Cortex-A76AE and Cortex-65AE are good solutions, for perception and path planning respectively, because together they deliver the absolute performance and throughput efficiency needed.

With a course plotted, vehicles must then decide on the commands given to their actuation systems to proceed safely. In summary, autonomous systems have four main steps:

- Sense
- Perceive
- Decide
- Actuate

In parallel to the revolutionary jump of robotic taxis, the automotive industry is also trying to improve the reliability of human driven cars. Progressive evolution of Advanced Driver Assistance Systems (ADAS) means that classes of minor accident, like nudging the car in front or colliding with a car when changing lanes, will become a thing of the past. ADAS is another opportunity where semiconductors are pivotal, and with a market value CAGR of over 18% forecast for the foreseeable future, it is an attractive market [2].

3. Functional safety

The high growth of the automotive market is attractive to semiconductor vendors. An accompanying challenge is how to develop systems suitably, where safety is a paramount concern. The practice of ensuring that products operate safely and continue to do so even when they go wrong is called functional safety engineering.

There are standards to guide functional safety engineering, such as ISO 26262 for automotive electronics and IEC 61508 for industrial electronics. ISO 26262 defines functional safety as “The absence of unreasonable risk due to hazards caused by malfunctioning behavior of electrical / electronic systems”. This means that systems must function correctly, with potentially unsafe faults detected and controlled to prevent a hazard. Predictability of failure modes is expected to enable thorough analysis.

The main concepts for functional safety are [4]:

- Identify level of risk
- Determine level of robustness in design
- Develop and test following guidelines
- Verify and validate
- Perform independent assessment
- Manage change and modifications

Products must be specified and developed suitably for their criticality and have a welldefined safety concept [5]. ISO 26262 defines four categories to guide the engineering activity, each an Automotive Safety Integrity Level (ASIL) [6].

Each level from ASIL A to ASIL D requires an increased level of engineering rigor. ISO 26262 makes various recommendations of the methods to apply, although alternatives can be used with justification.

As a generic rule of thumb, the following categorization can be made:

- Safety nominal
  - Helpful rather than essential
  - User can act to avoid hazards if aware of fault
  - Usually ASIL B, could be ASIL A

- Safety critical
  - Relied upon to always function or fail safely
  - Significant risk of hazard and loss of life if the fault is undetected
  - Usually ASIL D, could be ASIL C

Each development must describe its activity through the gathering of evidences in a safety case. The case must be scrutinized by an independent party or assessor, considering the argumentation made, good practice, and where applicable, guidance of relevant standards.

The Arm partnership has a strong track record of automotive design, predating the introduction of ISO 26262 in 2011 and subsequent revision in 2018. Quality engineering and dissemination of good practice are essential to functional safety, so multiple experts within Arm contribute to working groups and influence relevant standards using its knowhow, relationships and ecosystem.

4. What could go wrong?

Failures can occur throughout a system, in the design or implementation of a hardware component or software module, the way the pieces are integrated, or the conception of the overall system architecture. Faults are classified as systematic or random.

Systematic faults are those inherent in the design and relate to the previously listed safety concepts. They occur in hardware, software and at the system level in consequence to deficiency of the development, such as:

- The design and verification methodology
- Incorrect or ambiguous specification
- Training of people to understand the concepts and methodology
- Tools used during the development

Random faults occur during use and are caused by manufacturing, aging and circumstance. They occur in hardware and cause disruption to software. They can be permanent or transient and require management at the system level. Examples include:

- Permanent faults due to physical defects
- Transient faults due to environmental factors like radiation
- Intermittent physical defects are also classified as transient faults
- The statistical failure of asynchronous crossings are also transient faults

Figure 1 illustrates examples of systematic, random permanent and random transient faults.

