# Implémentation sur SoC des réseaux Bayésiens pour l'état de santé et la décision dans le cadre de missions de véhicules autonomes

Sara Zermani

**THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE**
*sous le sceau de l'Université Bretagne Loire*

pour obtenir le titre de
**DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE**
*Mention : Informatique*
**Ecole Doctorale MATHSTIC ED 601**


présentée par

Sara Zermani

Préparée au Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC) — CNRS, UMR 6285


# Implémentation sur SoC des réseaux Bayésiens pour l'état de santé et la décision dans le cadre de missions de véhicules autonomes

**Thèse soutenue le 21novembre 2017**
devant le jury composé de :

**Pierre Boulet**
Professeur des Universités, Université de Lille/ rapporteur

**Damien Querlioz**
Chargé de Recherche CNRS, Université Paris-Sud/ rapporteur

**Olivier Romain**
Professeur des Universités, Université de Cergy-Pontoise/ examinateur

**Jean-Philippe Diguet**
Directeur de recherche CNRS, Université de Bretagne Sud/ examinateur

**Reinhardt Euler**
Professeur des Universités, Université de Bretagne Occidentale/ directeur de thèse

**Catherine Dezan**
Maître de conférences, Université de Bretagne Occidentale/ co-directrice de thèse

**Remerciements**

Je tiens à remercier les personnes qui ont contribué à la réussite de cette thèse.

Tout d'abord, Catherine Dezan et Reinhardt Euler, qui ont su assurer l'encadrement avec fulgurance et bienveillance.

Je remercie également chaque membre du jury, Pierre Boulet, Damien Querlioz, Olivier Romain et Jean-Philippe Diguet, d'avoir accepté d'évaluer mon travail et de me donner l'opportunité de le défendre.

J'adresse des remerciements à toutes les personnes que j'ai pu côtoyer ces dernières années et qui ont été passionnées par le sujet de mes recherches.

# Resumé

Les véhicules autonomes, tels que les drones, sont utilisés dans différents domaines d'application pour exécuter des missions simples ou complexes. D'un coté, ils opèrent généralement dans des conditions environnementales incertaines, pouvant conduire à des conséquences désastreuses pour l'humain et l'environnement. Il est donc nécessaire de surveiller continuellement l'état de santé du système afin de pouvoir détecter et localiser les défaillances, et prendre la décision en temps réel. Cette décision doit maximiser les capacités à répondre aux objectifs de la mission, tout en maintenant les exigences de sécurité. D'un autre coté, ils sont amenés à exécuter des tâches avec des demandes de calcul important sous contraintes de performance. Il est donc nécessaire de penser aux accélérateurs matériels dédiés pour décharger le processeur et répondre aux exigences de la rapidité de calcul.

C'est ce que nous cherchons à démontrer dans cette thèse à double objectif. Le premier objectif consiste à définir un modèle pour l'état de santé et la décision. Pour cela, nous utilisons les réseaux Bayésiens, qui sont des modèles graphiques probabilistes efficaces pour le diagnostic et la décision sous incertitude. Nous avons proposé un modèle générique en nous basant sur une analyse de défaillance de type FMEA (Analyse des Modes de Défaillance et de leurs Effets). Cette analyse prend en compte les différentes observations sur les capteurs moniteurs et contextes d'apparition des erreurs. Le deuxième objectif était la conception et la réalisation d'accélérateurs matériels des réseaux Bayésiens d'une manière générale et plus particulièrement de nos modèles d'état de santé et de décision. N'ayant pas d'outil pour l'implémentation embarqué du calcul par réseaux Bayésiens, nous proposons tout un atelier logiciel, allant d'un réseau Bayésien graphique ou textuel jusqu'à la génération du bitstream prêt pour l'implémentation logicielle ou matérielle sur FPGA. Finalement, nous testons et validons nos implémentations sur la ZedBoard de Xilinx, incorporant un processeur ARM Cortex-A9 et un FPGA.

***Mots-clés***— Réseaux Bayésiens, Etat de santé, Décision, FMEA, FPGA, Synthèse de haut niveau, Implémentation matérielle/logicielle.

# Abstract

Autonomous vehicles, such as drones, are used in different application areas to perform simple or complex missions. On one hand, they generally operate in uncertain environmental conditions, which can lead to disastrous consequences for humans and the environment. Therefore, it is necessary to continuously monitor the health of the system in order to detect and locate failures and to be able to make the decision in real time. This decision must maximize the ability to meet the mission objectives while maintaining the security requirements. On the other hand, they are required to perform tasks with large computation demands and performance requirements. Therefore, it is necessary to think of dedicated hardware accelerators to unload the processor and to meet the requirements of a computational speed-up.

This is what we tried to demonstrate in this dual objective thesis. The first objective is to define a model for the health management and decision making. To this end, we used Bayesian networks, which are efficient probabilistic graphical models for diagnosis and decision-making under uncertainty. We propose a generic model based on an FMEA ( Failure Modes and Effects Analysis). This analysis takes into account the different observations on the monitors and the appearance contexts. The second objective is the design and realization of hardware accelerators for Bayesian networks in general and more particularly for our models of health management and decision-making. Having no tool for the embedded implementation of computation by Bayesian networks, we propose a software workbench covering graphical or textual Bayesian networks up to the generation of the bitstream ready for the software or hardware implementation on FPGA. Finally, we test and validate our implementations on the Xilinx ZedBoard, incorporating an ARM Cortex-A9 processor and an FPGA.

***Keywords***— Bayesian networks, Health management, Decision making, FMEA, FPGA, High level synthesis, Hardware/Software implementation.

# Table des matières

# - Chapitre 1 -

# Introduction générale

## 1.1 Contexte et problématique

De nos jours, les systèmes autonomes, tels que les drones, sont utilisés dans différents domaines d'application pour exécuter des tâches telles que la détection d'objets, la localisation, le suivi, à travers des missions simples ou complexes. La complexité des applications et l'opération dans des conditions environnementales incertaines peuvent entraîner des scénarios de défaillance, tels que les défaillances matérielles ou logicielles, les événements environnementaux (obstacles), etc. Un dysfonctionnement peut conduire à des conséquences désastreuses pour l'humain et l'environnement, ainsi à un coût élevé de réparation. Par conséquent, il est nécessaire d'évaluer continuellement l'état de santé du système afin de garantir la fiabilité de la mission. Si un scénario de défaillance survient, le système doit agir en temps réel, dans le but de prendre des décisions (replanification, reconfiguration, etc.) qui maximisent les capacités à répondre aux objectifs de la mission, tout en maintenant les exigences de sécurité. Un exemple d'une décision peut être l'abandon de la mission de manière sécurisée pour les biens et le matériel, ou la poursuite de la mission à l'aide d'une nouvelle configuration.

L'évaluation de l'état de santé d'un système se base sur des méthodes de diagnostic. Ces méthodes ont pour objectif de détecter et de localiser la ou les défaillances, à l'aide d'observations sur le système et d'un modèle représentant les relations causales entre ses composants. Plusieurs modèles peuvent être envisagés tels que les réseaux de neurones, les systèmes experts, les modèles d'analyse de données, les arbres de défaillances, les modèles logiques, les réseaux Bayésiens, etc. Un tableau comparatif entre ces modèles est donné dans [Naïm et al., 2007]. Les réseaux Bayésiens, que nous utilisons dans nos travaux, sont largement utilisés pour implanter un diagnostic complexe dans différents types de systèmes [Pearl and Russell, 1998], et sont les plus adaptés pour faire face à l'incertitude. Néanmoins, il est difficile de construire

un modèle Bayésien rationnel pour des applications réelles et complexes. Pour résoudre ce problème, l'analyse des modes de défaillance et de leurs effets (FMEA) [McDermott et al., 1999] peuvent être utilisés.

De plus, par la représentation des dépendances causales probabilistes dans les réseaux Bayésiens, l'autonomie de la décision peut être fournie, contrairement aux méthodes de recouvrement classiques (FDIR) [Portinale and Codetta-Raiteri, 2011], où les informations sont transférées au sol pour le dépannage.

D'un autre coté, les systèmes autonomes sont amenés à exécuter des tâches complexes (suivi, localisation, reconstruction de terrain), avec des demandes de calcul qui peuvent varier au cours d'une mission. Les cartes de calcul embarqué, peuvent inclure diverses unités parallèles comme les processeurs multidimensionnels, GPU, FPGA ou SoC pour atteindre les objectifs en temps réel [Martin and Chang, 2012]. Pour les systèmes embarqués, les circuits hybrides CPU / FPGA fournissent également un banc d'essai intéressant pour les implémentations matérielles/ logicielles adaptables. Le choix entre les versions matérielles ou logicielles peut être piloté par des objectifs de performance ou de consommation d'énergie ainsi que par l'équilibrage de charge des tâches de mission entre la partie système de traitement (CPU) et la partie logique programmable (FPGA). Un autre avantage d'une implémentation matérielle dédiée est de maintenir le CPU disponible pour exécuter d'autres tâches de mission de logiciel. L'implémentation sur FPGA permet également une reconfiguration partielle ou complète dans le cas où le plan de mission est modifié en raison d'un scénario de défaillance. Des outils de CAO ont été introduits pour la conception d'IP intégrée et plus généralement de SoC dédié [Crockett et al., 2014]. Néanmoins, aucun d'entre eux n'a ciblé les implémentations des réseaux Bayésiens pour l'état de santé et la décision.

## 1.2 Contributions

Nos principales contributions se déclinent en trois points :

1. **Développement d'un atelier logiciel pour l'implémentation matérielle et logicielle des réseaux Bayésiens sur SoC**

   Les réseaux Bayésiens par leur structure et la complexité de calcul nécessitent une implémentation efficace et optimale, qui répond aux besoins de l'embarqué et du temps réel. N'ayant pas d'outil pour l'implémentation embarquée du calcul par réseaux Bayésiens, nous proposons un atelier logiciel dédié aux SoCs hybrides. A partir d'une représentation graphique ou textuelle du réseau Bayésien, nous générons un algorithme de calcul embarquable et parallélisable, prêt à être utilisé en entrée des outils de synthèse de haut niveau (HLS). Nous utilisons par la suite les outils HLS pour adapter la solution selon le besoin en parallélisme, partitionnement, contraintes de ressource ou de latence. Aussi, la HLS permet la définition des interfaces et des ressources pour le stockage

et l'envoi des paramètres et la génération de la description HDL standard qui sera mappée sur la partie FPGA. Par la suite, le déploiement sur l'architecture du SoC se fait en utilisant Vivado Design Suite, et le test en ligne peut être réalisé.

2. **Proposition d'un modèle Bayésien pour l'état de santé et la décision pour les missions des véhicules autonomes**

   Les réseaux Bayésiens sont des modèles efficaces pour le diagnostic et la décision mais il reste difficile de construire le modèle, qui se base généralement sur des expertises. Pour remédier à cette problématique, nous avons proposé un modèle générique pour l'état de santé et la décision dédié aux véhicules autonomes et pouvant être pris en compte au cours de leur mission. Ce modèle se base sur une analyse de défaillance de type FMEA (Analyse des Modes de Défaillance et leurs Effets), prenant en compte les différentes observations sur les capteurs moniteurs et contextes d'apparition des erreurs. Nous avons aussi adapté l'outil logiciel à ce cas particulier d'étude, en proposant un algorithme de calcul générique pour celui-ci.

3. **Application du modèle au cas d'une mission de drone et de l'atelier logiciel sur la Zedboard de Xilinx**

   Pour valider notre modèle, nous avons intégré le réseau Bayésien de l'état de santé et de la décision à un cas d'étude d'une mission de drone. La mission prise en compte est une mission de recherche et de sauvetage nommée ici "Save Outback Joe", qui a correspond à une mission du "2014 Canberra Unmanned Aerial Vehicle Outback Challenge" [Tridgell, 2014]. Pour la validation de l'atelier logiciel, nous avons utilisé la carte ZedBoard de Xilinx, incorporant un processeur ARM Cortex-A9 et une partie FPGA, communiquant via des bus AXI (Advanced eXtensible Interface).

## 1.3 Plan du mémoire

Ce document est organisé de la manière suivante :

Dans le second chapitre, nous présentons des généralités sur les réseaux Bayésiens et leurs algorithmes de calcul, appelé inférence Bayésienne. Nous exposons aussi des extensions des réseaux Bayésiens, permettant d'ajouter l'aspect temporaire et décisionnel. Et nous finissons ce chapitre par des exemples d'application des réseaux Bayésiens pour le diagnostic et la prise de décision.

Le troisième chapitre est consacré aux méthodes et outils des implémentations sur SoC et aux accélérateurs matériels. Nous exposons aussi la conception d'accélérateur FPGA à partir de la synthèse de haut niveau, utilisé dans notre outil. Et nous finissons ce chapitre par des exemples d'implémentation des réseaux Bayésiens sur FPGA.

Les quatrième, cinquième, et sixième chapitres sont dédiés à nos contributions. Dans le Chapitre 4, nous détaillons notre atelier logiciel pour l'implémentation sur SoC hybride des réseaux Bayésiens. Dans le Chapitre 5, nous exposons notre modèle Bayésien générique pour l'état de santé et la décision dans le cas d'une mission de véhicule autonome, ainsi que l'application sur l'exemple de "Save Outback Joe". Et dans le Chapitre 6, nous présentons nos différentes expérimentations sur la carte Zedboard de Xilinx, évaluant ressources, performance, et énergie.

Finalement, dans le septième chapitre, nous concluons avec un rappel de la problématique et des contributions, ainsi que nos perspectives futures.

## 1.4 Publications

### Revue internationale à comité de lecture

1. S. Zermani, C. Dezan, C.Hireche, R. Euler, et J. Diguet, "**Embedded Context Aware Diagnosis for a UAV SoC Platform**", *Elsevier Embedded Hardware Design Journal (MICPRO' 17)*, 2017. [Zermani et al., 2017b]

### Conférences internationales à comité de lecture

2. S. Zermani, C. Dezan, et R. Euler, "**Embedded Decision Making for UAV Missions** ", *6th Mediterranean Conference on Embedded Computing (MECO' 17)*, Montenegro, juin 2017 ; [Zermani et al., 2017a]
3. S. Zermani, C. Dezan, C.Hireche, R. Euler, et J. Diguet, "**Embedded and Probabilistic Health Management for the GPS of Autonomous Vehicles**", *5th Mediterranean Conference on Embedded Computing (MECO' 16)*, Montenegro, juin 2016 (reward de gratitude pour la contribution dans les travaux scientifiques et de recherche) ; [Zermani et al., 2016]
4. S. Zermani, C. Dezan, R. Euler, et J. Diguet, "**Bayesian network based framework for the design of reconfigurable health management monitors**", *NASA/ESA Conference on Adaptive Hardware and systems (AHS'15)*, Montreal, juin 2015 ; [Zermani et al., 2015b]
5. S. Zermani, C. Dezan, H. Chenini, J. Diguet, et R. Euler, "**FPGA implementation of Bayesian network inference for an embedded diagnosis**", *IEEE International Conference on Prognostics and Health Management (PHM' 15)*, Austin, juin 2015 ; [Zermani et al., 2015a]
6. C. Dezan, et S. Zermani, "**Stochastic Reliability Evaluation of Sea-of-Tiles Based on Double Gate Controllable-Polarity FETs**", *International Symposium on Nanoscale architectures (NANOARCH' 14)*, Paris, juillet 2014. [Dezan and Zermani, 2014]

### Conférences nationales à comité de lecture

7. S. Zermani, C. Dezan, C.Hireche, R. Euler, et J. Diguet, "**Génération de composant "état de santé" pour monitorer le système embarqué de véhicule autonome**", *Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS' 16)*, Lorient, juillet 2016.

### Communications

8. S. Zermani, C. Dezan, R. Euler, et J. Diguet, "**Online Inference for Adaptive Diagnosis via Arithmetic Circuit Compilation of Bayesian Networks**", *Designing with Uncertainty Opportunities and Challenges workshop*, York, mars 2014 ;
9. S. Zermani, C. Dezan, et R. Euler, "**Bayesian networks for diagnosis of combinational circuits**", *GDR SoC-SiP*, Lyon, juin 2013.
10. S. Zermani, H. Chenini , C. Dezan, R. Euler, D. Heller, J. Diguet, D. Campbell, B. Chen, et G. Coppin "**SWARMS Project : Self-Adaptive HW/SW Architecture for Unmanned Aerial Vehicles (UAVs)**", *GDR SoC-SiP*, Nantes, juin 2016, *Séminaire des doctorantes et doctorants de la SIF*, Paris, avril 2016.

# - Chapitre 2 -

# Les réseaux Bayésiens

## Sommaire

## 2.1 Introduction

Les réseaux Bayésiens [Pearl, 1988] sont des modèles graphiques probabilistes utilisés pour comprendre et analyser le comportement des systèmes, sous incertitude. Aujourd'hui, ils représentent un formalisme complet, associant des méthodes statistiques et des technologies de l'intelligence artificielle. Ces modèles permettent de représenter par un graphe orienté et de stocker dans des variables les connaissances sur le système, et définir avec des probabilités les relations entre ces variables. Les modèles peuvent être construits manuellement ou à l'aide des algorithmes d'apprentissage à partir de bases de données. La résolution d'un réseau Bayésien consiste à propager des informations certaines au sein du réseau (ce qu'on appelle observations ou évidence) et calculer les nouvelles probabilités a posteriori des variables cibles, grâce à des algorithmes d'inférence [Naïm et al., 2007]. L'inférence permet d'agencer connaissances et observations.

Nous consacrons ce chapitre aux réseaux Bayésiens, où nous présentons une vue générale sur ces modèles en se focalisant sur les parties que nous utilisons dans nos travaux. Dans la Section 2.2, nous exposons des généralités fondamentales sur les réseaux Bayésiens. Nous donnons les définitions importantes et illustrons avec des exemples. Dans la Section 2.3, nous détaillons des algorithmes de calcul d'inférence, qui permettent le calcul des probabilités a posteriori d'un réseau Bayésien. Plus particulièrement, les deux approches que nous utilisons sont à base d'un arbre de jonction [Jensen et al., 1990; Lauritzen and Spiegelhalter, 1990] et à base de compilation en circuit arithmétique [Darwiche, 2001a, 2003; Huang, 1996]. Dans la Section 2.4, nous exposons des extensions des réseaux Bayésiens, et nous détaillons les réseaux Bayésiens dynamiques, ajoutant l'aspect temporel, ainsi que les diagrammes d'inférence, ajoutant l'aspect décision. Dans la Section 2.5, nous nous intéressons à la mise en œuvre des réseaux Bayésiens dans le diagnostic et la décision des systèmes autonomes. Et finalement, nous concluons sur ce chapitre et donnons un aperçu sur le chapitre suivant.

## 2.2 Généralités sur les réseaux Bayésiens

Le choix d'utilisation d'un modèle dépend du type de l'application. Les réseaux Bayésiens sont plus envisageables que d'autres modèles (réseau de neurones, système expert, arbre de décision, modèle d'analyse de données, arbre de défaillances) dans les cas des besoins suivants [Naïm et al., 2007] :

— Un problème d'incertitude, que ce soit dans les observations ou dans les règles de décision,

— Une fusion de connaissances de nature différente dans le même modèle (expertise, données, observations),

— Une représentation graphique intuitive et compréhensible par un non-spécialiste,
— Une polyvalence du réseau Bayésien qui peut être utilisé pour la prédiction, le diagnostic et la décision avec le même modèle,
— Une disponibilité d'outils logiciels comprenant l'apprentissage, l'intégration de la décision, etc.

Un réseau Bayésien est la représentation de la connaissance d'un système qui permet :

— **La prédiction** du comportement d'un système.
— **Le diagnostic** des causes d'un évènement observé dans le système.
— **Le contrôle** du comportement d'un système.
— **La simulation** du comportement d'un système.
— **L'analyse** des données d'un système.
— **La prise de décision** dans un système.

Les domaines d'application des réseaux Bayésiens sont donc vastes, pouvant rassembler plusieurs disciplines scientifiques telles que : l'intelligence artificielle, les probabilités et statistiques, la théorie de la décision, l'informatique et aussi les sciences cognitives. Par ailleurs ils sont moins adaptés aux applications apparentées à la résolution de problèmes ou à la démonstration de théorèmes. Parmi les divers domaines d'application nous citons :

— La santé

  Les réseaux Bayésiens ont été appliqués en premier lieux dans le diagnostic médical. Ils sont bien adaptés car ils offrent un modèle qui se base sur l'expertise humaine et des données statistiques. Par exemple la localisation de gènes à partir de l'analyse d'arbre généalogique [Friedman et al., 2000; Hart and Graham, 1997].

— L'industrie

  Les réseaux Bayésiens sont utilisés surtout dans les systèmes autonomes tels que les robots adaptatifs. Dans ce cas, le système doit être capable de se mettre à jour et de trouver une solution dans le cas d'un changement environnemental ou d'un endommagement [Skaanning, 2000].

— La défense

  Les réseaux Bayésiens sont utilisés dans la fusion de données, grâce à la capacité à prendre en compte des données incertaines et guider la recherche ou la vérification des données [Liu and Wu, 2011].

— L'informatique

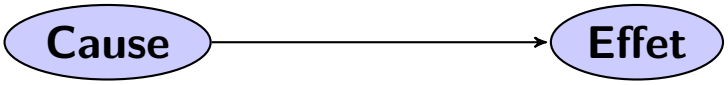
  Les réseaux Bayésiens sont utilisés dans le diagnostic de programmes informatiques, ainsi que l'intelligence des agents (logiciels, locaux à une machine, ou autonomes sur des réseaux ou sur Internet) [Kim and Valtorta, 1995; Horvitz et al., 1998].

### 2.2.1 Définition

Les réseaux Bayésiens sont des modèles qui représentent des connaissances incertaines sur des phénomènes complexes. Un réseau Bayésien peut se décomposer en deux parties (voir Figure 2.1) :

— une représentation graphique par un graphe orienté acyclique (DAG). Les variables aléatoires sont représentés par des nœuds contenant leurs états, et les relations de dépendance entre ces variables par des arcs. Les nœuds sont reliés par la relation causale (relation de cause à effet),

— une représentation probabiliste par un ensemble de tables de probabilités conditionnelles (CPT). A chaque nœud est associé une CPT de taille exponentielle au nombre de parents et leur nombre d'états.




| | |
|---|---|
| Cause= $état_0$ | $P_0$ |
| Cause= $état_1$ | $1 - P_0$ |

| | Cause=$état_0$ | Cause=$état_1$ |
|---|---|---|
| Effet= $état'_0$ | $P'_0$ | $P'_1$ |
| Effet= $état'_1$ | $1 - P'_0$ | $1 - P'_1$ |

**Figure 2.1:** *Exemple de la représentation par réseau Bayésien.*

La construction d'un réseau Bayésien se base sur la connaissance des composants du système étudié et leurs interactions, ainsi que les règles de la circulation de l'information (D-séparation) [Pearl, 2000]. Le calcul dans le réseau se base sur le théorème de Bayes. Dans ce qui suit, nous donnons une définition formelle des réseaux Bayésiens et nous donnons un exemple simple explicatif. Nous expliquons par la suite le principe de la D-séparation et le calcul dans les réseaux avec le théorème de Bayes, tout en illustrant sur l'exemple.

D'une manière formelle, un réseau Bayésien se définit comme suit [Jensen and Nielsen, 2007] :

— un graphe orienté acyclique (Directed Acyclic Graph ou DAG) $R = (N, A)$, où $N = (N_1, N_2, ..., N_m)$ est l'ensemble des nœuds et A l'ensemble des arcs de R,

— un espace probabiliste fini $(\Omega, Z, p)$, avec $\Omega$ un espace non-vide, $Z$ un sous espace de $\Omega$, et $p$ une mesure de probabilité dans $Z$ avec $\mathrm{p}(\Omega) = 1$.

— un ensemble de variables associées aux nœuds du graphe et définies sur $(\Omega, Z, p)$, tel que :

$$p(N_1, N_2, ..., N_m) = \prod_{i=1}^{m} p(N_i | Pa(V_i))$$

où $Pa(V_i)$ est l'ensemble des parents du nœud $V_i$ dans le graphe $R$.
$p(N_1, N_2, ..., N_m)$ représente ce qu'on appelle la distribution de probabilité jointe, et qui définit un réseau Bayésien sur l'ensemble N.

### 2.2.2 Exemple simple

Nous donnons un exemple simple et intuitif, qui nous sert pour l'explication de la construction d'un réseau Bayésien, le principe de circulation de l'information et la D-séparation dans un tel réseau, ainsi que le raisonnement et le calcul basé sur le théorème de Bayes.

L'exemple traite le problème de défaillances matérielles d'un ordinateur et peut être formulé comme suit :
*Sur un ordinateur, on considère deux types de défaillances matérielles : défaillance de la RAM, et défaillance du CPU. On peut constater la présence d'une défaillance lorsque l'écran de l'ordinateur est bleu ou le système bloque. Rajoutons l'information que la défaillance du CPU est observée par une surchauffe. Le but est de trouver la cause de la défaillance à partir des observations.*

### 2.2.3 Construction d'un réseau Bayésien

La construction d'un réseau Bayésien peut se faire de trois manières comme suit :

— Manuelle : défini par des experts humains.

— Automatique : défini par un algorithme d'apprentissage appliqué sur une base de données.

— Hybride : défini par la combinaison des deux approches. L'apprentissage peut affiner un premier modèle construit manuellement à partir de données disponibles.

Ici nous exposons la construction manuelle sur l'exemple et par la suite nous expliquons l'apprentissage. La construction peut ne pas être unique et cela dépend de l'utilisation du réseau.

La construction d'un réseau Bayésien se fait en trois étapes :

— Identification des variables et de leurs espaces d'états, qui représentent les nœuds dans le réseau Bayésien.

— Définition de la structure du réseau Bayésien, qui représente les liens entre les nœuds en respectant les influences des variables entre elles, ainsi que les règles de circulation de l'information.

— Définition de la loi de probabilité conjointe des variables, qui représente les tables de probabilités associées à chaque nœud.

1. Identification des variables et de leurs espaces d'états : Il s'agit de définir l'ensemble des variables caractérisant le problème traité qui représentent les nœuds du réseau. Pour chaque variable, définir par la suite l'espace d'états représentant l'ensemble des valeurs possibles. Nous utilisons la notation suivante : par exemple pour une variable $A$, l'ensemble des états est $(a_0, a_1, \dots)$. Pour l'exemple de la défaillance matérielle d'un ordinateur, les nœuds et les états pour chacun peuvent être les suivants :

— **Nœud CPU** : qui peut avoir l'état bon ou défaillant.

— **Nœud RAM** : qui peut avoir l'état bon ou défaillant.

— **Nœud Etat de l'ordinateur** : qui peut avoir l'état bon, écran bleu ou bloqué.

— **Nœud Surchauffe** : qui peut avoir l'état oui ou non.

2. Définition de la structure du réseau Bayésien : Un réseau Bayésien est un graphe orienté acyclique, dont il faut définir les liens et la direction des liens sans avoir de boucle. Pour cela nous pouvons nous baser sur le principe "cause à effet" qui veut dire la flèche doit partir de la cause (nœud parent) vers la conséquence (nœud fils), ou "effet à observation" qui veut dire une flèche de l'effet vers le moyen d'observation, tout en respectant les règles de circulation (D-séparation) de l'information selon les types des connexions comme indiqués dans le tableau suivant (Tableau 2.1) [Pearl, 2000] :

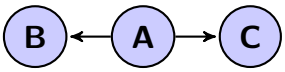
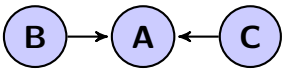
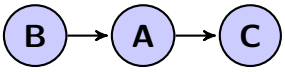
| Graphe | Type de connexion | Règles de circulation |
|---|---|---|
|  | Connexion divergente<br>(A cause B et C) | L'information peut circuler<br>de B à C si A n'est pas connu |
|  | Connexion convergente<br>(B et C causent A) | L'information peut circuler<br>de B à C si A est connu |
|  | Connexion en série<br>(B cause A et A cause C) | L'information peut circuler<br>de B à C si A n'est pas connu |

**Tableau 2.1:** *Les types de connexions dans un réseau Bayésien.*

Pour l'exemple de la défaillance matérielle d'un ordinateur, les liens peuvent être comme suit :

— **CPU et RAM vers Etat de l'ordinateur** (connexion convergente) : une défaillance sur le CPU ou sur la RAM cause un écran bleu ou un blocage de l'ordinateur.

— **CPU vers Surchauffe** (connexion divergente) : une défaillance sur le CPU cause une surchauffe en plus de l'état de l'ordinateur. Ou sinon on peut dire la défaillance du CPU est observée par une surchauffe.

3. Définition de la loi de probabilité conjointe des variables : Il s'agit de définir, par expertise, les tables de probabilités des variables du réseau. Deux types sont déterminés selon la position du nœud comme suit :

— Table a priori : pour les nœuds qui n'ont pas de parents, où l'expert précise la probabilité marginale de la variable.

— Table conditionnelle : pour les nœuds ayant au minimum un parent, où l'expert précise la dépendance pour chaque état d'une variable avec toutes les combinaisons d'états de ses parents.

La somme des probabilités d'une colonne (pour chaque état de la variable et tous les parents) doit être égale à 1.

Pour l'exemple de la défaillance matérielle d'un ordinateur, les tables de probabilités peuvent être définies comme suit :

— Table a priori : le nœud RAM (Tableau 2.2 ), et le nœud CPU (Tableau 2.3).

| | |
|---|---|
| RAM= bon | 0.5 |
| RAM= défaillant | 0.5 |

**Tableau 2.2:** *Table a priori de la RAM.*

| | |
|---|---|
| CPU= bon | 0.5 |
| CPU= défaillant | 0.5 |

**Tableau 2.3:** *Table a priori du CPU.*

Cela signifie que a priori, on attribue la probabilité de 0.5 pour un état bon et 0.5 pour un état défaillant pour les deux variables CPU et RAM.

— Table conditionnelle : le nœud Etat de l'ordinateur (Tableau 2.4) et le nœud Surchauffe (Tableau 2.5).

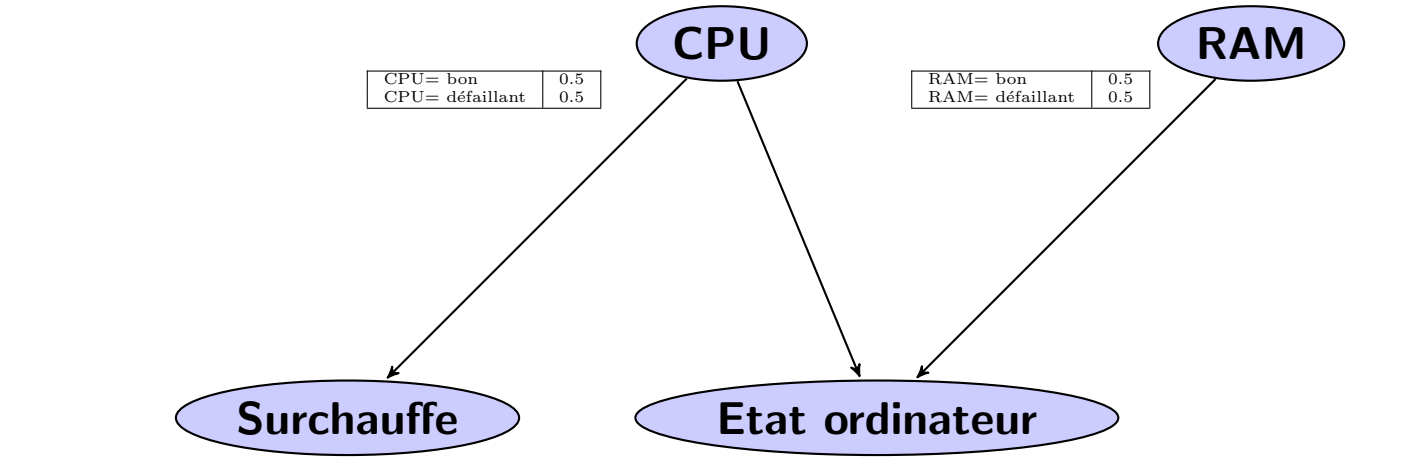
| | RAM= bon | | RAM= défaillant | |
|---|---|---|---|---|
| | CPU= bon | CPU=défaillant | CPU= bon | CPU=défaillant |
| Etat ordinateur= bon | 0.8 | 0.1 | 0.1 | 0 |
| Etat ordinateur= ecranbleu | 0.1 | 0.5 | 0.4 | 0.5 |
| Etat ordinateur= bloqué | 0.1 | 0.4 | 0.5 | 0.5 |

**Tableau 2.4:** *Table conditionnelle du nœud Etat de l'ordinateur.*

| | CPU=bon | CPU=défaillant |
|---|---|---|
| Surchauffe= oui | 0.3 | 0.9 |
| Surchauffe= non | 0.7 | 0.1 |

**Tableau 2.5:** *Table conditionnelle du nœud Surchauffe.*

Cela donne les relations conditionnelles entre les nœuds et leurs parents. Par exemple dans le cas de la RAM et du CPU en bon état, l'ordinateur a 80% de chance qu'il fonctionne correctement, 10% de chance que l'écran soit bleu, et 10% de chance que l'ordinateur bloque, par rapport à d'autres défaillances possibles et non la RAM ou le CPU. Par contre dans le cas ou les deux sont défaillants, il n'y a aucune chance pour que l'ordinateur fonctionne correctement et 50% de chance pour les deux autres cas.




| | |
|---|---|
| CPU= bon | 0.5 |
| CPU= défaillant | 0.5 |

| | |
|---|---|
| RAM= bon | 0.5 |
| RAM= défaillant | 0.5 |

| | CPU=bon | CPU=défaillant |
|---|---|---|
| Surchauffe= oui | 0.3 | 0.9 |
| Surchauffe= non | 0.7 | 0.1 |

| | RAM= bon | | RAM= défaillant | |
|---|---|---|---|---|
| | CPU= bon | CPU=défaillant | CPU= bon | CPU=défaillant |
| Etat ordinateur= bon | 0.8 | 0.1 | 0.1 | 0 |
| Etat ordinateur= ecranbleu | 0.1 | 0.5 | 0.4 | 0.5 |
| Etat ordinateur= bloqué | 0.1 | 0.4 | 0.5 | 0.5 |

**Figure 2.2:** *Un réseau Bayésien pour l'exemple "défaillance matérielle d'un ordinateur".*

A partir de ces 3 points, nous avons obtenu le réseau Bayésien de la Figure 2.2.

Cette représentation est la représentation la plus intuitive. Elle permet de trouver la cause (CPU ou RAM) de la défaillance d'un ordinateur selon les observations. Nous verrons comment le calcul se fait par la suite. Nous pouvons aussi proposer une autre modélisation, où nous mettons en avant le fait qu'un problème soit présent ou pas, et chercher la cause par la suite.

Le modèle est représenté dans la Figure 2.3, où les nœuds et les arcs sont les suivants :

— **Nœud Etat de l'ordinateur** : qui peut avoir l'état bon ou défaillant

— **Observation** : qui peut avoir l'état rien à signaler, écran bleu ou bloqué.

— **Nœud CPU** : qui peut avoir l'état bon ou défaillant.

— **Nœud RAM** : qui peut avoir l'état bon ou défaillant.

— **Nœud Surchauffe** : qui peut avoir l'état oui ou non.

Les relations entre les nœuds sont les suivantes :

— **Etat de l'ordinateur vers Observation** : l'état de l'ordinateur représente l'effet qui est observé par le nœud Observation.

— **Etat de l'ordinateur vers CPU et RAM** : l'état de l'ordinateur peut aussi être observé par les deux nœuds CPU et RAM.

— **CPU vers Surchauffe** : la défaillance du CPU est observée par une surchauffe.

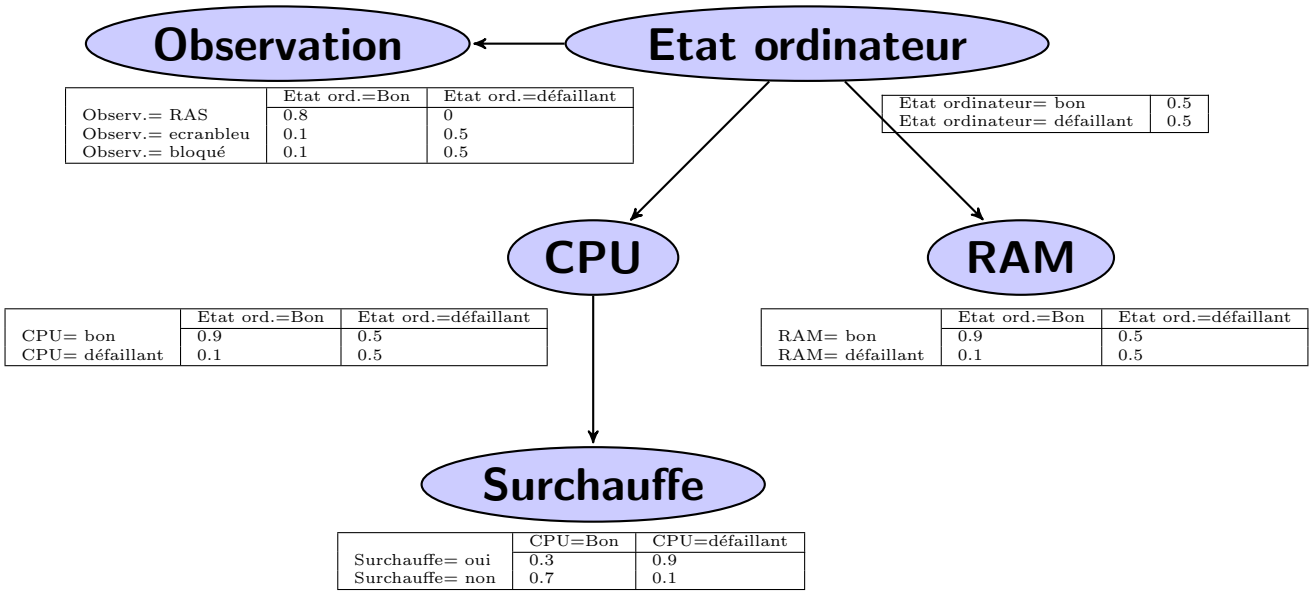



**Figure 2.3:** *Deuxième modélisation de l'exemple "défaillance matérielle d'un ordinateur".*

### 2.2.4 Apprentissage dans les réseaux Bayésiens

Les algorithmes d'apprentissage permettent, à partir d'une base de données (complète ou incomplète), de déterminer le modèle Bayésien correspondant (apprentissage de structures), ou d'obtenir la distribution des probabilités (apprentissage de paramètres) [Neapolitan, 2003].

1. Apprentissage de structures

   Il consiste à définir le modèle Bayésien qui représente le mieux le problème, à partir de données disponibles. La solution naïve pour trouver la meilleure structure est d'explorer l'espace des structures possibles, en associant un score à chacune et de choisir par la suite le modèle avec le score le plus élevé. Ce parcours exhaustif rend cet apprentissage complexe et exponentiel par rapport au nombre élevé des réseaux possibles. Plusieurs approches existent, en se basant généralement sur des heuristiques, afin de résoudre ce problème [Margaritis, 2003; Jordan, 2004; Naïm et al., 2007].

2. Apprentissage de paramètres

   Il consiste à définir pour une structure donnée les distributions de probabilités à partir de données disponibles. Deux cas sont possibles : le cas où les données sont complètes, et donc la résolution se fait à travers le maximum de vraisemblance, utilisant la fréquence des événements dans les données [Grossman and Domingos, 2004], et le cas où les données sont incomplètes, et où la résolution se fait par des algorithmes EM (Expectation-Maximisation) [Friedman, 1998], utilisant l'inférence pour calculer la distribution des probabilités jusqu'à l'obtention du modèle le plus vraisemblable.

### 2.2.5 Théorème de Bayes

Le raisonnement dans un réseau Bayésien pour le diagnostic revient à calculer les probabilités conditionnelles de certains nœuds, en fixant des observations certaines sur d'autres. Ce calcul se base sur la règle de Bayes.

Prenons deux variables $A(a_0, a_1, \dots.)$ et $B(b_0, b_1, \dots)$, avec $A$ nœud parent et $B$ nœud fils. Le calcul de la probabilité conditionnelle (a posteriori) du nœud $A$ (par exemple pour l'état $a_0$) avec l'observation sur le nœud $B$ (par exemple l'observation est l'état $b_0$) se fait en utilisant le théorème de Bayes comme suit (Equation 2.1) :

$$P(A = a_0|B = b_0) = \frac{P(B = b_0|A = a_0) * P(A = a_0)}{P(B = b_0)} \tag{2.1}$$

Nous appliquons le calcul à base du théorème de Bayes à l'exemple précédent (modèle intuitif). Dans l'exemple, le but est de calculer la probabilité de la défaillance de la RAM et du CPU selon les observations. Nous étudions 3 cas : un cas nominal (pas d'écran bleu, ni blocage, ni surchauffe), un cas d'observation d'un écran bleu, un cas d'observation d'un écran bleu et une surchauffe.

Notation : défaillant=déf., RAM=R, Etat ordinateur=E, Surchauffe=S, CPU=C.

1. Le cas nominal : les probabilités à calculer ici sont :

   (a) La probabilité de la RAM, sachant que l'état de l'ordinateur donne un bon fonctionnement et une absence de surchauffe : cela se traduit par l'Equation 2.2.

$$P(R = déf.|E = bon, S = non) = \frac{P(E = bon, S = non|R = déf.) * P(R = déf.)}{P(E = bon, S = non)} \tag{2.2}$$

   On a (par calcul des probabilités de base) :

$$P(E = bon, S = non|R = déf.) * P(R = déf.) = P(E = bon, S = non, R = déf.)$$

   et

$$\begin{aligned} P(E = bon, S = non, R = déf.) &= P(E = bon|R = déf., C = bon) * P(S = non|C = bon) \\ &\quad * P(C = bon) * P(R = déf.) \\ &\quad + P(E = bon|R = déf., C = déf.) * P(S = non|C = déf.) \\ &\quad * P(C = déf.) * P(R = déf.) \\ &= 0.1 * 0.7 * 0.5 * 0.5 + 0 * 0.1 * 0.5 * 0.5 \\ &= 0.0175 \end{aligned}$$

Et

$$\begin{aligned}P(E=bon,S=non) &= P(E=bon|R=bon,C=bon)*P(S=non|C=bon)\\ &\quad *P(C=bon)*P(R=bon)\\ &\quad +P(E=bon|R=bon,C=déf.)*P(S=non|C=déf.)\\ &\quad *P(C=déf.)*P(R=bon)\\ &\quad +P(E=bon|R=déf.,C=bon)*P(S=non|C=bon)\\ &\quad *P(C=bon)*P(R=déf.)\\ &\quad +P(E=bon|R=déf.,C=déf.)*P(S=non|C=déf.)\\ &\quad *P(C=déf.)*P(R=déf.)\\ &= 0.8*0.7*0.5*0.5+0.1*0.1*0.5*0.5\\ &\quad +0.1*0.7*0.5*0.5+0*0.1*0.5*0.5\\ &= 0.16\end{aligned}$$

Donc

$$\begin{aligned}P(R=déf.|E=bon,S=non) &= \frac{0.0175}{0.16}\\ &= 0.109\end{aligned}$$

Ce résultat confirme que si l'état de l'ordinateur est bon et s'il n'y a pas de surchauffe, la probabilité d'avoir une défaillance sur la RAM est très faible.

(b) La probabilité du CPU, sachant que l'état de l'ordinateur donne un bon fonctionnement et une absence de surchauffe : de la même façon que pour la RAM, nous obtenons une probabilité de 0.016 d'avoir une défaillance sur le CPU.

2. Le cas d'observation d'un écran bleu : Nous calculons les mêmes probabilités (CPU et RAM) en observant l'état de l'ordinateur à "écran bleu" (sans observation sur la surchauffe). Cela revient à calculer :
$P(R=déf.|E=ecranbleu)$ et $P(C=déf.|E=ecranbleu)$.

(a) Calcul probabilité de défaillance de la RAM sachant l'écran bleu (Equation 2.3 )

$$\begin{aligned}P(R=déf.|E=ecranbleu) &= \frac{P(E=ecranbleu|R=déf.)*P(R=déf.)}{P(E=ecranbleu)}\\ &= 0.60\end{aligned} \tag{2.3}$$

(b) Calcul probabilité de défaillance du CPU sachant l'écran bleu (Equation 2.4 )

$$P(C = déf.|E = ecranbleu) = \frac{P(E = ecranbleu|R = déf.) * P(C = déf.)}{P(E = ecranbleu)} = 0.667 \tag{2.4}$$

De ces résultats, avec l'observation uniquement sur l'état de l'ordinateur, la probabilité de défaillance est passée de 0.5 à 0.6 pour la RAM et 0.66 pour le CPU. Nous ne pouvons pas privilégier l'une des deux causes avec cette seule information.

3. Le cas d'observation d'un écran bleu et d'une surchauffe : Nous calculons les mêmes probabilités (CPU et RAM) pour comme information l'état de l'ordinateur à "écran bleu" et aussi une surchauffe. Cela revient à calculer :
$P(R = déf.|E = ecranbleu, S = oui)$ et $P(C = déf.|E = ecranbleu, S = oui)$.
   — Calcul probabilité de défaillance de la RAM sachant l'écran bleu et une surchauffe (Equation 2.5 )

$$P(R = déf.|E = ecranbleu, S = oui) = \frac{P(E = ecranbleu, S = oui|R = déf.) * P(R = déf.)}{P(E = ecranbleu, S = oui)} = 0.543 \tag{2.5}$$

   — Calcul probabilité de défaillance du CPU sachant l'écran bleu et une surchauffe (Equation 2.6 )

$$P(C = déf.|E = ecranbleu, S = oui) = \frac{P(E = ecranbleu|R = déf.) * P(C = déf.)}{P(E = ecranbleu, S = oui)} = 0.857 \tag{2.6}$$

De ces résultats, en rajoutant l'observation sur la surchauffe, la probabilité de défaillance du CPU est passée de 0.667 à 0.857, car l'information de la surchauffe a renforcé la croyance d'une défaillance au niveau du CPU. Et par rapport aux règles de la D-séparation, l'information passe du CPU vers la RAM, ce qui diminue la probabilité de la défaillance de la RAM à 0.543. Ici, nous pouvons privilégier la défaillance sur le CPU.

Nous pouvons constater que pour des réseaux plus grands, les calculs avec le théorème de Bayes deviennent complexes et coûteux en temps. Il existe des algorithmes, appelés algorithmes d'inférence pour résoudre ce problème. Nous présentons par la suite les différents algorithmes pour en détailler certains que nous avons utilisés dans nos travaux.

## 2.3 Inférence dans les réseaux Bayésiens

Les algorithmes d'inférence sont utilisés pour répondre aux requêtes de calcul des probabilités a posteriori. Ces calculs sont relativement complexes selon la structure du réseau (nombre de nœuds, les relations entre les nœuds, le nombre des parents d'un nœud, et le nombre d'états des nœuds), ce qui a donné lieu à de nombreuses recherches.

