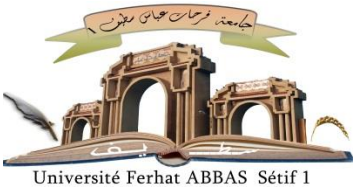
الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de L'Enseignement Supérieur et de la Recherche Scientifique



UNIVERSITÉ FERHAT ABBAS - SETIF 1

FACULTÉ DE TECHNOLOGIE

THESE

Présentée au Département d'Electronique

Pour l'obtention du diplôme de

DOCTORAT EN SCIENCES

Option : Electronique

Par

Benhamadouche Abdelouahab Djoubair

THÈME

Méthodologie et outil d'aide à la conception des systèmes multi-technologiques par VHDL

Soutenue le 26/02/2020 devant le Jury :			
HASSAM Abdelouahab	**Professeur**	**Univ. Ferhat Abbas Sétif 1**	**Président**
BOUKEZZOULA Naceur-Eddine	**Professeur**	**Univ. Ferhat Abbas Sétif 1**	**Directeur**
KERROUR Fouad	**Professeur**	**Université Constantine 1**	**Examinateur**
ZIET Lahcene	**Professeur**	**Univ. Ferhat Abbas Sétif 1**	**Examinateur**
BELHANI Ahmed	**MCA**	**Université Constantine 1**	**Examinateur**
MESSAOUDI Noureddine	**MCA**	**Université de Boumerdes**	**Examinateur**
DJAHLI Farid	**Professeur**	**Univ. Ferhat Abbas Sétif 1**	**Invité**

DEDICACES

A ma Mère

A toute ma famille

A mes amis

REMERCIEMENTS

A l'issue de ce travail, je tiens à adresser ma reconnaissance et mes remerciements à toutes les personnes qui ont contribué, chacune à sa manière, à l'accomplissement de cette thèse.

J'adresse tous mes remerciements à Monsieur Boukezzoula Naceur-Eddine, Professeur à l'Université de Sétif 1, d'avoir accepté d'être mon directeur de thèse.

Je tiens à remercier très chaleureusement Monsieur Djahli Farid, Professeur à université de Sétif 1, qui m'a encadré tout au long de cette thèse et qui m'a fait partager ses brillantes intuitions. Qu'il soit aussi remercié pour sa gentillesse, sa disponibilité permanente et pour les nombreux encouragements qu'il m'a prodiguée.

Je tiens à remercier vivement les membres de mon jury de thèse pour avoir accepté d'évaluer mon travail : Monsieur Hassam Abdelouahab, Professeur à l'Université de Setif1, d'avoir accepté de juger mon travail et de présider le jury de soutenance de cette thèse. Messieurs Kerrour Fouad, Professeur à Université Constantine 1, Ziet Lahcene, Professeur à l'Université de Sétif 1, Belhani Ahmed Maître de conférences à Université Constantine 1, Messaoudi Noureddine Maître de conférences à l'Université de Boumerdes, pour m'avoir fait l'honneur d'accepter d'être les examinateurs de cette thèse.

Il m'est impossible d'oublier Rabhi Abdou, Sahli Abdesslem et Ballouti Adel pour leur aide précieuse et leur soutien durant toutes ces années.

TABLE DES MATIERES

LISTE DES FIGURES

Chapitre 2 : État de l'art sur la conception système

Chapitre 3 : Méthodologie proposée pour la conception système

Chapitre 4 : Plateforme proposée pour la simulation C-HIL

Chapitre 5 : Cas d'étude, application et test

LISTE DES TABLEAUX

GLOSSAIRE

AFIS	Association française d'Ingénierie Système
AMBA	Advanced Microcontroller Bus Architecture
AMS	Analog and Mixed-Signal
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
CAN	Convertisseur Analogique Numérique
CHIL	Controller Hardware-In-the-Loop
CNA	Convertisseur Numérique Analogique
CUT	Circuit Under Test
DLL	Dynamic Link Library
DOD	American Department of Defence
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GMII	Gigabit media independent interface
HDL	Hardware Description Language
HIL	Hardware-in-the-loop
IAO	Ingénierie Assistée par Ordinateur
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INOSE	International Council on Systems Engineering
IP	Intellectual property
IS	Ingénierie Système ou Ingénierie des Systèmes
ISO	International Organization for Standardization
LAN	Local Area network
LUT	Look-Up Table
MAC	Media Access Controller
MBSE	Model Base System engineering

MDIO	Management Data Input/output
MEMS	Micro-Electro-Mechanical System
MII	Media Independent Interface
MIL	Model-in-the-loop
OSI	Open Systems Interconnection
PHIL	Power Hardware-In-the-Loop
PID	Proportional Integral Derivative
PID	Proportionnel Intégral Dérivé
PWM	Pulse Width Modulation
RMII	Reduced media-independent interface
SIL	Model-In-the-Loop
SOC	System On Chip
SysML	System Modeling Language
TCP	Transmission Control Protocol
UCF	User Constraint File
UDP	User Datagram Protocol
UML	Unified Modeling Language
USB	Universal Serial Bus
V&V	Vérification et Validation
VHDL	VHSIC Hardware Description Language
VHDL-AMS	VHDL Analog and Mixed Signal

Chapitre 1

Introduction

1 INTRODUCTION

Les travaux entrepris dans cette thèse portent sur la contribution au développement de méthodes et d'outils dédiés à la conception des systèmes multi-technologiques. Avant d'exposer les détails qui s'y rapportent, il est plus que nécessaire de définir le contexte de nos travaux. Dans ce chapitre, nous allons dans un premier temps introduire le contexte et les tendances actuelles de la conception système, ensuite nous allons présenter la problématique de recherche liée aux différents aspects techniques et structurels mis en jeu dans la mise en œuvre des nouveaux systèmes d'ingénierie. Par ailleurs, nous allons identifier les motivations, les objectifs et la contribution de notre travail de recherche.

2 CONTEXTE ET PROBLEMATIQUES

De nos jours, l'innovation technique et technologique s'est étendue sur toutes les facettes de notre vie quotidienne, cette innovation se manifeste par des produits fortement intégrés et de plus en plus complexes. Tiré par la demande croissante d'un marché en plein effervescence d'avantage de fonctionnalités sont embarqués dans ces produits. Ces produits ont évolué vers des systèmes multifonctionnels qui requièrent durant leur conception l'implication de divers secteurs d'applications scientifiques et industrielles. De ce fait, l'hétérogénéité de ces systèmes est devenue inéluctable, où un seul système est composé d'un ensemble lié d'unités fonctionnelles de diverses natures multidisciplinaires et multi-technologiques.

En effet, un système multi-technologique ou système multidisciplinaire est un système d'ingénierie issu de l'assemblage de sous-systèmes en interaction er de composants technologiques issus de différentes disciplines d'ingénierie, autrement dit ces systèmes associent, du point de vue fonctionnel et physique, diverses disciplines telles que la mécanique, l'électronique, l'automatique et l'informatique. Dans notre axe de recherche, qui porte sur la conception des systèmes d'ingénierie dans le domaine du génie électrique et de l'électronique, il existe une multitude de systèmes qui sont issues de la fusion de plusieurs disciplines technologiques, nous pouvons les classés selon les catégories suivantes :

- Système mécatronique (microsystème, MEMS, MOEMS…).
- Système sur puce (SOC, VLSI, ASIC…etc.).
- Système embarqué (Cyber Physical System CPS).
- Système de commande industriel.
- Système de conversion de la puissance électrique.

Une conséquence à l'hétérogénéité de ces systèmes est que leur processus de conception est divisé entre différentes méthodologies et outils de développement. Dans le cycle de vie d'un produit technologique la conception technique est le pilier de la mise en œuvre d'un produit fonctionnel et efficient. La conception de ces systèmes requiert des compétences multidisciplinaires nécessitant la collaboration de plusieurs groupes de différentes spécialités. Ces groupes ont leurs propres

méthodologies de travail et leurs propres outils de conception qui sont spécifiques à un domaine particulier.

D'autre part, l'évolution rapide des marchés concurrents exige la diminution du temps de développement d'un nouveau produit en gardant la qualité et les performances du système. Il devient donc nécessaire d'augmenter l'efficacité du processus de conception. Pour répondre à cette situation, les concepteurs de systèmes ont donc grand besoin d'outils permettant un développement rapide et à moindre coût. Ces attentes devront trouver une réponse dans des méthodologies et des outils de conception spécialement pensés pour les systèmes multidisciplinaires, qui doivent être unifiés et flexibles d'un côté et spécialisés de l'autre, et qui facilitent l'intégration de solutions analogiques, numériques, mixtes, matérielles et logicielles dans une approche interdisciplinaire.

Il n'existe pas une méthodologie universellement reconnue et répondant aux problématiques de conception des systèmes multidisciplinaires, de telles méthodologies s'appuient essentiellement sur un cahier des charges ad-hoc [1]. Cependant, nous devons adopter une méthodologie qui puisse être appliquée à tous les niveaux d'abstraction et qui puisse capturer les contraintes de conception et les composants à chaque niveau [2]. En outre, la méthodologie doit privilégier une vue système transcendante de la conception, de sorte qu'elle puisse offrir une productivité accrue et gérer plusieurs objectifs de conception à la fois. Ainsi, une méthodologie efficace doit respecter un certain nombre de pratiques, notamment :

- Favoriser l'exploration de l'espace architectural.
- Passage automatique dans le processus de conception de la spécification à la mise en œuvre.
- Rester indépendant le plus longtemps possible vis à vis de la technologie de mise en œuvre.
- Réutilisation des designs ; c'est la capacité d'utiliser des designs préexistants ou d'anciens systèmes.
- Prise en compte des contraintes matérielles au plus tôt dans le processus de conception.
- Vérification et évaluation du système dès les premières phases du processus de conception.
- Intégration de la vérification formelle des propriétés de la conception au fur et à mesure de la mise en œuvre.
- Utiliser les outils de conception assistée par ordinateur (CAO) pour les tâches répétitives et laborieuses.
- L'utilisation de techniques telles que le codesign, la simulation et la conception coopérative.

Pour renforcer cette démarche méthodologique, des outils d'aide à la conception sont nécessaires. La modélisation et la simulation sont l'un des outils les plus efficaces à la disposition des développeurs de systèmes complexes. Ces outils consistent en la construction de modèles représentatifs d'un système réel et à conduire différentes expériences sur ce modèle afin de comprendre le comportement de ce système et d'en améliorer les performances. La modélisation et la simulation nous permettent de créer un prototype entièrement numérique, ce qui permet de comprendre et d'optimiser les interactions essentielles entre la physique, les contrôles et l'environnement, au cours du processus de développement du produit.

La modélisation dans la conception des systèmes multidisciplinaires est devenue essentielle pour gérer les besoins élevés en matière de complexité, de fiabilité et de temps de mise sur le marché. La modélisation s'appuie sur la décomposition de la complexité et fait apparaître des sous-ensembles qui mobilisent chacune des technologies différentes. Le décloisonnement des univers technologiques est donc nécessaire à la mise en place d'une démarche de modélisation pertinente.

De plus, face à la complexité croissante des systèmes multi-technologiques, les procédures de validation, de vérification et de test (VV&T) sont devenues des pratiques indispensables pour l'évaluation et la qualification de nouveaux produits ainsi qu'au succès de la conception système [3]. Ces pratiques accompagnent chaque étape du cycle de développement pour prévenir l'occurrence d'erreurs ou de mauvais comportement [4]. Différentes techniques peuvent être utilisées pour exécuter ces procédures, de manière générale cela dépend de l'architecture du système et de sa mise-en-œuvre.

En outre, de nouvelles techniques de test telles que les techniques Hardware-In-the-Loop (HIL) ont émergé dans le monde de l'industrie et dans le domaine de la recherche scientifique pour être un moyen efficace pour la validation des systèmes complexes. Ces techniques viennent en remplacement aux pratiques de test qui nécessitent la construction de prototypes réels, ces techniques sont largement utilisées mais restent très couteuses et souvent hasardeuses.

3 MOTIVATION, OBJECTIFS ET CONTRIBUTION

En raison de la grande complexité et de la grande hétérogénéité des systèmes multi-technologique, beaucoup d'aspects technologiques et organisationnels rendent le développement de ces systèmes plus difficile et plus exposé à l'échec. Dans ce qui suit, nous allons identifier quelques obstacles qui trouble la résolution de ces problèmes.

- Les solutions proposées sont toujours dépendantes des outils propriétaires et ne font pas l'unanimité dans les différents domaines impliqués dans la conception d'un système particulier.
- L'entreprise technologique ne cherche pas de nouvelles solutions qui peuvent bouleverse son organisation technique.
- Les différents groupes disciplinaires qui interagissent lors de la conception système peinent à communiquer ensemble de manière efficace et utile.
- Les techniques de vérification sont faiblement exploitées pour le cas des systèmes multidisciplinaires, particulièrement dans les étapes post intégration.
- Les techniques de simulation HIL restent assez cher pour des projets à petit budget.
- La recherche sur la conception multidisciplinaire s'est concentrée sur l'aspect discret et l'aspect analogique d'un système, mais pas simultanément sur les deux ni sur leur cointégration.

Dans le domaine du génie électrique et de l'électronique la conception système exige différentes compétences dans différentes disciplines de l'ingénierie, ce qui implique une grande dépendance vis-

à-vis différents outils de développement avec des approches de conception différentes. Par conséquent, un nombre croissant d'outils utilisés dans la conception d'un système peut rendre la conception de plus en plus complexe. Dans cette thèse, nous nous intéressons en particulier à la conception des systèmes de conversion de la puissance électrique, la conception de ces systèmes implique un large éventail de domaines techniques tels que l'électronique, la thermique, l'électromagnétique, la programmation, etc.

Ce travail vise à améliorer la méthodologie de conception de ces systèmes en réduisant les délais de développement et en simplifiant les tâches d'ingénierie, à travers l'utilisation de méthodes et d'outils de conception unifiés et interdisciplinaires. Cette méthodologie doit couvrir une partie des tâches d'ingénierie système à partir de l'expression des besoins jusqu'au prototypage, et en passant par l'analyse des besoins et la construction de modèles avec des vues statiques et dynamiques.

Dans une approche multidisciplinaire, la conception système doit se baser sur des outils logiciels et une méthodologie adaptée. D'où la nécessité d'un processus de conception harmonisé capable d'effectuer des tâches d'ingénierie telles que la simulation multi-abstraction, le prototypage fonctionnel, la vérification et la validation. En parallèle, un outil de développement amont est nécessaire pour accélérer chaque étape de la conception, sans perdre en précision et en fiabilité, et qui aide à la construction d'un prototype sûre et opérationnel.

Dans ce travail, nous présentons une nouvelle méthodologie qui simplifie et accélère le processus de développement système. Dans cette méthodologie, le langage de description matériel VHDL et son extension AMS sont utilisés comme support pour la modélisation et la simulation de toutes les parties d'un système multi-technologique ; L'idée est d'utiliser les mêmes outils de modélisation pour concevoir la partie numérique et la partie analogique d'un système multi-technologique.

Pour adapter la vérification et la validation du système aux différentes phases de la conception nous déploierons l'utilisation de la modélisation VHDL-AMS à l'implémentation matérielle et à l'intégration des systèmes numériques sur les circuits reconfigurables FPGA. Enfin, nous proposons l'utilisation de la technique de simulation Hardware-In-the-Loop en associant les mêmes outils de modélisation ainsi que des cartes de développement FPGA. Nous considérons cette approche comme une extension inéluctable de la modélisation avec le langage VHDL-AMS.

Pour atteindre avec succès ces objectifs de recherche, il est indispensable que nous explorions les points suivants :

- ✓ Etudier et analyser les approches existantes pour la conception des systèmes multidisciplinaires qui permettent l'interaction entre plusieurs méthodes de conception et de modélisation.
- ✓ Proposer une méthode de conception basée sur la modélisation et sur la simulation multi-domaines permettant une vision globale et modulaire à partir du langage VHDL-AMS.
- ✓ Mettre en œuvre un outil qui permet de réaliser des simulations Hardware-In-the-Loop en se basant sur l'utilisation des circuits FPGA.

- ✓ Valider l'approche proposée et la technique de simulation introduite à travers des cas d'étude appliqués aux systèmes de conversion de la puissance électrique.

4 STRUCTURE DU MEMOIRE DE THESE

Ce mémoire de thèse s'articule autour de cinq chapitres :

Dans le **chapitre 2**, les notions impliquant les systèmes multidisciplinaires et la complexité de ces systèmes sont exposées. Ensuite, différentes démarches de conception système sont présentées en précision les étapes clés du processus de développement technique auxquelles nous nous intéressons. Dans notre approche de conception la modélisation est une étape cruciale pour le succès de la mise en œuvre système, partant de ce constat nous détaillerons son utilisation ainsi que les outils qui permettent son adhésion dans le processus de vérification et de validation système. Ensuite, nous détaillerons les techniques de simulation qui permettront une meilleure maitrise de l'évolution du processus de développement système.

Le **chapitre 3** est dédié à la présentation de notre méthodologie de conception des systèmes multidisciplinaires. Nous exposerons, les éléments nécessaires pour la mise en œuvre de notre approche, ainsi que les détails de la modélisation et de la simulation à travers l'association des langages de description matérielle VHDL-AMS/VHDL. Nous détaillerons par la suite le processus et la méthode de conception associés à notre approche, nous établirons les règles méthodologiques qui permettent de réussir chaque étape de conception.

Le **chapitre 4** traite essentiellement de la mise en œuvre d'une plateforme expérimentale dédiée à la simulation Hardware-In-the-Loop, cette plateforme est basée sur le prototypage FPGA ainsi que sur la modélisation VHDL-AMS. Les principaux éléments de construction de cette plateforme seront présentés ainsi que les différentes techniques utilisées pour son implémentation logicielle et matérielle.

Le **dernier chapitre** est consacré à l'illustration par un cas d'application pratique pour l'utilisation de notre approche ainsi que la plateforme développée. Le but de cette dernière partie est de traiter un exemple pratique de système de conversion de la puissance, c'est une application qui permet d'éclairer et d'illustrer de la meilleure façon possible les étapes de conception et d'implémentation déployée.

Finalement, le manuscrit s'achèvera par une conclusion générale et des perspectives pour entreprendre de nouveaux projets de recherche et de développement.

Chapitre 2

État de l’art sur la conception système

5 INTRODUCTION

Aujourd'hui, pour faire face à la complexité grandissante des systèmes d'ingénierie et particulièrement des systèmes électroniques, plusieurs méthodes et outils de conception ont vu le jour. En effet, la conception de ces systèmes nécessite l'association de plusieurs disciplines d'ingénierie (mécanique, optique, thermique, etc.), et par conséquent l'utilisation de plusieurs environnements de développement qui sont souvent incompatibles les uns avec les autres. S'ajoute à cette problématique des contraintes de coût, de performance et de temps de mise sur le marché.

Dans ce chapitre, nous présenterons les tendances actuelles pour la mise-en-œuvre de ces systèmes multidisciplinaires. Cet état de l'art n'a pas la prétention d'être exhaustif ; il a pour but de mettre en évidence les principaux éléments qui interviennent dans notre démarche de recherche. Dans un premier temps nous commencerons par définir l'ensemble du contexte de l'ingénierie système, nous donnerons par la suite un état des lieux des objectifs de l'ingénierie système et des pratiques existantes pour la conception technique, en particulier l'ingénierie système basée sur des modèles MBSE (Model Based System Engineering).

Etant donné que la modélisation est un facteur important dans notre démarche de conception, nous allons présenter les éléments-clés de cette phase essentielle de la conception système. Du coup, nous aurons aussi à exposer les principes de la simulation qui est profondément liée à la modélisation. Nous détaillerons par la suite quelques procédures de simulation, notamment la simulation Hardware-In-the-Loop qui sera un des moyens majeurs dédiés aux phases de vérification et de validation des systèmes étudiés.

1 L'INGENIERIE DES SYSTEMES

Cette section porte sur la présentation du processus actuel d'ingénierie système (IS). Elle commence par fournir des informations générales sur l'ingénierie système ainsi que les étapes génériques impliquées dans le processus de développement.

1.1 C'EST QUOI UN SYSTEME

Il existe un grand nombre de définitions du terme « système ». Nous allons voir quelques-unes afin de mettre en évidence les termes communs et leurs différences.

Selon « Charles S. Wasson » [5]

> Un système est un ensemble intégré d'éléments interopérables, chacun des éléments a des fonctionnalités explicitement spécifiées et limités, travaillant en synergie pour effectuer un traitement à valeur ajoutée permettant à un utilisateur de satisfaire les besoins opérationnels axés sur un objectif dans un environnement d'exploitation prescrit avec un résultat et une probabilité de réussite spécifiés.

D'autre définitions plus concises ont été intégrées dans les normes de l'ingénierie système telles que :

Selon l'INCOSE (International Council On Systems Engineering) [6]

> Un système est un ensemble d'éléments interdépendants qui interagissent entre eux de façon organisée et forment un ensemble unique.

Selon l'IEEE [7]

> Groupe de personnes, d'objets et de processus interdépendants, constitué dans le but de réaliser des objectifs définis ou de remplir un certain rôle opérationnel, par l'exécution de fonctions précises.

Selon la NASA [8]

> Un système est un ensemble de composants inters-reliés qui interagissent les uns avec les autres d'une manière organisée pour accomplir une finalité commune.

De ces différentes définitions et bien d'autres, nous retenons que :

> Un système est un ensemble uni de composants interdépendants qui collaborent à la réalisation d'un ensemble de tâches en vue de fournir un ensemble de services, cet ensemble est fonctionnel dans un environnement spécifique et interagit ainsi avec les éléments de cet environnement.

1.2 DEFINITION ET OBJECTIFS DE L'INGENIERIE SYSTEME (IS)

Depuis plusieurs décennies déjà, de nouvelles méthodes de conception ont été développées pour assister la mise-en-œuvre des systèmes complexes. L'ingénierie système est la première à être normalisée, elle comprend deux disciplines importantes : le domaine de la connaissance technique dans lequel opère l'ingénierie système, et le domaine de la gestion de l'ingénierie des systèmes [9].

Une synthèse de plusieurs définitions trouvées en littératures peut être faite pour définir l'IS. L'ingénierie système est une démarche méthodologique et des moyens interdisciplinaires qui contribuent à un projet collaboratif, et qui assurent l'ensemble des activités adéquates et équilibrées tout au long du cycle de vie pour concevoir, faire évoluer et vérifier un système apportant une solution économique et performante aux besoins d'un client, tout en satisfaisant l'ensemble des parties prenantes.

Dans le cadre du développement d'un produit et de son processus de fabrication associé, les objectifs de l'ingénierie système sont :

- ✓ Définir des méthodes et des moyens permettant de satisfaire au mieux les besoins techniques et organisationnels ;

- ✓ Rechercher l'équilibre entre les exigences, les contraintes, les performances, les coûts, les délais et les risques du projet ;
- ✓ Identifier, structurer et organiser les activités techniques ;
- ✓ Maîtriser la complexité liée à l'interaction de multiples composants du système ;
- ✓ Assurer la compatibilité fonctionnelle et organique du système avec ces exigences ;
- ✓ Ajuster les activités techniques aux besoins ;
- ✓ Maîtriser les informations nécessaires à la réalisation du système ;
- ✓ Réduire les délais de développement ;
- ✓ Faciliter la réingénierie du produit ;
- ✓ Une meilleure anticipation des problèmes et des risques concernant tant le projet que le système et son environnement tout au long du cycle de vie.

1.3 COMPLEXITE DE LA CONCEPTION DES SYSTEMES MULTIDISCIPLINAIRES

L'ingénierie des systèmes est indispensable pour la mise en œuvre des systèmes complexes, en particulier les systèmes multidisciplinaires, ces systèmes apportent une difficulté accrue au processus de conception. De ce fait, nous avons identifié les caractéristiques de la conception des systèmes multidisciplinaires qui peuvent être sources de complexité :

- ***Multi-compétences :*** Les connaissances requises pour faire de la conception multidisciplinaire sont réparties entre différents domaines de la science et de l'ingénierie de sorte qu'une seule personne n'est pas capable de posséder toutes les expertises requises.
- ***Multi-langages :*** Les membres d'une équipe multidisciplinaire, collaborant les uns avec les autres afin d'atteindre un objectif commun, sont souvent de cultures scientifiques différentes et utilisent leur propre langage. En conséquence, il devient difficile pour les participants de communiquer, de se comprendre et de régler les conflits éventuels.
- ***Multi-vues :*** L'implication d'experts de divers métiers implique une différence de points de vue sur le système à concevoir, cette situation nécessite un important besoin d'échanges et de coopération.
- ***Multi-objectifs :*** Les différents métiers ont leurs propres règles et contraintes. Celles-ci peuvent entrer en conflit avec les objectifs globaux de la conception ou avec des caractéristiques des solutions proposées par les autres métiers.
- ***Dynamique difficilement prévisible :*** La départementalisation, la distribution géographique des compétences métiers cloisonnent les disciplines et rendent moins prévisible le nombre d'itérations nécessaires pour trouver une solution.
- ***Multi-relations et couplage interdisciplinaire :*** Les interactions entre les différents phénomènes physiques, les antagonismes entre qualité, sûreté de fonctionnement, coûts et

délais à maîtriser peuvent faire que certains membres de l'équipe de conception peuvent modifier des paramètres pour leur propre besoins à l'insu des besoins des autres.

- ***Intégration interdisciplinaire :*** L'intégration des composants et sous-système, l'élaboration de procédures de vérification des sous-systèmes sont eux-mêmes des problèmes multidisciplinaires qui nécessitent l'imbrication des compétences.

- ***Complexité combinatoire :*** Les composantes physiques de chaque produit étant souvent nombreuses, le processus d'évaluation de toutes les configurations possibles s'avère coûteux en temps, même en simulation. De plus, pour ne pas écarter une piste de solution innovante trop tôt, une tendance est à repousser le plus longtemps possible la décision d'abandonner une voie investiguée.

Un système multidisciplinaire est source de complexité, à cause des différences de culture scientifique au sein de l'équipe qui conçoit un tel système. De ce fait, les acteurs de la conception peinent à communiquer et à trouver des accords quant aux choix de conception, chacun ayant ses propres contraintes métiers et ses objectifs de conception qui peuvent être contradictoires avec ceux des autres.

1.4 LE PROCESSUS D'INGENIERIE DES SYSTEMES

Le processus d'ingénierie des systèmes est un processus décisionnel complet, itératif et récursif, qui est appliqué séquentiellement. Il englobe toutes les activités créatives, manuelles et techniques nécessaires à la définition du produit, et devants être réalisées pour convertir une définition de système en une spécification de conception de système, qui est suffisamment détaillée pour la fabrication et le déploiement du produit. Le processus d'IS permet l'analyse, l'évaluation et le contrôle du système, il permet de suivre les décisions et de vérifier les exigences tout en respectant les coûts et le calendrier imposé [9].

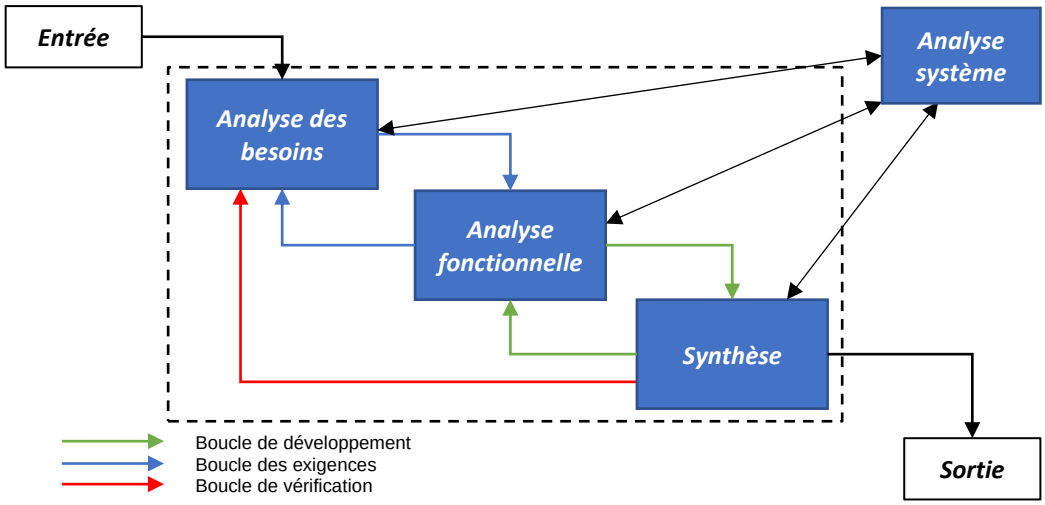


Figure 1. Le processus d'IS développé par le DoD [9].

Pour décrire l'approche d'ingénierie système, nous utilisons le processus d'IS développé par le département américain de la défense (DoD) (voir Figure 1) [9]. Cependant, il ne s'agit que d'une brève

description ou le processus d'IS est divisé en quatre étapes : L'étape d'analyse des exigences, l'étape d'analyse fonctionnelle, l'étape de synthèse de la conception et l'étape de contrôle et d'analyse système. En outre, l'entrée du processus d'ingénierie est définie par les exigences de la partie prenante (clients) et la sortie du processus renvoie les données qui sont nécessaires au développement du produit et aux étapes suivantes de son cycle de vie.

Dans le Tableau 1, une description succincte des étapes du processus d'ingénierie système selon le DOD [9].

Tableau 1. Description des étapes du processus d'ingénierie système

Etape	**Descriptions**

Entrée du processus	• Besoins, objectifs et exigences du client,
	  - Missions,
	  - Mesures d'efficacité,
	  - Environnements,
	  - Contraintes,
	• Base technologique,
	• Exigences de production découlant d'efforts de développement antérieurs,
	• Exigences de décision du programme,
	• Exigences appliquées selon les spécifications et les normes.

Analyse des besoins	• Analyser les missions et les environnements, • Identifier les exigences fonctionnelles, • Définir et affiner les exigences de performance et les contraintes de conception.
Analyse fonctionnelle	• Décomposer en fonctions de niveau inférieur, • Allouer les performances et autres exigences à tous les niveaux fonctionnels, • Définir et affiner les interfaces fonctionnelles (internes/externes), • Définir, affiner et intégrer l'architecture fonctionnelle.
Synthèse	• Transformer les architectures fonctionnelles en architectures physiques, • Définir des concepts alternatifs, des éléments de configuration et des éléments système, • Sélectionner les solutions de produits et de processus privilégiées, • Définir et affiner les interfaces physiques (internes/externes).
Contrôle et analyse du système	• Etudes de compromis, • Analyses d'efficacité, • Gestion des risques, • Gestion de la configuration, • Gestion des interfaces, • Gestion des données, • Mesures de performance.

Sortie de processus	• Dépendant du niveau de développement
	  - Base de données décisionnelle,
	  - Architecture des éléments du système et de la configuration,
	  - Spécifications et lignes de base.

1.5 CYCLE DE DEVELOPPEMENT

L'approche Ingénierie Système passe inévitablement par une description chronologique et séquentielle de la méthode de conception, appelée cycle de développement. Il existe un nombre important de cycles de développement et de représentations associées, ces cycles ont été proposés pour différents types de systèmes avec divers niveaux de complexité [10].

Les principaux cycles de développement sont : le cycle en V, le cycle en cascade et le cycle en spirale ; mais on pourrait également citer le cycle itératif, le cycle incrémental et un nombre important de cycles dérivés de ces 5 grandes familles. L'apparition de ces variantes s'explique par des besoins plus spécifiques dans certains domaines ainsi que leur adaptation à des outils particuliers de développement.

1.5.1 Cycle en V [11]

La Figure 2 illustre une des représentations du cycle de développement en V. Le cycle en V est aujourd'hui le cycle de développement le plus utilisé dans l'IS. Si les noms et le nombre d'étapes changent d'un ouvrage à l'autre, le principe général reste identique. Le cycle est composé d'une branche descendante correspondant à une démarche de conception (décomposition et définition) et d'une branche ascendante détaillant les phases de test et de vérification.

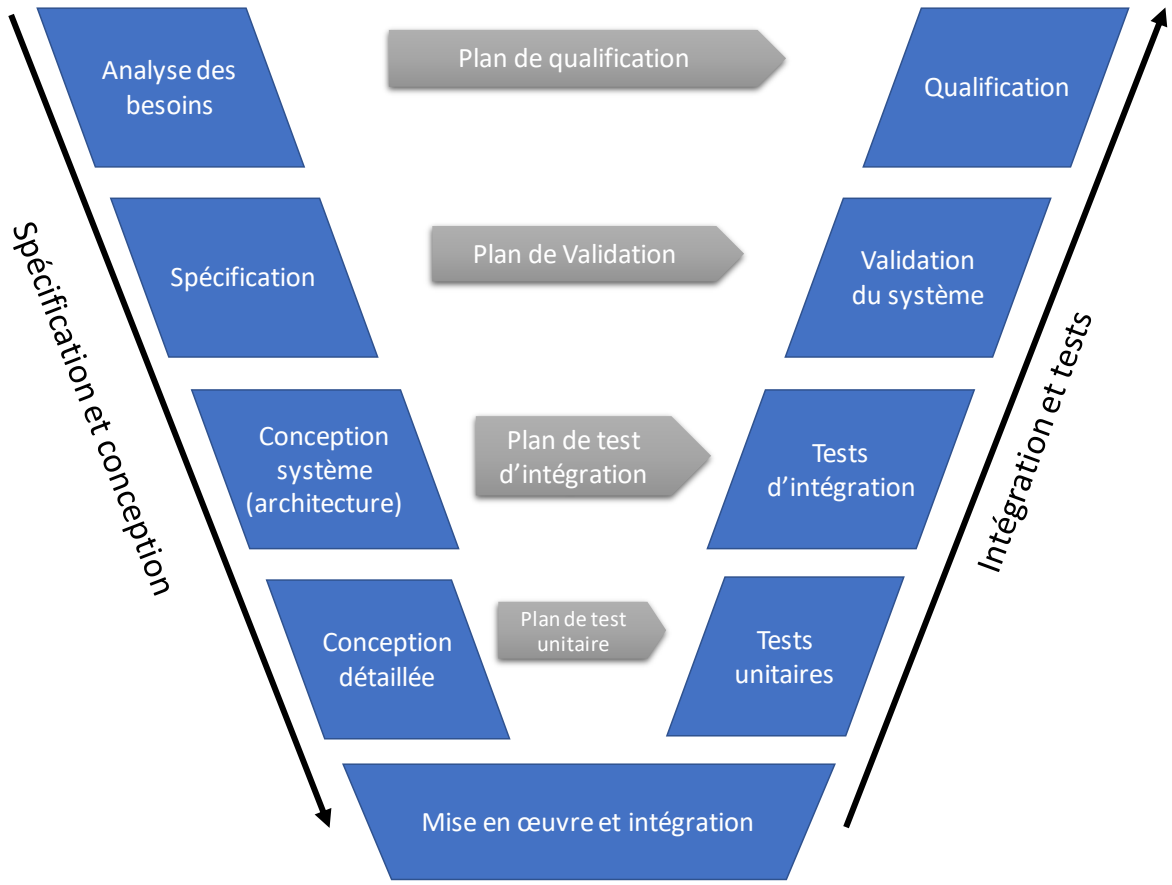


Figure 2. Exemple de représentation du cycle de développement en V

Voici de façon synthétique les différentes phases du cycle en V pour décrire un système :

Analyse des besoins : Expression bien formulée des besoins du client et de l'ensemble des parties prenantes ayant un lien avec le système à concevoir. Elle définit la mission du système.

Spécification : La spécification système définit de façon détaillée et rigoureuse les exigences que doivent respecter le système et les sous-systèmes. Elle exprime au travers de ces exigences ce que doit faire le système et non comment il doit le faire (aucun apport de solutions techniques). Les exigences peuvent être de deux types ; elles peuvent transmettre un besoin fonctionnel (exigences fonctionnelles) ou un besoin qualitatif (exigences non fonctionnelles). Cette phase constitue un point important pour la réussite d'un projet. La spécification doit éviter toute ambiguïté, tout oubli ou incohérence avec la phase amont. Durant cette phase, sont ajoutés les plans de validation qui consistent à définir les exigences qui devront être vérifiées, ainsi que la manière dont elles seront vérifiées.

Conception système : Le système va être décomposé de façon itérative en sous-systèmes. Les exigences issues de la spécification sont alors distribuées entre les différents sous-niveaux. On obtient un dossier de conception, composé de spécifications techniques et de l'architecture système. Ce dossier est accompagné du plan d'intégration et des tests d'intégration à effectuer. La validation de cette phase s'effectue au travers des prototypes, modèle et/ou simulation.

Développement des composants et réalisation : L'idée générale ici est de reprendre le cycle en V pour les sous-systèmes de façon itérative. On retrouve la spécification sous-système, la conception détaillée, la réalisation, les tests unitaires, l'intégration sous-système et la validation sous-système. Sans être normalisé, c'est en général le moment où la conception n'est plus assurée par les métiers de l'IS mais par un seul métier (réalisation d'une carte, d'un logiciel, d'une puce, d'une structure...). Il est essentiel d'avoir une définition parfaite des interfaces des composants développés par des équipes différentes afin d'assurer la réussite de l'intégration.

Intégration système : C'est la phase d'assemblage des composants. Chaque composant, après avoir fait l'objet d'une validation est inséré suivant le plan d'intégration défini en phase de conception. Le nouveau système constitué est alors soumis à une nouvelle phase de validation. Il en résulte le système intégré prêt à être validé à son tour.

Validation système : En suivant le plan de validation défini en phase de spécification, on vérifie exigence par exigence que le système intégré est conforme à ces spécifications. Le système est alors prêt pour la phase de qualification opérationnelle.

Qualification opérationnelle : C'est la phase de test de la conformité aux besoins opérationnels définis par le client et par les parties prenantes en phase d'analyse des besoins.

Le modèle de développement en V n'est pas parfait et souffre de certaines lacunes. La séquentialité et la linéarité du modèle en V ne font pas apparaitre l'ensemble des rétroactions des phases aval vers les phases amont ainsi que les cycles de développement intermédiaires. Par ailleurs, l'approche descendante (top down) du cycle en V ne facilite pas l'intégration de composants existants. Il est clair qu'un système complexe est analysé avec une approche descendante, mais concrètement

le système est constitué en majorité de composants existants sur le marché ou déjà développés. Le système est donc construit dans une approche ascendante (bottom-up) que ne montre pas le cycle en V. L'aspect séquentiel de la méthode oblige à bien concevoir chaque phase du cycle, mais en pratique c'est rarement le cas, c'est la conséquence de mauvaises interprétations des besoins client, des incohérences lors de la spécification ou suite aux mauvais choix de conception.

1.5.2 Cycle en Spirale

La Figure 3 donne un exemple de cycle de développement provenant des méthodes agiles, la méthode Spirale a été initialement utilisée pour le développement logiciel. Ce modèle reprend les différentes étapes du cycle en V mais permet de réitérer autant de fois que nécessaire le cycle complet, en apportant à chaque itération de nouvelles fonctionnalités, une meilleure réponse aux besoins du client et une meilleure robustesse. L'analyse de risques et l'utilisation de prototypes sont deux autres dominantes de cette méthode, où l'on estime qu'il est plus efficace d'expérimenter avec les utilisateurs (prototypage) puis de rectifier, que de réfléchir longuement pour tenter d'obtenir directement la bonne solution. Cette méthode, très employée dans le génie logiciel n'a encore jamais été utilisée pour les systèmes complexes [10].

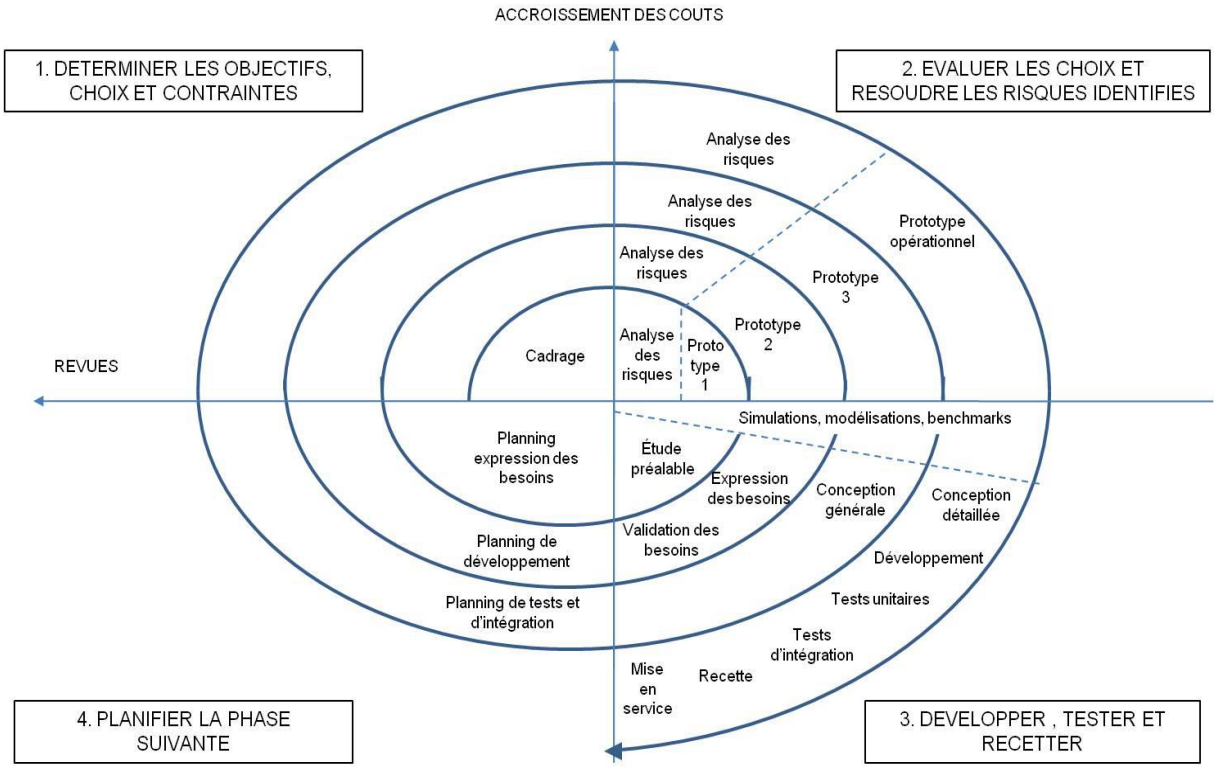


Figure 3. Représentation du cycle de développement en spirale

1.5.3 Cycle en cascade

Le modèle de cycle de vie en cascade a été mis au point depuis les années soixante. Les phases de conception ou de développement sont effectuées les unes après les autres, avec un retour sur les précédentes, pour vérifier la conformité avant de passer à la suivante (symbolisées dans la Figure 4 par des flèches vers le haut) [10].

Les avantages de ce modèle sont de deux natures, d'un point de vue technique, le processus est clair et systématique. Les informations sont très bien organisées, transmissibles et réutilisables. D'un point de vue gestion de projet, il fournit un cadre de référence pour la planification et le contrôle ainsi qu'une bonne visibilité des progrès et des résultats. Malgré ces avantages, ce modèle de cycle ne convient pas à la conception des systèmes complexes, car la conception de ces systèmes n'est pas séquentielle. Il est difficile de définir tous les besoins dès le début du projet. De plus, la validation ne peut se faire que tardivement vers la fin du cycle.

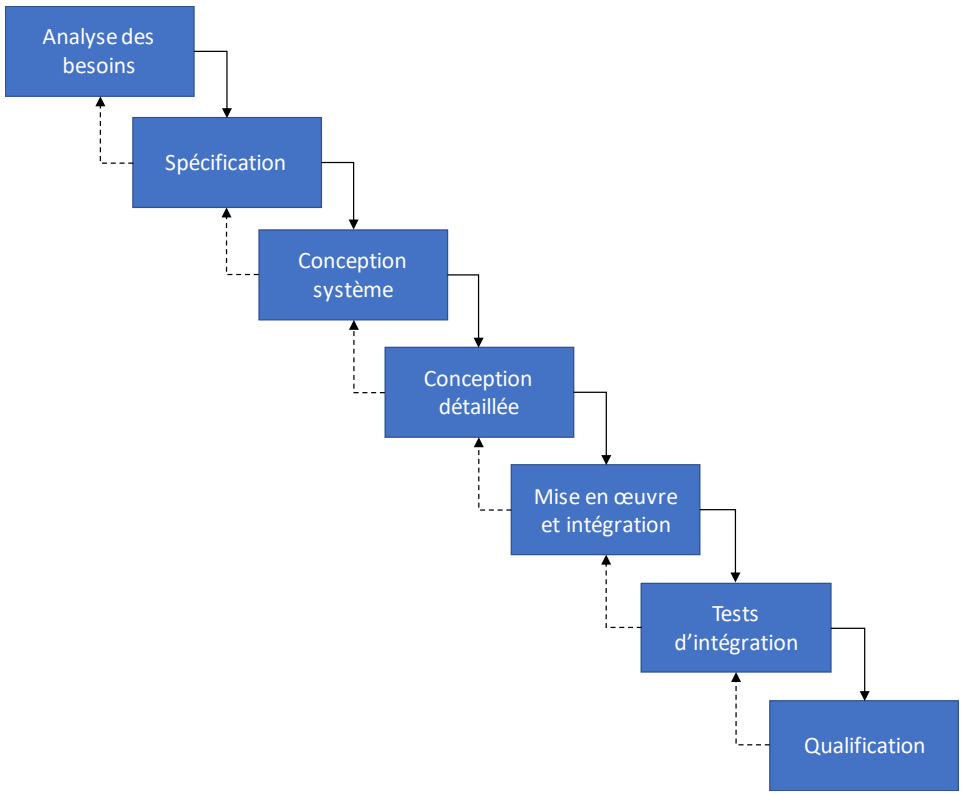


Figure 4. Représentation du cycle de développement en cascade

1.6 LES METHODES DE CONCEPTION

La conception d'un système est un processus méthodique complexe. Elle consiste à mener conjointement des activités afin de parvenir à une solution satisfaisant un besoin formulé et un ensemble de contraintes. Dans cette section, nous nous intéressons aux méthodes sur lesquelles peuvent s'appuyer les différents cycles présentés précédemment.