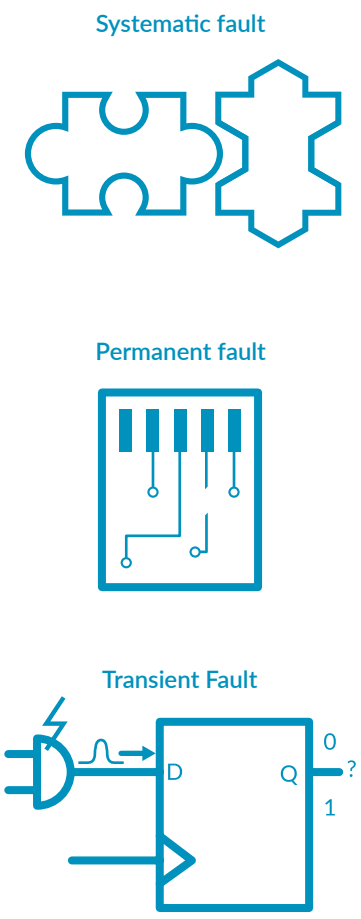


Figure 1
Example faults

5. Systematic faults

Developing the system and its components in a rigorous way is the primary means to the avoidance of systematic faults, including a strong verification methodology tailored to the intended safety integrity level [4]. Verification of functional safety designs must always be thorough, however for the higher integrity levels traditional stimulus-based testing techniques are expected to be surpassed through use of formal or semi-formal methods.

"Arm has long been a leader in verification of silicon IP and uses formal techniques extensively"

Formal techniques typically represent the design specification using mathematical constructs, such as algebras or finite state machines, which can be reasoned about logically using tools. Ideally the specification will be machine-readable, for example a modelling language, which enables tools to automatically check the correctness and verify that the implementation matches the model. Semi-formal specifications have more structure than natural language, they include pseudo code and structured diagrams or models. Semi formal definition reduces ambiguity and enables humans to reason about the design.

Arm has long been a leader in verification of silicon IP and uses formal techniques extensively. Thus, Arm's journey into functional safety has focused on other aspects, such as increased detail and rigor in requirements management and traceability for the design and verification data. The value of this work to functional safety goes beyond the obvious aspects of confirmation and evidence, as it significantly improves maintainability of the design and reproducibility of the design flow. Managing complexity in such a systematic and controlled way also facilitates the realization of more complex systems, regardless of whether safety is a requirement.

Development methodology is the only mitigation fully under the control of software developers. However, hardware can help guard against common software related defects. For example, Memory Protection Units (MPU) and Memory Management Units (MMU) can detect erroneous data accesses. When software is developed out of context, developers can also place requirements on the hardware capability through their assumptions of use. If the software is developed in parallel with the underlying hardware platform, a Hardware Software Interface (HSI) specification can be defined.

6. Decomposition

Developing and verifying software for higher levels of integrity (ASIL D) is very challenging. System level architecture is one way to mitigate limitations, including the practicality of developing a large complex software system, in its entirety, to the quality expected for ASIL D applications. ASIL decomposition provides a set way to subdivide a system into several more tractable independent requirements, each with a lower safety integrity than the original.

Practical decompositions include the development of two different software programs to perform the same or similar system function. Each program must be developed and executed independently, thus reducing the likelihood of a common cause failure to an acceptable level. ISO 26262 provides guidance on ASIL decomposition and most practical applications are to address the systematic capability.

7. Random fault detection

Random faults are detectable using hardware mechanisms that are typically supported at the system level by software. In some cases, the mechanism is mostly software based, utilizing minimal enabling hardware. Random faults in practice can be:

- Single point faults – one failure, such as an open or short circuit
- Multiple point faults – one or more failures, several simultaneous single point faults
- Latent faults – multiple point faults scenarios that only occur under fault conditions

Once the symptoms of a fault are detected, more thorough diagnostics or corrective action can be taken by the system. Detection mechanisms, also known as diagnostic capability, may include continuous detection or periodic tests. Examples include:

- Continuous detection
    - Temporal and/or spatial replication with comparison of results, e.g., Dual-Core Lockstep (DCLS)
    - Information redundancy, such as Error Detection/Correction Codes (EDC/ECC)
    - Algorithmic fault tolerance
    - Redundancy at the system level, including so called software lockstep

- Run-time diagnostics
    - Functional testing using software (Software Test Library)
    - Scan based Built-In Self-Test (BIST) of logic or memories
    - Periodic test pattern checking

Standards like ISO 26262 guide designers through the definition of metrics to quantify whether the diagnostic coverage is suitable.