Nous trouvons dans la littérature deux grandes familles d'algorithmes d'inférence. La première comprend les algorithmes d'inférence exacte, tels que les algorithmes de propagation de messages [Jensen et al., 1990; Lauritzen and Spiegelhalter, 1990], les algorithmes de conditionnement [Darwiche, 2001b], les algorithmes d'élimination de variables [Li and D'Ambrosio, 1994; Zhang and Poole, 1996], etc. Ces algorithmes exploitent la structure du réseau ou le regroupement des nœuds, afin de se ramener à un arbre tel que les algorithmes se basant sur l'arbre de jonction. La seconde comprend les algorithmes d'inférence approchée [Henrion, 1986; Kjærulff, 1994; Mengshoel et al., 2011], donnant des estimations de la probabilité a posteriori, en appliquant les méthodes exactes sur une partie du graphe, ou en utilisant des méthodes de simulation stochastique.

Nous présentons en détail deux algorithmes : le calcul d'inférence par arbre de jonction, qui est l'approche la plus répandue dans la littérature, et la plus utilisée dans les outils logiciels, et le calcul d'inférence basé sur la compilation en circuit arithmétique qui est une approche efficace pouvant offrir des performances en temps réel et qui nous servira par la suite pour l'implémentation matérielle.

### 2.3.1 L'inférence par arbre de jonction

Cette approche contient deux phases. Une première phase où le réseau Bayésien est converti vers une seconde structure (arbre de jonction) et une deuxième phase de calcul sur cette structure d'arbre [Lauritzen, 1992].

1. Construction de l'arbre de jonction à partir du réseau Bayésien

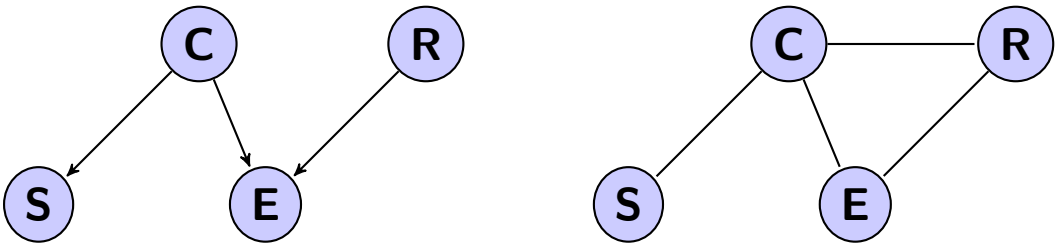
   La transformation d'un réseau Bayésien en un arbre de jonction se fait en 3 étapes :

   — La moralisation : pour obtenir un graphe non orienté.

   — La triangulation : rajouter des arcs pour obtenir un graphe triangulé.

   — La construction de l'arbre de jonction : en définissant un ensemble de cliques et de séparateurs.

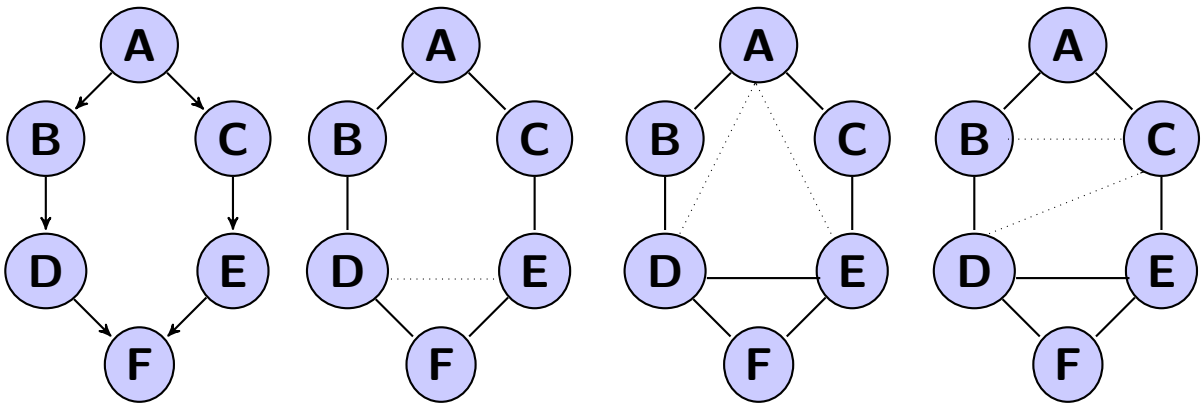
   (a) La moralisation : se fait en deux étapes : créer un graphe non orienté en supprimant les orientations des arcs et relier toute paire de nœuds parents par un arc non orienté [Huang, 1996]. La Figure 2.4 illustre cette transformation sur l'exemple de la défaillance matérielle d'un ordinateur.




Graphe Bayésien Graphe moral

**Figure 2.4:** *La moralisation du réseau Bayésien pour l'exemple "défaillance matérielle d'un ordinateur".*

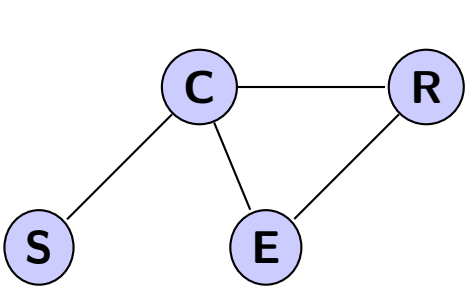
(b) La triangulation : se fait en réduisant tout cycle de plus de trois nœuds du graphe moral en cycles de trois nœuds maximum [Kjærulff, 1990]. Dans l'exemple de la défaillance matérielle d'un ordinateur, nous n'avons pas de cycles de plus de 3 nœuds. Pour illustrer la moralisation, nous prenons un autre exemple, où différents graphes triangulés peuvent être obtenus à partir du graph moral (Figure 2.5).



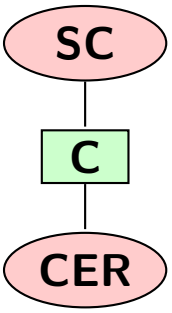

Graphe Bayésien Graphe moral Graphe triangulé 1 Graphe triangulé 2

**Figure 2.5:** *Exemple illustrant la triangulation.*

(c) La construction de l'arbre de jonction : se fait en reliant l'ensemble des cliques et des séparateurs du graphe triangulé, où les cliques sont les cycles de trois nœuds maximum et les séparateurs sont les intersections entre chaque paire de cliques voisines. Les arbres de jonction pour l'exemple de la défaillance matérielle de l'ordinateur sont illustrés dans la Figure 2.6.




Graphe moral




Arbre de jonction correspondant

**Figure 2.6:** *La construction de l'arbre de jonction pour l'exemple "défaillance matérielle d'un ordinateur".*

2. Calcul des probabilités conditionnelles par arbre de jonction [Jensen et al., 1990; Huang, 1996]

   Le calcul se base sur la propagation des messages dans un arbre. Les étapes à suivre pour effectuer le calcul des probabilités conditionnelles sont :

   — L'initialisation : consiste à donner à chaque nœud une probabilité potentielle en utilisant les tables de probabilités du réseau Bayésien.

   — La propagation globale : consiste à un passage de message entre les cliques et la mise à jour des probabilités potentielles.

   — La normalisation : consiste à calculer les probabilités conditionnelles.

   (a) L'initialisation : cette étape se fait en 3 phases :

      i. Encodage d'observations (table $\Lambda$ du Tableau 2.6) : pour chaque variable $X$ (avec les états $x_0, x_1, \dots$) du réseau, encoder l'observation comme suit :

         — Si $X$ est une observation : $\Lambda X(x_i) = 1, i \in \{0, 1, \dots\}$ lorsque $x_i$ est l'état observé par $X$, 0 sinon.

         — Si $X$ n'est pas une observation $\Lambda X(x_i) = 1, i \in \{0, 1, \dots\}$ pour tous les états $x_i$ de $X$.

         Le Tableau 2.6 illustre l'encodage d'observations pour le calcul de $P(R = déf.|E = ecranbleu, S = oui)$ de l'exemple de la défaillance matérielle d'un ordinateur.

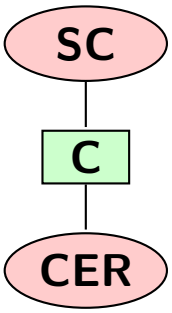
      ii. Initialisation des potentielles ($\Phi$) : se fait comme suit :

         — Pour chaque clique et séparateur C mettre $\Phi_C \leftarrow 1$.

         — Pour chaque clique mettre $\Phi_C \leftarrow \Phi_C P(X|Pa_X)$, où $X$ représente les variables de la clique et $Pa_X$ un parent de X et inclus dans la clique.

| Variables | Etats | $\Lambda$ |
|---|---|---|
| R | bon | 1 |
| | déf. | 1 |
| C | bon | 1 |
| | déf. | 1 |
| E | bon | 0 |
| | ecranbleu | 1 |
| | bloqué | 0 |
| S | oui | 1 |
| | non | 0 |

**Tableau 2.6:** *Encodage d'observation pour l'exemple "défaillance matérielle d'un ordinateur".*




Arbre de jonction

| S | C | $\Phi$(S,C) (1* P(S\|C)* P(C)) |
|---|---|---|
| oui | bon | 1* 0.3*0.5 |
| oui | déf. | 1* 0.9*0.5 |
| non | bon | 1* 0.7*0.5 |
| non | déf. | 1* 0.1*0.5 |

Potentielles SC

| C | $\Phi$(C) |
|---|---|
| bon | 1 |
| déf. | 1 |

Potentielles C

| C | E | R | $\Phi$(C, E, R) |
|---|---|---|---|
| bon | bon | bon | 1* 0.8*0.5*0.5 |
| bon | bon | déf. | 1* 0.1*0.5*0.5 |
| bon | ecranbleu | bon | 1* 0.1*0.5*0.5 |
| bon | ecranbleu | déf. | 1* 0.4*0.5*0.5 |
| bon | bloqué | bon | 1* 0.1*0.5*0.5 |
| bon | bloqué | déf. | 1* 0.5*0.5*0.5 |
| déf. | bon | bon | 1* 0.1*0.5*0.5 |
| déf. | bon | déf. | 1* 0*0.5*0.5 |
| déf. | ecranbleu | bon | 1* 0.5*0.5*0.5 |
| déf. | ecranbleu | déf. | 1* 0.5*0.5*0.5 |
| déf. | bloqué | bon | 1* 0.4*0.5*0.5 |
| déf. | bloqué | déf. | 1* 0.5*0.5*0.5 |

Potentielles CER

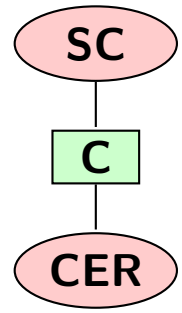
**Figure 2.7:** *L'initialisation des potentielles pour l'exemple "défaillance matérielle d'un ordinateur".*

La Figure 2.7 illustre l'initialisation des potentielles pour l'exemple de la défaillance matérielle d'un ordinateur.

iii. Initialisation des potentielles observables ($\Phi$) : identifier les cliques (C) contenant les évidences (e) puis mettre à jour les probabilités en mettant $\Phi_C = \Phi_C \Lambda_e$ (Figure 2.8)

(b) La propagation globale : cette étape se fait en deux phases :

i. Envoi des messages : tous les cliques envoient un message vers une clique choisie C en passant par tous leurs voisins. A chaque passage d'un message d'une clique C1 vers une clique C2 (soit R le séparateur de C1 et C2), les probabilités potentielles de C2 et R changent comme suit :




Arbre de jonction

| S | C | $\Phi$(S,C) |
|---|---|---|
| oui | bon | 0.15*1= 0.15 |
| oui | déf. | 0.45*1= 0.45 |
| non | bon | 0.35*0= 0 |
| non | déf. | 0.15*0= 0 |

Potentielles SC

| C | $\Phi$(C) |
|---|---|
| bon | 1 |
| déf. | 1 |

Potentielles C

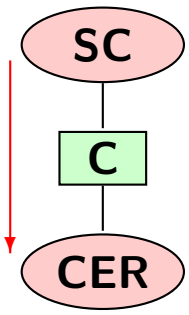
| C | E | R | $\Phi$(C, E, R) |
|---|---|---|---|
| bon | bon | bon | 0 |
| bon | bon | déf. | 0 |
| bon | ecranbleu | bon | 0.025 |
| bon | ecranbleu | déf. | 0.1 |
| bon | bloqué | bon | 0 |
| bon | bloqué | déf. | 0 |
| déf. | bon | bon | 0 |
| déf. | bon | déf. | 0 |
| déf. | ecranbleu | bon | 0.125 |
| déf. | ecranbleu | déf. | 0.125 |
| déf. | bloqué | bon | 0 |
| déf. | bloqué | déf. | 0 |

Potentielles CER

**Figure 2.8:** *L'initialisation des potentielles observables pour l'exemple "défaillance matérielle d'un ordinateur".*

$$\begin{cases} \Phi'_R = \Phi_R \\ \Phi_R = \sum\limits_{C1|R} \Phi_{C1} \\ \Phi_{C2} = \Phi_{C2}\frac{\Phi_R}{\Phi'_R} \end{cases}$$

La Figure 2.9 illustre l'envoi des messages en prenant comme clique choisie CER dans l'exemple de la défaillance matérielle d'un ordinateur.




Arbre de jonction

| S | C | $\Phi$(S,C) |
|---|---|---|
| oui | bon | 0.15 |
| oui | déf. | 0.45 |
| non | bon | 0 |
| non | déf. | 0 |

Potentielles SC

| C | $\Phi$(C) |
|---|---|
| bon | 0.15 + 0 |
| déf. | 0.45 + 0 |

Potentielles C

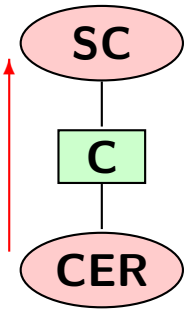
| C | E | R | $\Phi$(C, E, R) |
|---|---|---|---|
| bon | bon | bon | 0 |
| bon | bon | déf. | 0 |
| bon | ecranbleu | bon | (0.025 * 0.15)/ 1 |
| bon | ecranbleu | déf. | (0.1 * 0.15)/ 1 |
| bon | bloqué | bon | 0 |
| bon | bloqué | déf. | 0 |
| déf. | bon | bon | 0 |
| déf. | bon | déf. | 0 |
| déf. | ecranbleu | bon | (0.125 * 0.45)/ 1 |
| déf. | ecranbleu | déf. | (0.125 * 0.45)/ 1 |
| déf. | bloqué | bon | 0 |
| déf. | bloqué | déf. | 0 |

Potentielles CER

**Figure 2.9:** *L'envoi des messages pour l'exemple "défaillance matérielle d'un ordinateur".*

ii. propagation des messages : après réception de tous les messages, la clique C propage des messages vers toutes les autres cliques. Les mises à jour des probabilités potentielles se font comme dans la phase 1.

La Figure 2.10 illustre la propagation des messages en prenant comme clique choisie CER dans l'exemple de la défaillance matérielle d'un ordinateur.




Arbre de jonction

| C | E | R | $\Phi$(C, E, R) |
|---|---|---|---|
| bon | bon | bon | 0 |
| bon | bon | déf. | 0 |
| bon | ecranbleu | bon | 0.0375 |
| bon | ecranbleu | déf. | 0.015 |
| bon | bloqué | bon | 0 |
| bon | bloqué | déf. | 0 |
| déf. | bon | bon | 0 |
| déf. | bon | déf. | 0 |
| déf. | ecranbleu | bon | 0.005625 |
| déf. | ecranbleu | déf. | 0.005625 |
| déf. | bloqué | bon | 0 |
| déf. | bloqué | déf. | 0 |

Potentielles CER

| C | $\Phi$(C) |
|---|---|
| bon | 0.0375 + 0.015 |
| déf. | 0.005625 + 0.005625 |

Potentielles C

| S | C | $\Phi$(S,C) |
|---|---|---|
| oui | bon | (0.15 * 0.0525) / 0.15 |
| oui | déf. | (0.45 * 0.01125) / 0.45 |
| non | bon | 0 |
| non | déf. | 0 |

Potentielles SC

**Figure 2.10:** *La propagation des messages pour l'exemple "défaillance matérielle d'un ordinateur".*

(c) La normalisation : pour obtenir les probabilités conditionnelles d'une variable $X(x_0, x_1, \dots)$ selon les évidences $e$, il faut déduire les probabilités marginales à partir de la propagation globale pour avoir :
$P(X = x_i|e) = \frac{P(X=x_i,e)}{P(e)}, i \in \{0, 1, \dots\}$
Où $P(e) = \sum_{C|e} \Phi_C$, où $C$ est une clique contenant $e$,
et $P(X = x_i, e) = \sum_{C|e,x_i} \Phi_C$ où $C$ est une clique contenant $X$.
Du résultat de la propagation globale, nous pouvons déduire pour l'exemple de la défaillance matérielle d'un ordinateur que :
$P(e) = P(E = ecranbleu, S = oui) = 0.0375 + 0.015 + 0.005625 + 0.005625$, et $P(R = déf., e) = P(R = déf., E = ecranbleu, S = oui) = 0.015 + 0.005625$. Ce qui donne :
$P(R = déf.|E = ecranbleu, S = oui) = \frac{0.015+0.005625}{0.0375+0.015+0.005625+0.005625} = \frac{0.020625}{0.06375} = 0.543$

Le calcul d'inférence par arbre de jonction a une complexité d'ordre $O(n \exp(w))$ où $n$ est le nombre de nœuds et $w$ est la largeur de l'arbre de jonction. Plusieurs études ont été faites pour l'optimisation de ce calcul, par exemple les optimisations

du calcul d'inférence par élimination de variables [Chavira and Darwiche, 2007]. Nous trouvons aussi des travaux pour la parallélisation du calcul d'inférence et la conception d'accélérateurs matériels, que nous détaillons dans le chapitre suivant.

### 2.3.2 L'inférence par circuit arithmétique

Les algorithmes basés sur l'AC sont puissants et peuvent fournir des performances en temps réel [Darwiche, 2003]. L'inférence par AC se base sur la factorisation des fonctions multilinéaires et la dérivée partielle. Cette approche comprend deux phases. Une première phase où le réseau Bayésien est converti vers un circuit arithmétique (AC), et une deuxième phase de calcul sur l'AC.

1. La construction de l'AC
   La construction de l'AC peut se faire de deux façons, en se basant sur l'arbre de jonction ou en utilisant les fonctions multilinéaires (MLF). Ici, nous détaillons la deuxième méthode avec les deux points suivants :
   - La génération d'un polynôme : qui correspond à la définition de la fonction multilinéaire du réseau Bayésien.
   - La factorisation du polynôme : qui permet de simplifier le polynôme pour construire un AC optimal.

   (a) La génération d'un polynôme :
   Plusieurs méthodes existent pour la représentation en AC [Lian et al., 2011; Chavira and Darwiche, 2005], nous détaillons ici le principe général et nous exposons notre méthode de génération dans le Chapitre 4.
   Pour chaque réseau Bayésien, une MLF unique (notée $f$) est définie (Equation 2.7) :

$$f = \sum_{x} \prod_{xu \sim x} \lambda_x \theta_{x|u} \tag{2.7}$$

   - $\lambda_x$ (indicateurs d'évidence) : l'ensemble des évidences.
   - $\theta_{x|u}$ (Paramètres du réseau) : les valeurs des probabilités associées aux variables $X$ et $U$.
   - $\sim$ désigne la relation de compatibilité entre les instanciations [Darwiche, 2003].

   Comme pour l'arbre de jonction, pour chaque variable $X(x_0, x_1, \dots)$ du réseau, les indicateurs d'évidence sont définis comme suit :
   - Si $X$ est une observation : $\lambda x_i = 1, i \in \{0, 1, \dots\}$ lorsque $x_i$ est l'état observé par $X$, 0 sinon.

— Si $X$ n'est pas une observation : $\lambda x_i = 1, i \in \{0, 1, \dots\}$ pour tous les états $x_i$ de $X$.

$f$ est le polynôme représentant le réseau Bayésien, qui permet de calculer les probabilités des évidences "e" ($f(e) = P(e)$).

Pour notre exemple de la défaillance matérielle d'un ordinateur, notons :

— *RAM(bon, défaillant)*=$R(r_0, r_1)$.

— *CPU(bon, défaillant)*=$C(c_0, c_1)$.

— *Etat ordinateur (bon, ecranbleu, bloqué)*=$E(e_0, e_1, e_2)$.

— *Surchauffe (oui, non)*= $S(s_0, s_1)$.

Le polynôme correspondant est défini comme suit (Equation 2.8) :

$$\begin{aligned} f = {} & \lambda_{r_0}\theta_{r_0}\lambda_{c_0}\theta_{c_0}\lambda_{e_0}\theta_{e_0|r_0c_0}\lambda_{s_0}\theta_{s_0|c_0} + \lambda_{r_0}\theta_{r_0}\lambda_{c_0}\theta_{c_0}\lambda_{e_0}\theta_{e_0|r_0c_0}\lambda_{s_1}\theta_{s_1|c_0} + \\ & \lambda_{r_0}\theta_{r_0}\lambda_{c_0}\theta_{c_0}\lambda_{e_1}\theta_{e_1|r_0c_0}\lambda_{s_0}\theta_{s_0|c_0} + \lambda_{r_0}\theta_{r_0}\lambda_{c_0}\theta_{c_0}\lambda_{e_1}\theta_{e_1|r_0c_0}\lambda_{s_1}\theta_{s_1|c_0} + \\ & \lambda_{r_0}\theta_{r_0}\lambda_{c_0}\theta_{c_0}\lambda_{e_2}\theta_{e_2|r_0c_0}\lambda_{s_0}\theta_{s_0|c_0} + \lambda_{r_0}\theta_{r_0}\lambda_{c_0}\theta_{c_0}\lambda_{e_2}\theta_{e_2|r_0c_0}\lambda_{s_1}\theta_{s_1|c_0} + \\ & \lambda_{r_0}\theta_{r_0}\lambda_{c_1}\theta_{c_1}\lambda_{e_0}\theta_{e_0|r_0c_1}\lambda_{s_0}\theta_{s_0|c_1} + \lambda_{r_0}\theta_{r_0}\lambda_{c_1}\theta_{c_1}\lambda_{e_0}\theta_{e_0|r_0c_1}\lambda_{s_1}\theta_{s_1|c_1} + \\ & \lambda_{r_0}\theta_{r_0}\lambda_{c_1}\theta_{c_1}\lambda_{e_1}\theta_{e_1|r_0c_1}\lambda_{s_0}\theta_{s_0|c_1} + \lambda_{r_0}\theta_{r_0}\lambda_{c_1}\theta_{c_1}\lambda_{e_1}\theta_{e_1|r_0c_1}\lambda_{s_1}\theta_{s_1|c_1} + \\ & \lambda_{r_0}\theta_{r_0}\lambda_{c_1}\theta_{c_1}\lambda_{e_2}\theta_{e_2|r_0c_1}\lambda_{s_0}\theta_{s_0|c_1} + \lambda_{r_0}\theta_{r_0}\lambda_{c_1}\theta_{c_1}\lambda_{e_2}\theta_{e_2|r_0c_1}\lambda_{s_1}\theta_{s_1|c_1} + \\ & \lambda_{r_1}\theta_{r_1}\lambda_{c_0}\theta_{c_0}\lambda_{e_0}\theta_{e_0|r_1c_0}\lambda_{s_0}\theta_{s_0|r_0} + \lambda_{r_1}\theta_{r_1}\lambda_{c_0}\theta_{c_0}\lambda_{e_0}\theta_{e_0|r_1c_0}\lambda_{s_1}\theta_{s_1|c_0} + \\ & \lambda_{r_1}\theta_{r_1}\lambda_{c_0}\theta_{c_0}\lambda_{e_1}\theta_{e_1|r_1c_0}\lambda_{s_0}\theta_{s_0|c_0} + \lambda_{r_1}\theta_{r_1}\lambda_{c_0}\theta_{c_0}\lambda_{e_1}\theta_{e_1|r_1c_0}\lambda_{s_1}\theta_{s_1|c_0} + \\ & \lambda_{r_1}\theta_{r_1}\lambda_{c_0}\theta_{c_0}\lambda_{e_2}\theta_{e_2|r_1c_0}\lambda_{s_0}\theta_{s_0|c_0} + \lambda_{r_1}\theta_{r_1}\lambda_{c_0}\theta_{c_0}\lambda_{e_2}\theta_{e_2|r_1c_0}\lambda_{s_1}\theta_{s_1|c_0} + \\ & \lambda_{r_1}\theta_{r_1}\lambda_{c_1}\theta_{c_1}\lambda_{e_0}\theta_{e_0|r_1c_1}\lambda_{s_0}\theta_{s_0|c_1} + \lambda_{r_1}\theta_{r_1}\lambda_{c_1}\theta_{c_1}\lambda_{e_0}\theta_{e_0|r_1c_1}\lambda_{s_1}\theta_{s_1|c_1} + \\ & \lambda_{r_1}\theta_{r_1}\lambda_{c_1}\theta_{c_1}\lambda_{e_1}\theta_{e_1|r_1c_1}\lambda_{s_0}\theta_{s_0|c_1} + \lambda_{r_1}\theta_{r_1}\lambda_{c_1}\theta_{c_1}\lambda_{e_1}\theta_{e_1|r_1c_1}\lambda_{s_1}\theta_{s_1|c_1} + \\ & \lambda_{r_1}\theta_{r_1}\lambda_{c_1}\theta_{c_1}\lambda_{e_2}\theta_{e_2|r_1c_1}\lambda_{s_0}\theta_{s_0|c_1} + \lambda_{r_1}\theta_{r_1}\lambda_{c_1}\theta_{c_1}\lambda_{e_2}\theta_{e_2|r_1c_1}\lambda_{s_1}\theta_{s_1|c_1}. \end{aligned} \tag{2.8}$$

Ce polynôme nous permet de calculer les probabilités des évidences $P(e)$. Par exemple dans le cas où l'état de l'ordinateur donne un écran bleu et une présence de surchauffe ($E = e_1, S = e_0$), le calcul de P(e) se fait comme suit (Equation 2.9) :

$$\begin{aligned} P(e) &= f(\lambda_{r_0} = 1, \lambda_{r_1} = 1, \lambda_{c_0} = 1, \lambda_{c_1} = 1, \lambda_{e_0} = 0, \lambda_{e_1} = 1, \lambda_{e_2} = 0, \\ &\quad \lambda_{s_0} = 1, \lambda_{s_1} = 0) \\ &= \theta_{r_0}\theta_{c_0}\theta_{e_1|r_0c_0}\theta_{s_0|c_0} + \theta_{r_0}\theta_{c_1}\theta_{e_1|r_0c_1}\theta_{s_0|c_1} + \theta_{r_1}\theta_{c_0}\theta_{e_1|r_1c_0}\theta_{s_0|c_0} \\ &\quad + \theta_{r_1}\theta_{c_1}\theta_{e_1|r_1c_1}\theta_{s_0|c_1} \\ &= 0.5 * 0.5 * 0.1 * 0.3 + 0.5 * 0.5 * 0.5 * 0.9 + 0.5 * 0.5 * 0.4 * 0.3 \\ &\quad + 0.5 * 0.5 * 0.5 * 0.9 \\ &= 0.2625 \end{aligned} \tag{2.9}$$

(b) La factorisation et la construction de l'AC :
Le polynôme peut être simplifié en utilisant la factorisation puis représenté par un arbre (AC). Les nœuds feuilles de l'AC sont les $\lambda$ et les $\theta$ et les nœuds internes sont des multiplications (*) ou des additions (+) selon le polynôme après factorisation. L'AC n'est pas unique, sa représentation dépend de la factorisation.
Dans l'exemple précédent, le polynôme après factorisation peut être le suivant (Equation 2.10) :

$$\begin{aligned} f = \; & \lambda_{c_0}\theta_{c_0}(\lambda_{r_0}\theta_{r_0}(\lambda_{e_0}\theta_{e_0|r_0c_0} + \lambda_{e_1}\theta_{e_1|r_0c_0} + \lambda_{e_2}\theta_{e_2|r_0c_0})+ \\ & \lambda_{r_1}\theta_{r_1}(\lambda_{e_0}\theta_{e_0|r_1c_0} + \lambda_{e_1}\theta_{e_1|r_1c_0} + \lambda_{e_2}\theta_{e_2|r_1c_0}))(\lambda_{s_0}\theta_{s_0|c_0} + \lambda_{s_1}\theta_{s_1|c_0})+ \\ & \lambda_{c_1}\theta_{c_1}(\lambda_{r_0}\theta_{r_0}(\lambda_{e_0}\theta_{e_0|r_0c_1} + \lambda_{e_1}\theta_{e_1|r_0c_1} + \lambda_{e_2}\theta_{e_2|r_0c_1})+ \\ & \lambda_{r_1}\theta_{r_1}(\lambda_{e_0}\theta_{e_0|r_1c_1} + \lambda_{e_1}\theta_{e_1|r_1c_1} + \lambda_{e_2}\theta_{e_2|r_1c_1}))(\lambda_{s_0}\theta_{s_0|c_1} + \lambda_{s_1}\theta_{s_1|c_1}) \end{aligned} \tag{2.10}$$

La Figure 2.11 donne la représentation de l'AC à partir de la fonction multilinéaire factorisée.

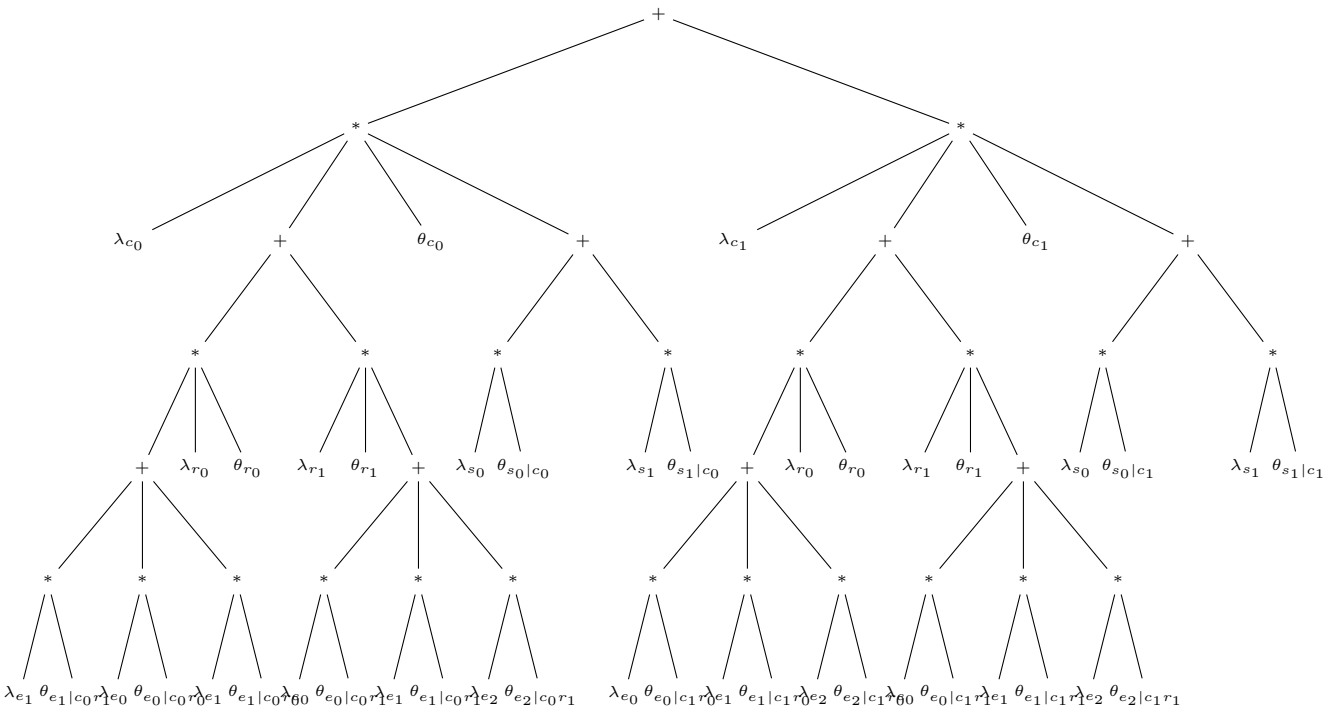



**Figure 2.11:** *Le circuit arithmétique pour l'exemple "défaillance matérielle d'un ordinateur".*

2. Le calcul d'inférence par l'AC
   A partir de l'AC ou du polynôme, le calcul des probabilités conditionnelles $P(X|e)$ ($X$ une variable et $e$ des évidences) peut se faire en se basant sur la dérivée partielle selon le théorème suivant :

   **Théorème 2.3.1**
   *Soit un réseau Bayésien ayant un polynôme $f$. Pour chaque variable $X$ et toutes évidences $e$ :*
   $\frac{\partial f}{\partial \lambda_x}(e) = P(x, e - X)$. *($e - X$ : les évidences sans la variable $X$)*

   De ce théorème, on peut déduire : $P(x|e) = \frac{1}{f(e)} \frac{\partial f}{\partial \lambda_x}(e)$ (rappel : $P(x|e) = \frac{P(x,e)}{P(e)}$.)
   Dans l'exemple précédent, pour calculer, par exemple :
   $P(R = déf.|E = ecranbleu, S = oui)$ $(= P(r_1|e_1 s_0))$, nous avons :

$$\begin{aligned} P(r_1, e_1, s_0) =& \frac{\partial f}{\partial \lambda_{r_1}}(e_1, s_1) \\ &= \lambda_{c_0}\theta_{c_0}\theta_{r_1}(\lambda_{e_0}\theta_{e_0|r_1c_0} + \lambda_{e_1}\theta_{e_1|r_1c_0} + \lambda_{e_2}\theta_{e_2|r_1c_0})(\lambda_{s_0}\theta_{s_0|c_0} + \lambda_{s_1}\theta_{s_1|c_0}) + \\ &\quad \lambda_{c_1}\theta_{c_1}\theta_{r_1}(\lambda_{e_0}\theta_{e_0|r_1c_1} + \lambda_{e_1}\theta_{e_1|r_1c_1} + \lambda_{e_2}\theta_{e_2|r_1c_1})(\lambda_{s_0}\theta_{s_0|c_1} + \lambda_{s_1}\theta_{s_1|c_1}) \\ &= \theta_{c_0}\theta_{r_1}\theta_{e_1|r_1c_0}\theta_{s_1|c_0} + \theta_{c_1}\theta_{r_1}\theta_{e_1|r_1c_1}\theta_{s_1|c_1} \\ &= 0.5 * 0.5 * 0.4 * 0.3 + 0.5 * 0.5 * 0.5 * 0.9 \\ &= 0.1425 \end{aligned} \tag{2.11}$$

   Et donc $P(r_1|e_1 s_0) = \frac{0.1425}{0.2625} = 0.543$

**Analyse**

Nous pouvons constater que le calcul explose par rapport au nombre de nœuds. En se basant sur la compilation en AC, une méthode proposée dans [Darwiche, 2003] a été développée permettant de calculer les probabilités conditionnelles de tous les nœuds à la fois. Cette approche se base sur l'évaluation et la dérivée partielle par passage de message dans le circuit arithmétique en deux phases :

- Upward-pass : Evaluation de l'AC. Résultat : $f(e)$.
- Downward-pass : Compilation de l'AC par dérivée partielle : Résultat : $\frac{\partial f}{\partial \lambda_x}(e)$.

Chaque nœud $v$ du circuit a deux registres $vr(v)$ et $dr(v)$. L'algorithme de calcul d'inférence est le suivant :

- Initialisation : mettre $dr(v) = 0$ pour tous les nœuds sauf le nœud racine où $dr(v) = 1$.

— Upward-pass : à chaque nœud $v$, calculer sa valeur et sauvegarder la dans $vr(v)$. Ce calcul se fait en évaluant l'AC de bas en haut et en sauvegardant les valeurs intermédiaires dans les registres des nœuds correspondants.

— Downward-pass : à chaque nœud $v$, et pour chaque parent $p$ :
$dr(v) = dr(p)$ si $v$ est un nœud addition,
$dr(v) = dr(p) * \prod_{v'} vr(v')$ si $v$ est un nœud multiplication et où les v' sont les autres fils de $p$ (vue comme une dérivée par rapport à $v$).

**Constatation**

Nous pouvons constater que cette méthode est plus adaptée à des implémentations matérielles, par rapport à la facilité de calcul et la possibilité de parallélisation, contrairement à la méthode de l'arbre de jonction où il est difficile d'implémenter la triangulation et la moralisation. Des optimisations de calcul et le parallélisme sont envisageables sur ce genre de calcul. Ces deux points seront abordés dans les chapitres suivants.

## 2.4 Extension des réseaux Bayésiens

Nous pouvons trouver plusieurs extensions des réseaux Bayésiens telles que :

— Les réseaux Bayésiens dynamiques : en ajoutant l'aspect temporel [Murphy, 2002].

— Les diagrammes d'influence [Jensen and Nielsen, 2007] : pour faire de la décision.

— Les réseaux Bayésiens orientés objets ou flous [Koller and Pfeffer, 1997] : en combinaison avec la notion d'objet ou la théorie des ensembles flous.

— Les réseaux Bayésiens distribués [Valtorta et al., 2002] : en divisant le réseau Bayésien en plusieurs sous-réseaux.

Dans cette section, nous présentons les deux premiers points que nous utilisons dans nos travaux.

### 2.4.1 Réseaux Bayésiens dynamiques

Un réseau Bayésien dynamique est un réseau Bayésien intégrant l'évaluation temporelle des variables [Dean and Kanazawa, 1989; Murphy, 2002]. Cela signifie, que l'état d'une variable à l'instant présent peut être influencé par l'état à un instant passé. De nombreuses applications ont besoin de ce type de modèle, telles que la reconnaissance de la parole [Bach and Jordan, 2005], l'apprentissage de réseaux génétiques, etc.

Nous pouvons construire un réseau Bayésien dynamique pour l'exemple de la défaillance matérielle d'un ordinateur, en supposant que le CPU et la RAM ont un système de vieillissement. La Figure 2.12 représente le réseau dynamique Bayésien correspondant, qui consiste à dupliquer le réseau Bayésien précédemment défini et rajouter les liens entre les variables à chaque temps en définissant les tables de probabilités.

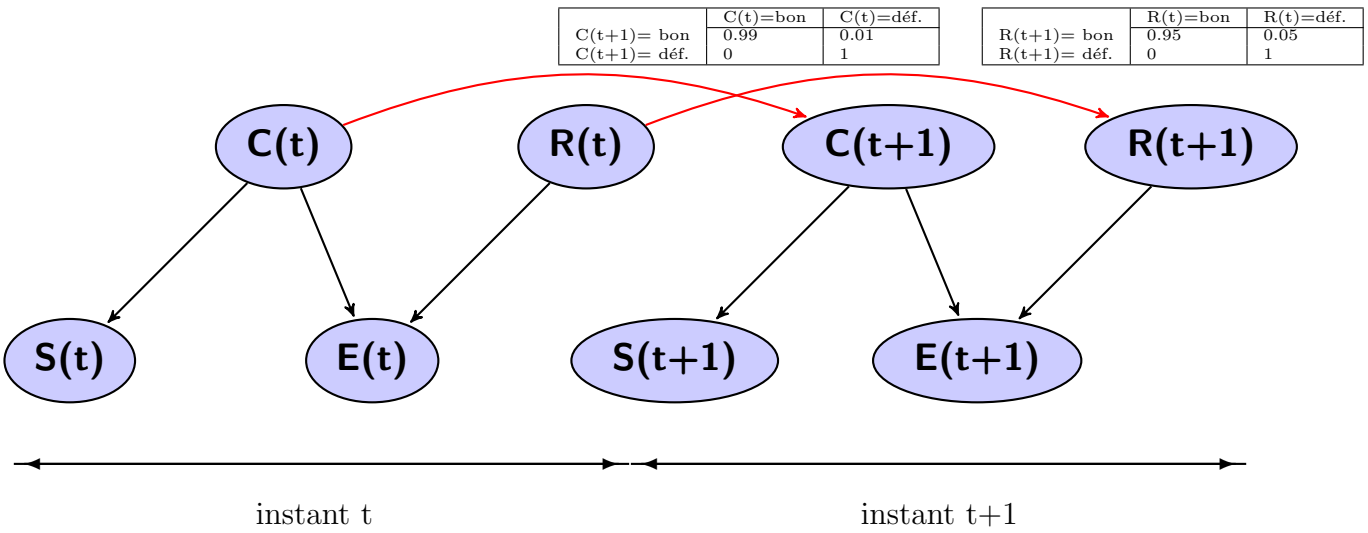



**Figure 2.12:** *Le réseau Bayésien dynamique pour l'exemple "défaillance matérielle d'un ordinateur".*

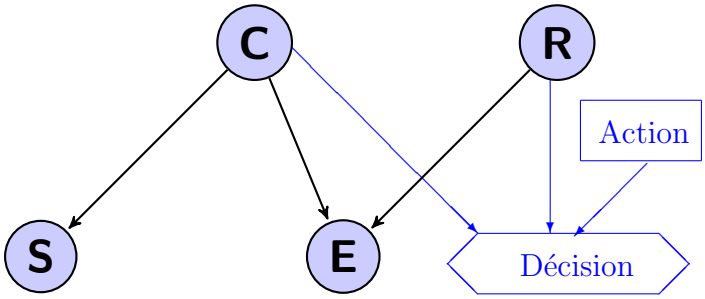
### 2.4.2 Diagramme d'influence

Les diagrammes d'influence sont des extensions des réseaux Bayésiens qui permettent la modélisation de la décision en situation d'incertitude [Chan and Shachter, 1992]. Un diagramme d'influence est composé de :

- **Nœuds chance** : qui sont les nœuds probabilistes contenant les tables de probabilités.
- **Arcs** : reliant les différents nœuds chance selon les dépendances conditionelles.
- **Nœuds action** : qui sont des nœuds contenant les différentes actions pour la décision.
- **Nœuds décision** : qui sont les nœuds reliés aux nœuds chance et aux nœuds action, et contenant une table d'utilité représentant les différents scores pour chaque action et différents états des nœuds chance.

Résoudre le diagramme d'influence revient à trouver l'action maximisant une fonction d'utilité [Jensen and Nielsen, 2007].

Nous pouvons imaginer un diagramme d'influence pour l'exemple de la défaillance matérielle d'un ordinateur. Nous pouvons rajouter une décision simple qui consiste à ne rien faire si la RAM et le CPU sont bons et sinon changer la RAM ou le CPU. Pour cela, nous rajoutons au réseau Bayésien un nœud action contenant les 3 actions possibles, ainsi qu'un nœud décision avec sa table d'utilité, contenant les différents scores des actions selon les états de la RAM et du CPU. La Figure 2.13 représente le diagramme d'influence de cet exemple.




| Action |
|---|
| Rien faire ($A_0$) |
| Changer RAM ($A_1$) |
| Changer CPU ($A_2$) |

| C | $c_0$ | $c_0$ | $c_0$ | $c_0$ | $c_0$ | $c_0$ | $c_1$ | $c_1$ | $c_1$ | $c_1$ | $c_1$ | $c_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | $r_0$ | $r_0$ | $r_0$ | $r_1$ | $r_1$ | $r_1$ | $r_0$ | $r_0$ | $r_0$ | $r_1$ | $r_1$ | $r_1$ |
| Action | $A_0$ | $A_1$ | $A_2$ | $A_0$ | $A_1$ | $A_2$ | $A_0$ | $A_1$ | $A_2$ | $A_0$ | $A_1$ | $A_2$ |
| Utilité | 100 | 0 | 0 | 0 | 100 | 0 | 0 | 0 | 100 | 0 | 50 | 50 |

**Figure 2.13:** *Diagramme d'influence pour l'exemple "défaillance matérielle d'un ordinateur".*

La fonction d'utilité est calculée pour chaque action. Dans l'exemple de la Figure 2.13, nous avons besoin de la probabilité des nœuds C et R, ainsi que de la table d'utilité. L'action choisie est celle qui maximise la fonction d'utilité. Le calcul se fait comme suit :

$$\text{Action choisie} = Max(U\_f_{A_0}, U\_f_{A_1}, U\_f_{A_2})$$

où chaque $U\_f_k(k = \{A_0, A_1, A_2\})$ représente la valeur de la fonction d'utilité de l'action $k$, et est égale à :

$$U\_f_k = \sum_i \sum_j U(A = k, C = i, R = j) * P(C = i) * P(R = j)$$
$$(i = \{c_0, c_1\} \quad et \quad j = \{r_0 r_1\})$$

Si nous supposons le cas d'une observation d'un écran bleu et d'une surchauffe, nous aurons par calcul d'inférence les probabilités suivantes :
$P(C = c_0 | S = s_0, E = e_1) = 0.143$
$P(C = c_1 | S = s_0, E = e_1) = 0.857$
$P(R = r_0 | S = s_0, E = e_1) = 0.457$

$P(R = r_1|S = s_0, E = e_1) = 0.543$
Et par conséquence nous aurons les valeurs de la fonction d'utilité suivantes :
$U\_f_A\_0 = 2.87$
$U\_f_A\_1 = 32.85$
$U\_f_A\_2 = 64.28$
Ce qui signifie pour ce cas que : Action choisie = $A_2$ , qui correspond à l'action changer le CPU.

## 2.5 Les réseaux Bayésiens dans le diagnostic et la prise de décision

Les réseaux Bayésiens sont largement utilisés pour mettre en œuvre le diagnostic complexe dans divers types de systèmes [Pearl and Russell, 1998; Darwiche, 2010; Ricks and Mengshoel, 2014]. Ils représentent des modèles de diagnostic efficace par rapport à la gestion de données incomplètes, incertaines et bruitées. Différentes études sur le diagnostic avec les réseaux Bayésiens ont été développées dans la littérature telle que le diagnostic par réseaux Bayésiens sur les systèmes d'alimentation électrique [Kaspi and Ratsaby, 2012]. Aussi dans [Bottone et al., 2008], un réseau Bayésien pour un système spécifique de surveillance et de contrôle d'une borne de terre par satellite a été présenté. Dans [Zheng and Mrad, 2013], la fiabilité des systèmes logiciels pour la surveillance des applications électriques avec des réseaux Bayésiens a été abordée. Aussi pour compléter le diagnostic par une décision ou un pronostic, les extensions des réseaux Bayésiens sont largement utilisées [Codetta-Raiteri and Portinale, 2015]. Deux références majeures en comptabilité avec notre thématique sont détaillées dans cette partie. La première consiste en l'étude de la gestion de l'état de santé des capteurs et des logiciels pour des drones à l'aide d'un modèle Bayésien [Schumann et al., 2013b, 2015, 2013a, 2011]. La seconde comprend la détection de défaillances, la décision et le pronostic dans les systèmes autonomes à l'aide des réseaux Bayésiens dynamiques combinés aux diagrammes d'influence [Codetta-Raiteri and Portinale, 2015; Portinale and Codetta-Raiteri, 2011].