1.6.1 Démarche séquentielle

Cette démarche, utilisée dans le cycle en V ou de type cascade présente deux principaux types de processus de conception. Il s'agit de la conception dite « Bottom-up » et de la conception dite « Top-down » [12]. Dans les systèmes d'ingénierie actuels, la conception est surtout de type « Top-down », cette approche se base sur une démarche de conception allant du plus abstrait au plus détaillé. Le point de départ est une description globale du système qui représente les attentes de l'utilisateur, puis la conception est enrichie d'un ensemble de fonctions qui couvre les spécifications du système. Le système est ainsi successivement partitionné et affiné, jusqu'à l'obtention d'une

définition détaillée. Dans cette approche les erreurs et les faiblesses du système sont découvertes au plus tôt car la validation se fait à chaque niveau d'abstraction avant de passer au niveau suivant qui est plus détaillé, ce qui permet de gagner un temps important lors de la conception.

L'inconvénient de cette approche séquentielle est que chaque séquence verrouille certains aspects de la conception devenant des contraintes supplémentaires à la séquence suivante. De plus, non seulement le système final n'est pas optimisé, mais en plus, la durée du processus de conception s'accroit puisque la tâche suivante ne peut commencer que lorsque la précédente est terminée. Face à ces obstacles, d'autres approches ont été développées.

1.6.2 Démarche itérative

Cette démarche issue de la communauté logicielle est basée sur le cycle en spirale. En effet, les approches traditionnelles séquentielles laissent peu de place aux changements. Cette impossibilité de changement est d'ailleurs un des premiers motifs d'échec des projets informatiques. Avec l'approche itérative, la notion de « gestion de projet » est remise en question au profit de la « gestion de produit » de façon à raisonner sur le « produit » plutôt que sur le « projet » [12].

Cette approche empirique adopte un processus de développement itératif et incrémental datant de 1986. Dans le cadre d'un projet logiciel, le client liste les fonctionnalités que le produit doit réaliser. L'équipe de développement estime le coût du projet et réalise un produit partiel mais utilisable dans un délai très court (Une itération). Le client peut ainsi se rendre compte très tôt de l'alignement du produit réalisé sur le besoin et émettre des retours précieux pour les futures itérations.

Pour conclure, cette démarche propose une grande flexibilité dans la conception des systèmes, mais elle nécessite en contrepartie de prendre en compte au plus tôt les possibilités d'évolution du produit pour permettre de développer facilement les fonctionnalités au fur et à mesure des changements.

1.6.3 Démarche de conception basée sur les modèles

Une des dernières tendances dans le domaine de l'ingénierie des systèmes est d'utiliser des modèles pour capturer et contrôler les informations des systèmes. Cette nouvelle approche appelée Model-Based Systems Engineering (MBSE) ce qui correspond littéralement à « l'ingénierie des systèmes basée sur les modèles ». L'approche MBSE permet de générer des modèles au niveau système, offrant ainsi la possibilité de créer une base de connaissances dans toutes les disciplines et donc de favoriser la collaboration, ainsi que la communication entre les différentes disciplines d'ingénierie impliquées à un niveau limité de détails et donc à un niveau d'abstraction plus élevé. Dès lors, l'ingénierie système basée sur les modèles (MBSE) est l'application formalisée de la modélisation permettant de supporter toutes les activités de conception, de développement et des autres phases du cycle de vie [13]. Un état de l'art des principales méthodologies pour l'implémentation de la MBSE a été publié par Jeff A. Estefan [14], cette synthèse regroupe l'essentiel des éléments-clés pour la mise en œuvre d'une méthodologie de conception basée sur les modèles.

2 INGENIERIE SYSTEME BASEE SUR LES MODELES "MBSE"

Selon la définition de l'INCOSE [15] *: « L'ingénierie système basée sur les modèles (MBSE), est l'application formalisée de la modélisation permettant la gestion des exigences du système, la conception, l'analyse, les activités de vérification et de validation et ce dès les étapes amont et tout au long du cycle de vie ».*

> ***Remarque :***
> A ne pas confondre avec la Model-driven engineering MDE, qui est une méthode d'ingénierie spécifique au domaine du logiciel, même avec l'existence de traits communs qui pousse à la confondre avec la MBSE, la MDE est basée sur des métamodèles et une pratique qui repose sur la transformation de modèle [16].

L'objectif principal de la MBSE est d'améliorer la capacité de capture, d'analyse, de partage et de gestion des informations système [16]. La MBSE est souvent représentée comme un passage d'une approche d'ingénierie système centrée sur les documents à une approche centrée sur les modèles. Dans une approche traditionnelle centrée sur les documents, les informations associées aux différentes tâches de conception telles que les spécifications, les descriptions de système, les rapports d'analyse, les études de compromis et les processus de qualification, figurent dans des documents, tandis-que dans une approche centrée sur les modèles, toutes ces informations (ou une partie) sont capturées dans un modèle système ou un ensemble de modèles. De ce fait, et en raison des avantages générés par la MBSE, cette approche entraînera des améliorations significatives dans les processus de conception, de réalisation et de déploiement du système ; en matière de fonctionnement, de sécurité et des coûts induits.

L'application de la MBSE repose sur trois piliers qui doivent être pris en considération. Ils sont « un langage de modélisation », « une méthode de modélisation » et « un outil de modélisation » [17] [18], tel que :

- Le langage de modélisation doit être un moyen de définir les éléments d'un modèle ainsi que les liens qui relient ces éléments, il peut être graphique (SysML, UML, MARTE…) ou textuel (Modelica, VHDL…). Le plus important est que ce langage doit être standardisé pour éviter toute ambiguïté dans la compréhension des modèles.
- La méthode de modélisation est une composition de règles qui permet à l'équipe de modélisation de construire le modèle du système de manière cohérente qui vise un objectif final commun. Elle permet de décrire comment utiliser le langage de modélisation pour générer les modèles du système. Plusieurs méthodes existent dans la littérature telle que :
  - INCOSE Object-Oriented Systems Engineering Method,
  - Weilkins System Modeling method,
  - IBM telelogic Harmony-SE.

- L'outil de modélisation est un outil logiciel qui est nécessaire pour construire les modèles du système tout en respectant les règles du ou des langages de modélisation. Voici quelques exemples d'outils :
    - Agilian (fournisseur : Visual Paradigm),
    - Enterprise Architect (fournisseur : Sparx Systems),
    - Rhapsody (fournisseur : IBM Rational),
    - Modelio (Créateur : Modeliosoft),
    - Papyrus (Créateur : Atos Origin).

2.1 LES OBJECTIFS DE LA MODELISATION SYSTEME

Comprendre l'objectif et la portée de la modélisation d'un système fournit la base pour l'établissement d'attentes réalistes pour l'effort de modélisation. Les objectifs de la modélisation d'un système peuvent mettre l'accent sur différents aspects du processus d'ingénierie des systèmes ou prendre en charge d'autres utilisations du cycle de vie, notamment : [13] [19].

- Spécifier et concevoir un nouveau ou une nouvelle version d'un système.
    - Représenter les concepts d'un système.
    - Spécifier et valider les exigences d'un système.
    - Spécifier les exigences des composants du système.
    - Concevoir le système.
    - Maintenir la traçabilité des exigences.
- Evaluer le système.
    - Réaliser des analyses de compromis.
    - Analyser les performances, le coût ou d'autres paramètres du système.
    - Vérifier que la conception du système répond à ses exigences.
    - Evaluer l'impact de changements d'exigences ou de conception.
- Caractériser un système existant (reverse engineering).
- Former les utilisateurs et concepteurs du système.

2.2 AVANTAGES DE LA MBSE

Dans [13] et [20] les auteurs ont énuméré un ensemble d'avantages que peut apporter la méthodologie MBSE au processus de conception système, notamment :

- Communications améliorées :
    - Compréhension partagée du système par l'équipe de développement et les autres parties prenantes,
    - Capacité à intégrer des représentations du système à partir de plusieurs points de vue.
- Risque de développement réduit :
    - Validation continue des exigences et vérification de la conception,
    - Des estimations de coûts plus précises pour développer le système.

- Qualité améliorée :
  - Exigences plus complètes, non ambiguës et vérifiables,
  - Traçabilité plus rigoureuse entre les exigences, la conception, l'analyse et les tests,
  - Intégrité de conception améliorée.
- Productivité accrue :
  - Analyse d'impact plus rapide des exigences et des modifications de conception,
  - Exploration plus efficace de l'espace commercial,
  - Réutilisation des modèles existants pour accompagner l'évolution de la conception,
  - Erreurs et temps réduits lors de l'intégration et des tests,
  - Génération automatisée des documents.
- Tirer parti des modèles tout au long du cycle de vie :
  - Soutenir la formation des opérateurs sur l'utilisation du système,
  - Soutenir le diagnostic et la maintenance du système.
- Transfert des connaissances amélioré :
  - Capture des conceptions existantes et anciennes,
  - Accès efficace et modification de l'information.

2.3 PROBLEMATIQUES ET DEFIS LIES A L'UTILISATION DE LA MBSE

Un certain nombre d'obstacles sur la voie de l'adoption industrielle de la MBSE ont été identifiés. Nous présentons ici une synthèse les problématiques et défis que doit surpasser la MBSE pour une meilleure adaptation au milieu de l'ingénierie système [21] [18] [22].

A. Outils et méthodologies : Dans le milieu industriel, la MBSE n'est qu'un autre outil de création de diagrammes qui représente les données de manière plus organisée à un niveau supérieur, mais ne procure aucun avantage dans les phases ultérieures de développement du produit, tel que l'analyse, la vérification, la validation etc. Ainsi, pour changer la perception industrielle, il est important de développer la capacité de traduire les modèles systèmes définis dans la MBSE en modèles analytiques ainsi qu'en modèle de simulation.

B. Questions de réglementation : Le système de réglementation repose encore principalement sur des documents. Les industriels sont donc sceptiques quant à l'acceptation des produits développés selon l'approche MBSE par les organismes de réglementation. Pour relever ce défi, nous devons montrer que les exigences réglementaires peuvent être capturées avec les informations requises au même niveau (ou même mieux) dans des représentations basées sur un modèle.

C. Multi-domaine : L'un des avantages de l'utilisation de la MBSE et qu'elle introduit un partage des modèles entre différents domaines d'ingénierie, sauf que l'effort de sa généralisation n'est pas toujours accepté. En outre, dans certains domaines, tels que la mécanique, les ingénieurs ne sont pas en mesure de comprendre les diagrammes SysML, ce problème peut être résolu en accentuant les efforts dans les formations spécialisées.

D. Outils de développement : Dans chaque domaine d'ingénierie, différents outils de développement sont utilisés. L'application de la MBSE nécessite donc le développement de nombreuses interfaces pour ces outils, pour ainsi maintenir la cohérence des données tout au long du processus de développement.

E. Problèmes hérités : La plupart des entreprises possèdent d'importantes base de données sur d'anciens produits qui sont centrés sur les documents. Mais il n'existe pas de processus de traduction simple (ou peu coûteux) permettant de convertir ces informations dans un format centré sur les modèles. Cela représente un gros obstacle pour les entreprises qui souhaitent appliquer la MBSE aux produits existants.

2.4 LE LANGAGE DE MODELISATION SYSML

Un autre élément important de l'approche MBSE concerne le langage utilisé. Le langage SysML (Systems Modeling Language), basé sur le langage UML 2.0 initié par l'OMG (Object Management Group) et l'INCOSE en 2001 est souvent vu par la communauté de l'ingénierie système comme le plus adapté [13]. La description du langage SysML ci-dessous n'est pas exhaustive, une spécification complète est disponible sur le site de l'OMG [23].

Ce langage permet de couvrir les différentes phases de conception des systèmes complexes, le langage SysML est basé sur une modélisation graphique utilisant trois types de diagrammes :

- Diagrammes pour la modélisation structurelle ou architecturale ;
- Diagrammes pour la modélisation du comportement ;
- Diagrammes pour la spécification des exigences.

La Figure 5 montre la composition de ces différents types de diagrammes. Le langage SysML permet de gérer une grande partie des activités de conception. En termes de spécification d'exigences, SysML permet, à partir du cahier des charges, de transcrire les exigences sous forme de diagrammes et de blocs ce qui aide à définir facilement les concepts du système à concevoir et à tracer rapidement les modèles déployés avec les exigences correspondantes. En termes de modélisation logique, SysML permet de décrire les architectures logiques du système et de définir des diagrammes de spécification des liens (association, composition, agrégation, etc.) entre les concepts. En termes de modélisation physique, SysML permet d'une part de définir l'architecture physique du système et d'autre part de décrire les lois physiques à associer aux composants et sous-systèmes en définissant les flux échangés (signaux, énergies, etc.) et les attributs physiques (surface, masse, etc.).

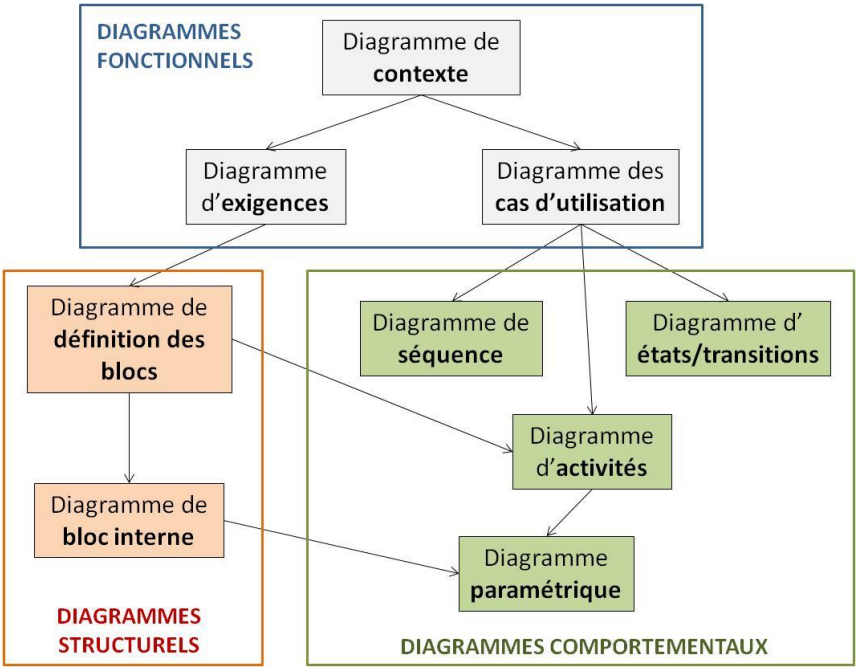


Figure 5. Les diagrammes du langage SysML

2.5 DEVELOPPEMENT BASE SUR UN MODELE POUR LES SYSTEMES DE CONTROLE

En ce qui concerne le développement des contrôleurs embarqués destinés aux systèmes de commande, la conception à base de modèles (MBSE) est une méthodologie couramment appliquée. Elle fait généralement référence au développement du système sur des outils spécialisés, ceci en réalisant des modèles de simulations de toutes les parties du système visé. C'est un moyen efficace de développement des systèmes numérique de contrôle, plutôt que de s'appuyer sur des prototypes physiques, la MBSE utilise des modèles comme représentation exécutables tout au long du cycle de développement [24].

3 LA MODELISATION

L'activité de modélisation est à la base des nouveaux processus de conception tel que la MBSE. Cette tâche consiste essentiellement à développer une description abstraite d'une réalité physique de telle façon qu'elle soit utile pour le processus de conception. Le modèle dépend du point de vue selon lequel on observe le système, mais aussi suivant l'utilisation que l'on souhaite faire de ce modèle au sein du processus de conception [25].

3.1 DEFINITION D'UN MODELE

Il existe de nombreuses définitions du mot modèle. Les définitions suivantes font référence à un modèle en tant que représentation d'aspects sélectionnés d'un domaine de modélisation :

- Une représentation physique, mathématique ou logique d'un système, d'une entité, d'un phénomène ou d'un processus [26] ;

- Une représentation d'un ou plusieurs concepts pouvant être réalisés dans le monde physique [13] ;
- Une représentation simplifiée d'un système à un moment donné dans le temps ou dans l'espace, destinée à promouvoir la compréhension du système réel [27] ;
- Une abstraction d'un système visant à comprendre, communiquer, expliquer ou concevoir les aspects présentant un intérêt pour ce système [28] ;
- Une approximation, une représentation ou une idéalisation de certains des aspects de la structure, du comportement, du fonctionnement ou d'autres caractéristiques d'un processus du monde réel, d'un concept ou d'un système [29] ;

Dans notre contexte, un modèle qui représente un système et son environnement revêt une importance particulière pour sa mise-en-œuvre, il doit spécifier, concevoir, analyser et vérifier des systèmes, ainsi que partager des informations avec d'autres parties prenantes. Une variété de modèles est utilisée pour représenter différents types de systèmes à différentes fins de modélisation.

3.2 PORTEE DU MODELE

Le modèle doit être conçu pour répondre à son objectif. En particulier, les types de modèles et les langages de modélisation associés doivent prendre en charge les besoins spécifiques à satisfaire. Supposons, par exemple, que des modèles soient construits pour soutenir le développement d'un avion. Un modèle d'architecture de système peut décrire l'interconnexion entre les pièces de l'avion, un modèle d'analyse de trajectoire peut analyser la trajectoire de l'avion et un modèle d'analyse d'arborescence de défaillances peut évaluer les causes potentielles de défaillance de l'avion. Pour chaque type de modèle, il convient de déterminer l'étendue, la profondeur et la fidélité appropriées pour répondre à l'objectif visé [6].

- ***L'étendue*** du modèle reflète la couverture des exigences système en fonction du degré auquel le modèle doit répondre aux exigences fonctionnelles, d'interface, de performances et physiques, ainsi qu'à d'autres exigences non fonctionnelles.
- ***La profondeur*** du modèle indique la couverture de la décomposition du système depuis le contexte du système jusqu'aux éléments du système.
- ***La fidélité*** du modèle indique le niveau de détail que le modèle doit représenter pour une partie donnée du modèle. La fidélité peut également faire référence à la précision d'un modèle de calcul.

3.3 TYPES DE MODELES

La plupart des systèmes commencent par un modèle mental qui est élaboré et traduit en plusieurs étapes pour former un modèle final ou un produit de simulation. Souvent, des modèles distincts sont préparés pour des points de vue distincts, tels que la fonctionnalité, les performances, la fiabilité, la disponibilité opérationnelle et les coûts.

Il est utile de classer les modèles pour nous aider à choisir le bon type de modèle pour le but et la portée prévus. Les modèles peuvent être classés de différentes manières, l'AFIS (Association

Française d'Ingénierie Système) distingue dans sa typologie trois types principaux de modèles [30]. Ces modèles sont distingués selon le rôle qu'ils jouent dans la conduite des activités d'IS. La Figure 6 montre une classification possible des modèles en 3 grandes catégories :

- ***Les modèles cognitifs*** sont utilisés pour l'analyse et l'exploration du problème ou du besoin à l'origine du système et pour la validation des concepts opérationnels de la solution apportée.
- ***Les modèles normatifs*** participent à la définition de la solution à différents niveaux d'abstraction et de granularité. A chaque niveau, des ***modèles prescriptifs*** représentent ce à quoi elle doit satisfaire et supportent ainsi la définition des exigences, des ***modèles constructifs*** représentent les architectures et leur fonctionnement.
- ***Des modèles prédictifs*** sont utilisés pour prévoir et valider le comportement des systèmes. Les techniques de spécification et de modélisation formelles permettant d'établir des preuves théoriques de conformité de comportement. Plus généralement, les exigences de performances et de sûreté de fonctionnement sont estimées grâce à ces modèles et à des techniques de simulation associées.

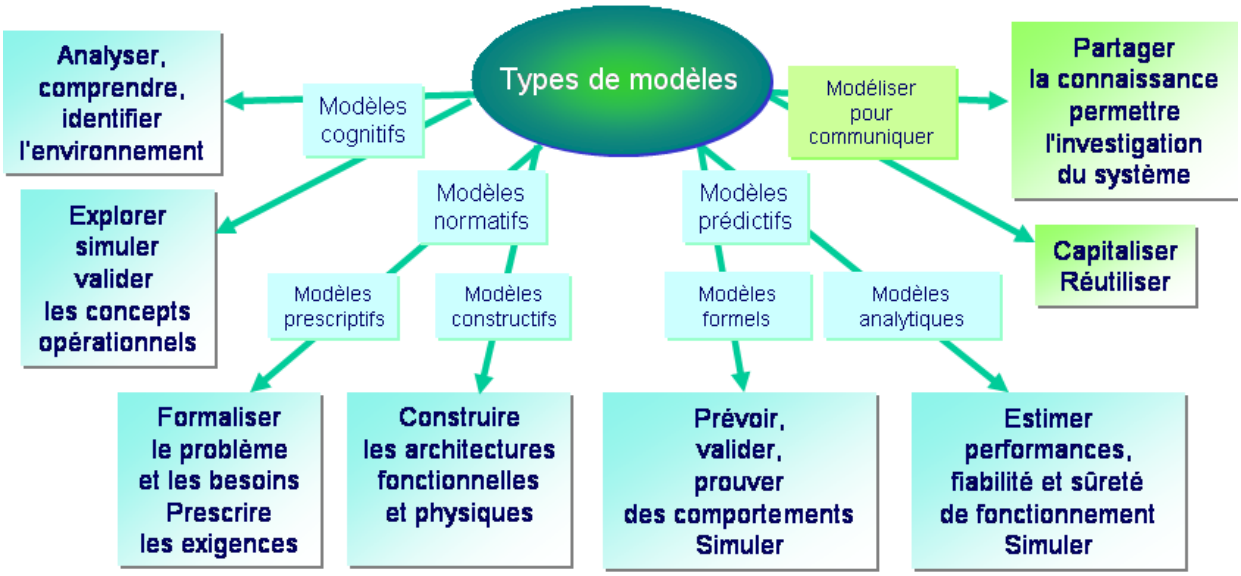


Figure 6. Typologie des modèles selon l'AFIS

4 VALIDATION & VERIFICATION

Un enjeu majeur de la conception des systèmes d'ingénierie est le processus de validation et de vérification qui est effectué à chaque étape du développement et de la construction d'un système jusqu'au produit final réalisé. L'activité de validation et de vérification (V&V) occupe une place essentielle en ingénierie système. Présente tout au long du cycle de développement, c'est une activité coûteuse qui demande une prise en compte particulière. Dans ce paragraphe, nous montrerons que la V&V va au-delà de la validation et de la vérification du système final. Pour améliorer la détection des fautes, les activités de V&V concernent aujourd'hui toute la chaîne de développement (produit,

activités, tâches, etc.), l'activité de V&V fait elle-même l'objet d'un processus de validation et de vérification.

Le standard ISO/IEC/IEEE 15288 [31] donne les définitions suivantes :

La vérification

- Le processus de vérification a pour but de fournir des preuves objectives qu'un système ou un élément de système satisfait à ses exigences et caractéristiques spécifiées.
- Le processus de vérification identifie les anomalies (erreurs, défauts ou fautes) dans tout élément d'information (par exemple, configuration système ou description de l'architecture), éléments du système mis en œuvre ou processus du cycle de vie à l'aide de méthodes, techniques, normes ou règles appropriées. Ce processus fournit les informations nécessaires pour déterminer la résolution des anomalies identifiées
- Le processus de vérification détermine que le "produit est construit correctement".

La validation

- Le processus de validation a pour but de fournir des preuves objectives que le système, lorsqu'il est utilisé, remplit ses objectifs commerciaux ou de mission et les exigences des parties prenantes, réalisant ainsi l'utilisation prévue dans l'environnement opérationnel prévu.
- L'objectif de la validation d'un système ou d'un élément de système est de gagner la confiance en sa capacité à réaliser la mission ou l'utilisation envisagée dans des conditions opérationnelles spécifiques. La validation est ratifiée par les parties prenantes. Ce processus fournit les informations nécessaires pour que les anomalies identifiées puissent être résolues par le processus technique approprié où l'anomalie a été produite.
- Le processus de validation détermine que le "bon produit est construit".

Les définitions des termes validation et vérification sont régulièrement sujettes à discussion. Ces termes ont une définition différente suivant le groupe concerné ou le domaine d'application. Il est fréquent de voir ces deux termes regroupés sous l'appellation V&V afin d'éviter toute ambiguïté. Les activités pouvant être très proches, elles sont parfois confondues. Nous allons voir ici qu'elles traitent de manière relativement semblable deux problématiques très différentes tout au long du développement du système.

La Figure 7 reprend en version simplifiée du cycle de développement d'un système, et illustre les principales activités de validation et de vérification. Les questions sous-jacentes à chacune des activités apparaissent sur le schéma de manière non-formelle. La vérification et la validation permettent ainsi de confirmer l'acceptabilité des éléments d'ingénierie des systèmes générés au cours des processus de transformation [32].

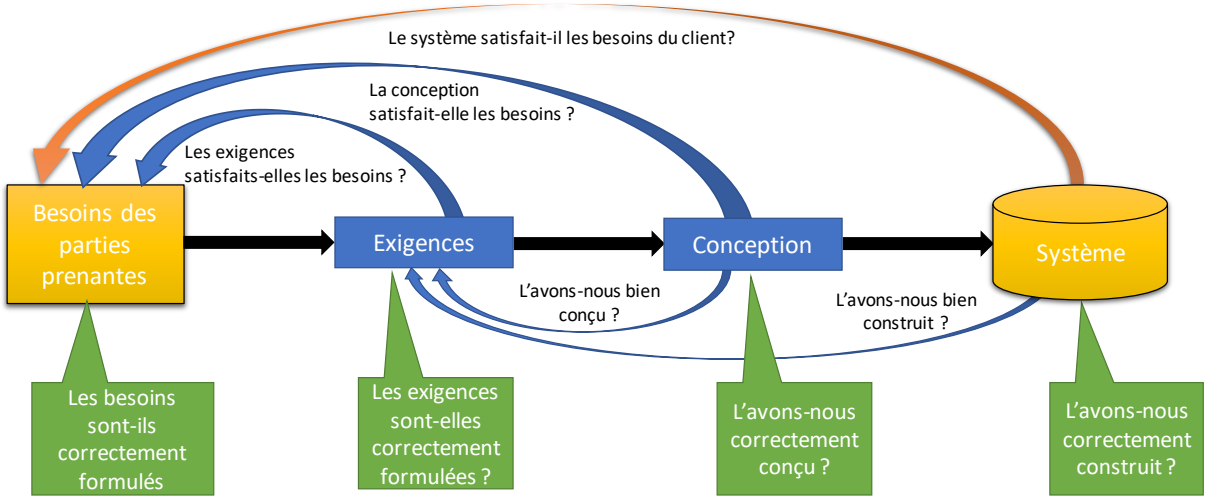


Figure 7. Principales activités de V&V durant le cycle de développement

- Les activités de vérification cherchent à s'assurer que la réalisation de l'activité a été bien faite, que le produit généré correspond à ses exigences de réalisation sans introduire de défauts. La vérification peut avoir lieu tout au long de l'activité pour confirmer que la réalisation se dirige convenablement vers le produit. Les méthodes de vérification sont variées : preuve, test, inspection et simulation.
- Les activités de validation cherchent à confirmer que la réalisation de l'activité répond à son objectif en vérifiant la conformité du produit de l'activité avec le besoin du "client". La validation va donner la confirmation que l'on se dirige vers le bon produit. Les méthodes de validation sont similaires aux méthodes de vérification : modélisation, preuve, test, inspection et simulation.

Il est à noter que les opérations de V&V sont planifiées et suivies au même titre que les autres activités. C'est le rôle du management technique de définir une stratégie de V&V, d'optimiser et de justifier le rapport couverture/coût, de planifier l'activité et en assurer la réalisation. Le contrôle de qualité s'assurera que ces opérations sont planifiées, exécutées conformément au plan d'exécution et techniquement correctes.

Les activités de validation et de vérification sont nombreuses et variées, il serait difficile et peu utile pour notre étude d'en faire une liste exhaustive. Le tableau 2 fournit néanmoins quelques exemples de ces activités tout au long du cycle de développement, et illustre quelques-unes des tâches que doit planifier le management technique. La ligne V&V propose deux exemples d'activités assurées par le contrôle qualité.

Le management technique possède plusieurs méthodes à sa disposition et choisira, selon la criticité, une ou un ensemble des méthodes de V&V tel que : le Test, la Preuve, l'Inspection, la Modélisation et la Simulation. Nous présenterons par la suite les méthodes de simulation qui sont des activités majeures pour réaliser les activités liées à la V&V.

Tableau 2. Exemples d'activités de validation et de vérification selon la phase de développement

Etape	Validation	Vérification
Exigences	S'assurer que les exigences et les besoins répondent correctement au besoins finaux des utilisateurs.	S'assurer qu'une exigence porte sur le quoi et pas sur le comment
Conception	S'assurer de l'absence d'incompatibilité de l'architecture logique avec les exigences non directement associables à l'aspect fonctionnel du système.	S'assurer que les interactions inter-fonctionnelles sont dûment identifiées et caractérisées.
V&V	S'assurer que le document de preuve satisfait au besoin exprimé par le document parent auquel il répond, en termes de conformité aux objectifs, de complétude, de faisabilité et d'optimisation.	S'assurer que les dossiers de preuves rassemblent les configurations appliquées, les procédures appliquées, les durées, etc.
Système	Fournir le dossier de preuves de satisfaction du besoin exprimé par le client.	S'assurer que le système est conforme aux spécifications.

5 APPORT DE LA SIMULATION

Face aux problématiques et aux besoins de l'ingénierie système, l'utilisation de la simulation et de la modélisation semble être un moyen d'y répondre. En effet, la simulation sert à conduire des expérimentations pour répondre à des questions lorsque le système réel n'est pas disponible ou pas accessible, ou bien, elle sert à acquérir de l'expérience pour se former ou s'entraîner, comme par exemple sur un simulateur de vol [33]. Selon [34] la simulation effectue des expériences dirigées vers un objectif en utilisant un modèle dynamique du système.

Les parties prenantes d'un projet d'ingénierie système utilise des modèles et des simulations pendant le cycle de vie à la fois pour vérifier leurs propres pensées et pour communiquer leurs concepts aux autres, ainsi les bénéfices apportés sont considérables :

- Les modèles et les simulations confirment la nécessité du système et son comportement attendu avant de poursuivre le développement d'un système réel,
- Les modèles et les simulations présentent une conception claire et cohérente à ceux qui vont développer, tester, déployer et faire évoluer le système, maximisant ainsi la productivité et minimisant les erreurs.
- La capacité à détecter les limitations et les incompatibilités au moyen des modèles du système et de la simulation au début d'un projet, ceci permet d'éviter des coûts plus élevés

et des dépassements de programme plus tard dans un projet, en particulier pendant le fonctionnement du système.

- La valeur de la modélisation et de la simulation augmente avec la volume physique ou la complexité du système en cours de développement.

Aujourd'hui, la simulation peut être utilisée tout au long des phases de conception, de développement et même durant l'exploitation d'un système : [10]

- **Conceptualisation** : Aide à la comparaison des solutions envisageables, aide à la faisabilité du projet.
- **Conception** : Augmente la confiance dans la solution proposée avant de passer à la phase de développement. Et permet d'évaluer si les exigences peuvent être respectées ou si la solution peut être améliorée.
- **Développement** : Aide au développement du système, aide à la validation du comportement du système à chaque étape du cycle de développement.
- **Transfert vers l'exploitation** : Entraînement à l'utilisation du système pour les futurs opérateurs du système.
- **Exploitation** : Aide à la réparation, à l'amélioration ou au diagnostic du système après sa mise en exploitation.
- **Retrait** : Aide à la validation du scénario de retrait avec prise en compte des mises à jour ou des pannes subies durant l'exploitation.

5.1 LA SIMULATION POUR LA VERIFICATION/VALIDATION

La distinction entre méthode d'exécution (Simulation) et méthode de modélisation ne semble pas toujours évidente, au regard de la littérature. Cela se ressent au travers de la notion de modèle (de simulation) qui parfois fait référence à du code et parfois à une représentation graphique de plus haut niveau (et voire même aux deux), mais aussi au niveau des taxonomies traditionnelles des modèles de simulation qui sont souvent liées au paradigme d'implémentation de la simulation plutôt qu'à un paradigme de modélisation. [33]

La simulation peut être utilisée afin d'effectuer les activités de vérification et de validation pendant tout le cycle de conception et de développement. Les simulations peuvent refléter les fonctions d'un système ou bien la structure détaillée du système. Ces simulations sont composées des représentations des éléments du système, connectés de la même manière que dans le système réel [6]. Généralement, la simulation est exécutée au travers de scénarios dans le domaine temporel pour simuler le comportement du système réel [33].

L'avantage de l'utilisation de la simulation est qu'elle est plus facile à mettre en œuvre que des techniques de preuve d'une part, et d'autre part, elle permet l'analyse des systèmes complexes plus facilement que les méthodes mathématiques/analytiques (en termes de temps par exemple). Cependant, elle ne peut pas être utilisée pour obtenir une preuve absolue, car le système est testé sur un certain nombre de scénarios, tous les cas ne sont donc pas abordés. De plus, le modèle de

simulation est conçu selon un objectif et selon la manière dont est perçu le système. Ceci peut poser un problème au niveau des hypothèses, qui peuvent être erronées ou manquantes, par exemple.

Quel que soit le type de techniques de simulation utilisées pour représenter la dynamique d'un système, de façon assez générale, on retrouve quatre types d'approches pour les plateformes de tests en fonction de la place de la simulation dans le cycle de conception [10]:

- Au début de la conception, il y a l'approche de simulation "Model in the loop" (MIL). On utilise le modèle décrivant le comportement du système à valider en le couplant avec un modèle de l'environnement, afin de constater si le modèle satisfait les exigences principales (du système).
- Ensuite, on trouve l'approche de simulation "Software in the loop" (SIL). Cette fois, on utilise le programme cible qui implémente le modèle (le programme final). On cherche ici à garantir la cohérence sémantique.
- Puis, apparaît l'approche de simulation "Processor in the loop" (PIL), parfois appelée "Controller in the loop". On s'intéresse ici à l'équivalence comportementale du programme après son intégration dans le processeur cible, cependant l'environnement reste simulé.
- Enfin, la dernière approche de simulation correspond à celle dite "Hardware in the loop" (HIL). Cette fois-ci on utilise le contrôleur physique final (par exemple l'automate programmable) mais l'environnement est toujours simulé avec en plus des contraintes de temps réel.

5.2 PROCESSUS DE CONCEPTION D'UNE SIMULATION

De façon générale, le processus de conception d'une simulation est constitué de trois phases principales : une première phase de modélisation, puis une seconde d'implémentation qui permet d'obtenir le modèle exécutable, et enfin une phase de tests (appelée généralement phase d'expérimentations) qui permet de tester le modèle de simulation [35].

Le système du monde réel (Système d'intérêt sur la Figure 8) qu'il soit existant ou hypothétique, correspond au système que l'on souhaite simuler pour conduire des expérimentations. Il s'agit ainsi du système que l'on souhaite modéliser pour pouvoir répondre à certaines questions. A partir de ce système du monde réel, la première étape de modélisation permet d'obtenir le modèle conceptuel, tandis que la deuxième étape, d'implémentation, permet d'obtenir le modèle de simulation (modèle exécutable) [33].

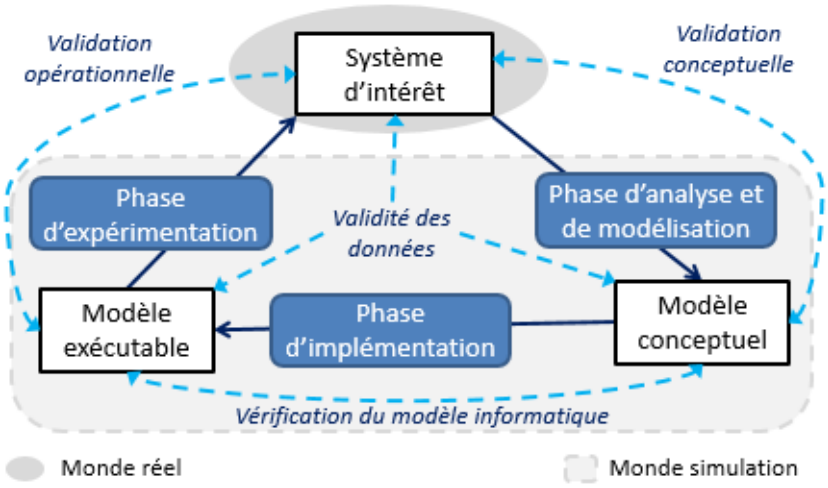


Figure 8. Processus de modélisation et de simulation simplifié [35]

Le passage du modèle conceptuel vers un modèle exécutable se fait via des étapes successives de spécification et d'implémentation. Il est à noter que le terme modèle (de simulation) est traditionnellement utilisé pour faire référence à un modèle informatique (computational model) sous la forme d'un programme (code) et non en référence à un modèle exprimé dans un langage de modélisation graphique de haut niveau. Le modèle de simulation y est ainsi vu comme un programme (du code) alors que le modèle conceptuel est plutôt perçu comme un modèle mental ou schématique. Pour s'assurer que le modèle de simulation est correct par rapport au modèle conceptuel, une vérification du modèle informatique est effectuée. Des mécanismes de vérification permettent de s'assurer que le modèle de simulation est une représentation fidèle du modèle conceptuel.

La phase d'expérimentation dans le processus de conception s'intéresse au système de simulation dans son ensemble (le modèle exécutable et ce qui permet de l'exécuter). Les expérimentations permettent de déterminer si les résultats de la simulation sont adaptés et assez précis pour l'objectif d'utilisation. D'autre part, il faut pouvoir garantir la validité des données entre le système d'intérêt et les modèles, que ce soit pour construire le modèle, l'évaluer, le tester ou lors de son utilisation (par rapport à l'objectif donné).

5.3 SIMULATION NUMERIQUE (INFORMATIQUE)

Pour les systèmes complexes de type multidisciplinaire, il est souvent difficile, voire impossible, de les étudier de manière analytique. C'est pourquoi la simulation numérique est utilisée à un stade précoce pour la conception de ces systèmes, notamment pour prendre en charge la dynamique du système plus efficacement. La simulation numérique a pour objectif principal d'approcher le plus fidèlement possible le comportement d'un phénomène physique complexe et d'obtenir les résultats les plus précis possibles. Cela signifie que le premier objectif est la précision des résultats de la simulation et que le temps de calcul n'est qu'en seconde priorité.

Traditionnellement, on recense quatre catégories de simulation, en fonction de la façon dont évoluent les variables d'états du système : statique, continue, à événements discrets et multi-agent [36].

- **La simulation statique** correspond à des variables d'états n'évoluant pas en fonction du temps, et est plus connue sous le nom de simulation de Monte-Carlo. La modélisation des phénomènes aléatoires se fait en utilisant des variables aléatoires et des lois de probabilités.
- **La simulation continue** qui est aussi appelé **simulation à temps continue** correspond à des variables d'états qui évoluent de façon continue en fonction du temps. La modélisation se fait sous forme de systèmes d'équations, où le temps et les relations d'états ne sont pas précisément représentés.
- **La simulation discrète** qui est aussi appelée **simulation à événements discrets**, correspond à une évolution des variables d'états lors d'événements discrets. Elle peut être dirigée par les événements (**event-driven**) ou par le temps (**time-driven**). La modélisation du système physique se fait avec des modèles mathématiques et/ou logiques qui représentent les changements d'états en des instants précis du temps simulé, ce qui implique une description précise de la nature du changement d'état et de son occurrence.
- **Simulation multi-agent** dans laquelle la simulation est segmentée en plusieurs entités qui interagissent entre elles. Elle est essentiellement utilisée dans les simulations sociales et économiques, dans lesquelles chaque agent décrit un individu ou un ensemble d'individus. Par nature, son fonctionnement est asynchrone [37].

5.3.1 Cosimulation

En ce qui concerne la simulation des systèmes multidisciplinaires, on définit une autre catégorie de simulation qui est la cosimulation. Les systèmes multidisciplinaires sont composés de multiples entités que nous modélisons séparément, et l'interaction de leurs modèles au sein du modèle système doit permettre de reproduire le comportement du système global. Il est entendu que chaque entité doit être modélisée en utilisant le formalisme le plus adéquat, leur intégration en un seul modèle nécessite donc de gérer le couplage entre les différents formalismes [38].

Une autre difficulté qui s'ajoute aux multiples formalismes, c'est le fait que tout système multidisciplinaire implique de multiples composants qui sont généralement décrits par des modèles à temps continue et des modèles à évènement discret. Ces composants ne peuvent souvent pas être modélisés et simulés dans un seul outil, où un composant individuel est mieux modélisé dans un outil spécifique à un domaine spécifique. Dans ces scénarios de simulation, le couplage de différents simulateurs est devenu essentiel pour garantir la fiabilité des résultats [39].

Dans une approche de conception pluridisciplinaire des systèmes complexes, la simulation n'est pas une tâche facile. Pour surmonter cette difficulté, deux approches sont proposées : d'un côté l'approche homogène qui consiste à utiliser un seul langage pour la spécification complète du fonctionnement du système, et de l'autre côté l'approche hétérogène qui se base sur l'utilisation de plusieurs langages de modélisation à la fois [40].

5.3.1.1 L'approche hétérogène

L'approche hétérogène permet de modéliser le système complet en utilisant des langages spécifiques. La technique de cosimulation permet l'utilisation de plusieurs simulateurs pour la validation globale de ce système. Pour cela, il faut disposer d'un modèle de communication qui décrit

la synchronisation et les interconnexions entre les différents modules. La difficulté réside dans la construction de ce modèle. Une autre technique, la technique mono-simulateur, consiste à traduire les langages utilisés pour la description du système entier vers une sorte de langage unique ou un format accepté par le simulateur [41] [42].

A. Technique mono-simulateur

Avec cette technique le système est toujours composé d'un ensemble de modules spécifiés dans différents langages mais la simulation nécessite le passage par un langage unifié ou un format connu par le simulateur (voir Figure 9).

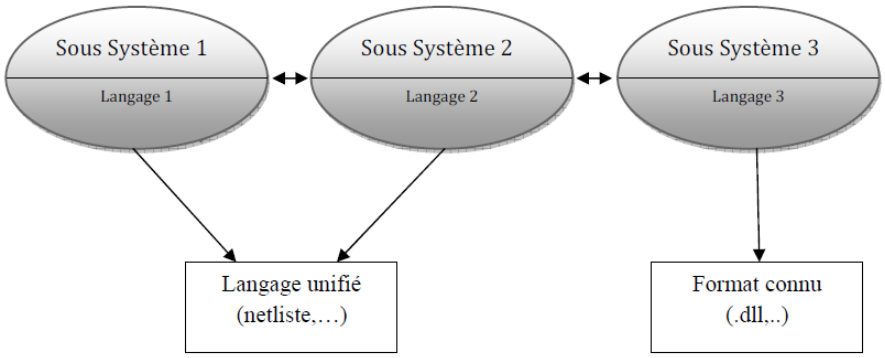


Figure 9. Le principe du technique mono-simulateur

La validation par simulation consiste à exécuter le modèle du système afin de reproduire le fonctionnement du système entier. Les modèles de chaque élément du système sont décrits en utilisant différent langages, mais lors de la simulation, ces différents modèles sont traduits dans une seule représentation pour qu'elle soit exécutée par le même simulateur.

B. Technique de cosimulation

La cosimulation consiste à exécuter des simulateurs communicants. Chacun des simulateurs exécute un élément ou plusieurs éléments du système qui sont décrit dans un langage approprié. Pour assurer l'échange correct des données et la communication entre ces sous-systèmes, le besoin d'un modèle de synchronisation s'impose. Ce modèle prend en compte les spécificités du modèle de simulation adopté par chaque simulateur.

Des interfaces de simulation assurent l'interconnexion entre les différents modèles. Ces interfaces communiquent à travers un bus de cosimulation qui peut être une mémoire partagée avec une structure bien définie permettant des interconnexions complexes ou autre technique de communication inter-processus. Les interfaces de simulation sont composées par des couches de communication et de synchronisation et selon les simulateurs utilisés ils peuvent implémenter des comportements assez complexes [43].

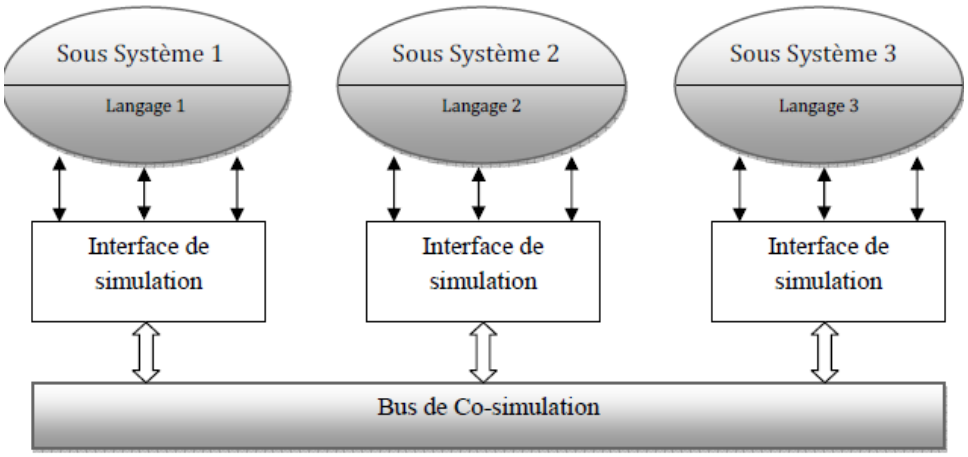


Figure 10. Le principe de la cosimulation

La cosimulation signifie l'utilisation combinée de différents simulateurs hétérogènes. Le couplage ou l'intégration de ces simulateurs constituera la base de la simulation de l'ensemble du système. La résolution numérique de chaque modèle peut être effectuée dans l'outil de simulation correspondant ou directement dans l'outil de simulation final, où tous les modèles sont intégrés. Lorsqu'un composant de loi de commande est inclus avec les autres modèles, nous parlons de la simulation Model-In-the-Loop (MIL) [44].

5.3.2 Cosimulation matériel/logiciel

La cosimulation matériel/logiciel intègre des techniques de conception matérielle et logicielle qui utilisent généralement divers langages, formalismes et outils dans une méthodologie de conception unique. L'utilisation d'un cadre unique pour cette tâche accélère le processus de conception, permettant de faire des compromis matériel/logiciel de manière dynamique au fur et à mesure de l'avancement de la conception, ce qui permet de soutenir considérablement le processus de vérification [45].

Ce concept de couplage matériel-simulation existe depuis plusieurs années. Dans [46] les auteurs présentent l'un des premiers systèmes de simulation Hardware-In-the-Loop (HIL) qui ne diffère pas des systèmes actuels. La simulation fonctionne sur du matériel PC standard, tandis que le couplage est réalisé avec des cartes d'entrée/sortie spécialisés.

