ALEBAS : une méthodologie de développement et d'analyse de sûreté de fonctionnement des systèmes embarqués

Jean Godot

NNT : 2018SACLS057

Thèse de doctorat de l'Université Paris-Saclay préparée à L'Université Paris Sud

Ecole doctorale n°580
Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

par

M. Jean Godot

ALEBAS: une méthodologie de développement et d'analyse de sûreté de fonctionnement des systèmes embarqués

Thèse présentée et soutenue à Saint-Quentin-en-Yvelines, le 27 mars 2018.

Composition du Jury :

Dr Samir Bouaziz	Professeur	Satie	Président du jury
Dr Mihaela Barreau	Enseignant/Chercheur	Université d'Angers	Rapporteur
Dr Jean-Yves Choley	Enseignant/Chercheur	Supmeca	Rapporteur
Dr Annie Bracquemond	Chef de projet	Vedecom	Examinatrice
Dr Cherif Larouci	Directeur du pôle S2ET	Estaca	Directeur de thèse
Dr Bertrand Barbedette	Enseignant/Chercheur	Estaca	Co-encadrant
Dr Sébastien Saudrais	Enseignant/Chercheur	Estaca	Co-encadrant
Dr Adil Alif	Docteur/Ingénieur R&D	Faar Industry	Co-encadrant

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : ALEBAS: une méthodologie de développement et d'analyse de sûreté de fonctionnement des systèmes embarqués

Mots clefs : Méthodologie - Sûreté de fonctionnement - Systèmes embarqués - Logiciel - Prototype - Industrie de la mobilité

Résumé : De nos jours, l'augmentation de la complexité des systèmes embarqués impose la prise en charge, dès les premiers prototypes et au niveau logiciel, d'exigences habituellement traitées plus tardivement voire même seulement lors du passage en série, comme notamment les contraintes de sûreté de fonctionnement. Nous proposons une méthodologie de développement de logiciels embarqués pouvant répondre aux besoins et contraintes de développement de ces logiciels dans la phase de prototypage. La flexibilité de la méthodologie garantit une meilleure gestion des évolutions récurrentes caractérisant les logiciels dans cette phase de prototypage. Aussi, grâce à sa structure bien adaptée, l'aspect automatisé de son intégration et de sa mise en place, notre méthodologie garantit une réduction significative des coûts du passage en série et de la mise en place de normes telles que l'ISO 26262 ou la DO-178. Pour illustrer les résultats que nous obtenons grâce à notre méthodologie et la chaîne d'outils associée, nous l'appliquons dans le cadre d'un projet industriel d'innovation qui consiste à la robotisation d'un véhicule prototype. Le cas d'étude se concentre sur la fonction « accélérateur » de ce prototype.

Title : ALEBAS: a development and safety analysis methodology of embedded systems

Keywords : Methodology - Safety - Embedded systems - Software - Prototyping - Mobility industry

Abstract : Nowadays, the increasing complexity of embedded systems requires the management, from the first prototypes and software level, of requirements usually handled later even only when passing in production, such as including dependability constraints. We propose a development methodology for embedded software that can meet the needs and constraints of developing this software in the prototyping phase. The flexibility of the methodology ensures better management of recurring changes characterizing the software in the prototyping phase. Also, due to its well-adapted structure, and automated aspect of its integration and implementation, our methodology ensures a significant cost reduction of serialization and implementation of norms such as ISO 26262 or DO-178. To illustrate the results, we apply our methodology and the associated toolchain to an innovative industrial project which consists to robotize a prototype car. The case study focuses on the accelerator function of the prototype.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Remerciements

Je remercie chaleureusement toutes les personnes qui m'ont aidé pendant l'élaboration de ma thèse, à commencer par mon directeur M. Cherif LAROUCI, pour son intérêt, son soutien et son suivi afin d'assurer le bon déroulement de la thèse.

Je tiens particulièrement à remercier mes encadrants M. Sébastien SAUDRAIS, M. Bertrand BARBEDETTE et M. Adil ALIF pour m'avoir accompagné tout au long de ces trois années. Leurs remarques constructives et leurs encouragements m'ont permis d'avancer tout au long de la thèse. Leurs relectures ont été capitales lors des périodes de rédaction des publications scientifiques et de ce manuscrit.

J'adresse mes remerciements à M. Khalid AYOUCHE et M. Emmanuel D'ARFEUILLE, respectivement directeur général et directeur technique de FAAR Industry, d'avoir accepté de monter ce projet de thèse CIFRE qui me tenait à cœur et pour les moyens fournis, afin de garantir la bonne conduite de ce projet.

Je tiens ensuite à remercier les membres du jury qui me font l'honneur de participer à la soutenance de cette thèse, notamment M. Samir BOUAZIZ pour avoir accepté de présider ce jury. Mes remerciements les plus sincères vont à Mme Mihaela BARREAU et M. Jean-Yves CHOLEY pour avoir accepté de rapporter ma thèse. Je remercie grandement Mme Annie BRACQUEMOND, avec qui j'ai eu l'opportunité d'échanger à plusieurs reprises sur mes travaux et qui m'a ainsi fait bénéficier de ses conseils.

Je remercie mes collègues de chez FAAR Industry et Pronergy pour leur soutien, leur accompagnement au cours de ces années et les conseils techniques.

Je tiens aussi à remercier les enseignants/chercheurs de l'ESTACA qui m'ont fait profiter de leurs conseils techniques ainsi que de leur partage d'expérience concernant le déroulement de la thèse. Je remercie les doctorants de l'ESTACA pour nos encouragements communs, nos discussions, dans les moments de doutes, au bout desquelles on finit toujours par relativiser, ainsi que pour l'ambiance conviviale.

Je ne peux évidemment pas oublier de remercier ma fiancée pour le soutien mutuel que nous nous sommes apportés au quotidien et qui nous a permis de poursuivre nos études. Enfin, mes remerciements vont à mes parents sur qui je peux toujours compter, ma famille et mes amis que j'ai quelque peu délaissés ces derniers mois pour achever cette thèse.

Sommaire

Liste des figures

Liste des tableaux

Listings

Glossaire

Sûreté de fonctionnement La propriété qui permet aux utilisateurs du système de placer une confiance justifiée dans le service qu'il leur délivre [Lap95]. XXI

Vote Plusieurs versions d'une même donnée sont calculées (ou reçues) "indépendamment". Une erreur est détectée lorsqu'une des versions au moins est distincte des autres. XXI

Acronymes

A Alerte. i–v, vii–ix, xi, xii, 40, 43, 45, 48, XXI

AADL Architecture Analysis and Design Language. 20, 21, XXI

ADAS Système avancé d'aide à la conduite –ou *Advanced Driver Assistance System*–. 2, 10, XXI

ADC Convertisseur Analogique/Numérique –ou *Analog/Digital Converter*–. i, 34, XXI

AdD Arbre de Défaillances –ou *Fault Tree*–. 5, 6, 15, 16, 18–20, 31, 35, 37, 50, 52, 54, 60, 63, 64, 66, 67, 69, 71, 72, 74, 76, 78, 79, 82, 84, 96, XXI

AEEL Analyse des Effets des Erreurs de Logiciel. 5, 18, XXI

AMDE Analyse des Modes de Défaillance et de leurs Effets. 5, 6, 15–20, 31, 35, 37, 39, 40, 54–58, 60, 61, 63, 64, 67, 69, 71, 72, 74, 79, 80, 82, 84, 96, XXI

AMDEC Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité. 29, XXI

APR Analyse Préliminaire des Risques. 15, 29, XXI

ASIL Automotive Safety Integrity Level. 29, 55, XXI

CAN Control Area Network. 33–35, 41, 46–48, 50, 52, 57, XI, XXI

D Détection. i–v, vii, ix, xi, 40, 43, 45, 48, XXI

DSL Domain Specific Language. XXI

E/E Électrique/Électronique. 22, 23, XXI

EAST-ADL Electronic Architecture and Software Tools - Architecture Description Language. XXI

ECU calculateur –ou *Electronic Control Unit*–. 34, 51, 54, 57, 75, 80, XXI

ER Événement Redouté. 29–31, 50, 51, XXI

ERlog Événement Redouté du logiciel. 31, 32, XXI

ERsys Événement Redouté du système. 29, XXI

FPTC Failure Propagation Transformation Calculus. XXI

FPTN Failure Propagation Transformation Notation. 20, XXI

Hazop Hazard and operability studies. 15, XXI

HiP-HOPS Hierarchically Performed Hazard Origin and Propagation Studies. 20, 21, XXI

HTML HyperText Markup Language. 67, 80, XXI

HW matériel –ou *Hardware*–. 57, XXI

I/O Entrées/Sorties –ou *Inputs/Outputs*–. 14, 35, 37, XXI

IDM Ingénierie Dirigée par les Modèles. 3, XXI

IF-FMEA Interface Focused - FMEA. 20, XXI

IHM Interface Homme-Machine. 30, XXI

MAAB MathWorks Automotive Advisory Board. ix, 32, 46, XXI

MBD Model-Based-Design. 3, 13, 14, 19, 27, XXI

MBSA Model-Based Safety Analysis. 19, 22, 24, XXI

MD Mode Dégradé. i–v, vii–ix, xi, xii, 40, 43, 45, 48, XXI

NCS Niveau de Continuité de Service. XXI

OASIS AutOmotive Analysis and Safety EngIneering InStrument. XXI

OEM Original Equipment Manufacturer. 22, 23, XXI

Open-PSA Open-Probabilistic Safety Assessment. 76, XXI

OPIIEC Observatoire des métiers du numérique, de l'ingénierie, des études et du conseil. 1, XXI

PC Performances Complètes. i–viii, xi, xii, 40, 43, 45, 48, XXI

PME Petites et Moyennes Entreprises. 2, 4, 27, XXI

PoC Preuve de Concept –ou *Proof of Concept*–. 5, 27, 63, XXI

PWM Pulse Width Modulation. 89, XXI

RD Règle de Développement. vii–ix, 40, 44, 46, XXI

SAML Safety Analysis Modeling Language. XXI

SdF Sûreté de Fonctionnement. xxii, 1, 4–6, 10, 14–24, 27, 28, 31, 32, 35, 37, 40, 50, 54, 55, 63, 66, 67, 69–71, 74, 76, 79, 84, 89, 90, 95–97, XXI

SW Logiciel –ou *Software*–. 57, 80, XXI

SysML Systems Modeling Language. 21, XXI

TRL Niveau de Maturité Technologique –ou *Technology Readiness Level*–. 2, XXI

UML Langage de Modélisation Unifié –ou *Unified Modeling Language*–. 21, XXI

V Vérification. i–ix, xi, 40, 43, 45, 48, XXI

XML eXtensible Markup Language. 67, 70, 76, 78, XXI

Introduction générale

Contexte

Les systèmes embarqués sont d'une importance stratégique dans l'économie moderne. Ils forment le premier secteur de croissance des technologies de l'information et de la communication. Ils jouent un rôle clé pour la compétitivité de nombreuses industries comme la production d'énergie, la productique, les télécommunications ou encore les transports : aérien, routier, ferroviaire et naval. Ces tendances sont mises en évidence par une étude de l'Observatoire des métiers du numérique, de l'ingénierie, des études et du conseil (OPIIEC) menée en 2014, sur le thème de « l'évolution des métiers et des besoins en formation pour les systèmes embarqués » [OPI14]. Cette étude fait suite à une première enquête conduite six ans plus tôt [OPI08], ainsi elle montre l'augmentation des besoins dans ce secteur d'activité au cours des années passées et dresse des prévisions pour les années futures.

Les systèmes embarqués combinent du matériel électrique/électronique et des fonctionnalités logicielles, qui sont conçus ensemble afin d'exécuter des tâches précises. Ils requièrent une attention particulière lors de leur développement du fait des limites et contraintes auxquelles ils sont confrontés. Ces limitations concernent notamment l'énergie disponible, la puissance de calcul, le volume d'occupation et le coût. Malgré ces restrictions, les systèmes embarqués doivent souvent, en plus des fonctions pour lesquelles ils sont conçus, être en mesure de gérer eux-mêmes leur consommation d'énergie, d'opérer leurs tâches dans un temps imparti (temps réel) et de plus en plus de garantir leur Sûreté de Fonctionnement (SdF).

Dans son livre [Rib12], le Pr. W. Ribbens explique que l'utilisation de l'électronique moderne, dans l'industrie automobile, prend son essor dans les années 70 pour deux raisons : (1) l'introduction de réglementation gouvernementale concernant les émissions des gaz d'échappement et l'économie de carburant, qui ont requis une meilleure gestion des moteurs à combustion que ce que pouvait proposer les techniques de cette époque et (2) la diminution relative des coûts d'utilisation de l'électronique. Depuis, la quantité de systèmes embarqués utilisés dans les véhicules ne cesse d'augmenter. Aujourd'hui, l'électronique compte pour 34% du prix du véhicule [OPI14] qui accueille en moyenne une

cinquantaine de calculateurs et jusqu'à une centaine pour les modèles haut de gamme. Cette progression est en grande partie permise grâce à la miniaturisation des composants électroniques. Au-delà de la réduction de la taille, la miniaturisation s'accompagne de la diminution de la consommation électrique par unité de calcul et du prix, alors que la vitesse de calcul augmente [Gou09]. Par conséquent, le potentiel des systèmes embarqués continue de s'accroitre considérablement et les mène à réaliser des fonctions toujours plus complexes et diversifiées au sein des véhicules. On les retrouve, aujourd'hui, dans toutes les fonctions typiques de l'automobile : le contrôle moteur, le freinage, la direction, la liaison au sol ou encore la sécurité et le confort. Dans ces secteurs, les systèmes embarqués remplacent ou supportent les fonctions mécaniques ; on les appelle alors les systèmes mécatroniques [Ber07]. On constate aussi de nouvelles tendances dans l'utilisation des systèmes embarqués dans l'automobile comme l'accroissement de l'assistance du conducteur dans la conduite (Système avancé d'aide à la conduite –ou *Advanced Driver Assistance System*– (ADAS)), qui aboutira à terme à son remplacement avec le véhicule 100% autonome et l'ouverture du véhicule à son environnement (communication avec les autres véhicules et les infrastructures) [MW13].

Ces nouvelles tendances dynamisent le monde automobile et se traduisent par une forte activité d'innovation. L'innovation est définie dans [BRS09] comme un processus à plusieurs étapes par lequel les organisations transforment des idées en produits, services ou processus nouveaux/améliorés pour s'avancer, rivaliser et se différencier avec succès dans leur marché. La progression d'une innovation est décrite grâce aux Niveaux de Maturité Technologique –ou *Technology Readiness Level (TRL)*–, pour évaluer son degré de maturité (échelle de 1 à 9) avant de l'incorporer à un système ou à un sous-système [Man95, ISO13]. Le marché automobile étant fortement concurrentiel (compétitivité), il est important pour les industriels du secteur de se démarquer de leurs concurrents en proposant des nouveautés qui améliorent l'expérience utilisateur et s'adaptent aux nouvelles réalités (véhicule autonome et connecté). Ainsi, l'automobile s'ouvre à d'autres secteurs d'activité comme l'intelligence artificielle, la perception (radar, lidar, caméra...), la localisation ou encore les télécommunications. Dans ce contexte, ce sont souvent des start-up ou des Petites et Moyennes Entreprises (PME) à fort potentiel qui intègrent ces nouveaux concepts. Pour ces entreprises, un de leur objectif premier est de susciter l'intérêt des grands équipementiers et constructeurs pour se faire connaître et ainsi travailler avec eux. Dans un second temps, il faut aussi convaincre les futurs utilisateurs de l'efficacité et de l'intérêt de leurs innovations car la plupart de ces nouveautés vont avoir un fort impact sur l'expérience utilisateur. Dans les deux approches, vis-à-vis des constructeurs ou des utilisateurs, cela passe très souvent par le développement de démonstrateurs afin de mettre en conditions réelles les intéressés. En effet, une technologie avec un niveau TRL relativement avancé, c'est-à-dire supérieure à 5, est susceptible d'intéresser un

industriel car à partir de ce stade, la technologie ressemble grandement à ce qu'elle sera une fois mise sur le marché. Il est donc plus facile pour les industriels et les futurs utilisateurs de se projeter et d'observer les qualités de ce qui leur est proposé.

Ce besoin de compétitivité a pour effet de réduire le temps de développement des démonstrateurs car chaque industriel souhaite être le premier à proposer une innovation pour se distinguer des autres. Ce désir d'écourter le développement a incité les acteurs a employé des approches comme le prototypage rapide. Dans le contexte des systèmes embarqués, le prototypage rapide consiste à mettre en place des moyens qui permettent de passer rapidement des simulations sur ordinateur à des tests sur cible embarquée dans un environnement simulé ou même directement dans l'environnement réel du véhicule. Une autre approche prend de plus en plus d'ampleur, c'est la modélisation. Elle est décrite par le Dr. J-M Jézéquel comme l'utilisation efficace d'une représentation simplifiée d'un aspect de la réalité pour un objectif donné [Jéz06, JCV12]. En informatique, elle peut être décrite comme la séparation des différents besoins fonctionnels et les préoccupations extra-fonctionnelles issues des exigences telles que la sécurité, la fiabilité, l'efficacité ou les performances. L'approche de l'Ingénierie Dirigée par les Modèles (IDM) consiste à fusionner (ou tisser) des solutions à ces différentes préoccupations dans du code. Ce que propose l'IDM est simplement de systématiser le processus que les ingénieurs expérimentés suivent à la main. L'ingénieur est moins confronté aux difficultés de programmation qu'avec les langages bas niveau, il peut alors se concentrer sur les problématiques fonctionnelles telles que les algorithmes de contrôle-commande. La conception de logiciel à base de modèle est plus communément appelée par son équivalent en anglais : le Model-Based-Design (MBD). Ainsi, dans un processus de développement en prototypage rapide (figure 1(a)) exploitant le MBD (figure 1(b)), le même modèle logiciel est utilisé tout au long du processus ce qui favorise la réutilisation de résultats antérieurs et donc les itérations rapides des créations logicielles.

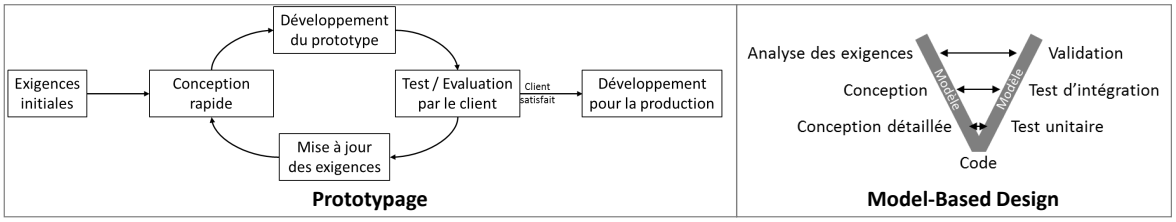


FIGURE 1 – (a) Modèle de développement en prototypage / (b) Model-Based Design (MBD)

Problématique

La croissance des performances des systèmes embarqués et la diversification des secteurs d'activité dans l'automobile mènent les systèmes embarqués à réaliser des fonctions

critiques. En effet, les systèmes embarqués et mécatroniques s'immiscent de plus en plus dans des fonctions qui impactent directement le comportement routier du véhicule (ABS, ESP, ACC, véhicule autonome) et peuvent ainsi provoquer en cas de panne des accidents pouvant porter atteinte à la sécurité des personnes, de l'environnement ou des biens. L'apparition de ces nouvelles contraintes participe à la complexification du développement de ces systèmes. Pour donner un cadre au développement de ces fonctions critiques à base de systèmes embarqués, il existe la norme internationale IEC 61508 [IEC10] qui concerne les industries traitant de la sécurité fonctionnelle des systèmes électriques/-électroniques programmables. Elle peut être employée dans de nombreux secteurs industriels mais il est parfois nécessaire de l'adapter pour un secteur particulier. Ainsi, pour traiter des problématiques spécifiques à l'automobile, la norme IEC 61508 a été dérivée et a donné lieu à la publication de la norme ISO 26262 en 2011 [ISO11]. L'ISO 26262 vise à garantir la sécurité fonctionnelle des systèmes électriques/électroniques dans les véhicules automobiles.

La SdF doit être prise en compte de plus en plus tôt dans les projets d'innovation à base de systèmes embarqués, c'est-à-dire dès les premiers prototypes, pour plusieurs raisons. Les risques encourus en cas de défaillance peuvent être très graves ; il est donc important de les anticiper le plus tôt possible afin d'éviter leur occurrence. Ensuite, l'apparition de la norme ISO 26262 a contribué à une plus grande sensibilisation des personnes aux contraintes de SdF. Les potentiels clients et futurs utilisateurs se soucient de plus en plus de cette problématique. Enfin, la réduction du temps de mise sur le marché implique de prendre en compte les exigences autres que celles fonctionnelles dès le prototypage afin de réduire les temps de développement et par conséquent le coût lors du passage en série.

Cependant, le prototypage fait l'objet de quelques particularités qui doivent être prise en compte pour assurer la bonne intégration des contraintes de SdF à ce stade. D'une part, de nombreuses PME sont impliquées dans des phases de prototypage, or leurs ressources et leurs compétences sont limitées, on ne peut pas leur imposer un processus de développement trop lourd qui requiert des effectifs importants et avec un haut niveau d'expertise en SdF. Certaines analyses de SdF notamment au niveau du logiciel embarqué sont encore conduites à la main, d'où la charge de travail supplémentaire. D'autre part, en phase de prototypage le processus de développement doit conserver une certaine flexibilité afin de pouvoir intégrer à tout moment de nouvelles exigences en fonction des évolutions du besoin client et des résultats obtenus à chaque version du prototype.

Les méthodologies actuelles se concentrent plutôt sur l'application d'analyse de SdF sur l'architecture fonctionnelle au niveau système. À ce stade, des décisions architecturales sont prises pour assurer une meilleure SdF comme par exemple la redondance de certains équipements. Les objectifs concernant la gestion de la SdF par le logiciel sont

également définis pendant cette phase. En revanche, au niveau logiciel, les analyses se font plus rares pour plusieurs raisons. Tout d'abord, les modèles sont plus larges qu'au niveau système (nombre de composants plus important) et par conséquent la quantité de résultats obtenue subit la même prise d'ampleur. Ensuite, dans les rares cas où des analyses sont menées jusqu'en phase de conception détaillée, elles sont conduites avec la même approche que pour les analyses systèmes, c'est-à-dire que les composants sont considérés comme des boites noires. Ainsi, les résultats obtenus ne sont pas toujours pertinents vis-à-vis de l'étude menée car les analyses reposent principalement sur la caractéristique structurale des modèles. Or, au stade de conception détaillée du logiciel, les composants présents sont des blocs basiques pour lesquels des informations comportementales ou de temps réel sont bien connues ; la prise en compte de ces caractéristiques, dans les analyses, permettrait d'affiner leurs résultats. À l'aide de ces analyses réalisées à un niveau avancé du développement logiciel, des recommandations d'implémentation du modèle pourraient être proposées afin de rendre le logiciel et par conséquent le système embarqué, plus robustes. Actuellement, la qualité de l'implémentation du modèle logiciel en phase de prototypage et plus particulièrement des stratégies de SdF repose essentiellement sur les compétences du développeur.

Contribution

La contribution de cette thèse est une méthodologie qui vise à améliorer les logiciels embarqués en se basant sur des recommandations issues d'analyse de SdF. Cette méthodologie, appelée ALEBAS, accompagne le développeur dans la transition entre les premiers développements du projet (Preuve de Concept –ou *Proof of Concept*– (PoC)) et les prototypes suivants qui intègrent plus d'exigences et notamment celles de SdF. ALEBAS peut également être employée dans des phases plus avancées du développement c'est-à-dire proche de la mise en production. La méthodologie proposée est basée sur l'utilisation des techniques de SdF suivantes : les Arbre de Défaillances –ou *Fault Tree*– (AdD) et l'Analyse des Modes de Défaillance et de leurs Effets (AMDE) logicielle (aussi appelée Analyse des Effets des Erreurs de Logiciel (AEEL)). Une des principales particularités de la méthodologie est de mener une analyse préalable des blocs basiques qui sont communément utilisés dans le développement logiciel. Les résultats obtenus à partir de l'analyse individuelle fournissent des données d'entrée précises et étoffées pour l'application de l'analyse des AdD et de l'AMDE logicielle. À partir de ces analyses, des recommandations peuvent être suggérées pour améliorer le modèle logiciel et dans certains cas, elles peuvent aussi concerner l'architecture système. La proposition de la méthode est accompagnée d'un outil logiciel qui automatise la construction des AdD et de l'AMDE.

La méthodologie ALEBAS contribue à :

- améliorer la pertinence des résultats des analyses réalisées en phase de conception détaillée ; ce qui implique en plus une réduction de la quantité de données obtenues.

- rendre accessible la prise en charge des contraintes de SdF en proposant des recommandations d'implémentations précises, pertinentes et proches du code ; ainsi elle facilite ce travail pour des ingénieurs/développeurs qui ne sont pas experts en SdF.

- conserver une flexibilité d'approche en n'imposant pas fortement le formalisme du modèle à analyser ; ainsi la méthodologie s'adapte aux différentes pratiques des entreprises et des développeurs. Toutefois, des règles de bonnes pratiques de développement sont indiquées pour garantir les meilleurs résultats des analyses.

- assurer la répétabilité et la cohérence des analyses de SdF grâce à leur automatisation ; ainsi elle favorise la gestion des évolutions récurrentes du logiciel en phase de prototypage.

L'ensemble du manuscrit comporte quatre chapitres : le premier expose les domaines concernés par cette thèse et les trois chapitres suivants décrivent la contribution tant au niveau théorique que pratique.

Le chapitre 1 présente un état de l'art sur les processus et les pratiques de développement des systèmes embarqués et mécatroniques, avec un regard particulier sur leur utilisation dans un contexte industriel et d'innovation. La suite de l'étude bibliographique s'intéresse aux méthodes et méthodologies de SdF qui sont employées lors du développement des systèmes embarqués.

Le chapitre 2 décrit la contribution originale de cette thèse qui repose sur la proposition de la méthodologie ALEBAS. Cette méthodologie a pour objectif d'améliorer les logiciels embarqués vis-à-vis de la SdF grâce à l'application d'analyse de SdF. La méthode intervient en phase de conception détaillée des modèles de logiciel, dans le but d'apporter des recommandations d'implémentation au développeur. Au début du chapitre, des règles générales de bonnes pratiques de développement et d'approche SdF à pratiquer lors des étapes précédant la conception détaillée du logiciel, sont présentées afin de formaliser les données d'entrée de la méthodologie ALEBAS, pour obtenir des résultats optimaux. Ensuite, l'organisation générale de la méthodologie ALEBAS est exposée avant de détailler chacune des tâches qui la composent.

Le chapitre 3 décrit la mise en œuvre de la méthodologie ALEBAS à l'aide d'un outil logiciel développé dans l'environnement Matlab. Cet outil logiciel s'adresse plus particulièrement à l'automatisation des analyses de SdF c'est-à-dire la génération des AdD et du tableau de l'AMDE logicielle.

Le chapitre 4 vise à éprouver la méthodologie ALEBAS et l'outil logiciel associé, sur un cas d'étude industriel. Ce cas d'étude est basé sur une fonction d'un prototype de véhicule autonome : la robotisation de la pédale d'accélérateur.

Chapitre 1

État de l'art et travaux connexes

Sommaire

L'industrie des transports fait face à de nouveaux challenges dans le secteur des systèmes embarqués et mécatroniques avec des innovations majeures comme dans l'automobile avec le déploiement des ADAS [PCSB16] en vue d'aboutir au véhicule autonome [PJ14]. Ainsi, la complexité et la nature multidisciplinaire des systèmes embarqués ont augmenté. Leurs développements requièrent de plus en plus d'attention depuis que les ordinateurs sont utilisés pour contrôler des processus temps réel et à caractère critique [Lev86]. Aujourd'hui, la SdF est une activité d'ingénierie incontournable lors du développement des systèmes embarqués et mécatroniques.

1.1 Systèmes embarqués et mécatroniques

Les systèmes embarqués combinent du matériel électrique/électronique et des fonctionnalités logicielles, conçus ensemble afin d'exécuter des tâches précises. Ils requièrent une attention particulière lors de leur développement du fait des limites et contraintes auxquelles ils sont confrontés. Ces limitations concernent notamment l'énergie disponible, la puissance de calcul, le volume d'occupation et le coût. Malgré ces restrictions, les systèmes embarqués doivent souvent, en plus des fonctions pour lesquelles ils sont conçus, être en mesure de gérer eux-mêmes leur consommation d'énergie, d'opérer leurs tâches dans un temps imparti (temps réel) et de plus en plus garantir leur SdF. Le potentiel des systèmes embarqués continue de s'accroître considérablement et les mène à réaliser des fonctions toujours plus complexes et diversifiées au sein des véhicules. Aujourd'hui, on les retrouve dans toutes les fonctions typiques de l'automobile : le contrôle moteur, le freinage, la direction, la liaison au sol ou encore la sécurité et le confort. Dans ces secteurs, les systèmes embarqués remplacent ou supportent les fonctions mécaniques ; on les appelle alors les systèmes mécatroniques [Ber07].

Le système mécatronique de la figure 1.1 intègre des composants multiphysiques (mécaniques, hydrauliques, pneumatiques ou encore thermiques), de l'électronique et du logiciel.

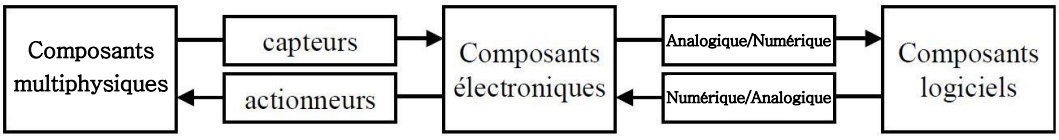


FIGURE 1.1 – Système mécatronique [SK97, Mih07]

Le développement d'un système embarqué, pour être mené correctement, requiert le suivi d'un processus à définir dès le commencent et celui-ci doit être adapté aux besoins du système à concevoir et aux capacités de développement. Ainsi, il existe différents types de cycle de développement présentant chacun différents avantages et inconvénients selon les objectifs du système à réaliser.

1.1.1 Cycles de développement

Le développement d'un système nécessite de suivre une démarche afin de travailler efficacement. Différents types de cycle de développement ont été développés [Rup10], chacun avec leurs avantages et inconvénients. Lors du démarrage d'un projet, il est important de choisir celui qui correspond le mieux aux attentes et objectifs. Un cycle de développement décrit les différentes étapes, leur enchaînement et la manière dont elles sont abordées. Nous abordons ci-dessous les modèles les plus largement utilisés et ceux en lien avec nos travaux.

1.1.1.1 Modèle en cascade

Le modèle en cascade est mentionné pour la première fois dans [Ben56] puis il est mis à jour par [Roy70] en 1970 qui le représente comme dans la figure 1.2. Il fait référence à un cycle de développement linéaire et séquentiel. Chaque étape doit être complétée avant de passer à la suivante, en revanche à la fin de chaque étape une revue est réalisée afin de contrôler que le projet est en bonne voie, sinon le projet peut être arrêté ou repris à des étapes antérieures. Ce type de processus peut être utilisé dans des projets pour lesquels il n'y a pas de doute sur les exigences et les moyens pour les atteindre.

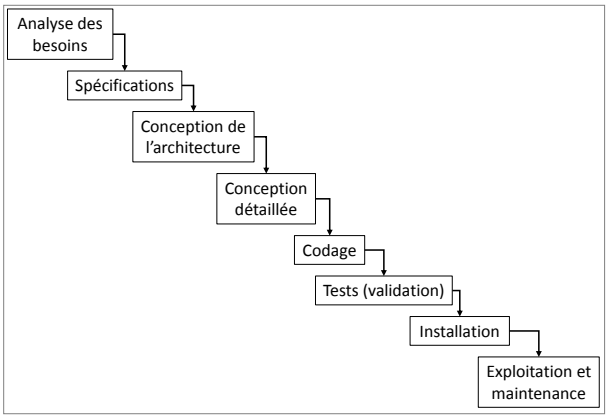


FIGURE 1.2 – Modèle de développement en cascade

1.1.1.2 Modèle en V

Le modèle de développement en V a été développé par la NASA et il est présenté pour la première fois en 1991 [FM91]. Ce modèle est une variation de celui en cascade représente par une forme en V comme illustré en figure 1.3(a). La branche descendante du V représente l'évolution des exigences en composants de plus en plus réduit et précis selon le principe de décomposition. La branche ascendante du V représente l'intégration et la vérification des composants à travers différentes étapes de tests et d'assemblage. Comme pour le modèle en cascade, le processus d'exécution du cycle en V est séquentiel. Chaque

étape doit être complétée avant de passer à la suivante. Les phases de test sont planifiées et préparées en parallèle d'une tâche correspondante de la phase de développement. Il est souvent considéré comme un cycle de développement générique et macroscopique, mais en pratique le développement s'apparente plus à une procédure itérative dans laquelle s'enchaîne plusieurs cycles en V [Sei04] (figure 1.3(b)).

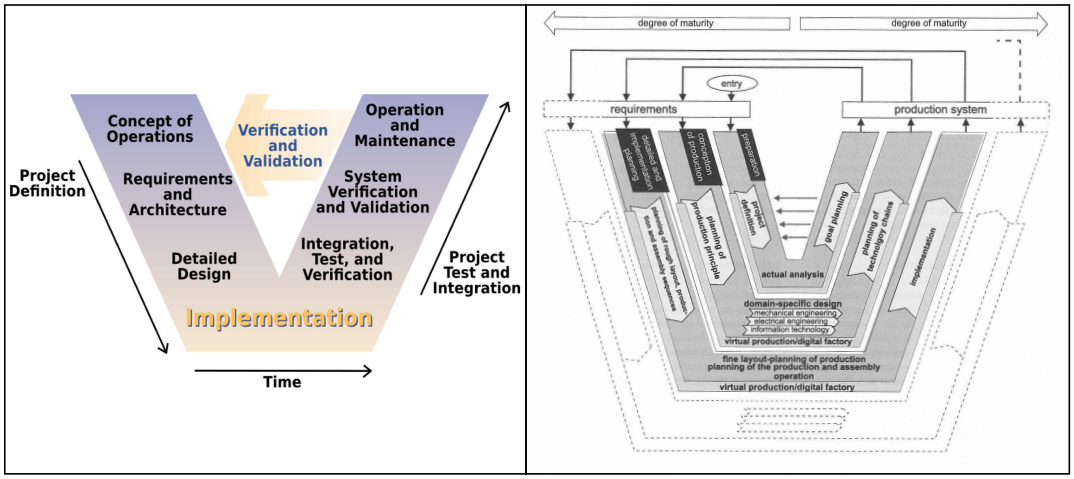


FIGURE 1.3 – (a) Modèle de développement en V / (b) Cycle en V successifs

1.1.1.3 Modèle de prototypage

Le prototypage est décrit par James Martin dans son livre « Rapid Application Development » [Mar91]. Le processus de développement en prototypage repose sur l'idée que les exigences ne sont pas figées dès le début du cycle de développement, bien au contraire, elles sont amenées à être affinées au cours du développement de manière itérative (figure 1.4). Les évolutions des exigences se font en accord avec le client qui évalue le prototype actuel et définit les changements à réaliser pour la version suivante. Le modèle de prototypage est à utiliser lorsque les exigences ne sont pas toutes connues au départ, c'est notamment le cas lors de développement de système innovant. Le risque est de devoir opérer de nombreuses itérations des prototypes et que le développement prenne une ampleur non prévue au démarrage. L'évaluation du client et la mise à jour des exigences doivent être menées avec précision pour limiter ces débordements.

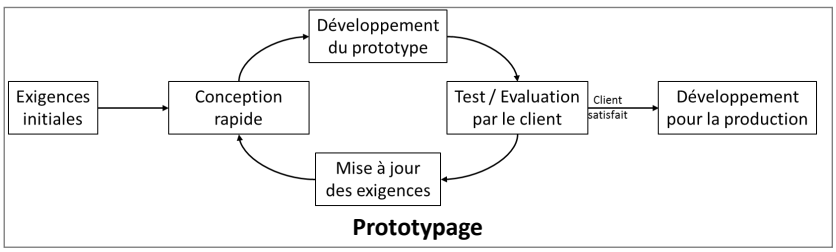


FIGURE 1.4 – Modèle de développement pour le prototypage

1.1.1.4 Modèle en spirale

Barry Boehm modifie le modèle en cascade en 1986 [Boe86] en introduisant plusieurs itérations qui sont petites au commencement puis prennent de plus en plus d'ampleur comme décrit dans la figure 1.5. Le modèle en spirale est similaire à un processus incrémental qui accorde plus d'importance à l'analyse des risques. Dans ce cycle de développement, le système à développer passe à plusieurs reprises dans les quatre phases : définition des objectifs, planification, analyse de risque, et développement et test. Chacune de ces phases pouvant être décomposées en plusieurs étapes. Ce type de processus est à utiliser dans le cadre de projet où les exigences sont complexes, des changements en cours de développement sont à prévoir et l'évaluation des risques est importante.

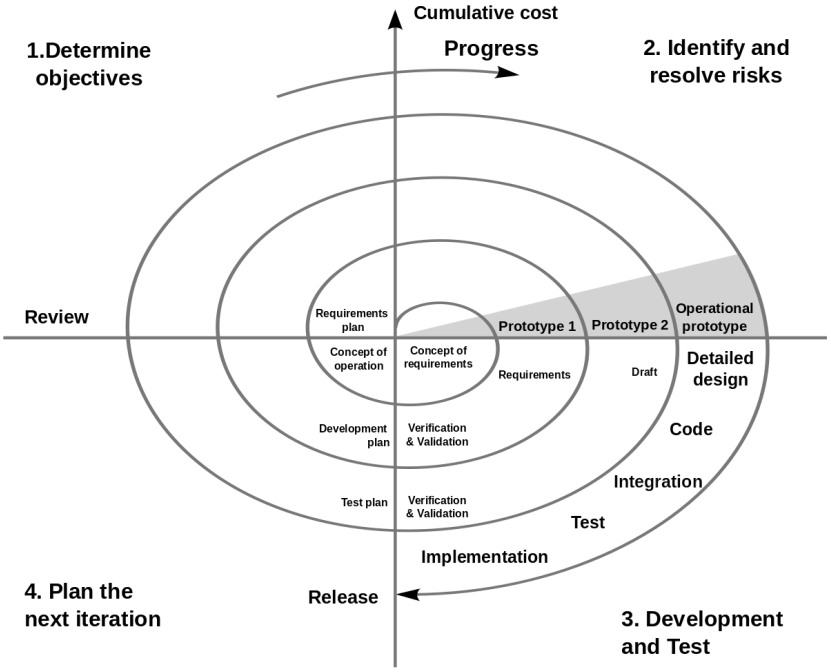


FIGURE 1.5 – Modèle de développement en spirale

1.1.2 Model-Based Design et outils de développement

Le MBD est l'application de la modélisation pour supporter les activités de définition des exigences, de conception, d'analyse, de vérification et de validation commençant dans les phases conceptuelles de conception et se prolongeant tout au long du cycle de développement [BFB14]. L'environnement de développement en MBD le plus populaire est Matlab-Simulink.

1.1.2.1 Matlab - Simulink

Simulink [1] est un environnement de diagramme fonctionnel destiné à la simulation multi-domaine et à l'approche de conception par modélisation (MBD). Il prend en charge la conception et la simulation au niveau système, la génération automatique de code, ainsi que le test et la vérification en continu des systèmes embarqués.

1.1.2.2 Chaîne d'outils MotoHawk

La chaîne d'outils MotoHawk [Woo] et les calculateurs associés sont destinés au développement de prototypes, de projets d'innovation et à une distribution en petite série, majoritairement, pour des systèmes embarqués dans les transports. MotoHawk est intégrée comme une bibliothèque complémentaire dans Simulink et est dédiée à l'implémentation de systèmes de contrôle embarqués. Les blocs de la bibliothèque MotoHawk sont utilisés pour interfacer les Entrées/Sorties –ou *Inputs/Outputs*– (I/O) matérielles (du calculateur) avec les I/O du logiciel. À partir des autres outils de la chaîne, du code est généré puis compilé afin de pouvoir être embarqué sur les calculateurs dédiés. Les calculateurs MotoHawk sont des « composants sur étagère », leur conception selon les normes de qualité du monde automobile en fait des calculateurs robustes. La chaîne d'outils MotoHawk est un outil de prototypage-rapide, ainsi il se concentre principalement sur la pratique des aspects fonctionnels plutôt que ceux de la SdF. L'entreprise ayant initié ces travaux de recherche est fournisseur de la chaîne d'outils MotoHawk et l'exploite dans ces projets de Recherche & Développement, ainsi cette thèse s'attache à poursuivre son utilisation.

1.2 Sûreté de fonctionnement (SdF)

Depuis plusieurs décennies, on observe une forte augmentation des performances des systèmes de notre quotidien tels que les moyens de transport ou les télécommunications. Un bon niveau de performance exige, entre autres, un fonctionnement sans défaillance. Parallèlement à cet accroissement des performances, la complexité de ces systèmes s'est également considérablement accrue. Or, la complexité implique une multiplication des sources potentielles de défaillances. La réduction des délais de mise sur le marché limite l'étendue des phases d'essai et donc la quantité de retours d'expérience. Le domaine de la sûreté de fonctionnement (SdF) a ainsi émergé avec ces méthodes d'ingénierie dans l'objectif d'anticiper les risques en amont et ainsi assurer le bon fonctionnement des systèmes durant leur cycle de vie. La SdF ne peut être caractérisée par une grandeur unique. La SdF d'un système repose sur quatre facteurs : sa fiabilité, sa disponibilité, sa maintenabilité et sa sécurité [AC08b, AC08a, Fau08, Gir05].

1. The MathWorks - www.mathworks.com

La **fiabilité** est définie comme l'aptitude d'un dispositif à fonctionner sans défaillance, dans des conditions données, pendant un temps donné. Lorsqu'on quantifie ce paramètre, on exprime cette aptitude par la probabilité d'atteindre un temps t sans défaillance.

La **disponibilité** est définie comme l'aptitude d'un dispositif, sous les aspects combinés de sa fiabilité, de sa maintenabilité et de la logistique de maintenance, à remplir ou à être en état de remplir une fonction à un instant donné ou dans un intervalle de temps donné.

La **maintenabilité** est définie comme l'aptitude d'un dispositif à être maintenu (prévention) ou rétabli (correction) dans son état de fonctionnement, en s'intéressant surtout au temps nécessaire pour réaliser ces opérations de maintenance. Lorsqu'on quantifie ce paramètre, on exprime cette aptitude par la probabilité d'effectuer l'opération de maintenance dans une durée au plus égale à un temps t.

La **sécurité** est définie comme l'aptitude d'un dispositif à ne pas générer d'événements critiques ou catastrophiques pouvant porter atteinte à la sécurité des personnes, de l'environnement ou des biens. Lorsqu'on quantifie ce paramètre, on exprime cette aptitude par la probabilité que le système évite de faire apparaître ces événements critiques. Le terme français « sécurité » est ambigu dans le contexte de la SdF car il recouvre indistinctement les termes anglais « safety » (menaces naturelles) et « security » (menaces malveillantes). Pour rendre compte de cette distinction en français certains préfèrent l'utilisation des termes sécurité-innocuité (safety) et sécurité-confidentialité (security).

Afin d'assurer une SdF optimale d'un système, des méthodes d'ingénierie ont été conçues pour être pratiquées au cours du processus de développement et du cycle de vie du système.

1.2.1 Méthodes classiques

Une étude SdF se conduit à l'aide de méthodes d'analyse qui ont été élaborées spécifiquement pour ce domaine. Les plus connues sont l'analyse des AdD et l'AMDE que nous aborderons ensuite. Il existe de nombreuses autres méthodes comme l'Analyse Préliminaire des Risques (APR) [MIL, Lie76], Hazard and operability studies (Hazop) [AotCI77] ou encore l'analyse des arbres d'événements [CE05].

1.2.1.1 Arbre de défaillances (AdD)

Les AdD ont été développés conjointement par Bell Telephone Laboratories et l'armée de l'air américaine au début des années 60. Leur objectif était d'étudier le système de contrôle de lancement des missiles balistiques intercontinentaux « Minuteman » [HB04]. Par la suite, les AdD ont été appliqués à de nombreux domaines : le spatial, l'aéronautique, le militaire, l'automobile, le nucléaire, la chimie ou encore la médecine. Avec la

progression des capacités des ordinateurs, la construction et l'évaluation des AdD ont été facilitées faisant ainsi de l'analyse des AdD un moyen efficace pour la gestion et la documentation de l'évaluation de la fiabilité d'un système. Aujourd'hui, l' analyse des AdD est la technique d'analyse de SdF déductive la plus communément utilisée [Pum99].

Un AdD représente graphiquement, la combinaison des événements et conditions qui contribuent à l'occurrence d'un événement redouté disposé au sommet de l'arbre. La majeure partie de la communauté travaillant sur les systèmes à caractère critique se réfère au guide de la commission nucléaire américaine [VGRH81] comme définition de la syntaxe de construction des AdD. Bien que les domaines d'application des AdD soient très variés, les symboles représentants les événements, les conditions et les opérateurs logiques demeurent remarquablement standards. Les symboles conventionnels et les plus communément utilisés sont exposés en figure 1.6. Les événements présents dans l'arbre peuvent s'accompagner de leur probabilité d'occurrence; ainsi, par combinaison, il est possible d'évaluer la probabilité d'occurrence de l'événement redouté au sommet de l'arbre.

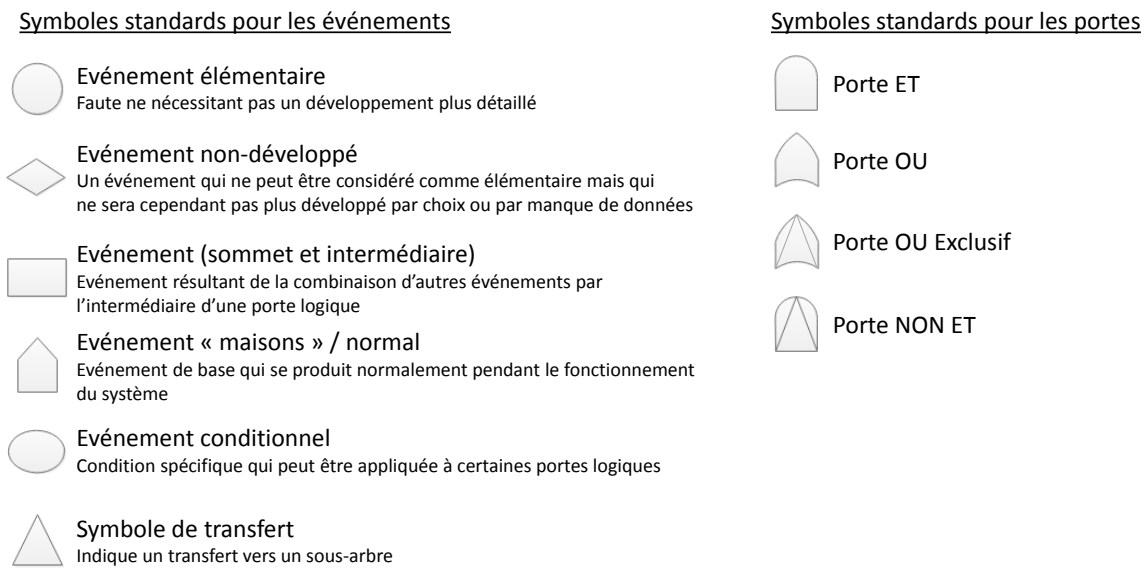


FIGURE 1.6 – Symboles standards pour la construction des AdD

La figure 1.7 présente un système simple d'éclairage composé de l'alimentation, d'un interrupteur et de deux lampes. Un exemple d'arbre de défaillances est aussi présenté dans cette figure 1.7; il a été construit à partir de l'étude de l'événement redouté « absence d'éclairage dans la pièce ».

1.2.1.2 Analyse des modes de défaillance et de leurs effets (AMDE)

L'AMDE est une technique inductive largement utilisée dans l'industrie [IEC91] pour réaliser des analyses de SdF. L'AMDE évalue, élimine et anticipe les potentiels modes de

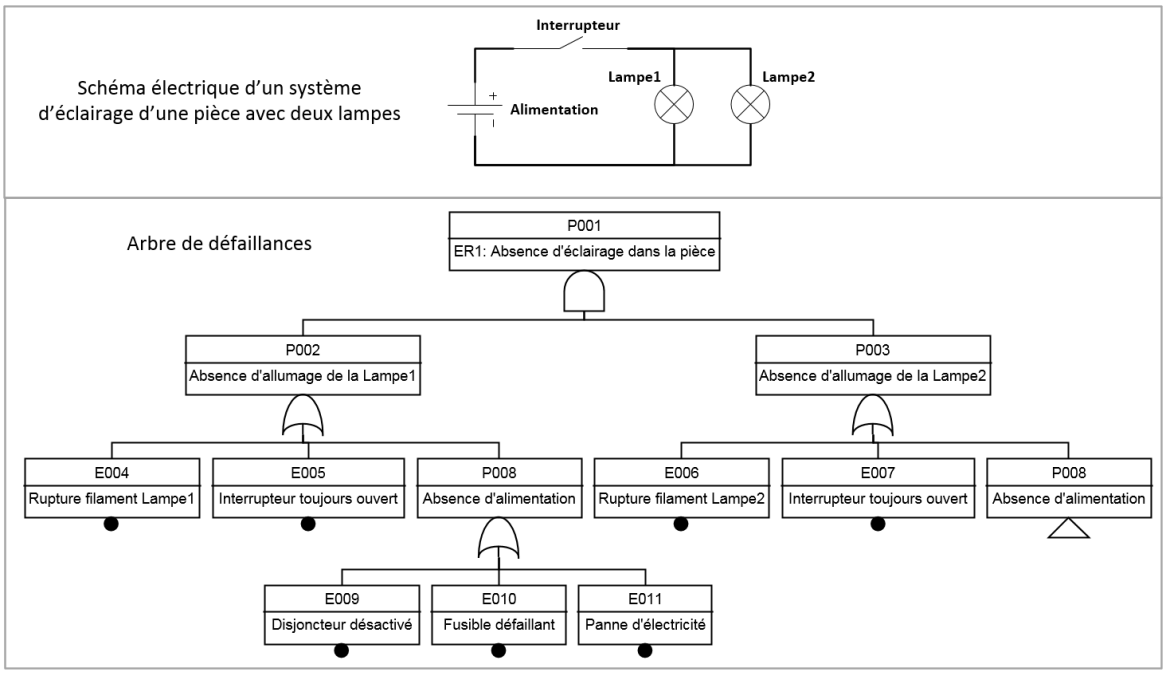


FIGURE 1.7 – Exemple d'un arbre de défaillances sur l'étude de l'événement redouté « absence d'éclairage dans la pièce » d'un système d'éclairage

défaillance d'un système. Toutes les informations de l'étude sont rassemblées dans un tableau spécifique qui comporte deux parties : la première inclut l'identification et l'évaluation des causes et de leurs effets ; la deuxième partie présente les actions recommandées et celles qui ont été appliquées. Le tableau 1.1 présente comme exemple un extrait de tableau résultant de l'AMDE menée sur le système d'éclairage de la figure 1.7. Le principal avantage de l'AMDE est son habilité à collecter et à identifier de façon exhaustive les défaillances initiales et leurs effets locaux. Elle peut aussi être utilisée dans un processus de maintenance pour le diagnostic et les solutions de réparation.

L'un des premiers articles discutant de l'AMDE logicielle est [Rei79]. Son approche a été plus largement introduite par Goddard [GRT00] qui décrit deux types d'AMDE logicielle pour les systèmes embarqués. Homkes en présente une troisième dans [HEK05].

- L'**AMDE logicielle au niveau système** peut être utilisée pour évaluer l'efficacité de l'architecture logicielle. Elle doit être pratiquée dès que possible dans le processus de développement logiciel.

- L'**AMDE logicielle détaillée** permet de valider que le logiciel a été construit pour atteindre les exigences de SdF spécifiées en analysant chaque composant. Dans [BW01], Bowles indique que l'AMDE logicielle détaillée est d'autant plus efficace lorsque le matériel n'est pas muni de protection. Les systèmes de contrôle embarqués font partie de ce type de système.

- L'**AMDE logicielle des interfaces** analyse les défaillances affectant les interfaces entre

les modules logiciels ou entre le logiciel et le matériel.

Pour désigner l'AMDE logicielle, on utilise aussi le terme AEEL.

TABLEAU 1.1 – Exemple de tableau AMDE basée sur le système d'éclairage

Analyse					**Correction**	
Fonction	**Élément**	**Mode de défaillance**	**Cause**	**Effet**	**Action corrective**	**Action appliquée**
Commande de l'éclairage	Interrupteur	Commande intempestive de l'interrupteur	Usure du mécanisme	Allumage intempestif des lampes	Réparation de l'interrupteur	Remplacement de l'interrupteur
					Remplacement de l'interrupteur	

1.2.2 Analyse de SdF sur le logiciel

Les analyses de SdF peuvent être conduites sur le logiciel mais elles peuvent être menées à différents niveaux : sur les exigences, l'architecture, la conception détaillée et le code. La plupart des analyses de SdF conduites sur le logiciel se font au niveau architectural (dans la continuité de l'analyse système). Ensuite, on trouve quelques propositions d'analyse au niveau du code, mais les analyses au niveau de la conception détaillée se font très rares.

Dans [LS83], l'analyse des arbres de défaillances est conduite sur du code Ada. Il est bien relevé que l'approche de l'analyse de logiciel est bien différente de celle menée sur du matériel. De plus, il est décrit que cette méthode est à employer sur des programmes à structure simple pour étudier la logique de l'algorithme. Il est indiqué la difficulté de mener une analyse d'AdD sur tout le code étant donnée la quantité de résultats, mais cette étude reconnaît que la construction d'arbres même partielle peut se révéler intéressante. La tolérance logicielle aux fautes est une procédure coûteuse pour déterminer les parties à protéger, ainsi même des arbres partiels peuvent aider à déterminer les modules et les fonctions critiques.

Dans [PS08], les travaux s'intéressent à la génération automatique de l'AMDE pour du code et l'exemple est mené sur du code JAVA. Le code source du logiciel à étudier est transformé en modèle de propagation de fautes afin de pouvoir être étudié. Il s'agit d'un graphique dans lequel les opérations sont les transitions et les variables sont les nœuds. Grâce à ce modèle, il est possible d'étudier la propagation et par conséquent l'impact d'une faute en entrée. Cette méthode peut être appliquée à d'autres langages mais l'outil de transformation en modèle est dépendant du langage. La principale limite de cette méthode réside dans son besoin de tracer les dépendances entre les différentes variables, pour certains programmes cela peut se révéler très problématique.

Dans [OYCS05], une approche est proposée pour générer des AdD à partir de diagrammes de blocs fonctionnels utilisés dans l'implémentation de logiciels critiques. La méthode proposée aborde les diagrammes de blocs fonctionnels de façon générique en s'intéressant aux opérations les plus communes : logique, arithmétique, de comparaison, de sélection et de retard unitaire. Pour chaque type d'opération et chaque mode de défaillance potentiel de ce bloc, un template d'AdD est développé contenant les fautes potentielles pouvant occasionner l'occurrence du mode de défaillance étudié et les conditions d'apparition. Les AdD du diagramme complet étudié sont construits par propagation dans celui-ci. À chaque fois qu'un bloc est rencontré et selon le type d'erreur propagé dans l'étude, le template d'arbre correspondant est ajouté à l'AdD.

Synthèse : L'application d'analyse des AdD et de l'AMDE visant le logiciel a été abordée dans quelques travaux. On constate que l'utilisation de ces méthodes peut s'appliquer à différents stades du développement. Elles présentent quelques difficultés dans les phases détaillées et de codage pouvant limiter leur utilisation et exhaustivité, mais ce qui n'empêche pas d'avoir des résultats intéressants. Malgré l'apparition des solutions de développement à l'aide de modèle (MBD), la pratique de ces analyses n'a été abordée que de manière générique, éclipsant ainsi des problématiques spécifiques aux outils et au type de logiciels étudiés, comme les notions de temps réel pour un logiciel embarqué.

