Component-Level ASIL Decomposition for Automotive Architectures

Alessandro Frigerio
Eindhoven University of Technology
Eindhoven, The Netherlands
a.frigerio@tue.nl

Bart Vermeulen
NXP Semiconductors
Eindhoven, The Netherlands
bart.vermeulen@nxp.com

Kees Goossens
Eindhoven University of Technology
Eindhoven, The Netherlands
k.g.w.goossens@tue.nl

***Abstract*—The Automotive industry is evolving towards a more electronics-assisted driving and self-driving functionality. The addition of complex subsystems has a great impact on the current vehicle architectures, leading to safety concerns. In this work we present a technique that follows the ISO 26262: Road Vehicles - Functional Safety standard to introduce redundancy in the architecture by using ASIL decomposition, and perform a safety analysis of the modelled system. A three-layer model is used to describe the application, the resources, and the physical space of the vehicle. In this paper we introduce novel model transformations to replicate parts of the application following ASIL decomposition rules. Finally, we perform a cost analysis and a probabilistic fault tree analysis on the architecture, making a comparison between different possible solutions. The advantages of these techniques, such as traceability and scalability, are shown by modelling and analysing the lateral control application of a real truck platooning system.**

***Index Terms*—ASIL decomposition, EE Automotive Architecture, Functional Safety, Probabilistic FTA**

I. Introduction

According to the SAE automation levels [1], a level 4 or 5 autonomous vehicle is required to perform all driving functions without the necessity of an immediate response from a backup driver. To achieve that, the system has to be safe, and redundancy has to be introduced to have backup modules that can complete the driving task even when a failure in the main module occurs. Different types of redundancy can be used, for example spatial or temporal redundancy. In this paper we focus on functional redundancy: a function is realized by diverse implementations, both in terms of software and heterogeneous hardware, to minimize the risk of a combined failure.

An important decision during the design of an autonomous vehicle is choosing which type of architecture to use. For example, a *centralized architecture* [2] consists of a single powerful central unit that performs all the necessary operations, such as sensor data fusion and vehicle control algorithms. The network will experience congestion near the central unit compared to a *distributed architecture* [3], where the sensors and actuators are connected to distributed controllers to minimize the cable lengths and the network congestion. Many variations are possible in this, both in the architecture (*domain based, zonal architecture, etc.*) and in the network topology (*ring, star, etc.*).

With our work we provide:

1) A methodology to explore generic automotive architecture solutions by using correct ASIL decomposition techniques, as specified by the ISO 26262 standard [5], in an automated way on a modelled system.
2) A complete Fault Tree Analysis (FTA) that includes the generation of a fault tree and the probabilistic FTA that calculates the probability of failure of the total system, and the Common Cause Faults analysis to identify possible violations of the independence rules required for redundancy.
3) A cost analysis of the system, with the possibility to tailor the cost metrics to fit any specific scenario.

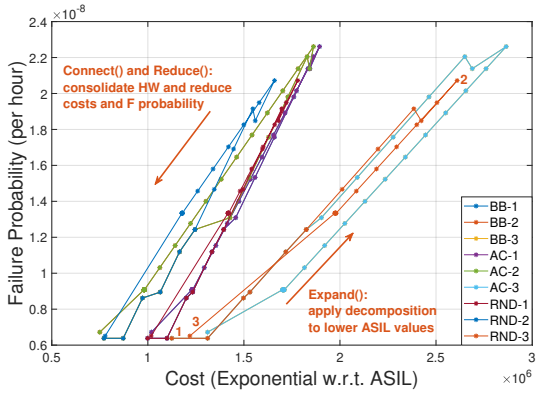

Fig. 1: Analysis of different level of safety for a lateral control application in a truck platooning system.

As a result, it is possible to evaluate the trade-offs between the safety and the cost of the architecture at each step during its development. Figure 1 shows the results of our techniques applied to a real ASIL D lateral control application used on the truck platooning system described in Section VIII. Each point in the plot corresponds to the cost and the failure probability of a particular architecture that implement the lateral controller application, with different levels of redundancy obtained by using our model transformations. The model used in this work is based on [4] and is described in Section IV.

As described in Section III, multiple types of ASIL decomposition are allowed. For example, the curve RND-3 in Figure 1 repeatedly decomposes ASIL D components into ASIL B + ASIL B or ASIL A + ASIL C components. The starting position 1 corresponds to an ideal system where only ASIL D components are used. This situation is often not possible to implement because of the non-availability or the cost of ASIL D component, especially for general purpose application. The *Expand()* transformation described

in Section VII substitutes with redundant implementation the selected modules, and in this case the cost metric described in Section VI leads to an increased cost of the system. The effect of having a redundant system composed by multiple less reliable components is also to increase the system probability of failure, and this depends on the failure rates assigned to the system, described together with the failure probability calculation in Section V. By repeatedly applying the *Connect()* transformation we can reduce the number of separate redundant blocks by merging them together, reducing both costs and failure probability of the system. In the final position 3, only ASIL B redundant components are used for executing the initial application functionality, while simple and cheaper ASIL D specific components are only used to manage the redundancy. The other curves in the plot correspond to different types of ASIL decomposition or different cost metrics, which lead to different costs and failure probabilities for the same initial system.

Finally, Section IX will explain the results of the experiments performed on the lateral control application of Section VIII.