5.4 Simulation Hardware-In-the-Loop (HIL)

La simulation HIL (Hardware-In-the-Loop) est une technique dans laquelle un équipement matériel est incorporé à la simulation d'un grand système. Cette approche offre de nombreux avantages que n'offrent pas les méthodes d'analyse et de test usuelles.

La simulation HIL est un concept largement utilisé, elle est principalement utilisée pour le prototypage rapide, les tests et l'optimisation des systèmes. Dans notre domaine d'application, celui des Systèmes de conversion de la puissance électrique, la simulation Hardware-In-the-Loop (HIL) permet entre autres aux concepteurs d'évaluer un algorithme de commande en couplant le contrôleur

numérique (partie "Hardware") à des simulateurs logiciels qui reproduisent le comportement dynamique du système de puissance.

Les raisons majeures qui motivent le recours à un simulateur HIL lors d'essais sur les systèmes de conversion de la puissance électrique sont nombreuses ; on peut notamment citer les arguments suivantes qui motivent l'utilisation d'un simulateur HIL : [47].

- Le système de puissance n'est pas disponible,
- Les coûts engendrés par la construction du système de puissance sont très élevés,
- Les essais comportent des risques importants pour la sûreté globale de l'opération (danger, puissance élevée mise en jeu, etc.).

En ce qui concerne la rapidité du calcul requis, les méthodes de simulation peuvent être subdivisées en trois classes [48]:

- Des méthodes de simulation sans limitations de temps,
- Des méthodes de simulation en temps réel,
- Des méthodes de simulation plus rapides qu'en temps réel.

Certains exemples d'application sont donnés à la Figure 11. Ici, la simulation en temps-réel signifie que la simulation d'un composant est effectuée de sorte que les signaux d'entrée et de sortie affichent les mêmes valeurs dépendantes du temps que le composant réel à fonctionnement dynamique. Cela devient un problème de calcul pour les processus dont la dynamique est plus rapide par rapport aux algorithmes et à la vitesse de calcul requise.

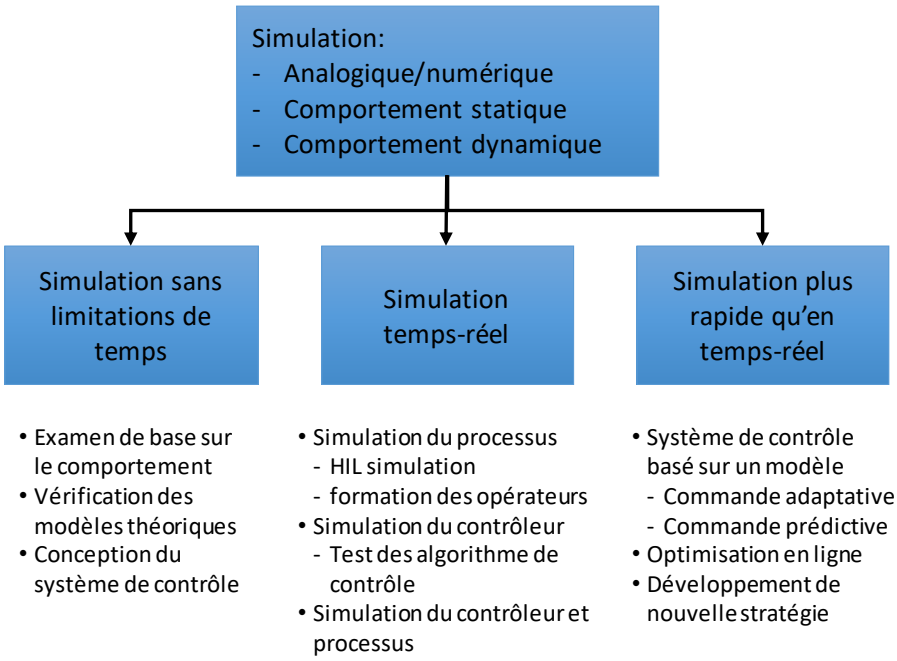


Figure 11. Classification des méthodes de simulation selon leur rapidité de calcul

5.4.1 Architectures

Généralement, un système de contrôle est composé d'un contrôleur, d'un processus a contrôlé, d'actionneurs et de capteurs. Dans ce cas, la simulation HIL peut être mise en œuvre de différentes manières, la stratégie de mise en œuvre est généralement une question de projet et de ressources [49]. Comme le montre le Tableau 3, nous utilisons la catégorisation donnée par [48] qui montre les configurations réalisables. Certaines configurations sont difficiles à mettre en œuvre et d'autre nécessitent des ressources matérielles importantes avec des contraintes temps-réel rigoureuses.

Tableau 3. Composants de processus réels et simulés pour la simulation HIL

Cas	**Actionneur**		**Procédé**		**Capteurs**	
	Réel	**Simulé**	**Réel**	**Simulé**	**Réel**	**Simulé**
1	X	—	—	X	—	X
2	X	—	—	X	X	—
3	X	—	X	—	X	—
4	—	X	—	X	—	X
5	—	X	—	X	X	—

La Figure 12 illustre une représentation graphique des différentes configurations possibles dans un même banc de test, où on peut passer d'une configuration à une autre au cours du même cycle de développement. Toutefois, ces transitions sont très difficiles à mettre en pratique car à chaque étape, une nouvelle interface physique doit être implémentée entre la partie simulée et l'actionneur ou le capteur réel. Dans ce cas, les interfaces physiques peuvent être de simples convertisseurs analogique/numérique, des convertisseurs de puissance, des relais électriques, ou même des interfaces de communication numériques.

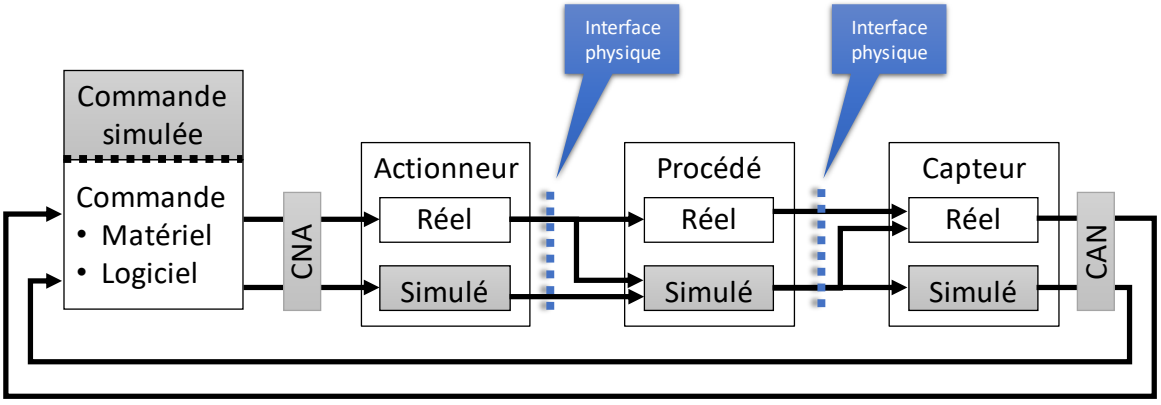


Figure 12. Les différentes configurations possibles pour la simulation Hardware-In-the-Loop

5.4.2 Intérêts de la simulation Hardware-In-the-Loop

Plusieurs raisons ont conduit au développement des plateformes de simulation HIL [50] [51]. Nous pouvons citer à titre d'exemple :

- Les scénarios critiques peuvent être testés sans compromettre le matériel ou les personnes. Les composants critiques pour la sécurité (tels que les variateurs de vitesse) peuvent être testés sans danger.
- L'automatisation des tests est possible. Les tests HIL sont des tests coût-efficacité. Un scénario est plus facile à configurer dans un environnement virtuel que dans un environnement réel. Ainsi, par exemple dans le domaine de la conception automobile, moins de tests de conduite et d'expériences sur banc d'essai sont nécessaires, mais également moins de prototypes de véhicules ou de composants de véhicules, ce qui entraîne des économies en termes de coûts de développement.
- D'une part, l'environnement virtuel peut être facilement modifié, de sorte qu'un contrôle réel (le calculateur réel) peut être testé dans différentes boucles de contrôle. D'autre part, une véritable boucle de contrôle (par exemple un moteur) peut être facilement testée avec différents contrôleurs simulés (ou algorithmes de contrôle). De simples modifications de paramètres permettent de simuler des conditions ambiantes difficiles (essais en hiver, pluie, températures élevées ou basses…etc.).
- Les tests étant immédiatement reproductibles, les défaillances des composants et les scénarios d'urgence associés peuvent être testés de manière reproductible.
- Le matériel peut être testé en parallèle dans différents environnements de travail (par exemple dans une chambre climatique).
- Si certaines parties d'un système sont trop complexes pour être modélisées, elles peuvent être remplacées dans la simulation par le matériel réel.
- Le matériel réel intégré (Embedded real hardware) peut accélérer toute la simulation.
- Dans l'automobile par exemple, le matériel et les logiciels du calculateur peuvent être testés à un stade précoce de développement, un vrai moteur ou une vraie transmission n'est pas nécessaire.

5.4.3 Types de HIL pour la commande des systèmes de puissance

Dans les systèmes de contrôle de puissance la plateforme de simulation HIL est généralement composée de trois éléments ; le matériel sous-test, le système simulé et l'interface. À partir de ces éléments, nous pouvons classer la simulation HIL en deux catégories principales : la simulation contrôleur HIL (**C-HIL)** et la simulation Power HIL (**P-HIL**) [52] [53] [54].

A. Simulation contrôleur HIL

La structure d'une simulation C-HIL typique est présentée à la Figure 13. La partie matérielle contient uniquement le contrôleur, et le système de puissance est entièrement modélisé et simulé par

la plate-forme de simulation numérique. Des signaux de faible puissance sont échangés dans ce cas entre le simulateur et le matériel. Pour tenir compte des effets de la chaîne d'acquisition, il est possible d'utiliser des convertisseurs analogique-numérique et numérique-analogique (ADC, DAC) associés à des circuits de conditionnement.

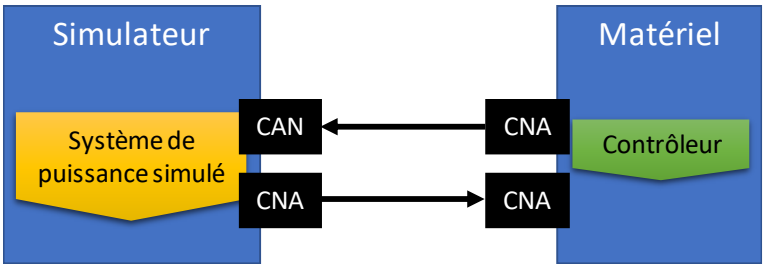


Figure 13. Structure d'une simulation C-HIL

Lorsque des convertisseurs de puissance sont utilisés dans le système simulé, les signaux d'interface sont les signaux de commande. Il convient également de noter qu'une simulation C-HIL peut être réalisée dans le même périphérique matériel où le contrôleur et le modèle de système en temps réel sont implémentés, où l'interfaçage est alors inhérent au périphérique (interfaçage entièrement numérique).

Une simulation C-HIL permet une première validation réaliste et fournit des premières garanties de fonctionnement expérimental du système électrique global. L'utilisateur peut ensuite valider le contrôleur développé dans une large gamme de conditions réalistes, notamment des conditions de fonctionnement sûres ou fortement défaillantes. Cela a le mérite de minimiser le temps et le coût de développement, ainsi que les risques de dommages.

B. Simulation Puissance HIL (Power HIL)

La configuration de base d'une simulation P-HIL est illustrée dans la Figure 14, comme pour une simulation C-HIL, une partie du système de puissance est simulée et l'autre partie étant réelle, sauf que dans ce cas la totalité ou une partie du matériel de puissance réel est utilisé (ex : système de conversion, turbine, batterie, réseau électrique, etc.) [55].

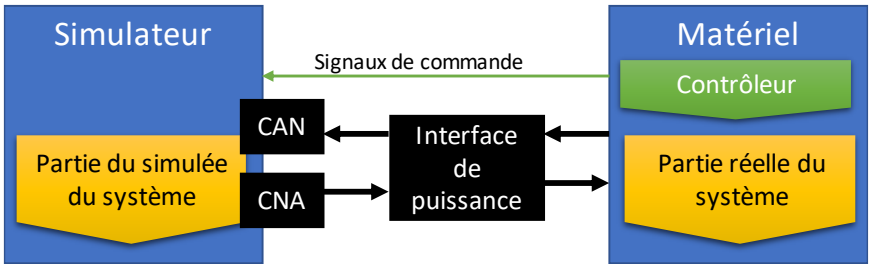


Figure 14. Structure d'une simulation P-HIL

Si le matériel sous-test absorbe ou génère une quantité de puissance considérable, des dispositifs d'amplification de puissance et de conversion spécialement conçus (par exemple, des actionneurs, des convertisseurs de puissance continue/continue) doivent être déployés pour interconnecter le simulateur et le matériel.

Un autre élément est à préciser, c'est que de par leurs structures, les plateformes de simulation P-HIL ne peuvent être que des plateformes temps réel. Le matériel de puissance réel peut être testé de manière réaliste, répétitive et sûre (équivalent du test sur le terrain) et même dans des conditions de fonctionnement extrêmes.

6 CONCLUSION

Dans ce deuxième chapitre, nous avons présenté le cadre de nos travaux de thèse qui est la conception système, notamment la conception des systèmes multidisciplinaires. De ce fait, nous avons exposé les bases fondamentales nécessaires à la compréhension des approches de conception des systèmes complexes en s'appuyant sur la description de l'approche de l'ingénierie des systèmes.

Pour formaliser et appréhender la conception système, nous avons décrit les démarches visant à répondre aux problématiques de la conception multi-abstraction et multi-niveaux. Dans ce contexte, nous avons mis en exergue les outils et les méthodes de conception système telles que la méthode d'ingénierie basée sur les modèles MBSE et les cycles de développement système.

Dans ce contexte, nous avons identifié l'importance de la modélisation et de la simulation pour accompagner le processus de vérification et de validation tout au long du cycle de conception système. Ces étapes de développement permettent de s'assurer que le système est conforme aux spécifications d'exigence et aux attentes de la partie prenante.

La simulation étant une phase importante dans notre démarche de conception, nous lui avons accordait une grande partie de ce chapitre. Nous avons ainsi présenté certaines définitions et certaines méthodes qui sont impliquées dans la conception des systèmes, particulièrement les systèmes de conversion de la puissance électrique. Par ailleurs., nous avons présenté la technique de simulation Hardware-In-the-Loop qui est une pièce maitresse de notre démarche de conception.

Chapitre 3

Méthodologie proposée pour la conception système

1 INTRODUCTION

Dans ce troisième chapitre, nous allons décrire la méthodologie de conception proposée pour le développement de produits multi-technologiques. Le principal but de cette méthodologie est de répondre aux exigences de l'ingénierie système et de soutenir les phases de conception et de vérification d'un système à concevoir. Ce chapitre présente les principaux concepts théoriques et les fondements de la méthodologie proposée, ainsi que les éléments qui permettent sa mise en œuvre et son exploitation.

La conception d'un système multi-technologique, en particulier les systèmes de conversion de puissance, nécessite différentes compétences dans différentes disciplines de l'ingénierie, ce qui implique une grande dépendance vis-à-vis d'un grand nombre d'outils de développement avec différentes approches de conception. Par conséquent, la diversité des outils utilisés dans le processus de conception d'un système peut rendre la conception de plus en plus complexe avec une issue incertaine.

L'approche proposée se base sur l'utilisation d'un cadre de conception dans lequel la simulation multidisciplinaire occupe une position centrale. Ce cadre de conception joue un rôle important dans la vérification des modèles et l'optimisation du processus de conception, ainsi que dans l'interopérabilité entre différents niveaux de modélisation. Cette approche se base sur l'utilisation des langages de description matérielle VHDL et VHDL-AMS, qui permettent de renforcer les différentes étapes de la conception système.

2 BESOIN D'UNE METHODOLOGIE DE CONCEPTION ADAPTEE

Généralement, la conception d'un système multi-technologique est effectuée dans un environnement collaboratif constitué de différentes équipes multidisciplinaires. Où chaque équipe disciplinaire conçoit et optimise les éléments liés à son domaine. Ensuite, viens la phase d'intégration pour rassembler ces différents éléments dans la même structure. Lorsque la mise en œuvre du système ne prend pas en considération toutes les problématiques liées à la conception multidisciplinaire, l'intégration va mettre à découvert, tardivement, les problèmes de compatibilité et de cohérence entre les différents composants multidisciplinaires du système.

Une des solutions les plus appropriées à cette problématique est l'utilisation d'une méthodologie de conception basée sur les modèles. En effet, cette approche apporte beaucoup de solution en ce qui concerne la cohérence de la conception multidisciplinaire et multi-niveaux en s'appuyant sur l'utilisation des modèles comme vecteur de connaissance intra-disciplinaire. Toutefois, d'autres problèmes de collaboration ont vu le jour, tel que l'utilisation d'une pléiade très disparate d'outils et de langages de modélisation selon le niveau d'abstraction ou le type de discipline impliquée dans la conception du système à concevoir.

En outre, la vérification et la validation du système sont assurées par des logiciels de simulation durant le processus de conception afin de parvenir à un design qui vérifie les exigences imposées dans

le cahier des charges. La multidisciplinarité des logiciels utilisés engendre des problèmes d'échanges de données aussi bien pour la modélisation que pour la simulation.

L'objectif principal de nos travaux de recherche vise à développer une nouvelle approche pour circonscrire la plupart des problèmes liés à la modélisation et à la conception des systèmes multidisciplinaires. Cet objectif reste très difficile à mettre en œuvre dans le cadre des travaux d'une thèse de doctorat, c'est pour cette raison que nous avons choisi de viser quelques aspects concrets afin de canaliser notre démarche de recherche.

- Harmoniser l'interfaçage entre les modèles des éléments pluridisciplinaires du système : l'utilisation de plusieurs langages de modélisation implique l'utilisation de plusieurs sémantiques qui sont souvent incompatibles les unes aux autres,
- Autoriser l'association de modèles issus des formalismes analogiques et discrets dans le même contexte de modélisation et de simulation,
- Autoriser l'utilisation et l'association de modèles issus de différents niveaux d'abstraction dans le même contexte de modélisation et de simulation,
- Autoriser le passage d'un formalisme à l'autre sans porter atteinte à l'intégrité du modèle système,
- Permettre une modélisation multidisciplinaire dans le même environnement de développement.

Ces objectifs auront un impact direct sur le processus de développement de manière à :

- Améliorer la collaboration entre les différentes disciplines sur tout le cycle de développement,
- Rendre la prise de décisions dans le cadre d'un projet de conception plus aisée grâce à la remontée fréquente et régulière d'informations opérationnelles et précises,
- Permettre la traçabilité des modifications apportées aux données de définition et exigences du système à concevoir,
- Un gain de temps de développement considérable,
- Une réduction des coûts de mise en œuvre.

3 LA CONCEPTION DES SYSTEMES DE CONVERSION DE PUISSANCE

Dans notre démarche de recherche, nous avons choisi d'appliquer notre approche de conception à la conception de système de conversion de la puissance électrique. Toutefois, l'approche que nous proposons peut-être facilement adapter à d'autres systèmes qui y sont similaires en termes de fonctionnement et/ou en termes de structure. De tels systèmes associent plusieurs disciplines technologiques, ainsi qu'un sous-système de commande et de régulation souvent numérique.

3.1 SYSTEME DE CONVERSION DE PUISSANCE

Le développement des systèmes de conversion de la puissance électrique implique plusieurs domaines technologiques, tels que l'électromécanique, l'automatisme, l'électronique, la thermique, l'électromagnétisme, la fluidique, etc. Ces différents domaines sont impliqués directement dans les différentes phases de développement du produit. Tout système de conversion de puissance met en œuvre un ou plusieurs convertisseurs statiques pilotés par un système de commande (voir Figure 15). Ce système de commande communique avec le système de puissance pour connaître l'état de ce dernier, afin de modifier son fonctionnement ou de le maintenir, tout en pilotant de manière adéquate les composants semi-conducteurs qui y sont associés.

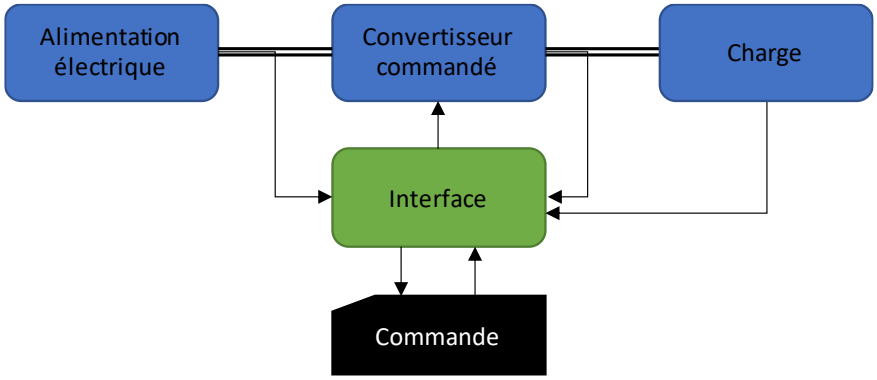


Figure 15. Structure générale d'un système de conversion de l'énergie électrique

La commande des systèmes de puissance connait une évolution croissante tant en performances qu'en support de mise en œuvre. La technologie de mise en œuvre a fini par converger vers des solutions numériques intégrés tels que les ASIC, les DSP, les microcontrôleurs et les FPGA [56].

Au-delà des solutions logicielles traditionnelles, les nouvelles solutions matérielles telles que les FPGA ou les ASIC sont aussi considérées comme étant des solutions numériques appropriées pour l'implantation des algorithmes de commande des convertisseurs d'électronique de puissance. L'implantation de ces algorithmes dans leur intégralité sur des cibles matérielles telles que les FPGA est une démarche qui nécessite une parfaite maîtrise des processus de conception et un travail spécifique d'adéquation entre le système à contrôler et l'architecture de commande à intégrer. Donc, un savoir-faire méthodologique est nécessaire aux concepteurs utilisant les composants de type FPGA afin de satisfaire l'ensemble des contraintes inhérentes à l'implantation du système, tout en apportant une flexibilité de développement suffisante.

Avec l'utilisation des FPGA les problématiques de conception des systèmes de contrôle s'accentuent pendant les phases de validation et de vérification, où l'évaluation et la mise au point ralentisse de façon considérable le processus de développement. En effet, l'exécution de ces tâches nécessite l'assemblage et la mise en marche de tout le système de puissance. Pour pallier ce problème, on utilise la simulation Hardware-In-the-Loop qui permet de tester l'implémentation réelle du système de contrôle avec les modèles exécutables du système de puissance, ceci permet une validation rigoureuse du système avant même son intégration effective.

3.2 APPORTS DES CIRCUITS FPGAS DANS LES SYSTEMES DE CONVERSION DE PUISSANCE

Un composant FPGA est un circuit intégré numérique composé essentiellement d'un grand nombre de blocs logiques programmables et reconfigurables. Aujourd'hui, les composants FPGAs sont devenus indispensables dans les systèmes numériques et sont utilisés dans de multiples domaines d'applications en raison des nombreux avantages obtenus lors de leur utilisation [57], [58]. Parmi tous ces avantages, on peut notamment citer :

1. L'augmentation croissante du niveau de performance temps réel tout en réduisant le coût et l'encombrement.
2. L'amélioration des performances. Ce mode d'implantation permet par exemple de réduire le temps d'exécution d'un algorithme afin de permettre au contrôleur à base de FPGA d'atteindre le niveau de performance des contrôleurs analogiques, sans présenter les inconvénients de ces derniers (Divergence, manque de souplesse, problème de compatibilité électromagnétique, etc.).
3. Leur grande souplesse de programmation permet de les réutiliser à volonté pour cibler des algorithmes différents en un temps très court, ceci à l'aide d'une même plate-forme de développement.
4. La rapidité et la facilité de reconfigurer un FPGA autant de fois que nécessaire pour implanter les fonctionnalités désirées.

En raison de tous ces avantages, les FPGAs sont aujourd'hui utilisés dans diverses applications nécessitant des traitements numériques importants tels que le traitement du signal et de l'image, le contrôle/commande des machines électriques, la mesure de vitesse, le contrôle des convertisseurs statiques de puissance, les équipements médicaux, la télécommunication, l'aéronautique, les transports, la bio-informatique, l'automobile, la robotique ou encore plus généralement l'accélération des calculs scientifiques.

3.3 PROTOTYPAGE A BASE DE CIRCUIT FPGA

Les outils de conception pour les FPGA permettent de transformer le modèle conceptuel spécifié à l'aide de schémas ou d'un langage textuel, en une suite de données binaires permettant de programmer le circuit FPGA. La Figure 16 montre un exemple de flot de conception générique d'un circuit FPGA, depuis la spécification de la fonctionnalité de l'algorithme jusqu'au fichier de configuration, connu sous le nom de « Bitstream File ». Ce fichier contient des informations sur la manière dont les éléments fonctionnels du composant FPGA doivent être reliés, en passant par les étapes de synthèse, de placement et de routage.

L'utilisation des langages de description matérielle s'est généralisée avec l'avènement des FPGAs, particulièrement pour la conception des systèmes numériques complexes. Ils offrent un processus de conception efficace et plus rapide que celui basé sur les schémas logiques, aussi ils permettent un passage plus fiable entre la spécification d'exigences et l'implémentation du système. Les langages HDL notamment le VHDL offrent beaucoup d'avantages à la conception des systèmes à base de FPGA, ils permettent une description du système à plusieurs niveaux d'abstraction, ainsi

qu'une méthode de conception modulaire et hiérarchisée avec réutilisation de codes anciens ou externes.

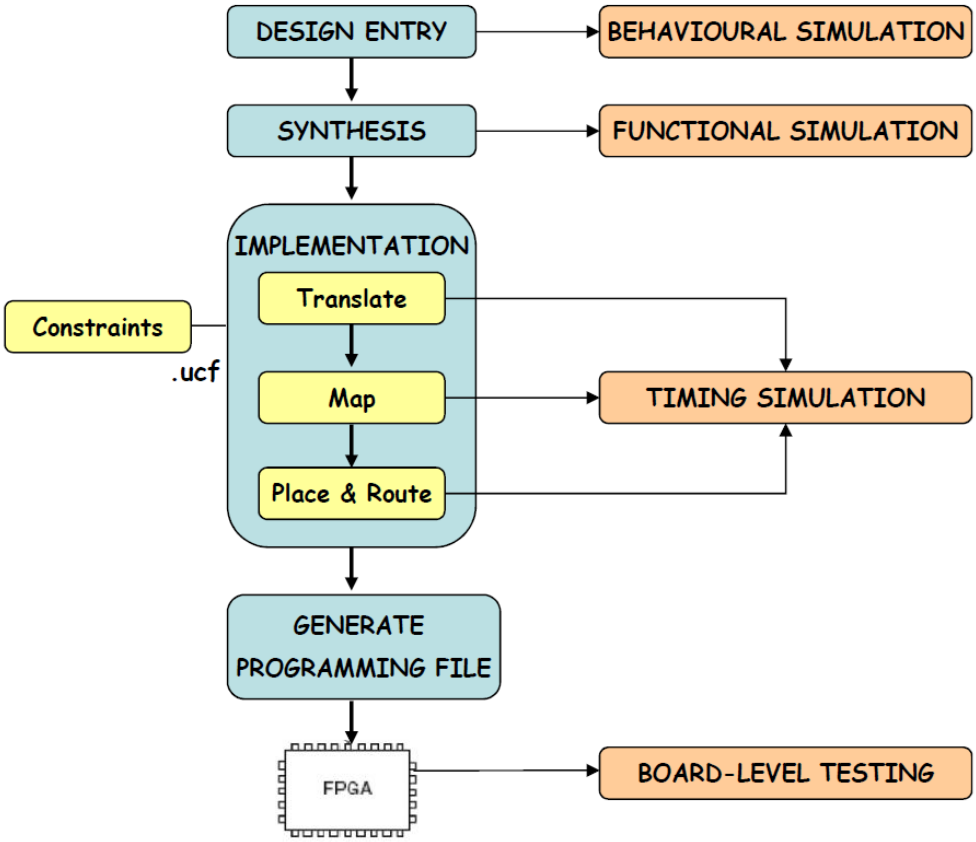


Figure 16. Flot de conception FPGA (prototypage)

Toutefois, des insuffisances constatées dans le flot de conception de certain type de système, à base de FPGA, freinent leur mise en œuvre matérielle. Ces insuffisances sont liées aux processus de vérification et de validation. Ceci est dû au fait que les tests sur les modèles décrits en VHDL sont réalisés dans un environnement de développement (IDE) dédié, qui ne permet pas de vérifier le fonctionnement des modèles dans leur environnement d'application. Les méthodes de vérification qu'on peut utiliser dans ces IDE concernent seulement la vérification du fonctionnement logique du code VHDL, la détection des problèmes temporels ainsi que l'estimation de l'énergie consommée par le circuit FPGA [59].

De nouvelles méthodes de vérification et de validation sont en plein développement, ces méthodes utilisent la cosimulation avec plusieurs logiciels de simulation comme moyen de vérification et de prototypage FPGA, ainsi des systèmes multidisciplinaires complets sont simulés pour évaluer le fonctionnement des contrôleurs numérique compte tenu du fonctionnement dynamique du système commandé [60] [62]. Ces méthodes utilisent pour la plupart des logiciels de simulation multi-domaines tel que le MATLAB/Simulink en le complétant avec des logiciels de développement dédiés aux FPGAs.

4 METHODOLOGIE DE CONCEPTION

Après avoir défini le contexte de notre démarche, ainsi que les différents aspects mis en jeu dans la conception système, Il nous faut à présent proposer une méthodologie qui réunisse ces différents éléments techniques et organisationnels. La méthodologie que nous proposons est basée sur la démarche d'ingénierie système basée sur les modèles (MBSE). L'utilisation de la MBSE implique l'utilisation d'un processus développement multidisciplinaire basé sur l'utilisation de modèles comme principal artefact [63]. Pour tirer pleinement parti de la MBSE, il est impératif de créer des liens entre l'ingénierie système et l'ingénierie disciplinaire [64], c'est pourquoi il faut lier la spécification des activités du processus de développement à la classe du système à concevoir.

4.1 LES PILIERS D'UNE METHODOLOGIE DE CONCEPTION

L'application d'une méthodologie d'ingénierie système basée sur les modèles repose sur trois piliers qui sont les processus, les méthodes et les outils [14] :

- ***Un processus*** est une séquence logique de tâches effectuées pour atteindre un objectif particulier. Un processus définit le « QUOI » qui doit être fait sans spécifier le « COMMENT », chaque tâche est effectuée. La structure d'un processus fournit plusieurs niveaux d'agrégation pour permettre l'analyse et la définition à différents niveaux de détail afin de prendre en charge différents besoins de prise de décision.
- ***Une méthode*** consiste en des techniques permettant d'exécuter un ensemble de tâches. En d'autres termes, elle définit le « COMMENT » de chaque tâche. À tous les niveaux, les tâches d'un processus de développement sont exécutées à l'aide de méthodes. Cependant, chaque méthode est aussi un processus en soi, avec une séquence de tâches à exécuter pour cette méthode particulière. En d'autres termes, le « COMMENT » à un niveau d'abstraction plus haut devient le « QUOI » au niveau inférieur suivant.
- ***Un outil*** est un instrument qui, appliqué à une méthode particulière, peut améliorer l'efficacité de la tâche. Pourvu qu'il soit appliqué correctement par quelqu'un qui possède les compétences et la formation appropriée. L'objectif d'un outil devrait être de faciliter la réalisation des « COMMENT ». De manière plus générale, un outil améliore le « QUOI » et le « COMMENT ». La plupart des outils utilisés pour soutenir l'ingénierie des systèmes sont basés sur des ordinateurs et des logiciels appelés aussi outils d'ingénierie assistée par ordinateur (IAO).

Sur la base de ces définitions, une méthodologie peut être définie comme un ensemble de processus, méthodes et outils associés. Une méthodologie est essentiellement considérée comme moyen d'application des processus, des méthodes et des outils connexes à une classe de problèmes qui ont tous quelque chose en commun.

Finalement, l'environnement de développement comprend le milieu, les objets externes, les conditions ou les facteurs qui influencent les actions d'un objet, d'une personne ou d'un groupe. Ces conditions peuvent être sociales, culturelles, personnelles, physiques, organisationnelles ou

fonctionnelles. Un environnement de projet doit avoir pour objectif d'intégrer et de soutenir l'utilisation des outils et des méthodes utilisés dans le cadre de ce projet. Ainsi, l'environnement active (ou désactive) le « QUOI » et le « COMMENT ».

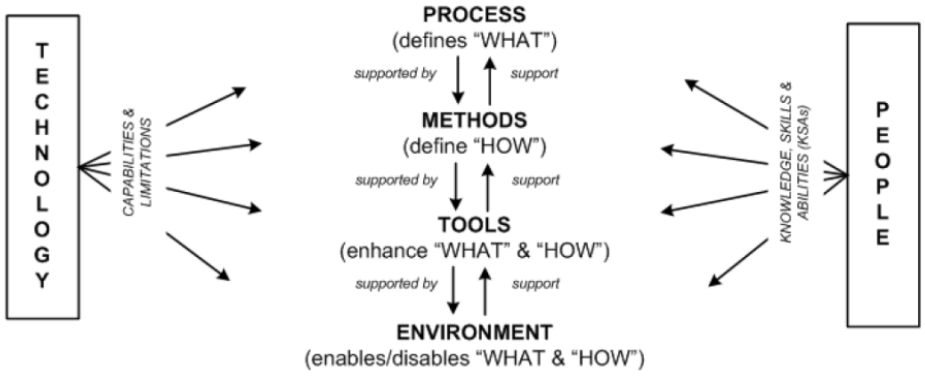


Figure 17. Processus, méthodes et outils de développement [14]

La Figure 17 illustre la relation entre les éléments précédemment cités, ainsi que l'effet de la technologie et des personnes sur ces éléments. Le concept de "méthodologie" va englober toutes ces notions, liant un processus à une méthode ainsi qu'à un langage.

5 OUTIL DE MODELISATION ET DE SIMULATION

L'approche que nous proposons s'appuie essentiellement sur l'utilisation d'un outil unique pour de modélisation et de simulation qui peut soutenir les différentes phases du processus de développement système. Aussi, cet outil doit surpasser les contraintes liées à la modélisation, la simulation, la vérification et la validation des systèmes multidisciplinaires.

Une multitude d'outils de modélisation et de simulation existent, qui sont destinés à la conception des systèmes multidisciplinaires, et qui satisfont les activités de conception liées à l'approche MBSE. Nous nous intéressons dans cette section aux langages de modélisation tels que des langages propriétaires (*MATLAB*, *MAST*), ou des langages du domaine public tel que *Modelica* ou des langages normalisés tel que *VHDL-AMS.* Les langages de modélisation sont des outils conceptuels offrant la possibilité aux concepteurs de formaliser leurs pensées et de conceptualiser la réalité sous forme explicite, ces langages peuvent être textuels ou graphiques [65].

Notre problématique de thèse est de réussir à modéliser dans un seul langage, unifié et à plusieurs niveaux d'abstraction, des systèmes multi-technologiques pouvant être régie en tout ou en partie, par les lois de conservation de l'énergie d'une part et qui embarque des éléments de traitement numérique d'autre part. A travers l'utilisation d'un même langage nous souhaitons être capables d'adopter aussi bien une vue structurelle qu'une vue comportementale du système ou de ces composants.

5.1 LES LANGAGES DE MODELISATION

Afin de choisir le bon langage de modélisation, nous devons définir un certain nombre de critères de sélection qui permettront de choisir le langage le plus approprié à notre méthodologie de conception. Les critères suivants doivent être respectés par le langage sélectionné :

- ✓ **La portée du langage :**
  - Le langage doit supporter la modélisation et la simulation comportementales et structurelles,
  - Le langage doit supporter la modélisation multi-domaines (électrique, mécanique, fluidique, thermique, ...etc.),
  - Le langage doit prendre en charge la modélisation des équations différentielles et algébriques (DAE),
  - Le langage doit prendre en charge la modélisation et la simulation continue, discrète et mixte.
- ✓ **La modularité :**
  - Le langage doit permettre une modélisation multi-niveau et multi-abstraction,
  - Le langage doit supporter la hiérarchisation des composants et la réutilisation,
  - Le langage doit supporter la transformation de et vers d'autres langages de modélisation tel que SysML.
- ✓ **La fiabilité du langage :**
  - Le langage doit être standardisé et normalisé,
  - Le langage doit être utilisé par une communauté active,
  - Doit avoir un environnement de simulation de préférence open-source.

Un certain nombre de langages satisfaits plus ou moins un certain nombre de ces critères, parmi ces langages on peut citer : MATLAB, MAST, VHDL-AMS, Verilog-AMS, SystemC-AMS, Modelica. D'ores et déjà, les langages de modélisation et de simulation tels que MATLAB/Simulink ainsi que le langage MAST sont à écarter, ces langages sont des langages propriétaires qui ne sont pas normalisés est leur pérennité dépend de leurs propriétaires respectifs. Concernant les autres langages de modélisation notamment les langages de description matériel (HDL - Hardware Description Language), nous cherchons un langage de modélisation qui sera adapté à la modélisation des systèmes numériques à base de circuits FPGA.

Dans la suite de cette section, nous allons présenter quelques caractéristiques de ces différents langages de modélisation.

5.1.1 MATLAB/Simulink

MATLAB est un langage de haut niveau puissant, particulièrement adapté à la démonstration des concepts mathématiques. MATLAB offre un environnement de travail utile pour un calcul rapide de modèle ainsi que des procédures de simulation complètes. Il est utilisé partout dans le monde et dans des centaines de disciplines différentes, le volume de données associé sur le net est énorme. MATLAB peut être complété avec plusieurs outils, appelés toolbox tels que :

- Simulink : Framework de simulation graphique,
- Control System Toolbox : Analyse de modèles linéaires invariants dans le temps,
- Neural Network Toolbox : Modélisation de réseaux de neurones,
- Optimization Toolbox : Optimisation des problèmes,
- Statistics Toolbox : Modélisation et analyse de modèles statistiques.

Simulink est une extension graphique de MATLAB permettant de créer des diagrammes à l'aide de blocs. Ces diagrammes représentent des systèmes et des fonctions mathématiques.

5.1.2 Modelica

Modelica est un langage orienté objet permettant de décrire les systèmes d'équation algébrique différentielle (DAE) combinés à des événements discrets [66], [67]. De tels modèles sont parfaitement adaptés à la représentation de flux d'énergie, de matériaux, de signaux ou d'autres interactions continues entre les composants du système.

Modelica est un langage non propriétaire orienté objet, il est adapté à la modélisation de systèmes physiques complexes. Modelica est construit sur une modélisation causale avec des équations mathématiques et des constructions orientées objet. Il est conçu pour prendre en charge un développement efficace de bibliothèques qui permet échange de modèles simplifiés. Enfin, OpenModelica propose un moteur de simulation gratuit et puissant pour effectuer des expériences pratiques et valider l'approche proposée.

5.1.3 VHDL-AMS

VHDL-AMS est un langage de description matérielle dont le rôle est de vérifier le comportement d'un système composé d'une partie analogique et d'une partie numérique. VHDL-AMS a été normalisé en décembre 1999 sous la référence IEEE 1076.1-1999 [68]. Il a été développé comme une extension du langage VHDL qui ne permet pas de représenter des modèles à temps continu. La partie AMS du langage permet de représenter des signaux analogiques et mixtes de manière continue. De plus, ces signaux supportent des discontinuités dans le temps et peuvent être décrits par des équations différentielles et algébriques. Ces signaux permettent donc de simuler des événements physiques tels que la vitesse, la force mécanique, la pression et l'écoulement fluidique, entre autres grâce à l'application généralisée des lois de Kirchhoff (conservation de l'énergie).

5.1.4 Verilog-AMS

Verilog-AMS est un dérivé du langage de description matériel Verilog, créé sous la tutelle d'Accellera (organisation de normalisation EDA) [69] [70]. Il comprend des extensions analogiques et des signaux mixtes AMS (Analog and Mixed-Signal) afin de définir le comportement des systèmes à signaux analogiques et mixtes. La norme Verilog-AMS a été instaurée dans l'intention de permettre aux concepteurs de systèmes à signaux analogiques et mixtes et de circuits intégrés de pouvoir créer et d'utiliser des modules qui encapsulent les descriptions de comportement de haut niveau, aussi bien que des descriptions structurelles de systèmes et de composants. Verilog-AMS définit un langage de modélisation standardisé par l'industrie pour les circuits à signaux mixtes. Il fournit à la fois le temps-continu et les sémantiques de modélisation d'événements. Il est donc approprié pour les circuits

analogiques, numériques et mixtes. Il est important de noter que le Verilog ne constitue pas un langage de programmation. Il s'agit d'un langage de description du matériel.

5.1.5 SystemC AMS

SystemC est un ensemble de bibliothèques "open source" de classes écrites en C++ qui permettent de modéliser au niveau système un système numérique sur puce et de vérifier que tous les composants de natures différentes communiquent correctement entre eux [71]. Ce langage est largement utilisé pour la modélisation des composants numériques et logiciels.

SystemC-AMS est une extension du langage SystemC [72] [73], cette extension permet de modéliser des composants analogiques et analogues. La version courante de SystemC-AMS est optimisée pour les applications dominées par le traitement de signal. En théorie, ce langage permet la modélisation des systèmes conservatifs et non conservatifs (orientés flot de signal) avec des équations dynamiques et statiques non linéaires. Son problème majeur est que rare sont les logiciels de simulation qui le prennent en charge.

5.2 CHOIX DU LANGAGE DE MODELISATION

D'après l'étude des caractéristiques des différentes langages cités ci-dessus, nous pouvons considérer que le langage VHDL-AMS est le plus adapté à nos attentes et à nos besoins. En effet, dans ce travail nous utilisons VHDL-AMS comme pilier de notre méthodologie de conception système. Il sera utilisé comme langage de support pour la modélisation et la simulation dans les différentes phases du processus de conception.

VHDL-AMS est à l'origine une extension du langage VHDL [74], et le langage VHDL fait toujours partie intégrante du standard VHDL-AMS [75] . Cependant, dans de nombreuses applications qui utilisent le VHDL-AMS, la partie numérique (qui est en VHDL) est généralement négligée ou séparée de la conception et seul l'aspect analogique du langage VHDL-AMS est utilisé [76] [78] . Dans d'autres travaux tels que [79] le VHDL est utilisé lors de la modélisation au niveau comportementale pour évaluer la partie analogique d'une conception complexe. Dans d'autres travaux [80] [82], la partie numérique permet de décrire le contrôleur numérique d'un système mixte, puis le code est transformé en une forme synthétisable pour une éventuelle implémentation, cependant, le code synthétisé n'est jamais vérifié avec le modèle de conception et aucune garantie que le résultat de l'implémentation correspondra à celui du modèle initialement étudié.

Contrairement aux utilisations précitées, dans ce travail le langage VHDL est exploité pour améliorer les pratiques de conception qui utilise le VHDL-AMS et permet ainsi la validation et la vérification des modèles de façon rapide est efficace. Dans ce travail, le VHDL-AMS fonctionne à pleine capacité, ceci à travers la validation des deux points suivants :

- Utilisation du langage VHDL-AMS pour la modélisation de systèmes mixtes à des fins de simulation et de vérification.
- Utilisation du langage VHDL-AMS et du VHDL afin de réaliser des simulations comportementales, l'implémentation matérielle et la validation de tout le système ciblé.

Mais avant d'aborder son utilisation dans notre méthodologie de conception, nous devons présenter l'essentiel des caractéristiques de ce langage.

5.3 LE LANGAGE VHDL-AMS

5.3.1 Présentation

Le langage IEEE 1076.1, appelé de manière informelle VHDL-AMS (dont la dernière révision est IEEE 1076.1-2017) est un sur-ensemble du langage IEEE Std 1076-2008 (VHDL) qui fournit des fonctionnalités pour la description et la simulation de systèmes analogiques et à signaux mixtes avec une sémantique conservative et non conservative pour la partie analogique d'un système multidisciplinaire [83] [84]. Le langage prend en charge la modélisation à de nombreux niveaux d'abstraction dans les domaines de l'énergie électrique et non électrique pour les systèmes pouvant être décrits par des équations différentielles ordinaires et des équations algébriques. Le langage ne spécifie aucune technique particulière pour résoudre les équations, mais définit plutôt les résultats à atteindre. La solution des équations peut inclure des discontinuités, l'interaction entre la partie numérique d'un modèle et sa partie analogique est prise en charge de manière flexible et efficace. Enfin, la simulation des signaux et du bruit dans le domaine fréquentiel est prise en charge [75].

En tant qu'extension du langage VHDL, le langage VHDL-AMS profite tout de même des capacités de ce langage pour modéliser des systèmes qui manipulent des signaux à évènement discret en utilisant des processus sensibles aux changements de signal. En plus des instructions concurrente prise en charge par le VHDL, le VHDL-AMS intègre d'autres types d'instruction simultanée qui permettent de simuler des valeurs à temps continu. Les interactions entre les parties discrètes et continues d'un modèle sont gérées de manière flexible et efficace par le VHDL-AMS [68].

5.3.2 Modélisation multi-domaines

Le standard VHDL-AMS permet la modélisation multi-domaines évitant l'obstacle à la communication entre les domaines scientifiques, ainsi la norme autorise l'échange entre collaborateurs ou différents groupes d'ingénierie dans le cadre la prise en charge de la hiérarchisation des modèles et la construction de bibliothèques spécialisés [84].

A cette flexibilité d'utilisation s'ajoute la possibilité pour les concepteurs d'aborder leurs modèles à différents niveaux d'abstraction. En effet, VHDL-AMS propose des mécanismes permettant de gérer [85]:

- Les abstractions comportementales (modélisation de la fonction du système),
- Les abstractions structurelles (le système est divisé en sous-systèmes qui peuvent eux-mêmes être modélisés au moyen de différentes abstractions),
- Les abstractions flot de données (signal flow) (enchaînement de blocs fonctionnels dont les entrées n'ont pas d'influence sur les sorties des blocs précédents).

