BORN TO ENGINEER

Unlocking Linux Instrument Cluster Potential

Making clusters safer – a journey of innovation

– Automotive Team @ QuEST

QuEST Global

contents

Unlocking Linux Instrument Cluster Potential

Abstract

Cluster design and development has seen an explosion in terms of innovation and adoption of multiple technologies and experimentation on different device and technology integrations. However, the safety aspect of clusters and in-car infotainment has also been thrown into sharp relief, especially after the Fiat Chrysler hacking and the subsequent recall of over 1.4 million vehicles.

Such a product recall has set Fiat Chrysler back by billions, and will most certainly result in ripple effects reaching other places that seemed reasonably secure. One of the most affected areas is perhaps experiments on using open source platforms like Linux in subsystems such as infotainment head units and instrument clusters. Due to the economic impact of this massive recall, the excitement about Android for Automobiles as well as the open source projects underway for instrument and car computer development has been lackluster.

We at QuEST Global, however, believe that there is still significant potential in terms of end-user benefit if we are able to get a Linux cluster compliant to ISO 26262 standards. In this paper, we explore some of the challenges to be overcome during development from scratch, or in porting of a cluster based on a proprietary OS, to a Linux kernel. We also outline some possible approaches to overcoming these challenges, effectively as well as cost efficiently.

Background

With open source being increasingly adopted in technology markets across the world, its scope is still being explored. And the question of why the automotive sector, too, cannot adopt open source, persists.

The path of the instrument cluster has gone from different devices functioning independently, to a cluster of instruments, to a controlled cluster, to a cluster integrated with the HMI and other control systems of the automobile, unified on to an on-board computer, and integrated with multiple systems on the cloud.

This path has opened up multiple worlds of opportunities of integration and overlap of functionalities, especially with the Internet of Things. Furthermore, the advent of head-up displays and advancements in connected vehicle technology have ended up creating a plethora of scenarios where technologies have overlapped and thus become greater than the sum of the parts. A fine example is the sharing of control of the automobile between the car itself and the mobile app.

However, with technology components interacting with each other, with absolutely no human supervision or monitoring, there are very real risks of hazards during the use of these programmable devices and components. Whether these hazards are caused by a system malfunction or a human intervention, the risks need to be considered and mitigated. This has given rise to the concept of functional safety and ISO 26262 – the framework for achieving functional safety compliance.

In this paper, we aim to explore the process of cluster development, and how functional safety can be achieved cost efficiently, in the development of Linux-based clusters.

The Cluster – and its Criticality

What people call the instrument cluster is no longer where the speed, rpm, fuel, and engine temperature gauges are located. It has grown to include a variety of information and analytics, such as:

- Alerts based on analysis of in-vehicle sensor data collected in real-time
- Infotainment integration that allows for connectivity to the internet, in-vehicle Wi-Fi, as well as connectivity to cloud platforms and applications
- Integration to media players and streaming devices
- Integration to mobile devices such as laptops, tablets, and mobile phones, further extensible to include wearables and other embedded systems
- Integration to other in-vehicle systems such as ADAS (Advanced Driver Assistance Systems)

The point here being that the principal function of the instrument cluster, as well as Human Machine Interface (HMI), has in many ways redefined itself. Today, the cluster's purpose includes a whole lot more. From merely giving information to the driver, to being the hub of all information-routing in the vehicle.

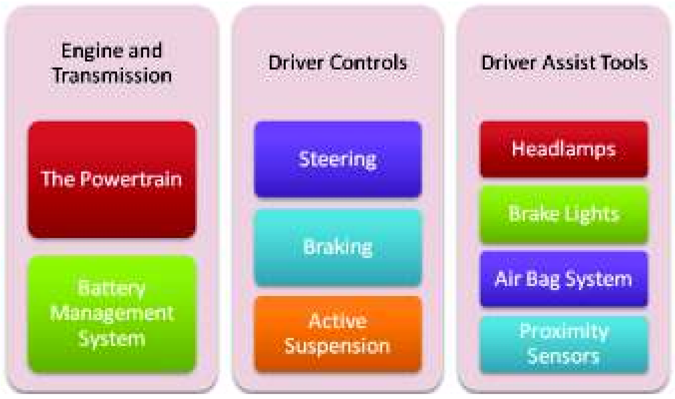
Fundamentally, the instrument cluster has increased in criticality. It now integrates all the in-vehicle systems to the world around it.