1.2.3 Model-Based Safety Analysis

Les techniques classiques d'évaluation de la SdF telles que les AdD et l'AMDE, sont toujours employées en revanche, elles sont traditionnellement pratiquées manuellement ce qui pose problème pour l'étude de systèmes complexes [SP11, Sha11]. Pour cette raison, ces méthodes classiques sont rarement réalisées plus d'une fois au cours du processus de développement et souvent en fin de développement alors que les choix de conception ont déjà été finalisés. Un autre problème soulevé est le manque de méthode systématique pour gérer conjointement les modèles fonctionnels et les artefacts de SdF comme ces deux pratiques sont traditionnellement menées séparément. Ainsi pour pallier à certaines de ces difficultés, des recherches se sont intéressées au développement de techniques d'évaluation de SdF plus robustes et plus efficaces grâce à de l'automatisation. Model-Based Safety Analysis (MBSA) introduit l'utilisation de modèle au cœur de la conception et du processus d'évaluation. MBSA étend le modèle fonctionnel nominal avec des informations sur les défaillances potentielles des composants pour obtenir le modèle de faute [JHMW06] ou aussi appelé le modèle d'erreur. En parallèle du développement d'une méthode MBSA, un outil logiciel est implémenté pour automatiser l'application de différentes techniques d'évaluation de la SdF comme la génération d'AdD ou l'AMDE. La suite de cette section décrit des exemples de méthodes MBSA existantes.

Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS)

Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [PM99, PMSH01] est une méthode d'analyse de SdF des systèmes par le biais de génération d'AdD et de tableaux AMDE. HiP-HOPS propose une amélioration de la méthode Failure Propagation Transformation Notation (FPTN). Les modèles de défaillances ne sont plus flottants avec HiP-HOPS car les équations de dysfonctionnement sont annotées directement dans les composants du modèle fonctionnel. Les AdD sont construits par propagation dans la structure du modèle système. De plus, pour la description des annotations, HiP-HOPS introduit une nouvelle approche inspirée de l'AMDE, appelée Interface Focused - FMEA (IF-FMEA). HiP-HOPS s'interface avec des outils industriels tels que Matlab - Simulink [PMSH01] ou SimulationX[2] et des langages de description d'architectures spécifiques, tel que Architecture Analysis and Design Language (AADL). En plus de ces caractéristiques pour l'analyse de SdF, la méthode HiP-HOPS a récemment été étendue avec des facultés d'optimisation architecturale des systèmes. Les critères d'optimisation sont basés sur l'amélioration de la fiabilité tout en réduisant le coût [MPS+11]. Au delà des contraintes des outils, la méthode HiP-HOPS peut être appliquée sur tout type d'architecture système dont les composants représentent du matériel ou du logiciel, et dont les composants s'échangent des flux de données ou d'énergie. Pour la gestion de la complexité, le modèle système peut être une hiérarchie de sous-systèmes incorporant eux-même des architectures d'éléments de plus bas niveau. Le modèle peut prendre la forme de modèle fonctionnel ou architectural c'est-à-dire que les composants peuvent aussi bien représenter des fonctions que des éléments, tels qu'un capteur, un processeur ou un module logiciel. HiP-HOPS n'est pas destinée à l'analyse de SdF pour du logiciel à une étape de conception détaillée. En effet, l'analyse ne considère pas les composants basiques qui sont utilisés à un niveau avancé de la conception du logiciel pour implémenter les algorithmes de contrôle/commande, tels que les blocs constant ou gain. Par conséquent, les améliorations et optimisations du modèle ne sont pas non plus adressées pour les modèles logiciels.

Altarica

Altarica [PR99, Poi09] est un langage de modélisation formelle qui a été développé pour supporter l'analyse de SdF et de fiabilité. C'est un langage de sûreté qui permet de décrire le comportement des systèmes lorsque des fautes se produisent. Altarica permet la modélisation hiérarchique d'un système grâce à un représentation par nœuds qui représentent des composants ou des sous-systèmes. Chaque nœud peut être représenté comme un automate d'interface défini par des variables d'état, des variables de flux et des

2. SimulationX - www.simulationx.com

événements. Les variables de flux permettent l'interaction avec les autres nœuds. Les variables d'état et les événements forment l'automate qui décrit le comportement du composant. La description de l'automate est spécifiée par des transitions et des expressions. Les expressions décrivent les contraintes qui doivent être vérifiées par les variables. Altarica est principalement un langage de modélisation textuelle. Certains outils proposent un support graphique. Parmi les suites logicielles supportant Altarica pour la modélisation, l'analyse et la simulation des modèles, on peut citer : Safety Designer (Dassault Systemes), SimFia (EADS Apsys), OpenAltarica tools (IRT SystemX et Altarica Association) ou encore ARC/Altarica Studio (Université de Bordeaux).

Build-It-Safe

Le projet et la chaîne d'outils Build-It-Safe [VLAL13] visent à proposer une méthodologie soutenant le processus d'analyse de SdF et reposent sur des approches basées sur les modèles dans le contexte du domaine de l'automobile. Leur méthodologie permet de réaliser les analyses de SdF sur différents types de modèles comme Langage de Modélisation Unifié –ou *Unified Modeling Language*– (UML) /Systems Modeling Language (SysML) ou Matlab-Simulink. Le mécanisme consiste à transformer le modèle étudié en un fichier XML ayant un formalisme bien défini pour pouvoir être utilisé dans l'outil d'analyse. Ensuite, la liste des composants est importée à l'aide du fichier XML dans l'outil d'analyse. Chacun des composants peut alors être annoté avec des informations sur ses défaillances et ses équations de dysfonctionnement, de manière similaire à HiP-HOPS. Les résultats sont plutôt concluants avec des modèles UML/SysML car ils ne contiennent que des informations structurelles, c'est-à-dire sur l'organisation des composants entre eux. En revanche, avec les modèles Matlab-Simulink, le même procédé basé sur la topologie du modèle manque de précision. En effet, ce type de modèle ne contient pas seulement l'information structurelle des composants entre eux mais il décrit aussi un comportement induit par les blocs élémentaires qui composent ce modèle. Ce comportement a une influence sur l'apparition des événements redoutés qui n'est pas gérée dans Build-It-Safe.

AADL Error Annex

AADL [SAE, FLVC05] est un langage spécifique de domaine pour la spécification et l'analyse de logiciels embarqués temps-réel. La syntaxe AADL est très riche et extensible. L'AADL Error Annex, par exemple, est une extension qui permet la modélisation du comportement en défaillance des composants. La modélisation est faite grâce à des automates stochastiques qui se composent d'un nombre fini d'état d'erreur. L'AADL Error Annex dans sa forme la plus simple peut être considéré comme une approche logique de défaillance, car il décrit la propagation et les comportements déviants à travers le système. Mais le modèle de défaillance n'est pas un modèle flottant (indépendant) car il s'interface

avec le modèle du système nominal. Il peut être considéré comme une approche hybride où les deux modèles de défaillance et nominal sont spécifiés. Cela permet la réutilisation de type d'erreur grâce à une bibliothèque d'erreur. Les machines à état de comportement en erreur peuvent aussi être réutilisées ou raffinées en les utilisant d'une façon similaire à l'approche objet. Il est aussi possible de modéliser les mitigations de fautes, ainsi que les comportements erronés silencieux. Certains travaux ont pour objectif l'unification d'AADL Error Annex et Altarica.

Synthèse : La plupart des méthodes MBSA portent une grande importance à la structure du système étudié. En effet, elles sont souvent pratiquées à un stade de conception où les composants sont considérés comme des boîtes noires, ainsi un composant peut avoir un comportement différent en fonction de sa position dans le système. Ceci implique pour l'étude SdF, la nécessité de conduire des analyses locales de chaque composant en tenant compte de leur implantation dans le système pour construire les modèles de faute. Du fait de travailler avec des boites noires, le comportement des composants et les notions de performances temps réel du système sont rarement abordés.

1.2.4 Norme automobile : ISO 26262

L'expansion de l'utilisation des systèmes Électrique/Électronique (E/E) soulève le problème de la coexistence de fonctions et services ayant des niveaux de criticité différents au sein d'un même système. Une fonction critique peut conduire à l'occurrence d'un événement redouté si une erreur apparaît lors de son exécution. Aussi, de nombreux acteurs sont impliqués dans le développement d'une voiture : le constructeur (Original Equipment Manufacturer (OEM)) et plusieurs équipementiers/fournisseurs (Tier 1, Tier 2) qui développent des sous-parties du système définies par le constructeur. Chaque entreprise à son propre processus de développement, ainsi il est nécessaire de définir et de suivre des règles rigoureuses de conception, afin de justifier les méthodes de travail et la documentation tout au long du développement. Par conséquent, toute les activités vouées à la SdF doivent être tracées. Il est important de noter que dans le domaine automobile, il n'existe aucune exigence légale qui vous impose des pratiques de développement, ni d'autorité de certification afin d'autoriser la mise sur le marché de nouveaux systèmes E/E. Des premiers OEM et équipementiers ont choisi volontairement de se conformer à la norme IEC 61508. Au contraire, dans l'aéronautique où les entreprises se doivent de suivre l'ARP 4754/ED-79 (SAE International) pour le développement des avions civils et des systèmes, ainsi que l'ED-12/DO-178 pour les considérations logicielles. Dans le domaine nucléaire, ils sont aussi contraints de respecter les normes 50-SG-D3 et 50-SG-D8 qui ne sont pas réservées au nucléaire. La norme IEC 61508 s'applique sur tout le processus de développement d'un système et elle décrit des étapes à respecter dans le but

d'assurer la SdF des composants E/E. Plus particulièrement, la norme IEC 61508 définit des objectifs pour la spécification, la conception, l'implémentation et l'évaluation des systèmes programmables et E/E.

Depuis 2011, une version dérivée de la norme IEC 61508 appelée ISO 26262 a été publiée, elle est destinée au secteur automobile. Cette norme est le résultat d'un travail entre les principales entreprises du secteur dans l'objectif de spécifier les pratiques pour : la documentation, l'interaction entre les acteurs, et les techniques et méthodes pour justifier la SdF des systèmes. Ainsi, elle facilite les échanges entre les OEM et les différents fournisseurs en indiquant une base d'exigences à suivre. La norme ISO 26262 s'adresse aux événements redoutés pouvant être causés par un dysfonctionnement des systèmes E/E eux-mêmes mais aussi par des soucis d'interaction entre eux. La norme est divisée en dix parties comme décrit en figure 1.8, chaque partie est destinée à une phase du processus de développement ou à la description de pratiques attenantes.

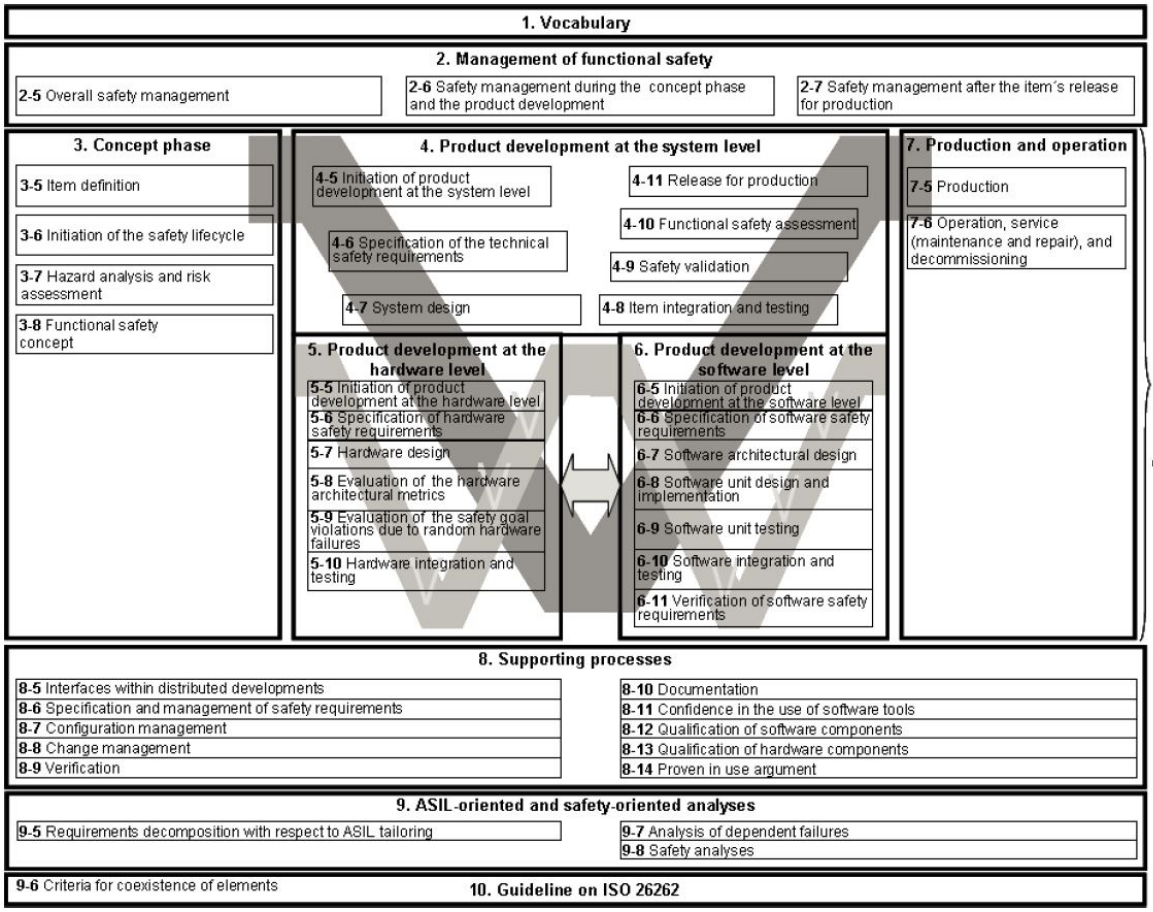


FIGURE 1.8 – Structure et répartition des parties de l'ISO 26262

1.3 Conclusion

Ce chapitre sur l'état de l'art et les travaux connexes a porté sur deux thématiques majeures : les systèmes embarqués et mécatroniques, et la SdF. Les notions et concepts, associés à ces deux domaines, servent de base à notre démarche.

Concernant les systèmes embarqués et mécatroniques, l'approche s'est orientée sur les cycles et méthodes de développement, afin de déterminer les pratiques menées dans le monde industriel. Pour la SdF, l'attention s'est portée sur les techniques d'analyse traditionnel et leur application sur du logiciel en phase avancée du développement ainsi que sur les méthodes d'analyse plus récentes comme le MBSA. Compte tenu de l'environnement industriel de la thèse, impliquée dans le monde des transports et plus particulièrement l'automobile, une présentation de la norme ISO 26262, visant à garantir la sécurité fonctionnelle des systèmes E/E dans les véhicules automobiles, vient compléter la perception du contexte.

L'étude de ces notions et de l'existant montre que, dans un processus de développement en prototypage, les développeurs se concentrent avant tout dans la réalisation des exigences fonctionnelles. La SdF n'est souvent abordée que dans un second temps. D'autre part, les analyses de SdF sont peu conduites sur du logiciel à un niveau de conception détaillée, elles sont plutôt pratiquées au stade de conception du système. Malgré l'apparition de nouveaux moyens de développement à base de modèle, on ne trouve que peu d'approche menant des analyses de SdF sur des modèles fonctionnels de logiciel. Pour celles existantes, en générale, elles ne prennent pas en compte toutes les caractéristiques d'un modèle logiciel. Elles se consacrent avant tout sur son aspect structurel mais un modèle logiciel en phase de conception détaillée décrit aussi un comportement et des performances temps réel. En ne portant que sur le critère structurel, les analyses de SdF menées ne sont pas assez précises, les résultats sont trop larges. En apportant plus d'informations aux analyses avec la prise en compte du comportement et de l'aspect temps réel, les résultats des analyses sont affinés.

Chapitre 2

Méthodologie ALEBAS

Sommaire

Les phases de prototypage et de développement de démonstrateurs sont principalement dédiées à la mise en œuvre des exigences fonctionnelles. Mais avec l'augmentation de la complexité des systèmes embarqués ainsi que leur implication dans des fonctions à caractère critique et les contraintes budgétaires, il est de plus en plus nécessaire de prendre en charge, dès les premiers prototypes, des exigences habituellement traitées plus tard, voire même seulement lors du processus de mise en production.

La figure 2.1 décrit le processus de développement, basé sur le MBD, du logiciel embarqué ainsi que les tâches de SdF associées à chaque étape du développement. Les méthodologies de développement actuelles intégrant la SdF réalisent principalement des analyses de SdF dans les phases de conception du système (fond quadrillées en orange dans la figure 2.1). C'est souvent à ce stade que l'on peut faire des choix techniques sur l'architecture du système pour assurer une meilleure fiabilité, comme par exemple la redondance d'un sous-système. Les méthodes de prototypage rapide conduisent à progresser rapidement dans le développement jusqu'à la phase d'implémentation du logiciel. Il existe peu de méthodologies pour réaliser des analyses de SdF à ce stade du développement (fond en pointillés bleus dans la figure 2.1) comme constaté dans l'état de l'art (chapitre 1). Or, le développement logiciel est dépendant du niveau d'expertise du développeur. Des erreurs peuvent être introduites malencontreusement dans le logiciel. C'est aussi à ce stade que l'on peut mettre en pratique des stratégies de SdF pour assurer une meilleure robustesse du logiciel et donc, par prolongement, du système. En analysant le modèle logiciel développé en phase de conception détaillée, il est possible de proposer des améliorations du logiciel pour corriger les fautes existantes et implémenter des solutions pour éliminer ou tolérer des fautes.

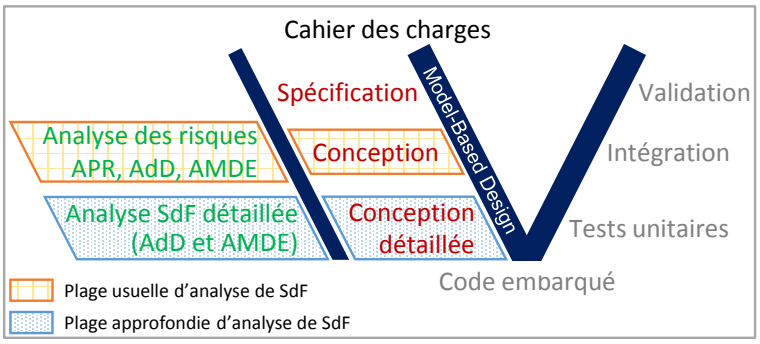


FIGURE 2.1 – Processus de développement intégrant la SdF pour une itération de prototype

Pour leur compétitivité, les PME sont très souvent impliquées dans le développement de nouveaux systèmes et par conséquent dans des phases de PoC et de prototypage. Les ressources des PME sont limitées et leur expertise ne couvre pas nécessairement la SdF. Il est donc difficile de leur imposer la pratique d'analyse de SdF à moins que les moyens de mise en œuvre soient relativement simples et flexibles pour s'adapter à leurs pratiques habituelles.

Compte tenu du contexte, il est nécessaire de proposer une méthodologie de développement en phase de prototypage qui favorise l'introduction de la SdF dans les logiciels embarqués. Les travaux menés sur ce sujet nous ont conduit à définir la méthodologie ALEBAS en se concentrant sur les critères suivants à remplir :

1. elle doit être relativement simple pour être exploitée par les développeurs qui ne sont pas nécessairement experts en SdF ;
2. elle doit être assez flexible pour s'adapter au processus de prototypage et aux différentes organisations et règles de développement appliquées dans les entreprises ;
3. elle doit pouvoir gérer les évolutions récurrentes du logiciel en phase de prototypage ;
4. elle doit être adaptée aux spécificités des modèles logiciels pour garantir leur bonne analyse ;
5. elle doit produire de la documentation contenant les résultats des analyses de SdF.

Dans ce chapitre, nous commencerons par présenter quelques bonnes pratiques à appliquer au cours du cycle de développement afin de rassembler les conditions optimales pour appliquer la méthodologie ALEBAS. Puis, l'organisation générale de la méthodologie ALEBAS ainsi que ses objectifs et son fonctionnement seront présentés. Ensuite, nous détaillerons chacune des étapes de la méthodologie et les moyens mis en place pour rendre les techniques utilisées adaptées à l'analyse de modèle logiciel embarqué, à l'étape de conception détaillée et en phase de prototypage.

2.1 Bonnes pratiques préalables

La méthodologie ALEBAS vise à améliorer les modèles de logiciel embarqué en phase de conception détaillée en se basant sur des recommandations obtenues à partir d'analyse de SdF. Avant d'atteindre le cœur de la méthodologie qui s'applique en phase de conception détaillée, quelques règles de bonnes pratiques sont définis afin d'établir plus largement la méthodologie de travail dans les phases amont de développement et de SdF. Cependant, pour répondre au besoin de flexibilité de la méthode, il n'est pas obligatoire de respecter ce processus et ces règles de développement ; nous ne pouvons pas les imposer aux utilisateurs car ceux-ci peuvent être propres aux pratiques des entreprises. Nos analyses de SdF peuvent tout de même être appliquées sur des modèles qui ne suivent pas exactement le cadre que nous proposons. Le processus de développement repose sur celui exposé précédemment en figure 2.1. Les règles proposées s'attachent notamment à la phase de conception et d'analyse des risques ainsi qu'au développement lors de la

conception détaillée. Pour illustrer les bonnes pratiques, lors de la conception détaillée, celles-ci sont d'abord présentées de façon générique puis elles sont inscrites dans l'environnement de développement de l'entreprise c'est-à-dire avec l'utilisation des outils Simulink et MotoHawk.

2.1.1 Spécification, conception et analyse des risques

Dans le cadre du prototypage, la spécification est évolutive à chaque itération de prototype. Les exigences changent en fonction des nouveaux besoins et afin de corriger ce qui n'a pas bien fonctionné avec le prototype précédent. Il est important de bien définir ou réajuster les exigences et les objectifs à atteindre pour cette nouvelle itération du prototype.

En phase de conception, les ingénieurs définissent l'architecture fonctionnelle du système ; elle se compose généralement de capteurs, de calculateurs et d'actionneurs. Les fonctions définies lors de la spécification sont réparties à chaque composant de l'architecture. La figure 2.2 décrit un exemple simple d'architecture système composée de deux capteurs, un calculateur et deux actionneurs. Concernant le calculateur à cette étape, l'objectif est de connaître le nombre et le type de chacune des entrées et sorties nécessaires pour respectivement l'acquisition des données et la commande des actionneurs.

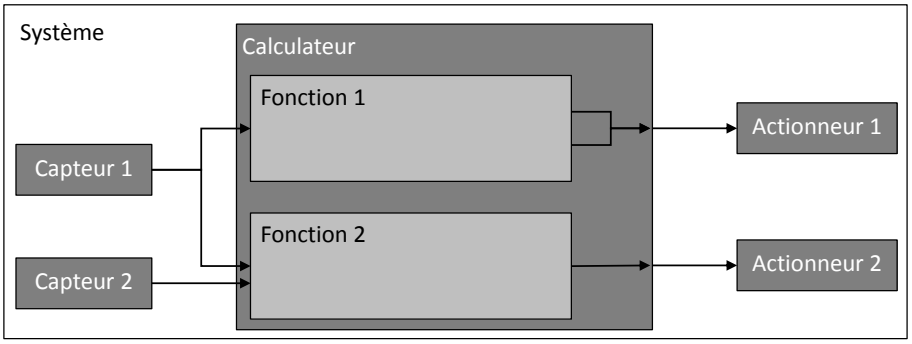


FIGURE 2.2 – Exemple de système simple

L'analyse des risques menée à ce stade a pour objectif de déterminer les Événements Redoutés du système (ERsys). Ces Événements Redoutés (ER) sont répartis en trois classes : l'absence ou la perte d'une fonction, le déclenchement intempestif d'une fonction et l'envoi de données erronées par la fonction. Comme le recommande les normes, les ER sont évalués pour qu'un niveau de criticité leur soit attribué. Dans l'automobile avec la norme ISO26262, ces niveaux sont appelés Automotive Safety Integrity Level (Automotive Safety Integrity Level (ASIL)). Les niveaux de criticité peuvent être déterminés par l'application de méthodes comme l'APR ou l'Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité (AMDEC). Cette évaluation permet d'identifier les ER les plus à risque et donc pour lesquels il faut être le plus vigilant et mettre en place des stratégies afin d'éviter

leur apparition. En connaissance des autres composants présents dans le système et leur relation, c'est-à-dire la topologie du système, il est possible en partant des ER, de remonter à leurs causes et notamment dans le cas des systèmes embarqués, à celles induites par le calculateur et son logiciel. Le logiciel n'étant pas encore développé, les causes potentielles de défaillances au niveau du calculateur se rapportent au dysfonctionnement de ses sorties.

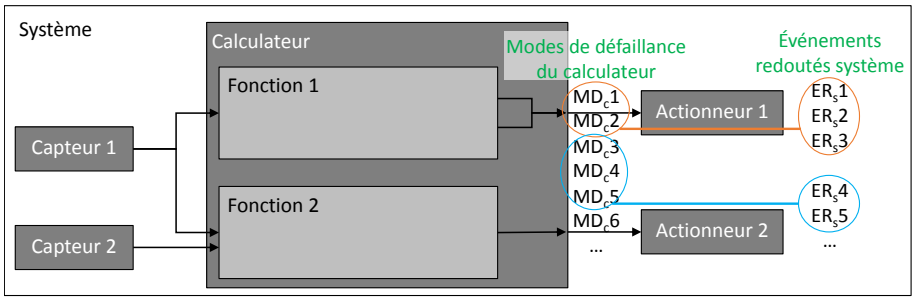


FIGURE 2.3 – Lien entre les événements redoutés du système et les modes de défaillance du calculateur

2.1.2 Conception détaillée / Analyse SdF détaillée

Le premier objectif à ce niveau est de construire l'architecture du modèle de logiciel embarqué. Pour concevoir cette architecture, on propose une structure type à employer à chaque développement logiciel. Ce type de modèle se structure en au moins deux niveaux [Bla07] comme décrit en figure 2.4 :

- Un niveau supervision qui assure la conduite globale du logiciel dans ces différents modes ou états, et si elle existe, l'Interface Homme-Machine (IHM).
- Un niveau contrôle-commande chargé de gérer les entités physiques c'est-à-dire les entrées/sorties de l'application pour qu'elles contribuent à satisfaire les objectifs fixés par le niveau de supervision.

Les deux niveaux sont couplés entre eux : dans le sens descendant pour l'assignation d'objectifs, dans le sens ascendant pour rendre compte de l'avancement vers les objectifs. La figure 2.5 expose l'application de cette structure du modèle logiciel à l'outil MotoHawk. Le niveau supérieur se compose d'une tâche de supervision qui rassemble toutes les informations pour connaître l'état du système et donner les ordres de passer aux états suivants. Si une IHM est présente, le niveau supérieur se compose d'une tâche « afficheObs » qui prépare les données pour les afficher correctement. Le niveau inférieur se compose d'une tâche de contrôle-commande qui contient l'acquisition des entrées, les algorithmes et les commandes à appliquer aux sorties. Le niveau inférieur contient aussi une tâche de diagnostic contrôlant les entrées et sorties. Dans le cas de la présence d'une IHM, une

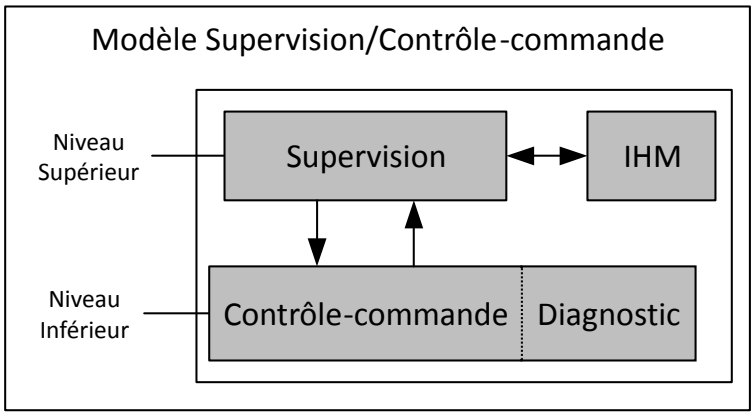


FIGURE 2.4 – Architecture logicielle pour un modèle de supervision/Contrôle-commande

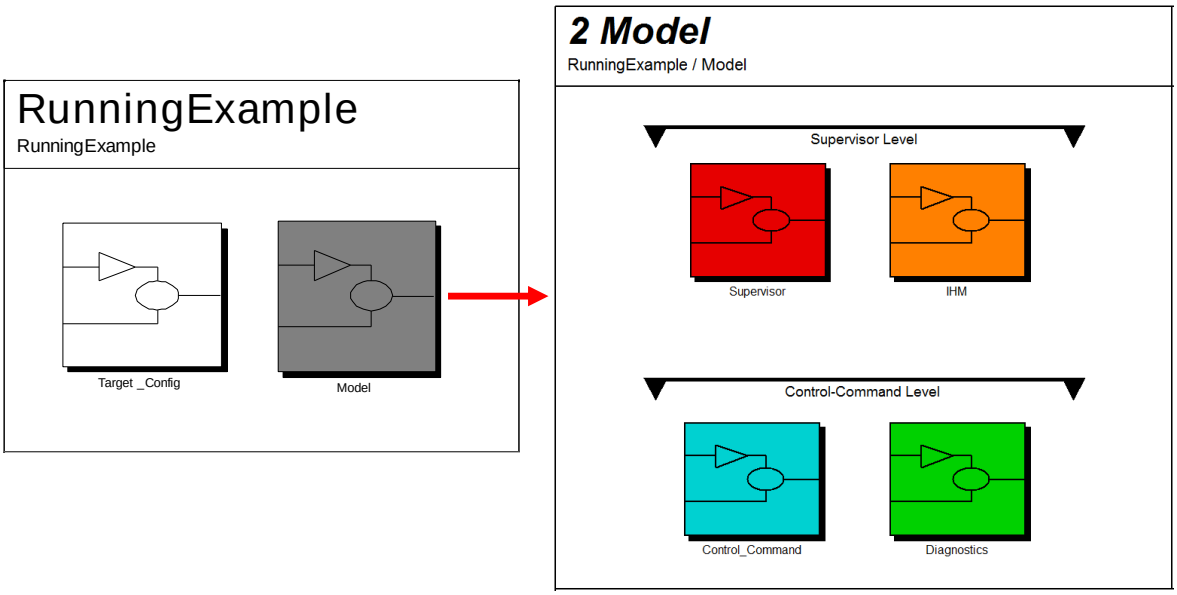


FIGURE 2.5 – Application de l'architecture logicielle à MotoHawk

tâche « collectObs » rassemble les données à transmettre à la tâche « afficheObs ». À ce stade, la communication entre les grandes fonctions constituant le modèle et l'environnement extérieur est assurée. Aussi, les contraintes de temps réel sont respectées avec la création des tâches et la configuration de leur périodicité d'exécution. En effet, chacune des fonctions principales, vues à ce stade comme des boites noires, sont reliées à une tâche.

En parallèle de la conception de l'architecture logicielle, l'étude de SdF se poursuit. Ici, le champ d'étude se réduit par rapport à l'analyse des risques au niveau système. On ne s'intéresse ici qu'aux risques pouvant être causés par le logiciel. Les causes potentielles d'occurrence d'un ER déterminées précédemment deviennent par projection des Événements Redoutés du logiciel (ERlog). Ainsi, des analyses des AdD et AMDE sont conduites sur les fonctions et sous fonctions qui viennent d'être créées en phase de conception pour déterminer comment elles peuvent causer l'apparition d'un ERlog. À ce stade, les

analyses de SdF reposent majoritairement sur l'étude de la structure du modèle logiciel. Après avoir défini les équations de dysfonctionnement des fonctions et sous-fonctions, le mécanisme de propagation est très efficace pour établir toutes les causes potentielles de chaque ERlog.

La phase de conception détaillée du logiciel embarqué se poursuit avec l'implémentation des stratégies de contrôle-commande. Il s'agit de programmer à l'aide de blocs élémentaires chacune des fonctions que le logiciel doit opérer. Des règles de développement sont spécifiées pour cadrer le travail du développeur. Ces règles reprennent ce qui se fait avec d'autres langages de programmation comme le code-C [Mot12], ou le MathWorks Automotive Advisory Board (MAAB) [Mat15] auxquelles s'ajoutent des conseils sur les blocs à privilégier et leur association avec d'autres blocs. De nouveau, cette étape est associée à une étude de SdF. Les analyses de SdF sont rarement effectuées jusqu'à ce niveau de détail, à cause de l'apparition de la dimension comportementale des blocs à ce stade. Mais avec l'établissement de normes comme l'ISO 26262, les analyses SdF doivent être conduites à chaque niveau de développement. Donc pour suivre cette consigne, les analyses sont conduites à la main ce qui impact le coût et qui est source d'erreur, d'autant plus lorsque des modifications sont nécessaires. Dans notre cas, une analyse SdF approfondie est menée sur le modèle réalisé en vue de l'enrichir avec les recommandations résultantes. Il est à noter que le modèle à cette étape est souvent très riche et peut contenir des milliers de blocs décrivant des comportements divers et variés : physique, électronique ou encore diagnostic. L'analyse SdF conduite doit gérer cette diversité, et être suffisamment flexible et automatisée pour suivre l'évolution constante du modèle. Pour cela, notre analyse SdF s'appuie sur des règles spécifiques prenant en compte les différents blocs utilisés avec les différents types de défaillances, les scénarios de dysfonctionnement et événements redoutés qui leurs sont associés. Ces règles sont dépendantes de chaque type de bloc élémentaire et de l'événement redouté étudié.

2.1.3 Exemple d'application

Cette section présente un exemple simple de logiciel embarqué développé à l'aide de Simulink et MotoHawk. Ce modèle logiciel servira d'exemple d'application de chacune des étapes de la méthodologie ALEBAS, au fur et à mesure de leur présentation dans la suite du document. Cela facilitera la compréhension et l'illustration de chacune des techniques utilisées. Le modèle logiciel exemple est présenté en figure 2.6. Le listing 2.1 décrit le code-C équivalent au modèle logiciel présenté en figure 2.6, ceci pour permettre aux personnes non-familières avec les modèles logiciels de comprendre l'algorithme.

Dans ce modèle, on retrouve d'abord le niveau supérieur qui est au sommet de l'arborescence du modèle. À ce niveau, les paramétrages de la cible matériel et de la compilation

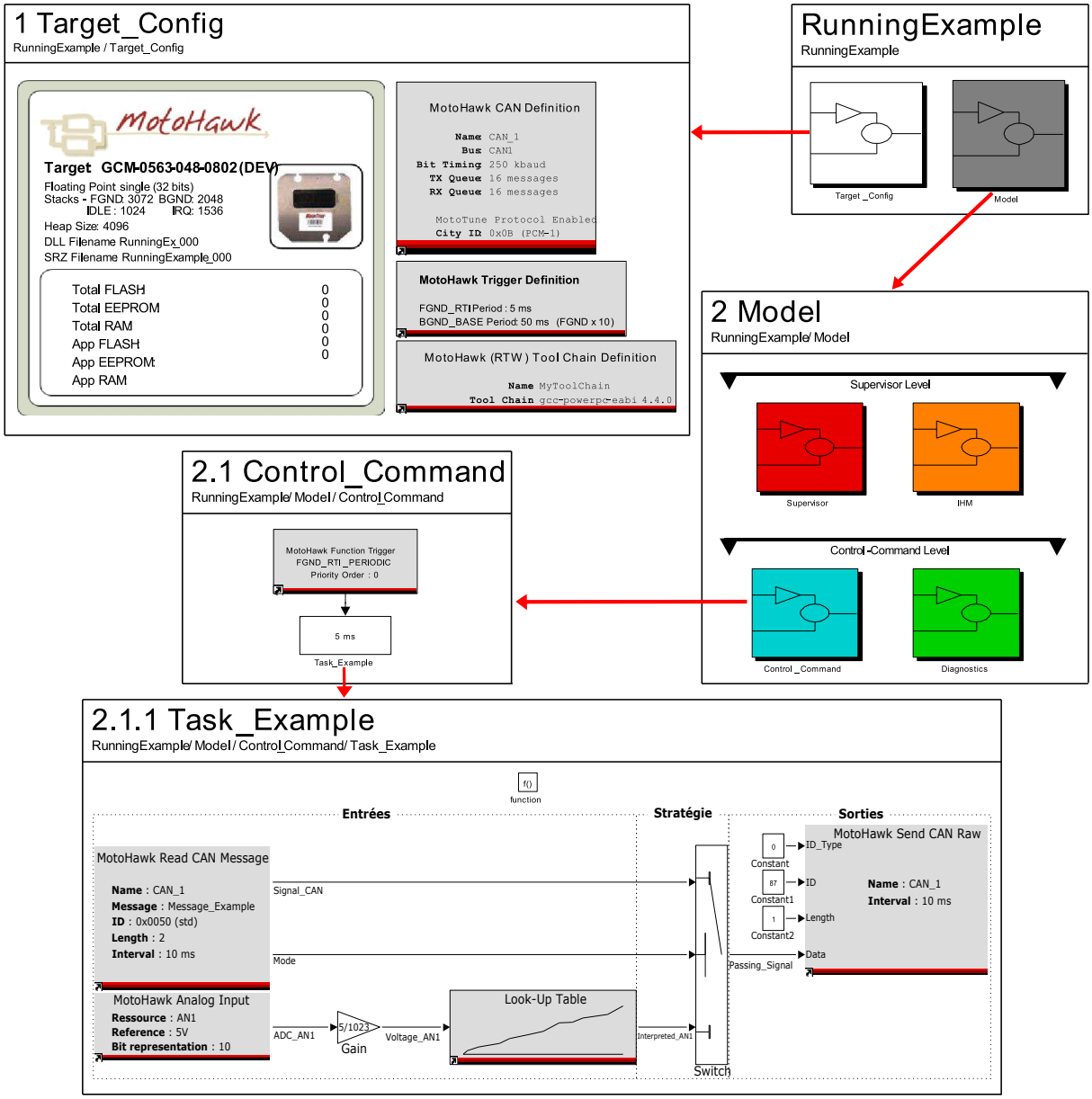


FIGURE 2.6 – Exemple simple

du code sont définis : choix du calculateur, paramètres du bus CAN, paramètres de l'horloge du système temps réel (Trigger), choix du compilateur et enfin la création des tâches qui contiennent les fonctions logicielles. Dans cet exemple, il y a une tâche déclenchée toutes les 5 millisecondes qui contient la fonction décrite ci-dessous. Pour les besoins de la présentation, la fonction n'a pas été découpée en sous-systèmes comme il aurait été nécessaire de le faire pour avoir une bonne architecture logicielle (un premier sous-système contenant les entrées, un autre l'algorithme et enfin un dernier avec les sorties) ; mais le choix a été fait de présenter un modèle simple et réduit pour cet exemple académique. Il s'agit d'une fonction composée de deux blocs d'entrées, six blocs intermédiaires et un bloc de sortie. Globalement, l'objectif de cette fonction est d'envoyer un message CAN

Listing 2.1 – Pseudo-code équivalent à la fonction se trouvant dans la tâche du modèle logiciel

```
/* Réception du message CAN "Message_Example"
contenant 2 signaux */
Signal_CAN = MotoHawkReadCANMessage(Message_Example.Signal);
Mode = MotoHawkReadCANMessage(Message_Example.Mode);

/* Acquisition du signal analogique */
ADC_AN1 = AnalogInput(AN1M.ADC);

/* Interprétation du signal analogique en volt
puis en donnée physique */
Voltage_AN1 = ADC_AN1 * Gain ;
Interpreted_AN1 = LookUpTable(Voltage_AN1) ;

/* Aiguillage du signal de sortie par le switch */
Switch (Mode)
{
case true: Passing_Signal = Signal_CAN
case false: Passing_Signal = Interpreted_AN1
}

/* Paramétrage du message CAN à envoyer */
MotoHawkSendCANRaw(ID_Type) = 1 ;
MotoHawkSendCANRaw(ID) = 87 ;
MotoHawkSendCANRaw(Length) = 1 ;

/* Insertion du signal sélectionné par le switch
dans le message CAN */
MotoHawkSendCANRaw(Byte) = Passing_Signal ;
```

contenant soit un premier signal reçu via un autre message CAN, soit un deuxième signal reçu via une entrée analogique. Ce choix est décidé selon le contenu d'un troisième signal reçu via le message CAN qui commande un « switch ». Ce « switch » gère l'aiguillage pour avoir en sortie soit le premier signal soit le deuxième.

Le premier bloc d'entrée est la réception d'un message CAN contenant deux signaux. Le premier signal reçu par CAN contient une donnée envoyée par un autre module du système dans lequel se trouve le calculateur. Le deuxième signal CAN est une commande qui contrôle l'aiguillage du bloc « switch » : soit l'entrée 1 soit l'entrée 3 du « switch » est passante. Le deuxième bloc d'entrée est l'acquisition d'un signal analogique via une entrée physique (pin) de l'calculateur –ou *Electronic Control Unit*– (ECU). Ce signal est reçu brut du Convertisseur Analogique/Numérique –ou *Analog/Digital Converter*– (ADC), il est nécessaire d'y appliquer quelques calculs, ici un gain et une table d'interpolation (Look-

up Table - LuT) afin d'obtenir une valeur d'ingénierie. Enfin, en fonction de la commande du bloc « switch » l'un des deux signaux d'entrée est aiguillé vers la sortie pour être envoyé via un message CAN sur un autre bus.

2.2 Architecture de la méthodologie ALEBAS

Cette section présente la méthodologie ALEBAS dans sa globalité et la manière dont elle répond aux contraintes exposées précédemment. L'intention générale de la méthodologie est de mener une analyse de SdF sur un modèle logiciel, afin d'en déduire les recommandations à implémenter pour rendre le logiciel plus sûr. L'organisation de la méthodologie proposée est illustrée en figure 2.7. Tout d'abord, ALEBAS requiert en entrée le modèle logiciel à étudier et à améliorer. Le modèle logiciel étudié a atteint la fin de la phase de conception détaillée. À ce stade, le modèle logiciel représente la stratégie du logiciel embarqué à l'aide de composants basiques qui sont organisés de façon hiérarchique, pour composer les fonctions et sous-fonctions requises pour répondre aux exigences fonctionnelles. Dans notre cas, le modèle logiciel est développé avec l'interface Simulink à l'aide de la librairie native et d'une librairie complémentaire MotoHawk. Cette dernière est dédiée à la programmation d'une gamme de calculateurs automobiles. Elle facilite l'interfaçage entre les I/O du logiciel et du matériel. La suite de la méthodologie peut être reproduite sur d'autres outils dédiés à la modélisation de logiciels embarqués, à partir du moment où ils sont basés sur les diagrammes de flux de données.

Ensuite, dans notre méthodologie débute la phase d'analyse de SdF qui est conduite sur le modèle logiciel. L'analyse de SdF proposée s'appuie sur deux techniques généralement combinées pour leur complémentarité : les AdD ❷ et l'AMDE ❸. L'utilisation de ces deux méthodes est recommandée pour l'analyse de logiciel pouvant induire des hasards dans le système [Nat04]. L'application des AdD sur le logiciel identifie les multiples combinaisons d'erreurs logicielles responsables de l'occurrence d'un événement redouté étudié. L'AMDE logicielle examine les effets causés par l'apparition d'une défaillance dans le logiciel sur le système. Elles sont très répandues dans le monde de l'ingénierie et souvent enseignées dans les écoles ; leur approche par des ingénieurs non-experts en SdF est par conséquent facilitée. L'utilisation de ces techniques sur un modèle logiciel issu de la phase de conception détaillée présente quelques difficultés notamment :

- la gestion de la grande quantité de données obtenues par les analyses ;
- les causes potentielles identifiées dans les AdD doivent concerner les événements redoutés étudiés ;
- le tableau de l'AMDE doit contenir des données précises et facilement exploitables.

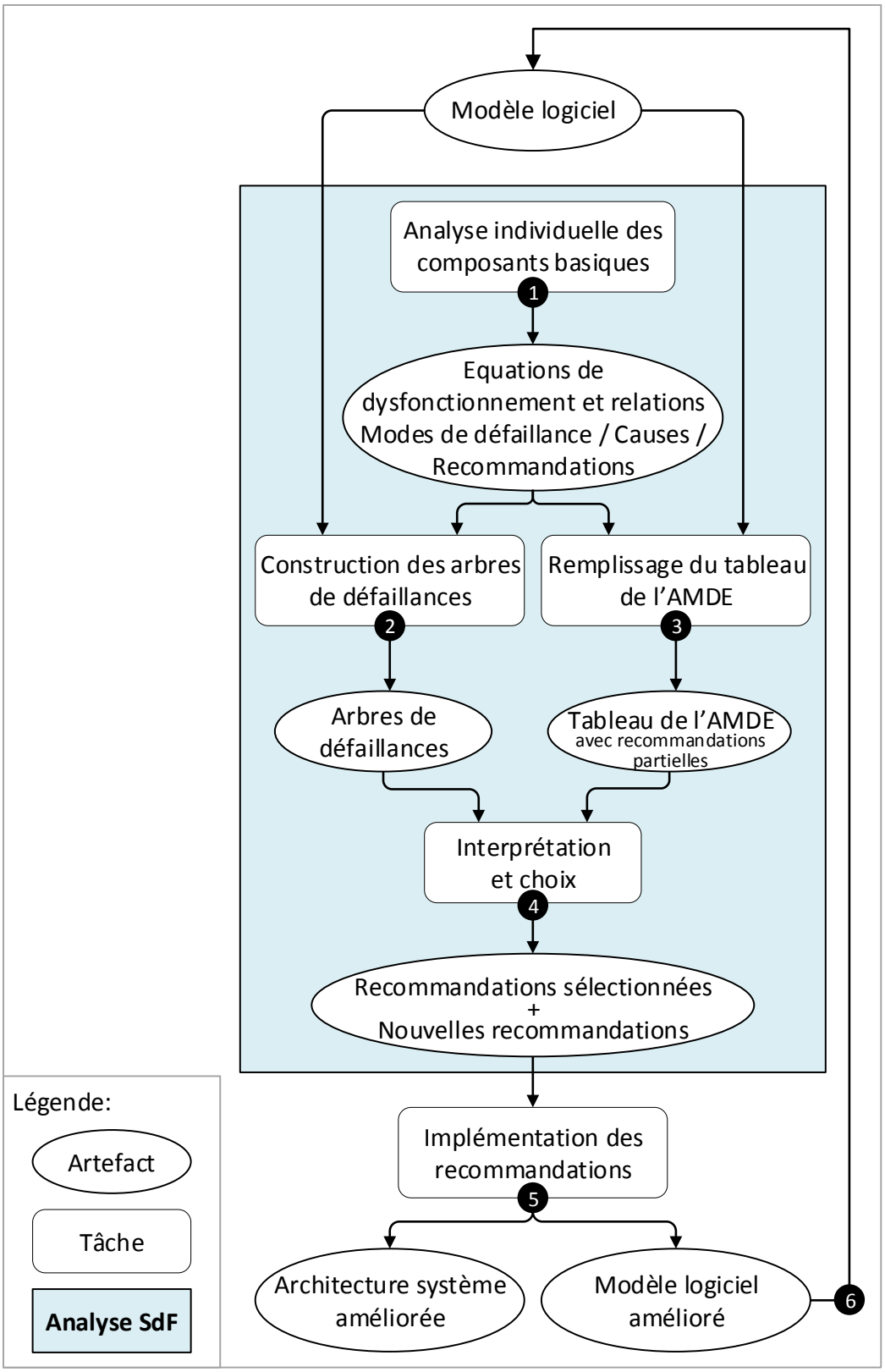


FIGURE 2.7 – Organisation de la méthodologie ALEBAS

Une des réponses à ces difficultés consiste à supporter les deux techniques AdD et AMDE par l'utilisation des relations « Modes de défaillance / Causes » des blocs basiques du modèle pour mener au mieux les processus de propagation inductive et déductive. Ces relations de « Modes de défaillance / Causes » ont été déterminées par des analyses individuelles de chacun des blocs basiques ❶. Ces analyses individuelles sont indépendantes du modèle en cours d'étude, elles ont été réalisées en amont et sont réutilisées à chaque étude. Le détail des moyens mis en place pour répondre à ces problématiques est exposé dans les présentations de chacune des techniques dans les sections dédiées suivantes.

L'ordre d'utilisation des deux techniques n'est pas imposé pour garantir la flexibilité de la méthodologie. Le choix est laissé à l'ingénieur en fonction de ce qu'il cherche à étudier. Néanmoins, pour une première itération de l'analyse de SdF, il est préférable de commencer par les AdD afin d'avoir une vision globale des éventuelles propagation d'erreur dans le modèle et les zones les plus sensibles. Il est ensuite plus facile de cibler les parties de l'AMDE à étudier en premier en fonction de ces zones sensibles avant d'élargir l'étude. Le résultat de l'analyse est l'obtention de recommandations, à partir de l'interprétation des deux méthodes, afin d'améliorer le modèle logiciel prototype et dans certains cas, notamment lorsque la partie de l'étude concerne les I/O du logiciel, les recommandations peuvent également porter sur l'amélioration de la configuration matérielle ou architecture système. Les recommandations sont déduites par une étude des AdD obtenus et le tableau de l'AMDE. Lors de l'analyse individuelle, des corrections sont proposées pour éliminer les causes ou tolérer les effets déterminés. Ces recommandations se retrouvent dans le tableau de l'AMDE et font l'objet de recommandations pour l'amélioration du logiciel.

Pendant le prototypage, le modèle logiciel est développé de manière itérative. Alors l'analyse de SdF est menée en parallèle de l'avancement du développement. Ainsi, la stratégie de SdF est aussi implémentée étape par étape en accord avec le processus de développement et les recommandations proposées. Les recommandations deviennent des exigences pour les versions suivantes du modèle logiciel. Le développeur est libre d'appliquer les recommandations qu'il souhaite selon son expertise du système. ALEBAS peut être appliquée de manière itérative afin d'étudier le nouveau modèle amélioré.

2.3 Analyse individuelle ❶

Les modèles logiciels peuvent devenir très larges, c'est-à-dire qu'ils sont composés de très nombreux blocs, pour réaliser l'algorithme du logiciel embarqué souhaité. Par conséquent, les AdD générés à partir de ces modèles subissent les mêmes effets de prise d'ampleur car pour un bloc logiciel, on retrouve plusieurs portes ou événements dans les AdD. Les AdD sont classiquement construits à partir de la structure du modèle et de l'or-

ganisation des blocs entre eux c'est-à-dire des relations de proche en proche existantes entre les blocs. Le comportement des blocs n'est pas pris en compte dans cette démarche. L'analyse individuelle de chacun des blocs vise à déterminer leurs modes de défaillance propres, leurs causes potentielles et les actions recommandées afin de les éviter ou de les tolérer. Ces caractéristiques sont dépendantes entre elles de la manière suivante : une faute est à l'origine de l'occurrence d'un mode de défaillance et pour tolérer ou éliminer une faute, cela nécessite l'application de recommandations spécifiques.

2.3.1 Démarche de l'analyse individuelle

L'analyse individuelle d'un composant basique commence avec l'identification des types de modes de défaillance auxquels il est susceptible d'être affecté. La classification de ces différents types de déviation de comportement est typiquement assistée par l'utilisation de mots clés qui sont similaires à ceux utilisés dans la méthode Hazop [Pum99] ou comme présentés dans [BS90]. Ces mots clés proposent un classement des défaillances selon plusieurs catégories [FMNP94] :

- l'omission qui fait référence à la défaillance d'un composant pour fournir sa sortie ;
- la commission qui fait référence à la provision inattendue d'une sortie ;
- la valeur qui fait référence à une donnée de sortie erronée ;
- le temps qui fait référence à une sortie qui serait fournie en avance ou en retard par rapport à ce qui est prévu.

Comme l'analyse est menée sur des composants basiques dont leur comportement est bien connu, il est possible d'affiner ou préciser la description des défaillances et plus précisément la valeur des défaillances. En effet, dans [Law96], un ensemble de phrases guides sont fournies selon la phase de développement. Certaines d'entre elles peuvent être utilisées pendant la phase de conception détaillée.

Lorsque les modes de défaillance sont définis, les causes de leur apparition peuvent être établies. Dans [ALRL04], les fautes sont catégorisées selon huit points de vue basiques comme décrit en figure 2.8 .

Pour un logiciel embarqué, la catégorie la plus adaptée pour classer les causes est « dimension » car les causes relèvent :

- des erreurs logicielles qui concernent le paramétrage des blocs, le choix du bloc ou son intégration dans le modèle ;
- des dysfonctionnements du matériel car le matériel et le logiciel sont très liés dans le cadre des systèmes embarqués.

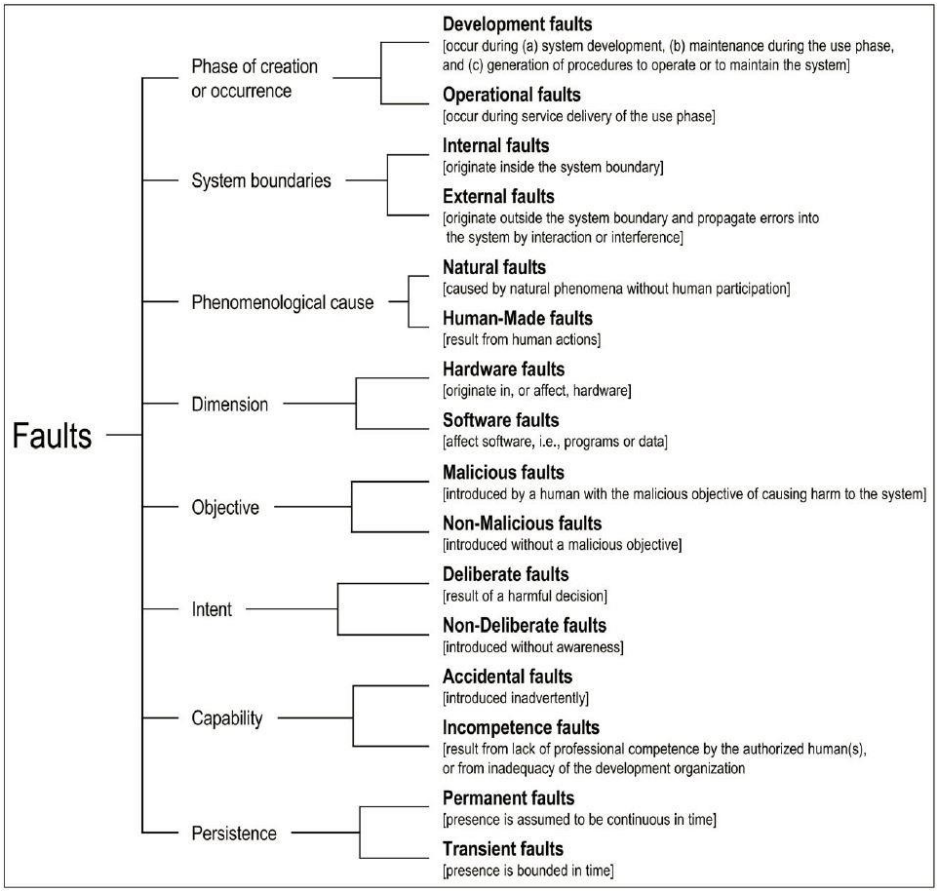


FIGURE 2.8 – Catégorisation des fautes élémentaires [ALRL04]

L'identification des causes repose sur l'étude du comportement des blocs, des tests pour produire les modes de défaillance et plus particulièrement sur l'expérience des analystes et des développeurs. Une fois les causes identifiées, les moyens pour les tolérer ou les éliminer doivent être déterminés. Dans [ALRL04], les moyens sont distingués en quatre catégories : la prévention, la prévision, la tolérance et l'élimination des fautes. Dans le cadre de notre approche, le logiciel est déjà implémenté, cela signifie que les fautes sont déjà introduites donc la prévention n'est pas appropriée. La prévision des fautes rentre dans notre méthodologie avec l'utilisation de l'AMDE. La tolérance et l'élimination des fautes sont les mieux adaptées pour gérer les causes décrites précédemment parce qu'elles peuvent être appliquées sur le logiciel existant et pendant le fonctionnement. Les solutions proposées pour tolérer ou éliminer une faute doivent être suffisamment détaillées et expliquées pour faciliter son application par le développeur : ce sont les actions recommandées. Il peut y avoir plusieurs recommandations pour une cause. Par exemple, il est possible de proposer :

- une stratégie de détection d'erreur qui sauvegarde les occurrences pour les indiquer pendant la maintenance du système ;

- une stratégie de détection d'erreur qui alerte directement l'utilisateur (par un voyant ou un signal sonore). La fonction reste défaillante mais l'utilisateur est prévenu ;
- l'intégration d'une redondance associée à une stratégie logicielle pour mettre le système en mode dégradé. Dans ce cas, la fonction n'offre pas toutes ces capacités mais reste partiellement en opération.