Le grand atout de ce langage est de supporter la modélisation et la simulation des différents éléments d'un système mixte et multidisciplinaire dans un environnement unique, optimisant ainsi l'analyse et la mise au point du système à concevoir. Cette souplesse d'emploi permet aux concepteurs

(électronicien, automaticien, mécanicien, …) de modéliser des éléments d'un système dans un domaine spécifique sans se soucier de leur intégration et interaction avec les autres éléments modélisés.

Un autre atout de VHDL-AMS est sa transparence : le concepteur peut construire ses propres modèles comportementaux ou structurels de manière simple et flexible. Et d'un autre côté, l'utilisateur dispose de ces modèles, où il peut les modifier et les adapter aux besoins de son travail. Ainsi, tous les modèles déjà créés sont archivables dans une bibliothèque pour être modifiés ou réutilisés plus tard. L'avantage majeur en matière de productivité est lorsqu'une bibliothèque comporte un nombre important de composants et qui seront mis à disposition pour être intégrés à des systèmes plus complexes.

Le VHDL-AMS permet d'associer des modèles de haut niveau et des modèles de bas niveau dans le même modèle global ayant une approche structurelle. Ainsi, on peut détailler certaines parties plus finement, alors que d'autres parties sont modélisées de manière plus abstraite, requérant ainsi un temps de calcul inférieur de simulation.

Aussi, le langage VHDL-AMS supporte la description des systèmes conservatifs, à travers l'utilisation des lois de Kirchhoff généralisées (Egalité des efforts et somme des flux égale à zéro) pour modéliser les systèmes physiques complexes. Une autre caractéristique importante du langage réside dans ses capacités de traitement des équations implicites : un système complexe peut être décrit à l'aide d'un système d'équations différentielles non orientées et linéaires ou non, sans avoir recours à la simplification ou à la linéarisation des équations.

5.3.3 Structure d'un modèle VHDL-AMS

Un modèle VHDL-AMS est constitué de deux parties principales : la spécification d'entité (désigné par le mot clé : **ENTITY**) qui correspond à la vue externe du modèle et l'architecture de l'entité (désigné par le mot clé : **ARCHITECTURE**) qui correspond à la vue interne du modèle. La Figure 18 montre la vue générale d'un modèle VHDL-AMS.

Entité d'un modèle VHDL-AMS

L'entité définit l'interface du modèle avec son monde extérieur (son environnement). Elle permet, après appel de bibliothèques utiles (mot clef : **LIBRARY**) et spécification du contenu à exporter (mot clef : **USE**), de communiquer entre le monde extérieur et le modèle au moyen de deux objets : **GENERIC** et **PORT**.

Les « génériques » sont des constantes à valeurs différées (les paramètres) qui peuvent être modifiés par la suite et servent à rendre le modèle plus général. Les ports sont les nœuds dynamiques par lesquels l'entité pourra recevoir et envoyer des informations à et vers son environnement.

Les ports peuvent être de plusieurs classes :

- Les ports de classe « signal » (mot clé : **SIGNAL**) définissent des canaux de communication directionnels ou bidirectionnels modélisant des signaux a évènement discret.

- Les ports de classe « quantité » (mot clé : **QUANTITY**) définissent des points de connexions analogiques directionnels d'entrée et bidirectionnels d'entrée/sortie pour les informations.
- Les ports de classe « terminal » (mot clef : **TERMINAL**) définissent des points de connexions analogiques pour lesquels les lois de conservation de l'énergie (lois de Kirchhoff pour les circuits électriques ou lois équivalentes pour les systèmes non électriques) sont satisfaites.

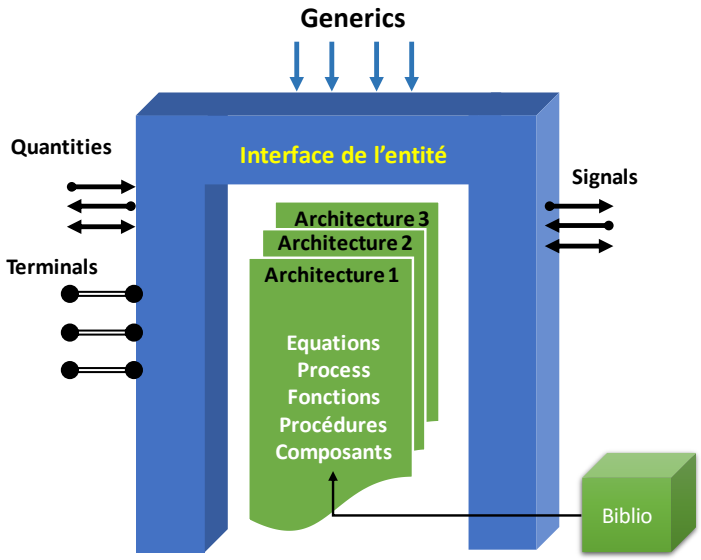


Figure 18. Vue générale d'un modèle VHDL-AMS

L'entité peut être écrite, compilée et mise à disposition avant de connaître l'architecture associée. Une fois l'analyse des besoins effectuée, on sait quels sont les ports d'entrée-sorties nécessaires. Cette souplesse permet de concevoir une ossature du système alors que l'architecture du modèle VHDL-AMS n'est pas encore définie.

Une entité qui ne définit aucune déclaration est un banc de test, souvent appelée "Testbench". Le modèle interne ne pouvant être excité de l'extérieur doit être autonome pour son fonctionnement et son exécution, les testbenchs sont des outils qui imbriquent les modèles sous test de mise en œuvre d'environnement de test contrôlé.

Architecture

Une architecture définit le comportement et/ou la structure du modèle. Elle est reliée à une unique entité et hérite ainsi de toutes les déclarations faites à son niveau (les génériques, les interfaces d'entrée/sortie, etc.). L'architecture de l'entité permet, après une zone de déclaration locale, de définir le fonctionnement du modèle par l'intermédiaire d'instructions concurrentes permettant de manipuler l'information numérique et d'instructions simultanées décrivant le comportement à temps continu du modèle. L'intérêt du VHDL-AMS est que toutes ces instructions peuvent cohabiter, offrant ainsi un support à la multi-abstraction et à la conception mixte.

Pour une seule entité donnée, il peut y avoir plusieurs architectures ce qui est une des forces du langage VHDL-AMS. L'intérêt de disposer de plusieurs architectures pour le même modèle est extrêmement important :

- Ça permet de tester plusieurs implémentations et de choisir la plus appropriée.
- Ça permet de passer d'un niveau d'abstraction à un autre sans changer l'interface du modèle.
- Ça permet de créer des architectures comportementales qui serviront de références à la validation des exigences.

5.3.4 Les classes d'objet

A. Terminaux et Natures

Le terminal est un objet VHDL-AMS utile pour la spécification de points de connexion pour lesquels les lois de conservation d'énergie sont satisfaites. Les terminaux peuvent être déclarés dans les interfaces des entités ou dans les zones de déclarations des architectures. Dans ce dernier cas, le terminal est local à l'architecture qui contient sa déclaration.

Les terminaux ne définissent explicitement aucune valeur mais doivent être associés à des quantités. Leur rôle donc est de faciliter la description des systèmes conservatifs. Les quantités associées sont appelées les quantités de branche (***branch quantity***).

Un terminal est associé à une « nature », qui représente un domaine physique ou une discipline particulière (électrique, mécanique, thermique, …etc.), permettant de faire les vérifications de cohérence d'associations. Chaque nature est caractérisée par deux classes de quantités liées à des effets physiques. Les quantités « **ACROSS** » représentent un effort et les quantités « **THROUGH** » représentent un flux, comme par exemple la tension et le courant, respectivement, pour les systèmes électriques. Ces deux classes de grandeurs permettent de définir la notion d'énergie et de puissance d'un système physique (Figure 19).

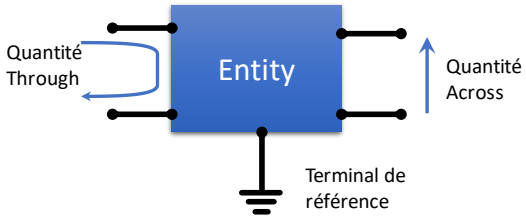


Figure 19. Les terminaux externes de l'entité

Une nature est l'association de deux types réels et d'un identificateur. Les natures sont associées aux mots clés effort "***across***" et flux "***through***" (Tableau 4).

Tableau 4. Flux et efforts relatifs à divers domaines physiques

Nature (domaine)	Effort (*across*)	Flux (*through*)
Electrique	Tension	Courant
Thermique	Température	Puissance (flux)
Mécanique de rotation	Angle de vélocité	Torsion
Mécanique de translation	Vélocité	Force
Magnétique	Force magnétique	Flux magnétique
Hydraulique	Pression	Flux (débit)
Radiatif	Dose	Photocourant

B. Quantités

Les quantités (***quantity***) représentent des variables à valeurs réelles du temps, typiquement les inconnues du système d'équations impliquées dans la résolution du modèle VHDL-AMS. Les quantités ne prennent pas leurs valeurs par affectation, mais par la résolution d'un système d'équations.

Il existe plusieurs variétés de quantités :

- Une quantité libre (***free_quantity***) peut être déclarée localement dans une architecture. Dans ce cas elle n'est disponible que localement. Elle peut être initialisée (sa valeur initiale par défaut est zéro), affectée et associée aux équations simultanées.
- Les quantités de branche (***branch_quantity***) permettent d'associer des quantités à des terminaux. Les quantités de branches héritent des types associés à la nature des terminaux définissant les branches (les deux terminaux d'une branche doivent être obligatoirement d'une même nature)
- Le langage VHDL-AMS définit aussi des quantités implicites (***implicit_quantity***) : c'est-à-dire des quantités qui n'ont pas besoin d'être déclarées, mais qui sont liées à d'autres quantités explicitement déclarées. Un exemple de quantités implicites est donné ci-dessous. Soit Q une quantité explicitement déclarée :
  - ***Q'dot*** représente la dérivée temporelle première de la quantité Q.
  - ***Q'Integ*** représente l'intégrale de la quantité Q sur un intervalle de temps allant de zéro au temps courant.

5.3.5 Types d'analyses

VHDL-AMS supporte différents types d'analyses, le simulateur utilisé doit fournir une indication explicite nommé ***DOMAIN***. Les types possibles d'analyses sont les suivants :

- Calcul du point de fonctionnement (DC) :
  Pour déterminer le point de fonctionnement, on utilise l'état « quiescent » (QUIESCENT_DOMAIN) (voir le code VHDL-AMS ci-après). Ceci permet de contourner une limitation des simulateurs, qui ne prennent en compte la construction BREAK pour les conditions initiales, comme le simulateur ***ADVanceMS*** par exemple.

```
IF (domain = quiescent_domain) USE          --DC analysis
        V == 0.0;
ELSE                                        --TR analysis
        I == C * V'dot;
END USE;                                    --End Domain
```

- Analyse temporelle (TRANSITOIRE) :
  On utilise l'analyse temporelle (TIME_DOMAIN) pour déterminer le comportement transitoire temporel d'un système.
- Analyse fréquentielle (AC NOISE) :

L'analyse fréquentielle en petits-signaux (FREQUENCY_DOMAIN) permet l'étude du régime permanent d'un circuit en ayant recours à des techniques de simulation simples et rapides. Le but est d'obtenir les caractéristiques en fréquence d'un circuit lorsqu'il est stimulé par des signaux sinusoïdaux de faibles amplitudes. L'analyse fréquentielle peut être l'analyse de bruit ou l'analyse harmonique.

5.3.6 Simulation mixte

Un modèle mixte inclut des parties ayant un comportement dirigé par les événements (parties logiques) et des parties ayant un comportement continu (parties analogiques) qui interagissent. La simulation mixte est un élément incontournable dans la conception des systèmes multidisciplinaires notamment des systèmes de conversion de puissance. Où les modèles des éléments de puissance (source, convertisseur statique, charge), des capteurs et des éléments analogiques sont décrits en VHDL-AMS, et de l'autre côté le langage VHDL est utilisé pour la description de la commande numérique.

Le langage VHDL-AMS offre la possibilité de passer d'une d'interface logique vers une interface analogique et vice-versa. Ces conversions se font par le biais de certaines procédures de programmation qui utilisent, par exemple, les processus numériques ou bien par l'utilisation de certaines fonctions et attributs propre au langage VHDL-AMS telles que :

- L'interface analogique numérique peut se faire par l'intermédiaire des quantités implicites ***S'RAMP*** et ***S'SLEW*** et de l'instruction ***BREAK on***.
- L'interface numérique analogique est disponible par l'intermédiaire du signal implicite ***Q'ABOVE***.

5.3.7 Prise en charge par le simulateur

Pour le langage VHDL-AMS, deux simulateurs cohabitent, un analogique (solveur) et l'autre numérique. Le simulateur analogique, après résolution des équations différentielles (à travers des calculs matriciels d'ordre élevé régis par les instructions concurrentes) produit des ASP (Analog Signal Point – point de simulation analogique) synonymes de convergence. Le simulateur numérique produit des événements à partir des équations logiques régies par des instructions concurrentes : c'est à dire des instructions qui permettent de modéliser le comportement d'un composant sous la forme d'évènements qui peuvent s'exécuter de manière asynchrone [86].

Les noyaux de simulation ne créent pas forcément les résultats au même instant. Le langage propose un mécanisme de synchronisation des noyaux pour autoriser l'échange des données entre les modèle discret et le modèle logique.

5.3.8 Avantages du langage

Lorsque le langage VHDL-AMS a été créé, il existait de nombreux langages de conception propriétaires pour chaque fondeur ou fournisseur. Ceci était un obstacle à la communication entre domaines scientifiques et posait de graves problèmes aux sociétés lorsqu'un intervenant de la chaîne venait à disparaître ou à être remplacé, car le portage des modèles était alors très délicat et nécessitait de nombreuses heures de travail supplémentaires.

VHDL-AMS est quant à lui un produit non propriétaire et normalisé par l'IEEE qui tend à être reconnu par le plus grand nombre. L'utilisation généralisée de ce langage facilite la communication entre les différents domaines scientifiques grâce à son approche multi-domaines native qui permet aussi bien à un électronicien qu'à un mécanicien ou même un chimiste de modéliser la partie d'un dispositif qui le concerne directement sans problèmes de dialogue avec les autres parties [83].

La grande force de ce langage est de permettre la simulation mixte en autorisant aussi bien les modélisations à temps continu (analogiques) qu'à évènements discrets (logiques) ou mélangeant les deux. A cette flexibilité d'emploi s'ajoute la possibilité pour les concepteurs d'aborder leurs modèles à différents niveaux d'abstraction. En effet, le VHDL-AMS propose des mécanismes qui permettent de gérer aussi bien les abstractions comportementales (c'est la fonction réalisée par le système qui est modélisée et non sa physique), que les abstractions structurelles (le système est divisé en sous-ensembles qui peuvent eux-mêmes être modélisés au moyen de différentes abstractions, …) ou bien de type workflow (enchaînement de blocs fonctionnels dont les entrées n'ont pas d'influence sur les sorties des blocs précédents). Les modèles créés avec VHDL-AMS peuvent donc aussi bien être descriptifs que prédictifs.

VHDL-AMS présente les avantages suivants :

- C’est une norme IEEE 1076.1-1999,
- Les modèles sont réutilisables (Modularité, bibliothèques),
- Les modèles sont multidisciplinaires (paquetages, ports conservatifs et non conservatifs, continus et non continus),
- Il offre des instructions simultanées et concurrentes,
- Il permet d’écrire des équations implicites et explicites,
- Il offre des formulations procédurales et simultanées,
- La possibilité de décrire des comportements continus par morceaux est possible.
- L’initialisation par l’utilisateur en cas de discontinuité est permise.
- Le langage définit une sémantique de conservation d’énergie (la direction du flot est définie dans la déclaration des quantités),
- Il offre des transformations de Fourier, de Laplace (’LTF) et transformé en Z (’ZTF).
- L’interaction entre les interfaces analogiques et numériques est permise aussi bien que celle entre signaux discrets et signaux continus.

5.3.9 Inconvénients et limitations

VHDL-AMS n'est cependant pas à même de proposer des instructions répondant à tous les besoins des concepteurs de modèles. Par exemple, l'utilisation des dérivations spatiales n'est pas prévue par le langage, ce qui rend délicat les modélisations géométriques. Seules les dérivations temporelles sont acceptées par VHDL-AMS.

Par ailleurs, même si le langage est à même de supporter des solutions à ses manques grâce à ses possibilités d'interfaçage avec d'autres langages (notamment le C/C++), la forme de ces interfaces n'est pas standardisée [77]. De ce fait, les modèles ayant recours à des langages extérieurs à VHDL-AMS ne sont généralement pas portables.

Enfin, le fait que les simulateurs actuels soient basés sur des extensions et des modifications d'anciens simulateurs, et pas encore sur de nouvelles techniques de simulation spécifiques, implique des limitations dans les possibilités de simulation qui empêchent l'implémentation de certaines instructions du langage. C'est le cas, par exemple, des instructions PROCEDURAL et DISCONNECT qui agissent sur la synchronisation des noyaux de simulation et sur la progression de la résolution des systèmes d'équations au cours dans le temps.

5.4 OUTILS DE SIMULATION VHDL-AMS

5.4.1 Environnement de simulation VHDL-AMS

La modélisation en utilisant le langage VHDL-AMS est basée sur différents éléments : structure d'un modèle (entité et architecture), classes d'objets (terminaux, natures et quantités), sémantique de connexion, les instructions (séquentielles, concurrentes et simultanées), types d'analyses, simulation mixte, et critère de solvabilité.

La Figure 20 montre les étapes du processus de simulation :

- Description du système : Ecriture du code du modèle,
- Compilation : Analyses syntaxique et sémantique, elle permet la détection d'erreurs locales qui ne concernent que l'unité compilée,
- Stockage dans la bibliothèque de travail et/ou appel d'une bibliothèque externe.
- Elaboration : Détection d'erreurs globales qui concernent l'ensemble de l'unité de conception,
- Simulation : Calcul du comportement du système,
- Résultat : Tracé la forme d'onde les signal et quantité.

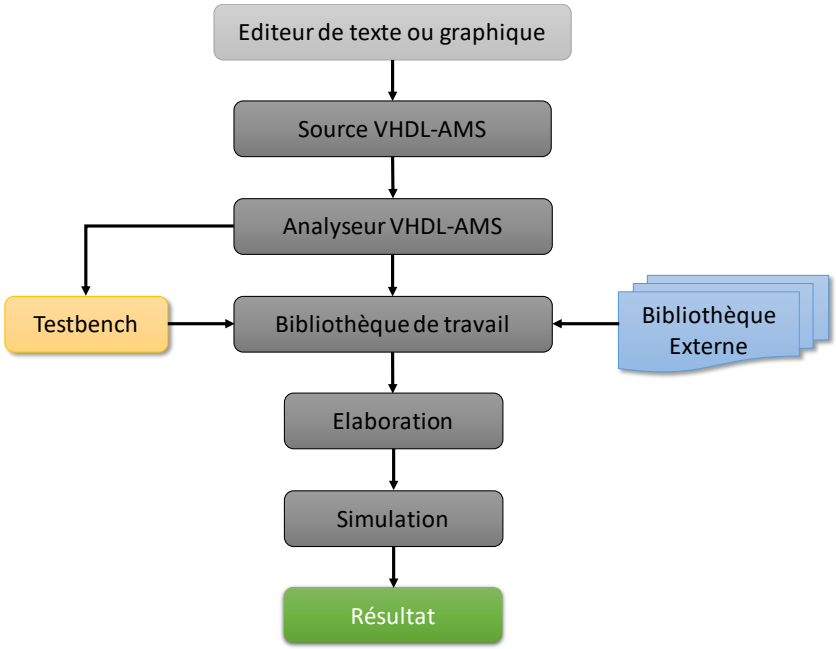


Figure 20. Processus de simulation VHDL-AMS

5.4.2 Principaux outils de simulation

Il existe sur le marché plusieurs outils qui supportent le langage VHDL-AMS. Cependant, les fabricants d'outils logiciels ne réalisent pas des implémentations totalement conformes à la norme : les instructions ne correspondent pas à 100% de ce qui était prévu par le standard. De plus, certaines instructions ne sont pas opérationnelles dans certains simulateurs. En dépit de ces manquements, nous pouvons citer quelques logiciels qui sont compatibles avec le langage VHDL-AMS.

- ANSYS Simplorer,
- Cadence Virtuoso AMS Designer,
- Dolphin Integration SMASH,
- Mentor Graphics Questa ADMS,
- Mentor Graphics SystemVision,
- Synopsys SaberRD,

6 PRINCIPE DE LA METHODOLOGIE DE CONCEPTION PROPOSEE

Durant le cycle de vie des systèmes multi-technologiques, améliorer la méthodologie de conception améliorera de façon considérable les métriques d'évaluation d'un système [12], ce qui permet d'avoir un système en complète concordance aux exigences et aux besoins visés. Pour ce faire, nous avons besoin d'une méthode et d'un processus de conception qui intégrent des tâches telles que la simulation à différents niveaux abstraction, le prototypage fonctionnel, la vérification et la validation, la simulation Hardware-In-the-Loop, etc. En parallèle, un outil de conception adéquat est nécessaire pour accélérer chaque étape de la conception, sans perdre en précision et en fiabilité.

L'approche que nous proposons s'appuie essentiellement sur l'utilisation d'un seul langage de description matériel qui est le VHDL-AMS et ce, tout au long du processus de conception système. Aussi, la validation et la vérification se fera à travers ce même langage qui garantira un prototypage efficace et rapide du système de commande numérique en se basant sur les circuits FPGA.

Dans ce travail, nous présentons une nouvelle méthodologie qui accélère et simplifie le processus de développement des systèmes multidisciplinaires, notamment les systèmes de conversion de puissance impliquant un contrôleur numérique intégré. La démarche que nous proposons est basée sur les langages de modélisation VHDL et VHDL-AMS ainsi que sur les composants programmables FPGA. L'idée est d'utiliser le même langage pour modéliser, concevoir et vérifier la partie numérique et la partie analogique d'un système multidisciplinaire et mixte. A la fin du processus de modélisation, la partie numérique est implémentée dans un composant FPGA, la vérification du système est réalisée à travers l'utilisation de la simulation C-HIL.

Dans cette méthodologie, le langage VHDL-AMS est utilisé comme support pour la modélisation et la simulation de toutes les parties d'un système, des modèles structurels et fonctionnels à différents degrés de raffinement sont ensuite utilisés. Le langage VHDL-AMS est bien adapté à la vérification de systèmes complexes à différentes phases de la conception, ainsi qu'à la validation de nouveaux modèles inexplorés. Le VHDL-AMS peut ainsi prévenir les défaillances du produit réel ainsi que des

comportements impropres [4]. Mais actuellement, les pratiques VHDL-AMS restent loin de la mise en œuvre physique et de la synthèse. Pour cette raison, dans la deuxième partie de ce travail, nous déploierons l'utilisation de la modélisation VHDL-AMS à la simulation Hardware-In-the-Loop basée sur les composants FPGA. Cette approche est considérée comme une extension inéluctable de l'utilisation du langage VHDL-AMS pour la conception des systèmes multidisciplinaires.

6.1 PROCESSUS ET METHODE DE CONCEPTION PROPOSES

Dans notre approche, nous visons à réduire les inadéquations dans les transitions entre chaque tâche du processus de développement d'un système multi-technologique, notamment les systèmes de conversion de puissance. Pour ce faire, nous nous basons sur l'association des langages VHDL et VHDL-AMS, ainsi que sur un processus de conception spécifique.

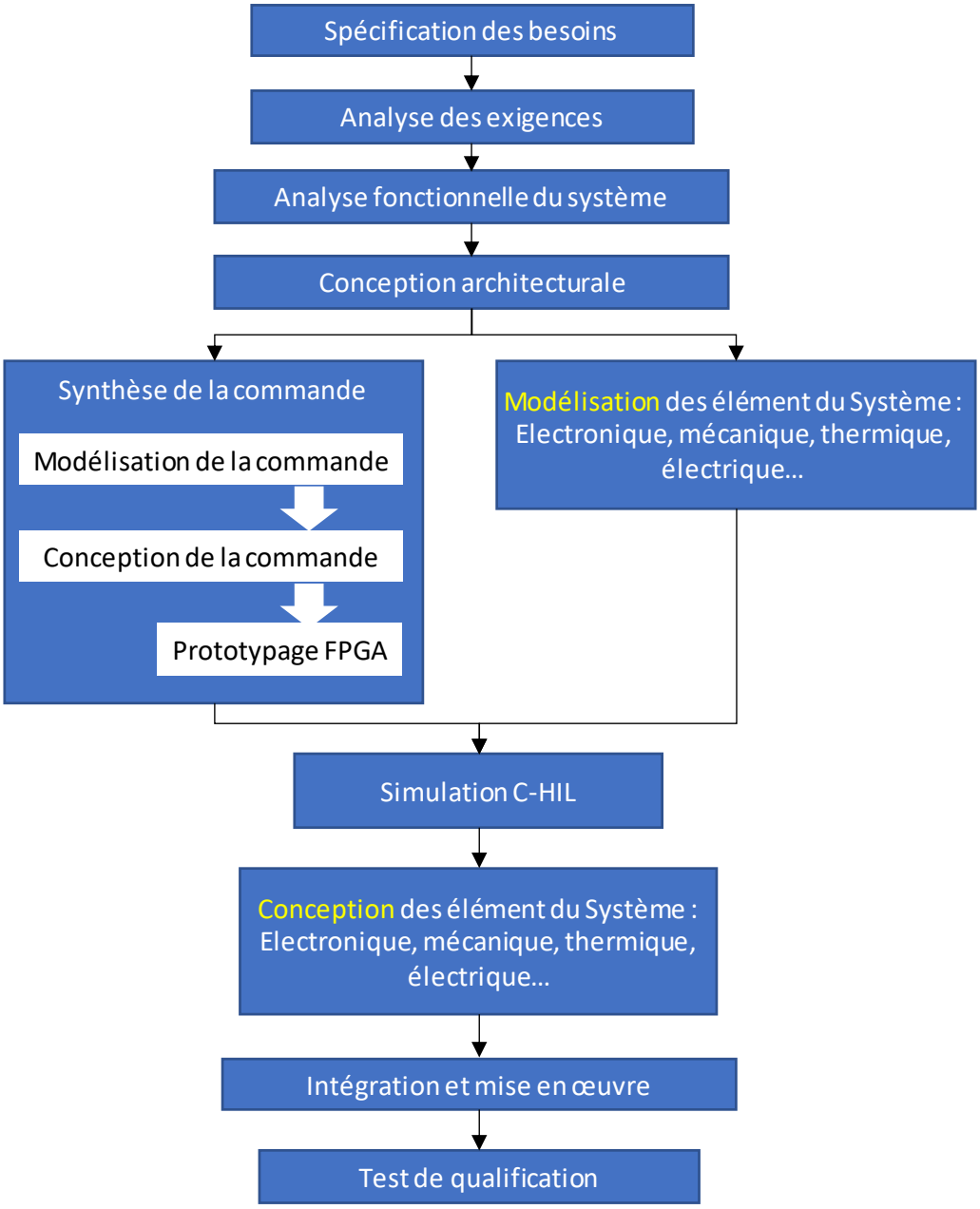


Figure 21. Processus de développement

Dans cette section, la méthodologie de conception proposée sera introduite. Dans cette méthodologie, qui est basée sur les méthodologies MBSE, la simulation et la modélisation occupent une place prédominante dans le processus de développement. La Figure 21 illustre un aperçu général du processus de développement, ce processus est basé sur une approche top-down qui implique plusieurs phases de conception. Après avoir spécifié les exigences du système, des phases de modélisation fonctionnelle et structurelle sont nécessaires pour identifier les différents composants multidisciplinaires associés à la conception du système. La modélisation et la simulation de la commande et des composants systèmes permettront d'accélérer la mise en œuvre du système global.

Comme pour les méthodologies MBSE usuelles, l'utilisation du langage SysML est importante pour la modélisation au niveau système. Lors de la description des premières phases de développement nous ferons référence à différents diagrammes de modélisation SysML qui permettent de structurer et harmoniser notre démarche de conception avec les approches déjà existantes.

Les paragraphes suivants détailleront le processus de conception et les étapes d'ingénierie système basée sur les modèles, et décrivent les concepts associés de gestion, de simulation et de vérification.

6.1.1 Spécification et analyse des exigences

La première phase dans le processus de développement d'un système est la spécification des exigences, vient ensuite une phase d'analyse qui a pour objectif d'examiner et de synthétiser les entrées du processus. Les exigences des parties prenantes sont traduites en exigences système qui définissent ce que le système doit faire (exigences fonctionnelles) et son niveau de performance (exigences de qualité de service).

Les activités essentielles de cette phase sont illustrées à la Figure 22. Cela commence par l'analyse et éventuellement l'affinage des exigences des parties prenantes. Dans l'étape suivante, ces exigences sont transformées en fonctions système requises, ou en exigences non fonctionnels. Pour la traçabilité, les exigences système identifiées sont liées aux exigences des parties prenantes associées.

Les exigences peuvent être appliquées au système global, à chaque sous-système (ou composant) et à l'interconnexion entre deux sous-systèmes. Par conséquent, les exigences peuvent être détaillées en les subdivisant en sous-exigences supplémentaires, créant ainsi une hiérarchie d'exigences.

La prochaine étape majeure de la phase d'analyse des exigences est la définition des cas d'utilisation du système. Un cas d'utilisation décrit un aspect opérationnel spécifique du système. Il spécifie le comportement tel que perçu par les acteurs et le flux de messages entre les acteurs et le cas d'utilisation. Toutefois, il faut d'abord définir les acteurs dans le contexte du système, un acteur peut être une personne, un autre système ou un matériel externe au système en développement.

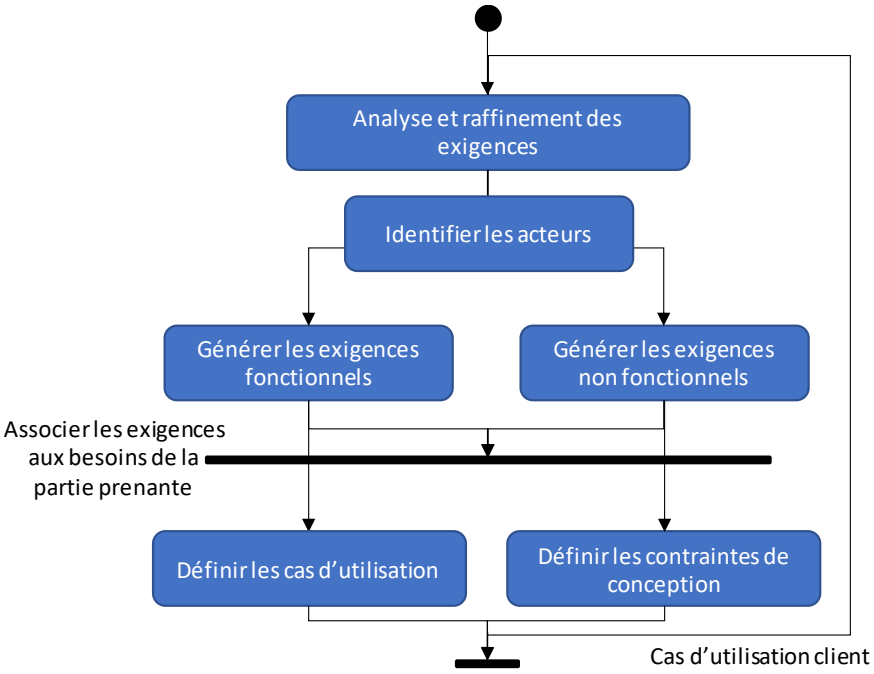


Figure 22. Spécification des exigences.

Les diagrammes SysML qui peuvent être utilisés dans cette phase sont :

- Le diagramme de contexte général.
- Le diagramme des exigences.

6.1.2 Analyse fonctionnelle du système

La phase d'analyse fonctionnelle du système est principalement axée sur la transformation des exigences fonctionnelles du système en une description cohérente des fonctions du système (opération). L'analyse est basée sur les cas d'utilisation, c'est-à-dire que chaque cas d'utilisation au niveau du système identifié lors de la phase précédente d'analyse des exigences est traduit en un modèle.

La Figure 23 détaille les tâches de modélisation. Tout d'abord, le contexte du modèle de cas d'utilisation est défini dans un diagramme synoptique interne. Les éléments de ce diagramme sont des instances de blocs SysML, représentant le cas d'utilisation et ses acteurs associés. Ensuite, viennent les tâches qui correspondent aux définitions des organigrammes fonctionnels du système, des scénarios de cas d'utilisation, des organigrammes d'état (à travers l'utilisation des diagrammes d'activité), des diagrammes de séquence et finalement des diagrammes d'état. L'ordre de ces tâches dépend des informations disponibles et du choix des concepteurs.

A la fin de cette phase est après plusieurs itérations, les modèles des cas d'utilisation sont analysés et éventuellement simulés pour vérifier les séquences générées. Les vérifications doivent aussi s'intéresser à des chevauchements possibles entre les différents diagrammes générés.

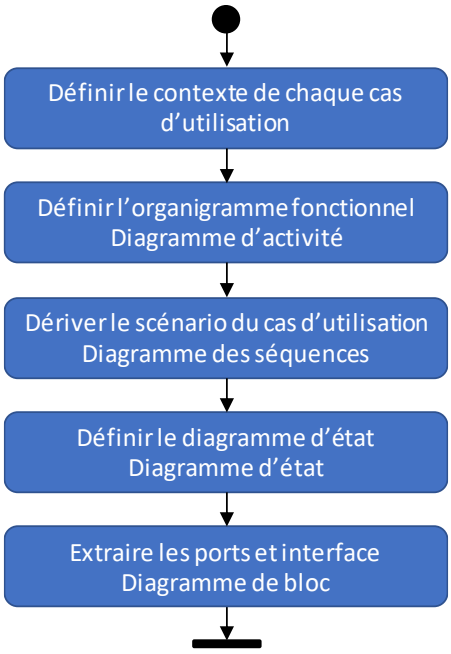


Figure 23. Flot de modélisation fonctionnelle du système.

Si cette phase est correctement achevée, elle doit permettre d'avoir une vue intégrale est sans ambiguïté des fonctionnalités exécutées ou autorisées par le système.

Les diagrammes SysML qui peuvent être utilisés sont :

- Diagramme des cas d'utilisation,
- Diagramme d'activité,
- Diagramme de séquences,
- Diagramme d'état.

6.1.3 Conception architecturale

Après avoir explicitement détaillé les spécifications et les fonctionnalités du système dans les phases de développement précédentes, la phase suivante définie la conception architecturale. Cette phase vise à synthétiser une solution qui satisfaits aux exigences. La phase de conception architecturale est axée sur la mise au point d'une architecture physique (c'est-à-dire un ensemble d'éléments, de système et d'algorithme) capable d'exécuter les fonctions requises dans les limites des contraintes de performances prescrites. La synthèse de conception suit une approche descendante. La Figure 24 décrit les sous-phases essentielles pour parvenir à une solution architecturale en utilisant les outils de modélisation SysML et VHDL-AMS.

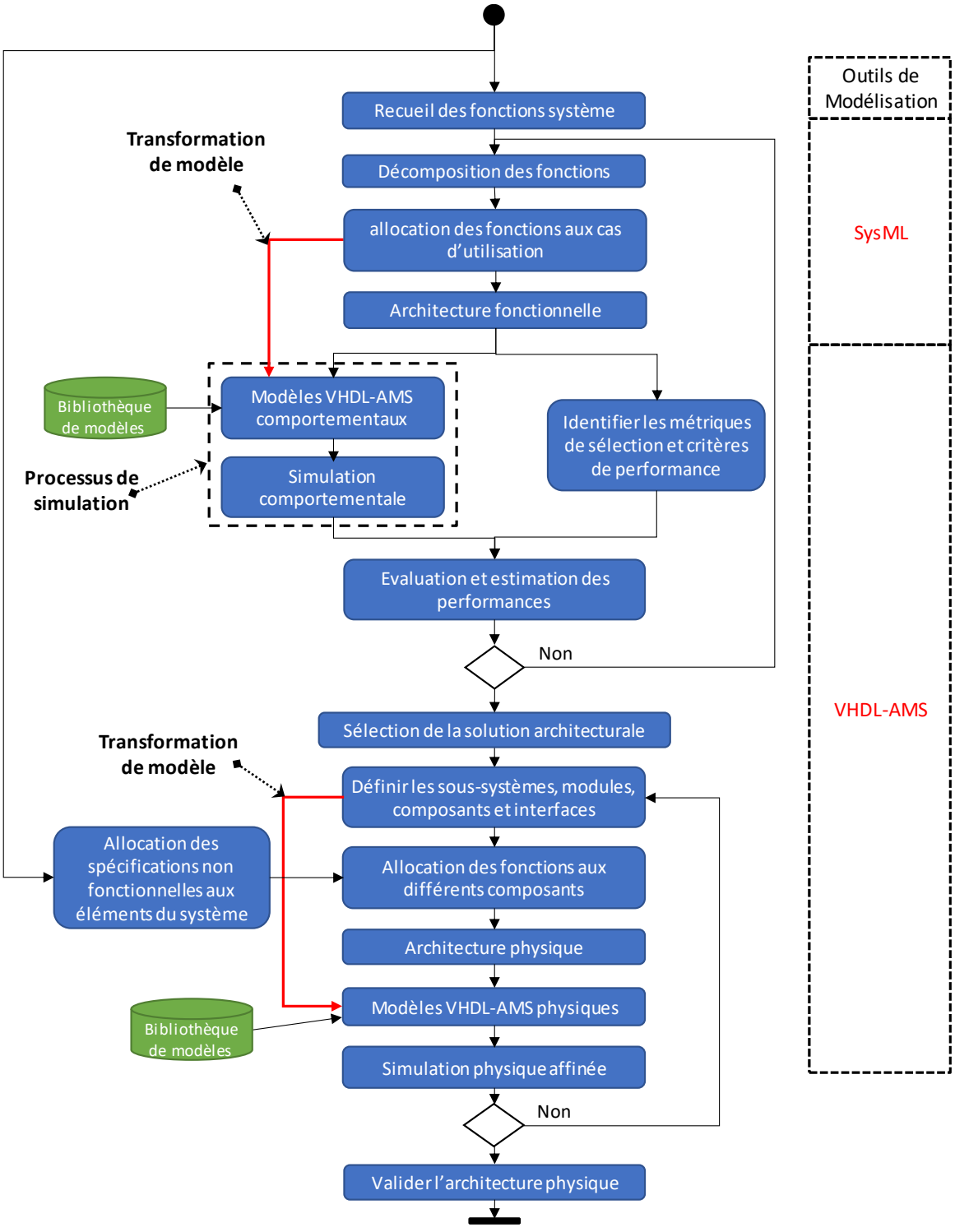


Figure 24. Processus de conception architecturale

A. Recueil des fonctions système

Cette première étape permet de lister les différentes fonctions exécutées par le système, ainsi que les informations et évènements échangés entre les différentes fonctions du système. Ces fonctions qui sont eux même tirées des spécifications du système, permettent l'identification et la mise en œuvre de la structure spéciale et opérationnelle du système (voir Figure 25).

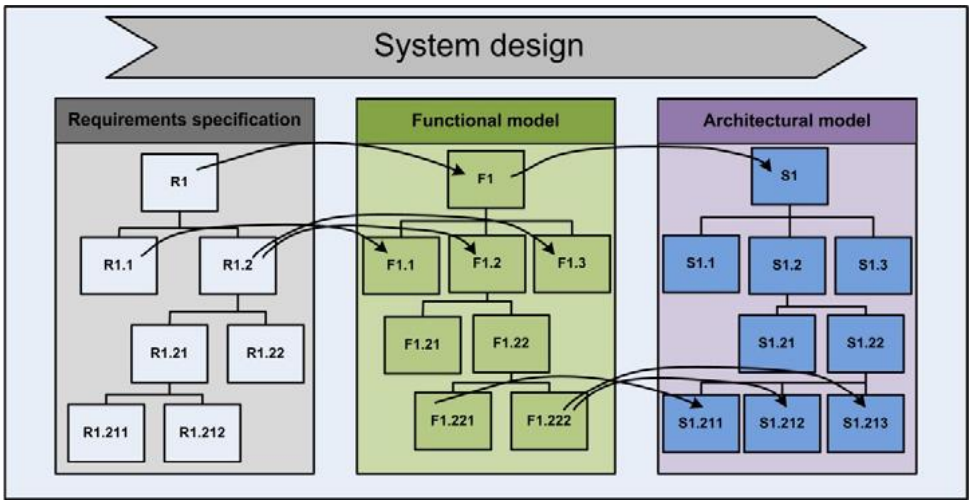


Figure 25. Elaboration de l'architecture à travers l'extraction des fonctionnalités du système

B. Décomposition des fonctions

Dans cette étape, les fonctions sont regroupées dans des sous-ensembles appelés fonctions-clés afin de prendre en charge l'analyse des architectures candidates et leur évaluation. Une fonction-clé système peut être un groupe de fonctions du système qui :

- Sont complémentaires ou étroitement couplées,
- Sont dépendantes d'une seule discipline d'ingénierie,
- Seront spécifiées par un modèle existant (réutilisation),
- Peuvent être réalisées par un seul composant physique,
- Seront réalisées via la réutilisation d'un composant existant (matériel ou logiciel),
- Peuvent être réutilisées dans le système,
- Répondent à une contrainte de conception commune.

C. Allocation des fonctions aux cas d'utilisation

Les cas d'utilisation définis dans la spécification sont ensuite alloués aux fonctions-clés et aux sous-fonctions définies précédemment. Ceci permet la traçabilité des fonctions et permet d'utiliser par la suite les scénarios déjà mis en œuvre.

D. Architecture fonctionnelle

Le résultat des étapes précédentes donne lieu à un ensemble de modules fonctionnels dont l'agencement représentes toutes les exigences du système en termes fonctionnel et représente l'ensembles des cas d'utilisation. L'assemblage des fonctions et de leurs interactions est défini puis représenté par des diagrammes de block interne (en SysML).

Une architecture fonctionnelle ou comportementale est un agencement de fonctions et de leurs sous-fonctions et interfaces (internes et externes) définissant le séquencement d'exécution, les conditions de contrôle ou de flux de données, ainsi que les exigences de performance permettant de satisfaire le référentiel d'exigences [31]. Un modèle d'architecture fonctionnel peut être décrit comme un ensemble de scénarios de fonctions et de modes opérationnels interdépendants. Le choix du groupement des fonctions ainsi que leur agencement permettent de satisfaire un ensemble ou l'ensemble des exigences. Plusieurs architectures candidates sont alors développées avec différents

arrangements et différents niveaux de raffinement, après évaluation, l'architecture la mieux adaptée est adoptée.

En ce qui concerne la vérification et l'évaluation de l'architecture, nous utiliserons la simulation. De ce fait, nous devons transformer l'architecture fonctionnelle candidate (structure architecturale et fonctions) en un modèle simulable. Pour cette tâche critique, nous allons utiliser le langage VHDL-AMS pour une simulation au niveau comportementale.

E. Identifier les métriques de sélection et critères de performance

Afin d'identifier la meilleure solution parmi un ensemble de solutions candidates, il est nécessaire d'identifier des critères d'évaluation. Des critères d'évaluation significatifs sont définis en collaboration avec les parties prenantes et une équipe représentant toutes les disciplines impliquées. En règle générale, les critères d'évaluation sont basés sur les contraintes du client, les caractéristiques de performance requises et les coûts induits.

Tous les critères d'évaluation ne sont pas égaux, certains sont plus importants que d'autres. Alors, les critères d'évaluation sont pondérés en fonction de leur importance relative dans la solution globale.

F. Processus de simulation

A partir d'un schéma de l'architecture fonctionnelle du système, les différents types d'éléments et les différentes connections entre ces éléments sont identifiés. Une fois les différents types d'éléments identifiés, on construit leurs modèles conceptuels. Puis, en instanciant ces modèles conceptuels génériques dans le schéma du système on obtient le modèle conceptuel du système. Le modèle de simulation du système est ainsi construit, nous pouvons élaborer des scénarios de test et exécuter des procédures de simulation selon les besoins de l'analyse souhaitée [33]. (Voir Figure 26)

Cette étape permet d'évaluer la perspicacité de l'architecture candidates, l'utilisation du langage de modélisation VHDL-AMS permet d'exécuter une série de simulation, qui permettront de comparer les architectures les unes aux autres, et de les évaluer par apport aux métriques fixées lors de l'étape précédente.

L'exécutions du processus de simulation implique l'exécution des points suivants :

- Les modèles SysML réalisés lors de la spécification et de l'analyse architecturale seront la base de la simulation comportementale, chaque module fonctionnel décrit dans l'architecture est remplacé par une entité VHDL-AMS. La structure du modèle de simulation globale est tirée de la structure architecturale. Les digrammes des cas d'utilisation, les digrammes d'activité, les diagrammes de séquence, et les digrammes d'état seront les principales sources d'information pour la création des modèles VHDL-AMS, il existe dans la littérature une multitude de méthodes de transformation de modèle SysML en modèle VHDL-AMS comme celles décrite dans [87] [88] [89]. Lors de cette étape de conception architecturale, la mise en œuvre de modèles comportementaux exécutables est suffisante pour l'analyse architecturale.