### 2.5.1 La gestion de l'état de santé des capteurs et des logiciels pour des drones

Les recherches dans ce domaine sont faites par la NASA, où la fiabilité du système est un facteur important car un léger dysfonctionnement peut avoir des conséquences désastreuses. En outre, ces systèmes opèrent dans des environnements incertains et doivent faire face à des défaillances matérielles ou logicielles. Il est alors nécessaire d'évaluer continuellement l'état de santé du système pour vérifier la fiabilité de la mission et pouvoir agir en temps réel, dans le but de prendre des décisions qui

maximisent les capacités à répondre aux objectifs de la mission, tout en maintenant les exigences de sécurité. Dans les dernières études, un modèle Bayésien fiable et robuste pour le diagnostic a été embarqué pour le monitoring en temps réel des comportements des logiciels et matériels, dans le but d'identifier les causes les plus probables dans le cas d'une défaillance afin de pouvoir réagir par la suite. Ici, nous exposons le modèle Bayésien utilisé pour le diagnostic et son implémentation sera développée dans le chapitre suivant.

Un modèle "Glues together" a été proposé (voir Figure 2.14) [Schumann et al., 2013a]. Ce modèle représente les relations de causalité et le comportement de différents composants matériels/logiciels pour une tâche donnée avec une certaine incertitude.
Le réseau Bayésien comporte quatre types de nœuds :

- **Nœuds état U :** reflètent les états non observables du système ou du composant.
- **Nœuds commande C** : représentent des actions ou des commandes extérieures.
- **Nœuds capteur S** : indiquent des mesures du processus par des capteurs logiciels ou matériels.
- **Nœuds santé H** : reflétent l'état de santé du système (H_U) ou l'état de santé des capteurs (H_S).

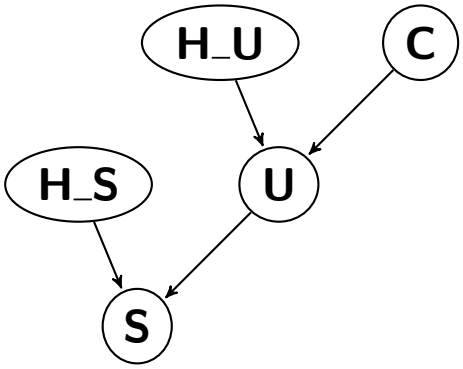



**Figure 2.14:** *Le modèle Bayésien "Glues together".*

Les tables de probabilités et les états des nœuds sont définis selon les cas d'étude. Les entrées sont les observations sur les nœuds capteurs et les nœuds commandes, et la sortie est la probabilité a posteriori du nœud état de santé $P(H\_U|C, S)$.

La Figure 2.15 représente un exemple simple illustrant le modèle exposé ci-dessus. Le modèle représente un monitoring de Pitch-up ou de Pitch-down dans un drone. Le nœud $C\_up$ $(oui, non)$ représente la commande de Pitch-up et le nœud $C\_down$ $(oui, non)$ représente la commande de pitch-down. Le nœud capteur $S\_acc$ $(augmentation, diminution)$ est un accéléromètre. Sa valeur augmente lorsque le drone est en Pitch-up et elle diminue lorsque le drone est en Pitch-down. Le nœud $U\_up$ $(oui, non)$ représente l'état inobservable du Pitch-up, et le nœud $U\_down$ $(oui, non)$ représente l'état inobservable du Pitch-down. Le nœud $H\_up$ $(Bon, défaillant)$ surveille l'état de santé du Pitch-up, le nœud $H\_down$ $(Bon, défaillant)$ surveille l'état de santé du Pitch-down, et le nœud $H\_acc$ $(Bon, défaillant)$ surveille l'état de santé de l'accéléromètre.

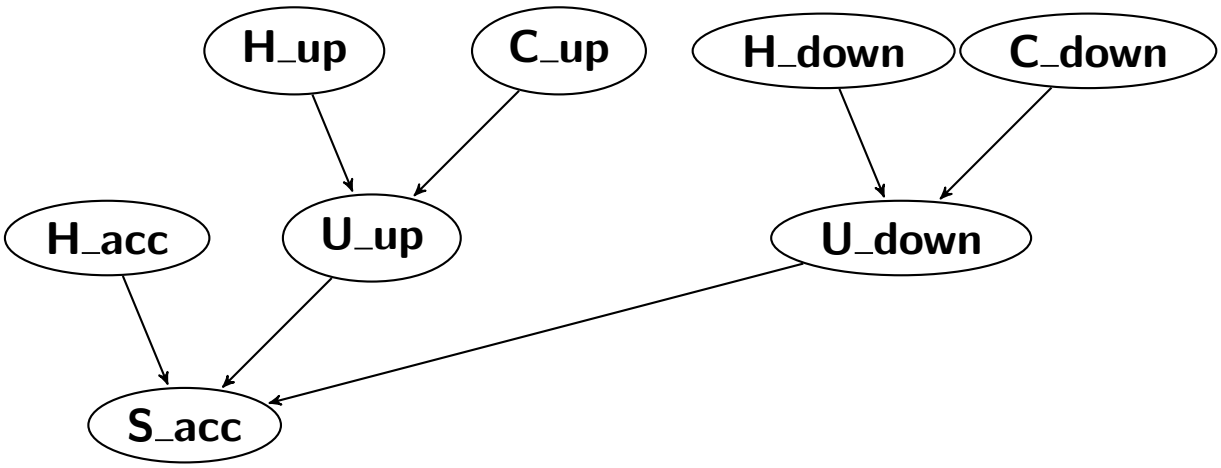



**Figure 2.15:** *Le modèle Bayésien "Glues together" pour l'exemple de Pitch-up et Pitch-down.*

#### Synthèse

Ce modèle n'a pas été automatisé et il est construit par expertise. Dans nos travaux, nous nous basons sur ce modèle enrichi par une analyse de défaillance pour une construction automatique des réseaux Bayésiens pour l'état de santé.

### 2.5.2 La détection de défaillances, la décision et le pronostic dans les systèmes autonomes

Les recherches dans ce domaine sont faites pour des engins spatiaux autonomes, dans le but d'améliorer les méthodes classiques de détection, d'identification et de recouvrement "FDIR" (fault detection, identification and recovery). Les méthodes FDIR classiques se basent sur des analyses de défaillance (Analyse des modes de défaillance et de leurs effets FMEA [Automotive Industry Action Group, 1993; McDermott et al., 1999], arbre de défaillance FTA [Schneeweiss, 1999]) et les observations sur l'état de fonctionnement du système. Le but de ces approches est la détection en temps réel des défaillances et l'application par la suite des actions de recouvrement pour mettre l'engin dans une configuration sûre. Avec les réseaux Bayésiens dynamiques et les diagrammes d'influence, les aspects pronostic et décision autonome sont rajoutés aux méthodes FDIR classiques.

Dans les derniers travaux intitulés ARPHA (Anomaly Resolution and Prognostic Health management for Autonomy), un réseau Bayésien dynamique est proposé pour faire le pronostic et le diagnostic des états des composants lors d'une mission, utilisant des observations sur des capteurs. Ce réseau Bayésien est rattaché à un ensemble d'actions de recouvrement qui peuvent être exécutés d'une manière autonome. Une table d'utilité, avec le score de chaque action selon les cas possibles

est définie, permettant de calculer à chaque instant, avec un diagramme d'influence, l'action à choisir dans le cas d'un problème.

#### Synthèse

L'application de cette approche a été réalisée pour le système d'alimentation électrique du Rover ExoMars avec 4 scenarios de défaillance et des actions de recouvrement, détaillés dans [Codetta-Raiteri and Portinale, 2015]. Ces travaux sont dédiés aux missions d'engins spatiaux et pour une conception logicielle en se basant sur le calcul d'inférence par arbre de jonction.

## 2.6 Conclusion

Dans ce chapitre, nous avons exposé un état de l'art sur les réseaux Bayésiens. Nous avons commencé par des définitions et des généralités sur les réseaux Bayésiens. Nous avons aussi abordé la construction des réseaux Bayésiens et le théorème de Bayes. Nous avons par la suite présenté des algorithmes d'inférence, permettant la résolution des réseaux Bayésiens en se basant sur le théorème de Bayes. Nous avons aussi donné des extensions des réseaux Bayésiens, pour finir par montrer l'utilisation des réseaux Bayésiens dans le diagnostic et la décision.

Le chapitre montre l'intérêt des réseaux Bayésiens et leur adéquation avec nos travaux. En revanche, la construction d'un réseau Bayésien reste difficile. Dans les chapitres suivants, nous donnons notre modèle pour l'état de santé des systèmes autonomes en nous basant sur une analyse de défaillances. Aussi nous avons montré que les calculs dans les réseaux Bayésiens sont parallélisables et embarquables. Le chapitre suivant est consacré aux accélérateurs matériels, aux implémentations sur SoC, aux outils de synthèse de haut niveau, pour présenter les implémentations des réseaux Bayésiens sur SoC, introduisant nos contributions.

# - Chapitre 3 -

# Accélérateurs matériels et conception par synthèse de haut niveau

## Sommaire

## 3.1 Introduction

Les accélérateurs matériels sont utilisés afin d'alléger ou remplacer le processeur lorsque les taches sont couteuses en termes de performance et de consommation. Plusieurs types d'accélérateurs ont vu le jour et font l'objet de sujets de recherche, tel que les GPU (les processeurs graphiques), les DSP (processeurs de traitement numérique du signal), les ASIC, et les FPGA (circuits logiques programmables) qui font l'objet de notre travail. Dans la Section 3.2 de ce chapitre, nous développons les différents types d'accélérateurs matériels et nous montrons leurs intérêts puis nous nous focalisons sur les FPGA comme accélérateurs matériels.

Les réseaux Bayésiens par leur structure et complexité, ainsi que la multitude des algorithmes d'inférence sont des sujets d'étude pour le contexte de l'embarqué, où une implémentation efficace et optimale est nécessaire. Des implémentations parallèles du calcul d'inférence sur GPU et FPGA ont été réalisées dans différentes recherches [Kozlov and Singh, 1994; Zheng and Mengshoel, 2013; Schumann et al., 2013b]. Nous donnons un aperçu des travaux réalisés à la fin de la Section 3.2.

La conception d'un accélérateur matériel pour FPGA utilise traditionnellement une description au niveau RTL (transferts entre registres) utilisant des langages tels que VHDL et Verilog pour une conception très proche du circuit numérique dédié. Le développement au niveau RTL peut être très fastidieux, d'où l'apparition des techniques de synthèse de haut niveau (High Level Synthesis ou HLS). La HLS utilise une description de haut niveau, qui est convertie, par les outils HLS, à une description RTL. Les techniques de HLS permettent une génération rapide d'un RTL optimisé, tout en respectant des exigences de performance, de surface et de consommation énergétique. Dans la Section 3.3 de ce chapitre, nous détaillons les techniques des outils HLS d'une façon générale, puis nous détaillons l'outil Vivado HLS de Xilinx, que nous utilisons pour nos travaux, où nous exposons les différentes manières d'avoir un code optimal selon les besoins en ressource et en performance.

## 3.2 Accélération matérielle et FPGA

### 3.2.1 Définition et intérêt des accélérateurs matériels

Les accélérateurs matériels sont des unités de traitement spécifiques ajoutées à des systèmes à base de processeurs standards et sur lesquels peut être déportée une partie des calculs, afin d'obtenir un calcul plus efficace en termes de performance et de consommation. Par exemple l'affichage d'une vidéo, ou le décodage d'une image, où les calculs sont généralement coûteux pour le CPU. Différents types d'accélérateurs matériels existent, dont les plus répandus sont :

— les processeurs graphiques (Graphical Processing Unit ou GPU), plus utilisés

dans les calculs liés à l'affichage et dans le calcul scientifique [Owens et al., 2008],

— les processeurs de traitement numérique du signal (Digital Signal Processing ou DSP), plus utilisés dans les calculs sur signaux numériques comme le filtrage, la compression, etc [Antoniou, 2016],

— les circuits intégrés propres à une application (Application Specific Integrated Circuit ou ASIC), sont des puces intégralement dédiées à la réalisation d'une tâche précise et qui donnent un meilleur ratio consommation/performance mais sont plus couteux [Smith, 1997; Kuon and Rose, 2007],

— les circuits logiques programmables (Field Programmable Gate Array ou FPGA), qui sont des accélérateurs matériels très polyvalents [El-Ghazawi et al., 2008].

### 3.2.2 Les FPGA comme accélérateurs matériels

Parmi les différents accélérateurs existants, les FPGA émergent plus particulièrement pour leurs performances et la diversité d'applications [Monmasson, 2017]. Cependant, il est possible de concevoir un circuit pour une application puis le reconfigurer totalement ou bien partiellement et le programmer pour un autre type d'application [Pocek et al., 2013].

De plus, il est généralement plus intéressant en termes de consommation énergétique et de rapidité de calcul, d'utiliser des accélérateurs FPGA. Par exemple, dans [Kindratenko and Pointer, 2006], on montre qu'en utilisant des FPGA, une accélération de facteur trois peut être obtenue pour des applications dans le domaine de la biophysique fonctionnant dans des supercalculateurs.

Depuis 2015, il existe des FPGA qui possèdent des millions de cellules logiques, des blocs de calcul spécifiques, des capacités mémoires importantes, etc. Cela permet de concevoir toutes sortes d'applications, même les plus complexes. Aussi, les FPGA actuels offrent un bon niveau de performances pour un temps de développement très court. Ceci convient donc très bien au contexte de l'embarqué et du temps réel.

De nos jours, les FPGA sont en pleine croissance, avec deux fournisseurs principaux : Xilinx et Altera détenant environ 80% du marché.

#### Description des FPGA

Un FPGA (Field Programmable Gate Array), signifiant littéralement Matrice de Portes Logiques Programmables, est un circuit électronique programmable. Il a la particularité d'être programmé pour exécuter une tâche voulue et peut être reprogrammé plusieurs fois.

Chaque FPGA comporte un nombre fixe de ressources, composées de [Rose et al., 1990] :

— blocs logiques programmables,

— éléments d'interconnexion,

— broches d'entrées/sorties,

— éléments mémoires embarqués,

— blocs de calcul spécifiques.

L'architecture classique d'un FPGA est représentée dans la Figure 3.1. Les blocs logiques configurables (Configurable Logic Block ou CLB) sont les éléments de base. Chaque CLB se compose de bascules Flip-Flop (FF), de Look-Up Table (LUT) et de multiplexeurs (voir Figure 3.2). Les LUTs permettent d'implémenter des fonctions logiques de base (AND, OR, etc.) et les FFs servent à stocker des informations binaires. Les LUTs et les FFs sont regroupés dans des slices, permettant ainsi de générer tout type de circuit numérique [Betz et al., 1999]. Dans les FPGA actuels, les constructeurs ont ajouté des mémoires à accès direct (BRAM) pour faciliter le stockage d'information et des blocs multiplieurs accumulateurs (DSP) pour accélérer les calculs mathématiques. On peut également trouver des modules de gestion d'horloge (DCM pour Digital Clock Manager). Une matrice d'interconnexion connecte tous ces éléments (CLB, BRAM et DSP) et les blocs d'entrée/sortie (I/O) se chargent de la communication avec l'extérieur.

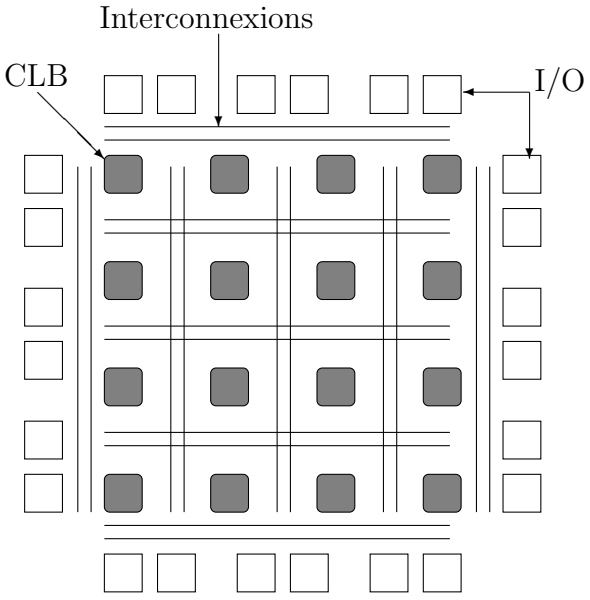



**Figure 3.1:** *Architecture d'un FPGA.*

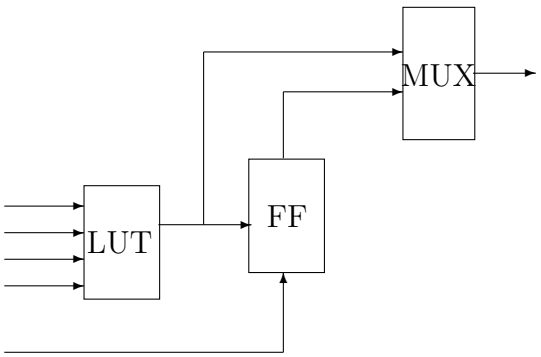



**Figure 3.2:** *Bloc logique configurable (CLB).*

### Place des FPGA dans le contexte de l'accélération matérielle

Malgré tous les avantages que présentent les FPGA et toutes les améliorations apportées ces dernières années [Pocek et al., 2013], leur position dans le monde de l'accélération matérielle reste faible. Au dela du coût de la reconfiguration, le développement pour FPGA demande des compétences techniques solides et un temps

considérable. En effet, la conception de circuits FPGA requiert l'utilisation du langage de bas niveau HDL (Hardware Description Language) comme VHDL et Verilog, classiquement non maitrisés par les développeurs. Pour améliorer ce dernier point, des techniques de synthèse de haut niveau sont utilisées, que nous détaillons dans la section suivante.

### 3.2.3 Accélération matérielle et implémentation sur FPGA des réseaux Bayésiens

Comme nous l'avons vu dans le chapitre précédent, les réseaux Bayésiens sont des modèles simples et efficaces. Cependant leur complexité augmente avec la densité et la largeur du réseau, ainsi qu'avec le nombre d'états de chaque nœud du réseau. Cette complexité augmente significativement la difficulté du calcul d'inférence. Afin d'utiliser des réseaux Bayésiens pour des problèmes en temps réel, il est important de développer des techniques pour accélérer le calcul. Une approche consiste à concevoir des accélérateurs matérielles spécialisés, vu la possibilité du parallélisme des réseaux Bayésiens.

Des implémentations matérielles et parallèles de réseaux Bayésiens ont été proposées au milieu des années 1990 pour l'inférence par arbres de jonction [Kozlov and Singh, 1994] sur les multiprocesseurs. Ensuite, à partir de 2008, des implémentations parallèles sur GPU ont été proposées dans [Silberstein et al., 2008; Zheng and Mengshoel, 2013], ce qui a permis une accélération intéressante. Par exemple, dans [Silberstein et al., 2008], un accélérateur GPU a été développé, parallélisant la propagation des messages dans l'arbre de jonction. Cette approche a été mise en œuvre sur un GPU NVIDIA et testée sur des réseaux Bayésiens de plusieurs applications. Avec cette solution, des accélérations allant de 0,68 à 9,18 ont été obtenues pour les réseaux Bayésiens étudiés.

Une implémentation FPGA de réseaux Bayésiens basée sur l'algorithme de l'arbre de jonction a été proposée dans [Kulesza and Tylman, 2006]. Dans cette dernière étude, deux composants spécifiques sont définis : le composant nœud et le composant séparateur. Le composant nœud est implémenté en tant que mémoire et le composant séparateur est équipé d'une unité arithmétique simplifiée capable de faire une addition, une multiplication, et une division. L'impact de la représentation des données (représentation en point fixe) sur la précision des résultats est évoquée mais pas entièrement évaluée.

Dans [Lin et al., 2010], une machine de calcul Bayésienne avec un débit élevé pour le calcul de l'inférence Bayésienne a été proposée. Cette machine est implémentée sur FPGA en utilisant des mémoires embarquées distribuées, mais en évitant les stalles de mémoire avec un schéma d'allocation de mémoire spécifique. La proposition d'architecture est basée sur un pipeline profond des nœuds de traitement et sur un planificateur optimal. Cette implémentation fonctionne avec une version adaptative en virgule flottante. Néanmoins, la qualité du calcul résultant dépend fortement de

la performance du compilateur/planificateur dédié.

Dans [Schumann et al., 2013b], une architecture reconfigurable dédiée basée sur les propriétés structurelles de la forme AC a été proposée.

L'architecture est composée d'un ensemble d'unités parallèles appelées blocs de calcul. Chaque bloc de calcul prend en charge une partie du calcul de l'AC et contient principalement les éléments suivants :

— une Unité Logique Arithmétique (ALU), qui s'occupe du calcul à partir de l'AC, avec des multiplications et additions,

— une mémoire BRAM pour les tables de probabilités et une mémoire BRAM pour les indicateurs d'évidence,

— une interface mémoire/ multiplexeur pour charger ou stocker des opérandes de et vers la mémoire, déclencher des transferts de résultats et coordonner des charges d'entrées,

— une unité de contrôle pour décoder et transmettre les instructions,

— une mémoire de bloc-notes pour enregistrer les résultats locaux intermédiaires, calculés au cours de la traversée ascendante de l'AC, pour être réutilisés pendant la traversée descendante.

La mise en œuvre choisie pour l'opérateur arithmétique est basée sur une représentation en point fixe de 18 bits pour être exécutée par le DSP intégré du FPGA. Les blocs communiquent via un bus spécifique pour obtenir l'inférence globale du réseau Bayésien. Le nombre de blocs à considérer dépend des ressources informatiques que l'utilisateur veut allouer.

Dans cette dernière étude, l'accélérateur matériel FPGA a été conçu dans le langage de description matérielle VHDL, sans outil de synthèse de haut niveau, en utilisant les outils de la synthèse logique ALTERA QUARTUS II.

## 3.3 Conception d'accélérateurs FPGA à partir de la synthèse de haut niveau

### 3.3.1 La génération classique et la génération en HLS

La conception d'un circuit passe par plusieurs étapes pour obtenir un FPGA mettant en œuvre l'application souhaitée. Deux méthodes existent pour la génération de circuits FPGA : l'approche classique se basant sur les transferts entre registres (Register Transfer Level ou RTL) et l'approche de haut niveau se basant sur la synthèse de haut niveau (High Level Synthesis ou HLS).

1. Flot de conception RTL

   Le flot de conception classique par RTL est représenté dans la Figure 3.3 [Sjoholm and Lindh, 1997]. Le point d'entrée pour la conception classique d'un

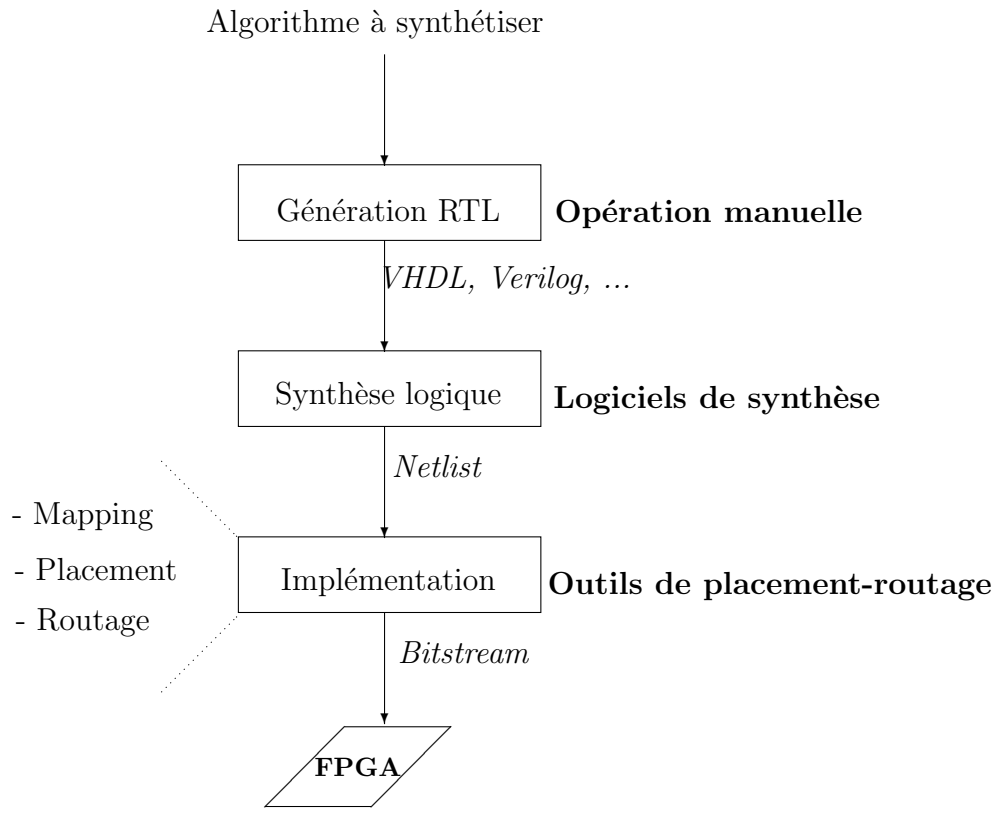



**Figure 3.3:** *Flot de conception RTL.*

circuit FPGA est la description RTL, où à partir d'un algorithme le développeur écrit manuellement une première version en langage de description matérielle, tel que VHDL, Verilog, ou SystemVerilog. A partir de cette description, plusieurs étapes sont nécessaires pour arriver à la génération du circuit souhaité, où l'ensemble des étapes est appelé synthèse. Tout d'abord, le code RTL passe par une phase de synthèse logique, où le RTL est traduit à un ensemble d'éléments logiques interconnectés appelé netlist. Cette netlist peut être composé d'additionneurs, de blocs mémoires, etc. Différents logiciels de synthèse existent comme XST de l'environnement ISE de Xilinx, Quartus II d'Altera, etc. Cette étape donne aussi l'utilisation en surface et la fréquence. Si les caractéristiques matérielles (surface et fréquence) ne correspondent pas aux contraintes et aux besoins, une modification de la description RTL est nécessaire pour chercher une meilleure solution. Après la phase de la synthèse logique, il y a la phase d'implémentation physique pour une cible FPGA donnée. Cette phase se compose du mapping, du placement et du routage. Le mapping se charge de traduire la netlist en des composants dans le FPGA,

généralement des LUTs et des FFs. Le placement et le routage consistent à placer les ressources dans le FPGA et à les relier entre elles par des interconnexions. Le résultat final obtenu est un bitstream, qui est un flux de données binaires à charger dans le FPGA et permettant sa programmation.

2. Flot de conception HLS

   La conception à base de synthèse de haut niveau consiste à automatiser la conception du RTL [Coussy and Morawiec, 2008]. Le point d'entrée d'un flot de HLS (voir Figure 3.4) est une description dans un langage de haut niveau (C, C++). L'utilisateur peut donner des indications spécifiques (types des opérateurs, le partage des ressources, etc.) et en sortie, un ou plusieurs fichiers sont obtenus décrivant l'application dans un langage de description matérielle (VHDL, Verilog, etc). A la génération, les caractéristiques matérielles (surface, fréquence et latence) sont données. Ainsi l'utilisateur peut garder la solution ou essayer de l'améliorer, selon le besoin, en utilisant des commandes spécifiques du logiciel de HLS. Ces outils HLS permettent par conséquence une exploration rapide de l'espace des solutions.

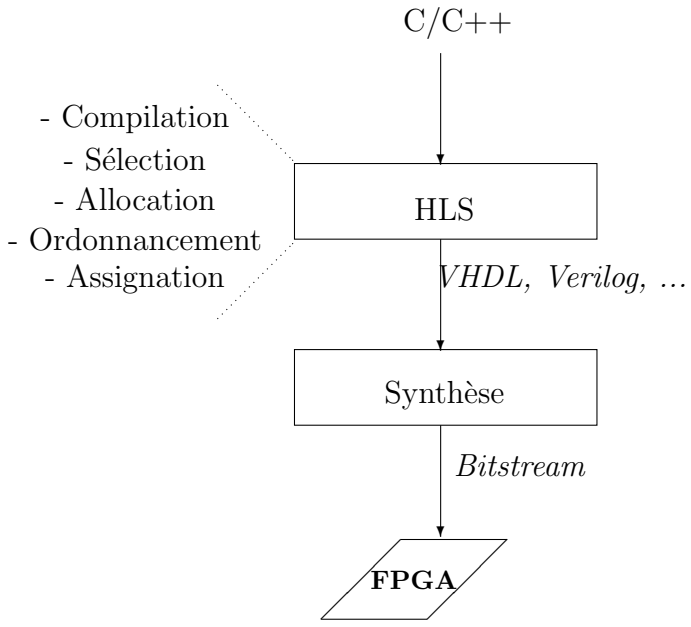



**Figure 3.4:** *Flot de conception HLS.*

Les étapes de la synthèse de haut niveau sont (voir Figure 3.4) :

(a) **La compilation** permet de vérifier la syntaxe et la sémantique et traduire la description en une représentation intermédiaire. Ainsi elle se charge des optimisations telles que la propagation de constantes et la transformation de boucles [Nicolau and Potasmann, 1991; Bacon et al., 1994].

(b) **La sélection** permet de définir les types d'opérateurs à partir de la bibliothèque de composants matériels. Le choix des opérateurs dépend des indications en surface, vitesse, consommation, etc. [Bakshi et al., 1996].

(c) **L'allocation** permet de fixer le nombre d'instances nécessaires dans l'architecture finale [Gutberlet et al., 1992] . Cela peut se faire par l'utilisateur dans le cas de contraintes de ressources ou automatiquement dans le cas de contraintes de temps.

(d) **L'ordonnancement** permet d'affecter l'instant de début d'exécution de chaque opérateur respectant les contraintes exigées et les dépendances de données fournies par la compilation [Walker and Chaudhuri, 1995].

(e) **L'assignation** permet d'associer une instance d'opérateur à chaque opération et un registre à chaque variable Cong and Xu [2008].

Plus de détails sur le flot de conception de haut niveau et ses spécificités sont donnés dans [Coussy et al., 2009].

De nombreux outils existent pour la synthèse haut niveau, nous pouvons citer quelques uns :

— **Handle-C** [Han], créé en 1996, permet de définir des sections parallèles et de construire des canaux de communication (FIFOs) entre les unités. Il supporte aussi les types de variables et signaux avec des précisions au niveau bit.

— **Stream-C** [Gokhale et al., 2000], basé sur le modèle de programmation Communicating Sequential Process (CSP) de Hoare [Hoare, 1983] et cible des applications opérant sur des flux de données.

— **SPARK** [Gupta et al., 2003] transforme du C en VHDL. Cette HLS a été conçu pour étudier les impacts des transformations et des heuristiques appliqués à la génération du code VHDL à partir de code C.

— **Catapult** [cat] est un des outils industriels les plus répandus. Il prend en entrée du C++ ou du SystemC et génère du code RTL pour FPGA ou ASIC. Ces dernières avancées sont l'intégration de méthodes de vérification du code généré.

— **Vivado Xilinx HLS** [Viv] fait partie de la suite d'outils Xilinx. Il se base sur un environnement de développement (EDI) Eclipse, intégrant la compilation et la simulation RTL. Il génère du SystemC, VHDL et Verilog. Il est possible d'exporter directement une application sous forme de bloc IP depuis le HLS, facilement intégrables dans un projet.

Dans nos travaux, nous nous appuyons sur l'outil Vivado HLS de Xilinx, pour l'implémentation d'un accélérateur FPGA sur SoC du calcul de l'inférence Bayésienne. Dans la section suivante, nous donnons le flot de conception pour FPGA à base de Vivado HLS ainsi que les différentes spécifications du logiciel.

3. Flot de conception SoC-FPGA

   Les processeurs et les FPGA sont les noyaux les plus performants de la plupart des systèmes embarqués. Parmi les améliorations les plus récentes du monde FPGA, on retrouve les périphériques SoC- FPGA (System on Chip- FPGA). Un SoC FPGA intègre un noyau de processeur dur et une logique programmable sur le même circuit. L'intégration de la fonctionnalité de gestion de haut niveau des processeurs et des opérations en temps réel, le traitement des données ou les fonctions d'interface d'un FPGA dans un seul dispositif forment un système embarqué encore plus puissant (voir l'architecture générale dans la Figure 3.5). Les trois plus grands fournisseurs FPGA, Xilinx, Altera et Microsemi (Actel) commencent tous à fabriquer de tels circuits.

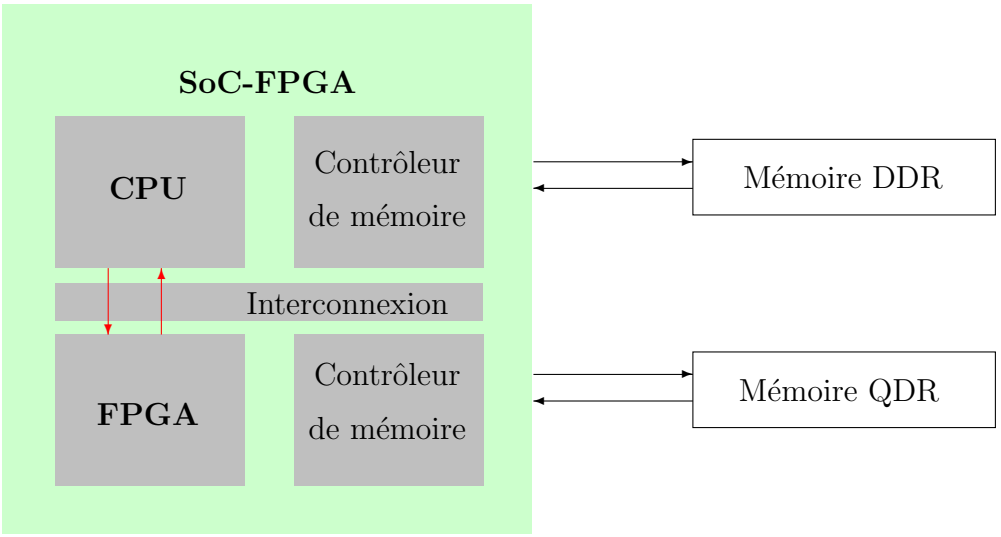



**Figure 3.5:** *Architecture SoC-FPGA*

Le flot de conception pour un SoC se base sur ce qu'on appelle le Co-Design logiciel/matériel. Le Co-Design permet d'exploiter le partage matériel et logiciel dans le but de respecter les contraintes de conception telles que le coût, les performances et la consommation énergétique [Teich, 2012].

La Figure 3.6 représente le flot de conception pour un SoC. Dans l'étape de partitionnement, le choix entre les parties à réaliser en matériel et les parties en logiciel est défini. Par la suite, la synthèse logicielle consiste en la génération du code exécutable sur le processeur et la synthèse matérielle à générer le bitstream pour l'exécution sur le FPGA. La synthèse d'interface est une étape essentielle, où les transferts de données entre la partie matérielle et la partie

logicielle sont gérés, ainsi que les protocoles et les modes de communication.

Les outils de synthèse de haut niveau permettent de simplifier ce flot, en utilisant des directives, que nous détaillons par la suite avec les outils Xilinx. Xilinx SDSoC est un nouvel environnement qui permet aussi une estimation des ressources et des performances avec des propositions d'architecture, ainsi que pour le partitionnement.

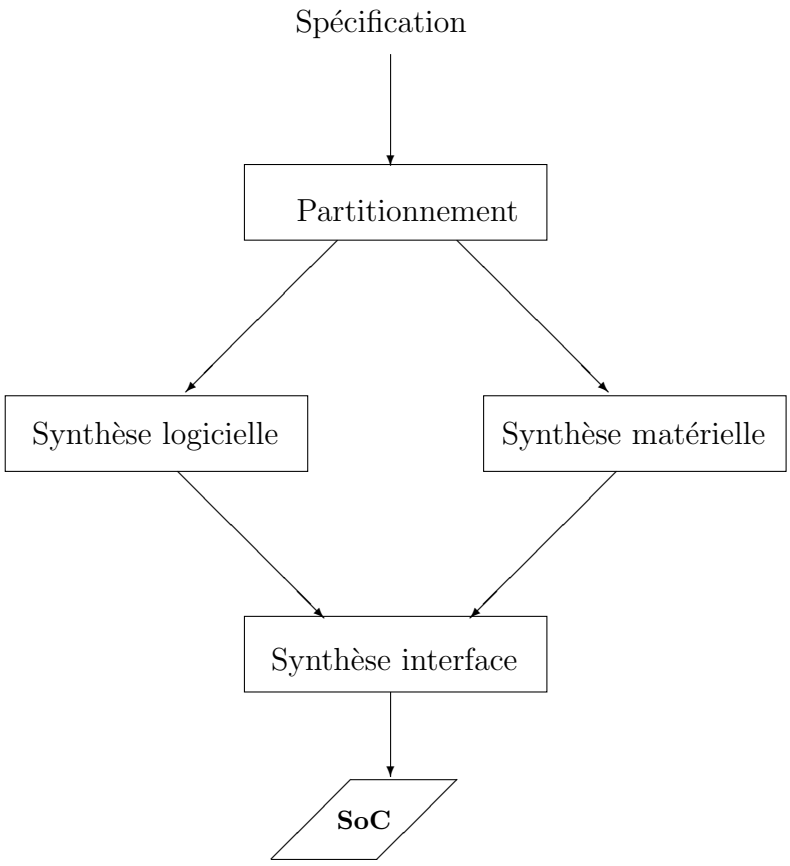



**Figure 3.6:** *Flot de conception SoC.*

### 3.3.2 Flot de conception avec Vivado HLS de Xilinx et gestion des optimisations et interfaces

Vivado HLS est un logiciel de Xilinx pour la synthèse de haut niveau [Xil, a]. Il fait la synthèse d'une fonction C ( C, C++ ou SystemC) qui peut être intégrée dans un système matériel. Il offre un ensemble complet de fonctionnalités pour la mise en œuvre la plus optimale du programme C et il est facilement intégrable dans les outils de conception Xilinx.

La Figure 3.7 donne le flot de conception Vivado HLS avec les fichiers d'entrée et de sortie. Vivado HLS permet de :

— compiler, simuler et déboguer l'algorithme C,

— synthétiser l'algorithme C et donner la description RTL, avec ou sans directives d'optimisation des utilisateurs. Ces directives d'optimisation permettent de donner des indications spécifiques par rapport au partage des ressources, l'accès aux mémoires, etc., Nous détaillons cette partie par la suite.

— avoir des rapports complets et les capacités d'analyse,

— vérification automatisée de l'implémentation du RTL,

— préparer l'implémentation RTL dans un des formats disponibles.

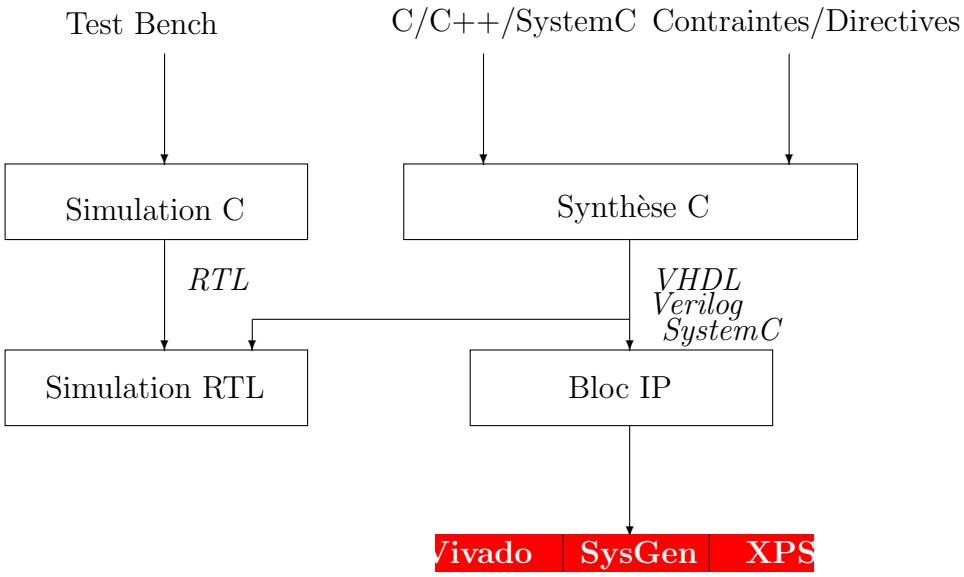



**Figure 3.7:** *Flot de conception Vivado HLS.*

L'allocation et l'assignation sont les processus au cœur de la synthèse de Vivado HLS. Nous expliquons cela sur l'exemple suivant :

```
int foo(char x, char a, char b, char c) {
char y;
y = x*a+b+c;
return y
}
```

Pendant le processus d'ordonnancement, le HLS détermine dans quelle opération du cycle d'horloge il faut se produire. Les décisions prises lors de la planification tiennent compte de la fréquence d'horloge, des informations de synchronisation de

la bibliothèque des périphériques cibles, et de toute directive d'optimisation spécifiée par l'utilisateur. Les directives d'optimisation sont détaillées dans la partie suivante.

Pour l'exemple, la phase d'ordonnancement est représentée dans la Figure 3.8. La multiplication et la première addition doivent être exécutées dans le premier cycle d'horloge, puis la deuxième addition. Ainsi, la sortie est disponible à la fin du deuxième cycle d'horloge.

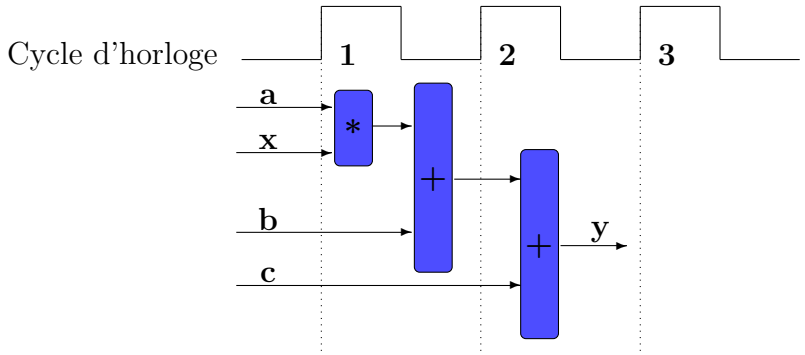



**Figure 3.8:** *Phase d'ordonnancement.*

La Figure 3.9 représente la phase d'assignation, où les ressources matérielles sont déterminées pour les opérations définies à l'ordonnancement. Il est possible d'utiliser un DSP48 pour implémenter à la fois une multiplication et une addition. Une ressource DSP48 est un bloc de calcul disponible dans l'architecture FPGA qui offre l'équilibre idéal de haute performance.

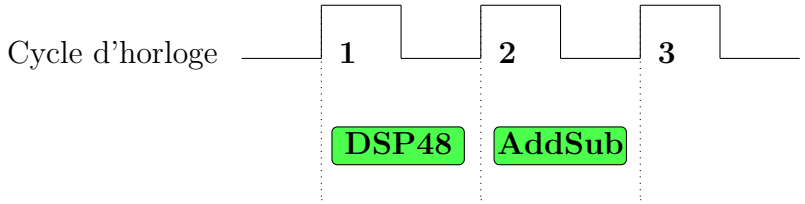



**Figure 3.9:** *Phase d'assignation.*

Pour les entrées/sorties (I/O) de cet exemple, ce sont des ports de données simples. Etant donné que chaque variable d'entrée est un type char, les ports de données d'entrée sont tous sur 8 bits. Le résultat de la fonction est un type int de 32 bits et le port de données de sortie est de 32 bits.

Nous pouvons voir les avantages de l'implémentation du code C dans le matériel. Toutes les opérations se terminent en seulement deux cycles d'horloge. Dans un CPU, même cet exemple de simple calcul prend beaucoup plus de cycles d'horloge.

### Gestion des optimisations et des interfaces dans Vivado HLS

A partir d'un code C, plusieurs implémentations matérielles sont possibles, avec différentes performances et ressources utilisées. Pour la gestion de l'implémentation selon le besoin, des directives sont proposées pour la gestion des interfaces, les ressources utilisées, les optimisations, etc.

### Gestion des interfaces

Les communications entre le processeur et le FPGA se font via des bus AXI (Advanced eXtensible Interface) [Xil, c].

Il existe trois types d'interface AXI :

— AXI4 : pour les besoins de cartes mémoire à haute performance,

— AXI4-Lite : pour une communication simple, à faible débit, mappée en mémoire,

— AXI4-Stream : pour les données de diffusion à grande vitesse.

La communication se fait à travers 3 types de port :

— Port GP : interface esclave 2 x 32 bits ou interface maître 2 x 32 bits. Les ports GP sont conçus pour une flexibilité maximale, permettant l'accès au registre de la partie logicielle vers la partie matérielle et inversement,

— Port HP : interface esclave 4 x 64 bits. Les ports HP sont conçus pour un accès maximal de bande passante à la mémoire externe,

— Port ACP : interface esclave A x 64 bits. Les ports ACP permettent à un accélérateur matériel d'accéder à faible latence au cache des processeurs.

Vivado HLS prend en charge l'ajout d'interfaces AXI suivants :

— AXI4-Stream (axis)

— AXI4-Lite esclave (s_axilite)

— AXI4 master (m_axi)

L'exemple suivant montre les directives à rajouter pour l'utilisation d'un AXI4-Stream de deux tableaux A et B sur Vivado HLS.

```
void  ExempleAXI(int A[50], int B[50]) {
#pragma HLS INTERFACE axis port=A
#pragma HLS INTERFACE axis port=B
int i;
for(i = 0; i < 50; i++){
B[i] = A[i] + 5;}
}
```

### Gestion des optimisations

Une liste des directives principales d'optimisation fournies par Vivado HLS est donnée dans le Tableau 3.1 [Xil, b].

| Directive | Description |
|---|---|
| ALLOCATION | limiter le nombre d'opérations, de noyaux ou de fonctions. Cela force le partage des ressources mais ça augmente la latence. |
| ARRAY_MAP | Regrouper des petits tableaux dans un seul grand tableau. Cela aide à réduire l'utilisation des BRAM. |
| ARRAY_PARTITION | Diviser un grand tableau en plusieurs petits tableaux. Cela améliore l'accès aux données. |
| ARRAY_RESHAPE | Reformer un tableau de plus grande largeur. Cela améliore l'accès aux BRAM sans utiliser plusieurs. |
| DATA_PACK | Former un seul scalaire avec un mot de plus grande largeur. |
| DATAFLOW | Permet le pipeline des tâches, pour s'exécuter simultanément. Cela permet de minimiser l'intervalle. |
| DEPENDENCE | Surmonter les dépendances et permettre aux boucles d'être pipelinées. |
| EXPRESSION_BALANCE | Désactiver l'équilibrage automatique des expressions. |
| FUNCTION_INSTANTIATE | Optimiser localement différentes instances de la même fonction. |
| INLINE | Supprimant toute la hiérarchie des fonctions. Cela permet l'optimisation logique et améliore la latence. |
| LATENCY | Spécifier une contrainte de latence minimale et maximale. |
| LOOP_FLATTEN | Affaisser les boucles imbriquées en une seule boucle. |
| LOOP_MERGE | Fusionner les boucles consécutives. Cela pour réduire la latence globale et améliorer l'optimisation logique. |
| PIPELINE | Réduire l'intervalle d'initiation en autorisant l'exécution simultanée des opérations dans une boucle ou une fonction. |
| STREAM | Spécifier qu'un tableau doit être implémenté comme FIFO ou RAM lors de l'optimisation dataflow. |
| UNROLL | Créer pour les boucles plusieurs opérations indépendantes plutôt qu'une seule collection d'opérations. |

**Tableau 3.1:** *Directives d'optimisation Vivado HLS*

En plus des directives d'optimisation, Vivado HLS fournit des paramètres de configuration pour modifier le comportement par défaut de la synthèse (voir Le Tableau 3.2 [Xil, b]).

| Directive | Description |
|---|---|
| Config Array Partition | Limiter le nombre d'opérations, de noyaux ou de fonctions. Déterminer comment les tableaux sont partitionnés. |
| Config Bind | Déterminer le niveau d'effort à utiliser pendant la phase d'assignation. |
| Config Compile | Diviser un grand tableau en plusieurs petits tableaux. |
| Config Dataflow | Spécifier le canal par défaut et la profondeur FIFO dans l'optimisation dataflow. |
| Config Interface | Contrôler les ports d'I/O non associés aux arguments de la fonction. |
| Config RTL | Fournir un contrôle sur la sortie RTL. |
| Config Schedule | Déterminer le niveau d'effort à utiliser pendant la phase d'ordonnancement. |

**Tableau 3.2:** *Configurations Vivado HLS*

Les optimisations utilisées dans nos travaux sont détaillées dans les chapitres suivants.