Le choix de la recommandation à appliquer dépend de trois caractéristiques : la continuité de service (disponibilité), la complexité et le coût. Ces trois caractéristiques sont liées dans la mesure où si l'on cherche à augmenter la continuité de service, la complexité et le coût vont également croitre. Pour notre approche, nous avons privilégié le critère de continuité de service dans la mesure où nous nous intéressons à des fonctions à caractère critique. Nous pouvons noter qu'il existe des méthodes d'optimisation qui permettent de trouver le meilleur compromis entre les trois critères comme dans [Les17]. Le niveau de continuité de service à tenir est dépendant du niveau de risque de l'événement redouté qui peut potentiellement se produire. Ainsi, pendant l'application de notre AMDE, si l'analyse de SdF a été préalablement réalisée au niveau de l'architecture système, les actions recommandées sont proposées par la corrélation entre le niveau de risque de l'événement sur le système et la continuité de service comme présentée dans le tableau 2.1. En d'autres termes, plus le niveau de risque potentiel de l'événement redouté étudié est important, plus la continuité de service doit être importante. Cette corrélation peut être automatisée mais, bien souvent, il est nécessaire de le faire valider par l'analyste et le développeur.

TABLEAU 2.1 – Description de la continuité de service à appliquer en fonction du niveau de risque

Niveau de risque	**Description de la continuité de service à établir**
4 (ASIL D)	Performances Complètes (PC) : La fonction et les performances sont assurées.
3 (ASIL C)	Mode Dégradé (MD) : La fonction et les performances sont partiellement assurées.
2 (ASIL B)	Alerte (A) : La fonction n'est pas assurée mais l'utilisateur est prévenu.
1 (ASIL A)	Détection (D) : La fonction n'est pas assurée et l'utilisateur n'est pas prévenu mais la défaillance est enregistrée et pourra être relevée et corrigée lors de la maintenance.
0 (QM)	Vérification (V) et/ou Règle de Développement (RD) : La fonction ne nécessite pas de surveillance. En cas de défaillance, elle n'est ni assurée ni détectée volontairement.

La dernière étude à mener pour l'analyse individuelle d'un bloc est son comportement vis-à-vis de l'acquisition en entrée du bloc d'une valeur défaillante. La valeur reçue a été affectée d'une défaillance par un bloc en amont et on souhaite connaître comment

cette défaillance se propage lorsqu'elle atteint le bloc en cours d'étude. Ce travail peut être mené grâce à la connaissance précise, par un expert, des blocs étudiés et il peut aussi être conduit par la réalisation de tests en soumettant volontairement le bloc que l'on souhaite étudié aux différents types d'erreur et d'observer son comportement en sortie.

Pour résumer, l'analyse individuelle d'un bloc se déroule en trois parties :

- l'étude des types de modes de défaillance propres au bloc étudié ;
- l'étude des relations « Modes de défaillance / Causes / Recommandations » ;
- l'étude de la propagation des événements redoutés.

La réalisation des analyses individuelles des blocs reposent sur l'expérience de l'analyste. Il est recommandé de mener ces analyses conjointement avec des experts en développement logiciel et en électronique embarqué.

2.3.2 Analyse individuelle de blocs Simulink/MotoHawk

La présentation de l'analyse individuelle des blocs est conduite par l'utilisation de quelques blocs présents dans le modèle exemple. Les blocs étudiés ont été choisis de manière à montrer la majorité des cas et résultats de l'analyse. Les blocs étudiés se composent d'un bloc déclencheur de tâche (Trigger), un bloc d'entrée avec l'acquisition d'un signal analogique, deux blocs intermédiaires avec le « Gain » et le « Switch » puis un bloc de sortie avec l'envoi d'une trame CAN. Les sous-sections suivantes présentent chacune un extrait de l'analyse individuelle pour chaque composant sélectionné.

2.3.2.1 Entrée analogique (*MotoHawk Analog Input*)

Ce bloc permet l'interfaçage entre le logiciel et le matériel. Ce bloc donne accès à la valeur de tension mesurée par le convertisseur analogique/numérique de la ressource sélectionnée (broche du calculateur). La valeur mesurée représente le ratio du niveau de tension mesurée sur la broche du calculateur par rapport à la référence (par défaut 5V), elle est codée sur la résolution en nombre de bits disponibles (par défaut 10 bits). La figure 2.9 expose le bloc d'entrée analogique.

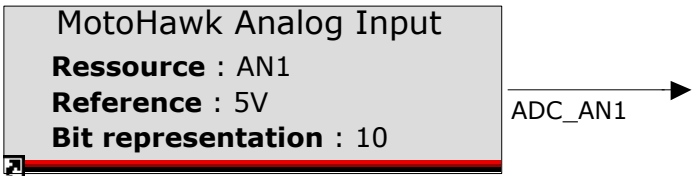


FIGURE 2.9 – Bloc d'entrée analogique (*Analog Input*) issu de la librairie MotoHawk

Étude des modes de défaillance potentiels de l'entrée analogique

- Omission : l'entrée analogique n'est pas concernée par ce mode de défaillance car à partir du moment où le bloc est dans le modèle, il fournit au moins en sortie sa valeur initiale « 0 » ;
- Commission : le déclenchement de l'échantillonnage des données n'est pas géré par ce bloc, il dépend des blocs « trigger » de tâche ou de blocs conditionnés dans les sous-systèmes parents ;
- Erroné : ce bloc est susceptible de fournir des valeurs erronées. L'erreur peut être subtile c'est-à-dire qu'elle appartient à l'intervalle attendu mais n'est pas correcte. Elle peut aussi être grossière c'est-à-dire que la valeur fournie est en dehors de l'intervalle attendu ou le comportement est complètement différent de celui en opération normale ;
- Temps : la notion de temps réel n'est pas gérée par ce bloc mais par des blocs parents comme les blocs « trigger » de tâche.

Étude de propagation

- Si le signal matériel reçu via le calculateur est correct alors la sortie du bloc logiciel ne peut être défaillante qu'en fonction de ses modes de défaillance propres ;
- Si le signal matériel est défaillant alors le bloc logiciel se comporte de façon linéaire, on retrouve la même défaillance en sortie.

Étude des relations « Modes de défaillance / Causes / Recommandations »

Les relations de « Modes de défaillance / Causes / Recommandations » pour le bloc d'entrée analogique sont rassemblées dans l'annexe A.1. Une partie de cette étude est présentée ci-dessous, elle concerne l'étude du mode de défaillance de type « erreur grossière » et plus précisément lorsque le signal analogique reçu a un comportement erratique. On note que s'agissant d'un bloc d'interfaçage entre le matériel et le logiciel, les causes de défaillances sont surtout susceptibles d'être déclenchées par un problème matériel. D'une manière générale, la principale difficulté est de détecter la défaillance et surtout de distinguer les différentes défaillances potentielles. La détection est parfois facilitée par des moyens de diagnostic existants comme la surveillance des circuits d'acquisition des entrées du calculateur qui permet de vérifier pour chaque entrée son statut. Mais lorsque l'erreur vient de l'extérieur, il est nécessaire d'implémenter des stratégies plus complexes soit en se basant sur un seul signal d'entrée, en étudiant son comportement et en le comparant à un comportement attendu (par exemple étude de la dérivée), soit il est parfois nécessaire d'avoir des redondances de signaux d'entrée pour effectuer les comparaisons. Dans la partie de l'étude individuelle ci-dessous, des exemples de recommandations sont présentées pour la cause de type matériel « défaillance du signal reçu ».

Mode de défaillance : *Erreur grossière* - Comportement erratique/aléatoire du signal

Cause : Logiciel - Mauvais réglage du paramètre « Data Type »

Cause : Matériel - Défaillance de l'entrée matérielle du calculateur

Cause : Matériel - Défaillance du signal reçu (ex : panne capteur, perturbations électromagnétiques...)

Recommandations

V : Contrôle du bon fonctionnement du signal lors des phases de tests et vérifications au cours du processus de développement.

D : V + Implémentation d'une stratégie logicielle pour détecter le comportement erratique par rapport au fonctionnement normal (ex : utilisation de la dérivée). Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance.

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation).

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance).

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction.

2.3.2.2 Gain

Ce bloc est un opérateur mathématique qui multiplie la donnée d'entrée du bloc par la valeur du gain : $Sortie = Entrée \times Gain$

FIGURE 2.10 – Bloc Gain de la librairie Simulink

Étude des modes de défaillance potentiels du bloc gain

- Omission : le gain n'est pas concerné par ce type de mode de défaillance car à partir du moment où le bloc est dans le modèle, il effectuera forcément un calcul ou donnera une valeur par défaut ;
- Commission : le déclenchement de l'exécution de ce bloc ne dépend pas de lui-même mais d'autres blocs comme les « trigger » de tâche ou de blocs conditionnés qui se trouvent dans les sous-systèmes parents ;
- Erroné : ce bloc est susceptible de fournir des valeurs erronées. L'erreur peut être subtile c'est-à-dire qu'elle appartient à l'intervalle attendu mais n'est pas correcte.

Elle peut aussi être grossière c'est-à-dire que la valeur fournie est en dehors de l'intervalle attendu ou le comportement est complètement différent de celui en opération normale ;

- Temps : la notion de temps réel n'est pas gérée par ce bloc mais par des blocs parents comme les blocs « trigger » de tâche.

Étude de propagation

- Si la donnée d'entrée du bloc est correcte alors sa sortie ne peut être défaillante qu'en fonction de ses modes de défaillance propres ;
- Si le signal d'entrée est défaillant alors le bloc logiciel se comporte de façon linéaire, on retrouve le même type de défaillance en sortie.

Étude des relations « Modes de défaillance / Causes / Recommandations »

Les relations de « Modes de défaillance / Causes / Recommandations » pour le bloc Gain sont rassemblées dans l'annexe A.2.

Pour tous les blocs intermédiaires, c'est-à-dire les blocs qui se trouvent entre les entrées et les sorties d'interfaçage entre le matériel et le logiciel, leurs modes de défaillance propres vont principalement être causés par des erreurs logicielles. En ce qui concerne le matériel, on retrouve simplement les modules de mémoire RAM et/ou ROM qui peuvent affecter les calculs en cas de défaillance. Pour les causes logicielles, des recommandations peuvent suggérer des règles de développement comme l'association du bloc en question avec d'autres ou son remplacement. L'extrait de l'étude individuelle du bloc « Gain » ci-dessous s'attache au mode de défaillance de type erreur grossière qui implique un comportement erratique de la sortie du bloc Gain. La cause logicielle « mauvais choix du paramètre datatype » est présentée avec ces recommandations pour améliorer le logiciel.

Mode de défaillance : *Erreur grossière* - comportement erratique de la sortie
Cause : Matériel - Défaillance de la mémoire
Cause : Logiciel - Mauvais choix du paramètre « datatype »
Recommandations
RD : Le bloc « gain » peut poser des problèmes lors de la compilation du code. Préférer l'utilisation du bloc « Product » pour opérer des multiplications. Assurer que le résultat soit dans un intervalle donné par l'utilisation d'un bloc « saturation » après le gain.

V : Contrôler le choix du paramètre « datatype ».
D : V + Contrôler le résultat du calcul vis-à-vis d'un intervalle attendu s'il est connu.
A : V + D + Alerte du conducteur par un système visuel ou sonore (nécessite un système de signalisation).
MD : Les stratégies de MD ou PC seront mises en place autour des blocs de sortie impac-
PC : tés par cette erreur.

2.3.2.3 Switch

Ce bloc permet d'aiguiller en sortie, soit le signal de l'entrée 1, soit le signal de l'entrée 3 en fonction du statut (vrai ou faux) de la condition arrivant dans l'entrée 2.

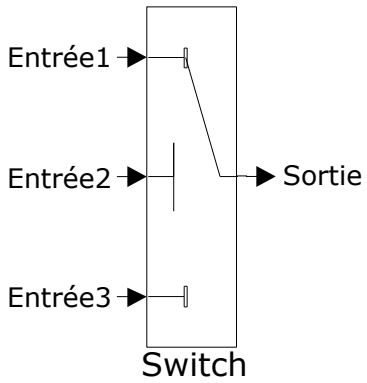


FIGURE 2.11 – Bloc Switch de la librairie Simulink

Listing 2.2 – Pseudo-code équivalent au bloc Switch

```
Switch (Entrée2)
{
case true: Sortie = Entrée1
case false: Sortie = Entrée2
}
```

Étude des modes de défaillance potentiels du bloc « switch »

- Omission : Si la condition est erronée alors une sortie sera omise et par complémentarité l'autre sera délivrée de façon inattendue.
- Commission : Si la condition est erronée alors une sortie sera délivrée de façon inattendue et par complémentarité l'autre sera omise.
- Erroné : certains paramètres du bloc peuvent affecter le signal passant.
- Temps : la notion de temps réel n'est pas gérée par ce bloc mais par des blocs parents comme les blocs « trigger » de tâche.

Étude de propagation

- Si la donnée d'entrée du bloc est correcte, alors sa sortie ne peut être défaillante qu'en fonction de ses modes de défaillance propres vus précédemment ;
- En revanche, si le signal d'entrée est défaillant alors le bloc logiciel se comporte de façon linéaire, on retrouve le même type de défaillance en sortie.

Étude des relations « Modes de défaillance / Causes / Recommandations »

Les relations de « Modes de défaillance / Causes / Recommandations » du bloc Switch sont rassemblées dans l'annexe A.3. Certains bloc intermédiaires comme le « switch » ou les « sous-systèmes enabled/trigger » n'opèrent pas de calcul mais ils participent à l'aiguillage de signaux ou au déclenchement de certaines parties du logiciel. Ce type de bloc peut alors mener à l'apparition de modes de défaillance de type Omission/ Commission. L'apparition de ces modes propres au bloc sont dus à des erreurs de paramétrage.

Mode de défaillance : *Omission/Commision* - comportement erratique du choix de l'entrée passante vers la sortie

Cause : Logiciel - Mauvais choix de la condition

Recommandations

RD : L'entrée 2 doit contenir un signal booléen et la condition doit être réglée sur u2~=0 (consigne définie par le MAAB [Mat15])

Mode de défaillance : *Omission/Commision* - Le signal de sortie est bloqué sur l'une des deux entrées

Cause : Logiciel - Mauvais choix de la condition

Mode de défaillance : *Erreur grossière* - comportement erratique de la valeur de sortie

Cause : Logiciel - Mauvais réglage du paramètre datatype

Mode de défaillance : *Erreur subtile* - signal de sortie légèrement erroné (ex : un arrondi de la valeur du signal)

Cause : Logiciel - Mauvais réglage du paramètre « datatype »

2.3.2.4 Envoi d'une trame CAN

Étude des modes de défaillance potentiels du bloc d'envoi d'une trame CAN

- Omission : l'envoi d'une trame CAN n'est pas concerné par ce mode de défaillance car à partir du moment où le bloc est dans le modèle, il effectuera forcément un calcul ;
- Commission : le déclenchement de l'échantillonnage des données n'est pas géré par ce bloc, il dépend des blocs « trigger » de tâche ou de blocs conditionnés dans les sous-systèmes parents ;

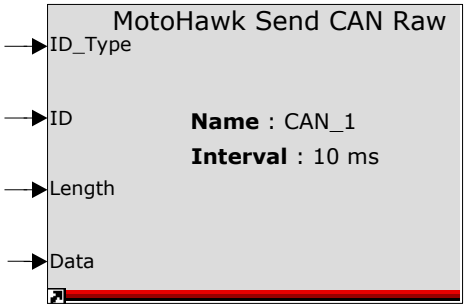


FIGURE 2.12 – Bloc d'envoi d'une trame CAN (*Send CAN Raw*) de la librairie MotoHawk

- Erroné : ce bloc est susceptible de fournir des valeurs erronées. L'erreur peut être subtile c'est-à-dire qu'elle appartient à l'intervalle attendu mais n'est pas correcte. Elle peut aussi être grossière c'est-à-dire que la valeur fournie est en dehors de l'intervalle attendu ou le comportement est complètement différent de celui en opération normale ;

- Temps : les blocs d'envoi de trame CAN ont la particularité d'avoir des paramétrages de leur périodicité d'envoi qui peut prendre le dessus sur les blocs « trigger » qui se trouveraient en amont de la hiérarchie du modèle. En cas de mauvais réglage, les messages peuvent arriver en avance ou en retard.

Étude de propagation

- Si la donnée d'entrée du bloc est correcte alors sa sortie ne peut être défaillante qu'en fonction de ses modes de défaillance propres vus précédemment.

- En revanche si le signal matériel est défaillant alors le bloc logiciel se comporte de façon linéaire, on retrouve la même défaillance en sortie.

Étude des relations « Modes de défaillance / Causes / Recommandations »

Les relations de « Modes de défaillance / Causes / Recommandations » du bloc d'envoi brut d'une trame CAN sont rassemblées dans l'annexe A.4. Les blocs d'interfaçage de sortie sont très similaires aux blocs d'entrée, leurs modes de défaillance potentiels dépendent en grande partie de causes matérielles qu'il faut être capable d'identifier pour appliquer les meilleures corrections possibles.

Mode de défaillance : *Erreur grossière* - message non-envoyé
Mode de défaillance : *Erreur grossière* - comportement erratique du message envoyé (mauvais ID, données erronées)
Cause : Matériel - Défaillance de la sortie (chip) CAN
Recommandations

V : Contrôle du bon fonctionnement de l'envoi du message lors des phases de tests et vérifications au cours du processus de développement.

D : V + Implémentation d'une stratégie logicielle pour détecter un problème sur la sortie CAN. Utiliser le bloc « CAN_Fault_status » pour connaître l'état du bus CAN. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance. Si le message porte des données à haute importance, ajouter un calcul de « checksum » à envoyer dans le message qui sera contrôlé par le/les module(s) à la réception et si besoin un signal de synchronisation qui s'incrément de 1 en 1 jusqu'à 15 puis redémarre. Cette synchronisation sera également contrôlée par le/les module(s) à la réception.

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation).

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance).

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction.

Cause : Matériel - Mauvaise résistance de terminaison

Cause : Matériel - Perturbations électromagnétiques (Fils non torsadés ou non-blindés si nécessaire)

Cause : Matériel -

Mode de défaillance : *Temps* - périodicité supérieure à celle attendue (pour les messages périodiques)

2.3.2.5 Trigger de tâche

Ce bloc est à l'origine de toutes les tâches logicielles créées avec MotoHawk. Il doit être associé à un bloc sous-système « function-call » qui contient la stratégie logicielle de cette tâche. Le rôle de ce bloc est de gérer le déclenchement périodique ou événementiel des tâches. Dans le cas d'une tâche périodique, on paramètre l'intervalle de la périodicité. Dans le cas d'une tâche événementielle, on paramètre l'événement déclencheur comme la réception d'une trame CAN par exemple.

Étude des modes de défaillance potentiels du bloc trigger

- Omission : le « trigger » n'est pas concerné par ce mode de défaillance car à partir du moment où le bloc est dans le modèle, il effectuera forcément un calcul ;

- Commission : le déclenchement de l'échantillonnage des données n'est pas géré par ce bloc, il dépend des blocs « trigger » de tâche ou de blocs conditionnés dans les sous-systèmes parents ;

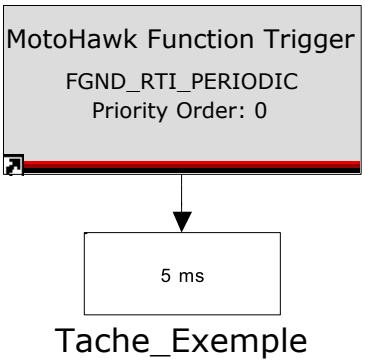


FIGURE 2.13 – Bloc Trigger de tâche

- Erroné : ce bloc est susceptible de fournir des valeurs erronées. L'erreur peut être subtile c'est-à-dire qu'elle appartient à l'intervalle attendu mais n'est pas correcte. Elle peut aussi être grossière c'est-à-dire que la valeur fournie est en dehors de l'intervalle attendu ou le comportement est complètement différent de celui en opération normale ;

- Temps : la notion de temps réel n'est pas gérée par ce bloc mais par des blocs parents comme les blocs « trigger » de tâche.

Étude de propagation

- Si la donnée d'entrée du bloc est correcte alors sa sortie ne peut être défaillante qu'en fonction de ces modes de défaillance propres vus précédemment ;

- En revanche si le signal matériel est défaillant alors le bloc logiciel se comporte de façon linéaire, on retrouve la même défaillance en sortie.

Étude des relations « Modes de défaillance / Causes / Recommandations »

Les relations de « Modes de défaillance / Causes / Recommandations » sont rassemblées dans l'annexe A.5. Les blocs de type « Trigger » influencent la temporisation d'exécution du code qu'elle soit périodique ou événementielle. Les modes de défaillance propres de ces blocs sont principalement liés au matériel comme le processeur qui s'il est trop chargé peut provoquer des retards d'exécution.

Mode de défaillance : *Retard important voire non exécution*
Cause : Matériel - Défaillance du processeur
Cause : Matériel - Surcharge du processeur
Cause : Logiciel - problème de priorisation
Mode de défaillance : *Avance/Retard* - Exécution erratique
Cause : Matériel - Défaillance du processeur

Cause : Matériel - Surcharge du processeur
Mode de défaillance : *Retard léger*
Cause : Matériel - Surcharge du processeur

2.4 Arbres de défaillances (AdD) ❷

Dans le cadre de la phase d'analyse de SdF des modèles logiciels dans la méthodologie ALEBAS, une partie de l'analyse des modèles consiste à employer la technique des AdD. Cette technique est bien adaptée pour être appliquée sur des diagrammes de flux comme ceux fournis par Simulink. En effet, la propagation déductive se fait des sorties vers les entrées du modèle en remontant les flux. Nous avons montré dans la bibliographie les limites de l'utilisation des AdD basés sur une propagation de proche en proche, c'est-à-dire uniquement sur l'information structurelle du modèle. Nous présentons dans la suite notre processus de propagation déductive qui est supporté par les informations collectées lors de l'analyse individuelle des blocs et qui rend les AdD plus pertinents.

2.4.1 Conception des arbres : propagation déductive

Cette partie expose le mécanisme de propagation déductive qui permet de construire des AdD à partir d'un modèle logiciel embarqué. La technique des AdD est une méthode déductive, c'est-à-dire qu'à partir d'un événement redouté assigné à une sortie du modèle, les causes potentielles d'occurrence de cet évènement sont identifiées en naviguant dans le modèle en remontant les flux de données. c'est-à-dire que le modèle est parcouru des blocs de sortie vers les blocs d'entrée. La première étape du mécanisme de propagation consiste à identifier les sorties du modèles. Deux solutions sont possibles dans ce sens :

- dans le cas de l'étude d'un modèle purement Simulink, les blocs de sortie sont les blocs « Outport » qui se trouvent dans le niveau le plus haut du modèle ;
- dans le cas de l'étude d'un modèle Simulink/MotoHawk, les blocs de sortie sont ceux qui permettent la commande des sorties matérielles du calculateur comme : les sorties analogiques ou digitales ou encore les envois de messages CAN.

Ensuite pour chacun de ces blocs de sorties, leurs ER potentiels doivent être identifiés. Un arbre est créé par événement redouté de chaque sortie du modèle.

Le mécanisme de propagation déductif va être confronté à trois types de liens dans le modèle :

- **lien physique unique** : il s'agit du cas le plus répandu dans ce genre de modèle. Un lien entre les blocs partage une donnée unique. Dans le cas de la propagation déductive, un port d'entrée n'est relié qu'à un seul port de sortie en amont.

- **lien physique commun** : un lien entre les blocs partage plusieurs données. La difficulté réside dans le suivi du flux, il faut retrouver à quel moment la donnée rentre dans le bus.

- **pas de lien physique** : les données sont distribuées entre les blocs sans lien visible, par l'intermédiaire de la mémoire du système (de l'ECU dans l'embarqué). Dans ce cas, un bloc de lecture peut avoir plusieurs blocs d'écriture en amont. Il peut aussi s'agir de bloc ayant une influence du fait de leur positionnement hiérarchique en amont comme les « trigger ».

Ces trois types de lien impliquent d'établir différentes règles de propagation. Pour les liens physiques, il suffit de suivre les liaisons entre les blocs. Pour le cas des liens physiques communs, il est nécessaire d'identifier un paramètre supplémentaire pour déterminer le flux à suivre et ainsi, ne conserver que les chemins corrects. Pour les liens non-physiques, il faut chercher les autres blocs qui partagent la même donnée échangée et qui sont susceptibles de la remplacer. Il faut aussi rechercher dans la hiérarchie si des blocs « trigger » sont présents.

2.4.2 Influence du comportement sur la propagation

Une autre particularité des modèles de logiciel embarqué à ce stade du développement est que les blocs ont un comportement qui peut affecter la propagation des ER. Comme les blocs utilisés à ce stade sont basiques, c'est-à-dire qu'ils opèrent des opérations relativement simples, leur comportement est bien connu et il sera le même quelle que soit la fonction logicielle développée. L'analyse individuelle des blocs a permis d'établir le comportement de chacun des blocs vis-à-vis de la propagation des ER. Grâce à la prise en compte de cette information lors de la propagation déductive, il est possible de réduire le champ des chemins empruntés par une erreur se propageant dans le modèle. Ainsi, les causes potentielles de l'apparition d'un événement redouté à la sortie du modèle se précisent. La figure 2.14 présente la propagation déductive lorsqu'elle est effectuée par simple relation de proche en proche entre les blocs.

On remarque que pour une sortie et qu'importe l'événement redouté étudié, tous les chemins en amont sont empruntés. En appliquant la combinaison de nos règles de propagation dédiées au modèle de logiciel embarqué et le comportements de propagation de chacun des blocs, on obtient pour l'étude de trois événements redoutés différents des chemins de propagation différents (voir figure 2.15). Les événements redoutés et les chemins associés de cette exemple s'organisent comme suit : non-respect de la périodicité d'envoi du message CAN (bleu) ; valeur erronée du signal dans le message CAN (orange) ; comportement intempestif du signal CAN (vert)

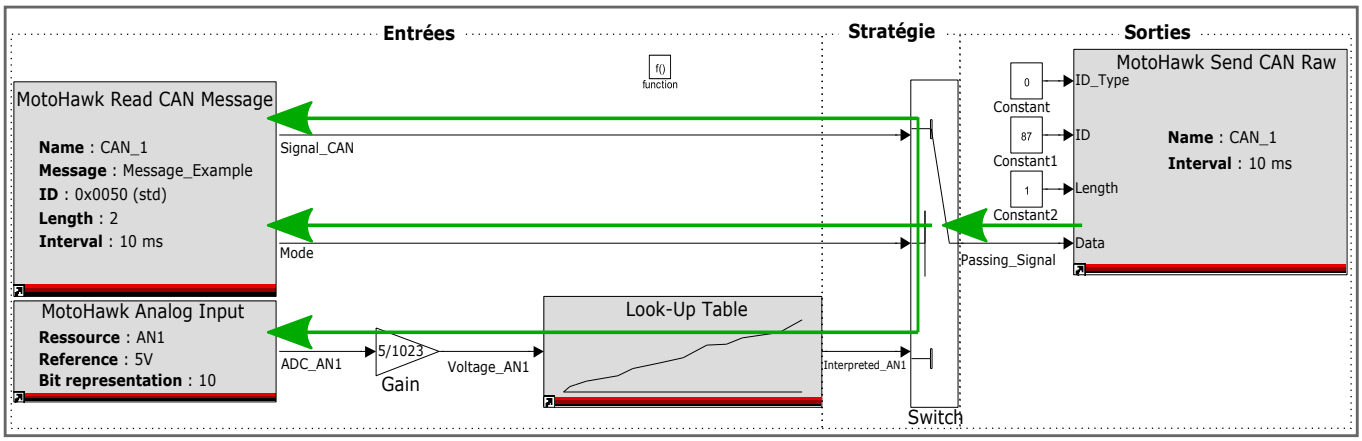


FIGURE 2.14 – Propagation déductive basée sur la structure du modèle

2.4.2.1 Exemple d'application des arbres de défaillances

Cette section compare les résultats des AdD obtenus selon la méthode de propagation uniquement structurelle et celle intégrant les données comportementales et de temps réel. On note tout d'abord pour la méthode de propagation uniquement structurelle que la propagation est identique quelque soit l'événement redouté étudié. L'AdD obtenu avec cette méthode est présenté en annexe B dans la figure B.1. Les arbres obtenus en appliquant l'analyse des AdD selon la méthodologie ALEBAS sont également exposés en annexe B dans les figures B.3, B.4 et B.2.

TABLEAU 2.2 – Caractéristiques du modèle logiciel étudié

	Modèle logiciel
Nombre de blocs	9
Nombre d'entrées	2
Nombre de sorties	1

TABLEAU 2.3 – Caractéristiques des arbres de défaillances obtenus

	Analyse structurelle	**ALEBAS Erroneous**	**ALEBAS Omission/-Commission**	**ALEBAS Temps réel**
Nombre de nœuds	32	22	16	9
Nombre de sources potentielles de défaillances (feuilles de l'arbre)	22	16	12	6

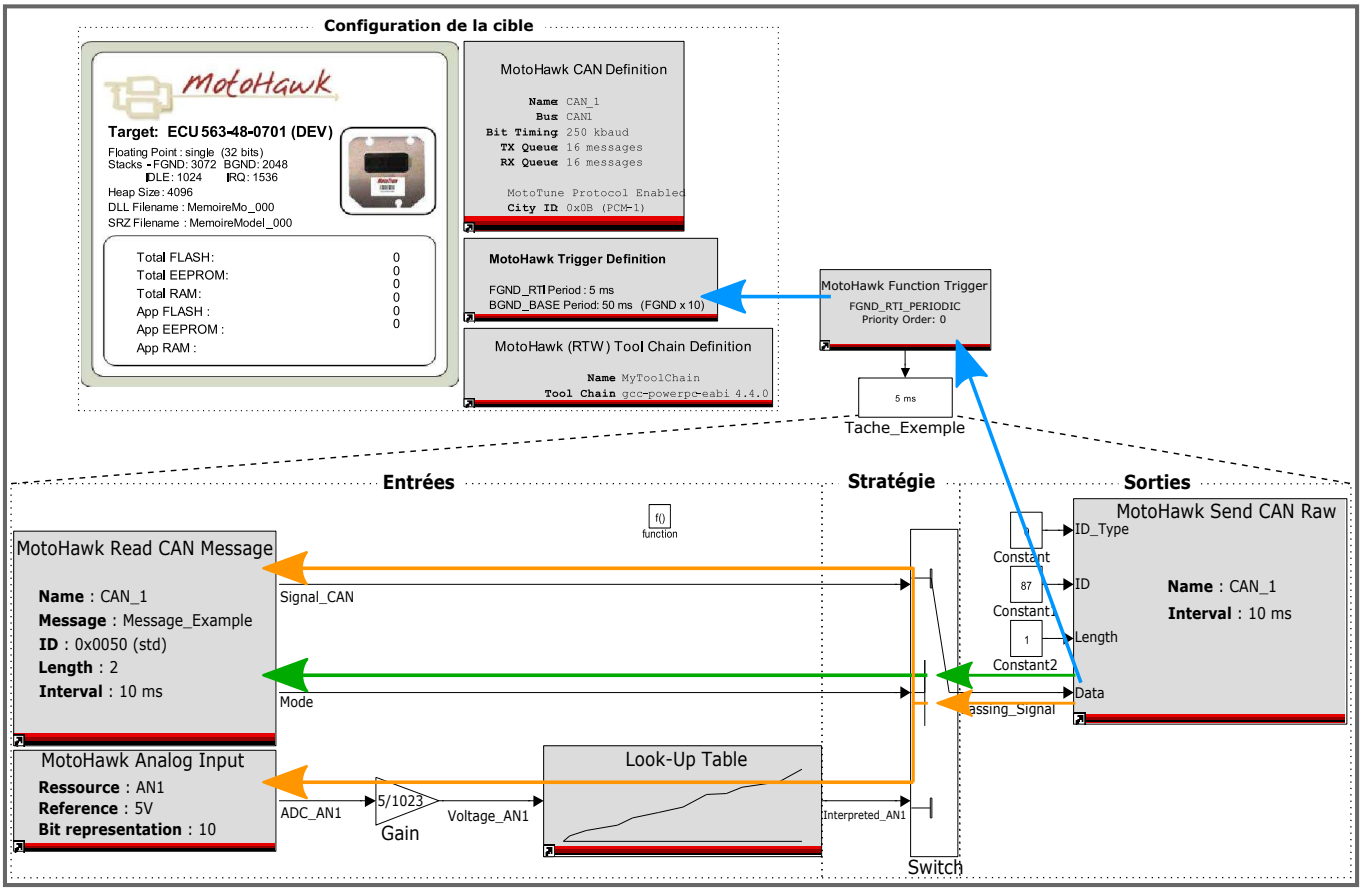


FIGURE 2.15 – Propagation déductive avec prise en compte du comportement des blocs

2.5 Analyse des modes de défaillance et de leurs effets appliquée au logiciel (AMDE logicielle) ❸

Après les AdD, ALEBAS repose aussi sur l'application d'une AMDE sur le modèle de logiciel embarqué. L'AMDE logicielle est une méthode inductive, c'est-à-dire qu'elle vise à déterminer l'impact de l'apparition d'une défaillance d'un des composants sur les sorties du système étudié. Pour cette étude, le modèle logiciel est parcouru de ses entrées vers ses sorties en suivant le flux des données qui s'y propagent.

L'AMDE logicielle consiste à remplir un tableau qui facilite l'analyse de SdF. Ce tableau est complété en récoltant les informations sur les blocs au cours de la propagation. La propagation et la récolte des informations s'appuient sur les analyses locales réalisées précédemment. Le tableau est composé de neuf colonnes comme présenté plus loin dans 2.18. Le contenu de chaque colonne est décrit ci-dessous :

- **Fonction** : cette colonne comporte le nom de la fonction du bloc en cours d'étude. Dans le cas des modèles logiciels, il s'agit du nom du sous-système parent du bloc étudié.

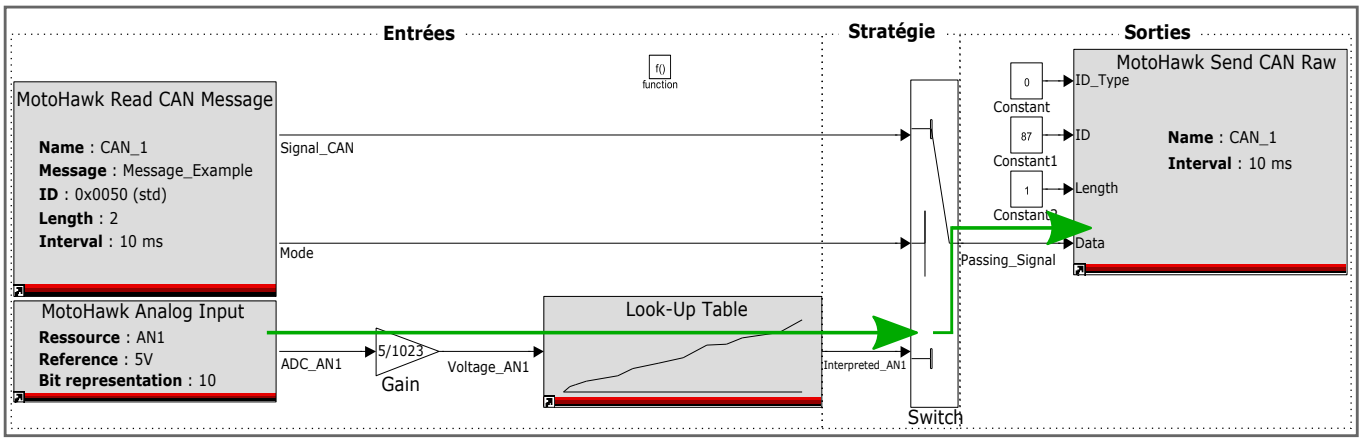


FIGURE 2.16 – Propagation inductive

- **Bloc** : cette colonne indique le nom du bloc en cours d'étude.
- **Mode de défaillance potentiel** : dans cette colonne est décrit les potentiels comportements déviants du bloc étudié. Ces informations proviennent de l'analyse locale.
- **Cause potentielle** : cette colonne définit les fautes potentielles à l'origine de l'occurrence du mode de défaillance présenté dans la colonne précédente.
- **Effet potentiel à l'échelle de l'ECU** : cette colonne indique le comportement déviant que peut subir le(s) bloc(s) de sortie du modèle logiciel lorsqu'il est impacté par l'occurrence du mode de défaillance étudié. Cette relation est déterminée lors du processus de propagation.
- **Effet potentiel à l'échelle du système** : cette colonne décrit l'effet potentiel de l'occurrence du mode de défaillance étudié sur le système complet dans lequel est intégré le logiciel embarqué. Pour l'automobile par exemple, il s'agira de l'effet sur la voiture. Cet effet est déterminé par la projection de l'effet potentiel à l'échelle du logiciel dans l'analyse de SdF système, où cet effet est un mode défaillance du système à ce niveau, comme présenté en figure 2.17. Par propagation au niveau du système, il a été déterminé l'effet de ce mode de défaillance. Cette colonne peut être complétée seulement si l'analyse de SdF au niveau système existe.
- **Niveau de risque** : cette colonne définit le niveau de risque associé à l'effet potentiel sur le système qui a été déterminé dans la colonne précédente. Par exemple pour l'automobile, ce niveau de risque correspond aux niveaux ASIL.
- **Action recommandée** : cette colonne décrit les moyens proposés pour tolérer ou éliminer une faute. En fonction de la faute étudiée, la recommandation peut concerner une amélioration de l'architecture logiciel ou la correction de paramètres d'un

ou plusieurs blocs ou l'application de vérifications ou encore des améliorations touchant l'architecture du système.

- **Action appliquée** : cette colonne sert au développeur à indiquer les recommandations qu'il a choisi d'appliquer dans la nouvelle version du logiciel. Si besoin, il décrit l'effet de cette amélioration. Cette colonne permet d'avoir un suivi des modifications.

La difficulté majeure dans l'utilisation de l'AMDE à un stade avancé du développement est de rendre l'analyse suffisamment compréhensible pour être efficacement exploitée. L'analyse individuelle des composants basiques facilite cette approche, elle permet au moment de remplir le tableau AMDE d'y récupérer des informations spécifiques et pertinentes.

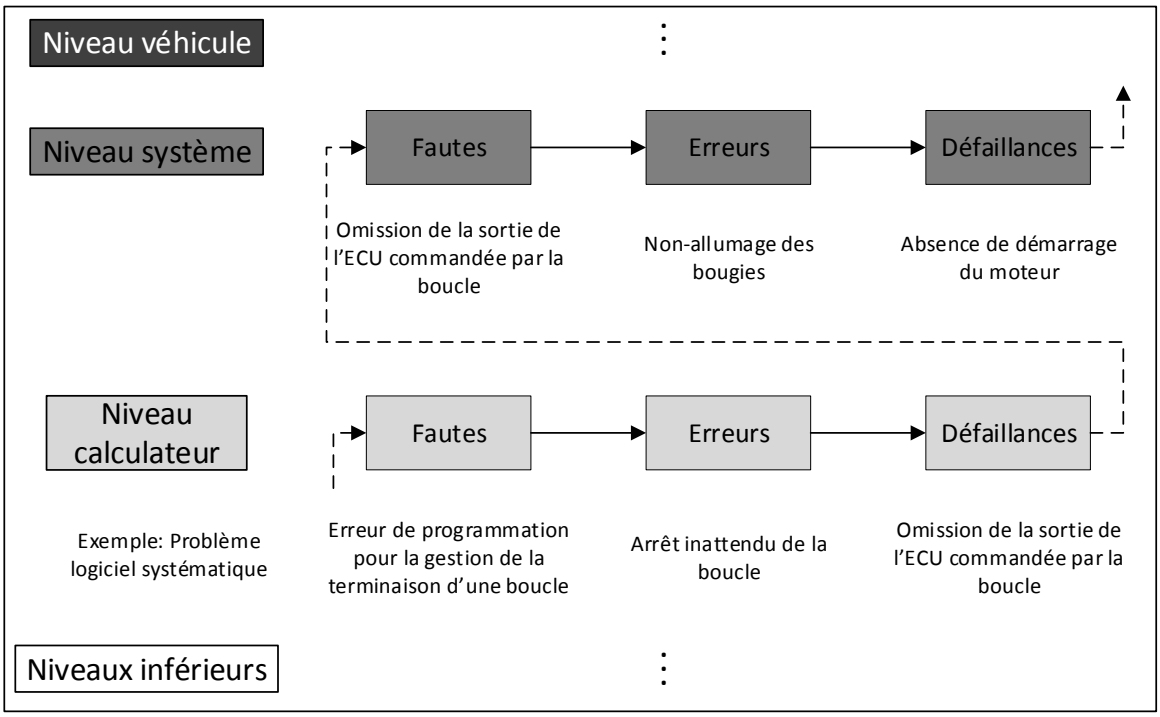


FIGURE 2.17 – Relation entre les fautes, les erreurs et les défaillances sur plusieurs niveaux d'abstraction

2.5.1 Propagation inductive

L'un des principaux objectifs de l'AMDE est de connaître l'impact d'un composant qui aurait un comportement déviant sur le logiciel. Dans le cadre des logiciels embarqués, l'intérêt va se porter sur l'impact aux sorties du modèle logiciel c'est-à-dire les interfaces entre le logiciel et le matériel. Pour mener cette étude, une navigation descendante est

conduite. La navigation descendante suit le flux de données dans le modèle logiciel de ses entrées vers ses sorties.

Le premier objectif de l'AMDE est de rassembler les informations sur le comportement local des composants. Étant donné que les composants étudiés dans le cadre des logiciels embarqués en phase de conception détaillée sont des composants basiques, le comportement local correspond au comportement individuel qui a été préalablement étudié lors de l'analyse individuelle. À chaque fois qu'un composant est rencontré lors de la propagation inductive, ces informations de modes de défaillance sont importées dans la colonne du tableau AMDE correspondante.

2.5.2 Recommandations SdF adaptées

La colonne « Recommandation » du tableau de l'AMDE logicielle est remplie en récupérant les recommandations proposées lors de l'analyse individuelle des composants basiques pour la cause correspondante. Dans le cas où l'analyse système n'a pas été réalisée alors le niveau de risque n'est pas connu donc toutes les recommandations sont ajoutées. Par contre, si le niveau de risque est connu grâce à l'analyse système réalisée en amont, alors une seule recommandation est proposée et adaptée à ce niveau de risque. Ces recommandations proposent différents niveaux de continuité de service comme décrits précédemment dans le tableau 2.1 de la section 2.3.1.

2.5.3 Exemple d'application de l'AMDE

Cette partie présente l'application de l'AMDE logicielle selon la méthodologie ALEBAS sur l'exemple d'application. La figure 2.18 est un extrait du tableau AMDE obtenu pour l'analyse du modèle logiciel de l'exemple. Cet extrait adresse l'analyse du bloc d'entrée « Analog Input » du modèle logiciel (colonne Blocs). Dans les informations issues de l'analyse individuelle, il a été relevé que ce composant peut produire plusieurs modes de défaillance ; celui étudié dans cet extrait est « Signal bloqué dans des valeurs basses » (colonne Modes de défaillance). Ensuite, dans l'analyse individuelle, il existe plusieurs causes à l'apparition de ce mode de défaillance, celles présentent dans cet extrait sont les causes matériel –ou *Hardware*– (HW) suivantes : « connexion filaire défectueuse » et « convertisseur analogique-numérique défectueux » (colonne Causes). À l'aide de la propagation inductive, il est déterminé que le bloc impacté dans le cas de l'apparition d'une défaillance du bloc analogique est le bloc de sortie d'envoi d'une trame CAN « MotoHawk Send CAN Raw ». Les différents comportements déviants de ce bloc, récupérés de l'analyse individuelle, sont ajoutés à la colonne « Effets potentiels des défaillances à l'échelle du Logiciel –ou *Software*– (SW) / ECU ». Pour le remplissage des autres colonnes, trois situations différentes sont décrites afin de montrer les variantes possibles de l'analyse.

Tout d'abord, si l'on se place dans le cas où l'analyse système n'a pas été réalisée alors les colonnes « Effets potentiels des défaillances sur le système » et « Niveau de risque » restent vides. La colonne « Recommandations » est complétée avec la liste de toutes les recommandations possibles qui sont récupérées de l'analyse individuelle pour les causes étudiées.

Deux autres solutions à présenter nécessitent d'inscrire le modèle logiciel dans deux contextes différents. Les contextes suivants requièrent en réalité un modèle logiciel plus complexe que celui utilisé comme exemple d'application, mais l'objectif ici est simplement de montrer différents comportements de l'analyse en les illustrant avec des systèmes différents. Le premier contexte est celui d'un tableau de bord, c'est-à-dire que le modèle logiciel présenté pourrait servir à envoyer deux données, alternativement, par le bus CAN à un afficheur. Les résultats de ce contexte dans l'AMDE logicielle sont entourés par des pointillés bleus dans la figure 2.18. Ces deux données qui transitent sont celles reçues en entrée du modèle ; le « Signal_CAN » est la température du moteur et l'entrée analogique est la température intérieure du véhicule. Les effets potentiels des défaillances à l'échelle du SW/ECU sont par projection des modes de défaillance dans l'analyse système, on peut alors trouver leur impact sur le système. Ces informations d'effets sur le système sont rassemblées dans la colonne « Effets potentiels des défaillances à l'échelle du système ». La colonne « Niveau de risque » est complétée par le niveau de risque le plus important des effets sur le système. Dans la colonne « Recommandations », seule la recommandation correspondante au niveau de risque à éviter est exposée.

Pour le second contexte, il inscrit le modèle logiciel de l'exemple dans un système de « bypass » de la pédale d'accélérateur, dans le cadre de la robotisation d'un véhicule pour le rendre autonome. Le modèle logiciel sert à basculer la commande de l'accélérateur entre le signal de la pédale et une consigne envoyée par le calculateur principal du véhicule autonome. Cet exemple sera abordé plus longuement dans le chapitre 4 comme cas d'étude. Les résultats de ce contexte dans l'AMDE logicielle sont entourés en orange dans la figure 2.18. Le remplissage des colonnes « Effets potentiels des défaillances sur le système » et « Niveau de risque » est toujours effectué de la même manière. Cette fois en cas de défaillance, les risques sur le système impactent le comportement du moteur dans le comportement routier de la voiture, ce qui implique un niveau de risque important : niveau 3. Ainsi, ce niveau de risque entraîne des recommandations plus importantes à mettre en place.

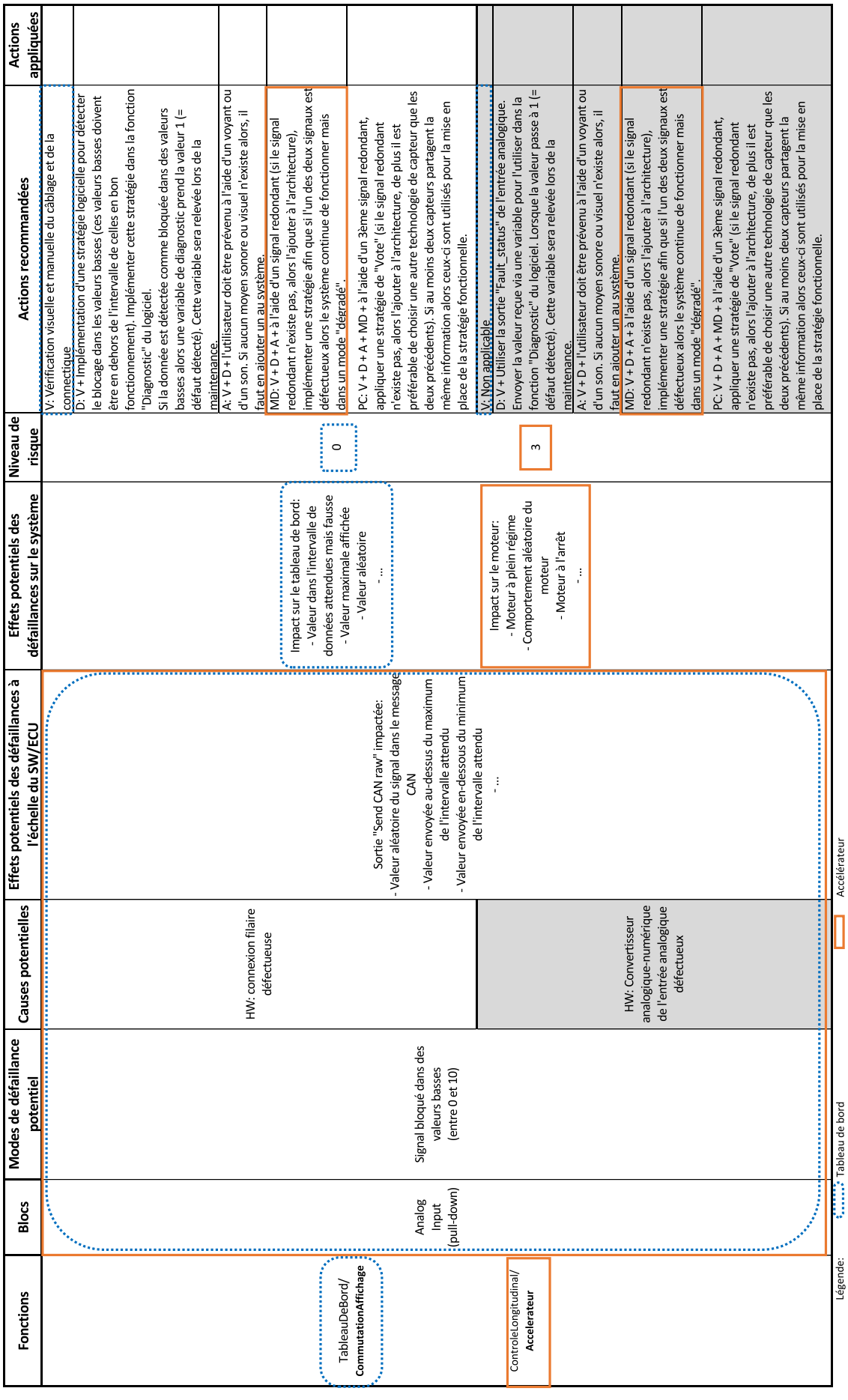


FIGURE 2.18 – Extrait d'AMDE logicielle appliquée à l'exemple

2.6 Interprétation des résultats et choix de recommandations ❹

À partir des résultats obtenus par l'application des deux techniques d'analyse de SdF, les AdD et l'AMDE, l'ingénieur doit définir les améliorations à apporter au logiciel embarqué. Tout d'abord, l'AMDE propose des premières recommandations. Dans le cas où le lien entre l'analyse logicielle et l'analyse système n'a pas été faite, les recommandations ne sont pas proposées spécifiquement au niveau de risque de l'événement qui pourrait être occasionné par la défaillance du logiciel. L'ingénieur peut ainsi commencer à réduire le champ des recommandations proposées en apportant cette information du niveau de risque associé à chaque comportements déviants des sorties du logiciel. Ainsi, les recommandations à conserver sont celles qui sont en accord avec le niveau de risque système. Ensuite, l'affinage de la sélection des recommandations à appliquer nécessite une importante expertise. L'objectif est d'être capable d'établir les recommandations à appliquer, en combinant les compétences suivantes : une bonne connaissance du logiciel embarqué développé (technologies matérielles et logicielles) et une bonne faculté d'interprétation des AdD et de l'AMDE. Il est parfois nécessaire de réunir plusieurs personnes pour être capable de rassembler l'ensemble des compétences requises. Le processus d'obtention des recommandations à appliquer peut dépendre de l'expérience des analystes et des pratiques de l'entreprise. Dans tous les cas, cela requiert de combiner les résultats de l'analyse de SdF sur le logiciel embarqué.

Nous proposons l'approche suivante pour obtenir les recommandations à appliquer. Tout d'abord, une nouvelle sélection des recommandations peut être réalisée en parcourant le tableau de l'AMDE et en identifiant les recommandations qui sont déjà implémentées dans le logiciel. Ces recommandations peuvent dans ce cas être écartées en notifiant, dans la dernière colonne du tableau AMDE « Actions appliquées », qu'elles sont déjà appliquées. Ensuite, à l'aide des AdD, il est possible de déterminer les parties du logiciel qui sont le plus à l'origine de l'occurrence d'événements redoutés, par exemple, certains nœuds (modes de défaillance) se retrouvent dans plusieurs AdD. Par expérience, l'analyste est capable de cibler certains blocs qu'il considère généralement plus enclins à produire un mode de défaillance. En associant cette connaissance à la position de ces blocs et à leurs modes de défaillance dans les AdD, l'analyste est capable d'indiquer des recommandations pour pallier à l'apparition potentielle de problèmes via ces blocs.

Grâce à cet affinage de la sélection des recommandations et l'ajout de nouvelles recommandations appropriées, le modèle logiciel peut être amélioré en implémentant ces préconisations.

2.7 Implémentation des recommandations ❺ et itération de la méthodologie ALEBAS ❻

2.7.1 Implémentation des recommandations ❺

L'implémentation des recommandations consiste à suivre la liste des recommandations qui ont été établies précédemment et à les appliquer dans le logiciel. L'impact des recommandations peut impliquer de simples optimisations des paramètres des blocs, jusqu'à l'implémentation de stratégies complexes afin de maintenir les performances du logiciel. Au fur et à mesure de l'implémentation des recommandations, celles-ci doivent être annotées dans la colonne « Recommandations appliquées » du tableau de l'AMDE, afin d'indiquer ce qui a été réalisé dans le modèle pour mettre en place cette recommandation. Par la suite, cette colonne « Recommandations appliquées » peut servir pour le suivi de version du modèle logiciel.

Dans certains cas, les améliorations requièrent des modifications au niveau du système et du matériel. Normalement, ces besoins de modifications ont déjà été détectés lors de l'analyse système. Il peut s'agir par exemple, du besoin d'une redondance d'un capteur pour assurer les performances du système, qui implique l'ajout d'un signal d'entrée pour le calculateur et le logiciel. Dans le cas de notre analyse et des résultats obtenus, ces besoins de modifications peuvent être à nouveau décelés au niveau du logiciel. Ceci peut servir de vérification et de confirmation vis-à-vis des analyses réalisées à un niveau d'abstraction plus élevé.