© 2016, QuEST Global Services

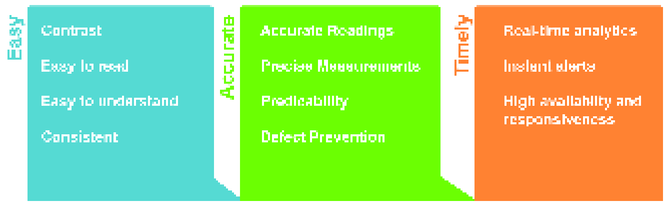
Unlocking Linux Instrument Cluster Potential

Designing the Cluster

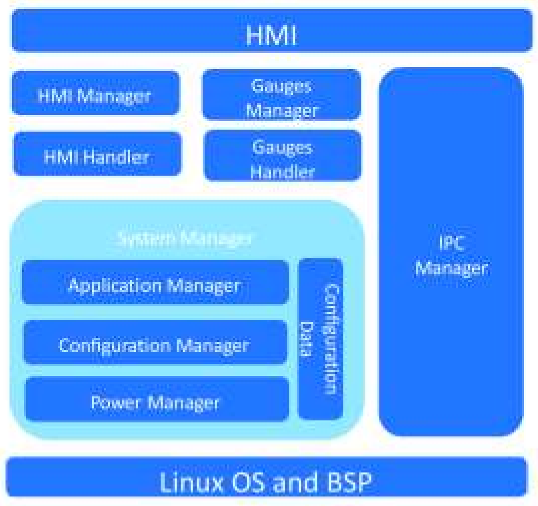
In order to design the cluster, we integrate some of the following in-vehicle functions:



Some of the principal features the cluster should possess are:



The journey to this goal starts with the development of the Operating System-compatible hardware and Board Support Package (BSP). A potential cluster architecture based on Linux OS can be designed as follows:



Functions and Functional Safety

Now, let us take a step back and look at the functional safety aspect of clusters. Since the cluster has transformed from being just a few dials, to a Car-Computer or Carputer, with features and functions, using Bluetooth, GPS, maps, and proximity sensors to name just a few. The result is an explosion of complexity in the architecture of the cluster, and of course, its functions.

Functional Safety

The concept of Functional Safety was introduced to ensure no damage to human life or property would be caused during the operation of the equipment. Specific to automobiles, Functional Safety pertains to operations of automotive equipment, including the components of the cluster and the in-vehicle infotainment (IVI) and other engine and drive train components.

Here the safety standards prescribe the following:

1. The Automotive Safety Integrity Level required for each of the devices that together form the cluster
2. Ensuring the hardware, software, and safety functions are individually and collectively performing in accordance with the purpose of their design

Functional Safety Compliance in Clusters

Development Stage Challenges

The first set of challenges are faced during the development phase, where the system architecture, design, development, and integration come under the scanner.

The first challenge here lies in the initiation of development from the concept stage, and the hazard analysis and risk assessment. The complexity lies in the fact that the devices, the operating system, the applications, and the interfaces must all be designed, developed, integrated, tested, and validated within the confines of ISO 26262 guidelines.

© 2016, QuEST Global Services

Unlocking Linux Instrument Cluster Potential

Here, we use a framework called IDEAS, which helps understand the landscape and scope for compliance testing.

IDEAS stands for:

- **Identify** – In this stage, we specifically pinpoint the integration points between different physical devices, the operating system, HMI, applications, and other components. Each of these integration points are to be tested for the entire set of permutations and combinations of information and command flows.
- **Document** – Documenting the integration points identified for testing ensures the test scenarios and cases are as detailed and exhaustive as possible to ensure 100% compliance.
- **Execute** – Each test scenario is executed along with the corresponding data sets and commands issued during the course of the drive.
- **Analytics** – The test data is then analyzed and validated against projections and limits. This analysis also helps identify potential risks, which are resolved by revising the risk mitigation plan with the new risks.
- **Simulation** – Simulating models and scenarios based on the test cases helps complete the cycle of testing, where scenario discovery also happens alongside validation of test data.

At the initiation of development, the safety plan prepared during the concept phase is revisited and revised according to the findings of the IDEAS testing and validation sequences. This augmented safety plan ensures better compliance with the standards required for an ASIL C or B certification.