II. Related Work

In this section we present works that relate to this paper in terms of modelling of autonomous vehicle architectures, ASIL decomposition, fault tree analysis, and application of the ISO 26262 standard in the design of automotive systems. As mentioned in the introduction, ASIL decomposition is generally used as a top-down method applied on the Functional Safety Requirements (FSRs). In [6] an overview of the ASIL decomposition technique is given, focusing on points of the standard that could be misinterpreted. In [7] and [8] the authors describe a methodology to allocate the ASIL values that are assigned to a FSR to the architecture components following the decomposition rules. In [9] the ASIL decomposition technique is applied to a hybrid braking system using the Hip-HOPS tool for safety analysis and design optimization. Similar work is done also in the aerospace field, as in [10] where the authors describe a method to allocate Development Assurance Levels. As opposed to these works, in our work we use the ASIL decomposition to modify the system architecture in order to accommodate a decomposed FSR.

In [11] design space exploration of a system architecture is performed with SMT solvers to improve the system safety level by integrating safety mechanisms which are similar to the transformations we use in this work to introduce system redundancy. In [12] the design space exploration is performed in an AUTOSAR environment, in which Software Components are allocated to the hardware modules in the architecture and a time analysis that considers the execution time and the scheduling of the components is performed. In this paper we assume that the allocation of the application is given, and that it can be modified via the ASIL decomposition rules.

We base our automotive architecture models on [4], extending that work with a complete FTA and Cost Analysis to evaluate design trade-offs in a quantitative manner.

In terms of fault tree analysis for automotive systems, [13] describes an algorithm to generate the fault tree from a system description. Our fault tree generation described in Section V is based on this work. In [14] a generic structure for the failure of an automotive component is described, with details on how to refine the abstraction level on the hardware resources failure modes.

The use of ASIL decomposition does not always follow the correct rules and practices, as shown in multiple publications such as [15]. With our work we offer automated model transformations that follow the rules of ISO 26262, as well as validation and quantification techniques for any automotive architecture described by the model of Section IV, regardless of how it was generated.

III. ISO 26262 and ASIL decomposition

The ISO 26262: *Road Vehicles - Functional Safety* is a standard published in 2011 to address functional safety of automotive systems. It takes into account the full product lifecycle, from the concept idea to production and post-production operations, according to a V-Model [5]. The standard defines the *Automotive Safety and Integrity Level* (ASIL) as an adaptation of the Safety and Integrity Level (SIL) from the IEC 61508 standard [16]. ASIL is a risk-based parameter that classifies an hazardous scenario into five levels, where QM (Quality Management) is the lowest and D the most critical, based on *Severity*, *Probability of Exposure* and *Controllability*.

The standard describes the ASIL decomposition procedure as a technique to reduce the criticality of an FSR by splitting it into multiple redundant requirements, each with a lower ASIL value. Figure 2 shows the possible decomposition patterns that are illustrated in the standard. The notation *ASIL X(Y)* is used to track the original FSR and to ensure that the proper system level analysis is performed at ASIL Y level.

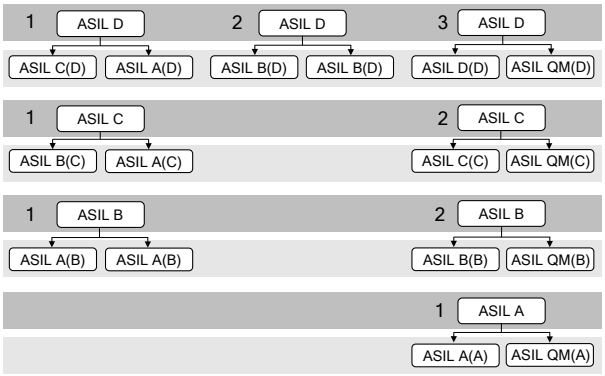

Fig. 2: ASIL decomposition patterns.

As the standard suggests, the ASIL decomposition technique is generally applied on the requirements level. In this work instead we use this technique in a later phase of the V-Model, applying it on component level, modifying a given architecture instead of allocating the requirements to existing components. An FSR can be related to one or more functionalities, which are implemented in the automotive system. In the following section we establish which rules are required to apply the decomposition technique on this lower level, ensuring that the

required independence between the redundant components is achieved.

IV. Automotive Architecture Model

To perform a system-level safety analysis of an automotive architecture, information about the functionality of the system and its implementation are both required. In particular, we need the ASIL requirements and specifications of the software and hardware resources to carry out an ISO 26262-based analysis. Moreover, environmental information such as temperature, vibrations, EMI, etc. is required to establish *Freedom From Interference* (FFI) [5] and to validate ASIL decomposition over redundant parts of the system. To complete the independence analysis of the redundant parts, we must also be able to identify Common Cause Faults in the system. Finally, a quantifiable metric is required to compare between different architecture solutions.

We use the model of [4] to describe the previous requirements and perform the functional safety analysis. The model consists of three layers, each described by a graph:

$G = (N, E)$	Application graph
$H = (R, L)$	Resource graph
$F = (P, C)$	Physical graph

Each graph contains a set of nodes and a set of edges. Each edge is an ordered pair (i, j), where i is the source node and j is the sink node.

The application graph is formed by the set of application nodes N and the set of channels E, which forms the logical relationships between the nodes. We assume that the FSR was obtained from a preceding *Hazard Analysis and Risk Assessment* (HARA). Each application node n has an ASIL value $A(n)$ that is defined by the FSR.