- Single Point Fault Metric (SPFM)
- Latent Fault Metric (LFM)

The level of coverage is a safety design choice based on the system's functional requirements and safety integrity level. Although safety standards typically propose or mandate target diagnostic metrics, the failure rate is a vital characteristic. Failure rate is defined by safety standards as Probabilistic Metric of Hardware Failure (PMHF). It can be expressed as Failure In Time (FIT) where one FIT is the number of failures within a billion (109) hours. Table 1 shows the metrics recommended by ISO 26262. Automotive safety architects can use the PMHF and other argumentation to deviate, above or below, the recommended metrics for each part of the design and thus the overall system.

Table 1: Proposed system-level metrics for automotive systems

TARGET METRIC	ASIL A	ASIL B	ASIL C	ASIL D
SPFM	Nominal	≥90%	≥97%	≥99%
LFM	Nominal	≥60%	≥80%	≥90%
PMHF	≤1000 FIT	≤100 FIT	≤100 FIT	≤10 FIT

8. Hardware metrics for decomposed systems

Regardless of safety architecture, the system still needs to achieve its target metrics for detecting random faults. Decomposition can be applied to lower the detection capability needed for a specific piece. The principle being that if two independent capabilities are executed in parallel, each with modest detection capability, the combined system has enhanced detection.

In complex cases like autonomous vehicles, the metrics achieved by decomposition can be very hard to prove, so hardware is increasingly expected to address the detection requirements of the highest safety integrity even if the software is decomposed. Hardware suitable for the system's highest safety integrity significantly simplifies the safety architecture and safety case.

To address the needs of autonomous vehicles, Arm has expanded its range of lockstep capable processor cores to include the high-performance Cortex-A family. Lockstep is explained in detail later in this paper.

Starting with Arm Cortex-A76AE, it is now possible to execute complex algorithms, computer vision and machine learning with the diagnostic capability expected for ASIL D systems provided directly by the hardware. It delivers the simplification of diagnostic coverage needed to realize the most complex of decomposed systems. Where part of a system requires only an ASIL B capability, the Split-Lock processor is configurable without lockstep under software control.

ISO 26262 defines a nomenclature to represent decomposed requirements [6]. It comprises the ASIL of the original requirement and the lower decomposed requirement. So, for example, if an ASIL D requirement is decomposed to two ASIL B requirements it will be expressed as ASIL B of D in the form ASIL B(D).

There is also an associated portfolio of automotive enhanced system IP to enable the compute needed, comprising:

- Arm CMN-600AE Cache Coherent Interconnect
- Arm GIC-600AE Generic Interrupt Controller
- Arm MMU-600AE System Memory Management Unit

9. Fault detection mechanisms

The most suitable mechanism to detect and potentially correct a fault depends on the circuit to be protected.

Information redundancy

One of the best-known techniques is Error Detection/Correction Codes (EDC/ECC), which includes parity, extended hamming codes [7], Cyclic Redundancy Codes (CRC) [8] and Reed-Solomon codes [9]. They introduce information redundancy at the expense of slightly increased data. EDCs are highly effective at detecting faults with bit cells within memories and ECC enables correction of the localized transient faults inevitably induced by radiation. Permanent faults can also be tolerated using ECC.

Parity and Hamming codes are relatively low cost, typically requiring about 13% overhead for a 64b data chunk, comprising bit cells to store the code and logic to generate and check it. EDC is less effective at detecting the multi-bit errors caused by single point faults in the address decoder, which makes 99% SPFM hard to achieve using EDC or ECC within a single memory.

Similar capabilities and limitations exist for interconnects, where single point faults in elements, such as cross-bar switches and routers, can cause multi-bit errors beyond the Hamming distance of the error detection codes typically used. EDC, usually parity, is also effective at protecting registers.