2.7.2 Exemple d'application des recommandations

En reprenant l'exemple, on applique les recommandations proposées dans le tableau AMDE. Ici, on ne traite pas le cas où l'analyse système n'a pas été réalisée car dans ce cas, on ne connaît pas quelles recommandations appliquer. On ne se préoccupe pas non plus du contexte « Tableau de bord » car les recommandations pour cet exemple ne concernaient pas de modifications logicielles. L'exemple abordé est celui de l'accélérateur dont l'analyse indique qu'il faut implémenter une stratégie de passage en « Mode Dégradé » en cas de détection des défauts. Pour cet exemple, on n'implémentera que le premier niveau de recommandation c'est-à-dire « Détection » car cet exemple est plus amplement détaillé dans le cas d'étude. L'objectif pour cette recommandation est : « Implémentation d'une stratégie logicielle pour détecter le blocage dans les valeurs basses (ces valeurs basses doivent être en dehors de l'intervalle de données lors du bon fonctionnement). Implémenter cette stratégie dans la fonction « diagnostic » du logiciel. Si la donnée est détectée comme bloquée dans les valeurs basses alors une variable de diagnostic prend la valeur 1 (= défaut détecté). Cette variable sera relevée lors de la maintenance. » Le modèle

logiciel décrivant l'implémentation de cette recommandation est présenté en figue 2.19.

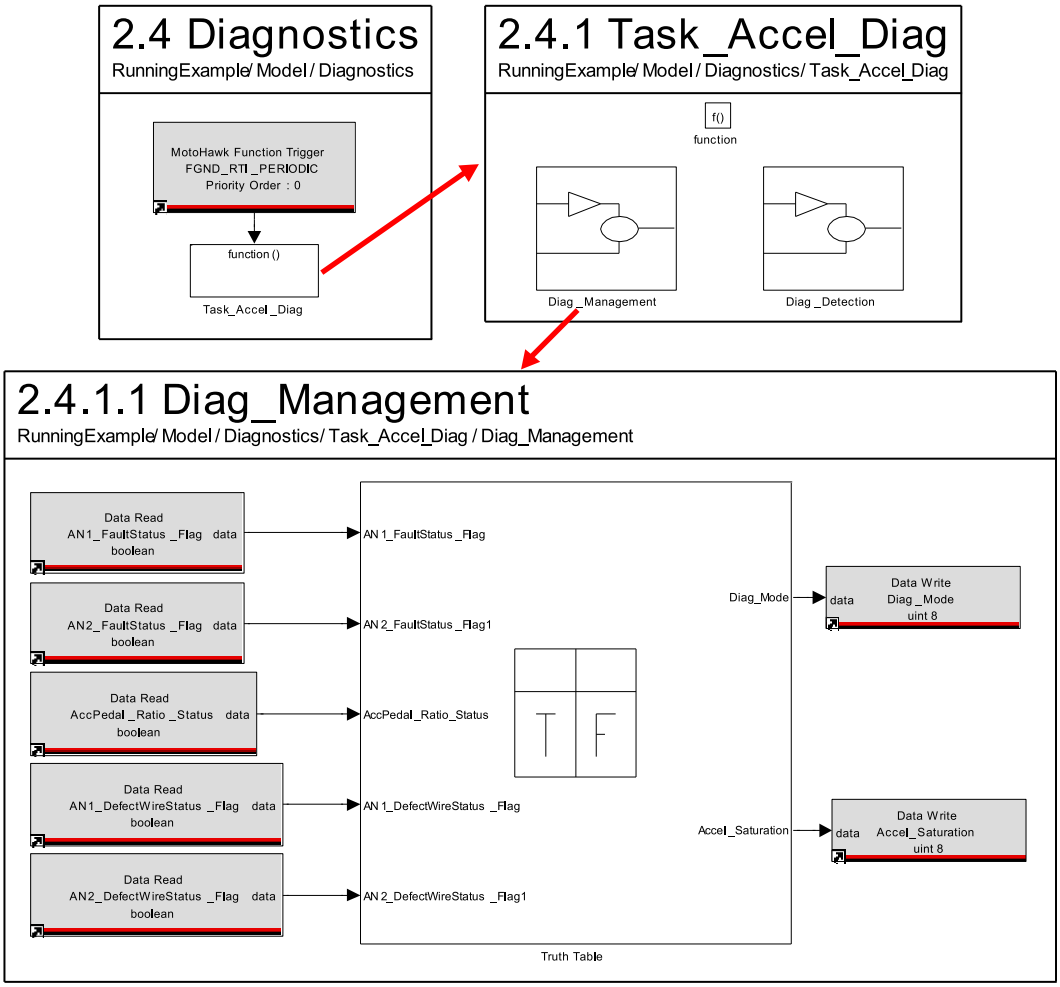


FIGURE 2.19 – Partie du modèle logiciel, fonction diagnostic, gérant le statut du système surveillé

2.7.3 Itération de la méthodologie ALEBAS ❻

Une fois que le logiciel a été modifié, il est possible de répéter la méthodologie afin d'opérer une nouvelle analyse sur le nouveau modèle de logiciel. Par l'application d'itérations d'ALEBAS, il est possible de découvrir de nouvelles recommandations du fait des modifications qui ont été apportées au logiciel. L'attention de l'analyse, à ce stade, portera notamment sur les parties du modèle logiciel qui ont été modifiées et celles qui sont impactées par ces modifications.

Au fur et à mesure de l'avancement dans le cycle de développement, les prototypes évoluent et s'enrichissent comme le logiciel associé. La méthodologie ALEBAS est appliquée pour chaque évolution du logiciel au cours du cycle de développement du système. Le renouvellement de la méthodologie peut profiter des analyses réalisées précédemment pour encore une fois focaliser les analyses sur les parties du logiciel qui se sont enrichies. L'objectif est de capitaliser sur le travail effectué auparavant.

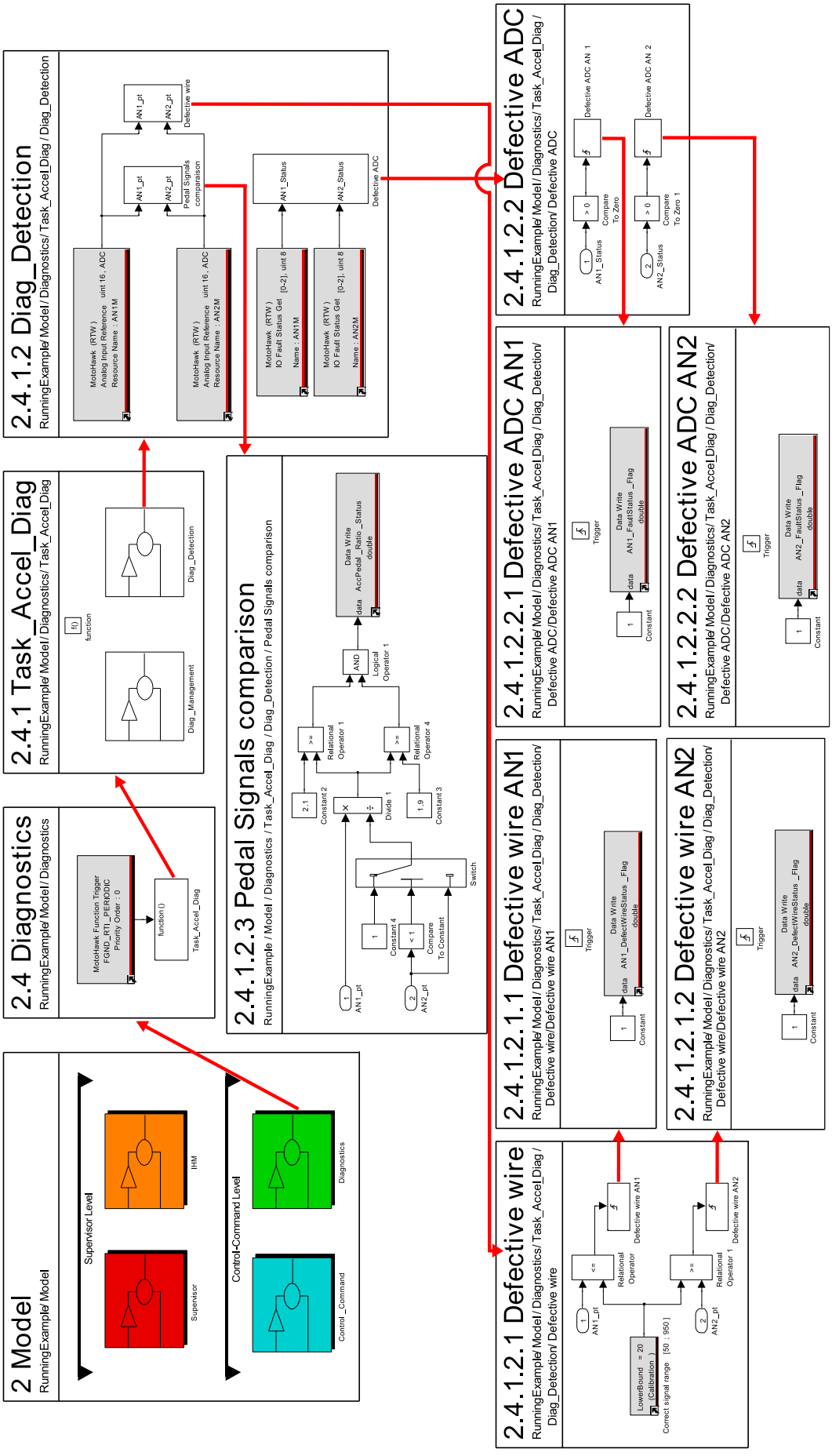


FIGURE 2.20 – Partie du modèle logiciel, fonction diagnostic, intégrant les recommandations de détection de défauts

2.8 Conclusion

Ce chapitre a présenté ALEBAS, une méthodologie destinée à améliorer les modèles de logiciel embarqué vis-à-vis des contraintes de SdF. ALEBAS s'inscrit dans un contexte de développement de systèmes innovants pour lesquels le cycle de développement est souvent incrémental et itératif. En effet, le système et le logiciel embarqué associé, évoluent au cours de différentes phases allant de la PoC puis aux prototypages et jusqu'à son homologation pour la série. Le logiciel prend aujourd'hui une importance considérable au sein des systèmes mécatroniques embarqués et les fonctions qu'il réalise sont de plus en plus complexes et à caractère critique. Il est donc nécessaire de répondre à des exigences de SdF pour assurer la sécurité des personnes, des biens et de l'environnement. La charge de travail pour le développement fonctionnel est déjà relativement importante et complexe ; à celle-ci vient s'ajouter les activités de SdF. Les équipes impliquées dans les phases de prototypage sont relativement restreintes pour une meilleure flexibilité et dynamique du projet. Il se révèle donc important d'accompagner les ingénieurs dans l'intégration des études de SdF dans leur processus de développement.

Dans ce contexte, la méthodologie ALEBAS vise à anticiper au plus tôt dans le cycle de développement du système, l'application des exigences de SdF pour escompter une mise en série future à un coût moindre du système. Cette méthodologie repose sur l'application d'une analyse de SdF à l'aide des techniques des AdD et de l'AMDE sur les modèles logiciels. ALEBAS est conçue de manière à convenir au processus itératif et incrémental en phase de prototypage. De plus, elle vise à capitaliser sur les résultats obtenus lors des analyses précédentes pour limiter la charge de travail à chaque itération. L'originalité de la méthodologie proposée est de conduire les analyses de SdF en phase de conception détaillée du logiciel ; afin d'apporter au plus près du code des améliorations visant à rendre le logiciel plus sûr. Les analyses de SdF menées au niveau de l'architecture système ou logiciel requièrent l'application d'une analyse locale de chacun des composants présents dans le modèle. Au stade de conception détaillée, le nombre de blocs utilisés dans le modèle est plus grand qu'à des niveaux d'abstraction plus élevés. La charge de travail pour l'application d'analyses locales devrait donc augmenter, mais à ce niveau, nous sommes en présence de composants basiques qui opèrent des fonctions simples : calcul mathématique, condition, logique... Ce type de bloc a un comportement indépendant de son environnement, il n'est donc pas nécessaire d'appliquer une analyse locale (dans son environnement) pour chacun des blocs ; mais une analyse individuelle de chaque type de bloc permet d'effectuer le travail qu'une seule fois. Afin de profiter pleinement des données récoltées lors de l'analyse individuelle et d'éviter à l'analyste d'avoir à consulter ces résultats pour mener la construction des AdD et du tableau AMDE, le processus d'analyse devra être automatisé, ce sujet est traité dans le chapitre suivant.

L'exemple d'application a permis d'illustrer les résultats des analyses et l'amélioration du logiciel.

Chapitre 3

Mise en œuvre de la méthodologie ALEBAS

Sommaire

3.1 Introduction

En phase de prototypage, le logiciel embarqué est amené à évoluer régulièrement. Ces changements imposent le renouvellement de l'analyse de SdF, afin d'être en adéquation avec la nouvelle version du logiciel embarqué. L'analyse de SdF permet de déterminer les parties du logiciel qu'il faut rendre plus sûres et ainsi aider le développeur à implémenter les stratégies de SdF. La mise à jour perpétuelle de l'analyse de SdF peut rapidement devenir laborieuse et source d'erreur si elle est totalement effectuée manuellement. Pour cette raison, une chaîne d'outils logiciels a été développée pour automatiser l'analyse de SdF sur la base de la méthodologie ALEBAS. La figure 3.1 illustre le périmètre d'action de l'automatisation de la méthodologie proposée. L'outil logiciel d'automatisation se charge notamment des tâches de construction des AdD ❷ et du remplissage du tableau de l'AMDE ❸. Les tâches d'analyse individuelle des composants basiques ❶ et d'interprétation des résultats et choix ❹ nécessitent une expertise humaine importante, il n'a donc pas été envisagé de les automatiser pour le moment. L'étape d'implémentation des recommandations ❺ n'a pas encore été automatisée car il est nécessaire, dans un premier temps, d'avoir un retour utilisateur et d'éprouver les tâches d'étude de SdF automatisée.

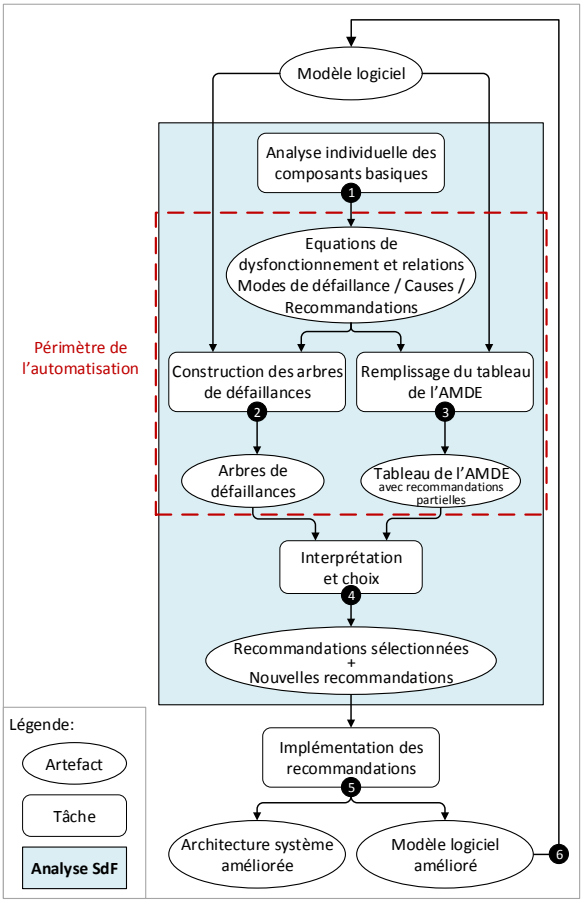


FIGURE 3.1 – Périmètre d'automatisation de la méthodologie ALEBAS

Le synoptique de la procédure d'automatisation est proposé dans la figure 3.2. On retrouve les tâches de la méthodologie ALEBAS avec : l'analyse individuelle des composants basiques ❶, la construction des AdD ❷ et de remplissage du tableau de l'AMDE ❸. Ces tâches se décomposent en opérations pour leur automatisation. Les données d'entrée de l'outil logiciel sont le modèle de logiciel embarqué à étudier, la base de données SdF contenant les résultats de l'analyse individuelle, et optionnellement des paramètres pour orienter l'analyse si nécessaire. Les données attendues en sortie sont des AdD facilement visualisables et pertinents vis-à-vis de l'étude d'un logiciel embarqué, et un tableau récapitulant l'AMDE du logiciel embarqué étudié contenant des recommandations partielles d'amélioration du logiciel, afin d'assurer une meilleure SdF. Dans ALEBAS, ces recommandations sont ensuite affinées et complétées lors de la tâche d'interprétation ❹ des AdD et de l'AMDE.

Grâce à cette chaîne d'outils logiciels, les principales tâches répétitives et pour lesquelles la plus value humaine est limitée, sont automatisées. L'outil logiciel actuel a été développé sous Matlab. Il permet l'étude de modèles issus de l'outil Simulink ayant été développés uniquement avec la librairie native du même nom ou avec cette librairie native combinée à la librairie complémentaire MotoHawk [Woo]. Concernant les données de sortie de l'outil, il a été choisi de les soumettre à d'autres formalismes que ceux de Matlab comme le format eXtensible Markup Language (XML) ou HyperText Markup Language (HTML), afin de faciliter le partage des résultats et leur interopérabilité pour les exploiter dans d'autres environnements.

Les sections suivantes décrivent chacune des opérations du processus d'automatisation qui permettent, à partir du modèle logiciel et de la base de données SdF, de générer les AdD et le tableau de l'AMDE.

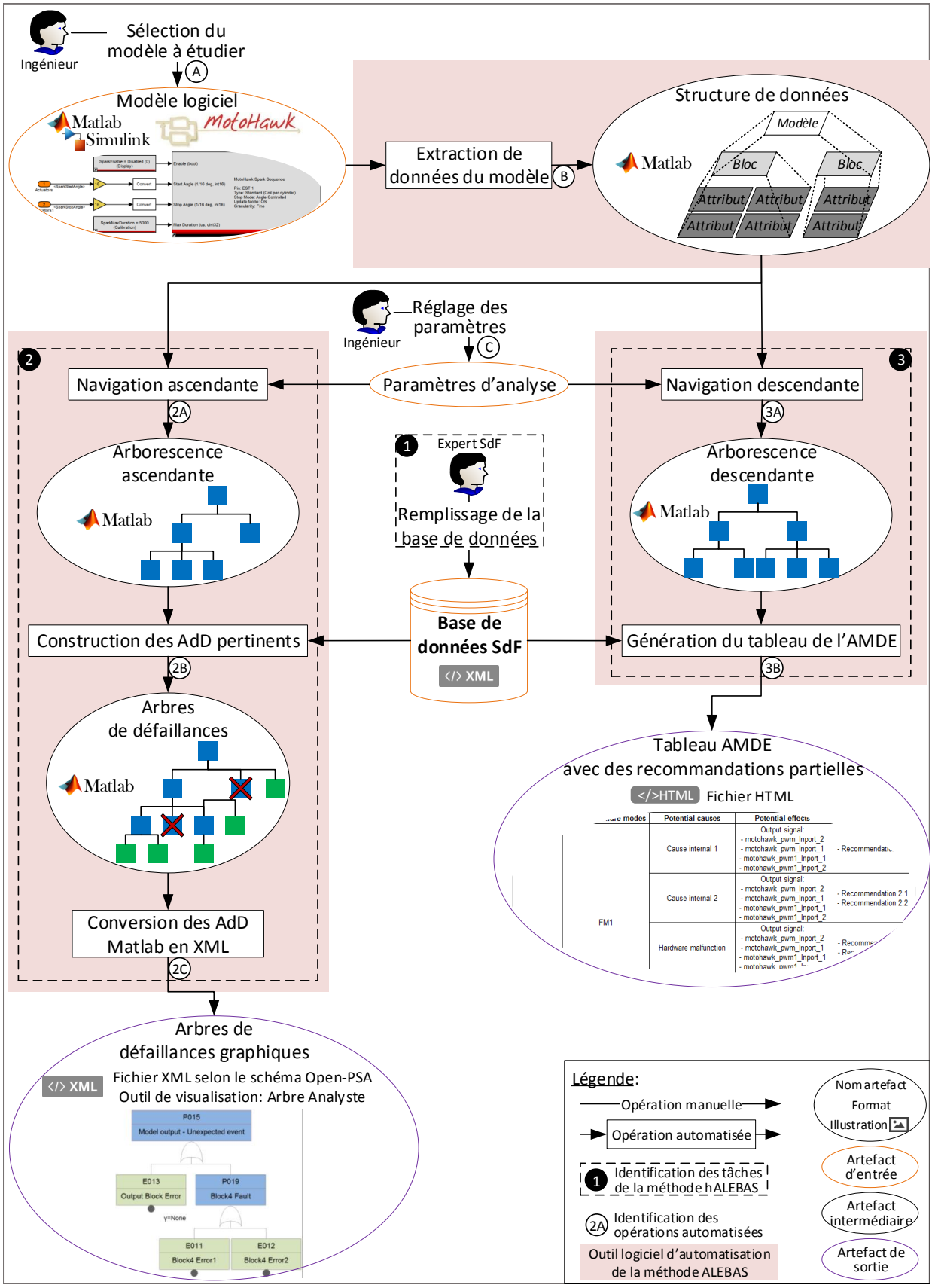


FIGURE 3.2 – Synoptique de l'automatisation de la construction des AdD et du remplissage de l'AMDE pour l'analyse SdF d'un modèle de logiciel embarqué dans le cadre de la méthodologie ALEBAS

3.2 Mise en forme des données d'entrée

Certains pré-requis sont nécessaires avant de pouvoir opérer la construction des AdD et le remplissage du tableau de l'AMDE. En premier lieu, il est indispensable de vérifier la conformité du modèle de logiciel embarqué à étudier, selon certains critères détaillés dans la section suivante. Ensuite, la construction des AdD et le remplissage du tableau de l'AMDE dépendent des informations issues de l'analyse individuelle des composants basiques. Les informations issues de l'analyse individuelle sont recueillies dans une base de données formalisée pour permettre à l'outil d'automatisation de les exploiter. L'architecture et le remplissage de cette base de données SdF sont décrits dans la deuxième section suivante.

3.2.1 Conformité du modèle de logiciel embarqué et identification de son type

L'entrée principale du processus d'automatisation est le modèle de logiciel embarqué à étudier Ⓐ (figure 3.2). Pour garantir que le processus d'automatisation ne rencontre pas d'erreur au cours de son exécution et pour qu'il produise des résultats corrects, il est nécessaire de vérifier que le modèle fourni en entrée répond à certains critères pour lesquels l'outil d'automatisation a été développé. La figure 3.3 expose l'algorigramme qui vérifie la conformité du modèle logiciel lorsque celui-ci a été sélectionné par l'utilisateur.

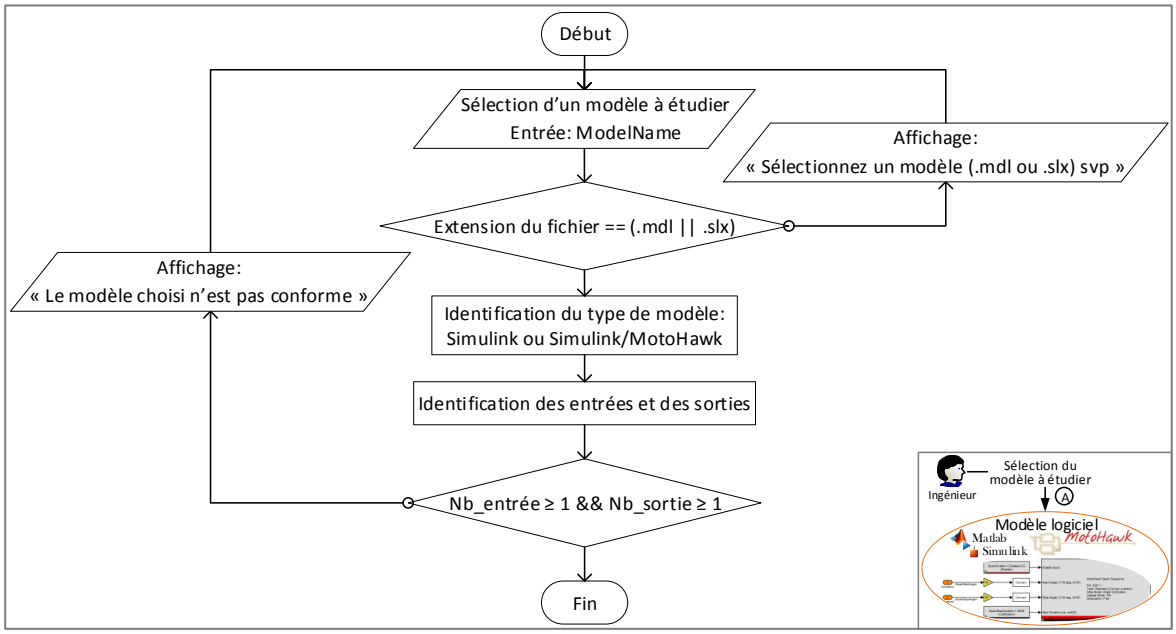


FIGURE 3.3 – Algorigramme : Vérification de la conformité du modèle de logiciel embarqué et identification de son type

La première vérification consiste à contrôler le type de fichier qui a été sélectionné.

Les modèles logiciels développés avec les versions de Simulink avant 2012 sont de type « .mdl », après ce sont des fichiers « .slx ». Si le fichier sélectionné correspond à l'un des deux formats alors une identification plus détaillée du type de modèle et de ses entrées/-sorties est opérée. D'une part, il est détecté si le modèle logiciel à étudier ne contient que des blocs Simulink. Dans ce cas, les entrées et sorties du modèle logiciel à identifier sont les blocs de type « Inport » et « Outport » qui se trouvent à la racine du modèle. D'autre part, s'il s'agit d'un modèle composé à la fois de blocs Simulink et MotoHawk alors les entrées et sorties n'ont pas de position particulière dans le modèle. Elles sont identifiées en comparant chacun des blocs du modèle à une liste préalablement établie des types de blocs d'entrée et de sortie. Une fois que le processus d'identification des entrées et sorties du modèle est terminé, on vérifie que le modèle de logiciel embarqué contient au moins une entrée et une sortie. Dans le cas contraire, le modèle sélectionné n'est pas celui d'un logiciel embarqué, alors l'analyse de SdF ne peut pas être conduite sur ce modèle.

Ces vérifications contrôlent les critères requis sur le modèle pour assurer le bon déroulement des opérations qui suivent dans le processus d'automatisation.

3.2.2 Architecture et remplissage de la base de données SdF

Les informations récoltées lors de l'analyse individuelle des blocs ❶ sont capitales pour l'obtention de résultats de qualité par la méthodologie ALEBAS. Elles constituent des données d'entrée indispensables à l'outil d'automatisation. Pour stocker ces informations et les rendre accessibles, il a été choisi de construire une base de données au format XML. Le schéma XML a été conçu spécifiquement pour contenir ces informations nécessaires à l'analyse de SdF dans le cadre de la méthodologie proposée. Les données obtenues à partir de l'analyse individuelle des blocs et qui doivent donc être stockées dans la base de données SdF sont : les équations de dysfonctionnement et les relations entre « modes de défaillance / causes / recommandations ». Le schéma XML développé pour ranger ces informations est présenté en figure 3.4.

Le remplissage de la base de données est indépendant de l'application de l'analyse de SdF ; il repose sur l'application d'une analyse individuelle sur chaque composant basique. La base de données nécessite d'être préalablement remplie avant de pouvoir être utilisée pour l'analyse de SdF des modèles logiciels. Le remplissage de la base de données n'est à réaliser qu'une seule fois, elle est ensuite réutilisée à chaque analyse de SdF quel que soit le modèle de logiciel embarqué. Une mise à jour de la base de données est simplement requise dans le cas où de nouveaux blocs apparaissent dans les nouvelles versions des librairies ou si des blocs évoluent.

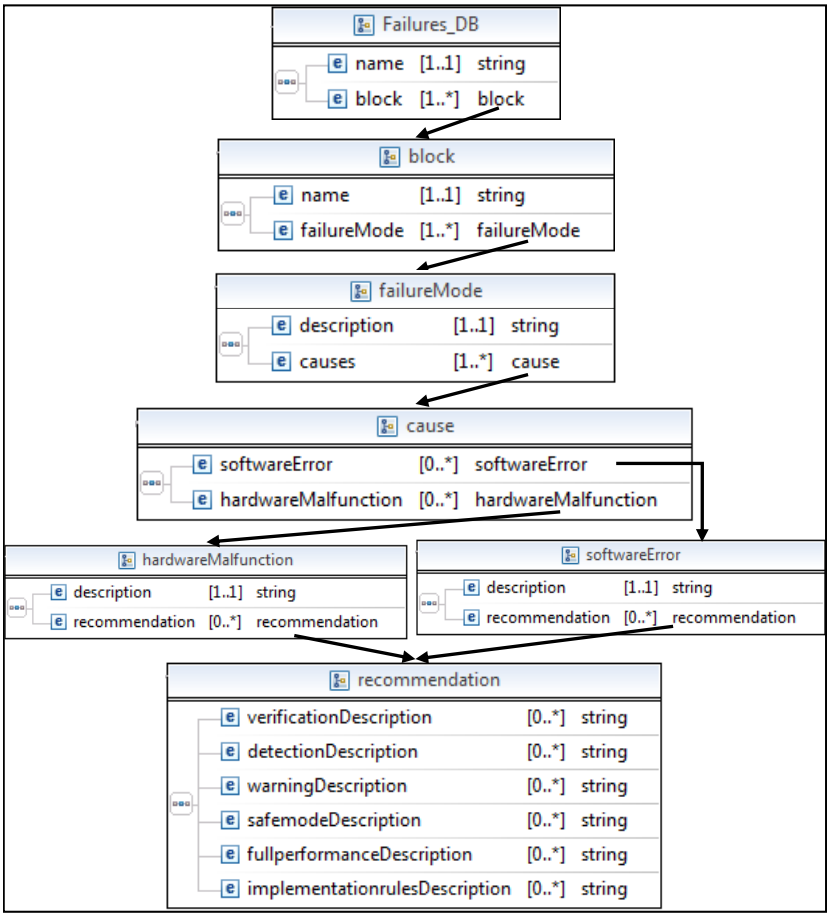


FIGURE 3.4 – Schéma XML de la base de données

3.3 Extraction des données du modèle

La première étape de notre processus d'automatisation consiste à extraire Ⓑ (figure 3.2) du modèle logiciel à étudier, les données nécessaires pour générer par la suite les AdD et le tableau AMDE. Un modèle logiciel développé sous Simulink/MotoHawk contient beaucoup d'informations. Dans le cadre d'une analyse de SdF, l'intérêt se porte sur la topologie du modèle. La figure 3.5 est le méta-modèle qui décrit les informations qu'il est nécessaire d'extraire du modèle Simulink/MotoHawk pour connaître sa topologie. Globalement, les informations extraites permettent d'identifier le modèle étudié, les blocs qui le composent et de connaître la topologie du modèle par l'intermédiaire des relations qui existent entre chacun des ports de l'ensemble des blocs.

La figure 3.6 est l'algorigramme décrivant le processus d'extraction des données du modèle à étudier. La première étape consiste à créer et initialiser la structure selon le méta-modèle (3.5) pour qu'elle puisse ensuite être complétée avec les données du modèle, des blocs et des liens entre les blocs. Les premières informations récupérées concernent l'identification du modèle : son nom, son chemin d'accès et le chemin d'accès au dossier qui le contient. Ensuite, une première caractéristique sur la topologie est récupé-

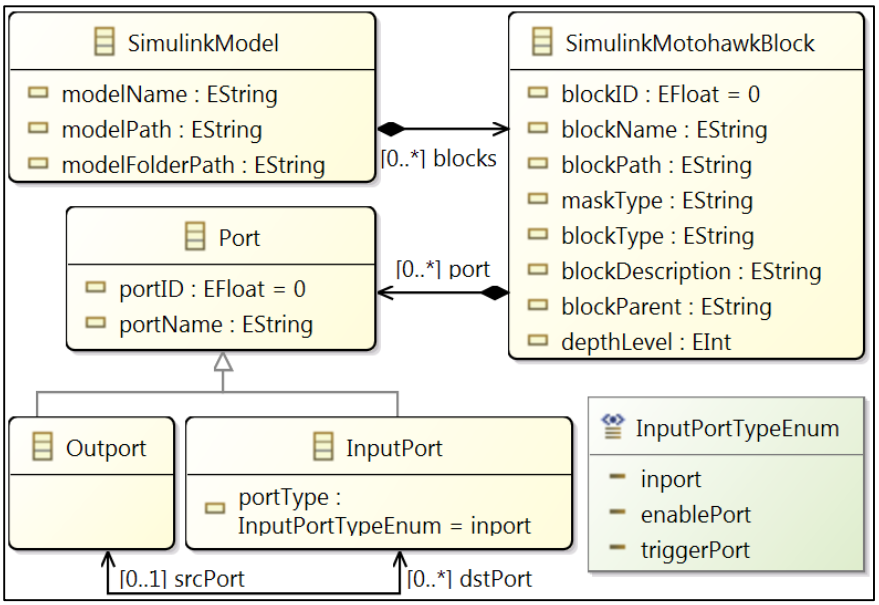


FIGURE 3.5 – Extrait du méta-modèle Simulink contenant les données à récupérer

rée, il s'agit de la profondeur maximale du modèle c'est-à-dire le niveau le plus bas dans l'arborescence des sous-systèmes inclus dans le modèle. Puis, tous les noms des blocs sont ajoutés à la structure afin de pouvoir les identifier, et ainsi les parcourir un par un pour récupérer les informations complémentaires suivantes : l'identifiant, le nom, le chemin dans le modèle, le type de bloc, le type de masque, la description, les sous-systèmes parents et enfin la position en profondeur dans le modèle. Pour chacun des ports de chaque blocs, leur identifiant et leur nom sont récupérés. On note une particularité pour les ports d'entrée ; en effet, il en existe de différents types :

- « Inport » : ports par lesquels transitent les flux de données
- « EnablePort » : ces ports reçoivent des flux conditionnés qui vont activer/désactiver l'exécution du sous-système qui contient ce port. Tant que la condition est vraie, le sous-système est exécuté.
- « TriggerPort » : ces ports reçoivent des flux conditionnés qui vont activer/désactiver l'exécution du sous-système qui contient ce port. Le sous-système n'est exécuté que lors du changement de statut de la condition, c'est-à-dire lorsqu'elle passe de « Vrai » à « Faux » ou l'inverse.

Cette phase d'extraction des données du modèle permet de préparer le terrain pour les tâches de création des AdD et de l'AMDE. Elle permet de travailler avec une structure plus simple que celle d'origine du modèle et surtout elle ne contient que les données nécessaires pour l'analyse.

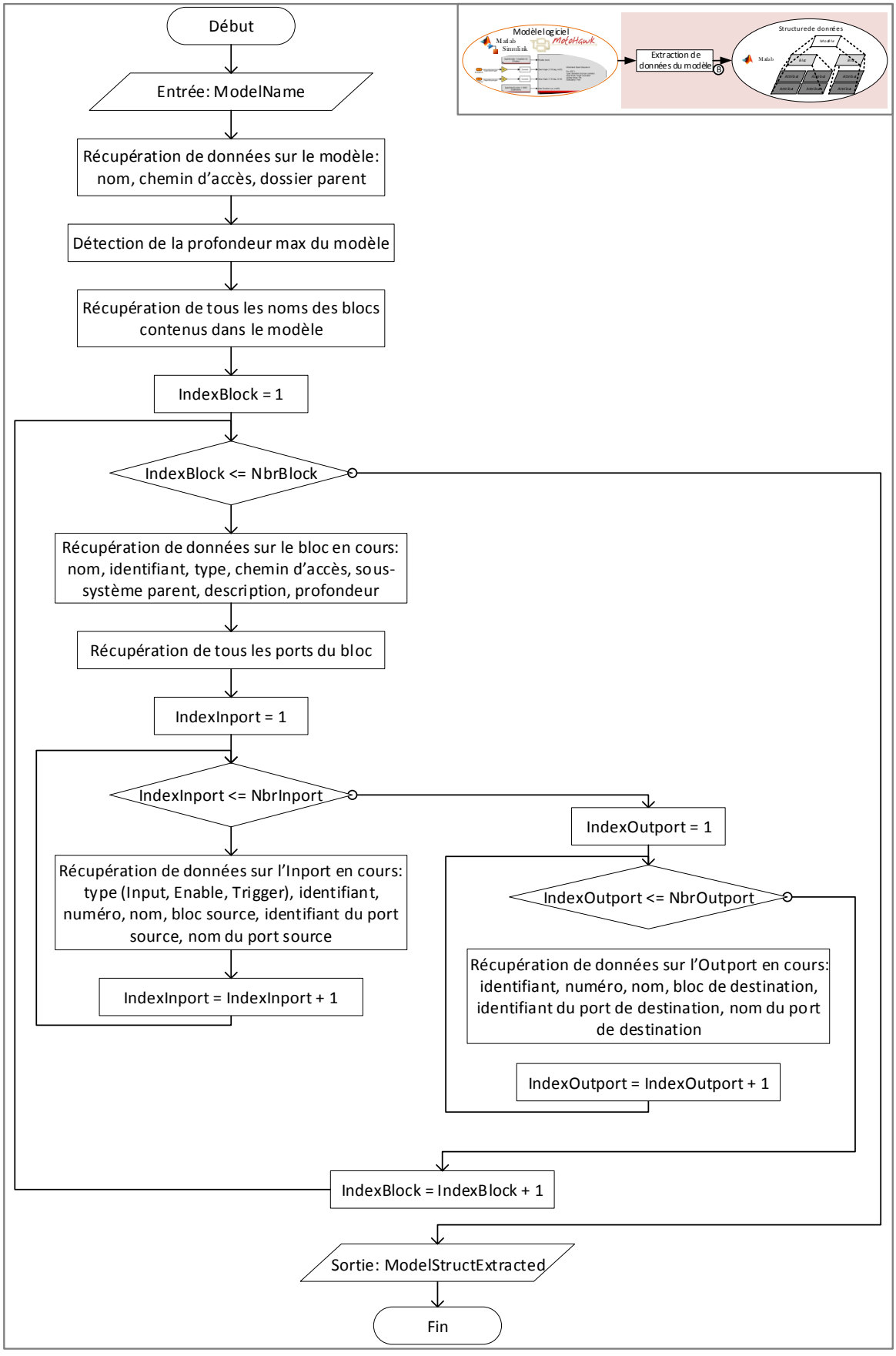


FIGURE 3.6 – Algorigramme : Extraction de données du modèle

3.4 Paramétrage de l'analyse

L'outil propose à l'utilisateur d'appliquer quelques paramètres Ⓒ (figure 3.2) s'il le souhaite, afin de limiter le périmètre de son analyse. Le premier paramètre permet de limiter la profondeur de l'analyse dans le modèle car l'utilisateur peut considérer qu'à partir d'un certain seuil de profondeur, les fonctions qui se trouvent en dessous, sont relativement simples et donc ne sont pas nécessairement à prendre en compte dans l'analyse de SdF. Le second paramétrage consiste à pouvoir écarter de l'analyse des sous-systèmes. Certains sous-systèmes peuvent avoir déjà été examinés lors de précédentes analyses sur ce modèle ou peuvent aussi être des sous-systèmes réutilisés, provenant d'autres modèles qui ont déjà été analysés et éprouvés. Ainsi, l'utilisateur peut choisir de les écarter de cette nouvelle analyse pour alléger les résultats.

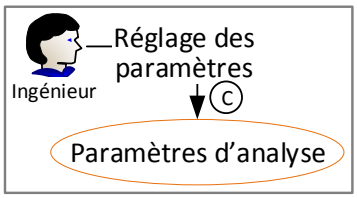


FIGURE 3.7 – Partie du processus d'automatisation pour le paramétrage de l'analyse

3.5 Automatisation de la construction des AdD

L'obtention des AdD et du tableau AMDE repose avant tout sur l'étude de la propagation, réciproquement déductive et inductive, d'erreur dans le modèle logiciel.

À ce stade, les données du modèle sont stockées dans une structure Matlab. Elle contient les informations d'identification du modèle et des blocs, ainsi que les relations qui existent entre les blocs pour connaître la topologie du modèle. Ces données ne sont pas classées de façon organisée, elles ont été rangées aléatoirement dans la structure. Le premier objectif est de réorganiser cette structure, afin que les informations sur la topologie ne soit pas simplement stockées textuellement mais que l'architecture même de la structure rende compte de cette topologie du modèle. L'organisation des données sous forme d'arborescence permet de rendre compte de la topologie du modèle. Sous Matlab, il n'existe pas de solution native pour représenter des arborescences et les manipuler. Une classe a été développée par M. Tinevez [1] pour combler ce manque. Cette classe permet d'implémenter des structures de données sous forme d'arborescence et est accompagnée de commandes facilitant l'exploitation de ce type de structure.

1. https://tinevez.github.io/matlab-tree/index.html

3.5.1 Navigation ascendante

La navigation ascendante (2A) (figure 3.2) consiste à traverser le modèle logiciel en remontant des sorties vers les entrées et en suivant les liens entre les composants. Une arborescence est créée pour chaque sortie du modèle logiciel. Cette arborescence permet de connaître tous les chemins en amont qui conduisent à une sortie. L'algorigramme décrivant la navigation ascendante est présenté en figure 3.8. Pour une analyse complète de tout le modèle, il est nécessaire d'étudier toutes les sorties. Mais si l'utilisateur souhaite cibler son analyse sur certaines sorties, l'outil propose à l'utilisateur de choisir les sorties qu'il tient à analyser. Après la sélection des sorties à étudier, chaque arborescence est construite de la manière suivante. Le sommet de l'arborescence est une sortie du modèle qui a été identifiée précédemment lors de l'extraction des données du modèle. Pour compléter les niveaux suivants de l'arborescence, lorsqu'un bloc est ajouté, on s'intéresse aux ports d'entrée de ce bloc pour connaître le bloc source. Il existe différents types de lien entre les blocs qui nécessitent des traitement différents pour suivre la propagation.

Les liens classiques physiques : dans ce cas, un port d'entrée n'a qu'un seul bloc source, l'identification du bloc source est connue dans les données récupérées lors de l'extraction.

Les sous-systèmes : il s'agit d'un cas particulier car lorsqu'un bloc sous-système est rencontré, il est nécessaire d'aller étudier les blocs qui se trouvent à l'intérieur, sauf si celui-ci à été écarté de l'analyse lors du paramétrage par l'utilisateur. On ne passe donc pas directement de ses ports de sorties à ses ports d'entrées, mais il faut identifier clairement le port de sortie par lequel on est arrivé pour ensuite identifier le bloc « Outport » correspondant, se trouvant à l'intérieur du sous-système. Une fois cette correspondance réalisée, la propagation se poursuit classiquement de ports en ports jusqu'à rencontrer un bloc « Inport », indiquant la sortie du sous-système. Comme pour l'« Outport », il est cette fois bien nécessaire d'identifier le bloc « Inport » et de connaître le port d'entrée correspondant du bloc sous-système parent, afin de pouvoir poursuivre la propagation.

Pas de lien physique : les données sont distribuées entre les blocs sans lien visible, par l'intermédiaire de la mémoire du système (de l'ECU dans l'embarqué). Les blocs concernés sont par exemple le « Data Write/Data Read » pour la librairie MotoHawk. Dans ce cas, lorsque l'on remonte des sorties vers les entrées, le premier bloc rencontré est celui de lecture « Data Read », il est alors nécessaire d'identifier tous les blocs « Data Write » associés, c'est-à-dire ceux qui modifient la même variable transportée pour poursuivre la traversée du modèle.

Après chaque étude d'un bloc, celui-ci est marqué c'est-à-dire que l'un de ces champs dans sa structure est « IsAnalyzed » et la valeur de ce champ est mis à un. Ainsi, dans le cas où des boucles sont présentes dans le modèle logiciel, il ne faut parcourir une boucle qu'une seule fois, sinon la construction de l'arbre rentrerait dans une boucle infinie. C'est

pour cette raison, qu'en début de construction d'un arbre, tous les marqueurs sont remis à zéro pour au contraire cette fois être sûr de bien passer par tous les chemins. L'arborescence est complétée jusqu'à ce qu'il n'y ait plus de blocs ajoutés à l'arborescence, car les blocs d'entrée du modèle ont tous été atteints.

Pour résumer, une arborescence ascendante est composée au sommet d'un bloc de sortie du modèle logiciel, les nœuds intermédiaires de l'arbre sont des blocs intermédiaires du modèle, enfin, les feuilles de l'arbre sont des entrées du modèle logiciel.

3.5.2 Construction des arbres de défaillances pertinents

La figure 3.9 présente l'algorigramme qui permet d'obtenir les AdD pertinents. Pour construire les AdD pertinents (2B) (figure 3.2), l'arborescence ascendante précédemment obtenue est parcourue de haut en bas. Pour le premier bloc au sommet de l'arbre ascendant, qui est aussi une sortie du modèle logiciel, ces informations de comportement en condition de défaillances sont récupérées dans la base de données SdF. Chacun de ses modes de défaillance devient une racine (sommet) de nouveaux arbres, qui vont ensuite devenir, en les complétant, les AdD pertinents. Ensuite, pour compléter un arbre de défaillance, les causes potentielles internes d'apparition du mode de défaillance pour ce bloc sont récupérées dans la base de données et ajoutées à l'AdD. Puis, l'apparition de ce mode de défaillance peut être due aux données qui arrivent en entrée de ce bloc, donc on remonte l'arborescence ascendante afin d'étudier les blocs en amont. En fonction du mode de défaillance étudié, l'arborescence est remontée en suivant les connexions et les équations de dysfonctionnement décrite dans la base de données. À chaque fois qu'un nœud est rencontré dans l'arborescence ascendante (un nœud correspond à un composant basique), et qu'au moins une de ces équations de dysfonctionnement est en lien avec les modes de défaillance du bloc prédécesseurs, alors ces modes de défaillance sont ajoutés à l'AdD. Ce processus se poursuit jusqu'à atteindre chacune des feuilles de l'arborescence ascendante, à moins que les équations de dysfonctionnement indiquent de ne pas poursuivre la propagation. Ces AdD sont construits sous forme de structures arborescente dans Matlab. Toutes les informations sont stockées dans ces arbres mais leur lisibilité est difficile. Un affichage graphique des AdD est nécessaire pour faciliter leur exploitation.

3.5.3 Conversion des AdD Matlab en XML

Pour obtenir une meilleure visualisation des AdD, la chaîne d'outils de la méthodologie ALEBAS offre la possibilité de transformer les structures arborescentes de Matlab en fichier XML. Ce fichier XML répond au schéma spécifique Open-Probabilistic Safety Assessment (Open-PSA). Open-PSA définit un formalisme XML permettant de décrire un

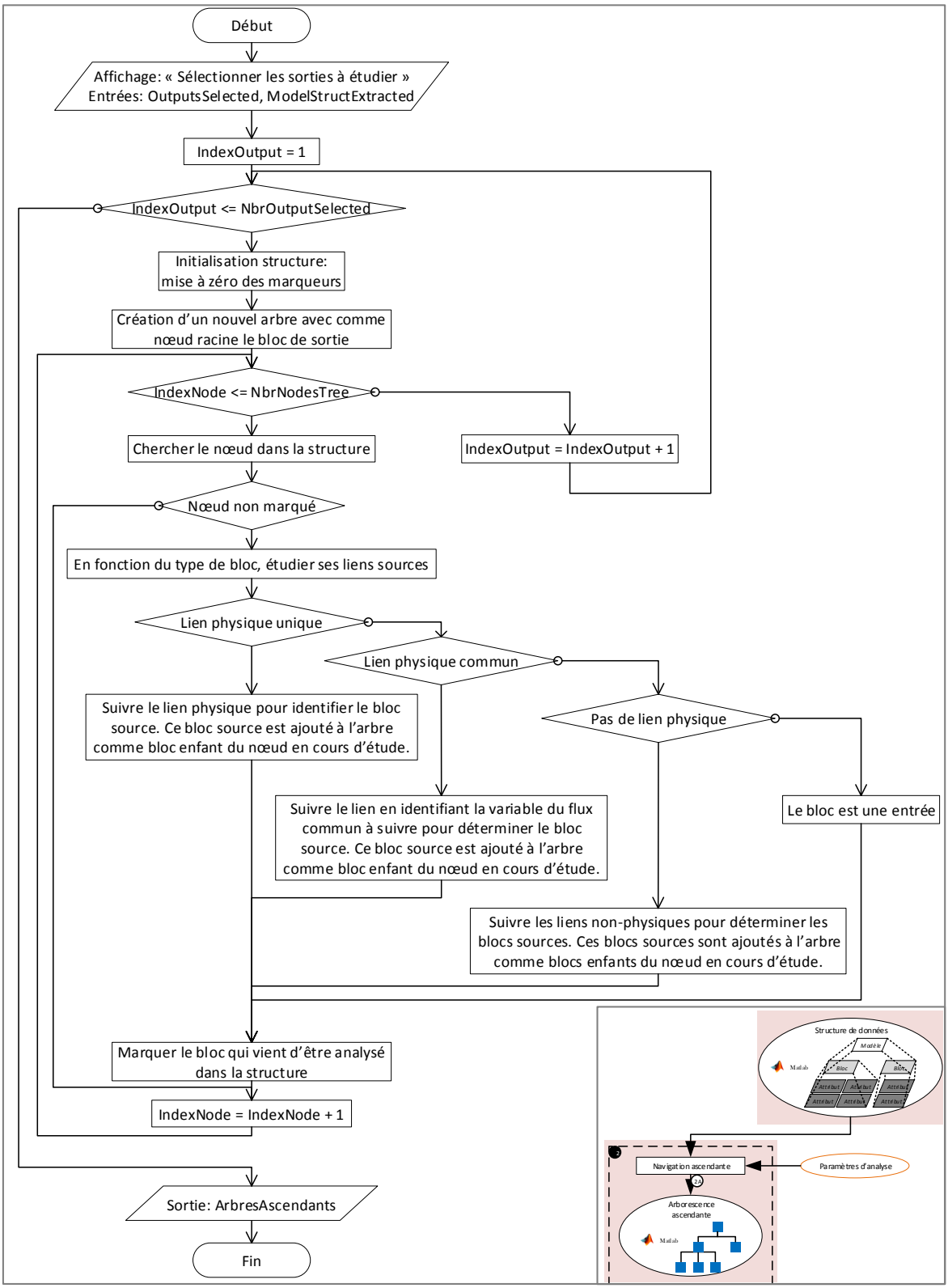


FIGURE 3.8 – Algorigramme : Navigation ascendante

arbre de défaillances. Il a pour but une interopérabilité complète. Ce type de fichier peut être lu par le logiciel Arbre Analyste [Cle, CTR14]. Ce logiciel est dédié à la construction d'AdD et propose des algorithmes de calcul de fiabilité. Dans notre cas, l'utilisation du logiciel se limite à un moyen de visualisation graphique des AdD qui ont été construits automatiquement à l'aide de notre chaîne d'outils. La conversion est réalisée en parcourant les AdD qui viennent d'être construits. Tous les nœuds des arbres sont parcourus de haut en bas. Pour chaque nœud, les informations sur le mode de défaillance qui y est contenu sont ajoutées dans la balise adéquate du fichier XML. L'algorigramme n'est pas

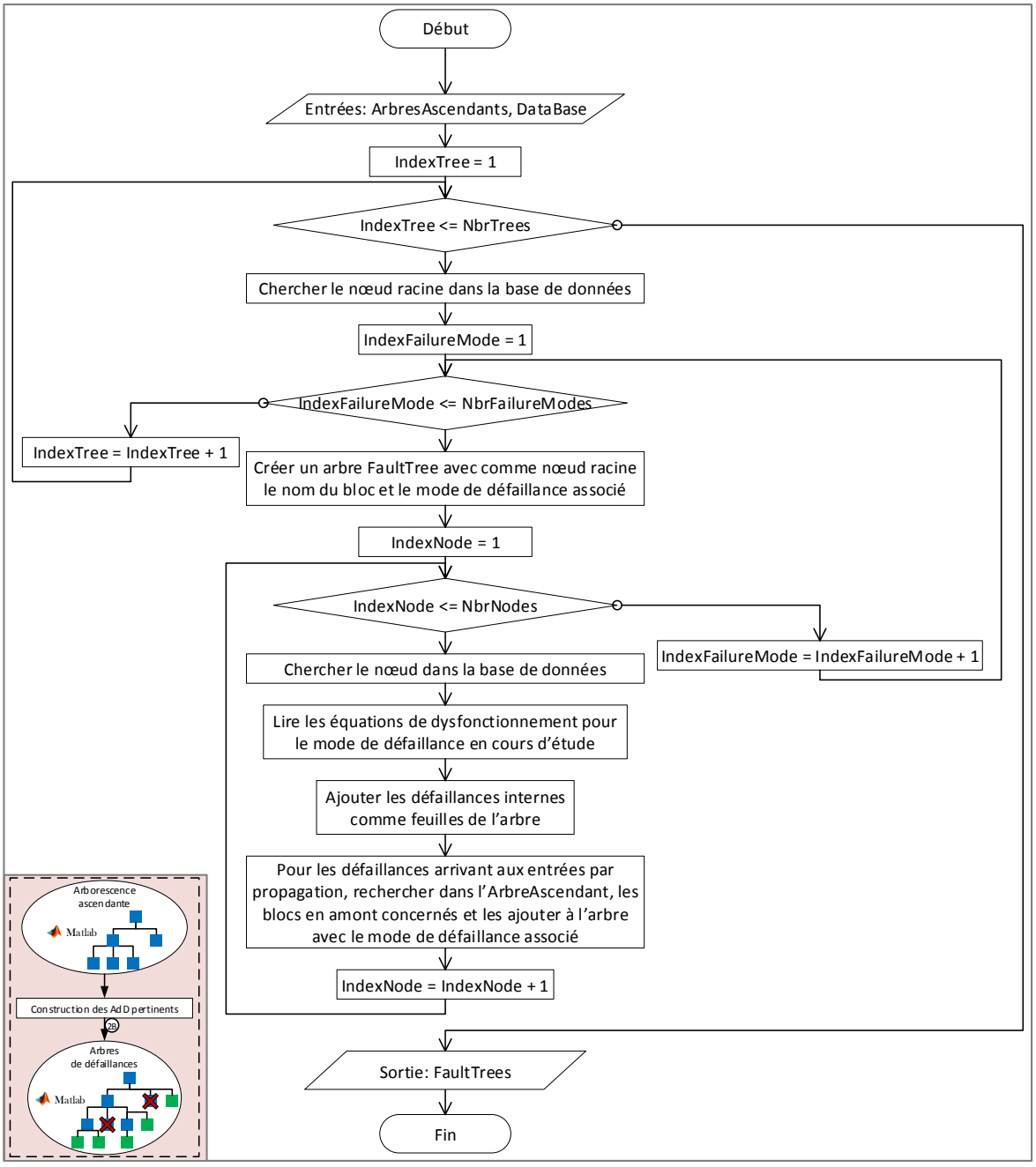


FIGURE 3.9 – Algorigramme : construction des arbres de défaillances pertinents

détaillé ici, car cette étape n'apporte pas de valeur ajoutée aux arbres ; il ne s'agit que d'un changement de format.

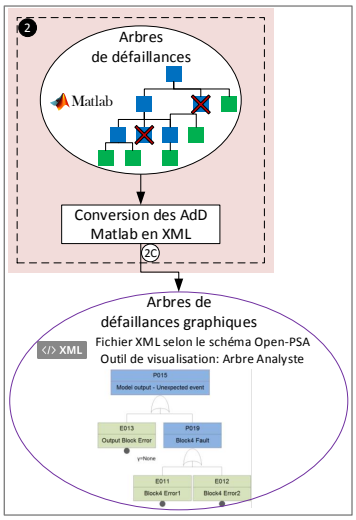


FIGURE 3.10 – Partie du processus d'automatisation pour le paramétrage de l'analyse

3.6 Automatisation du remplissage du tableau de l'AMDE

Parallèlement et indépendamment de la construction des AdD, ALEBAS implique l'application d'une AMDE sur le modèle de logiciel embarqué. Pour les mêmes raisons que la construction des AdD, c'est-à-dire la charge de travail importante et la source d'erreur pour mettre à jour régulièrement les analyses de SdF du fait des évolutions fréquentes du modèle logiciel, la génération du tableau AMDE a été automatisée. Les sections suivantes présentent les opérations qui sont réalisées pour décomposer la tâche de génération du tableau de l'AMDE dans le processus d'automatisation.