The resource graph contains the set of resources R and a set of links L that describe the hardware architecture of the system. Each resource r has an ASIL value $A(r)$ that describes the maximum safety level obtainable by that resource, also called *ASIL-X-ready resource*.

The physical graph is composed of the set of physical locations P and the physical connections C. In our Functional Safety Analysis, the physical locations contain information about the environmental factors required for the FFI analysis.

Automotive applications often have feedback loops to correct the inputs of the controllers, meaning that the application graph G will often be a *Directed Cyclic Graph* (DCG).

A. An Example Sensor-to-Actuator Application

Figure 3 shows an example of a simplified redundant system in which a data fusion system collects the data from a camera and a GPS to control the steering actuator.

In the application layer, the communication between the different modules is described by explicit *communication nodes*, such as the camera stream or the GPS coordinates. The function $MapG(N) \rightarrow \mathcal{P}(R)$ defines the mapping of the application nodes to the resources. For example, the GPS coordinates are sent via the CAN bus, the CAN-to-Ethernet

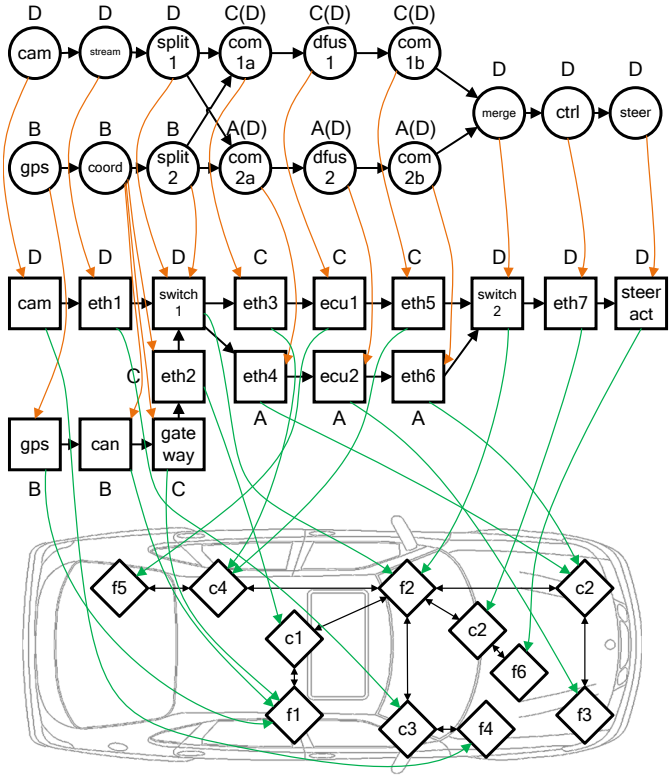

Fig. 3: Redundant system with camera and GPS data fusion.

gateway and the Ethernet connection *eth2*. The physical locations *c1*, *c2*, etc. represent the space in the vehicle in which the cables are placed. For example, *c4* connects the front of the car, where the switches are positioned, with the rear, where *ecu2* is located. The function $MapH(R) \rightarrow \mathcal{P}(P)$ defines the positioning of the resources into the physical space.

We use two special nodes in the application layer to identify the redundant blocks: the *splitter* and the *merger* nodes. A splitter replicates the input data to its output ports, while a merger compares the redundant inputs and ensures only correct output data is forwarded. The combination of these two types of nodes creates the redundant blocks, in which each branch that connects the two is a subsystem that works in parallel with the others. In the example in Figure 3, the splitter and merger nodes are implemented in the Ethernet switches.

Having an explicit redundancy pattern in the model allows us to identify the redundant parts of the system in order to apply and verify the ASIL decomposition rules. The independence of the separate redundant branches (e.g. in terms of used resources or locations) is necessary to validate the decomposition.

V. Fault Tree Generation and Analysis

The model of a generic system can be used to generate the system fault tree. During the construction of the Safety Case, the fault trees are generally manually constructed by a safety expert. We follow instead the algorithm described in [4], exploring the *application graph* from actuators to sensors, to automatically generate the fault tree. For each node, a pattern is generated. For example, the fault tree part related to the node *com a1* of Figure 3 is shown in Figure 4. The input

nodes are connected via an OR gate, and the only exception to this rule is the *merger* node, which will propagate the effect of its inputs via an *AND* instead of an *OR* gate by having the possibility to chose which of its input data shall be propagated.

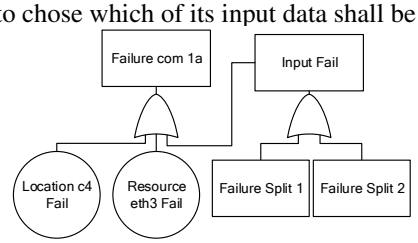

Fig. 4: Result of fault tree generation for the node *com a1* from Figure 3.

A fault tree is a non-cyclic graph, but the *application graph* can contain cycles. In the generation phase, when a cycle is detected the fault tree is cut, since the cyclic dependencies are not analyzed with the FTA.