Software Test Library and test patterns

Fault detection mechanisms don't have to be hardware based. Consistency checking can be performed using software within the application framework. Alternatively, software routines can functionally test the hardware by exercising its features in a consistent, deterministic way that achieves a quantifiable level of diagnostic coverage.

"STLs require low-level access to the hardware and on a CPU can require the highest privilege to broaden the logic within its reach. "

Test routines, known as a Software Test Library (STL) or sometimes Software BIST (SBIST), are widely used for automotive microcontrollers to periodically test the processor and other components during windows of quiescence. Achieving the target coverage of permanent faults in complex blocks, such as performance processors with deeply nested logic, for ASIL B use cases is challenging and can be impractical. STLs can operate with or without disturbing the system's state, although have the advantage that they can be aborted quickly if the system must urgently respond to an interrupt event.

Test patterns directed at specific low-level circuits are a typical part of an STL, they can be efficient and provide fine-grained control over testing and enables targeted testing following a fault. Tests can also be high-level, such as a test pattern for an image processing pipeline or graphical display unit. Test patterns can also be applied directly by the hardware, with or without software control.

STLs require low-level access to the hardware and on a CPU can require the highest privilege to broaden the logic within its reach. Where there is an operating system, the STL will be tightly coupled to ensure it is run at a time that minimizes disruption and availability to service interrupts. Coverage can often be enhanced by inserting bespoke hardware blocks that make hard-to-reach logic accessible.

Testing can be event driven, such as between tasks or during idle time. In time triggered systems a dedicated time slice will be allocated to testing, as shown by Figure 2. Following a quantum of computation, a context switch (C/S) will occur to start the next quantum. If testing is due the context will be saved, tests run on the processor, then a context restore. When the next quantum starts, an application will be context switched to the processor.

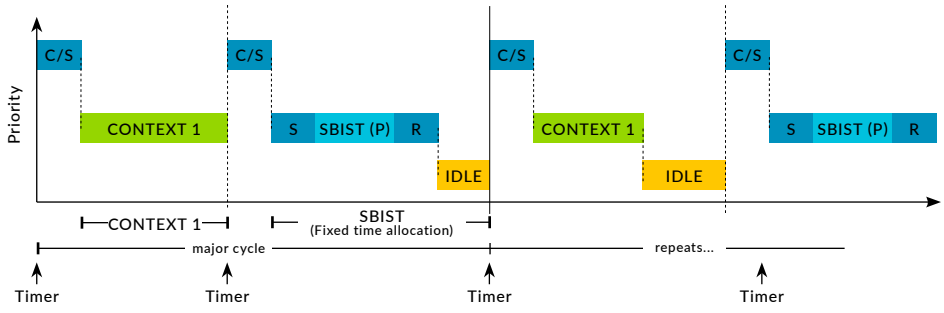


Figure 2: Use of a Software Test Library in a time triggered system

Run-time scan diagnostics

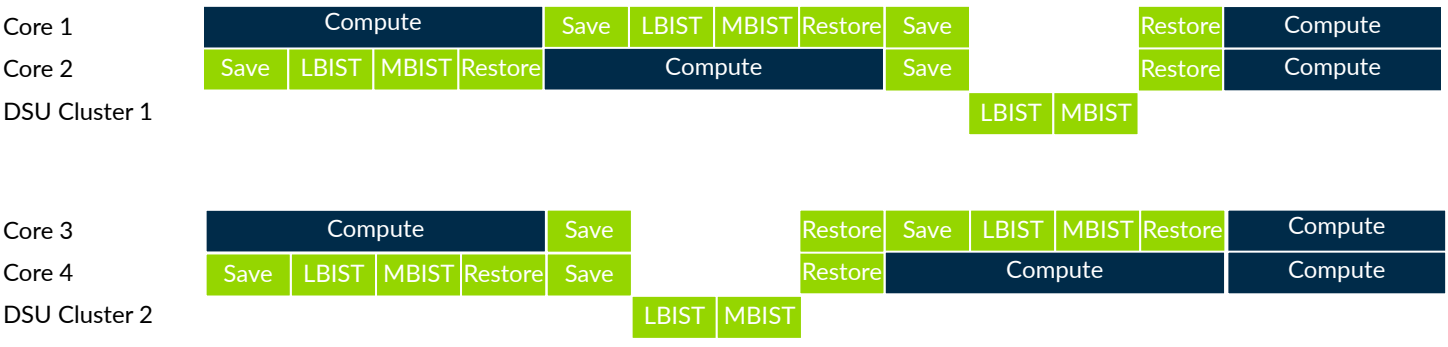
When diagnostic coverage is needed for an ASIL B system, many system designers look to STLs. They can be applied after development of a chip, are within the control of the system designer, and are well known due to their extensive use in cost limited MCU systems.