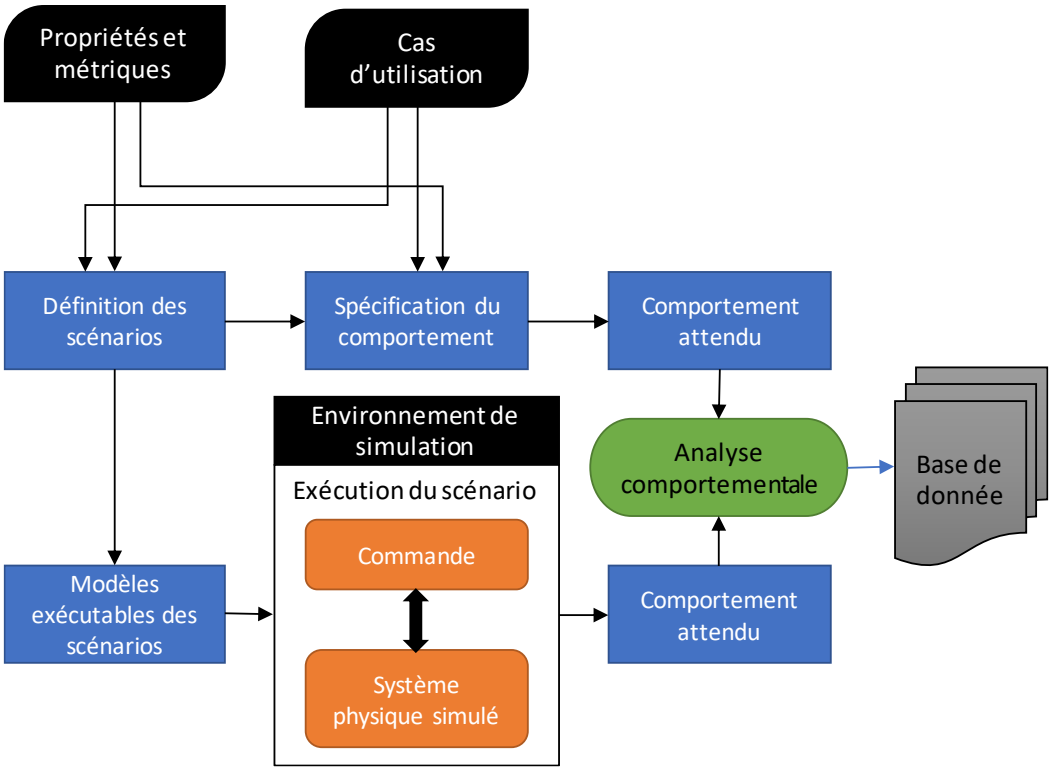


Figure 26. Principe de la vérification par simulation

- De plus, la réutilisation des modèles VHDL-AMS (multi-abstraction) existants est préférable si l'architecture intègre des fonctions similaires. Ceci permet un gain de temps considérable et offre des niveaux de modélisation affinés. Aussi, les bibliothèques VHDL-AMS et VHDL existantes contiennent souvent des modèles déjà élaborés qui représentent des composants ou des modules réels (Régulateur, convertisseur de puissance, mémoire, diode, etc.).

- Des modèles spécifiques appelé « entité testbench » doivent être élaborer pour effectuer une simulation VHDL-AMS, ces modèles peuvent représenter les cas d'utilisation définis dans les phases précédentes du développement du système.

- La simulation comportementale s'exécute rapidement car les détails de la mise en œuvre sont ignorés tout en conservant les détails du fonctionnement. Les modèles comportementaux détaillés sont utiles pour l'évaluation de l'architecture fonctionnelle. De plus, les simulations comportementales sont flexibles, ce qui permet de modifier et de vérifier rapidement la structure d'une architecture ou les propriétés d'une fonction.

G. Vérification et estimation des performances

L'évaluation des propriétés du système et des métriques de sélections permet de classer l'ensemble des architectures considérées. Pour ce faire, un ensemble de tâches doivent être effectuées. Après la mise en œuvre des scénarios de simulation, les modèles de simulation sont exécutés et ensuite comparés avec le comportement attendu qui préalablement est déterminé. L'analyse de l'ensembles des comportements permet de créer une base de données des comportements et des résultats de simulation validés.

H. Sélection de la solution architecturale

A la fin de plusieurs itérations de l'ensemble des tâches précédentes, une architecture est choisie. C'est l'architecture fonctionnelle qui vérifie au plus haut degré les exigences et les contraintes du système à concevoir.

I. Définition des sous-systèmes et des composants physiques

De la même manière que pour les cas d'utilisation et les fonctions clés, il convient d'allouer les fonctions clés aux composants du système et aux composants contenus dans la décomposition physique. Dans la vision physique, les organes réels de l'architecture (logiciel ou matériel) sont définis et choisis soit parmi des composants existants (composants commercialisés sur étagère ou modules déjà conçus) ou à défaut, l'assemblage physique du système est spécifié (spécifications techniques des besoins des composants matériels et logiciels restant à réaliser). La définition des sous-systèmes et composants physiques répond à plusieurs critères tels que : le regroupement des fonctions communes dans un même composant, ou selon les contraintes temporelles (même périodicité) ou spatiales lorsque des données sont partagées.

Le résultat de cette étape est une bibliothèque de composants physiques sous forme d'un ensemble de blocs, ainsi que des tables d'allocation des modules fonctionnels sur les composants physiques.

J. Définition les éléments d'interface

La définition des interfaces des composants est essentielle pour l'agrégation des éléments de l'architecture dans un ensemble homogène est opérationnel. Les interfaces sont des constituants d'architectures qui possèdent certaines caractéristiques spécifiques, ainsi, toute interface relie au minimum deux constituants de l'architecture. Les interfaces principales d'un système concernent :

- Les interfaces internes entre constituants du système.
- Les interfaces physiques et logiques avec les ressources partagées (alimentation en puissance, bus de données).
- Les interfaces logiques entre les fonctions partagées avec d'autres systèmes.
- Les interfaces externes du système avec d'autres systèmes, la structure ou l'environnement.

La nature des interfaces est également spécifiée (électronique, électrique, fluidique, mécanique, Optique, Homme-machine, etc.). De plus, les caractéristiques concernant une information échangée peuvent être spécifiées telles que : la nature de l'information, le format et la dynamique de l'échange de l'information.

K. Architecture physique

Les composants et les sous-systèmes sont ensuite assemblés les uns avec les autres. Les interactions sont définies et représentées dans des diagrammes de bloc sur l'ensemble du système physique. Une vérification de l'architecture est nécessaire pour examiner le fonctionnement de chaque composant, ainsi que le flot de donnée traversant les différentes interfaces de l'architecture.

L. Processus de la simulation détaillée

Dans cette étape, le processus de simulation utilisé est le même que celui exécuté lors de la simulation comportementale, sauf que pour cette étape les modèles VHDL-AMS sont différents. La description de chaque élément physique est détaillée puis affinée dans un niveau d'abstraction approprié au type d'analyse souhaitée. Les détails devront refléter la structure des modèles et leurs comportements dynamiques.

Dans cette phase, nous pouvons d'ores déjà exécuter un modèle numérique de la commande, ceci permet un gain de temps important dans la conception système. Aussi, cette phase permet l'analyse avancée du comportement dynamique du système et de ces différents composants.

En ce qui concerne la simulation mixte, un environnement de simulation spécifique doit être utilisé, il est choisi par apport à sa capacité de supporter la modélisation numérique et mixte avec le langage VHDL-AMS, nous y reviendrons au choix du logiciel de simulation dans le chapitre suivant.

M. Validation de l'architecture physique

Le même processus d'évaluation que celui de la Figure 26 est utilisé. Néanmoins, Il faut ajouter les propriétés technologiques et les métriques dynamiques du système. Aussi, les simulations et le fonctionnement du système lors de la simulation comportementale seront la base de l'évaluation de l'architecture physique, ainsi que les différents composants et sous-systèmes technologiques choisis.

6.1.4 Synthèse de la commande

La méthodologie que nous proposons est dédiée aux applications de contrôle. De ce fait la synthèse de l'algorithme de contrôle prend une place importante dans la conception système. L'approche avancée permet de synthétiser une commande destinée aux circuits FPGA à partir d'un modèle VHDL-AMS. Pour parvenir à un tel résultat, nous devons suivre les étapes décrites dans le processus de synthèse illustré dans la Figure 27.

L'utilisation du langage de modélisation VHDL-AMS permet de simuler dans le même environnement des modèles analogiques avec des modèles numériques destinés à une implémentation dans des circuits FPGA. En l'absence de cette solution, nous devrons utiliser la cosimulation entre deux ou plusieurs logiciels de simulation, par la suite nous devrions réaliser des cycles de vérification fastidieux qui nécessitent beaucoup de temps de développement.

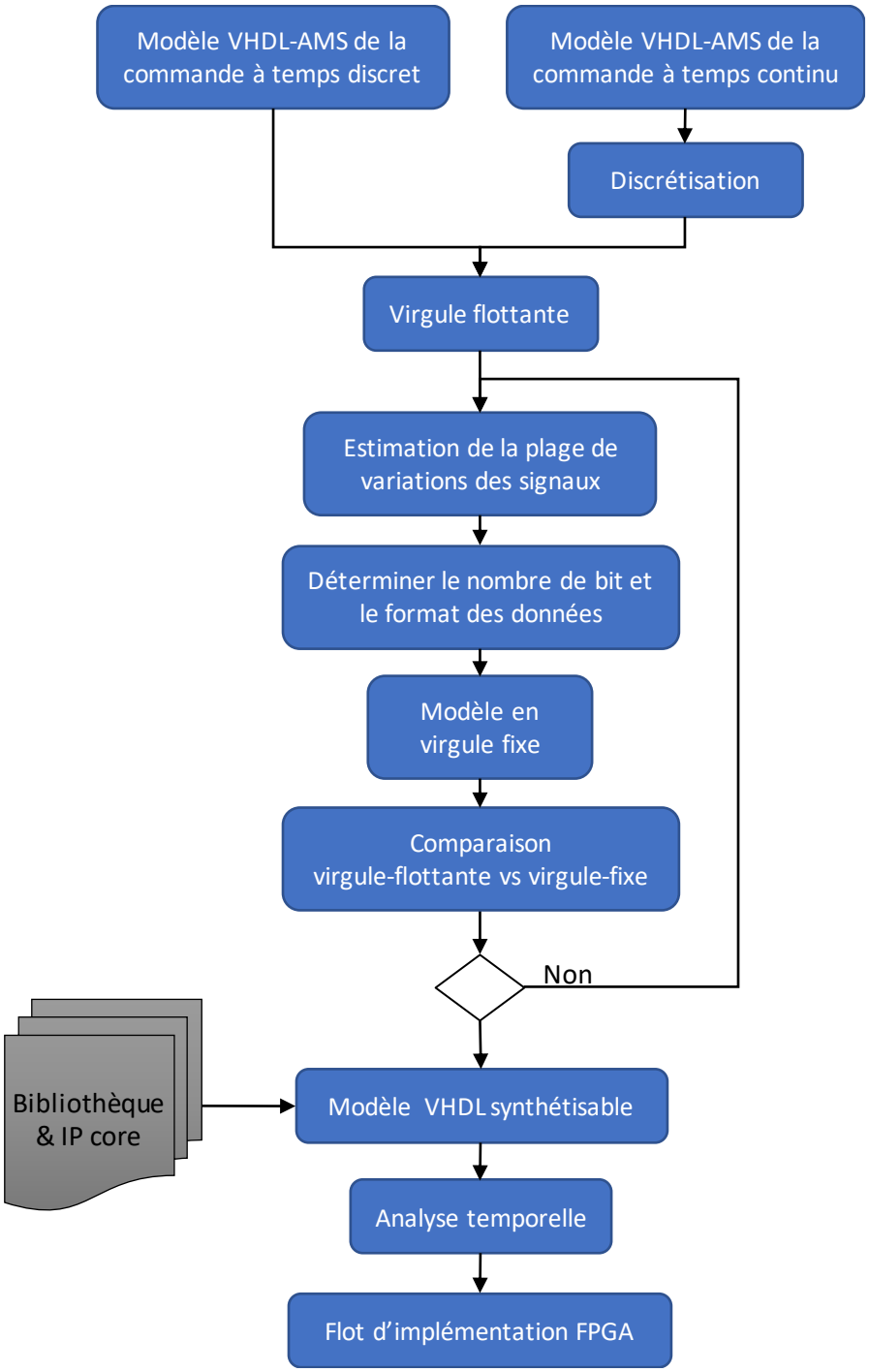


Figure 27. Processus de synthèse du modèle de commande numérique

➢ **Etape 1 : Approximation dans le domaine discret**

Le début du processus de synthèse de la commande diffère selon le type de commande utilisée, si la commande est une commande à temps continue, cela nécessite sa transformation par discrétisation en utilisant les méthodes de transposition usuelles [90], où des modèles à temps continue sont remplacer par leur équivalents à temps discret à travers des transformations numériques appropriées tels que la méthode d'approximation d'Euler ou la méthode d'approximation bilinéaire. A la fin de cette étape les modèles utilisés sont toujours décrits en VHDL-AMS, ces modèles utilisent des variables de type réel avec une représentation en virgule flottante. Aussi, les algorithmes

et les opérations de contrôle qui y sont définis sont cadencés à travers un signal d'horloge qui échantillonne et synchronise les différents éléments du contrôleur.

- **Etape 2 : Analyse et estimation des champs de variation**

Avant de passer à un format de donnée numérique, il faut analyser toutes les variables impliquées dans l'algorithme de commande afin d'estimer leurs intervalles de variation, de cette manière on peut normaliser les variables et simplifier le calcul numérique.

- **Etape 3 : Quantification et numérisation**

L'implantation efficace des algorithmes de commande dans des solutions numériques matérielles requiert l'utilisation de l'arithmétique en virgule fixe, ceci afin de satisfaire les différentes contraintes liées aux ressources matérielles visées. Le modèle ainsi développé doit être quantifié à travers le développement d'un modèle à virgule fixe de l'algorithme numérisé. Le passage en virgule fixe de l'architecture de la commande peut être une source d'erreurs et une tâche fastidieuse qui allonge nettement le temps de développement des algorithmes. C'est pourquoi une comparaison des performances entre le modèle de référence (en virgule flottante) et le modèle quantifié est indispensable pour évaluer l'impact du format de la virgule fixe choisi sur la précision des calculs et les performances de contrôle.

Dans cette étape le modèle est toujours en VHDL-AMS, mais il faut adapter son interface aux interfaces des autres éléments du système ce qui garantit l'intégrité du flux de donnée dans le système. Ensuite, il faut refaire de nouvelles simulations avec les mêmes scénarios que ceux utilisés dans les précédentes étapes afin de :

- Vérifier le bon fonctionnement du modèle numérisé,
- Vérifier si le modèle est toujours conforme aux spécifications,
- Controller la précision de la conversion du modèle.

Si les résultats ne correspondent pas aux attentes fixées, il faut choisir un nouveau format de donnée et refaire la quantification des signaux impliqués dans le modèle. Par exemple, en élargissant les plages de variation des signaux, ou en augmentant le nombre de bits dans le format binaire des données.

- **Etape 4 : Génération de code VHDL**

Jusqu'à maintenant nous avons utilisé la sémantique du langage VHDL-AMS, mais dans la suite du développement du modèle, il faut réécrire les modèles produits dans le langage VHDL et assainir le code pour qu'il soit synthétisable. Dans cette étape, nous pouvons recourir à des bibliothèques ou des IP-cores qui peuvent améliorer les performances des algorithmes de contrôle, telles que des bibliothèques VHDL pour le calcul trigonométrique ou des modules de contrôle spécifiques aux circuits FPGA visés.

D'autres simulations peuvent être effectuées pour vérifier la cohérence et l'exactitude du code. Aussi, dans certaines applications de contrôle l'analyse temporelle est nécessaire pour la validation du code VHDL retenu.

- **Etape 5 : Programmation du circuit FPGA**

Dans cette dernière étape, le flot de conception générique pour la conception FPGA est utilisé. Le résultat de ce flot de conception est un fichier Bitstream qui est destiné pour être embarqué dans le circuit FPGA. Cette étape est dédiée à l'implantation de la commande numérique sur une carte de développement intégrant un composant FPGA. Elle est principalement destinée à la vérification et à la validation de l'implantation numérique de l'algorithme de commande sur cible FPGA dans un environnement de simulation Hardware-In-the-Loop. Lors de cette étape, nous utilisons les outils de conception dédié, ces outils permettent la synthèse, le placement et routage et ensuite l'implémentation du code binaire dans le circuit cible. Aussi, des étapes de configuration d'ajustement et d'optimisation sont possibles pour ajuster le design au circuit FPGA utilisé.

6.1.5 Simulation Contrôleur HIL

La dernière phase de développement dans notre approche de conception est la simulation Hardware-In-the-Loop. A ce stade, nous devons insérer le code généré du contrôleur numérique dans une plateforme dédiée à la simulation HIL avec VHDL-AMS, cette plateforme que nous avons conçue sera détaillée dans le prochain chapitre.

Dans notre approche nous utilisons le prototypage avec simulation Hardware-In-the-Loop basé sur les composants FPGA. Cette technique permet la vérification et la validation de l'implémentation sur FPGA de la partie numérique d'un système de conversion de puissance. La simulation est exécutée en associant un logiciel de simulation VHDL-AMS qui permet de reproduire un environnement de test crédible et réaliste. Ce qui permet de passer plus facilement à la qualification du système de commande dans un contexte réel.

Les phases de développement qui interviennent par la suite dans le flot de conception que nous proposons sont les mêmes phases utilisées dans d'autres approches de conception, soit l'intégration réelle du système et le test de qualification.

7 CONCLUSION

Dans ce chapitre, nous avons d'abord justifié la nécessité grandissante d'une méthodologie de conception adaptée aux systèmes multi-technologiques, notamment les systèmes de conversion de la puissance électrique. Ensuite, nous avons présenté les différents piliers qui soutiennent une méthodologie de conception cohérente, et qui aident à concevoir un système robuste qui reproduit les exigences du client ainsi que les exigences techniques de conception.

Le premier pilier de la méthodologie proposée est le langage de description matériel VHDL-AMS, ce langage permet une grande souplesse de conception, ainsi le concepteur se permet de passer d'une conception analogique à une conception numérique de manière simple et pertinente. Ce langage présente de nombreux avantages par rapport à ces concurrents tels que le langage SystemC-AMS ou Modelica.

Nous avons ensuite proposé un nouveau flot de conception et de prototypage basé sur les composants programmables FPGA. Ce flot de conception descendant (Top-Down) permet d'utiliser

une approche basée sur le modèle, cette approche nous permet d'utiliser le langage VHDL-AMS pour la conception, la modélisation et la vérification du système, ainsi que le langage SysML pour la spécification des exigences et la conception architecturale. Les modèles sont utilisés pour la validation des exigences et pour la réalisation effective du système. Par ailleurs, les modèles VHDL-AMS comportementaux sont traduits vers des descriptions VHDL synthétisables qui permettent de réaliser des simulations Hardware-In-the-Loop basées sur les composants reconfigurables FPGA.

Les avantages majeurs de l'approche proposée résident dans la facilité de conception apportée, la rapidité du prototypage et le coût faible de ces outils de développement. Toutefois, pour la mise en œuvre de cette approche une plateforme de simulation HIL dédiée doit être utilisée, la conception et l'utilisation de cette plateforme sont présentées dans les prochains chapitres.

Chapitre 4

Plateforme proposée pour la simulation C-HIL

1 INTRODUCTION

La méthodologie que nous proposons dans ce travail est basée sur les langages de description matérielle VHDL et VHDL-AMS, le premier est utilisé pour la description de système numérique et le second pour la modélisation de systèmes multidisciplinaires mixte analogique/numérique. Conformément aux objectifs de notre démarche, le passage de la simulation à la validation effective du système à concevoir passe par l'utilisation des techniques de simulation Hardware-In-the-Loop.

Dans ce quatrième chapitre nous allons présenter une plateforme de simulation Hardware-In-the-Loop qui est dédiée à la vérification et la validation de systèmes multi-technologiques basés sur les composants FPGA. La plateforme proposée, permet de faire le lien entre la modélisation, la simulation et la validation des algorithmes de commande numérique dans un environnement contrôlé, ce qui permet la configuration et la mise au point des différents paramètres du système à concevoir.

Dans ce qui suit, nous allons détailler le développement de la plateforme proposée en établissant les choix de conception qui permettent de valider les objectifs visés. Dans le même contexte, nous allons présenter les différents éléments logiciels et matériels qui sont nécessaires à la mise en œuvre de cette plateforme, ainsi qu'à son fonctionnement.

2 PRESENTATION DE LA PLATEFORME DE SIMULATION HIL-AMS

2.1 PRINCIPE DE FONCTIONNEMENT

Le but de notre démarche consiste à accomplir des procédures de simulation Hardware-In-the-Loop multi-domaine à base de circuit FPGA, à travers l'utilisation du langage de description matérielle VHDL-AMS. Afin de pouvoir atteindre ce but, nous devons réunir dans une même plateforme logiciel/matériel le prototypage FPGA et la simulation VHDL-AMS.

La Figure 28 met en évidence le principe de base de cette plateforme. A travers des interfaces de communication logicielles et matérielles, les données sont transférées de part et d'autre de la plateforme pour permettre au processus de simulation de s'exécuter. La structure de cette plateforme est organisée autour de trois composants : un ordinateur hôte avec le simulateur VHDL-AMS, une carte de développement FPGA et une application pour l'interface de communication logicielle/matérielle.

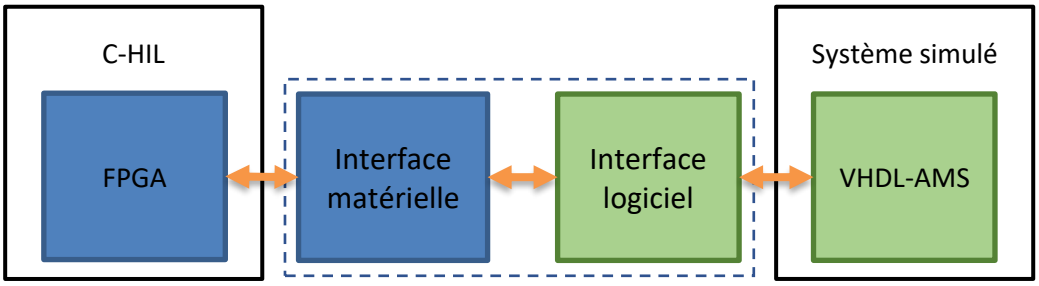


Figure 28. Stratégie de simulation HIL à base de circuits FPGA et du langage VHDL-AMS

L'outil que nous proposons cible un grand nombre de domaines d'application allant des systèmes sur puce (SOC System On Chip) aux systèmes mécatroniques. Les applications concernées sont des applications qui impliquent une ou plusieurs disciplines d'ingénierie, en plus d'un sous-système de commande numérique. La structure du système est modélisée en VHDL-AMS et le sous-système de commande est intégré sur un composant FPGA.

Cet outil peut être utilisé dans différentes activités de conception, comme par exemple :

- La validation de l'architecture physique du système,
- La validation de nouveaux algorithmes de commande,
- La démonstration de la faisabilité de la commande,
- Le réglage et la mise au point de la commande au cours de la simulation,
- Le débogage in-situ de d'implémentation sur circuit FPGA.

2.2 OBJECTIFS ET CARACTERISTIQUES

L'objectif de cet outil est de proposer un environnement de simulation Hardware-In-the-Loop à base de circuits FPGA et qui utilise la simulation VHDL-AMS, cet outil doit être performant, simple d'utilisation et fiable. Afin de développer la plateforme de simulation proposée, nous avons défini un certain nombre de spécifications qui aideront à construire l'architecture de cette plateforme, ainsi que les différentes fonctionnalités supportées. De ce fait, cette plateforme doit être conforme aux points suivants :

- Pouvoir utiliser un logiciel de simulation multi-langage et multiplateforme.
- Le logiciel doit être compatible avec le langage VHDL et le langage VHDL-AMS avec une prise en charge maximal des deux standards (IEEE 1076 et IEEE 1076.1).
- Le logiciel doit avoir une interface de communication logicielle pour permettre le contrôle de la simulation par des applications externes.
- Disposer d'une carte de développement FPGA d'une capacité suffisante permettant l'implantation des composants nécessaires pour la réalisation d'une simulation Hardware-In-the-Loop.
- La carte de développement FPGA doit disposer d'un moyen de communication rapide pour dialoguer avec l'ordinateur hôte (USB, Série, Ethernet, PCI Express …).
- La carte FPGA doit être suffisamment paramétrable pour ajuster son fonctionnement aux paramètres du logiciel de simulation.
- Le choix des éléments de la plateforme doit toujours prendre en considération le coût et les performances.

2.3 CHOIX DU LOGICIEL DE SIMULATION

Contrairement au langage VHDL qui est largement utilisé dans l'industrie, le langage VHDL-AMS, pareillement que les autres langages de modélisation AMS (Analog and Mixed Signal), reste peu utilisé dans l'industrie qui a adopté des outils de développement intégré plus accompli. Par conséquent, peu d'entreprises ont investi dans le développement de logiciels de simulation qui supportent ce type de

langage. Les principaux outils de simulation qui prennent en charge le langage VHDL-AMS sont : SIMPLORER™, Portunus™, SMASH™ et AdvanceMS™. Tous ces logiciels n'implémentent pas complètement la norme IEEE 1076.1 du langage VHDL-AMS, mais chacun s'en approche plus ou moins.

Actuellement, SMASH est le simulateur supportant le langage VHDL-AMS le plus complet sur le marché. Il s'approche d'une couverture totale de la norme, assez loin devant les autres simulateurs [85], [91]. De ce fait, SMASH de Dolphin Integration est le simulateur que nous avons choisi pour la mise en œuvre de la plateforme proposée, vu sa capacité de prendre en charge la simulation multi-domaine et la modélisation multi-langage. Une autre qualité de ce logiciel est qu'il intègre une interface de programmation applicative (API) qui permet de personnaliser et de contrôler ces fonctionnalités. Avec cette API, SMASH peut être considéré comme une plateforme ouverte, dans laquelle les développeurs de logiciels peuvent ajouter de nouvelles routines et de nouvelles fonctionnalités. Dans ce travail, ce dernier aspect est l'atout le plus intéressant de SMASH, il nous permet de concevoir la plateforme recherchée.

3 LE LOGICIEL DE SIMULATION SMASH

SMASH™ est un simulateur à signaux mixtes qui permet, entre autres, le développement et la vérification des circuits intégrés, des systèmes électroniques et de manière générale les systèmes multidisciplinaires. Grâce à ses capacités multi-domaines et son support multi-langages, il permet d'améliorer et de simplifier la conception interdisciplinaire avec un temps de développement rapide et une productivité accrue. Dans sa version de base il ne dispose pas d'éditeur graphique (Figure 29).

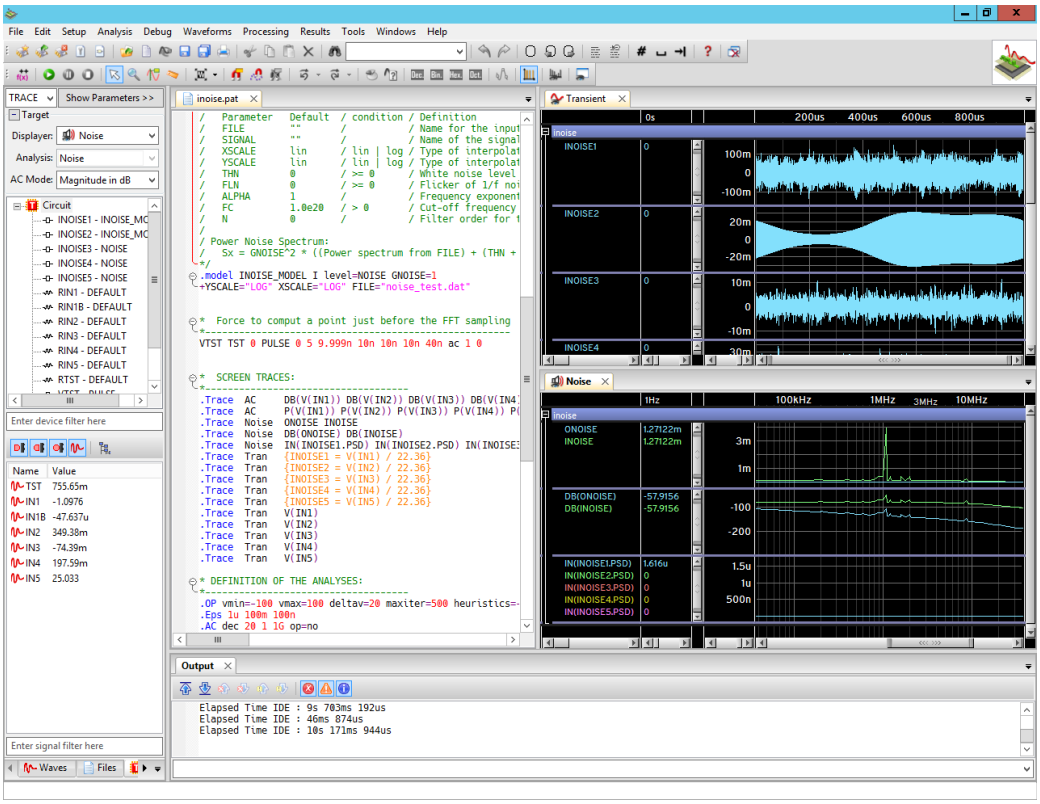


Figure 29. Environnement de développement SMASH

Une des caractéristiques les plus intéressante de l'outil SMASH est sa capacité multi-langage. Il est compatible avec SPICE, Verilog, Verilog-AMS, VHDL, TML, C et VHDL-AMS et permet aux utilisateurs de créer des modèles en mélangeant les différents langages et les niveaux de description. Aussi, il possède une API de programmation qui permet de l'interfacer avec d'autres outils de conception. Cette capacité de programmation est très importante pour permettre à ce logiciel de profiter d'autres solutions pour la simulation et pour la gestion de la simulation qui n'y sont pas intégrées nativement.

3.1 FLOT DE SIMULATION

SMASH utilise une approche HDL compilée par rapport à une approche interprétée. Cela signifie qu'une description HDL sera toujours compilée avant d'être simulée. Ci-dessous sont listées les différentes étapes d'une simulation HDL avec le logiciel SMASH, qui sont par ailleurs représenté sur la Figure 30 :

→ Compilation et catalogage :
Les modèles HDL sont compilés dans des bibliothèques partagées binaires et ensuite ajoutés au catalogue principal des modèles.

→ Élaboration :
Création et association de chaque instance de modèle de la conception en commençant par le niveau supérieur.

→ Simulation :
Exécution du modèle élaboré.

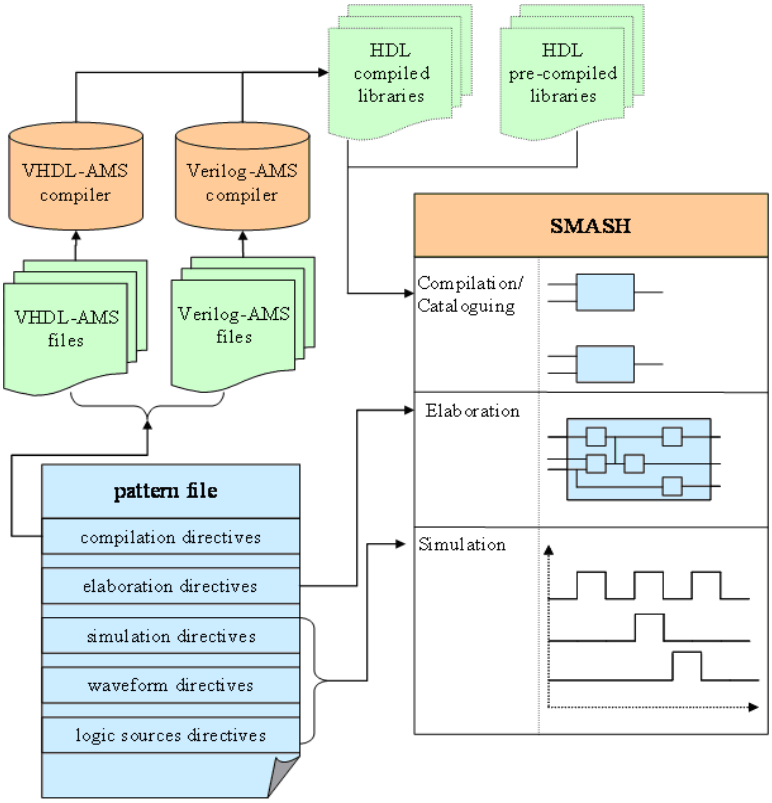


Figure 30. Flot de simulation de SMASH [92]

3.1.1 Compilation

La phase de compilation consiste à compiler les modèles HDL, qu'ils soient Verilog, VHDL, Verilog-AMS ou VHDL-AMS, quel que soit leur niveau d'abstraction, dans des modèles binaires prêts à être exécutés par le simulateur. Les avantages de cette méthode de compilation sont les suivants :

- Chargement plus rapide des modèles une fois la compilation initiale effectuée car il n'est pas nécessaire de les analyser et de les vérifier à nouveau.
- Assurer un certain niveau de protection des modèles.

3.1.2 Elaboration

La phase d'élaboration consiste à créer une représentation du modèle en mémoire pour la simulation. Cela implique l'allocation et l'association de chaque instance de modèle de la conception, en commençant par le niveau supérieur.

3.1.3 Catalogage

Avant de commencer l'élaboration d'une conception, les modèles doivent être compilés dans des bibliothèques partagées et cataloguées. Le catalogue stocke tous les modèles disponibles pour une éventuelle instanciation.

3.1.4 La simulation

La phase de simulation consiste à exécuter la représentation du modèle en mémoire pour l'analyse choisie. Cela peut impliquer une simple propagation des changements de valeur du signal dans la hiérarchie et les connexions, ainsi qu'une exécution plus complexe des descriptions comportementales.

3.2 SYNCHRONISATION DE LA SIMULATION A SIGNAUX MIXTES

L'une des principales fonctions d'un simulateur à signaux mixtes est la synchronisation de deux algorithmes distincts afin d'échanger des informations sans erreur, sans perte de précision ou en temps de calcul et de communication. Le simulateur idéal de signaux mixtes déplacerait les algorithmes analogiques et logiques indépendamment dans le temps, en interagissant uniquement pour l'échange de données entre les domaines analogique et logique.

Certaines des méthodes les plus utilisées pour la synchronisation des algorithmes analogiques et logiques sont le "Lock-step" et le "Roll-back" (voir Figure 31) [93]. SMASH implémente les deux méthodes et permet à l'utilisateur de sélectionner la méthode appropriée en fonction du modèle simulé.

3.2.1 Méthode du pas bloqué (Lock-step)

Cette technique impose un pas identique pour la simulation logique et la simulation analogique égale au plus petit pas requis, qui est imposé par le simulateur analogique. La synchronisation est ainsi toujours assurée, au prix cependant de ne plus profiter pleinement de la latence du côté logique. De plus, la synchronisation fréquente n'est pas toujours nécessaire, car à chaque point de calcul il y'a résolution du système d'équations du modèle, ce qui augmente le temps de simulation global.

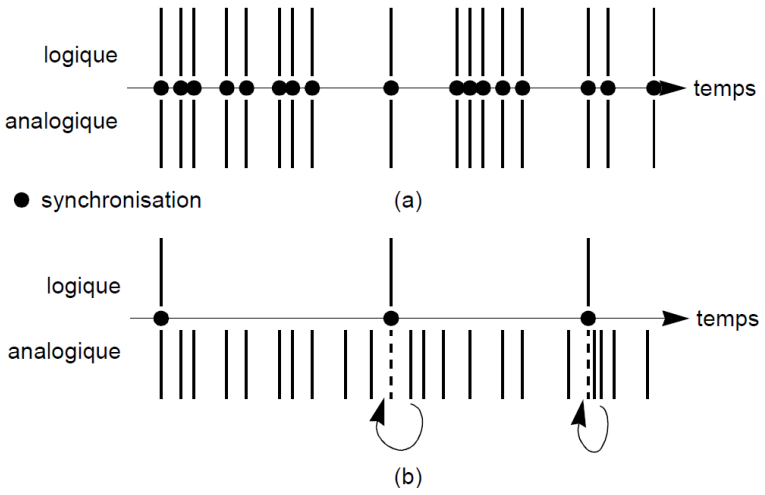


Figure 31. Méthodes de synchronisation du temps : (a) pas bloqué, (b) avec retour en arrière

3.2.2 Méthode avec retour en arrière (roll-back)

Cette technique permet à chaque simulateur d'utiliser indépendamment son propre pas temporel jusqu'à l'occurrence d'un événement qui nécessite une synchronisation, afin d'échanger des données entre les deux moteurs de simulation analogique et numérique. Les événements du côté logique sont très facilement identifiables alors que ceux du côté analogique dépendent de la précision des calculs. Pour maintenir la justesse de la simulation, le simulateur analogique en avance dans le temps doit revenir en arrière pour retrouver la synchronisation. Il faut noter que ce retour en arrière ne requiert pas forcément un nouveau calcul de l'état de la partie analogique, une simple interpolation peut s'avérer suffisante.

3.3 L'API DE SMASH

En informatique, API est l'acronyme d'**A**pplication **P**rogramming **I**nterface, que l'on traduit en français par interface de programmation applicative. L'API peut être résumée à une solution informatique qui permet à des applications de communiquer entre elles et de s'échanger mutuellement des services ou des données. Il s'agit en réalité d'un ensemble de fonctions, de commandes, de protocoles et d'objet que le programmeur utilise pour la création d'une application logicielle ou pour interagir avec un système externe. L'API facilite via un langage de programmation, l'accès aux services d'une application. L'API fournit aux développeurs des commandes standard pour effectuer des opérations courantes afin qu'ils n'aient pas à écrire le code à partir de zéro [94].

L'interface de programmation applicative API de SMASH offre un moyen de personnaliser le logiciel SMASH, soit avec des extensions, des plugins ou des « hooks », ces applications logicielles sont écrites avec des langages de programmation standard. Avec cette API, SMASH devient une plate-forme ouverte, où les utilisateurs peuvent développer des fonctionnalités supplémentaires et les brancher au produit principal. L'API est fournie sous trois formes ; une API C/C++, une API Tcl et une API ActiveX. Il existe donc trois manières de communiquer avec SMASH :

- La première consiste à utiliser l'API C. Dans ce cas, les add-ons sont des fichiers bibliothèques (DLL) qui doivent être écrites en C ou C ++. Les add-ons appellent simplement les fonctions exportées par SMASH. SMASH permet alors de créer et de compiler ces DLL.
- La seconde consiste à utiliser la liaison Tcl (Tool Command Language) de l'API SMASH. C'est le moyen le plus simple mais il existe un certain nombre de limitations (toutes les fonctions de l'API ne sont pas liées au Tcl). L'approche Tcl est recommandée à chaque fois que les performances ne sont pas une priorité.
- La troisième voie consiste à utiliser l'API ActiveX. Dans ce cas, les add-ons sont des exécutables. Ils peuvent être écrits en langage Visual Basic, en langage Visual C ++ ou en langage Java.

L'API SMASH est un ensemble de fonctions exportées, qui permet de développer des applications qui agissent comme des "add-ons" au-dessus de SMASH lui-même. À l'aide de l'API, un processus défini par l'utilisateur peut appeler la base de données SMASH pour extraire des informations, mais également utiliser des ressources SMASH (diverses routines d'utilitaires, fenêtres, messages, invités, etc.).

4 MISE EN ŒUVRE DE LA PLATEFORME DE SIMULATION

Quant à la mise en œuvre de la plateforme de simulation HIL-AMS, nous proposons de relier le logiciel de simulation SMASH avec une carte de développement FPGA, cette liaison sera assurée grâce à l'utilisation d'une interface logicielle/matérielle qui permettra de piloter la simulation de part et d'autre de la plateforme. Pour ce faire, plusieurs solutions sont proposées dans l'API de SMASH (voir section 3.3 ci-dessus), toutefois, après différents tests de performance, nous avons choisi de s'appuyer dans notre travail sur une procédure qui utilise le logiciel SMASH comme un outil de cosimulation.

Dans le logiciel SMASH, l'API de cosimulation est une procédure qui permet d'intégrer de nouvelles fonctionnalités au logiciel SMASH. Cette méthode fait intervenir des outils externes afin d'améliorer et d'affiner les résultats de simulation, de réduire le temps de simulation, de vérifier les éléments fonctionnels d'un système avant le prototypage réel et ainsi de valider plus vite le système visé.

La Figure 32 montre les différents liens qui permettent de réaliser cette procédure de simulation. Dans le cadre de nos objectifs, nous devons concevoir un environnement de simulation qui permet d'établir les échanges de données et de commandes nécessaires entre le logiciel SMASH et une carte de développement (dans notre cas une carte de développement à base de composants FPGA), ceci à travers l'utilisation d'une application logicielle dédiée. Cette application doit garantir l'intégrité de l'échange de données en synchronisant l'exécution des modèles dans les différentes parties associées à cet environnement.

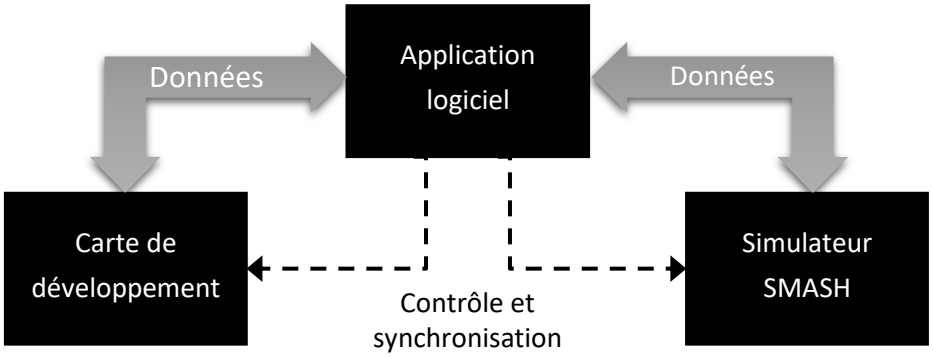


Figure 32. Schéma de principe de la plateforme de simulation HIL

Afin de mettre en œuvre ces liens de communication, nous devons disposer d'une plateforme logicielle/matériel qui permet de rassembler les éléments illustrés dans Figure 33, chaque élément doit configurer et adapter aux autres éléments adjacents. Ces éléments sont définis ci-dessous :

- Ordinateur hôte : c'est la machine prévue pour l'exécution du logiciel SMASH et de l'application développée.
- Modèle VHDL-AMS : c'est l'ensemble des modèles qui représente la partie analogique ou analogue du système considéré.
- Application C++ : c'est l'application logicielle qui doit gérer la simulation HIL en contrôlant le simulateur SMASH, elle est programmée en C++ pour garantir les meilleures performances.
- Interface de communication : c'est l'interface qui permet l'échange de données entre l'application logicielle et la carte de développement FPGA, plusieurs possibilités sont envisageables et qui dépendent essentiellement de la configuration de la carte FPGA.
- Modèle sous-test : c'est le modèle de la commande numérique du système que nous voulons vérifier dans la carte FPGA.

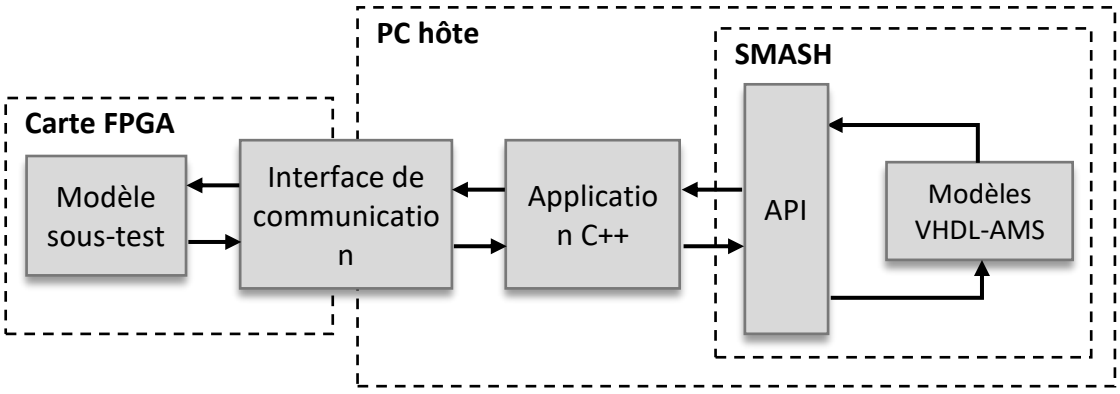


Figure 33. Eléments constituants la plateforme de simulation FPGA & VHDL-AMS

Avant de détailler les éléments qui constituent la plateforme proposée, nous devons faire des choix en ce qui concerne les méthodes de communication et de synchronisation qui conviennent le plus aux besoins de notre conception et aux éléments méthodologiques que nous nous sommes déjà imposés, à savoir l'utilisation :

- ✓ Du langage VHDL-AMS,
- ✓ Du logiciel SMASH,
- ✓ De la procédure API de cosimulation,
- ✓ D'une carte de développement FPGA.

Dans la suite de cette section, nous détaillerons les méthodes de communication et de synchronisation que nous avons choisies pour avoir le plus de flexibilité et le maximum de performance.

4.1 L'INTERFACE DE COMMUNICATION

L'interface de communication est un élément essentiel de la plateforme proposée, elle permet de réaliser le transfert de données entre la carte de développement FPGA et l'ordinateur hôte. Une interface de communication est composée d'une interface physique et d'un protocole de communication.

Le choix du type de communication est important pour la mise en œuvre de la plateforme, mais plusieurs critères sont à prendre en considération pour sélectionner la meilleure solution, parmi ces critères on peut citer :

- ➢ La bande passante et la vitesse de communication,
- ➢ Les latences matérielles et logicielles,
- ➢ La disponibilité de l'interface dans les cartes de développement FPGA,
- ➢ Le volume des ressources utilisé dans le circuit FPGA,
- ➢ Le coût.

En prenant en considération ces critères, nous pouvons sélectionner un des moyen de communication parmi les plus utilisées dans le domaine de la simulation HIL, dont on peut citer :

- PCI Express : la plus rapide des solutions existantes, elle permet d'atteindre des débits de plusieurs GO/S. Elle est disponible seulement sur les cartes de développement FPGA haute de gamme.

- USB : Elle permet des débits de transfert relativement corrects, mais la plupart des cartes FPGA l'utilise pour la programmation et le débogage, nous ne pouvons l'utilisée en même temps pour le transfert de données. Aussi, l'intégration de cette interface sur le composant FPGA requiert beaucoup de ressources matérielles.

- Réseau LAN : Le débit entre deux terminaux peut atteindre quelques dizaines de MO/S. c'est une solution qui garantit un rapport coût-performance appréciable.

Pour les besoins de notre application, nous avons choisi la solution du réseau LAN qui est la plus adaptée pour notre application. Le débit est suffisant pour faire de la simulation HIL et la plupart des cartes de développement FPGA y sont équipées. Cette solution rend notre implémentation compacte, portable et à faible coût.

Interface Ethernet

Le réseau Ethernet est une technologie largement utilisée dans la communication, l'instrumentation et les systèmes embarqués. Défini en tant que norme IEEE 802.3, il offre des configurations de réseau prenant en charge des débits de transfert de données théoriques de 10Mbit/s (10BASE-T), 100Mbit/s (100BASE-T) et 1Gbit/s (1000BASE-T), le type de réseau le plus utilisé et le réseau 100BASE-T. Dans la technique de simulation HIL, il permet le transfert de données entre les différents blocs matériels des plateformes de simulation. Cependant, pour des pas de simulation très court d'autres méthodes sont utilisées.