## 3.4 Conclusion

Dans ce chapitre, nous avons exposé un état de l'art sur les accélérateurs matériels. Nous avons commencé par définir et citer les différents accélérateurs matériels existants, en se focalisant sur les FPGA, qui font l'objectif de nos travaux. Nous avons aussi donné un aperçu des accélérateurs matériels pour les réseaux Bayésiens. Nous avons par la suite présenté la conception des accélérateurs FPGA, et les outils de la synthèse de haut niveau et plus particulièrement Vivado HLS de Xilinx.

Ce chapitre montre l'intérêt de l'accélération matérielle qui peut être mis à profit grâce aux outils disponibles. En revanche, ces outils n'ont pas été exploités dans le cas des réseaux Bayésiens. Le chapitre suivant est consacré à notre première contribution, qui est l'atelier logiciel pour l'implémentation des réseaux Bayésiens. Cet atelier logiciel prend comme entrée un réseau Bayésien graphique ou textuel et donne en sortie le bitstream prêt pour l'implémentation logicielle ou matérielle sur FPGA, en passant par les outils de synthèse de haut niveau.

# - Chapitre 4 -

# Atelier logiciel pour l'implémentation des réseaux Bayésiens

## Sommaire

## 4.1 Introduction

Nous présentons dans ce chapitre nos contributions apportées à l'implémentation de l'inférence Bayésienne et de la décision sur un SoC hybride. Pour cela nous proposons tout un atelier logiciel, afin de générer des implémentations HW/SW, car il n'existe pas d'outil efficace et dédié pour embarquer directement l'inférence Bayésienne sur un SoC hybride. Aussi, ces implémentations peuvent contribuer à l'adaptation de l'implémentation en fonction des contraintes de performances et de la disponibilité des ressources.

La Figure 4.1 donne un aperçu sur notre démarche, qui se fait en deux phases :

1. Hors-ligne : dans cette phase, nous commençons par la compilation d'un réseau Bayésien en AC avec une méthode hiérarchique et modulaire. Puis, nous appliquons quelques optimisations possibles par rapport à la connaissance des nœuds évidences. Par la suite, nous générons la description C de haut niveau de l'AC en se basant sur une décomposition hiérarchique de l'arbre, et la description C du diagramme d'influence de la décision. Et finalement, nous utilisons les outils de synthèse de haut niveau (Vivado HLS de Xilinx) pour des directives de synthèse selon le besoin en parallélisme, partitionnement, et contraintes de ressource ou de latence. Aussi afin de définir les interfaces et les ressource utilisées pour le stockage et l'envoi des paramètres. Et finalement pour produire la description HDL standard dans le format VHDL ou le format SystemC qui sera mappé sur la partie FPGA du SoC pour l'implémentation HW. Cette dernière description est ensuite déployée sur l'architecture du SoC, comprenant un processeur pour l'implémentation SW, l'IP généré, les interfaces de communication entre le processeur et l'IP pour l'envoi des paramètres et un IP "timer" pour évaluer les performances HW / SW, utilisant Vivado Design Suite (Xilinx).
2. En-ligne : cette phase est exécutée en temps réel. Les valeurs de certaines informations de capteurs et de commandes sont observées, ce qui permet de fixer les indicateurs d'évidence puis calculer l'inférence et la décision. Une comparaison entre la version SW et HW est établie en terme de performance. Les versions HW et SW sont réalisés avec le même outil. Les deux peuvent être embarquées et la plus adéquate sera sélectionnée par un système adaptatif en fonction des besoins de performance ou de la disponibilité en ressource.

Dans ce chapitre, nous nous intéressons plus particulièrement à la phase hors-ligne pour la génération du bitstream pour l'implémentation HW sur un support programmable, qui demande une expertise plus spécifique. La phase en-ligne de l'implémentation HW/SW sur SoC sera plus détaillée dans le Chapitre 6, dans le cadre de l'application sur une carte hybride Zynq. Dans la Section 4.2, nous détaillons la partie hors-ligne de l'atelier logiciel pour la génération du bitstream pour un réseau Bayésien. Nous exposons la description des réseaux Bayésiens, la compilation en AC et les optimisations et nous détaillons la décomposition modulaire et hiérarchique du calcul d'inférence par AC. Nous présentons aussi la génération du code C et exposons les directives de synthèse utilisées pour gérer les données, les contraintes de ressources, et de performances, en utilisant l'outil de synthèse de haut niveau Vivado HLS. Dans la Section 4.3, nous détaillons la partie hors-ligne de

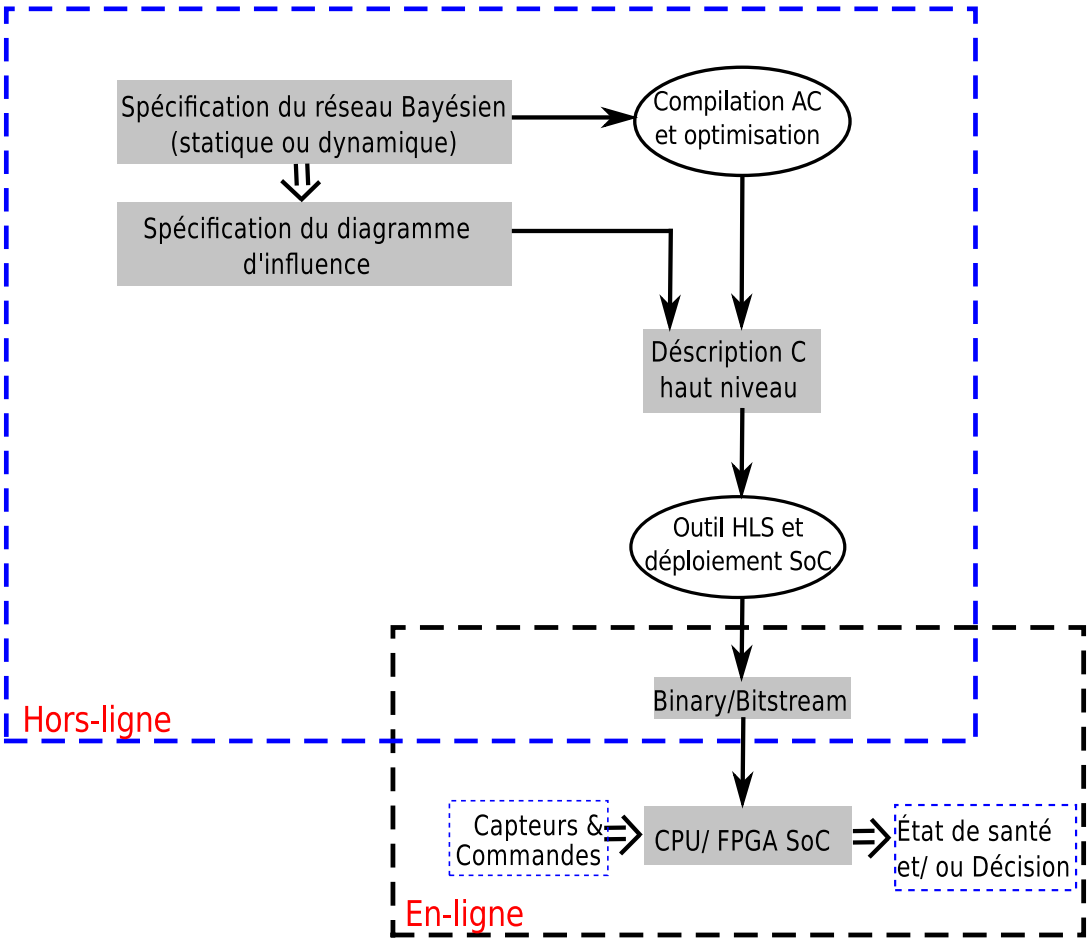



**Figure 4.1:** *Atelier logiciel pour l'implémentation HW/SW des réseaux Bayésiens.*

l'atelier logiciel pour le calcul de la décision. Nous exposons la description des diagrammes d'influence de la décision, et la génération du code C pour la synthèse de haut niveau. Et finalement, nous concluons ce chapitre.

## 4.2 Génération hors-ligne du bitstream pour un réseau Bayésien

Dans notre outil, les réseaux Bayésiens peuvent être décrits en format .m (Matlab utilisant la ToolBox BNT [Murphy, 2001]) ou en format .net (Hugin [Lauritzen, 2014], SamIam [Sam], etc.). Le but est de générer le bitstream qui sera par la suite implémenté sur une carte FPGA. Les étapes de cette génération sont résumées dans la Figure 4.2.

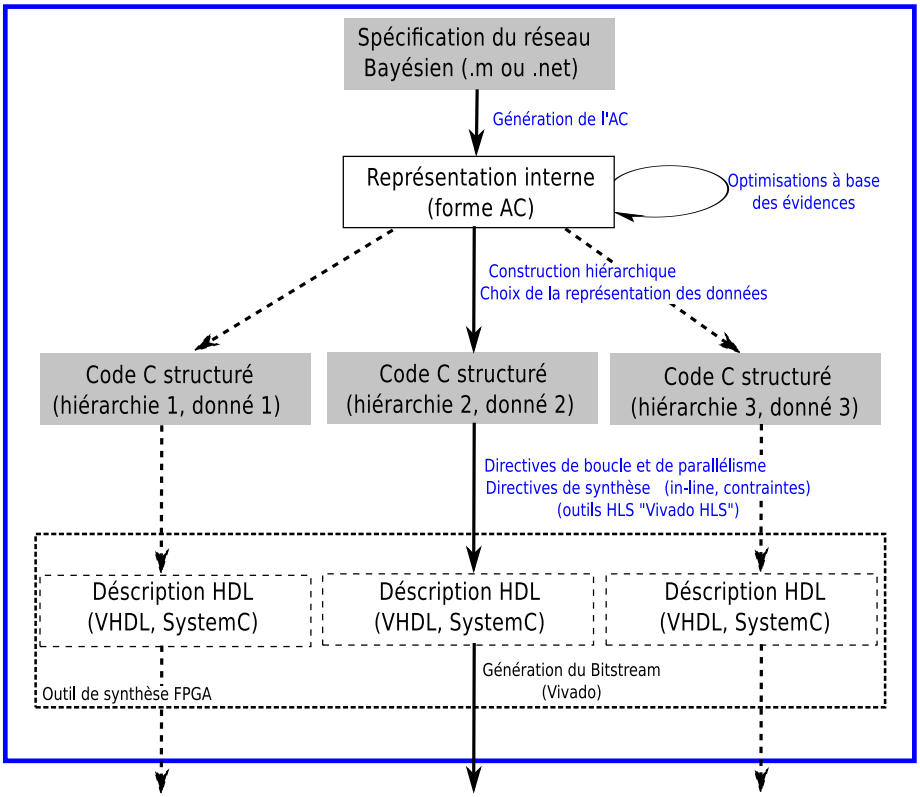



**Figure 4.2:** *Détails de la phase hors-ligne de l'atelier logiciel proposé.*

### 4.2.1 Spécification du réseau Bayésien

La spécification d'un réseau Bayésien peut se faire d'une manière graphique ou textuelle à partir des divers outils existants. Chaque description comporte :

- des nœuds : représentant les variables aléatoires,
- des arcs : représentant les dépendances entre les nœuds,
- des tables de probabilités : représentant les valeurs à priori et relations conditionnelles entre les nœuds.

1. Spécification générale en .m :
   Pour le .m, la description se fait d'une manière textuelle comme suit :
   - la définition des nœuds et des arcs : se fait à travers une matrice $N * N$ initialement nulle, où $N$ est le nombre de nœuds du réseau Bayésien. Les arcs sont représentés par un *1* dans la matrice entre les deux variables concernées (cf. Algorithme 1 Partie 1),
   - la définition du nombre d'états pour chaque nœud : se fait à travers un vecteur de taille $N$ (cf. Algorithme 1 Partie 2),
   - la définition des tables de probabilités : se fait à travers de vecteurs $V\_A_i$, après la génération de la structure du réseau Bayésien (bnet) (cf. Algorithme 1 Partie 3).

**Algorithme 1 :** Partie 1 (définition des nœuds et des arcs en .m)

dag =zeros ($N$,$N$) {définir une matrice nulle de taille N}
**pour tout** les nœuds du réseau Bayésien $A_i$ **faire**
  $A_i = i$ {respectant un ordre topologique, les parents avant les fils}
**fin pour**
**pour tout** arc entre deux variables $A_i$, $A_j$ **faire**
  dag ($A_i$, $A_j$)=1 {définir les arcs}
**fin pour**

**Algorithme 1 :** Partie 2 (définition du nombre d'états des nœuds)

**pour tout** les nœuds du réseau Bayésien **faire**
  nodes_sizes (i)=nb_etat {définir le vecteur des états nodes_sizes, où nb_etat est le nombre d'états}
**fin pour**

**Algorithme 1 :** Partie 3 (définition des tables de probabilités)

bnet= mk_bnet (dag, nodes_size) {génération de la structure du réseau Bayésien}
**pour tout** les nœuds du réseau Bayésien **faire**
  bnet.CPD{$A_i$}= tabular_CPD(bnet,$A_i$, $V\_A_i$) {définir les tables de probabilités}
**fin pour**

2. Spécification générale en .net :
   Pour le .net, les parties importantes de la description textuelle du réseau généré graphiquement sont les suivantes :
   — la définition des nœuds et de leurs états : (cf. Algorithme 2 Partie 1).
   — la définition des tables de probabilités : (cf. Algorithme 2 Partie 2).
3. Exemple d'une spécification en .m et en .net :
   Ici, nous donnons un exemple que nous utilisons pour toutes les illustrations de ce chapitre (cf. Figure 4.3). Le réseau Bayésien se compose de :

   — Nœuds : 4 nœuds A, B, C et D avec deux états chacun,
   — Arcs : A vers B et C, et D vers C,
   — Tables de probabilités : représentées par les $\theta$.

**Algorithme 2 :** Partie 1 (définition des nœuds et de leurs états en .net)

**pour tout** les nœuds du réseau Bayésien $A_i$ **faire**
  node $A_i$
  {
    states= ( "$a_{i_0}$"  "$a_{i_1}$"  ...) {définir les états}
  }
**fin pour**

**Algorithme 2 :** Partie 2 (définition des tables de probabilités)

**pour tout** les nœuds du réseau Bayésien **faire**
  potential ($A_i$ | les fils de $A_i$)
  {
    data = la table de probabilités
  }
**fin pour**

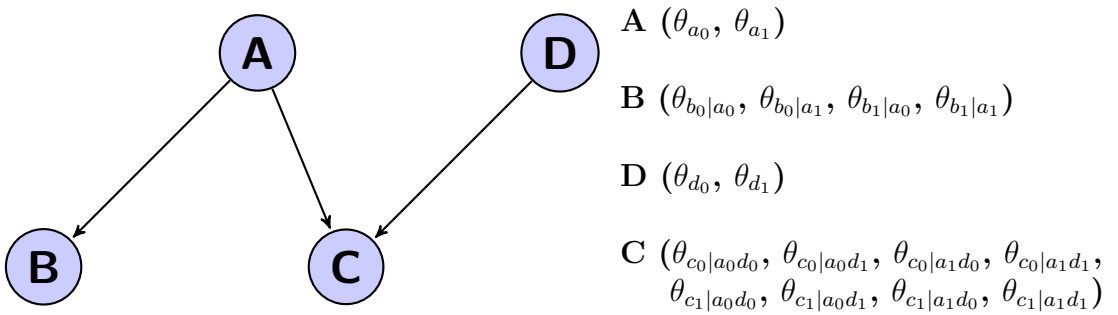



**Figure 4.3:** *Un exemple de réseau Bayésien.*

La description en .m de l'exemple est la suivante :

```
N = 4;
dag = zeros(N,N);
A = 1; B = 2; D = 3; C = 4;
dag(A, [B C ]) = 1;    dag(D,C) = 1;

node_sizes = [2 2 2 2];
bnet = mk_bnet(dag, node_sizes);

bnet.CPD{A}=tabular_CPD(bnet,A,[0.5 0.5]);
bnet.CPD{B}=tabular_CPD(bnet,B,[0.4 0.7 0.6 0.3]);
bnet.CPD{D}=tabular_CPD(bnet,D,[0.6 0.4]);
bnet.CPD{C}=tabular_CPD(bnet,C,[0.8 0.1 0.5 0.3  0.2 0.9 0.5 0.7]);
```

La description en .net de l'exemple, avec des valeurs numériques aléatoires des tables de probabilités, est représentée comme suit :

```
node A
{
    states = ("a0"  "a1");
}
node B
{
    states = ("b0"  "b1");
}
node D
{
    states = ("d0"  "d1");
}
node C
{
    states = ("c0"  "c1");
}

potential (A |)
{
    data = (0.5 0.5);
}
potential (B | A)
{
    data = ((0.4 0.6)
         (0.7 0.3));
}
potential (D |)
{
    data = (0.6 0.4);
}
potential (C | A D)
{
    data = (((0.8 0.2)
               (0.1 0.9))
             ((0.5 0.5)
              (0.3 0.7)));
}
```

Pour la suite de l'atelier logiciel, nous travaillons avec Matlab. Nous avons mis en place un parser afin de traduire le .net en .m.

### 4.2.2 Compilation en AC

A cette étape, le but est d'avoir une représentation interne pour le calcul d'inférence générée automatiquement à partir de la spécification .m du réseau Bayésien. Dans notre cas, nous nous basons sur le calcul d'inférence à partir de la compilation en Circuit Arithmétique (détaillée dans l'état de l'art). Notre approche consiste à créer l'AC d'une façon modulaire et hiérarchique pour faciliter la décomposition et la modification dans le cas d'ajout ou de suppression de nœuds.

La compilation en AC est donnée par l'algorithme suivant (Algorithme 3 ) :

**Algorithme 3 :** Compilation en AC

Ordonner les nœuds par ordre topologique et en ASAP
Créer un (+) pour le premier nœud, ayant $N_0$ fils (*) {$N_0$ est le nombre d'états du nœud}
A chaque (*) faire correspondre son indicateur d'évidence $\lambda$ et son paramètre $\theta$
**pour tout** les autres nœuds, par ordre **faire**
  **si** le nœud a au moins un parent **alors**
    Ajouter un (+) à tout les (*) du dernier père ayant $N_i$ fils (*)
    A chaque (*) faire correspondre son indicateur d'évidence $\lambda$ et son paramètre $\theta$
  **sinon**
    Ajouter un (+) à tout les (*) du dernier nœud de la liste avant le nœud en question, ayant $N_i$ fils (*) {$N_i$ est le nombre d'états du nœud}
    A chaque (*) faire correspondre son indicateur d'évidence $\lambda$ et son paramètre $\theta$
  **fin si**
**fin pour**

L'ordre topologique ici veut dire que les pères sont avant les fils, par niveau et en ASAP (avant les fils du voisin du meme niveau). Comme pour notre exemple l'ordre est $A$,$D$,$B$,$C$ et non $A$,$B$,$D$,$C$ car dans l'AC le $D$ va être automatiquement lié à $A$ car son fils ($C$) est lié à $A$, ainsi suivant l'algorithme, le $A$ est le dernier nœud de la liste pour le $D$.

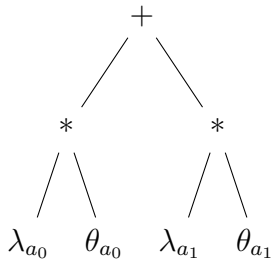
Dans ce qui suit, nous donnons les étapes de création de l'AC pour l'exemple précédent, où les indicateurs d'évidence et les paramètres pour chacun des nœuds sont représentés dans le Tableau 4.1.

| Variables | Indicateurs d'évidence | Paramètres |
|---|---|---|
| A | $(\lambda_{a_0},\lambda_{a_1})$ | $(\theta_{a_0}, \theta_{a_1})$ |
| B | $(\lambda_{b_0},\lambda_{b_1})$ | $(\theta_{b_0\|a_0}, \theta_{b_0\|a_1}, \theta_{b_1\|a_0}, \theta_{b_1\|a_1})$ |
| D | $(\lambda_{d_0},\lambda_{d_1})$ | $(\theta_{d_0}, \theta_{d_1})$ |
| C | $(\lambda_{c_0},\lambda_{c_1})$ | $(\theta_{c_0\|a_0d_0}, \theta_{c_0\|a_0d_1}, \theta_{c_0\|a_1d_0}, \theta_{c_0\|a_1d_1}, \theta_{c_1\|a_0d_0}, \theta_{c_1\|a_0d_1}, \theta_{c_1\|a_1d_0}, \theta_{c_1\|a_1d_1})$ |

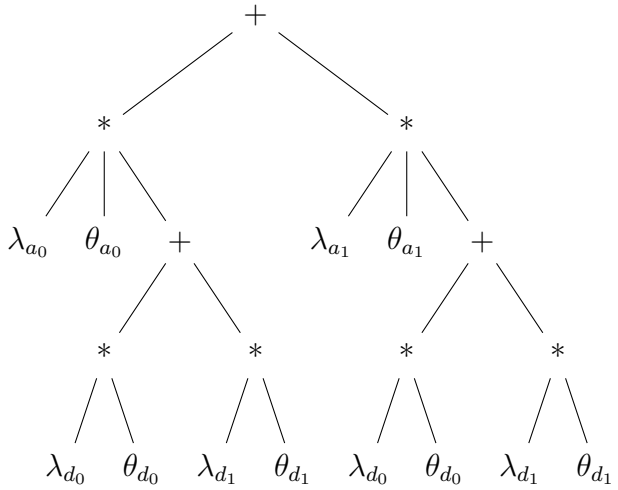
**Tableau 4.1:** *Les indicateurs d'évidence et les paramètres pour la compilation AC de l'exemple.*

**Etape de création de l'AC**

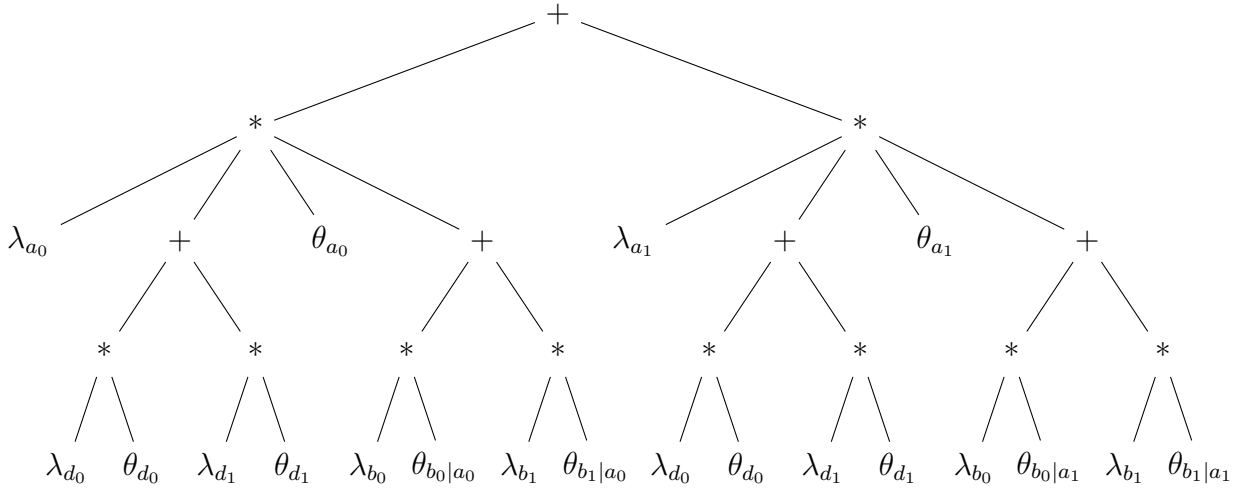
— Création de l'AC : étape 1 ( nœud A) :



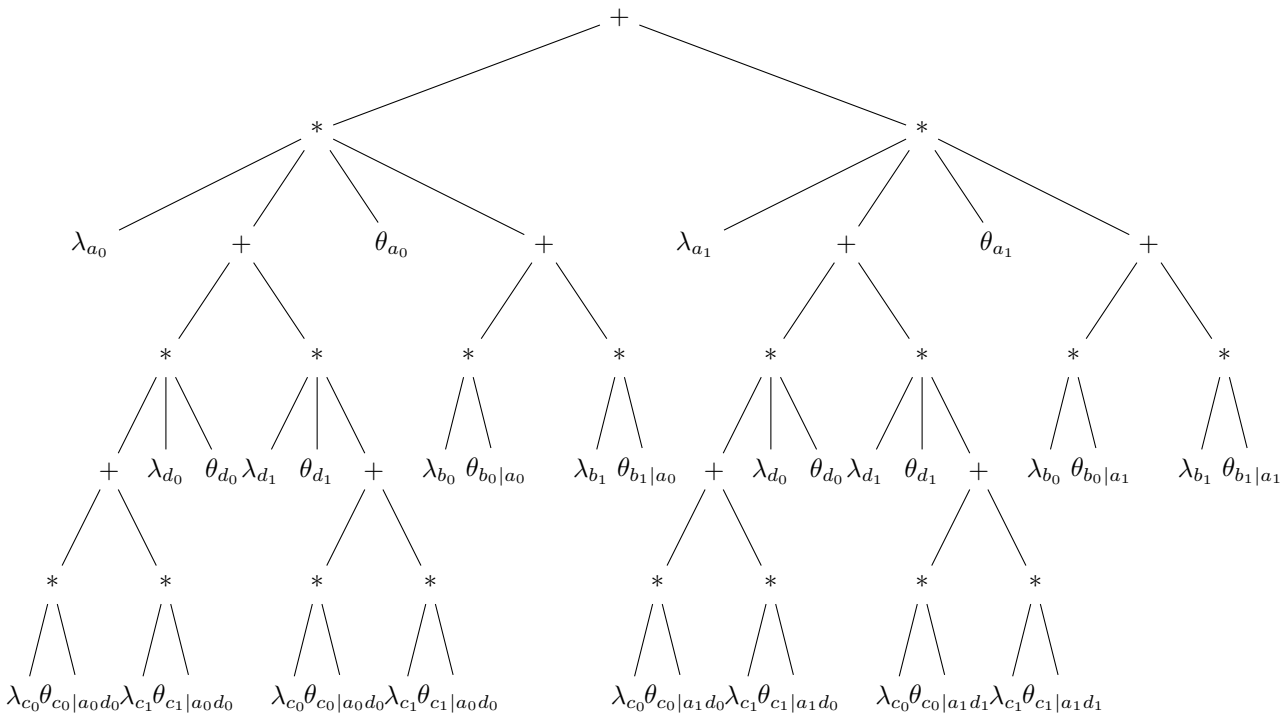

— Création de l'AC : étape 2 (nœud A, nœud D) :




— Création de l'AC : étape 3 (nœud A, nœud D, nœud B) :



— Création de l'AC : étape 4 (nœud A, nœud D, nœud B, nœud C ) :




### 4.2.3 Décomposition hiérarchique de l'AC

En observant l'arbre AC, on peut déduire la possibilité d'une décomposition hiérarchique et parallèle.
Dans l'arbre AC, on trouve :

— une alternance de nœuds addition (+) et de nœuds multiplication (*),

— chaque nœud multiplication (*) a un seul parent (un nœud (+)),

— les feuilles sont toujours des fils de nœud (*).

Ici, nous commençons par expliquer la décomposition de l'atelier proposé dans Schumann et al. [2013b] (abordée dans l'état de l'art). Puis, nous exposons notre décomposition. Et finalement, nous donnons un exemple comparant les deux approches.

1. Décomposition de Schumann :
   Un atelier logiciel a été proposé qui génère une architecture reconfigurable en fonction des propriétés structurelles de l'AC. Cette architecture repose sur une décomposition en blocs multi-mode qui peuvent atteindre différents types de calculs.

   Les différents modes correspondent à différentes parties extraites des opérations dans l'AC comme suit :

— mode a) : fonctionnement avec quatre entrées et 3 opérations élémentaires (soit addition ou multiplication),

— mode b) : fonctionnement avec trois entrées et 2 opérations élémentaires,

— mode c) : fonctionnement avec 2 entrées et 1 opération élémentaire.

La Figure 4.4 représente le bloc multi-mode et ces trois modes.

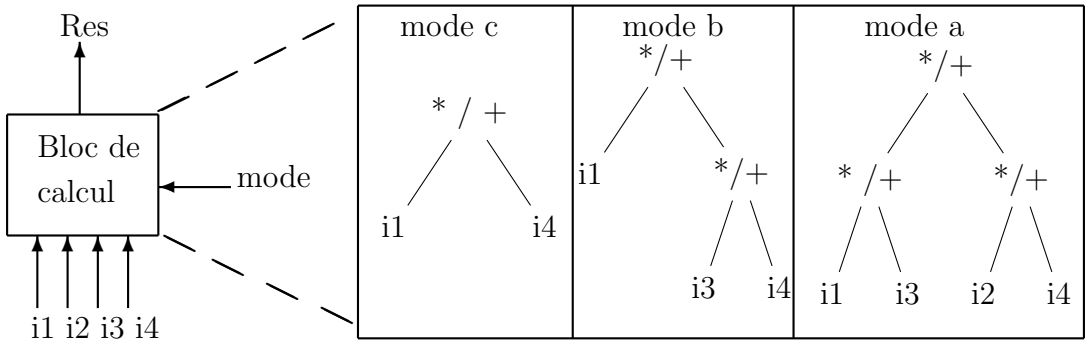



**Figure 4.4:** *Le bloc multi-mode de Schumann.*

2. Notre décomposition :
Contrairement à la décomposition donnée par Schumann utilisant des blocs génériques, nous avons opté pour une décomposition dédiée et incrémentale avec des optimisations dans le but de minimiser les ressources et la latence (à voir dans les résultats).
Avec une hypothèse de travail, où chaque nœud possède que 2 états (par exemple, pour une variable A les états sont $a_0$ et $a_1$ ), cependant on peut généraliser la démarche pour $n$ états.
Chaque niveau de l'AC peut être décomposé en blocs BE1, BE0 comme suit :

$$\begin{cases} BE1 = \lambda_{a_0} * \theta_{a_0|x} * BE0_{a_0} + \lambda_{a_1} * \theta_{a_1|x} * BE0_{a_1} \\ \\ BE0_{a_j} = \prod_{k=1}^{i} v(k) & \text{si } i > 0 \\ \quad = 1 & \text{si } i = 0, \text{ où } i \text{ est le nombre de sous-branches } v \text{ associées au nœud } A \text{ de l'état } a_j \ (j \text{ égal à 0 ou à 1}) \end{cases}$$

Nous savons aussi que les $\lambda$ (exemple pour une variable $A$ : $\lambda_{a_0}$ et $\lambda_{a_1}$) ne peuvent avoir que les valeurs :

$$\begin{cases} (\lambda_{a_0}, \lambda_{a_1}) = (1,1) & \text{si le noeud A n'est pas observable} \\ (\lambda_{a_0}, \lambda_{a_1}) = (1,0) & \text{si le noeud A est observable sur } a_0 \\ (\lambda_{a_0}, \lambda_{a_1}) = (0,1) & \text{si le noeud A est observable sur } a_1 \end{cases}$$

Prenant en compte ces informations, BE1 peut être défini comme suit :

$$\text{BE1} = \begin{cases} \theta_{a_0|x} * BE0_{a_0} & \text{si } \lambda_{a_1} = 0 \\ \theta_{a_1|x} * BE0_{a_1} & \text{si } \lambda_{a_0} = 0 \\ \theta_{a_0|x} * BE0_{a_0} + \theta_{a_1|x} * BE0_{a_1} & \text{sinon} \end{cases}$$

Pour les feuilles, où le nombre de fils est égal à 0, BE1 est optimisé comme suit :

$$\text{BE1} = \begin{cases} \theta_{a_0|x} & \text{si } \lambda_{a_1} = 0 \\ \theta_{a_1|x} & \text{si } \lambda_{a_0} = 0 \\ 1 & \text{sinon} \end{cases}$$

La Figure 4.5 représente les blocs utilisés dans notre décomposition, et la Figure 4.6 représente l'optimisation de BE1 pour les feuilles.

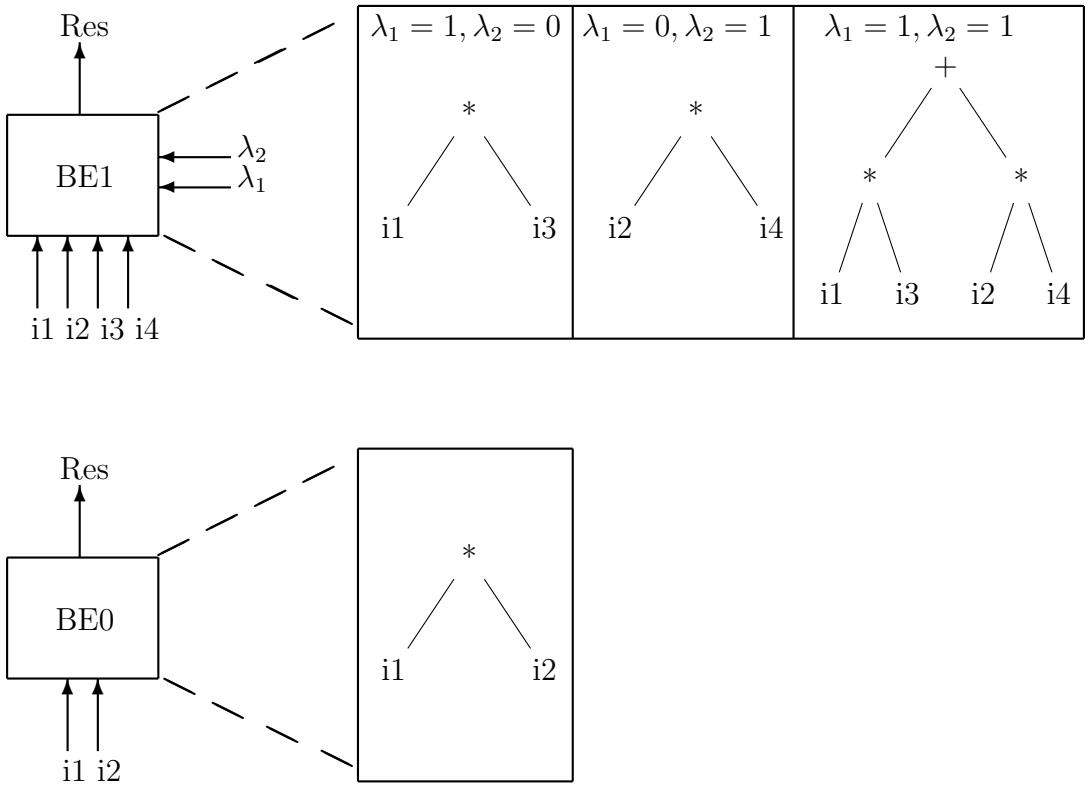



**Figure 4.5:** *Les blocs BE1, BE0 de notre décomposition.*

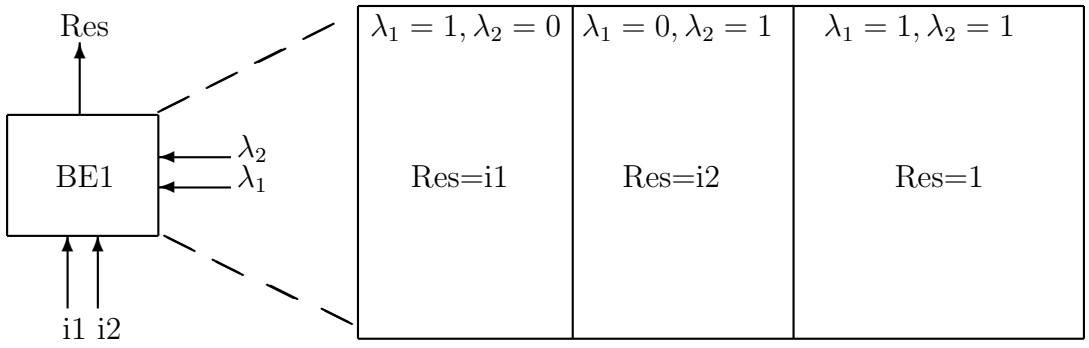



**Figure 4.6:** *Le bloc BE1 pour les feuilles.*

Dans notre cas, nous utilisons généralement des nœuds à 2 états mais cette approche peut être généralisée pour le cas de $n$ états où :

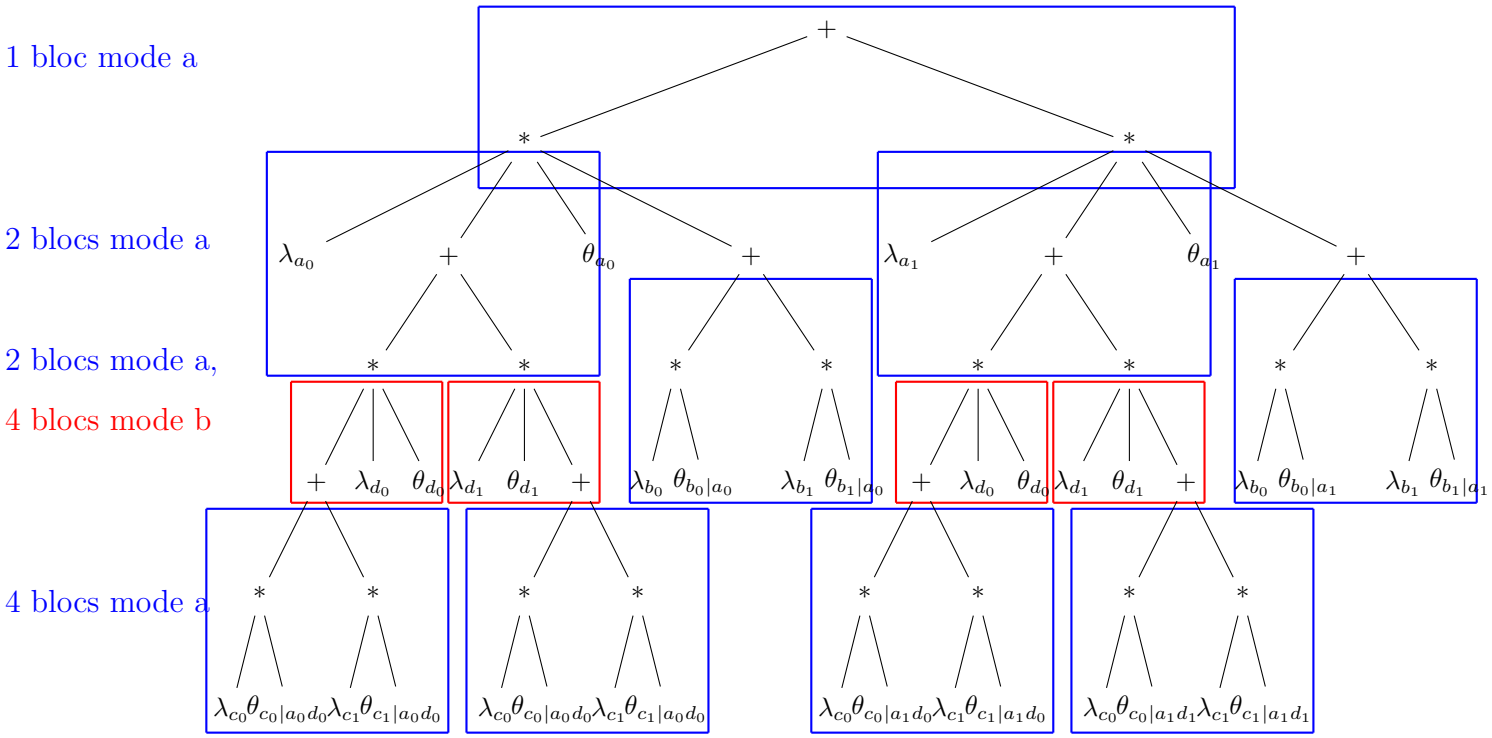
$$\begin{cases} BE1 = \sum_{i=1}^{n} \lambda_{a_i} * \theta_{a_i|x} * BE0_{a_i} & n \text{ est le nombre d'états du nœud A} \\ BE0_{a_i} = \prod_{k=1}^{j} v(k) & \text{si } j > 0 \\ \quad = 1 & \text{si } j = 0, \text{ où } j \text{ est le nombre de sous-branches } v \text{ associées au nœud } A \text{ de l'état } a_i \end{cases}$$

Pour une variable A à $n$ états ($\lambda_{a_1}$, $\lambda_{a_2}$, .... $\lambda_{a_n}$ ), un seul $\lambda$ peut être égal à 1 si présence d'une observation ou tous les $\lambda$ sont égaux à 1 si absence d'observation, ce qui donne un bloc BE1 défini comme suit :

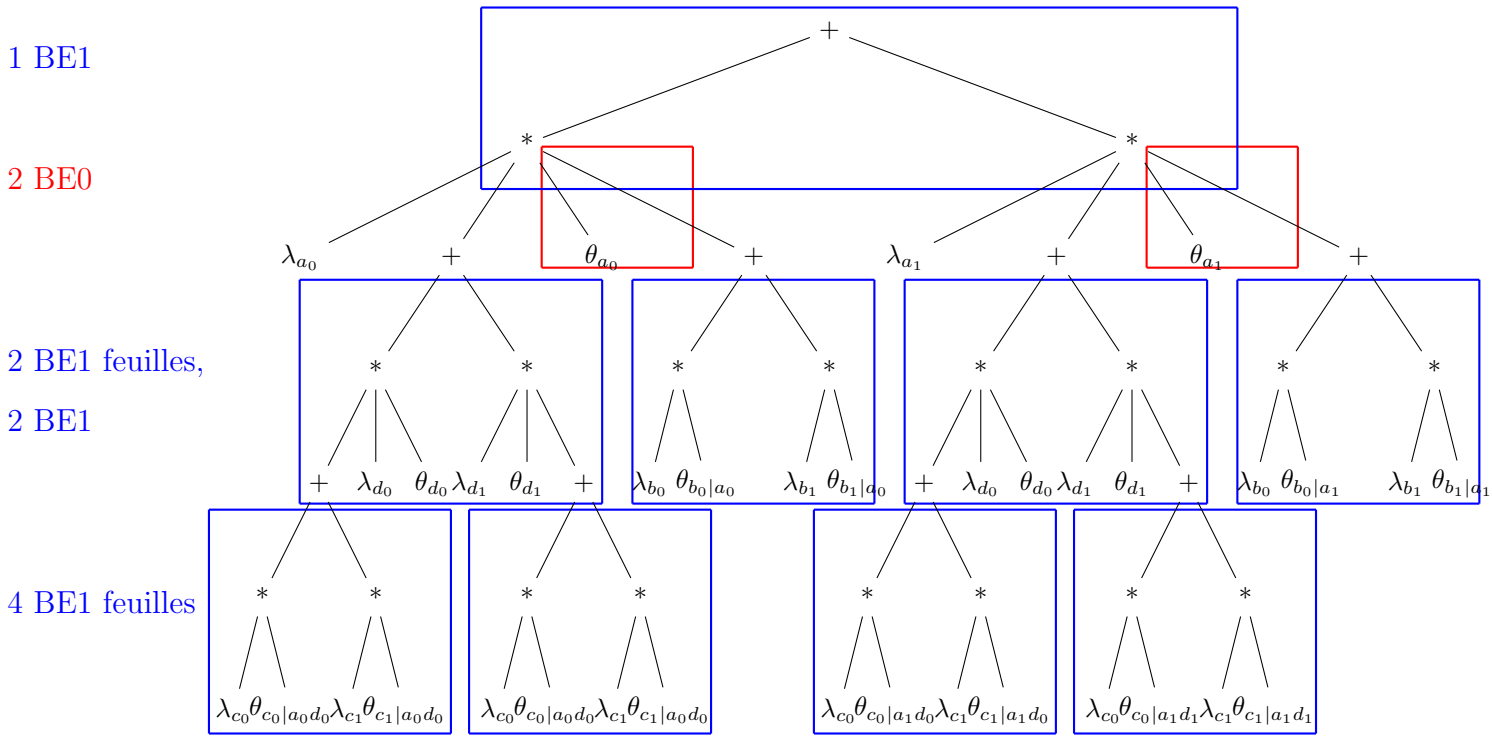
$$\text{BE1} = \begin{cases} \theta_{a_i|x} * BE0_{a_i} & \text{si } \exists!\ \lambda_{a_i} = 1 \\ \sum_{i=1}^{n} \theta_{a_i|x} * BE0_{a_i} & \text{sinon} \end{cases}$$

3. Exemple comparatif
   En appliquant cette décomposition de Schumann et notre décomposition sur l'exemple précédent, nous aurons :

— Décomposition de Schumann pour l'exemple :




— Notre décomposition :



Pour cet exemple, avec les blocs de Schumann, pour un maximum parallèle, 4 blocs mode a sont nécessaires avec 2 (*) et 1 (+) et 4 blocs mode b avec 2 (*) [total 12 (*), 4 (+)]. Pour notre cas nous avons besoin de 4 BE1 optimisés pour les feuilles qui n'utilisent ni de (*) ni de (+) , 2 BE1 où chaque BE1 est composé de 2 (*) et 1 (+) , et de 2 BE0 composé chacun de 1 (*) [total 6 (*) et 4 (+)]. Des comparaisons sur d'autres cas d'étude seront exposées dans le Chapitre 6 (application sur une architecture hybride).

### 4.2.4 Optimisations

Dans certains cas, selon les contextes d'utilisation, les nœuds observables et les nœuds non observables sont connus en avance. Dans ce cas nous aurons :

— pour un nœud observable $A(a_0, a_1) : (\lambda_{a_0}, \lambda_{a_1}) = (1, 1)$,

— pour les nœuds non observables $B(b_0, b_1) : (\lambda_{b_0}, \lambda_{b_1}) = (1, 0)$ ou $(0, 1)$.