What many system designers don't realize is that for complex hardware, scan-based techniques are often more suitable, which is why they are applied extensively for advanced System-on-Chip (SoC) systems. Scan techniques readily achieve 90% diagnostic coverage for all sizes of design, whereas an STL could be as low as 50% coverage. The high coverage makes scan techniques a good fit for testing periodically at run-time, as well as diagnostic checks at key-on, key-off and following fault detection by another mechanism, making scan techniques suitable for SPFM and LFM coverage of logic. Periodic test of logic will be complemented by dedicated memory testing using Memory BIST (MBIST), and in practice a combination of MBIST and STL with coverage "topped up" by LBIST is often applied.

There are two approaches, Logic BIST (LBIST) and directed scan test, that both exploit the scan-chains designed into the SoC for post-manufacture functional test. They require integration of a hardware controller for testing the SoC and software control for run-time use. The affected parts of the system must be quiescent, so software is needed to suspend the relevant activity, which might be to stop or hot-detach a processor.

Availability of compute is important, so run-time diagnostics are scheduled to ensure that at least one CPU is always available. Testing of shared cluster logic, such as cache controllers, requires all cores in the cluster to be suspended, which motivates at least two clusters within the system. Figure 3 illustrates an operational sequence to test two CPU clusters, each with two cores, whilst maintaining CPU availability. It shows testing of the shared logic in the DynamIQ Shared Unit (DSU) of cluster 2 with both its CPUs suspended. In the meantime, compute occurs in cluster 1, ensuring availability of at least one processor.

Figure 3: Save and restore sequence for periodic testing



LBIST relies on scanning sequences generated from a seeded pseudorandom number generator or similar. Generating the sequence in the device eliminates the need to store large test patterns. A limitation is that some circuits, such as comparators, are impractical to test exhaustively using random sequences.

Directed scan testing can be tailored to test all types of circuit and can be very efficient because the sequences are purposely generated. The obvious downside is the memory needed to store the sequences, although many complex systems already have very significant off-chip storage and memory making this approach attractive. The directed approach also has a fine granularity similarly to most STLs used to test CPUs.

All types of scan-testing have two significant limitations. The power required to scan the sequences at speed is higher and more concentrated than normal operation, and this needs to be managed carefully by running the scan-chains with a lower frequency than the system clock, often tens of MHz.

Periodic testing including STLs and scan-based techniques provide no direct coverage of transient faults, however incorrect flip-flop states lying dormant within the component are flushed by the local test and reset sequence. Transient fault coverage can be improved by ancillary mechanisms such as watchdog timers, software consistency checks and selective flop-flop hardening. Design tools can help designers identify the most critical SoC flip-flops to swap for hardened cells or triple redundant flip-flop macros.

Memory BIST

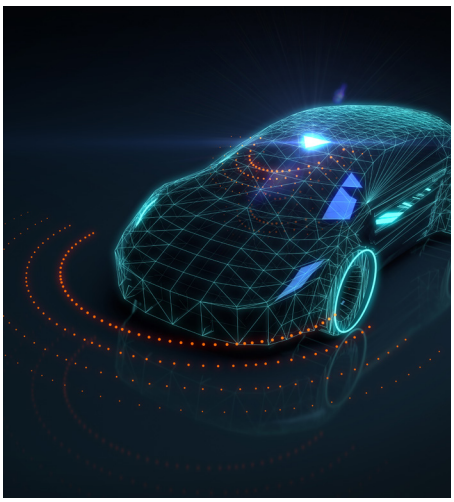
Before the system starts it's important to identify whether memories are free from permanent faults. Complementary to the continuous protection of memories using EDC/ECC is MBIST. It enables more precise testing to pin-point the location of bit-cell faults and distinguishes cell faults from other classes.