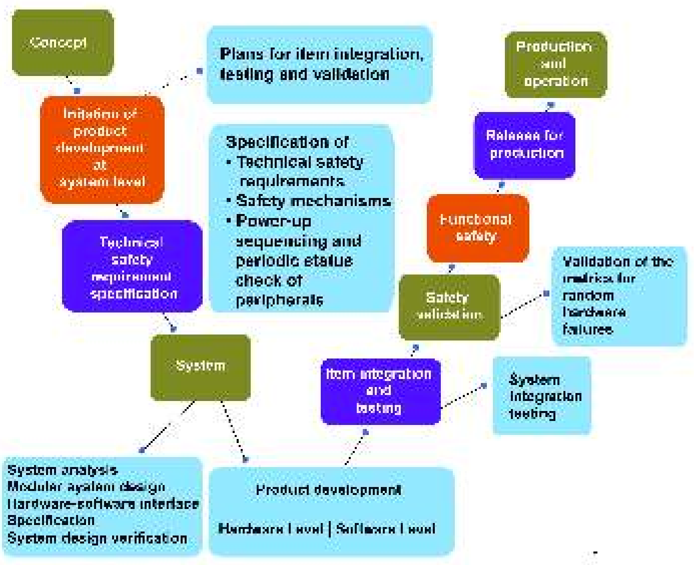
Functional Safety Concept

The second challenge lies in the specification of safety requirements, from the safety goals. We use our cyclic analysis approach combined with agile development. This ensures documenting of each safety requirement and outlining the specifications for each one, as well as the diagnostics to be run during the power-up sequence.

System Design

The third challenge lies in the design of the system itself. The complexity arises from the fact that the system has to be nimble, modular to allow for seamless integration, as well as extensibility, and yet be safe and secure, both from within, as well as with the connected environment.

To ensure proper system design, we plan to follow the V model of Development with additional checks. The sequence of activities that needs to be carried out in a typical V Model development for System Design compliant to ISO 26262 is as given in the figure below.


Concept
Plans for item integration, testing and validation
Production and operation
Initiation of product development at system level
Specification of
• Technical safety requirements
• Safety mechanisms
• Power-up sequencing and periodic status check of peripherals
Release for production
Functional safety
Technical safety requirement specification
Safety validation
Validation of the metrics for random hardware failures
System
Item integration and testing
System integration testing
System analysis
Modular system design
Hardware-software interface
Specification
System design verification
Product development
Hardware Level | Software Level

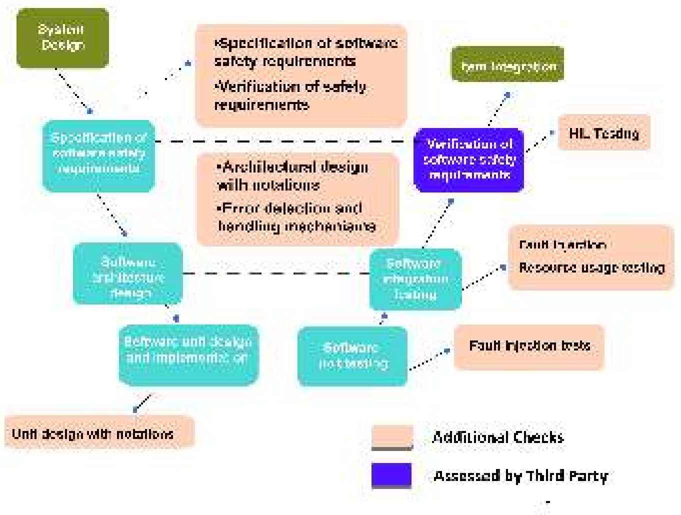
Typical V Model System Design

As per this workflow, all of the development phases (Requirement Analysis, Architectural Design, Detailed Design and Coding) shall be mapped to the respective types of tests as in a typical V model of Development.