Dans ce cas, nous décomposons BE1 en deux sous blocs, un pour les nœuds observables et l'autre pour les non observables, comme suit :

$$\text{BE1\_observable} = \begin{cases} \theta_{a_0|x} * BE0_{a_0} & \text{si } \lambda_{a_1} = 0 \\ \theta_{a_1|x} * BE0_{a_1} & \text{si } \lambda_{a_0} = 0 \end{cases}$$

$$\text{BE1\_Non\_observable} = \theta_{b_0|x} * BE0_{b_0} + \theta_{b_1|x} * BE0_{b_1}$$

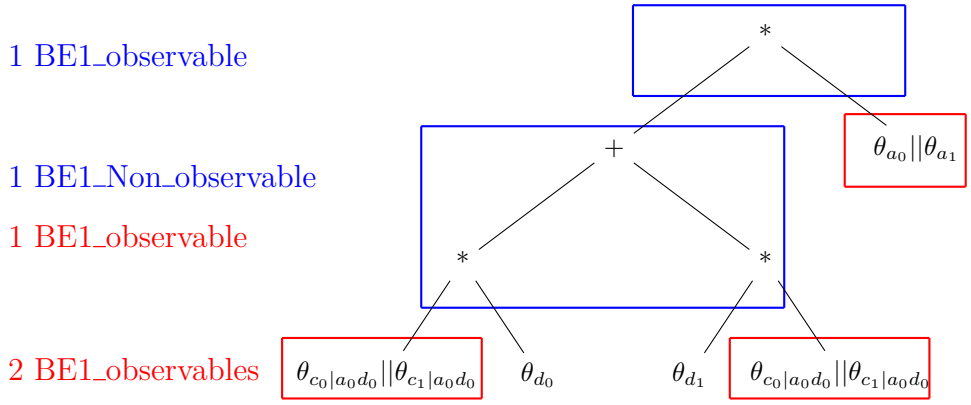
Et pour les feuilles :

$$\text{BE1\_observable} = \begin{cases} \theta_{a_0|x} & \text{si } \lambda_{a_1} = 0 \\ \theta_{a_1|x} & \text{si } \lambda_{a_0} = 0 \end{cases} \quad \text{(on le note } BE1 = \theta_{a_0|x}||\theta_{a_1|x})$$

$$\text{BE1\_Non\_observable} = 1 \quad \text{(cela signifie de supprimer les branches (+) feuilles)}$$

Cette transformation se rapproche d'une propagation de constante qui permet de simplifier les calculs dans les différents blocs.

En appliquant cette nouvelle décomposition sur l'exemple précédent, prenons par exemple les nœuds A et C comme nœuds observables et B et D comme nœuds non observables. Nous passons de 4 BE1 feuilles, 2 BE1 et 2 BE0 [total 6 (*) et 4 (+)] à 2 BE1_observables (pas de (*) ni de (+)) et 1 BE1_Non_observable (2 (*) et 1 (+)) et 1 BE1_observable (1(*)) [total 3 (*) et 1 (+)], comme suit :




### 4.2.5 Génération du code C pour la synthèse de haut niveau

La génération du code C se base sur la décomposition hiérarchique. La génération se fait automatiquement à partir d'un réseau Bayésien en passant par les étapes de compilation et de décomposition.

Une fois le code C généré, nous modifions le code pour qu'il soit plus efficace, en se basant sur les points suivants :

1. La représentation des données :
   les deux types de données en entrée sont :
   - Les paramètres $\theta$ : qui sont des probabilités inférieures ou égales à 1. Ils peuvent être représentés par des flottants pour plus de précision ou en point fixe pour minimiser les ressources et l'utilisation mémoire. En se fixant une largeur par rapport à un seuil, nous pouvons avoir un compromis entre la précision et les ressources.
   - Les paramètres $\lambda$ : qui sont des valeurs booléennes. Pour plus de performances et pour un stockage optimal, nous proposons de concaténer les $\lambda$ par 32 ou 64.
2. L'organisation de la mémoire de données :
   La taille des tables de probabilités et des indicateurs d'évidence est importante et augmente avec le nombre de nœuds du réseau Bayésien. Pour cela nous proposons d'utiliser des directives de synthèse permettant l'organisation de la mémoire en stockage interne et utilisant des BRAM, ou en stockage externe en utilisant des AXI Stream.
   - Dans le cas de l'AXI Stream, les paramètres sont envoyés du processeur via le stream, lu par l'accélérateur et stocké en mémoire interne en registre ou BRAM. Une fois le traitement fini, les résultats seront envoyés au processeur via l'interface Stream.

Un exemple de la fonction à générer et à rajouter pour utiliser l'interface Stream :

```
#include <hls_stream.h>
#include <ap_axi_sdata.h>
typedef ap_axis<32,2,5,6> intSdCH;  //  les para-
mètres du stream

float NomF(hls::stream<intSdCH>&inStream,
hls::stream<intSdCH>&outStream)

#pragma HLS INTERFACE axis port=inStream   // direc-
tive axi stream
#pragma HLS INTERFACE axis port=outStream

// L'envoi des données et la récupération via le Stream
for (int i=0; i<size;c++)
{
#pragma HLS PIPELINE
 #pragma HLS UNROLL
intSdCH InT1= inStream.read();       // lire  les données
union {    unsigned int ival; float oval; } convertert1;
convertert1.ival = InT1.data;          //conver-
tir du uint à float
 T1[c] = convertert1.oval;    // stockage des données

// Appel de la fonction  (resultat Res)

// L'envoi du résultat   via le Stream
union {    unsigned int oval; float ival; } converter1;
     converter1.ival=Res;
     valOut1.data=converter1.oval;
// pour indiquer la fin
     valOut1.last = 0;
     valOut1.strb = -1;
     valOut1.keep = 15;
     valOut1.user = 0;
     valOut1.id = 0;
     valOut1.dest = 0;
     outStream.write(valOut1);   // écriture du résultat
```

— Dans le cas de la BRAM, les paramètres sont stockés dans une BRAM, connectée à l'accélérateur via un port GP et au processeur via un contrôleur BRAM pour l'accès aux adresses des paramètres. Pour cela, il suffit juste de rajouter les directives suivantes dans la fonction globale :

```
#pragma HLS INTERFACE bram port=T1 //  variable T1 en  BRAM
#pragma HLS RESOURCE variable=T1 core=RAM_1P_BRAM
```

Le détail de l'interfaçage et de la connexion avec le processeur, ainsi que des avantages et inconvénients de chaque utilisation seront données dans le Chapitre 6 (application sur une architecture hybride)..

3. Gestion des ressources et des performances :
   Selon la disponibilité des ressources et des exigences en performances, nous pouvons rajouter des directives de synthèse afin d'utiliser plus ou moins de parallélisme, de limiter les ressources ou mettre à plat le code.

   — Mettre à plat le code peut permettre le partage de ressources. Pour cela il faut ajouter la directive suivante dans la fonction souhaitant la mettre à plat :

   ```
   #pragma HLS INLINE  on     // mettre à plat
   #pragma HLS INLINE  off   //  garder la fonction en bloc
   ```

   Dans notre cas, nous utilisons cette directive pour les blocs BE0 et BE1 pour les laisser en blocs ou les mettre à plat. Une étude comparative sera donnée dans le Chapitre 6 (application sur une architecture hybride).

   — Afin de limiter les ressources, nous utilisons la directive suivante pour les blocs BE1 et BE0 pour limiter le nombre d'instances des blocs à utiliser.

   ```
   #pragma HLS ALLOCATION instances=BE1 limit=4 function
   ```

   — Afin de pouvoir utiliser les variables des tables de probabilités en parallèle, nous utilisons la directive suivante, permettant de découper les tableaux en plusieurs registres indépendants. Selon le besoin en parallélisme et le nombre de blocs disponibles, nous pouvons faire un partitionnement "complet" où tous les paramètres sont stockés dans des registres indépendants, ou en "bloc " pour un partitionnement partiel par rapport à un facteur.

   ```
   #pragma HLS ARRAY_PARTITION variable=T1 block fac-
   tor=4 dim=1
    // découpage en blocs de  facteur 4
   #pragma HLS ARRAY_PARTITION variable=T1 complete
   // découpage complet
   ```

## 4.3 Génération hors-ligne du bitstream pour un réseau de décision

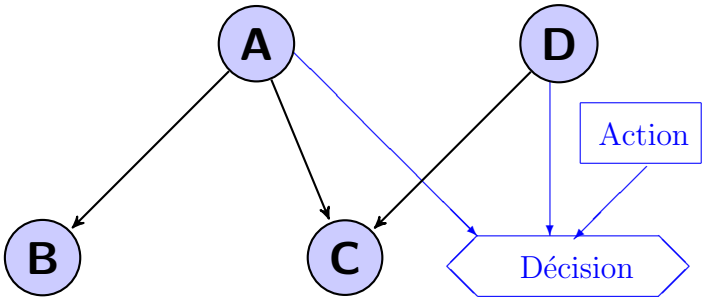
Les diagrammes d'influence pour la décision peuvent aussi être décrits en format .m ou en format .net. Le but est de générer le bitstream qui sera par la suite implémenté sur une carte FPGA.

La génération se fait en deux étapes :

1. Décrire les nœuds chances, les nœuds actions et définir la table d'utilité.
2. Générer le code C pour la synthèse de haut niveau.

### 4.3.1 Spécification du diagramme d'influence pour les réseaux de décision

Les nœuds chance du diagramme d'influence sont des nœuds du réseau Bayésien définis précédemment, et pour définir un réseau de décision, on rajoute des nœuds actions et une table d'utilité. Prenant l'exemple précédent, la spécification est la même que celle définie précédemment pour la partie réseau Bayésien en rajoutant la partie décision comme suit : (cf. Figure 4.7).




| Action |
|---|
| $A_0$ |
| $A_1$ |

| A | $a_0$ | $a_0$ | $a_0$ | $a_0$ | $a_1$ | $a_1$ | $a_1$ | $a_1$ |
|---|---|---|---|---|---|---|---|---|
| D | $d_0$ | $d_0$ | $d_1$ | $d_1$ | $d_0$ | $d_0$ | $d_1$ | $d_1$ |
| Action | $A_0$ | $A_1$ | $A_0$ | $A_1$ | $A_0$ | $A_1$ | $A_0$ | $1_1$ |
| Utilité | U0 | U1 | U2 | U3 | U4 | U5 | U6 | U7 |

**Figure 4.7:** *Un exemple de réseau de décision*

Le .net est défini comme suit, où nous avons deux actions $A_0$ et $A_1$, et la table d'utilité est donnée avec des valeurs aléatoires dans la variable "data".

```
.....

decision Action
{
    states = ("A0"  "A1" );
}

potential (Decision |  Action D A)
{
    data = (((2 10)
             (4 20))
            ((30 50)
             (7 100)));

// data  = (((U0 U1) (U2 U3)) ((U4 U5)(U6 U7))) de la Figure 4.7
}
```

### 4.3.2 Génération du code C pour la synthèse de haut niveau

La génération du code C se base sur le calcul de la fonction d'utilité ($U_{f\_A_k}$) pour chaque action $A_k$, et la décision est le choix de l'action ayant à maximiser cette fonction (définie dans l'état de l'art). Le code généré est le suivant (cf Algorithme 4), où les nœuds du réseau Bayésien liés au nœud décision sont représentés par $X_i$ et la table d'utilité est représentée par $U$.

**Algorithme 4 :** le code C généré à partir d'un réseau de décision

**pour tout** les états $x\_1_i$ du premier nœud du réseau Bayésien $X_1$ relié au nœud Décision **faire**
  ...
  **pour tout** les états $x\_n_i$ du $n$ ième nœud du réseau Bayésien $X_n$ relié au nœud Décision **faire**
    **pour tout** les actions $A_k$ **faire**
      $U_{f\_A_k} += U(A_k, X_i = x\_1_i, ..., X_n = x\_n_i) * P(X_1 = x\_1_i) * ...P(X_n = x\_n_i)$ {calculer la fonction d'utilité pour chaque action}
    **fin pour**
  **fin pour**
**fin pour**
Décision= Maximum ($U_{f\_A_k}$)  { parmi tous les $U_{f\_A_k}$ de chaque $A_k$ }

De même que pour le réseau Bayésien, nous rajoutons des directives de synthèse en nous basant sur les points suivants :

1. La représentation des données :
   En plus des paramètres du réseau Bayésien, nous avons pour la partie décision comme entrées le nombre d'actions et la table d'utilité.
   Les valeurs de la table d'utilité peuvent être des entiers ou des flottants.

2. L'organisation de la mémoire de données :
   La taille de la table d'utilité croit de manière exponentielle en nombre de nœuds chance. Taille de la table d'utilité = $2^{(nb\_noeuds)} * nb\_actions$
   avec 2 états pour chaque nœud. D'une manière générale, pour $N$ nœuds avec les états $\{ nb_1, nb_2, ...., nb_n\}$ :
   Taille de la table d'utilité = $nb_1 * nb_2 * .... * nb_n * nb\_actions$
   Selon la taille de la table, nous pouvons choisir entre l'envoi en Stream ou le stockage en BRAM avec la même démarche que pour les réseaux Bayésiens.

3. Gestion des ressources et des performances :
   Comme nous pouvons le constater, le code de la décision contient des boucles imbriqués et selon le besoin en performance et la disponibilité des ressources,

nous utilisons les deux directives suivantes pour exploiter au mieux le parallélisme de boucles :

```
#pragma HLS PIPELINE
#pragma HLS UNROLL
```

Aussi, nous pouvons utiliser le partitionnement comme dans les réseaux Bayésiens pour stocker indépendamment les valeurs de la table d'utilité afin d'utiliser plus de parallélisme si besoin. Des exemples seront données dans le Chapitre 6 (application sur une architecture hybride).

## 4.4 Synthèse et conclusion

Dans ce chapitre, nous avons exposé notre première contribution pour l'implémentation de l'inférence Bayésienne et de la décision sur un SoC hybride. Nous avons proposé tout un atelier logiciel pour générer des implémentations HW et SW, en nous focalisant sur la génération hors ligne du bitstream pour l'implémentation HW. Nous avons montré la passage de la spécification du réseau Bayésien à la génération du bitstream en passant par une représentation interne et une décomposition hiérarchique, des optimisations et de la génération du code C de haut niveau pour la synthèse de haut niveau.

Cette première contribution a abouti à des publications dans des conférences internationales [Zermani et al., 2015b,a, 2017a], avec une mise en œuvre dans le contexte de l'état de santé et la décision pour les systèmes autonomes (détaillé dans le Chapitre 6).

Pour montrer l'intérêt de notre outil, nous avons utilisé des modèles Bayésiens existants. Dans le chapitre suivant nous exposons notre deuxième contribution en proposant une spécification d'un réseau Bayésien et d'un réseau de décision pour l'état de santé et de décision des systèmes autonomes et adaptatifs en se basant sur une analyse de défaillance. Nous donnons aussi des cas d'étude pour des scénarios de défaillance de véhicules autonomes. Ainsi se confirme l'adaptabilité de notre atelier logiciel pour ce type de réseaux.

# - Chapitre 5 -

# Approche Bayésienne pour l'état de santé et la décision lors d'une mission d'un véhicule autonome

## Sommaire

## 5.1 Introduction

Nous présentons dans ce chapitre nos contributions au monitoring et à la décision dans les systèmes autonomes et adaptatifs, plus particulièrement les missions de véhicules embarqués. Notre première contribution consiste en la génération d'un modèle de réseau Bayésien pour le monitoring de l'état de santé et les scénarios de défaillance d'applications/ composants. Ce travail se base sur une analyse de défaillance de type FMEA (Analyse des Modes de Défaillances et de leurs Effets). La seconde contribution est la génération, à partir du réseau Bayésien, de l'état de santé et des actions de recouvrement, d'un réseau de décision afin d'élaborer une mission auto-adaptative. Nous exposons aussi l'adaptation de notre atelier logiciel à ce cas particulier d'étude.

La Figure 5.1 donne un aperçu de notre démarche. Les réseaux Bayésiens pour l'évaluation de l'état de santé des sous-systèmes de mission sont générés hors ligne à partir de l'analyse FMEA. Au cours d'une mission, des observations provenant de capteurs sont considérés comme des entrées du réseau Bayésien et l'état de santé des sous-systèmes est calculé avec la phase en ligne de notre atelier logiciel en SW ou en HW. Cet ensemble d'informations est acheminé vers le module de prise de décision, qui définit le nouveau plan de mission parmi les actions de recouvrement disponibles, en cas de présence d'un scénario de défaillance. Si le nouveau plan utilise de nouveaux composants, les réseaux Bayésiens de l'évaluation de leur état de santé peuvent être générés à nouveau, en remplaçant partiellement ou complètement les parties inutilisables.

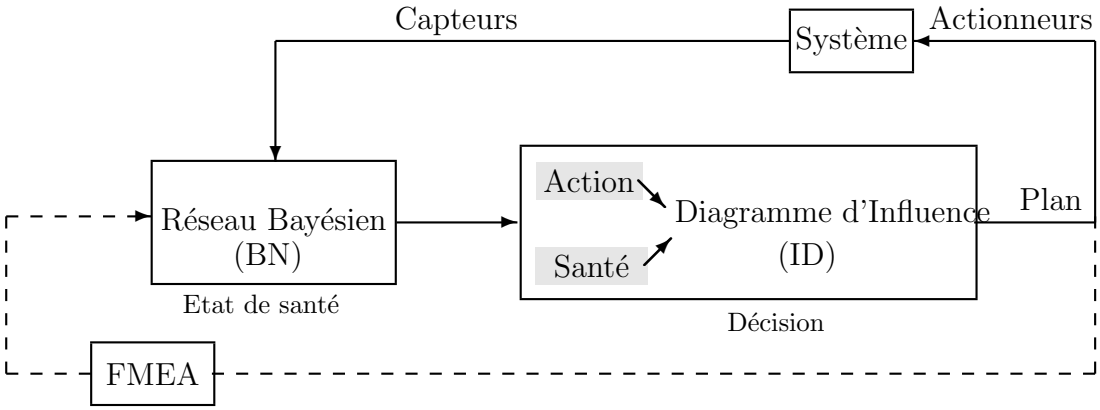



**Figure 5.1:** *Monitoring de l'état de santé et de la décision pour un système autonome.*

Dans la Section 5.2, nous exposons la génération du réseau Bayésien à partir de l'étude FMEA, en l'illustrant par deux exemples de scénarios de défaillance, et nous montrons l'adaptation de notre atelier logiciel à ces cas particuliers de réseaux. Ensuite, dans la Section 5.3 nous développons le modèle de décision par diagramme

d'influence pour la décision lors d'un scénario de défaillance, en l'illustrant par une mission de navigation et nous montrons l'adaptation de l'atelier logiciel. Et finalement, nous concluons sur cette partie et nous donnons un aperçu sur le chapitre suivant.

## 5.2 Génération d'un modèle Bayésien pour l'état de santé à base de l'analyse FMEA

Les réseaux Bayésiens ont l'avantage de prendre en compte l'incertitude et sont largement utilisés pour mettre en œuvre le diagnostic complexe. Néanmoins, il est difficile de construire un modèle de réseau Bayésien rationnel pour des applications réelles et complexes. Afin de résoudre ce problème, l'Analyse des Modes de Défaillances et de leurs Effets (FMEA) [McDermott et al., 1999] peut être utilisée.

Le but ici est de montrer comment, à partir de cette analyse de type FMEA, on peut construire le réseau Bayésien pour le monitoring de l'état de santé du composant/application.

### 5.2.1 L'analyse de type FMEA

L'analyse FMEA est une méthode pour identifier les modes de défaillance d'un produit ou d'un processus. Elle permet d'identifier pour l'ensemble des couples {composant, fonction}, les modes de dégradation de la fonction considérée, les causes de ces modes de dégradation et leurs conséquences sur le système étudié [Automotive Industry Action Group, 1993].

Dans notre cas, nous nous basons sur cette analyse pour générer les réseaux Bayésiens. Pour chaque application/composant, nous définissons un ensemble de types d'erreur qui représentent les sources de défaillance. Ces défaillances sont monitorées par des capteurs internes matériels ou logiciels et par des contextes d'apparition des défaillances renforçant la fiabilité (des capteurs, des applications, des prévisions, ou des historiques). Les capteurs eux même peuvent être défaillants, l'information pouvant aussi être renforcée par des contextes d'apparition.

Le Tableau 5.1 résume le résultat de cette analyse où :

- $U\_E_i$ représente un type d'erreur $i$ ($i \in \{1, .., p\}$),
- $A\_E_{i\_j}$ représente des contextes d'apparition $j$ ($j \in \{1, .., q\}$) d'un type d'erreur $i$,
- $S\_E_i$ représente le capteur moniteur matériel ou logiciel du type d'erreur $i$,
- $A\_H\_E_{i\_u}$ représente des contextes d'apparition du moniteur $u$ ($u \in \{1, .., r\}$) d'un type d'erreur $i$.

  ($p$ représente le nombre de types d'erreur, $q$ représente le nombre de contextes d'apparition, et $r$ représente le nombre de contextes d'apparition du moniteur.)

| Type d'Erreur | Moniteur | Contexte d'Apparition (Type d'Erreur) | Contexte d'Apparition (Moniteur) |
|---|---|---|---|
| $U\_E_i$ | $S\_E_i$ | $A\_E_{i\_j}$ | $A\_H\_E_{i\_u}$ |

**Tableau 5.1:** *Table FMEA associée à un type d'erreur*

A titre d'illustration, prenons un exemple de la détection d'obstacle. Le monitoring peut se faire par un capteur de mesure de distance Laser, par exemple. La fiabilité de l'information de la présence d'un obstacle donnée par ce capteur peut être renforcée par le type de terrain (contexte d'apparition). Nous considérons plus probable d'avoir un obstacle dans un milieu urbain que dans une foret, par exemple, ou plus dans la journée que dans la nuit. Aussi, la fiabilité du capteur Laser peut être renforcée par les prévisions météo. Nous possédons de meilleures informations du capteur Laser lorsque le ciel est dégagé, par exemple. Le Tableau 5.2 donne la table FMEA associée à l'erreur sur la détection d'obstacle.

| Type d'Erreur | Moniteur | Contexte d'Apparition (Type d'Erreur) | Contexte d'Apparition (Moniteur) |
|---|---|---|---|
| Détection d'obstacle | Capteur de mesure de distance (Laser) | - Terrain<br>- Obscurité | - Météo |

**Tableau 5.2:** *Table FMEA associée à l'erreur sur la détection d'obstacle.*

### 5.2.2 Génération d'un modèle de réseau Bayésien générique pour l'état de santé

La table FMEA peut être traduite sous forme de réseau Bayésien en respectant les relations cause-à-effet et le principe de la D-séparation [Pearl, 2000] pour la circulation de l'information.

Pour chaque type d'erreur, on construit un sous-réseau Bayésien, où on associe un nœud $U\_E_i$ à l'état interne inobservable de l'erreur. Cette erreur peut être observée par un moniteur, donc on lui associe un nœud $S\_E_i$ qui est fils du nœud $U\_E_i$. Cette information donnée par le moniteur peut être renforcée par contextes d'apparition qui causent l'erreur, donc on associe un nœud $A\_E_{i\_j}$ pour chacun de ces contextes d'apparition, qui sont des fils de $U\_E_{i\_u}$ pour respecter la D-séparation pour la circulation de l'information entre le moniteur et les contextes d'apparition.

Le moniteur est un capteur, pouvant être défaillant, et pour surveiller son état de santé on l'associe à un nœud $H\_E_i$ qui est parent du nœud $S\_E_i$. Aussi l'état de santé du capteur peut être influencé par des contextes d'apparition du moniteur, donc on associe à chacun un nœud $A\_H\_E_{i\_u}$. Reprenant l'exemple de l'erreur sur la détection d'obstacle, le sous-réseau Bayésien correspondant est donné dans la Figure 5.2.

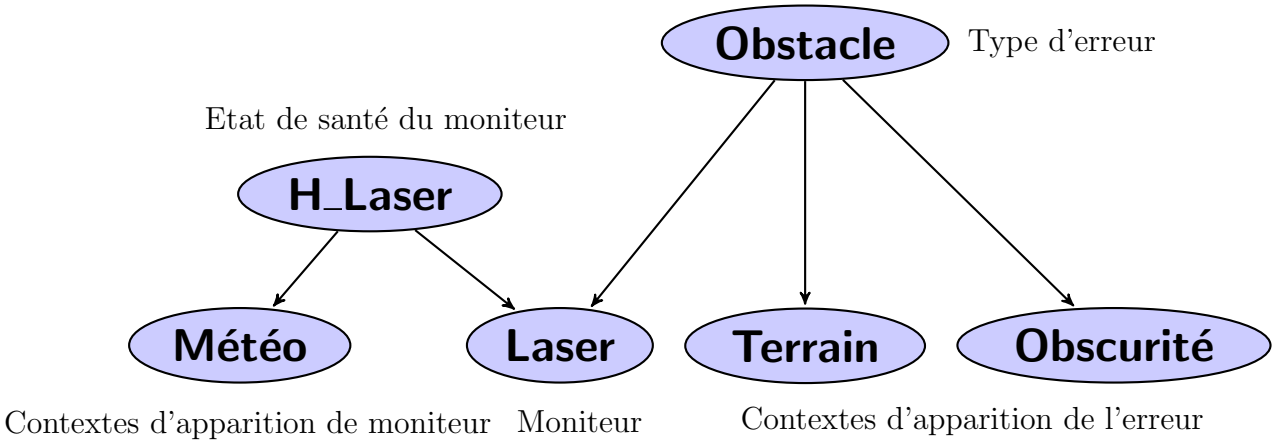
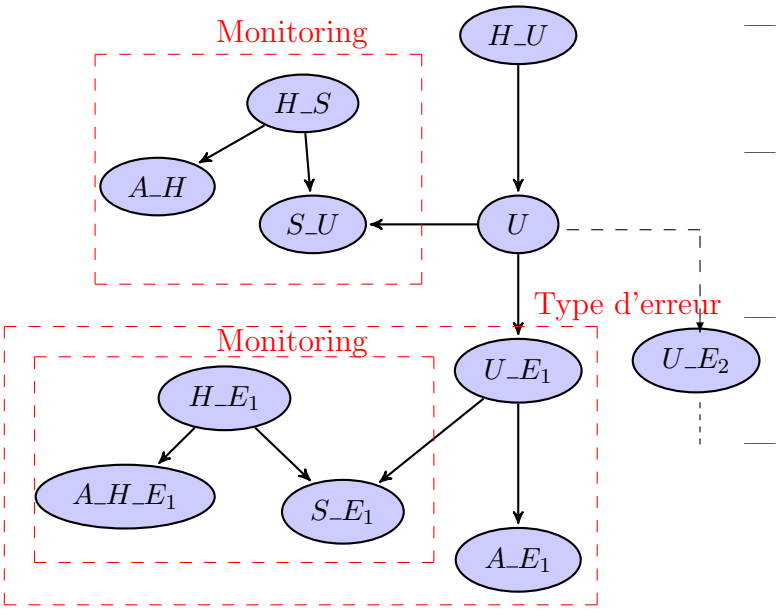



**Figure 5.2:** *Le sous-réseau Bayésien de l'erreur sur la détection d'obstacle.*

En se basant sur ce principe, nous proposons un modèle de réseau Bayésien pour l'état de santé, dans le cas statique et dynamique.

1. Réseau Bayésien global cas statique :
   Pour construire le réseau Bayésien global, on associe un nœud inobservable à l'application/tache/composant $U$. Le but étant de surveiller l'état de santé de ce système, on associe un nœud $H\_E$ à l'état de santé, parent du nœud $U$. L'état du système est observé par un moniteur du système, donc on lui associe un nœud $S\_U$, qui est fils du nœud $U$. De même que pour les types d'erreur, le nœud moniteur a comme parent le nœud $H\_S$, qui a comme fils les nœuds contexte d'apparition de l'état de santé du moniteur $A\_H$. L'état de santé du système dépend aussi des types d'erreur, et pour la circulation de l'information et la D-séparation, les nœuds $U\_E_i$ sont fils du nœud $U$. Le réseau Bayésien générique obtenu est représenté dans la Figure 5.3 avec les différents types de nœuds. Les entrées du modèle sont les valeurs des capteurs qui représentent les états observables (évidences). Les nœuds "contexte d'apparition" peuvent aussi être observables pour renforcer la croyance sur l'erreur ou sur l'état de santé des capteurs. La sortie du modèle est la probabilité a posteriori de l'état de santé du sous-système (nœud $H\_U$) par rapport aux évidences données.
2. Réseau Bayésien global cas dynamique :
   Certaines applications évoluent avec le temps, et dans ce cas nous proposons une généralisation de la construction en 2 temps ($t-1$ et $t$). Pour cela, on génère les deux sous-réseaux Bayésiens pour les deux temps, de la même manière que pour le cas statique avec les moniteurs et les contextes d'apparition. Les deux sous-réseaux sont reliés par les nœuds observables, qui sont fils du nœud $U$ de l'instant $t$, pour respecter les règles de la D-séparation et de la circulation d'information. Le réseau Bayésien générique obtenu est représenté dans la Figure 5.4.




— **Nœud** $H$ : reflète l'état de santé du sous-système ($H\_U$) ou des capteurs ($H\_S$ ou $H\_E_i$).

— **Nœud** $S$ : indique les mesures de traitement au moyen de capteurs HW/SW du sous-système ($S\_U$) ou des erreurs ($S\_E_i$).

— **Nœud** $U$ : reflète l'état 'inobservable' du sous-système ($U$) ou des erreurs ($U\_E_i$).

— **Nœud** $A$ : reflète le contexte d'apparition des erreurs ($A\_E_i$) ou de l'état de santé des capteurs ($A\_H$ ou $A\_H\_E_i$).

**Figure 5.3:** *Modèle générique à partir d'une analyse de type FMEA pour le cas statique.*

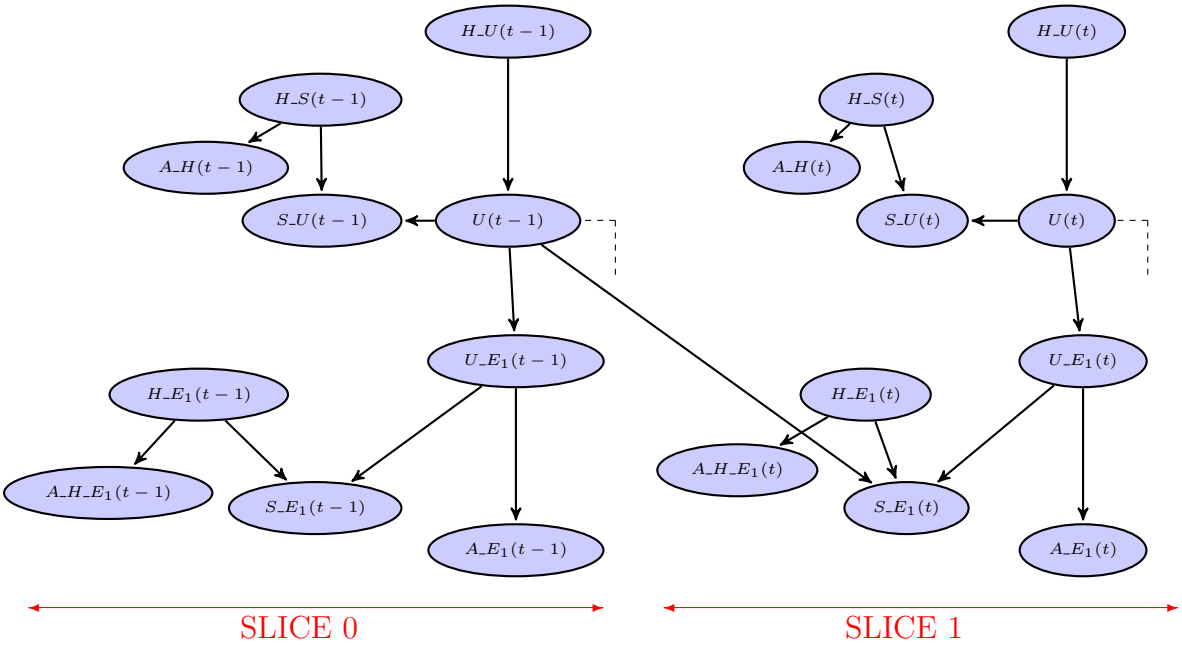



**Figure 5.4:** *Modèle générique à partir d'une analyse de type FMEA pour le cas dynamique.*

### 5.2.3 Exemples de scénarios de défaillance : GPS et énergie

Les scénarios de défaillance représentent des événements inattendus qui peuvent affecter la mission. Dans cette partie, nous exposons deux exemples de scénarios de défaillance pour le cas de monitoring de véhicules autonomes. Le premier est la localisation par GPS (statique) et le second est la consommation d'énergie (dynamique).

1. Le cas de la position par GPS :

   La navigation de véhicules autonomes nécessite des informations de géolocalisation pour assurer un positionnement précis. Ces informations peuvent être fournies par un Système de positionnement global [Grewal et al., 2007](GPS) ou par un SLAM [Durrant-Whyte and Bailey, 2006] (localisation et cartographie simultanées) ou autres méthodes.

   Le système GPS est souvent utilisé mais la fiabilité de la position donnée par le GPS dépend de facteurs contextuels qui affectent le signal satellitaire lors de sa propagation et sa réception. Afin de générer le réseau Bayésien, il faut d'abord définir les principales erreurs de la position par GPS et construire la table FMEA.

   — Principe du GPS et les principales erreurs :

   Le GPS est un système de navigation qui calcule la position d'un véhicule au moyen de signaux émis par des satellites aux récepteurs GPS. Dans un système de positionnement par satellite, les satellites (émetteurs) sont mobiles et chaque satellite a une horloge atomique qui garantit la précision du temps.

   Pour calculer une position, il suffit de connaître les positions de référence (positions satellites) et ses distances par rapport au récepteur GPS [Salós et al., 2010]. Pour calculer ces distances (mesures de pseudo-distance), le temps et la vitesse de propagation du signal doivent être connus. Le récepteur GPS détermine la distance entre l'antenne et l'antenne satellite. Par conséquent, la position est calculée en se basant sur l'observation de la mesure de pseudo-distance et de phase.

   La distance entre un satellite et le récepteur GPS (mesure de pseudo-distance) est calculée en multipliant le temps de propagation du signal par la vitesse. Dans le cas du GPS, les signaux passent à travers les différentes couches de l'atmosphère (ionosphère et troposphère) pour atteindre le récepteur GPS. Par conséquent, les signaux sont réfractés par les particules chargées d'atomes, des molécules, etc., qui ont un impact sur la direction et la vitesse de propagation du signal. En plus de ces deux sources d'erreur, il peut y avoir d'autres sources d'erreur.

La Figure 5.5 représente les erreurs du GPS, qui sont [Langley, 1997] :

- Horloge satellite : l'erreur de dérive dans les horloges atomiques des satellites GPS.
- Données éphémérides : une erreur dans la position du satellite transmise.
- Délai ionosphérique : une réfraction des particules sur les signaux présents dans l'ionosphère, qui a la particularité d'être ionisé, et qui dévie le trajet du signal. Le retard ionosphérique dépend de l'angle d'élévation du satellite, des tempêtes solaires ou de la proximité d'un équateur /pôles.
- Délai troposphérique : la présence d'atomes et de molécules neutres affecte la propagation des signaux satellites. La réfraction des signaux dépend de la température, de la pression du gaz dans la troposphère (azote, oxygène, argon et dioxyde de carbone) et de la vapeur d'eau.
- Multipath : un signal peut atteindre le récepteur GPS via plusieurs chemins. Ces différentes voies sont prises en raison de la présence d'obstacles (bâtiments, murs, planchers, toits, arbres) sur lequel les signaux sont reflétés.
- Récepteur : une erreur dans la mesure de la portée du récepteur peut être causé par une erreur d'horloge du récepteur, un bruit thermique, la précision du logiciel due à des vibrations ou à la haute altitude.

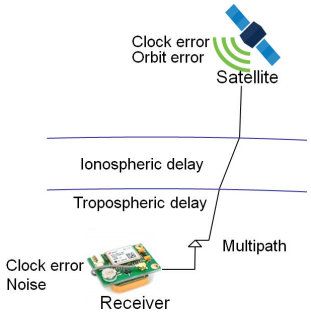



**Figure 5.5:** *Les erreurs GPS [Langley, 1997].*

— La table FMEA du GPS :

A partir de cette étude des principales erreurs de la position donnée par le GPS, nous pouvons établir la table FMEA correspondante. Nous ne prenons pas en compte les deux premières erreurs, qui sont rares dans les nouveaux GPS différentielles, et la dernière erreur peut être considérée comme l'erreur sur le moniteur, qui est le récepteur GPS. Ce qui donne la table FMEA représentée par le Tableau 5.3.

| Type d'erreur | Monitoring | Contexte d'apparition (Type d'erreur ou monitoring) |
|---|---|---|
| Ionosphère | Mesure à double fréquence | - Faible angle d'élévation<br>- Tempêtes solaires<br>- Proximité géomagnétique de l'équateur ou des pôles |
| Troposphère | Modèle en fonction de la température, de la pression et de l'angle d'élévation du satellite | - Humidité<br>- Vapeur |
| Multipath | Détection d'obstacle par SW ou capteur laser | - Terrain urbain<br>- Météo (Monitoring) |

**Tableau 5.3:** *La table FMEA de la position donnée par le GPS*

— Le réseau Bayésien de la position par GPS :

Le réseau Bayésien pour l'évaluation de l'état de santé de la position par GPS peut être obtenu à partir de la table FMEA du GPS en utilisant le modèle proposé. Le modèle évalue principalement le moniteur (le récepteur GPS) et les 3 types d'erreurs, qui eux-mêmes sont surveillées par des capteurs logiciels ou matériels, ainsi que leurs contextes d'apparition. Les capteurs eux-mêmes peuvent être défaillants et leurs santés sont renforcés par les contextes d'apparition. Le réseau Bayésien pour l'´etat de santé de la position par GPS est donné dans la Figure 5.6. Le nœud $H\_GPS$ représente la sortie du modèle et reflète l'état de santé du système. Le nœud $U\_GPS$ représente l'état "inobservable", qui est surveillé par le récepteur GPS ( nœud $S\_GPS$). Le récepteur peut être défaillant, et donc il a un nœud de santé ($H\_S\_GPS$). Les nœuds contextes d'apparition qui renforcent la fiabilité de l'état de santé du capteur récepteur GPS sont l'altitude et les vibrations. D'autre part, la santé de la position par GPS est surveillée par les erreurs externes tels que le multipath, l'ionosphère et la troposphère représentés par $U\_multipath$, $U\_ionosphere$ et $U\_troposphere$. Leur monitoring avec des modèles, des applications ou des capteurs sont représentés par des nœuds capteurs et des nœuds contexte d'apparition pour renforcer la fiabilité de ces monitorings.

2. Le cas de la consommation énergétique :

   La surveillance de la consommation énergétique est importante dans le cas d'une mission de navigation autonome. Nous proposons une extension du modèle générique en un modèle dynamique pour l'évaluation de la consommation énergétique, car elle évolue dynamiquement.

   Nous supposons que la consommation énergétique évolue d'une manière linéaire dans le mode nominal (sans perturbation ou événement interne/externe).

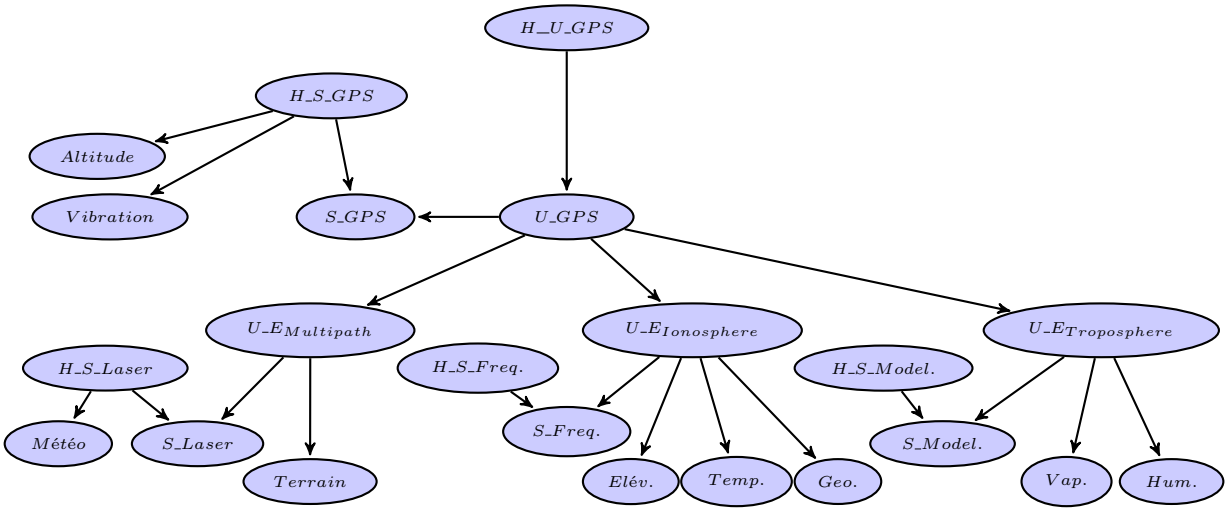



**Figure 5.6:** *Réseau Bayésien pour l'état de santé de la position par GPS.*

Les événements extérieurs tels que le vent peuvent augmenter la consommation d'énergie, ainsi que toute autre application/composant utilisés dans la période de temps. Le niveau d'énergie est donné par un capteur, qui peut être défaillant. La croyance sur la fiabilité de ce capteur peut être renforcée par des contextes d'apparition (la température, par exemple).

Le réseau Bayésien correspondant est un réseau dynamique sur deux temps (2TBN). La partie intra-slice du modèle correspond au modèle générique, et se construit de la même manière que pour le GPS avec les événements internes/externes comme types d'erreur. Les nœuds "commande" (C) représentent l'activation des applications/composants externes, et peuvent modifier la consommation. La partie inter-slice du modèle représente l'évolution du réseau entre deux temps (slice 0, slice 1). L'état de l'énergie (nœud $U\_Energy$) à l'instant $t-1$ est liée à l'état de l'énergie à l'instant $t$ à travers des nœuds observables (événements externes et applications) qui modifient l'état de l'énergie à l'instant $t$. Le réseau Bayésien pour l'état de santé de la consommation énergétique est donnée dans la Figure 5.7.

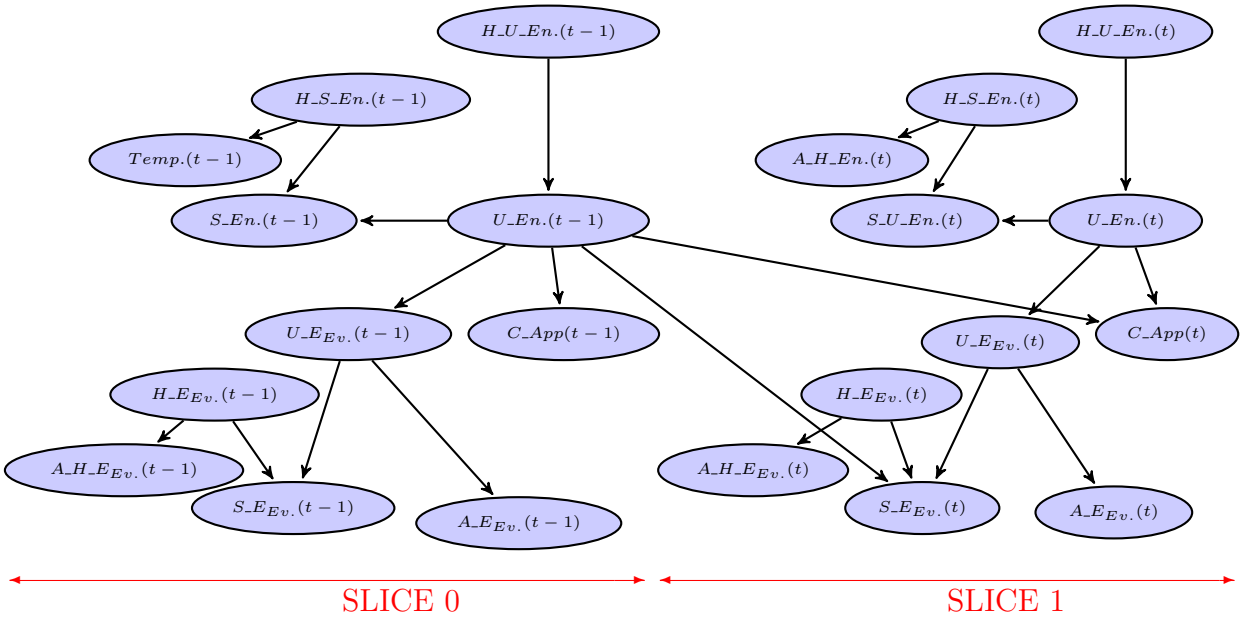



**Figure 5.7:** *Réseau Bayésien pour l'état de santé de l'énergie.*

### 5.2.4 Atelier logiciel pour le cas particulier de la FMEA

Ici, nous exposons l'adaptation de notre atelier logiciel à ce cas particulier de réseaux Bayésiens générés automatiquement à partir d'une analyse de type FMEA. Cette adaptation permet un traitement plus rapide de l'atelier par rapport aux patterns réguliers du modèle généré.

Nous abordons les points suivants de l'atelier :

1. Spécification du réseau Bayésien :

   La spécification du réseau Bayésien se fait directement à partir de la table FMEA et des tables de probabilités (définies par apprentissage ou par simulation), que ce soit pour le cas statique ou dynamique.

   La génération automatique d'un réseau Bayésien à partir de ces informations est intégrée à l'outil pour faciliter la construction dans le cas du monitoring de l'état de santé d'un système.

   Cette génération dépend des informations suivantes, récupérées de la table FMEA :

   — le nombre de contextes d'apparition du moniteur ($n$) (ou pour chaque moniteur),

   — le nombre de types d'erreur ($p$),

   — le nombre de contextes d'apparition pour chaque type d'erreur ($q$),

   — le nombre de contextes d'apparition pour le moniteur de chaque type d'erreur ($r$).

- Le cas statique :

La génération du réseau Bayésien pour le cas statique se fait comme suit :

(a) créer le nœud représentant l'état de santé du système ($H\_U$) et le nœud représentant l'état inobservable du système ($U$), fils du nœud état de santé,

(b) créer un sous-réseau Bayésien du moniteur : contenant le nœud qui représente le moniteur du système ($S\_U$), fils du nœud ($U$), le nœud état de santé de ce moniteur ($H\_S$), parent du nœud moniteur, et tous les nœuds contextes d'apparition de l'état de santé du moniteur ( $A\_H_1$, ... $A\_H_n$), des fils du nœud état de santé,

(c) créer des sous-réseaux Bayésiens pour chaque type d'erreur : contenant le nœud inobservable du type d'erreur ($U\_E_i$), fils du nœud ($U$), le sous-réseau Bayésien du moniteur du type d'erreur (comme dans b), et les nœuds contextes d'apparition des types d 'erreur, fils de $U\_E_i$,

(d) spécifier les états et les tables de probabilités pour chaque nœud comme suit :

— Dans notre cas, deux états pour chaque nœud (état bon ou mauvais, état présent ou absent, état supérieur au seuil ou inférieur au seuil, etc.).