MBIST is traditionally an offline technique, however Arm has developed an online capability that can execute test sequences alongside the system with minimal impact [10]. Available in Arm Cortex-R52, the major advantage of this technique is increased system availability compared to offline tests, which is especially valuable for diagnosis following EDC faults if they are frequent.

Bespoke detection mechanisms

One of the more labor-intensive ways of realizing detection mechanisms is to design a specific circuit to detect the expected failure modes. In most cases it's desirable to have a small targeted mechanism that works at a high-level to detect faults. An example of where that fits is on-chip interconnects where modest transaction consistency checks and information redundancy can provide moderately high coverage of faults relating to sequence, loss, insertion and corruption.

Some faults are hard to detect abstractly, so the hope of low-cost fault detection is often



unachievable, with many discrete mechanisms needed to build up the coverage, especially for the highest coverage targets. Bespoke mechanisms take more time to design and verify, although special cases such as flip-flop and register protection using EDC have a regular structure and so easier to realize. Register protection using EDC can even be inserted by the SoC integrator within the netlist using commercially available design tools.

Bespoke mechanisms are often believed to be a golden solution until investigated and the facts laid out. Many mechanisms increase area and power costs to the point where duplication is often cheaper to both the silicon integrator and the IP provider.

Lockstep

Lockstep is the duplication of a circuit or component, such as a processor, more than once with comparators to check that all relevant outputs are consistent between the copies. The aim is to detect a broad range of faults and achieve 99% SPFM in the primary copy of the logic within a few cycles.

Comparison of additional internal signals can improve the fault detection time. Faults in the secondary lockstep copy are detected by the primary copy which provides a degree of latent fault coverage, which can be enhanced by a Software Test Library that exercises a significant and quantifiable part of the design.

Transient faults can affect clock and power networks such that identical circuits suffer a common fault. Offsetting the lockstep copy in time by a few cycles provides temporal redundancy to reduce such faults. Spatial diversity is also necessary, so the chip floorplan must be structured to keep the two copies separate.

Lockstep is a straightforward way to achieve very high diagnostic coverage. The downside of duplicating large components is considerably increased area and power cost. Area can be saved by sharing the larger memories between the instances and protecting them with an ECC, as shown by Figure 4. This can lower the area of a lockstep processor or controller to less than double the design without lockstep.

Although lockstep is often seen as an expensive “brute force” approach, smaller scale duplication can be the most cost-effective option. The alternative of bespoke checkers has a high design cost and passing signals between small circuits is expensive because it necessitates additional consistency checking. Such checking can cost more area than duplicating the whole circuit and consume significantly more dynamic power within the intermediary checkers.

