Model-Based Software Design, A.Y. 2023/24

Laboratory 3 Report

Components of the working group (max 2 people)

- Matteo Gravagnone, s319634
- Danilo Guglielmi, s318083

Functional Safety Concept

One pedal

Functional safety architecture

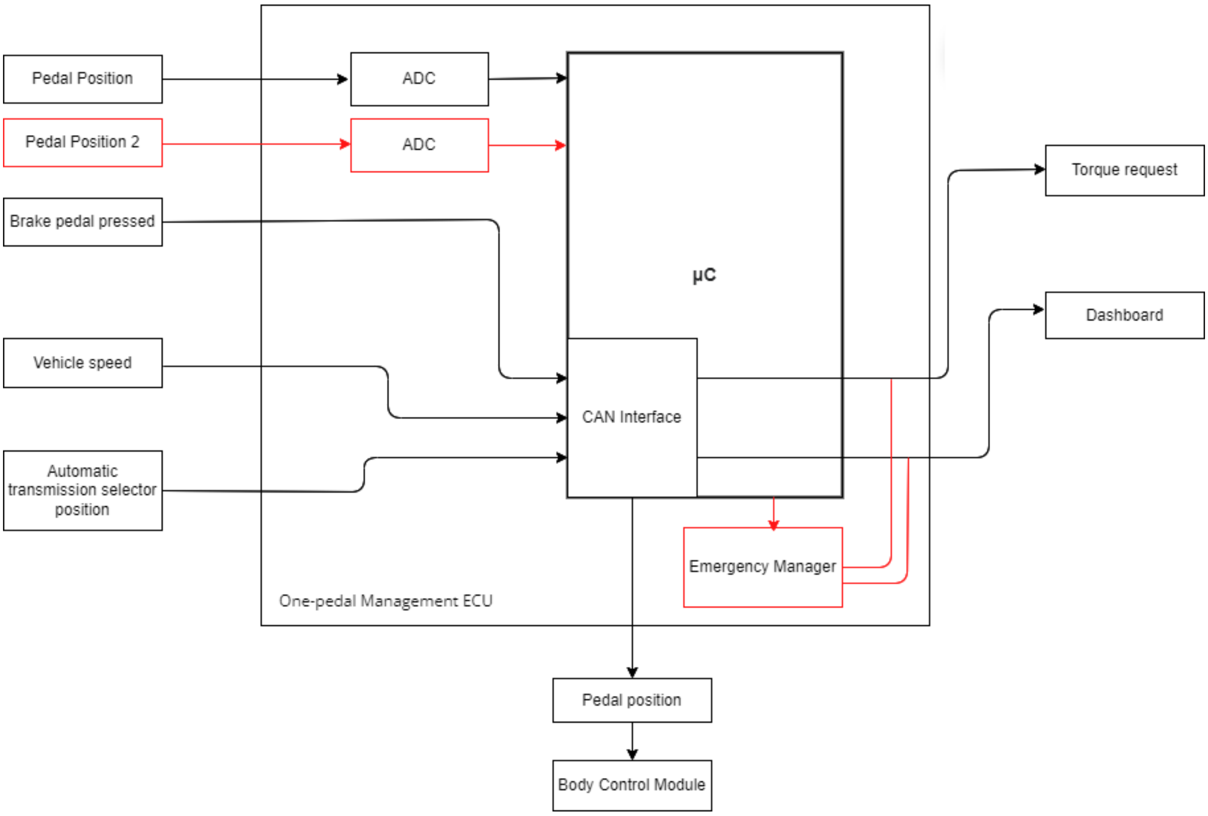


Figure 1 Functional safety architecture (from the safety concept)

Attributes of the safety goals

Fill in the attribute/parameters of the safety goal

Safety goal	**Attributes/Parameters of the safety goal**				
	Integrity (ASIL)	Safe state	Fault tolerance time	Warning concept	Degradation concept
SG1	C	Switch to N	100 ms	The driver must be notified on the dashboard	Motor is turned off
SG2	B	Switch to N	100 ms	The driver must be notified on the dashboard	Motor is turned off
SG3	B	Warning of the malfunction	100 ms	The driver must be notified on the dashboard	Warning system is deactivated

Functional (and technical) safety requirements and allocation

		Define functional safety requirements		Allocation of requirements on systems and elements	
		Safety requirements	**Remark**	**If applicable, allocate the safety requirements to other Items / Systems**	**If applicable, allocate the safety requirements to equipment other technologies to minimize risk. That could be e.g. hydraulic, mechanical equipment**
Safety goals	**The vehicle must not accelerate unintentionally**	SR1: If the pedal position interpreted is not valid (between 0 and 1), the torque request is set to 0.	No	No	Hydraulic braking system
		SR2: The torque should be limited in the correct interval depending on the current state: if B between [-80, 80], if D between [0; 80], if R between [-40, 0], 0 if N or P.	No	No	Hydraulic braking system
	The vehicle must not decelerate unintentionally	SR1: If the pedal position interpreted is not valid (between 0 and 1), the torque request is set to 0.	No	No	No
		SR2: The torque should be limited in the correct interval depending on the current state: if B between [-80, 80], if D between [0; 80], if R between [-40, 0], 0 if N or P.	No	No	No
	The vehicle should be able to detect malfunctions in the warning system	SR1: Monitor the functionality of the warning system periodically with specific diagnostic routines.	No	No	No
		SR2: Notify the driver with a specific error message and activate a dashboard light if a warning malfunction is detected.	No	Warning lamp in the Cockpit-Display	No