3.6.1 Navigation descendante

La navigation descendante (3A) (figure 3.2) consiste à traverser le modèle des entrées vers les sorties. Une arborescence est construite pour chaque entrée du modèle. L'outil offre la possibilité à l'utilisateur d'écarter des entrées de l'analyse. La navigation descendante dans le modèle logiciel est très similaire à la navigation ascendante. La seule particularité est pour les liens physiques, dans ce sens de propagation, un port de sortie peut avoir plusieurs ports de destination.

3.6.2 Génération du tableau de l'AMDE

À partir des arborescences descendantes et de la base de données SdF, le tableau AMDE peut être généré (3B) (figure 3.2). Pour une bonne visibilité, la flexibilité et l'ac-

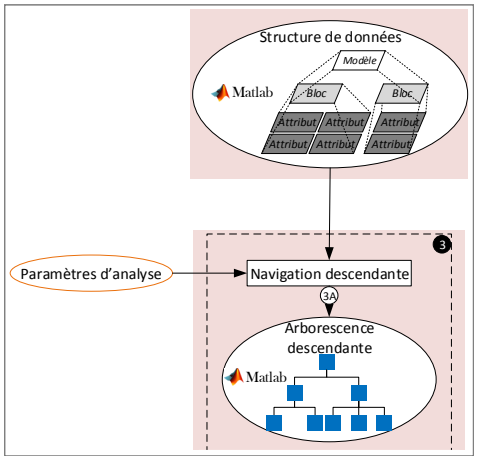


FIGURE 3.11 – Partie du processus d'automatisation pour la navigation descendante

cessibilité, le tableau AMDE est généré sous la forme d'un fichier HTML. Le fichier HTML est construit en parcourant les arborescences descendantes. L'algorigramme 3.12 décrit le processus de génération du tableau AMDE. Le processus commence par la construction de la structure du tableau. Ensuite, les arbres descendants sont parcourus pour compléter le tableau à partir des informations de chacun des nœuds. Lorsqu'un bloc est rencontré dans l'arborescence, son nom est ajouté dans la colonne « bloc », le nom de son sous-système parent dans la colonne « fonction ». Ensuite, ce bloc est cherché dans la base de données pour récupérer les informations sur ces modes de défaillance. Chacun des modes de défaillance de ce bloc sont ajoutés dans la colonne « Mode de défaillance » du tableau. Puis, pour chaque modes de défaillance, leurs causes sont ajoutées à la colonne « Cause » du tableau AMDE. Après, pour toutes les feuilles se trouvant sous le nœud en cours d'étude, il s'agit de blocs de sorties du modèle, leurs modes de défaillance sont récupérés dans la base de données et sont ajoutés dans la colonne « Potential effects on SW/ECU ». Le remplissage des colonnes « Potential effects on the system » et « Risk Level » n'est pas encore automatisé, cela nécessite d'avoir l'analyse système dans un formalisme particulier pour pouvoir y récupérer les données nécessaires. L'automatisation se poursuit tout de même avec le remplissage de la colonne « Recommandations » à l'aide de la base de données, dans laquelle chaque cause est associée à plusieurs recommandations. Une fois que tous les nœuds de tous les arbres ont été parcourus pour remplir, à l'aide de la base de données, les colonnes du tableau AMDE, alors sa génération est terminée.

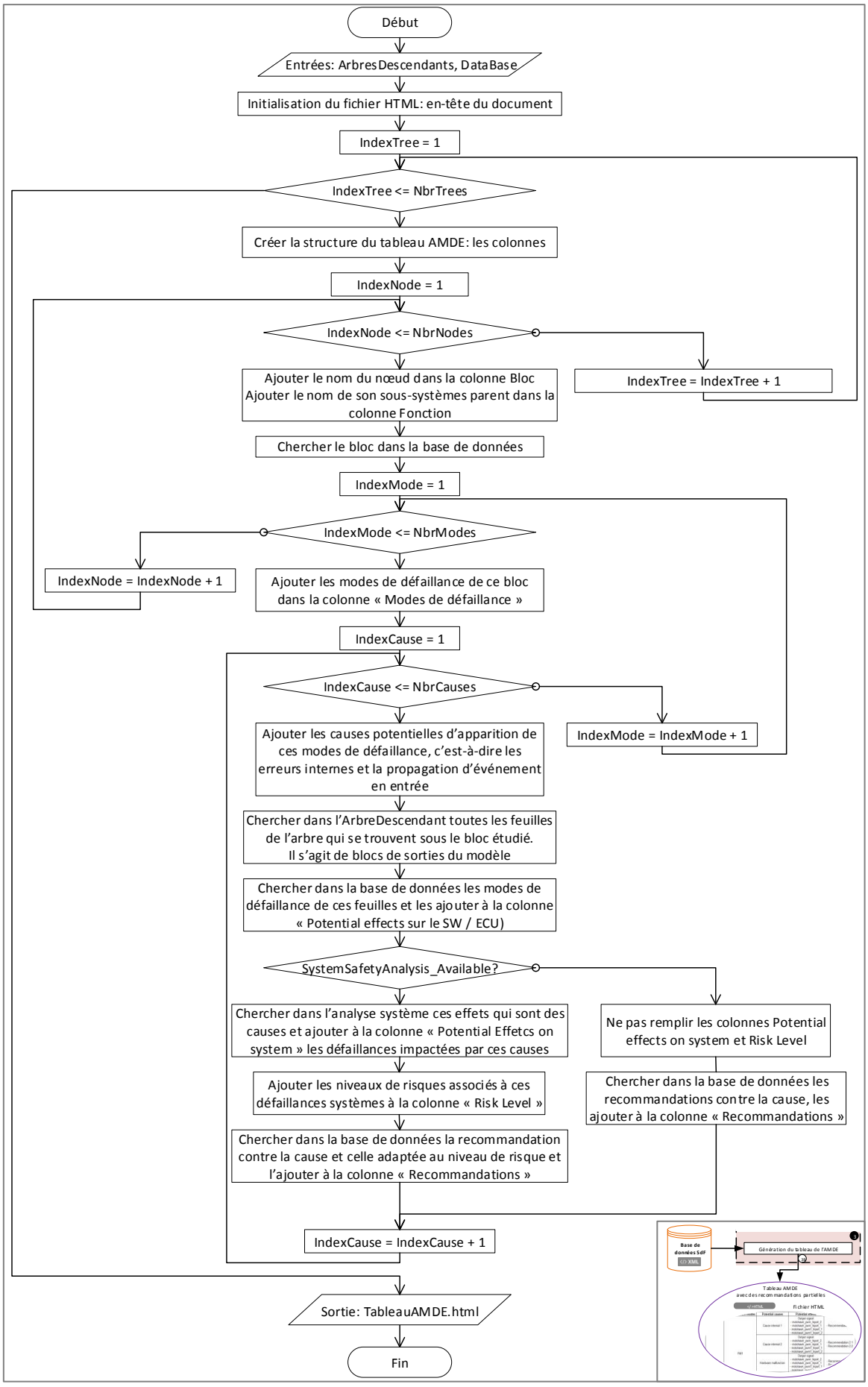


FIGURE 3.12 – Algorigramme : Génération du tableau de l'AMDE

3.7 Interface graphique de la chaîne d'outils

Pour faciliter la manipulation de la chaîne d'outils et le paramétrage des études, une interface graphique a été développée. L'interface graphique permet de suivre la progression de l'analyse. L'interface graphique de la chaîne d'outils est présentée en figure 3.13. L'étape 1 de l'interface graphique correspond à l'opération de sélection du modèle Ⓐ et l'extraction des données Ⓑ. Les étapes 2 et 3 servent au paramétrage de l'analyse Ⓒ avec le choix de la profondeur et des sous-systèmes à écarter. L'étape 4 coïncide avec la construction des AdD ❷, jusqu'à l'obtention de ceux-ci au format XML. Enfin, l'étape 5 permet la génération du tableau de l'AMDE ❸. Après chaque exécution d'une étape du processus, son statut passe à l'état « OK » et l'étape suivante peut être utilisée.

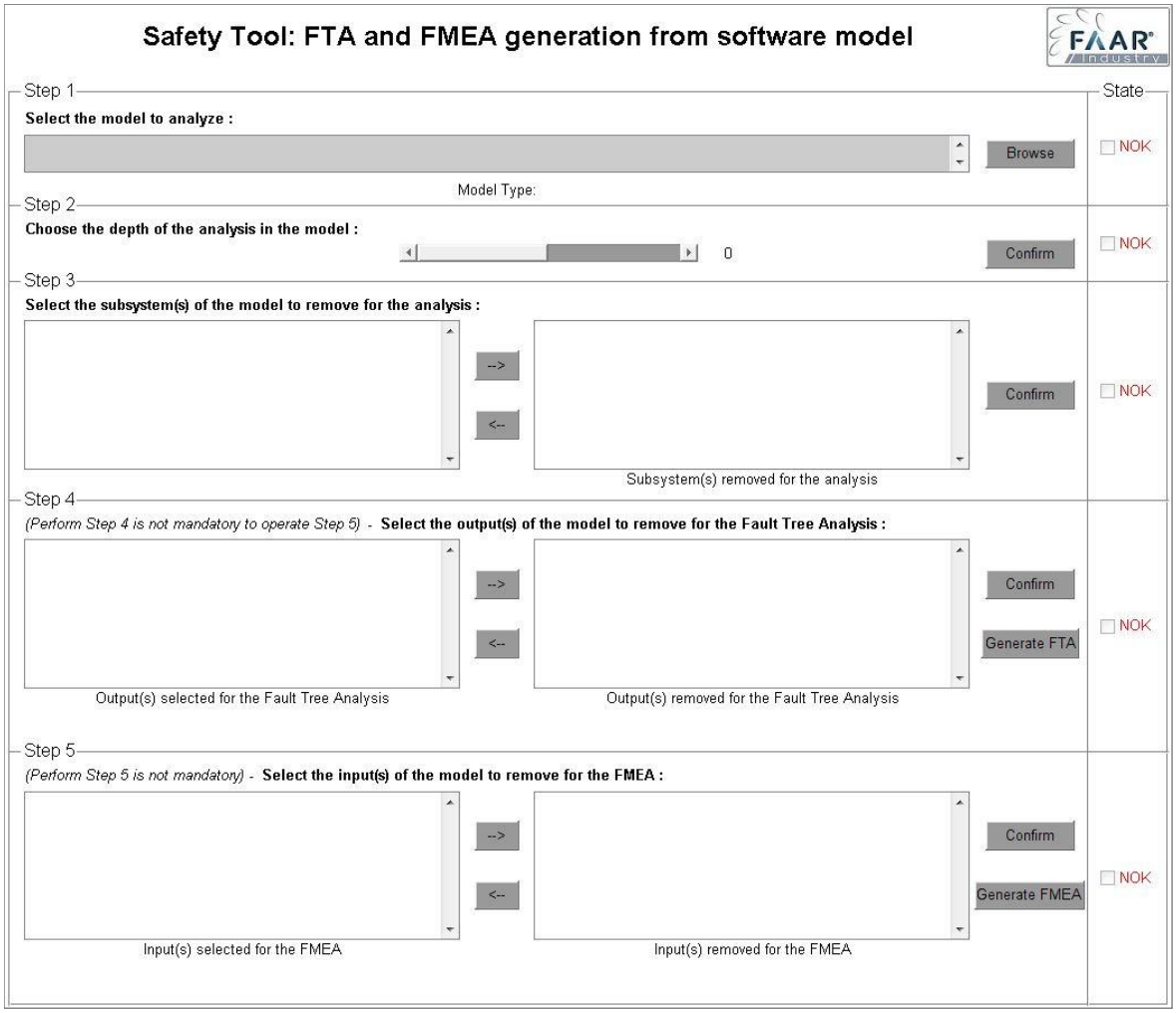


FIGURE 3.13 – Interface graphique de l'outil développé

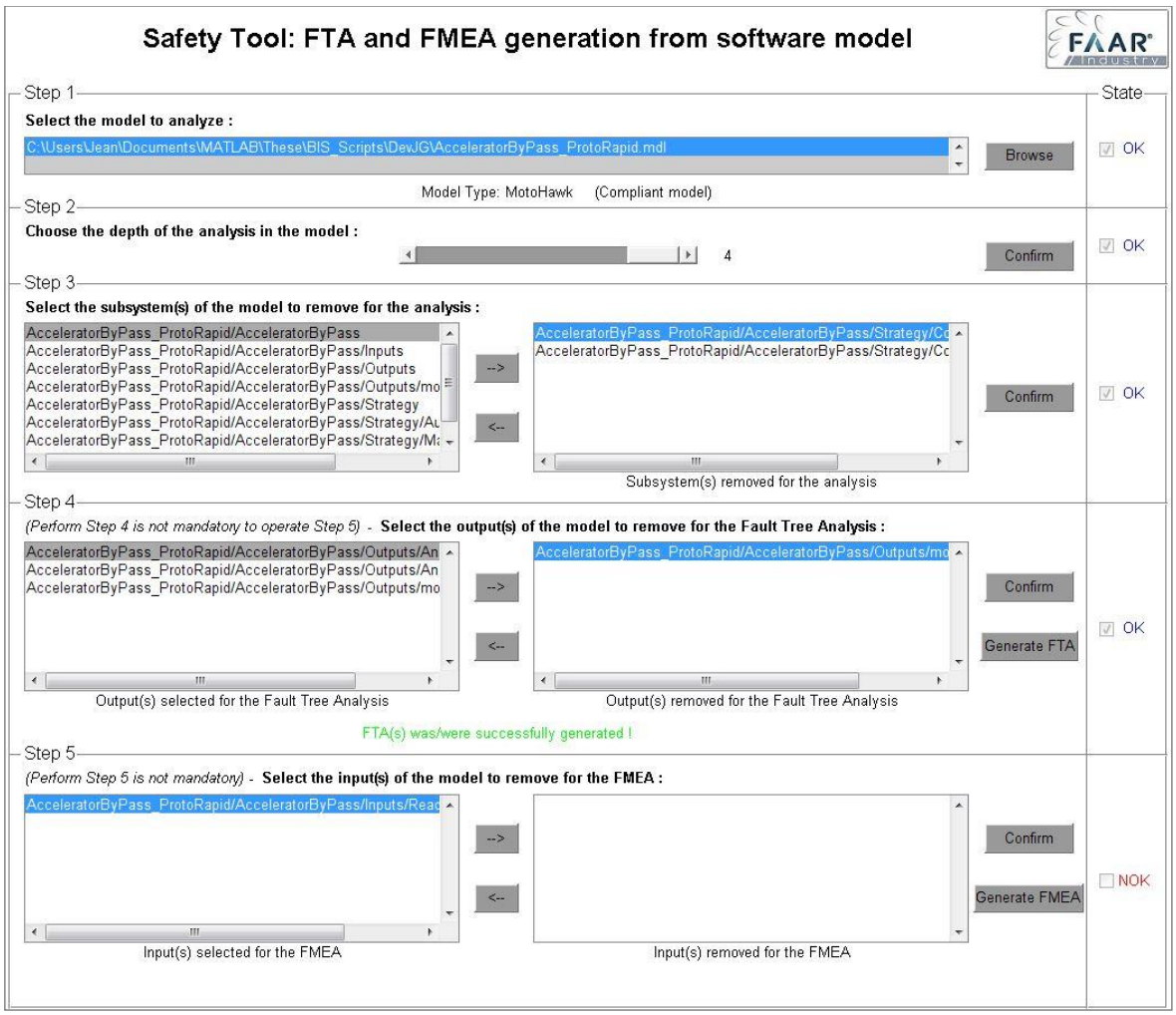


FIGURE 3.14 – Interface graphique de l'outil développé

3.8 Conclusion

L'application de la méthodologie ALEBAS permet d'améliorer les modèles de logiciel embarqué vis-à-vis de la SdF en appliquant des techniques d'analyse. Pour profiter pleinement de cette méthodologie qui nécessite certaines opérations complexes et parfois sources d'erreur, une automatisation est nécessaire pour faciliter son utilisation. De plus, dans un contexte de prototypage dans lequel le logiciel est amené à évoluer régulièrement, l'itération des analyses devient une tâche conséquente. L'automatisation allège le travail de l'ingénieur. La chaîne d'outil développée s'est concentrée sur l'automatisation des techniques de SdF. Ainsi, à partir du modèle de logiciel embarqué à étudier et une base de données de SdF préalablement complétée avec des informations sur chacun des composants basiques, les AdD et un tableau AMDE sont automatiquement générés. L'AMDE propose des recommandations partielles d'améliorations du modèle logiciel. Une interprétation des AdD et de l'AMDE par un ingénieur permet d'affiner et/ou de compléter les recommandations. Différents paramètres sont possibles avant de générer les AdD et l'AMDE pour flexibiliser l'étude et centrer les analyses sur certaines parties du modèle qui intéressent l'analyste.

Chapitre 4

Cas d'étude : Leurrage de la pédale d'accélérateur pour un prototype de véhicule autonome

Sommaire

Dans ce chapitre, la méthodologie ALEBAS est mise en pratique sur un cas d'étude réel et industriel. Ce cas d'étude est dédié au système de « leurrage » ou « bypass » en anglais de la pédale d'accélérateur, qui est utilisé dans des prototypes de véhicules autonomes. La présentation de ce cas d'étude a pour objectif de montrer dans quelles mesures la méthodologie ALEBAS peut être utilisée et les avantages qu'elles procurent notamment en observant les améliorations qui seront apportées au logiciel. La première partie de ce chapitre décrit le projet de prototypage de la robotisation d'un véhicule et les particularités et objectifs concernant le cas d'étude qui porte sur le leurrage de l'accélérateur. Cette partie présente la première version du logiciel prototype qui répond aux exigences fonctionnelles mais pour lequel la SdF n'a pas été traitée. La seconde partie de ce chapitre consiste à appliquer la méthodologie ALEBAS afin d'améliorer le modèle logiciel prototype en y ajoutant des stratégies de SdF et de diagnostics qui rendent le système plus sûr.

4.1 Prototypage de la robotisation d'un véhicule

En vue du salon « Intelligent Transport System » (ITS) de Bordeaux en 2015, FAAR Industry et l'Université Technologique de Belfort Montbéliard (UTBM) se sont associés afin de réaliser une première mondiale : la gestion du passage d'un carrefour par trois véhicules autonomes réels. Il a donc été nécessaire de préparer trois véhicules autonomes. Dans le cadre de la réalisation d'un prototype de véhicule autonome, le développement se fait très souvent à partir d'un véhicule de série existant pour des questions de coût. Sur un véhicule existant, il est nécessaire de prendre la main sur les trois fonctions principales de la conduite : l'accélérateur, le frein et la direction. L'objectif est d'être capable de commander ces fonctions, afin de pouvoir remplacer les actions du conducteur par des commandes électroniques. Ce stade du développement est la robotisation du véhicule. Ce projet a conduit à robotiser un véhicule de type Renault Grand Scenic de 2012 équipé d'un moteur 1.5 DCI avec une boîte de vitesse automatique (figure 4.1). Dans le cadre du cas d'étude visant à mettre en œuvre la méthodologie de développement proposée, seule la partie sur la robotisation de l'accélérateur de ce véhicule est abordée. Cette fonction suffit à montrer différents cas d'utilisation et différents résultats de la méthode. À la suite de ce projet, l'un des trois véhicules a été conservé par l'entreprise et sert de plateforme d'essai aussi bien pour des projets internes que pour des clients qui souhaitent tester leurs technologies. Ainsi, le cas d'étude de la thèse a porté sur la robustification du logiciel de contrôle commande en charge des organes de conduite : accélérateur, frein et direction.

FIGURE 4.1 – Prototype de véhicule autonome basé sur un Renault Grand Scénic sur lequel le cas d'étude a été réalisé

4.1.1 Accélérateur standard

Avant de décrire la robotisation de l'accélérateur, il est nécessaire de décrire le fonctionnement de la commande d'accélérateur dans sa configuration standard. L'architecture standard du système d'accélérateur est décrit en figure 4.2

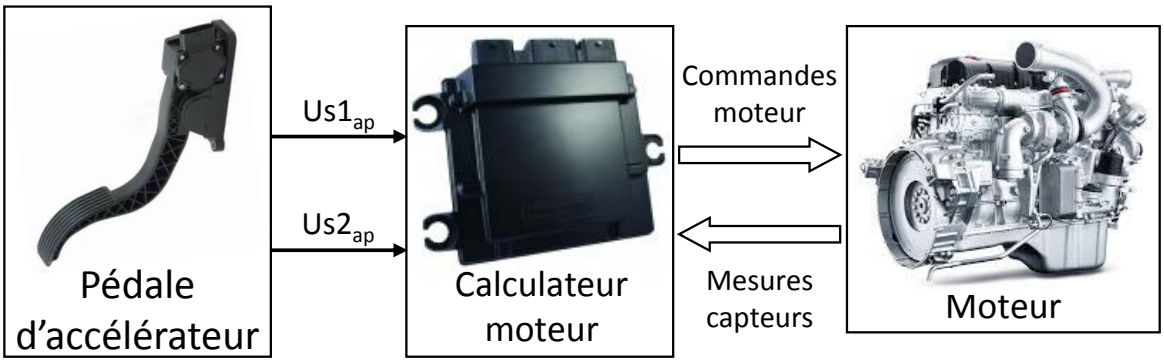


FIGURE 4.2 – Synoptique de l'architecture standard de la fonction accélérateur d'un véhicule

Dans la configuration standard de l'accélérateur, la requête du conducteur se fait par l'intermédiaire de la pédale d'accélérateur. Les pédales d'accélérateur n'ont aujourd'hui plus de liaison mécanique, elles transmettent des signaux électroniques au calculateur moteur. Elles se composent généralement de deux potentiomètres dont leurs signaux analogiques indiquent chacun la position de la pédale. Cette redondance des potentiomètres permet de vérifier l'intégrité du signal de position de la pédale d'accélérateur. Les potentiomètres de la pédale utilisée dans ce cas d'étude présentent la relation suivante : $U_{s1AP} = 2 * U_{s2AP}$ (AP : accélérateur pédale). La figure 4.3 présente ce ratio qui existe entre les deux potentiomètres et la relation entre la valeur des signaux analogiques des potentiomètres, en fonction de la position de la pédale d'accélérateur. Ainsi, lorsque la pédale évolue de 0 à 5% d'appui, on note que les signaux des potentiomètres n'évoluent pas ; il

s'agit de la garde de l'accélérateur. Puis de 5% à 100%, les deux potentiomètres croissent linéairement en conservant le rapport de deux entre eux. Les deux signaux de la pédale ont des plages de fonctionnement entre [0.4 ; 3.8]V et [0.2 ; 1.9]V. Ils n'utilisent pas toute la plage disponible de [0 ; 5]V pour permettre de distinguer des problèmes de déconnexion de fils ou de court-circuit qui donneraient les valeurs extrêmes 0V ou 5V par rapport aux intervalles de fonctionnement nominal.

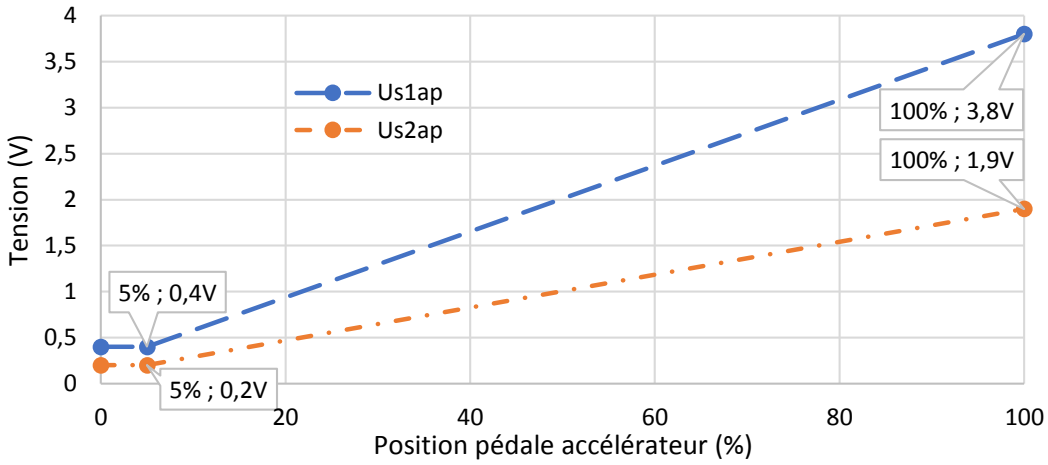


FIGURE 4.3 – Évolution de la tension des signaux analogiques des potentiomètres de la pédale d'accélérateur en fonction de la position de la pédale

Ensuite, ces signaux provenant des potentiomètres sont interprétés par le calculateur moteur qui reçoit également de nombreuses données d'autres capteurs, afin de calculer les bonnes commandes des actionneurs du moteur : injection, admission d'air, allumage...

4.1.2 Architecture du système de leurrage de l'accélérateur

Pour prendre la main sur l'accélérateur, plusieurs solutions sont possibles :

1. avoir accès au logiciel du calculateur moteur et le modifier en y ajoutant une stratégie permettant de recevoir des consignes d'accélérateur, soit via la pédale (mode standard), soit via un autre calculateur (mode autonome). Cette maîtrise et cet accès du calculateur ne sont possibles que par l'équipementier ou le constructeur automobile qui l'a développé.

2. remplacer le calculateur moteur par un calculateur dit « ouvert », c'est-à-dire que l'on peut reprogrammer. Cette solution nécessite de re-développer toute la stratégie de contrôle du moteur.

3. concevoir un actionneur mécatronique qui remplace l'action mécanique du conducteur sur la pédale.

4. introduire entre la pédale d'accélérateur et le calculateur moteur un module qui va pouvoir, soit laisser passer les signaux de la pédale en mode standard, soit les remplacer par d'autres consignes en mode autonome.

Dans notre cas, la solution n°4 a été sélectionnée. Il s'agit du meilleur compromis entre la faisabilité, la transparence d'action pour le conducteur, la discrétion de l'intégration et le coût.

Le synoptique du système de leurrage de la pédale d'accélérateur est présenté en figure 4.4.

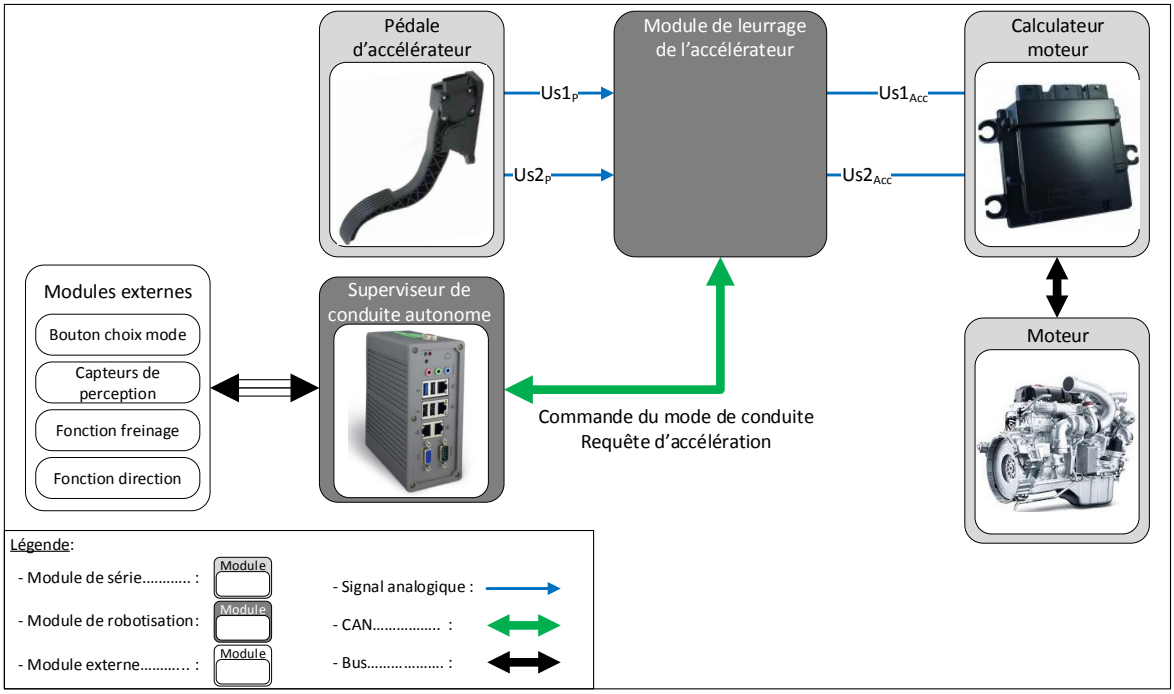


FIGURE 4.4 – Synoptique du système de leurrage de la pédale d'accélérateur

Dans cette architecture réalisant la fonction de leurrage de l'accélérateur, les composants de la configuration standard sont conservés et non modifiés : la pédale d'accélérateur, le calculateur moteur et le moteur. Le module de leurrage de la pédale d'accélérateur doit être capable d'aiguiller les signaux, soit de la pédale, soit de l'ordinateur autonome vers le calculateur moteur. Il doit être capable de reproduire les signaux de la pédale. Autour de cette fonction principale, d'autres sous-fonctions définies dans la spécification sont à réaliser :

- Le passage en mode autonome est commandé par le superviseur de conduite autonome via l'envoi d'une trame CAN à destination du module de leurrage

- Le retour au mode manuel est commandé, soit par le superviseur de conduite autonome via l'envoi d'une trame CAN à destination du module de leurrage, soit en cas

de détection d'un appui sur la pédale d'accélérateur (l'appui sur la pédale de frein n'est pas abordé dans ce cas d'étude)

Pour réaliser ces fonctions, deux modules ont été insérés pour aiguiller les commandes d'accélération, soit celles venant de la pédale d'accélérateur en mode standard, soit celles venant d'un autre calculateur. Le premier module ajouté est un calculateur qui est programmable avec la chaîne d'outils MotoHawk. Le second module est un module matériel composé de deux circuits indépendants et identiques. Ces circuits se composent d'un convertisseur de signaux Pulse Width Modulation (PWM) vers des signaux analogiques 0-5V, qui permet d'émuler les signaux pédales en mode autonome, et d'un relais permettant, soit de laisser passer les signaux pédale d'origine, soit de les remplacer par le signal analogique venant du convertisseur. Ce convertisseur assure une part de SdF compte tenu de sa conception. En effet, dans le cas de la perte de l'alimentation du boîtier ou de la non-réception du signal « Safety pulse », les signaux provenant de la pédale d'accélérateur sont passants, c'est-à-dire que le mode manuel est évidemment privilégié. Ces modules ont été choisis car il s'agit de modules sur étagère, ils n'ont donc pas nécessité un coût supplémentaire de conception et de développement.

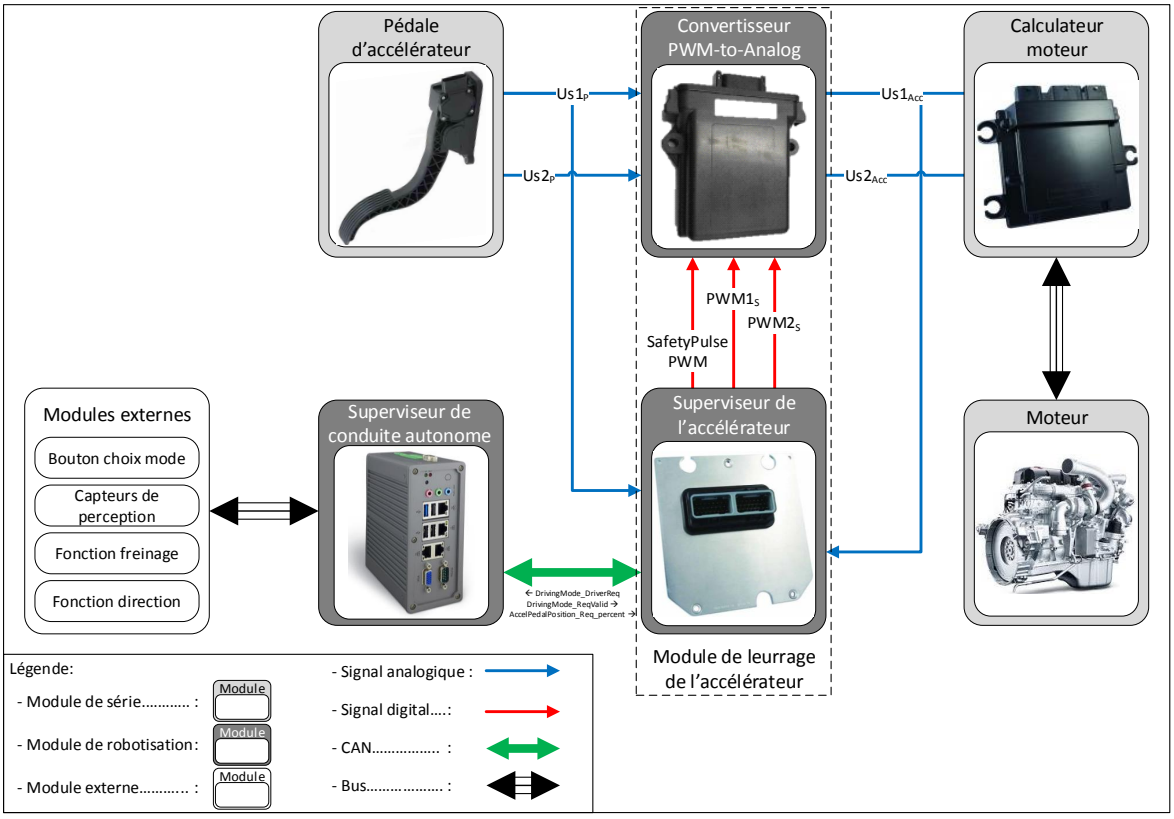


FIGURE 4.5 – Architecture du système de leurrage de la pédale d'accélérateur

La figure 4.5 décrit l'architecture du système de leurrage avec l'utilisation de ces deux composants. Dans cette architecture système, le superviseur de leurrage de l'accélérateur

est connecté à l'ordinateur principal du véhicule autonome (superviseur de conduite autonome). Ce superviseur de conduite autonome transmet au calculateur les commandes de changement de mode de conduite (manuel ou autonome) et, dans le cas du mode autonome, la requête d'accélération. Ensuite, le calculateur commande le convertisseur à l'aide du signal appelé « Safety pulse », afin soit de laisser passer les signaux de la pédale, soit d'appliquer d'autres commandes reçues par le calculateur via les liaisons PWM1 et PWM2. Enfin, les signaux de sortie du convertisseur sont transmis au calculateur moteur. Le superviseur reçoit aussi en entrée un signal de la pédale d'accélérateur permettant de connaître son état (appuyé ou non), afin de désactiver le mode autonome à l'aide de la pédale. Dans ce processus de prototypage, des premiers essais de ce module de leurrage ont été réalisés. Il a été décelé qu'une régulation des signaux simulés en mode autonome est nécessaire, afin de garantir que la requête d'accélération demandée par le superviseur de conduite autonome soit correctement effectuée par le module de leurrage, d'où l'ajout d'une relecture d'un des signaux simulé : Us1acc dans l'architecture du système.

Pour faciliter la conduite des essais sans avoir à travailler directement sur le véhicule, un banc de test a été réalisé afin de reproduire en partie le système. Ce banc de test est présenté en figure 4.6. Sur ce banc, le superviseur de conduite autonome est remplacé par un ordinateur de bureau (non visible sur la figure). Le calculateur moteur et le moteur du véhicule sont remplacés par un calculateur qui commande un moteur électrique. Chacune des liaisons filaires de ce banc de test possèdent des connecteurs intermédiaires permettant de simuler des pannes en les débranchant simplement ou en y injectant des signaux perturbateurs.

4.1.3 Logiciel prototype

Pour l'application de la méthodologie d'analyse de SdF, la situation initiale est le cas où au moins une première version du système de leurrage de l'accélérateur avec son logiciel a été développé, sans que la problématique de la SdF n'ait été traitée. L'architecture de ce modèle logiciel suit les bonnes pratiques proposées. On retrouve les quatre parties majeures : le superviseur, l'IHM, le contrôle-commande et le diagnostic. On ne traitera pas la partie concernant IHM dans cette présentation. Pour cette première version de logiciel, la partie « diagnostic » ne contient aucune stratégie (elle est vide) car le développement ne s'est pas préoccupé de la SdF. La figure 4.8 présente la partie du logiciel qui a été développée pour réaliser la supervision du système. La supervision du système de leurrage de la pédale d'accélérateur gère le comportement général c'est-à-dire ses différents états ou modes de fonctionnement.

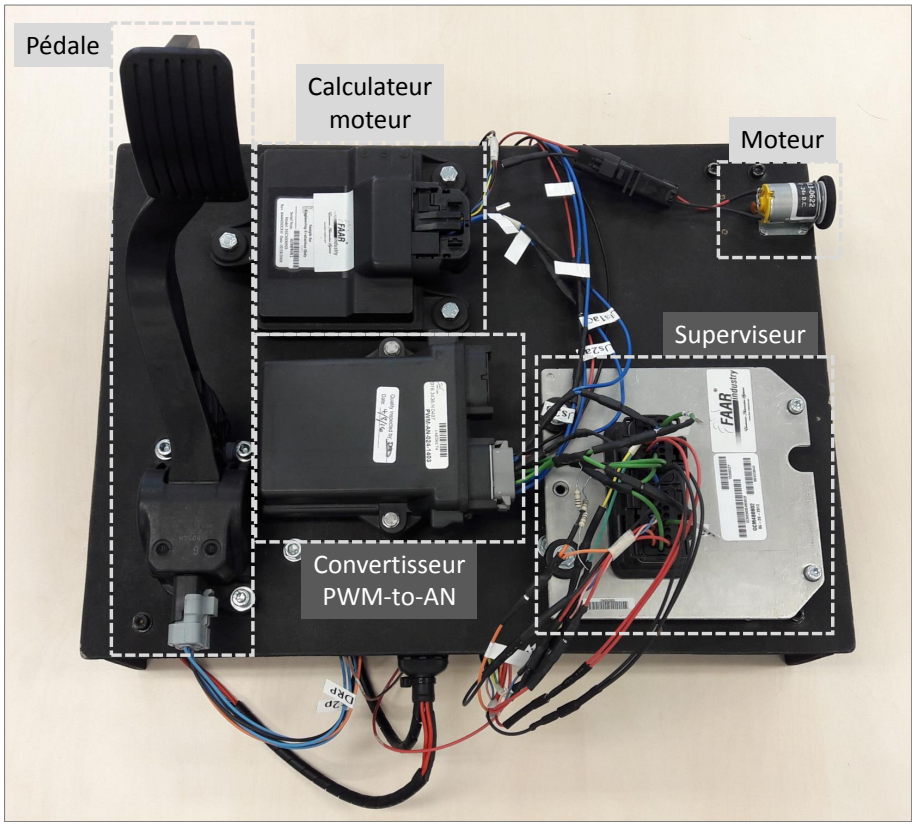


FIGURE 4.6 – Banc de test

4.1.4 Test du logiciel prototype

Le logiciel prototype présenté précédemment a fait l'objet d'une série de tests afin de contrôler son bon fonctionnement. D'un point de vue fonctionnel, les résultats sont concluants, toutes les fonctions définies dans la spécification sont réalisées correctement :

- La commande de l'accélérateur en mode manuel, c'est-à-dire à partir de la pédale, se comporte correctement.
- La commande de passage en mode autonome est reçue correctement par le module de leurrage et les signaux de la pédale sont bien remplacés par des signaux simulés.
- Les signaux simulés se comportent correctement, ils reproduisent bien les signaux et suivent les commandes d'accélération reçues par le superviseur de conduite autonome grâce à la boucle de régulation.
- La commande de retour en mode manuel envoyée par le superviseur de conduite autonome permet bien de reprendre le contrôle de l'accélérateur avec la pédale.
- En mode autonome, l'appui sur la pédale d'accélérateur permet bien de revenir en mode manuel.

Après la réalisation de ces tests visant à vérifier le comportement fonctionnel du système, d'autres essais sont conduits afin de vérifier la robustesse du système. Le logiciel prototype n'ayant pas été développé en prenant en compte les problématiques de SdF, il est clair que certains tests vont produire l'apparition d'événements redoutés, l'objectif ici est de les dévoiler afin de montrer plus tard après avoir appliquer la méthodologie ALEBAS qu'en appliquant ces mêmes tests alors les événements redoutés ne se produisent plus. Le calculateur moteur d'origine est déjà pourvu de solutions de diagnostic et de protection afin d'éviter des comportements dangereux. Ainsi, la plupart des tests de simulation de panne sont déjà tolérés, grâce à lui dans de nombreux cas, le système passe en mode dégradé. Certains essais ont tout de même conduit à l'apparition d'événements redoutés notamment :

- Lorsque le système est en mode autonome, si le signal de la pédale d'accélérateur est perdu en entrée du superviseur d'accélérateur, l'utilisateur ne peut pas repasser en mode manuel par ce biais, ainsi l'accélération est conservée selon la requête de le superviseur de conduite autonome.
- Si le signal Us1acc, relu par le superviseur d'accélérateur pour la boucle de régulation, subit des pertes de tension, des accélérations intempestives peuvent être provoquées.

Il est normal qu'à ce stade les performances de robustesse du système ne soient pas optimales car aucune précaution n'a été prise au niveau du module de leurrage. Ainsi, au lieu de redémarrer tout le développement en prenant en compte ces nouvelles contraintes, nous appliquons la méthode ALEBAS afin de capitaliser sur le travail réalisé et le logiciel existant, afin de déterminer les améliorations qui peuvent être apportées.

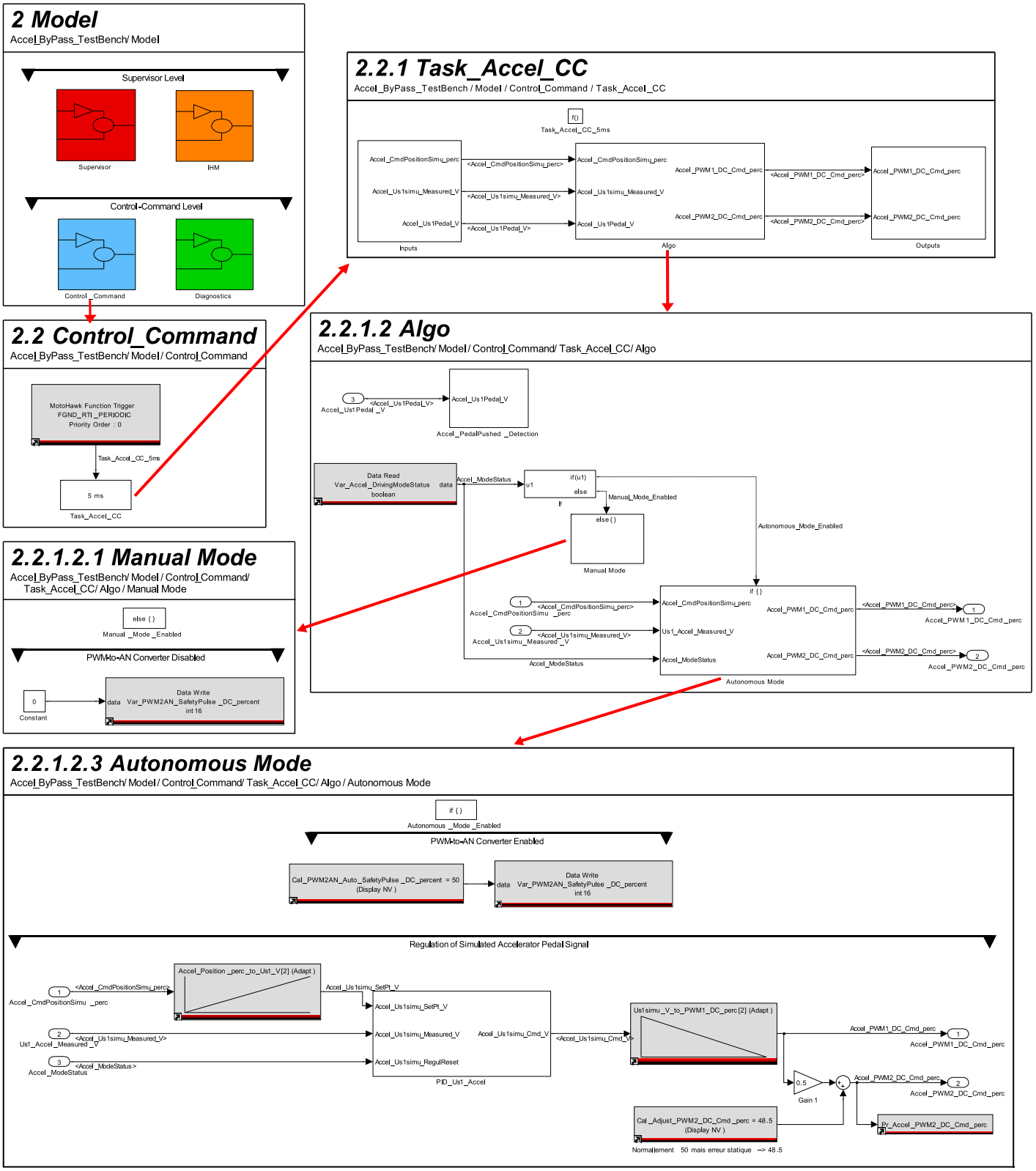


FIGURE 4.7 – Fonction de contrôle-commande du modèle logiciel du système de leurrage de l'accélérateur

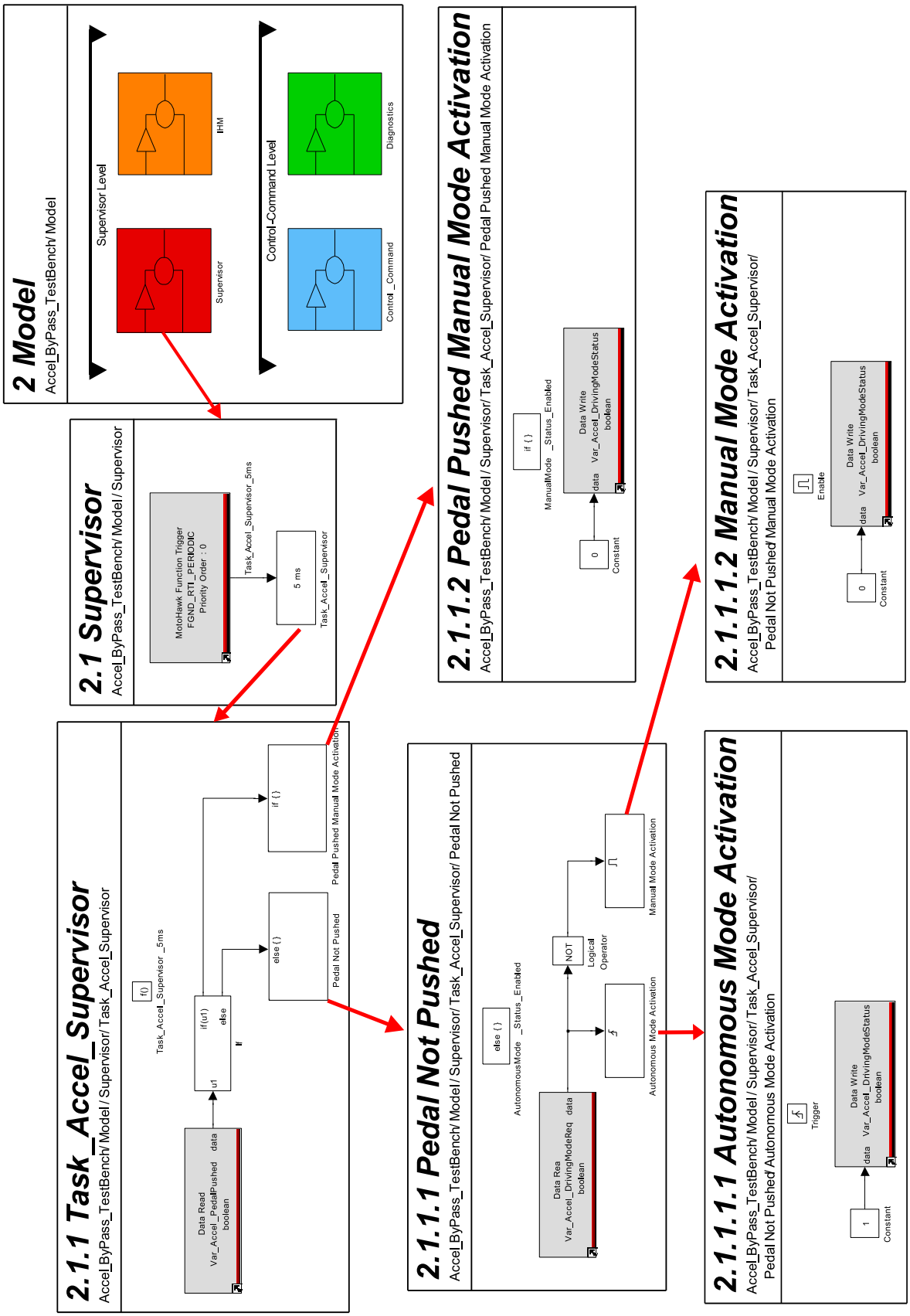


FIGURE 4.8 – Fonction de supervision du modèle logiciel du système de leurrage de l'accélérateur

4.2 Application de la méthodologie ALEBAS

L'objectif de l'application de la méthodologie est de rendre le logiciel plus robuste en y ajoutant des stratégies de détection de défauts ou de tolérance aux fautes ou encore d'activation de modes dégradés. Comme nous ne maîtrisons pas toutes les stratégies implémentées dans le calculateur de contrôle moteur (propriété du constructeur), il est difficile de mener correctement une analyse au niveau système. La méthodologie ALEBAS est ainsi appliquée directement sans connaissance d'analyse en amont et sera restreint aux modules de leurrage.

4.2.1 Analyse des risques et analyse SdF système

En amont de l'application d'analyse de SdF au niveau du modèle logiciel comme proposé dans ALEBAS, une analyse des risques du système est conduite. La fonction « accélérateur » des voitures est, depuis de nombreuses années maintenant, réalisée par l'intermédiaire d'une pédale d'accélérateur électronique et d'un calculateur moteur (ECM : Engine Control Module). Avant, la pédale actionnait directement le carburateur à l'aide d'un câble. Aujourd'hui, des projets de standardisation, entre constructeurs et équipementiers, adressent la fonction « accélérateur » à base d'électronique [ABD+13]. Ainsi, plusieurs études ont déjà été menées sur la gestion de la SdF de la fonction accélérateur standard [Ros16, AM14]. Pour notre analyse des risques, nous avons alors récupéré les informations proposées par les études existantes et les niveaux ASIL qui leur ont été attribués. Les risques sont les suivants :

- Absence d'accélération (QM)
- Accélération intempestive et significative : la limite d'acceptabilité est une valeur qui conduit à ce que cette accélération puisse rendre le véhicule incontrôlable pour le conducteur (ASIL B)
- Décélération intempestive et significative : la limite d'acceptabilité est une valeur qui conduit à une chute importante du régime moteur, qui peut rendre le véhicule incontrôlable pour le conducteur (QM)
- Accélération bloquée à une valeur constante (QM) (ce risque peut aussi être décrit comme l'absence de décélération)

Dans le cycle de développement et d'analyse du système, après l'analyse des risques, l'architecture du système est établi et fait l'objet d'une analyse de SdF afin d'étudier la propagation éventuelle de fautes dans celui-ci. L'architecture système de notre cas d'étude est celle présentée précédemment en figure 4.5. Dans ce système, nous n'avons pas la

maîtrise de tous les éléments notamment concernant le calculateur moteur de part son caractère fermé (propriété du constructeur) et son importante complexité de stratégie fonctionnelle et de diagnostic. Dans cette situation, il est plus difficile de dresser une analyse pertinente du système. Dans le cadre du prototypage, cette situation n'est pas un cas particulier. Ce constat illustre la nécessité d'une méthodologie flexible comme ALEBAS qui permet de réaliser des analyses SdF même lorsque les conditions ne sont pas optimales. Dans ces conditions, nous faisons le choix d'appliquer directement les analyses sur le modèle logiciel prototype en se focalisant sur le module de leurrage de l'accélérateur.

4.2.2 Génération des AdD et du tableau de l'AMDE

L'application du cœur de la méthodologie ALEBAS repose sur l'étude du modèle logiciel prototype à l'aide d'une analyse des AdD et d'une AMDE logicielle . Les AdD et le tableau de l'AMDE ont été générés grâce à la chaîne d'outils qui a été développée. Le modèle logiciel a été étudié à partir de toutes ces entrées et ces sorties, sur toute sa profondeur et à travers tous les sous-systèmes.

Les AdD obtenus sont présentés en annexe C.1. Le tableau AMDE obtenu est présenté en annexe C.2. L'AMDE logicielle se concentre sur l'apport de recommandations dédiées à des causes précises d'apparition d'une défaillance.

L'étude et les apports de ces analyses sur le modèle logiciel sont présentés dans la sous-section suivante.

4.2.3 Combinaison/interprétation des résultats - Amélioration du logiciel et du système

Cette étape vise à étudier les AdD et le tableau AMDE générés afin d'en déduire des améliorations à apporter au modèle logiciel, afin de le rendre plus robuste. A l'aide des AdD, plusieurs constatations peuvent être faites. Tout d'abord, on constate une grande similitude entre les AdD concernant les sorties de commande PWM1 et PWM2, 93% des noeuds des AdD sont identiques (AdD concernés : C.1 C.2 ; C.4 C.5 ; C.7 C.8 ; C.10 C.11 ; C.13 C.14 ; C.16 C.17 ; C.19 C.20 ; C.22 C.23 ; C.25 C.26). En effet, dans la stratégie fonctionnelle du logiciel, le processus de calcul de la commande PWM2 est dépendant du calcul de PWM1. Ainsi, si une erreur se propage dans le processus de calcul de PWM1, PWM2 est également impacté. Leur évolution est alors similaire, ce qui ne permet pas au calculateur moteur de détecter de défaut. Cette situation est problématique car elle peut conduire à l'apparition d'accélérations intempestives.

Notre attention se porte également sur les moyens de désactivation du mode autonome ; pour cette fonction il existe deux possibilités soit l'appui sur la pédale d'accéléra-

teur soit via une requête envoyée par CAN par le superviseur de conduite autonome. Ces deux solutions de désactivation se retrouvent à la base de nombreux AdD avec des événements redoutés basiques qui leur sont associés (AdD concernés : C.3 ; C.6 ; C.7 ; C.8 ; C.9 ; C.10 ; C.11 ; C.12 ; C.18 ; C.21 ; C.22 ; C.23 ; C.24).

Concernant la désactivation du mode autonome à partir de la pédale d'accélérateur, la stratégie actuelle du logiciel ne repose que sur un seul signal émis par la pédale d'accélérateur, donc dans le cas d'une défaillance de ce signal, le mode autonome peut soit se désactiver de façon intempestive soit rendre inactif l'appui sur la pédale pour sortir du mode autonome. Concernant la désactivation à partir de la réception d'une trame CAN envoyée par le superviseur de conduite autonome, aucune stratégie de diagnostic n'est associée à ce message.

Ainsi, concernant les problématiques de dépendance trop importante entre les commandes PWM1 et PWM2, et de détection de l'appui pédale à partir d'un seul signal, alors une amélioration de l'architecture système est proposée. Cette nouvelle architecture ajoute la relecture du signal $Us2_{acc}$ afin de calculer la commande PWM2 de façon indépendante par rapport à PWM1, aussi l'ajout de la lecture d'un deuxième signal de la pédale $Us2_p$ permet de renforcer la détection d'un appui pédale pour la désactivation du mode autonome. La nouvelle architecture est présentée en figure 4.9, les deux ajouts des relectures des signaux $Us2_p$ et $Us2_{acc}$ sont indiqués par des lignes bleues épaisses dans le schéma.