— La majorité des tables de probabilités peuvent également être générés automatiquement. Pour les noeuds $U\_E$ liés aux types d'erreur, pour le noeud $U$ principal et pour les noeuds $S$, les valeurs sont fixées à 0 ou 1, si l'information cause à effet est confirmée, et à 0,5 si l'information est neutre. Pour les noeuds $H$, les probabilités représentent les valeurs a priori pour un bon état de santé et peuvent être initialement à 0,5. Uniquement pour les noeuds $A$, les probabilités représentent l'influence du contexte d'apparition sur la précision du capteur associé. Ces valeurs peuvent être définies par expertise, ou par simulation en appliquant des algorithmes d'apprentissage (algorithme EM, par exemple).

- Le cas dynamique (2TBN) :

La génération du réseau Bayésien pour le cas dynamique se fait comme suit, en généralisant l'algorithme du cas statique :

(a) générer le sous-réseau Bayésien pour le SLICE0, comme pour la cas statique,

(b) dupliquer ce sous-réseaux Bayésien pour le SLICE1, en incluant :

— l'ajout à tous les nœuds moniteurs ($S$) des types d'erreur du SLICE1 comme parent le nœud $U$ du SLICE0,

— la modification des tables de probabilités des nœuds moniteurs ($S$), qui dépendent dans ce SLICE des nœuds $U\_E_i$ et $H\_S_i$ ainsi que du nœud $U\_E_i$ du SLICE0.

2. Compilation en AC :

   Par rapport à la régularité du réseau Bayésien, la création de l'AC peut se faire d'une manière plus simple et plus rapide, en se basant sur les propriétés du BN statique ou dynamique sous-jacent.

   De plus l'indépendance entre les sous-réseaux des types d'erreur permet la construction modulaire et hiérarchique. La génération de l'AC peut être expliquée à partir du réseau Bayésien extrait de la table FMEA et des tables de probabilités. Ce dernier point peut être intéressant pour une génération en ligne des réseaux d'état de santé selon le besoin.

   - Le cas statique :

   La construction de l'AC se base sur les relations entre les nœuds parent et les nœuds fils. S'il y a une indépendance entre les sous-réseaux des fils, les sous-AC des fils sont indépendants entre eux et puis reliés au parent.

   Dans notre modèle, tous les sous-réseaux Bayésiens des types d'erreurs sont indépendants entre eux et du sous-réseau du moniteur. Et tous ces sous-réseaux sont liés avec le reste du réseau par le nœud inobservable ($U$) du système. Cela permet de construire l'AC d'une manière hiérarchique, modulaire et parallèle. La construction de l'AC se fait comme suit :

   (a) générer les subAC1 du sous-réseau du moniteur : le nombre de subAC1 dépend du nombre d'états de $U$, pour la construction de l'AC global et parce que le sous-réseau est relié à $U$. Si le nombre d'états de $U$, par exemple est égale à 2 ($u_0$,$u_1$), nous aurons un subAC1 prenant en compte les paramètres de $S\_U|u_0$ et un deuxième pour $S\_U|u_1$,

   (b) pour tous les types d'erreur $U\_E_i$ : générer les subAC2_i pour tout les sous-réseaux des erreurs, où le nombre de subAC2_i dépend aussi du nombre d'états de $U$,

   (c) générer l'AC complet, avec subAC , le nœud $U$ et le nœud $H$.

   Exemple de construction de l'AC pour les caractéristiques de la table FMEA suivants :

   — 2 types d'erreur,

   — 1 contexte d'apparition pour le moniteur et pour les types d'erreurs,

   — chaque nœud possède deux états (exemple $A(a_0, a_1)$).

   La Figure 5.8 représente cet exemple.

   Les étapes de construction :

   (a) Les subAC1 du moniteur : construire les subAC1 du moniteur pour tous les états de $U$ (exemple $U(u_0, u_1)$, cf Figure 5.9).

   (b) Les subAC2 d'un type d'erreur : construire les subAC2 de chaque type d'erreur pour tous les états de $U$ (exemple $U(u_0, u_1)$, cf Figure 5.10).

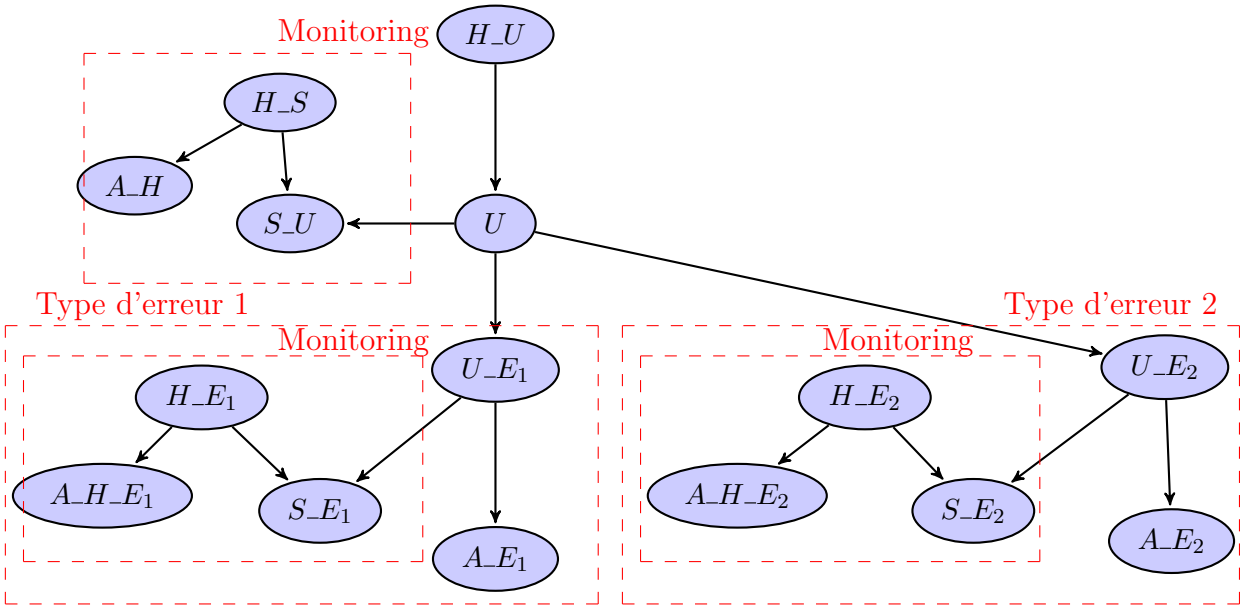
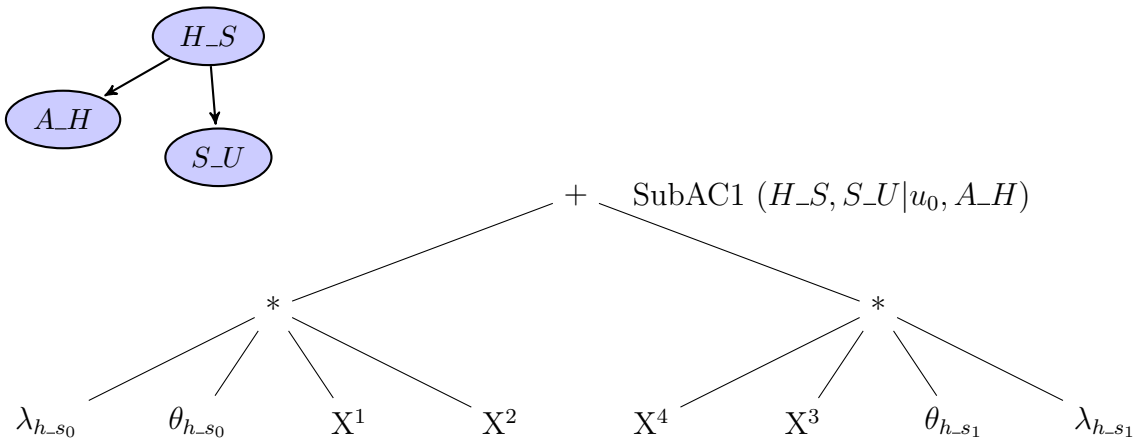


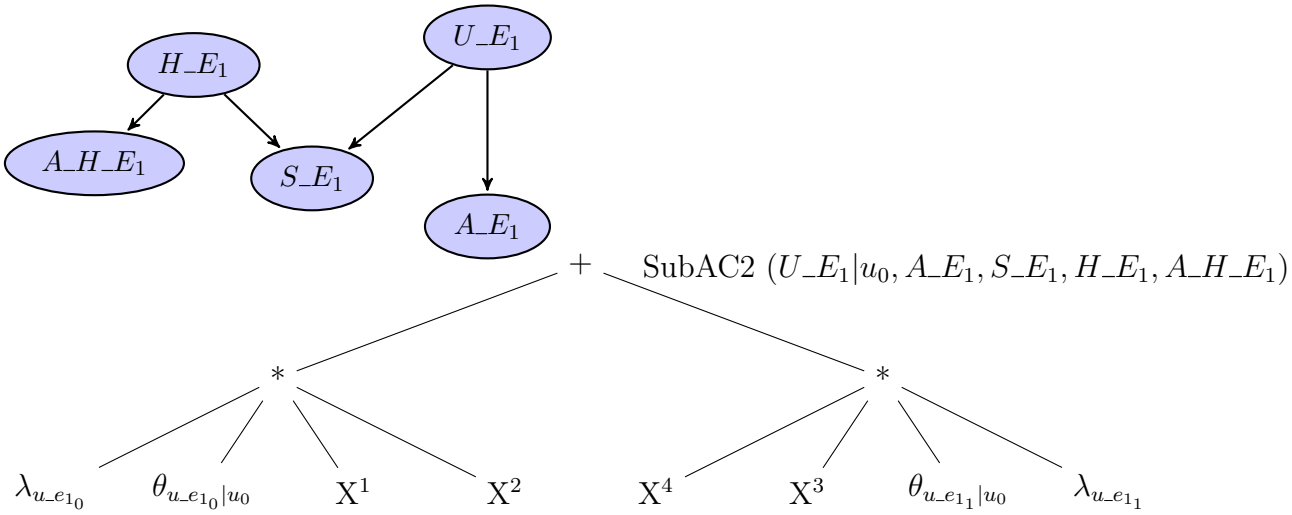

**Figure 5.8:** *Réseau Bayésien de l'exemple de la construction de l'AC.*




$$X^1 = BE1(\lambda_{a\_h_0}, \theta_{a\_h_0|h\_s_0}, \lambda_{a\_h_1}, \theta_{a\_h_1|h\_s_0}),\ X^2 = BE1(\lambda_{s\_u_0}, \theta_{s\_u_0|h\_s_0u_0}, \lambda_{s\_u_1}, \theta_{s\_u_1|h\_s_0u_0}),$$

$$X^3 = BE1(\lambda_{a\_h_0}, \theta_{a\_h_0|h\_s_1}, \lambda_{a\_h_1}, \theta_{a\_h_1|h\_s_1}),\ X^4 = BE1(\lambda_{s\_u_0}, \theta_{s\_u_0|h\_s_1u_0}, \lambda_{s\_u_1}, \theta_{s\_u_1|h\_s_1u_0}).$$

**Figure 5.9:** *Le sous-réseau Bayésien du moniteur et le SubAC1 correspondant pour le cas* $S\_U|u_0$.




$$X^1 = SubAC1(H\_E_1, S\_E_1|u\_e_{1_0}, H\_E_1), X2 = BE1(\lambda_{a\_e_{1_0}}, \theta_{a\_e_{0_0}|u\_e_{1_0}}, \lambda_{a\_e_{1_1}}, \theta_{a\_e_{1_1}|u\_e_{1_0}}),$$

$$X^3 = SubAC1(H\_E_1, S\_E_1|u\_e_{1_1}, H\_E_1), X^4 = BE1(\lambda_{a\_e_{1_0}}, \theta_{a\_e_{0_0}|u\_e_{1_1}}, \lambda_{a\_e_{1_1}}, \theta_{a\_e_{1_1}|u\_e_{1_1}}).$$

**Figure 5.10:** *Le sous-réseau Bayésien d'un type d'erreur et le SubAC2 correspondant pour le cas* $U\_E_1|u_0$.

(c) l'AC complet : construire l'AC global incluant les SubAC1 et les SubAC2 (cf. Figure 5.11).

- Le cas dynamique :

La méthode peut être généralisée au cas dynamique, où l'algorithme est plus complexe par rapport à la relation entre les deux SLICE.

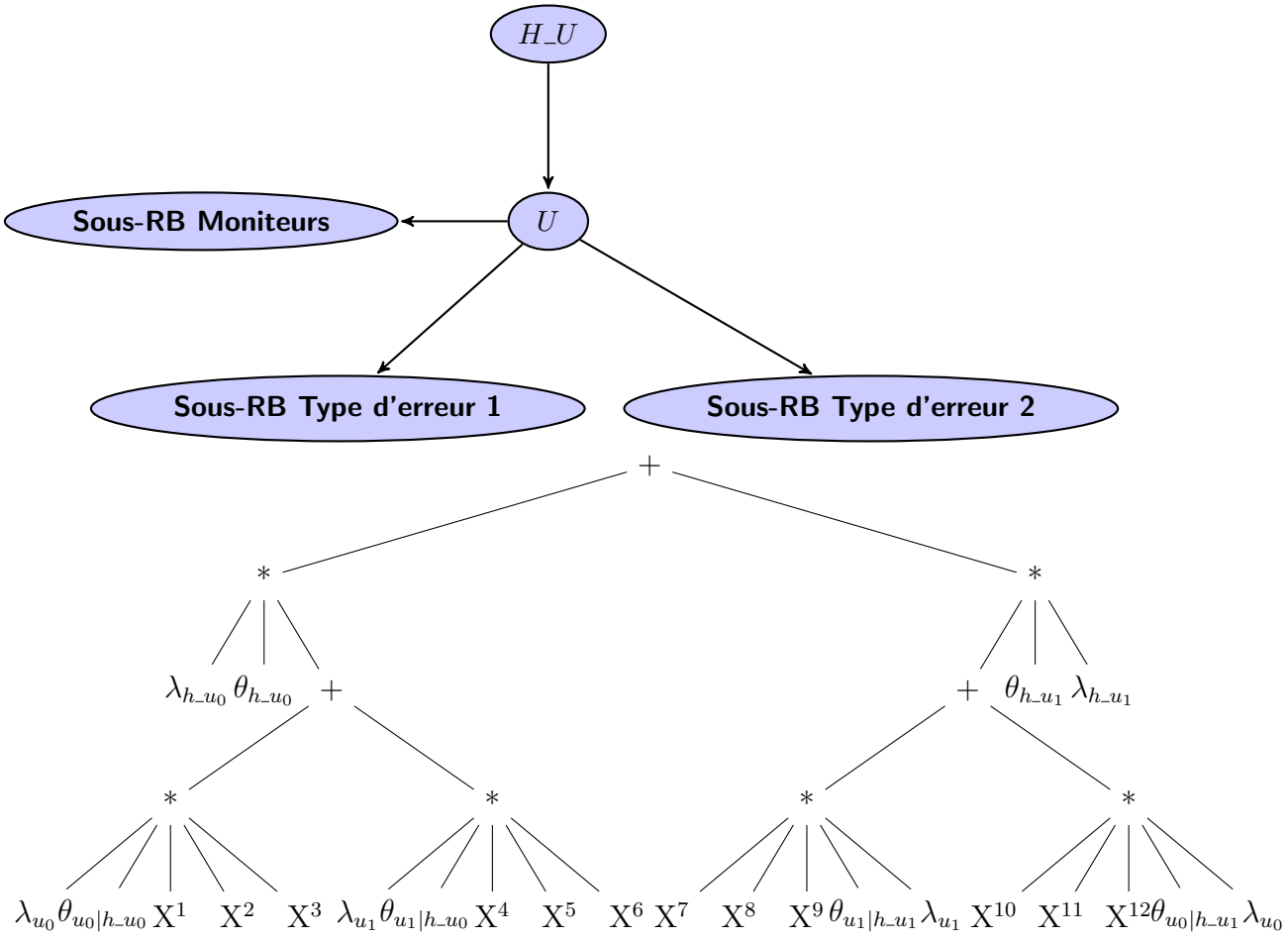
Vu les dépendances entre les nœuds moniteurs et le nœud $U$ du SLICE0, où $U$ est le parent des nœuds moniteurs des types d'erreur, la construction se fait de manière hiérarchique et modulaire comme suit :

(a) construire le SubAC (comme pour le cas statique) pour le SLICE0 en incluant :

— les SubAC1 des types d'erreur du SLICE1 de la même manière que les types d'erreur du SLICE0,

— comme les nœuds capteur sont des fils du nœud $U$ du SLICE1, faire les SubAC1 pour chaque état du nœud $U$.

(b) construire l'AC complet (comme pour le cas statique) pour les parties restantes.




$$\mathrm{X}^1 = \mathrm{X}^{12} = SubAC1(H\_S, S\_U|u_0, A\_H),\quad \mathrm{X}^4 = \mathrm{X}^9 = SubAC1(H\_S, S\_U|u_1, A\_H),$$

$$\mathrm{X}^2 = \mathrm{X}^1 1 = SubAC2(U\_E_1|u_0, A\_E_1, S\_E_1, H\_E_1, A\_H\_E_1),$$

$$\mathrm{X}^3 = \mathrm{X}^1 0 = SubAC2(U\_E_2|u_0, A\_E_2, S\_E_2, H\_E_2, A\_H\_E_2),$$

$$\mathrm{X}^5 = \mathrm{X}^8 = SubAC2(U\_E_1|u_1, A\_E_1, S\_E_1, H\_E_1, A\_H\_E_1),$$

$$\mathrm{X}^6 = \mathrm{X}^7 = SubAC2(U\_E_2|u_1, A\_E_2, S\_E_2, H\_E_2, A\_H\_E_2).$$

**Figure 5.11:** *L'AC complet de l'exemple.*

3. Optimisations :

   Dans ce type de réseaux :

   — les nœuds observables sont les nœuds $S$,

   — les nœuds non observables sont les nœuds $H$ et $U$,

   — les nœuds qui peuvent être soit observables soit non observables, selon la disponibilité de l'information, sont les nœuds $A$.

Donc les optimisations, proposées dans le chapitre précédent, peuvent être appliqués sur les nœuds $S$, $H$ et $U$.

4. Génération du code C pour la synthèse de haut niveau :
   Le code C de la compilation AC pour ce type de réseau peut être généralisé comme suit (cas statique) :

**Algorithme 5 :** Code C de l'AC généré par FMEA

**Entrée:** Indicateurs d'évidence et tables de probabilités
**pour** chaque état $s$ du nœud $U$ **faire**
Calculer SubAC2 pour tous les états $U\_E_i$ ($f\_U_{i-s}$)
**fin pour**
Multiplier tous les $f\_U_{i-s}$ par les valeurs de la table de probabilité de $U$ relatives à $s$ (1)
**pour** tout état $s$ du nœud $U$ **faire**
Calculer SubAC1 pour le moniteur ($f\_U_s$) avec les nouveaux paramètres obtenus par (1)
**fin pour**
Calculer la probabilité de l'état de santé du système $P(H\_U|e)$
**Sortie:** Health probability of the system

L'algorithme peut également être généralisé au cas dynamique.

Une fois le code C généré, les directives de synthèse utilisées pour ce type de réseau, pour l'organisation de la mémoire de données et la gestion des ressources et des performances, sont les suivants :

(a) Stockage interne utilisant des blocs BRAM pour les tables de probabilités. Nous optons pour cette représentation, car le taux de mise à jour des tables de probabilités est faible et peut être considéré statique au cours d'un scénario de mission. Les indicateurs d'évidence (valeurs booléennes) sont mis à jour pour chaque calcul à la vitesse du capteur. Pour minimiser le temps et optimiser les ressources, ils sont concaténés par 32 et envoyés via un bus $AXI\_lite$ ou le contrôleur $AXI\_BRAM$, selon la taille du réseau. Le résultat du calcul (l'état de santé) est envoyé via le bus $AXI\_lite$. Si de nombreux résultats doivent être envoyés en même temps, nous pouvons utiliser des blocs BRAM avec le contrôleur $AXI\_BRAM$.

(b) Accès parallèle aux tables de probabilités depuis les blocs BRAM pour un calcul parallèle, avec la directive $ARRAY\_PARTITION$. Les indicateurs d'évidence sont concaténés en mots de 32 bits avant d'être envoyés à l'IP. Une fois reçus, ils sont séparés en valeurs booléennes et stockés séparément dans les registres FF (Flip Flop). Si nous avons besoin d'un grand nombre d'indicateurs d'évidence, nous pouvons utiliser le partitionnement sur la réception des indicateurs d'évidence concaténées.

(c) Directives pour l'optimisation des ressources et/ou des performances. Tel que *Unroll* et *Pipeline* pour minimiser le temps nécessaire pour séparer les indicateurs d'évidence et explorer les différentes branches de l'AC en même temps dans différentes boucles. Ainsi *Inline* et *Limit allocation resource* pour pouvoir partager les ressources et contrôler le nombre d'allocations des opérations.

## 5.3 Modèle de décision par diagramme d'influence pour la mission d'un véhicule

Dans une mission, les états de santé des composants/applications servent à alimenter le réseau de décision afin de pouvoir choisir des actions de recouvrement dans le cas d'une défaillance. Comme pour les modèles de l'état de santé, nous proposons un modèle générique, basé sur les diagrammes d'influence, pour la prise de décision lors d'un scénario de mission, en se basant sur les informations des états de santé et des différentes actions de recouvrement possibles.

### 5.3.1 Modèle générique pour la décision et des scénarios de défaillance

Le modèle générique pour la prise de décision lors d'un scénario d'une mission est défini comme suit :

— Les nœuds chance : sont les nœuds $H\_U$ des composants/ applications du scénario de la mission,

— Le nœud action : comprend les différentes actions de recouvrement pour le scénario de mission dans le cas d'une défaillance,

— La table d'utilité : représente les différents scores pour chaque action lors des différents scénarios de défaillance.

La Figure 5.12 représente le modèle générique pour la décision pour les scénarios de défaillance.

### 5.3.2 Exemple de décision lors d'une mission de drone

Nous présentons un scénario complet d'une mission de drone avec différents scénarios de défaillance et actions de recouvrement (save Outback Joe [Tridgell, 2014]) pour montrer l'intégration d'une mission à notre modèle d'état de santé et de décision. Puis, nous illustrons la décision par diagrammes d'influence sur un scénario de défaillance.

1. La mission "save Outback Joe" : la mission consiste à chercher, à l'aide d'un drone, une personne vétu d'un gilet jaune et à la sauver par la suite.

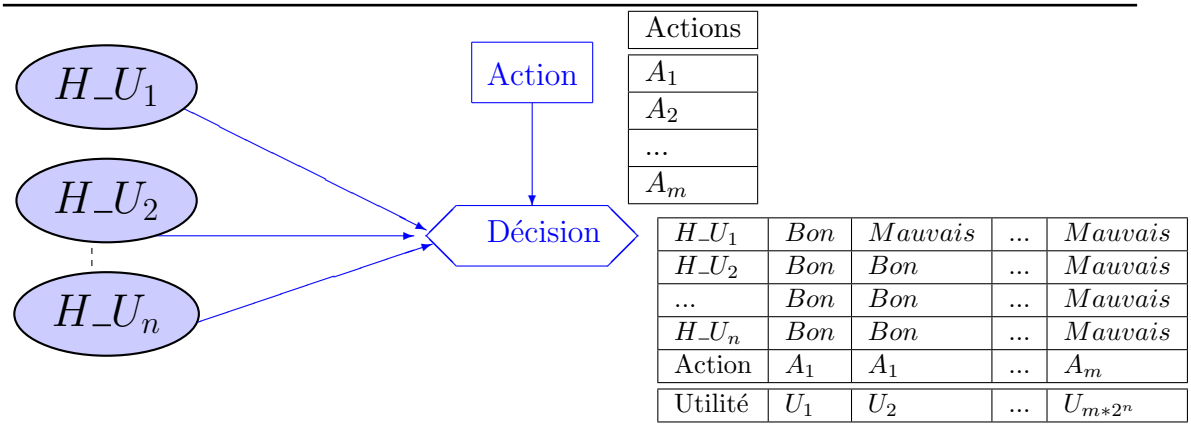



**Figure 5.12:** *Modèle générique pour la prise de décision lors d'une mission.*

Cette mission contient les étapes suivantes :

- Décollage ($E_0$).
- Se diriger vers la zone de recherche (en suivant une trajectoire connue) ($E_1$).
- Patrouille : chercher la personne au gilet jaune avec des algorithmes de traitement d'image pour détecter des objets ($E_2$).
- Identification : voir si l'objet détecté correspond à la recherche ($E_3$).
- Atterrissage : lorsque la personne est trouvée ou à la fin de la mission ($E_4$).

Au cours de ces différentes étapes de la mission, des scénarios de défaillance peuvent apparaître tels que :

- Une défaillance sur le GPS ($H\_FS_{GPS}$) ou la batterie ($H\_FS_{Batterie}$) à n'importe quel moment peut causer une perte du drone ou un crash.
- Une perturbation météorologique ($H\_FS_{Météo}$) tel que le vent, les nuages, la pluie peuvent causer des défaillances matérielles et logicielles.
- Des obstacles ($H\_FS_{Obstacle}$) tels que les oiseaux ou d'autres drones peuvent causer des défaillances matérielles ou un crash.

A partir de ces scénarios des actions de recouvrement peuvent être générées tels que :

- Changement de mode de localisation ($A_1$) si le GPS ne fonctionne plus.
- Atterrissage d'urgence ($A_2$) dans le cas d'une batterie insuffisante ou endommagée.
- Stand-by ($A_3$) ou changement de trajectoire ($A_4$) dans le cas d'un obstacle.
- Stand-by ou diminution de la vitesse ($A_5$) dans le cas d'une perturbation météorologique, etc.

La Figure 5.13 résume cette mission.

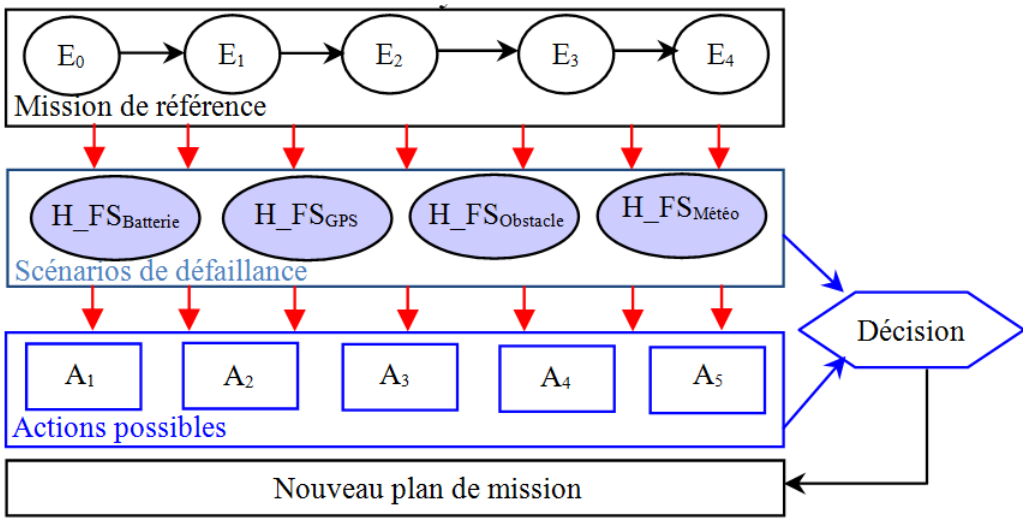



**Figure 5.13:** *Machine à état de la mission "save Outback Joe".*

L'intégration de cette mission à notre modèle se fait comme suit :

- Pour chaque étape de la mission, définir les réseaux Bayésiens d'état de santé et de la décision pour chaque composant matériel ou logiciel utilisé, à partir de la table FMEA du composant.
- Définir pour chaque étape de la mission, les actions de recouvrement possibles selon les scénarios de défaillance utilisés.
- Définir la table d'utilité pour les décisions intégrant les scénarios de défaillance et les actions, qui représente le score de chaque action selon les différents états des scénarios de défaillance.

L'interaction entre la mission et notre modèle est résumé dans la Figure 5.14.

2. Pour illustrer la décision, nous prenons un scénario de défaillance du GPS et de la batterie avec comme actions de recouvrement, un changement de méthode de localisation ou un atterrissage d'urgence. Par conséquent, nous avons deux réseaux Bayésiens pour l'état de santé (la localisation GPS et la consommation d'énergie) et 3 actions. Le modèle de décision peut donc être représenté par un diagramme d'influence, reliant les deux réseaux Bayésiens et les actions de recouvrement par une table d'utilité (cf. Figure 5.15). Cette table donne un score pour chaque action de recouvrement par rapport aux états de santé de la localisation par GPS et de la consommation énergétique.

   La table d'utilité peut être définie par un expert, évaluant les risques de chaque action selon les cas. Le choix de l'action de recouvrement la plus adéquate est donnée en calculant la fonction d'utilité ($U_f$) pour chaque action.

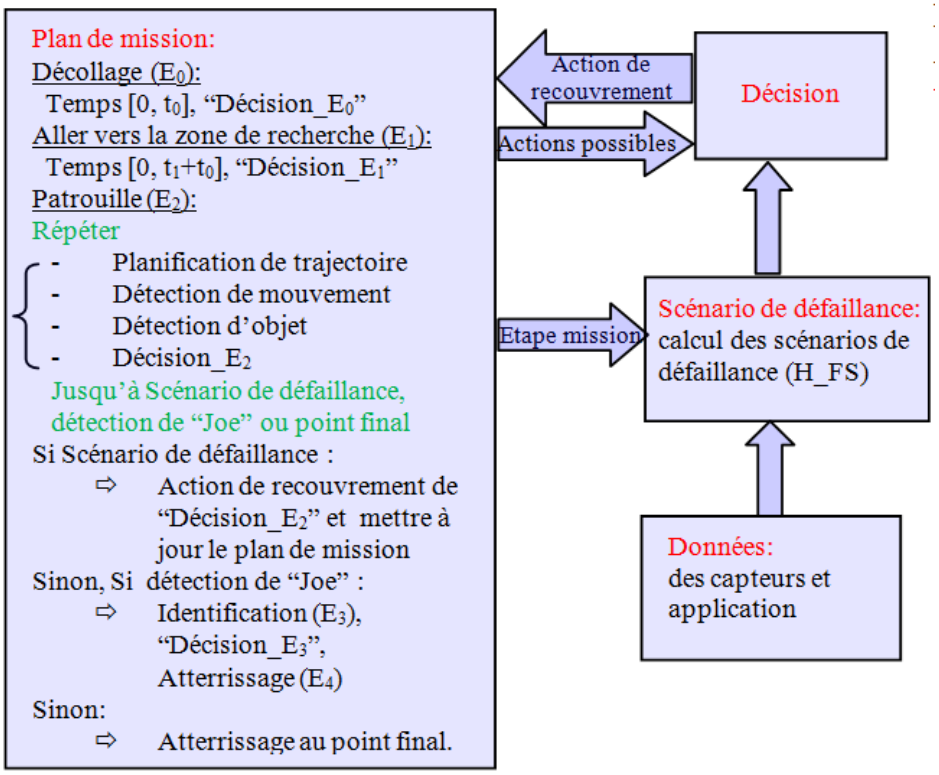
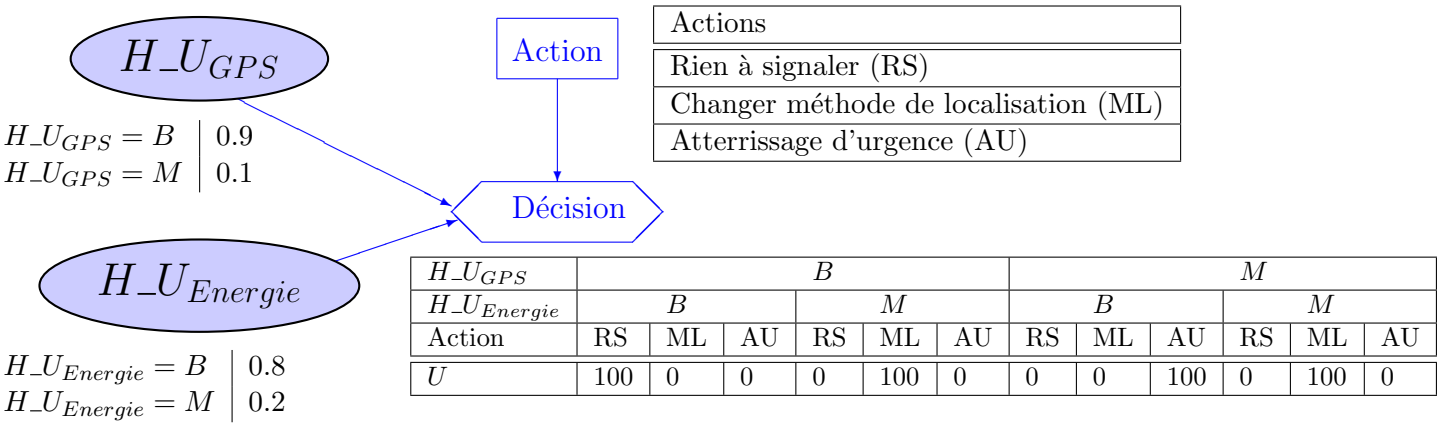



**Figure 5.14:** *L'interaction entre la mission "save Outback Joe" et notre modèle.*




| $H\_U_{GPS}$ | $B$ | | | | | | $M$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H\_U_{Energie}$ | $B$ | | | $M$ | | | $B$ | | | $M$ | | |
| Action | RS | ML | AU | RS | ML | AU | RS | ML | AU | RS | ML | AU |
| $U$ | 100 | 0 | 0 | 0 | 100 | 0 | 0 | 0 | 100 | 0 | 100 | 0 |

**Figure 5.15:** *Réseau de décision pour un exemple de mission.*

Pour calculer la fonction d'utilité, nous avons besoin de la probabilité de la santé du GPS et de l'énergie. La fonction d'utilité de chaque action est égale à la somme des produits de l'action avec la probabilité adéquate de l'état de santé du GPS et de l'énergie. L'action de recouvrement est l'action qui maximise la fonction d'utilité.

Pour l'exemple de la Figure 5.15 :

Action de recouvrement = $Max(U\_f_{RS}, U\_f_{ML}, U\_f_{AU})$

où chaque $U\_f_k(k = \{RS, ML, AU\})$ est égale à :

$$U\_f_k = \sum_i \sum_j U(A = k, H_{GPS} = i, H_{Energie} = j) * P(H_{GPS} = i) * P(H_{Energie} = j)$$
$$(i = \{Bon, Mauvais\} \quad et \quad j = \{Bon, Mauvais\})$$

Dans cet exemple, si nous prenons les probabilités des deux états de santé des scénarios de défaillance et la table d'utilité de la Figure 5.15, $U\_f_{NR} = 72$, $U\_f_E = 20$ and $U\_f_{LM} = 8$ Ce qui signifie qu'il n'y a rien à signaler dans ce cas.

### Validation du modèle de l'état de santé et de la décision

La Figure 5.16 montre un exemple d'exécution pour la validation du modèle du monitoring et de la décision avec le réseau du GPS et de l'énergie et les différentes actions de recouvrement.

La Figure 5.16. (a) montre l'intérêt des nœuds contextes d'apparition. Pour cela, nous avons dans un premier temps calculé la probabilité de l'état de santé du système sans avoir d'observation sur les contextes d'apparition, et puis en considérant des observations sur ces nœuds. Par exemple, à partir du temps t= 0 jusqu'à t= 4, nous n'observons aucun problème dans le réseau du GPS, la probabilité de la santé du système est de 0,835 sans les contextes d'apparition puis elle passe à 0,915 avec les observations des contextes d'apparition. A l'instant t= 4, nous supposons un problème dans le réseau du GPS. Sans observation sur les nœuds contextes d'apparition, la probabilité de la santé du système est égale à 0,365 mais en rajoutant les observations sur les contextes d'apparition, la probabilité diminue à 0,143. Cela signifie que les nœuds contextes d'apparition aident à renforcer les informations données par les capteurs.

La Figure 5.16. (b) montre le monitoring de la consommation d'énergie dans un cas nominal. Avec des observations sur les capteurs et les contextes d'apparition, la consommation d'énergie est dans un état 'bon' mais diminue avec le temps parce que le réseau Bayésien associé évolue dynamiquement.

La Figure 5.16. (c) montre l'évolution de la prise de décision dans le temps en fonction des probabilités de l'état de santé du réseau du GPS et de la consommation d'énergie. Du temps t= 0 à t= 4, le maximum des fonctions d'utilité est obtenue pour l'action 'rien à signaler', ce qui reflète le cas "pas de problème dans la mission". Au temps t = 4, le maximum des fonctions d'utilité est obtenue pour l'action 'changer méthode de localisation', ce qui reflète "un problème dans la mission", c'est-à-dire que le scénario de défaillance du GPS est détecté.

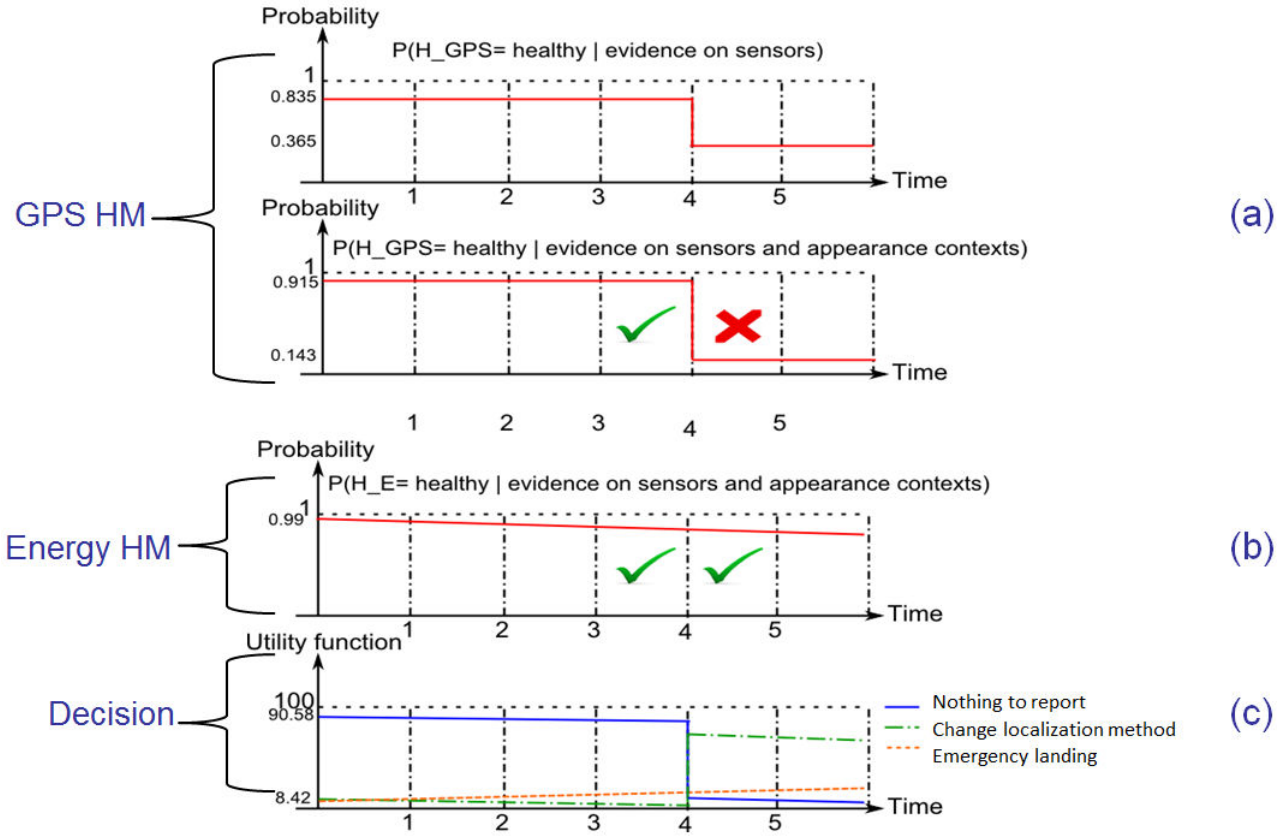



**Figure 5.16:** *Validation du modèle sur la mission de navigation [Zermani et al., 2017a].*

### 5.3.3 Atelier logiciel pour le cas particulier de la décision de mission

Ici, nous exposons l'adaptation de notre atelier logiciel au cas particulier de réseaux de décision générés automatiquement à partir des réseaux Bayésiens de l'état de santé et des actions de recouvrement.

Nous abordons les points suivants de l'atelier :

1. Génération du réseau de décision :

   La génération du diagramme d'influence en (.net) peut se faire automatiquement à partir des réseaux Bayésiens, des actions de recouvrement et de la table d'utilité.

2. Génération du code C pour la synthèse de haut niveau : Le code C du calcul de la fonction d'utilité pour les actions de recouvrement dans un réseau de décision d'un scénario de mission peut être généralisé comme suit :

**Algorithme 6 :** Prise de décision lors d'un scénario de mission

**Entrée:** Etats de santé $HM_1$, ..., $HM_n$, Actions $A_1$, ..., $A_n$ et table d'utilité $U$

**pour tout** état $i_1$ de $HM_1$ **faire**

...

**pour tout** état $i_n$ de $HM_n$ **faire**

**pour tout** action $A_k$ **faire**

// Calculer la fonction d'utilité pour chaque action

$$U_{f\_A_k} = U_{f\_A_k} + U(A_k, HM_1 = i_1, ..., HM_n = i_n) * P(HM_1 = i_1) * ... * P(HM_n = i_n)$$

**fin pour**

**fin pour**

...

**fin pour**

**Sortie:** Maximum de $U_f\_A_k$

Une fois le code C généré, nous appliquons les directives de synthèse, expliquées dans le chapitre précédent, pour la représentation des données, l'organisation de la mémoire des données et la gestion des ressources et des performances.

## 5.4 Synthèse et conclusion

Dans ce chapitre, nous avons abordé les contributions apportées à la gestion de l'état de santé des composants/ applications et à la prise de décision lors de scénarios de mission des systèmes autonomes et adaptatifs. Nous avons proposé un modèle de réseau Bayésien générique (statique et dynamique) pour le monitoring en se basant sur une analyse de défaillance. Nous avons complété cette étude par la proposition d'un réseau de décision, avec des diagrammes d'influence, prenant en compte les états de santé donné par le monitoring, des actions de recouvrement et une table d'utilité. Nous avons aussi montré l'adaptation de l'atelier logiciel, proposé dans le chapitre précédent, à ce cas d'étude.

Cette deuxième contribution a abouti à des publications dans un journal et deux conférences internationales [Zermani et al., 2017b, 2016, 2017a], avec une mise en œuvre qui sera détaillée dans le chapitre suivant où nous exposons des expérimentations sur notre plateforme cible, la Zedboard de Xilinx (SoC hybrid), dans le but de valider nos approches.

# - Chapitre 6 -

# Application sur une architecture hybride

## Sommaire

## 6.1 Introduction

Nous présentons dans ce chapitre nos différentes expérimentations sur un SoC hybride incorporant un processeur ARM pour l'implémentation SW et une partie FPGA pour l'implémentation HW [Crockett et al., 2014]. Le but est de valider notre outil, détaillé dans le Chapitre 4, et d'évaluer les ressources, les performances, et la consommation énergétique selon les différents critères, cités dans les chapitres précédents. Notre plateforme cible est la carte ZedBoard de Xilinx [Zed], incorporant un processeur Zynq hybride. Comme le montre la Figure 6.1, l'architecture est construite autour du processeur ARM Cortex-A9 (processeur Zynq PS). Le processeur communique avec des accélérateurs HW dédiés, implémentés dans la partie FPGA, en utilisant la logique programmable via des bus AXI (Advanced eXtensible Interface).

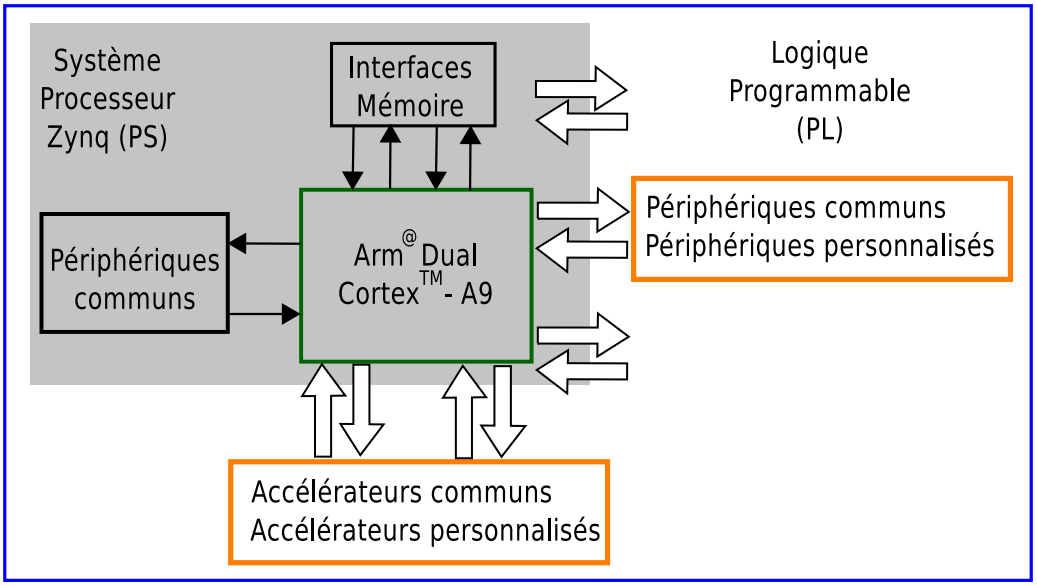



**Figure 6.1:** *L'architecture hybride du processeur Zynq pour le déploiement SoC [Crockett et al., 2014].*

Dans ce chapitre, nous commençons par expliquer le contexte de nos expérimentations. Nous donnons par la suite quelques résultats des expérimentations de l'atelier logiciel pour des exemples d'illustration et des réseaux Bayésiens virtuels. Nous exposons, aussi, différents résultats concernant l'approche "état de santé à partir de l'analyse FMEA et décision". Et finalement, nous concluons ce chapitre.

## 6.2 Contexte des expérimentations

Après la génération du code C, utilisant la partie Hors-ligne de notre atelier logiciel, nous nous basons sur l'outil Vivado de Xilinx pour l'implémentation sur la ZedBoard. 3 étapes indispensables pour une exécution sur la carte sont :

1. La synthèse de haut niveau et la génération du RTL :
   Nous utilisons l'outil Vivado HLS, non seulement pour la préparation des interfaces et les différentes directives (Chapitre 4), mais aussi pour la synthèse et la génération du RTL. Vivado HLS nous donne une estimation de la latence et des ressources utilisées en DSP, LUT et FF. Pour la Zedboard, le nombre total de DSP est égal à 220, le nombre de LUT et de FF est de 53200 et 106400, respectivement.
2. La génération du bitstream :
   Une fois le RTL généré, nous utilisons Vivado pour intégrer l'IP HLS à l'architecture ainsi que les différents Blocs HW nécessaires pour l'interface et la communication entre le CPU et l'IP. Dans notre cas, il s'agit des blocs BRAM, des contrôleurs BRAM, des DMA, et d'un Timer dans le but de comparer les temps SW et HW. Après la validation de l'architecture, le bitstream est généré et nous pouvons évaluer les ressources utilisées en BRAM, DSP, LUT et FF ainsi que la puissance (qui nous permettra de calculer la consommation énergétique).
3. Exécution sur la ZedBoard :
   Une fois le bitstream généré et exporté, nous utilisons Xilinx SDK pour tester et exécuter le SW et le HW de nos approches et pour calculer les performances de chacune pour obtenir l'accélération ainsi que la consommation énergétique. La consommation énergétique est calculée par l'Equation 6.1.

$$Consommation\ énergétique = Puissance * \frac{Temps\_Cyles}{Fréquence\_Horloge}, \tag{6.1}$$

   où $Fréquence\_Horloge = 100Mhz$.