ASIL preliminary architecture[1]

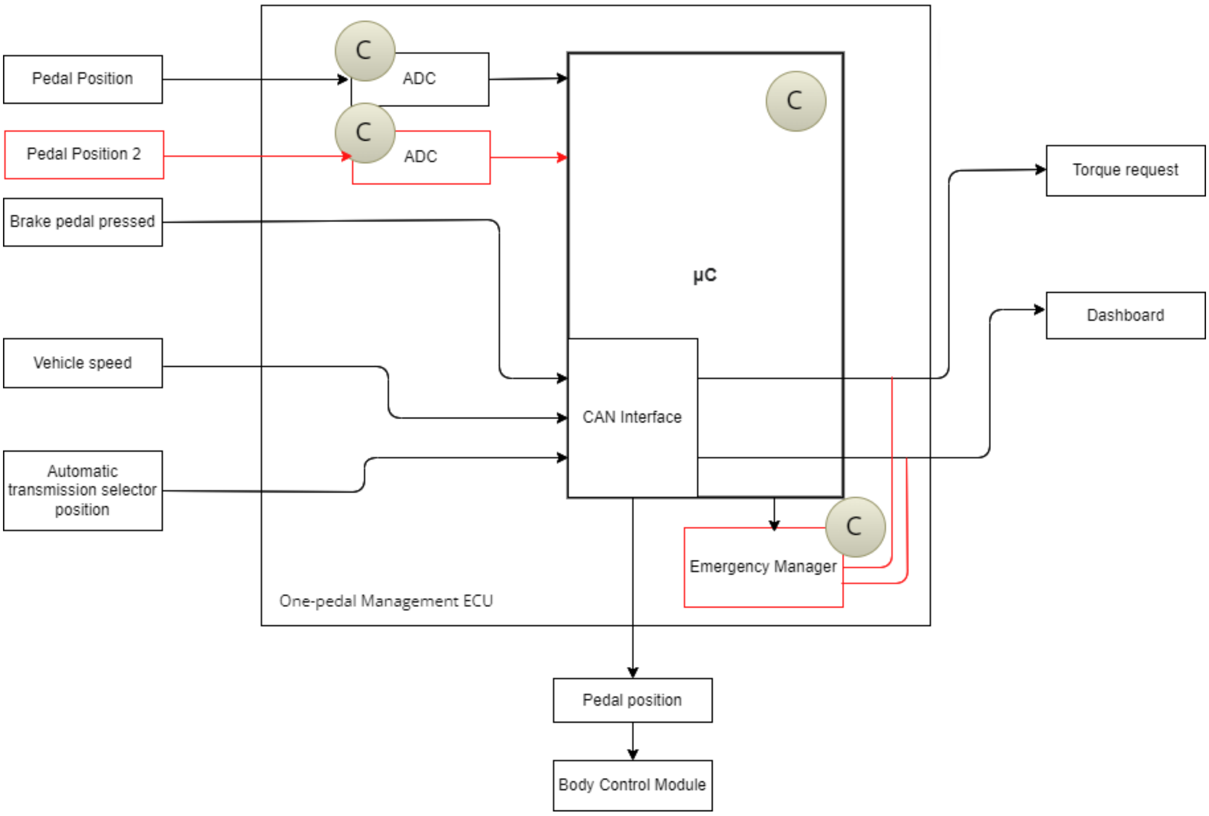


Figure 2 Preliminary architecture without ASIL decomposition

[1] See document `02-iso26262.pdf`, slides 89, 90, 91, 92, 93.

Implementations[2]

Functional redundancies

The system can have at least 2 circuitries that can read the pedal position at the same time. The µC can be replaced, in case of failure, by a simpler circuit, called Emergency Manager, that warns the driver of the failure and sets the torque request to zero. In our implementation, the µC is responsible for warning and actions, and its failure is not managed.

Implemented plausibility checks

- *BrakePedalPressed* is a boolean value so plausibility checks are not needed.
- *ThrottlePedalPosition* needs to be in the range [0; 1] so if the position is not valid, a warning flag is set. A secondary *ThrottlePedalPositionRedundancy* input is also checked in the same range, a warning flag is set. If the discrepancy between the two variables is larger than 0.1 (10% of the pedal travel), a warning flag is set. An overall *PedalWarningFlag* is set to true if at least one of the previous three flags is set.
- *AutomaticTransmissionSelectorState* is seen as an integer value between 0 and 4. If it is not in this range, a warning flag called *SelectorWarningFlag* is set.
- *TorqueRequest_Nm* is limited in the correct interval depending on the current AutomaticTransmissionState value: if B between [-80, 80], if D between [0; 80], if R between [-40, 0], 0 if N or P.
- If at least one between *PedalWarningFlag* and *SelectorWarningFlag* is set, the controller sets *Warning* to true, switches to N and sets the torque request to zero.