In this work, we assign a failure rate λ to each base event of the generated fault tree. Table I shows the failure rates values that were used in the experiments for this work for the resource layer. The failure rates are related to the type of resource and to their ASIL specification. We assume *splitter* and *merger* resources to be highly reliable components and their failure rates are lower by a factor of 10 compared to other types of resources. The λ value related to the physical position is 10^{-11} failures per hour and it conveys information about the environmental conditions and the probabilities of accidents to impact a specific position of the vehicle.

TABLE I: Resources failure rates (failures/hour).

Resource Type	QM	A	B	C	D
Splitter or Merger	10e-6	10e-7	10e-8	10e-9	10e-10
Other	10e-5	10e-6	10e-7	10e-8	10e-9

To compute the total failure probability of the system, we translate the fault tree into a Binary Decision Diagram (BDD) using an *If-Then-Else* (ITE) structure as described in [17]. This step will reduce the complexity of calculating the failure probability of the system directly on the fault tree. A variable ordering is created by traversing the fault tree from top to bottom and from left to right, so that the base events that impact more directly the Top Level Event come first. Each base event is represented in the *ITE* structure as *ITE(b_event,1,0)*, where *b_event* is the variable. The following equations are followed to perform an operation $<op>$ between two *ITE* structures *J = ITE(X,f1,f2)* and *G = ITE(Y,g1,g2)* :

$$J<op>G = ITE(X, f1<op>G, f2<op>G) \ \ if\ X<Y \quad (1)$$

$$J<op>G = ITE(X, f1<op>g1, f2<op>g2)\ if\ X=Y \quad (2)$$

The possible operations depend on the gates present in the fault tree: addition when an *OR* gate is found, multiplication in case of an *AND* gate. The constructed *ITE* structure for the Top Level Events corresponds to the BDD that represents the system, where the variables correspond to the fault tree's base events and have an associated failure rate. We calculate the total system failure probability by following the BDD rules.

The conversion between fault tree and BDD computational cost grows exponentially with the number of redundant blocks that are present in the application graph. That is due to the recursive nature of the *ITE* operations of Equations 1 and 2 combined with the effect of calculating the *ITE* structure for each of the possible paths present in the graph, *whose number increases exponentially with each new ASIL decomposition*. To improve the scalability of the method for systems that have more than a few separate redundant blocks we introduce a new approximation technique that aims to reduce the number of possible different paths present in the application graph.

The approximation consists in removing the failure events related to the nodes that form the redundant branches from the fault tree, connecting directly the *splitter*'s failure event to the corresponding *merger*'s input failure event via an OR gate.

For the example of Figure 3, the failure probability of the system without the approximation is $2.04180e{-}7$ fph, compared to $2.04179e{-}7$ fph with the approximation. The approximation lowers the number of nodes in the fault tree from 87 to 51, but more importantly, halves the number of possible paths, for each of the *n* ASIL decomposition a 2^n factor. The fact that the failure events related to the branches do not have a significant impact on the failure probability of the system is due to the *merger* node: the AND gate on its inputs leads to a multiplication of the failure rates of these events. The order of magnitude of multiplied λ can be expected to be much lower compared to the rest, as illustrated in Table I. For more complex applications (695 nodes in the fault tree for position 1 of Figure 1) it is not possible to calculate the failure probability without the approximation.

The approximation also requires that the nodes in the separate branches do not share common *base events*, which would also invalidate the independence of the redundant parts as required by the ASIL decomposition rules of the ISO 26262 standard. The algorithms check these assumptions. An analysis of the nodes that form the branches highlights possible Common Cause Faults (CCFs) when the same *base event* is found on nodes belonging to different branches. For example, when the nodes $dfus_1$ and $dfus_2$ of Figure 3 are mapped on the same ECU, the base event related to the failure of the hardware component will be present for both the nodes and a warning for a potential CCF is issued. The warning informs the user that the ASIL decomposition is not valid, as well as the results obtained by using the approximation in the failure probability calculation.

VI. Cost Calculation

To identify the trade-offs between the failure probability and the cost of an architecture, we perform a cost calculation based on the ASIL values of the resources. We consider different cost metrics to highlight the flexibility of the tool in the exploration.

Table II shows one possible cost metric, where the cost of the splitter and merger resources is lower than other resource types, assuming that since they perform a very specific functionality, their cost could be lower because of a simpler safety analysis and certification process. As previously shown in Figure 1, different cost metrics can be used and can be tailored to a specific case.

TABLE II: Exponential Cost Metric 1

res Type	QM	A	B	C	D
Functional	5	50	500	5000	50000
Communication	4	40	400	4000	40000
Sensor	8	80	800	8000	80000
Actuator	8	80	800	8000	80000
Splitter	1	10	100	1000	10000
Merger	1	10	100	1000	10000

VII. ASIL-Oriented Model Transformations

A. Expand, Connect and Simplify transformations

Transformations can be applied to the model described in the previous section to modify the level of redundancy of the system. To adhere to the ISO 26262 standard the ASIL decomposition rules must be applied on the component level as explained in [4]. Equation 3 expresses the relationship between the resulting ASIL value, the FSR and its implementation in the resource layer.

$$ASIL(node) = min(A(node), A(MapG(node))) \quad (3)$$

Equation 4 expresses the relation to the ASIL value of a redundant block in relationship with the ASIL values of the *splitter*, *merger* and the redundant branches. These formulas are used to assign the correct ASIL values on the transformed modules obtaining the desired ASIL for the redundant system.

$$ASIL_{block} = min(ASIL(split), \sum_{b \in branches} ASIL_b, ASIL(merg)) \quad (4)$$

We perform three transformations on the graphs: *Expand()*, *Connect()*, *Reduce()*.