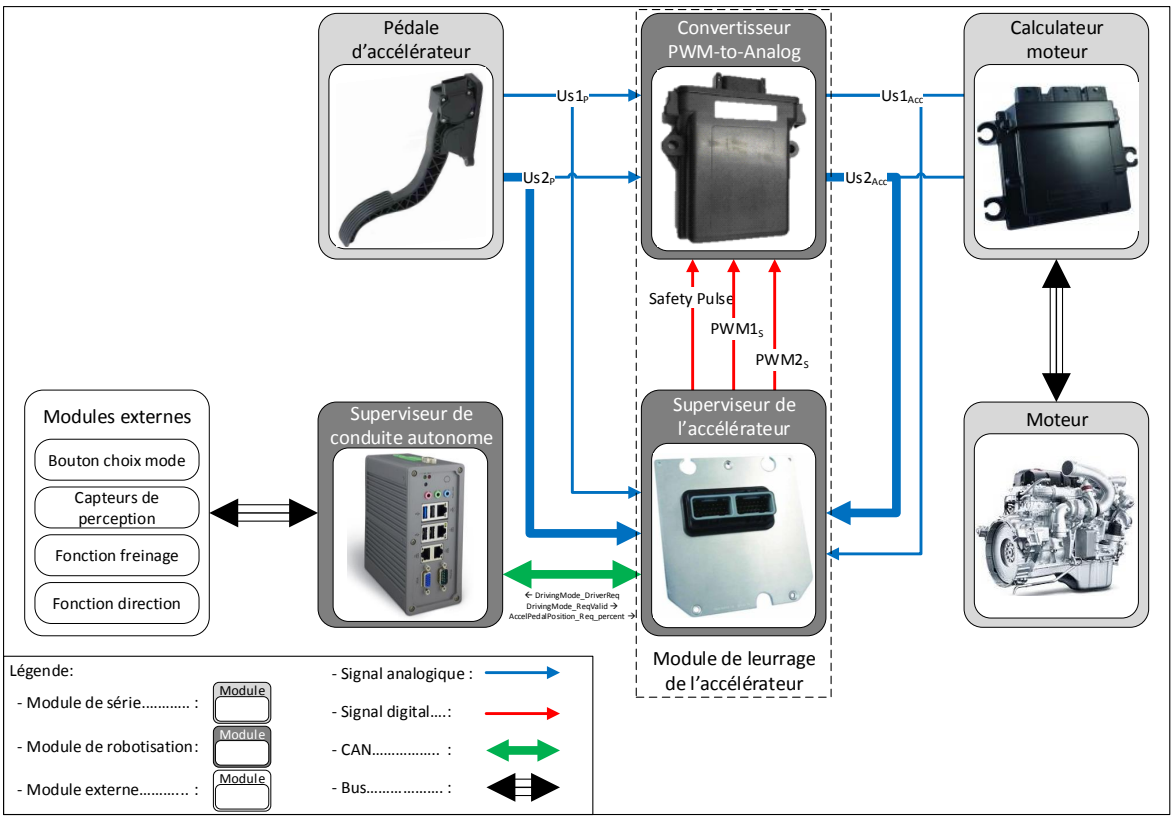


FIGURE 4.9 – Architecture améliorée du système de leurrage de la pédale d'accélérateur

La figure 4.10 présente un extrait du modèle sur l'acquisition des signaux analogiques qui viennent d'être ajoutés. La figure 4.11 expose l'indépendance de calcul de la commande PWM2.

Concernant la réception de la trame CAN, des moyens de protection sont proposés dans le tableau AMDE comme le montre la figure 4.12, qui est un extrait du tableau qui concerne cette trame CAN. Ainsi, des solutions de protection très efficaces comme le calcul d'un « checksum » et l'ajout d'un compteur sont proposés. Ces solutions permettent de vérifier que la trame CAN est bien active et de valider l'intégrité des données reçues. La figure 4.13 montre l'implémentation de cette stratégie de calcul du « checksum » et l'ajout du compteur dans la fonction diagnostic du modèle logiciel.

Notre attention dans ces AdD se porte aussi sur les variables présentes dans le modèle logiciel. Il est important d'observer les différentes parties du modèle, dans lequel une variable est mise à jour. Par exemple, la variable « Var_Accel_DrivingModeStatus » est mise à jour trois fois dans le modèle et lue une seule fois comme le montre l'extrait d'un AdD présenté en figure 4.14. Il faut vérifier qu'il n'y a pas de risque qu'une mise à jour de la variable ne soit écrasée par une autre avant d'avoir été lue. Il faut aussi essayer de factoriser les mises à jour des variables, ceci n'étant pas réalisé correctement dans le modèle actuel. La variable « Var_Accel_DrivingModeStatus » est mise à jour à deux endroits différents pour passer en mode manuel : dans les sous-systèmes « Pedal Pushed Manual Mode Activation » et « Manual mode Activation ». Dans les deux cas, la variable reçoit la valeur zéro, la question est de savoir s'il ne serait pas possible de réaliser cette opération qu'une seule fois, en opérant une factorisation des cas possibles qui mènent à cet état de la variable. On remarque dans le tableau AMDE (figure 4.14) qu'il est également indiqué de prendre des précautions concernant l'écriture sur une variable.

La figure 4.15 présente la nouvelle stratégie logicielle concernant l'écriture sur la variable « Var_Accel_DrivingModeStatus ». Cette stratégie optimise le nombre d'écriture sur la variable « Var_Accel_DrivingModeStatus » à deux : l'une pour mettre la variable à 1 et l'autre à 0 en fonction de l'état du système.

Les signaux d'entrées du calculateur sont à la base de nombreux AdD, ils peuvent donc être à l'origine de l'apparition de nombreux événements redoutés. Il est préférable de vérifier l'intégrité de ces signaux avant leur utilisation dans la stratégie logicielle. Concernant l'acquisition de la trame CAN, des améliorations ont déjà été ajoutées précédemment, nous nous intéressons maintenant aux signaux analogiques. Le tableau AMDE propose des recommandations pour protéger l'acquisition des signaux analogiques comme présenté dans la figure 4.16. Ces recommandations proposent notamment l'activation du port « Fault_Status » pour les blocs analogiques, ces ports permettent de connaître l'état matériel des entrées analogiques : l'ADC. Dans ces recommandations, il est aussi proposé à l'aide de redondance d'appliquer une stratégie de comparaison de signaux, afin de va-

lider leur intégrité. Cette stratégie peut être appliquée dans notre cas car nous avons une redondance sur l'acquisition des signaux pédales (Us_p) et des signaux de commandes envoyés au calculateur moteur (Us_{acc}). Aussi, les intervalles de fonctionnement des signaux sont bien connus, par exemple pour $Us1_p$ et $Us1_{acc}$, les valeurs brutes de l'ADC doivent être comprises entre 80 et 770, ainsi une stratégie de détection de valeurs en dehors de cet intervalle peut aussi être ajoutée. La figure 4.17 expose l'application de ces recommandations sur les signaux analogiques.

Dans cette partie, il a été montré par quelques exemples comment l'analyse des AdD et l'AMDE qui ont été générés automatiquement par la chaîne d'outils, sont exploités dans le cadre de la méthodologie ALEBAS pour améliorer le modèle logiciel. On remarque qu'à partir d'un modèle logiciel répondant uniquement aux contraintes fonctionnelles, il est possible d'ajouter de nombreuses stratégies logicielles qui permettent de rendre le logiciel plus robuste. Après avoir implémenté les recommandations nécessaires, les tests logiciels réalisés précédemment sur le logiciel prototype ont été de nouveau effectués. Cette fois, aucun comportement douteux n'a pu être observé, dans les cas qui posaient problème précédemment, comme par exemple une déviation de la donnée sur l'acquisition d'un signal analogique qui produisait une accélération intempestive, alors le système détecte cette fois-ci un défaut qui conduit le système à passer en mode par défaut c'est-à-dire le mode manuel. Ces stratégies auraient peut-être pu être implémentées directement par un développeur expérimenté mais l'utilisation de la méthodologie ALEBAS permet de n'oublier aucun point et surtout de justifier et de documenter l'application de ces stratégies logicielles pour la SdF. L'analyse des AdD est celle qui requiert le plus d'automatisme sans pour autant nécessité une grande expertise technique. En effet, cette analyse consiste notamment à comparer les arbres de défaillances générés, afin d'en déceler les similarités qui indiquent des modes communs de défaillances qu'il faut par la suite essayer de limiter dans la stratégie logicielle. Le tableau AMDE pour sa part donne directement les recommandations qu'il est possible d'appliquer, il est ainsi très abordable même pour un non-expert. La seule difficulté va résulter dans la quantité relativement importante de données à traiter. Le principal défaut de l'AMDE générée automatiquement repose sur la qualité des informations qui sont préalablement ajoutées dans la base de données pour la construction automatique des tableaux.

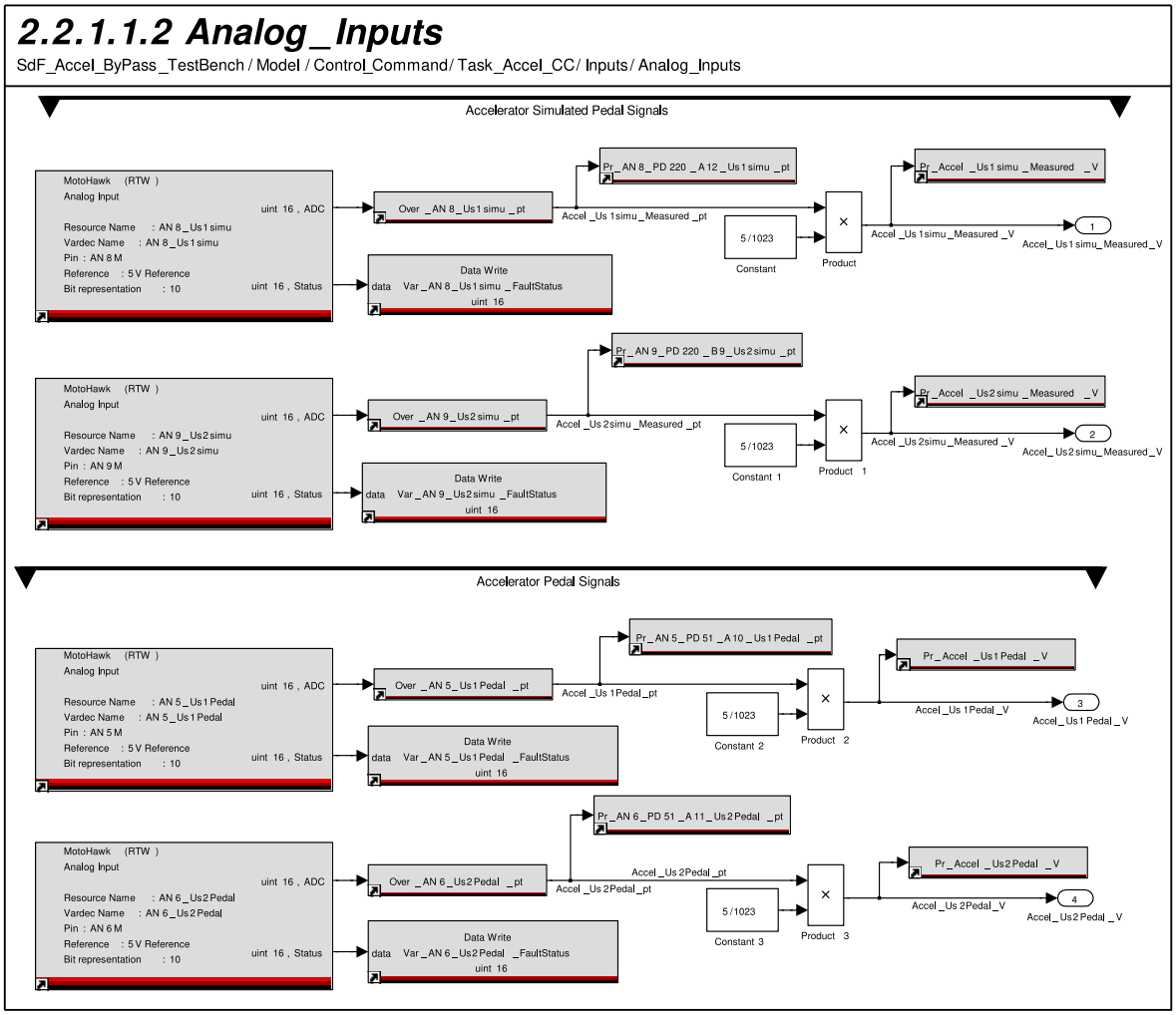


FIGURE 4.10 – Extrait du modèle logiciel concernant l'acquisitions des entrées analogiques

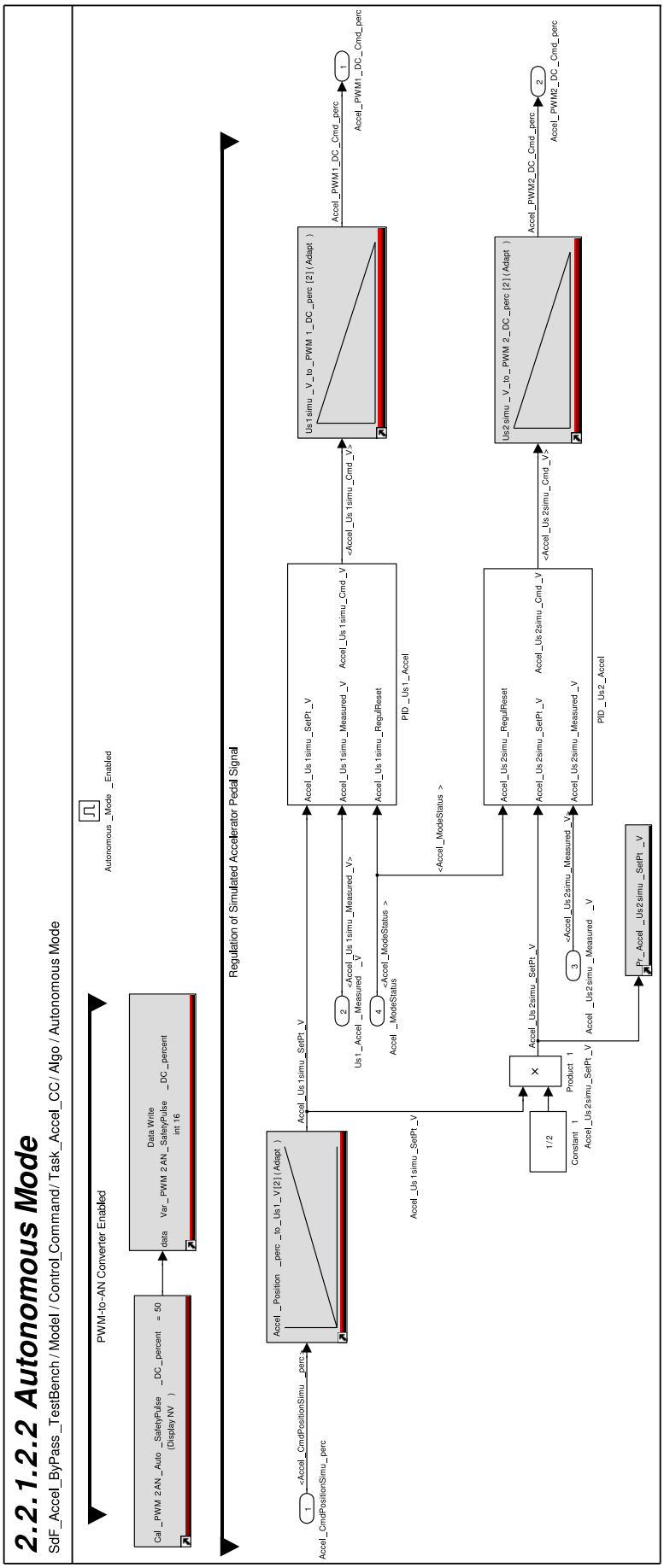


FIGURE 4.11 – Extrait du modèle logiciel concernant le calcul indépendant des commandes PWM1 et PWM2

Functions	Blocks	Potential failure modes	Potential causes	Potential effects (SW & ECU scope)	Potential effects on system	Risk level	Recommended actions	Applied actions
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Inputs /CAN_Inputs	Read CAN Message2	Signal value error (stuck, random...)	Bus is improprely terminated or defective ECU input (Chip CAN) or environmental perturbations	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- During test phase or maintenance, check the integrity of the CAN Input (Chip) - Apply V + Add the block CAN Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a strategy of checksum computing between the sender and receiver nodes. Add also a counter which must stay synchronized between both nodes. If an error is detected, the system pass in a limp homemode - Apply V + D + W + SM + Add redundancy signal to be able to switch on an other input if this one failed. Thus the system can continue to function properly	

FIGURE 4.12 – Extrait du tableau AMDE indiquant des recommandations pour assurer la réception d'une trame CAN

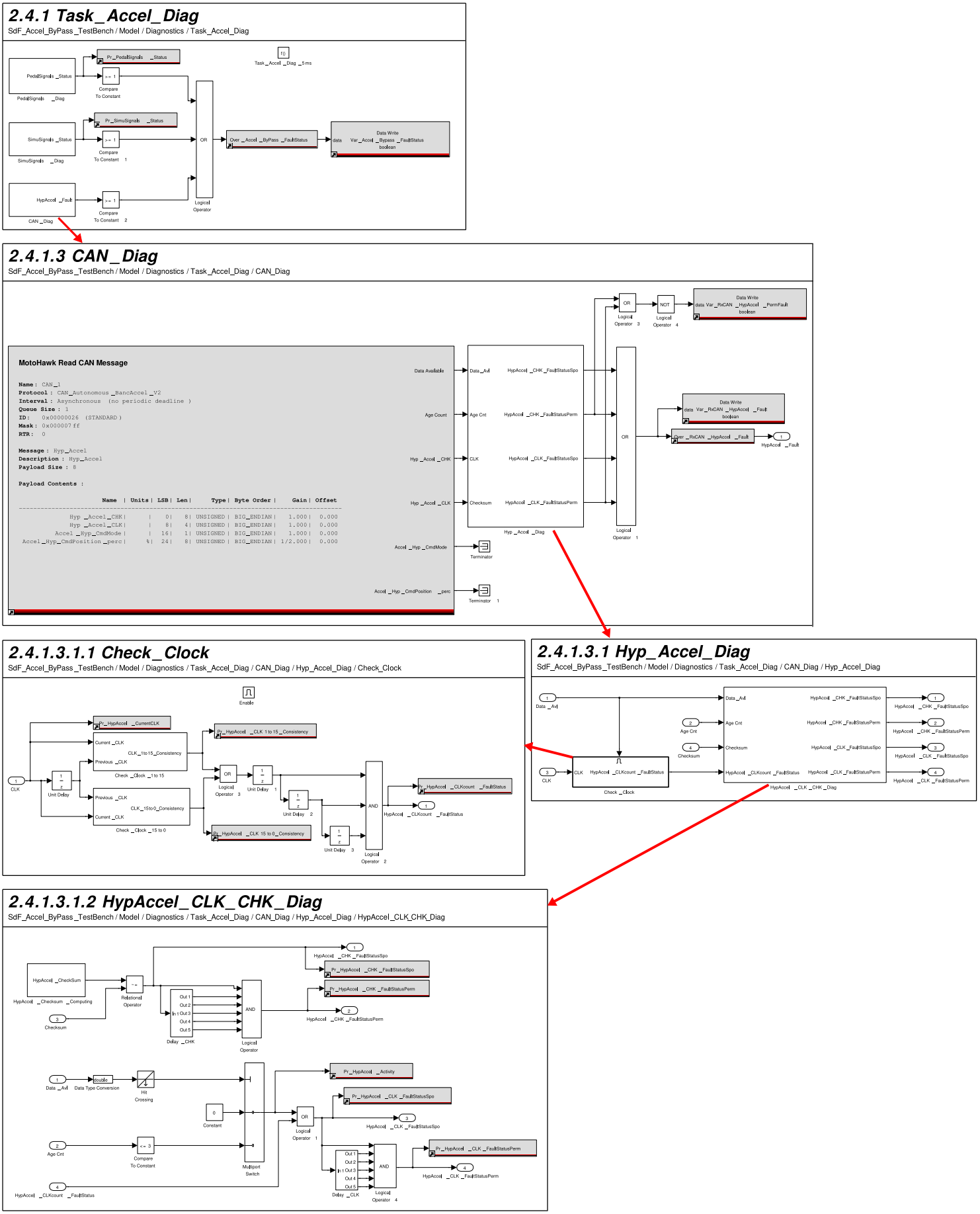
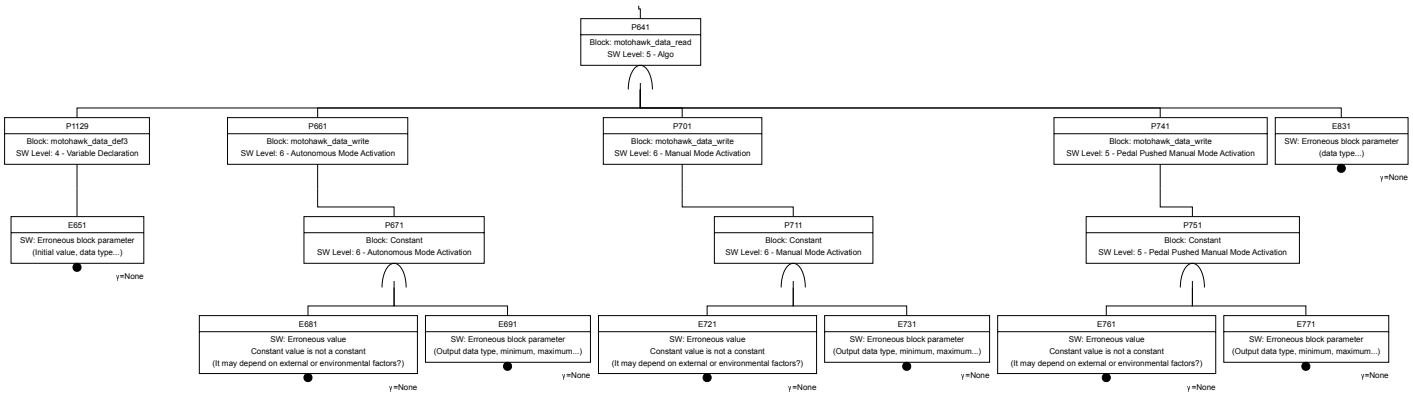


FIGURE 4.13 – Extrait du modèle logiciel exposant la stratégie mise en place pour le diagnostic du bus CAN



Functions	Blocks	Potential failure modes	Potential causes	Potential effects (SW & ECU scope)	Potential effects on system	Risk level	Recommended actions	Applied actions
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Algo	motohawk_data_read	Erroneous value	Avoid multiple "Data write" blocks	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency omission/commission ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency omission/commission ; Frequency erroneous motohawk_pwm2: Duty cycle omission/commission			- Development rules: in case of multiple writing of the same value, try to factorize	

FIGURE 4.14 – Extraits d'un AdD et du tableau AMDE concernant les multiples mises à jour de variable ("write")

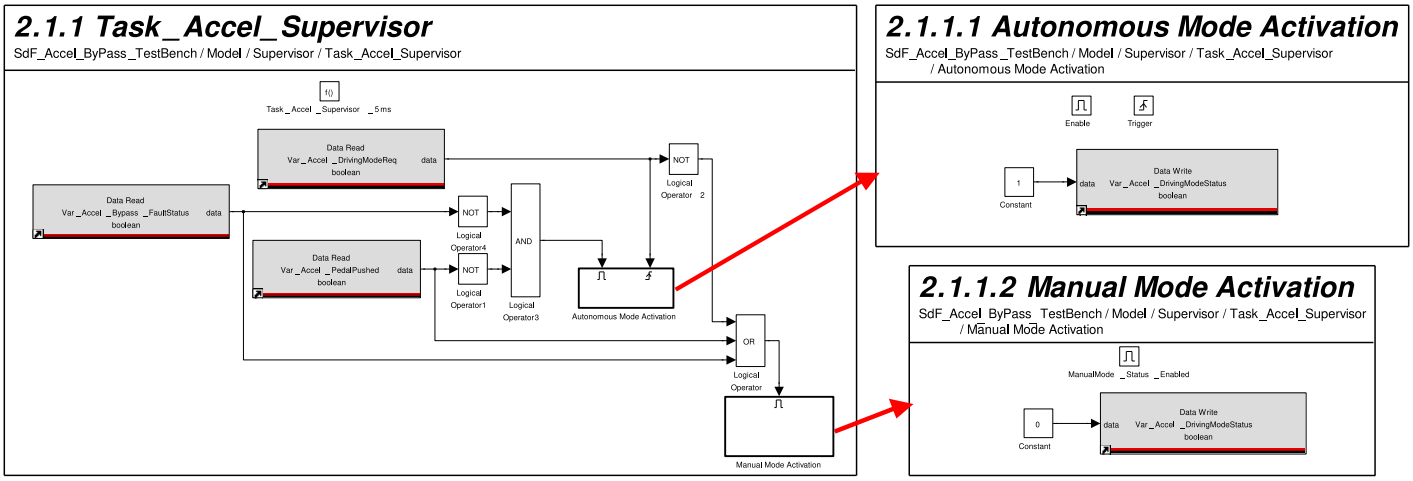


FIGURE 4.15 – Extrait du modèle logiciel exposant les améliorations apportées pour réduire le nombre d'écriture dans une variable

Functions	Blocks	Potential failure modes	Potential causes	Potential effects (SW & ECU scope)	Potential effects on system	Risk level	Recommended actions	Applied actions
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Inputs /Analog_Inputs	motohawk_ain	Signal stuck at low value (pull-down)	Input signal failed: Failed sensor or Defective wire/connection or disconnected or Short-Circuit to ground	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect value below the minimum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
			Failed ADC	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multi le failure switch in a safemode.	

FIGURE 4.16 – Extrait du tableau AMDE indiquant des recommandations pour assurer la réception de signaux analogiques

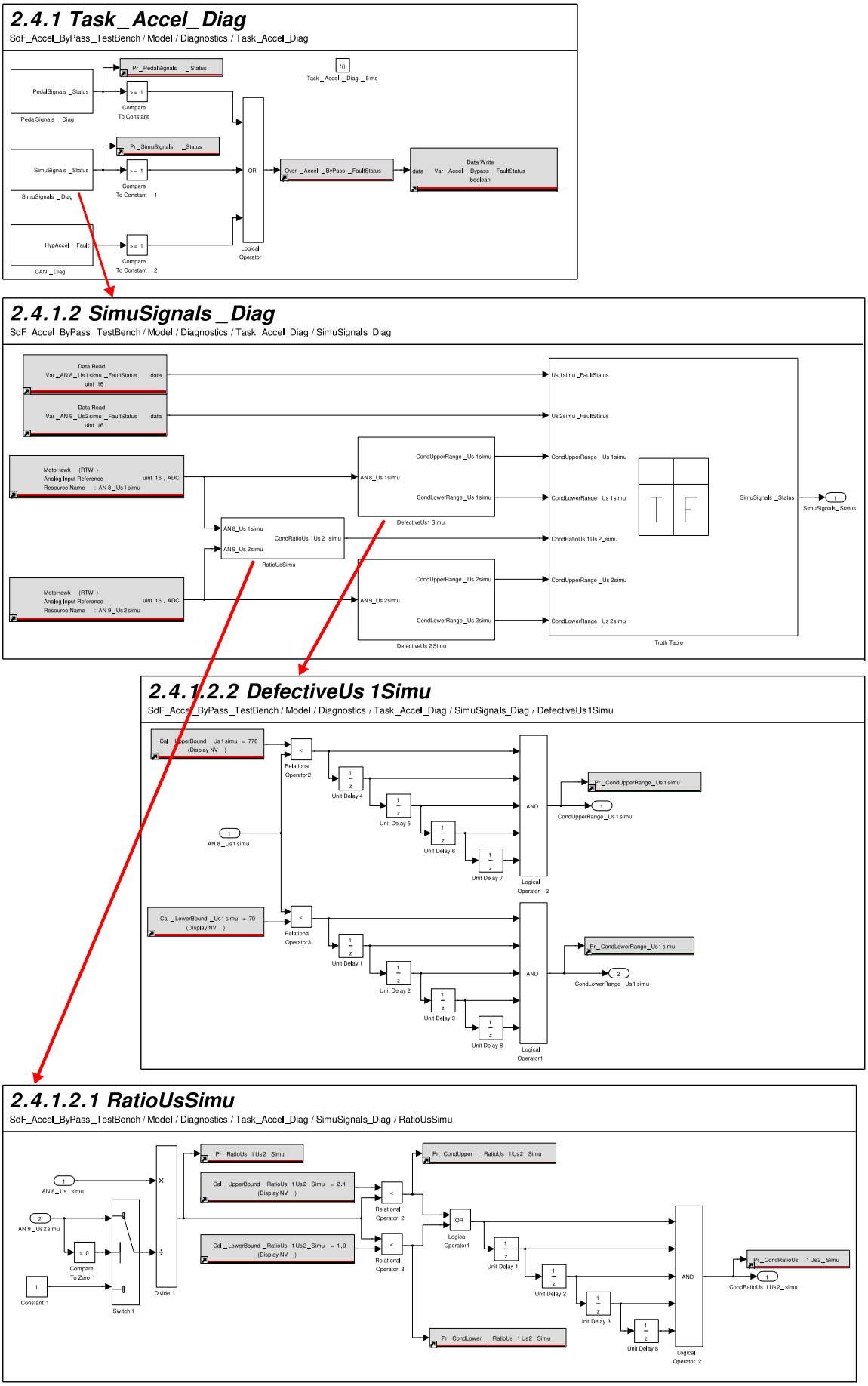


FIGURE 4.17 – Extrait du modèle logiciel exposant la stratégie mise en place pour le diagnostic des entrées analogiques

4.3 Conclusion

Ce chapitre a décrit l'application de la méthodologie ALEBAS sur un cas d'étude réel de prototypage d'un système embarqué. Ce cas d'étude concerne la robotisation de trois véhicules en vue de les rendre autonomes. La description du cas d'étude s'est restreinte à la fonction de robotisation de l'accélérateur pour limiter la présentation. La première partie de ce chapitre a décrit l'approche quant au besoin de robotiser l'accélérateur jusqu'au choix de l'architecture la mieux adaptée pour le prototypage. La fonction de robotisation de l'accélérateur se traduit finalement en un système de leurrage (« bypass ») des signaux pédales transmis au calculateur moteur. Comme il est souvent le cas lors du développement d'un prototype, une première version du logiciel embarqué du module de leurrage a été développée pour répondre uniquement aux contraintes fonctionnelles afin de vérifier la faisabilité et les performances de la solution choisie. Le système développé touche une fonction évidemment critique du véhicule : l'accélérateur car on imagine facilement qu'en cas de dysfonctionnements les conséquences peuvent être dramatiques. Ainsi, c'est dans un second temps que les contraintes de SdF sont abordées. La méthodologie ALEBAS est ainsi pratiquée pour faciliter la mise en place de la SdF dans le modèle logiciel prototype. Ce cas d'étude a illustré les difficultés auxquelles on peut être confronté lors du développement d'un prototype comme le manque d'informations précises sur certains composants comme pour le calculateur moteur dans notre cas. Cette situation a exposé la nécessité d'une méthodologie flexible permettant de s'adapter à ce genre de situation. Grâce à cette flexibilité, les analyses ont tout de même pu être conduite sur le modèle logiciel prototype et de nombreuses recommandations et stratégies logicielles ont été implémentées pour rendre le système plus sûr. L'analyse n'ayant pas été conduite au niveau système, de nombreuses recommandations avec différents niveaux de protection ont été proposées, dans cette situation, c'est au développer et à un expert de déterminer les recommandations à privilégier. Toutefois la génération des analyses AdD et AMDE facilite l'approche de la méthode et tend à accompagner le développeur dans l'implémentation de stratégie de SdF. L'utilisation de la méthodologie ALEBAS permet de lister exhaustivement les erreurs que peut produire le logiciel et surtout de justifier et de documenter l'application des stratégies de SdF. Cependant, la pertinence des analyses générées automatiquement reposent sur la qualité des informations contenues dans la base de données.

Conclusion générale

Ce travail de thèse s'intéresse à la prise en charge des contraintes de SdF dès les phases de prototypage d'un système embarqué et plus particulièrement le logiciel. L'objectif est de proposer une solution pour réduire le coût de l'intégration des contraintes de SdF en accompagnant les ingénieurs dans le développement. L'approche doit suivre les besoins liés au développement de systèmes embarqués et mécatroniques innovants. Pour répondre à ces besoins, nous avons proposé la méthodologie ALEBAS qui repose sur l'application d'analyses de SdF sur les modèles de logiciel embarqué au stade de conception détaillée. À partir de l'application de ces analyses de SdF, des recommandations sont déduites pour améliorer les modèles logiciels dans l'optique de les rendre plus sûrs. ALEBAS s'accompagne d'une chaîne d'outils logiciels pour l'automatisation de certaines tâches de l'analyse de SdF, afin d'alléger le travail de l'ingénieur en facilitant l'application de la méthodologie.

Dans un premier temps, nous avons défini précisément le contexte pour lequel nos recherches se sont orientées. L'objectif général est de faire face à l'augmentation de la complexité des systèmes embarqués de par leur implication toujours plus importante dans des fonctions à caractère critique.

L'état de l'art a permis de déterminer les besoins spécifiques du contexte tels que la nécessité de réactivité et de flexibilité lors du développement de systèmes innovants en phase de prototypage. Le processus de développement pour ces projets est de type incrémental et itératif. Le système progresse pendant le développement et prend des formes différentes allant de la preuve de concept à des prototypes et des démonstrateurs de plus en plus complets, jusqu'à atteindre l'objectif fixé avant industrialisation. Pour le développement de ces systèmes innovants, dans un premier temps, le challenge relève souvent principalement de problématiques fonctionnelles. Les contraintes de SdF ne sont souvent abordées que dans un second temps. Du point de vue de la SdF, l'état de l'art a permis d'établir qu'elle est principalement introduite dans les étapes d'analyse fonctionnelle et d'architecture système et logicielle. Il existe peu de méthodes qui descendent jusqu'à un niveau de détail important tel que l'étape de conception détaillée du logiciel. La première raison est que la SdF doit avant tout être envisagée avec des composants d'un niveau d'abstraction élevé dans le but de se préoccuper des fonctions principales du système et

d'en avoir une vision globale. Cependant, le développeur lors de son implémentation des stratégies logicielles de SdF est très peu assisté. La qualité de son implémentation repose essentiellement sur ses compétences algorithmiques et sa connaissance de l'outil de développement. Quelques méthodologies ont envisagé de poursuivre les analyses de SdF à un niveau de détail avancé du logiciel, mais le problème a été abordé de la même manière que pour les analyses à des niveaux architecturaux, notamment avec l'application d'analyse locale des composants. Or, les composants sont très nombreux à un niveau de conception détaillée et l'analyse locale nécessite l'étude manuelle de chacun des composants. La charge de travail devient alors monumentale.

La méthodologie ALEBAS est proposée pour répondre à ces différentes problématiques. ALEBAS est conçue de manière à accompagner le développeur de logiciel embarqué pour implémenter des stratégies de SdF, comme par exemple des moyens de tolérance aux fautes. À partir de la conduite d'analyse des AdD et d'une AMDE sur le modèle logiciel, des recommandations sont proposées pour améliorer le logiciel en vue de rendre le système plus sûr. Ces deux types d'analyse sont reconnues pour leur complémentarité et parce qu'elles sont répandues dans le monde de l'ingénierie, même pour des personnes qui ne pratiquent pas la SdF quotidiennement. Pour permettre l'analyse des AdD et l'AMDE, malgré la quantité importante de composants dans les modèles, on applique une analyse individuelle de chacun des composants basiques des librairies utilisées pour développer les modèles logiciels. Ces analyses individuelles sont indépendantes du logiciel développé à contrario de l'analyse locale abordée habituellement. Grâce aux informations récoltées lors de l'analyse individuelle (équations de dysfonctionnement et relations modes de défaillance / causes / recommandations), les AdD sont plus pertinents. Les éléments rassemblés dans les AdD concernent exactement l'événement redouté étudié qui se trouve au sommet de l'arbre. Pour l'AMDE logicielle, l'analyse individuelle permet de proposer des recommandations pour chaque cause d'apparition d'un mode de défaillance. Les recommandations proposées sont très proches du code pour aider au mieux le développeur. De plus, les recommandations proposées s'adaptent au niveau de risque évalué préalablement pour le mode de défaillance associé. Afin de compléter ces analyses, une tâche d'interprétation, réalisée par un ingénieur, est tout de même nécessaire afin d'affiner les résultats obtenus. Après avoir défini toutes les recommandations à appliquer pour améliorer le logiciel, il ne reste au développeur qu'à implémenter les solutions proposées. ALEBAS peut être répétée pour contrôler les nouvelles parties du logiciel qui ont été ajoutées. On note que l'efficacité de la méthodologie dépend fortement de la qualité de l'analyse individuelle.

Pour rendre l'utilisation de la méthodologie proposée encore plus abordable, les tâches conséquentes de construction des AdD et du tableau de l'AMDE ont été automatisées à l'aide d'un outil logiciel. Les résultats de l'analyse individuelle des composants basiques

ont été rassemblés au sein d'une base de données, afin de pouvoir être exploités par l'outil logiciel d'automatisation. Cette automatisation facilite également l'utilisation répétitive d'ALEBAS dans le cas des cycles de développement itératifs. L'outil logiciel développé permet différents réglages dans le but de flexibiliser l'analyse comme, par exemple, la possibilité de cibler certaines parties du modèle à étudier. Il est notamment possible de limiter la profondeur d'analyse, d'écarter des sous-systèmes, des entrées, des sorties ou des types de causes (logicielle, matérielle) ou encore de n'étudier que les parties du modèle qui ont évoluées depuis la version précédente. L'outil allège considérablement la charge de travail du développeur favorisant la mise en place d'une approche de SdF.

Enfin, une étude de cas a été conduite sur une sous-fonction de la robotisation d'un prototype de véhicule autonome. La fonction, sur laquelle ALEBAS a été testée, est la fonction de leurrage de l'accélérateur, afin de pouvoir commander l'accélération du véhicule par un système embarqué. L'objectif est de montrer l'évolution d'un logiciel embarqué, qui au départ a été développé en se concentrant sur les problématiques fonctionnelles et pour lequel les contraintes de SdF n'ont pas été suffisamment traitées. Grâce à l'application de la méthodologie, des améliorations sont apportées au logiciel afin de rendre l'ensemble du système embarqué plus sûr et d'assurer une plus grande continuité de service des fonctions.

Améliorations et Perspectives

Des améliorations à court-terme sont envisagées concernant la chaîne d'outils logiciels d'automatisation de la méthodologie ALEBAS. Il peut être mis en œuvre une identification des parties du modèle qui ont été modifiées en suivant les recommandations proposées; ainsi, l'itération des analyses ne pourrait porter que sur ces parties de modèle. Cette optimisation requiert l'application d'une analyse d'impact pour connaître ce qui a changé et ce qui a été affecté par ces modifications. Il est envisageable d'automatiser la tâche d'implémentation des recommandations en préparant en amont des extraits de modèle appliquant ces recommandations. A l'aide de ces extraits, il serait aussi intéressant de pouvoir déterminer si des recommandations sont déjà mises en œuvre dans le modèle.

À plus long terme, la question de l'automatisation complète de la méthodologie se pose. La tâche d'interprétation des résultats est celle qui risque d'être la plus complexe, car elle fait appel à des compétences humaines et relève souvent de l'expérience de l'ingénieur. Toutefois, une réflexion sur l'utilisation des réseaux bayésiens pour pallier à ce problème, ou en partie, pourrait être conduite. Grâce à l'utilisation de cette méthode, les probabilités pourraient être utilisées afin d'évaluer les blocs en fonction de leur implication dans l'apparition d'événements redoutés. Par conséquent, il serait possible d'au-

tomatiser l'identification des parties du modèle les plus fragiles, car elles se composent, par expérience, de blocs reconnus plus à risque. D'autre part, l'une des difficultés relevées consiste à déterminer de manière précise les conséquences de l'apparition d'une erreur dans le modèle logiciel. L'utilisation des équations de dysfonctionnement permet d'obtenir un premier niveau de détail, mais il pourrait être intéressant de discuter de l'utilisation de l'injection de fautes associée à de la simulation pour tenter de déterminer des effets plus précis.

Bibliographie

[ABD+13] Audi, BMW, Daimler, Porsche, and VW. Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units. Technical report, 2013. 96

[AC08a] Jean-François Aubry and Eric Châtelet. Sûreté de fonctionnement des systèmes de commande - Principes et méthodes. *Techniques de l'ingénieur*, (Automatique - Robotique), 2008. 14

[AC08b] Jean-François Aubry and Eric Chatelet. Sûreté de fontionnement des systemes de commande - Exemple d'application et rappels sur les RdP. *Techniques de l'ingénieur*, (Automatique - Robotique), 2008. 14

[ALRL04] Algirdas Azizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1) :11–33, 2004. XI, 38, 39

[AM14] Alexander Altby and Davor Majdandzic. *Design and implementation of a fault-tolerant drive-by-wire system*. PhD thesis, 2014. 96

[AotCI77] Chemical Industry Safety Association and Health Council of the Chemical Industries. *A Guide to Hazard and Operability Studies*. 1977. 15

[Ben56] Herbert D. Benington. Production of Large Computer Programs. In *Advanced programming methods for digital computers*, 1956. 11

[Ber07] Joseph Berreta. *Electronique, électricité et mécatronique automobile*. 2007. 2, 10

[BFB14] Giacomo Barbieri, Cesare Fantuzzi, and Roberto Borsari. A model-based design methodology for the development of mechatronic systems. *Mechatronics*, 24 :833–843, 2014. 13

[Bla07] Pierre Blandin. Architecture logicielle type pour les applications de contrôle-commande, 2007. 30

[Boe86] Barry W. Boehm. A Spiral Model of Software Development and Enhancement. *ACM SigSoft Software Engineering Notes*, 2(4) :22–42, 1986. 13

[BRS09] Anahita Baregheh, Jennifer Rowley, and Sally Sambrook. Towards a multidisciplinary definition of innovation. *Management Decision*, 47(8) :1323–1339, 2009. 2

[BS90] A. Bondavalli and L. Simoncini. Failure classification with respect to detection. *Proceedings. Second IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 47–53, 1990. 38

[BW01] John B Bowles and Chi Wan. Software failure modes and effects analysis for a small embedded control system. *IEEE Proceedings annual reliability and maintainability symposium*, pages 1–6, 2001. 17

[CE05] A Clifton and C A Ericson. *Hazard Analysis Techniques for System Safety (1)*. 2005. 15

[Cle] Emmanuel Clement. Arbre Analyste - http ://www.arbre-analyste.fr/. 78

[CTR14] Emmanuel Clement, Thierry Thomas, and Antoine B Rauzy. Arbre Analyste : un outil d'arbres de défaillances respectant le standard OPEN-PSA et utilisant le moteur XFTA. In *Congrès LAMBDA-MU 19*, pages 1–7, 2014. 78

[Fau08] Jean Faucher. Sûreté de fonctionnement - Concepts et enjeux. *Techniques de l'ingénieur*, (Génie industriel - Maintenance), 2008. 14

[FLVC05] Peter H Feiler, Bruce Lewis, Steve Vestal, and Ed Colbert. An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard : A Basis for Model-Based Architecture-Driven Embedded Systems Engineering. In *Architecture Description Languages*, pages 3–15. 2005. 21

[FM91] Kevin; Forsberg and Harold Mooz. The Relationship of System Engineering to the Project Cycle. In *National Council On Systems Engineering (NCOSE)*, Chattanooga (USA, TN), 1991. 11

[FMNP94] P Fenelon, J A McDermid, M Nicholson, and D J Pumfrey. Towards integrated safety analysis and design. *ACM SIGAPP Applied Computing Review - Special issue on saftey-critical software*, 2(1) :21–32, 1994. 38

[Gir05] Marc Giraud. Sûreté de fonctionnement des systèmes - Principes et méthodes. *Techniques de l'ingénieur*, (Electronique - Photonique), 2005. 14

[Gou09] Laurent Gouzenes. Des composants électroniques toujours plus petits et performants. *Réalités Industrielles*, pages 32–35, 2009. 2

[GRT00] Peter L Goddard, Raytheon, and Troy. Software FMEA techniques. In *Annual Reliability and Maintainability Symposium*, pages 118–123, 2000. 17

[HB04] Desmond N. D. Hartford and Gregory B. Baecher. *Risk and uncertainty in dam safety*. Thomas Telford, 2004. 15

[HEK05] Rick Homkes, Donna Evanecky, and Henry Kraebber. Applying FMEA to Software. In *American Society for Engineering Education Annual Conference & Exposition*, 2005. 17

[IEC91] IEC60812. Functional safety of electrical/electronic/programmable electronic safety/related systems, analysis for system reliability-procedure for failure mode and effect analysis (FMEA). 1991. 16

[IEC10] IEC61508. Sécurité fonctionnelle des systèmes électriques/électroniques/-électroniques programmables relatifs à la sécurité, 2010. 4

[ISO11] ISO26262. Road Vehicles Functional Safety, 2011. 4

[ISO13] ISO16290. Systèmes spatiaux - Définition des Niveaux de Maturité de la Technologie (NMT) et de leurs critères d'évaluation, 2013. 2

[JCV12] Jean-Marc Jézéquel, Benoît Combemale, and Didier Vojtisek. *L'ingénierie dirigée par les modèles : des concepts à la pratique*. Ellipses edition, 2012. 3

[Jéz06] Jean-Marc Jézéquel. L'ingénierie des modèles. Projet TRISKELL - INRIA / Irisa, 2006. 3

[JHMW06] Anjali Joshi, Mats P E Heimdahl, Steven P Miller, and Mike W Whalen. Model-Based Safety Analysis. Technical Report NASA report, 2006. 19

[Lap95] J. C. Laprie. *Guide de la sûreté de fonctionnement*. Cépaduès edition, 1995. XIX

[Law96] J. Dennis Lawrence. Software safety hazard analysis. Technical report, U.S. Nuclear Regulatory Commission, 1996. 38

[Les17] Patrick Leserf. *Optimisation de l'architecture de systèmes embarqués par une approche basée modèle*. PhD thesis, 2017. 40

[Lev86] Nancy G Leveson. Safety : Why , What , and How. *ACM Computing Surveys (CSUR)*, 18(2) :125 – 163, 1986. 10

[Lie76] C. Lievens. *Sécurité des systèmes*. Edition ce edition, 1976. 15

[LS83] Nancy G Leveson and Janice L Stolzy. Safety Analysis of Ada Programs Using Fault Trees. *IEEE Transactions on Reliability*, R-32(5) :479–484, 1983. 18

[Man95] John C Mankins. Technology Readiness Level. Technical report, NASA, 1995. 2

[Mar91] James Martin. *Rapid Application Development*. Macmillan edition, 1991. 12

[Mat15] MathWorks Automotive Advisory Board (MAAB). Control Algorithm Modeling Guidelines Using Matlab, Simulink and Stateflow, 2015. 32, 46, ix

[Mih07] Alin Gabriel Mihalache. *Modelisation et evaluation de la fiabilite des systemes mecatroniques : application sur systeme embarque*. PhD thesis, 2007. XI, 10

[MIL] MIL-STD-882 Norme militaire américaine. 15

[Mot12] Motor Industry Software Reliability Association (MISRA). *Guidelines for the Use of the C Language in Critical Systems*. 2012. 32

[MPS+11] Adachi Masakazu, Yiannis Papadopoulos, Septavera Sharvia, David Parker, and Tetsuya Tohdo. An approach to optimization of fault tolerant architectures using HiP-HOPS. *Software - Practice and Experience*, 2011. 20

[MW13] Markus Maurer and Hermann Winner. *Automotive Systems Engineering*. Springer, 2013. 2

[Nat04] National Aeronautics and Space Administration. NASA-GB-8719.13 : Software Safety Guidebook. Technical report, 2004. 35

[OPI08] *Etude sur le marché et les compétences autour des logiciels embarqués*. L'observatoire des métiers du numérique, de l'ingénierie, des études et du conseil (OPIIEC), 2008. 1

[OPI14] *Etude sur l'évolution des métiers et des besoins en formation pour les systèmes embarqués*. L'observatoire des métiers du numérique, de l'ingénierie, des études et du conseil (OPIIEC), 2014. 1

[OYCS05] Younju Oh, Junbeom Yoo, Sungdeok Cha, and Han Seong Son. Software safety analysis of function block diagrams using fault trees. *Reliability Engineering & System Safety*, 88 :215–228, 2005. 19

[PCSB16] Aneesh Paul, Rohan Chauhan, Rituraj Srivastava, and Mriganka Baruah. Advanced Driver Assistance Systems. In *SAE International Mobility Conference*, feb 2016. 10

[PJ14] Sharon L Poczter and Luka M. Jankovic. The Google Car : Driving Toward A Better Future ? *Journal of Business Case Studies*, 10(1) :7–14, 2014. 10

[PM99] Y Papadopoulos and J a McDermid. Hierarchically performed hazard origin and propagation studies. In *SAFECOMP'99*, pages 139–152, 1999. 20

[PMSH01] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering and System Safety*, 71 :229–247, 2001. 20

[Poi09] Gérald Point. *AltaRica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*. PhD thesis, 2009. 20

[PR99] G. Point and A. Rauzy. AltaRica : Constraint automata as a description language. In *Journal européen des systèmes automatisés*, volume 33, pages 1033–1052. Hermès, 1999. 20

[PS08] Chris Price and Neal Snooke. An automated software FMEA. In *International System Safety Regional Conference*, 2008. 18

[Pum99] David John Pumfrey. *The Principled Design of Computer System Safety Analyses*. PhD thesis, 1999. 16, 38

[Rei79] Donald J. Reifer. Software Failure Modes and Effects Analysis. *IEEE Transactions on Reliability*, R-28(3) :247–249, 1979. 17

[Rib12] William Ribbens. *Understanding Automotive Electronics (7th Edition) - An Engineering Perspective*. Butterworth-Heinemann, 2012. 1

[Ros16] Hans-leo Ross. *Functional Safety for Road Vehicles : New Challenges and Solutions for E-mobility and Automated Driving*. 2016. 96

[Roy70] Winston W. Royce. MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS Dr. Winston W. Rovce INTRODUCTION. In *IEEE Wescon*, 1970. 11

[Rup10] Nayan B. Ruparelia. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35(3) :8, 2010. 11

[SAE] SAE International Standard. Architecture Analysis and Design Language (AADL) - http ://www.aadl.info. 21

[Sei04] Inhalt Seite. Entwicklungsmethodik für mechatronische Systeme Design methodology for mechatronic systems VDI-Handbuch Konstruktion VDI-Handbuch Mikro-und Feinwerktechnik. 2004. 12

[Sha11] Septavera Sharvia. *Integrated Application of Compositional and Behavioural Safety Analysis*. PhD thesis, 2011. 19

[SK97] Devdas Shetty and Richard A. Kolk. *Mechatronics System Design*. Cengage le edition, 1997. XI, 10

[SP11] Septavera Sharvia and Yiannis Papadopoulos. Integrated Application of Compositional and Behavioural Safety Analysis. In *Advances in Intelligent and Soft Computing (AISC)*, 2011. 19

[VGRH81] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. *Fault Tree Handbook*. Systems an edition, 1981. 16

[VLAL13] Frédérique Vallée, Agnès Lanusse, Adil Alif, and Youssef Laarouchi. Model-Based Safety Assessment : Experiments conducted in the Build-It Safe project . In *ICSSEA'13*, 2013. 21

[Woo] Woodward. MotoHawk Control Solutions : Product Guide - http ://www.woodward.com/motohawkcontrolsolution.aspx. 14, 67

Annexe A

Analyse individuelle des blocs

A.1 Analyse individuelle du bloc d'entrée analogique

Mode de défaillance : *Erreur grossière* - Signal bloqué dans les valeurs basses (entre 0 et 10 en valeur de l'ADC)

Cause : Matériel - Défaillance du signal (ex : panne capteur, perturbations électromagnétiques...)

Recommandations

- V : Contrôle du bon fonctionnement du signal lors des phases de tests et vérifications au cours du processus de développement
- D : V + Implémentation d'une stratégie logicielle pour détecter les valeurs basses. Elles doivent être en dehors de l'intervalle du signal en opération normale. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance
- A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)
- MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)
- PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Matériel - Court-circuit à la masse

Recommandations

- V : Contrôle des connexions lors des phases de tests et vérifications au cours du processus de développement

D : V + Utilisation de la sortie diagnostic du bloc pour détecter le court-circuit à la masse. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Matériel - Fil défectueux ou déconnecté

Recommandations

V : Contrôle des connexions lors des phases de tests et vérifications au cours du processus de développement

D : V + Implémentation d'une stratégie logicielle pour détecter les valeurs basses. Elles doivent être en dehors de l'intervalle du signal en opération normale. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Matériel - Entrée matérielle du calculateur défaillante

Recommandations

V : Contrôle du bon fonctionnement de l'entrée lors des phases de tests et vérifications au cours du processus de développement

D : V + Utilisation de la sortie diagnostic du bloc pour détecter la panne de l'entrée. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Mode de défaillance : *Erreur grossière* - Signal bloqué dans les valeurs hautes (entre 1010 et 1023 en valeur de l'ADC)

Cause : Matériel - Défaillance du signal (ex : panne capteur, perturbations électromagnétiques...)

Recommandations

V : Contrôle du bon fonctionnement du signal lors des phases de tests et vérifications au cours du processus de développement

D : V + Implémentation d'une stratégie logicielle pour détecter les valeurs hautes. Elles doivent être en dehors de l'intervalle du signal en opération normale. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Matériel - Court-circuit à sur une alimentation supérieure ou égale à 5V

Recommandations

V : Contrôle des connexions lors des phases de tests et vérifications au cours du processus de développement

D : V + Utilisation de la sortie diagnostic du bloc pour détecter le court-circuit sur une alimentation. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Matériel - Entrée matérielle du calculateur défaillante

Recommandations

V : Contrôle du bon fonctionnement de l'entrée lors des phases de tests et vérifications au cours du processus de développement

D : V + Utilisation de la sortie diagnostic du bloc pour détecter la panne de l'entrée. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Mode de défaillance : *Erreur grossière* - Comportement erratique/aléatoire du signal

Cause : Matériel - Défaillance du signal (ex : panne capteur, perturbations électromagnétiques...)

Recommandations

V : Contrôle du bon fonctionnement du signal lors des phases de tests et vérifications au cours du processus de développement

D : V + Implémentation d'une stratégie logicielle pour détecter le comportement erratique par rapport au fonctionnement normal (ex : utilisation de la dérivée). Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Matériel - Entrée matérielle du calculateur défaillante

Recommandations

V : Contrôle du bon fonctionnement de l'entrée lors des phases de tests et vérifications au cours du processus de développement

D : V + Utilisation de la sortie diagnostic du bloc pour détecter la panne de l'entrée. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Logiciel - Mauvais réglage du paramètre « Data Type »

Recommandations

V : Contrôle du bon fonctionnement du signal sur toute sa plage de fonctionnement lors des phases de tests et vérifications au cours du processus de développement

D : V + Implémentation d'une stratégie logicielle pour détecter le comportement erratique par rapport au fonctionnement normal (ex : utilisation de la dérivée). Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

Mode de défaillance : *Erreur subtile* - Signal figé dans la plage de valeur de fonctionnement du signal en opération normale

Cause : Matériel - Défaillance du signal (ex : panne capteur, perturbations électromagnétiques...)