---

[2] In the ISO26262 the implementations are based on a document called *Technical Safety Concept*, but for simplicity we move straight from the *Functional Safety Concept* to software implementations.
A guideline for the implementation phase can be found in the document `02-iso26262.pdf` from slide 81, in particular slide 86.

Software testing

Implemented unit tests

Describe in English the test performed to verify the correct functionality of the safety mechanism implemented.

In our model 4 different units, related to safety mechanisms, were tested.
For each of them, a test harness, with its own *.slx* file, was created through Simulink Test and modified with a set of proper test inputs for our coverage tests.
Decision was chosen as *Structural coverage level* in the coverage metrics settings of the harness in order to obtain values for branch and statement coverage.

- *controller_Harness_PedalPosition*: this harness is used to check the behavior of the mechanism related to the throttle pedal position, which implements the check about the range of both input pedal positions and the difference, in magnitude, between them, which shall not be larger than 0.1 in normal behavior.

  Summary

  Model Hierarchy/Complexity

		Decision	Execution
1. ThrottlePedalSafety	3	100%	100%
2. . . . Compare To Constant		NA	100%
3. . . . Compare To Constant1		NA	100%
4. . . . Compare To Constant2		NA	100%
5. . . . Compare To Constant3		NA	100%
6. . . . Compare To Constant4		NA	100%

- *controller_Harness_Selector*: this harness tests the mechanisms that checks the values of the AutomaticTransmissionStateSelector. The part of the harness related to the input has been modified in order to “force” invalid values of the input, otherwise Simulink would not allow them when requiring a cast to the enum TransmissionState (The subsystem which implements the safety mechanism therefore has not been modified in any way).
  In addition, the logic has been made more elaborated in order to include a decision block (Switch).

  Summary

  Model Hierarchy/Complexity

		Decision	Execution
1. Subsystem	2	100%	100%
2. . . . Compare To Constant2		NA	100%
3. . . . Compare To Constant3		NA	100%

- *controller_Harness_Torque*: this harness tests the mechanism that saturates the torque to the limit values according to the current transmission state. Therefore, it switches to different states and uses both correct and incorrect values of computed torque.

  Summary

  Model Hierarchy/Complexity Run 4

		Decision	Execution
1. Subsystem1	15	100%	100%

- *controller_Harness_Warning*: it is used to evaluate the behavior of the mechanism that, based on the flag which would be computed from the first two mechanisms, decides whether to switch to N and force zero torque or, otherwise, keeps the computed current transmission state and torque request.

Summary

Model Hierarchy/Complexity		Run 5 Decision	Execution
1. Subsystem2	3	100%	100%

Report generated through Simulink have been included in the provided files.

Implemented integration tests

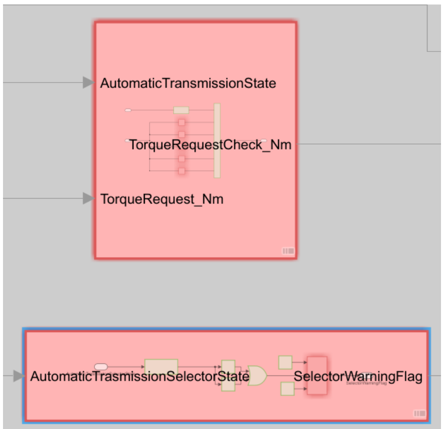
Describe, in English, the scenarios tested at the integration level to verify the proper integration between the various units implementing the safety mechanisms.

controller_Harness: this harness includes the overall controller and, in addition, also the connection to the plant which computes realistic values of the vehicle speed. A variety of inputs has been chosen to cover the considered problems, as the safety mechanisms shall work together.

Summary

Model Hierarchy/Complexity		Decision	Execution
1. controller	53	85%	100%
2. . . . Chart	33	100%	NA
3. . . . . . . SF: Chart	32	100%	NA
4. . . . . SF: BRAKE	11	100%	NA
5. . . . Subsystem	1	50%	100%
6. . . . . . . Compare To Constant2		NA	100%
7. . . . . . . Compare To Constant3		NA	100%
8. . . . Subsystem1	14	60%	100%
9. . . . Subsystem2	2	100%	100%
10. . . . ThrottlePedalSafety	2	100%	100%
11. . . . . . . Compare To Constant		NA	100%
12. . . . . . . Compare To Constant1		NA	100%
13. . . . . . . Compare To Constant2		NA	100%
14. . . . . . . Compare To Constant3		NA	100%
15. . . . . . . Compare To Constant4		NA	100%

The following image helps us tell why the decision does not reach 100% everywhere.
The StateFlow Chart, responsible for the computation of the TransmissionState and TorqueRequest, always produces values in the valid range and, as a consequence, the saturation blocks do not actually intervene.



Regarding the *AutomaticTransmissionSelectorState*, the model requires Enum: TransmissionState as a data type, therefore values not allowed in the enumeration cannot be selected.
To reach 100% decision everywhere, we would have to modify the model to have a “relaxed” input for the selector and to force incorrect values of torque request.