The function *Expand(app_node)* allows us to implement a redundant solution for the application node by substituting it with the structure shown in Figure 5, as presented in [4]. A splitter n_s is added for each input and a merger n_m is added for each output of the application node. In the example scenario, the node n is substituted by two redundant nodes, n_1 and n_2, and connected to the splitter and the merger via communication nodes. In case of expanding a communication node, the structure will be slightly different, containing only one communication node per branch and new communication nodes as input and output of the splitter and merger. The new ASIL value of the redundant block is calculated as in Equation 4. Depending on the ASIL values assigned to the resulting nodes we can obtain a higher or equal ASIL value for the block. Even if this transformation adds 7 extra nodes to the graph, the resulting cost could still be lower based on the cost metric that is used, since the splitter and the merger nodes can be executed by specific resources, while n_1 and n_2 are now mapped on less critical cheaper resources.

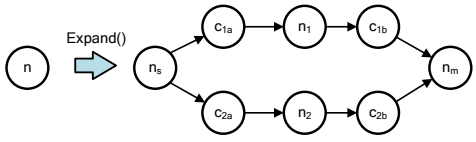

Fig. 5: Result of transformation *Expand(n)* applied on a functional node *n*.

The function *Connect(RBlock1, RBlock2)* takes two consecutive redundant blocks and merges them into a single one, as shown in Figure 6. In order for this transformation to be ASIL-equivalent, the following conditions must be satisfied:

1) The ASIL values of *RBlock1* and *RBlock2* must be the same;
2) The number of redundant branches in *RBlock1* and *RBlock2* must be the same;
3) The middle communication node *c* must not be connected to any external node different from n_m and f_s;
4) For each branch in *RBlock1* there must be a branch in *RBlock2* with the same ASIL value.

When the four conditions are satisfied, if the branches with the same ASIL are connected and n_m, *c*, and f_s are removed, then the resulting redundant block will have the same ASIL value as the combination of *RBlock1* and *RBlock2*.

In case of a single-fault scenario, the *Connect()* transformation does not change the reliability of the system. Note that in a two or more-faults scenario the transformed system is more prone to experience a system failure.

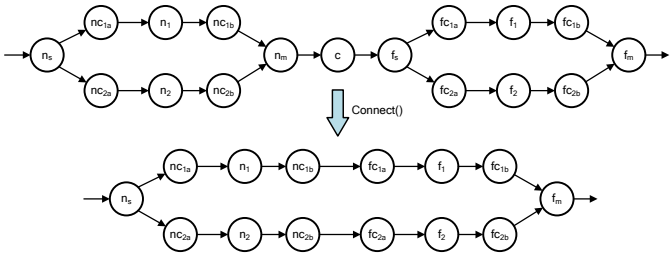

Fig. 6: Transformation *Connect(BlockN,BlockF)*

Last, the new *Reduce(app_comm1, app_comm2)* transformation simplifies two consecutive communication nodes. This situation could happen as a consequence of one of the previous transformations and the two consecutive communication nodes are not necessary for the analysis since they both contain the same information. The nodes *app_comm1* and *app_comm2* are substituted with a single communication node which is connected to the graph. If the ASIL values of *app_comm1* and *app_comm2* were different, the lowest ASIL value of the two is assigned to the resulting node.

The results of the *application* layer transformations are mapped to the resource graph and the ASIL decomposition is verified via the Common Cause Fault analysis described in Section V.

B. Effects of the model transformations and mapping on the system failure probability and cost

The failure probability and the cost of the system are strictly related to the *resource* and *physical layers*. The mapping of the application takes an important part in deciding which hardware modules are used in the final architecture. To evaluate the effects of the model transformations we first assume that for each new node in the *application graph* a new resource with the appropriate ASIL value is used in the *resource graph*.

The example of Figure 7 shows an expansion of a node n, which has 1 input and 2 outputs in the original graph. The initial failure probability of the system is $7.07e-9$, and after the transformation it becomes $6.39e-9$. The use of reliable splitter and merger resources reduces the failure

probability by replicating the functionality of node n. The cost of reliable resources with a specific role can be lower than reliable general purpose hardware. The shift of the high reliability requirements from the processing hardware to the safety management parts can thus reduce the system cost.

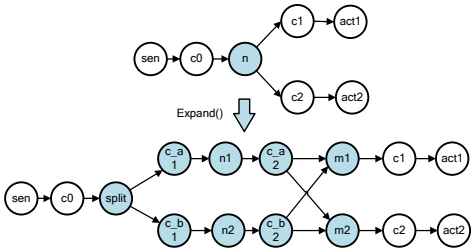

Fig. 7: Expansion of a node with 1 input and 2 outputs.

Figure 8 shows an example of the *Expansion()* transformation applied to a node with 3 inputs and 3 outputs. In this scenario, the resulting failure probability is $1.28e{-}8$ fph, compared to the initial system that has $1.21e-8$ fph. Even if the splitter and merger resources have individually lower λ than a functional resource with the same ASIL level, the introduction of 6 new resources in the system increases the complexity and each resource adds its individual failure rate to the probability calculation. This shows that it is not always beneficial to introduce redundancy in the system, depending on the λ values of the resources that are being used and the system configuration. In this scenario, a 1 input to 2 output node is worth to expand, while a 3 to 3 is not.