Au niveau matériel, la mise en place d'une connexion Ethernet demande plusieurs composants, qui prennent en charge la couche physique et de liaison du modèle OSI (Open Systems Interconnection) [95] :

- Un connecteur pour pouvoir connecter un câble Ethernet à la carte (e.g. RJ45) ;
- Un transformateur Ethernet qui permet d'isoler électriquement l'interface du réseau tout en assurant le passage des données ;
- Un transceiver Ethernet qui gère la couche physique du réseau, c'est-à-dire la génération et la réception des signaux analogiques sur le support physique ;
- Un composant appelé MAC (Media Access Control) permet de gèrer la couche de liaison, c'est-à-dire la gestion des trames Ethernet reçues et émises sur le réseau.

La plupart des plateformes FPGA intègre un connecteur RJ45, un transformateur Ethernet ainsi qu'un transceiver Ethernet discret. Si la plateforme comporte un processeur discret ou embarqué dans le FPGA, un module MAC Ethernet est déjà intégré comme périphérique avec le processeur et connectée au transceiver.

Ce module permet l'envoi et la réception de paquets Ethernet bruts ; tout ce qui relève de l'insertion et du filtrage d'adresse MAC, de l'insertion et de la vérification de checksum doit être pris en charge de manière logicielle par le pilote qui permet à la pile IP d'accéder au module MAC. Ce module comporte une interface MII pour se connecter aux transceivers externes 10 ou 100 Mbps ayant une interface MII, ou une interface RMII via un adaptateur RMII/MII ; cependant les transceivers gigabit Ethernet (ayant une interface GMII ou RGMII) ne peuvent pas être utilisés avec ce module.

Selon le modèle OSI, différents supports et protocoles de communication sont possible, nous avons choisi un protocole UDP/IP (**U**ser **D**atagram **P**rotocol / **I**nternet **P**rotocol), c'est un protocole de base utilisé dans la transmission sans connexion. Il est utilisé lorsque les performances de fonctionnement élevées sont nécessaires, quelles que soient les autres restrictions de communication.

Caractéristiques du protocole UDP

Les caractéristiques du protocole UDP qui justifient son utilisation dans la conception de la plateforme de simulation HIL que nous proposons sont décrites ci-dessous :

- Transmission très simple entre deux entités,

- L'UDP est une variante plus simple du protocole TCP (Transmission Control Protocol) qui permet, par sa simplicité, un gain de temps et une augmentation du débit par rapport à ce dernier,
- Il fonction en mode non-connecté, cela veut dire qu'il n'y a pas de vérification si les données envoyées sont bien arrivées ou non à destination,
- L'utilisation du protocole UDP permet de faciliter le débogage,
- Ce protocole permet de transmettre des données dans les deux sens en même temps.

Structure d'un datagramme

Un paquet UDP est conçu pour être encapsulé dans un datagramme IP et permettre un échange de données entre deux terminaux, sans échange de donnée pour une configuration initiale. Il comporte un en-tête suivi des données proprement dites à transporter.

Tableau 5. Structure d'un datagramme UDP

+	Bits 0 - 7	8 - 15	16 - 23	24 - 31
-96	Adresse Source			
-64	Adresse Destination			
-32	Zéros	Protocole	Taille UDP	
0	Port Source		Port Destination	
32	Longueur		Somme de contrôle	
64	Données			

De 0 à 64, la trame UDP est définie où :

- Le port Source : indique depuis quel port le paquet a été envoyé.
- Le port de Destination : indique à quel port le paquet doit être envoyé.
- La longueur : indique la longueur totale (exprimée en octets) du segment UDP (en-tête et données). La longueur minimale est donc de 8 octets (taille de l'en-tête).
- La somme de contrôle (CheckSum) : celle-ci permet de s'assurer de l'intégrité du paquet reçu quand elle est différente de zéro.

4.2 ADAPTATION DE MODELE ET AFFINAGE

Outre le mécanisme de synchronisation entre le moteur de simulateur VHDL-AMS et les événements discrets provenant du circuit FPGA, un autre aspect de la simulation HIL est la transformation du format de données.

Une étape essentielle dans le processus de conception est la conversion de la partie commande d'un modèle à temps continu vers un modèle numérique. Cette transformation de modèle ne doit pas affecter le reste des modèles du système. Pour cette raison nous devons ajouter des interfaces qui adaptent le format et le type des données entre les modèles concernés. Afin d'y parvenir, trois aspects sont à prendre en considération :

- Des modèles de convertisseurs analogiques-numériques (CAN) doivent être ajoutés pour assurer l'interfaçage entre les éléments analogiques et les éléments numériques du système. Si ces convertisseurs ne correspondent pas à la spécification d'exigence initialement établie, nous pouvons ajouter des fonctions ou des attributs du langage VHDL-AMS qui permettent de transformer des variables analogiques (quantité) en variables discrètes (signal).
- Quel que soit le mode d'acquisition du signal numérique, il est important de définir un format de représentation des données manipulées dans la description VHDL. Le format des données dépend essentiellement du circuit FPGA visé, des performances et de la précision ; il est donc déterminé en conciliant tous ces aspects dans la conception de la solution numérique.
- En ce qui concerne la simulation HIL, nous devons intégrer des routines et des instructions VHDL-AMS pour autoriser la capture des valeurs des signaux à partir de l'API de SMASH. Aussi, l'application logicielle aura pour rôle de convertir le format de données dans les deux sens de la communication impliquant la carte de développement FPGA et le simulateur SMASH.

5 APPLICATION LOGICIELLE

L'application logicielle a plusieurs missions qui correspondent aux tâches qui permettront l'utilisation et l'exploitation de la plateforme. Elles regroupent les fonctionnalités suivantes :

- Contrôle total du logiciel de simulation SMASH via son API,
- Communication bidirectionnelle pour envoyer les commandes et les données vers SMASH,
- Communication bidirectionnelle pour envoyer les commandes et les données vers la carte de développement FPGA,
- Récupération des données à partir du flux Ethernet UDP,
- Empaquetage des données dans le flux Ethernet UDP,
- Adaptation des formats de données.

Cette application logicielle a donc été conçue et évaluée parallèlement avec la réalisation de la partie de la plateforme sur le circuit FPGA, du fait de l'enchevêtrement des fonctionnalités de part et d'autre de la plateforme. Le langage choisi pour son implémentation est le langage C++, ce choix a été motivé pour deux raisons ; la première est la puissance des applications C++ et la deuxième est due au fait que le logiciel SMASH fournit les fonctions de son API ainsi que des bibliothèques nécessaires dans le même langage.

Le schéma présenté dans la Figure 34 décrit l'architecture de l'application et met en évidence le flux de données correspondant, ce qui permet le contrôle pour la communication avec le logiciel SMASH et l'interface du réseau Ethernet. Pour que cette application soit en mesure de réaliser une simulation HIL, elle doit exécuter les trois principales tâches décrites ci-dessous :

1. La première est que l'application doit exécuter et contrôler des échanges de données entre l'application elle-même et le modèle VHDL-AMS simulé. Cet échange de données concerne

la mise à jour des valeurs des signaux entrants et la capture des valeurs des signaux sortants. Cette tâche communique directement avec le moteur de simulation à temps discret de SMASH. Pour ce faire, cette fonction doit :

- Créer un pointeur qui gère les signaux VHDL-AMS pour permettre à l'application de capturer leurs valeurs pendant la durée de la simulation.
- Instancier les pilotes pour tous les signaux qu'on souhaite mettre à jour, afin de permettre à l'application de programmer leurs nouvelles valeurs en temps voulu pendant la simulation.

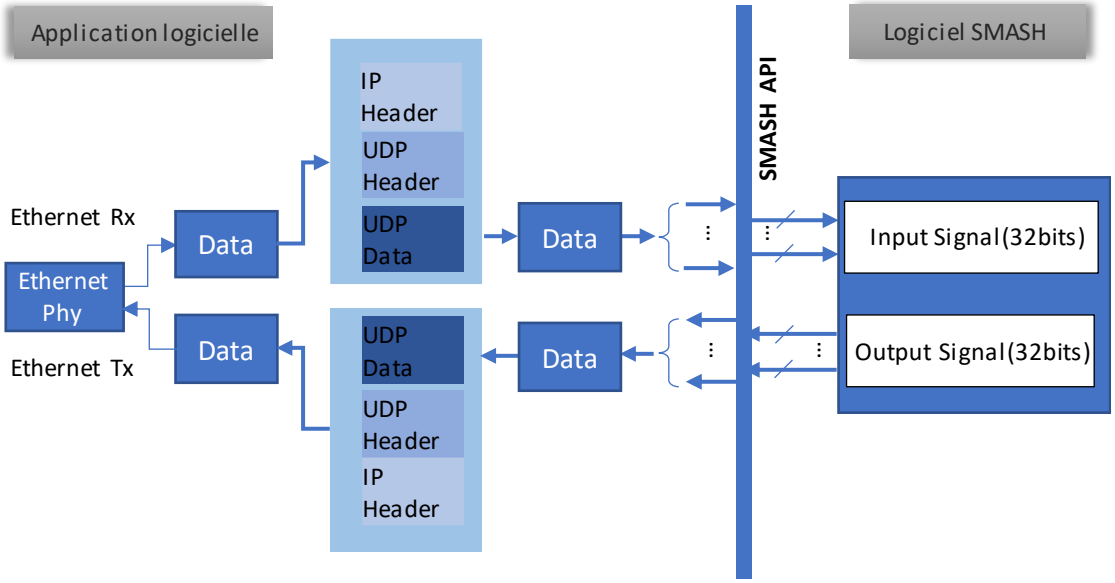


Figure 34. Flux de données dans l'application logicielle

2. La deuxième tâche est l'établissement de la communication avec la carte de développement FPGA. Cette tâche permet le transfert des données entre l'ordinateur hôte (qui exécute SMASH) et le circuit FPGA. Lorsque l'application collecte les valeurs de tous les signaux ciblés, les données sont ensuite assemblées dans un datagramme, puis transférées via la liaison Ethernet dans un paquet UDP vers la carte FPGA. Ensuite, l'application est mise en mode d'écoute et attend une réponse du FPGA. Le datagramme reçu est ensuite extrait du paquet UDP, ceci afin d'extraire les données nécessaires qui seront imposées comme nouvelles valeurs aux signaux d'entrée. Pendant ce temps, l'exécution de la simulation est suspendue jusqu'à l'arrivée de nouvelles valeurs.

3. Les tâches précitées ne peuvent être effectuées sans synchronisation, la fonction la plus critique de cette application est de pouvoir gérer et synchroniser la réception et la transmission des données entre le simulateur et la carte FPGA. Nous utilisons la synchronisation inhérente du logiciel qui gère le transfert des données entre les moteurs de simulation analogique et numérique.

5.1 ORGANIGRAMME FONCTIONNEL DE L'APPLICATION

L'organigramme de la Figure 35 montre les étapes clés de l'exécution du programme de l'application qui permet le transfert des données du logiciel SMASH vers la carte FPGA et vice versa.

Après initialisation de la carte FPGA et du simulateur SMASH, l'application prend le contrôle de ce dernier, ainsi elle va charger le projet (Modèle VHDL-AMS principal) qu'on veut simuler, elle recherche les signaux nécessaires pour exécuter la simulation HIL. Ensuite, l'application récupère le ***handle*** de chaque signal (fonction de l'API qui permet de s'accrocher à un signal) qui permet par la suite de contrôler le ***driver*** du signal.

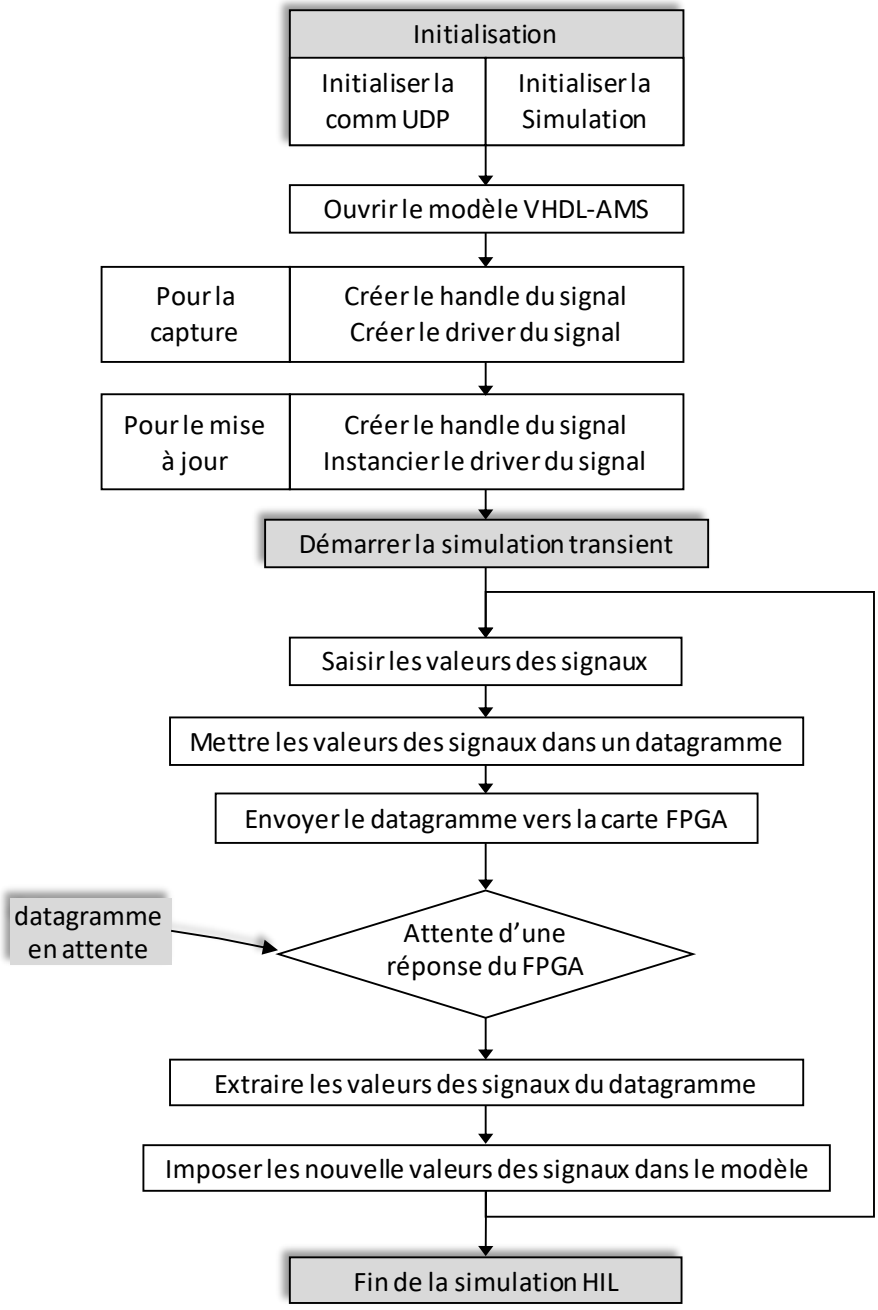


Figure 35. Organigramme fonctionnel de l'application

Ci-dessous les principales directives du programme de l'application qui permettent de lire et de modifier les valeurs des signaux ciblent à chaque pas de simulation :

1) Le programme récupère les valeurs des signaux que nous voulons mesurer à partir du modèle VHDL-AMS. Afin de capturer ces données, une fonction API SMASH est utilisée comme ci-dessous :

```
// S'accrocher au handle de signal
void * handle1 = SMASH_SignalGetHandle ("Signal_name" ) ;
// Obtenir la valeur du signal
String Value = SMASH_SignalGetValueAsString ( handle1 ) ;
```

2) Ensuite, les données sont encapsulées dans un datagramme et envoyées via la liaison Ethernet vers la carte de développement FPGA.

3) Par la suite, l'application est mise en mode d'écoute et attend une réponse de la carte FPGA. Une fois reçu, le datagramme est décompressé pour extraire les données générées par le contrôleur numérique qui est intégré dans le circuit FPGA.

4) Les données reçues sont les nouvelles valeurs des signaux de contrôle. Pour rendre cela effectif, le code C++ suivant est utilisé :

```
// S'accrocher le handle de signal
void * handle2 = SMASH_SignalGetHandle (" Signal name " ) ;
// Instancier le driver du signal
void * driver 2 = SMASH_SignalInstantiateDriver ( handle2 , 1 ) ;
// Programmer une nouvelle valeur entière de 32 bits sur le driver du signal pour le
   prochain pas de calcul
int data;
SMASH_SignalScheduleInt32 ( driver1 , data , 1 ) ;
```

5) Toutes les étapes précédentes sont effectuées à chaque pas de simulation en utilisant une fonction spécifique de l'API SMASH « ***SMASH_HookAddCallback ()*** ». Cette fonction permet de rappeler l'exécution du programme à chaque étape de la simulation, ce qui permet à l'application développée de se synchroniser et de gérer l'échange de données entre le moteur de simulation et la carte FPGA.

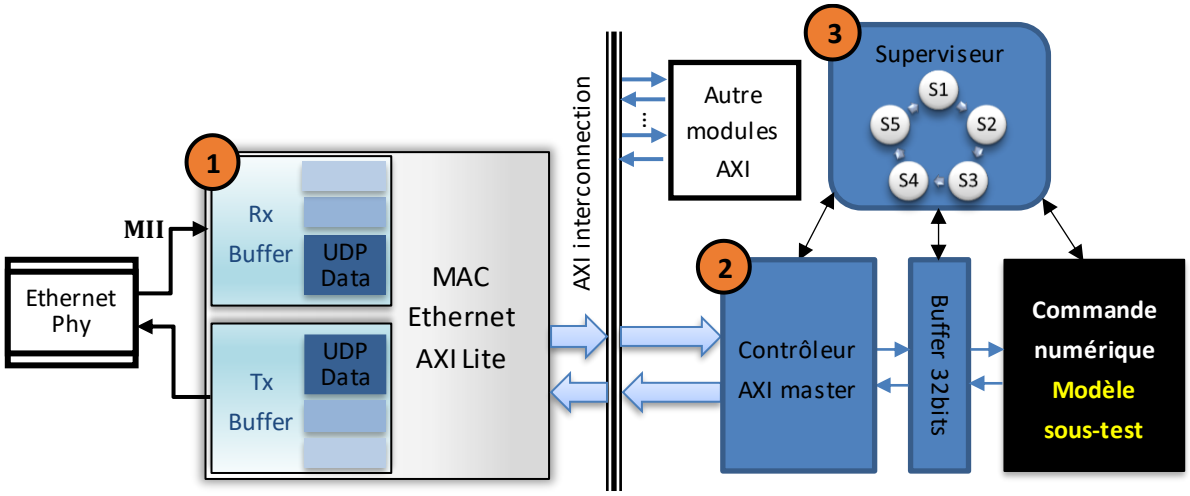
6 APPLICATION MATERIELLE COTE FPGA

6.1 PRESENTATION

L'approche présentée dans ce travail utilise la simulation Hardware-In-the-Loop avec un circuit FPGA comme matériel de contrôle dans la boucle. Comme indiqué dans les sections précédentes, notre approche doit satisfaire un certain nombre d'exigences dont un coût minimum. De ce fait, l'implémentation que nous proposons côté FPGA doit utiliser un minimum de ressources avec des

performances suffisantes pour valider la plateforme de simulation. Une carte de développement FPGA d'entrée de gamme est alors employée, elle comprend un circuit FPGA de faible densité avec une interface Ethernet.

L'objectif de cette application est de transférer les données entre le modèle de la commande numérique (implanté dans le circuit FPGA) et le simulateur SMASH. Nous devons donc passer par différentes couches du réseau pour extraire les données reçues ou pour insérer des données de réponse dans un paquet Ethernet. Pour pouvoir exécuter les différentes fonctionnalités nécessaires à accomplir cette mission, l'architecture de l'application FPGA doit comprendre une configuration minimale, les différents éléments de cette architecture sont illustrés dans la Figure 36.



**. Ethernet Phy est un composant électronique requis pour implémenter la couche physique d'un modèle OSI*
***. Interface MII introduite par la norme 100Base-T permet d'assurer la liaison entre la couche MAC et la couche physique.*

Figure 36. Application matérielle programmée sur FPGA

Pour construire une application FPGA reconfigurable, flexible et évolutive, nous avons utilisé le bus d'interconnexion intégré AXI, ce bus permet de connecter plusieurs unités dans le même design FPGA. Avec une librairie importante de modules qui y sont compatibles avec le bus AXI, nous pouvons ajouter à notre conception de nouvelles fonctionnalités qui pourront par exemple améliorer les performances, aider au diagnostic et au débogage de l'implémentation.

D'après la Figure 36, la configuration minimale contient trois blocs essentiels, à savoir :

1. Le contrôleur Ethernet MAC (contrôleur d'accès au support) : De nombreux contrôleurs Ethernet peuvent être utilisés, cela dépend de plusieurs paramètres tels que le type de composant FPGA ciblé, la vitesse de connexion ou le protocole de communication souhaité. Nous avons choisi un module AXI Ethernet lite MAC de la société Xilinx, qui utilise un volume réduit de ressources matériel et peut être contrôlé via l'interface AXI lite. Ce choix est compatible avec les exigences que nous nous sommes imposées.

2. Le contrôleur AXI Master : L'utilisation des bus d'interconnexion nécessite la mise en place d'un module qui gère le flux des données entre les différents éléments connectés aux bus. Dans l'architecture proposée, le bloc « contrôleur AXI Master » a deux fonctions principales.

La première fonction est de piloter le bus AXI, ce qui permet au contrôleur d'accéder aux mémoires de réception et d'émission du contrôleur MAC (RX buffer, TX buffer). La seconde fonction est de réaliser les échanges de données avec le module sous-test (contrôleur numérique).

3. Le superviseur : Ce module gère et synchronise toutes les opérations effectuées par les précédents modules et permet la transmission des datagrammes dans le temps imparti.

6.2 L'INTERFACE AXI

Le bus AXI fait partie de la famille ARM AMBA (**A**dvanced **M**icrocontroller **B**us **A**rchitecture), ces interfaces sont depuis longtemps devenues un des standards les plus utilisés dans l'industrie. AMBA est une norme ouverte et librement disponible pour la connexion et la gestion de blocs fonctionnels dans un système sur puce (SoC). Elle facilite le développement des solutions multiprocesseurs, avec un grand nombre de contrôleurs et de périphériques intégrés [96].

Le bus AXI est une spécification qui définit les protocoles pour la mise en œuvre de systèmes d'interconnexion de haute fréquence et de large bande passante, dans une large gamme d'applications. La version AXI4 comprend trois types d'interfaces :

- AXI4 : Pour les besoins en mémoire haute performance mappée.
- AXI4-Lite : Pour une communication mappée en mémoire simple et à faible débit (pour exemple, vers et depuis les registres de contrôle et les registres d'état).
- AXI4-Stream : Pour les données en streaming à haute vitesse.

Les interfaces AXI4 et AXI4-Lite se composent de cinq canaux différents :

- Un canal de lecture d'adresse (***Read Address***),
- Un canal de lecture de données (***Read Data***),
- Un canal d'écriture d'adresse (***Write Address***),
- Un canal d'écriture de données (***Write Data***),
- Un canal d'écriture de réponse (***Write Response***).

Les données peuvent circuler simultanément dans les deux sens entre le maître et l'esclave. Les tailles de transfert peuvent être choisis. La limite dans AXI4 est une transaction en burst (rafale) comportant jusqu'à 256 transferts de données. L'interface AXI4-Lite n'autorise qu'un seul transfert de données par transaction.

La Figure 37.a montre comment une transaction AXI4 de lecture utilise les canaux de ***Read Address*** et ***Read Data***. Aussi la Figure 37.b montre une transaction AXI4 de lecture utilisant les canaux de ***Write Address***, ***Write Data*** et ***Write Response***.

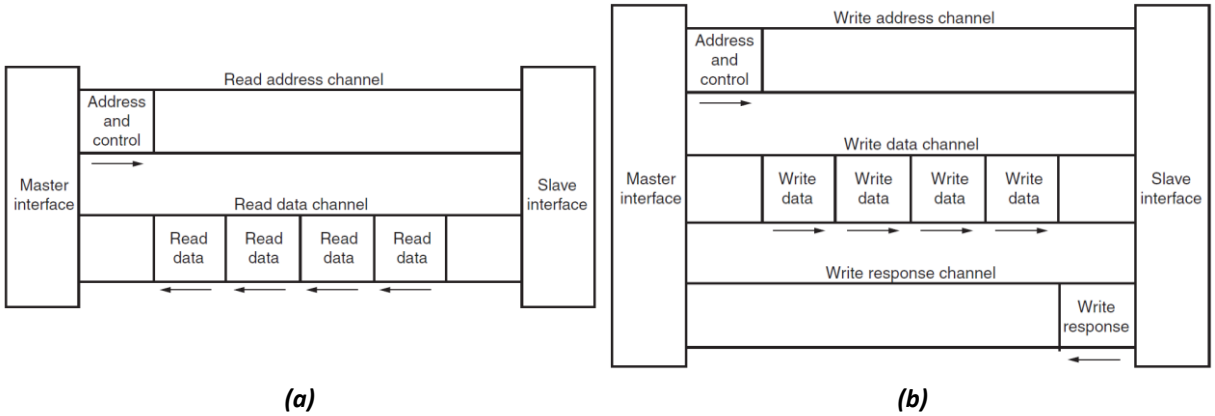


Figure 37. Architecture des canaux de lecture et d'écriture.

Chaque canal de l'architecture AXI4 comporte un nombre important de signaux qui permettent d'établir des transactions de lecture ou d'écriture entre un maitre (ou plusieurs) et un esclave (ou plusieurs).

6.3 AXI ETHERNET LITE MAC (MEDIA ACCESS CONTROLLER)

En ce qui concerne l'intégration et le contrôle de la couche de liaison Ethernet, beaucoup de solution sont disponibles. Ceci dit, vu que le matériel utilisé est un composant FPGA de Xilinx, une solution présentée par ce fournisseur est choisie. Nous avons utilisé le module AXI Ethernet lite MAC qui est une hard-macro IP fournie par Xilinx. Il peut être personnalisé par le concepteur, qui n'a cependant pas accès à sa description HDL. Cela facilite le travail du développeur lors de la mise en œuvre de la couche de liaison des données d'un module compatible Ethernet.

Dans la suite de cette section, nous présenterons quelques détails techniques qui permettent de comprendre les raisons qui ont motivé notre choix, ainsi que les bases du fonctionnement de ce module. Les informations de cette section sont basées sur sa fiche technique donnée par son fournisseur [97].

AXI Ethernet Lite MAC prend en charge la norme IEEE802.3 [98], il utilise l'interface MII (Media Independent Interface) pour communiquer avec les périphériques PHY (Physical Layer) d'un côté, et de l'autre côté communique avec un processeur via l'interface AXI4 ou AXI4-Lite. Le module fournit une interface de 10 mégabits par seconde (Mbps) et 100 Mbps (également appelée Fast Ethernet), offrant les fonctions nécessaires pour fournir une interface Ethernet avec le moins de ressources possibles.

Principales caractéristiques :

1. Licence gratuite,
2. Interface esclave AXI configurable basée sur la spécification AXI4 ou AXI4-Lite,
3. L'interface d'E/S directe mappée en mémoire vers la mémoire à double accès de transmission et de réception de données,
4. Interface MII (**M**edia **I**ndependent **I**nterface) pour la connexion à des émetteurs-récepteurs PHY 10/100 Mbps externes,

5. Mémoire interne à double port TX et RX de 2000 octets pour stocker les données d'un seul paquet,
6. En option des mémoires tampons doubles, pingpong de 4 Ko, pour TX et RX,
7. Reçoit et émet des interruptions,
8. En option une interface MDIO pour l'accès PHY.

Le schéma synoptique du contrôleur AXI Ethernet Lite est illustré sur la Figure 38.

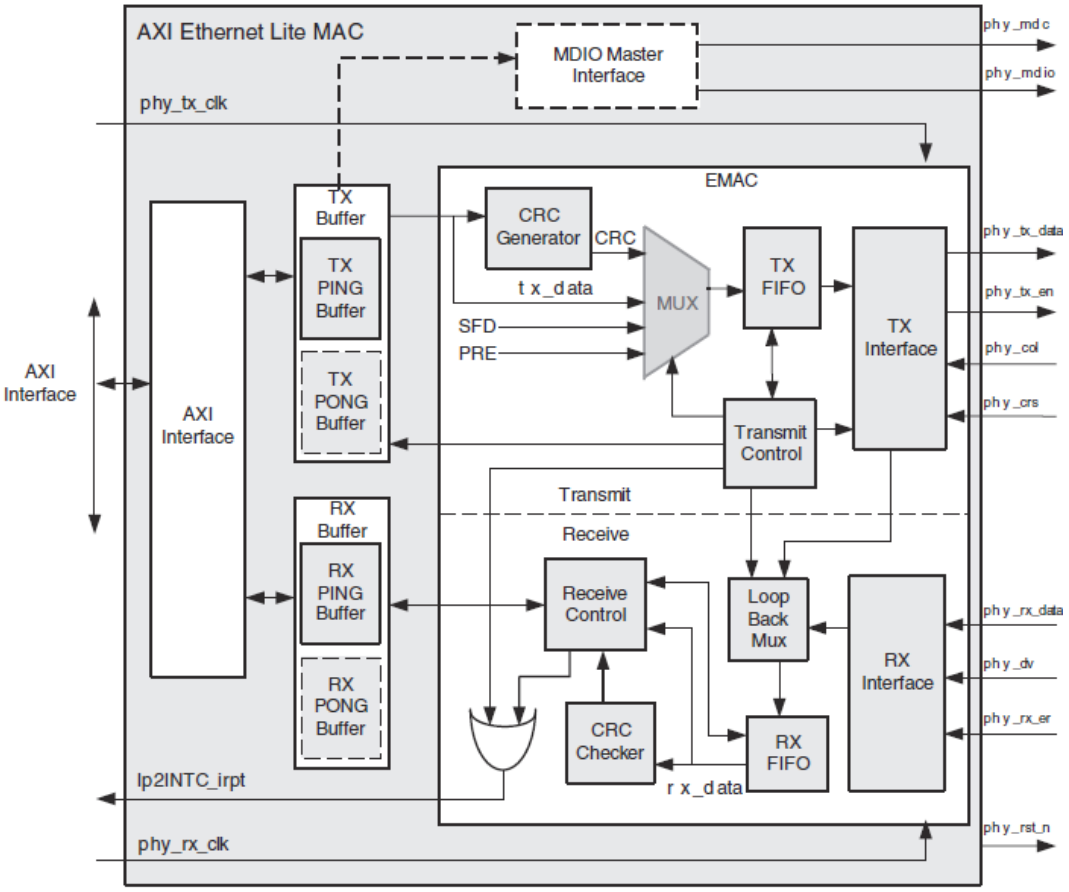


Figure 38. Schéma de principe du MAC AXI Ethernet Lite

Le module est piloté via le bus AXI Lite, où un processeur externe peut lire et écrire dans les registres internes du contrôleur, ces registres permettent de configurer le fonctionnement du contrôleur, et de cette manière de contrôler de flux de données entrant et sortant. Le contrôleur MAC met à disposition du processeur une mémoire pour les données en cours de réception et une mémoire pour les données en cours d'envoie. Différents signaux d'interruption permettent au processeur de visualiser les évènements importants en cours.

Bien que ce contrôleur MAC soit conçu initialement pour une utilisation exclusive avec un processeur intégré, néanmoins, nous avons réussi à le contrôler à travers un code VHDL fonctionnel.

6.4 SUPERVISEUR ET AXI MASTER

Pour arriver à des performances suffisantes qui sont nécessaire au processus de simulation HIL, le module de gestions du bus AXI et le module superviseur sont intégrés dans la même entité, ce qui permet de réduire les latences entre les différents blocs fonctionnels et par conséquent réduire les ressources matérielles réservées.

Le pilotage de l'interface AXI est généralement affecté à un processeur, mais son implémentation fonctionnelle par un module VHDL sur FPGA est possible. L'utilisation d'un processeur facilite la mise en œuvre des protocoles de communication de haut niveau tel que le TCP/IP ou le UDP/IP, ainsi que la gestion native du bus AXI. Cependant, l'utilisation d'un module VHDL permet de réduire considérablement les ressources utilisées dans le circuit FPGA et d'augmenter le débit de transmission. L'implémentation de ce module est réalisée à travers l'utilisation d'un ensemble de machines à état fini (FSM, Finit state machine) et d'algorithmes. Ces entités vont effectuer une série d'opérations qui contrôle et synchronise la communication des données entre le module de commande numérique et l'interface réseau.

Pareillement que les activités décrites dans le modèle de communication OSI, les opérations de supervision et de contrôle sont organisées selon quatre couches fonctionnelles, qui sont :

La couche « application » : c'est l'ensemble des services qui permettent de créer les données utilisateurs et les transmettent à travers le réseau. Les tâches affectées à cette couche sont :

- Gestion du flux de données au niveau du module de commande numérique sous-test,
- Préparation des données utiles récupérées pour leurs envois vers le réseau,
- Récupération des données utiles arrivées par le réseau.

La couche « transport » : Elle rassemble les données utiles dans un datagramme pour le compte de la couche réseau selon le protocole UDP, et inversement elle extrait les données utiles pour les couches supérieures. Les tâches qui y sont affectées sont :

- Création des datagrammes UDP,
- Extraction des données utiles à partir des datagrammes reçus.

La couche « réseau et physique » : dans le cas de la simulation C-HIL elle permet d'assurer la liaison entre l'ordinateur hôte et la carte FPGA à travers le protocole IP, pour ce faire il faut :

- Identifier les adresses physiques MAC et les adresses IP de l'ordinateur hôte et de la carte FPGA,
- Préparer les paquets de données,
- Configurer le contrôleur Ethernet MAC,
- Envoi des paquets à travers le réseau physique.

La couche « AXI » : c'est une couche matérielle, elle permet de gérer la communication entre la couche précédente et le contrôleur Ethernet MAC à travers son interface AXI-lite. Les tâches affectées à cette couche sont définies dans le protocole du bus AXI-lite.

- Opération de lecture des données,
- Opération d'écriture des réponses,
- Opération d'écriture des données.

Les différentes opérations de supervision précitées sont réalisées en exécutant les algorithmes de communication et en programmant correctement les différents registres du contrôleur MAC. Ces opérations permettent d'établir la liaison réseau entre la carte FPGA et l'ordinateur hôte.

6.4.1 Encapsulation des données

Les données ne sont pas transmises telles quelles sur le réseau, car chaque protocole a besoin d'informations bien précises pour exécuter correctement son travail. De ce fait, nous avons besoins de préparer à chaque couche de liaison un format bien précis qui implique, entre autres, les données utiles, les datagrammes et les paquets. La Figure 39 illustre un aperçu du processus d'encapsulation et de décapsulation des données.

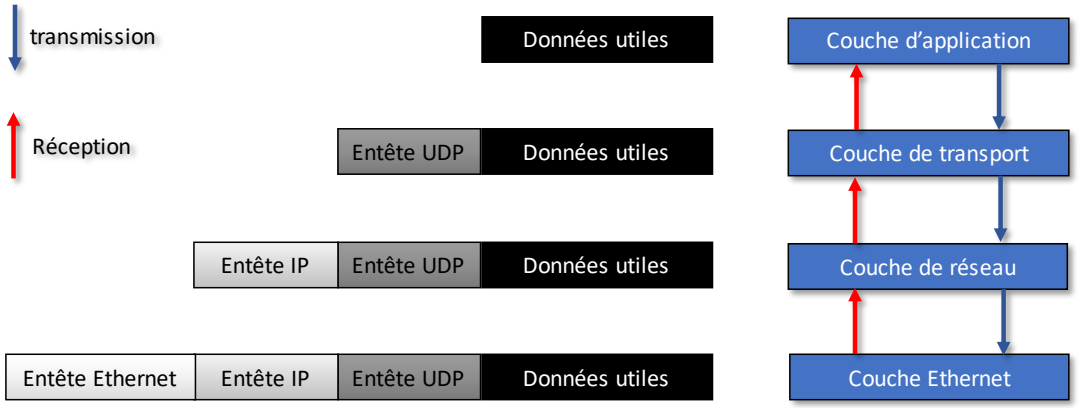


Figure 39. Encapsulation des données à travers le protocole UDP/IP

Le format final de la trame que doit récupérer le contrôleur MAC est telle qu'indiquée sur la figure ci-dessous.

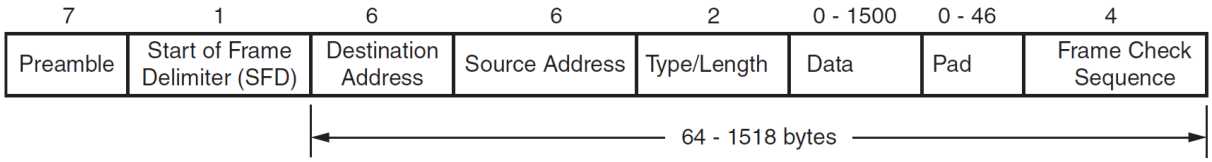


Figure 40. Trame de données Ethernet [97]

6.4.2 Traitement des données

Le module de supervision est chargé de l'essentiel des opérations qui permettent la transmission et la réception des données nécessaires à la simulation C-HIL. Concernant la transmission des données du module de commande numérique (carte FPGA) vers le simulateur (ordinateur hôte), les données sont encapsulées dans un format qui garantit leur intégrité et qui permet aux différents matériels associés d'identifier leur source et leur destination. Parallèlement à cette opération, le module superviseur configure les registres du contrôleur Ethernet MAC à travers l'utilisation du bus AXI. Aussi, à la fin de l'encapsulation, la trame de données est enregistrée dans le buffer du contrôleur MAC. Enfin,

un dernier registre à configurer permet d'enclencher la transmission des données à travers la couche physique du réseau.

Dans le sens de la réception, à l'arrivée de la trame des données dans le buffer de réception du contrôleur MAC, ce dernier émet une interruption vers le module superviseur qui démarrera aussitôt l'algorithme de réception. Les données utiles sont alors extraites et vérifiées avant de les envoyer à leur tour vers le module de commande numérique.

Afin d'établir une simulation C-HIL avec le module de commande sous-test, une machine à états finis (FSM) a été implémentée conformément au diagramme de la Figure 41. La FSM contrôle le bus AXI et émet les signaux de commandes pour gérer la communication avec le contrôleur MAC. Lorsqu'un paquet est reçu sur le buffer RX, la FSM recherche le datagramme UDP et vérifie la somme de contrôle des données (CheckSsm), puis les données nécessaires à la simulation C-HIL sont extraites et envoyées au contrôleur sous test (CUT). La FSM est placée en mode 'attente' jusqu'à ce que le CUT donne une réponse, cette réponse est insérée dans un nouveau datagramme UDP et envoyée à son tour vers le PC hôte.

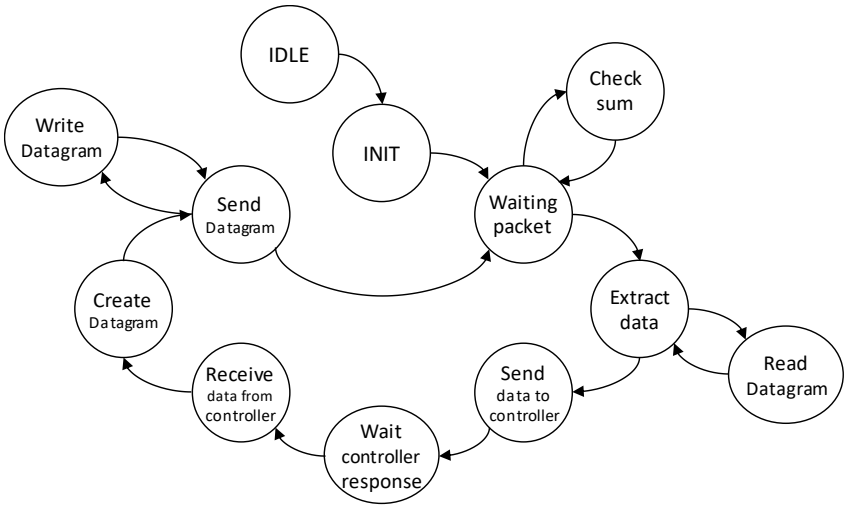


Figure 41. Diagramme de la machine à états finis du superviseur

6.5 IMPLEMENTATION SUR FPGA

Parmi les trois grands fabricants de FPGA qui sont Xilinx, Altera et Actel, Xilinx est le plus dominant sur le marché. Compte tenu des exigences que nous avons fixées pour la conception de la plateforme HIL-AMS, les FPGA type SRAM sont la solution la plus appropriée en raison de leur prix, de leur degré élevé d'intégration et de leur flexibilité.

Le composant FPGA retenu pour les expérimentations est un FPGA d'entrée de gamme, c'est un Spartan-XC6SLX9 de Xilinx. Il a été choisi pour son coût modeste et la disponibilité de ses outils de développements, ce choix est entièrement lié au choix de la carte de développement. Vu les besoins de notre conception, il n'était pas nécessaire d'utiliser un composant sophistiqué pour développer nos fonctions numériques. Ces fonctions sont écrites en VHDL, leur portage de la gamme Spartan vers des versions de haute performance et même vers des FPGA d'autres constructeurs est garanti par le langage.

Une autre raison du choix du Spartan est l'environnement de développement ISE de Xilinx qui est utilisé par une communauté d'utilisateurs assez large et dispose de tous les outils nécessaires pour le prototypage FPGA. Aussi, ces outils offrent au développeur des composants sous licence gratuite pour synthétiser des fonctions spécialisées pour l'interfaçage et le traitement du signal.

Les principales caractéristiques du FPGA Spartan-XC6SLX9 sont résumées dans le Tableau 6.

Tableau 6. Aperçu des caractéristiques du FPGA Spartan-6 SLX9

Device	Logic Cells(1)	Configurable Logic Blocks (CLBs)			DSP48A1 Slices(3)	Block RAM Blocks		CMTs(5)	Memory Controller Blocks (Max)(6)	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices(2)	Flip-Flops	Max Distributed RAM (Kb)		18 Kb(4)	Max (Kb)						
XC6SLX4	3,840	600	4,800	75	8	12	216	2	0	0	0	4	132
XC6SLX9	9,152	1,430	11,440	90	16	32	576	2	2	0	0	4	200
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232

(2) : Chaque slice de Spartan-6 FPGA contient quatre LUT et huit bascules.
(3) : Chaque slice DSP48A1 contient un multiplicateur 18 x 18, un additionneur et un accumulateur.
(4) : Les blocs RAM ont une taille de base de 18 Ko.

6.5.1 Plateforme matérielle

Afin de valider la conception de la plateforme HIL-AMS, nous avons évalué quelques cartes de développement FPGA d'entrée de gamme en termes de ressources matérielles, performances et interface de communication. Après avoir estimé les ressources matérielles utilisées pour l'implémentation de l'application FPGA, nous avons choisi la carte de développement spartan-6 FPGA LX9 MicroBoard. Les particularités de cette carte de développement qui nous intéressent le plus sont son faible coût et la disponibilité d'une interface réseau Ethernet qui est compatible avec le module AXI Ethernet.

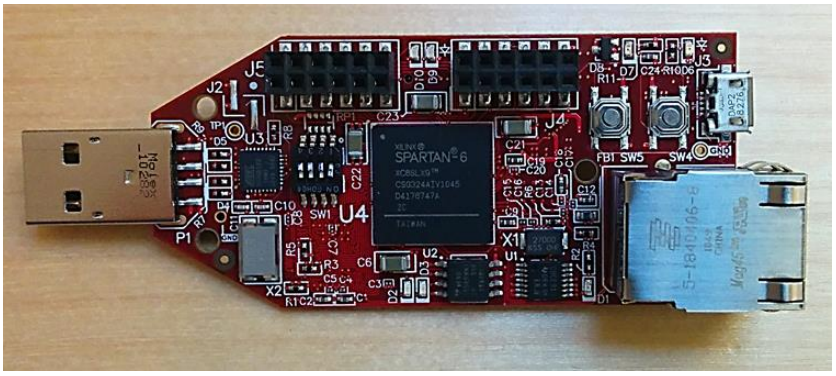


Figure 42. Carte de développement Avnet Xilinx Spartan 6 LX9 Microboard.

6.5.2 Ressources matérielles utilisées

Le Tableau 7 répertorie les ressources FPGA utilisées pour implémenter les composants nécessaires à l'application matérielle de la plateforme HIL-AMS. Les ressources consommées dépendent de la technologie FPGA utilisée. Et les proportions sont exprimées en fonction des ressources disponibles dans le circuit Spartan-6 XC6SLX9.

Tableau 7. Ressources FPGA utilisées pour l'implémentation matérielle.

Modules	Bascule	LUT	Bloc DSP
Ressources disponibles	11440	5720	16
AXI_Ethernet_lite	492 [4,3%]	494 [8,6]	0
Supervisor+AXI Master	197 [1,7%]	220 [3.8]	0
Global	689 [6%]	714 [12.4]	0

Dans ce tableau, l'utilisation globale des LUT (LookUp-Table ⇔ bloc logique) pour l'implantation des principaux modules utilisés, qui sont le module AXI Ethernet lite et le module superviseur, ne représente que 12,4% des LUT disponibles. Ces résultats montrent que la plateforme proposée ne nécessite que peu de ressources FPGA. Le reste des ressources qui représente plus 87% des LUT du circuit Spartan-6SLC9 peuvent accueillir le module de commande numérique ainsi que d'éventuels modules qui peuvent embarqués d'autres fonctionnalités additionnelles.

7 CONCLUSION

Au cours de ce chapitre, nous avons exposé les étapes que nous avons poursuivies pour parvenir au développement d'une plateforme de simulation Hardware-In-the-Loop. Cet outil est destiné à supporter la méthodologie de conception que nous avons entreprise dans le chapitre 3. Cette méthodologie ainsi que la plateforme susmentionnée utilisent la modélisation VHDL-AMS comme ossature de développement des systèmes multi-technologiques.

La plateforme HIL-AMS que nous avons développée est un outil matérielle/logicielle unique de simulation, elle permet de relier le monde de la simulation au prototypage réel. Elle permet de tester et de vérifier par simulation des sous-systèmes numérique, généralement des éléments de commande. Ces modules numériques sont implémentés dans des composants programmables de type FPGA et le reste du système est modélisé puis simulé en utilisant le langage VHDL-AMS.