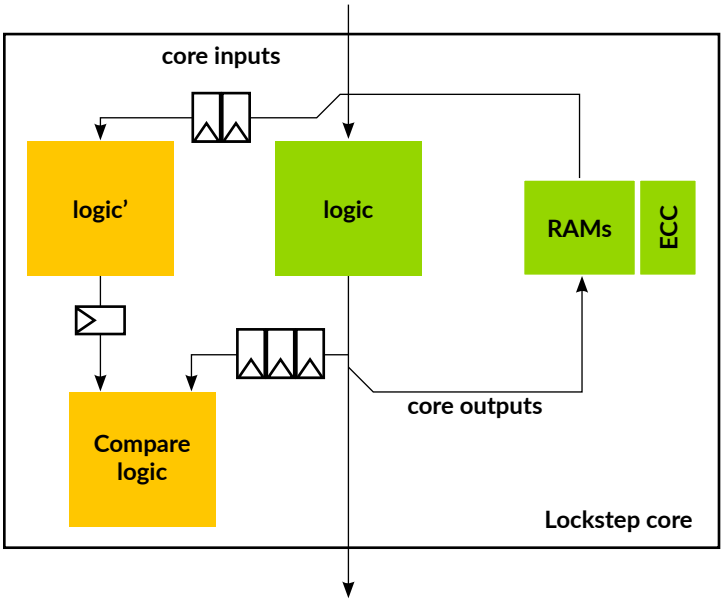


Figure 4: Dual Core LockStep processor with shared memories

Split-Lock

An enhancement to the lockstep approach is so called Split-Lock, where it is possible to disengage the secondary copy of the circuit when quiescent, such as during boot or a reset state, freeing it for use as an additional system resource. The technique is especially applicable to processors because it allows the system designer to select whether they have high-diagnostic coverage, or increased compute performance at a lower coverage.

Split-Lock is available for all of Arm's recent lockstep processor cores including Cortex-A76AE and Cortex-R52. Besides increased flexibility for system designers, its valuable to SoC vendors because they can target a broader range of applications with a single device.

A further advantage, especially relevant to autonomous vehicles, is the ability to improve availability by falling back from lockstep to 'Split' configuration as part of a reboot sequence, all under the control of software. In many core systems significant flexibility is possible, and software can often be relocated to the most appropriate core at runtime. Moreover, with careful isolation of memory to contain faults, as enabled by CMN-600AE, part of the system can remain operational, with the failed part taken offline for diagnostics and rebooted. Such flexibility can be employed to create effective system recovery strategies. However, there will always be significant challenges at the system-level if full functionality is to be maintained, because lockstep alone provides detection of faults, rather than preventing corruption of memory within permitted address regions.

10. Why safety documentation packages?

Licensable silicon IP is typically developed to be generic, without knowledge of how it will be used, besides limitations laid down through assumptions of use. Such generic developments are classified by ISO 26262 as Safety Element out of Context (SEooC) [11] and to enable safe use there must be documentation to describe it and the product's enabling capabilities.

For each functional safety development, Arm creates a functional safety documentation package to enable silicon partners to complete a safe development of their SoC and prove they've met their functional safety targets. ISO 26262 sets strict limitations on use of so-called black box IP, which has unknown provenance. That means the IP must ideally have been developed for functional safety, following ISO 26262:2018 part 11, and include appropriate supporting collateral; or be a quality managed development with sufficient supporting evidence to support qualification under ISO 26262 part 8 clause 13.

Safety documentation packages are specific to each product following a common format for ease of use. They comprise a safety manual, a Failure Modes and Effects Analysis (FMEA) report, a Development Interface Report (DIR), an example Failure Modes and Effects Diagnostics Analysis (FMEDA) and Dependent Failure Analysis (DFA) reports which can be adapted by the SoC designer to reflect their specific instantiation. Collectively the documents describe:

- The process followed including description of confirmation measures
- The fault detection and control features
- Assumptions of use
- How configuration parameters impact the design
- The interface between Arm and the SoC designer

Safety cases are hierarchical in use, requiring input from each IP supplier. Figure 5. shows how the documentation, such as the FMEA, provided by Arm and other IP suppliers supports silicon partners to enable the tier-1s and OEMs.