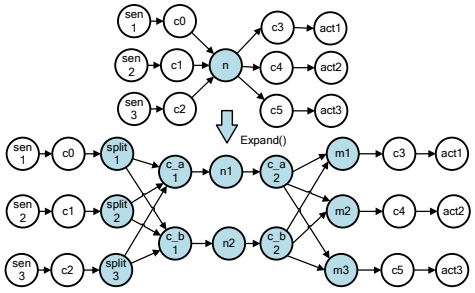

Fig. 8: Expansion of a node with 3 inputs and 3 outputs.

For Figure 6, the initial graph failure probability is $5.49e{-}9$ and after the *Connect()* transformation it becomes $4.26e{-}9$. By removing the hardware on which n_m, c, and f_s are mapped, the resulting system achieves a lower failure probability.

The *Reduce()* transformation is instead used mostly inside the redundant branches, and will not result in an impactful variation of the system failure probability, while reducing the cost by reducing the number of used resources.

The mapping has a strong impact on the outcome of the analysis. Figure 9 shows two different usages of the resources by the same application. In the first case, the system failure probability is $8.29e{-}9$ fph, while in the second one $4.26e{-}9$ fph. Assuming that the application nodes mapped on the same resource do not interfere with each other or increase the failure rate of the resource because of its sharing, the second solution will have a less complex hardware architecture that can achieve a higher grade of reliability. Advanced mapping algorithms can be used to identify the minimum set of necessary resources to achieve the minimum failure probability for the system, but we defer these techniques to future work.

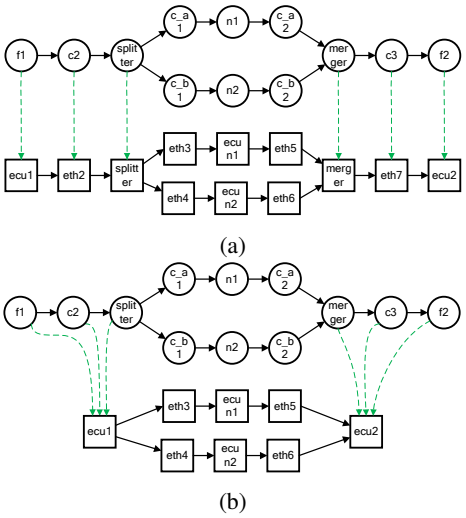

Fig. 9: Individual resource for each application node (a) and shared resources for multiple application nodes (b).

VIII. The EcoTwin Platooning Project

In this work we apply the analysis and transformations to the truck platooning project of the EcoTwin consortium [18]. In a truck platoon trucks autonomously follow a primary vehicle with a short distance, allowing significant fuel savings. The EcoTwin project focused on the development of a level 2+ [1] system architecture by using a Safety Executive Pattern, in which two channels, a main and a safety backup one, work in parallel and are monitored by a Health Management System.

The starting point consisted of a non-redundant system that did not satisfy the functional safety requirements because of the non-availability of ASIL D ready resources to run the self-driving algorithms. In our work we retrace the manual design process with our automated framework, which analyses the architecture providing cost and reliability information at each decomposition step. Our framework's input is a valid graph in terms of FSR, which models the ideal system where ASIL D resources are available. Our model transformations ensure the previously manual steps are automated and checked for cost and reliability at each step.

The application graph in Figure 10 describes the model of the original lateral control application. *Virtual* splitters are used when multiple sensors are acquiring information about the same objects and their data is used together in a sensor fusion algorithm. A single channel analyses sensors data to create a world model and provides the steering control signal for the steering actuator.

IX. Results

The experiments performed in this work transform the original *application* graph of the EcoTwin project shown in Figure 10 into the *application* graph shown in Figure 11. In the resulting graph two redundant branches are generating the actuator input signals, compared to the initial graph in which the autonomous driving functionality was executed by a single

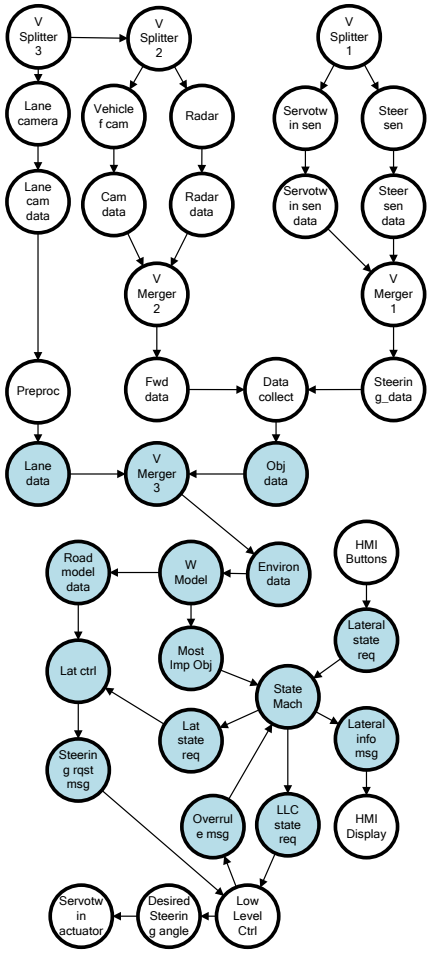

Fig. 10: Original non-redundant input application graph. In blue the nodes that are expanded in the experiments.

flow. Multiple metrics and types of transformations can be used, as shown in Figure 1, to obtain the redundant system: *BB, AC, RND* refer to the ASIL decomposition possibilities of Figure 2, while the numbers correspond to different cost metrics. To explain the results, we use the cost metric of Table II and an *Expand()* transformation that sets an ASIL value of the redundant branches according to the decompositions [2-1-1-1] of Figure 2, and assigns the original ASIL value to the splitters and mergers. Figure 12 shows the results of the analysis in terms of Failure Probability vs Costs.