Recommandations

V : Contrôle du bon fonctionnement du signal lors des phases de tests et vérifications au cours du processus de développement

D : V + Implémentation d'une stratégie logicielle pour détecter le dysfonctionnement à l'aide d'un signal redondant. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Mode de défaillance : *Erreur subtile* - Signal bridé dans la plage de valeur de fonctionnement du signal en opération normale

Cause : Logiciel - Mauvais réglage du paramètre « Reference »

Recommandations

V : Contrôle du bon fonctionnement du signal sur toute sa plage de fonctionnement lors des phases de tests et vérifications au cours du processus de développement

Cause : Logiciel - Mauvais réglage du paramètre « Bit representation »

Recommandations

V : Contrôle du bon fonctionnement du signal sur toute sa plage de fonctionnement lors des phases de tests et vérifications au cours du processus de développement

A.2 Analyse individuelle du bloc gain

Mode de défaillance : *Erreur grossière* - comportement erratique de la sortie

Cause : Matériel - Défaillance de la mémoire

Recommandations

V : Contrôle du bon fonctionnement du signal lors des phases de tests et vérifications au cours du processus de développement

D : V + Implémentation d'une stratégie logicielle pour détecter un problème de mémoire. Utiliser le bloc « MotoHawk_nv_status » pour connaître l'état de la mémoire. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Logiciel - Mauvais choix du paramètre « datatype »

Recommandations

RD : Le bloc gain peut poser des problèmes lors de la compilation du code. Préférez l'utilisation du bloc « Product » pour opérer des multiplications. Assurez que le résultat soit dans un intervalle donné par l'utilisation d'un bloc « saturation » après le gain.

V : Contrôler le choix du paramètre « datatype »

D : V + Contrôler le résultat du calcul vis à vis d'un intervalle attendu s'il est connu.

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : Les stratégies de MD ou PC seront mises en place autour des blocs de sortie impac-

PC : tés par cette erreur.

Mode de défaillance : *Erreur grossière* - sortie du gain bloquée à une valeur

Cause : Matériel - Défaillance de la mémoire

Recommandations

V : Contrôle du bon fonctionnement du signal lors des phases de tests et vérifications au cours du processus de développement

D : V + Implémentation d'une stratégie logicielle pour détecter un problème de mémoire. Utiliser le bloc « MotoHawk_nv_status » pour connaître l'état de la mémoire. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Mode de défaillance : *Erreur subtile* - comportement correct mais résultat légèrement erroné

Cause : Logiciel - Valeur du gain à optimiser

Recommandations

RD : Préférez l'utilisation du bloc « Product » pour opérer des multiplications auquel vous associez un bloc « motohawk_table_1d » qui permettra de déterminer le gain en fonction du paramètre extérieur (nécessite un capteur indiquant l'état ou la valeur du paramètre extérieur).

Cause : Logiciel - Choix d'un gain fixe alors qu'il dépend d'un paramètre extérieur (Température, usure...)

Recommandations

V : Vérifier et si possible optimiser la valeur du gain

A.3 Analyse individuelle du bloc switch

Mode de défaillance : *Omission/Commision* - comportement erratique du choix de l'entrée passante vers la sortie

Cause : Logiciel - Mauvais choix de la condition

Recommandations

RD : L'entrée 2 doit contenir un signal booléen et la condition doit être réglée sur u2~=0 (consigne définie par le MAAB [Mat15])

Mode de défaillance : *Omission/Commision* - Le signal de sortie est bloqué sur l'une des deux entrées

Cause : Logiciel - Mauvais choix de la condition

Recommandations

RD : L'entrée 2 doit contenir un signal booléen et la condition doit être réglée sur u2~=0 (consigne définie par le MAAB [Mat15])

Mode de défaillance : *Erreur grossière* - comportement erratique de la valeur de sortie

Cause : Logiciel - Mauvais réglage du paramètre datatype

Recommandations

RD : L'entrée 2 doit contenir un signal booléen et la condition doit être réglée sur u2~=0 (consigne définie par le MAAB [Mat15])

V : Contrôler le choix du paramètre datatype

D : V + Contrôler que le signal de sortie est égale ou à l'entrée1 ou à l'entrée 3.

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : Les stratégies de MD ou PC seront mises en place autour des blocs de sortie impac-

PC : tés par cette erreur.

Mode de défaillance : *Erreur subtile* - signal de sortie légèrement erroné (ex : un arrondi de la valeur du signal)

Cause : Logiciel - Mauvais réglage du paramètre datatype

Recommandations

RD : L'entrée 2 doit contenir un signal booléen et la condition doit être réglée sur u2~=0 (consigne définie par le MAAB [Mat15])

V : Contrôler le choix du paramètre datatype

D : V + Contrôler que le signal de sortie est égale ou à l'entrée1 ou à l'entrée 3.

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : Les stratégies de MD ou PC seront mises en place autour des blocs de sortie impac-

PC : tés par cette erreur.

A.4 Analyse individuelle du bloc d'envoi d'une trame CAN

Mode de défaillance : *Erreur grossière* - message non-envoyé

Mode de défaillance : *Erreur grossière* - comportement erratique du message envoyé (mauvais ID, données erronées)

Cause : Matériel - Défaillance de la sortie (chip) CAN

Recommandations

V : Contrôle du bon fonctionnement de l'envoi du message lors des phases de tests et vérifications au cours du processus de développement

D : V + Implémentation d'une stratégie logicielle pour détecter un problème sur la sortie CAN. Utiliser le bloc « CAN_Fault_status » pour connaître l'état du bus CAN. Lorsque la défaillance est détectée, une variable prend la valeur 1, cette variable sera contrôlée au moment de la maintenance. Si le message porte des données à haute importance, ajouter un calcul de checksum à envoyer dans le message qui sera contrôlé par le/les module(s) à la réception et si besoin un signal de synchronisation qui s'incrément de 1 en 1 jusqu'à 15 puis redémarre. Cette synchronisation sera également contrôlée par le/les module(s) à la réception

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Matériel - Mauvaise résistance de terminaison

Recommandations

Cause : Matériel - Perturbations électromagnétiques (Fils non torsadés ou non-blindés si nécessaire)

Recommandations

V : Contrôle du câblage : torsadage et blindage

D : V + Implémentation d'un calcul de checksum à envoyer dans le message qui sera contrôlé par le/les module(s) à la réception et si besoin un signal de synchronisation qui s'incrément de 1 en 1 jusqu'à 15 puis redémarre. Cette synchronisation sera également contrôlée par le/les module(s) à la réception

A : V + D + Alerte du conducteur par un système visuelle ou sonore (nécessite un système de signalisation)

MD : V + D + A + Implémentation d'une stratégie logicielle passant le système dans un mode dégradé (nécessite une redondance matérielle pour s'assurer de la défaillance)

PC : V + D + A + Implémentation d'une stratégie logicielle de vote basée sur au moins deux redondances matérielles permettant d'assurer les performances complètes de la fonction

Cause : Matériel -

Recommandations

Mode de défaillance : *Temps* - périodicité supérieure à celle attendue (pour les messages périodiques)

A.5 Analyse individuelle du bloc trigger

Mode de défaillance : *Retard important voire non exécution*

Cause : Matériel - Défaillance du processeur

Recommandations

Cause : Matériel - Surcharge du processeur

Recommandations

Cause : Logiciel - problème de priorisation

Recommandations

Mode de défaillance : *Avance/Retard* - Exécution erratique

Cause : Matériel - Défaillance du processeur

Recommandations

Cause : Matériel - Surcharge du processeur

Recommandations

Mode de défaillance : *Retard léger*

Cause : Matériel - Surcharge du processeur

Recommandations

Annexe B

Résultats des analyses de SdF appliquées à l'exemple (chapitre 2)

FIGURE B.1 – Arbre de défaillance obtenu par l'analyse uniquement structurelle

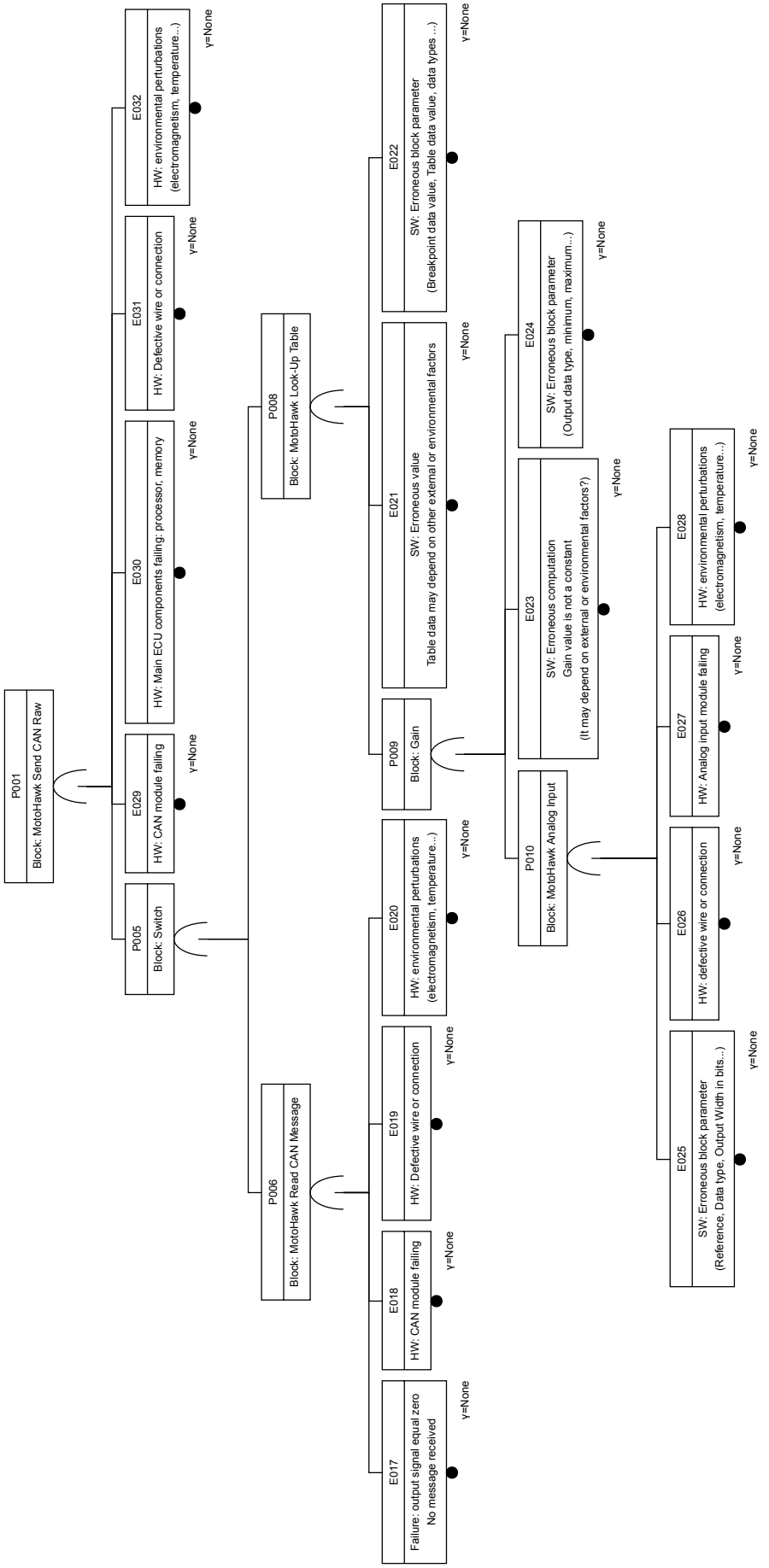


FIGURE B.2 – Arbre de défaillance obtenu via Alebas pour l'étude de l'événement redouté « valeur erroné du signal dans le message CAN »

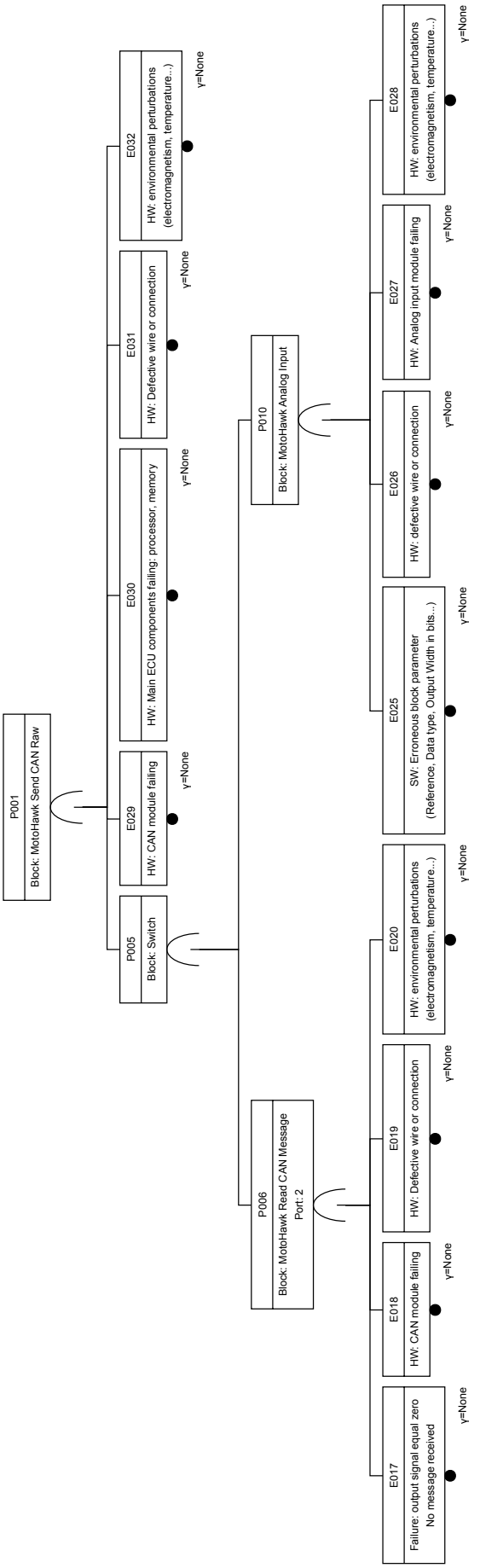


FIGURE B.3 – Arbre de défaillance obtenu via Alebas pour l'étude de l'événement redouté « comportement intempestif du signal CAN »

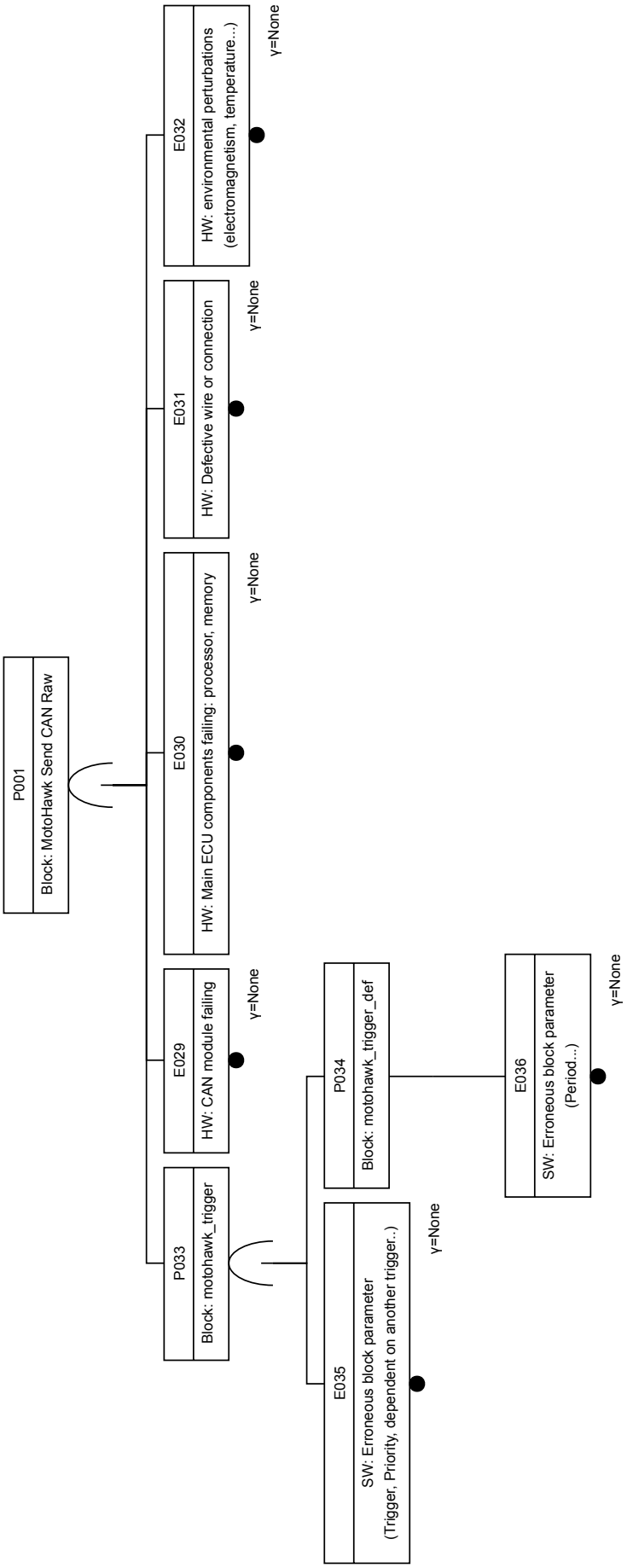


FIGURE B.4 – Arbre de défaillance obtenu via Alebas pour l'étude de l'événement redouté « non-respect de la périodicité d'envoi du message CAN »

Annexe C

Résultats des analyses de SdF appliquées sur le cas d'étude (chapitre 5)

C.1 Arbres de défaillances obtenus à partir du modèle prototype du cas d'étude de leurrage de la pédale d'accélérateur

Dans cette annexe, les arbres de défaillances obtenus grâce à la méthode ALEBAS et la chaine d'outils logiciels associée, sont présentés. Certains arbres sont présentés dans leur version complète et réduit (coupe minimale) pour essayer de permettre une meilleur lisibilité.

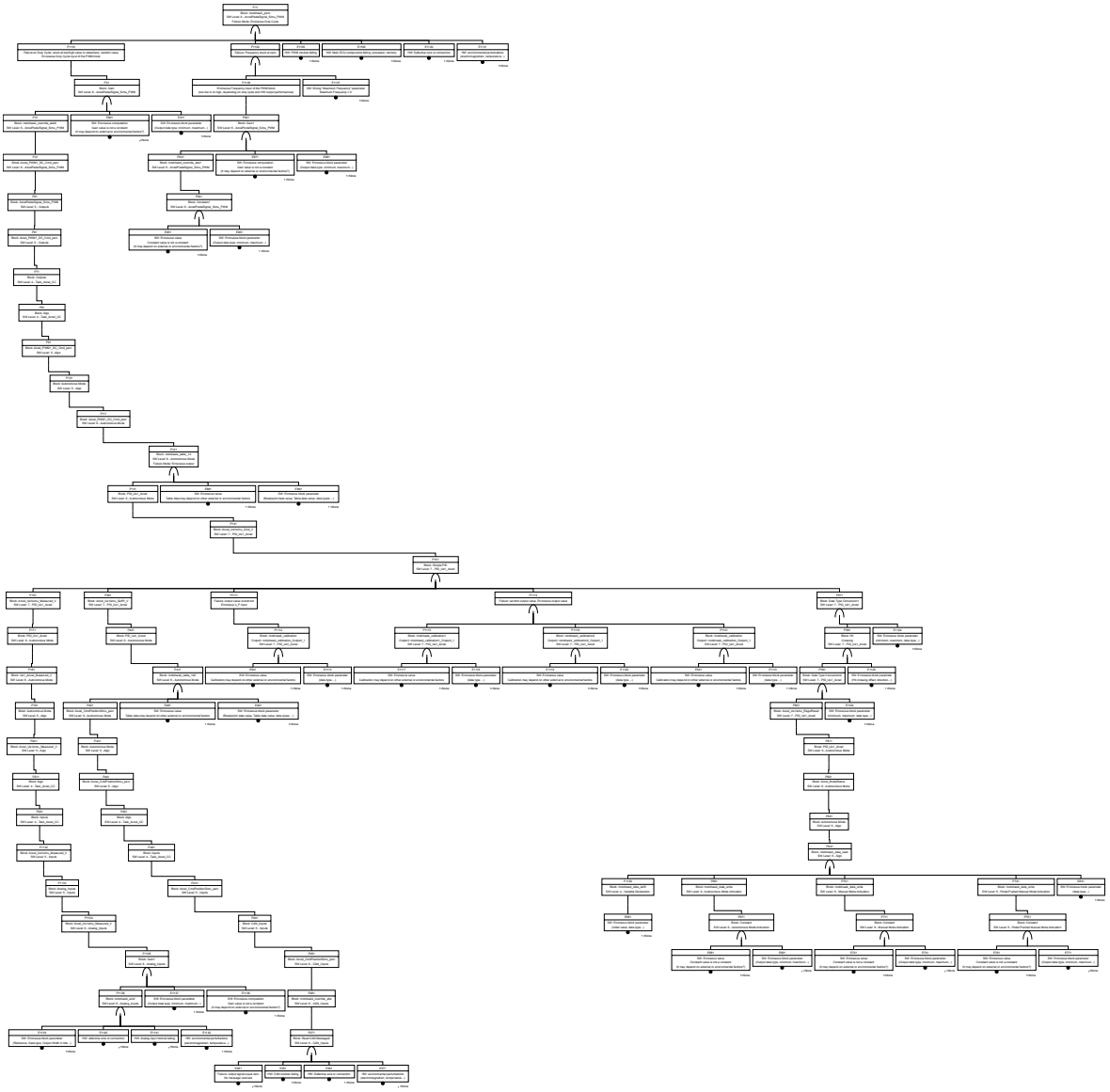


FIGURE C.1 – Arbre de défaillances complet : Duty cycle erroné de la sortie PWM1

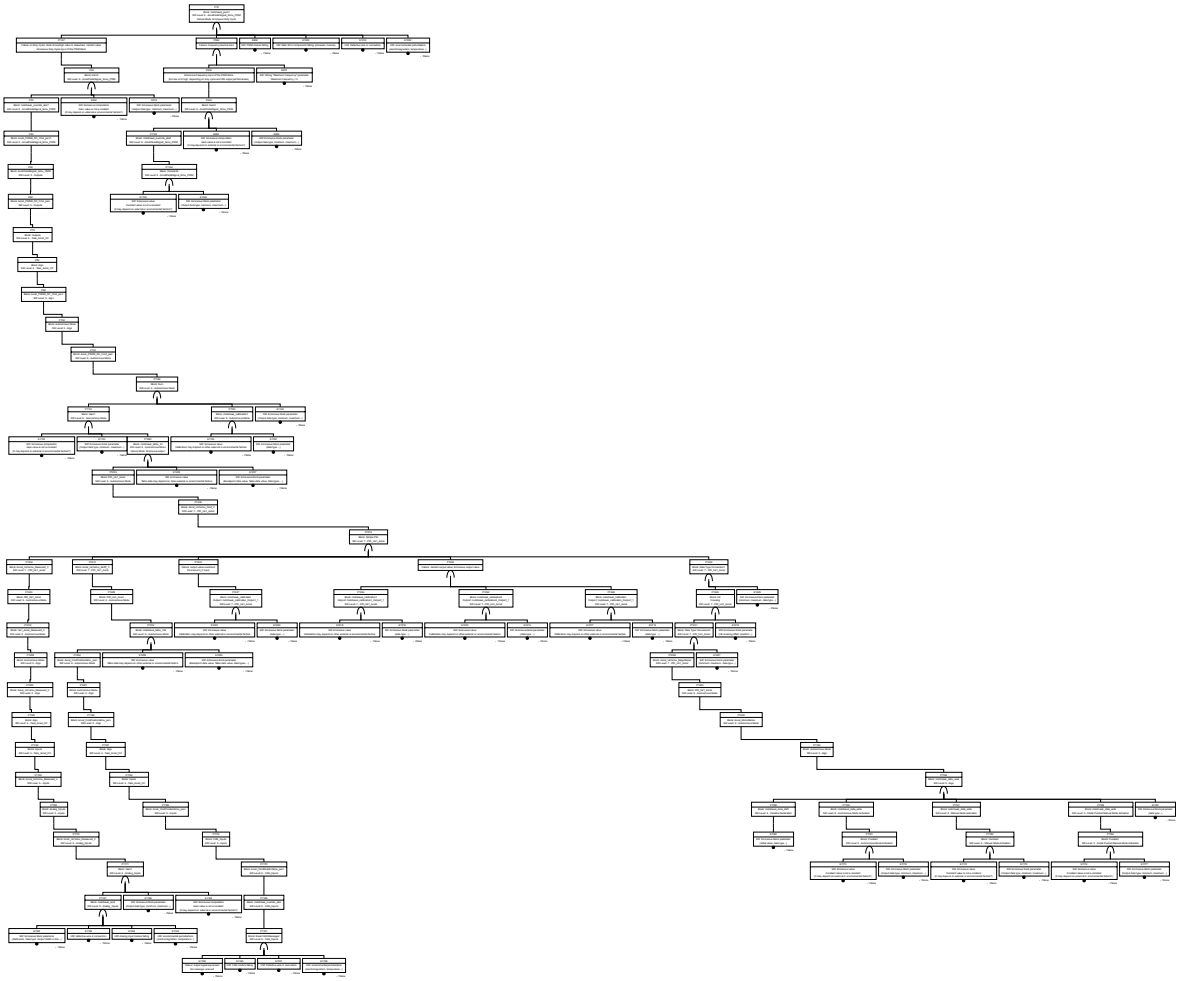


FIGURE C.2 – Arbre de défaillances complet : Duty cycle erroné de la sortie PWM2

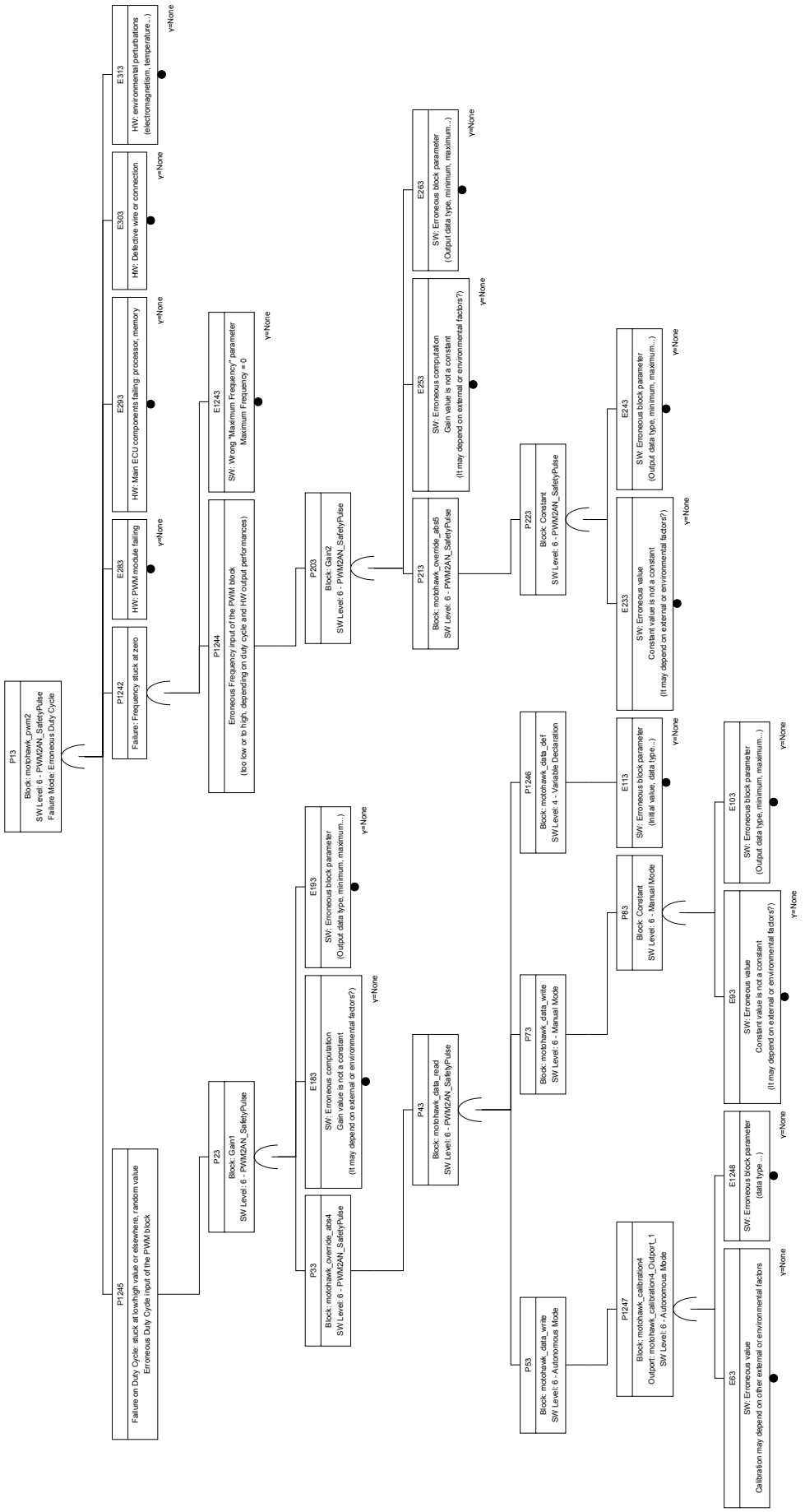


FIGURE C.3 – Arbre de défaillances complet : Duty cycle erroné de la sortie Safety Pulse

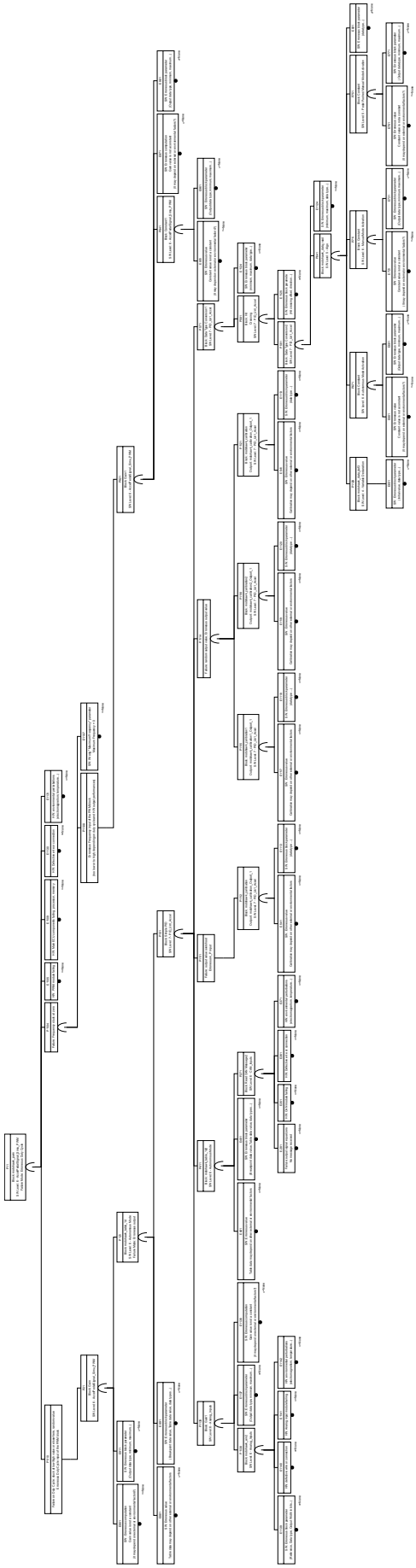


FIGURE C.4 – Arbre de défaillances coupe minimale : Duty cycle erroné de la sortie PWM1

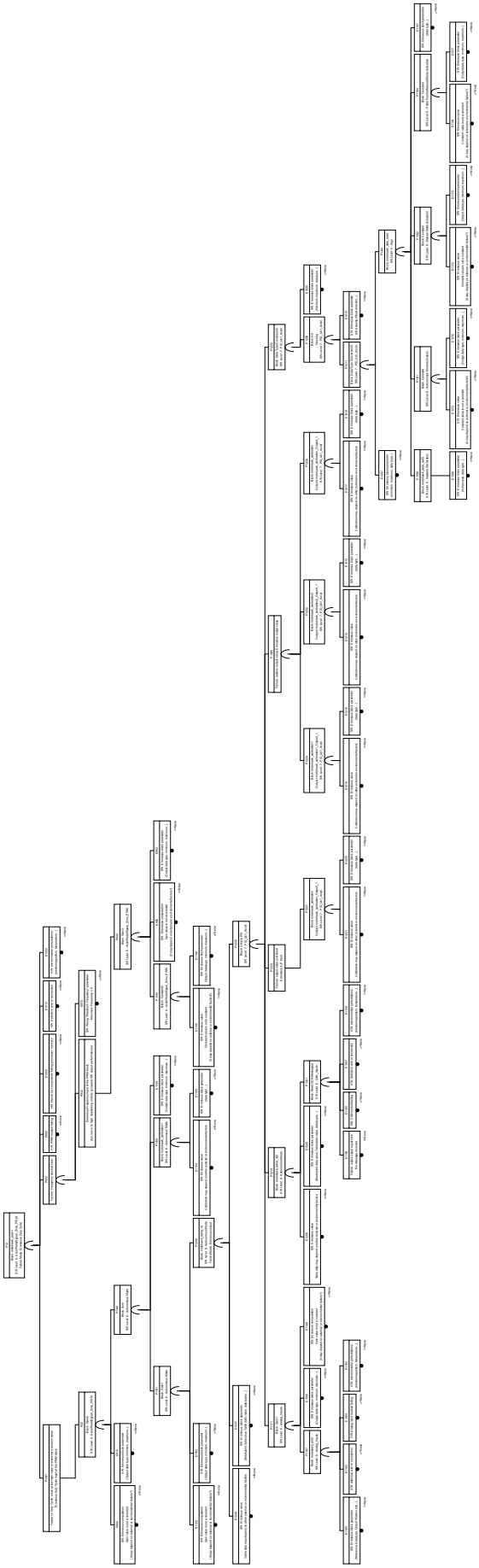


FIGURE C.5 – Arbre de défaillances coupe minimale : Duty cycle erroné de la sortie PWM2

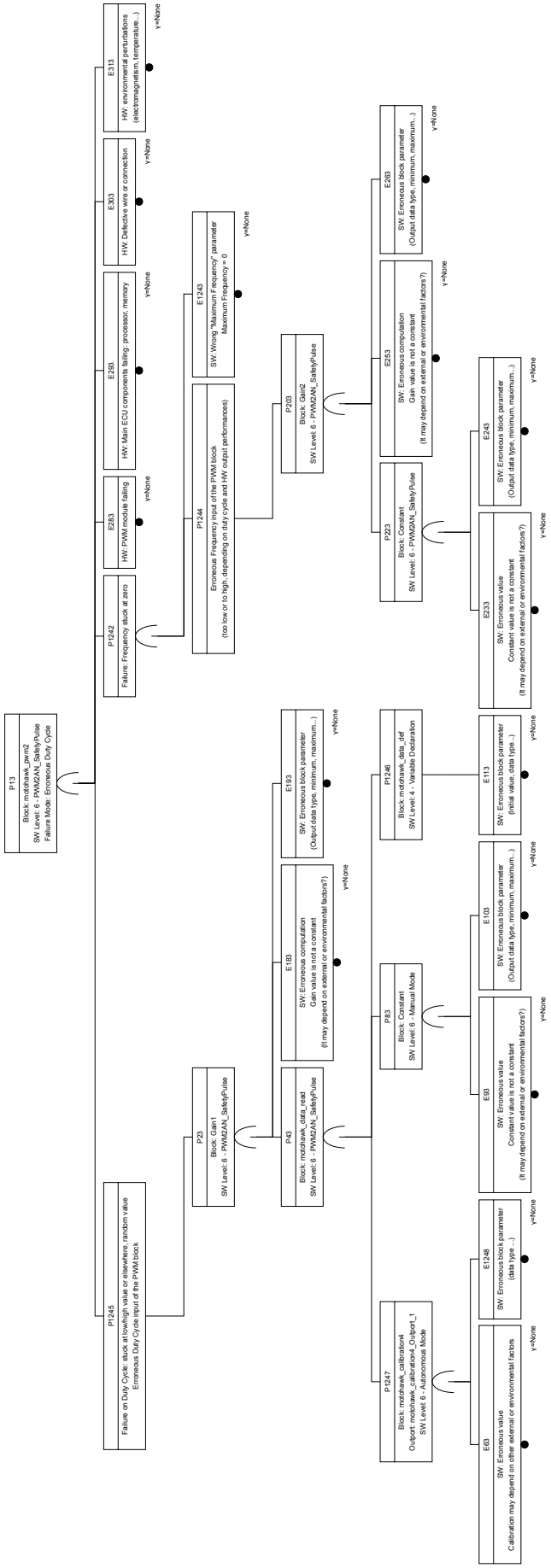


FIGURE C.6 – Arbre de défaillances coupe minimale : Duty cycle erroné de la sortie Safety Pulse

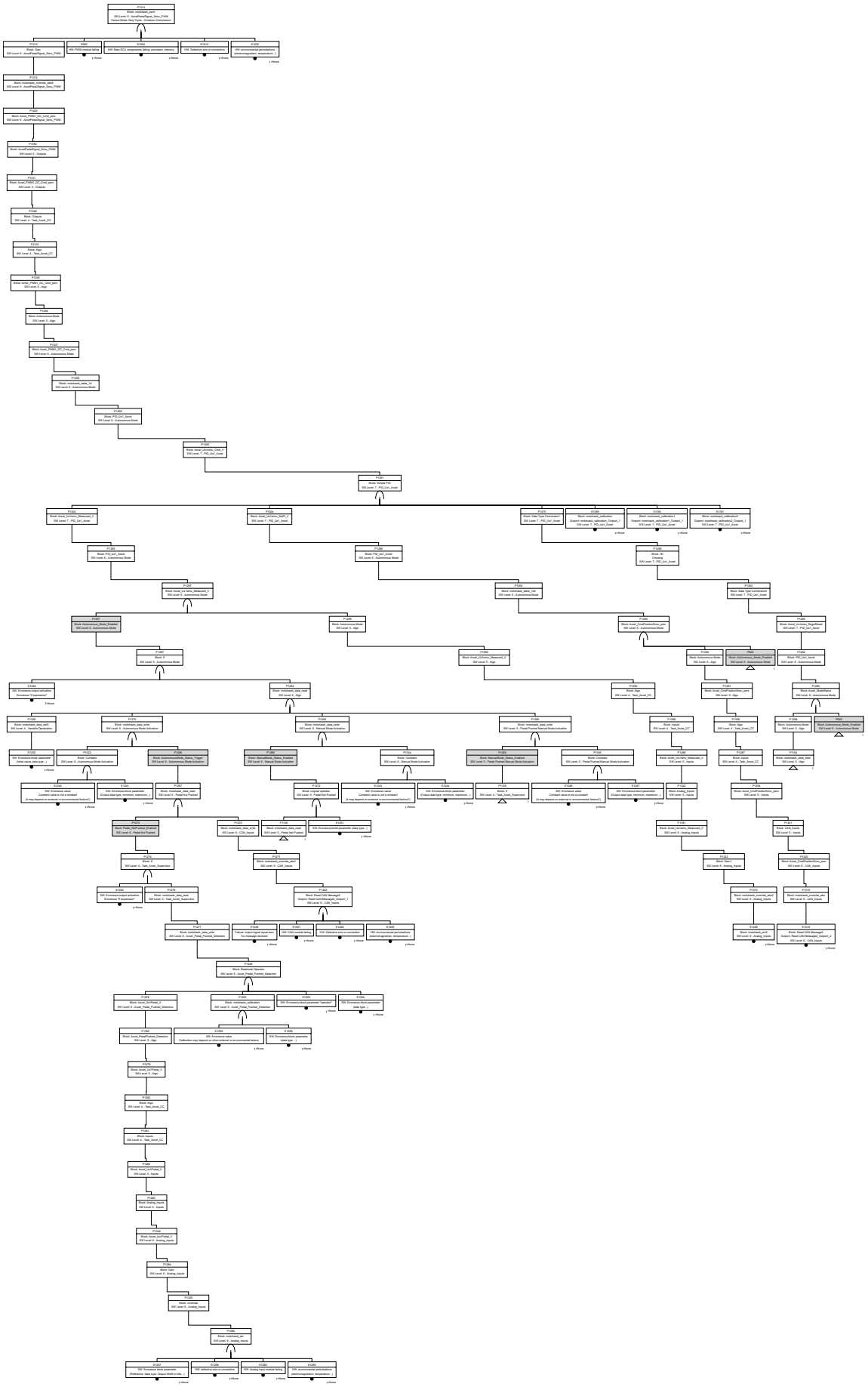


FIGURE C.7 – Arbre de défaillances complet : Omission ou commission du Duty Cycle de la sortie PWM1

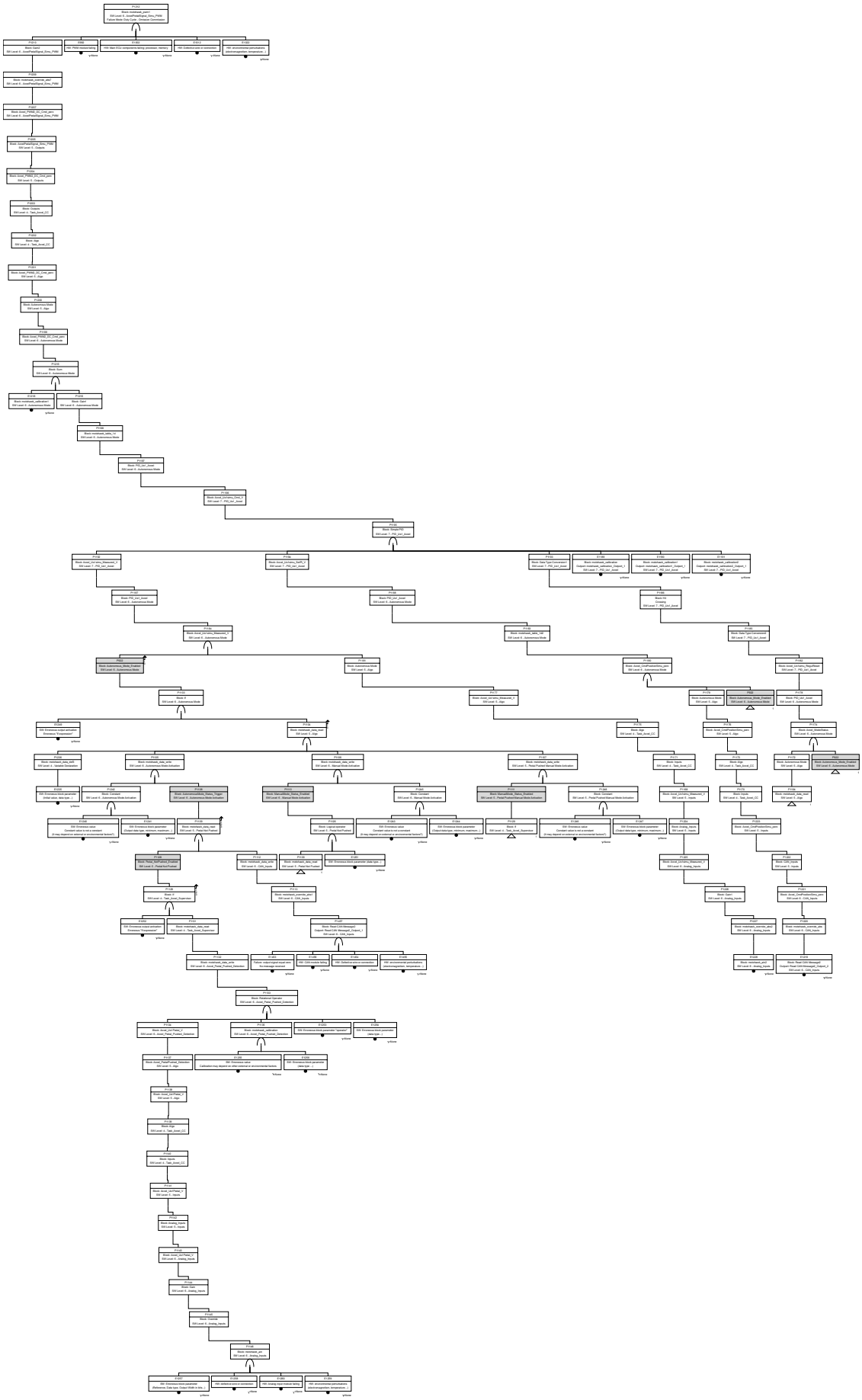


FIGURE C.8 – Arbre de défaillances complet : Omission ou commission du Duty cycle de la sortie PWM2

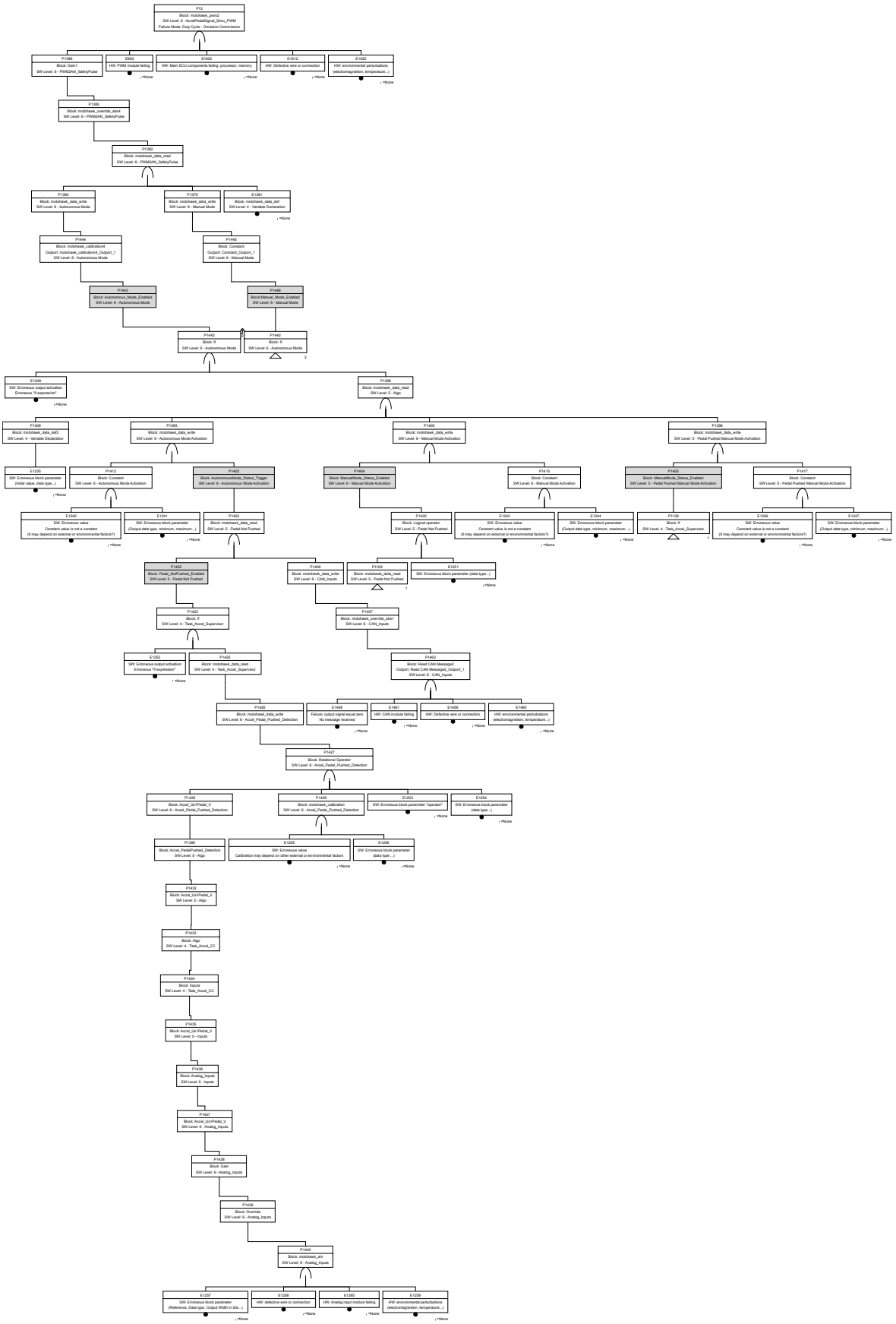


FIGURE C.9 – Arbre de défaillances complet : Omission ou commission du Duty cycle de la sortie Safety Pulse

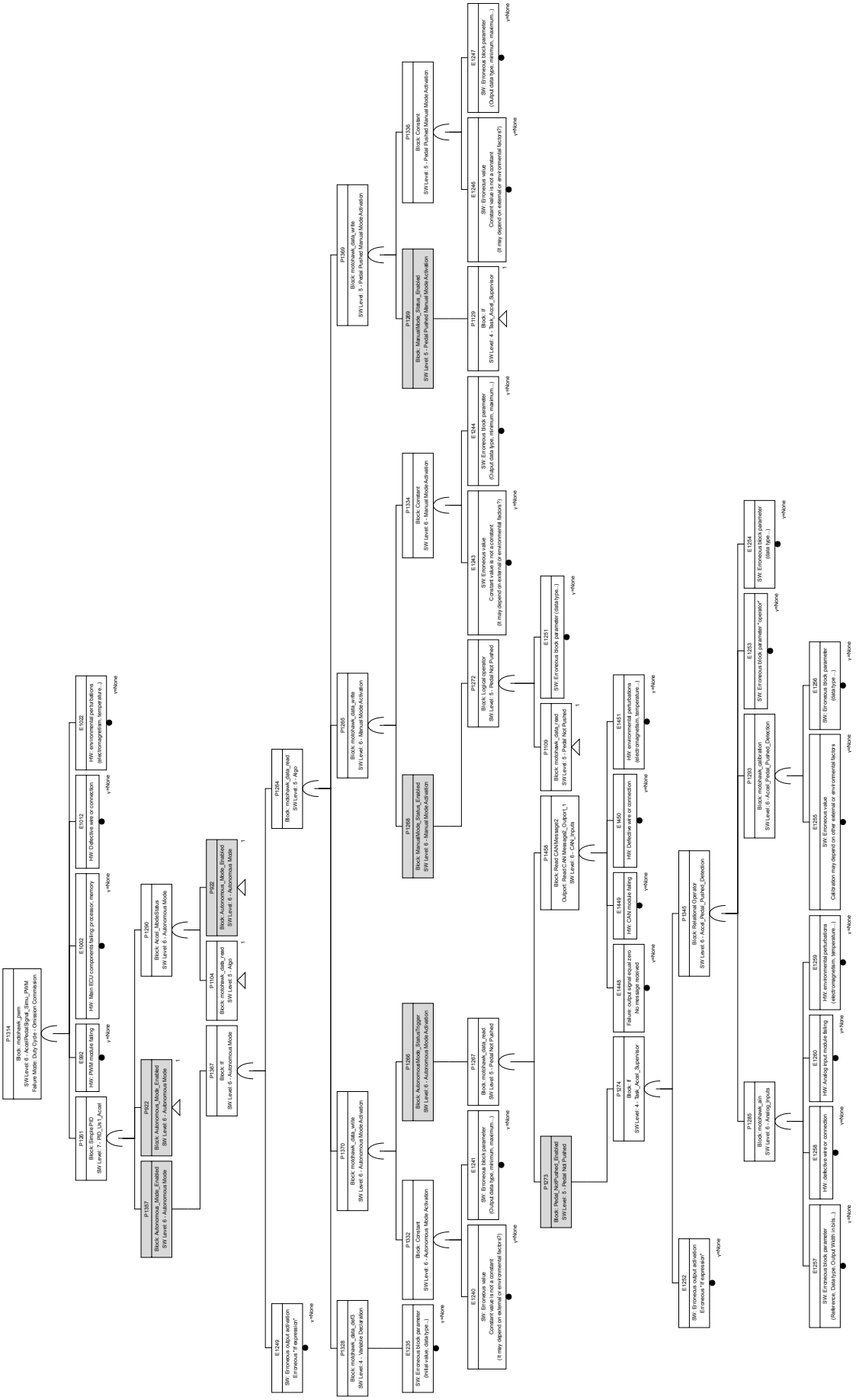


FIGURE C.10 – Arbre de défaillances coupe minimale : Omission ou commission du Duty Cycle de la sortie PWM1

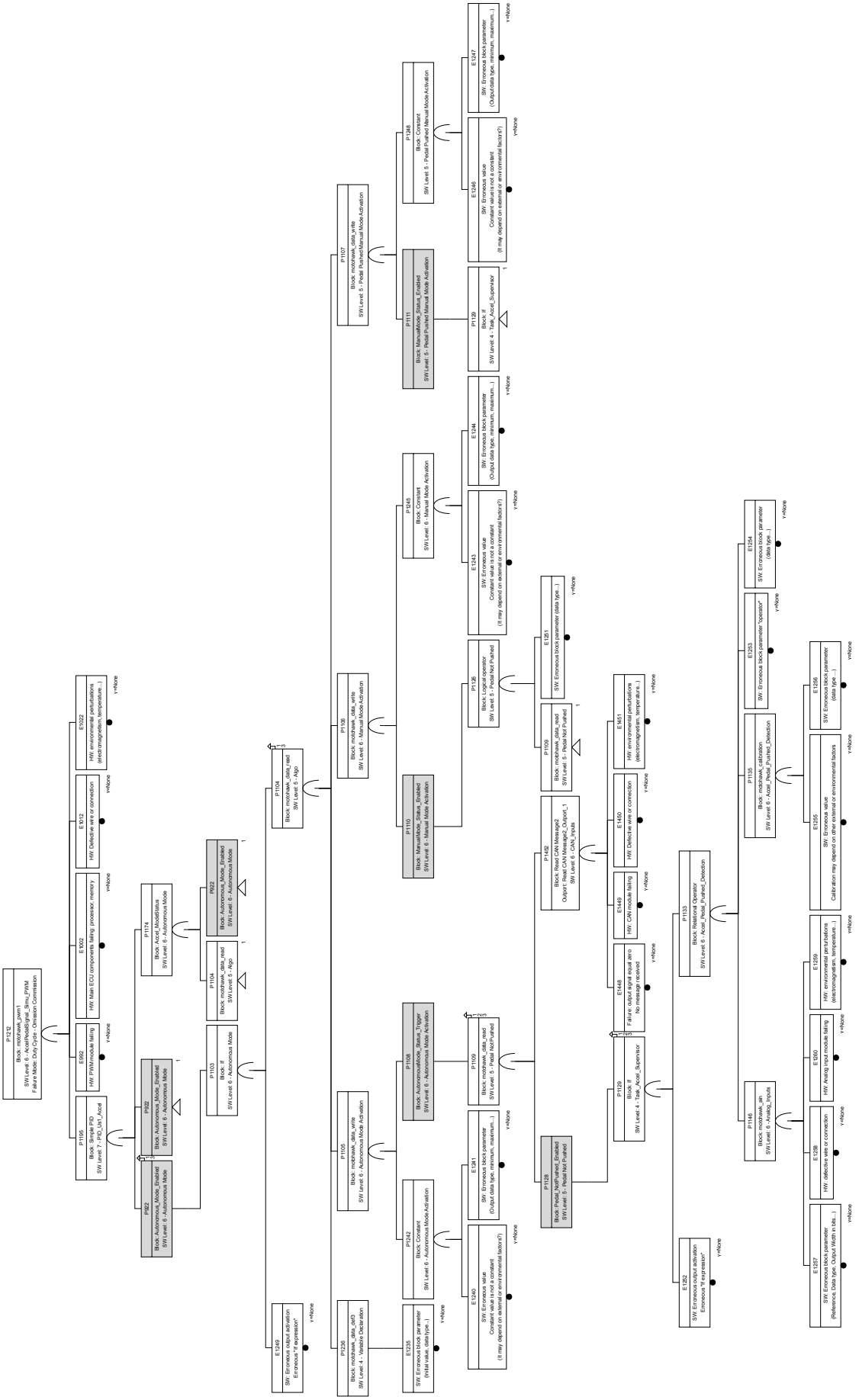


FIGURE C.11 – Arbre de défaillances coupe minimale : Omission ou commission du Duty cycle de la sortie PWM2