## 6.3 Evaluation des approches proposées et résultats

Dans cette section, nous exposons nos expérimentations et résultats en deux parties :

1. Les résultats concernant notre atelier logiciel :
   Pour cela nous commençons par 3 exemples d'illustration, inspirés de l'étude de Schumann [Schumann et al., 2011]. Nous montrons à travers ces 3 exemples les points suivants :

- la comparaison entre l'inférence par AC et par arbre de jonction pour confirmer le choix de notre algorithme,
- les gains des optimisations proposées,
- la comparaison entre la méthode de décomposition de Schumann et notre décomposition, sous contraintes de ressource et de performance,
- la comparaison entre une représentation de données en virgule fixe et en flottant,
- les implémentations SW/HW sur la ZedBoard et les accélérations obtenues.

Puis nous généralisons à un réseau Bayésien virtuel, où nous évaluons par rapport à la taille du réseau Bayésien certains points des exemples précédents.

2. Les résultats du cas particulier des FMEA et de la décision dans un scénario de mission :
   Comme pour la partie précédente, nous commençons par 2 exemples de scénarios de défaillance (le GPS et l'énergie) et la décision dans un scénario de mission de navigation. Nous montrons à travers ces exemples les implémentations HW/SW sur la Zedboard en évaluant les ressources, les performances et les consommations énergétiques, et en appliquant des directives pour une adaptation selon les besoins. Puis nous généralisons à des réseaux Bayésiens FMEA et des réseaux de décision génériques où nous abordons les points suivants :
   - les résultats des implémentations SW/HW en variant le nombre de types d'erreur du réseau Bayésien,
   - les différentes stratégies d'implémentation avec une variation du nombre de réseaux Bayésiens de type FMEA,
   - le choix d'implémentations SW/HW selon le nombre d'actions et de réseaux Bayésiens d'un réseau de décision.

### 6.3.1 Résultats pour la validation de l'atelier logiciel

Les 3 exemples d'illustration sont représentés dans la Figure 6.2. Ces réseaux Bayésiens sont inspirés de l'étude de Schumann pour le diagnostic d'un système de drone où :

- Le premier (Figure 6.2-a) représente un monitoring d'une opération d'écriture dans un fichier. Le réseau se compose d'un nœud commande externe d'écriture (nœud $C$), et d'un nœud capteur logique (noeud $S$) indiquant si le fichier est plein ou s'il ya suffisamment d'espace pour écrire.
- Le deuxième (Figure 6.2-b) représente un monitoring d'un tangage vers le haut ou vers le bas. Le réseau se compose d'un nœud commande externe de tangage vers le haut (nœud $C1$) ou vers le bas (nœud $C2$), et d'un nœud capteur matériel "accéléromètre" (nœud $S$) indiquant une accélération si l'UAV est en tangage vers le haut et un ralentissement si le tangage est vers le bas.

— Le troisième (Figure 6.2-c) représente un monitoring du calcul de l'altitude. Le réseau se compose d'un nœud commande externe de calcul d'altitude (nœud $C$), et de trois nœuds capteur matériel (nœud $S1$, nœud $S2$ et nœud $S3$) indiquant l'altitude. En plus des nœuds cités, à chaque réseau Bayésien est associé un nœud état interne du système (nœud $U$), et deux nœuds état de santé $H$ (un pour le système et un pour le capteur).

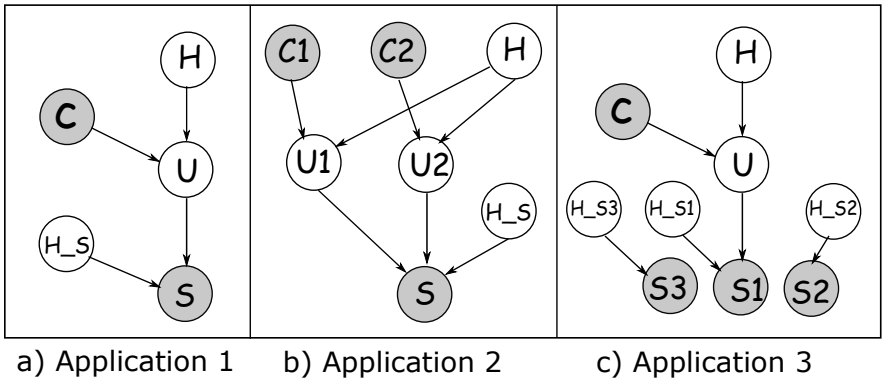



**Figure 6.2:** *Les exemples d'illustration.*

1. L'inférence JT vs l'inférence AC :
   Nous commençons par une étude comparative entre l'inférence avec l'algorithme arbre de jonction (JT), disponible sous Matlab, et l'algorithme AC, que nous avons proposé. Nous utilisons l'algorithme JT parce que c'est un algorithme classique connu. Le Tableau 6.1 représente le temps de calcul avec les deux algorithmes ainsi que les accélérations AC obtenues. Nous pouvons observer que les accélérations AC varient entre 2 et 3 pour ces 3 applications. La meilleure accélération est obtenue avec l'Application 1, où le nombre de nœuds est plus petit et les dépendances entre les nœuds sont moins complexes, et par conséquence les tailles des tables de probabilités sont plus petites.

| Réseau Bayésien | Temps JT ($ms$) | Temps AC ($ms$) | Accélération |
|---|---|---|---|
| Application 1 | 3.5 | 1.1 | 3.18 |
| Application 2 | 5.9 | 2.1 | 2.81 |
| Application 3 | 9.1 | 3.5 | 2.60 |

**Tableau 6.1:** *Algorithme JT vs algorithme AC.*

Notons que :

$$\text{Accélération} = \frac{\text{temps JT}}{\text{temps AC}} \tag{6.2}$$

**Analyse et synthèse**

Ces résultats montrent que l'algorithme AC est plus efficace que l'algorithme JT. Ce résultat n'est pas vraiment nouveau (voir [Lian et al., 2011]) mais il met l'accent sur le gain obtenu avec l'algorithme. La variation de l'accélération dépend principalement du nombre de nœuds, des dépendances entre les nœuds et de la taille des tables de probabilités.

2. Les optimisations :

   Les Figures 6.3 et 6.4 présentent les résultats obtenus en appliquant les optimisations proposées lors de la connaissance des nœuds observables et non observables. Dans ces applications les nœuds observables sont les nœuds $S$ et $C$ (leurs $\lambda$ sont à $(1,0)$ ou $(0,1)$), et les nœuds non observables sont les nœuds $H$ et $U$ (leurs $\lambda$ sont à $(1,1)$). Prenons par exemple l'Application 1, l'AC sans optimisations contient 5 niveaux de nœuds (+), et chaque nœud (+) contient 2 nœuds (*) avec 3 paramètres. En supprimant les calculs redondants, nous trouvons 44 multiplications et 13 additions dans l'AC. En utilisant les optimisations, nous commençons par supprimer les branches $\lambda$. Puis on supprime la branche gauche ou droite du nœud $C$ (niveau 3 de l'AC) et du nœud $S$ (niveau 5), car une des branches est égale à 0. Par conséquence, le nombre de multiplications est réduit à 12 et le nombre d'additions à 5.

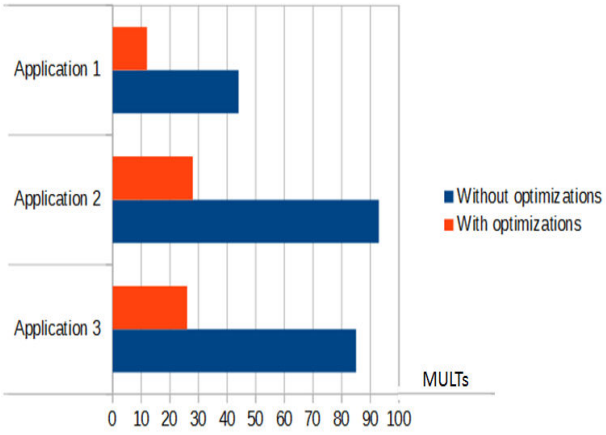



**Figure 6.3:** *Les optimisations sur le nombre de multiplications.*

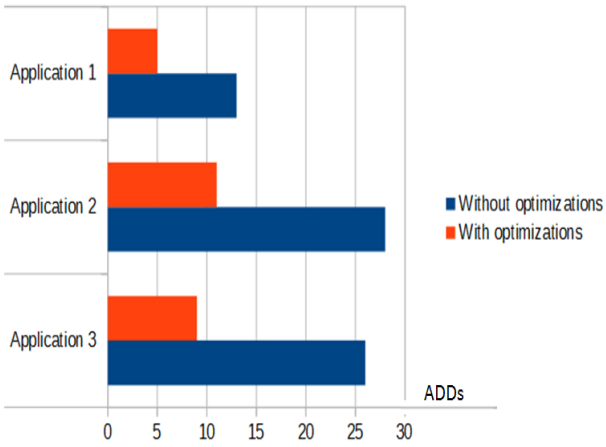



**Figure 6.4:** *Les optimisations sur le nombre d'additions.*

**Analyse et synthèse**

Ces résultats montrent l'intérêt des optimisations que nous avons proposé lors de la connaissance des noeuds observables et des noeuds non observables. Cela permet de simplifier le graphe, et par conséquent, de réduire le nombre d'opérations, c'est-à-dire les additions (ADD) et les multiplications (MULT).

3. Décomposition Schumann vs notre décomposition :
   Avant de donner les résultats de la comparaison avec la décomposition de Schumann, nous donnons un résultat avec notre approche entre deux propositions d'implémentation possibles :
   — Proposition 1 : trois accélérateurs HLS indépendants.
   — Proposition 2 : un accélérateur HLS adaptatif pour gérer les trois applications.

   La Figure 6.5 représente la surface occupée sur le FPGA et le Tableau 6.2 résume la latence, pour les trois exemples avec les deux propositions, pour un parallélisme maximal. Nous pouvons voir que la Proposition 1 utilise plus de ressource et donc moins de latence. Nous expliquons cela par le fait que les 3 applications sont indépendantes et que chacune utilise des ressources sur le FPGA. La Proposition 2 donne une solution qui utilise moins de ressources FPGA et offre de bonnes performances. La latence de chacune est donc la latence de l'application la plus complexe (ici Application 2) et la latence globale des 3 applications à 330 cycles, ce qui peut être expliqué par le pipeline (détaillé plus tard). En fonction des besoins de la mission, nous pouvons choisir de charger la Proposition 1 ou 2 avec les schémas de reconfiguration appropriés (reconfiguration partielle ou totale). Pour la comparaison qui suit, nous avons pris la Proposition 2.

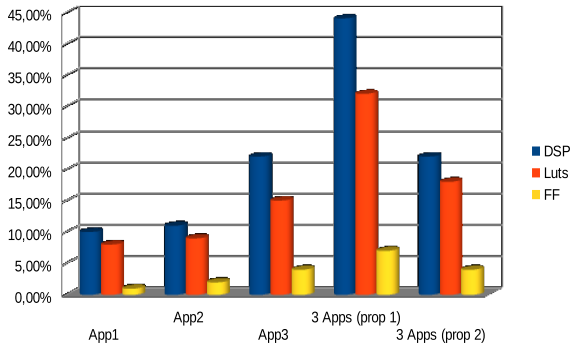



**Figure 6.5:** *Les ressources utilisées pour les deux propositions d'implémentation.*

| | | Latence (*cycles*) |
|---|---|---|
| Proposition 1 | Application 1 | 272 |
| | Application 2 | 306 |
| | Application 3 | 280 |
| Proposition 2 | All applications | 333 |

**Tableau 6.2:** *La latence pour les deux propositions d'implémentation.*

Nous comparons la décomposition de Schumann, basée sur des blocs génériques, et notre décomposition des blocs dédiés (BE0, BE1). Nous comparons les deux méthodes sous contraintes de temps puis sous contraintes de ressources. Nous évaluons sous contraintes de temps, la surface occupée sur le FPGA par chaque méthode pour une même latence. Ensuite, nous évaluons sous contraintes de ressource, la latence pour la même surface disponible.

Nous considérons d'abord les modèles sous contraintes de latence et nous comparons les ressources résultantes après la synthèse (synthèse sous contraintes de temps). Nous entendons par synthèse sous contraintes de temps que pour

la même latence nous évaluons la surface occupée sur le FPGA. Dans une deuxième étape, nous prenons la méthode de Schumann et la méthode avec des blocs élémentaires et comparons la latence pour le cas où les contraintes sont sur les ressources du FPGA (synthèse sous contraintes de temps). Par synthèse sous contraintes de ressource, on entend l'évaluation de la zone occupée sur le FPGA pour la même latence.

La Figure 6.6 présente les résultats de comparaison de la synthèse sous contraintes de temps. Pour une latence de 80 cycles, notre approche, basé sur le *BE* (Bloc Elémentaire), utilise moins de ressources que la représentation de Schumann, car les blocs de Schumann sont génériques et contiennent plus de ressources, inutilisables selon le cas.

La Figure 6.7 présente les résultats de la synthèse sous contraintes de ressource. Nous varions le nombre de LUTs et évaluons la latence obtenu. Nous pouvons voir que notre approche produit de meilleurs résultats en termes de latence. Avec une petite surface, l'exploitation du parallélisme n'est pas significative avec les blocs de Schumann et donc le calcul prend plus de temps, mais il l'est dans notre approche car nos blocs sont plus petits. Plus la surface disponible augmente, plus le parallélisme est exploité et par conséquence la latence diminue, jusqu'à un maximum de parallélisme à 17%.

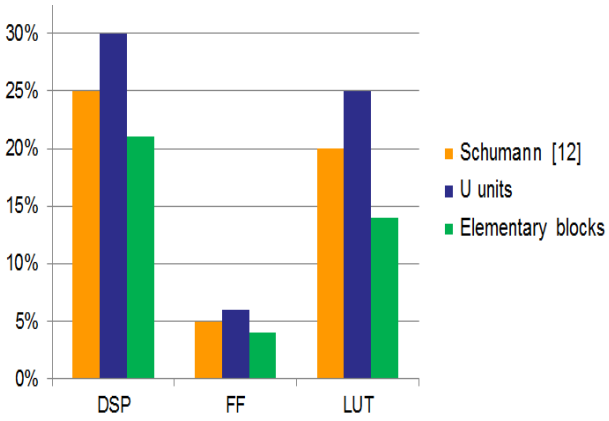



**Figure 6.6:** *Décomposition Schumann vs notre décomposition sous contraintes de temps.*

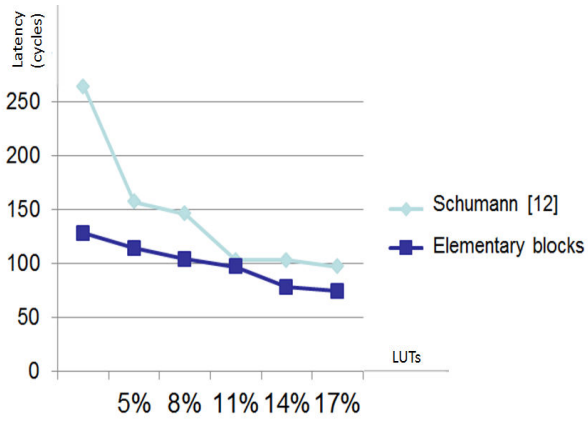



**Figure 6.7:** *Décomposition Schumann vs notre décomposition sous contraintes de ressource.*

### Analyse et synthèse

Ces résultats montrent que notre solution avec des blocs élémentaires (BE) dédiés offre une meilleure latence avec moins de ressources. Cela revient à la taille des BE qui est plus petite et donc nous pouvons utilisés un grand nombre en parallèle, contrairement aux blocs de Schumann qui sont génériques et par conséquence, ils utilisent plus de ressources.

4. La représentation des données :

   Dans les expérimentations précédentes, les données sont représentées en flottant. Dans les réseaux Bayésiens, les valeurs des tables de probabilités et les valeurs intermédiaires sont des probabilités inférieures ou égales à 1. Donc, il est possible d'utiliser une représentation en virgule fixe avec 1 bit pour la partie entière et différentes longueurs, selon la précision souhaitée, pour la partie fractionnaire. Nous comparons les ressources, les performances et les variations de précision pour choisir la longueur la plus appropriée.

   En se basant sur les règles de l'arithmétique des intervalles sur la multiplication et l'addition, on peut définir la valeur de l'intervalle du résultat en utilisant l'enchaînement des BE.

   Soit $x$ et $y$ les entrées pour une addition ou une multiplication et associons à chacune d'elles un intervalle :

$$D_x = [x_{min}, x_{max}], D_y = [y_{min}, y_{max}]$$

   Les résultats des opérations d'addition ou de multiplication varient en considérant les intervalles suivants :

$$D_{x+y} = [x_{min} + y_{min}, x_{max} + y_{max}],$$

$$\begin{aligned} D_{x*y} = [&min(x_{min}y_{min}, x_{min}y_{max}, x_{max}y_{min}, x_{max}y_{max}), \\ &max(x_{min}y_{min}, x_{min}y_{max}, x_{max}y_{min}, x_{max}y_{max})] \end{aligned}$$

   Si l'on prend $n$ comme la longueur de la partie fractionnaire, l'intervalle de la marge d'erreur ($e$) des entrées est $D_e = [0, 2^{-n}]$. Par exemple, en appliquant ces règles sur un BE1 (2 multiplications et une addition), $D_{BE1} = [0, 3*2^{-n+1}]$. La marge d'erreur d'un AC dépend du nombre d'opérations sur le chemin le plus long, qui dépend lui-même de la profondeur de l'AC. Par conséquence, l'erreur associée à une implémentation matérielle de l'AC est définie par l'expression : $(2^P - 1) * 2^{-n+2}$ ($P$ est la profondeur du circuit arithmétique par rapport aux nœuds (+) et $n$ est la longueur de la partie fractionnaire).

   Nos résultats sont résumés dans le Tableau 6.3 et la Figure 6.8. Les ressources et la latence sont données pour les 3 applications avec la Proposition 1, et la marge d'erreur est détaillée pour chacune d'entre elles.

   Le Tableau 6.3 montre que le nombre de ressources utilisées avec la représentation en virgule fixe est plus faible que dans le cas d'une représentation en virgule flottante. La latence et le nombre de DSP diminuent en continu jusqu'à 18 bits, puis stagnent (18 est la taille d'une entrée d'un DSP).

| | flottant | fixe (1,32) | fixe (1,24) | fixe (1,18) | fixe (1,12 |
|---|---|---|---|---|---|
| Latence | 74 | 78 | 48 | 24 | 24 |
| DSPs | 61 | 60 | 30 | 18 | 18 |
| FFs | 6231 | 2094 | 1572 | 1242 | 834 |
| LUTs | 9509 | 1497 | 1159 | 967 | 691 |
| Marge d'erreur app 1 | - | $15 * 2^{-30}$ | $15 * 2^{-22}$ | $15 * 2^{-16}$ | $15 * 2^{-10}$ |
| Marge d'erreur app 2 | - | $63 * 2^{-30}$ | $63 * 2^{-22}$ | $63 * 2^{-16}$ | $63 * 2^{-10}$ |
| Marge d'erreur app 3 | - | $31 * 2^{-30}$ | $31 * 2^{-22}$ | $31 * 2^{-16}$ | $31 * 2^{-10}$ |

**Tableau 6.3:** *Les résultats de la représentation des données.*

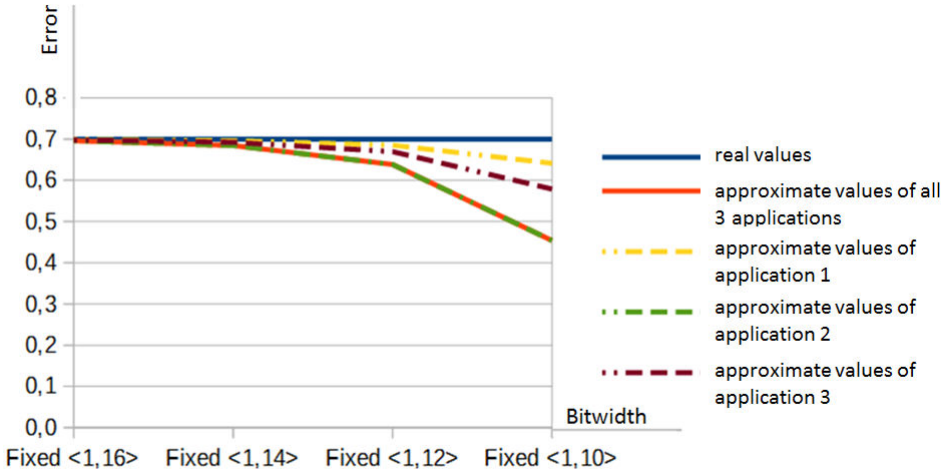



**Figure 6.8:** *La précision avec des représentations en virgule fixe [Zermani et al., 2015b].*

Dans la Figure 6.8, nous montrons un exemple pour le cas où la valeur de la représentation en virgule flottante est 0,7 et nous calculons les valeurs approximatives pour les 3 applications ($p = 6$, la profondeur du plus grand AC de l'Application 2, $p = 5$ pour l'Application 3 et $p = 4$ pour l'Application 1). On peut dire que la valeur appropriée de $n$ est égale à 12 pour cet exemple si la marge d'erreur est égale à 0,1. Si nous considérons chaque application séparément, $n$ est égal à 10 pour l'Application 1 et $n$ est égal à 11 pour l'Application 3, avec la même marge d'erreur.

### Analyse et synthèse

Ces résultats montrent qu'en fonction de la marge d'erreur que nous acceptons, nous pouvons déterminer la taille minimale en virgule fixe réduisant ainsi la consommation de ressources et augmentant les performances de l'implémentation matérielle.

5. Les implémentations SW/HW sur la ZedBoard :
   Nous passons maintenant à l'implémentation en-ligne de notre atelier logiciel sur la ZedBoard, afin de comparer les performances SW et HW. Avant d'exposer ces résultats, nous développons l'architecture et les interfaces utilisées pour ces expérimentations. Les interfaces rajoutées à l'architecture concernent la communication entre l'IP HLS et le CPU. Cette communication consiste à l'envoi des données du CPU à l'IP (les tables de probabilités, les tables d'évidence pour les réseaux Bayésiens, et aussi la table d'utilité et les actions pour les réseaux de décision). Pour cela nous proposons ici deux approches, une se basant sur une mémoire interne pour le stockage des tables de probabilités (lorsqu'elles sont fixes) et la deuxième sur une mémoire externe (lorsqu'elles sont variables). Par la suite, nous montrons avec l'approche en BRAM une solution intermédiaire, ou les tables peuvent être fixes avec un stockage en BRAM mais modifiables grâce à un contrôleur BRAM. Les Figures 6.9 et 6.10 représentent les deux architectures pour ces deux approches.

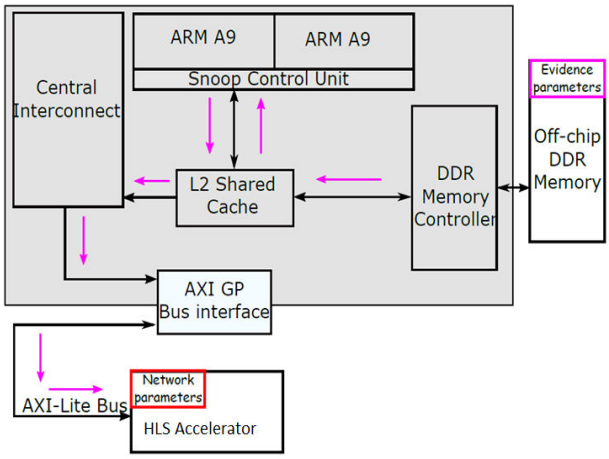



**Figure 6.9:** *Architecture en mémoire interne et connexion via AXI GP [Zermani et al., 2015a].*

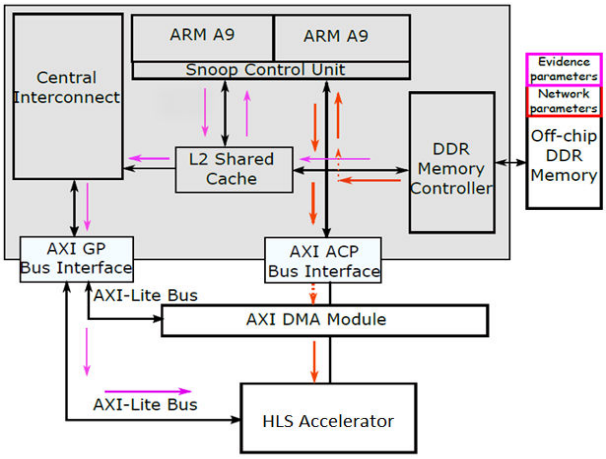



**Figure 6.10:** *Architecture en mémoire externe et connexion via AXI ACP [Zermani et al., 2015a].*

Lorsque nous supposons que la mémoire est interne pour les tables de probabilités, ainsi que les tables d'utilité et les actions pour les réseaux de décision (voir Figure 6.9), il reste juste l'envoi des tables d'évidence, qui lui est mis à jour à chaque calcul. Vu que les évidences sont concaténées par 32, nous les envoyons du CPU directement par un port AXI à usage général (GP). Les autres paramètres sont stockés en interne, qui veut dire dans la mémoire RAM embarquée ou dans les registres de la logique programmable. Lorsque nous supposons que la mémoire est externe (voir Figure 6.10), l'IP HLS est connecté au CPU par un bloc AXI DMA via un port AXI ACP, pour un envoi plus rapide et en flot. Pour les évidences, elles peuvent être envoyées comme pour la mémoire interne, mais aussi avec l'AXI DMA si leur nombre explose.

Une fois l'architecture validée et le bitstream généré, nous pouvons écrire le code C sur le SDK de Xilinx afin d'exécuter en SW et en HW nos 3 applications, et comparer les performances.

Le Tableau 6.4 représente les accélérations obtenues lors de la comparaison des implémentations SW et HW avec les deux architectures. On constate une bonne accélération dans les deux approches. L'accélération est meilleure dans le cas d'AXI GP car les paramètres sont stockés sur le FPGA. Dans l'AXI ACP, un temps de transfert des paramètres est ajouté au temps de calcul.

| | Accélération AXI GP Figure 6.9 | Accélération AXI ACP Figure 6.10 |
|---|---|---|
| Application 1 | 4.45 | 3.67 |
| Application 2 | 4.96 | 3.46 |
| Application 3 | 4.99 | 3.47 |

**Tableau 6.4:** *Les accélérations selon l'architecture.*

Avec :

$$\text{Accélération} = \frac{\text{Temps SW}}{\text{Temps HW}} \tag{6.3}$$

### Analyse et synthèse

Ces résultats montrent que nos deux architectures proposées pour l'implémentation matérielle, que ce soit avec une mémoire de stockage interne ou externe, offrent de bonnes accélérations. Ceci s'explique par le parallélisme qui est exploité dans les implémentations matérielles. L'architecture avec une mémoire de stockage interne donne une meilleure accélération car dans l'architecture avec une mémoire de stockage externe un temps de transfert des données est nécessaire

6. Généralisation pour un réseau Bayésien virtuel :

   Passons maintenant à la généralisation pour un réseau Bayésien virtuel, pour voir la capacité de notre atelier, en surface et performance pour différentes tailles de réseaux Bayésiens. Pour cela, nous considérons un AC virtuel avec différentes profondeurs (P), qui peut englober tous les types de réseau Bayésien. L'AC est un arbre avec une alternance de branches (+) et de branches (*). Pour simplifier, nous avons limité le nombre de fils des $BE0$ à 2. La profondeur est constituée du nombre d'étages (+) dans l'AC (nombre de BE1s séquentiels). Nous donnons des résultats pour les 3 points suivants :

— la généralisation de la comparaison JT vs AC,

— La généralisation de l'implémentation SW/HW avec les deux architectures (mémoire interne, mémoire externe),

— la généralisation de l'étude de la représentation des données en virgule fixe et en virgule flottante.

(a) La généralisation de la comparaison JT vs AC :
Le Tableau 6.5 représente la comparaison pour un cas général de l'algorithme JT et de l'algorithme AC. Nous confirmons que la version AC est plus efficace que la version JT. L'accélération augmente d'une manière exponentielle et on voit que le temps d'inférence avec AC est linéaire.

| | Temp JT ($ms$) | Temps AC ($ms$) | Accélération |
|---|---|---|---|
| P=2 (7 nodes, 26 $\theta$, 14 $\lambda$ ) | 13.0 | 12.1 | 1.07 |
| P=4 (31 nodes, 122 $\theta$, 62 $\lambda$) | 42.4 | 18.7 | 2.26 |
| P=6 (127 nodes, 506 $\theta$, 254 $\lambda$ ) | 162.5 | 24.4 | 6.65 |
| P=8 (511 nodes,2042 $\theta$, 1022 $\lambda$) | 627.3 | 31.6 | 19.85 |

**Tableau 6.5:** *Algorithme JT vs algorithme AC pour un AC virtuel.*

(b) La généralisation de l'implémentation SW/HW :
Nous calculons les accélérations obtenues par l'implémentation HW. Nous utilisons les deux architectures proposées dans la Figure 6.9 et la Figure 6.10. Nous limitons le nombre de ressources et nous supposons que l'architecture peut utiliser au maximum 12 blocs parallèles BE0 et BE1 pour tous les cas d'étude.
Les résultats en ressource et performance sont présentés dans le Tableau 6.6. Le nombre de DSP est petit pour P = 2 car 4 BE0 et 1 BE1 en parallèle sont suffisants. BE0 correspond à 3 DSP (en représentation virgule flottante) et BE1 correspond à 8 DSP (le nombre total de DSP sur le ZedBoard est égal à 220). Lorsque la profondeur P est supérieure à 4, la contrainte de ressource est appliquée au processus de synthèse, pour correspondre à la capacité du FPGA.
Nous observons une bonne accélération, qui croît avec le nombre de nœuds, justifiés par l'exploitation du parallélisme, dans le cas de la mémoire interne. Pour le cas de la mémoire externe, les résultats sont moins bons, et cela est dû au temps de transfert. Néanmoins, l'accélération augmente lorsque le réseau Bayésien est plus complexe (temps de calcul plus important que le temps de transfert) et que le parallélisme est mieux exploité.

| AC virtuel | Ressource | Accélération AXI GP | Accélération AXI ACP |
|---|---|---|---|
| P=2 | (20% DSP, 02% FF, 07% LUT) | 1.52 | 0.699 |
| P=4 | (60% DSP, 12% FF, 38% LUT) | 3.73 | 1.83 |
| P=6 | (60% DSP, 28% FF, 68% LUT) | 5.14 | 3.27 |
| P=8 | (60% DSP, 38% FF, 92% LUT) | 6.15 | 4.81 |

**Tableau 6.6:** *Ressource et performance pour un AC virtuel.*

(c) La généralisation de la représentation des données :
La Figure 6.11 représente la généralisation de la représentation des données en virgule fixe et en virgule flottante. De même que pour les 3 applications, les tables de probabilités et les calculs intermédiaires sont des valeurs inférieures ou égales à 1. Nous évaluons les réseaux Bayésiens virtuels de différentes profondeurs avec différentes largeurs de données et nous comparons avec la représentation en flottant en termes de ressources, performances et précisions.

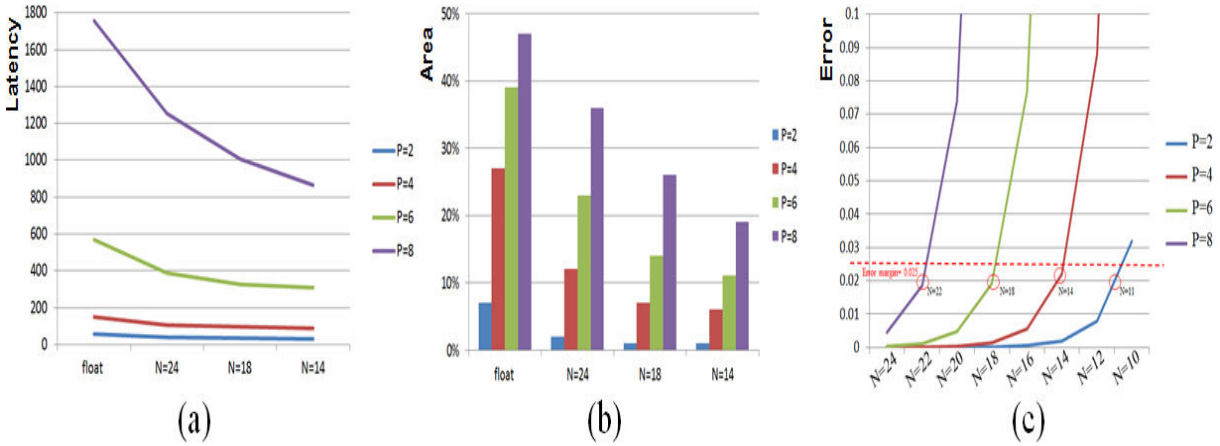



**Figure 6.11:** *La généralisation des expérimentations de la représentation des données [Zermani et al., 2015a].*

Dans la Figure 6.11.(a) et la Figure 6.11.(b), nous pouvons voir moins d'utilisation de ressources et nous obtenons de meilleures performances avec la représentation en virgule fixe. Ceci s'explique par la réduction de la taille des registres et le nombre de DSP utilisés pour les BE0 et BE1. Par exemple, lorsque $N = 24$, nous avons besoin de 2 DSP pour BE0 et 4 pour BE1 et lorsque N $\leq$ 18 nous avons besoin d'un seul DSP pour BE0 et de 2 pour BE1. Dans la Figure 6.11.(c), nous évaluons la marge d'erreur par l'expression : $(2^{2P-1} * (7/3) + 2^{P+2} - 2) * 2^{-N}$ (prouvé par récurrence). La valeur de la marge d'erreur augmente avec $N$ pour chaque $P$. Ces résultats nous permettent de choisir la largeur adéquate si

la marge d'erreur est fixée. Par exemple, pour $P = 4$, si la marge d'erreur est égale à 0,025, la largeur adéquate est 14 et elle est de 18 pour P = 6.

**Analyse et synthèse**

Ces résultats montrent l'intérêt d'une implémentation matérielle des réseaux Bayésiens avec notre approche pour des réseaux Bayésiens de différentes tailles et de profondeurs pour confirmer et compléter l'interprétation des résultats précédents. La première partie des résultats confirme que l'inférence par AC est plus performante que l'inférence par JT. Le temps d'inférence par AC augmente d'une manière linéaire par rapport à la taille du réseau Bayésien contrairement à l'inférence JT qui est exponentielle. La deuxième partie des résultats montre que l'implémentation matérielle offre une meilleure accélération dans le cas de la mémoire interne, qui croît avec le nombre de nœuds, justifiés par l'exploitation du parallélisme. Cependant, l'accélération croit avec le nombre de nœuds dans le cas de la mémoire externe, car le temps de calcul devient plus important que le temps de transfert. La troisième partie des résultats montrent que l'utilisation de la représentation en virgule fixe permet la réduction des ressources et que la valeur de la marge d'erreur augmente avec le nombre de nœuds et la profondeur.

### 6.3.2 Résultats pour la validation de l'approche Bayésienne pour l'état de santé et la décision

L'architecture avec l'AXI DMA offre une bonne accélération, mais nous avons déduit que le temps de communication est important, lorsque les données sont stockées en externe, et les données ne sont pas modifiables lorsqu'elles sont stockées en interne. Pour les réseaux Bayésiens à base de FMEA, nous proposons une architecture où les données sont stockées en BRAM et peuvent être modifiables grâce à un contrôleur BRAM (AXI BRAM Controller), qui gère le transfert de données entre les blocs BRAM et la mémoire DDR. Cette architecture permet une transmission rapide des données. Notez que les tables de probabilités peuvent être envoyées dès qu'elles sont fixées et qu'elles peuvent être modifiées en ligne via le contrôleur AXI BRAM. Mais en pratique, le taux de mise à jour est faible et les tables de probabilités peuvent être considérées statiques au cours d'une mission. Les indicateurs d'évidence (valeurs booléennes) sont fournis par les capteurs et sont mis à jour pour chaque calcul à la vitesse du capteur. Pour minimiser le temps et optimiser les ressources, ils sont concaténés par 32 et envoyés via l'AXI BRAM Controller. Les résultats de l'état de santé ou de la décision sont envoyés via un bus AXI lite. Si de nombreux résultats doivent être envoyés, nous pouvons également utiliser une BRAM pour la sortie. La Figure 6.12 représente cette architecture pour l'implémentation HW/ SW sur la ZedBoard des expérimentations.

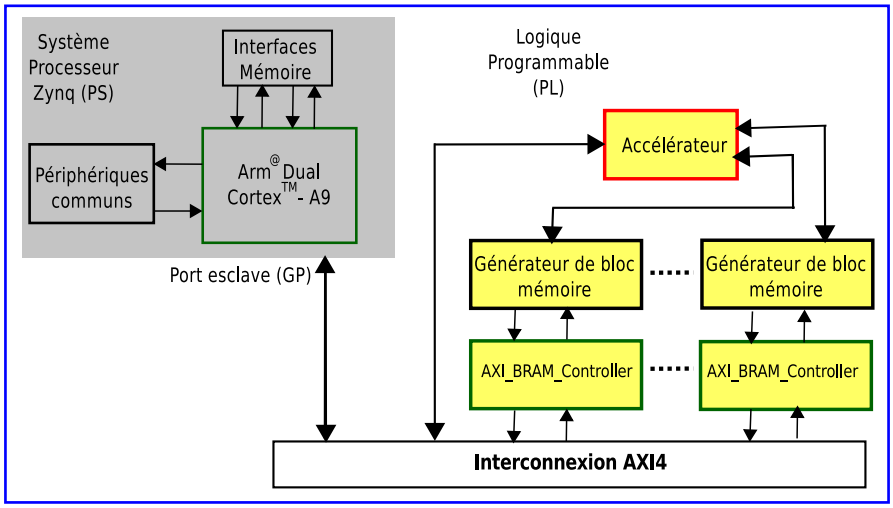



**Figure 6.12:** *L'architecture pour les implémentations à base de l'analyse FMEA [Zermani et al., 2017b].*

1. Mission de navigation :

   L'exemple de scénarios de mission pour illustrer les résultats de l'étude FMEA est la mission de navigation utilisant les deux réseaux Bayésiens du GPS et de l'énergie et la décision (décrite en 5.3.2).

   Pour cette mission, nous commençons par l'évaluation de l'implémentation HW/SW en ressource, performance et consommation énergétique des réseaux Bayésiens de l'état de santé du GPS et de l'énergie. Pour cela, nous exposons les résultats avec les différentes options de parallélisme de données et le partage de ressources. Par la suite, nous évaluons l'implémentation HW/SW de la décision de la mission en utilisant les implémentations des états de santé.

   Le Tableau 6.7 montre l'évaluation, hors ligne, de la latence et des ressources pour les implémentations HW de l'état de santé du GPS, l'énergie, ainsi que le réseau global incluant les deux réseaux en parallèle. Les résultats sont donnés pour les solutions matérielles avec différents degrés de parallélisme des données et en utilisant la directive *Array_Partition*.

   Les résultats montrent que l'impact du parallélisme des données est plus important dans le cas du GPS. Nous pouvons expliquer cela par une faible dépendance conditionnelle entre les nœuds sur le réseau du GPS, contrairement à l'énergie. La différence entre les deux réseaux est que le réseau du GPS a besoin de données différentes pour un calcul parallèle, et le réseau de l'énergie a besoin de mêmes données pour différents calculs, ce qui explique la grande quantité de ressources dans ce cas et une plus grande latence. Pour le réseau global GPS + Energie, nous observons que les ressources sont additionnées et la latence est égale à la plus grande latence qui est celle de l'énergie. L'explication est que les deux réseaux fonctionnent indépendamment et en parallèle.

| | | BRAM | DSP | LUT | FF | Latence |
|---|---|---|---|---|---|---|
| Array_Partition_1 | GPS | 3% | 10% | 16% | 08% | 145 |
| | Energie | 3% | 29% | 26% | 12% | 164 |
| | GPS+Energie | 4% | 39% | 37% | 18% | 166 |
| Array_Partition_4 | GPS | 10% | 15% | 19% | 10% | 113 |
| | Energie | 9% | 29% | 28% | 13% | 144 |
| | GPS+Energie | 13% | 44% | 42% | 20% | 146 |
| Array_Partition_8 | GPS | 15% | 29% | 24% | 12% | 106 |
| | Energie | 13% | 29% | 30% | 14% | 141 |
| | GPS+Energie | 24% | 58% | 51% | 25% | 143 |

**Tableau 6.7:** *Ressource et latence pour l'implémentation HW du cas GPS et Energie.*

Une partie importante de la surface est utilisée pour une meilleure latence avec un partitionnement différent des données, mais nous pouvons la réduire à l'aide de la directive Inline, permettant de partager les ressources entre les deux réseaux, ou la directive Allocation permettant de limiter les ressources d'allocation pour réduire les instances de calcul (multiplications et additions). Un exemple est donné dans la Figure 6.13 pour montrer le compromis ressource/ latence dans le cas du GPS + Energie (en utilisant *Array_Partition_1*). Nous pouvons déduire que la directive Inline optimise la surface utilisée grâce au partage des ressources mais avec une augmentation de la latence. Cependant, diviser les ressources en deux ne change pas la latence. Ceci est dû à la forme de l'arbre du modèle et à l'alternance des additions et des multiplications.

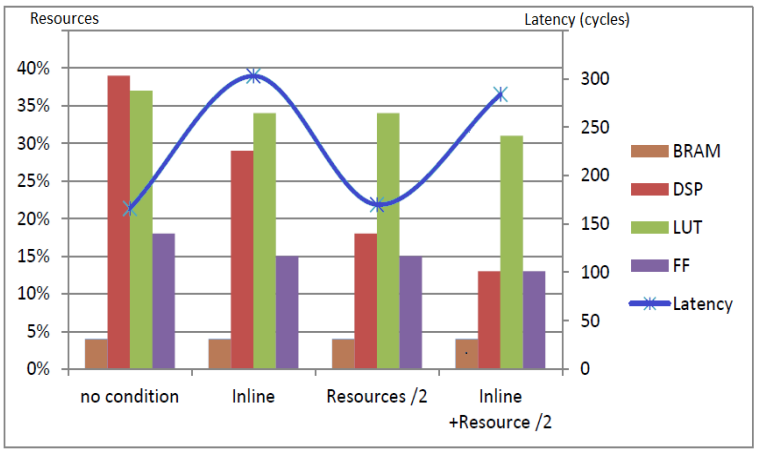



**Figure 6.13:** *Ressource et latence pour l'implémentation HW du cas GPS + Energie, en appliquant les directives [Zermani et al., 2017b].*

Le Tableau 6.8 montre les performances en ligne pour l'implémentation HW/SW, sur le réseau du GPS , le réseau de l'Energie et le réseau global avec le GPS

et l'Energie. Nous évaluons pour chaque cas de partitionnement : le temps d'exécution global HW et SW, le temps d'exécution HW sans le temps de transfert des tables de probabilités (ils peuvent être envoyés une seule fois), l'accélération HW et la consommation d'énergie sur la carte.

| | | Temps HW/SW | Temps HW2 | Acc1 | Acc2 | Conso. d'énergie ($\mu$ J) HW/SW |
|---|---|---|---|---|---|---|
| Array_Partition_1 | GPS | 325/1054 | 215 | 3.24 | 4.90 | 5.89/17.97 |
| | Energie | 334/1301 | 224 | 3.89 | 5.80 | 6.69/22.18 |
| | GPS+Energie | 407/2355 | 298 | 5.78 | 7.90 | 8.99/40.15 |
| Array_Partition_4 | GPS | 296/1054 | 186 | 3.56 | 5.66 | 5.74/17.97 |
| | Energie | 314/1301 | 204 | 4.14 | 6.37 | 6.10/17.97 |
| | GPS+Energie | 387/2355 | 277 | 6.08 | 8.50 | 8.80/40.15 |
| Array_Partition_8 | GPS | 279/1054 | 169 | 3.77 | 6.23 | 5.63/17.97 |
| | Energie | 311/1301 | 201 | 4.18 | 6.47 | 6.13/17.97 |
| | GPS+Energie | 384/2355 | 274 | 6.13 | 8.59 | 9.10/40.15 |

**Tableau 6.8:** *Performance HW et SW des réseaux GPS et Energie (où Acc1= Temps SW/ Temps HW, Acc2= Temps SW/ Temps HW sans les temps de transfert des tables de probabilités (HW2)).*

Nous observons une bonne accélération HW dans tous les cas car l'implémentation SW est séquentielle. Une meilleure accélération est observée dans le cas de l'Energie qui est due à la complexité de calcul provenant du réseau Bayésien dynamique. Le modèle global a une bonne accélération parce que les deux réseaux HW sont en parallèle. Nous remarquons également que l'accélération augmente avec le partitionnement ce qui s'explique par l'utilisation de plus de ressources et donc plus de parallélisme.

La consommation d'énergie sur la carte est plus faible dans le cas des implémentations HW. Par exemple, la puissance dans le cas des implémentations HW avec $Array\_Partition\_1$ est égale à $1.81W$ pour le GPS et $2.01W$ pour l'Energie par rapport à une puissance de $1.70W$ pour les implémentations $SW$. Néanmoins, la consommation d'énergie est inférieure dans n'importe quelle version HW en raison du temps d'exécution réduit des versions HW. La réduction de la consommation d'énergie, fournie par la version HW, est montrée dans la dernière colonne du Tableau 6.8.

Nous intégrons maintenant la décision, où nous pouvons avoir une implémentation tout en SW (réseau Bayésien + décision), tout en HW, ou en mixte des deux (réseau Bayésien en HW et décision en SW). Nous généralisons cette approche par la suite car le choix des implémentations dépend du nombre de réseaux Bayésiens et du nombre d'actions de la décision.