The model transformations are:

1) *Expand()* the *functional* and *communication* nodes that form the decision part of the system, highlighted in Figure 10. All these transformation but one increase the cost of the system and its failure probability. That is because for each node introduced by the replication a new resource is added to the resource layer, creating complexity in the architecture. The initial cost value is 998800 with a failure probability of $6.37e-9$ fph, point A in Figure 12, reaching 1843000 and $2.14e-8$ fph at point B, where the maximum expansion is reached.
2) *Connect()* and *Reduce()* to connect the redundant blocks. The curve is linear, since all the nodes that are removed were ASIL D, connecting branches with ASIL B values. The achieved cost is 1229000, with a failure probability

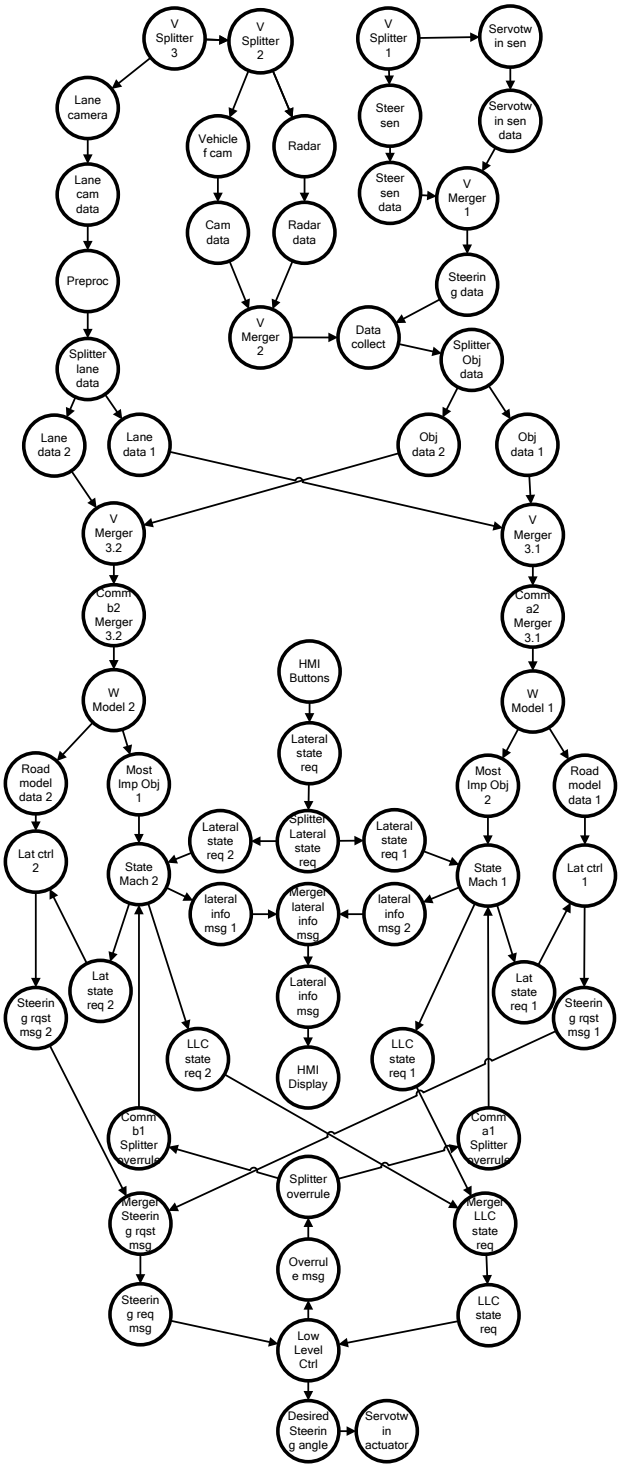

Fig. 11: Redundant output application graph

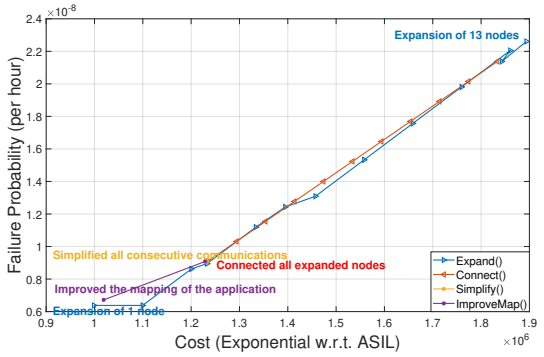

Fig. 12: Analysis of different level of redundancy for a lateral control application in a truck platooning system.

of $9.07e{-}9$ fph at point C of the curve. The cost of the current system as well as its failure probability are higher than the original system, but the mapping has not been optimized yet. Moreover, the redundant system utilizes ASIL D components in the modified part of the application graph only for the *splitter* and the *merger* nodes and only ASIL B for the computational resources. In a real project it is likely that no ASIL D components are available (at a low cost) thus requiring these transformations.