La plateforme réalisée est divisée en trois parties : le logiciel de simulation SMASH, une application matérielle implémentée sur FPGA et une application logicielle, cette dernière partie permet de contrôler le flux de données entre le simulateur et une carte de développement FPGA. L'une des premières exigences que nous avons validées est que la plateforme sera réalisée avec un coût minimum et avec des performances tangibles. Pour ces différentes raisons, nous avons choisi une carte développement à base de FPGA Spartan 6 avec une liaison haut débit Ethernet pour le transfert des données.

Finalement, nous avons réussi à implémenter les différentes parties de la plateforme et de tester l'intégrité du chemin de données entre le logiciel et le matériel impliqués. Toutefois, pour que la plateforme soit validée intégralement il faut utiliser des exemples d'applications pratiques, ensuite de réaliser des simulations HIL en mesurant les différentes métriques de performance.

Chapitre 5

Cas d’étude, application et test

1 INTRODUCTION

Ce chapitre décrit comment la méthodologie élaborée au chapitre 3 va être utilisée pour la conception d'un système de conversion de puissance. A travers un cas d'étude pratique, nous allons approuver les différentes étapes du processus de conception, notamment la modélisation et la simulation via les langages VHDL et VHDL-AMS. De plus, cet exemple permettra la validation de la plateforme de simulation HIL que nous avons mise en œuvre.

Parmi plusieurs exemples testés, nous avons choisi de retenir le système à base de convertisseur de puissance continu/continu. Nous détaillerons dans cet exemple les étapes de modélisation et de simulation et enfin la mise en application de la plateforme conçue pour simulation Hardware-In-the-Loop. Les différentes étapes de modélisation du système sont ainsi présentées, en particulier le sous-système de commande numérique. En outre, les méthodes de conversion et de transformation de modèle sont utilisées pour passer d'une représentation à une autre, ceci jusqu'à la synthèse d'un modèle optimisé pour l'implémentation sur un circuit FPGA.

2 PRESENTATION DU CAS D'ETUDE

Afin de valider les contributions des travaux de thèse, nous avons choisi une application industrielle à notre méthodologie de conception, nous avons choisi un exemple de convertisseur abaisseur DC/DC commandé avec un régulateur PID. Cet exemple est suffisant pour illustrer notre concept et mettre en avant les avantages et l'apport de notre démarche de conception système. Le système à concevoir est composé d'une partie analogique et mixte et d'une partie purement numérique. La première étape de son développement consiste à mettre en œuvre une architecture et de vérifier son adéquation avec les exigences préalablement formulées. De nouveaux éléments sont à intégrés dans l'architecture du système à mesure des transformations et des conversions de modèle.

2.1 DESCRIPTION DU CONVERTISSEUR DE PUISSANCE DC/DC DEVOLTEUR (BUCK)

Le système à concevoir consiste en un convertisseur de puissance DC/DC avec un contrôleur en boucle fermée (voir Figure 43) ; un convertisseur de puissance est un exemple typique de système AMS, il implique un comportement analogique et un comportement numérique. Aussi, il peut inclure d'autres fonctionnalités des autres disciplines de la physique telles qu'un comportement thermique ou une chaîne de traction mécanique.

Le système de puissance utilisé permet de régler la tension aux bornes de la charge de manière à ce qu'elle correspond à la valeur consignée. Pour ce faire, un contrôleur PID (proportionnel-intégral-dérivatif) suit l'erreur *e(t)* de la tension mesurée *Vm* avec la tension consignée V_{ref} , il génère ensuite un signal commande *u(t)*. Ce signal de commande module un signal PWM, ce dernier est appliqué à l'interrupteur de puissance qui permet d'ajuster le flux de puissance électrique en contrôlant la tension de sortie.

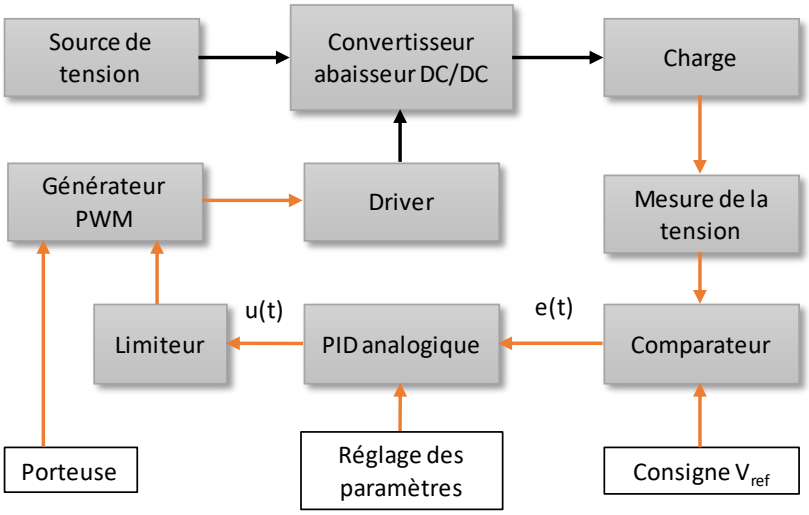


Figure 43. Architecture fonctionnelle du système à concevoir

L'élément central du système considéré est un convertisseur de puissance DC/DC de type dévolteur (Buck). La figure décrit la structure de ce convertisseur qui est basé sur un seul interrupteur de puissance commandé, en l'occurrence un MOSFET, et une diode de roue libre. Cette configuration peut être la base d'une description VHDL-AMS qui utilise un modèle structurel, toutefois d'autres types de modèles peuvent être développés.

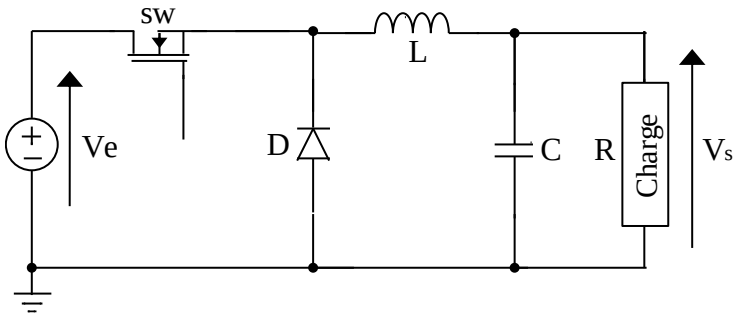


Figure 44. Schéma de principe d'un convertisseur dévolteur "Buck"

Comme pour les autres parties du système global, nous pouvons utiliser des modèles de types structurel, comportemental, des modèles moyens, des modèles en LUT (Look-up table ⇔ table de correspondance), ou bien la combinaison de ces différents types dans un seul modèle générique.

Moyennant l'utilisation de ces différents types de modèles, le VHDL-AMS offre la possibilité aux développeurs de réaliser une analyse fonctionnelle de l'architecture système, et de valider le fonctionnement qui correspond aux spécifications des besoins de la partie prenante. Chaque composant du système peut être facilement modélisé à différents niveaux d'abstraction et à différents niveaux de granularité, en fonction de la précision et de la complexité requise pour chaque modèle. Ce qui permet d'effectuer le type d'analyse appropriée et le type de simulation qui correspond le mieux à la phase courante du processus de développement en cours.

Toutefois, comme nous l'avons déjà indiqué dans les paragraphes précédents, nous avons préalablement fixé notre choix en ce qui concerne la méthode de régulation utilisée. Nous avons choisi une commande à base de contrôleur PID, ce choix vient du fait que ce type de commande est

largement utilisé en industrie, et il permet d'illustrer clairement les différentes étapes de conception de la commande numérique et de son implémentation sur circuit FPGA.

3 BOUCLE DE CONTROLE

Dans la conception des systèmes électriques et des systèmes de puissance, l'analyse fonctionnelle permet de contribuer à la mise en œuvre des sous-systèmes de commande. Lors de cette analyse le développeur utilise des modèles mathématiques à temps continu pour concevoir des sous-systèmes de commande analogique. En utilisant des méthodes de discrétisation adéquates le développeur peut créer des modèles de commande discrets équivalents aux modèles de la commande analogique conçue [56]. Ceci, permet de synthétiser l'architecture du contrôleur numérique et finalement de réaliser son implémentation dans une plateforme dédiée.

Dans le présent cas d'utilisation, nous utilisons un régulateur PID pour illustrer les différentes étapes de modélisation qui permettent de passer d'un modèle à temps continu vers un modèle à temps discret en utilisant le langage VHDL-AMS.

3.1.1 Régulateur PID

Les régulateurs PID sont de loin le type de régulateurs le plus utilisé dans les applications de contrôles industriellesFigure 43, car ils possèdent une structure simple et de bonne performance. L'utilisation des modèles analogiques et des modèles numériques est largement rapportée dans la littérature scientifique [99].

Le principe du régulateur PID est d'agir sur l'erreur *e(t)* formé par la différence entre la consigne et la sortie du procédé à commander *s(t)*, ceci au moyen de trois actions :

- Une action proportionnelle s'exprimant par le gain Kp ;
- Une action intégrale se manifestant par le gain Ki ;
- Une action dérivée caractérisée par le gain Kd.

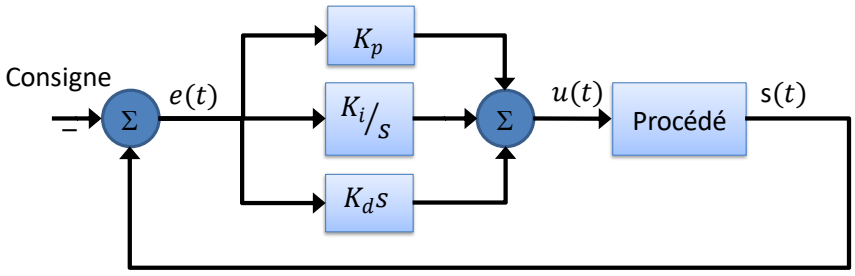


Figure 45. Principe d'un régulateur PID

Ces trois composantes seront combinées afin d'obtenir un compromis entre la précision, la stabilité et la rapidité de la réponse *u(t)*. La forme la plus classique du régulateur PID est représenté sur la Figure 45, la formule générale de la commande *u(t)* est alors exprimée dans le domaine temporel telle que :

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \qquad (1)$$

On peut alors donner la fonction de transfert d'un tel contrôleur par :

$$H_{PID}(s) = K_p + \frac{K_i}{s} + K_d . s = \frac{K_i + K_p . s + K_d . s^2}{s} \qquad (2)$$

La formule (1) permet de faire une analyse temporelle du système afin de choisir les coefficients du contrôleur PID. Différentes méthodes existent pour la mise au point de ces coefficients, dans ce cas nous utilisons l'analyse en domaine fréquentiel en se basant sur la formule (2). Via cette formule en peut analyser la stabilité et la robustesse du contrôleur utilisé.

3.1.2 Discrétisation du contrôleur PID

Afin de mettre en œuvre l'algorithme de contrôle en utilisant une technologie numérique, l'équation (2) doit être discrétisée. La discrétisation peut être réalisée de plusieurs manières en utilisant des méthodes d'approximation [100], ces méthodes sont utilisées pour passer à une représentation dans le domaine fréquentiel discret (domaine en z).

Nous avons choisi la méthode Backward d'Euler pour discrétiser le contrôleur. L'application de cette méthode revient à remplacer la variable s de l'équation (2) par :

$$s = \frac{z - 1}{T_e \cdot z} = \frac{1 - z^{-1}}{T_e}$$

Où *Te* est la période d'échantillonnage.

Après la transformation en Z, on obtient la fonction de transfert donnée par l'équation 3 :

$$H_{PID}(z) = K_p + K_i \cdot T_e \cdot \frac{z}{z - 1} + \frac{K_d}{Te} \cdot \frac{z - 1}{z} \qquad (3)$$

Pour la simulation comportementale et l'analyse fréquentielle du système, nous pouvons utiliser l'équation (3), par contre pour la mise en œuvre numérique, nous sommes davantage intéressés par la formule du domaine temporel discret donnée par l'équation :

$$u[k] = u[k - 1] + K_1 e[k] + K_2 e[k - 1] + K_3 e[k - 2] \qquad (4)$$

Avec :

$$K_1 = K_p + Te\ K_i + \frac{K_d}{Te} \qquad K_2 = -K_p - 2\frac{K_d}{Te} \qquad K_3 = \frac{K_d}{Te}$$

Le schéma équivalent pour l'équation (4) est donné par la Figure 46, ce schéma permet de visualiser la structure fonctionnelle du régulateur PID et le chemin de données qui correspond à son modèle à temps-discret.

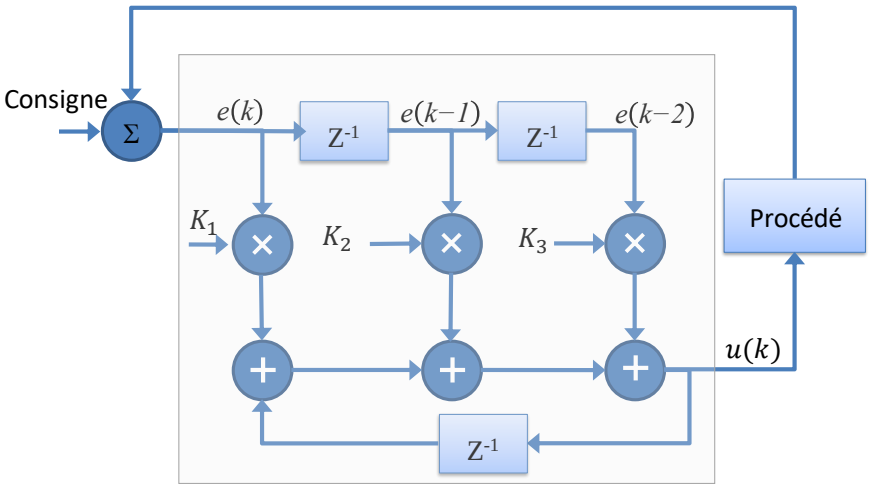


Figure 46. Structure d'un PID discret.

3.1.3 Implémentation en VHDL-AMS

L'un des grands avantages du langage VHDL-AMS est d'apporter une grande souplesse au passage d'un type de modèle à un autre, et d'un niveau d'abstraction plus élevé à un autre plus bas. Pour l'analyse et la simulation de la commande en temps continue nous pouvons utiliser directement les formules (1) et (2) pour décrire des modèles comportementaux tels qu'illustré dans la Figure 47.

```
architecture Temp_Arch of PID is
      constant K : REAL := 1.0;
      constant Kd : REAL := 1.0;
      constant Ki : REAL := 1.0;
begin
      Ut == K * Et + Ki * Et'INTEG + K_d * Et'DOT;
end architecture Temp_Arch;
```

```
architecture Laplace_Arch of PID is
      constant num : real_vector := (1.0, 1.0, 1.0);
      constant den : real_vector := (0.0, 1.0, 0.0);
begin
      Ut == k * Ut'ltf(num, den);
end architecture Laplace_Arch;
```

Figure 47. Extrait du code VHDL-AMS du contrôleur PID dans le domaine temporel

Les codes illustrés dans la Figure 47 permettent d'entrevoir un aperçu du code de l'architecture VHDL-AMS qui permet de décrire le fonctionnement d'un contrôleur PID. En outre, pour voir l'interface qui permet de connecter le bloc PID aux autres blocs du système il faut examiner le code de l'entité VHDL-AMS qui est présenté dans la Figure 48.

```
Entity PID is
      port ( quantity Et : in REAL;
             quantity Ut : out REAL
           );
End PID;
```

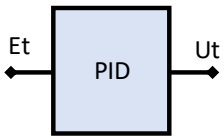


Figure 48. Entité et interface du bloc "contrôleur PID"

Dans les deux cas utilisés pour la modélisation en temps continue, nous utilisons la même interface, ce qui permet de préserver l'intégrité du système globale alors même qu'on a changé le contenu d'un des modèles de simulation.

Concernant le modèle dans le domaine fréquentiel, nous utilisons pour décrire la formule donnée en (3) une des méthodes offertes par le langage VHDL-AMS qui utilise l'attribut ***ZTF***. Cet attribut permet de définir la fonction de transfert du bloc PID sans avoir à modifier son interface, on précise seulement la valeur de la période d'échantillonnage *Te*. Le code et l'interface illustré dans la Figure 49 permet ainsi d'étudier la stabilité du système pour différentes valeurs de *Te*. C'est une étape très importante qui permet de valider la configuration et la fréquence de fonctionnement du contrôleur PID.

```
architecture Z_Arch of PID is
      constant Te : real := 1.0 / 10000.0;
      constant num : real_vector := (1.0, 1.0, 1.0);
      constant den : real_vector := (0.0, 1.0, 0.0);
begin
      Ut == Et'ztf(num, den, Tsrnp);
end Z_Arch;
```

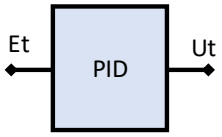


Figure 49. Code VHDL-AMS et interface du contrôleur PID dans le domaine fréquentiel

La dernière étape dans le processus de conception en VHDL-AMS est de décrire le contrôleur PID en utilisant le modèle discret de la formule (4), le code VHDL-AMS sera basé sur un ensemble d'instructions et de déclarations de type discret. Ce code permet de décrire le comportement cyclique et séquentiel du contrôleur PID. La Figure 50 montre le code de l'architecture, les instructions de type discret doivent être intégrées dans un « *process* » cadencé par un signal d'horloge « *clk* », ces instructions transcrivent le schéma fonctionnel de la Figure 46.

```
architecture Discret of PID_Discret is
      signal U_k_1, error_k_1, error_k_2 : real;
      constant K1 : REAL := 1.0;
      constant K2 : REAL := 1.0;
      constant K3 : REAL := 1.0;
begin
      process (clk)
      begin
            if clk'event and clk = '1' then
                  U_k_1 <= U_k;                    -- U(k-1)
                  error_k_1 <= error_k;            -- error(k-1)
                  error_k_2 <= error_k_1;          -- error(k-2)
                  U_k == U_k_1 + K1*error + K2 *error_k_1 + K3*error_k_2;
            end if;
      end process;
end Discret;
```

Figure 50. Code VHDL-AMS du contrôleur PID dans le domaine temporel discret

Lors de cette étape nous utilisons des signaux discrets de type réel, il n'est pas question ici de faire un code destiné à la programmation d'un FPGA. Le modèle décrit dans cette phase permet l'adaptation de l'interface avec le système simulé et de vérifier l'algorithme de commande. Pour ne altérer le code du système global, nous avons empaqueté le modèle PID discret dans l'interface du modèle PID analogique.

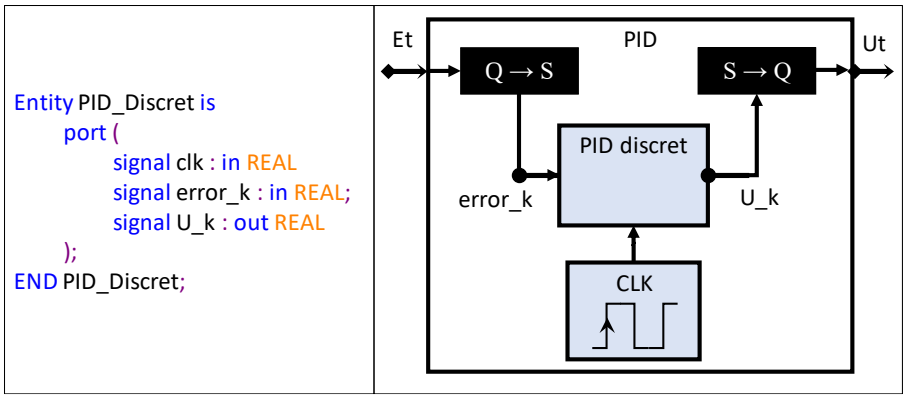


Figure 51. Code VHDL-AMS pour l'interface du contrôleur PID avec le reste du système

Selon le niveau d'abstraction et la finesse de la description souhaitée nous pouvons utiliser différentes méthodes de conversion des quantités analogiques vers des signaux analogiques. VHDL-AMS dispose de plusieurs attributs qui permettent cette conversion. Nous pouvons aussi utiliser des modules de conversion analogique/numérique (CAN) et numérique/analogique (CNA), cette solution ne correspond pas aux besoins de cette phase de modélisation mais elle sera essentielle pour la simulation HIL.

A ce niveau de modélisation, nous allons user des potentialités du langage et du logiciel de simulation qui permet de faire de simple affectation contrôlée pour passer du domaine continue vers le discret et vice-versa. Pour passer de l'analogique vers le discret ou d'une quantité vers un signal nous utilisons la séquence d'instructions suivante :

```
process (clk)
begin
   if clk'event and clk = '1' then
       S <= Q;
   end if;
end process;
```

Et pour passer du discret vers l'analogique ou d'une quantité vers un signal nous utilisons la séquence d'instructions ci-dessous, où l'instruction **break** permet de synchroniser le simulateur analogique sur le simulateur discret. C'est ainsi qu'on peut mettre à jour la valeur de la quantité Q avec la valeur du signal S.

```
Q == S;
break on S;
```

Par ailleurs, à ce stade de la modélisation le code VHDL-AMS du contrôleur PID (Figure 50) est déjà compatible avec le langage VHDL, lors des dernières transformations nous avons éliminé les déclarations et les instructions qui appartiennent à l'extension AMS, cette extension est dédiée exclusivement à la description du monde analogique, elle ne peut être implantée sur circuit FPGA. Voici quelques exemples d'instruction qu'il faut retirer du code.

- Déclaration des objets de type terminal,
- Déclaration des objets de nature multi-physique,
- Déclaration des quantités,

- L'utilisation des fonction mathématiques,
- Les attributs qui interagissent avec des quantités tels que : ***RAMP***, ***LTF***, ***ZOH***, ***DOT***, ***INTEG***…etc,
- Les instructions ***break on***,
- Les fonctions ***procedural***,

Le but de ces transformations est de passer vers un code VHDL synthétisable sur FPGA, une fois le code vérifié, nous devons le convertir vers un format de calcul en virgule fixe. Pour ce faire différentes étapes de mise en œuvre sont à élaborer.

3.1.4 Transformation en virgule fixe

L'implémentation d'une commande numérique sur FPGA nécessite l'utilisation des données dans un format en virgule fixe. Bien que l'utilisation d'un format en virgule flottante soit possible, néanmoins il reste marginal, car ce format consomme beaucoup de ressources FPGA et n'apporte guère d'améliorations significatives [55] [101]. Pour passer d'une représentation de données de type réel à une représentation en virgule fixe, il est recommandé de suivre dans l'ordre les étapes suivantes :

- Estimation de la plage de variation des données par simulation VHDL-AMS,
- Détermination du format de données,
- Modélisation VHDL en virgule fixe,
- Vérification par comparaison des simulations entre le format réel et le format en virgule fixe,
- Génération d'un modèle synthétisable.

Dans notre application, le contrôleur PID peut être implémenté en virgule fixe en utilisant un format binaire signé de 32Q16 bits (largeur de mot de 32 bits avec 16 bits de précision) tel que présenté ci-dessous.

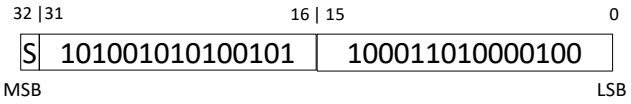


Figure 52. Format binaire choisi pour les signaux d'E/S du PID.

Nous avons choisi ce format pour définir l'entrée et la sortie du contrôleur, ce qui est suffisant pour reproduire le format des interfaces utilisé réellement pour ce genre de contrôleur. L'arithmétique à virgule fixe est utilisée pour les différents étages du contrôleur PID, alors les variables internes de ce bloc sont codées dans des formats différents selon leur plage de variation et en fonction de la précision souhaitée. Toutes les représentations des données doivent être réalisées dans les limites des ressources FPGA, qui regroupent essentiellement : la limite du nombre de broches disponibles, la limite du nombre de LUT et la limite du nombre d'unités de calcul DSP.

Une augmentation injustifiée de la taille du format binaire augmente inutilement le temps d'exécution et la surface occupée sur le circuit FPGA. Par contre, le choix d'une taille trop faible du format binaire peut conduire à la saturation des signaux et à une perte de précision, conduisant ainsi

à l'altération des performances du contrôleur numérique, voir à son dysfonctionnement. De ce fait, à ce stade, on doit choisir avec la plus grande attention le nombre de bits des formats binaires pour ces blocs intermédiaires, en prenant en considération les contraintes logicielles et les remarques précédemment mentionnées, afin de réaliser un codage approprié du flot de données.

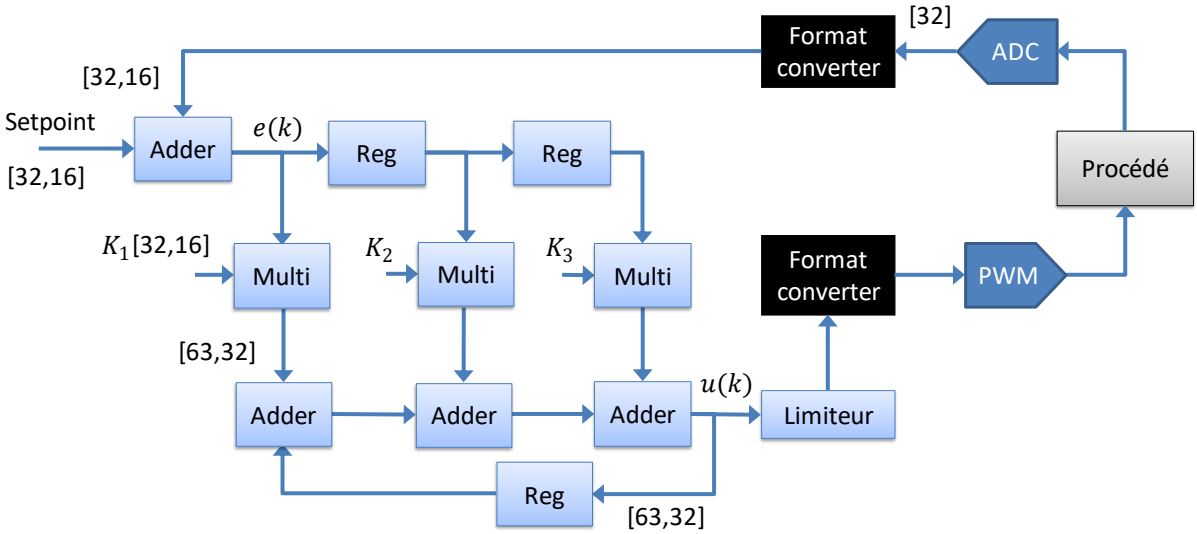


Figure 53. Structure du contrôleur PID avec format des données

La Figure 53 représente une structure simplifier du contrôleur PID numérique, cette structure utilise les principaux éléments du circuit FPGA impliqués dans la mise en œuvre matérielle. Les différents éléments de cette structure sont associés dans un processus en pipeline et synchronisés pour donner une réponse tous les 3 cycles d'horloge.

Pour la vérification du contrôleur implanté, nous pouvons intégrer son code dans une entité d'un niveau supérieur (voir Figure 54). Ce bloc permet d'adapter les interfaces du contrôleur au modèle globale du système et au modèle de simulation comportementale. D'autant que le code VHDL est toujours compatible avec la simulation VHDL-AMS, exception faite lorsque celui-ci intègre des bibliothèques spécifiques aux FPGA ou des bibliothèque propriétaires.

Il est à noter que le code VHDL synthétisable doit respecter un certain nombre de règles et de standard, pour qu'il soit compatible avec une implantation sur FPGA, ces usages sont largement (Bezerra & Lettnin, 2014) [102] [103].

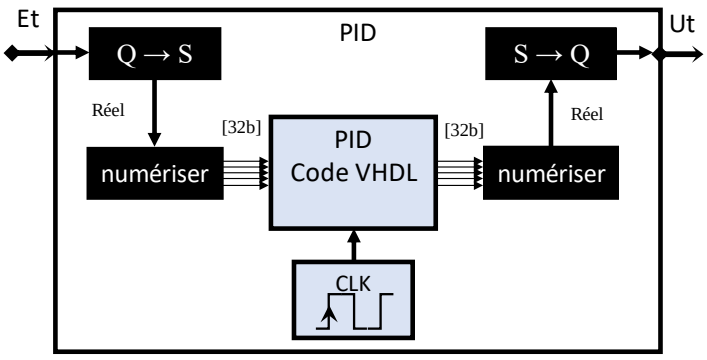


Figure 54. Empaquetage du contrôleur PID numérique.

4 VHDL-AMS SIMULATION

4.1 1ERE SIMULATION

Au tout début du processus de conception, l'ensemble du système est modélisé en VHDL-AMS, ensuite simulé en mode temps-continu avec les paramètres présentés dans le tableau 1. L'objectif de ce premier scénario est de vérifier le fonctionnement de chaque élément du système par apport aux spécifications préalablement définis. Cette simulation, dite comportementale, permet aussi de vérifier l'architecture développée.

Tableau 8. Paramètres du système

Paramètres	Valeurs
Fréquence PWM	10kHz
Vi	24V
Inductance	1 mH
Capacité	1 mF
Resistance de la charge	1 Ω
Kp	1,5
Ki	80
Kd	20

Pour montrer la portabilité des modèles VHDL-AMS, nous avons utilisé un autre logiciel de simulation qui est le logiciel ***SystemVision*** de la société ***Mentor Graphics***, dans cette phase de modélisation la plupart des logiciels de simulation VHDL-AMS sont plus ou moins identiques en termes de prise en charge de la sémantique du langage. Par contre les différences sont palpables du point de vue des performances et de l'ergonomie des logiciels. La Figure 55 montre le schéma-bloc du système à simuler dans l'environnement ***SystemVision***.

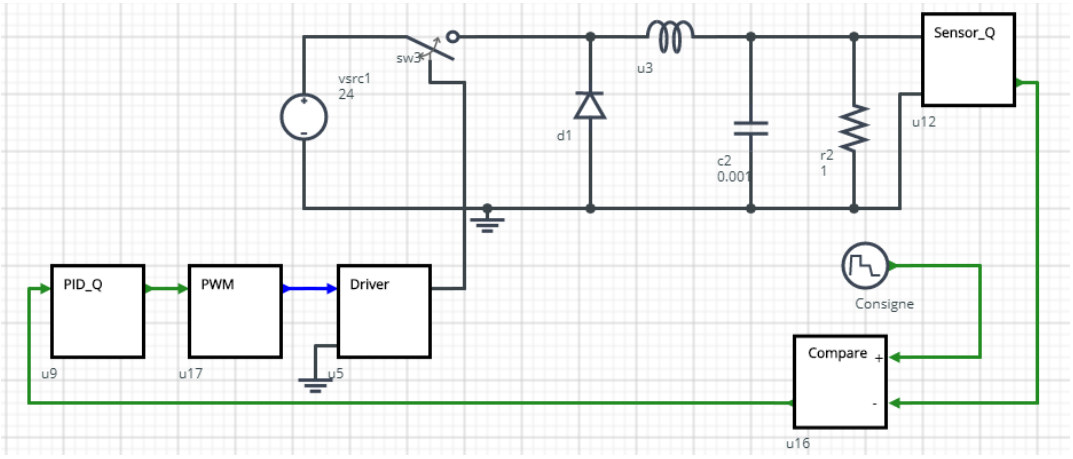


Figure 55. Schéma-bloc du système dans l'environnement SystemVision

Les résultats de simulation pour ce scénario sont reportés sur la Figure 56. Nous avons effectué une simulation du système avec des modèles comportementaux pour 300ms, avec différentes valeurs de la consigne. Ces résultats permettent de vérifier le bon fonctionnement des modèles du système, et de valider un certain nombre de paramètres tel que la fréquence de commutation, ainsi que les

coefficients du contrôleur PID. Aussi, les résultats validés seront considérés comme une référence pour la vérification des prochaines étapes de modélisation.

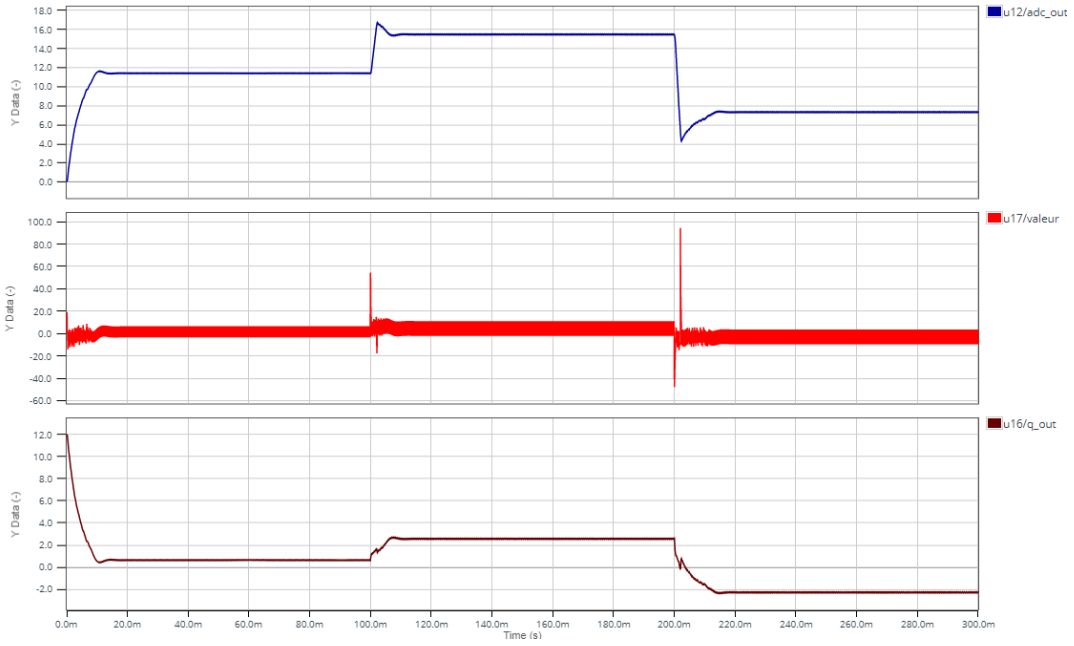


Figure 56. Tension à la charge, commande PID et erreur avec la consigne de référence

4.2 2EME SIMULATION

Dans cette deuxième étape de vérification, le modèle utilisé pour représenter le contrôleur PID est le modèle numérisé en virgule fixe, ce modèle est entièrement codé en VHDL synthétisable. Pour pouvoir l'utiliser il faut l'insérer dans une entité pour l'adapter et l'interfacer avec le modèle de simulation système.

Pour réaliser les différents plans de simulation nous allons utiliser le langage VHDL-AMS avec le logiciel SMASH. Toutefois, le logiciel de simulation ne dispose pas d'outil de modélisation graphique comme pour SystemVision, de ce fait nous devons créer des testbench pour contenir les modèles et créer des scénarios de test. La Figure 57 illustre une vue structurelle d'un testbench, où les éléments du système y sont intégrés en plus des sources de signaux qui établissent les paramètres des différents scénarii utilisés.

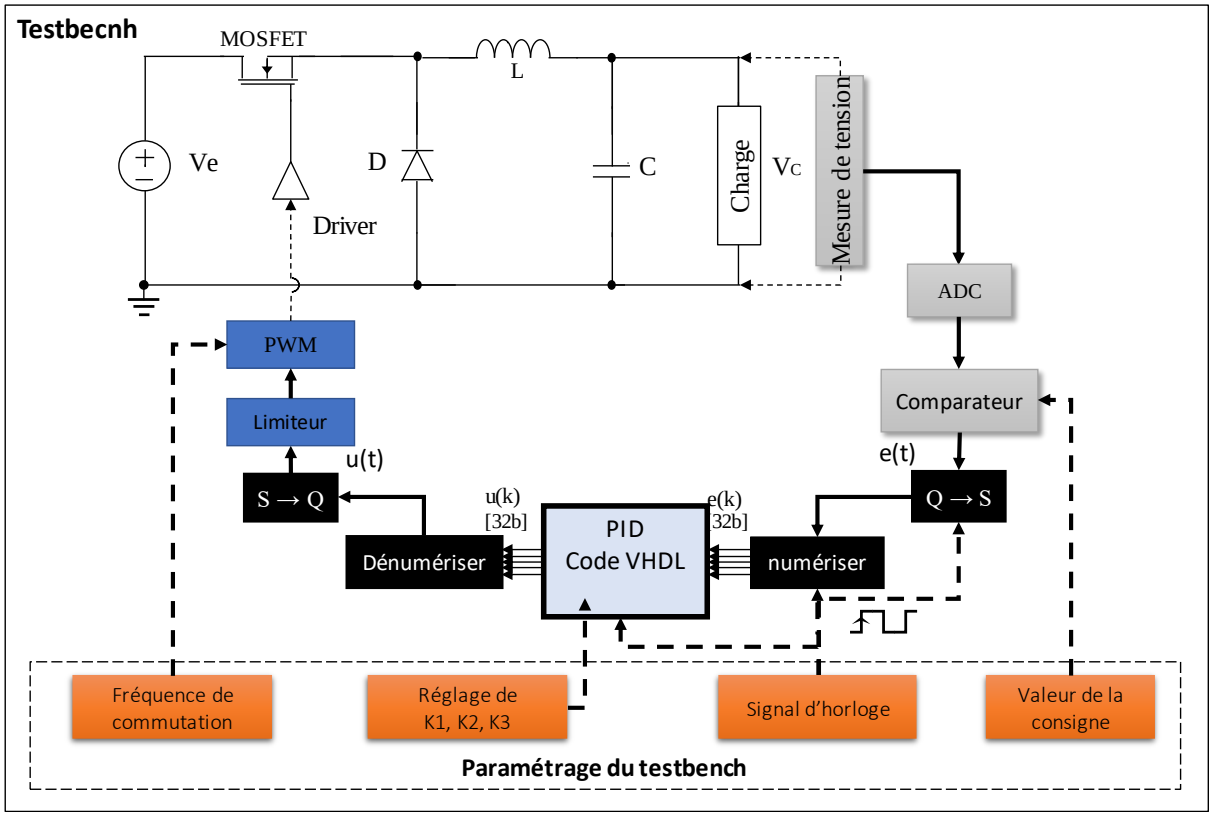


Figure 57. Vue structurelle d'un testbench

Les résultats de simulation illustrés dans la Figure 58 montrent l'évolution de la tension de sortie par rapport à la référence consignée. Ces résultats permettent de vérifier le bon fonctionnement du système, et ce, en les comparant avec les résultats de la simulation comportementale exécutée dans la phase précédente. D'un autre côté, la Figure 59 montre une partie des signaux impliqués dans le fonctionnement du PID numérique, ainsi que les données échangées à l'intérieur du contrôleur. En effet, ça permet de s'assurer de l'intégrité du flux de données entre chaque phase du traitement.

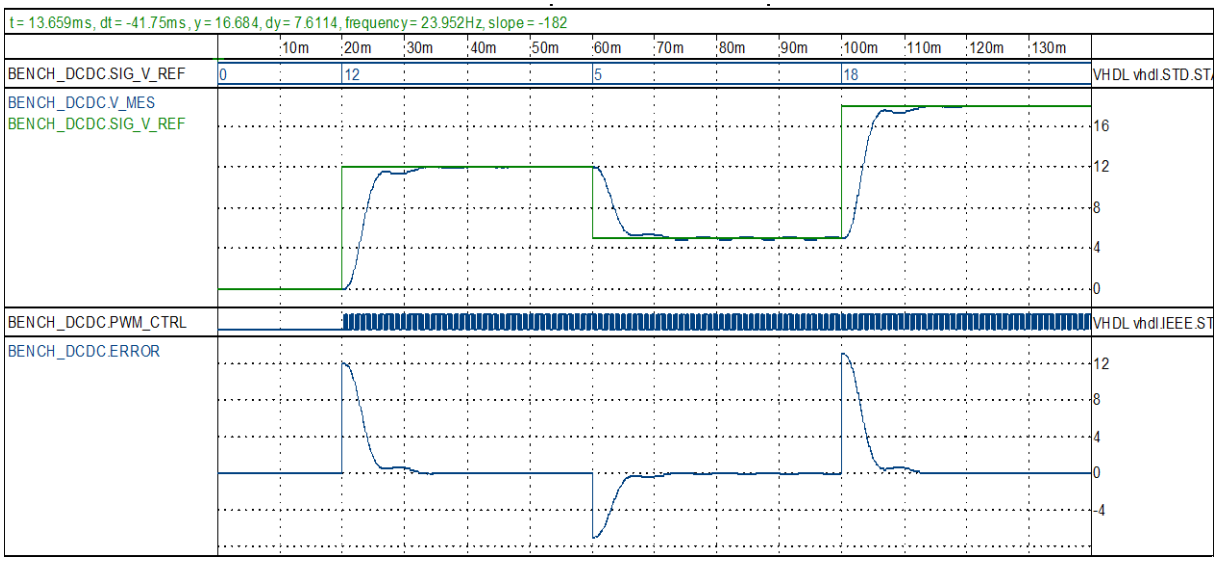


Figure 58. Tension des sortie, signal PWM et différence avec la consigne

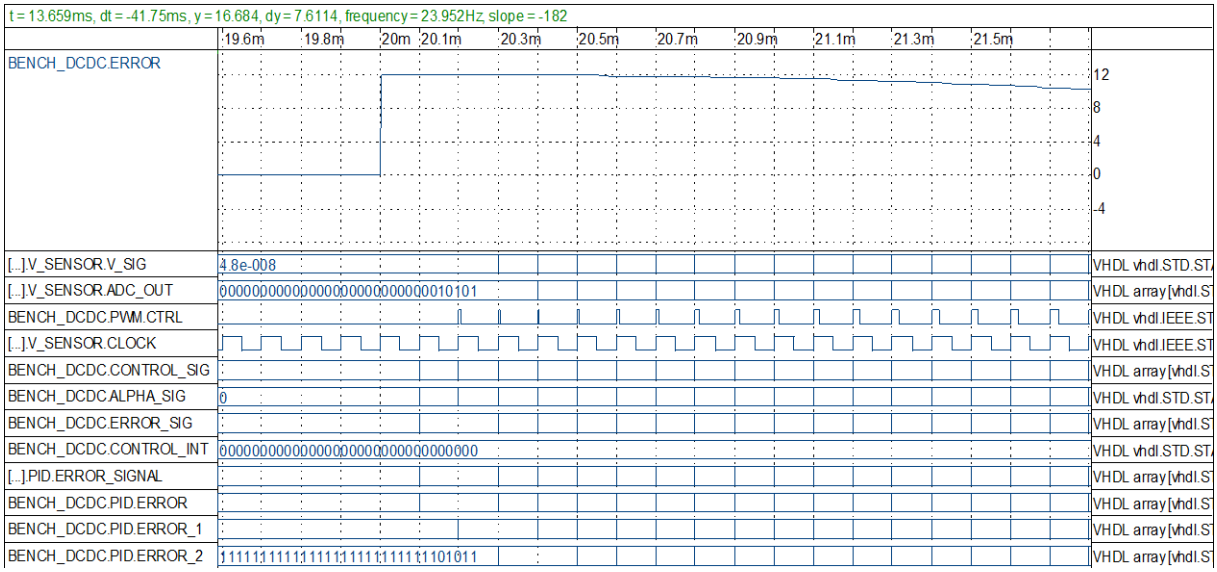


Figure 59. Zoom sur les signaux impliqués dans le fonctionnement du contrôleur PID

Ces simulations, dites transitoires, ont été effectuées sur le logiciel SMASH en mode analogique/discret avec un pas de simulation variable. Ces simulations ont permis de valider la procédure de conversion des éléments de commande dans leur forme numérique, particulièrement le format de représentation des données qui a été réalisé en virgule fixe. Par ailleurs, l'objectif de ces différentes simulations et ces différentes analyses n'est pas d'optimiser le fonctionnement du système, mais de montrer le processus d'élaboration de la commande numérique d'un système de puissance à travers l'utilisation du langage VHDL-AMS.

Le dernier modèle exécuté est le modèle destiné à être implanté dans le circuit FPGA, ce modèle est vérifié via la simulation VHDL-AMS mais pour tester son implémentation effective sur FPGA nous devons utiliser la plateforme de simulation HIL que nous avons développée.

5 IMPLEMENTATION EXPERIMENTALE ET RESULTATS DE SIMULATION HIL

Afin de valider notre approche de conception, nous devons valider le fonctionnement de la plateforme développée avec le cas d'étude que nous avons élaboré dans les sections précédentes. Pour ce faire, nous avons intégré le contrôleur numérique PID dans la partie matérielle de la plateforme HIL-AMS. Plusieurs configurations ont été mises en œuvre, nous présentons dans cette section les dernières améliorations et les derniers résultats obtenus.

Dans la Figure 60, nous présentons la répartition des éléments du système à simulé sur les différentes parties de la plateforme de simulation HIL-AMS. Dans cette configuration, nous pouvons tester le fonctionnement réel du contrôleur PID sur FPGA en utilisant le la modélisation VHDL-AMS du système dans la chaine de simulation. Il faut séparer les modules numériques des module analogique, l'interfaçages et synchronisation entre les deux parties se fait à travers l'application logicielle.

Pour pouvoir exécuter des simulation HIL, il faut déplacer les éléments de réglages de la simulation vers les modèles VHDL-AMS dans le logiciel SMASH, ce qui permet de diriger la simulation et de régler facilement les différents générateurs.