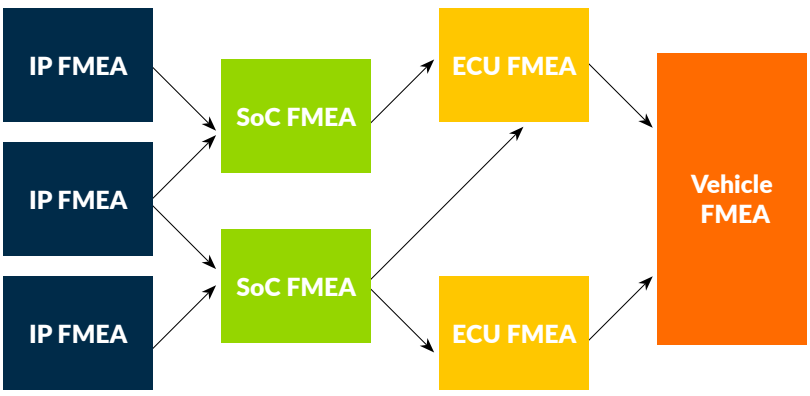
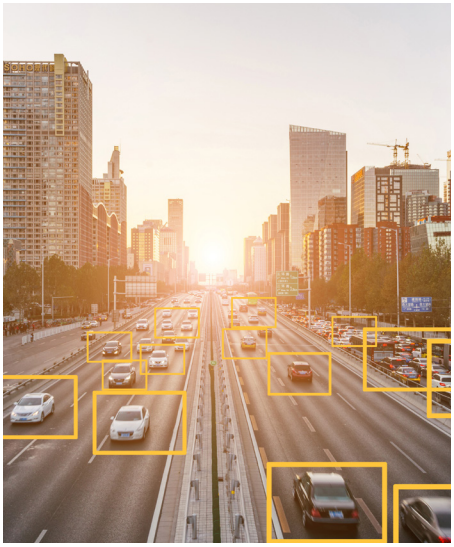


Figure 5: Arm's FMEA deliverables enable the wider automotive market



11. Conclusion on the future of SoC for the automotive revolution

The automotive market is undergoing several significant phases of change. Vehicle differentiation is now primarily delivered by software and compute integration, rather than electro-mechanical aspects. The uptake in ride hailing applications and the prospect of robotic vehicles is an even more disruptive change, that will rewrite the competitive landscape for road vehicles.

Enabling this new wave of computing requires unprecedented levels of computational performance, and Arm has developed new CPUs and a highly capable line of NPUs to enable it. The real “game changer” is to deliver the performance needed at a variety of safety integrities, so SoCs can be deployed in ADAS applications with immediate volume, and robotic taxi and autonomous vehicle platforms as they emerge.

Arm Cortex-A76AE and associated system IP were designed specifically for these applications, enabling ASIL B to ASIL D capable functions through the flexible Split-Lock processing capability. When in Dual Core LockStep mode, ASIL D software is enabled, and the diagnostics for ASIL B(D) decomposed systems are also made more practical to prove. Moreover, in many cases the high-performance with lockstep removes the burden of ripping algorithms to pieces to form the decompositions, as needed for other processor architectures. When in Split mode, ASIL B applications are supported, making Cortex-A76AE a highly versatile choice.

With these new processing capabilities now being designed into SoCs by lead partners, the automotive revolution is on-track to safely deploy the world’s greatest achievement in artificial intelligence.

12. References

[1] UBS Global Research, “Who will win the race to autonomous cars?”, Q-Series market report, 8th May 2018.

[2] IHS Markit, “Automotive semiconductor market tracker”, 18th July 2018.

[3] UK Department for Transport, “Reported road casualties in Great Britain: 2017 annual report”, 27th September 2018.

[4] International Organization for Standardization, “Road vehicles - Functional safety, Part 2: Vocabulary”, ISO 26262 Part 2, December 2018.

[5] D. Sexton, A. Priore, J. Botham, “Effective Functional Safety Concept Generation in the Context of ISO 26262”, SAE Int. J. Passeng. Cars – Electron. Electr. Syst. 7(1), 2014

[6] International Organization for Standardization, “Road vehicles - Functional safety, Part 1: Vocabulary”, ISO 26262 Part 1, December 2018.

[7] R.W. Hamming, “Error detecting and error correcting codes”, Bell System Technical Journal, 29(2): 147–160, 1950.

[8] W.W. Peterson, D.T. Brown, “Cyclic Codes for Error Detection”. Proc. of the IRE. 49(1): 228–235, January 1961.

[9] I.S. Reed, G. Solomon, “Polynomial Codes over Certain Finite Fields”, Journal of the Society for Industrial and Applied Mathematics (SIAM), 8(2): 300–304, 1960.

[10] A. Becker, Short burst software transparent on-line MBIST, IEEE 34th VLSI Test Symposium (VTS), 25-27 April 2016.

[11] International Organization for Standardization, “Road vehicles - Functional safety, Part 10: Vocabulary”, ISO 26262 Part 10, December 2018.