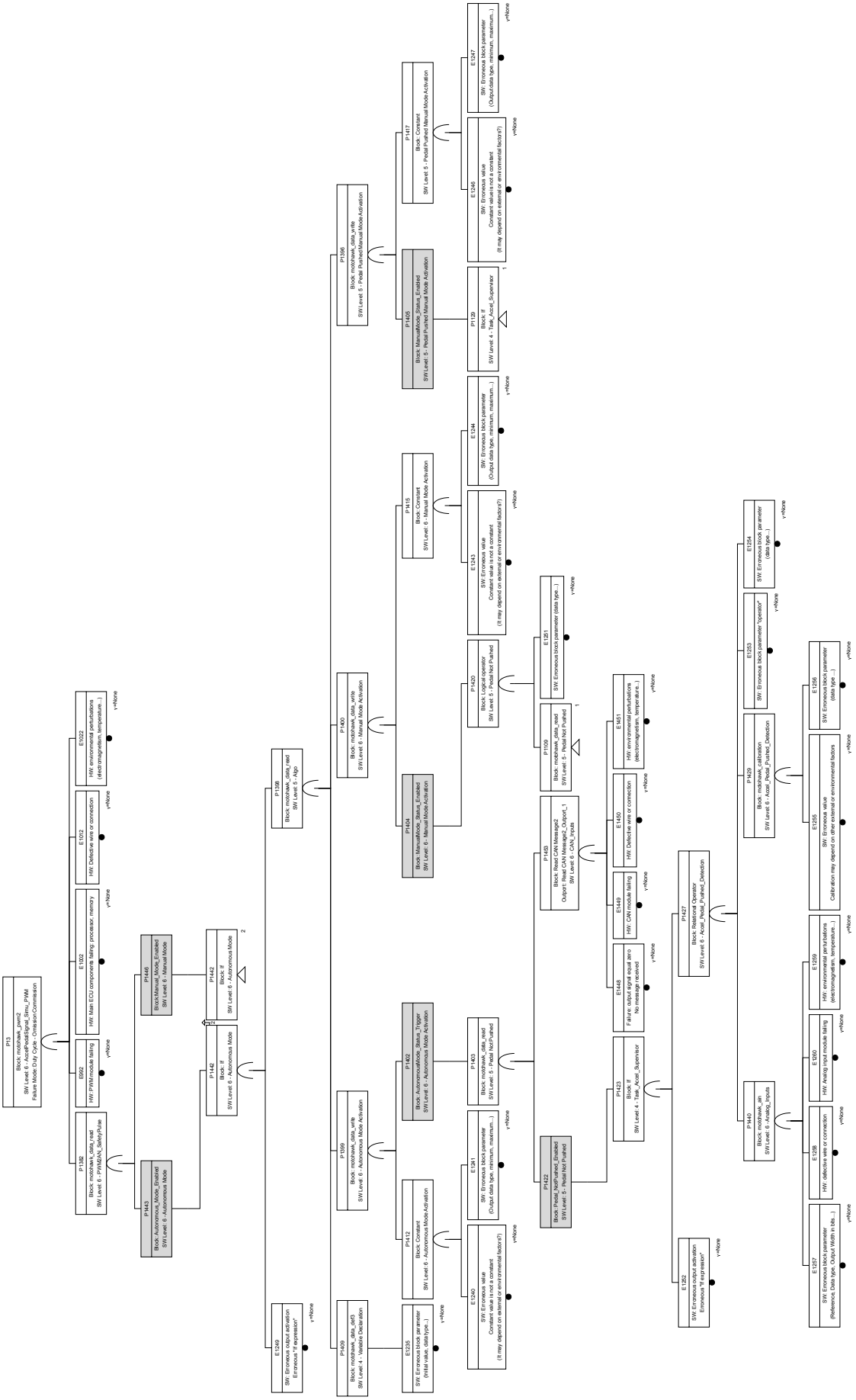


FIGURE C.12 – Arbre de défaillances coupe minimale : Omission ou commission du Duty cycle de la sortie Safety Pulse

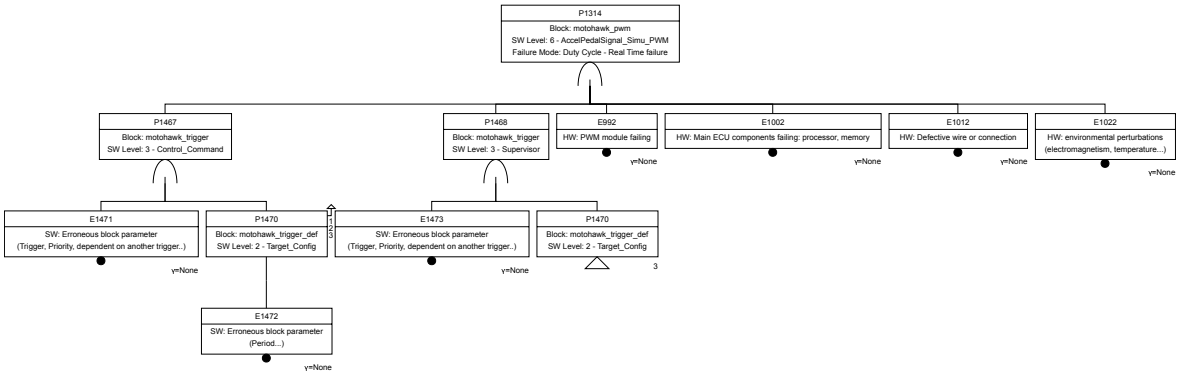


FIGURE C.13 – Arbre de défaillances coupe minimale : Défaillance temps réel du Duty Cycle de la sortie PWM1

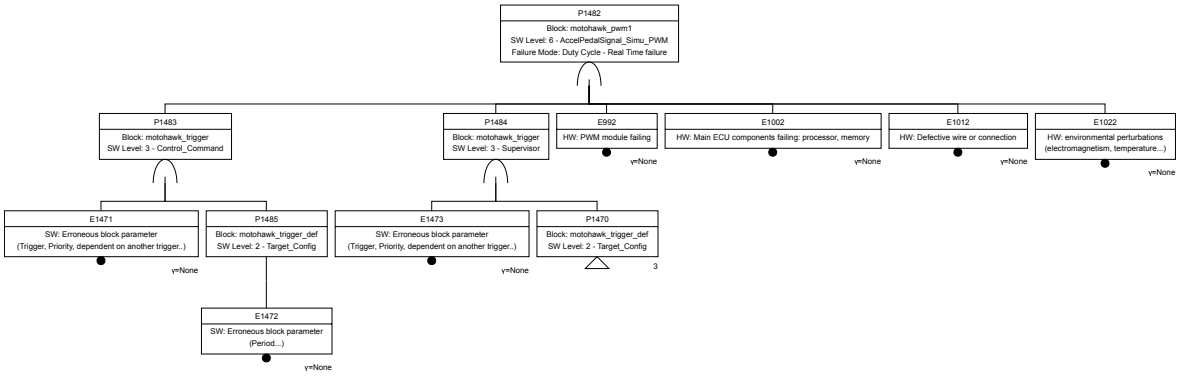


FIGURE C.14 – Arbre de défaillances coupe minimale : Défaillance temps réel du Duty cycle de la sortie PWM2

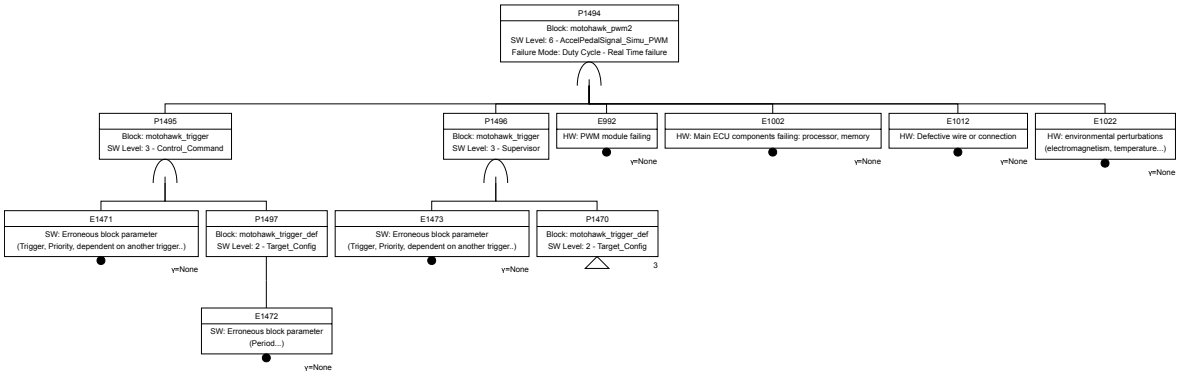


FIGURE C.15 – Arbre de défaillances coupe minimale : Défaillance temps réel du Duty cycle de la sortie Safety Pulse

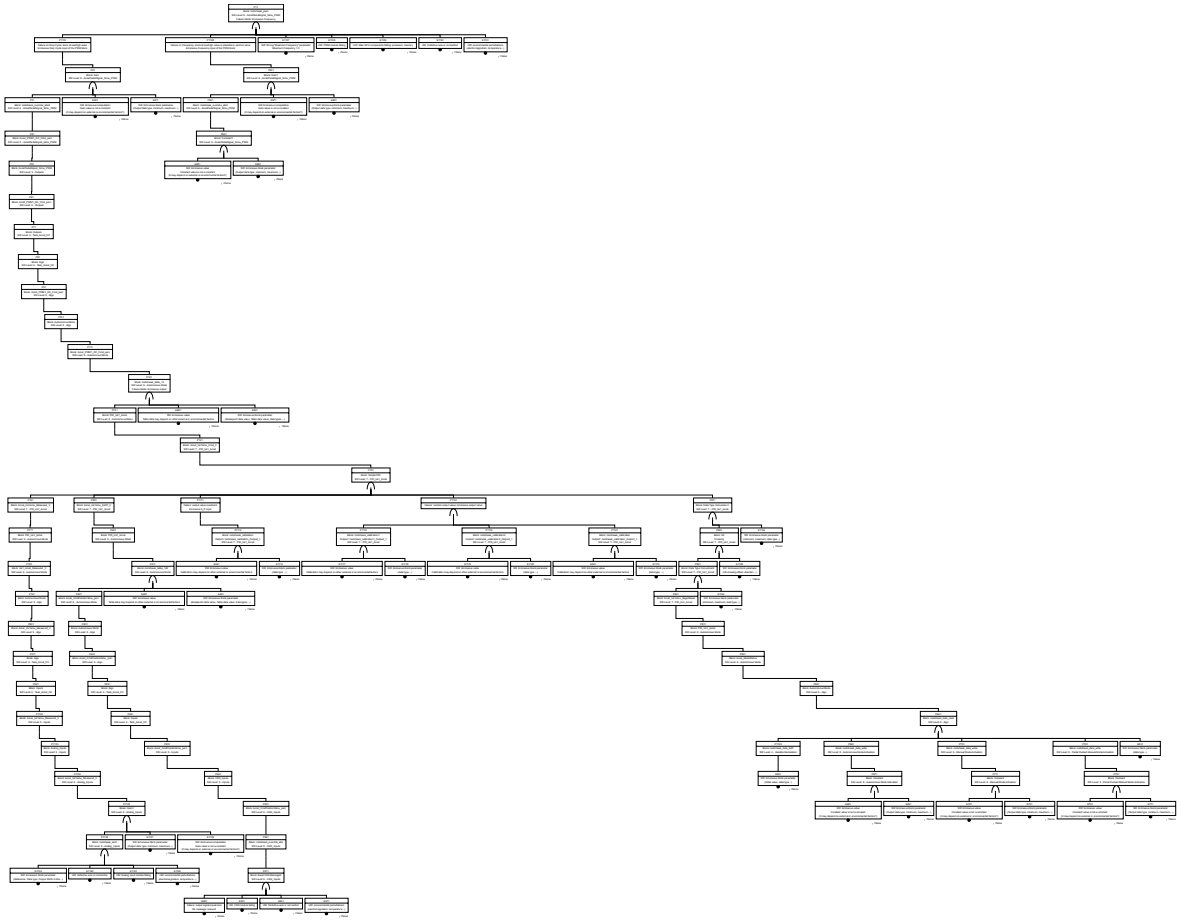


FIGURE C.16 – Arbre de défaillances complet : Fréquence erroné de la sortie PWM1

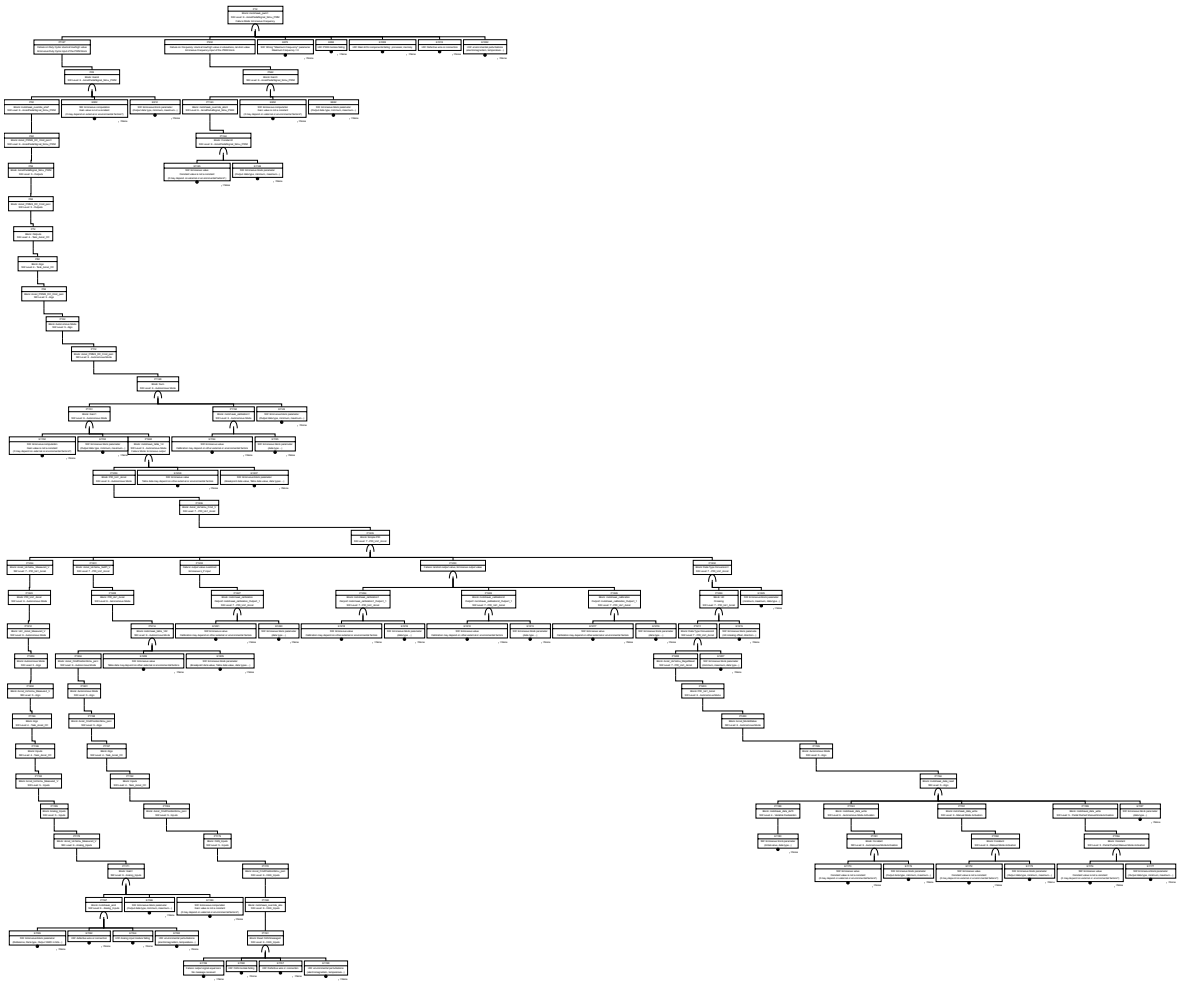


FIGURE C.17 – Arbre de défaillances complet : Fréquence erroné de la sortie PWM2

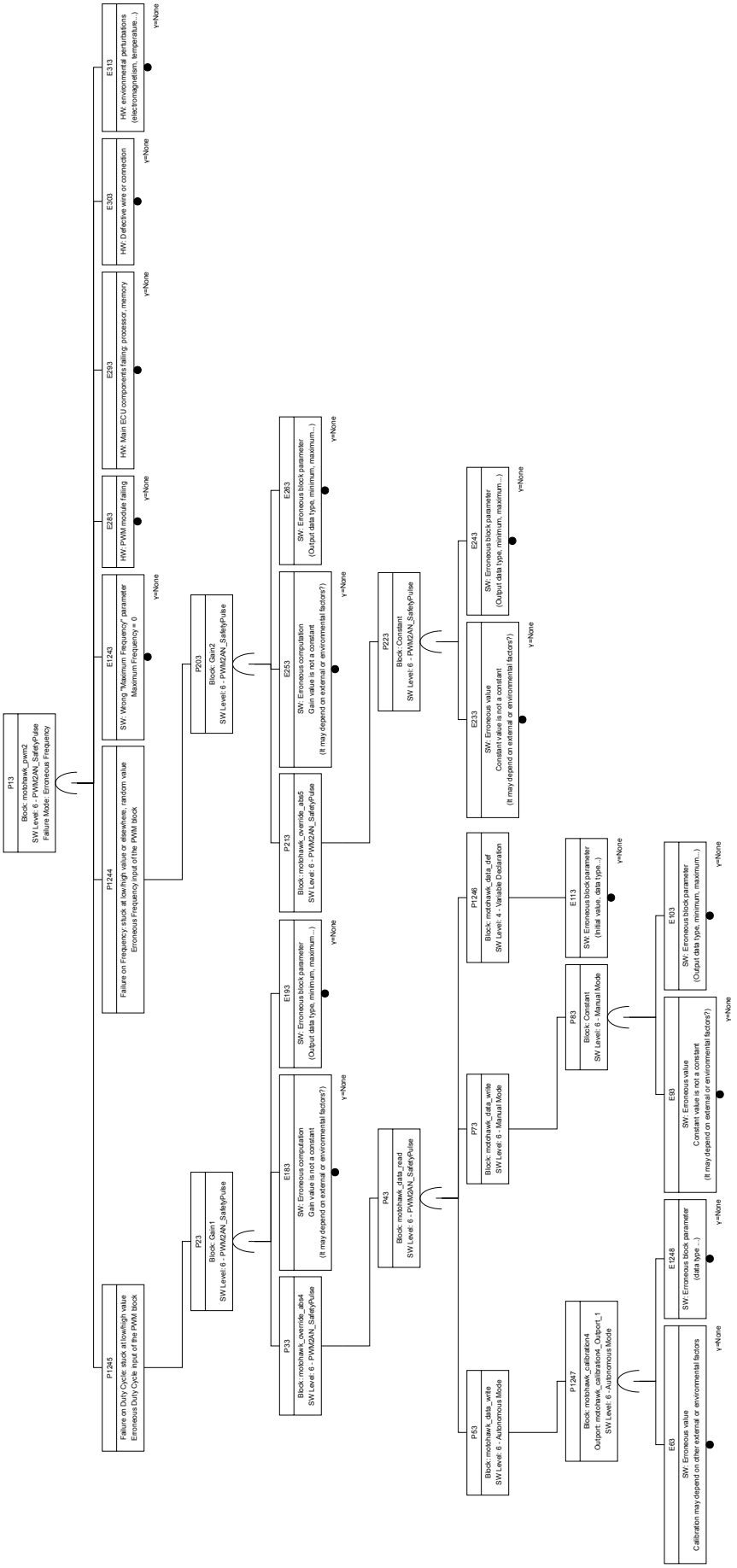


FIGURE C.18 – Arbre de défaillances complet : Fréquence erroné de la sortie Safety Pulse

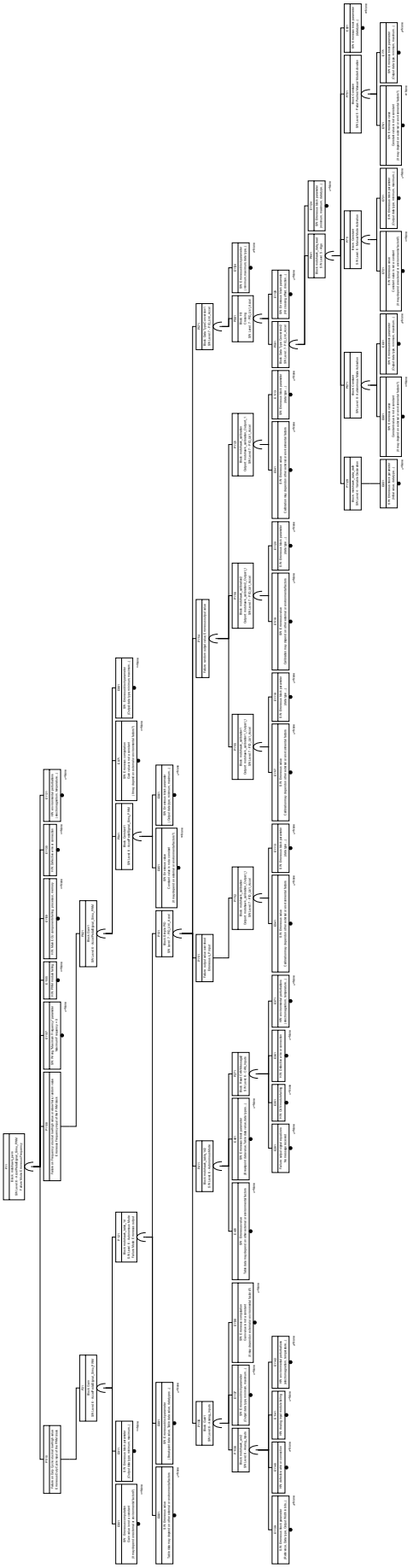


FIGURE C.19 – Arbre de défaillances coupe minimale : Fréquence erroné de la sortie PWM1

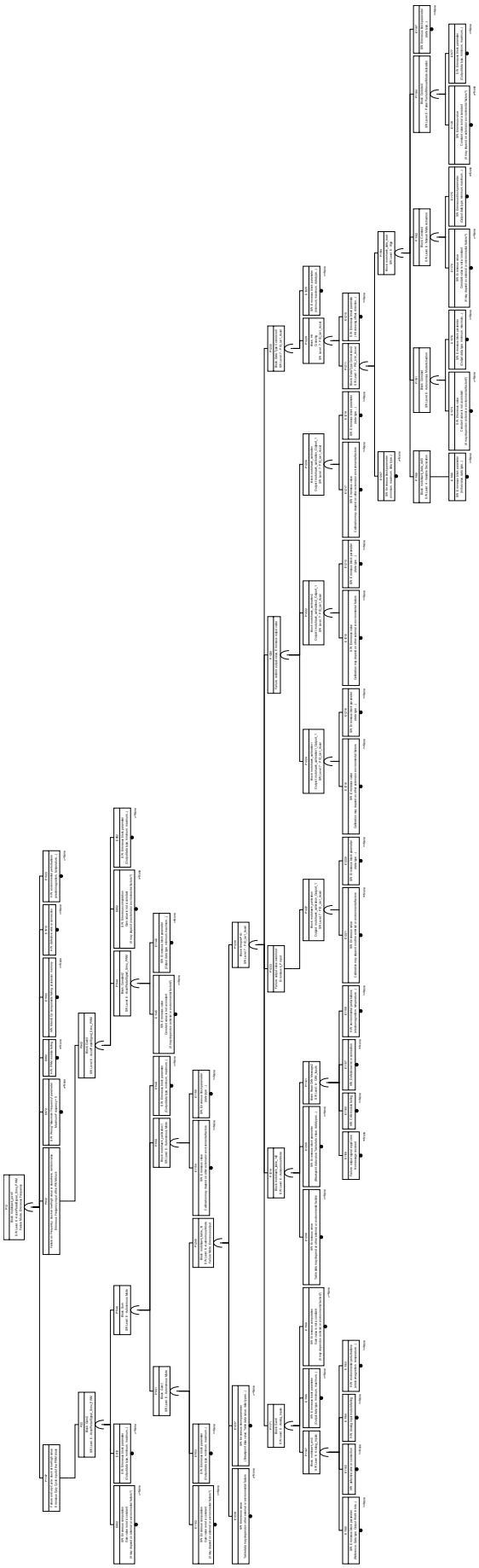


FIGURE C.20 – Arbre de défaillances coupe minimale : Fréquence erroné de la sortie PWM2

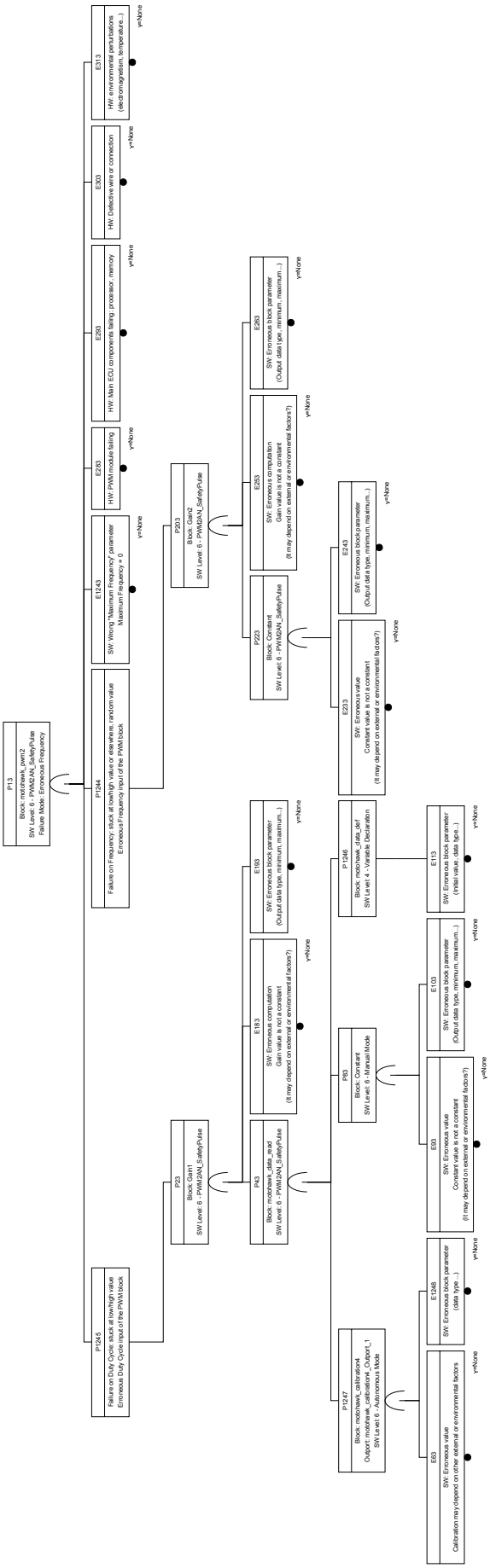


FIGURE C.21 – Arbre de défaillances coupe minimale : Fréquence erroné de la sortie Safety Pulse

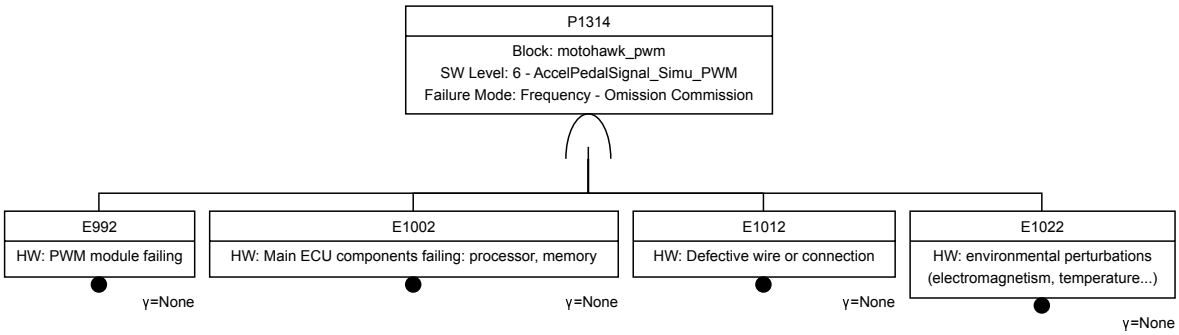


FIGURE C.22 – Arbre de défaillances coupe minimale : Omission ou commission de la fréquence de la sortie PWM1

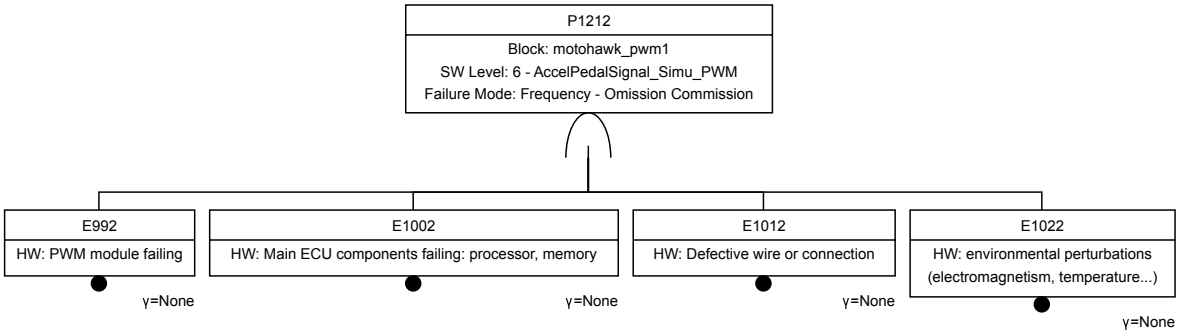


FIGURE C.23 – Arbre de défaillances coupe minimale : Omission ou commission de la fréquence de la sortie PWM2

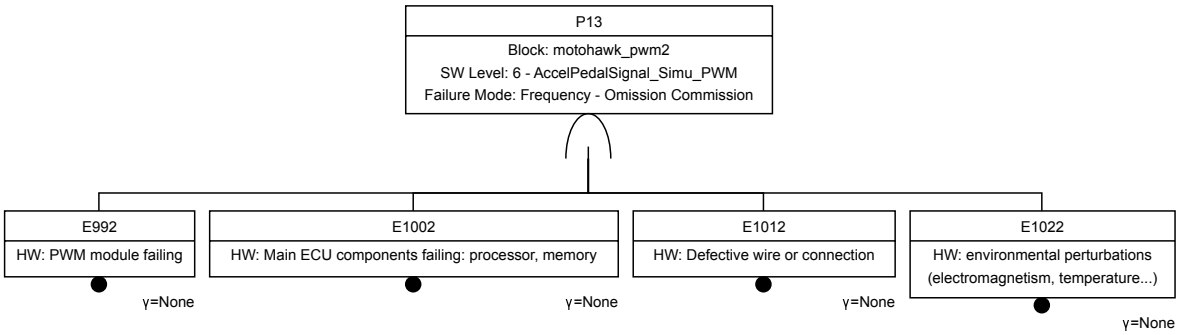


FIGURE C.24 – Arbre de défaillances coupe minimale : Omission ou commission de la fréquence de la sortie Safety Pulse

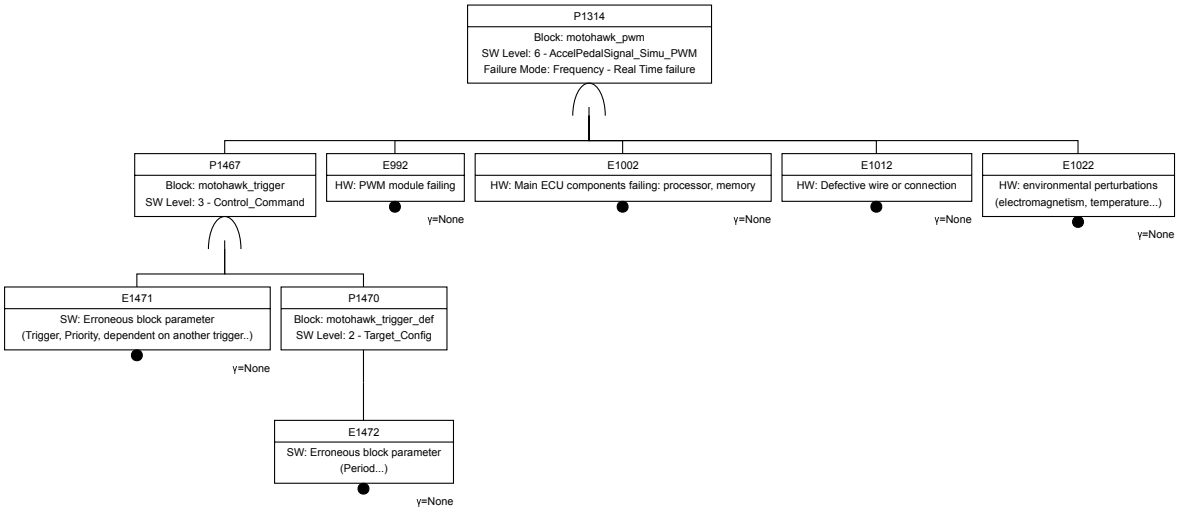


FIGURE C.25 – Arbre de défaillances coupe minimale : Défaillance temps réel de la fréquence de la sortie PWM1

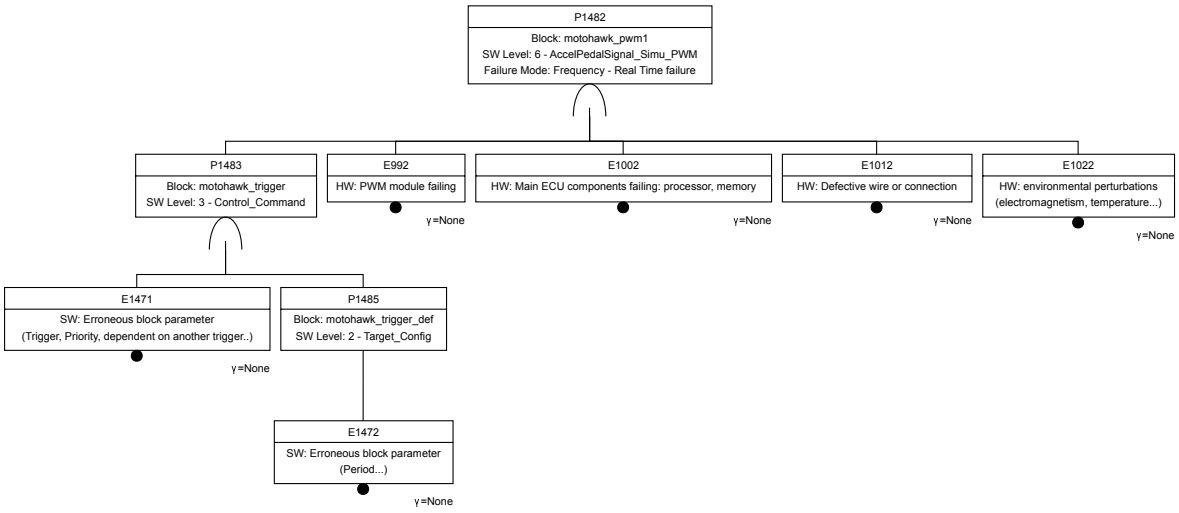


FIGURE C.26 – Arbre de défaillances coupe minimale : Défaillance temps réel de la fréquence de la sortie PWM2

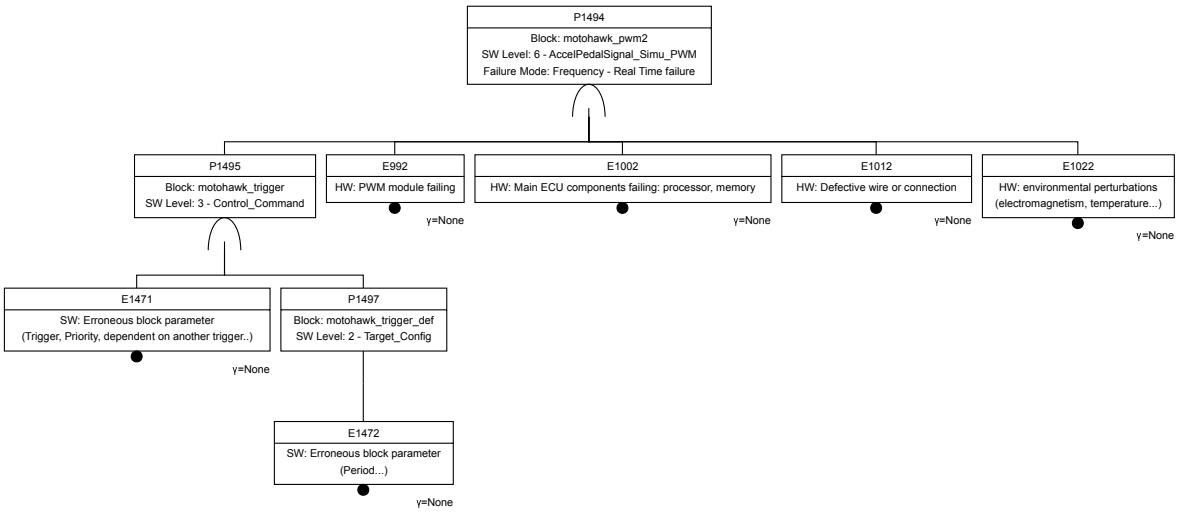


FIGURE C.27 – Arbre de défaillances coupe minimale : Défaillance temps réel de la fréquence de la sortie Safety Pulse

C.2 Tableau AMDE obtenu à partir du modèle prototype du cas d'étude de leurrage de la pédale d'accélérateur

Accelerator bypass - FMEA

Functions	Blocks	Potential failure modes	Potential causes	Potential effects (SW & ECU scope)	Potential effects on system	Risk level	Recommended actions	Applied actions
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Algo/Accel_PedalPushed_Detection	RelationalOperator	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty Cycle - Omission Commission motohawk_pwm1: Duty Cycle - Omission Commission motohawk_pwm2: Duty Cycle - Omission Commission			- Define Boolean data type	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Algo/Autonomous Mode	Gain1	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm1: Erroneous Duty Cycle ; Erroneous Frequency			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	
			Compilation issue	Output signal: motohawk_pwm1: Erroneous Duty Cycle ; Erroneous Frequency			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	
			Wrong gain value	Output signal: motohawk_pwm1: Erroneous Duty Cycle ; Erroneous Frequency			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Algo/Autonomous Mode/PID_Us1_Accel	Data Type Conversion1	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision, to optimize the memory occupation and the data type waiting by next block	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Algo/Autonomous Mode/PID_Us1_Accel	Data Type Conversion2	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision, to optimize the memory occupation and the data type waiting by next block	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Algo/Autonomous Mode	Sum	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Algo/Autonomous Mode	motohawk_table_1d	Erroneous value	Table data may depend on other external or environmental factors	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account	
			Erroneous block parameter (Breakpoint data value, Table data value, data types...)	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Apply tests to validate these parameters	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Algo/Autonomous Mode	motohawk_table_1d2	Erroneous value	Table data may depend on other external or environmental factors	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account	
			Erroneous block parameter (Breakpoint data value, Table data value, data types...)	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Apply tests to validate these parameters	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Algo/Manual Mode	Constant	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm2: Duty cycle erroneous			- Define data type according to the expected precision and to optimize the memory occupation	
			Wrong constant value	Output signal: motohawk_pwm2: Duty cycle erroneous			- Define the factors which could significantly influence the constant value. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value and use a table data	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Algo	motohawk_data_read	Erroneous value	Erroneous block parameter (DataType...)	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency omission/commission ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency omission/commission ; Frequency erroneous motohawk_pwm2: Duty cycle omission/commission			- Apply tests involving the entire operating range	
			Avoid multiple "Data write" blocks	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency omission/commission ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency omission/commission ; Frequency erroneous motohawk_pwm2: Duty cycle omission/commission			- Development rules: in case of multiple writing of the same value, try to factorize	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Inputs /Analog_Inputs	Gain	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	
			Compilation issue	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	

				omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous				
			Wrong gain value	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Inputs /Analog_Inputs	Gain1	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	
			Compilation issue	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	
			Wrong gain value	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Inputs /Analog_Inputs	Sum	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Inputs /Analog_Inputs	motohawk_ain	Signal stuck at low value (pull-down)	Input signal failed: Failed sensor or Defective wire/connection or disconnected or Short-Circuit to ground	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect value below the minimum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
			Failed ADC	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
		Signal stuck at high value (pull-down)	Input signal failed: Failed sensor or Short-Circuit to power	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect value above the maximum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	

				Failed ADC	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
			Signal stuck at low value (pull-up)	Input signal failed: Failed sensor or Short-Circuit to ground	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect value below the minimum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
				Failed ADC	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
			Signal stuck at high value (pull-up)	Input signal failed: Failed sensor or Defective wire/connection or disconnected or Short-Circuit to power	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect value above the maximum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
				Failed ADC	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D +

							W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
		Signal stuck elsewhere	Bad chosen "Data type" parameter	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- Apply tests covering the entire operating range	
			Bad "Output width in bits" value	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- Apply tests covering the entire operating range	
			Failed ADC	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
			Input signal failed: Failed sensor	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the sensor - Apply V + Implement a software strategy to detect stuck value (the value does not change for several execution). When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
		Random value In and Out the operating range	Failed ADC	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
			Input signal failed: Failed sensor or Environmental perturbations: EMC, temperature, humidity, vibration, altitude...	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the sensor - Apply V + Implement a software strategy to detect value out the operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure	

							to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
		Within range but wrong	Failed ADC	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
			Input signal failed: Failed sensor or Defective wire/connection or disconnected or Environmental perturbation	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle erroneous			- During test phase or maintenance, check the integrity of the sensor - Apply V + Add a redundancy signal to implement comparison strategy When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + D + W + implement a software strategy to define which signal is trustable if it is possible, the function can continue in a safamode by using this signal. if it is not possible, it requires a third signal to perform safemode(SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Inputs /Analog_Inputs	motohawk_ain2	Signal stuck at low value (pull-down)	Input signal failed: Failed sensor or Defective wire/connection or disconnected or Short-Circuit to ground	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect value below the minimum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
			Failed ADC	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
		Signal stuck at high value (pull-down)	Input signal failed: Failed sensor or Short-Circuit to power	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect	

				erroneous ; Frequency erroneous			value above the maximum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
			Failed ADC	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
		Signal stuck at low value (pull-up)	Input signal failed: Failed sensor or Short-Circuit to ground	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect value below the minimum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
			Failed ADC	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
		Signal stuck at high value (pull-up)	Input signal failed: Failed sensor or Defective wire/connection or disconnected or Short-Circuit to power	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the sensor, the wire and the connector and check the wire section - Apply V + Implement a software strategy to detect value above the maximum operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then	

							partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
			Failed ADC	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
		Signal stuck elsewhere	Bad chosen "Data type" parameter	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Apply tests covering the entire operating range	
			Bad "Output width in bits" value	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Apply tests covering the entire operating range	
			Failed ADC	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
			Input signal failed: Failed sensor	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the sensor - Apply V + Implement a software strategy to detect stuck value (the value does not change for several execution). When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
		Random value In and Out the operating range	Failed ADC	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In	

							case of multiple failure, switch in a safemode.	
			Input signal failed: Failed sensor or Environmental perturbations: EMC, temperature, humidity, vibration, altitude...	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the sensor - Apply V + Implement a software strategy to detect value out the operating range and add a saturation to avoid the fault propagation. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode	
		Within range but wrong	Failed ADC	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the ADC - Add "Output Creation Status" parameter and apply strategy to detect the bad status. When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + Add a redundancy signal and apply D + W to each signal + a comparison between both signals. If the comparison result is bad and the failure mode is detected for one of the signal then partially trust the other one by applying safemode (SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
			Input signal failed: Failed sensor or Defective wire/connection or disconnected or Environmental perturbation	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- During test phase or maintenance, check the integrity of the sensor - Apply V + Add a redundancy signal to implement comparison strategy When the error is detected then a diagnostic variable take the value 1 (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or message in a display - Apply V + D + W + implement a software strategy to define which signal is trustable if it is possible, the function can continue in a safamode by using this signal. if it is not possible, it requires a third signal to perform safemode(SafeMode is defined according to the function) - Apply V + D + W + SM + Add a third redundancy to implement a "Vote" strategy: if one signal failed, the function can continue according both others. In case of multiple failure, switch in a safemode.	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC/Inputs /CAN_Inputs	Read CAN Message2	No message received	Conflict between message periodicity or event and task periodicity or event	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Check the periodicity or event definition in DBC file and check the parent task and triggers. Apply tests involving different concerned triggers - Add Age count port to the block, it detects if no message is available. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
			Bus is disconnected, shorted, improprely terminated, the baud rate is incorrect or the bit error rate are high enough to cause hardware failures or defective Chip CAN	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- During test phase or maintenance, check the integrity of the wires and the connector and check the wire section - Apply V + Add the block CAN Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
		Message erratically received	Conflict between message periodicity or event and task periodicity or event	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency			- Check the periodicity or event definition in DBC file and check the parent task and triggers. Apply tests involving different concerned triggers - Add Age count port to the block, it detects if no message is available. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure	

				omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			to the user for instance: buzzer or a lamp in the dashboard or message in a display	
			Bus is improprely terminated or defective ECU input (Chip CAN) or environmental perturbations	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- During test phase or maintenance, check the integrity of the CAN Input (Chip) - Apply V + Add the block CAN Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a strategy of checksum computing between the sender and receiver nodes. Add also a counter which must stay synchronized between both nodes. If an error is detected, the system pass in a limp homemode - Apply V + D + W + SM + Add redundancy signal to be able to switch on an other input if this one failed. Thus the system can continue to function properly	
		Message rate is below stated minimum (for periodic message)	Conflict between message periodicity or event and task periodicity or event	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Check the periodicity or event definition in DBC file and check the parent task and triggers. Apply tests involving different concerned triggers - Add Age count port to the block, it detects if no message is available. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
			Bus is improprely terminated or defective ECU input (Chip CAN) or environmental perturbations	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- During test phase or maintenance, check the integrity of the CAN Input (Chip) - Apply V + Add the block CAN Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a strategy of checksum computing between the sender and receiver nodes. Add also a counter which must stay synchronized between both nodes. If an error is detected, the system pass in a limp homemode - Apply V + D + W + SM + Add redundancy signal to be able to switch on an other input if this one failed. Thus the system can continue to function properly	
		Signal value error (stuck, random...)	Bus is improprely terminated or defective ECU input (Chip CAN) or environmental perturbations	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Frequency erroneous ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- During test phase or maintenance, check the integrity of the CAN Input (Chip) - Apply V + Add the block CAN Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a strategy of checksum computing between the sender and receiver nodes. Add also a counter which must stay synchronized between both nodes. If an error is detected, the system pass in a limp homemode - Apply V + D + W + SM + Add redundancy signal to be able to switch on an other input if this one failed. Thus the system can continue to function properly	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/AccelPedalSignal_Simu_PWM	Constant1	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation	
			Wrong constant value	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous			- Define the factors which could significantly influence the constant value. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value and use a table data	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/AccelPedalSignal_Simu_PWM	Constant2	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation	
			Wrong constant value	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define the factors which could significantly influence the constant value. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value and use a table data	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/AccelPedalSignal_Simu_PWM	Gain	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	
			Compilation issue	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	
			Wrong gain value	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/AccelPedalSignal_Simu_PWM	Gain1	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	

			Compilation issue	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	
			Wrong gain value	Output signal: motohawk_pwm: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/AccelPedalSignal_Simu_PWM	Gain2	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	
			Compilation issue	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	
			Wrong gain value	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/AccelPedalSignal_Simu_PWM	Gain3	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	
			Compilation issue	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	
			Wrong gain value	Output signal: motohawk_pwm1: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/AccelPedalSignal_Simu_PWM	motohawk_pwm	Frequency stuck at zero	Wrong "Maximum Frequency" parameter (Maximum Frequency = 0)	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- Check during implementation. Apply tests	
			PWM module failling	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	
			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
		Frequency stuck at low/high value or elsewhere and random value	PWM module failling	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	
			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
		Duty cycle stuck at low/high value or elsewhere and random value	PWM module failling	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	

							- Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	
			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/AccelPedalSignal_Simu_PWM	motohawk_pwm1	Frequency stuck at zero	Wrong "Maximum Frequency" parameter (Maximum Frequency = 0)	Output signal: motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- Check during implementation. Apply tests	
			PWM module failling	Output signal: motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	
			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
		Frequency stuck at low/high value or elsewhere and random value	PWM module failling	Output signal: motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	
			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
		Duty cycle stuck at low/high value or elsewhere and random value	PWM module failling	Output signal: motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	

			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm1: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/PWM2AN_SafetyPulse	Constant	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation	
			Wrong constant value	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- Define the factors which could significantly influence the constant value. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value and use a table data	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/PWM2AN_SafetyPulse	Gain1	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	
			Compilation issue	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	
			Wrong gain value	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/PWM2AN_SafetyPulse	Gain2	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- Define data type according to the expected precision and to optimize the memory occupation. Replace the "gain" block with a "product"	
			Compilation issue	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- The "gain" block may produce wrong compilation. Replace the "gain" block with a "product"	
			Wrong gain value	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- Define the factors influencing the value described in the table data. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/PWM2AN_SafetyPulse	motohawk_data_read	Erroneous value	Erroneous block parameter (DataType...)	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- Apply tests involving the entire operating range	
			Avoid multiple "Data write" blocks	Output signal: motohawk_pwm2: Duty cycle erroneous ; Frequency erroneous			- Development rules: in case of multiple writing of the same value, try to factorize	
Accel_ByPass_TestBench/Model /Control_Command/Task_Accel_CC /Outputs/PWM2AN_SafetyPulse	motohawk_pwm2	Frequency stuck at zero	Wrong "Maximum Frequency" parameter (Maximum Frequency = 0)	Output signal: motohawk_pwm2: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- Check during implementation. Apply tests	
			PWM module failling	Output signal: motohawk_pwm2: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	
			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm2: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
		Frequency stuck at low/high value or elsewhere and random value	PWM module failling	Output signal: motohawk_pwm2: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	
			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm2: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and	

				real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
		Duty cycle stuck at low/high value or elsewhere and random value	PWM module failling	Output signal: motohawk_pwm2: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the integrity of the PWM module - Apply V + Add the outport Fault Status. If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display - Apply V + D + W + Add a redundancy signal to allow the receiver module to perform dignostic and pass in limphome mode (if possible send the fault status via CAN bus or discrete output) - Apply V + D + W + SM + Add a third redundancy to allow the receiver module to implement perform strategy to conserve all the performances	
			Main HW components failling (Processor, memory...)	Output signal: motohawk_pwm2: Duty cycle erroneous ; Duty cycle omission/commission ; Duty cycle real time ; Frequency erroneous ; Frequency omission/commission ; Frequency real time			- During test phase or maintenance, check the ECU integrity - Apply V + Add blocks to check the processor statistics (motohawk_idle_cpu and motohawck_stack_free) and the memory (). If the fault is detected a diagnostic variable take the value 1. (This variable will be checked during monitoring or maintenance). Implement this strategy in "diagnostic" part of the software model - Apply V + D + Add a mean to indicate the failure to the user for instance: buzzer or a lamp in the dashboard or message in a display	
Accel_ByPass_TestBench/Model /Supervisor/Task_Accel_Supervisor/Pedal Not Pushed/Autonomous Mode Activation	Constant	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Define data type according to the expected precision and to optimize the memory occupation	
			Wrong constant value	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Define the factors which could significantly influence the constant value. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value and use a table data	
Accel_ByPass_TestBench/Model /Supervisor/Task_Accel_Supervisor/Pedal Not Pushed	LogicalOperator	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Define Boolean data type	
Accel_ByPass_TestBench/Model /Supervisor/Task_Accel_Supervisor/Pedal Not Pushed/Manual Mode Activation	Constant	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Define data type according to the expected precision and to optimize the memory occupation	
			Wrong constant value	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Define the factors which could significantly influence the constant value. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value and use a table data	
Accel_ByPass_TestBench/Model /Supervisor/Task_Accel_Supervisor/Pedal Not Pushed	motohawk_data_read	Erroneous value	Erroneous block parameter (DataType...)	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Apply tests involving the entire operating range	
			Avoid multiple "Data write" blocks	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency			- Development rules: in case of multiple writing of the same value, try to factorize	

				omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission				
Accel_ByPass_TestBench/Model /Supervisor/Task_Accel_Supervisor/Pedal Pushed Manual Mode Activation	Constant	Erroneous output value	Wrong parameter data type	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Define data type according to the expected precision and to optimize the memory occupation	
			Wrong constant value	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Define the factors which could significantly influence the constant value. If necessary, add an input(s) to the SW which must take into account in the fluctuations of the value and use a table data	
Accel_ByPass_TestBench/Model /Supervisor/Task_Accel_Supervisor	motohawk_data_read1	Erroneous value	Erroneous block parameter (DataType...)	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Apply tests involving the entire operating range	
			Avoid multiple "Data write" blocks	Output signal: motohawk_pwm: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm1: Duty cycle omission/commission ; Frequency omission/commission motohawk_pwm2: Duty cycle omission/commission ; Frequency omission/commission			- Development rules: in case of multiple writing of the same value, try to factorize	

Annexe D

Publications et Institutions

Publications

Les travaux présentés dans cette thèse ont donné lieu aux publications suivantes :

- J. Godot, S. Saudrais, A. Alif, B. Barbedette, and C. Larouci, « Safety Analysis Generation from Prototyping Models for Transportation Systems », in *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC'16)*, Avril 2016, Pisa (Italy)
- J. Godot, A. Alif, B. Barbedette, and S. Saudrais, « Nouvelle méthodologie de développement de logiciels embarqués dans la phase de prototypage », in *Congrès Lambda Mu 20 de Maîtrise des Risques et de Sûreté de Fonctionnement*, Octobre 2016, Saint Malo (France)
- J. Godot, A. Alif, S. Saudrais, B. Barbedette, C. Larouci, « Safety analysis of heterogeneous software models at implementation stage », in *SAE World Congress eXperience (WCX 17)*, Avril 2017, Detroit (Michigan, USA)

FAAR Industry

FAAR Industry est une entreprise d'ingénierie française fondée en 2004. Ses activités sont principalement dédiées à l'industrie de la mobilité (automobile, aéronautique, ferroviaire et autres véhicules professionnels ou de loisir) et peuvent s'étendre à d'autres industries telles que la robotique, l'énergie ou encore le médical. FAAR Industry accompagne ses clients pour le développement de systèmes embarqués sur des projets pouvant aller de la preuve de concept jusqu'à de la production en petite série. Son savoir-faire concerne ainsi :

- la rédaction de cahier des charges et spécification
- la conception des architectures électriques et électroniques
- le développement de logiciels de contrôle
- l'intégration du système dans le véhicule
- le câblage
- le management de projet

Ainsi, FAAR Industry est notamment impliquée dans des projets innovants et d'actualité tels que les véhicules autonomes, propres et connectés.

C'est à la suite de l'apparition de la norme de SdF ISO 26262 dans l'automobile et des nouveaux besoins des clients qui se dirigent vers le développement de système d'aide à la conduite et plus globalement le véhicule autonome que FAAR Industy s'est intéressée à la problématique de la SdF et a ainsi initiée ce doctorat en partenariat avec l'ESTACA.

http://www.faar-industry.com/

ESTACA

L'École Supérieure de Techniques Aéronautiques et de Construction Automobile (ESTACA) est une école d'ingénieur dédiée à l'industrie des transports : automobile, aéronautique, ferroviaire et spatial. Elle accueille un laboratoire de recherche l'ESTACA'Lab qui s'organise autour de deux pôles :

- le pôle Systèmes et Énergie Embarqués pour les Transports (S2ET)
- le pôle Mécanique des Matériaux Composites et Environnement (2MCE)

Les projets menés par le pôle S2ET sont dédiés à l'électrification et l'hybridation des chaînes de traction et visent à rendre les véhicules plus propres, plus intelligents et plus sûrs. C'est donc avec cette entité que le doctorat a été mené.

http://www.estaca.fr