3) Finally, after applying both the *Connect* and the *Reduce* transformations, the mapping modification described in Section VII-B is performed where possible, obtaining a system with a cost of 1019000 units and a failure probability of $6.72e{-}9$ fph at point D. The final results uses ASIL D components only for the *splitter* and *merger* nodes, which are specific functions related to the redundancy management, while functional nodes are implemented by ASIL B components, which are generally available for general purpose application in automotive systems. This solution achieves a failure probability and a cost similar to the initial ideal (but infeasible) architecture, where only ASIL D components were used. More mapping optimization techniques can be used to further reduce the costs and the failure probability of the system.

X. Conclusions

In this paper we presented a framework for safety analysis of automotive systems. The ASIL decomposition technique of the ISO 26262 standard is followed to modify the redundancy of the system on the implementation-level. The automotive system is described with a 3-layer model, on which transformations can be applied to modify the configuration and maintaining the required ASIL value. Fault trees are generated from the model and are used to perform Common Cause Fault analysis for validation of the decomposition and a probabilistic analysis is performed to obtain the system failure probability. A cost is calculated from the model to analyze the trade-offs between safety and cost of the system with different configurations. This framework can take part of the building of the Safety Case for an automotive system, providing meaningful information to the system architects, as well as traceability of the FSRs on the architecture, scalability, automatic fault trees generation, quantitative metrics and validation of the ISO 26262 standard safety techniques.

The methodology is proven on the lateral control application of a real truck platooning system, obtaining a complete analysis of the system to guide the system implementation.

Acknowledgment

The work in this paper is supported by TU/e Impuls program, a strategic cooperation between NXP Semiconductors and Eindhoven University of Technology. The authors thank all reviewers for their helpful comments and suggestions that helped improve and clarify this paper.

References

[1] SAE, "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems", *SAE Standard J3016*, 2014.

[2] S. Sommer *et al.* "RACE: A centralized platform computer based architecture for automotive applications", in *IEEE Int. Electric Vehicle Conference*. IEEE, 2013.

[3] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car - part II: a case study on the implementation of an autonomous driving system based on distributed architecture", in *IEEE Trans. on Industrial Electronics*, vol. 62, no. 8, 2015.

[4] A. Frigerio, B. Vermeulen, and K. Goossens, "A generic method for a bottom-up asil decomposition", in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018.

[5] ISO, "ISO 26262-2018: Road vehicles - functional safety - part 9: ASIL-oriented and safety-oriented analyses", 2011.

[6] J. G. D'Ambrosio and R. Debouk, "ASIL decomposition: the good, the bad, and the ugly", SAE Technical Paper, Tech. Rep., 2013.

[7] Y. Papadopoulos *et al.*, "Automatic allocation of safety integrity levels", in *Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety*. ACM, 2010.

[8] M. S. Dhouibi, J. M. Perquis, L. Saintis, M. Barreau, "Automatic Decomposition and Allocation of Safety Integrity Level Using System of Linear Equations" in *Proceedings of the Fourth International Conference on Performance, Safety and Robustness in Complex Systems and Applications*. Springer, 2014 .

[9] L. da Silva Azevedo, D. Parker, M. Walker, Y. Papadopoulos, and R. E. Araujo, "Assisted assignment of automotive safety requirements", in *IEEE software*, vol. 31, no. 1, 2014.

[10] P. Bieber, R. Delmas, C. Seguin. "DALculus theory and tool for development assurance level allocation" in *Proc. of the 30th Int. Conference on Computer Safety, Reliability and Security*. Springer, 2011.

[11] K. Delmas, R. Delmas, and C. Pagetti, "Smt-based synthesis of fault-tolerant architectures", in *Int. Conference on Computer Safety, Reliability, and Security*. Springer, 2017.

[12] F. Maticu, P. Pop, C. Axbrink, and M. Islam, "Automatic functionality assignment to AUTOSAR multicore distributed architectures", SAE Technical Paper, Tech. Rep., 2016.

[13] M. L. McKelvin, Jr., G. Eirea, C. Pinello, S. Kanajan, and A. L. Sangiovanni-Vincentelli, "A formal approach to fault tree synthesis for the analysis of distributed fault tolerant systems", in *Proceedings of the 5th ACM Int. Conference on Embedded Software*. ACM, 2005.

[14] M. Ghadhab, S. Junges, J.-P. Katoen, M. Kuntz, and M. Volk, "Model-based safety analysis for vehicle guidance systems", in *Computer Safety, Reliability, and Security*. Springer, 2017.

[15] D. D. Ward and S. E. Crozier, "The uses and abuses of ASIL decomposition in ISO 26262", in *7th IET International Conference on System Safety, incorporating the Cyber Security Conference*. IET, 2012.

[16] IEC, "IEC 61508 Edition 2.0. Principles and Use in the Management of Safety", 2010.

[17] J. Andrews and R. Remenyte, "Fault tree conversion to binary decision diagrams", 2005.

[18] T. Bijlsma and T. Hendriks, "A fail-operational truck platooning architecture", in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017.