Le Tableau 6.9 montre les ressources HW utilisées pour la version tout en HW et la version mixte, ainsi que les performances en ligne, l'accélération obtenue par rapport à la version tout en SW, et la consommation énergétique (avec $Array\_Partition\_4$). Le réseau de cette mission inclut deux réseaux Bayésiens

(GPS, Energie) et 3 actions (Ne rien changer, Passer à une autre méthode de localisation, Atterrir d'urgence). Pour ce cas, nous avons un meilleur temps d'exécution pour la version tout en HW et avec plus de ressources, que la version mixte mais les deux accélérations restent bonnes. Cela signifie que pour cette mission, mettre les réseaux Bayésiens en HW est plus important que la décision. Pour la décision, selon le besoin en performance, nous pouvons choisir une implémentation HW ou SW.

| | Ressource | | | | Temps | Accélération | Conso. d'énergie |
|---|---|---|---|---|---|---|---|
| | BRAM | DSP | LUT | FF | (cycles) | | ($\mu$ J) |
| Tout en HW | 14% | 50% | 34 % | 21% | 419 | 6.28 | 9.81 |
| Mixte | 13% | 44% | 42 % | 20% | 665 | 3.96 | 15.11 |

**Tableau 6.9:** *Ressource et performance de l'implémentation de la décision.*

**Analyse et synthèse**

Ces résultats montrent dans un premier temps l'impact des directives de synthèses sur la latence, les ressources et consommation d'énergie des états de santé. Nous pouvons conclure que l'intérêt du partitionnement sur les performances est plus important lorsque les dépendances conditionnelles sont faibles. Cela permet d'explorer le parallélisme de calcul. L'utilisation du partitionnement à grand niveau donne une meilleure latence mais augmente la surface. Cela peut être géré par des directives pour limiter ou partager les ressources. La consommation d'énergie en HW reste toujours inférieure à la version SW en raison du temps d'exécution réduit. Ces résultats montrent dans un deuxième temps l'intérêt d'une implémentation HW de la décision. Avec 3 actions, une bonne accélération est obtenue que ce soit en SW (état de santé en HW) ou en HW. Nous concluons que la décision, dans ce cas là, peut être implémentée en SW ou en HW car le temps de calcul de la décision est moins important que celui de l'état de santé et le choix peut être fait selon le besoin et les contraintes.

2. Généralisation pour des réseaux Bayésiens de type FMEA générique et la décision :

   Nous généralisons l'approche à partir des modèles de mission pour des réseaux Bayésiens de type FMEA en présentant des expérimentations en trois étapes :

   — La variation du nombre de types d'erreur d'un réseau Bayésien.

   — La variation du nombre de réseaux Bayésiens et un nombre fixe de types d'erreur.

   — L'étude de la décision en variant le nombre de réseaux Bayésiens et le nombre d'actions.

(a) La variation du nombre de types d'erreur d'un réseau Bayésien :

La Figure 6.14 montre les ressources utilisées et la latence pour différents nombres de types d'erreur d'un réseau Bayésien de type FMEA, et la Figure 6.15 donne pour les mêmes réseaux Bayésiens, le temps d'exécution HW et SW, l'accélération et l'énergie consommées.

De la Figure 6.14, nous pouvons déduire que le pourcentage des ressources utilisées augmente avec le nombre de types d'erreur, ce qui signifie plus de calcul en parallèle, parce que les sous-réseaux des types d'erreur sont indépendants. La latence croît aussi avec le nombre de types d'erreur, ce qui est dû à la lecture des données, principalement avec *Array_Partition_1* . En utilisant *Array_Partition_8* , nous réduisons la latence car la lecture des données peut être effectuée en parallèle, et nous réduisons également l'écart de latence entre les réseaux.

De la Figure 6.15, nous confirmons que nous pouvons avoir une bonne accélération avec l'implémentation HW, qui croît avec le nombre de types d'erreurs, en raison du parallélisme du calcul. A partir du parallélisme des données, les écarts entre les temps HW des différents réseaux diminuent et les écarts d'accélération, par conséquent, augmentent. De la même façon, la consommation d'énergie HW croît avec le nombre de types d'erreur, mais l'écart avec la version SW diminue avec la directive *Array_Partition_8* en raison de la réduction significative du temps d'exécution, même si la consommation d'énergie est plus grande.

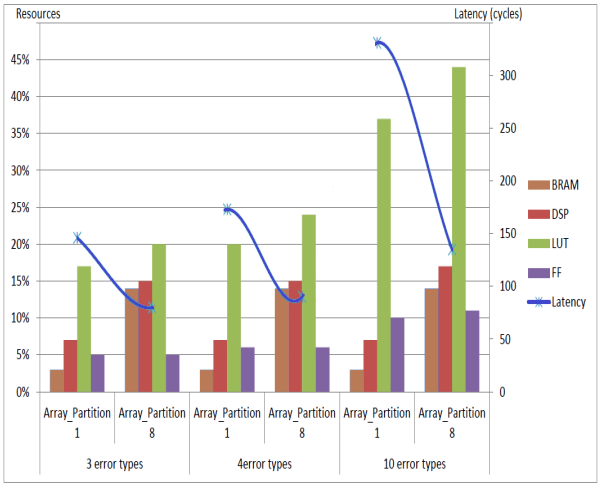



**Figure 6.14:** *Ressource HW et latence d'un réseau Bayésien de type FMEA en variant le nombre de types d'erreur.*

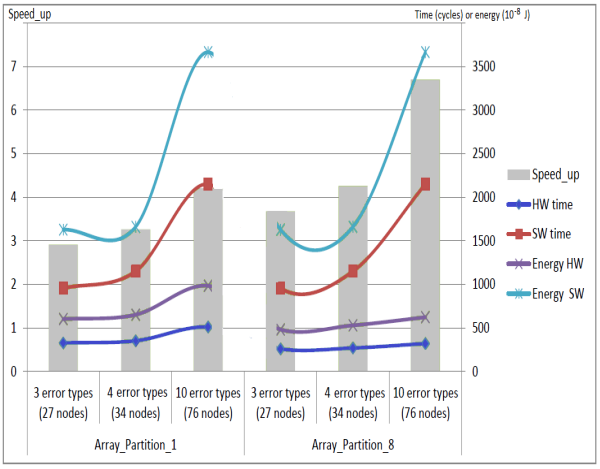



**Figure 6.15:** *Performance HW et SW d'un réseau Bayésien de type FMEA en variant le nombre de types d'erreur.*

A partir de ces expériences, nous pouvons conclure que l'exploitation du parallélisme croît avec le nombre de types d'erreurs et d'avantage quand les données sont accessibles en parallèle. Cela donne un meilleur temps HW et une meilleure consommation d'énergie. Par exemple, pour un réseau Bayésien avec 3 types d'erreur, la puissance consommée est de 1.84W et de 1.87W (*Array_Partition*_1 et *Array_Partition*_8, respectivement) et pour un réseau Bayésien avec 10 types d'erreur, la puissance consommée est de 1.93W et 1.95W par rapport à 1.70W pour la version SW. Néanmoins, la consommation d'énergie des versions HW est toujours inférieure à la consommation d'énergie des versions SW.

(b) La variation du nombre de réseaux Bayésiens :

La Figure 6.16 et la Figure 6.17 résument les résultats avec la variation du nombre de réseaux Bayésiens en utilisant un partitionnement de données moyen et tous les réseaux Bayésiens avec 3 types d'erreur. Pour ces expérimentations, nous proposons deux stratégies : la première est que tous les réseaux Bayésiens sont calculés séparément et en parallèle (avec des limitations de ressources si nécessaire), et la seconde est qu'un seul module est utilisé pour tous les réseaux Bayésiens, et les calculs sont en pipeline. La Figure 6.16 montre les ressources utilisées et la latence pour les différents cas, et la Figure 6.17 donne pour les mêmes cas le temps d'exécution HW et SW, l'accélération et la consommation d'énergie.

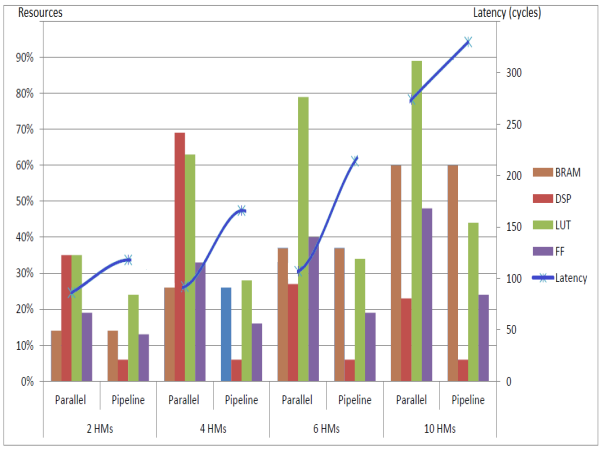



**Figure 6.16:** *Ressource HW et latence pour un ensemble de réseaux Bayésiens*

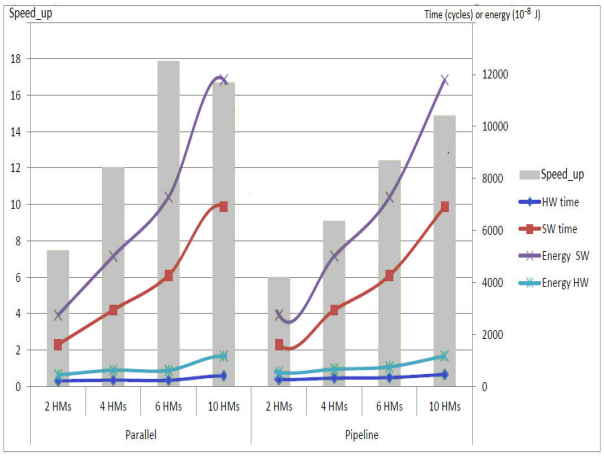



**Figure 6.17:** *Performance HW et SW pour un ensemble de réseaux Bayésiens.*

De la Figure 6.16, nous pouvons déduire que l'utilisation des ressources croît avec le nombre de réseaux Bayésiens dans le cas de la stratégie parallèle, car chaque réseau Bayésien se calcule indépendamment et il n'y a pas de partage de ressources. Notez qu'il est nécessaire de limiter les

ressources en cas de 6 et 10 réseaux Bayésiens, sinon nous aurons un dépassement de capacité du FPGA. Dans le cas de la stratégie pipeline, l'utilisation des ressources croît lentement, à cause de l'utilisation des mêmes ressources pour le calcul. Dans le cas de la stratégie parallèle, la latence croît aussi avec le nombre de réseaux Bayésiens, ce qui est dû à la lecture des données, mais l'écart de latence entre 2 réseaux Bayésiens, 4 réseaux Bayésiens et 6 réseaux Bayésiens est faible. Par contre, la latence de 10 réseaux Bayésiens est plus importante, en raison d'une grande limitation des ressources. Dans le cas de la stratégie pipeline, la latence croît aussi, avec le nombre de réseaux Bayésiens, puisque l'intervalle de calcul augmente avec le nombre de réseaux Bayésiens.

De la Figure 6.17, nous confirmons que nous pouvons avoir une bonne accélération avec l'implémentation HW. Dans le cas de la stratégie parallèle, nous observons que l'accélération est moins importante pour 10 HM en raison de la grande limitation des ressources. L'accélération et l'énergie consommée sont meilleures dans le cas de la stratégie parallèle en raison de l'intervalle de calcul entre deux HM dans la stratégie pipeline.

(c) L'étude de la décision en variant le nombre de réseaux Bayésiens et le nombre d'actions :

Le Tableau 6.10 résume les résultats de la décision avec la variation du nombre de réseaux Bayésiens et d'actions, en utilisant la stratégie pipeline pour les réseaux Bayésiens avec un partitionnement de données moyen et tous les réseaux Bayésiens avec 3 types d'erreur. Pour ces expérimentations, nous proposons une implémentation tout en HW et une implémentation mixte (réseaux Bayésiens en HW et décision en SW) dans le but d'évaluer la décision.

| | | | Ressource | | | | Temps (cycles) | Accélération | Conso. ($\mu$ J) |
|---|---|---|---|---|---|---|---|---|---|
| | | | BRAM | DSP | LUT | FF | | | |
| 2 Réseaux Bayésiens | 2 Actions | Tout en HW | 14% | 14% | 23% | 12% | 310 | 5.96 | 6.33 |
| | | Mixte | 14% | 6% | 20% | 11% | 528 | 3.50 | 10.50 |
| | 10 Actions | Tout en HW | 14% | 24% | 32% | 16% | 355 | 7.33 | 7.56 |
| | | Mixte | 14% | 6% | 20% | 11% | 1328 | 1.96 | 26.42 |
| 10 Réseaux Bayésiens | 2 Actions | Toute en HW | 61% | 41% | 78% | 34% | 6001 | 11.12 | 170.42 |
| | | Mixte | 60% | 6% | 44% | 24% | 60370 | 1.10 | 1521.32 |
| | 10 Actions | Tout en HW | 80% | 60% | 80% | 40% | 11025 | 21.3 | 330.84 |
| | | Mixte | 60% | 6% | 44% | 23% | 232475 | 1.02 | 5835.12 |

**Tableau 6.10:** *Ressource et performance de l'implémentation de la décision en variant le nombre de réseaux Bayésiens et d'actions.*

Nous pouvons constater que dans tous les cas l'implémentation tout en HW est plus performante avec une bonne accélération. Lorsque le nombre de réseaux Bayésiens est petit, une implémentation mixte peut être envisageable lorsque le nombre d'actions est également petit. Nous pouvons

aussi constater que le temps HW évolue d'une manière linéaire en nombre d'actions et d'une manière exponentielle en nombre de réseaux Bayésiens, ce qui est dû à la taille de la table d'utilité qui est exponentielle au nombre de réseaux Bayésiens et des multiplications de toutes les valeurs de cette table par les probabilités des réseaux Bayésiens dans le calcul de la fonction d'utilité. Les ressources augmentent, également, avec le nombre de réseaux Bayésiens et d'actions, ce qui peut être expliqué par le parallélisme dans le calcul des fonctions d'utilité des différentes actions, ainsi que les produits des multiplications des probabilités des réseaux Bayésiens et de la table d'utilité.

**Analyse et synthèse**

Ces résultats montrent dans un premier temps l'impact en performance, ressources et consommation d'énergie d'un réseau générique de type FMEA en variant le nombre de types d'erreur. Nous pouvons conclure que l'accélération HW et les ressources augmentent avec le nombre de types d'erreur, car chaque type d'erreur représente un sous-réseau Bayésien indépendant. Ces résultats montrent dans un deuxième temps l'impact en performance, ressources et consommation d'énergie en variant le nombre de réseaux Bayésiens de type FMEA, avec deux stratégies : en parallèle ou en pipeline. Nous pouvons conclure que dans le cas d'un petit nombre de réseaux Bayésiens la stratégie parallèle est la plus adéquate mais pour un grand nombre de réseaux Bayésiens, il est préférable d'utiliser la stratégie pipeline. Ces résultats montrent dans un troisième temps l'impact en performance, ressources et consommation d'énergie en faisant varier le nombre de réseaux Bayésien de type FMEA et d'actions. Nous pouvons conclure que dans tous les cas une implémentation HW est la plus performante mais peut utiliser beaucoup de ressources. L'implémentation mixte (état de santé en HW et Décision en SW) est intéressante lorsque le nombre d'actions est petit.

Le choix de l'implémentation est surtout fait selon le besoin en performance consommation d'énergie et les ressources disponibles.

## 6.4 Synthèse et conclusion

Dans ce chapitre, nous avons présenté nos différents résultats et expérimentations sur un SoC hybride (la carte Zedboard de Xilinx), où nous avons détaillé les architectures utilisées pour les implémentations HW/SW et les interfaces de communication.

Ces résultats donnent une idée, à concepteur non expert, de comment obtenir le meilleur compromis ressource/performance/consommation énergétique compte tenu

des paramètres du réseau.

Pour cela nous avons commencé par montrer l'intérêt de notre approche Bayésienne en utilisant 3 exemples applicatifs. Nous avons exposé nos résultats et analyses pour montrer l'intérêt de l'utilisation de l'inférence AC, l'apport de la décomposition et des optimisations que nous avons proposées, et le grain en utilisent une représentation des données en virgule fixe. Nous avons aussi exposé notre implémentation sur la carte Zedboard, et nous avons montré l'intérêt des implémentations HW/SW en termes de ressources, performances et consommation d'énergie. Nous avons fini cette première partie par une généralisation de cette étude à des réseaux Bayésiens de différentes tailles et profondeurs.

Nous avons par la suite montré les expérimentations de l'implémentation sur SoC de l'approche Bayésienne de l'état de santé et de la décision. Nous avons exposé les résultats pour un exemple de mission, en variant les différentes directives de synthèse pour avoir un compromis entre les ressources et les performances. Par la suite nous avons généralisé l'implémentation afin de pouvoir évaluer les limites. Pour cela nous avons d'abord fait varier le nombre de types d'erreur dans un réseau Bayésien de type FMEA. Nous avons par la suite fait varier le nombre de réseaux Bayésiens à implémenter en parallèle ou en pipeline. Et finalement nous avons fait varier pour la décision le nombre de réseaux Bayésiens et le nombre d'actions.

Dans le chapitre suivant nous rappelons les problématiques et les contributions, et nous concluons avec les perspectives.

# - Chapitre 7 -

# Conclusion générale

## 7.1 Conclusion

Avec cette thèse nous avons atteint principalement deux objectifs. Le premier consiste à développer un atelier logiciel pour l'implémentation matérielle et logicielle des réseaux Bayésiens sur SoC, non proposé auparavant. Et le deuxième consiste à proposer un nouveau modèle Bayésien pour l'état de santé et la décision pour les missions des véhicules autonomes embarqués. Ce travail a abouti à trois contributions que nous avons détaillées tout au long de ce mémoire.

Notre atelier logiciel pour l'implémentation matérielle et logicielle des réseaux Bayésiens permet à un non expert dans les systèmes embarqués de pouvoir implémenter son modèle sur un SoC hybride en HW ou en SW selon le besoin en ressource, performance, et consommation énergétique.

Dans notre atelier logiciel, nous partons d'une représentation graphique ou textuelle du réseau Bayésien, et nous proposons un algorithme efficace et optimal pour l'implémentation embarquée et parallèle de l'inférence Bayésienne. Cet algorithme est par la suite intégré aux outils de synthèse de haut niveau (HLS) pour adapter la solution selon le besoin en définissant les directives de synthèse nécessaires, les interfaces de communication entre le HW et le SW, ainsi que les ressources pour le stockage et l'envoi des paramètres. Le HLS permet aussi de générer le HDL qui sera mappé sur la partie FPGA. Par la suite, le déploiement sur l'architecture du SoC se fait en utilisant Vivado Design Suite, puis l'exécution sur un SoC et le test en ligne peut être réalisé avec le SDK de Vivado.

Pour la validation de notre atelier logiciel, nous avons utilisé la carte ZedBoard de Xilinx, incorporant un processeur ARM Cortex-A9 et une partie FPGA, communiquant entre eux via des bus AXI (Advanced eXtensible Interface).

Notre modèle de l'état de santé et de la décision se base sur les réseaux Bayésiens. Ce sont des modèles graphiques probabilistes largement utilisés dans diagnostic complexe, et les plus adaptés pour faire face à l'incertitude.

Ce modèle a pour objectif de localiser une défaillance, qui peut-être dû à des changements environnementaux, des problèmes matériels, etc. et d'être capable de choisir une action de recouvrement la plus adéquate en plein mission, contrairement aux solutions classiques, où les informations sont transférées au sol pour le dépannage.

Nous avons proposé un modèle générique dédié aux missions de véhicules autonomes qui s'inspire d'une analyse de type FMEA (Analyse des Modes de Défaillances et de leurs Effets), prenant en compte les différentes observations sur des capteurs et des contextes d'apparition.

Pour valider notre modèle, nous avons intégré le réseau Bayésien de l'état de santé et de la décision à un cas d'étude d'une mission de drone. La mission prise en compte est une mission de recherche et de sauvetage de type " Save Outback Joe ".

Ces travaux nous ont permis de conclure essentiellement que :

— Les réseaux Bayésiens sont des modèles efficaces pour l'état de santé et la décision à condition de bien définir le modèle et les différents paramètres associés. Nous proposons un front end qui permet de faciliter leur utilisation.

— Les implémentations matérielles des réseaux Bayésiens peuvent nous permettre d'atteindre une bonne accélération de calcul avec une consommation énergétique réduite, et cela dépend principalement de la taille du réseau Bayésien et des dépendances entre les nœuds, ainsi que la bonne gestion des interfaces et des ressources.

## 7.2 Perspectives

Ces travaux de thèse ouvrent la voie à des perspectives à différents niveaux. Nous donnons quelques pistes d'amélioration à court et moyen terme au niveau de l'atelier logiciel, au niveau du modèle Bayésien, et au niveau applicatif.

1. Niveau atelier logiciel, des améliorations sont possibles sur les parties suivantes :
   — Sur la partie spécification du réseau Bayésien et la compilation en AC : nous avons proposé des générations automatiques de l'AC avec des variables à deux états et nous avons généralisé l'approche pour plusieurs états qui reste à intégrer dans le logiciel.
   — Sur la partie cible : nous avons choisi comme cible un SoC-FPGA mais notre outil peut être applicable sur des multi-cibles (GPU, multi-cœurs) en utilisant de l'openCL à la place du C en front end des outils de synthèse.

   - Sur la partie modèle : il y a une possibilité d'étendre l'atelier à d'autres modèles probabilistes pour la prise de décision comme le MDP (Processus de Décision Markovien).
2. Niveau modèle Bayésien, des améliorations sont possibles sur les parties suivantes :
   - Sur la partie FMEA : il est possible de rajouter des nœuds "Solutions" permettant de voir la solution qui augmente la probabilité de l'état de santé globale dans le cas d'un problème.
   - Sur la partie variable : nous avons utilisé des variables discrètes dans notre approche mais il est possible d'utiliser des variables continues ou un mix des deux.
   - Sur la partie inférence Bayésienne : nous avons opté pour une solution exacte du calcul d'inférence mais il reste possible d'utiliser des algorithmes de calcul approché.
   - Sur la partie apprentissage : le modèle proposé peut aussi être amélioré en intégrant l'apprentissage de paramètres, qui permettrait d'affiner les tables de probabilités. Cette partie exige des tests réels, des informations d'expert ou des simulations.
3. Niveau applicatif, des améliorations sont possibles sur les parties suivantes :
   - Sur la partie application : la proposition a été développée et testée théoriquement pour un cas d'étude traitant des scénarios de défaillance lors d'une mission de drone. Cela peut être étendu à d'autres cas de systèmes autonomes et aussi testé réellement sur un drone par exemple.
   - Sur la partie reconfiguration de la mission : nous avons pensé à l'utilisation des FPGA pour leurs avantages en consommation et en accélération de calcul mais aussi pour la possibilité de la reconfiguration dynamique totale ou partielle. En effet, lors d'un changement de plan de mission ou à la fin d'une étape de mission, la partie FPGA dédié au réseau Bayésien peut être réutilisée pour le réseau Bayésien de la nouvelle mission. Par rapport à la structure du modèle générique, nous pouvons imaginer plusieurs façons de minimiser la reconfiguration.
   - Sur la partie génération automatique et en ligne des réseaux Bayésiens pour l'état de santé : Les réseaux Bayésiens de l'état de santé ont une structure régulière avec un calcul automatisé, donc avec une intelligence artificielle, l'analyse de type FMEA peut être mise à jour en ligne pour intégrer des dysfonctionnements non prévus et générer les réseaux Bayésiens correspondants au cours de la mission.

Parmi ces propositions, la partie apprentissage des paramètres est la tâche la plus critique car avoir une bonne précision des probabilités de défaillance permet de modéliser de manière plus adéquate les aléas d'une mission et de produire des scénarios de mission plus efficaces.

# Liste des figures

# Liste des tableaux

# Bibliographie


Handle-C. https://www.mentor.com/products/fpga/handel-c/. Accessed : 2016-09-30. 47

Samiam : Sensitivity analysis, modeling, inference and more. http://reasoning.cs.ucla.edu/samiam/. Accessed : 2016-09-30. 57

Vivado-HLS. https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html/. Accessed : 2016-09-30. 47

Xilinx. https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf, a. Accessed : 2016-09-30. 49

Xilinx. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug902-vivado-high-level-synthesis.pdf, b. Accessed : 2016-09-30. 53

Xilinx. https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf, c. Accessed : 2016-09-30. 52

Zedboard. http://zedboard.org/product/zedboard. Accessed : 2016-09-30. 102

Catapult. https://www.mentor.com/hls-lp/catapult-high-level-synthesis/. Accessed : 2016-09-30. 47

A. Antoniou. *Digital Signal Processing*. Mcgraw-Hill, 2nd edition, 2016. ISBN 0071846034, 9780071846035. 41

AIAG. Automotive Industry Action Group. *Potential Failure Mode and Effects Analysis (FMEA) : Reference Manual*. Chrysler Corporation, 1993. URL https://books.google.fr/books?id=XhDVPwAACAAJ. 36, 79

F.R. Bach and M.I Jordan. Discriminative training of hidden markov models for multiple pitch tracking [speech processing examples]. In *2005 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '05, Philadelphia, Pennsylvania, USA, March 18-23, 2005*, pages 489–492, 2005. doi : 10.1109/ICASSP.2005.1416347. URL https://doi.org/10.1109/ICASSP.2005.1416347. 31

D.F. Bacon, S.L. Graham, and O.J. Sharp. Compiler transformations for high-performance computing. *ACM Comput. Surv.*, 26(4) :345–420, December 1994. ISSN 0360-0300. doi : 10.1145/197405.197406. URL http://doi.acm.org/10.1145/197405.197406. 46

S. Bakshi, D. D. Gajski, and H. Juan. Component selection in resource shared and pipelined dsp applications. In *Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European*, pages 370–375, Sep 1996. doi : 10.1109/EURDAC.1996.558231. 47

V. Betz, J. Rose, and A. Marquardt, editors. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA, USA, 1999. ISBN 0792384601. 42

S. Bottone, D. Lee, M. O'Sullivan, and M. Spivack. Failure prediction and diagnosis for satellite monitoring systems using Bayesian networks. In *Military Communications Conference. MILCOM 2008. IEEE*, pages 1–7, Nov 2008. doi : 10.1109/MILCOM.2008.4753131. 34

B.Y. Chan and R.D. Shachter. Structural controllability and observability in influence diagrams. In *Proceedings of the Eighth International Conference on Uncertainty in Artificial Intelligence*, UAI'92, pages 25–32, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. ISBN 1-55860-258-5. URL http://dl.acm.org/citation.cfm?id=2074540.2074544. 32

M. Chavira and A. Darwiche. Compiling Bayesian networks with local structure. In *IJCAI*, 2005. 27

M. Chavira and A. Darwiche. Compiling Bayesian networks using variable elimination. In *IJCAI*, 2007. 27

D. Codetta-Raiteri and L. Portinale. Dynamic bayesian networks for fault detection, identification, and recovery in autonomous spacecraft. *Systems, Man, and Cybernetics : Systems, IEEE Transactions on*, 45(1) :13–24, 2015. 34, 37

J. Cong and J. Xu. Simultaneous fu and register binding based on network flow method. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, pages 1057–1062, New York, NY, USA, 2008. ACM. ISBN

978-3-9810801-3-1. doi : 10.1145/1403375.1403629. URL http://doi.acm.org/10.1145/1403375.1403629. 47

P. Coussy and A. Morawiec. *High-Level Synthesis : From Algorithm to Digital Circuit*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 1402085877, 9781402085871. 46

P. Coussy, D. D. Gajski, M. Meredith, and A. Takach. An introduction to high-level synthesis. *IEEE Design Test of Computers*, 26(4) :8–17, July 2009. ISSN 0740-7475. doi : 10.1109/MDT.2009.69. 47

L.H. Crockett, R.A. Elliot, M.A. Enderwitz, and R.W. Stewart. *The Zynq Book : Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014. 4, 102, 132

A. Darwiche. Recursive conditioning. *Artif. Intell.*, 126(1-2) :5–41, February 2001a. ISSN 0004-3702. doi : 10.1016/S0004-3702(00)00069-2. URL http://dx.doi.org/10.1016/S0004-3702(00)00069-2. 10

A. Darwiche. Recursive conditioning. *Artif. Intell.*, 126(1-2) :5–41, February 2001b. ISSN 0004-3702. doi : 10.1016/S0004-3702(00)00069-2. URL http://dx.doi.org/10.1016/S0004-3702(00)00069-2. 21

A. Darwiche. A differential approach to inference in Bayesian networks. *J. ACM*, 50(3) :280–305, 2003. 10, 27, 30

A. Darwiche. Bayesian networks. *Communications of the ACM*, 53(12) :80–90, 2010. ISSN 0001-0782. doi : 10.1145/1859204.1859227. 34

T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comput. Intell.*, 5(3) :142–150, December 1989. ISSN 0824-7935. doi : 10.1111/j.1467-8640.1989.tb00324.x. URL http://dx.doi.org/10.1111/j.1467-8640.1989.tb00324.x. 31

C. Dezan and S. Zermani. Stochastic reliability evaluation of sea-of-tiles based on double gate controllable-polarity fets. In Jacques-Olivier Klein, Csaba Andras Moritz, and Sorin Cotofana, editors, *IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2014, Paris, France, July 8-10, 2014*, pages 169–170. IEEE Computer Society/ACM, 2014. doi : 10.1109/NANOARCH.2014.6880507. URL https://doi.org/10.1109/NANOARCH.2014.6880507. 6

H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping : part i. *IEEE Robotics Automation Magazine*, 13(2) :99–110, June 2006. ISSN 1070-9932. doi : 10.1109/MRA.2006.1638022. 83

T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell. The promise of high-performance reconfigurable computing. *Computer*, 41(2) :69–76, Feb 2008. ISSN 0018-9162. doi : 10.1109/MC.2008.65. 41

N. Friedman. The bayesian structural em algorithm. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 129–138, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-555-X. URL http://dl.acm.org/citation.cfm?id=2074094.2074110. 17

N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian networks to analyze expression data. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, RECOMB '00, pages 127–135, New York, NY, USA, 2000. ACM. ISBN 1-58113-186-0. doi : 10.1145/332306.332355. URL http://doi.acm.org/10.1145/332306.332355. 11

M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski. Stream-oriented fpga computing in the streams-c high level language. In *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No.PR00871)*, pages 49–56, 2000. doi : 10.1109/FPGA.2000.903392. 47

M.S. Grewal, L.R. Weill, and A. P. Andrews. *Global Positioning Systems, Inertial Navigation, and Integration*. Wiley-Interscience, 2007. ISBN 0470041900. 83

D. Grossman and P. Domingos. Learning bayesian network classifiers by maximizing conditional likelihood. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 46–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi : 10.1145/1015330.1015339. URL http://doi.acm.org/10.1145/1015330.1015339. 17

S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. Spark : a high-level synthesis framework for applying parallelizing compiler transformations. In *16th International Conference on VLSI Design, 2003. Proceedings.*, pages 461–466, Jan 2003. doi : 10.1109/ICVD.2003.1183177. 47

P. Gutberlet, J. Muller, H. Kramer, and W. Rosenstiel. Automatic module allocation in high level synthesis. In *Design Automation Conference, 1992., EURO-VHDL '92, EURO-DAC '92. European*, pages 328–333, Sep 1992. doi : 10.1109/EURDAC.1992.246223. 47

P.E. Hart and J. Graham. Query-free information retrieval. *IEEE Expert : Intelligent Systems and Their Applications*, 12(5) :32–37, September 1997. ISSN 0885-9000. doi : 10.1109/64.621226. URL http://dx.doi.org/10.1109/64.621226. 11

M. Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In *UAI '86 : Proceedings of the Second Annual*

*Conference on Uncertainty in Artificial Intelligence, University of Pennsylvania, Philadelphia, PA, USA, August 8-10, 1986*, pages 149–164, 1986. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1776&proceeding_id=1002. 21

C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 26(1) : 100–106, January 1983. ISSN 0001-0782. doi : 10.1145/357980.358021. URL http://doi.acm.org/10.1145/357980.358021. 47

E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumière project : Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 256–265, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-555-X. URL http://dl.acm.org/citation.cfm?id=2074094.2074124. 11

A. Huang, C.and Darwiche. Inference in belief networks : A procedural guide. *International Journal of Approximate Reasoning*, 15 :225–263, 1996. 10, 21, 23

F.V. Jensen and T.D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2nd edition, 2007. ISBN 0387682813. 12, 31, 33

F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4 : 269–282, 1990. ISSN 0723-712X. 10, 21, 23

M.I Jordan. Learning in graphical models. *STATISTICAL SCIENCE*, 19(1) :140–155, 2004. 17

G. Kaspi and J. Ratsaby. Parallel processing algorithm for Bayesian network inference. In *27th IEEE Convention of Electrical Electronics Engineers in Israel (IEEEI)*, pages 1–5, Nov 2012. doi : 10.1109/EEEI.2012.6377114. 34

Y.G. Kim and M. Valtorta. On the detection of conflicts in diagnostic bayesian networks using abstraction. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 362–367, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-385-9. URL http://dl.acm.org/citation.cfm?id=2074158.2074199. 11

V. Kindratenko and D. Pointer. A case study in porting a production scientific supercomputing application to a reconfigurable computer. In *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 13–22, April 2006. doi : 10.1109/FCCM.2006.5. 41

U. Kjærulff. Triangulation of graphs – algorithms giving small total state space. Technical report, 1990. 22

U. Kjærulff. Reduction of computational complexity in bayesian networksthrough removal of weak dependences. In *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence*, UAI'94, pages 374–382, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1-55860-332-8. URL http://dl.acm.org/citation.cfm?id=2074394.2074442. 21

D. Koller and A. Pfeffer. Object-oriented bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, pages 302–313, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-485-5. URL http://dl.acm.org/citation.cfm?id=2074226.2074262. 31

A. V. Kozlov and J. P. Singh. A parallel lauritzen-spiegelhalter algorithm for probabilistic inference. In *Proceedings of Supercomputing '94*, pages 320–329, Nov 1994. doi : 10.1109/SUPERC.1994.344295. 40, 43

Z. Kulesza and W. Tylman. Implementation of Bayesian network in fpga circuit. In *Proceedings of the International Conference on Mixed Design of Integrated Circuits and System*, pages 711–715, June 2006. doi : 10.1109/MIXDES.2006.1706677. 43

I. Kuon and J. Rose. Measuring the gap between fpgas and asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2) :203–215, Feb 2007. ISSN 0278-0070. doi : 10.1109/TCAD.2006.884574. 41

R. B. Langley. The gps error budget. *GPS world*, 8(3) :51–56, 1997. 84, 132

S. L. Lauritzen and D. J. Spiegelhalter. Readings in uncertain reasoning. chapter Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1-55860-125-2. URL http://dl.acm.org/citation.cfm?id=84628.85343. 10, 21

S.L Lauritzen. Hugin, (version 8.0) [computer software]. 2014. 57

S.T Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87 :1098–1108, 1992. 21

Z. Li and B. D'Ambrosio. Efficient inference in bayes networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11(1) : 55 – 81, 1994. 21

Z. Lian, Y. Jinsong, W. Jiuqin, and X. Wei. Real time diagnosis with compiling Bayesian networks. In *6th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 542–546, June 2011. doi : 10.1109/ICIEA.2011.5975645. 27, 106

M. Lin, I. Lebedev, and J. Wawrzynek. High-throughput bayesian computing machine with reconfigurable hardware. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '10, pages 73–82, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-911-4. doi : 10.1145/1723112.1723127. URL http://doi.acm.org/10.1145/1723112.1723127. 43

B. Liu and X. Wu. Mission reliability analysis of missile defense system based on dodaf and bayesian networks. In *2011 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, pages 777–780, June 2011. doi : 10.1109/ICQR2MSE.2011.5976725. 11

D. Margaritis. Learning bayesian network model structure from data. Technical report, 2003. 17

G. Martin and H. Chang. *Winning the SoC revolution : experiences in real design*. Springer Science & Business Media, 2012. 4

R.E. McDermott, R.J Mikulak, and M.R Beauregard. The basics of fmea. 1999. 4, 36, 79

O.J. Mengshoel, D.C. Wilkins, and .D Roth. Initialization and restart in stochastic local search : Computing a most probable explanation in bayesian networks. *IEEE Transactions on Knowledge and Data Engineering*, 23(2) :235–247, 2011. URL http://cogcomp.org/papers/MengshoelWiRo11.pdf. 21

E. Monmasson. Fpgas : Fundamentals, advanced features, and applications in industrial electronics [book news]. *IEEE Industrial Electronics Magazine*, 11(2) : 73–74, June 2017. ISSN 1932-4529. doi : 10.1109/MIE.2017.2704499. 41

K.P Murphy. The bayes net toolbox for matlab. *Computing Science and Statistics*, 33 :2001, 2001. 57

K.P. Murphy. *Dynamic Bayesian Networks : Representation, Inference and Learning*. PhD thesis, 2002. AAI3082340. 31

P. Naïm, P.H. Wuillemin, P. Leray, O. Pourret, and A. Becker. *Réseaux bayésiens*. Algorithmes. Eyrolles, 2007. URL https://hal.archives-ouvertes.fr/hal-00412267. 3, 10, 17

R.E. Neapolitan. *Learning Bayesian Networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003. ISBN 0130125342. 17

A. Nicolau and R. Potasmann. Incremental tree height reduction for high level synthesis. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*,

DAC '91, pages 770–774, New York, NY, USA, 1991. ACM. ISBN 0-89791-395-7. doi : 10.1145/127601.127767. URL http://doi.acm.org/10.1145/127601.127767. 46

J. D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5) :879–899, May 2008. 41

J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7. 10

J. Pearl. Causality : models, reasoning, and inference. *Cambridge University Press. ISBN 0*, 521(77362) :8, 2000. 12, 14, 80

J. Pearl and S. Russell. *Bayesian networks*. Computer Science Department, University of California, 1998. 3, 34

K. Pocek, R. Tessier, and A. DeHon. Birth and adolescence of reconfigurable computing : a survey of the first 20 years of field-programmable custom computing machines. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 1–17, April 2013. doi : 10.1109/FPGA.2013.6882273. 41, 42

L. Portinale and D. Codetta-Raiteri. Arpha : An fdir architecture for autonomous spacecrafts based on dynamic probabilistic graphical models. In *Proc. IJCAI workshop on AI on Space, Barcelona*, 2011. 4, 34

B. Ricks and O.J Mengshoel. Diagnosis for uncertain, dynamic and hybrid domains using bayesian networks and arithmetic circuits. *International Journal of Approximate Reasoning*, 55(5) :1207–1234, 2014. 34

J. Rose, R. J. Francis, D. Lewis, and P. Chow. Architecture of field-programmable gate arrays : the effect of logic block functionality on area efficiency. *IEEE Journal of Solid-State Circuits*, 25(5) :1217–1225, Oct 1990. ISSN 0018-9200. doi : 10.1109/4.62145. 41

D. Salós, C. Macabiau, A. Martineau, B. Bonhoure, and D. Kubrak. Nominal gnss pseudorange measurement model for vehicular urban applications. In *Position Location and Navigation Symposium (PLANS), 2010 IEEE/ION*, pages 806–815. IEEE, 2010. 83

W.G. Schneeweiss. Advanced fault tree modeling. *j-jucs*, 1999. 36

J. Schumann, O.J. Mengshoel, and T. Mbaya. Integrated software and sensor health management for small spacecraft. In *Proceedings of the 2011 IEEE Fourth International Conference on Space Mission Challenges for Information Technology*, SMC-IT '11, pages 77–84, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4446-5. doi : 10.1109/SMC-IT.2011.25. URL http://dx.doi.org/10.1109/SMC-IT.2011.25. 34, 103

J. Schumann, T. Mbaya, O. Mengshoel, K. Pipatsrisawat, A. Srivastava, A. Choi, and A. Darwiche. Software health management with bayesian networks. *Innovations in Systems and Software Engineering*, 9(4) :271–292, 2013a. 34, 35

J. Schumann, K.Y. Rozier, T. Reinbacher, O.J. Mengshoel, T. Mbaya, and C. Ippolito. Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. In *Proceedings of the 2013 Annual Conference of the Prognostics and Health Management Society (PHM2013)*, October 2013b. 34, 40, 44, 64

J. Schumann, P. Moosbrugger, and K.Y. Rozier. R2u2 : Monitoring and diagnosis of security threats for unmanned aerial systems. In *Runtime Verification*, pages 233–249. Springer, 2015. 34

A. Silberstein, M.and Schuster, A. Geiger, D.and Patney, and J.D. Owens. Efficient computation of sum-products on gpus through software-managed cache. In *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ICS '08, pages 309–318, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-158-3. doi : 10.1145/1375527.1375572. URL http://doi.acm.org/10.1145/1375527.1375572. 43

S. Sjoholm and L. Lindh. *VHDL for Designers*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997. ISBN 0134734149. 44

C. Skaanning. A knowledge acquisition tool for bayesian-network troubleshooters. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, pages 549–557, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-709-9. URL http://dl.acm.org/citation.cfm?id=2073946.2074010. 11

M. J. S. Smith. *Application-specific Integrated Circuits*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. ISBN 0-201-50022-1. 41

J. Teich. Hardware/software codesign : The past, the present, and predicting the future. *Proceedings of the IEEE*, 100(Special Centennial Issue) :1411–1430, May 2012. ISSN 0018-9219. doi : 10.1109/JPROC.2011.2182009. 48

A. Tridgell. Canberra uav outback challenge. 2014. 5, 94

M. Valtorta, Y.G. Kim, and J. Vomlel. Soft evidential update for probabilistic multiagent systems. *Int. J. Approx. Reasoning*, 29(1) :71–106, January 2002. ISSN 0888-613X. doi : 10.1016/S0888-613X(01)00056-1. URL http://dx.doi.org/10.1016/S0888-613X(01)00056-1. 31

R. A. Walker and S. Chaudhuri. Introduction to the scheduling problem. *IEEE Design Test of Computers*, 12(2) :60–69, Summer 1995. ISSN 0740-7475. doi : 10.1109/54.386007. 47

S. Zermani, C. Dezan, H. Chenini, J.P. Diguet, and R. Euler. Fpga implementation of bayesian network inference for an embedded diagnosis. In *Prognostics and Health Management (PHM), 2015 IEEE Conference on*, pages 1–10. IEEE, 2015a. 6, 75, 112, 115, 133

S. Zermani, C. Dezan, R. Euler, and J.P. Diguet. Bayesian network-based framework for the design of reconfigurable health management monitors. In *Adaptive Hardware and Systems (AHS), 2015 NASA/ESA Conference on*, pages 1–8. IEEE, 2015b. 6, 75, 110, 133

S. Zermani, C. Dezan, C. Hireche, R. Euler, and J. P. Diguet. Embedded and probabilistic health management for the gps of autonomous vehicles. In *2016 5th Mediterranean Conference on Embedded Computing (MECO)*, pages 401–404, June 2016. doi : 10.1109/MECO.2016.7525792. 6, 100

S. Zermani, C. Dezan, and R. Euler. Embedded decision making for uav missions. In *2017 6th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, June 2017a. doi : 10.1109/MECO.2017.7977165. 6, 75, 99, 100, 132

S. Zermani, C. Dezan, C. Hireche, R. Euler, and J.P. Diguet. Embedded context aware diagnosis for a UAV soc platform. *Microprocessors and Microsystems - Embedded Hardware Design*, 51 :185–197, 2017b. doi : 10.1016/j.micpro.2017.04.013. URL https://doi.org/10.1016/j.micpro.2017.04.013. 6, 100, 117, 118, 133

N.L. Zhang and D. Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5 :301–328, 1996. 21

L. Zheng and O.J. Mengshoel. Optimizing parallel belief propagation in junction trees using regression. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 757–765, 2013. doi : 10.1145/2487575.2487611. URL http://doi.acm.org/10.1145/2487575.2487611. 40, 43

L. Zheng and N. Mrad. Validation of strain gauges for structural health monitoring with Bayesian belief networks. *Sensors Journal, IEEE*, 13(1) :400–407, Jan 2013. ISSN 1530-437X. doi : 10.1109/JSEN.2012.2217954. 34

## Implémentation sur SoC des réseaux Bayésiens pour l'état de santé et la décision dans le cadre de missions de véhicules autonomes


### Résumé

Les véhicules autonomes, tels que les drones, sont utilisés dans différents domaines d'application pour exécuter des missions simples ou complexes. D'un coté, ils opèrent généralement dans des conditions environnementales incertaines, pouvant conduire à des conséquences désastreuses pour l'humain et l'environnement. Il est donc nécessaire de surveiller continuellement l'état de santé du système afin de pouvoir détecter et localiser les défaillances, et prendre la décision en temps réel. Cette décision doit maximiser les capacités à répondre aux objectifs de la mission, tout en maintenant les exigences de sécurité. D'un autre coté, ils sont amenés à exécuter des tâches avec des demandes de calcul important sous contraintes de performance. Il est donc nécessaire de penser aux accélérateurs matériels dédiés pour décharger le processeur et répondre aux exigences de la rapidité de calcul.

C'est ce que nous cherchons à démontrer dans cette thèse à double objectif. Le premier objectif consiste à définir un modèle pour l'état de santé et la décision. Pour cela, nous utilisons les réseaux Bayésiens, qui sont des modèles graphiques probabilistes efficaces pour le diagnostic et la décision sous incertitude. Nous avons proposé un modèle générique en nous basant sur une analyse de défaillance de type FMEA (Analyse des Modes de Défaillance et de leurs Effets). Cette analyse prend en compte les différentes observations sur les capteurs moniteurs et contextes d'apparition des erreurs. Le deuxième objectif était la conception et la réalisation d'accélérateurs matériels des réseaux Bayésiens d'une manière générale et plus particulièrement de nos modèles d'état de santé et de décision. N'ayant pas d'outil pour l'implémentation embarqué du calcul par réseaux Bayésiens, nous proposons tout un atelier logiciel, allant d'un réseau Bayésien graphique ou textuel jusqu'à la génération du bitstream prêt pour l'implémentation logicielle ou matérielle sur FPGA. Finalement, nous testons et validons nos implémentations sur la ZedBoard de Xilinx, incorporant un processeur ARM Cortex-A9 et un FPGA.



**Mot clés :** Réseaux Bayésiens, Etat de santé, Décision, FMEA, FPGA, Synthèse de haut niveau, Implémentation matérielle/logicielle.


## SoC implementation of Bayesian networks for health management and decision making for autonomous vehicles missions


### Abstract

Autonomous vehicles, such as drones, are used in different application areas to perform simple or complex missions. On one hand, they generally operate in uncertain environmental conditions, which can lead to disastrous consequences for humans and the environment. Therefore, it is necessary to continuously monitor the health of the system in order to detect and locate failures and to be able to make the decision in real time. This decision must maximize the ability to meet the mission objectives while maintaining the security requirements. On the other hand, they are required to perform tasks with large computation demands and performance requirements. Therefore, it is necessary to think of dedicated hardware accelerators to unload the processor and to meet the requirements of a computational speed-up.

This is what we tried to demonstrate in this dual objective thesis. The first objective is to define a model for the health management and decision making. To this end, we used Bayesian networks, which are efficient probabilistic graphical models for diagnosis and decision-making under uncertainty. We propose a generic model based on an FMEA (Failure Modes and Effects Analysis). This analysis takes into account the different observations on the monitors and the appearance contexts. The second objective is the design and realization of hardware accelerators for Bayesian networks in general and more particularly for our models of health management and decision-making. Having no tool for the embedded implementation of computation by Bayesian networks, we propose a software workbench covering graphical or textual Bayesian networks up to the generation of the bitstream ready for the software or hardware implementation on FPGA. Finally, we test and validate our implementations on the Xilinx ZedBoard, incorporating an ARM Cortex-A9 processor and an FPGA.



**Keywords:** Bayesian network, Health management, Decision making, FMEA, FPGA, High level synthesis, HW/SW implementation.