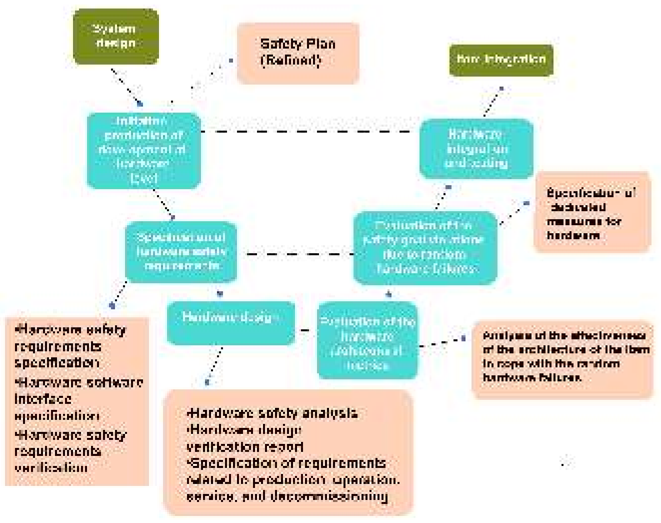
© 2016, QuEST Global Services

Unlocking Linux Instrument Cluster Potential

ISO 26262-Compliant Software Development Process and Additional Checks

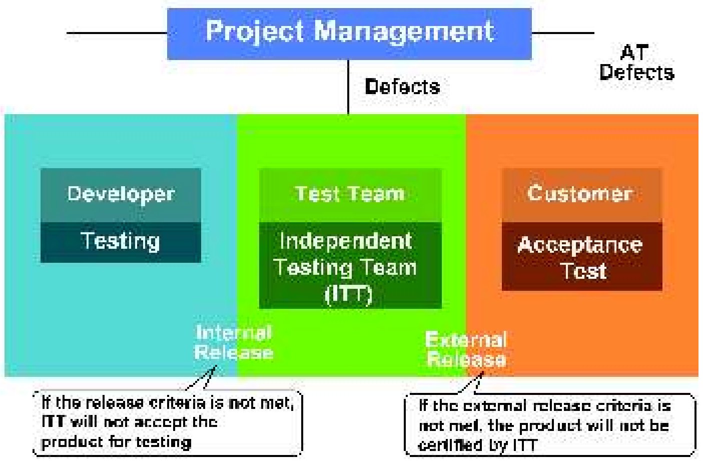

System Design
•Specification of software safety requirements
•Verification of safety requirements
Item Integration
Specification of software safety requirements
Verification of software safety requirements
HIL Testing
•Architectural design with notations
•Error detection and handling mechanisms
Software architecture design
Software integration testing
Fault Injection
Resource usage testing
Software unit design and implementation
Software unit testing
Fault injection tests
Unit design with notations
Additional Checks
Assessed by Third Party

ISO 26262-Compliant Hardware Development Process and Additional Checks


System design
Safety Plan (Refined)
Item Integration
Initiation of product development at hardware level
Hardware integration and testing
Specification of dedicated measures for hardware
Specification of hardware safety requirements
Evaluation of the safety goal violations due to random hardware failures
Hardware design
Evaluation of the hardware architectural metrics
Analysis of the effectiveness of the architecture of the item to cope with the random hardware failures
•Hardware safety requirements specification
•Hardware software interface specification
•Hardware safety requirements verification
•Hardware safety analysis
•Hardware design verification report
•Specification of requirements related to production, operation, service, and decommissioning

Hardware Development Process and Checks

The V-Model is an extension of the software development methodology of the Waterfall Model with a specific emphasis on continuous correspondence with the testing team. The connection between developer testing, independent testing, and acceptance testing is as depicted in the figure alongside.


Project Management
AT Defects
Defects
Developer
Testing
Test Team
Independent Testing Team (ITT)
Customer
Acceptance Test
Internal Release
External Release
If the release criteria is not met, ITT will not accept the product for testing
If the external release criteria is not met, the product will not be certified by ITT

Development to Prototype and Production

Taking the development concept to a prototype, and eventually into production, poses an entirely new set of challenges, such as testing and validating performance as well as failures.

System Integration Testing

The principal challenge in prototyping is in the fact that even successful trials on the prototype vehicle do not guarantee a successful transition to production.

Safety Validation

Validation that the safety plan is working as described is a huge responsibility on the automobile manufacturer, OEM, as well as other stakeholders, and is of paramount importance. Here, each safety specification has to be validated against real-life scenarios, including situations where human error is involved.

As compliance requires the components to behave within the defined hazard levels even in the event of a human error, the validation of technical safety specifications is quite an uphill task.

Furthermore, the criticality of the function also warrants that the safety validation be performed by an external agency, in order to maintain lack of bias, as well as eliminate all possible errors.

The most important aspect of validation is the behavior of the cluster under varied degrees of stress and load. Testing and validating behavior on events of

Unlocking Linux Instrument Cluster Potential

random hardware failures, and to start with, simulating these scenarios is also a massive responsibility in itself.

QuEST Approach

We have set out on a quest for developing the building blocks of an ISO 26262-compliant, Linux-based cluster, while ensuring cost efficiency in the process of compliance-achievement. Our approach to this goal includes a three-part strategy, outlined below.