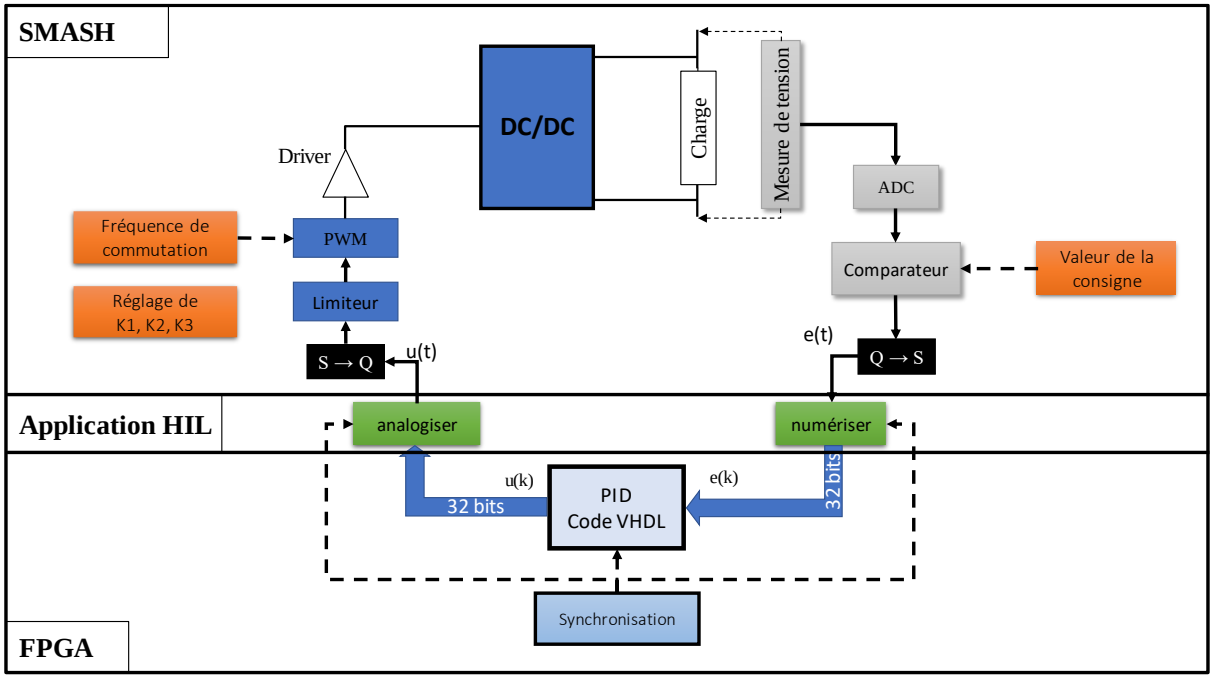


Figure 60. Vue d'ensemble de la simulation HIL-AMS

La plateforme de simulation HIL-AMS a été développée au tour du langage VHDL-AMS. Elle repose sur le logiciel de simulation SMASH, une application logicielle et une carte développement FPGA. Pour pouvoir exécuter une simulation sur cette plateforme, nous devons réaliser les tâches et configurations suivantes :

- Assembler dans le même projet le code VHDL du contrôleur PID numérique avec le code de l'application matérielle (côté FPGA), à savoir le contrôleur AXI Master, le contrôleur MAC Ethernet AXI Lite et le superviseur,
- Configurer les paramètres de l'application,
- Générer un fichier bitstream du projet pour programmer le circuit FPGA,
- Etablir la liaison physique Ethernet et la configuration des adresses réseau,
- Exécuter l'application software,
- Démarrer le logiciel SMASH,
- Ouvrir le projet de simulation du système,
- Régler les testbenchs et les générateurs de stimulus correspondant au scénario de simulation,
- Initialiser et tester la communication réseau et le protocole UDP/IP,
- Commencer la simulation.

5.1 SIMULATION HIL-AMS

Pour tester et évaluer la plateforme de simulation développée, nous allons comparer les résultats obtenus lors de la simulation VHDL-AMS du système considéré avec les résultats récupérés de la simulation HIL-AMS. Afin de comparer efficacement les deux modes de simulation, nous devons utiliser les mêmes paramètres de simulation que celles utilisées pour les simulations VHDL-AMS précédemment réalisées.

Considérons le même scénario que celui utilisé pour la simulation précédente, la Figure 61 montre les résultats de la simulation HIL-AMS, où la forme de la tension de charge est identique à la forme obtenue avec la simulation VHDL-AMS. Dans la **Error! Reference source not found.**, un agrandissement des courbes obtenues à 20 ms du temps de simulation montre une réponse retardée par rapport à la simulation VHDL-AMS. Ce retard est dû au temps de latence dans le canal de transmission et aux contraintes de synchronisation associées à la mise en œuvre de l'application côté FPGA.

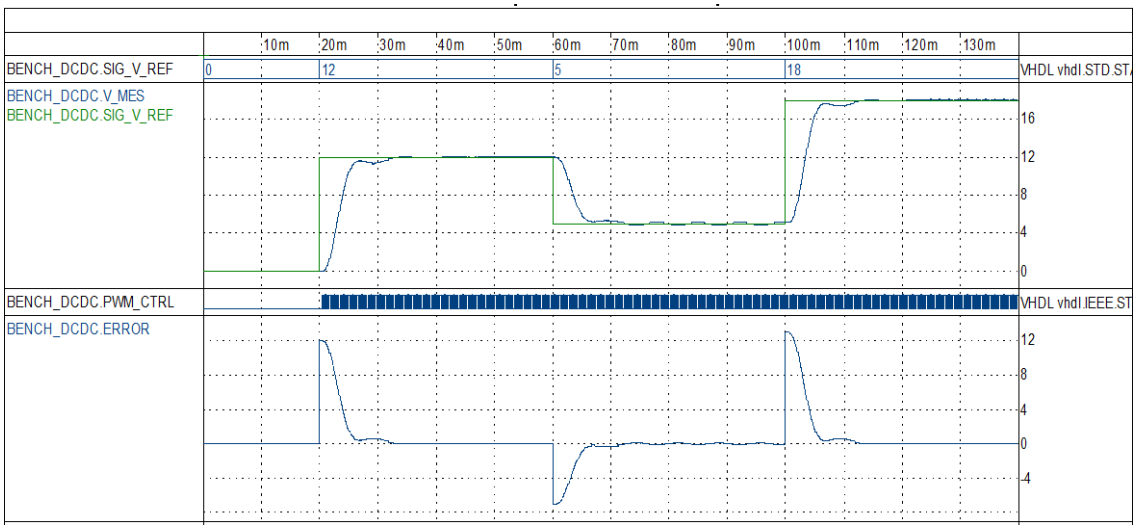


Figure 61. Tension des sortie, signal PWM et différence avec la consigne pour la simulation HIL

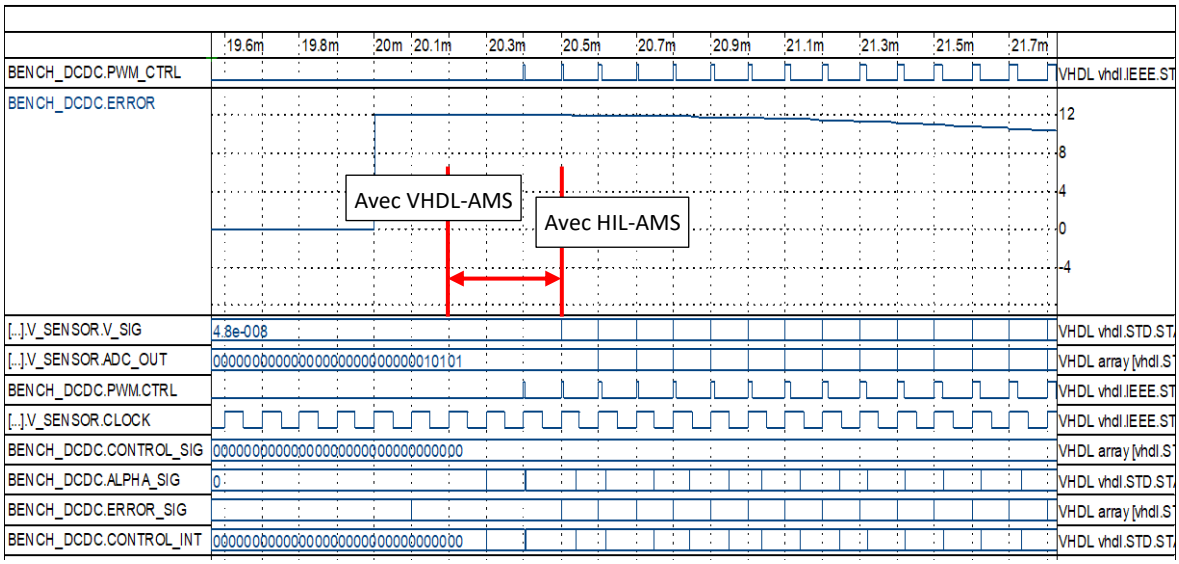


Figure 62. Zoom sur les signaux impliqués dans le fonctionnement du contrôleur PID pour la simulation HIL

5.2 RESSOURCES FPGA

Le Tableau 9 répertorie les ressources FPGA utilisées pour implémenter chaque composant de la plateforme HIL-AMS, de même que le contrôleur PID numérique. La même carte de développement que celle décrite dans le chapitre 4 est utilisée avec le même usage des ressources pour les parties de la plateforme de simulation développée.

Tableau 9. Ressources FPGA utilisées pour les différents modules utilisés

Modules	**Bascule**	**LUT**	**Bloc DSP**
Ressources disponibles	11440	5720	16
AXI_Ethernet_lite	492 [4,3%]	494 [8,6]	0
Supervisor+AXI Master	197 [1,7%]	220 [3.8]	0
PID_dig	127 [1,1%]	217 [3,8]	12[75%]
Global	816 [7.1%]	931 [16.2]	12[75%]

Les résultats expérimentaux sont obtenus pour un débit de transfert de données à 100 Mb/s avec des vecteurs de données de 32 bits. En ce qui concerne les ressources utilisées pour l'implémentation de toute l'application FPGA, ils sont les mêmes que ceux mentionnés dans le tableau 3 du chapitre 4. Le module *PID_dig* représente la partie du contrôleur numérique L'utilisation globale de LUT ne représente que 16,2% des LUT disponibles et l'intégration du contrôleur PID ne représente que 3,8%. Ces résultats montrent qu'il y a suffisamment de ressources pour remplacer le PID numérique par un contrôleur plus complexe et plus performant.

5.3 PERFORMANCES TEMPORELLES

Pour mettre en évidence les avantages de la mise en œuvre de notre approche en termes des performances temporelles, nous avons exécuté plusieurs scénarios de simulation en mode logiciel

(VHDL-AMS uniquement) et en mode de simulation HIL-AMS avec un pas de simulation variable. Les temps de simulation mesurés ont été effectués sur un ordinateur avec un processeur Intel Core i5 cadencé à une fréquence de 2,5 GHz avec 4 Go de mémoire.

Dans cette section, nous nous intéressons à la comparaison de différents résultats temporels, essentiellement le temps d'usage du CPU (durée d'utilisation du CPU) et le temps d'usage du noyau logique, ce temps indique la durée d'utilisation du CPU pour résoudre la partie discrète du système via le moteur de simulateur à temps discret.

Le Tableau 10 résume le temps d'usage du CPU et le temps d'usage du noyau logique pour différents scénarios exécutés. Ces scénarios permettent de simuler le même système pour différentes durées de simulation et pour différentes fréquences de commutation, ce qui implique un nombre différent de points de simulation analogiques et discrets qui nécessitent une utilisation nuancée des ressources matérielles.

Tableau 10. Temps consommé pour différents scénarios de simulation.

	Mode de simulation	Temps CPU total	Temps noyau logique	Amélioration du temps de simulation
Scenario 1 140ms 10kHz	Logiciel	7,83 s	5,38 s	83,66% 6,55 s
	HIL_AMS	1,28 s	0,109 s	
Scenario 2 1000ms 10kHz	Logiciel	39,14 s	30,735s	75% 29,35 s
	HIL_AMS	9,79 s	0,733 s	
Scenario 3 140ms 50kHz	Logiciel	29,22 s	21,62s	80,10% 23,52 s
	HIL_AMS	5,70 s	0,608 s	
Scenario 4 1000ms 50kHz	Logiciel	224,7 s	182,39 s	82,45% 184,48 s
	HIL_AMS	39,42 s	3,76 s	
Scenario 5 2000ms 5kHz	Logiciel	42,23 s	35,15 s	73,36% 30,98 s
	HIL_AMS	12,25 s	1,14 s	
Scenario 6 5000ms 5kHz	Logiciel	68,56 s	63,73 s	72,61% 49,78 s
	HIL_AMS	18,78 s	1,36 s	
Scenario 7 2000ms 20kHz	Logiciel	148,72 s	132,32 s	82,76% 123,08 s
	HIL_AMS	25,64 s	2,53 s	
Scenario 8 5000ms 20kHz	Logiciel	356,51 s	312,92 s	83,16% 296,46 s
	HIL_AMS	60,05 s	4,88 s	

Nous remarquons que le temps de simulation globale s'est considérablement amélioré d'environ 80%, vu que les logiciels de simulation mixte allouent de grandes ressources de calcul pour le noyau de simulation discret. Dans notre approche, la simulation numérique est exécutée exclusivement par le dispositif FPGA, ce qui libère le logiciel SMASH et le processeur pour exécuter seulement la partie analogique.

D'autre part, nous avons mesuré le temps de latence pour différents cas de simulation. Nous avons noté que le temps de latence pour la transmission réseau et le traitement des données dans le circuit FPGA est d'environ 55 µs et le temps de latence pour le traitement logiciel est d'environ 2,5 ms. Toutefois, ces performances temporelles dépendent de l'ordinateur, des paramètres de simulation, de l'interface réseau et de la complexité du système.

6 CONCLUSION

Dans ce chapitre, nous avons illustré à travers un exemple d'application les étapes de conception et de modélisation, en se basant sur la méthodologie de conception élaborée, où les étapes de modélisation et de transformation de modèles VHDL-AMS en modèles VHDL synthétisable sont explicités. Un exemple de convertisseur statique de puissance DC/DC dévolteur commandé par un régulateur PID numérique est utilisé. Après la modélisation comportementale des différents composants du système, nous avons exécuté différentes simulations en utilisant le langage VHDL-AMS seul dans les premières phases de vérification, ensuite en cosimulation avec le langage VHDL. Pour finir, nous avons réalisé des simulations Hardware-In-the-Loop en utilisant la plateforme développée et en intégrant la commande numérique dans un circuit FPGA,

A travers cet exemple d'application, nous avons évalué les performances de la plateforme développée et son apport vis-à-vis de la conception des systèmes multidisciplinaires. L'application étudiée donne un aperçu des capacités de cette méthode avec de multiples avantages en termes de conception. Les tests effectués montrent une réduction significative du temps de simulation avec un usage des ressources FPGA assez faible, le composant FPGA utilisé est un composant d'entrée de gamme ce qui signifie que le coût est assez faible en comparaison avec les plateformes commerciales de simulation Hardware-In-the-Loop existantes.

L'utilisation de matériels et logiciels de conception plus performants permettra de réduire les latences de communication et de traitement, dans certaines applications ça pourrait mener au développement d'un processus de simulation Hardware-In-the-Loop temps réel.

Conclusions et perspectives

CONCLUSIONS ET PERSPECTIVES

Notre objectif dans cette thèse était de développer et de valider une méthodologie de conception à partir des langages VHDL et VHDL-AMS pour la conception des systèmes multi-technologiques. Nous avons centré notre étude sur la conception des systèmes de conversion de la puissance électrique, constitué d'une partie électrique et une partie de commande numérique.

Pour mener à bien cette étude nous avons organisé notre travail en trois phases de développement correspondant aux objectifs de thèse :

1. Etablir un processus de développement adapté à la problématique de conception des systèmes multi-technologiques, en se basant sur l'utilisation du langage de description matériel VHDL-AMS.
2. Elaborer une plateforme pour la simulation Hardware-In-the-Loop basée sur les circuits FPGA, et en faire un support pratique de développement qui accompagne la méthodologie développée, tout en profitant de la puissance du langage VHDL-AMS.
3. Valider les outils méthodologiques et les outils de simulation développés à travers des cas d'étude tirés du domaine du génie électrique.

La réalisation de ces objectifs a nécessité l'exploration de plusieurs thématiques de recherche qui concerne la conception et l'ingénierie des systèmes. Nous avons ainsi abordé certaines méthodologies et techniques de conception d'un côté, et de l'autre côté nous avons appréhendé les différentes problématiques qui concernent, en autres, la modélisation, la simulation et la vérification.

Ainsi, dans le **chapitre 2** nous avons réalisé un état de l'art des pratiques de conception des systèmes multidisciplinaires, en nous appuyons sur une rétrospective de la démarche générale de conception. Pour ce faire, nous avons présenté les approches visant à répondre aux problématiques de la conception multi-abstraction et multi-niveaux. Dans ce contexte, nous avons mis en exergue les outils et les méthodologies de conception système telles que les méthodologies basées sur les modèles. De plus, nous avons montré l'importance des phases validation et de vérification qui permettent d'atteindre les spécifications d'exigence rapidement et efficacement.

Aussi, nous avons rappelé l'importance de la phase de modélisation dans le cycle de conception des systèmes multidisciplinaires, ainsi une étude non exhaustive des outils de modélisation et de simulation qui sont en concordance avec nos objectifs a été donnée. Cela nous a permis de présenter différentes techniques de simulation qui permettent d'analyser et d'évaluer des systèmes complexes de manière plus pratique et plus efficace, ces techniques utilisent des éléments matériels dans une boucle de simulation pour tester et valider une partie du système à concevoir.

Dans le **chapitre 3**, nous avons présenté notre démarche méthodologique fondée sur la conception multi-domaine. Cette démarche permet de prendre en considération l'hétérogénéité et la complexité architecturale des systèmes multidisciplinaires aux prémices de leur conception. Pour la mise en œuvre de cette démarche, nous nous sommes basés sur trois piliers méthodologiques qui sont un processus, une méthode et un outil de conception. Ces trois piliers sont interdépendants et ils sont centrés autour de la modélisation et de simulation par le langage VHDL-AMS.

A cet effet, nous avons détaillé la démarche proposée, en particulier les éléments-clés du processus de conception tels que la transformation des descriptions de modèle VHDL-AMS vers des descriptions VHDL synthétisable, ainsi que la vérification par simulation à travers des concepts de l'ingénierie basée sur le modèle. Dans ce chapitre nous avons montré l'importance du langage VHDL-AMS dans notre travail en le comparant à d'autres langages de modélisation. Cela nous a permis de positionner notre choix sur le langage de description matériel VHDL-AMS pour la modélisation multi-domaine des systèmes étudiés, où le langage VHDL-AMS s'est révélé parfaitement adapté à nos objectifs de conception.

Dans le **chapitre 4**, nous nous sommes focalisés sur la construction d'une plateforme de simulation Hardware-In-the-Loop « HIL-AMS », en mettant l'accent sur les choix architecturaux de cette plateforme ainsi que sur le choix des outils logiciels et matériels mis en œuvre. Dans un système de conversion de la puissance, cette plateforme permet de réaliser des simulations C-HIL. Ainsi, le système de puissance est simulé entièrement en VHDL-AMS et le sous-système de commande numérique est implémenté sur FPGA.

Cette plateforme repose sur l'utilisation du logiciel de simulation SMASH de Dolphin Integration, nous avons ainsi exposé les avantages de ce logiciel en ce qui concerne la modélisation multi-langage et multi-plateforme, comparé à d'autres logiciels de simulation VHDL-AMS, il reste le plus performant et le plus conforme aux standards IEEE 1076.1 du langage VHDL-AMS.

Une application logicielle est réalisée pour relier et contrôler le flux de données nécessaire à une simulation C-HIL. La synchronisation, la transformation des formats de donnée et l'interfaçage des parties associées à cette technique de simulation sont quelques-unes des tâches que cette application doit exécuter.

Par ailleurs, nous avons décrit le fonctionnement de la plateforme HIL-AMS en détaillant les différents éléments de son architecture logicielle et matérielle. Côté matériel, nous avons utilisé un FPGA d'entrée de gamme qui s'est avéré suffisant pour cette application, dans le souci d'optimiser les ressources consommées et de rendre notre implémentation la plus générique possible, nous avons combiné l'utilisation de l'interface extensible AXI-lite avec des codes VHDL dédiés pour atteindre de meilleures performances au moindre coût.

Pour valider l'apport de la méthodologie proposée et de la plateforme mise en œuvre, dans le **chapitre 5** nous avons illustré par un exemple d'application les étapes de conception et de modélisation, en se basant sur l'approche élaborée où les étapes de modélisation et de transformation de modèles VHDL-AMS en modèles VHDL synthétisable sont explicités, en même temps des simulations sont exécutées en utilisant le langage VHDL-AMS seul dans les premières phases de

vérification, ensuite en cosimulation avec le langage VHDL. Pour finir, nous avons testé la plateforme de simulation en réalisant une simulation hardware avec un exemple de commande de convertisseur statique de puissance, la commande étant implantée dans le circuit FPGA et le reste du système simulé dans le logiciel SMASH en utilisant des modèles VHDL-AMS. Ainsi, nous avons évalué les performances de la plateforme et son apport vis-à-vis de la conception des systèmes multidisciplinaires.

A l'issue de ce travail de thèse, plusieurs perspectives se dégagent :

- Il est nécessaire de poursuivre les tests expérimentaux pour améliorer l'apport de la plateforme développée, des cas d'étude plus poussés sont à prévoir dans d'autres domaines technologiques tels que les MEMS, la mécatronique ou la robotique. Ceci afin de valider les performances et l'efficacité de cette nouvelle technique de simulation C-HIL.

- De par les résultats que nous avons déjà obtenus, Il est possible d'utiliser d'autres langages de modélisation multi-domaines tel que le langage SystemC-AMS, Verilog-AMS ou Modelica, cela peut ouvrir la voie vers l'intégration dans le même environnement de conception d'autres approches de modélisation et de simulation multi-langage et multi-plateforme.

- Afin d'améliorer les performances de la plateforme HIL-AMS, nous envisageons d'utiliser d'autres interfaces de communication come le PCI-Express qui permettra d'accélérer la simulation, en réduisant les temps de réponse de la carte de développement FPGA. Ceci permettra d'explorer la possibilité de mise en œuvre d'une technique de simulation Hardware-In-the-Loop temps-réel.

- Actuellement, de nouvelles solutions d'interfaçage logiciel sont développées pour la cosimulation telle que l'interface FMI (Functional Mock-up Interface), ces solutions sont à considérer pour l'interfaçage avec d'autres logiciels de simulation multi-domaine en remplacement au logiciel SMASH. Ces alternatives permettent de se soustraire du logiciel SMASH et d'étendre le champ des opportunités offertes à la simulation Hardware-In-the-Loop multi-domaine.

Références bibliographiques

REFERENCES BIBLIOGRAPHIQUES

[1] J. Verries, « Approche pour la conception de systèmes aéronautiques innovants en vue d'optimiser l'architecture : application au systèmes portes passagers », thesis, Toulouse 3, 2010.

[2] A. Sangiovanni-Vincentelli, « Is a Unified Methodology for System-Level Design Possible? », *IEEE Design Test of Computers*, vol. 25, n° 4, p. 346-357, juill. 2008, doi: 10.1109/MDT.2008.104.

[3] O. Balci, « Validation, verification, and testing techniques throughout the life cycle of a simulation study », *Ann Oper Res*, vol. 53, n° 1, p. 121-173, déc. 1994, doi: 10.1007/BF02136828.

[4] D. Teegarden, « DO-254 Compliant Design and Verification with VHDL-AMS », Mentor Graphics Corporation, Technical White Paper mentorpaper_36582, 2007.

[5] C. S. Wasson, *System Analysis, Design, and Development: Concepts, Principles, and Practices*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2005.

[6] INCOSE, *Systems engineering handbook: a guide for system life cycle processes and activities*. International Council on Systems Engineering., 2015.

[7] IEEE, « IEEE Guide for Developing System Requirements Specifications », IEEE, 1998. doi: 10.1109/IEEESTD.1998.88826.

[8] NASA, *NASA Systems Engineering Handbook: NASA/SP-2016-6105 Rev2*. Place of publication not identified: 12th Media Services, 2018.

[9] D. A. U. DOD, *Systems Engineering Fundamentals, January 2001*. Fort Belvoir: Defense Acquisition University, 2001.

[10] D. Foures, « Validation de modèles de simulation », phd, Université de Toulouse, Université Toulouse III - Paul Sabatier, 2015.

[11] D. M. Buede, *The Engineering Design of Systems*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2009.

[12] A. Warniez, « Métriques d'intégration pour le choix d'architectures dans la conception des systèmes mécatroniques », thesis, Châtenay-Malabry, Ecole centrale de Paris, 2015.

[13] S. Friedenthal, A. Moore, et R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. Elsevier, 2011.

[14] J. A. Estefan, « Survey of model-based systems engineering (MBSE) methodologies », *Incose MBSE Focus Group*, vol. 25, n° 8, p. 1-12, 2007.

[15] INCOSE, « SYSTEMS ENGINEERING VISION 2020 ». International Council on Systems Engineering (INCOSE), sept. 2007, [En ligne]. Disponible sur : https://sdincose.org/wp-content/uploads/2011/12/SEVision2020_20071003_v2_03.pdf.

[16] B. Nastov, « Contribution à une méthode outillée pour la conception de langages de modélisation métier interopérables, analysables et prouvables pour l'Ingénierie Système basée sur des Modèles », thesis, Montpellier, 2016.

[17] L. Delligatti, *SysML Distilled: A Brief Guide to the Systems Modeling Language*, 1 edition. Upper Saddle River, NJ: Addison-Wesley Professional, 2013.

[18] J. Kößler et K. Paetzold, « Integration of MBSE into existing development processes-Expectations and challenges », présenté à DS 87-3 Proceedings of the 21st International Conference on Engineering Design (ICED 17) Vol 3: Product, Services and Systems Design, Vancouver, Canada, 21-25.08. 2017, 2017, p. 051-060.

[19] P. Leserf, « Optimisation de l'architecture de systèmes embarqués par une approche basée modèle », Université de Toulouse, 2017.

[20] S. Friedenthal, R. Griego, et M. Sampson, « INCOSE Model Based Systems Engineering (MBSE) Initiative », janv. 2009.

[21] A. Maheshwari, « Industrial Adoption of Model-Based Systems Engineering: Challenges and Strategies », MSAA, Purdue University, West Lafayette, Indiana, 2015.

[22] R. Cloutier, *Systems Engineering Simplified*, 1 edition. Boca Raton: Routledge, 2015.

[23] OMG, « OMG | Object Management Group », 2019. https://www.omg.org/.

[24] K. Fåhraeus, « Enhancement of the Mechatronic Development Process with Software in the loop Simulation: An embedded control case study », MS, KTH, School of Industrial Engineering and Management, STOCKHOLM, SWEDEN, 2015.

[25] D. Guihal, « Modélisation en langage VHDL-AMS des systèmes pluridisciplinaires », thesis, Toulouse 3, 2007.

[26] DOD, « DoD Modeling and Simulation (M&S) Glossary », US Department of Defense, Arlington, VA, USA, DOD-5000.59-M, janv. 1998. Consulté le: juill. 27, 2019. [En ligne]. Disponible sur: https://apps.dtic.mil/docs/citations/ADA349800.

[27] G. Bellinger, « Modeling & Simulation - An Introduction », 2004. http://www.systems-thinking.org/modsim/modsim.htm (consulté le mars 01, 2018).

[28] D. Dori, *Object-Process Methodology: A Holistic Systems Paradigm*. Berlin Heidelberg: Springer-Verlag, 2002.

[29] IEEE, « IEEE Standard Glossary of Modeling and Simulation Terminology », IEEE, IEEE Std 610.3-1989, mai 1989.

[30] AFIS, « Typologie des modèles », 2010. http://www.afis.fr (consulté le juill. 27, 2018).

[31] ISO/IEC/IEEE, « ISO/IEC/IEEE International Standard - Systems and software engineering – System life cycle processes », *ISO/IEC/IEEE 15288 First edition 2015-05-15*, p. 1-118, mai 2015, doi: 10.1109/IEEESTD.2015.7106435.

[32] M. J. Ryan et L. S. Wheatcraft, « On the use of the terms verification and validation », présenté à INCOSE International Symposium, 2017, vol. 27, p. 1277-1290.

[33] S. Prat, « Intégration de techniques de vérification par simulation dans un processus de conception automatisée de contrôle commande », Thèse, Université de Lorient, 2017.

[34] T. Ören et B. Waite, « Modeling and simulation body of knowledge index: an invitation for the final phases of its preparation », *SCS M&S Magazine*, vol. 4, n° 4, 2010.

[35] R. Sargent, « December."Verification and validation of simulation models" », présenté à Proceedings of the 2011 Winter Simulation Conference, Phoenix, Arizona, 2011, p. 183-198.

[36] S. Robinson, *Simulation: The Practice of Model Development and Use*, 2nd ed. 2014. Houndmills, Basingstoke, Hampshire, UK ; New York, NY: Red Globe Press, 2014.

[37] N. Seddari, « Outils formels et opérationnels pour la modélisation et la simulation des systèmes complexes », Doctorat LMD, Université 20 Août 1955 Skikda, Skikda, 2015.

[38] T. Paris, « Modélisation de systèmes complexes par composition », 2019.

[39] V. Savicks, « Integrating Formal Verification and Simulation of Hybrid Systems », phd, University of Southampton, 2016.

[40] G. Nicolescu, H. Boucheneb, L. Gheorghe, et F. Bouchhima, « Methodology for efficient design of continuous/discrete-events co-simulation tools », *High Level Simulation Languages and Applications-HLSLA. SCS, San Diego, CA*, p. 172-179, 2007.

[41] M. Ben Ayed, « Environnemnet de Co-Simulation / Emulation des systèmes Continus / Discrets », Doctorat, École Nationale d'Ingénieurs de Sfax, 2013.

[42] L. Gheorghe, « Continuous/Discrete Co-Simulation Interfaces from Formalization to Implementation », phd, École Polytechnique de Montréal, 2009.

[43] F. Bouchhima, G. Nicolescu, M. Aboulhamid, et M. Abid, « Discrete-continuous simulation model for accurate validation in component-based heterogeneous SoC design », in *16th IEEE International Workshop on Rapid System Prototyping (RSP'05)*, juin 2005, p. 181-187, doi: 10.1109/RSP.2005.22.

[44] N. Schetinin, N. Moriz, B. Kumar, A. Maier, S. Faltinski, et O. Niggemann, « Why do verification approaches in automation rarely use HIL-test? », in *2013 IEEE International Conference on Industrial Technology (ICIT)*, Cape Town, févr. 2013, p. 1428-1433, doi: 10.1109/ICIT.2013.6505881.

[45] A. Hoffmann, T. Kogel, et H. Meyr, « A framework for fast hardware-software co-simulation », in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, Munich, Germany, 2001, p. 760-764, doi: 10.1109/DATE.2001.915114.

[46] S. Alles, C. Swick, S. Mahmud, et F. Lin, « Real time Hardware-In-the-Loop vehicle simulation », in *[1992] Conference Record IEEE Instrumentation and Measurement Technology Conference*, mai 1992, p. 159-164, doi: 10.1109/IMTC.1992.245158.

[47] S. Bacha, J. Guiraud, D. Roye, L. Munteanu, et A. Bractcu, « Méthodologie de simulation temps reel Hardware-In-the-Loop–application aux systèmes éoliens », présenté à MOSIM'06, Actes de la 6ième Conférence Francophone de Modélisation et Simulation, 2006.

[48] R. Isermann et J. Scha, « Hardware-In-the-Loop simulation for the design and testing of engine-control systems », *Control Engineering Practice*, p. 11, 1999.

[49] M. A. A. Sanvido, « Hardware-In-the-Loop simulation framework », ETH. Zürich, Zürich, 2002.

[50] C. Köhler, *Enhancing Embedded Systems Simulation: A Chip-Hardware-In-the-Loop Simulation Framework*. Vieweg+Teubner Verlag, 2011.

[51] N. Pallo, T. Foulkes, T. Modeer, E. Fonkwe, P. Gartner, et R. C. N. Pilawa-Podgurski, « Hardware-In-the-Loop co-design testbed for flying capacitor multilevel converters », in *2017 IEEE Power and Energy Conference at Illinois (PECI)*, févr. 2017, p. 1-8, doi: 10.1109/PECI.2017.7935758.

[52] W. Ren *et al.*, « Interfacing Issues in Real-Time Digital Simulators », *IEEE Trans. Power Delivery*, vol. 26, n° 2, p. 1221-1230, avr. 2011, doi: 10.1109/TPWRD.2010.2072792.

[53] W. Ren, « Accuracy Evalaution [evaluation] of Power Hardware-In-the-Loop (PHIL) Simulation », Florida State University, 2007.

[54] M. O. O. Faruque et V. Dinavahi, « Hardware-In-the-Loop Simulation of Power Electronic Systems Using Adaptive Discretization », *IEEE Transactions on Industrial Electronics*, vol. 57, n° 4, p. 1146-1158, avr. 2010, doi: 10.1109/TIE.2009.2036647.

[55] M. Dagbagi, « Simulation temps-réel embarquée de systèmes électriques au moyen de FPGA », thesis, Cergy-Pontoise, 2015.

[56] I. D. Landau et G. Zito, *Digital Control Systems: Design, Identification and Implementation*. London: Springer-Verlag, 2006.

[57] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, et M. W. Naouar, « FPGAs in industrial control applications », *IEEE Transactions on Industrial informatics*, vol. 7, n° 2, p. 224-243, 2011.

[58] E. Monmasson et M. N. Cirstea, « FPGA design methodology for industrial control systems—A review », *IEEE transactions on industrial electronics*, vol. 54, n° 4, p. 1824-1842, 2007.

[59] Y. Hu, « Exploring formal verification methodology for FPGA-based digital systems », Sandia National Laboratories, New Mexico, California, SAND2012-7926, 2012.

[60] T. M. Bhatt et D. McCain, « Matlab as a development environment for FPGA design », présenté à Proceedings of the 42nd annual Design Automation Conference, 2005, p. 607-610.

[61] S. Mittal, S. Gupta, et S. Dasgupta, « System generator: The state-of-art FPGA design tool for dsp applications », présenté à Third International Innovative Conference On Embedded Systems, Mobile Communication And Computing (ICEMC2 2008), 2008, p. 187-190.

[62] Y. P. Siwakoti et G. E. Town, « Design of FPGA-controlled power electronics and drives using MATLAB Simulink », présenté à 2013 IEEE ECCE Asia Downunder, 2013, p. 571-577.

[63] C. Eckl, M. Brandstätter, et J. Stjepandic, « Using the" Model-based Systems Engineering" Technique for Multidisciplinary System Development. », présenté à ISPE CE, 2015, p. 100-109.

[64] H. Kim, D. Fried, et P. Menegay, « Connecting SysML models with engineering analyses to support multidisciplinary system development », présenté à 12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2012, p. 5632.

[65] V. M. MONTHE DJIADEU, « Développement des systèmes logiciels par transformation de modèles : application aux systèmes embarqués et à la robotique », thesis, Brest, 2017.

[66] H. Elmqvist, S. E. Mattsson, et M. Otter, « Modelica-a language for physical system modeling, visualization and interaction », présenté à Proceedings of the 1999 IEEE international symposium on computer aided control system design (Cat. No. 99TH8404), 1999, p. 630-639.

[67] P. Fritzson et V. Engelson, « Modelica—A unified object-oriented language for system modeling and simulation », présenté à European Conference on Object-Oriented Programming, 1998, p. 67-90.

[68] E. Christen et K. Bakalar, « VHDL-AMS-a hardware description language for analog and mixed-signal applications », *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, n° 10, p. 1263-1272, oct. 1999, doi: 10.1109/82.799677.

[69] P. Frey et D. O'Riordan, « Verilog-AMS: Mixed-signal simulation and cross domain connect modules », présenté à Proceedings 2000 IEEE/ACM International Workshop on Behavioral Modeling and Simulation, 2000, p. 103-108.

[70] K. Kundert et O. Zinke, *The designer's guide to Verilog-AMS*. Springer Science & Business Media, 2006.

[71] accellera, « SystemC », 2016. https://www.accellera.org/downloads/standards/systemc (consulté le oct. 01, 2019).

[72] H. Al-Junaid et T. Kazmierski, « Analogue and mixed-signal extension to SystemC », *IEE Proceedings-Circuits, Devices and Systems*, vol. 152, n° 6, p. 682-690, 2005.

[73] C. Grimm, M. Barnasconi, A. Vachoux, et K. Einwich, « An introduction to modeling embedded analog/mixed-signal systems using SystemC AMS extensions », présenté à DAC2008 International Conference, 2008, vol. 23.

[74] IEEE, « Std 1076-2008, IEEE Standard VHDL Language Reference Manual », IEEE, IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002), 2008.

[75] IEEE, « IEEE Standard VHDL Analog and Mixed-Signal Extensions », IEEE, IEEE std 1076.1-2017, janv. 2018.

[76] A. Laouamri, F. Djahli, A. Rabhi, A. Benhamadouche, A. Ballouti, et A. Bendjadou, « A Virtual Prototype of Proton Exchange Membrane Fuel Cell Using VHDL-AMS Language », *Journal of fuel cell science and technology*, vol. 6, n° 2, p. 024502, 2009.

[77] A. Rezgui, L. Gerbaud, et B. Delinchant, « VHDL-AMS electromagnetic automatic modeling for system simulation and design », *IEEE Transactions on Magnetics*, vol. 50, n° 2, p. 1013-1016, 2014.

[78] L. Zerioul, M. Ariaudo, et E. Bourdel, « RF transceiver and transmission line behavioral modeling in VHDL-AMS for wired RFNoC », *Analog Integrated Circuits and Signal Processing*, vol. 92, n° 1, p. 103-114, 2017.

[79] Y.-A. Chapuis, L. Zhou, H. Fujita, et Y. Hervé, « Multi-domain simulation using VHDL-AMS for distributed MEMS in functional environment: Case of a 2D air-jet micromanipulator », *Sensors and Actuators A: Physical*, vol. 148, n° 1, p. 224-238, 2008.

[80] L. A. Barragán, D. Navarro, J. Acero, I. Urriza, et J. M. Burdío, « FPGA implementation of a switching frequency modulation circuit for EMI reduction in resonant inverters for induction heating appliances », *IEEE Transactions on Industrial Electronics*, vol. 55, n° 1, p. 11-20, 2008.

[81] Y.-A. Chapuis, L. Zhou, D. Casner, H. Ai, et Y. Hervé, « FPGA-in-the-loop for control emulation of distributed MEMS simulation using VHDL-AMS », présenté à 2010 First Workshop on Hardware and Software Implementation and Control of Distributed MEMS, 2010, p. 92-99.

[82] S. Jovanovic, P. Poure, S. Saadate, et S. Weber, « Design of a fully digital controller for a shunt three-phase active filter using VHDL-AMS language », *International Journal of Electronics*, vol. 95, n° 10, p. 1055-1071, 2008.

[83] F. Pêcheux, C. Lallement, et A. Vachoux, « VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems », *IEEE transactions on Computer-Aided design of integrated Circuits and Systems*, vol. 24, n° 2, p. 204-225, 2005.

[84] P. J. Ashenden, G. D. Peterson, et D. A. Teegarden, *The System Designer's Guide to VHDL-AMS: Analog, Mixed-Signal, and Mixed-Technology Modeling*, Pap/Cdr. San Francisco, Calif: Morgan Kaufmann, 2002.

[85] A. Rezgui, « Interopérabilité de modèles dans le cycle de conception des systèmes électromagnétiques via des supports complémentaires : VHDL-AMS et composants logiciels ICAr », thesis, Grenoble, 2012.

[86] H. Boussetta, « Modélisations multi-physiques et simulations globales de systèmes autonomes sur puce », Thèse, Institut Polytechnique De Grenoble, 2010.

[87] F. Bouquet, J.-M. Gauthier, A. Hammad, et F. Peureux, « Transformation of SysML structure diagrams to VHDL-AMS », présenté à 2012 Second Workshop on Design, Control and Software Implementation for Distributed MEMS, 2012, p. 74-81.

[88] J.-M. Gauthier, « Combining Discrete and Continuous Domains for SysML-Based Simulation and Test Generation », thesis, Besançon, 2015.

[89] D. Chaves Café, « Multi-level modeling for verification and synthesis of complex systems in a multi-physics context. », thesis, CentraleSupélec, 2015.

[90] P. Dorato et D. Petersen, « Digital Control Systems », in *Advances in Computers*, vol. 23, M. C. Yovits, Éd. Elsevier, 1984, p. 177-252.

[91] C. A. Tugui, « Design Methodology for High-performance Circuits Based on Automatic Optimization Methods. », thesis, Supélec, 2013.

[92] Dolphin Integration, « SMASH - User Manual ». Dolphin Integration, 2019, [En ligne]. Disponible sur : https://www.dolphin-design.fr/.

[93] A. Vachoux, « Modélisation de Systèmes Intégrés Analogiques et Mixtes Introduction à VHDL-AMS », EPFL, 2002.

[94] [2] Francisco, *A Problem-Oriented Approach for Dynamic Verification of Heterogeneous Embedded Systems*. KIT Scientific Publishing, 2014.

[95] T. Bollengier, « Du Prototypage à l'Exploitation d'Overlays FPGA », Ecole Doctorale MathSTIC - ED 601, 2018.

[96] ARM Ltd, « AMBA », *ARM Developer*, 2016. https://developer.arm.com/architectures/system-architectures/amba (consulté le juin 12, 2016).

[97] Xilinx Inc, « LogiCORE IP AXI Ethernet Lite MAC v2.0, Product Guide ». 2013, Consulté le : janv. 01, 2016. [En ligne]. Disponible sur: https://www.xilinx.com/products/intellectual-property/axi_ethernetlite.html.

[98] IEEE_802, « IEEE Standard for Ethernet », Standard IEEE 802.3-2018 (Revision of IEEE Std 802.3-2008), 2018.

[99] K. J. Åström et T. Hägglund, *PID controllers: theory, design, and tuning*, 2e éd. Instrument society of America Research Triangle Park, NC, 1995.

[100] V. N. R. Yaramasu et B. Wu, *Predictive Control of Wind Energy Conversion Systems*. New York: Wiley-Blackwell, 2016.

[101] A. Sanchez, A. de Castro, et J. Garrido, « A Comparison of Simulation and Hardware-in-the- Loop Alternatives for Digital Control of Power Converters », *IEEE Trans. Ind. Inf.*, vol. 8, no 3, p. 491-500, août 2012, doi: 10.1109/TII.2012.2192281.

[102] W. F. Lee, *Vhdl: Coding and Logic Synthesis With Synopsys*, First Edition, First Printing. San Diego: Academic Press Inc, 2000.

[103] C. Sisterna, « Design and Coding Style Guidelines for Synthesizable VHDL-FPGA Code ». c7technology, 2015, [En ligne]. Disponible sur: http://www.c7technology.com/docs/technical_notes/C7T_AN06_HDL_FPGA_Guidelines_V1_0.pdf.

ملخص

تتناول هذه الأطروحة المنهجيات الخاصة بتصميم ونمذجة أنظمة التكنولوجيا المتعددة، وكذلك الأدوات التي تساعد المطورين في عملية التصميم. على الرغم من وجود العديد من تقنيات التصميم، إلا أنه من الصعب دائمًا إدارة تنفيذ نظام معقد ذو تغايريه عالية ويتطلب استخدام أدوات مختلفة لدعم التصميم. لذا يصبح من الضروري وضع منهجية تبسط وتسرع عملية التطوير والتحقق. في هذا الإطار، يقترح العمل المقدم في هذه الأطروحة منهجية التصميم القائمة على النمذجة وعلى المحاكاة، وتدعم هذه المنهجية لغات النمذجة والمحاكاة VHDL و VHDL-AMS. هذه المنهجية تجعل من الممكن دمج تقنية النمذجة في مستويات مختلفة من التجريد وفي مراحل مختلفة من تصميم النظام من أجل تنفيذ عمليات التحقق بسرعة وبكفاءة. وأخيرًا، وامتدادًا للمنهجية المقترحة، فقد صممنا نظامًا أساسيًا للمحاكاة HIL مبنية على دوائر قابلة لإعادة التعديل FPGA، من أجل إرساء التحقق المتكرر والتحقق في الأنظمة متعددة التقنيات ويساعد على بناء النموذج الحقيقي. ثم اختبار المنهجية والمنصة التي تم إعدادها في هذا العمل والتحقق من إمكانياتها من خلال أمثلة تطبيقية من أنظمة تحويل الطاقة الكهربائية.

الكلمات المفتاحية: أنظمة التكنولوجيا المتعددة، منهجيات التصميم، النمذجة، المحاكاة، VHDL-AMS، VHDL، HIL، FPGA، تقنيات التحقق، أنظمة تحويل الطاقة الكهربائية.

Résumé

Cette thèse aborde les méthodologies de conception des systèmes multi-technologiques, ainsi que les outils qui assistent les développeurs dans cette démarche de conception. Bien qu'il existe une multitude de techniques de conception, il est toujours ardu de maîtriser la mise en œuvre d'un système complexe à forte hétérogénéité et qui requiert l'utilisation de différents outils d'aide à la conception. Alors, il devient indispensable de mettre en œuvre une méthodologie qui simplifie et accélère le processus de développement et de vérification. Dans ce contexte, le travail présenté dans cette thèse propose une méthodologie de conception basée sur la modélisation et sur la simulation multi-domaine, cette méthodologie a pour support les langages de description matérielle VHDL et VHDL-AMS, elle permet d'intégrer la modélisation à différents niveaux d'abstraction et aux différentes étapes du processus de conception système, afin de valider rapidement et efficacement ce dernier. Enfin, et dans le prolongement de l'approche proposée, nous avons conçu une plateforme de simulation Hardware-In-the-Loop basée sur les circuits reconfigurables FPGA, ceci afin de permettre une vérification et une validation plus affinée des systèmes à concevoir, ce qui aide à construire rapidement un prototype fonctionnel et robuste. La méthodologie et la plateforme mises en œuvre dans ce travail ont été testées et validées à travers un cas d'étude issus du domaine de la conversion de la puissance électrique.

Mots clés : Système multi-technologiques, Méthodologies de conception, Modélisation, Simulation, VHDL-AMS, VHDL, FPGA, Simulation HIL, Systèmes de conversion de puissance.

Abstract

This thesis deals with design methodologies of mixed-technology systems, as well as tools that assist developers in this design process. Although there are multitude design techniques, it is always difficult to manage the implementation of a complex system with high heterogeneity and which requires the use of different tools for design support. So, it becomes essential to put forward a methodology that simplifies and speeds up the process of development and verification. In this context, the work presented in this thesis proposes a methodology of design based on the modeling and an iterative simulation, this methodology is supported by material description languages VHDL and VHDL-AMS. This methodology integrates modeling techniques at different levels of abstraction and different stages of system design to quickly and efficiently validate the targeted system. Finally, as an extension to the proposed approach, we have designed a Hardware-in-the-loop simulation platform based on FPGA circuits, in order to enable affined verification and validation of mixed-technology systems and helps to quickly build a functional and efficient prototype. The methodology and the platform implemented in this work have been tested and validated through case studies derived from electrical power conversion systems.

Keywords: Multi-technology System, Design Methodologies, Modeling, Simulation, VHDL-AMS, VHDL, FPGA, HIL Simulation, Power Conversion Systems.