Subset of Hardware Mechanisms to Achieve Safety Compliance

One of the challenges of the cluster is validating each data stream that is received, detecting errors, as well as fixing the errors in transmission. Here, we propose the use of a multilevel validation system that encompasses:

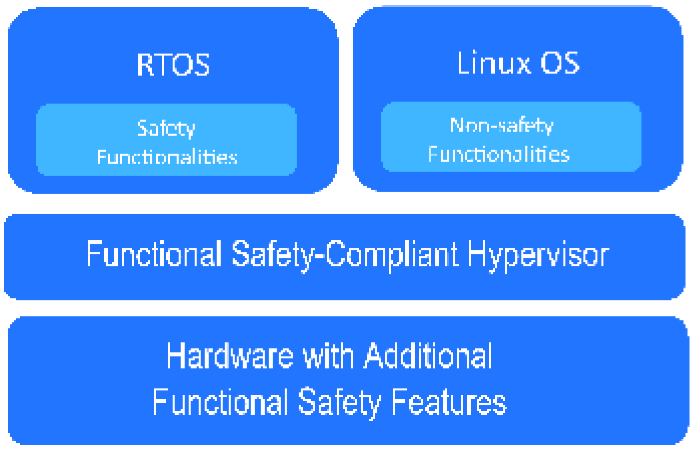
- **Parity Bit Checking** – Using an odd or even parity bit added to each data packet, we are able to perform error checks at a the fundamental level. Using such a layer-by-layer validation, the data validation accuracy increases manifold.
- **Memory Monitoring** – Using error detection and correction codes (EDC), we can detect single-bit and multi-bit failures in a stream. In order to achieve this, each word of the data stream is extended by additional bits of redundant data that produce an altered hamming code at a minimum distance of four. This helps identify each word against the transmitted data stream.
- **Modified Checksum** – Here, checksums are created by preset algorithms that use all the words in the memory, thus helping in validating data streams and detecting single-bit errors.

Possible Architecture Options

Another significant addition to enhancing safety of the cluster is the use of an ISO 26262-compliant hypervisor. Using the triumvirate architecture model that uses the OS, the hypervisor, and the software as the basis for fulfilling its goals, the cluster performance can thus be segregated into safety-critical and safety-noncritical functions.

These functions are then rerouted based on the functional safety level required, thus ensuring a contiguous compliance with safety specifications both at the process, as well as at the device level.

Furthermore, the addition of the ISO 26262-compliant hypervisor adds an additional safety layer by serving as an intermediary between commands and the automobile's safety critical processes.


RTOS
Safety Functionalities
Linux OS
Non-safety Functionalities
Functional Safety-Compliant Hypervisor
Hardware with Additional Functional Safety Features

Design Options

As system design is a significant contributor to safety, we propose to leverage our IDEAS framework that emphasizes the use of simulation to discover issues as well as solutions. Our designs are based on statistical and numerical models that ensure the design by itself will lend security and robustness to the system.

This, in addition to the segregation of safety-critical sections of code, instantly increases the failure-safety quotient manifold – but we do not plan to stop there. We propose to add multiple layers of both redundancy and isolation in order to ensure that minor or even negligible variance from the safety compliance curve will not result in a hazard.

System Monitoring

Combining some of the above-stated safety additions to a continuous monitoring system, we propose to achieve an even greater degree of fail-safety. The data generated from system monitoring is further analyzed for errors and risks, which in turn contributes to enhancements in design and other safety features.

© 2016, QuEST Global Services

Unlocking Linux Instrument Cluster Potential

Redundancy

As a design objective, we aim to include redundancy in the system, both at a hardware, as well as a software level in order to allow the system to enjoy a significant degree of robustness, when it comes to handling errors of data creation or transmission.

The Road Ahead

On the system design and hardware side, internally, we have developed a Freescale i.MX6-based, generic hardware platform that can function as a starting reference for a high-end instrument cluster design. On the software side, we have ported the Linux distribution to our hardware platform and other EVKs, and we are now in the process of developing some of the building blocks required for making it an ISO 26262-compliant, Linux-based cluster. Our team has presented a related paper titled **Case Study: Verifying an Existing System compliant with ISO 26262,** which outlines the roadmap of our goals and aspirations of this project at the **Embedded Safety and Security Summit (ESSS**) organized by LDRA India in Bengaluru, in June 2015.

QuEST

BORN TO ENGINEER

www.quest-global.com

© 2016, QuEST Global Services